VLSI Physical Design

Offline Internship Report

Team ID: 587579

Report Submitted by

SAHU ARABINDA KAILASH

EN. NO: - 210230111008

In partial fulfilment for the award if the degree of

Bachelor of Engineering

in

Electronics and Communication Engineering

Dr. S. & S.S. Ghandhy Government Engineering College, Surat

Gujarat Technological University, Ahmedabad

June-July 2024

Dr. S. & S.S Ghandhy Government Engineering College, Surat

Near Vanita Vishram Swimming Pool, Majura Gate, Ring Road, Surat -395001

Certificate

Date:

This is to certify that the project report submitted along with the project entitled ***VLSI Physical Design*** has been carried out by ***Sahu Arabinda Kailash*** under my guidance in partial fulfilment for the degree of Bachelor of Engineering in Electronics and Communication Engineering, 7th Semester of Gujarat Technological University, Ahmedabad during the academic year 2024-25.

Prof. P. V. Pithadiya
(Internal Guide)

Prof. T. P. Dave
(Head of Department)

Letter head certificate signed by Internship mentor

सरदार वल्लभभाई राष्ट्रीय प्रौद्योगिकी संस्थान, सूरत
SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY, SURAT
સરદાર વલ્લભભાઈ રાષ્ટ્રીય પ્રૌદ્યોગિકી સંસ્થા, સુરત
शिक्षा मंत्रालय, भारत सरकार द्वारा NITSER अधिनियम के तहत स्थापित राष्ट्रीय महत्व का संस्थान
(An Institute of National Importance, Established under NITSER Act by Ministry of Education, Govt. of India)
SVNIT

Date: 15/07/2024

TO WHOM IT MAY CONCERN

This is to certify that ***Sahu Arabinda Kailash*** , a student of Dr. S. & S.S. Ghandhy Government Engineering College, Surat has successfully completed his internship in the field of VLSI Physical Design from 29th June, 2024 to 12th July, 2024 (Total number of Weeks: 02) under the guidance of Dr. Pinalkumar J. Engineer (Associate Professor, Department of Electronics Engineering, SVNIT Surat).

His internship activities include VHDL module simulations, Advanced adder design implementations, Multiplier design implementations and ORFS tool usage for RTL to GDS flow.

During the duration of his internship program with us, he had exposed to different processes and was found diligent, hardworking, and inquisitive.

We wish him every success in his life and career.

For SVNIT, Surat
Authorised Signature with Industry Stamp

S.V.N.I.T. * स.व. राष्ट्रीय प्रौद्योगिकी संस्थान * National Institute of Technology
डी.ई.सी.ई.
D.E.C.E.

इच्छानाथ, सूरत - 395007, गुजरात, भारत, दूरभाष: (का) 91-0261-2259571, 2259582-84 वेबसाईट: https://www.svnit.ac.in
Ichchhanath, Surat-395007, Gujarat, India. Phone: (O) 91-0261-2259571, 2259582-84 Website: https://www.svnit.ac.in

DECLARATION OF ORIGINALITY

I hereby declare that the Internship report submitted along with the Internship entitled VLSI Physical design submitted in partial fulfilment for the degree of Bachelor of Engineering in Electronics and Communication engineering to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me at SVNIT under supervision of Dr. Pinalkumar J. Engineer and that no part of this report has been directly copied from any students reports or taken from any other source, without providing due reference.

Date:

Place: Surat

SAHU ARABINDA KAILASH . _____ .

210230111008

ACKNOWLEDGMENT

This is the place to admit that while there appears only author on the cover, this work just as any other, is a product of the interaction with and support during our internship work, among them, first I express my gratitude to my guides Dr. Pinalkumar J. Engineer & Prof. Dharmesh J. Patel for their affection throughout guidance, advice, and encouragement. Special thanks to my college for giving me the invaluable knowledge. Also, I interacted with some other students having their study and research there for guidance of software installation, I am grateful to them also for providing us guidance. Above all I am thankful to almighty God for everything.

SAHU ARABINDA KAILASH

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Symbol	Abbreviations
SVNIT	Sardar Vallabhbhai National Institute of Technology
RTL	Register Transfer Level
GDS	Graphic Design System
IC	Integrated Circuit
VLSI	Very Large-Scale Integration
RCA	Ripple Carry Adder
CSA	Carry Save Adder
CLA	Carry Look Ahead Adder
E.C.	Electronics and Communication
nm	nano meter
FET	Field Effective Transistor
TSMC	Taiwan Semiconductor Manufacturing Company
GAA	Gate-All-Around
AI	Artificial Intelligence
ASIC	Application Specific Integrated Circuit
SoC	System-on-Chip
EDA	Electronic Design Automation
GTU	Gujarat Technological University
DGGEC	Dr. S. & S.S. Ghandhy Government Engineering College
ALU	Arithmetic Logic Unit
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
LSB	Least Significant Bit
MSB	Most Significant Bit
DSP	Digital Signal Processing
EDA	Electronic Design Automation
CAD	Computer-aided Design
ORFS	OpenROAD-flow-scripts
SDC	Synopsys Design Complier

GUI	Graphic User Interface
HDL	Hardware Description Language
WSL	Windows Subsystem Linux
VMBox	Virtual Machine Box
PDK	Process Design Kit
CTS	Clock Tree Synthesis
BLIF	Berkely Logic Interchange Format
PDN	Power Distribution Network
SDC	Synopsys Design Constraints
SPEF	Standard Parasitic Exchange Format
um	micro meter

ABSTRACT

The main objective of the internship program is to learn about the RTL to GDS Design flow of Digital circuits to form an IC. In the journey of this learning, we learned about advanced adders and multipliers and some other components. Also, during the program, we learned about opensource tools and software related to ASIC designing process. The SVNIT has great environment and guidance for learnings. The internship is focused to the vlsi physical design flow in digital field. In duration of these internship, I learned a lot about Physical design aspect of VLSI and looking forward for VLSI ASIC design related works and projects.

CHAPTER 1: INTRODUCTION

In the E.C Engineering various Electronics devices are used such as resistor, transistors, diodes, and ICs. Among those devices IC are game changer technology. And all this possible due to semiconductors. Such as Silicon, graphene, etc. are generally used to form IC.

The current IC technology window reached is of 5-3nm only. Further research to narrow the channel length of FETs is going on. The SoC such as Tensor G3 used in Google Pixel 8 Series, M3 used in latest MacBook, Exynos 2400 in galaxy S24 series, Qualcomm's Snapdragon S gen 3 in Samsung galaxy S24 ultra are one of the latest technology-based ASICs designed by foundries such as TSMC, GAA, and Samsung. Recently Nvidia become first company to research on IC based AI neural engines and render engines. All this discussion to ensure that our internship program for VLSI physical design flow is to learn about designing of ASICs.

In most of Indian ASIC based companies are service based such as designing or testing and verification is carried out. I chose this topic of internship to explore about ASIC and RTL-to-GDS design flow.

1.1 INTERNSHIP DESCRIPTION

The Summer Internship of 2 weeks is a part of GTU academics extra curriculum activity allowing students to get hands on experience before placement drives in campus. This report outlines the experiences and accomplishments during the internship period at the VLSI Research Lab in Electronics Engineering Department of SVNIT, under the mentorship and supervision of Dr. Pinalkumar J. Engineer, Associate Professor at SVNIT. Duration of internship as mentioned by GTU Notification 29th June 2024 to 12th July 2024. During this period in 11 working days, the internship focused on the VLSI physical design flow in digital domain, providing exposure in implementing various digital circuits as RTL-to-GDS design such as RCA, CLA, CSA, Array Multipliers etc. VLSI technology is pivotal in the enhancement or advancement of modern electronics, enabling the creation of complex IC by combining thousands or millions of transistors onto a single chip. The

physical design phase in VLSI involves conversion of a high-level abstract design into a detailed manufacturable layout. This phase is crucial as it directly impacts the performance, power consumption, heat dissipation, and overall efficiency of IC.

1.2 SCOPE AND OBJECTIVE OF INTERNSHIP

1.2.1 Objectives

The primary objectives of internship were:

- To gain practical knowledge of the VLSI physical design flow.
- To implement and optimize physical designs for different types of digital circuits.
- To understand the tools and methodologies used in the VLSI design flow.

1.2.2 Scope of internship

During the internship the below mentioned tasks were undertaken:

- Implementation of different advanced adders and multiplier circuits in Xilinx tool for RTL and Simulation.
- Designing and Optimization of various integrated circuits.
- Use of opensource EDA tools for layout design, verification, and optimization.

This internship provided valuable insights into the intricacies of VLSI physical design, bridging the gap between theoretical knowledge and practical application. The experience gained has equipped me with the skills and understanding necessary to contribute effectively to the field of VLSI design.

CHAPTER 2: DESIGN ON XILINX

The initial stage of VLSI physical design flow is implementing the design module and verify the simulation output. For this we can create files such as Verilog file with ".v" extension, VHDL modules with ".vhd" file extension and ".tb" file extension for test bench. Those files are used to form the RTL, Schematics and Verification by simulation of logics with test vectors at Test bench. Apart from general digital circuits we learned in VLSI design subject and DSD subject of GTU syllabus I learned about multipliers and Advanced 8 bit and extensible digital circuit designs.

2.1 ADVANCED ADDER DESIGNS

In advanced adder architecture i learned about 3 kinds of adders used in forming ALUs and intermediate stage of multipliers used in ALU. The 3 kinds of Adder design I learned are 1) Ripple Carry Adder, 2) Carry Save Adder, 3) Carry Look Ahead Adder. Those adders have their specific application and use cases based on their design parameters such as size, delay, performance for n bits architecture and area occupied on chip etc. I have designed and implemented as per given instructions using VHDL programming and of 8 bits architecture for all these kinds. Let's see more about different adder designs in further sections.

2.1.1 Ripple Carry Adder (RCA)

A Ripple Carry Adder is digital binary adder circuit designed to two n-bit (n no. of bits in binary number). It is the extension to basic RCA concept, which scaled to handle any no of bits. The main building block of the RCA consists of Full Adders. As we know the Full Adder has three inputs as two operands and Carry input, and two outputs as Sum and Carry out. An n-bit RCA consists of chain of n Full adders (i.e. an 8-bit RCA consists of 8 Full Adder blocks. The RCA provides a n-bit Sum output and Carry output based on provided 2 n-bit operands and carry in signal usually zero (0).

A n-bit RCA does bit-wise addition of two operands using full adders, where the sum output of full adder will be summing output of corresponding bit and carry forwarded to next stage of full adder carry input. The term "Ripple" in RCA refers to the way of the carry propagation through the adders. The Carry out from the LSB adder ripples through the MSB of adder. This ripple propagation of carry causes delay proportional to the number of bits, making the adder slow for large width of bits. Whereas generally when we use an 8-bit RCA, and for addition of 16-bit operands we use RCA 2 times to save the area and provide performance. It is comparatively slower than CSA and CLA but design wise simplest among all other advanced n bit adders. Also, the hardware area is minimal as it comprises of series of full adders only.

The general time delay of Full Adders are as follows

T_{sum} for sum calculation typically involves *XOR* gate, and T_{carry} for carry generation typically involves *AND* and *OR* gates. Now, for the time delay calculation of n-bit RCA, consists of n full adders. Each stage carry output propagated after T_{carry} delay hence n times T_{carry} (i.e. n * T_{carry}). The sum output of corresponding bits available only after carry input of previous stage. The MSB calculated after n-1 bit's carry out generation, hence we can say that for an n-bit RCA the time delay is given by following equation

$$T_{rca} = n * (T_{carry} + T_{sum}) \quad \text{..... 2.1.1}$$

Assuming $T_{carry} = T_{sum}$, for 8-bit RCA we get delay of 16-units time.

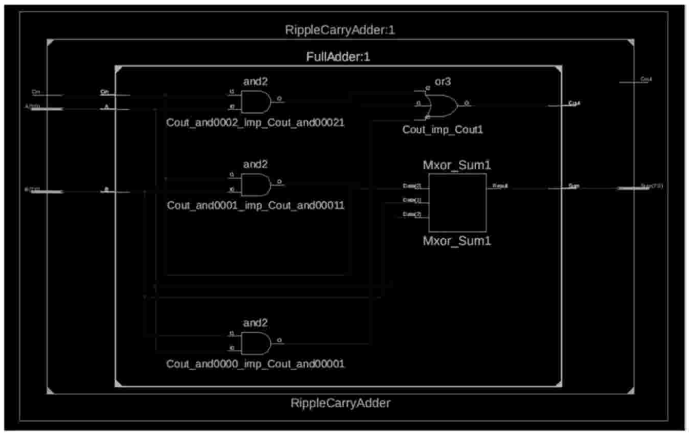

Fig 2.1.1 8-Bit RCA RTL From Xilinx Design Implementation

I have implemented 8-bit RCA during the internship also, implemented 4-bit design but showing results for 8-bit RCA only.

Here I have used the generate function in VHDL so by which in RTL view we can see that there is only one full adder block is being shown.

Fig 2.1.2 8-Bit RCA Simulation Output From Xilinx Design Implementation

RCA is basically used in simple arithmetic units where speed isn't the primary concern, for learning and educational purposes due to its basic simple design and used in those embedded systems where low power consumption and simplicity are more important parameters than speed.

2.1.2 Carry Look Ahead Adder (CLA)

Carry Look Ahead Adder is another kind of binary adder which is used in addition of more than 2-bit two operands. In comparison of RCA, it is faster but design is way more complex than that of RCA. The delay due to sequential carry propagation in RCA, is resolved in CLA by using a special combinational logic circuit. This Combinational circuit generates the carry of each stage in single machine cycle. The carry of each stage generated in advance.

The key concept of CLA is its different combinational logic circuit and partial full adder addition. The intermediate combinational circuit generates 2 signals i.e. G (generated carry bits) P (Propagated carry bits). The G signal is a bit position which generates carry if both corresponding bits of operands are logic high '1', in Boolean logic expressed as

$$G_i = A_i \cdot B_i \quad \text{2.1.2}$$

The P signal is a bit position which generates carry if any of the corresponding bits if operands is logic high '1', in Boolean logic expressed as

$$P_i = A_i + B_i \quad \text{2.1.3}$$

The Carry Look ahead logic generates the Carry signal of i^{th} bit in previous stage, with combinational logic of Boolean logic expressed as

$$C_{i+1} = G_i + P_i \cdot C_i \quad \text{2.1.3}$$

This combinational logic helps in generating carry parallelly reducing the delay of carry generation. The Corresponding Sum bits are generated with help of the xoring of corresponding bits of operands.

$$S_i = P_i \oplus C_i \quad \text{2.1.4}$$

The output of all bits sums and final carry is generated in 4 machine cycles but, the only problem in CLA concept is the combinational logic for parallel carry generation requires an *'and gate'* and *'or gate'* of n+1 inputs for n-bits width of CLA. i.e. for a 4-bit CLA in combinational logic stage it requires an *'and gate'* of 5 inputs. By this the fan in and fan out delay by which indirectly we get propagational delays. Although this delay is smaller than RCA but the design complexity, costly design, power consumption causes its limitations for some other kinds of specific application.

Time delay calculation for CLA, here we have 3 different parts. i.e. G and P signal generation, Sum generation and Carry Calculation delay. For G and P, we can express delay by following equation

$$t_{G+P} = \max (t_{AND}, t_{OR}) \quad \text{..... 2.1.5}$$

The sum calculation delay involves only xor gate delay hence we can express as

$$t_S = t_{XOR} \quad \text{..... 2.1.6}$$

The Carry calculation involves multiple levels of 'AND' & 'OR' gates. For an n-bit CLA, the carry-out of the last bit involves a maximum of $\log_2(n)$ levels of gates. By which we can express carry delay by the following equation

$$t_C = \log_2(n) * (t_{OR} + t_{AND}) \quad \text{..... 2.1.7}$$

By all above 3 three equations the final delay for CLA n bit will be given by following equation

$$t_{CLA} = t_{XOR} + \log_2(n) * (t_{OR} + t_{AND}) + \max (t_{AND}, t_{OR}) \quad \text{..... 2.1.8}$$

For example, in an 8-bit CLA all logic gates XOR, AND & OR having unit time delay, from the equation 2.1.8 we can say that 8-unit time delay.

I have implemented 8-bit CLA during the internship also, implemented 4-bit design but showing results for 8-bit CLA only. This implementation is similar kind as RCA. And design is extensible to 16-bits or any n-bit as designed with generate function.

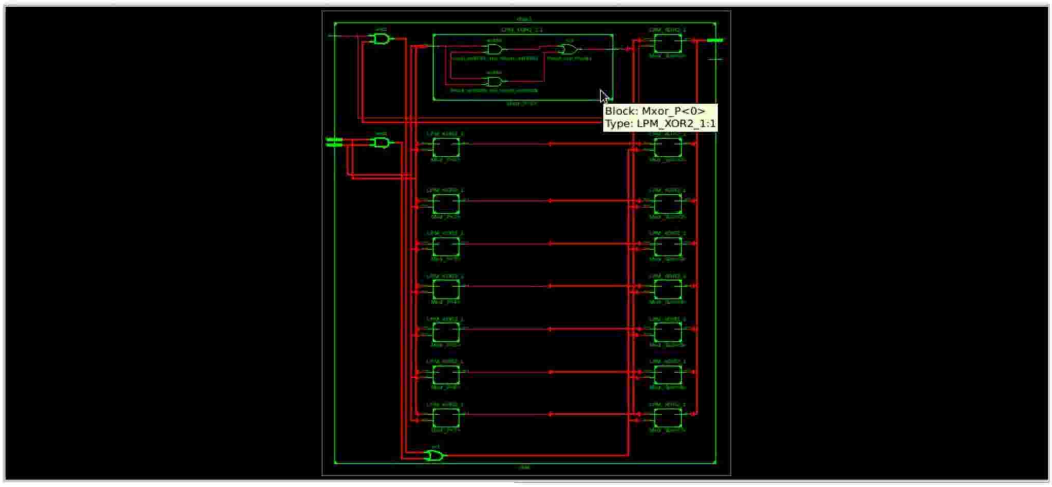

Fig 2.1.3 8-Bit CLA RTL From Xilinx Design Implementation

Fig 2.1.4 8-Bit CLA Simulation Output From Xilinx Design Implementation

2.1.3 Carry Save Adder (CSA)

A CSA is a type of binary digital adder used primarily in computer arithmetic operations. It is designed to add multiple operands simultaneously, which is particularly used in multiplication and accumulation operations. The CSA significantly speeds up the process

by not propagating carry bits immediately but rather saving them for later addition. For an n-bits width CSA, the structure primarily consists of multiple stages of addition, where each stage handles the addition of three inputs and generates two outputs a sum of n-bits width and a carry bit. The key component of the CSA is basically a full adder.

For an n-bit CSA, in stage 1 of initial addition n Full Adders, each taking three input bits and generating n sum bits and n carry bits. In stage 2 of addition of sum and carry bits sum bits from the first stage are passed to the next stage and carry bits are shifted left and then added. This stage of shifting and adding continues until we have n-bit sum output and a single bit carry output.

Timing calculations for CSA, According to structure we have initially a full adder delay. Each full adder delay associated with generating sum and carry bit where, delay of full adder expressed by following equation

$$t_{FA} = t_{XOR} + t_{AND} + t_{OR} \quad \text{..... 2.1.5}$$

The next delay is of CSA logic delay. The critical path delay is determined by the longest chain of carry propagation. For an n-bit CSA, the carry is not propagated immediately hence, the delay depends on the depth of carry-save stages. Now each stage involves full adder delays, typically uses $\log_2(n)$ stages for adding n bits.

The final delay for adding n-bit three operands, the delay determined by the depth of stages and delay of a single full adder. The final delay expressed as following equation,

$$T_{CSA} = k \oplus t_{FA} \quad \text{..... 2.1.6}$$

where k is the number of stages.

For example, by considering t_{FA} = 4-unit delay which consists 2-unit delay for XOR gate, 1 unit delay each for '*AND*' and '*OR*' gates. For an 8-bit CSA no. of stages in CSA is 3 from earlier discussion and final delay will of 12-unit time only. Hence, we can conclude that The CSA is an efficient adder for handling multiple inputs simultaneously without the immediate carry propagation delay.

I have implemented 8-bit CSA design on Xilinx here are the results of following design implementations in figures.

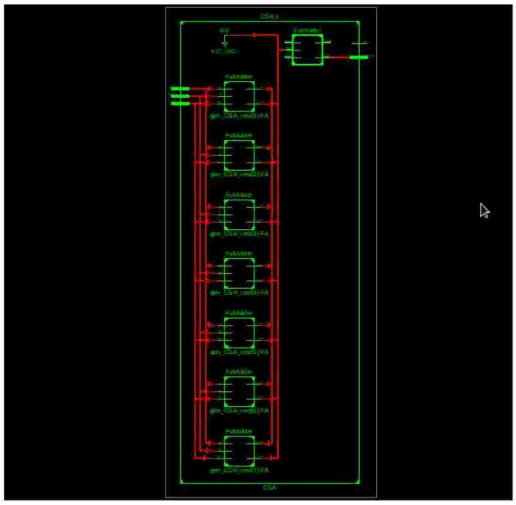

Fig 2.1.5 8-Bit CLA RTL From Xilinx Design Implementation

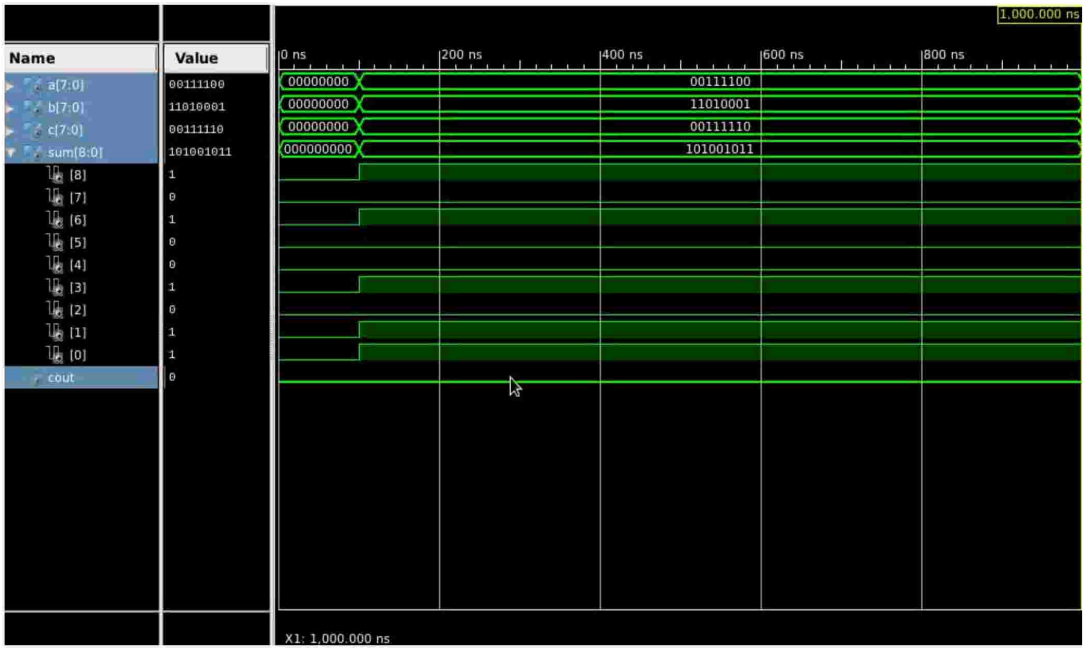

Fig 2.1.6 8-Bit CSA Simulation Output From Xilinx Design Implementation

2.1.4 Comparison of Adders

Table 2.1 Comparison of adders.

Parameter	RCA	CLA	CSA
Basic Operation	Sequential carry propagation	Parallel carry computation	Saves intermediate carries for later addition
Sum Calculation	Sequential	Parallel	Parallel with multiple stages
Carry Calculation	Sequential	Parallel using generates and propagate signal	Saved for later stages
Gate count	Low	High	Moderate to High
Area	Small	Large	Moderate
Power Consumption	Low	High	Moderate
Speed	Slow	Fast	Fast in multi-operand addition
Complexity	Low	High	Moderate
Scalability	Poor	Good	Good
Example 8-bit delay	16 units	9 units	12 units

2.2 MULTIPLIER DESIGNS

Multipliers are fundamental components in digital electronics, used to perform arithmetic multiplication of binary numbers. They are essential in various applications such as digital signal processing (DSP), computer graphics, and general-purpose computing. The general types of multipliers used in digital electronics are Array Multipliers, Carry Save Array Multipliers, Wallace Tree Multipliers, Sequential Multipliers, Booth Multipliers, Pipelined Multipliers, Bit-Serial Multipliers. Among these Array Multipliers are most basic multipliers and used in understanding of Multipliers Functions. The multipliers used and

applied in different application on basis of their speed, efficiency, versatility, accuracy, integration etc. The general applications of multipliers are Digital Signal Processing, Computer graphics, Cryptography, Machine Learning, Embedded systems, etc.

2.2.1 Array Multipliers

Array Multipliers also known as Fast Multipliers are straight-forward and efficient method for performing binary multiplication. They use an array of adders to sum partial products generated by multiplying each bit of one operand by each bit of the other operand. Let's understand design a timing of 2-bit, 3-bit and 4-bit array multipliers and understand difficulties in n-bit multiplier designs.

In array multiplier the first stage to get partial products hence for a 2-bit array multiplier let's have 2 operands X and Y of 2 bits which generate the product of 4 bits.

Table 2.2 2-bit Multiplication

		X_1	X_0	<= Multiplicand operand
	×	Y_1	Y_0	<= Multiplier operand
		X_1Y_0	X_0Y_0	<= Partial products with Y_0
C_2	X_1Y_1	X_0Y_1	×	<= Partial products with Y1 and weigh added by 2^1
C_2	$X_1Y_1 + C_1$	$X_1Y_0 + X_0Y_1$	X_0Y_0	<= Addition Results
P_3	P_2	P_1	P_0	<=Product bits

In above calculation we can see that X is multiplicand and Y is multiplier. And we get product of 4-bits. There are 4 partial products in above multiplication by multiplying the bits Y_0 and Y_1 bitwise to X vector and we required 4 *'and gates'* for the partial product generation. Further in next stage the addition of each partial products with the corresponding weights is done and we get respective product bits generated. In this 2-bit array multiplier we have 2 half adders for addition as we have addition for P_1 bit partial products which generates the Carry C_1, carried to the next stage and there another half adder

adds the next partial product and generated carry considered as 4th bit P_3, of the product bits.

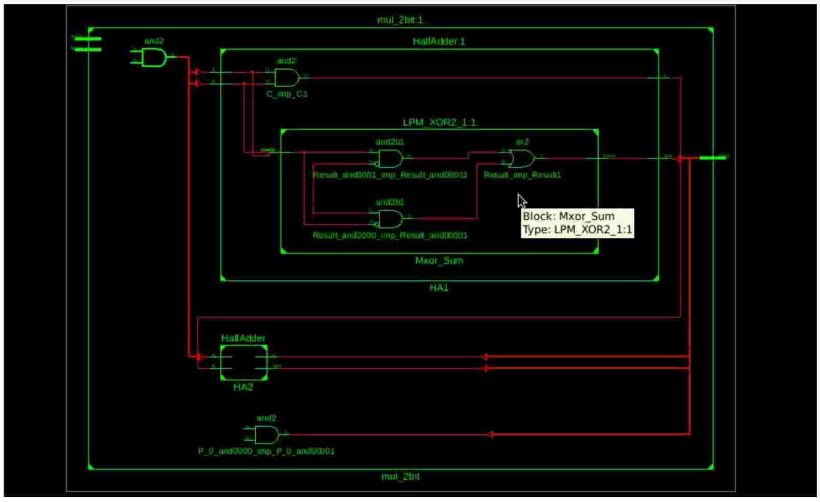

Fig 2.2.1 2-Bit Multiplier RTL From Xilinx Design Implementation

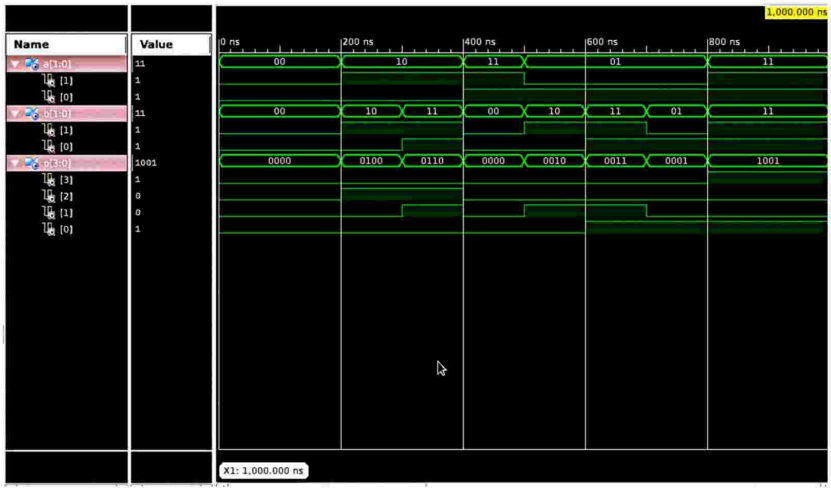

Fig 2.2.2 2-Bit Multiplier Simulation Output From Xilinx Design Implementation

Now in 3-bit fast multiplier let's see the design and the architecture. Similarly considering two 3 bits operands X and Y where X is multiplicand and Y is multiplier, and we have P vector of 6 bits.

Table 2.3 3-bit Multiplication

			X_2	X_1	X_0	<= Multiplicand operand
		×	Y_2	Y_1	Y_0	<= Multiplier operand
			X_2Y_0	X_1Y_0	X_0Y_0	<= Partial products with Y_0
		X_2Y_1	X_1Y_1	X_0Y_1	×	<= Partial products with Y_1 and weigh added by 2^1
	X_2Y_2	X_1Y_2	X_0Y_2	×	×	<= Partial products with Y_2 and weigh added by 2^2
C_{23}	X_2Y_2	X_2Y_1+	X_1Y_1+	X_1Y_0+	X_0Y_0	<= Addition Results and carry results where C_{1x} shows the carry generated by addition at first stage and then C_{2x} shows the carry generated by second stage of addition
	$+C_{22}$	$C_{21}+$	$C_{11}+$	X_0Y_1		
		$C_{12}+$	X_1Y_1+			
		X_1Y_2	X_0Y_2			
P_5	P_4	P_3	P_2	P_1	P_0	<=Product bits

In above calculation no. of adders may not be easy t find but no. of *'and gates'* are clearly visible which is 9. For P_0 we don't require adder hence P_0 is directly fetched. Now for P_1 bit we must use a half adder as no carry in from previous stage. The Sum output of that half adder will be P_1 and Carry will be carried forward as C_{11}. Now in next stage partial products and carry generated, added using full adders. And by reference to my circuit analysis during work I concluded that it requires 3 half adders and 3 full adders in structure.

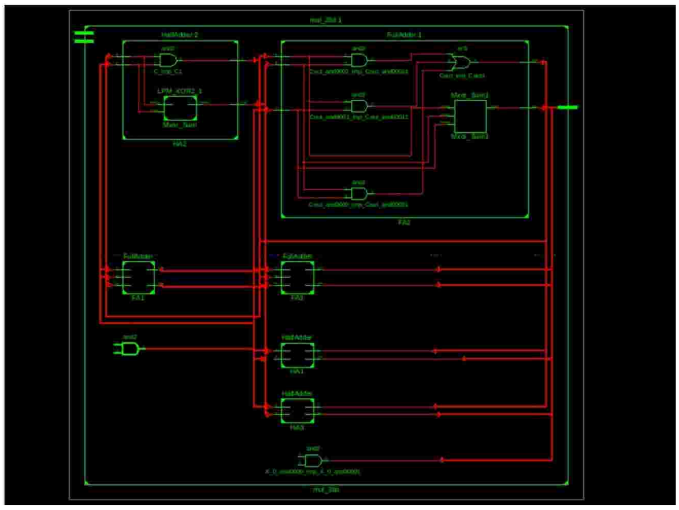

Fig 2.2.3 3-Bit Multiplier RTL From Xilinx Design Implementation

Fig 2.2.4 3-Bit Multiplier Simulation Output From Xilinx Design Implementation

Now let's see about 4-bit array multiplier. Similarly considering the A and B vector of 4 bits and output of 8 bits.

				A3	A2	A1	A0
				B3	B2	B1	B0
				A3B0	A2B0	A1B0	A0B0
			A3B1	A2B1	A1B1	A0B1	
		A3B2	A2B2	A1B2	A0B2		
	A3B3	A2B3	A1B3	A0B3			
P7	P6	P5	P4	P3	P2	P1	P0

Fig 2.2.5 4-Bit Multiplier Multiplication Method

From reference to above figure, we can say that we require 16 *'and gates'* to get partial products. And with reference to my lab-work I concluded that it requires 4 half adders and 8 full adders. By which we can conclude for an n-bit multipliers no of required components given by following equations

No. of *'and gates'* = n^2 ..... 2.2.1

No. of half adders = n 2.2.2

No. of full adders = n(n-2) 2.2.3

where n is width of input of operands of multipliers.

Now considering time delays taken by each multiplier and look at any kind of disadvantage. For a 2-bit, 3-bit, 4-bit or n-bit multiplier design we require only 1 gate delay for partial products. Now for 2-bit multiplier we have 2 half adders and the 2nd half adder is dependent on 1st adder for carry. Let's consider 2-unit delay required to propagate carry from 1st half adder hence total delay will be of 5-unit (i.e. 1+2+2).

Now in case of 3-bit multiplier in addition we have 3 levels 1st for half adder with 1-unit delay. 2nd level of full adders for intermediate stage with 2-unnit delay. And similar at final stage 3rd level of 2-unit delay. The total delay about of 6-unit time. Similarly in case if 4-bit multiplier we have final level of adders in 4th stage hence the total time delay can be of 8-unit times.

The design constraint of this fast multipliers or array multipliers is no. of logic and gates and full adder blocks required to get a design for n-bit width. Such as for a 8-bit array multiplier we need 64 *'and gates'* and about 8 half adders and 48 full adders for simplicity if we use all full adders implementation then also we require 56 different full adders as design of such array multiplier in VHDL is not easy because we can't use generate logic for addition stage. Also, the chip size and power consumption increase with increase in width of multiplier input. Now let's see the design implementation of 4-bit multiplier on Xilinx.

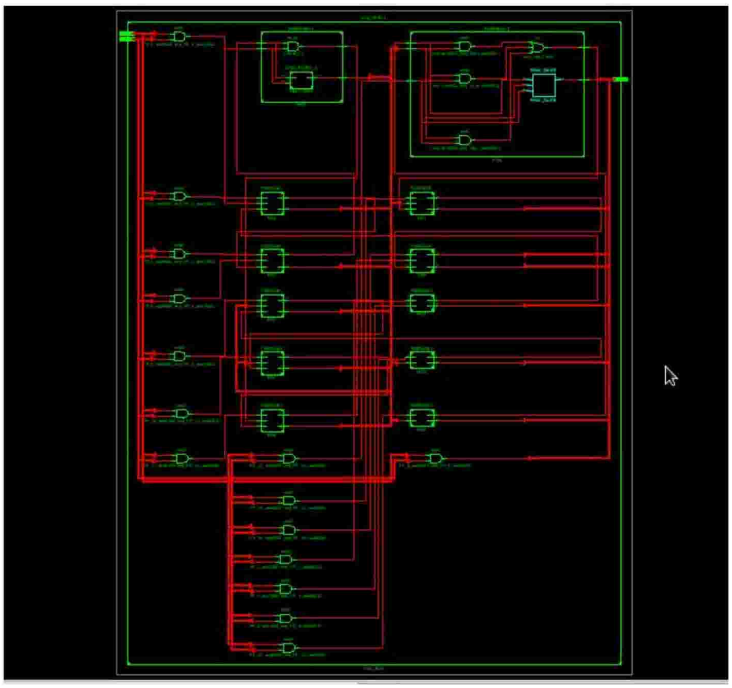

Fig 2.2.6 4-Bit Multiplier RTL From Xilinx Design Implementation

Fig 2.2.7 4-Bit Multiplier Simulation Output From Xilinx Design Implementation

2.2.2 Booth Multipliers

Booth multiplier is kind of multiplier works on basis of booth's algorithm of multiplication that multiplies two signed binary numbers in 2's complement notation. Booth uses desk calculators that were faster at shifting than adding and created the algorithm to increase the speed and performance of the multiplier. This multiplier is used more due to its ease in scaling and faster multiplication. Also, the design is not that complex as compare to fast multipliers.

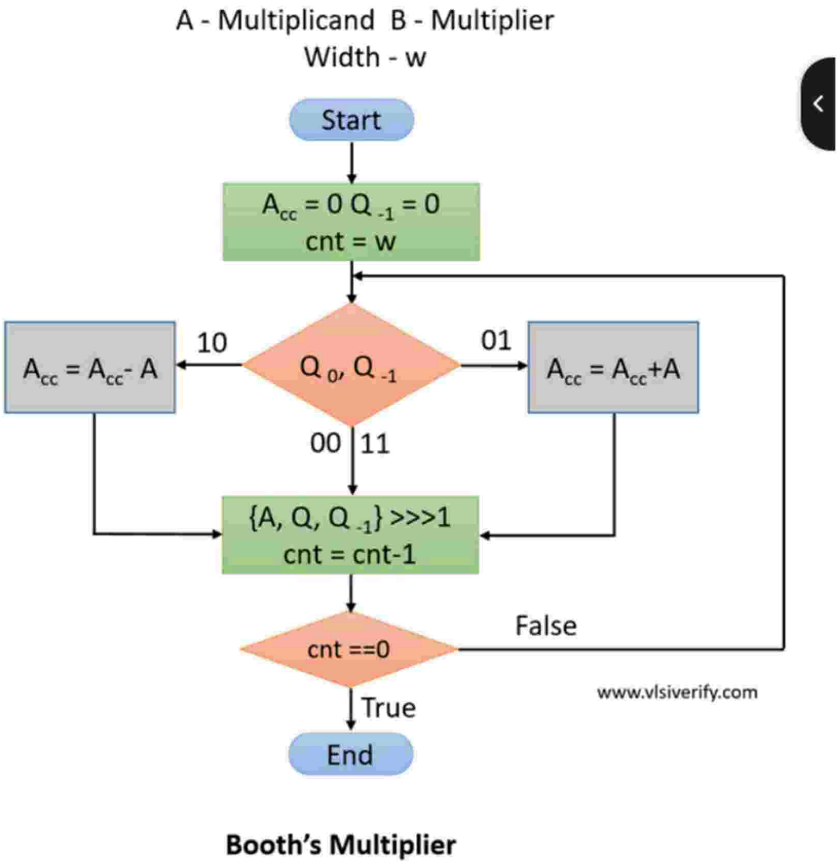

Fig 2.2.8 Booth's Algorithm Flow Chart

Here A is the input operand Multiplicand, B is the multiplier operand. Q register is initially same as B, Q_0 holds the LSB of the Q register. Q_{-1} is a 1 bit variable or register. A_{CC} is Accumulator register for holding result of intermediate addition and substruction. Count holds the maximum width of multiplicand or multiplier.

The disadvantages of booth logic are high power consumption and larger chip area occupation. Even though these disadvantages we use Booth Multiplier in few applications such as Arithmetic units, DSP, Scientific computing and machine learning and neural networks.

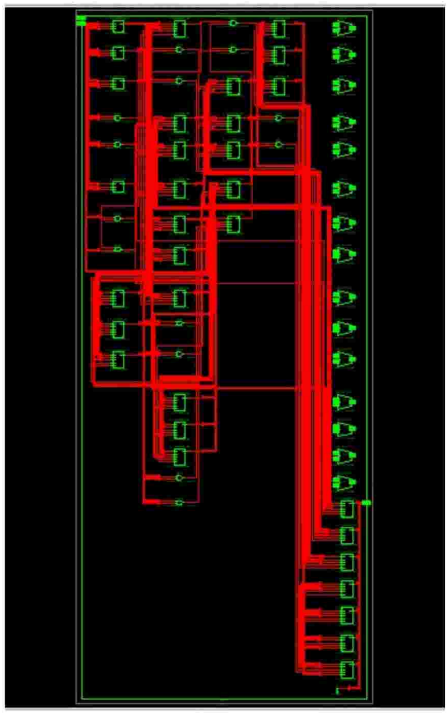

Fig 2.2.9 8-Bit Booth Multiplier RTL From Xilinx Design Implementation

Fig 2.2.10 8-Bit Booth Multiplier Simulation Output From Xilinx Design Implementation

2.2.3 Multipliers Comparison

Table 2.4 Comparison of Multipliers

Parameter	Booth Multiplier	Array Multiplier
Algorithm	Booth's Algorithm	Patial product addition
Basic Operation	Encoding and reducing number of additions	Summing partial products
Handling signed numbers	Efficiently handles signed numbers	Requires separate handling of sign
Number of operations	Reduces operations for sequences of 1's	Number of operations proportional to n^2
Speed	Faster for large no. of consecutive 1's	Slower for large bit-widths
Hardware Complexity	More complex	Simple straight forward
Scalability	More scalable	Less scalable
Power consumption	Lower	Higher
Efficiency	More in certain patterns	Uniform performance
Implementation	More complex logic control	Simple repetitive
Best use case	Signed multiplication large bit-widths	Smaller bit-widths, simpler applications

CHAPTER 3: VLSI PHYSICAL DESIGN FLOW

3.1 INTRODUCTION

After design implementation on Xilinx and having RTL of combinational logic circuits, we require a physical chip or IC fabricated and packaged to use it in day-to-day life. As I discussed earlier in introduction chapter that SoC are designed and fabricated to use in mobile phones. That IC is first having architectural design in three domains of VLSI design flow, physical, behavioral, structural domains. Then the fabrication of IC is done. We till now done design implementation in behavioral domain and structural domain to check whether our thoughts or idea can give the similar output as our requirement. Now we will explore about physical domain. The Geometrical design or Layout design is done in physical domain. This process done in sophisticated software tools called as Electronics Design Automation (EDA) or CAD tools. The software EDA tool we used for physical layout is OpenROAD. Now let us explore more about OpenROAD tool next section.

3.2 INTRODUCTION TO OpenROAD EDA TOOL

The OpenROAD EDA tool is Linux based opensource tool which is used to generate RTL-GDSII layout of Verilog language designed entities or modules. If we talk about OpenROAD tool then it consists of various tool chain for different steps of the RTL-GDSII flow layout planning. This chain of tools is called as OpenROAD-flow-scripts (ORFS). The ORFS consists of Magic, Yosys, OpenSTA, iverilog, gtkwave, klayout, OpenLane, OpenROAD. Each tool has its own use in different stages of RTL-GDSII flow. Likewise Magic and Klayout are for layout planning. Yosys used as logic synthesis tool which synthesizes the raw verilog file to required SDC file for next steps of routing, netlisting and floorplanning of ICs. Earlier stages there was no tool for clock tree synthesis but now we have tools like OpenSTA or OpenCLOCK for static timing analysis and clock tree synthesis. iverilog tool used to have HDL simulations inside the ORFS. gtkwave is waveform viewer. And at last, the main OpenROAD or OpenLane where we call all this tools by its GUI and command prompt window in GUI. Let us see how we can install and use all those tools to design IC from RTL to GDSII.

3.1 Step-by-step Installation Process.

The basic required system for using ORFS is Ubuntu Linux Based System. For latest Ubuntu 24.04 LTS we require at least 8 GB RAM recommended along with 100 GB of free space HDD or SSD and a quad processor with 2.4 GHz clock or better than that. A stable internet connectivity is also recommended probably ethernet or Wi-Fi as software installation is become bad to worst if internet is not working properly and may stuck in between any step of installation process. A dual boot laptop or desktop, or an ubuntu based system is required, do not try, or waste your time trying any other methods of Linux installation in your system such as WSL or VMBox. Simply install ubuntu linux in your system and do the step to install ORFS.

To install Ubuntu 24.04 by your own efforts you should have to create bootable pen-drive of 16 GB, and Then by entering to BIOS setup doing some changes you will be able to do boot in Ubuntu and Windows on single system. After installation of Ubuntu do 2 to 3 times of check of booting and try to install ORFS after 4 to 5 hours of installation of Ubuntu in system, if you using your old Ubuntu system then you can go through the installation steps as given below.

Installation steps for ORFS are as follows

Step 1:

Open the command terminal upgrade, and update the packages and whole ubuntu for new releases may required in further steps by using command **sudo apt-get update, sudo apt upgrade (**reboot recommended after packages upgradation)

Step 2:

Install the git package which will be used to clone GitHub repositories to your system by using the command, **sudo apt install git –assume-yes**

Step 3:

Install the dependencies and build essential by prompting command, **sudo apt install -y build-essential**

Step 4:

Install the Python using following command, sudo apt install -y pyhton3-venv

Step 5:

Next step is to make the default 'sh' to be 'bash' rather than 'dash' you can do it by following command, **sudo rm /bin/sh && sudo ln -s /bin/bash /bin/sh**

Step 6:

After that you should create directory or work area in ubuntu for ORFS by using following command, **mkdir -p Work/vlsi/tools**

Step 7:

Now we will move to tools directory and install Yosys and OpenROAD in that directory. The command should be followed as given, **cd ~/Work/vlsi/tools git clone –recursive https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts**

Step 8:

Setup of OpenROAD and Yosys by following command, **mkdir Open-ROAD-flow-scripts, cd ~/Work/vlsi/tools/OpenROAD-flow-scripts, sudo ./setup.sh**

Step 9:

Installation of OpenROAD and Yosys by following command, **./build_openroad.sh –local (**this step will take about 30-40 minutes to install without any error).

Step 10:

Installation of dependencies of Magic VLSI tool by using following commands, **cd && sudo apt-get update && sudo apt-get install m4 --assume-yes && sudo apt-get install tcsh --assume-yes && sudo apt-get install csh --assume-yes && sudo apt-get install libx11-dev --assume-yes && sudo apt-get install tcl-dev tk-dev --assume-yes && sudo apt-get install libcairo2-dev --assume-yes && sudo apt-get install mesa-common-dev libglu1-mesa-dev --assume-yes && sudo apt-get install libncurses-dev --assume-yes**

Step 11:

Installing Magic VLSI tool by using following command as given below, **git clone https://github.com/RTimothyEdwards/magic.git && mkdir yosys && cd yosys && sudo make && sudo make install** (this process of installation may take 30-45 minutes depending on stability and connectivity of internet you providing.

Step 12:

Installation of the iverilog tool for HDL simulations in ORFS,

sudo apt-get install -y iverilog –assume -yes

Step 13:

Installation of the gtkwave waveform simulation tool in ORFS,

sudo apt install gtkwave –assume -yes

Step 14:

Installation of the Klayout, layout planning toll in ORFS,

sudo apt install klayout –assume -yes

Step 15:

Installation of docker in ubuntu to install the OpenLane by docker method,

sudo apt-get remove docker docker-engine docker.io containerd runc –assume-yes
sudo apt-get update –assume-yes
sudo apt-get install\ apt-transport-https\ca-certificates\curl\gnupg\lsb-release –assume-yes
curl-fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg –dearmor -o/usr/share/keyrings/docker-archieve-keyring.gpg –yes
echo\ “deb [arch=amd6 signed-by=/usr/share/keyrings/docker-archieve-keyring.gpg] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >/ddev/null
sudo apt-get update –assume-yes
sudo apt-get install docker-ce-cli containered.io –assume-yes

Step 16:

Installation of OpenLane tool under the ORFS tool chain,

```
cd
git clone https://github.com/The-OpenROAD-Project/OpenLane.git
mkdir OpenLane
cd OpenLane
sudo make
sudo make install
```

By following above steps, we can install all the required tools for the ORFS tool chain, now we can proceed the step to install the opensource PDK supported by OpenROAD which will provide design constraints and standard cell library for our design implementation. By default, with OpenLane and ORFS we get skywater130 PDK and nangate45 PKD support, for other PDKs such as GF180MCU, SKY90FD, GlobalFoundries are such PDKs.

The installation of the PDK in the system is not that tough you can go through website link given below to install PDKs the link for installation process is: http://www.opencircuitdesign.com/open_pdks/install.html

After installation of all the tools we need to verify either all the tools are successfully installed or not. These current steps support verilog language synthesis only. For implementing and synthesis of VHDL modules in yosys gnat and ghdl, so that we can use VHDL files and parse for synthesis.

3.2 Installation Verification Process.

To verify the installation of ORFS tools the best is to use them directly or simply enter each command one by one by which information of the this ORFS tools or version may be visible to you in terminal or if the version or information is not visible then you must reinstall the respective tool by following respective step mentioned above again. The command to verify each tool installation given in following table.

Table 3.1 Command table for verification

Command	Command Output
yosys -V	Version of yosys installed will be displayed
openroad -version	Version of openroad will be visible
gtkwave -v	gtkwave installed version will be displayed
iverilog -V	iverilog installed version will be displayed
docker --version	docker version will be displayed
opensta	either GUI of opensta or information will be displayed

klayout -v	klayout version will be displayed
magic --version	magic tool installed version will be displayed
./flow.tcl -help (this command should be used by moving to OpenLane directory and then run this command)	This will display the information of OpenLane flow script.
ls $PDK_ROOT/sky130A or ls $PDK_ROOT/pdk_name	This will list the content of PDK directory.

3.3 RTL TO GDSII FLOW

The ORFS has different tools by which we can go through RTL to GDSII flow easily if we provide proper parameters and PDK files. Later, we will discuss about how we can get RTL to GDSII step by step. Now let us understand about what is RTL to GDSII flow. The ORFS is autonomous software but we will use it in manual order step by step to debug in between any error. Now see the flow of RTL to GDSII in following figure.

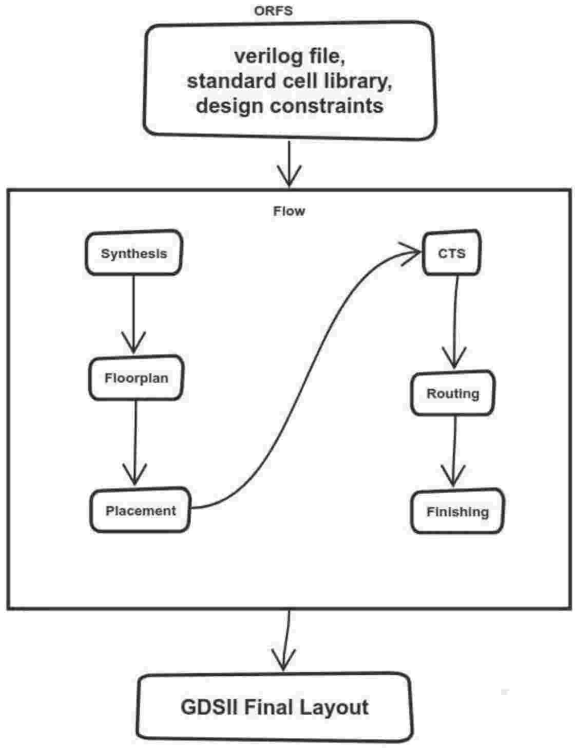

Fig 3.3.1.a RTL to GDSII Design Flow

As per the last figure we need to create a verilog file, and then from PDK we must select any PDK and then use that PDK's library file and design constraint file during the synthesis process through the yosys.

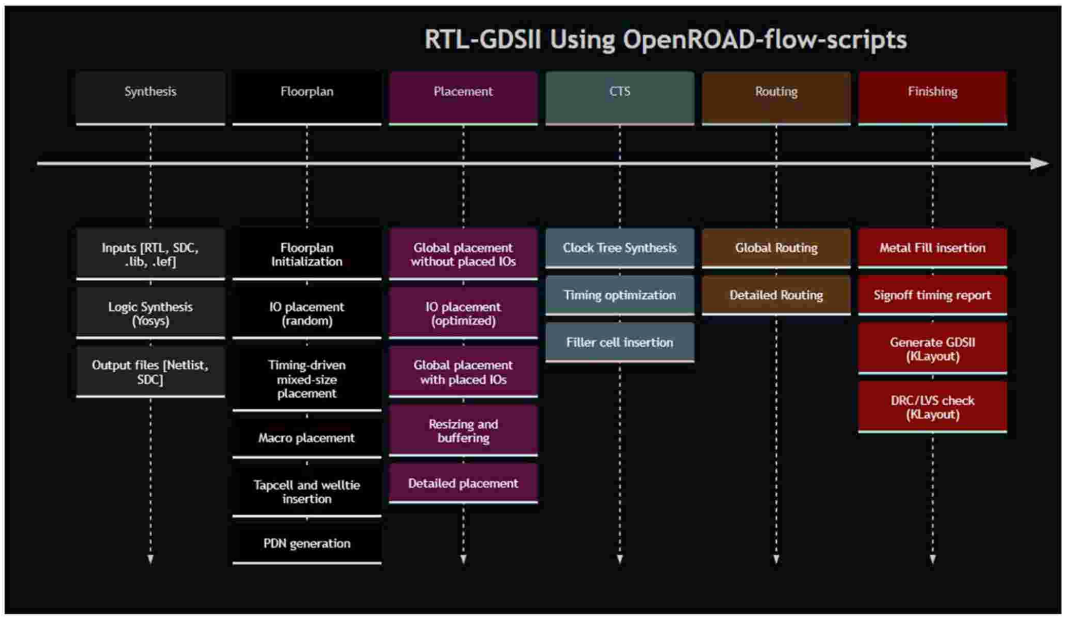

Fig 3.3.1.b RTL To GDSII Flow Using ORFS

Now let us understand about each part of Physical design flow given in figure 3.3.1.b

Synthesis: The process of synthesis is to synthesize the netlist file of the module using selected standard cell library provided by the vendor. The input of the synthesizer used yosys in the ORFS are verilog file and standard cell library which gives output of Gate level netlist and SDC of the designed module.

Floorplan: The Floorplan will initiate the basic area of die and core which will be used to design on actual silicon substrate. It helps in random IO placement which further rectified according to design and tapcell insertion for the design.

Placement: The placement in RTL to GDSII consists of many parts such as Global cell placement, IO placement, Global Placement with placed Ios, Resizing and buffer settings and detailed placement, provides various parameters and organizes the different objects such as IO pins, Buffers placement and give proper distribution.

Clock Tree Synthesis: This is done with help of opensta tool, which is used whenever we have any kind of clock driven part inside the chip and it will help in dividing the clock and proper clock functioning.

Routing: This part helps in interconnections between vias and metals layers of the different layers and interconnections between different blocks of design inside the chip.

Finishing: In general, wherever we using checkpoint, report and repair we are checking for any kind of faults and error in design which is termed as Finishing of design.

Now let us see the step-by-step process of getting RTL to GDSII final layout from a verilog file.

Design realization and implementation by using PDK of vendor nangate45 which support 45nm technology window design, and consist of design constraints and standard cell library for design implementation. The PDK will be provided by the vendor from whom you wished to fabricate your IC, here we using open-source PDK for our designs

Note:- Commands and outputs are in Italic and Bold. Also due to irregularity in sentence width the justify alignment isn't used during the steps as its seems too bad format, as of example you can see the above paragraph ending.

Step 1:To create a verilog file you can use by default text editor and save it in any folder in home directory or if you installed by following our method then you can do it by saving it in tools folder's any new folder, along with .v extension.

Step 2:Open terminal and write command: ***yosys*** .To access the yosys for synthesis.

Step 3:Open your verilog file by using command: ***read_verilog /$file_path/file_name.v***
read_verilog /home/darshit-mehta/run01/CarryLookAheadAdder_8bit.v
In this step we will get output message on terminal as
Generating RTLIL representation for module `\CarryLookAheadAdder_8bit'.
Successfully finished Verilog frontend.

Step 4:Checking, expanding, and cleaning of the design hierarchy , which will be done by following command: ***hierarchy -check -top CarryLookAheadAdder_8bit***
By performing this step you will get the output such that
analyzing design hierarchy..
Top module: \ CarryLookAheadAdder_8bit
Removed unused 0 modules

Note:- The setup we got limitation of setting only one entity module, if your verilog code has more than one entity module then you may get error at previous step or any of the next step. Hence to ensure the least error chances implement your design based in basic gates rather than using more than one entity.

Step 5:Conversion “processes” into multiplexers and registers (the internal representation of behavioral verilog code) by using command: ***proc*** . After this you should wait until any error message shows or proceed to next step if no error message shows.

Step 6:Perform some basic optimizations and cleanups by using following command: ***opt***
This will show message at last that ***Finished OPT passes. (There is nothing left to do.)***
After getting this message you can proceed to next step.

Step 7:Analyze and optimize finite state machine for your design by using following command: ***fsm***
By following this step you get last message as
Executing FSM_MAP pass (mapping FSMs to basic logic). After this repeat Step 6.

Note: From this onward after every step, it is good practice to use command: ***opt*** for performing basic optimization and cleanups.

Step 8:Analyze memories and create circuits to implement them for your design by using following command: ***memory***
This kind of message will be shown in ending if this process
Executing MEMORY_MAP pass (converting memories to logic and flip-flops).

Step 9:Map coarse-grain RTL cells (adders, etc.) to fine-grain logic gates (AND ,OR ,NIT,ETC...) by using following command: ***techmap***
Successfully finished Verilog frontend. This message may be displayed or if you get any error you can restart process or debug it if error is understandable.

Step 10:Map registers to available hardware flip-flop from lib by using following command: ***dfflibmap -liberty /home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.lib***
$/home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45 this is the file path of standard cell library provided by vendor. Wait till the process end and proceed to next step.

Step 11:Map logic to available hardware gates by using the command:
abc -liberty /home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.lib
By performing this step you will message displaying some useful information.
Re-integrating ABC results.
ABC RESULTS: NAND2_X1 cells: 3
ABC RESULTS: AOI21_X1 cells: 3
ABC RESULTS: AND2_X1 cells: 4
ABC RESULTS: OR2_X1 cells: 4

ABC RESULTS: AOI221_X1 cells: 1
ABC RESULTS: NOR2_X1 cells: 5
ABC RESULTS: XOR2_X1 cells: 4
ABC RESULTS: OAI21_X1 cells: 4
ABC RESULTS: XNOR2_X1 cells: 12
ABC RESULTS: internal signals: 115
ABC RESULTS: input signals: 17
ABC RESULTS: output signals: 9
Removing temp directory.

Step 12:Next step is to flattening of design using command: ***flatten***
Executing FLATTEN pass. (flatten design) {This message may be seen}

Step 13:Replacement of undef values with defined contants this is important as if we don't do this then our design may get any error in any next step. The command for this task is : ***setundef -zero***

Step 14:Removal of unused cells and wires is also important to optimize our design, The command for this task is: ***clean -purge***

Step 15:Next step is technological mapping of i/o pads (or buffers) by using the command: ***iopadmap -outpad BUF_X2 A:Z -bits***

Step 16:Use command : ***clean .***To remove unused cells and wires.

Step 17:To do some important calculations such as die area or core area of IC we require some stats this statistics data obtain by using command: ***stat -liberty /home/Darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.lib***
Here "/home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.lib" is path of the standard cell library file, which will give stats related such as no. components, wires, cells, processes and at last the chip area value in micrometers^2 units.
This will determine the chip core area and die area in substrate during fabrication process.
Printing statistics.

=== CarryLookAheadAdder_8bit ===

Number of wires: 38
Number of wire bits: 66
Number of public wires: 5
Number of public wire bits: 26
Number of memories: 0
Number of memory bits: 0

Number of processes: ***0***
Number of cells: ***49***
AND2_X1 ***4***
AOI21_X1 ***3***
AOI221_X1 ***1***
BUF_X2 ***9***
NAND2_X1 ***3***
NOR2_X1 ***5***
OAI21_X1 ***4***
OR2_X1 ***4***
XNOR2_X1 ***12***
XOR2_X1 ***4***

Chip area for module '\CarryLookAheadAdder_8bit': 59.052000

This will be displayed message may vary according to your components.

Step 18:Rename the object in the design is the next step , we can assign short auto-generated names to all the selected wires and cells with private name or use this command: ***rename -enumerate*** to do same.
Overwriting the design to verilog file is also included in this step this is done by using command: ***write_verilog -noattr CarryLookAheadAdder_8bit1_final.v***

Step 19:Now to generate BLIF file or netlist we must use the following command:
write_blif -buf BUF_X2 A Z
CarryLookAheadAdder_8bit1_mapped_withbuf.blif

Step 20:Now we should check the netlist generated and stored in BLIF file by previous step with help of graphviz, it will generate schematic graph of our netlist. This can be done by using command: ***show -stretch***
In this step you will be able to see the netlist as shown in figure.

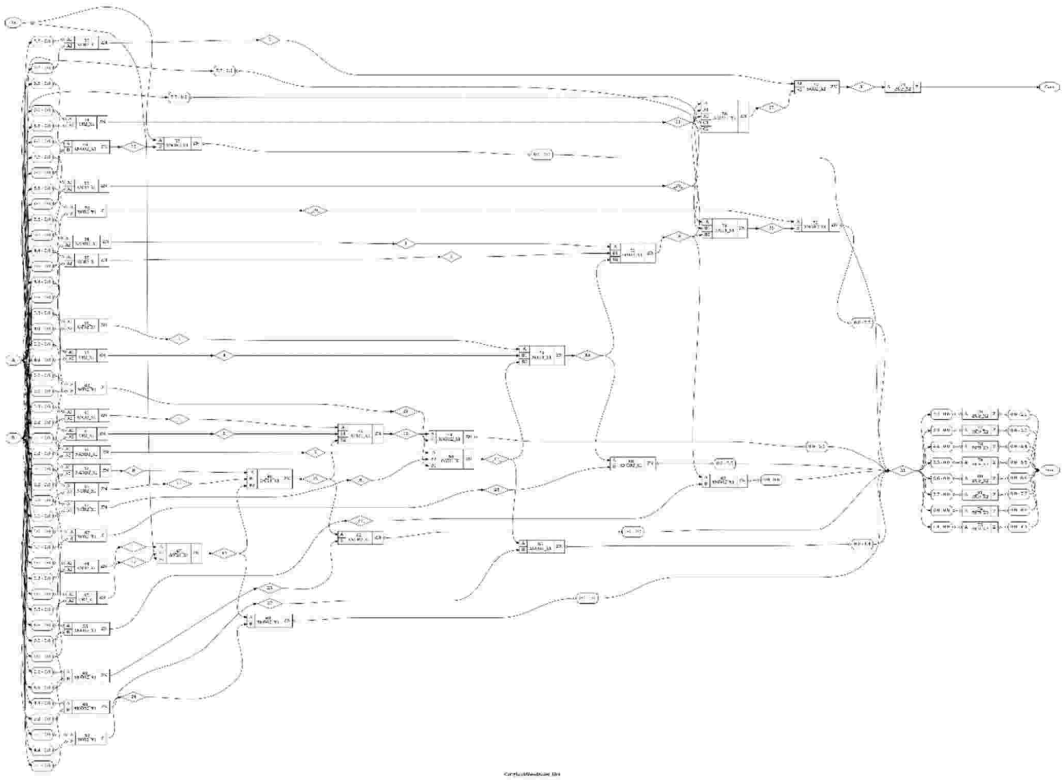

Fig 3.3.2 RTL to Gate Level Netlist of CarryLookAheadAdder_8bit

Step 21:Now you have exit the yosys using command: ***exit*** , and interact with Openroad GUI by using command: ***openroad -gui***

Note: Until this step we were working over yosys the synthesis tool now we must go for further steps to generate final GDS layout using OpenROAD tool of ORFS. The GUI of OpenROAD is command line-based tool hence at each step has different commands, the commands will be same font as until we see in previous steps.

Step 22:Create a SDC file with text editor and give name as SDC and save in .csv extension. This file also in same directory where you have previously saved verilog file. You may notice some changes that verilog file if you did not save the synthesized verilog file in different name. The content of SDC file is
create_clock [get_ports clk] -name core_clock -period 0.4850
set_all_input_output_delays
Now you must save this SDC file and final verilog file in folder with following path, $/home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test . After that you may proceed for further steps.

Note: The chip area will give you idea about estimation of chip area and die area, You can either you can keep same as chip area or round it to nearest ten for sake of simplicity and for larger areas you have to choose area by trial and error method. Although we have a

relation that if u = utilization efficiency of chip, then die area = (chip area * 100) / u, where u ranges between 30% to 90% practically may more different. And according to die area you can set core area.

Step 23:Now we will put following commands in command prompt of openroad GUI.

set design "CarryLookAheadAdder_8bit"
set top_module "CarryLookAheadAdder_8bit"
set synth_verilog "CarryLookAheadAdder_8bit1_final.v"
set sdc_file "CarryLookAheadAdder_8bit.sdc"
#set die_area {0 0 42 42}
set die_area {0 0 42 42}
#set core_area {2.5 2.5 40 45}
set core_area {2.5 2.5 40 45}

After having this command in the command window now you can press enter to set these parameters to the openroad for the design.

Step 24:In this step you have to give command of reading the design files to the openroad by using commands given below,

read_libraries
read_verilog $synth_verilog
link_design $top_module
read_sdc $sdc_file

Step 25:In this step we will having STA network instance count for all other parameters required to generate the final layout, this step will help in creating all other parameters.

utl::metric "IFP::ord_version" [ord::openroad_git_describe]
 # Note that sta::network_instance_count is not valid after tapcells are added.
utl::metric "IFP::instance_count" [sta::network_instance_count]

Step 26:In this step we will initiate process of floorplanning.

***initialize_floorplan -site $site ***
***-die_area $die_area ***
-core_area $core_area

By which you will get floorplan figure.

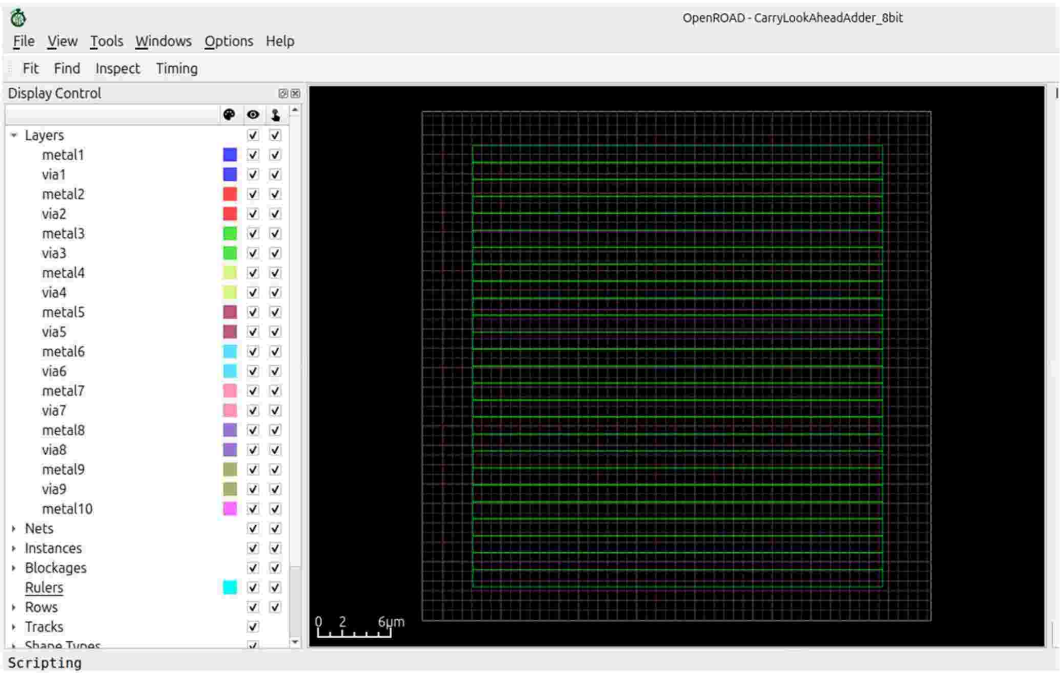

Fig 3.3.3 Floorplan Command Output Window.

Step 27:Now to optimize the design we must remove unused or unrequired buffers by using commands,

source $tracks_file

 # remove buffers inserted by synthesis

remove_buffers

We will get output at output console: ***[INFO RSZ-0026] Removed 16 buffers.***

Step 28:Now next step is random IO placement by using the following commands

 # IO Placement (random)

place_pins -random -hor_layers $io_placer_hor_layer -ver_layers $io_placer_ver_layer

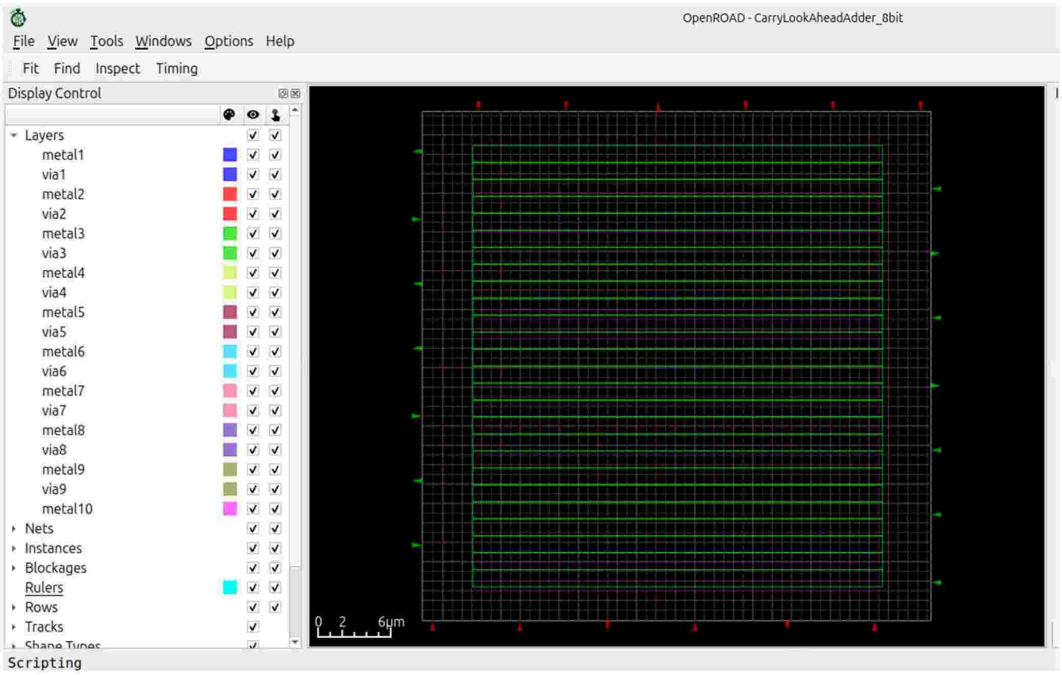

Fig 3.3.4 IO_Placement_Random Command Output Window

Step 29:Next step is macro-placement which can be done by using command,

```
# Macro Placement
if { [have_macros] } {
global_placement -density $global_place_density
macro_placement -halo $macro_place_halo -channel $macro_place_channel
}
```

Step 30:Now our next target is to create tapcell insertion by using following command,

```
# Tapcell insertion
eval tapcell $tapcell_args
```

This will show some significant changes in output window, which shown in Fig 3.3.5

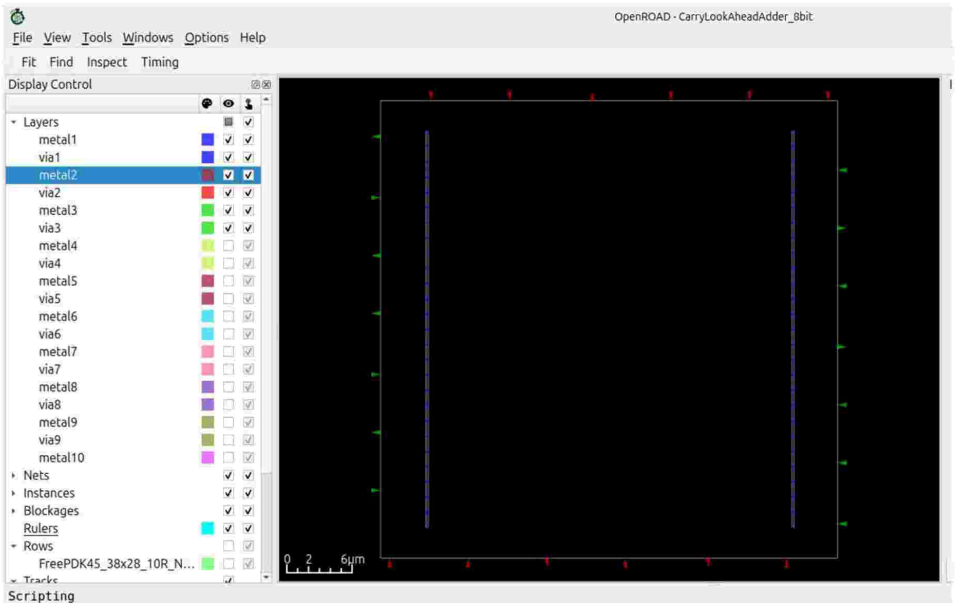

Fig 3.3.5 Tapcell Insertion Command Output Window

Step 31:Insertion of power distribution network using following command,

 # Power distribution network insertion
source $pdn_cfg
pdngen

This command renders some significant changes to output window,

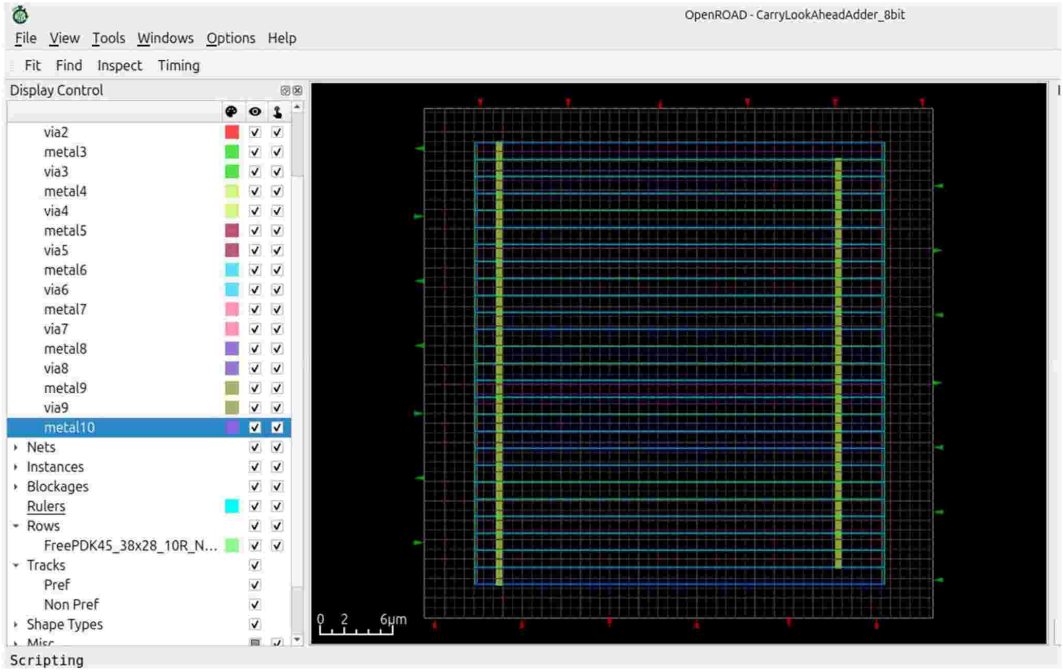

Fig 3.3.6 PDN Command Output Window

Step 32:The next step is of global placement of different parts layers interlinks, we can see some red instances in the chip design area by using following commands for global placement,

 # Global placement
foreach layer_adjustment $global_routing_layer_adjustments {
lassign $layer_adjustment layer adjustment
set_global_routing_layer_adjustment $layer $adjustment}
***set_routing_layers -signal $global_routing_layers ***
-clock $global_routing_clock_layers
set_macro_extension 2
***global_placement -routability_driven -density $global_place_density ***
-pad_left $global_place_pad -pad_right $global_place_pad

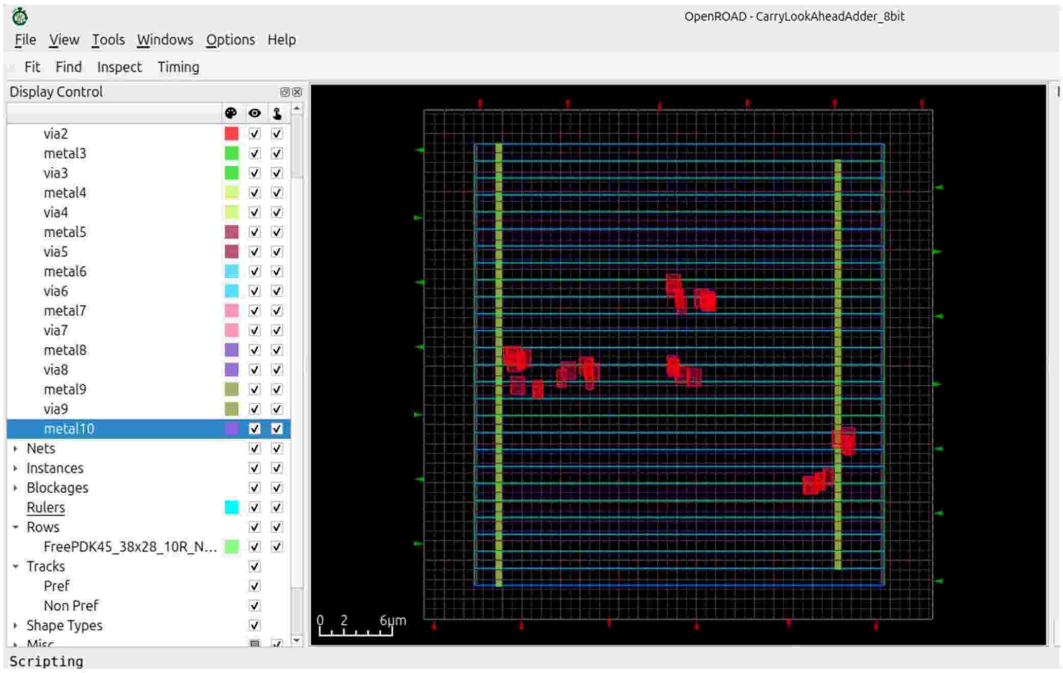

Fig 3.3.7 Global Placement Command Output Window

Step 33:Now we will organize the Input and Output Placements, by using following commands, ***# IO Placement***
place_pins -hor_layers $io_placer_hor_layer -ver_layers $io_placer_ver_layer

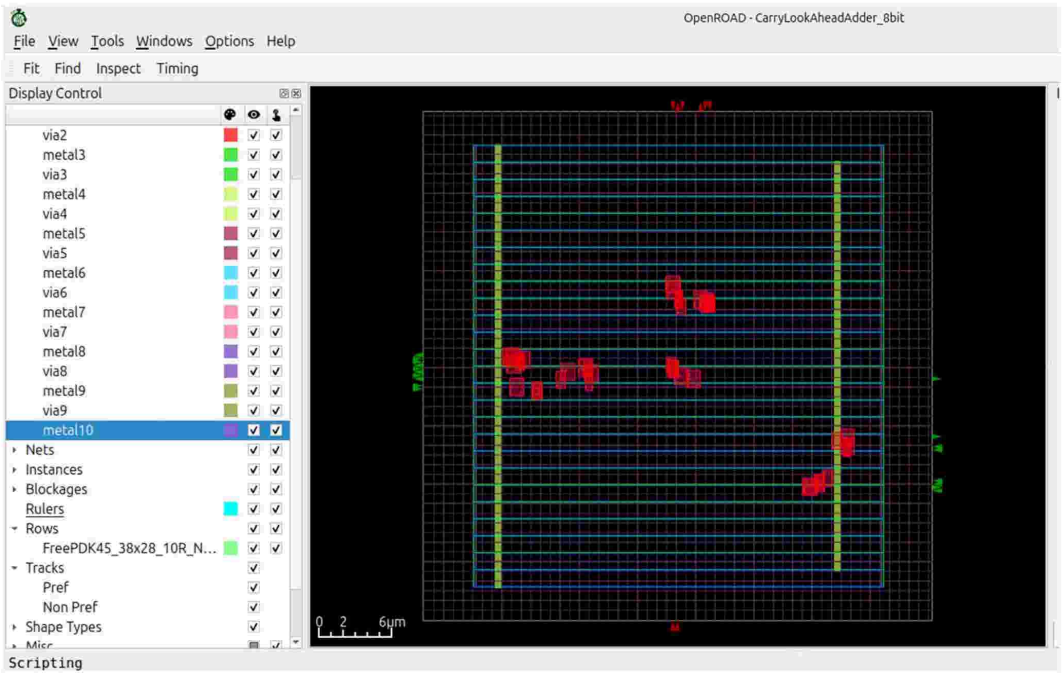

Fig 3.3.8 IO Placement Command Output Window

Step 34:Now we must set a checkpoint by using following command,

```
# checkpoint
set global_place_db [make_result_file ${design}_${platform}_global_place.db]
write_db $global_place_db
```

Step 35:Now we have to optimize and repair the design by maximum of slew, capacitance and fanout violations and normalizing the slew by following commands,

```
# Repair max slew/cap/fanout violations and normalize slews
source $layer_rc_file
set_wire_rc -signal -layer $wire_rc_layer
set_wire_rc -clock  -layer $wire_rc_layer_clk
set_dont_use $dont_use
estimate_parasitics -placement
repair_design -slew_margin $slew_margin -cap_margin $cap_margin
repair_tie_fanout -separation $tie_separation $tielo_port
repair_tie_fanout -separation $tie_separation $tiehi_port
set_placement_padding -global -left $detail_place_pad -right $detail_place_pad
detailed_placement
# post resize timing report (ideal clocks)
report_worst_slack -min -digits 3
report_worst_slack -max -digits 3
report_tns -digits 3
# Check slew repair
report_check_types -max_slew -max_capacitance -max_fanout -violators
utl::metric "RSZ::repair_design_buffer_count"
[rsz::repair_design_buffer_count]
utl::metric "RSZ::max_slew_slack" [expr [sta::max_slew_check_slack_limit] *
```

100]
utl::metric "RSZ::max_fanout_slack" [expr
[sta::max_fanout_check_slack_limit] * 100]
utl::metric "RSZ::max_capacitance_slack" [expr
[sta::max_capacitance_check_slack_limit] * 100]

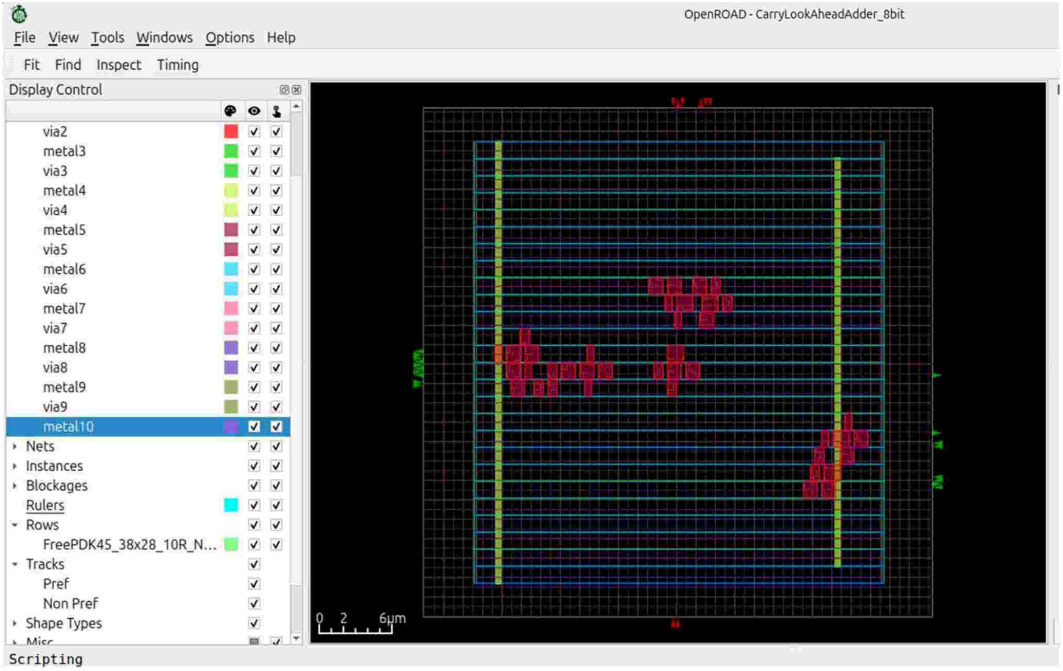

Fig 3.3.9 Repair Design Command Output Window

The console outputs and analysis we get at console window are,
Placement Analysis
----
total displacement 47.8 u
average displacement 0.5 u
max displacement 2.3 u
original HPWL 420.3 u
legalized HPWL 490.7 u
delta HPWL 17 %
worst slack 0.202
worst slack -0.042
tns -0.070

Step 36: Now next part is also important as we do CTS in next part, using following command group,
 # Clock Tree Synthesis
 # Clone clock tree inverters next to register loads
 # so cts does not try to buffer the inverted clocks.
repair_clock_inverters

```
clock_tree_synthesis -root_buf $cts_buffer -buf_list $cts_buffer \
-sink_clustering_enable \
 -sink_clustering_max_diameter $cts_cluster_diameter
# CTS leaves a long wire from the pad to the clock tree root.
repair_clock_nets
# place clock buffers
detailed_placement
# checkpoint
set cts_db [make_result_file ${design}_${platform}_cts.db]
write_db $cts_db
```

This will set buffers and clock buffers and give output of no. of valid clock nets according to design, The most important parameter out by this commands is Using max wire length which is 693 um in our design.

Placement Analysis

---

total displacement	0.0 u
average displacement	0.0 u
max displacement	0.0 u
original HPWL	490.7 u
legalized HPWL	490.7 u
delta HPWL	0 %

Step 37:Now setup and hold timing repair for the chip design in another part of CTS, by using following commands,

```
# Setup/hold timing repair
set_propagated_clock [all_clocks]
# Global routing is fast enough for the flow regressions.
# It is NOT FAST ENOUGH FOR PRODUCTION USE.
set repair_timing_use_grt_parasitics 0
if { $repair_timing_use_grt_parasitics } {
# Global route for parasitics - no guide file requied
global_route -congestion_iterations 100
estimate_parasitics -global_routing
} else {
estimate_parasitics -placement
}
repair_timing -skip_gate_cloning
# Post timing repair.
report_worst_slack -min -digits 3
report_worst_slack -max -digits 3
report_tns -digits 3
report_check_types -max_slew -max_capacitance -max_fanout -violators -
digits 3
utl::metric "RSZ::worst_slack_min" [sta::worst_slack -min]
utl::metric "RSZ::worst_slack_max" [sta::worst_slack -max]
utl::metric "RSZ::tns_max" [sta::total_negative_slack -max]
utl::metric "RSZ::hold_buffer_count" [rsz::hold_buffer_count]
```

Those commands will give rendered output for virtual clock core_clock cannot be

propagated which may differ design by design, and an Placement Analysis as shown in down outputs,

Placement Analysis

----

total displacement 12.4 u
average displacement 0.1 u
max displacement 2.6 u
original HPWL 495.3 u
legalized HPWL 509.7 u
delta HPWL 3 %

Step 38:Let us have now detailed placement of our design using following commands,

```
# Detailed Placement
detailed_placement
# Capture utilization before fillers make it 100%
utl::metric "DPL::utilization" [format %.1f [expr [rsz::utilization] * 100]]
utl::metric "DPL::design_area" [sta::format_area [rsz::design_area] 0]
# checkpoint
set dpl_db [make_result_file ${design}_${platform}_dpl.db]
write_db $dpl_db
set verilog_file [make_result_file ${design}_${platform}.v]
write_verilog $verilog_file
```

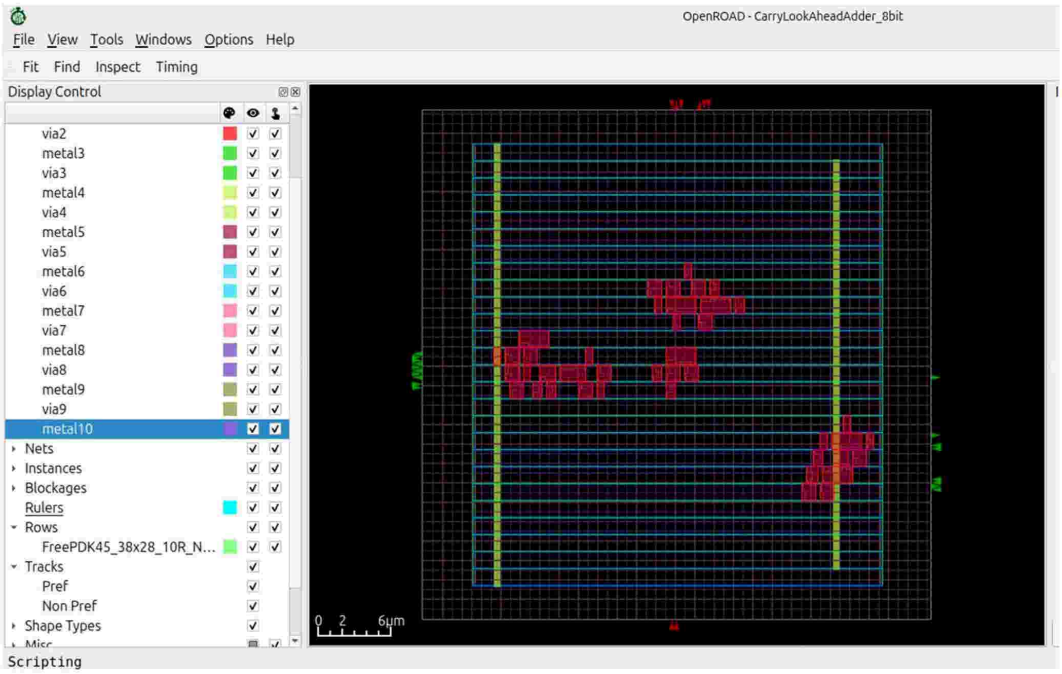

Fig 3.3.10 Detailed Placement Command Window Output

The console will give some outputs parameters of design given as also enlists the default vias layer by layer vias,

Step 39:After all this rendering we now do global routing by using following commands,

```
# Global routing
pin_access -bottom_routing_layer $min_routing_layer \
        -top_routing_layer $max_routing_layer
set route_guide [make_result_file ${design}_${platform}.route_guide]
global_route -guide_file $route_guide \
  -congestion_iterations 100
set verilog_file [make_result_file ${design}_${platform}.v]
write_verilog -remove_cells $filler_cells $verilog_file
```

This command will give some parameters such as,

```
Reading tech and libs
Units:              2000
Number of layers:    21
Number of macros:    135
Number of vias:      30
Number of viarulegen: 19

Reading design.
Design:             CarryLookAheadAdder_8bit
Die area:           ( 0 0 ) ( 84000 84000 )
Number of track patterns: 20
Number of DEF vias:      0
Number of components:    94
Number of terminals:     26
Number of snets:         2
Number of nets:          59
```

And list of vias different metal vias,

This will be also part of global routing report

```
Start pin access.
 Complete 57 pins.
 Complete 17 unique inst patterns.
 Complete 42 groups.
#scanned instances     = 94
#unique  instances     = 21
#stdCellGenAp          = 474
#stdCellValidPlanarAp  = 0
#stdCellValidViaAp     = 335
#stdCellPinNoAp        = 0
#stdCellPinCnt         = 134
#instTermValidViaApCnt = 0
#macroGenAp            = 0
#macroValidPlanarAp    = 0
#macroValidViaAp       = 0
#macroNoAp             = 0
Complete pin access.
```

Step 40:Now we will repair any possible defects of Antenna by using following commands,

 # Antenna repair
repair_antennas -iterations 5
check_antennas
utl::metric "GRT::ANT::errors" [ant::antenna_violation_count]

This will be give output of any net and pin violations, and give output as
Found 0 net violations.
Found 0 pin violations.

Step 41: Now the next part is filler placement in design by using following commands,

 # Filler placement
filler_placement $filler_cells
check_placement -verbose
 # checkpoint
set fill_db [make_result_file ${design}_${platform}_fill.db]
write_db $fill_db
There will be significant changes in output window of design,

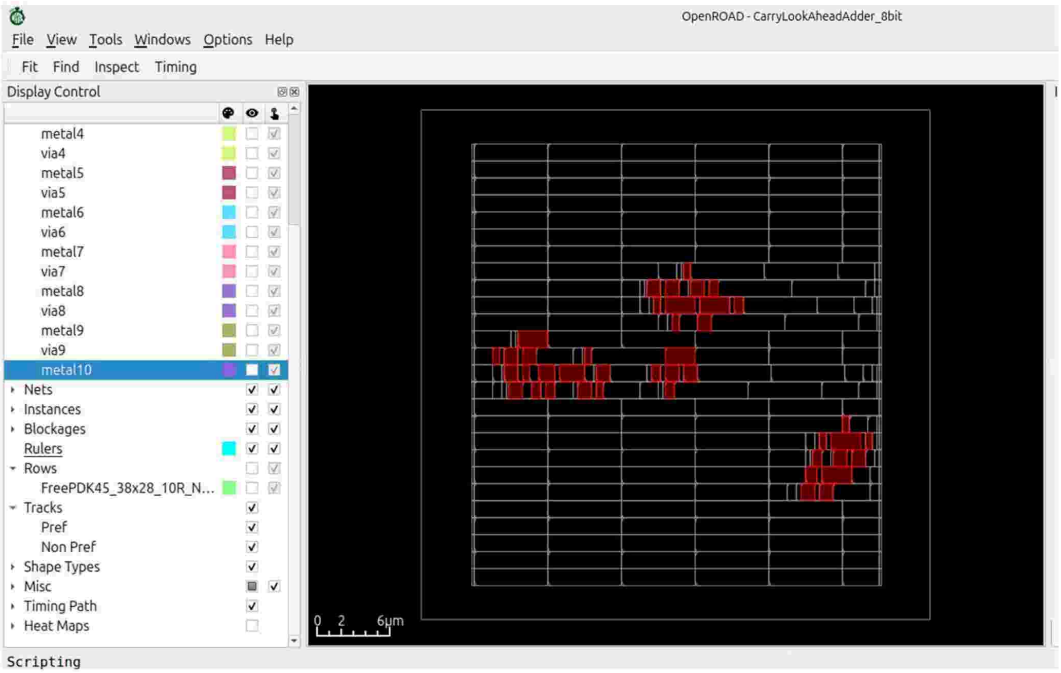

Fig 3.3.11 Filler Placement Command Window Output

Step 42:Now we will be doing detailed routing of our design using following commands,

 # Detailed routing
 # Run pin access again after inserting diodes and moving cells
***pin_access -bottom_routing_layer $min_routing_layer ***
-top_routing_layer $max_routing_layer

```
set_thread_count [exec getconf _NPROCESSORS_ONLN]
detailed_route
-output_drc [make_result_file"${design}_${platform}_route_drc.rpt"] \
-output_maze [make_result_file "${design}_${platform}_maze.log"] \
-no_pin_access \
-save_guide_updates \
-bottom_routing_layer $min_routing_layer \
-top_routing_layer $max_routing_layer \
-verbose 0
write_guides [make_result_file "${design}_${platform}_output_guide.mod"]
set drv_count [detailed_route_num_drvs]
utl::metric "DRT::drv" $drv_count
check_antennas
utl::metric "DRT::ANT::errors" [ant::antenna_violation_count]
set routed_db [make_result_file ${design}_${platform}_route.db]
write_db $routed_db
set routed_def [make_result_file ${design}_${platform}_route.def]
write_def $routed_def
```

This commands will run at instances to provide different output checks and only some parameters which required internally to display design

Step 43:This will help in extraction of SPEF

```
# Extraction
if { $rcx_rules_file != "" } {
  define_process_corner -ext_model_index 0 X
  extract_parasitics -ext_model_file $rcx_rules_file
  set spef_file [make_result_file ${design}_${platform}.spef]
  write_spef $spef_file
  read_spef $spef_file
} else {
  # Use global routing based parasitics inlieu of rc extraction
  estimate_parasitics -global_routing
}
```

These commands will check the give some error checks and final SPEF file for detailed information of interconnections between different components and gates, pins to the chip and design.

Step 44:To get the final report of the design and final design we can use the following commands,

```
# Final Report
report_checks -path_delay min_max -format full_clock_expanded \
  -fields {input_pin slew capacitance} -digits 3
report_worst_slack -min -digits 3
report_worst_slack -max -digits 3
report_tns -digits 3
report_check_types -max_slew -max_capacitance -max_fanout -violators -
digits 3
```

```
report_clock_skew -digits 3
report_power -corner $power_corner
report_floating_nets -verbose
report_design_area
utl::metric "DRT::worst_slack_min" [sta::worst_slack -min]
utl::metric "DRT::worst_slack_max" [sta::worst_slack -max]
utl::metric "DRT::tns_max" [sta::total_negative_slack -max]
utl::metric "DRT::clock_skew" [expr abs([sta::worst_clock_skew -setup])]
# slew/cap/fanout slack/limit
utl::metric "DRT::max_slew_slack" [expr [sta::max_slew_check_slack_limit]
* 100]
utl::metric "DRT::max_fanout_slack" [expr
[sta::max_fanout_check_slack_limit] * 100]
utl::metric "DRT::max_capacitance_slack" [expr
[sta::max_capacitance_check_slack_limit] * 100];
# report clock period as a metric for updating limits
utl::metric "DRT::clock_period" [get_property [lindex [all_clocks] 0] period]
# not really useful without pad locations
#set_pdnsim_net_voltage -net $vdd_net_name -voltage $vdd_voltage
#analyze_power_grid -net $vdd_net_name
```

The final report can be found in results folder of test for all detailed information and parameters if the designed chip, This is the final GDSII layout given in figure below,

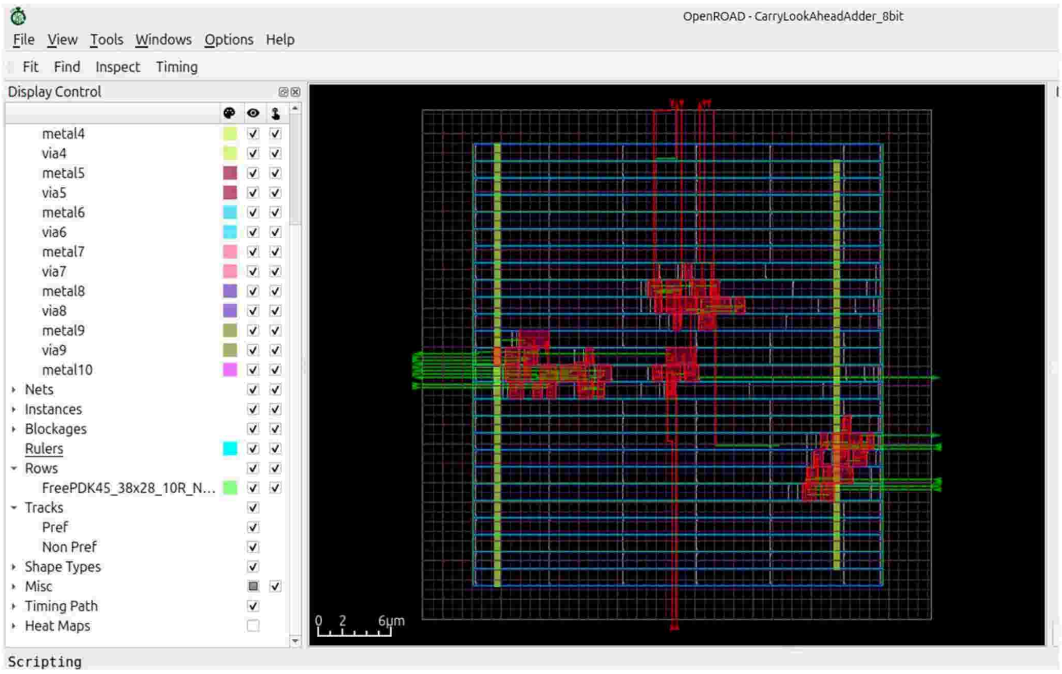

Fig 3.3.12 The Final CarryLookAheadAdder_8bit Design Output

3.4 SOME DESIGN IMPLEMENS

I have designed some designs implemented on basis of the PDK nangate45 which uses 45nm technology.

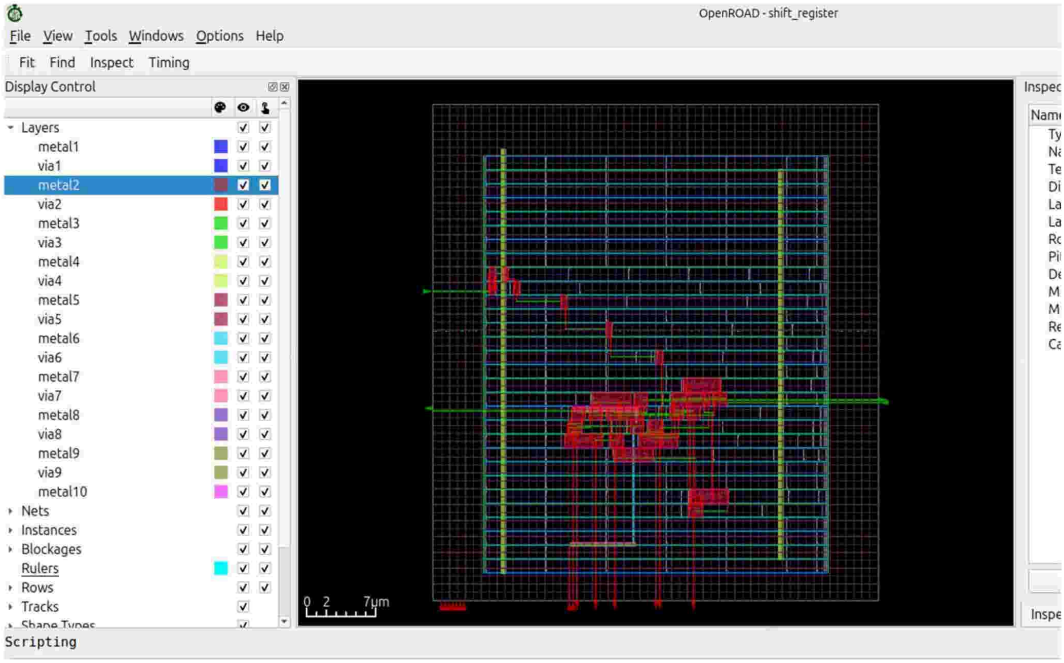

Fig 3.4.1 Shift_Register Final GDSII Layout Design

The CSA design implemented using nangate45 standard cell library, with set die_area {0 0 60 60}, set core_area {5 2 56 60}, max wire length of 693 um.

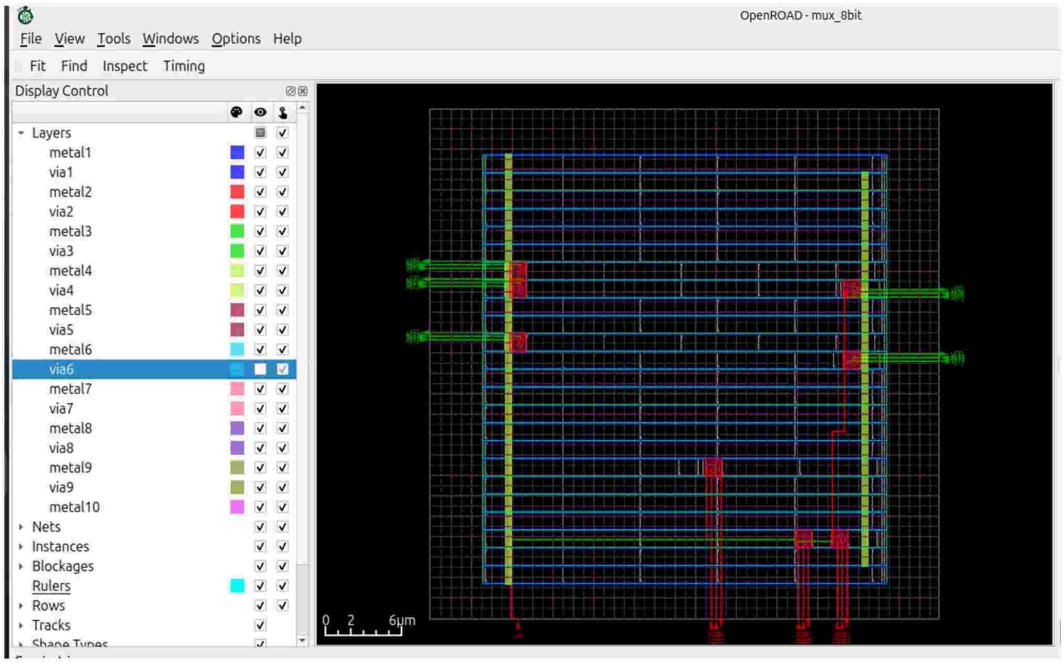

Fig 3.4.2 mux_8bits Final GDSII Layout Design

The RCA design implemented using nangate45 standard cell library, with set die_area {0 0 40 40}, set core_area {4 2 36 37}, max wire length of 693 um.

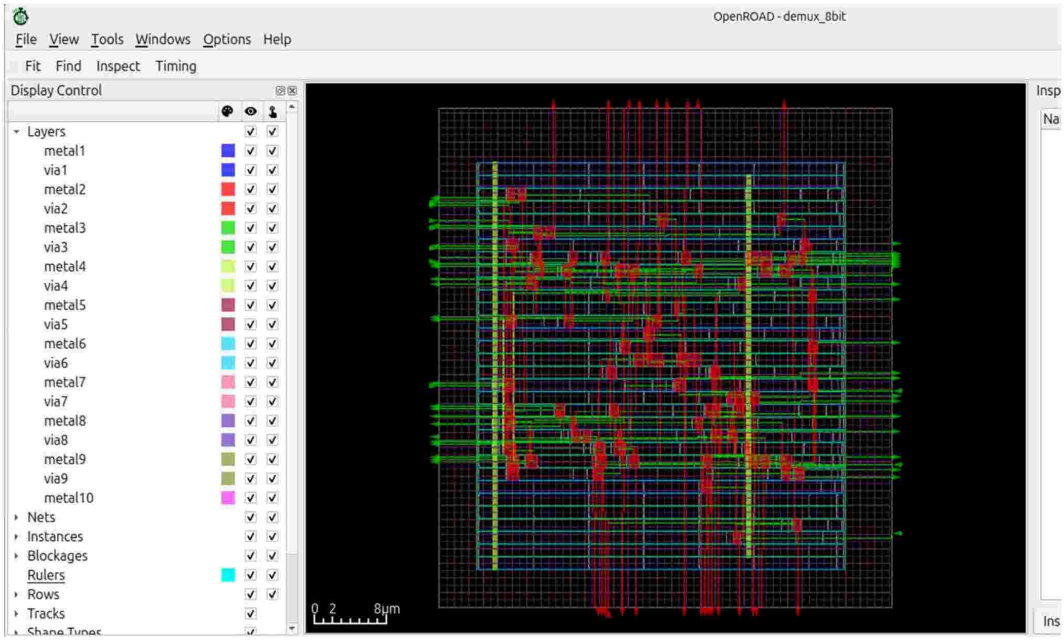

Fig 3.4.3 demux_8bits Final GDSII Layout Design

The 8-bit Multiplier design implemented using nangate45 standard cell library, with set die_area 0 0 50 55}, set core_area {4 3 45 50}, max wire length of 693 um.

CHAPTER 4: SUMMARY

Let me summarize the activities I done during my internship. Initially I was doing design implementations on Xilinx and trying to install ORFS. I had learned about the different kind of advanced adders used in computer architecture also, learned about the multipliers which applied as DSP, and convolutors for Machine Learning and Deep Learning Model architectures. I had explored about the lab facilities available to students and researchers in the SVNIT. The installation ORFS was not easy, I have spent over a week to just get that ORFS on my system. And after getting the software ORFS I explored about how to use it, I got familiar to code hub which is know as github. Before all this I learned how to use ubuntu basics as that software is a opensource hence not available in Windows based system. The ORFS supports many other PDKs also which make its more useful for initial research and design implementations.

References/ Bibliography

As per the guidelines and the references given by GTU, Ahmedabad. I have tried to keep this thesis report / Internship Report as sole creator and writer me, although 100% cannot be done by me hence I have taken references of designs and some terms and important images and parts from the web, knowingly or some others may show in plagiarism report, Some of references used by me knowingly are listed below,

- Invent Logics - Shop Now for Xilinx FPGA development boards (allaboutfpga.com)
- The OpenROAD Project · GitHub
- Online Collaborative Whiteboard | Sketchboard
- VLSI Physical Design with Timing Analysis - Course (nptel.ac.in)
- OpenROAD Flow Scripts Tutorial — OpenROAD Flow documentation (openroad-flow-scripts.readthedocs.io)
- OpenLane using Codespaces : RTL-to-GDSII flow. - Part-2 (youtube.com)
- https://www.youtube.com/watch?v=_AXknRBk4QI
- OpenROAD and efabless OpenLane at the CHIPKIT tutorial, ISCA-2020 (youtube.com)
- And there my be some other similar references can be listed in Plagiarism report which are unknowingly references.

STUDENT'S DAILY DIARY/ LOG

Name of student: Sahu Arabindo K.

Date	Day	Name of completed Task	Work of the day	Sign of the Industry supervisor
29-06-24	Saturday	Installation of xilinix & use of xilinix on a system.	On the day I have given task to install xilinix vivado design suite on my system & learn how to use xilinix to do logic simulation & abstract to RTL design of circuit.	
30-06-24	Sunday	Holiday	—	—
01-07-24	Monday	Ripple Carry Adder	Learning about the RCA logic of binary addition, its logical circuit building & realisation. And done RTL from verilog / VHDL on xilinix ISE.	
02-07-24	Tuesday	Carry Look Ahead Adder	Learning about the CLA logic of binary addition of n-bit width, its logical circuit realisation and RTL from verilog / VHDL on xilinix ISE.	
03-07-24	Wednesday	Carry Save Adder	Learning about the CSA logic of binary addition of n-bit width, its logical circuit realisation and RTL from verilog / VHDL on xilinix ISE.	

SDD-2 (?)

STUDENT'S DAILY DIARY/ LOG

Name of student: Sahu Arabinda K.

Date	Day	Name of completed Task	Work of the day	Sign of the Industry supervisor
04-07-24	Thursday	Fast - Array Multipliers	learning about the fast Multiplier logic of upto 8 bit (4,8,16) width its logical circuit realisation and RTL from verilog /VHDL on xilinx ISE	
05-07-24	Friday	Booth Multiplier	Learning about the booth algorithm of nbit binary multiplication and its logical circuit realisation. And done RTL from verilog / VHDL on xilinix ISE.	
06-07-24	Saturday	Holiday	—	—
07-07-24	Sunday	Holiday	—	—
08-07-24	Monday	Installation of Openroad flow scripts & openroad Tool.	Installing the ORFS, before that making system dual boot as OFFS supported in Unix only. Then installing the ORFS & all tools related to open Road Projectflow scripts.	

SDD-3

STUDENT'S DAILY DIARY/ LOG

Name of student: Sahu Arabindak.

Date	Day	Name of completed Task	Work of the day	Sign of the Industry supervisor
09-07-24	Tuesday	Learning & Exploring about Tools in ORFS	Learning about tools such as Yosys Synthesizer, STA pre and post by OpenSTA, Migration tool, Magic VLSI DRC, LVS checker tools & Learn about the flow of Physical design	
10-07-24	Wednesday	Step by Step Implementation of CLA RTL-To-GDS	Step by Step process mentioned in report, following those Steps ~~the~~ i implemented CLA from verilog code to ~~RT~~ RTL & then RTL to GDS inbetween many processes. Using ORFS & nangate 45 PDK.	
11-07-24	Thursday	Other Design Implementation in ORFS	on this day implemented various other designs such as multiplexer 8:8, demux 8:1 & shift register of 8bit width. using nangate 45 PDK.	
12-07-24	Friday	Presentation & Report Submission.	Presentation & Report Submission for the evaluation of the Internship.	

WPR-1.1

GUJARAT TECHNOLOGICAL UNIVERSITY
(Established under Gujarat Act No. 20 of 2007)
ગુજરાતટેકનોલોજીકલ યુનિવર્સિટી
(ગુજરાત અધિનિયમ ક્રમાંક: ૨૦/૨૦૦૭ દ્વારા સ્થાપિત)

Annexure I

Enrollment no: 210230111008

STUDENT'S WEEKLY RECORD OF INTERNSHIP

NAME OF STUDENT: SAHU ARABINDA

DIARY OF THE WEEK: Dt: 29-06-24 TO 05-07-24

DEPARTMENT: E.C. SEM: 7th

NAME OF THE ORGANISATION: SVNIT, Surat

NAME OF THE PLANT/SECTION/DEPARTMENT: Electronics Engineering Department

NAME OF OFFICER INCHARGE OF THE PLANT/SECTION/DEPARTMENT: Prof. J. Engineer

DESCRIPTION OF THE WORK DONE IN BRIEF

During week-I I am assigned work related to abstract level of logical digital circuits to the RTL of the circuits. During this week I learned about different kind of advanced adders & multipliers used in COA, done some simulations on xilinix related to different Digital circuits, done their timing analysis on paper and go through the process of synthesizing verilog / VHDL code to RTL only.

WPP1.7

GUJARAT TECHNOLOGICAL UNIVERSITY
(Established under Gujarat Act No. 20 of 2007)
ગુજરાતટેકનોલોજીકલ યુનિવર્સિટી
(ગુજરાત અધિનિયમ ક્રમાંક: ૨૦/૨૦૦૭ દ્વારા સ્થાપિત)

TOTAL HOURS: 36 Hrs

SIGNATURE OF STUDENT

- The above entries are correct and the grading of work done by Trainee is EXCELLENT / VERY GOOD / GOOD / FAIR / BELOW AVERAGE / POOR

Signature of Faculty Mentor

Signature of officer-in-charge of Dept. / Section / Plant

Date: 2

Date:

- Grading of Work, for trainee may be given depending upon your judgement about his Punctuality, Regularity, Sincerity, Interest taken, Work done etc.

WPR1.3

GUJARAT TECHNOLOGICAL UNIVERSITY
(Established under Gujarat Act No. 20 of 2007)
ગુજરાતટેકનોલોજીકલ યુનિવર્સિટી
(ગુજરાત અધિનિયમ ક્રમાંકઃ ૨૦/૨૦૦૭ દ્વારા સ્થાપિત)

SUPPLEMENTRY NOTES
(add additional sheets if required)

WPR 2.1

GUJARAT TECHNOLOGICAL UNIVERSITY
(Established under Gujarat Act No. 20 of 2007)
ગુજરાતટેકનોલોજીકલ યુનિવર્સિટી
(ગુજરાત અધિનિયમ ક્રમાંક: ૨૦/૨૦૦૭ દ્વારા સ્થાપિત)

Annexure I

Enrollment no: 210230111008

STUDENT'S WEEKLY RECORD OF INTERNSHIP

NAME OF STUDENT: SAHU ARABINDA KAILASH

DIARY OF THE WEEK: Dt: 06-07-24 TO 12-07-24

DEPARTMENT: E.C. SEM: 7th

NAME OF THE ORGANISATION: SVNIT, SURAT

NAME OF THE PLANT/SECTION/DEPARTMENT: Electronics Engineering Department

NAME OF OFFICER INCHARGE OF THE PLANT/SECTION/DEPARTMENT: Prof B. J. Engineer

DESCRIPTION OF THE WORK DONE IN BRIEF

During week-2 I was instructed to Install ORFS & its tools in my system to do RTL to GDS of physical Design flow. In this week I learned about the physical design flow the IC designing & Implemented some designs such as CLA, shift registers, mux 1:8 & demux 8:1 mentioned in report & some small designs just to understand the ORFS design tool.

WPR.2.7

GUJARAT TECHNOLOGICAL UNIVERSITY
(Established under Gujarat Act No. 20 of 2007)
ગુજરાતટેકનોલોજીકલ યુનિવર્સિટી
(ગુજરાત અધિનિયમ ક્રમાંક: ૨૦/૨૦૦૭ દ્વારા સ્થાપિત)

TOTAL HOURS: 30 hours

Arabinda K
SIGNATURE OF STUDENT

- The above entries are correct and the grading of work done by Trainee is EXCELLENT / VERY GOOD / GOOD / FAIR / BELOW AVERAGE / POOR

Signature of Faculty Mentor

Signature of officer-in-charge of Dept. / Section / Plant

Date:

Date:

- Grading of Work, for trainee may be given depending upon your judgement about his Punctuality, Regularity, Sincerity, Interest taken, Work done etc.

UPR 2.3

GUJARAT TECHNOLOGICAL UNIVERSITY
(Established under Gujarat Act No. 20 of 2007)
ગુજરાતટેકનોલોજીકલ યુનિવર્સિટી
(ગુજરાત અધિનિયમ ક્રમાંક: ૨૦/૨૦૦૭ દ્વારા સ્થાપિત)

SUPPLEMENTRY NOTES
(add additional sheets if required)

GUJARAT TECHNOLOGICAL UNIVERSITY
(Established under Gujarat Act No. 20 of 2007)
ગુજરાતટેકનોલોજીકલ યુનિવર્સિટી
(ગુજરાત અધિનિયમ ક્રમાંક: ૨૦/૨૦૦૭ દ્વારા સ્થાપિત)

Annexure 2

Feedback Form by Industry expert

Student Name: Satu Arabinda Kailash Date: 15-07-2024

Work Supervisor: Prof P. J. Engineer Title:

Company/Organization: Sardar Vallabhbhai National Institute of Technology

Enrollment No: 21023011 008

Internship Address: SVNIT, Electronics deparment, Ichchhanath, Keval chowk, Surat

Dates of Internship: From 29-06-2024 to 02-07-24

Please evaluate your intern by indicating the frequency with which you observed the following behaviors:

Parameters	Needs improvement	Satisfactory	Good	Excellent
Shows interest in work and his/her initiatives				✓
Produces high quality work and accepts responsibility				✓
Uses technical knowledge and expertise				✓
Analyzes problems effectively			✓	
Communicates well and writes effectively				✓

Overall performance of student intern: (Needs improvement/ Satisfactory/Good/Excellent):

Excellent

Additional comments, if any:

Signature of Industry person with name and Stamp:

Signature of the Faculty Mentor

Evaluation Rubrics

List of Documents to be prepared for Submission:

- Detail report duly signed and approved by the internal/external mentor
- Presentation softcopy approved by the internal/external mentor
- CD – Scan copy of report and presentation

Student Details	
Enrollment Number	210230111 0008
Student Name	SAHU ARABINDA KAILASH
Branch	ELECTRONICS & COMMUNICATION
Code and Name of the Institute	023 - Dr. S. & S.S. GHANDHY GOVERNMENT ENGINEERING COLLEGE, SURAT
Mentor Details	Name: Dr. Pinalkumar J. Engineer Designation: Mobile No: 9426148051
Mode of Internship Carried Out	Online / ~~Offline~~
Title of the Project/ Internship carried out	VLSI Physical Design
Nature of Work Carried Out	Small fabrication / experimental results/ simulations/ Application development / Design and / or Analysis of System(s) etc... Other please Specify_____

Evaluation Rubrics

Enrollment No: 210230111008 **Branch:** E. C.

Name of the Students: SAHU ARABINDA KAILASH

Date of Evaluation: _____

Internal Evaluation – 20 Marks PA(I)
(To be carried out by the mentor in consultation with Industry)
Minimum Passing Marks: 10

Parameter	Excellent	Very Good	Good	Not up the level of Satisfaction	Obtained Marks
Mark range	10	09-08	07-05	Below 5	
Student regularity during the Internship period and proactiveness/responsiveness towards the given tasks **10 marks**	10				10
Quality of the prepared report **10 marks**	10				10
Total Marks Obtained Out of 20 PA(I)					20

External Evaluation – 80 Marks ESE(V)
Minimum Passing Marks: 40

Parameter	Excellent	Very Good	Good	Average	Not up the level of Satisfaction	Obtained Marks
Mark range	20-17	16-15	14-12	11-10	Below 10	
Adequacy and Quality of the Work carried Out **(20 Marks)**						
Tools and Techniques Used and to achieve the objectives of the work **(20 Marks)**						
Work Plan and Execution and Outcome achieved **(20 Marks)**						
Quality of the report and presesentation Skill **(20 Marks)**						
Total Marks Obtained Out of 80 ESE(V)						

External Examiner Name: P. J Engineer **External Examiner Name:** _____

Signature: _____ **Signature:** _____

Content Checked For Plagiarism:

VLSI Physical Design Offline Internship Report Team ID: 587579 Report Submitted by SAHU ARABINDA KAILASH EN. NO: - 210230111008 In partial fulfilment for the award if the degree of Bachelor of Engineering in Electronics and Communication Engineering Dr.S. & S.S. Ghandhy Government Engineering College, Surat -5651423241043097450 Gujarat Technological University, Ahmedabad June-July 2024 1042386250 Dr.S Ghandhy Government Engineering College, Surat Near Vanita Vishram Swimming Pool, Majura Gate, Ring Road,Surat -395001 Certificate Date: This is to certify that the project report submitted along withthe project entitled VLSI Physical Design has been carried out by Sahu Arabinda Kailash undermy guidance in partial fulfilment for the degree of Bachelor of Engineering in Electronics and Communication Engineering, 7th Semester of Gujarat Technological University, Ahmedabad during the academic year 2024-25.Prof. P. V. Pithadiya Prof.T. P. Dave (Internal Guide)(Head of Department) Letter head certificate signed by Internship mentorDECLARATION OF ORIGINALITY I hereby declare that the Internship report submitted along with the Internshipentitled VLSI Physical design submitted in partial fulfilment for the degree of Bachelor of Engineeringin Electronics and Communication engineering to Gujarat Technological University, Ahmedabad, is a bonafide record oforiginal project work carried out by me at SVNIT under supervision of Dr.

Pinalkumar J.Engineer and that no part of this report has been directly copied from any students reports or taken from any other source, without providing due reference. Date: Place: Surat SAHU ARABINDA KAILASH .. 210230111008 ACKNOWLEDGMENT This is the place to admit that while there appears onlyauthor on the cover, this work just as any other, is a product of theinteraction with and support during our internship work, among them, first I express my gratitudeto my guides Dr.Engineer & Prof.Dharmesh J.Patel for their affection throughout guidance, advice, and encouragement.Special thanks to my college for giving me the invaluable knowledge.Also, I interacted with some other students having their study and research there for guidance of software installation, I am grateful to them also for providing us guidance.Above all I am thankful to almighty God for everything.SAHU ARABINDA KAILASH Table of Contents LIST OF FIGURESI LIST OF TABLESIII LIST OF ABBREVIATIONSIV ABSTRACTVI CHAPTER 1: INTRODUCTION 1 1.1 INTERNSHIP DESCRIPTION1 1.

2 SCOPE AND OBJECTIVE OF INTERNSHIP2 1.2.1 Objectives3 1.2.2 Scope of Internship3 CHAPTER 2: DESIGN ON XILINX3 2.1 ADVANCED ADDER DESIGN3 2.1.1 Ripple Carry Adder3 2.1.2 Carry Look Ahead Adder5 2.1.3 Carry Save Adder8 2.1.4 Comparison of Adders 10 2.2 MULTIPLIER DESIGNS11 2.2.1 Array Multiplier 12 2.2.2 Booth Multiplier 18 2.2.3 Comparison of Multipliers 20 CHAPTER 3: VLSI PHYSICAL DESIGN FLOW 21 3.1 INTRODUCTION 21 3.2 INTRODUCTION TO OpenROAD EDA TOOL 21 3.2.1 Step-by-step Installation Process 22 3.

2.2 Installation Verification Process 25 3.3 RTL TO GDSII FLOW 26 3.4 SOME DESIGNS IMPLEMENTS 46 CHAPTER 4: SUMMARY49 BIBLIOGRAPHY 50 LIST OF FIGURES Fig no Figure Title Page no.2.1.1 8-Bit RCA RTL From Xilinx Design Implementation 4 2.1.2 8-Bit RCA Simulation Output From Xilinx Design Implementation 5 2.1.3 8-Bit CLA RTL From Xilinx Design Implementation 7 2.1.4 8-Bit CLA Simulation Output From Xilinx Design Implementation 8 2. 1.5 8-Bit CSA RTL From Xilinx Design Implementation 10 2. 1.6 8-Bit CSA Simulation Output From Xilinx Design Implementation 10 2.2.1 2-Bit Multiplier RTL From Xilinx Design Implementation 13 2.2.2 2-Bit Multiplier Simulation Output From Xilinx Design Implementation 13 2.2.3 3-Bit Multiplier RTL From Xilinx Design Implementation 14 2.2.4 3-Bit Multiplier Simulation Output From Xilinx Design Implementation 15 2.

2.5 4-Bit Multiplier Multiplication Method 15 2.2.6 4-Bit Multiplier RTL From Xilinx Design Implementation 17 2.2.7 4-Bit Multiplier Simulation Output From Xilinx Design Implementation 17 2.2.8 Booth's Algorithm Flow Chart 18 2. 2.9 8-Bit Booth Multiplier RTL From Xilinx Design Implementation 19 2.2.10 8-Bit Booth Multiplier Simulation Output From Xilinx Design Implementation 19 3.3.1.a RTL to GDSII Design Flow 26 3.3.1.b RTL To GDSII Flow Using ORFS 27 3.3.2 RTL to Gate Level Netlist of CarryLookAheadAdder_8bit 32 3.3.3 Floorplan Command Output Window 34 3.3.4 IO_Placement_Random Command Output Windows 35 3.3.5 Tapcell Insertion Command Output Window 36 3.3.6 PDN Command Output Window 36 3.3.7 Global Placement Command Output Window 37 3.3.8 IO Placement Command Output Window 38 3.

Adder E.C. Electronics and Communication nm nano meter FET Field Effective Transistor TSMC Taiwan Semiconductor ManufacturingCompany GAA Gate-All-Around AI Artificial Intelligence ASIC Application Specific Integrated Circuit SoC System-on-Chip EDA ElectronicDesign Automation GTU Gujarat Technological University DGGEC Dr.S. & S.S. Ghandhy Government Engineering College ALU Arithmetic Logic Unit VHDL Very High-Speed Integrated Circuit HardwareDescription Language LSB Least Significant Bit MSB Most Significant Bit DSP Digital Signal Processing EDA Electronic Design Automation CAD Computer-aided Design ORFS OpenROAD-flow-scripts SDC Synopsys Design Complier GUI Graphic UserInterface HDL Hardware Description Language WSL Windows Subsystem Linux VMBox Virtual Machine Box PDK ProcessDesign Kit CTS Clock Tree Synthesis BLIF Berkely Logic Interchange Format PDN Power Distribution NetworkSDC Synopsys Design Constraints SPEF Standard Parasitic Exchange Format um micro meter ABSTRACT The mainobjective of the internship program is to learn about the RTL to GDS Design flowof Digital circuits to form an IC.In the journey of this learning, we learned about advanced adders and multipliers and some other components.Also, during the program, we learned about opensource tools and software related to

ASIC designing process.The SVNIT has great environment and guidance for learnings.The internship is focused to the vlsi physical design flow in digital field.In duration of these internship, I learned a lot about Physical design aspect of VLSI and looking forward for VLSI ASIC design related works and projects.CHAPTER 1: INTRODUCTION In the E.C Engineering various Electronics devices are used such as resistor, transistors, diodes, and ICs.Among those devices IC are game changer technology.And all this possible due to semiconductors.Such as Silicon, graphene, etc.are generally used to form IC.The current IC technology window reached is of 5-3nm only.Further research to narrow the channel length of FETs is going on.The SoC such as Tensor G3 used in Google Pixel 8 Series, M3 usedin latest MacBook, Exynos 2400 in galaxy S24 series, Qualcomm's Snapdragon S gen 3 inSamsung galaxy S24 ultra are one of the latest technology-based ASICs designed by foundries such

as TSMC, GAA, and Samsung.Recently Nvidia become first company to research on IC based AI neural engines and render engines.All this discussion to ensure that our internship program for VLSI physical design flow is to learn about designing of ASICs.In most of Indian ASIC based companies are service based such as designing or testing and verification is carried out.I chose this topic of internship to explore about ASIC and RTL-to-GDS design flow. INTERNSHIP DESCRIPTION The Summer Internship of 2 weeks is a part of GTU academics extra curriculum activity allowing students to get hands on experience before placement drives in campus.This report outlines the experiences and accomplishments during the internship period at the VLSIResearch Lab in Electronics Engineering Department of SVNIT, under the mentorship and supervision of Dr.Engineer, Associate Professor at SVNIT. Duration of internship as mentioned by GTU Notification 29th June 2024 to 12th July 2024.During this period in 11 working days, the internship focused on the VLSI physical design flow in digital domain, providing exposure in implementing various digital circuits as RTL-to-GDS design such as RCA, CLA, CSA, Array Multipliers etc.VLSI technology is pivotal in the enhancement or advancement of modern electronics, enabling the creation of complex IC by combining thousands or millions of transistors onto a single chip.The physical design phase in VLSI involves conversion of a high-level abstract design into a detailed manufacturable layout.

This phase is crucial as it directly impacts the performance, power consumption, heat dissipation, and overall efficiency of IC.SCOPE AND OBJECTIVE OF INTERNSHIP 1.2.1 Objectives The primary objectives of internship were: To gain practical knowledge of the VLSI physical design flow.To implement and optimize physical designs for different types of digital circuits.To understand the tools and methodologies used in the VLSI design flow.1.2.2 Scope of internship During the internship the below mentioned tasks were undertaken: Implementation of different advanced adders and multiplier circuits in Xilinx tool for RTL and Simulation.Designing and Optimization ofvarious integrated circuits.Use of opensource EDA tools for layout design, verification, and optimization.This internship provided valuable insights into the intricacies of VLSI physical design, bridging the gap between theoretical knowledge and practical application.The experience gained has equipped me with the skills and understanding necessary to contribute effectively to the field of VLSI design.CHAPTER 2: DESIGN ON XILINX The initial stage of VLSI physical design flow is implementing the design module and verify the simulation output.For this we can create files such as Verilog file with ".v" extension, VHDL modules with ".vhd" file extension and ".

tb" file extension for test bench.Those files are used to form the RTL, Schematics and Verification by simulation of logics with test vectors at Test bench.Apart from general digital circuits we learned in VLSI design subject and DSD subject of GTU syllabus I learned about multipliers and Advanced 8 bit and extensible digital circuit designs.ADVANCED ADDER DESIGNS In advanced adder architecture i learned about 3 kinds of adders used in forming ALUs and intermediate stage of multipliers used in ALU.The 3 kinds of Adder design I learned are 1) Ripple Carry Adder, 2) Carry Save Adder, 3) Carry Look Ahead Adder.Those adders have their specific application and use cases based on their design parameters such as size, delay, performance for n bits architecture and area occupied on chip etc.I have designed and implemented as per given instructions using VHDL programming and of 8 bits architecture for all these kinds. Let's see more about different adder designs in further sections.Ripple Carry Adder (RCA) A Ripple Carry Adder is digital binary adder circuit designed to two n-bit (n no.of bits in binary number).It is the extension to basic RCA concept, which scaled to handle any no of bits.The main building block of the RCA consists of Full Adders.As we know the Full Adder has three inputs as two operands and Carry input, and two outputs as Sum and Carry out.An n-bit RCA consists of chain of n Full adders (i.e. an 8-bit RCA consists of 8 Full Adder blocks.

The RCA provides a n-bit Sum output and Carry output based on provided 2 n-bit operands and carry in signal usually zero (0).A n-bit RCA does bit-wise addition of two operands using full adders, where thesum output of full adder will be summing output of corresponding bit and carry forwardedto next stage of full adder carry input.The term "Ripple" in RCA refers to the way of the carry propagation through the adders.The Carry out from the LSB adder ripples through the MSB of adder.This ripple propagation of carry causes delay proportional to the number of bits, making the adder slow for large width of bits.Whereas generally when we use an 8-bit RCA, and for addition of 16-bit operands we use RCA 2 times to save the area and provide performance.It is comparatively slower thanCSAandCLAbut design wise simplest among all other advanced n bit adders.Also, the hardware area is minimal as it comprises of series of full adders only.The general time delay of Full Adders are as follows Tsum for sum calculation typically involves XOR gate, and Tcarry for carry generation typically involves AND and OR gates.

Now, for the time delay calculation of n-bit RCA, consists of n full adders.Each stage carry output propagated after Tcarry delay hence n times Tcarry (i.e. n * Tcarry).The sum output of corresponding bits available only after carry input of previous stage.The MSB calculated after n-1 bit's carry out generation, hence we can say that for an n-bit RCA the time delay is given by following equation Trca = n * (Tcarry+ Tsum) 2.1.1 Assuming Tcarry = Tsum, for 8-bit RCA we get delay of 16-units time.871763211454 Fig 2.1.1 8-Bit RCA RTL From Xilinx Design Implementation I have implemented 8-bit RCA during the internship also, implemented 4-bit design but showing results for 8-bit RCA only.Here I have used the generate function in VHDL so by which in RTL view we can see that there is only one full adder block is being shown.Fig 2.1.2 8-Bit RCA Simulation Output From Xilinx Design Implementation RCA is basically used in simple arithmetic units where speed isn't the primary concern, for learning and educational purposes due toits basic simple design and used in those embedded systems where low power consumption andsimplicity are more important parameters than speed.Carry Look Ahead Adder (CLA) Carry Look Ahead Adder is another kind of binary adder which is used in addition of more than 2-bit two operands.

In comparison of RCA, it is faster but design is way more complex than that of RCA.The delay due to sequential carry propagation in RCA, is resolved in CLA by using a special combinational logic circuit.This Combinational circuit generates the carry of each stage in single machine cycle. The carry of each stage generated in advance.The key concept of CLA is its different combinational logic circuit and partial full adder addition.The intermediate combinational circuit generates 2 signals i.e. G (generated carry bits) P (Propagated carry bits).The G signal is a bit position which generates carry if both corresponding bits of operands are logic high '1', in Boolean logic expressed as Gi = Ai Bi 2.1.2 The P signal is a bit position which generates carry if any of the corresponding bits if operands is logic high '1', in Boolean logic expressed as Pi = Ai + Bi 2. 1.3 The Carry Look ahead logic generates the Carry signal of ith bit in previous stage, with combinational logic of Boolean logic expressed as Ci +1 = Gi + Pi Ci 2.1.3 This combinational logic helps in generating carry parallelly reducing the delay of carry generation.The Corresponding Sum bits are generated with help of the xoring of corresponding bits of operands.Si P

Ci2.1.4The output of all bits sums and final carry is generated in 4 machine cyclesbut, the only problem in CLA concept is the combinational logic for parallel carry generationrequires an 'and gate' and 'or gate' of n+1 inputs for n-bits width of CLA.i.e. for a 4-bit CLA in combinational logic stage it requires an 'and gate' of 5 inputs.By this the fan in and fan out delay by which indirectly we get propagational delays.Although this delay is smaller than RCA but the design complexity, costly design, power consumption causes its limitations for some other kinds of specific application.Time delay calculation for CLA, here we have 3 different parts.i.e. G and P signal generation, Sum generation and Carry Calculation delay.For G and P, we can express delay by following equation tG P = max (tAND, tOR 2.1.5 The sum calculation delay involves only xor gate delay hence we can express as tS = tXOR 2.1.6 The Carry calculation involves multiple levels of 'AND' & 'OR' gates.For an n-bit CLA, the carry-out of the last bit involves a maximum of log2(n) levels of gates.By which we can express carry delay by the following equation tC = log2(n) * (tOR + tAND) 2.

1.7 By all above 3 three equations the final delay for CLA n bit will be given by following equation tCLA = tXOR + log2(n) * (tOR + tAND) + max (tAND, tOR2.1.8 For example, in an 8-bit CLA all logic gates XOR, AND & OR having unit time delay, from the equation 2.1.8 we can say that 8-unit time delay.I have implemented 8-bit CLA during the internship also, implemented 4-bit design but showing results for 8-bit CLA only.This implementation is similar kind as RCA.And design is extensible to 16-bits or any n-bit as designed with generate function.1.3 8-Bit CLA RTL From Xilinx Design Implementation Fig 2.1.4 8-Bit CLA Simulation Output From Xilinx Design Implementation Carry Save Adder (CSA) A CSA is a type of binary digital adder used primarily in computer arithmetic operations.It is designed to add multiple operands simultaneously, which is particularly used in multiplication and accumulation operations.The CSA significantly speeds up the process by not propagating carry bits immediately but rather saving them for later addition.For an n-bits width CSA, the structure primarily consists of multiple stages of addition, where each stage handles the addition of three inputs and generates two outputs a sumof n-bits width and a carry bit. The key component of the CSA is basically a full adder.For an n-bit CSA, in stage 1 of initial addition n Full Adders, each taking three input bits and generating n sum bits and n carry bits.

In stage 2 of addition of sum and carry bits sum bits from the first stage are passed to the next stage and carry bits are shifted left and then added.This stage of shifting and adding continues until we haven-bitsum output and a single bit carry output.Timing calculations for CSA, According to structure we have initially a full adder delay.Each full adder delay associated with generating sum and carry bit where, delay of full adder expressed by following equation tFA tXOR + tAND + tOR1.5 The next delay is of CSA logic delay.The critical path delay is determined by the longest chain of carry propagation.For an n-bit CSA, the carry is not propagated immediately hence, the delay depends on the depth of carry-save stages.Now each stage involves full adder delays, typically uses log2(n) stages for adding n bits. The final delay for adding n-bit three operands, the delay determined by the depth of stages and delay of a single full adder.The final delay expressed as following equation, TCSA = k tFA1.6 where k is the number of stages.For example, by considering tFA = 4-unit delay which consists 2-unit delay for XOR gate, 1 unit delay each for 'AND' and 'OR' gates.For an 8-bit CSA no.

of stages in CSA is 3 from earlier discussion and final delay will of 12-unit time only.Hence, we can conclude that The CSA is an efficient adder for handling multiple inputs simultaneously without the immediate carry propagation delay.I have implemented 8-bit CSA design on Xilinx here are the results of following design implementations in figures.13925550 Fig 2.1.5 8-Bit CLA RTL From Xilinx Design Implementation 190611642 Fig 2.1.6 8-Bit CSA Simulation Output From Xilinx Design Implementation Comparison of Adders Table 2.1 Comparison of adders.Parameter RCA CLA CSA Basic Operation Sequential carry propagation Parallel carry computation Saves intermediatecarries for later addition Sum Calculation Sequential Parallel Parallel with multiple stages Carry Calculation SequentialParallel using generates and propagate signal Saved for later stages Gate count Low High Moderateto High Area Small Large Moderate Power Consumption Low High Moderate Speed Slow Fast Fastin multi-operand addition Complexity Low High Moderate Scalability Poor Good Good Example 8-bit delay 16units 9 units 12 units MULTIPLIER DESIGNS Multipliers are fundamental components in digital electronics, usedto perform arithmetic multiplication of binary numbers.They are essential in various applications such as digital signal processing (DSP), computer graphics, and general-purpose computing.

The general types of multipliers used in digital electronics are Array Multipliers, Carry SaveArray Multipliers, Wallace Tree Multipliers, Sequential Multipliers, Booth Multipliers, Pipelined Multipliers, Bit-Serial Multipliers.Among these Array Multipliers are most basic multipliers and used in understanding of Multipliers Functions.The multipliers used and applied in different application on basis of their speed, efficiency, versatility, accuracy, integration etc.The general applications of multipliers are Digital Signal Processing, Computer graphics, Cryptography, Machine Learning, Embedded systems, etc.Array Multipliers Array Multipliers also known as Fast Multipliers are straight-forward and efficient method for performing binary multiplication.They use an array of adders to sum partial products generated by multiplying each bit of one operand by each bit of the other operand.Let's understand designatimingof2-bit, 3-bit and 4-bit array multipliers and understand difficulties in n-bit multiplier designs.In array multiplier the first stage to get partial products hence for a 2-bit array multiplier let's have 2 operands X and Y of 2 bits which generate the product of 4 bits.Table 2.2 2-bit Multiplication X1 X0 <= Multiplicand operand × Y1 Y0 <= Multiplier operand X1Y0

X0 Y0 <= Partial products with Y0 C2 X1Y1 X0 Y1 × <= Partial productswith Y1 and weigh added by 21 C2 X1Y1 + C1 X1Y0 + X0 Y1X0 Y0 <= Addition Results P3 P2 P1 P0 <=Product bits In above calculation wecan see that X is multiplicand and Y is multiplier.And we get product of 4-bits.There are 4 partial products in above multiplication by multiplying the bits Y0 and Y1 bitwise to X vector and we required 4 'and gates' for the partial product generation. Further in next stage the addition of each partial products with the corresponding weights is done and we get respective product bits generated.In this 2-bit array multiplier we have 2 half adders for addition as wehave addition for P1 bit partial products which generates the Carry C1, carried to thenext stage and there another half adder adds the next partial product and generated carry considered as 4th bit P3, of the product bits.743123539750 Fig 2.2.1 2-Bit Multiplier RTL From Xilinx Design Implementation 743123285986 Fig 2.2.2 2-Bit Multiplier Simulation Output From Xilinx Design Implementation Now in 3-bit fast multiplier let's see the design and the architecture.Similarly considering two 3 bits operands X and Y where X is multiplicand and Y is multiplier, and we have P vector of 6 bits.

3 3-bit Multiplication X2 X1 X0 <= Multiplicand operand × Y2 Y1 Y0 <= Multiplieroperand X2Y0 X1Y0 X0 Y0 <= Partial products with Y0 X2Y1 X1Y1 X0 Y1 ×<= Partial products with Y1 and weigh added by 21 X2Y2 X1Y2 X0 Y2 × × <= Partial products with Y2 and weigh added by 22 C23 X2Y2 +C22 X2Y1+ C21 + C12+ X1Y2 X1Y1 + C11 + X1Y1 + X0 Y2 X1Y0 +X0 Y1 X0 Y0 <= Addition Results and carry results where C1x shows the carrygenerated by addition at first stage and then C2x shows the carry generated by secondstage of addition P5 P4 P3 P2 P1 P0 <=Product bits In above calculation no.of adders may not be easy t find but no.of 'and gates' are clearly visible which is 9.For P0 we don't require adder hence P0 is directly fetched.Now for P1 bit we must use a half adder as no carry in from previous stage.The Sum output of that half adder will be P1 and Carry will be carried forward as C11.Now in next stage partial products and carry generated, added using full adders.And by reference to my circuit analysis during work I concluded that it requires 3

half adders and 3 full adders in structure.10687051810385 Fig 2.2.3 3-Bit Multiplier RTL From Xilinx Design Implementation 659130276225 Fig 2.2.4 3-Bit Multiplier Simulation Output From Xilinx Design Implementation Now let's see about 4-bit array multiplier.Similarly considering the A and B vector of 4 bits and output of 8 bits. 430530678180 Fig 2.2.5 4-Bit Multiplier Multiplication Method From reference to above figure, we can say that we require 16 'and gates' to get partial products.And with reference to my lab-work I concluded that it requires 4 half adders and 8 full adders.By which we can conclude for an n-bit multipliers no of required components given by following equations No.of 'and gates' = n2 2.2.1 No. of half adders = n 2.2.2 No. of full adders = n(n-2) 2.2.3 where n is width of input of operands of multipliers.Now considering time delays taken by each multiplier and look at any kind of disadvantage.For a 2-bit, 3-bit, 4-bit or n-bit multiplier design we require only 1 gate delay for partial products.

Now for 2-bit multiplier we have 2 half adders and the 2nd half adder is dependent on 1st adder for carry.Let's consider 2-unit delay required to propagate carry from 1st half adder hence total delay will be of 5-unit (i.e. 1+2+2). Now in case of 3-bit multiplier in addition we have 3 levels 1st for half adder with 1-unit delay.2nd level of full adders for intermediate stage with 2-unnit delay.And similar at final stage 3rd level of 2-unit delay.The total delay about of 6-unit time.Similarly in case if 4-bit multiplier we have final level of adders in 4th stage hence the total time delay can be of 8-unit times.The design constraint of this fast multipliers or array multipliers is no.of logic and gates and full adder blocks required to get a design for n-bit width. Such as for a 8-bit array multiplier we need 64 'and gates' and about8 half adders and 48 full adders for simplicity if we use all full addersimplementation then also we require 56 different full adders as design of such array multiplierin VHDL is not easy because we can't use generate logic for addition stage.Also, the chip size and power consumption increase with increase in width of multiplier input.

Now let's see the design implementation of 4-bit multiplier on Xilinx.8115300 Fig 2.2.6 4-Bit Multiplier RTL From Xilinx Design Implementation 697230350520 Fig 2.2.7 4-Bit Multiplier Simulation Output From Xilinx Design Implementation Booth Multipliers Booth multiplier is kindof multiplier works on basis of booth's algorithm of multiplication that multiplies two signed binarynumbers in 2's complement notation.Booth uses desk calculators that were faster at shifting than adding and created the algorithm to increase the speed and performance of the multiplier.This multiplier is used more due to its ease in scaling and faster multiplication.Also, the design is not that complex as compare to fast multipliers.4083051603375 Fig 2.2.8 Booth'sAlgorithm Flow Chart Here A is the input operand Multiplicand, B is the multiplier operand.Q register is initially same as B, Q0 holds the LSB of the Q register.Q-1 is a 1 bit variable or register.ACC is Accumulator register for holding result of intermediate addition and substruction.

Count holds the maximum width of multiplicand or multiplier.The disadvantages of booth logic are high power consumption and larger chip area occupation.Even though these disadvantages we use Booth Multiplier in few applications such as Arithmetic units, DSP, Scientific computing and machine learning and neural networks. 15417801066800 Fig 2.2.9 8-Bit Booth Multiplier RTL From Xilinx Design Implementation 7389810 Fig 2.2.10 8-Bit Booth Multiplier Simulation Output From Xilinx Design Implementation Multipliers Comparison Table 2.4 Comparison of Multipliers Parameter Booth Multiplier Array Multiplier Algorithm Booth's Algorithm Patial product additionBasic Operation Encoding and reducing number of additions Summing partial products Handling signed numbers Efficientlyhandles signed numbers Requires separate handling of sign Number of operations Reduces operations for sequencesof 1's Number of operations proportional to n2 Speed Faster for large no.of consecutive 1's Slower for large bit-widths Hardware Complexity More complex Simple straight forwardScalability More scalable Less scalable Power consumption Lower Higher Efficiency More in certain patterns Uniformperformance Implementation More complex logic control Simple repetitive Best use case Signed multiplication large bit-widthsSmaller bit-widths, simpler applications CHAPTER 3: VLSI PHYSICAL DESIGN FLOW INTRODUCTION After design implementation onXilinx and having RTL of combinational logic circuits, we require a physical chip or IC

fabricated and packaged to use it in day-to-day life.As I discussed earlier in introduction chapter that SoC are designed and fabricated to use in mobile phones.That IC is first having architectural design in three domains of VLSI design flow, physical, behavioral, structural domains.Then the fabrication of IC is done.We till now done design implementation in behavioral domain and structural domain to check whether our thoughts or idea can give the similar output as our requirement.Now we will explore about physical domain.The Geometrical design or Layout design is done in physical domain.This process done in sophisticated software tools called as Electronics Design Automation (EDA) or CAD tools.The software EDA tool we used for physical layout is OpenROAD.Now let us explore more about OpenROAD tool next section.INTRODUCTION TO OpenROAD EDA TOOL The OpenROAD EDA tool is Linux based opensource tool which is used to generate RTL-GDSII layout of Verilog language designed entities or modules.If we talk about OpenROAD tool then it consists of various tool chain for different steps of the RTL-GDSII flow layout planning.This chain of tools is called as OpenROAD-flow-scripts (ORFS).The ORFS consists of Magic, Yosys, OpenSTA, iverilog, gtkwave, klayout, OpenLane, OpenROAD.Each tool has its own use in different stages of RTL-GDSII flow.

Likewise Magic and Klayout are for layout planning.Yosys used as logic synthesis tool which synthesizes the raw verilog file to required SDC file for next steps of routing, netlisting and floorplanning of ICs.Earlier stages there was no tool for clock tree synthesis but now we have tools like OpenSTA or OpenCLOCK for static timing analysis and clock tree synthesis.iverilog tool used to have HDL simulations inside the ORFS.gtkwave is waveform viewer.And at last, the main OpenROAD or OpenLane where we call all this tools by its GUI and command prompt window in GUI.Let us see how we can install and use all those tools to design IC from RTL to GDSII.Step-by-step Installation Process.The basic required system for using ORFS is Ubuntu Linux Based System.For latest Ubuntu 24.04 LTS we require at least 8 GB RAM recommended along with 100 GB of free space HDD or SSD and a quad processor with 2.4 GHz clock or better than that.A stable internet connectivity is also recommended probably ethernet or Wi-Fi as software installationis become bad to worst if internet is not working properly and may stuck in

between any step of installation process.A dual boot laptop or desktop, or an ubuntu based system is required, do not try, or waste your time trying any other methods of Linux installation in your system such as WSL or VMBox.Simply install ubuntu linux in your system and do the step to install ORFS.To install Ubuntu 24.04 by your own efforts you should have to create bootable pen-drive of 16 GB, and Then by entering to BIOS setup doing some changes you will be able to do boot in Ubuntu and Windows on single system.After installation of Ubuntu do 2 to 3 times of check of booting andtry to install ORFS after 4 to 5 hours of installation of Ubuntu in system,if you using your old Ubuntu system then you can go through the installation stepsas given below.Installation steps for ORFS are as follows Open the command terminal upgrade, and update the packages and whole ubuntu for new releases may required in further steps by usingcommandsudo apt-get update, sudo apt upgrade (reboot recommended after packages upgradation) Install the git packagewhich will be used to clone GitHub repositories to your system by using the command,sudo apt install git –assume-yes Install the dependencies and build essential by prompting command, sudoapt install -y build-essential Install the Python using following command, sudo apt install -y pyhton3-venv

Next step is to make the default 'sh' to be 'bash' rather than 'dash' youcan do it by following command, sudo rm /bin/sh && sudo ln -s /bin/bash /bin/shAfter that you should create directory or work area in ubuntu for ORFS by usingfollowing command, mkdir -p Work/vlsi/tools Now we will move to tools directory and install Yosysand OpenROAD in that directory.The command should be followed as given, cd ~/Work/vlsi/tools git clone –recursive https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts Setup of OpenROAD and Yosys by following command, mkdir Open-ROAD-flow-scripts, cd ~/Work/vlsi/tools/OpenROAD-flow-scripts, sudo ./setup.sh Installation of OpenROAD and Yosys by following command, ./build_openroad.sh –local (this step will take about 30-40 minutes to install without any error).Installation of dependencies of Magic VLSI tool by using following commands,cd && sudo apt-getupdate && sudo apt-get install m4 --assume-yes && sudo apt-get install tcsh --assume-yes && sudoapt-get install csh --assume-yes && sudo apt-get install libx11-dev --assume-yes && sudo apt-get install tcl-devtk-dev --assume-yes && sudo apt-get install libcairo2-dev --assume-yes && sudo apt-get install mesa-common-dev libglu1-mesa-dev --assume-yes

&& sudo apt-get install libncurses-dev --assume-yes Installing Magic VLSI tool by using following command asgiven below,git clone https://github.com/RTimothyEdwards/magic.git && mkdir yosys && cd yosys && sudo make && sudo make install (thisprocess of installation may take 30-45 minutes depending on stability and connectivity of internet youproviding.Installation of the iverilog tool for HDL simulations in ORFS,sudo apt-get install -y iverilog–assume -yes Installation of the gtkwave waveform simulation tool in ORFS,sudo apt install gtkwave –assume-yes Installation of the Klayout, layout planning toll in ORFS,sudo apt install klayout –assume -yesInstallation of docker in ubuntu to install the OpenLane by docker method, sudo apt-get removedocker docker-engine docker.io containerd runc –assume-yessudo apt-get update –assume-yessudo apt-get install\ apt-transport-https\ca-certificates\curl\gnupg\lsb-release –assume-yescurl-fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o/usr/share/keyrings/docker-archieve-keyring.gpg –yesecho\ “deb [arch=amd6 signed-by=/usr/share/keyrings/docker-archieve-keyring.

gpg] https://download.com/linux/ubuntu\$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >/ddev/nullsudo apt-get update –assume-yessudo apt-get install docker-ce-cli containered.io –assume-yes Installation of OpenLane tool under the ORFS tool chain, cdgit clone https://github.com/The-OpenROAD-Project/OpenLane.gitmkdir OpenLanecd OpenLanesudo makesudo make install By following above steps, we can install all therequired tools for the ORFS tool chain, now we can proceed the step to installthe opensource PDK supported by OpenROAD which will provide design constraints and standard cell libraryfor our design implementation.By default, with OpenLane and ORFS we get skywater130 PDK and nangate45 PKD support, for other PDKs such as GF180MCU, SKY90FD, GlobalFoundries are such PDKs.The installation of the PDK in the system is not that tough you can go through website link given below to install PDKs the link for installation process is: http://www.opencircuitdesign.com/open_pdks/install.html After installation of all the tools we need to verify either all the tools are successfully installed or not.

These current steps support verilog language synthesis only.For implementing and synthesis of VHDL modules in yosys gnat and ghdl, so that we can use VHDL files and parse for synthesis.Installation Verification Process.To verify the installation of ORFS tools the best is to use them directlyor simply enter each command one by one by which information of the this ORFStools or version may be visible to you in terminal or if the version orinformation is not visible then you must reinstall the respective tool by following respective stepmentioned above again.The command to verify each tool installation given in following table.Table 3.1 Command table for verification Command Command Output yosys -V Version of yosys installed willbe displayed openroad -version Version of openroad will be visible gtkwave -v gtkwave installed versionwill be displayed iverilog -V iverilog installedversion will be displayed docker --version docker version will be displayed opensta either GUI ofopensta or information will be displayed klayout -v klayout version will be displayed magic --version

magic tool installed version will be displayed ./flow.tcl -help (this command should be used by moving to OpenLane directory and then run this command) This will display the information of OpenLane flowscript.ls $PDK_ROOT/sky130A or ls $PDK_ROOT/pdk_name This will list the content of PDK directory.RTL TO GDSII FLOW The ORFS has different tools by which we can go through RTL to GDSII flow easily if we provide proper parameters and PDK files.Later, we will discuss about how we can get RTL to GDSII step by step. Now let us understand about what is RTL to GDSII flow.The ORFS is autonomous software but we will use it in manual order step by step to debug in between any error.Now see the flow of RTL to GDSII in following figure.9786511616710 Fig 3.3.1.a RTL to GDSII Design Flow As per the last figure we need to createa verilog file, and then from PDK we must select any PDK and then usethat PDK's library file and design constraint file during the synthesis process through the yosys.1906846666 Fig 3.3.1.b RTL To GDSII Flow Using ORFS Now let us understand about each part of Physical design flow given in figure 3.

3.1.b Synthesis: The process of synthesis is to synthesize the netlist file of the module using selected standard cell library provided by the vendor.The input of the synthesizer used yosys in the ORFS are verilog file and standard cell library which gives output of Gate level netlist and SDC of the designed module.Floorplan: The Floorplan will initiate the basic area of die and core which will be used to design on actual silicon substrate.It helps in random IO placement which further rectified according to design and tapcell insertion for the design.Placement: The placement in RTL to GDSII consists of many parts such as Globalcell placement, IO placement, Global Placement with placed Ios, Resizing and buffer settings and detailedplacement, provides various parameters and organizes the different objects such as IO pins, Buffers placementand give proper distribution.Clock Tree Synthesis: This is done with help of opensta tool, which is usedwhenever we have any kind of clock driven part inside the chip and it willhelp in dividing the clock and proper clock functioning.Routing: This part helps in interconnections between vias and metals layers of the different layers and interconnections between different blocks of design inside the chip.Finishing: In general, wherever we using checkpoint, report and repair we are checking for any kind of faults and error in design which is termed as Finishing of design.Now let us see the step-by-step process of getting RTL to GDSII final layout from a verilog file.Design realization and implementation by using PDK of vendor nangate45 which support 45nm technology window design, and consist

design constraints and standard cell library for design implementation.The PDK will be provided by the vendor from whom you wished to fabricate your IC, here we using open-source PDK for our designs Note:- Commands and outputs are in Italic and Bold.Also due to irregularity in sentence width the justify alignment isn't used during the steps as its seems too bad format, as of example you can see the above paragraph ending.To create a verilog file you can use by default text editor and saveit in any folder in home directory or if you installed by following our method then you can do it by saving it in tools folder's any new folder, alongwith .v extension.Open terminal and write command: yosys .To access the yosys for synthesis.Open your verilog file by using command: read_verilog /$file_path/file_name.v read_verilog /home/darshit-mehta/run01/CarryLookAheadAdder_8bit.vIn this step we will get output message on terminal asGenerating RTLIL representation for module `\CarryLookAheadAdder_8bit'.Successfully finished Verilog frontend.Checking, expanding, and cleaning of the design hierarchy , which will be done by

following command: hierarchy -check -top CarryLookAheadAdder_8bitBy performing this step you will get the output suchthat analyzing design hierarchy..Top module: \ CarryLookAheadAdder_8bitRemoved unused 0 modules Note:- The setup we got limitation of settingonly one entity module, if your verilog code has more than one entity module thenyou may get error at previous step or any of the next step.Hence to ensure the least error chances implement your design based in basic gates rather than using more than one entity.Conversion "processes" into multiplexers and registers (the internal representation of behavioral verilog code) by using command: proc .After this you should wait until any error message shows or proceed to next step if no error message shows.Perform some basic optimizations and cleanups by using following command: optThis will show message at last that Finished OPT passes.(There is nothing left to do.) After getting this message you can proceed to next step.Analyze and optimize finite state machine for your design by using following command: fsmBy following this step you get last message as Executing FSM_MAP pass (mapping FSMs to basic logic).After this repeat Step 6.Note: From this onward after every step, it is good practice to use command: opt for performing basic optimization and cleanups.Analyze memories and create circuits to implement them for your design by using following

command: memoryThis kind of message will be shown in ending if this process Executing MEMORY_MAPpass (converting memories to logic and flip-flops).Map coarse-grain RTL cells (adders, etc.) to fine-grain logic gates (AND ,OR ,NIT,ETC...) by using following command: techmapSuccessfully finished Verilog frontend.This message may be displayed or if you get any error you can restart process or debug it if error is understandable.Map registers to available hardware flip-flop from lib by using following command: dfflibmap -liberty /home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.lib$/home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45 this is the file path of standard cell library provided by vendor.Wait till the process end andproceed to next step.Map logic to available hardware gates by using the command: abc -liberty /home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.libBy performing this step you will message displaying some useful information.Re-integrating ABC results.ABC RESULTS: NAND2_X1 cells: 3 ABC RESULTS: AOI21_X1 cells: 3 ABC RESULTS: AND2_X1 cells:

4 ABC RESULTS: OR2_X1 cells: 4 ABC RESULTS: AOI221_X1 cells: 1 ABC RESULTS: NOR2_X1 cells:5 ABC RESULTS: XOR2_X1 cells: 4 ABC RESULTS: OAI21_X1 cells: 4 ABC RESULTS: XNOR2_X1 cells:12 ABC RESULTS: internal signals: 115 ABC RESULTS: input signals: 17 ABC RESULTS: output signals:9 Removing temp directory.Next step is to flattening of design using command: flatten Executing FLATTEN pass. (flatten design) {This message may be seen} Replacement of undef values with defined contants this is important as if we don't do this then our design may get any error in any next step.The command for this task is : setundef -zero Removal of unused cells andwires is also important to optimize our design, The command for this task is: clean -purge Next step is technological mapping of i/o pads (or buffers) by using the command:iopadmap -outpad BUF_X2 A:Z -bits Use command : clean .To remove unused cells and wires.To do some important calculations such as die area or core area of ICwe require some stats this statistics data obtain by using command: stat -liberty /home/Darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.lib Here "/home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test/Nangate45/Nangate45_typ.lib" is path of the standard cell library file, which will give stats related such as no.

components, wires, cells, processes and at last the chip area value in micrometers2 units.This will determine the chip core area and die area in substrate during fabrication process.Printing statistics.=== CarryLookAheadAdder_8bit === Number of wires: 38 Number of wire bits: 66 Number ofpublic wires: 5 Number of public wire bits: 26 Number of memories: 0 Number ofmemory bits: 0 Number of processes: 0 Number of cells: 49 AND2_X1 4 AOI21_X1 3 AOI221_X1 1 BUF_X2 9 NAND2_X1 3 NOR2_X1 5 OAI21_X1 4 OR2_X1 4 XNOR2_X1 12 XOR2_X14 Chip area for module '\CarryLookAheadAdder_8bit': 59.052000 This will be displayed message may vary according to your components.Rename the object in the design is the next step , we can assign short auto-generated names to all the selected wires and cells with private name or use this command: rename -enumerate to do same.Overwriting the design to verilog file is also included in this step this is done by using command: write_verilog -noattr CarryLookAheadAdder_8bit1_final.v Now to generate BLIF file or netlist we must use the following command: write_blif -buf BUF_X2 A Z CarryLookAheadAdder_8bit1_mapped_withbuf.blif Now we should check the netlist generated and stored in BLIF file by previous step with help of graphviz, it will generate schematic graph of our netlist.This can be done by using command: show -stretchIn this step you will be able to see the netlist as shown in figure.Fig 3.

3.2 RTL to Gate Level Netlist of CarryLookAheadAdder_8bit Now you have exit the yosys using command: exit , and interact with OpenroadGUIbyusing command: openroad -gui Note: Until this step we were working over yosys the synthesis tool now we must go for further steps to generate final GDS layout using OpenROAD tool of ORFS.The GUI of OpenROAD is command line-based tool hence at each step has different commands, the commands will be same font as until we see in previous steps.Create a SDC file with text editor and give name as SDC and save in .csv extension.This file also in same directory where you have previously saved verilog file.You may notice some changes that verilog file if you did not save the synthesized verilog file in different name.The content of SDC file is create_clock [get_ports clk] -name core_clock -period 0.4850 set_all_input_output_delays Now you must save this SDC file and final verilog file in folder with following path, $/home/darshit-mehta/Work/vlsi/tools/OpenROAD-flow-scripts/tools/OpenROAD/test .After that you may proceed for further steps.Note: The chip area will give you idea about estimation of chip area anddie area, You can either you can keep same as chip area or round itto nearest ten for sake of simplicity and for larger areas you have to choose

area by trial and error method.Although we have a relation that if u = utilization efficiency of chip, then die area = (chip area * 100) / u, where u ranges between 30% to 90% practically may more different.And according to die area you can set core area.Now we will put following commands in command prompt of openroad GUI.set design "CarryLookAheadAdder_8bit"set top_module "CarryLookAheadAdder_8bit"set synth_verilog "CarryLookAheadAdder_8bit1_final.v" set sdc_file "CarryLookAheadAdder_8bit.sdc" #set die_area {0 0 42 42}set die_area {0 0 42 42}#set core_area {2.5 2.5 40 45} set core_area {2.5 2.5 40 45} After having this command in the command window now you can press enter to set these parameters to the openroad for the design.In this step you have to give command of reading the design files tothe openroad by using commands given below,read_libraries read_verilog $synth_verilog link_design $top_module read_sdc $sdc_file In thisstep we will having STA network instance count for all other parameters required to generatethe final layout, this step will help in creating all other parameters.utl::metric "IFP::ord_version" [ord::openroad_git_describe] # Note that sta::network_instance_count is not valid after tapcells are added.utl::metric "IFP::instance_count" [sta::network_instance_count] In this step we will initiate process of floorplanning.

initialize_floorplan -site $site \ -die_area $die_area \ -core_area $core_areaBy which you will get floorplan figure. 3.3 Floorplan Command Output Window.Now to optimize the design we must remove unused or unrequired buffers by usingcommands,source $tracks_file# remove buffers inserted by synthesis remove_buffersWe will get output at output console: [INFORSZ-0026] Removed 16 buffers.Now next step is random IO placement by using the following commands# IO Placement (random)place_pins -random -hor_layers $io_placer_hor_layer -ver_layers $io_placer_ver_layer Fig 3.3.4 IO_Placement_Random Command Output Window Next step is macro-placement which can be done by usingcommand, # Macro Placementif { [have_macros] } {global_placement -density $global_place_densitymacro_placement -halo $macro_place_halo -channel $macro_place_channel} Now our next targetistocreate tapcell insertion by using following command,# Tapcell insertioneval tapcell $tapcell_argsThis will show some significant changes in output window, which shown in Fig 3.3.5 Fig 3.3.5 Tapcell Insertion Command Output Window145414422 Insertion of power distribution network using following command,# Powerdistribution network insertionsource $pdn_cfgpdngenThis command renders some significant changes to output window, Fig 3.

3.6 PDN Command Output Window The next step is of global placement of different partslayers interlinks, we can see some red instances in the chip design area by using following commands for global placement,# Global placementforeach layer_adjustment $global_routing_layer_adjustments {lassign $layer_adjustment layer adjustmentset_global_routing_layer_adjustment $layer $adjustment}set_routing_layers-signal $global_routing_layers \-clock $global_routing_clock_layersset_macro_extension 2global_placement -routability_driven -density $global_place_density \-pad_left $global_place_pad -pad_right $global_place_pad Fig 3.3.7 Global Placement Command Output Window-57361174200 Now we will organize the Input and Output Placements,by using following commands, # IO Placementplace_pins -hor_layers $io_placer_hor_layer -ver_layers $io_placer_ver_layer Fig 3. 3.8 IO Placement Command Output Window102871422 Now we must set a checkpoint by using following command,# checkpointset global_place_db [make_result_file ${design}_${platform}_global_place. db]write_db $global_place_db Now we have to optimize and repair the design by maximum of slew,capacitance and fanout violations and normalizing the slew by following commands,# Repair max slew/cap/fanout violationsand normalize slewssource $layer_rc_fileset_wire_rc -signal -layer $wire_rc_layerset_wire_rc -clock -layer $wire_rc_layer_clkset_dont_use $dont_useestimate_parasitics -placementrepair_design -slew_margin $slew_margin -cap_margin $cap_marginrepair_tie_fanout -separation $tie_separation $tielo_portrepair_tie_fanout -separation $tie_separation $tiehi_portset_placement_padding -global -left $detail_place_pad -right $detail_place_paddetailed_placement# post resize timing report (ideal clocks)report_worst_slack -min -digits 3report_worst_slack -max -digits 3report_tns -digits 3# Check slew repairreport_check_types -max_slew -max_capacitance -max_fanout -violatorsutl::metric "RSZ::repair_design_buffer_count" [rsz::repair_design_buffer_count]utl::metric "RSZ::max_slew_slack" [expr [sta::max_slew_check_slack_limit] * 100]utl::metric "RSZ::max_fanout_slack" [expr [sta::max_fanout_check_slack_limit] * 100]utl::metric "RSZ::max_capacitance_slack" [expr[sta::max_capacitance_check_slack_limit] * 100] Fig 3. 3.9 Repair Design Command Output WindowThe console outputs and analysis we get at console window are,Placement Analysis----------total displacement 47.

8 uaverage displacement 0.5 umax displacement 2.3 uoriginal HPWL 420.3 ulegalized HPWL 490.7 udelta HPWL 17 %worst slack 0.202worst slack -0.042tns -0. 070-91226-196426 Now next part is also important as we do CTS in next part, usingfollowing command group,# Clock Tree Synthesis# Clone clock tree inverters next to register loads# socts does not try to buffer the inverted clocks. repair_clock_invertersclock_tree_synthesis -root_buf $cts_buffer -buf_list $cts_buffer \-sink_clustering_enable \ -sink_clustering_max_diameter $cts_cluster_diameter# CTS leaves a long wire fromthe pad to the clock tree root. repair_clock_nets# place clock buffersdetailed_placement# checkpointset cts_db [make_result_file ${design}_${platform}_cts.db]write_db $cts_dbThis will set buffers and clock buffers and give output of no.of valid clock nets according to design, The most important parameter out by this commands is Using max wire length which is 693 um in our design.

Placement Analysis----------total displacement 0.0 uaverage displacement 0.0 umax displacement 0.0 uoriginal HPWL 490.7 ulegalized HPWL 490.7 udelta HPWL 0 % Now setup and hold timing repair for the chip designin another part of CTS, by using following commands,# Setup/hold timing repairset_propagated_clock [all_clocks]# Global routingis fast enough for the flow regressions.# It is NOT FAST ENOUGH FOR PRODUCTION USE.set repair_timing_use_grt_parasitics 0if { $repair_timing_use_grt_parasitics } {# Global route for parasitics - no guide filerequiedglobal_route -congestion_iterations 100estimate_parasitics -global_routing} else {estimate_parasitics -placement}repair_timing -skip_gate_cloning# Post timing repair.report_worst_slack -min -digits 3report_worst_slack -max -digits 3report_tns -digits 3report_check_types -max_slew -max_capacitance -max_fanout -violators -digits 3utl::metric"RSZ::worst_slack_min" [sta::worst_slack -min]utl::metric "RSZ::worst_slack_max" [sta::worst_slack -max]utl::metric "RSZ::tns_max" [sta::total_negative_slack -max]utl::metric "RSZ::hold_buffer_count" [rsz::hold_buffer_count]Those commands will give rendered output for virtual clock core_clockcannot be propagated which may differ design by design, and an Placement Analysis as shown

in down outputs,Placement Analysis----------total displacement 12.4 uaverage displacement 0.1 umax displacement 2.6 uoriginal HPWL 495.3 ulegalized HPWL 509.7 udelta HPWL 3 % Let us have now detailed placement of our design using following commands,# Detailed Placementdetailed_placement# Capture utilization before fillers make it 100%utl::metric "DPL::utilization" [format %.1f [expr [rsz::utilization] * 100]]utl::metric "DPL::design_area" [sta::format_area [rsz::design_area] 0]# checkpointset dpl_db [make_result_file ${design}_${platform}_dpl.db]write_db $dpl_dbset verilog_file [make_result_file ${design}_${platform}.v]write_verilog $verilog_file Fig 3.3.10 Detailed Placement Command Window Output The console will give some outputs parameters of designgiven as also enlists the default vias layer by layer vias, After all this renderingwe now do global routing by using following commands,# Global routingpin_access -bottom_routing_layer $min_routing_layer \ -top_routing_layer$max_routing_layerset route_guide [make_result_file ${design}_${platform}. route_guide]global_route -guide_file $route_guide \ -congestion_iterations 100set verilog_file [make_result_file ${design}_${platform}.

v]write_verilog -remove_cells $filler_cells $verilog_fileThis command will give some parameters such as,Reading tech and libsUnits: 2000Numberof layers: 21Number of macros: 135Number of vias: 30Number of viarulegen: 19Reading design.Design: CarryLookAheadAdder_8bitDie area: (0 0) (84000 84000)Number of track patterns: 20Numberof DEF vias: 0Number of components: 94Number of terminals: 26Number of snets: 2Number of nets:59 And list of vias different metal vias, This will be also part of globalrouting report Start pin access.Complete 57 pins.Complete 17 unique inst patterns.Complete 42 groups.#scanned instances = 94#unique instances = 21#stdCellGenAp = 474#stdCellValidPlanarAp = 0#stdCellValidViaAp = 335#stdCellPinNoAp = 0#stdCellPinCnt= 134#instTermValidViaApCnt = 0#macroGenAp = 0#macroValidPlanarAp = 0#macroValidViaAp = 0#macroNoAp = 0Complete pin access.Now we will repair any possible defects of Antenna by using following commands,# Antennarepairrepair_antennas -iterations 5check_antennasutl::metric "GRT::ANT::errors" [ant::antenna_violation_count]This will be give output of any net and pin violations,and give output asFound 0 net violations.Found 0 pin violations.

Now the next part is filler placement in design by using following commands,# Fillerplacementfiller_placement $filler_cellscheck_placement -verbose# checkpointset fill_db [make_result_file ${design}_${platform}_fill.db]write_db $fill_db There will be significant changes in output window of design, Fig 3.3.11 Filler Placement Command Window Output Now we will be doing detailed routing of ourdesign using following commands,# Detailed routing# Run pin access again after inserting diodes and moving cellspin_access -bottom_routing_layer $min_routing_layer \ -top_routing_layer $max_routing_layerset_thread_count [exec getconf _NPROCESSORS_ONLN]detailed_route -output_drc [make_result_file"${design}_${platform}_route_drc.rpt"] \-output_maze [make_result_file "${design}_${platform}_maze.log"] \-no_pin_access \-save_guide_updates \-bottom_routing_layer $min_routing_layer \-top_routing_layer $max_routing_layer \-verbose 0write_guides [make_result_file "${design}_${platform}_output_guide.mod"]set drv_count [detailed_route_num_drvs]utl::metric "DRT::drv" $drv_countcheck_antennasutl::metric "DRT::ANT::errors" [ant::antenna_violation_count]set routed_db [make_result_file ${design}_${platform}_route.db]write_db $routed_dbset routed_def [make_result_file ${design}_${platform}_route.def]write_def $routed_defThis commands will run at instances to provide different output checks and only someparameters which required internally to display design This will help in extraction of SPEF# Extractionif{ $rcx_rules_file != "" } { define_process_corner -ext_model_index 0 X extract_parasitics -ext_model_file $rcx_rules_file set spef_file [make_result_file ${design}_${platform}.spef] write_spef $spef_file read_spef $spef_file} else { # Use global routing based parasitics inlieu of

rc extraction estimate_parasitics -global_routing} These commands will check the give some error checks and final SPEF file for detailed information of interconnections between different components and gates, pins to thechip and design.To getthefinal report of the design and final design we can use the following commands,# Final Reportreport_checks -path_delay min_max -format full_clock_expanded \ -fields {input_pin slew capacitance} -digits 3report_worst_slack -min -digits 3report_worst_slack -max -digits 3report_tns -digits 3report_check_types -max_slew -max_capacitance -max_fanout -violators -digits 3report_clock_skew -digits 3report_power -corner $power_cornerreport_floating_nets -verbosereport_design_areautl::metric "DRT::worst_slack_min" [sta::worst_slack -min]utl::metric "DRT::worst_slack_max" [sta::worst_slack -max]utl::metric "DRT::tns_max" [sta::total_negative_slack -max]utl::metric "DRT::clock_skew" [expr abs([sta::worst_clock_skew -setup])]# slew/cap/fanout slack/limitutl::metric "DRT::max_slew_slack" [expr [sta::max_slew_check_slack_limit] * 100]utl::metric "DRT::max_fanout_slack" [expr [sta::max_fanout_check_slack_limit] * 100]utl::metric "DRT::max_capacitance_slack" [expr [sta::max_capacitance_check_slack_limit] *100];# report clock period as a metric for updating limitsutl::metric "DRT::clock_period" [get_property [lindex [all_clocks] 0]period]# not really useful without pad locations#set_pdnsim_net_voltage -net $vdd_net_name -voltage $vdd_voltage#analyze_power_grid -net $vdd_net_name The finalreport can be found in results folder of test for all detailed information and parametersif the designed chip, This is the final GDSII layout given in figure below, Fig3.

3.12 The Final CarryLookAheadAdder_8bit Design Output2046804377055 SOME DESIGN IMPLEMENS I have designed some designs implemented on basis of the PDK nangate45 which uses 45nm technology.4.1 Shift_Register Final GDSII Layout Design The CSA design implemented using nangate45 standard cell library, with set die_area {0 0 60 60}, set core_area {5 2 56 60}, max wire length of 693 um.4.2 mux_8bits Final GDSII Layout Design The RCA design implemented using nangate45 standard cell library, with set die_area {0 0 40 40}, set core_area {4 2 36 37}, max wire length of 693 um.4.3 demux_8bits Final GDSII Layout Design The 8-bit Multiplier design implemented using nangate45 standard celllibrary, with set die_area 0 0 50 55}, set core_area {4 3 45 50}, maxwire length of 693 um. CHAPTER 4: SUMMARY Let me summarize the activities I done during my internship.Initially I was doing design implementations on Xilinx and trying to install ORFS.I had learned about the different kind of advanced adders used in computer architecturealso, learned about the multipliers which applied as DSP, and convolutors for Machine Learning andDeep Learning Model architectures.I had explored about the lab facilities available to students and researchers in the SVNIT.The installation ORFS was not easy, I have spent over a week to just get that ORFS on my system.And after getting the software ORFS I explored about how to use it, I got familiar to code hub which is know as github.Before all this I learned how to use ubuntu basics as that software is a opensource hence not available in Windows based system.

The ORFS supports many other PDKs also which make its more useful for initial research and design implementations.References/ Bibliography As per the guidelines and the references given by GTU, Ahmedabad.I have tried to keep this thesis report / Internship Report as sole creatorand writer me, although 100nnot be done by me hence I have taken references ofdesigns and some terms and important images and parts from the web, knowingly or someothers may show in plagiarism report, Some of references used by me knowingly are listedbelow, Invent Logics-ShopNowfor Xilinx FPGA development boards (allaboutfpga.com) The OpenROAD Project · GitHub Online Collaborative Whiteboard | Sketchboard VLSI Physical Design with Timing Analysis - Course (nptel.ac.in) OpenROAD Flow Scripts Tutorial — OpenROAD Flow documentation (openroad-flow-scripts.readthedocs.io) OpenLane using Codespaces : RTL-to-GDSII flow.

- Part-2 (youtube.com) https://www.youtube.com/watch?v=_AXknRBk4QI OpenROAD and efabless OpenLane at the CHIPKIT tutorial, ISCA-2020 (youtube.com) And there may be some other similar references can be listed in Plagiarism report which are unknowingly references.

Matched Sources

www.ijirst.org	http://www.ijirst.org/articles/IJIRSTV1I7110.pdf
www.svnit.ac.in	https://www.svnit.ac.in/web/department/Electronics/15SRLab.html
cloudsecurityalliance.org	https://cloudsecurityalliance.org/
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/and
www.claconnect.com	https://www.claconnect.com/en/
en.wikipedia.org	https://en.wikipedia.org/wiki/Pi
en.wikipedia.org	https://en.wikipedia.org/wiki/Circled_plus
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/Ci
www.w3.org	https://www.w3.org/WAI/WCAG21/Understanding/character-key-shortcuts.html
www.quora.com	https://www.quora.com/What-is-the-meaning-of-an-n-bit-computer
en.wikipedia.org	https://en.wikipedia.org/wiki/A
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/timing
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/of
www.ijeetc.com	https://www.ijeetc.com/v3/v3n1/11_A0134_(92-103).pdf
en.wikipedia.org	https://en.wikipedia.org/wiki/Graphical_user_interface
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/by
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/is
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/to
www.dictionary.com	https://www.dictionary.com/browse/the
shop.app	https://shop.app/
www.merriam-webster.com	https://www.merriam-webster.com/dictionary/now
about.readthedocs.com	https://about.readthedocs.com/

GUJARAT TECHNOLOGICAL UNIVERSITY

CERTIFICATE FOR COMPLETION OF ALL ACTIVITIES AT ONLINE PROJECT PORTAL

B.E. SEMESTER VII, ACADEMIC YEAR 2024-2025

Date of certificate generation : 20 September 2024 (19:39:46)

This is to certify that, ***Sahu Arabinda Kailash*** (Enrolment Number - 210230111008) working on project entitled with ***VLSI Physical Design*** from ***Electronics & Communication Engineering*** department of ***DR. S & S.S. GHANDHY GOVERNMENT ENGINEERING COLLEGE, SURAT*** had submitted following details at online project portal.

Internship Project Report	**Completed**

Name of Student : Sahu Arabinda Kailash

Name of Guide : Mrs.Pithadia Parul Vasantlal

Signature of Student : Arabinda

*Signature of Guide :

Disclaimer :
This is a computer generated copy and does not indicate that your data has been evaluated. This is the receipt that GTU has received a copy of the data that you have uploaded and submitted as your project work.

*Guide has to sign the certificate, Only if all above activities has been Completed.