DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to Visvesvaraya Technological University (VTU), Belagavi,
Approved by AICTE and UGC, Accredited by NAAC with 'A' grade & ISO 9001 – 2015 Certified Institution)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560 111, India

DEPARTMENT OF ELCTRONICS & COMMUNICATION ENGINEERING

(Accredited by NBA Tier 1: 2022-2025)

Project Report on

DESIGN OF CORDIC CORE – RTL TO GDS

Submitted in partial fulfillment for the award of the degree of

Bachelor of Engineering
in
Electronics & Communication Engineering

Submitted by

AKASH V KASHYAP	**1DS21EC023**
AKSHAY N NAYAK	**1DS21EC025**
CHETAN PATIL	**1DS21EC055**
CHIRAAYU M V	**1DS21EC056**

Under the Guidance of

Dr. SANTOSH KUMAR R
Assistant Professor
Department of Electronics and Communication Engineering
DSCE, Bengaluru

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI-590018, KARNATAKA, INDIA
2024-25

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to Visvesvaraya Technological University (VTU), Belagavi,
Approved by AICTE and UGC, Accredited by NAAC with 'A' grade & ISO 9001 – 2015 Certified Institution)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560 111, India

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

(Accredited by NBA Tier 1: 2022-2025)

CERTIFICATE

Certified that the project report entitled **"Design of CORDIC Core – RTL To GDS**" carried out by **Akash V kashyap , Akshay N Nayak , Chetan Patil , Chiraayu M V** bearing **USN 1DS21EC023 ,1DS21EC025 ,1DS21EC055 ,1DS21EC056** is a bonafide student of **DAYANANDA SAGAR COLLEGE OF ENGINEERING**, an autonomous institution affiliated to VTU, Belagavi in partial fulfillment for the award of Degree of **Bachelor of Electronics & Communication Engineering** during the year **2024-2025**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements with respect to the work prescribed for the said Degree.

Signature of the Guide	Signature of the HOD	Signature of the Principal
Dr. Santhosh Kumar R	**Dr. Shobha.K.R**	**Dr. B G Prasad**
Assistant Professor	**Professor & Head**	**Principal**
Dept. of ECE, DSCE	**Dept. of ECE, DSCE, Bengaluru**	**DSCE, Bengaluru**
Bengaluru		

Name of the Examiners	Signature with date
1.	**.....**
2.	**.....**

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to Visvesvaraya Technological University (VTU), Belagavi,
Approved by AICTE and UGC, Accredited by NAAC with 'A' grade & ISO 9001 – 2015 Certified Institution)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560 111, India

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

(Accredited by NBA Tier 1: 2022-2025)

DECLARATION

We, **Akash V kashyap (1DS21EC023), Akshay N Nayak (1DS21EC025), Chetan Patil (1DS21EC055) and Chiraayu M V (1DS21EC056),** respectively,hereby declare that the project work entitled "Design of CORDIC Core – RTL to GDS" has been independently done by us under the guidance of **Dr. Santhosh Kumar R, Assistant Professor,** ECE department and submitted in partial fulfillment of the requirement for the award of the degree of **Bachelor of Electronics & Communication Engineering** at **Dayananda Sagar College of Engineering**, an autonomous institution affiliated to VTU, Belagavi during the academic year 2024-2025.

We further declare that we have not submitted this report either in part or in full to any other university for the award of any degree.

AKASH V KASHYAP	**1DS21EC023**
AKSHAY N NAYAK	**1DS21EC025**
CHETAN PATIL	**1DS21EC055**
CHIRAAYU M V	**1DS21EC056**

PLACE: Bengaluru -78

DATE:

ACKNOWLEDGEMENT

The satisfaction and euphoria accompanying the successful completion of any task would be incomplete without the mention of people who made it possible and under constant guidance and encouragement the task was completed. We sincerely thank the **Management of Dayananda Sagar College of Engineering, Bengaluru.**

We express our sincere regards and thanks to **Dr. B G Prasad, Principal, Dayananda Sagar College of Engineering, Bengaluru.** His constant encouragement guidance and valuable support have been an immense help in realizing this project.

We express our sincere regards and thanks to **Dr. Shobha.K.R, Professor & Head, Department of Electronics & Communication Engineering, Dayananda Sagar College of Engineering, Bengaluru.** Her incessant encouragement guidance and valuable technical support have been an immense help in realizing this project. Her guidance gave us the environment to enhance our knowledge, and skills and to reach the pinnacle with sheer determination, dedication, and hard work.

We would like to express profound gratitude to our Project guide **Dr Santhosh Kumar R , Assistant Professor , Department of Electronics & Communication Engineering, Dayananda Sagar College of Engineering, Bengaluru** who has encouraged us throughout the project. **His/Her** moral support enabled us to complete our project work successfully.

We express our sincere thanks to Project Coordinators **Dr.S.Thenmozhi**, **& Mrs.Bindhu H.M,** of the **Department of Electronics and Communication Engineering** for their continues support and guidance. We thank all teaching and non-teaching staff of the Department of Electronics and Communication Engineering for their kind and constant support throughout the academic Journey.

AKASH V KASHYAP	**1DS21EC023**
AKSHAY N NAYAK	**1DS21EC025**
CHETAN PATIL	**1DS21EC055**
CHIRAAYU M V	**1DS21EC056**

ABSTRACT

The project focuses on the design and implementation of a highly configurable Coordinate Rotation Digital Computer (CORDIC) algorithm using physical design flows. The algorithm simplifies the computation of transcendental functions such as trigonometric, logarithmic, and hyperbolic functions through micro-rotations, utilizing additions, subtractions, and bit-shifting. Implemented in Verilog, this IP core supports three architecture styles—combinatorial, iterative, and pipelined—to cater to different performance requirements. Functional verification, gate-level synthesis, and physical design were performed using Cadence tools, ensuring efficient, robust, and scalable implementation.

The project includes a comprehensive physical design process covering floor planning, power planning, placement, and routing, followed by static timing analysis (STA) for timing verification. Each architecture demonstrates distinct trade-offs in terms of speed, area, and power. The iterative and pipelined designs achieved high operating frequencies of 100MHz, respectively, while maintaining low power consumption. The physical design was completed using Cadence Innovus, culminating in a verified and optimized layout.

Key metrics such as timing, power, and area were analyzed to ensure the design's adherence to performance specifications. The CORDIC IP core provides versatile functionality, including rotation and vectoring modes, enabling its use in applications like digital signal processing, telecommunications, and image processing. This modular and customizable design approach ensures adaptability for varied computational needs.

The CORDIC IP Core is a powerful solution for high-speed and low-power computation of mathematical functions. It has been validated through extensive simulations and analysis, making it an efficient choice for integration into complex VLSI systems. This work demonstrates the effectiveness of leveraging modern EDA tools and optimization techniques in developing scalable and high-performance IP cores.

Keywords: *CORDIC, IP Core, Physical Design, Static Timing Analysis, VLSI, Cadence Tools*

Table of Contents

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
CORDIC	Coordinate Rotation Digital Computer
RTL	Register Transfer Level
SDC	Synopsys Design Constraints
LEF	Library Exchange Forma
GDS	Graphic Data System
SPEF	Standard Parasitic Exchange Format
SDF	Standard Delay Format
PVT	Process Voltage Temperature
DRC	Design Rule Check
LVS	Layout Versus Schematic
CTS	Clock Tree Synthesis
GUI	Graphical User Interface
SI	Signal Integrity
VSS	Ground Voltage Supply
VDD	Power Voltage Supply

1. INTRODUCTION

1.1 Overview

The CORDIC algorithm, short for COordinate Rotation Digital Computer, is a computational method that simplifies the calculation of various mathematical functions like trigonometric, logarithmic, and square root operations. First introduced by Jack E. Volder in 1959, CORDIC is based on basic geometric principles and uses iterative calculations to perform complex mathematical operations. Over time, it has evolved into an efficient solution for hardware implementations, particularly in applications where simplicity and cost-effectiveness are critical.

One of the unique strengths of CORDIC lies in its ability to perform a wide range of mathematical computations using only basic operations like addition, subtraction, shifts, and comparisons. This makes it particularly useful for systems that lack hardware multipliers. By performing a series of small micro-rotations, the algorithm computes functions such as sine, cosine, arctangent, and others. These characteristics make it a practical choice for applications in signal processing, robotics, and telecommunications. Since its inception, the CORDIC algorithm has seen significant advancements, including improvements in architecture design and optimization for specific use cases. In 1971, John Walther demonstrated its versatility by showing that with minor adjustments; the same algorithm could handle a broader range of functions, including exponential and logarithmic calculations. This ability to unify the computation of multiple mathematical functions has contributed to its widespread adoption in areas like image processing, scientific computation, and embedded systems.

Today, CORDIC continues to be a valuable tool in many fields due to its simplicity and adaptability. It has found use in generating waveforms, designing digital filters, solving kinematic equations for robotics, and enabling three-dimensional vector transformations for graphics and animation.

At its core, CORDIC performs vector rotation**s** by rotating a vector in a Cartesian plane by a given angle. This is achieved through iterative calculations involving only addition, subtraction, comparison, and bit-shifting operations**.** These operations are simple and well-suited for hardware implementations, particularly in systems where more complex operations, such as multiplication, may not be available or are too costly.

CORDIC's iterative nature allows it to compute a variety of functions, such as **sine, cosine,** arc tangent, and square root**,** with each iteration refining the result. The angle of rotation in the algorithm is adjusted step-by-step, and each step uses the previous result to compute the next. The key advantage here is that all these computations can be carried out with just shifts and adds, making CORDIC an attractive alternative to traditional methods like polynomial expansions, which require more complex operations such as multiplications and divisions.

Because of the simplicity of its operations, CORDIC is particularly useful in hardware with limited resources**,** where efficient computation is necessary, such as in embedded systems, signal processing, and communication devices. Additionally, CORDIC can be adapted to various applications by modifying its iterative steps to compute different mathematical functions, depending on the specific needs of the system. This flexibility, combined with the algorithm's efficiency, makes CORDIC a widely adopted choice in many digital systems, especially when no hardware multiplier is available or when a compact, low-power design is essential

The CORDIC (Coordinate Rotation Digital Computer) algorithm is a well-established method for performing a wide range of mathematical operations, including trigonometric, hyperbolic, logarithmic, and square root calculations. Initially introduced by Jack Volder in 1959, CORDIC is particularly well-suited for hardware implementation due to its simplicity and efficiency. The algorithm operates using only addition, subtraction, bit-shifts, and compares—basic operations that are computationally inexpensive and can be easily realized in hardware. This makes CORDIC an ideal choice for systems that require

real-time calculations and high-speed data processing, such as in embedded systems, digital signal processing (DSP), telecommunications, and computer graphics.

One of the key features of CORDIC is its iterative nature. It works by rotating a vector in a coordinate system, successively reducing the angle of rotation in each iteration until the desired angle is achieved. Through this process, CORDIC can compute trigonometric functions such as sine, cosine, and tangent, as well as hyperbolic functions like sinh and cosh. The algorithm is also capable of calculating logarithmic and exponential functions, square roots, and even multiplication and division, making it extremely versatile. This flexibility allows CORDIC to be used in a wide variety of applications, all with the same basic architecture.The primary advantage of the CORDIC algorithm over other methods is its low hardware complexity. Unlike traditional algorithms such as the Taylor series or polynomial approximations, which require costly multiplications and divisions, CORDIC reduces the problem to a series of simple shift-and-add operations. These operations can be executed very quickly in hardware, especially in FPGA (Field-Programmable Gate Array) or ASIC (Application-Specific Integrated Circuit) implementations. This makes CORDIC highly suitable for systems that need to perform mathematical calculations on large datasets in real-time, such as digital filters, modulators, demodulators, and communications systems.

Another important benefit of CORDIC is its scalability. The accuracy of the results can be adjusted by increasing or decreasing the number of iterations. This feature makes it adaptable to various levels of precision, which is critical in hardware design where resource constraints such as memory and processing power are often a limiting factor. For example, a hardware implementation of CORDIC can be optimized for specific applications, where a balance between speed, accuracy, and power consumption is required. By adjusting the number of iterations, designers can achieve the necessary trade-offs for different applications without the need to redesign the core algorithm.

The CORDIC algorithm can be implemented in various modes, which further contributes to its versatility. In the "rotate" mode, CORDIC can compute the sine and cosine of a given angle, while in "vector" mode, it can compute the arctangent of a given ratio of two

numbers. These modes make it particularly useful for applications such as robotics, where angles and vector rotations are critical. Additionally, CORDIC is often used in applications where fast and low-power computation is a priority, such as mobile devices, satellites, and automotive systems. In conclusion, the CORDIC algorithm stands out for its ability to perform a wide range of mathematical functions with low computational complexity, making it highly suitable for hardware implementations. Their iterative natures, adaptability in terms of accuracy, and ease of integration into various systems have contributed to its widespread use in real-time digital applications. The fact that CORDIC can handle multiple functions with minimal hardware overhead sets it apart from other algorithms, such as Taylor series or lookup tables, making it an excellent choice for embedded and hardware-based systems.CORDIC's ability to compute multiple mathematical functions with a single hardware structure is another reason for its widespread adoption. In traditional software or hardware designs, different functions often require separate implementations. For example, trigonometric functions like sine and cosine might require different calculation methods than logarithmic or square root functions. CORDIC, however, uses the same iterative approach to solve all these functions, which simplifies the overall hardware design and reduces the need for multiple processing units. This consolidation leads to reduced complexity in both design and implementation, which is highly beneficial in resource-constrained environments such as embedded systems or mobile devices.

Furthermore, the algorithm's compatibility with fixed-point arithmetic is a key factor that enhances its usefulness in practical hardware applications. Many embedded systems and digital processors rely on fixed-point arithmetic for efficiency, as it allows faster computations with lower power consumption compared to floating-point arithmetic. CORDIC is well-suited for fixed-point operations because it primarily uses simple additions, subtractions, and bit shifts, which can be easily adapted to fixed-point representations.

1.2 Problem Statement

Older methods like the Taylor series were commonly used to calculate mathematical functions such as sine, cosine, and logarithms. While effective, these methods involve complex operations like factorials and polynomials, making them computationally intensive and slower. This made them less suitable for modern applications that require faster and more efficient hardware solutions.The CORDIC algorithm offers a simpler and more efficient alternative by using basic operations like addition, subtraction, and shifts instead of multipliers. This approach reduces hardware complexity and improves speed. However, earlier implementations of CORDIC lacked flexibility and were not optimized for modern physical design techniques, limiting their scalability and performance in advanced systems.

This project addresses these limitations by developing a highly configurable and improved version of the CORDIC algorithm. By integrating advanced physical design processes like power planning, placement, and routing, this design is optimized for high-performance applications and ready for tapeout. The highly configurable nature of this design makes it adaptable for a wide range of applications, including signal processing, robotics, and 3D graphics, while ensuring efficiency and scalability.

When implementing mathematical functions like trigonometric, exponential, logarithmic, or square root calculations in hardware or software, multiple algorithms are available. Below, we explore a few alternative methods and compare them with the CORDIC algorithm, highlighting why CORDIC was chosen for this project.

When designing systems to compute mathematical functions like trigonometric, exponential, logarithmic, or square root operations in hardware or software, various algorithms are available, each with its strengths and limitations.

Methods such as Taylor series expansion, polynomial approximations, lookup tables, and iterative techniques like Newton-Raphson provide alternatives for these calculations. However, each approach presents trade-offs in terms of computational complexity, memory usage, and hardware efficiency. The CORDIC algorithm stands out for its simplicity, hardware-friendliness, and versatility.

Taylor Series Expansion:

The Taylor series is a powerful mathematical tool that approximates functions by summing an infinite series of their derivatives evaluated at a specific point. For example, sine and cosine functions can be expressed as an infinite series of terms While the method is highly accurate for small ranges, it demands significant computational resources in hardware due to the need for division, multiplication, and factorial operations. Additionally, the method becomes memory-intensive when higher-order terms are included to increase precision. While useful for software applications, its complexity makes it less suitable for hardware like FPGAs.

The Taylor series provides a way to approximate functions using an infinite sum of terms based on the function's derivatives at a single point. For example, the sine function can be approximated as:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots.$$

Pros:

- Provides high accuracy for small values of xxx.
- Flexible, allowing the computation of a wide variety of functions.

Cons:

- Computationally expensive in hardware, requiring multiple multiplications and divisions.
- Memory-intensive for high precision due to the large number of terms needed.
- Not optimal for hardware like FPGAs due to the need for complex arithmetic operations

Polynomial Approximation (Chebyshev Polynomials)

Polynomial approximations, such as Chebyshev polynomials, simplify mathematical functions into a set of polynomials with coefficients that minimize error over a specified range. These methods offer a good trade-off between precision and complexity. However, their hardware implementation requires multiple multipliers and accumulators, which increases resource utilization and latency. Though effective for specific functions, they lack the general-purpose flexibility of CORDIC, which supports trigonometric, hyperbolic, and exponential functions using a unified algorithm. Polynomial approximations, such as Chebyshev polynomials, are used to approximate functions with a series of polynomials.: These polynomials minimize the approximation error over a specified interval. For example, for a function f(x)f(x)f(x), the Chebyshev approximation might look like:

$$f(x) \approx T0(x) + T1(x) + T2(x) + \cdots f(x)$$

where Tn(x) are the Chebyshev polynomials of the first kind.

Pros:

- Offers a good trade-off between accuracy and computational efficiency.
- Can be tailored to specific ranges, minimizing approximation error.

Cons:

- Requires hardware multipliers for polynomial evaluation, which increases complexity.

Lookup Tables (LUTs)

Lookup tables precompute values for mathematical functions and store them in memory, allowing functions like sine, cosine, or logarithm to be retrieved instantly. This method is extremely fast and efficient in applications where memory is abundant and computation speed is critical. However, LUTs are limited by their resolution and become impractical for applications requiring high precision or a large dynamic range. CORDIC overcomes this by computing values dynamically with a small memory footprint, making it ideal for resource-constrained environments.

Equation for Sine using LUT:

$$sin\ (x) \approx LUT[x]$$

Pros:

- Very fast, as the function evaluation is reduced to a memory lookup.
- Simple hardware implementation.

Cons:

- Requires large amounts of memory to store function values, limiting scalability for high precision.
- Limited by the resolution of the stored data; higher precision requires larger tables.

Why CORDIC Was Chosen?

While each of the alternative algorithms has its own strengths, CORDIC is particularly suited for hardware implementations because of its simplicity and efficiency. Unlike the Taylor series or polynomial approximations, which require multiple complex operations like multiplication and division, CORDIC uses iterative shift-and-add operations that can be easily parallelized, making it ideal for FPGA or ASIC-based implementations.

Additionally, CORDIC is highly flexible, capable of computing a variety of functions (trigonometric, hyperbolic, exponential, and square roots) with minimal changes to the core algorithm, unlike methods such as Newton-Raphson or LUTs, which are specialized for specific operations. This versatility, combined with its low resource usage and high speed, made CORDIC the best choice for this project.

CORDIC (Coordinate Rotation Digital Computer) was chosen for this project due to its unique advantages in terms of hardware efficiency and flexibility. One of the key reasons for selecting CORDIC is its ability to perform complex mathematical operations, such as trigonometric, logarithmic, and square root calculations, using only simple arithmetic operations—primarily addition, subtraction, and bit-shifting. This makes it well-suited for hardware implementations, especially in FPGA or ASIC designs where resources like multipliers and dividers are often limited or unavailable. CORDIC's iterative nature also allows for a highly efficient design that can be easily scaled for different levels of precision by adjusting the number of iterations.

Moreover, CORDIC can compute a wide range of functions with a single algorithmic structure, making it far more versatile compared to alternatives like Taylor series or polynomial approximations, which typically require different implementations for different functions. While methods such as Newton-Raphson or lookup tables are specialized and offer high-speed solutions for specific operations, they lack the flexibility of CORDIC in handling various mathematical functions with minimal hardware changes. This adaptability makes CORDIC particularly attractive for applications in signal processing, image processing, and telecommunications, where multiple functions might need to be computed on-the-fly. Furthermore, its ability to work with fixed-point arithmetic makes CORDIC well-suited for low-power and high-speed applications, reinforcing its suitability for embedded systems and real-time applications.

1.3 Objectives

The main objective of this project is to develop a highly configurable CORDIC IP core that can efficiently compute a variety of mathematical functions such as trigonometric, logarithmic, and exponential operations. By utilizing the CORDIC algorithm, the design simplifies hardware requirements by relying on basic operations like addition, subtraction, and shifts, reducing computational complexity and improving performance.

A crucial aspect of this project is to focus on physical design and preparing the CORDIC IP core for tapeout. This involves transforming the Verilog RTL code into a physical design, which includes essential steps such as floorplanning**,** clock tree synthesis (CTS)**,** and static timing analysis (STA)**.** These steps are vital for optimizing the design to meet power, area, and timing constraints. The final goal is to generate the GDS file, which represents the layout for fabrication.

Additionally, the project aims to provide a highly configurable architecture that allows users to customize the CORDIC core based on their specific needs. This flexibility ensures the core can be adapted for a wide range of applications, maintaining efficiency in both area and performance while meeting different requirements for speed, power, and resource usage.

.

1.4 Motivation

This project provides valuable hands-on experience with physical design and industry-standard tools. By going through the full design process, from writing RTL code in Verilog to generating the final GDS file, we gain practical knowledge in key areas like floorplanning**,** clock tree synthesis (CTS)**,** and static timing analysis (STA). These steps are essential for ensuring the design meets performance, power, and area requirements in real-world applications. Using tools such as Cadence Innovus provides direct experience with the same software commonly used in the industry, preparing us for real-world design challenges.

The project is also motivated by the need for efficient hardware solutions in fields like signal processing, communications, and scientific computing. Many applications in these areas require complex calculations like trigonometric and logarithmic functions. Traditional methods, such as using Taylor series, can be slow and resource-intensive. The CORDIC algorithm, however, simplifies these calculations using basic operations like addition and shifting, which makes it much more efficient and suitable for hardware implementations.

Additionally, there is a strong focus on developing highly configurable hardware. Today's devices need to adapt to a variety of requirements, from low-power systems to high-performance applications. The CORDIC IP core provides this flexibility, allowing users to fine-tune the design to meet specific performance, power, and area needs, making it ideal for a wide range of use cases.

The motivation behind this project stems from the increasing demand for efficient, high-performance digital circuits in modern electronic systems. With the rapid growth of applications in fields such as telecommunications, automotive, and consumer electronics, the need for optimized digital systems, such as processors and communication modules, has become more critical.

The CORDIC algorithm, known for its simplicity and efficiency in computing trigonometric, hyperbolic, and square root functions, offers a powerful solution to these challenges. Implementing a pipelined CORDIC processor ensures high throughput and performance, addressing the growing need for real-time processing in modern electronic devices.Another key motivation for this project is to explore and demonstrate the complete design flow for digital circuit design, from behavioral modeling to physical implementation. Using industry-standard tools such as Xilinx Vivado, Cadence Genus, and Innovus enables the student to gain hands-on experience with state-of-the-art design and verification techniques. By combining theoretical knowledge with practical application, this project aims to bridge the gap between academia and industry, equipping us with the skills needed for the rapidly evolving semiconductor industry. The ability to work with such advanced tools and methodologies is essential for future engineers aspiring to contribute to the development of cutting-edge technologies.

Finally, this project serves as an excellent opportunity to delve into the intricacies of digital design, including functional verification, synthesis, and physical design. The ability to optimize designs for power, area, and performance (PAP) and to generate a finalized GDS file for fabrication is a crucial skill in the semiconductor industry. The successful completion of this project will not only provide valuable insights into the entire design process but will also serve as a solid foundation for further research and development in the field of VLSI design, paving the way for the creation of more advanced and efficient digital systems in the future.

2. LITERATURE SURVEY

S.No	Paper/Study	Existing Approach	Improvement in Our Project
1.	Volder, J.E. (1959)	CORDIC introduced as a method to compute trigonometric functions using only addition, subtraction, and shifting.	Introduced the foundational pipelined algorithm. Implemented the physical design process for optimized performance and efficient resource utilization.
2.	Walther, J. (1971)	CORDIC expanded to handle a wider range of functions like logarithms, square roots, and exponentials.	We enhanced the implementation by making the CORDIC core highly configurable, supporting different modes like vector, rotate, and iterative operation.
3.	Zhang, L. et al. (2018)	Proposed optimized CORDIC implementations for higher speed, with various improvements in the iterative process.	We further optimized the CORDIC architecture, implementing floorplanning**,** CTS (Clock Tree Synthesis)**,** and STA (Static Timing Analysis) to ensure higher performance and better area and power efficiency.
4.	Nguyen, D. et al. (2020)	Introduced parallel CORDIC techniques to improve performance.	We introduced a pipelined design for faster computation and provided comprehensive physical design solutions such as power planning and placement.
5.	NPTEL CORDIC Algorithm (2021)	The NPTEL course covers the basic CORDIC algorithm for computing trigonometric functions and its hardware implementation.	Our project improves upon this by creating a highly configurable CORDIC IP core, adding advanced physical design techniques and preparing the design for tape-out using industry-standard tools like Cadence Innovus and Genus

1. Volder, J.E. (1959)

Existing Approach:

In 1959, Volder introduced the CORDIC (Coordinate Rotation Digital Computer) algorithm as an efficient method to compute trigonometric functions using only simple arithmetic operations—addition, subtraction, and bit-shifting. The algorithm works by performing iterative rotations on a vector in a Cartesian plane, with each rotation reducing the angle by a fixed amount. This approach is particularly advantageous in hardware, as it avoids the need for multiplication and division, which are more resource-intensive operations. By using only additions, subtractions, and shifts, CORDIC provided a computationally inexpensive method for calculating trigonometric functions, making it ideal for early digital computers with limited hardware resources and processing power.

Improvement in Our Project:

Our project builds upon Volder's foundational work by optimizing the CORDIC algorithm for modern applications. We enhanced the design by implementing a pipelined version of the algorithm, which allows the system to process multiple iterations simultaneously. This pipelined approach improves throughput and reduces latency, making the CORDIC core significantly faster and more efficient than Volder's original method, which operated in a serial manner. Additionally, the pipelined design allows for better scalability, enabling the core to handle more complex operations without a substantial increase in processing time. This improvement ensures that our CORDIC core can perform high-speed calculations in real-time applications, such as digital signal processing (DSP) and communications.

Enhanced Resource Utilization:

Another significant improvement in our project is the optimization of resource utilization through physical design techniques. By incorporating floorplanning, placement, and routing strategies, we have significantly reduced the area and power consumption of the CORDIC core while maintaining high performance. This is crucial for modern FPGA or ASIC designs, where efficient use of resources is key to achieving optimal performance in

power-constrained or area-limited environments. The pipelined CORDIC core, along with the physical design optimizations, allows the algorithm to be deployed in practical hardware systems with minimal overhead, offering a compact and power-efficient solution for a wide range of applications.

2. Walther, J. (1971)

Existing Approach:

In 1971, Walther expanded upon Volder's CORDIC algorithm by broadening its scope to calculate a wider range of functions, such as logarithms, square roots, and exponentials, in addition to the original trigonometric functions. Walther's approach allowed CORDIC to handle not only trigonometric calculations but also more complex mathematical operations, making it a versatile tool for hardware implementations. This expansion utilized the same iterative process of coordinate rotation, which ensured that the CORDIC algorithm remained efficient and fast, even as it was applied to more complex functions. The extension of CORDIC to a variety of functions marked a significant improvement in the practical applicability of the algorithm in different domains, such as scientific computing, control systems, and communications.

Improvement in Our Project:

Our project further advances Walther's expanded CORDIC by designing a highly configurable CORDIC IP core that supports multiple modes of operation, including vector, rotate, and iterative modes. This configurability enables the core to adapt to a range of applications, from calculating trigonometric functions in real-time DSP systems to more complex mathematical operations like logarithms and square roots in scientific computations. The ability to switch between these modes allows users to optimize the CORDIC core for specific tasks, maximizing both performance and resource efficiency. The introduction of such flexibility ensures that our design can handle a broader set of use cases compared to Walther's more rigid implementation, making it suitable for diverse hardware systems.

Optimized Performance and Resource Utilization:

Additionally, we implemented advanced design techniques to further optimize performance and resource usage. The use of pipelining, parallelism, and a configurable architecture ensures that our CORDIC core operates at higher speeds, uses less power, and can scale with more complex applications. We also implemented dynamic adjustments for precision and iteration count based on the specific needs of the application. These improvements make the CORDIC core more adaptable and efficient in modern hardware environments, enabling it to deliver higher performance in terms of both computation and resource consumption than Walther's original design.

3. Zhang, L. et al. (2018)

Existing Approach:

In 2018, Zhang et al. proposed several optimizations to enhance the performance of the CORDIC algorithm, particularly focusing on improving the iterative process. Their approach reduced the latency of the iterations and improved the throughput by refining the steps involved in each iteration. These improvements enabled the algorithm to perform faster while maintaining its simple hardware structure. Additionally, Zhang et al. explored the potential for optimizing CORDIC to achieve higher speed in hardware, making it a more viable solution for applications that require rapid computation, such as signal processing and high-frequency trading systems. Their research significantly advanced the CORDIC algorithm, providing a foundation for further performance improvements in future implementations.

Improvement in Our Project:

In our project, we have taken Zhang et al.'s work a step further by incorporating modern physical design techniques to optimize not just the algorithm itself, but the entire CORDIC core. We integrated floorplanning, Clock Tree Synthesis (CTS), and Static Timing Analysis (STA) into the design process, which ensures that the core operates at its highest possible frequency, meets timing constraints, and consumes less power. By implementing CTS, we

can manage clock distribution efficiently, while STA ensures that the design satisfies all timing requirements, preventing errors in signal propagation. This comprehensive approach enhances the core's performance and reliability, addressing the limitations of Zhang et al.'s optimizations, which were mainly focused on algorithmic improvements rather than the full hardware design.

Optimized Area and Power Efficiency:

We also paid special attention to optimizing the area and power consumption of the CORDIC core. Our design features strategic power planning techniques, such as power domain partitioning, power ring additions, and special routing to minimize power dissipation. The use of efficient placement and routing algorithms ensures that the core consumes minimal chip area while achieving high performance. By combining these physical design techniques with the algorithmic improvements from Zhang et al., we have created a CORDIC core that offers superior speed, efficiency, and scalability, making it well-suited for modern hardware applications that require high throughput and low power consumption.

4. Nguyen, D. et al. (2020

Existing Approach:

Nguyen et al. (2020) introduced parallel CORDIC techniques aimed at improving the performance of the algorithm by processing multiple iterations simultaneously. By utilizing parallelism, they were able to reduce the overall latency of the CORDIC computation, which allowed the algorithm to process multiple data points in less time. This improvement was particularly beneficial for applications that require real-time computation, such as radar systems and high-speed signal processing. Their approach demonstrated how parallel CORDIC can be leveraged to speed up the algorithm and enhance its ability to handle large datasets or perform complex calculations more quickly.

Improvement in Our Project:

We have further advanced Nguyen et al.'s parallel CORDIC approach by incorporating pipelined processing into the design. The pipelined structure ensures that multiple stages of the computation are processed concurrently, further reducing latency and increasing throughput. By allowing each iteration of the CORDIC algorithm to process in parallel, our design maximizes the performance of the core while maintaining its low computational overhead. This pipelined architecture, combined with the parallelism introduced by Nguyen et al., allows us to deliver even higher performance and efficiency, particularly in high-speed systems that require minimal delays, such as wireless communications and digital signal processing.

Physical Design Optimization:

In addition to parallelization, we have integrated comprehensive physical design strategies, such as power planning, placement, and routing, to optimize the CORDIC core for modern hardware platforms. Our power planning techniques ensure that the core operates efficiently without excessive power consumption, while the placement and routing optimizations reduce area and improve signal integrity. By applying these physical design techniques alongside the parallel and pipelined approaches, we have significantly enhanced the performance, scalability, and energy efficiency of the CORDIC core, making it suitable for real-time, high-performance applications in modern embedded systems and processors.

NPTEL CORDIC Algorithm (2021)

Existing Approach: The NPTEL course on the CORDIC algorithm provides a detailed introduction to the basic CORDIC algorithm, primarily focusing on its hardware implementation for computing trigonometric functions. The course presents the CORDIC algorithm as an efficient solution for hardware computation, requiring only addition, subtraction, and bit-shifting, making it ideal for systems with limited computational resources. The NPTEL course emphasizes the fundamental concepts behind the algorithm,

its iterative nature, and how it can be implemented in hardware for calculating functions like sine, cosine, and other trigonometric operations. The content provides a solid understanding of the algorithm and its application in simple hardware designs, but the course primarily focuses on the core mathematical functions and the basic principles without delving into more advanced design and optimization techniques.

Improvement in Our Project: Our project builds upon the foundational concepts covered in the NPTEL course, but it goes beyond the basic hardware implementation of the CORDIC algorithm. We have designed a highly configurable CORDIC IP core, which can operate in different modes such as vector, rotate, and iterative modes. This configurability allows our core to adapt to a wide range of applications, from simple trigonometric calculations to more complex operations like logarithms and square roots. While the NPTEL course provides the basic building blocks for CORDIC, our design improves upon it by offering greater flexibility and performance, tailored for modern applications that require high speed and low power consumption. Furthermore, we have incorporated advanced physical design techniques such as floorplanning, placement, power optimization, and timing analysis, which were not covered in the NPTEL course.

In addition to improving the algorithm's flexibility, we have focused on advanced physical design techniques to optimize the implementation for modern hardware platforms. Using industry-standard tools like Cadence Innovus and Genus, we have incorporated features like power domain partitioning, Clock Tree Synthesis (CTS), and Static Timing Analysis (STA) to ensure that our CORDIC core operates efficiently in terms of power consumption and performance. These design techniques help to ensure that the core meets timing constraints, reduces power dissipation, and makes efficient use of the available area on an FPGA or ASIC. The NPTEL course, while a good starting point for understanding the algorithm, does not address these advanced physical design considerations, which are critical for optimizing the performance and resource utilization of modern CORDIC cores. Our project takes the core CORDIC principles and adapts them to meet the demands of contemporary hardware, ensuring higher performance, lower power, and better scalability for real-world applications.

3. PROBLEM ANALYSIS & DESIGN

3.1 Block diagram and working principle of proposed system

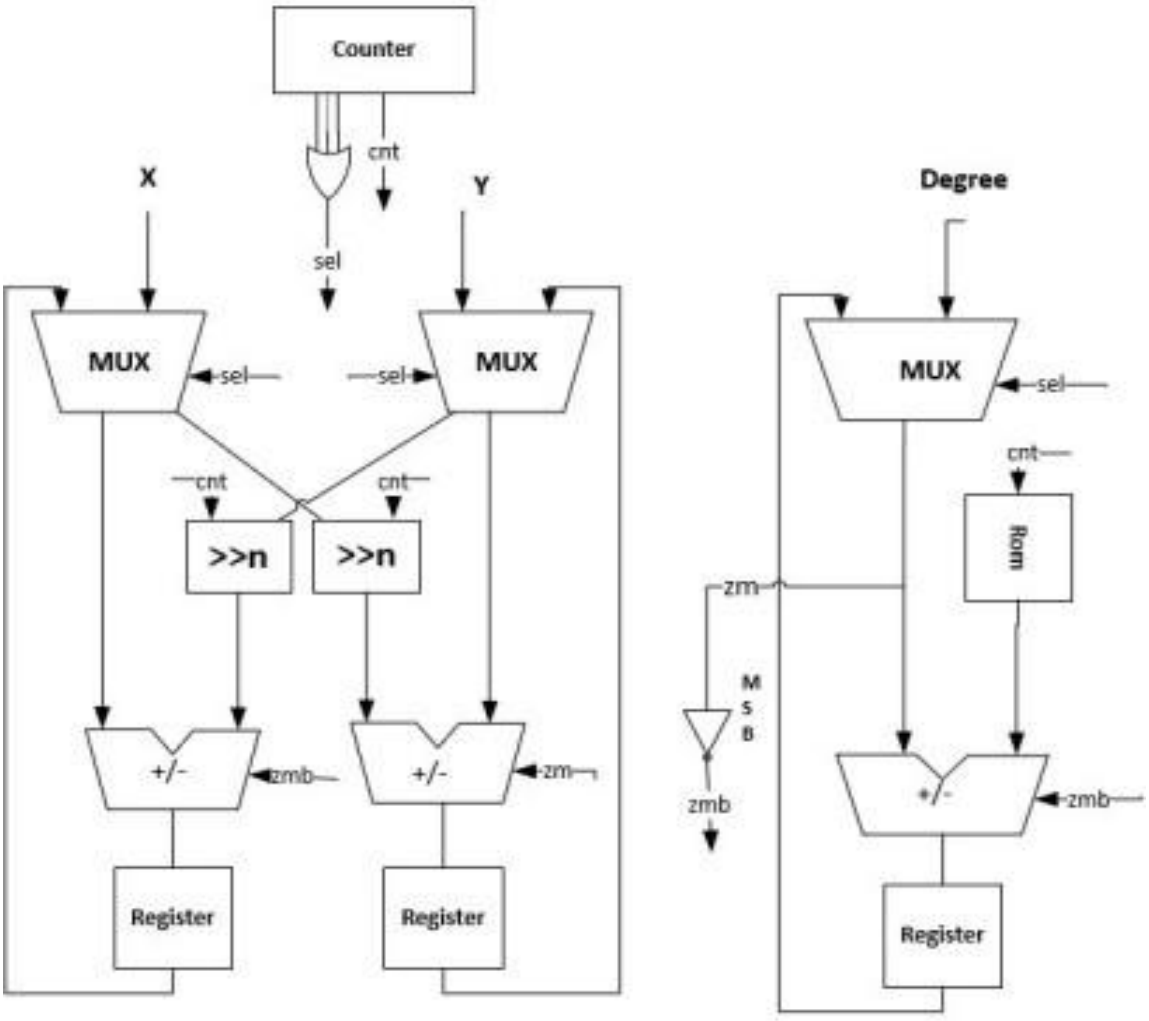


Figure 1: CORDIC Core Block diagram

Working: The CORDIC (COordinate Rotation Digital Computer) algorithm operates by rotating a vector in a two-dimensional plane to compute functions like sine, cosine, and square roots. Initially, a vector is set with specific coordinates, typically along the x-axis, and the goal is to rotate it by a series of small, pre-defined angles to achieve a final angle that represents the desired function. This is done using only simple operations such as addition, subtraction, and bit shifting, making it particularly efficient for hardware implementations where multiplications might be too costly or unavailable.

The algorithm proceeds iteratively, with each iteration performing a small rotation of the vector by a specific angle. These rotations are determined by powers of two, and at each step, the direction of rotation (clockwise or counter clockwise) is chosen based on whether the desired final angle is greater or smaller than the current angle achieved by the previous rotations. Each iteration improves the accuracy of the vector's direction, gradually bringing it closer to the target angle. By rotating the vector in small steps, the CORDIC algorithm can achieve precise results without the need for complex multiplications.As the iterations continue, the components of the vector (the x and y coordinates) are updated, each time refining the estimate of the sine and cosine of the angle being computed. After a set number of iterations, the vector will have rotated sufficiently, and the x and y coordinates will correspond to the cosine and sine of the desired angle. The iterative nature of CORDIC allows it to converge rapidly, providing an efficient way to compute these trigonometric functions. The process can also be adapted to compute other mathematical functions, such as square roots and logarithms, with the same iterative steps.

The number of iterations required for convergence depends on the level of precision needed for the specific application. In most cases, the algorithm is designed to stop once the angle error is smaller than a predefined threshold, meaning the result is accurate enough for practical purposes. The simplicity of the algorithm, using only basic arithmetic operations, makes it highly attractive for applications where computational resources are limited, such as in embedded systems or hardware devices with restricted processing power.

3.2 Software Design

The software design for a configurable CORDIC IP Core focuses on making it easy to use and efficient. It allows users to set up and test the CORDIC algorithm for tasks like calculating sin, cos, and arctan. The software supports different modes of operation and provides tools to simulate and analyze the hardware design, making it suitable for applications like signal processing and telecommunications.

The implementation of a configurable CORDIC IP Core involves multiple software tools, each serving a specific purpose in the design, verification, synthesis, and physical implementation stages. Below is a detailed explanation of the software used:

EDA Tools Used:

- Xilinx Vivado
- Cadence Incisive
- Cadence Genus
- Cadence Innovus

Figure 2: Logos of the software used Xilinx Vivado and Cadence

1. Xilinx Vivado :

Purpose:

- Used for initial RTL design, functional simulation, and FPGA implementation.
- Provides a development environment for synthesizing the CORDIC core and testing it on FPGA platforms.

Key Features:

- **RTL Design Entry:** Writing Verilog/VHDL code for the CORDIC algorithm.
- **Simulation:** Testing functionality using built-in waveform viewers.
- **Synthesis:** Converts the RTL code into gate-level netlists for FPGA implementation.
- **FPGA Deployment:** Allows testing the CORDIC core on hardware.

Where Used in the Project:

- **RTL Design:** The initial Verilog implementation of the CORDIC algorithm was written and simulated in Xilinx Vivado.
- **Functional Verification:** Basic waveforms were generated for checking the correctness of the sin, cos, and arctan outputs.

2. Cadence Incisive :

Purpose:

- Performs functional verification of the CORDIC design at the RTL level.

Key Features:

- **Simulation:** Verifies the design using testbenches written in SystemVerilog or Verilog.
- **Waveform Viewer:** Analyzes signals and debugging issues in the design.
- **Code Coverage:** Ensures all parts of the design are tested effectively.

Where Used in the Project:

- **Functional Verification:** After writing the RTL code, Cadence Incisive was used to simulate the design with comprehensive testbenches.
- **Waveform Analysis:** Ensured that the design behaved as expected under various input scenarios, such as different angles and vector inputs.
- **Debugging:** Helped identify and fix logical errors in the RTL code by analyzing simulation outputs.

3. Cadence Genus:

Purpose:

- Used for logic synthesis to convert the RTL design into a gate-level netlist.

Key Features:

- **Synthesis:** Maps the RTL code to standard cells from the technology library.
- **Timing Analysis:** Checks if the design meets timing constraints.
- **Optimization:** Minimizes area and power while maintaining performance.

Inputs Required:

1. **RTL Code:** The Verilog/VHDL design.
2. **SDC File:** Describes timing constraints for synthesis.
3. **Library File:** Contains the characteristics of standard cells.

Outputs Generated:

- Gate-level netlist.
- Timing reports and constraints.

4. Cadence Innovus:

Purpose:

- Performs physical design, including floorplanning, placement, routing, and clock tree synthesis.

Key Features:

- **Floorplanning:** Defines the placement of blocks on the chip.
- **Power Planning:** Creates power rings and straps to ensure proper power distribution.
- **Placement and Routing:** Arranges and connects the components of the design.
- **DRC and LVS Checks:** Ensures the layout complies with design and manufacturing rules.

Workflow in the Project:

1. Initial Design and Simulation: Xilinx Vivado.
2. Functional Verification: Cadence Incisive.
3. Logic Synthesis: Cadence Genus.
4. Physical Design: Cadence Innovus.
5. Timing Analysis: Cadence Innovus.
6. Automation: TCL scripts for Genus and Innovus.

Software design is a critical phase in the development lifecycle, bridging the gap between user requirements and implementation. It focuses on creating a structured framework that ensures functionality, scalability, and maintainability of the system. By employing various tools and methodologies, software design translates complex requirements into logical modules and interactions, enabling seamless integration with hardware or other software components. A robust design not only enhances efficiency but also ensures adaptability to future changes, laying a strong foundation for successful project execution.

4. IMPLEMENTATION

4.1 Overview of System Implementation

The system implementation of the CORDIC algorithm in our project follows a structured approach, starting from the design of the algorithm itself, progressing through functional verification, gate-level synthesis, and physical design, and finally culminating in optimized performance. The aim is to design a highly configurable, efficient CORDIC IP core that can be utilized in various applications such as signal processing, cryptography, image processing, and other computationally intensive tasks that require real-time performance. This section provides an overview of the different stages and components involved in implementing the CORDIC core, ensuring that the final product meets the desired specifications in terms of area, power, and performance.

The first step in the implementation process is designing the CORDIC algorithm in hardware using Verilog HDL (Hardware Description Language). The CORDIC algorithm, known for its efficiency in computing trigonometric, logarithmic, and other mathematical functions, is implemented in different configurations such as combinational, iterative, and pipelined versions. Each configuration has its own trade-offs in terms of speed, resource utilization, and complexity. The Verilog design encapsulates these different configurations, allowing for flexible operation and easy adaptation to various application requirements.

Once the Verilog design is written, functional verification is performed to ensure that the implementation of the CORDIC core behaves as expected. This involves simulating the design using tools like Cadence Incisive, which helps in generating waveform outputs and checking for correctness. Functional verification is crucial to detect and resolve any issues related to the logic of the design before proceeding to the next steps. After verifying that the design is free from errors, the next phase involves generating the gate-level netlist through synthesis, a process carried out using Cadence Genus. This step translates the high-level Verilog code into a gate-level representation, which can be used for further optimization and physical design.

Following synthesis, the physical design process begins with floorplanning, placement, and routing. Using Cadence Innovus, the floorplanning process ensures that the functional blocks of the CORDIC core are optimally placed on the chip, considering factors like area, timing, and power constraints. Placement then arranges the cells in the most efficient configuration, while routing establishes the interconnects between the different functional blocks. This step is critical to ensuring that the final design meets the required timing constraints and can operate at the desired clock speeds. The physical design phase is iterative and involves continuous adjustments to achieve the best performance, power, and area balance.

Power planning is another important aspect of the physical design process. The power planning stage involves partitioning the chip into different power domains and applying power rings, strips, and special routes to ensure that the power delivery network is optimized for the core's needs. This step also helps in managing power dissipation, which is a critical factor in modern system-on-chip (SoC) designs. In our project, power planning ensures that the CORDIC core consumes minimal power while maintaining high performance, making it suitable for mobile and embedded applications that require low energy consumption.

Finally, static timing analysis (STA) is performed using Cadence Tempus to ensure that the timing of the design meets the required specifications. STA helps verify setup and hold time violations, ensuring that the CORDIC core can operate reliably at high speeds. If any timing issues are detected, adjustments are made to the design through optimization techniques such as resizing cells or adjusting the clock tree. Once the design passes timing analysis, the CORDIC core is considered ready for tape-out, meaning it can be fabricated on silicon or deployed on FPGA hardware. Through this detailed system implementation process, we achieve a highly efficient, configurable, and optimized CORDIC IP core suitable for various applications.

4.2 Flowchart

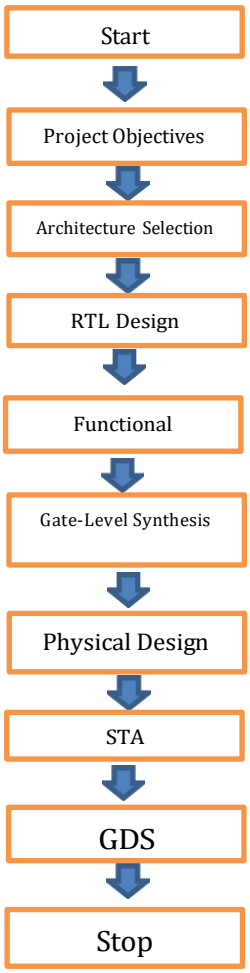


Figure 3: Flow of the project

1. RTL Design:

In our project, the initial step begins with writing the Register Transfer Level (RTL) design using Verilog. This RTL design defines the operations and data flow of the CORDIC algorithm, which is central to our project. We use RTL to describe how the input vectors, angles, and control signals are processed through the system, including the iterations for vector or rotation modes. The design includes the core logic for the iterative calculations, which are performed in parallel to ensure fast computation. Additionally, we define registers to hold the intermediate values of x, y, and angle, and the control logic ensures the correct sequence of operations across multiple clock cycles.

The flowchart begins with an initialization phase, where all necessary variables are set up. In the case of the CORDIC algorithm, this involves initializing the input variables, including the vector coordinates (x_in, y_in), the angle (theta_in), and the mode of operation (rotate or vector mode). Depending on the mode, the CORDIC core will perform different operations. After initialization, the first decision point in the flowchart determines which operation mode the CORDIC core is in. There are typically two modes in a CORDIC implementation:

- **Rotate Mode**: In this mode, the CORDIC algorithm is used to rotate a vector (x, y) by an input angle θ to compute trigonometric functions like sine and cosine.
- **Vector Mode**: In this mode, the CORDIC algorithm computes the angle for a given x, y input, which is used to calculate the arctangent (inverse trigonometric function).

2. **Synthesis for Gate-Level Representation**

After completing the RTL design, we proceed to Synthesis using industry-standard tools like Cadence Genus or Synopsys Design Compiler. The RTL code is synthesized into a gate-level netlist, where high-level operations are mapped to logic gates and flip-flops from a standard library of cells. During this phase, the tools optimize the design based on the constraints provided (e.g., area, speed, power). The synthesis process ensures that the CORDIC operations are efficiently translated into logic gates, and any timing violations or functional issues are addressed at this stage. This step is crucial for ensuring that the logic implemented in RTL is correctly represented as a physical design that can be used in subsequent stages of the project, such as placement and routing.

3. **Physical Design Implementation**

Following synthesis, the Physical Design phase begins, which involves transforming the synthesized gate-level netlist into a physical layout. In our project, the placement step

involves determining the location of logic gates and functional blocks on the chip, considering factors like area and power distribution.

The routing step ensures that all signals are connected properly between components, while maintaining timing and minimizing wire delays. The physical design process ensures that the final layout of the CORDIC core meets the necessary performance, power, and area constraints. In our case, specific techniques like Clock Tree Synthesis (CTS) and Static Timing Analysis (STA) are applied to optimize the design for better timing closure and efficiency.

4. GDSII Generation for Tape-Out

The final step in the design flow is generating the GDSII file, which represents the physical layout of the design and is used for chip fabrication. In our project, once the physical design is completed, the GDSII file is generated using tools like Cadence Innovus. This file contains detailed geometric information about the chip layout, including the placement of all components, routing of connections, and metal layers.

The GDSII file is the final output of the design process, and it is used to send the design to manufacturing for tape-out. This stage is critical as it ensures that the CORDIC core's physical implementation is ready for fabrication, and any adjustments made during physical design, such as optimizing power consumption or improving timing, are reflected in the final layout.

In our project, the GDSII file is generated using Cadence Innovus, which is an advanced tool for physical design. Innovus is specifically designed to handle the complex tasks involved in creating high-performance, power-efficient layouts for ASICs. Once the placement and routing steps are completed, Innovus ensures that the design meets the required specifications for timing, power, and area. It then performs final optimizations to address any remaining issues before generating the GDSII file. This file contains the exact layout of the CORDIC core, including detailed information on the metal layers, vias, and cell placements, all of which are essential for the chip fabrication process. Cadence Innovus also performs final checks for design rule violations and ensures the layout is manufacturable, thus preparing the design for tape-out, the final step before fabrication.

4.3 Algorithm

[1] Consider a unit circle of radius 'r'

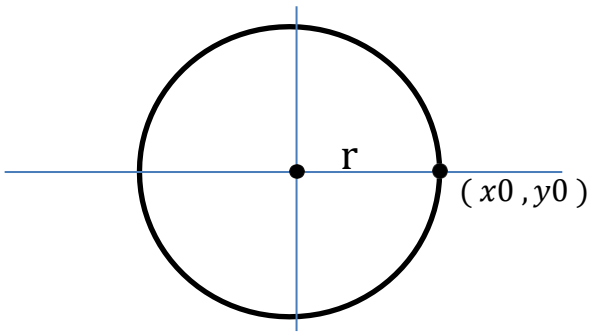


Figure 4: Circle with unit radius 'r' centred at x_0,y_0

[2] Initially consider $(x0, y0)$ as [1 ,0] , therefore $\Theta = 0$

[3] We need to find some angle i.e $\Theta + \alpha$

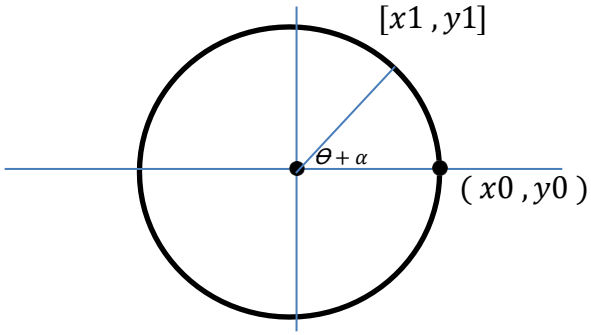


Figure 5: Angle to be measured $\Theta + \alpha$

[4] We need to find $[x1, y1]$ because $x1 = rcos(\Theta + \alpha)$, here r = 1 therefore

$x1 = cos(\Theta + \alpha)$, similarly y1=$sin(\Theta + \alpha)$

[5] To find $x1$ and $y1$, we perform Matrix operations as follows

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} * \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \qquad (1)$$

For first iteration $j = 1$ and $[xi, yi]$ are initial coordinates $[x0, y0]$

[6] The above equation can be modified by factoring a cosθ, and then we will get the following,

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \cos\Theta * \begin{bmatrix} 1 & -\tan\Theta \\ \tan\Theta & 1 \end{bmatrix} * \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \quad (2)$$

[7] Now, for nth iterations, equation becomes

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos\Theta_n * \begin{bmatrix} 1 & -\tan\Theta_n \\ \tan\Theta_n & 1 \end{bmatrix} * \begin{bmatrix} X_n \\ Y_n \end{bmatrix}$$

[8] After further reduction, due to tan property , for every half angle the tan value becomes half

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos\Theta_n * \begin{bmatrix} 1 & -S_n * (1/2^n) \\ S_n * (1/2^n) & 1 \end{bmatrix} * \begin{bmatrix} X_n \\ Y_n \end{bmatrix}$$

4.4 Implementation

RTL Design:

In our project, the RTL (Register Transfer Level) design was implemented in Verilog to describe the digital logic and functional flow of the CORDIC algorithm. This phase involved defining the key components such as data registers for x, y, and angle, and control logic for iterative operations. The RTL design was modular, supporting different operational modes like rotation and vector, making it highly adaptable. We designed the system with a focus on sequential operations, where each iteration refined the computation, utilizing simple arithmetic operations like addition, subtraction, and shifts. This approach made the design efficient for both FPGA-based prototyping and ASIC implementations.

After completing the Verilog code, we validated the functionality using Xilinx Vivado for simulation. The RTL simulations tested the correctness of the iterative computations, ensuring that outputs like sine, cosine, and arctangent matched expected results across various test cases. These simulations also verified the control signals for handling different modes and operational parameters, ensuring a robust design. The modularity of the RTL code facilitated easy debugging and reuse, enabling seamless integration into the synthesis flow and subsequent stages of physical design. This process ensured that the design was functionally sound and ready for synthesis into a gate-level netlist.

Inputs:

1. `clk` (Clock Signal)

- Type: `wire`

- Description: Synchronizes the operation of the pipeline stages. Each iteration of the CORDIC algorithm is computed on the rising edge of this clock.

2. rst` (Reset Signal)

- Type: `wire`

- Description: Initializes all internal registers (`x`, `y`, `z`) to zero when set high. Useful for resetting the system during startup or error conditions.

3. x_in` (16-bit Signed Input for X-coordinate)

- Type: `signed [15:0]`

- Description: Represents the initial X-coordinate of the input vector in Cartesian space. When calculating trigonometric functions, it is typically initialized to a scaled constant (CORDIC gain adjusted).

4. `y_in` (16-bit Signed Input for Y-coordinate)

- Type: `signed [15:0]`

- Description: Represents the initial Y-coordinate of the input vector in Cartesian space. Usually initialized to zero for calculating sine and cosine.

5. theta_in` (16-bit Signed Input for Angle)

- Type: `signed [15:0]`

- Description: The angle of rotation in fixed-point representation. Used to determine how much the input vector should rotate.

Outputs:

1. x_out` (16-bit Signed Output for X-coordinate)

- Type: `signed [15:0]`

- Description: Final X-coordinate after all iterations, representing the cosine value of the input angle (scaled by CORDIC gain).

2. `y_out` (16-bit Signed Output for Y-coordinate)

- Type: `signed [15:0]`

- Description: Final Y-coordinate after all iterations, representing the sine value of the input angle (scaled by CORDIC gain).

Verilog Code:

```
module cordic_pipeline(
    input wire clk,
    input wire rst,
    input wire signed [15:0] x_in,
    input wire signed [15:0] y_in,
    input wire signed [15:0] theta_in,
    output wire signed [15:0] x_out,
    output wire signed [15:0] y_out )
);

    parameter ITERATIONS = 16;

    reg signed [15:0] x [0:ITERATIONS-1];
    reg signed [15:0] y [0:ITERATIONS-1];
    reg signed [15:0] z [0:ITERATIONS-1];

    wire signed [15:0] arctan_table [0:15];
```

$assign\ arctan\_table[0] = 16'd11520;\ //\ atan(2\hat{}0)\ 45\ degrees$

$assign\ arctan\_table[1] = 16'd6800;\ //\ atan(2\hat{} - 1)\ 26.565\ degrees$

$assign\ arctan\_table[2] = 16'd3552;\ //\ atan(2\hat{} - 2)\ 14.036\ degrees$

$assign\ arctan\_table[3] = 16'd1804;\ //\ atan(2\hat{} - 3)\ 7.125\ degrees$

$assign\ arctan\_table[4] = 16'd906;\ //\ atan(2\hat{} - 4)\ 3.576\ degrees$

$assign\ arctan\_table[5] = 16'd455;\ //\ atan(2\hat{} - 5)\ 1.790\ degrees$

$assign\ arctan\_table[6] = 16'd227;\ //\ atan(2\hat{} - 6)\ 0.895\ degrees$

$assign\ arctan\_table[7] = 16'd114;\ //\ atan(2\hat{} - 7)\ 0.448\ degrees$

$assign\ arctan\_table[8] = 16'd57;\ //\ atan(2\hat{} - 8)\ 0.224\ degrees$

$assign\ arctan\_table[9] = 16'd28;\ //\ atan(2\hat{} - 9)\ 0.112\ degrees$

$assign\ arctan\_table[10] = 16'd14;\ //\ atan(2\hat{} - 10)\ 0.056\ degrees$

$assign\ arctan\_table[11] = 16'd7;\ //\ atan(2\hat{} - 11)\ 0.028\ degrees$

$assign\ arctan\_table[12] = 16'd4;\ //\ atan(2\hat{} - 12)\ 0.014\ degrees$

$assign\ arctan\_table[13] = 16'd2;\ //\ atan(2\hat{} - 13)\ 0.007\ degrees$

$assign\ arctan\_table[14] = 16'd1;\ //\ atan(2\hat{} - 14)\ 0.004\ degrees$

$assign\ arctan\_table[15] = 16'd0;\ //\ atan(2\hat{} - 15)\ 0.002\ degrees$

$genvar\ i;$

$generate$

$for\ (i = 0;\ i < ITERATIONS;\ i = i + 1)\ begin : cordic\_stage$

$always\ @(posedge\ clk\ or\ posedge\ rst)\ begin$

$if\ (rst)\ begin$

```
        x[i] <= 0;
        y[i] <= 0;
        z[i] <= 0;
    end else if (i == 0) begin

        x[i] <= x_in;
        y[i] <= y_in;
        z[i] <= theta_in;
    end else begin

        if (z[i-1] < 0) begin
            x[i] <= x[i-1] + (y[i-1] >>> i);
            y[i] <= y[i-1] - (x[i-1] >>> i);
            z[i] <= z[i-1] + arctan_table[i-1];
        end else begin
            x[i] <= x[i-1] - (y[i-1] >>> i);
            y[i] <= y[i-1] + (x[i-1] >>> i);
            z[i] <= z[i-1] - arctan_table[i-1];
        end
    end
  end
end
```

```
    endgenerate


    assign x_out = x[ITERATIONS - 1];
    assign y_out = y[ITERATIONS - 1];


endmodule
```

Test bench Code:

```
module tb_cordic_pipeline;
    reg clk;
    reg rst;
    reg signed [15:0] x_in;
    reg signed [15:0] y_in;
    reg signed [15:0] theta_in;
    wire signed [15:0] x_out;
    wire signed [15:0] y_out;


    cordic_pipeline cordic_inst (
        .clk(clk),
        .rst(rst),
        .x_in(x_in),
        .y_in(y_in),
        .theta_in(theta_in),
```

```
    .x_out(x_out),
    .y_out(y_out)
);

initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    rst = 1;
    #10;
    rst = 0;

    x_in = 16'd32768;
    y_in = 16'd0;
    theta_in = 16'd0;
    #100;

    x_in = 16'd32768;
    y_in = 16'd0;
    theta_in = 16'd11520;
    #100;
```

$x\_in = 16'd32768;$

$y\_in = 16'd0;$

$theta\_in = 16'd23040;$

#100;

$x\_in = 16'd32768;$

$y\_in = 16'd0;$

$theta\_in = -16'd11520;$

#100;

$x\_in = 16'd32768;$

$y\_in = 16'd0;$

$theta\_in = 16'd46080;$

#100;

$x\_in = 16'd32768;$

$y\_in = 16'd0;$

$theta\_in = 16'd7680;$

#100;

$stop;

end

```
        initial begin
$monitor("Time: %0t | X_in: %0d | Y_in: %0d | Theta_in: %0d | X_out: %0d | Y_out: %0d",
                $time, x_in, y_in, theta_in, x_out, y_out);
        end

    endmodule
```

The functional verification of the CORDIC pipeline module was performed using a testbench designed to rigorously test its behavior under various input conditions. The testbench simulated different combinations of input vectors (`x_in`, `y_in`) and angles (`theta_in`) to validate the accuracy of the sine and cosine outputs. The RTL code for the CORDIC pipeline was developed and synthesized using Xilinx Vivado, leveraging its robust simulation and debugging tools to ensure proper functionality and timing compliance before proceeding to physical design stages.

The testbench in Xilinx Vivado provided comprehensive verification by simulating the CORDIC pipeline across multiple iterations and edge cases, ensuring its robustness and reliability. The outputs (`x_out`, `y_out`) were cross-verified against expected results from theoretical calculations to confirm the correctness of the trigonometric operations. Vivado's waveforms and analysis tools facilitated detailed debugging, allowing us to optimize the RTL code for better performance and resource utilization, ensuring a smooth transition to the synthesis and implementation phases.

Functional Verification using Cadence Incisive:

Functional verification is a critical step to ensure the RTL design behaves as intended under different scenarios. Using Cadence Incisive, we performed comprehensive simulations of the CORDIC pipeline to verify its functionality, correctness, and edge-case handling. Incisive's robust simulation capabilities provided insights into the design's behavior at a detailed level, enabling us to validate the iterative computations and outputs against expected results for various inputs. Testbenches were developed to simulate a wide range of input cases, including boundary conditions, to confirm the design's accuracy and resilience.A key reason for leveraging Cadence Incisive was to compare its results and performance with earlier simulations done in Xilinx Vivado. This allowed us to check for consistency across different verification environments. Using Incisive, we also generated detailed coverage reports, ensuring all paths and conditions in the RTL code were exercised during testing. The tool's advanced debugging features facilitated faster root-cause analysis for any discrepancies identified during simulation, ensuring a highly reliable design.

By verifying the RTL code in both AMD and Cadence environments, we established a robust cross-platform validation approach. This ensured that the design would function reliably in subsequent stages, such as synthesis and physical implementation. The use of Cadence Incisive enhanced confidence in the design's functionality, bridging any potential gaps between development and industry-standard verification practices.

In addition to standalone verification, we compared the simulation results obtained from Cadence Incisive with those from Xilinx Vivado. This cross-platform verification ensured that the functionality remained consistent across different design flows. Cadence Incisive also provided advanced coverage metrics, such as code coverage and functional coverage, to ensure that all parts of the RTL code were exercised during simulation. This comprehensive verification process ensured that the CORDIC pipeline design was functionally correct and ready for the subsequent synthesis and physical design phases.

Commands to visualize the Waveforms in Cadence incisive

- First step is to compile the Verilog design.
- Second step is to elaborate and optimize the design.
- Third step is to run the simulation.

Setup the design environment by clicking terminal and invoke into the Virtuoso environment.

Step:1 Create logical library by the following procedures:

a) mkdir < directory name>- Create library directory and provide a logical library name.

b) vi <file name>.v – To write the Verilog code

c) vi <file name>_tb.v – To write the test bench

d) vi cds.lib – Create cds lib for mapping and make the following entries in new tab DEFINE <directory name>_lib ./<directory name>.lib

e) mkdir <directory name>.lib - create a work directory for logical mapping.

f) vi hdl.var – Create variable library and make the following entry in new tab DEFINE WORK <directory name>_lib

Step:2 Compilation process – Following commands are used for compiling design and test bench files.

a) ncvlog <file name>.v –mess

b) ncvlog <file name>_tb.v –mess

Step:3 Elaborate process- elaboration switches comes in handy to access read/write and connectivity access.

a) ncelab <testbench module name> –access +rwc –mess

Step:4 Simulate the design

a) ncsim <testbench module name> –gui

Step:5 NCSIM Design Browser window pops up

a) Select the< testbench module name> instance and click SEND the SELECTED SIGNALS TO THE TARGETED WAVEFORM WINDOW.

b) In the waveform window press RUN.

c) Now you will be able to visualize the waveforms

Synthesis:

The synthesis process is crucial for transforming the RTL design into a gate-level representation that can be implemented in hardware. To perform synthesis using Cadence Genus, several key input files are required. First, the RTL code (typically written in Verilog or VHDL) is the primary input, as it defines the logic and functionality of the design. Additionally, constraints files, such as the *sdc (Synopsys Design Constraints)* file, are essential. These files specify timing constraints, input-output relationships, clock definitions, and other physical design constraints that guide the synthesis process. These constraints ensure that the final gate-level design meets the desired performance and area requirements.

Another important input for Cadence Genus synthesis is the technology library files, which include information about the available cells and their characteristics in the chosen fabrication process (e.g., 7nm or 65nm). These libraries provide information about cell delays, area, power, and other properties, which help the synthesis tool optimize the design for the target technology. Additionally, for power analysis, a power constraints file may be used to guide power optimization during synthesis. With these input files, Cadence Genus can efficiently map the RTL design onto the chosen technology, optimizing it for timing, area, and power before moving to the next stages of implementation.

SDC file:

$set\ sdc\_version$ 2.0

$set\_units\ -capacitance$ $1000fF$

$set\_units\ -time$ $1000ps$

 # Set the current design

current_design cordic_pipeline

create_clock −name "clk" −period 10.0 *−waveform* {0.0 5.0} [*get_ports clk*]

set_clock_gating_check −setup 0.0

set_wire_load_mode "enclosed"

1. **set sdc_version 2.0**`: This command sets the version of the SDC (Synopsys Design Constraints) file to version 2.0, which defines timing and design constraints for the synthesis tool.

2. **set_units -capacitance 1000fF**` and `**set_units -time 1000ps** : These commands set the units for capacitance to femtofarads (fF) and time to picoseconds (ps) for proper design interpretation.

3. **current_design cordic_pipeline**`: This specifies that the current design being worked on is "cordic_pipeline," guiding the synthesis tool to focus on it.

4. `**create_clock -name "clk" -period 10.0 -waveform {0.0 5.0} [get_ports clk]**`: This defines the clock for the design, with a 10 ns period and a waveform that transitions between 0 and 5 ns.

5. **set_clock_gating_check -setup 0.0**`: This command disables clock gating checks during setup to allow flexibility in the design's clocking.

6. **set_wire_load_mode "enclosed"`:** This sets the wire load model to "enclosed," which helps estimate wire capacitance for routing optimization.

TCL File:

```
read_hdl /home/vlsi/Cordic/cordic.v

read_libs /home/install/FOUNDRY/digital/45nm/dig/lib/slow.lib

elaborate cordic_pipeline

syn_generic

syn_map

read_sdc cordic.sdc

syn_opt

# gui_show

# gui_hide

check_design

check_timing_intent

report_qor > cordic_qor.rep

report_timing > cordic_timing.rep

report_power > cordic_power.rep

report_area > cordic_area.rep

write_netlist cordic_pipeline > cordic_synth.v

write_sdc > cordic_sdc.sdc
```

1. `**read_hdl /home/vlsi/Cordic/cordic.v**`: Reads the Verilog HDL file for the CORDIC design, which contains the logic description of the module.

2. **read_libs /home/install/FOUNDRY/digital/45nm/dig/lib/slow.lib**: Loads the technology library file (slow.lib) for synthesis, providing cell definitions for the target process (45nm).

3. **elaborate cordic_pipeline**`: Performs elaboration of the design, which means interpreting the Verilog code and resolving design objects.

4. **syn_generic**: Generates a generic RTL representation of the design for synthesis.

5. `**syn_map**`: Maps the RTL design to specific cells in the target technology library (45nm), optimizing for area and timing.

6. **read_sdc cordic.sdc**`: Reads the SDC (Synopsys Design Constraints) file, which contains timing and physical constraints for the design.

7. **syn_opt**: Performs optimization of the synthesized design to improve timing, area, and power.

8. **check_design**: Verifies that the design is correct and all logical components are in place.

9. **check_timing_intent:** Ensures that the design's timing constraints are met and properly implemented.

10. `**report_qor > cordic_qor.rep`:** Generates a Quality of Results (QoR) report, which includes overall design metrics like area, timing, and power.

11. **report_timing > cordic_timing.rep**`: Generates a detailed timing report showing the setup and hold times of the design.

12. `**report_power > cordic_power.rep**`: Generates a power report showing the estimated power consumption of the design.

13. **report_area > cordic_area.rep`:** Generates an area report showing the total area used by the design in the layout.

14. **write_netlist cordic_pipeline > cordic_synth.v`:** Writes the synthesized netlist to a Verilog file (`cordic_synth.v`), which contains the final design in terms of gates and connections.

15. **write_sdc > cordic_sdc.sdc`:** Writes the design constraints (SDC) to a new file (`cordic_sdc.sdc`), which is used in the subsequent steps of implementation.

After running the synthesis process in Cadence Genus, we obtain several important output files. These include the synthesized Verilog netlist (`cordic_synth.v`), which represents the final RTL design mapped to the target technology cells, and the SDC file (`cordic_sdc.sdc`), which contains the timing and design constraints applied during synthesis. These files are critical for the next steps in the design flow, as they provide a detailed and optimized representation of the design's logic and constraints.

The synthesized Verilog netlist and SDC files generated by Genus are then used as input for Cadence Innovus, the tool responsible for the physical design of the chip. Innovus takes these files to perform the placement and routing of the cells, ensuring that the design meets the timing, area, and power requirements. The SDC file guides Innovus in applying the necessary timing constraints during the physical design process, while the synthesized netlist ensures that the logic is correctly implemented on the physical layout.

Physical Design:

The next step in the design process is physical design, which is carried out using Cadence Innovus. This phase involves translating the synthesized netlist and constraints into a physical layout on the chip. Innovus handles key tasks like floorplanning, placement of cells, clock tree synthesis, routing, and optimizing for performance, power, and area. Using the Verilog netlist and SDC files from Genus, Innovus ensures that the design meets all timing and design constraints, preparing it for tape-out and fabrication.

Physical design begins with creating the intended digital design and its testbench, typically in Verilog (.v format). The first step is to verify the functionality of the design using functional simulation tools such as the Incisive Simulator (with commands like ncvlog, ncelab, and ncsim). After ensuring the functionality, the design undergoes synthesis to generate a gate-level netlist and other required files like the Block Level SDC, Liberty file (.lib), and LEF files, which are crucial for the physical design process. The gate-level netlist is the result of synthesis and provides a description of the design at the logical gate level, while the SDC file specifies the constraints for timing, clock domains, and resets. Liberty files are used to define the cell characteristics (timing, power, etc.), and LEF files provide information about the layer-specific

Inputs for Physical Design

1. Gate Level Netlist: This is the output of the synthesis stage and serves as the foundation for physical implementation.

2. Block Level SDC: The timing constraints file generated during synthesis to guide the design implementation.

3. Liberty File (.lib): Captures timing, power, and functional characteristics of standard cells.

4. LEF Files (Layer Exchange Format): Contains abstract physical information about standard cells, including pin positions, layout layers, and blockages.

Outputs from Physical Design

1. GDS II File: The final physical design in Graphical Data Stream format for fabrication.

2. SPEF (Standard Parasitic Exchange Format) and SDF (Standard Delay Format): Used for post-layout timing and parasitic extraction verification.\

Initiating Innovus

1. Ensure the synthesis is complete for the target design and open a terminal in the corresponding workspace.

2. Start the Cadence tools using the command: `Innovus` and press Enter.

3. The Innovus GUI opens, and the terminal enters the Innovus command prompt, ready for tool-specific commands.

Physical Design Stages

Physical design is carried out in five key stages:

1. Design Creation: Write the RTL design and verify its functionality with a testbench.

2. Floor Planning: Define the core area, IO boundaries, and pin placement.

3. Power Planning: Establish a robust power distribution network.

4. Placement: Position standard cells and IO pins efficiently.

5. Clock Tree Synthesis (CTS): Design the clock distribution network.

6. Routing: Physically connect all components using metal layers.

Importing the Design

To begin, import the design into Innovus using the mandatory inputs:

- Use script files (.globals, .view/.tcl) or GUI options.

- For GUI: Navigate to File -> Import Design, load netlist, LEF, and SDC files, and set the "Top cell" to "Auto Assign."

- Liberty files with PVT (Process, Voltage, Temperature) combinations must also be included:

- Slow.lib: Models maximum delays under worst-case conditions.

- Fast.lib: Models minimum delays under best-case conditions.

These inputs allow for setup and hold timing analysis using delay corners. Delay corners are combinations of library sets and RC (resistance-capacitance) corners.

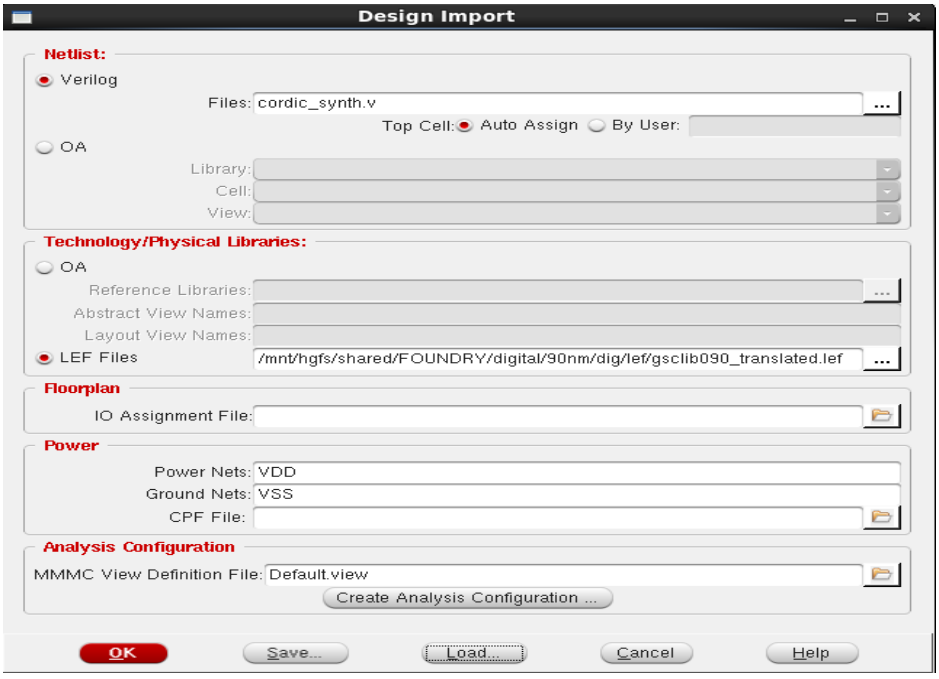


Figure 6: Design import window

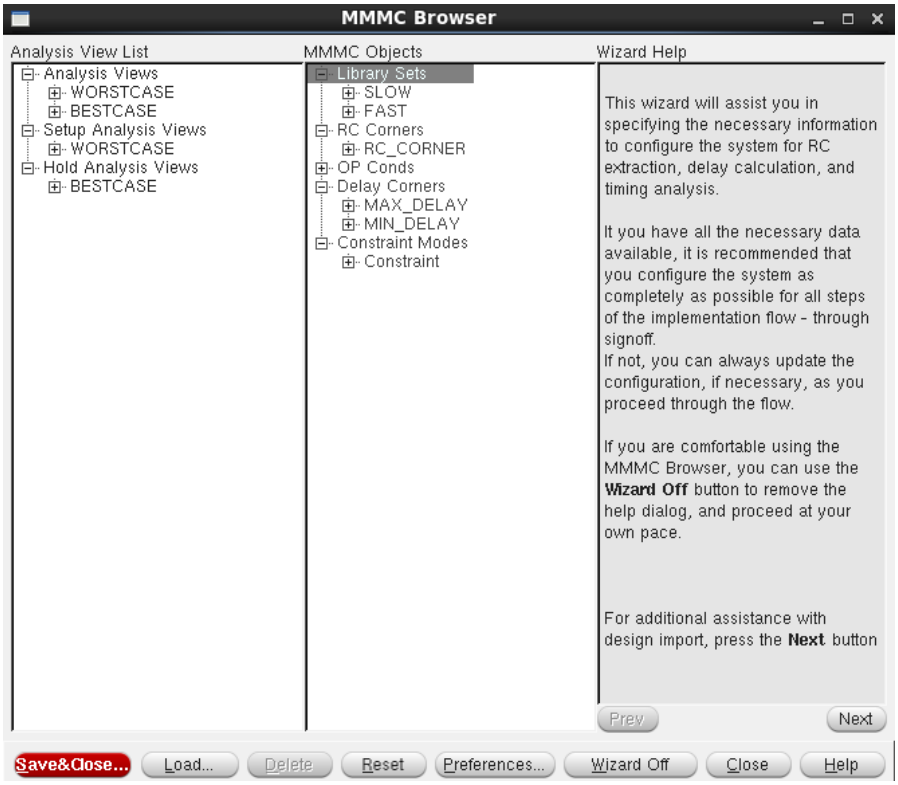


Figure 7: MMMC Window

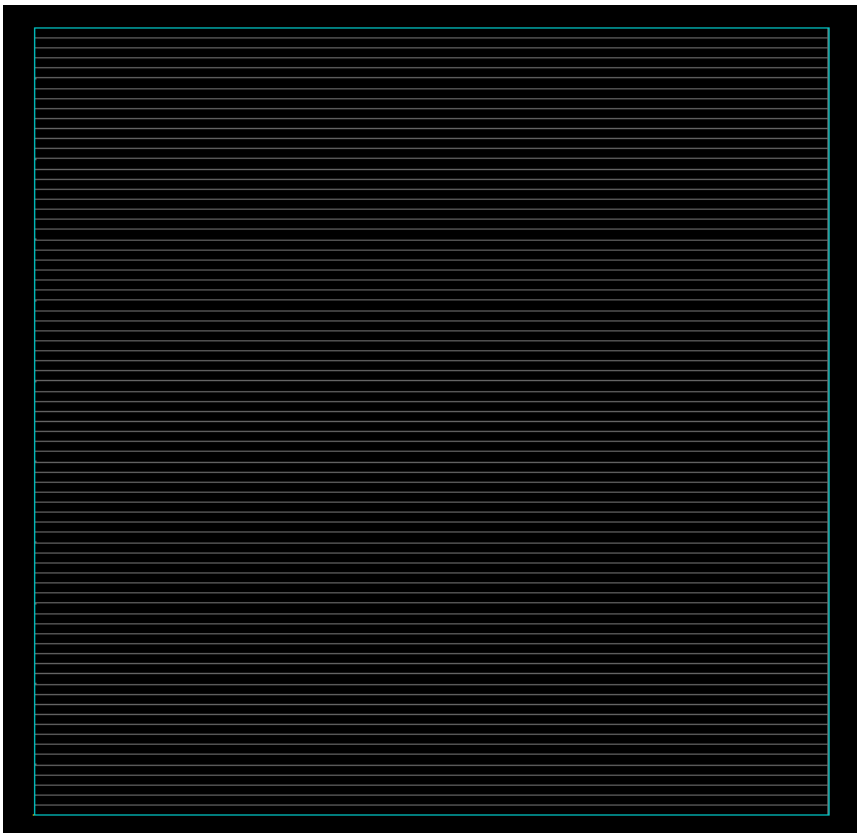


Figure 8: Blank Die

Slow.lib and Fast.lib:

In the context of physical design, slow.lib and fast.lib are two different variations of the Liberty file, which contains the timing characteristics of standard cells under different Process, Voltage, and Temperature (PVT) conditions. The slow.libnfile represents the timing data under the worst-case conditions, typically using a slow process corner, lower voltage, and higher temperature. This file is used for worst-case analysis, where the goal is to ensure that the design will meet timing constraints even under suboptimal conditions. It models slower cell delays, which helps ensure that the design will function correctly in the most challenging operational environments.

On the other hand, fast.lib contains the timing data for the best-case conditions, typically representing a fast process corner, higher voltage, and lower temperature. This file models the fastest possible cell delays, ensuring that the design performs optimally under ideal conditions. Fast.lib is used for best-case analysis, which helps to validate the design's performance under the most favorable operating conditions. Both slow.lib and fast.lib are crucial for performing static timing analysis (STA), where the design is validated for setup and hold violations, helping to ensure that the circuit will work correctly in both worst-case and best-case scenarios.

Floor Planning

The next stage of physical design involves floorplanning, which refers to the arrangement of the design's core area and cell placement. The design space is divided into a "core area" where all the standard cells and pins will be placed, along with "power planning" to ensure that there is a proper distribution of power throughout the design. During floorplanning, one specifies the aspect ratio, core utilization, and channel spacing. Floorplan constraints are applied to allocate sufficient space for the design’s modules while maintaining design efficiency. Power planning includes adding global power rings and power strips around the core to ensure that power is evenly distributed across the design. These components help minimize resistance and optimize the power delivery network for the standard cells.

Define the aspect ratio, core utilization, and channel spacing between the core and IO boundaries:

1. Set Aspect Ratio (height-to-width ratio of the core).

2. Define Core Utilization (percentage of the core area used for placement).

3. Adjust Channel Spacing to ensure proper routing and IO alignment.\

Unplaced pins appear as yellow blocks and must be positioned along IO boundaries.

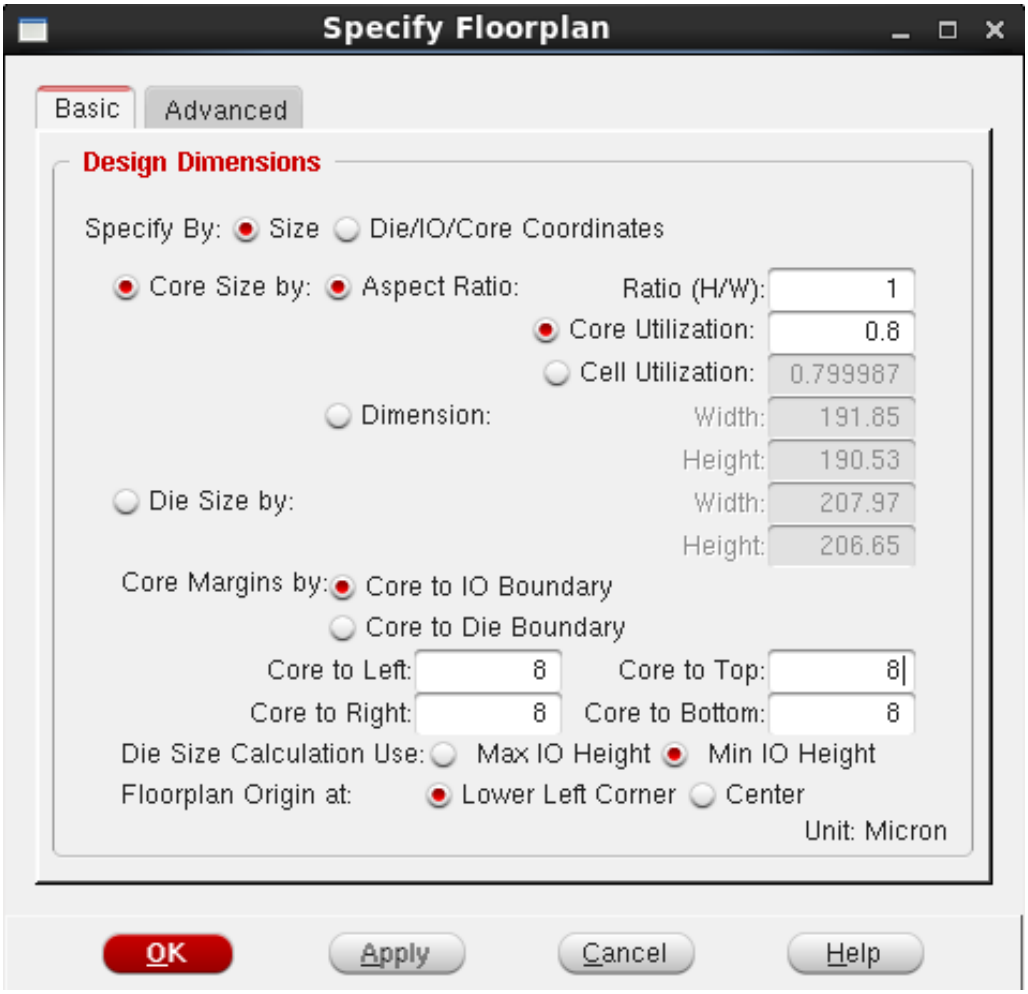


Figure 9: Floorplan Specifications window

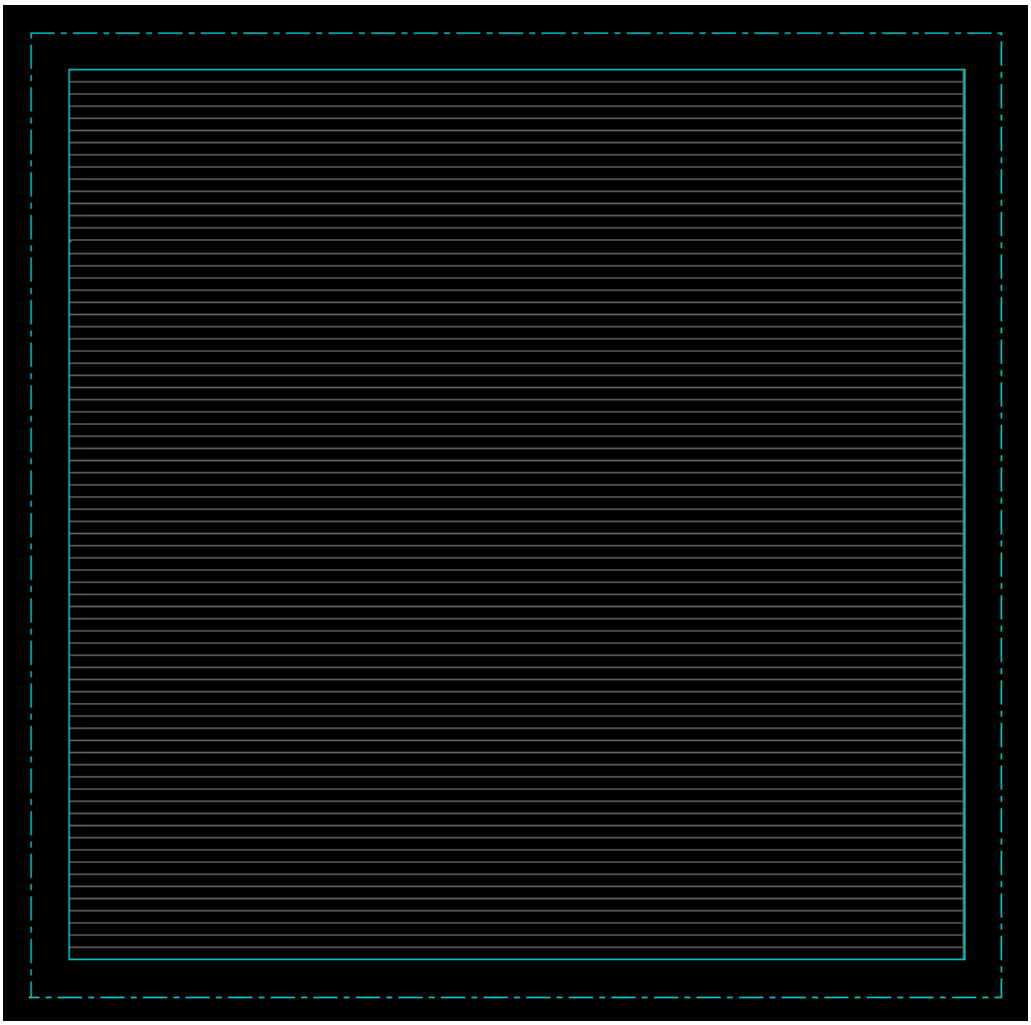


Figure 10: Post Floorplan view of die

Power Planning

Steps under power planning:

1. Connect global net connects

2. Adding power rings

3. Adding power strips

4. Special Route

-Under connect global net connects, we create 2 pins, one for VDD and one for VSS. Connecting to corresponding global nets as mentioned in globals file/power and ground nets.

1. Select Power -> Connect globals nets to create "Pin" and "connect to global net" as shown and use "Add to list".

2. Click on "Apply" to direct the tool in enforcing the pins and net connects to design and then close the window.

-In order to tap in power from a distant power supply, wider nets and parallel to improve efficiency. Moreover, the cells that would be placed inside the core area are expected to have shorter nets for more resistance.

-Hence power rings [Around core boundary] and power strips [Across core boundary] are added which satisfies the above conditions.

-Select power -> power planning -> Add ring to add power rings around core boundary.

-Select the nets from browse option or directly type in the global net names separated by a space being case and spelling sensitive.

-Select the highest metal marked 'H' [Horizontal] for top and bottom and metals marked 'V' [Vertical] for right and left. This because highest metals have highest widths thus lower resistance.

-Click on update after the selection and "Set Offset: Center in Channel" in order to get the minimum width W and minimum spacing of the corresponding metals and then click "OK".

-Similarly power strips are added using similar content to that of power rings.

Factors to be considered under power strips:

1. Nets

2. Metal and its direction

3. Width and spacing [Updated]

4. Set to set distance = (Minimum Width of metal + Minimum spacing) x 2

-On adding power strips, the power mesh set up is complete as shown. However, there are no vias that could connect metal 9 or metal 8 directly with metal 1 [VDD or VSS of standard cells are generally made up of metal 1].

-The connection between the highest and lowest metals is done through stacking of vias done using "Special Route".

-To perform Special Route, select Route -> Special Route -> Add nets -> OK.

-After this special route is complete, All the standard cell rows turn to the Color coded for the Metal 1.

The complete Power Planning process makes sure every Standard cell receives enough power to operate smoothly.

The power distribution network ensures that all cells receive sufficient power:

1. Global Nets: Create VDD and VSS connections to global power nets.

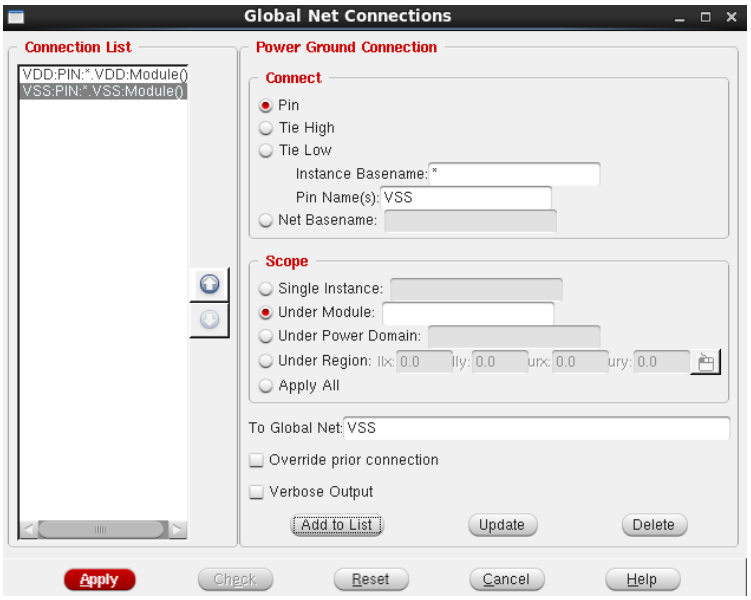


Figure 11: Global net connections window

2. Power Rings: Surround the core boundary with metal rings to distribute power.

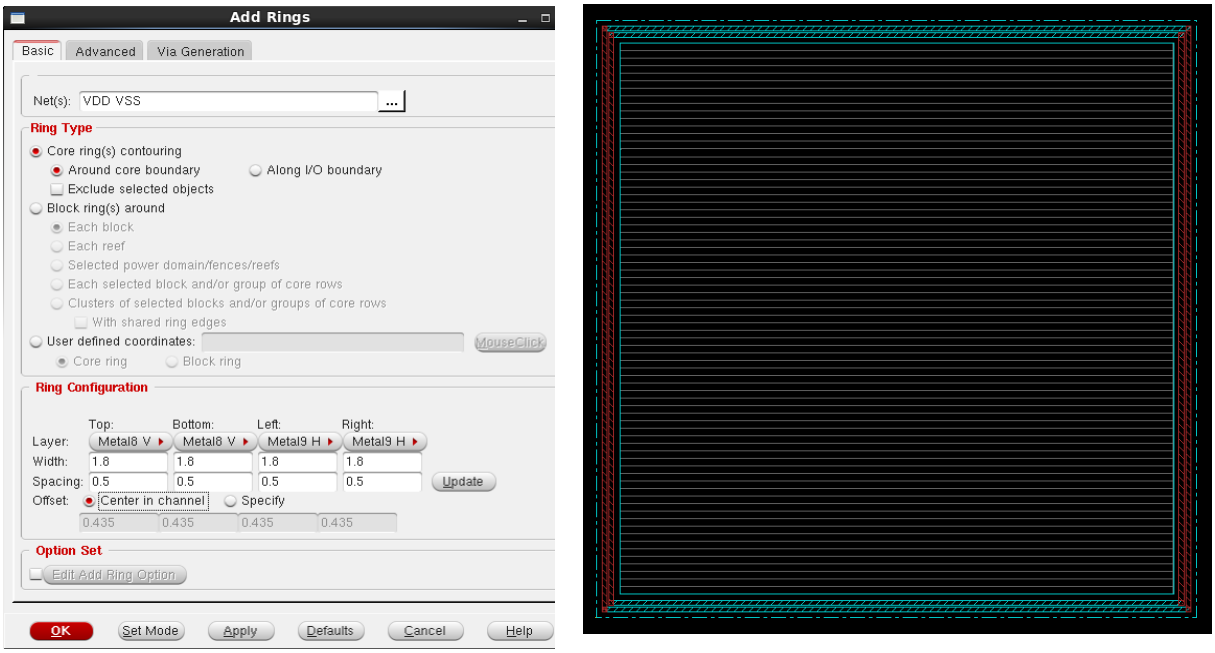


Figure 12: Addition of the metal rings to the die

3. Power Strips: Lay down horizontal and vertical strips across the core area for localized power supply.

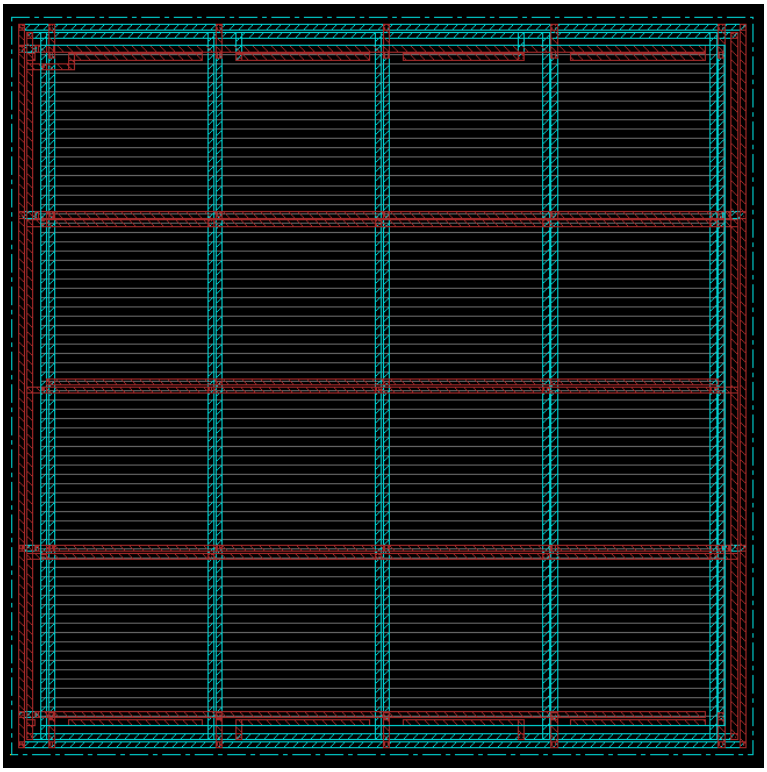


Figure 13: Power stripped Die

4. Special Routes: Add vias to connect upper and lower metal layers.

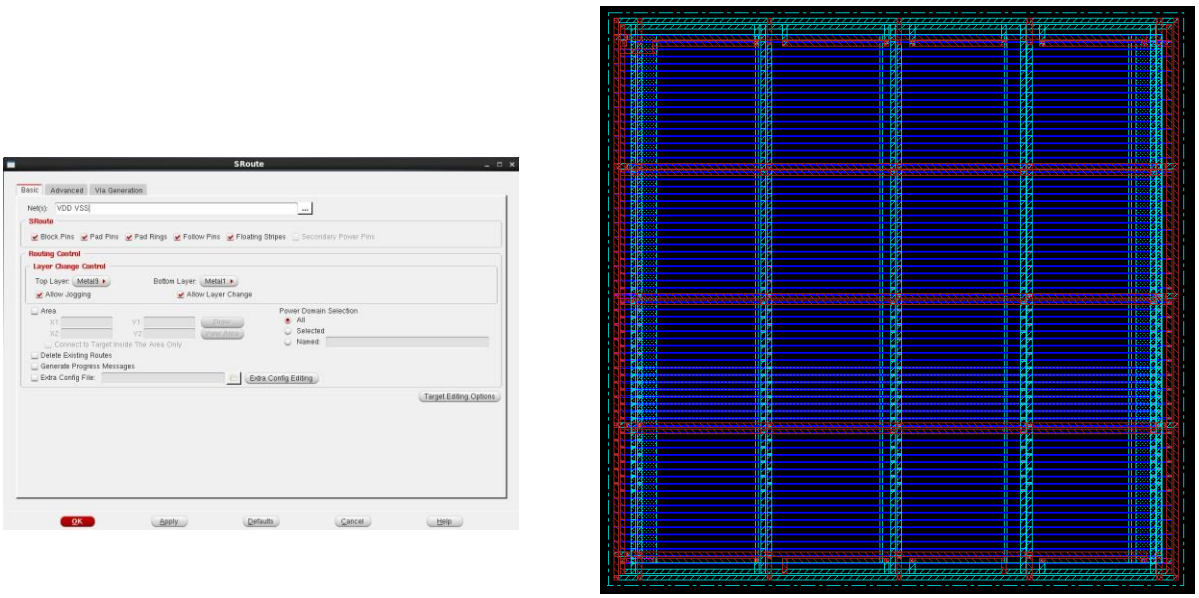


Figure 14: Special Routing the Die

Pre-Placement

-After power planning, a few physical cells are added namely End Caps and Well Taps.

-End Caps: They are physical cells which are added to the left and right core boundaries acting as blockages to avoid standard from moving out of boundary.

-Well Taps: They act like shunt resistance to avoid latch up effects.

-To add End Caps, Select Place -> Physical Cell -> Add End Caps and "Select" the FILL from the available list.

-Higher FILLS have higher Widths

-As shown below, the End Caps are added below your Power mesh.

-To add Well Taps, Select Place -> Physical Cell -> Add Well Tap -> Select -> FILL X [X->strength of fill is equal to 1,2,4 etc] -> Distance Interval [Could be given in range of 30-45u] -> OK.

Add End Caps and Well Taps:

- End Caps: Physical cells at core boundaries prevent cells from shifting.

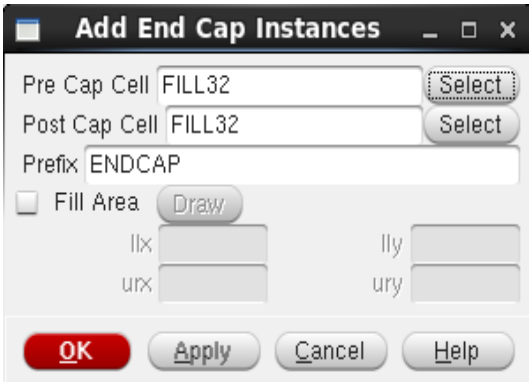


Figure 15: Addition of end cap

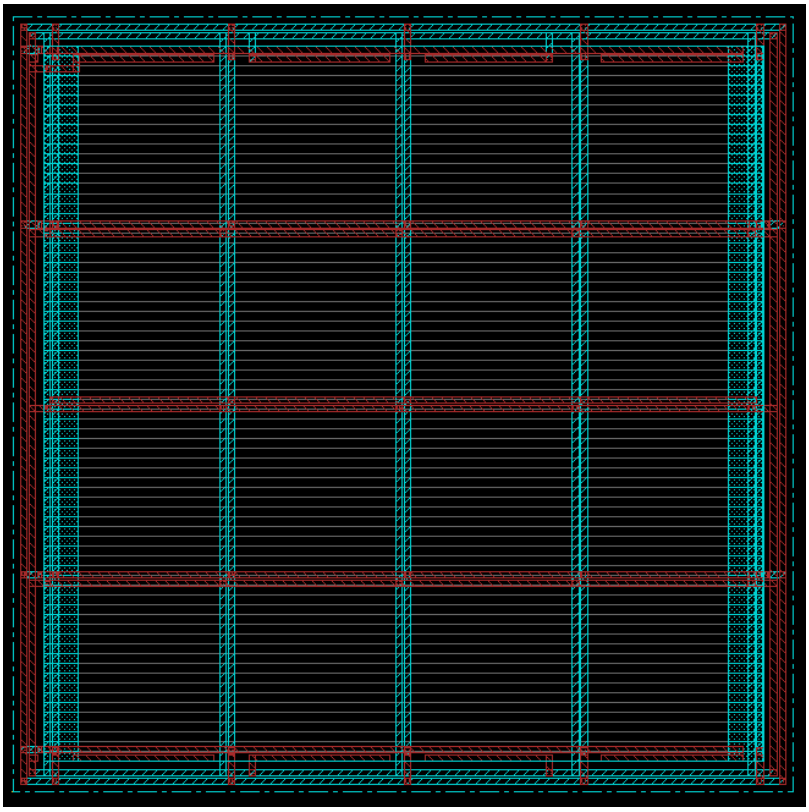


Figure 16: Die with the end caps

- Well Taps: Provide resistance to prevent latch-up effects.

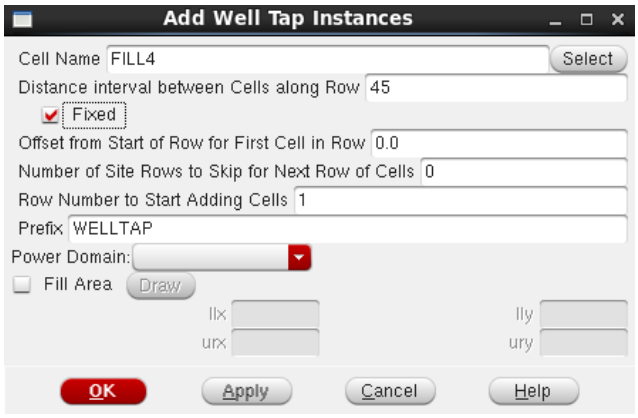


Figure 17: Addition of the Well Taps

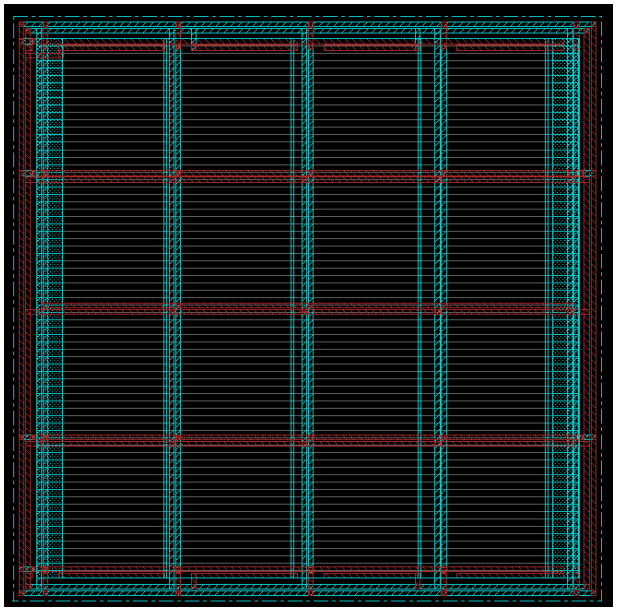


Figure 18: Die with Well Taps

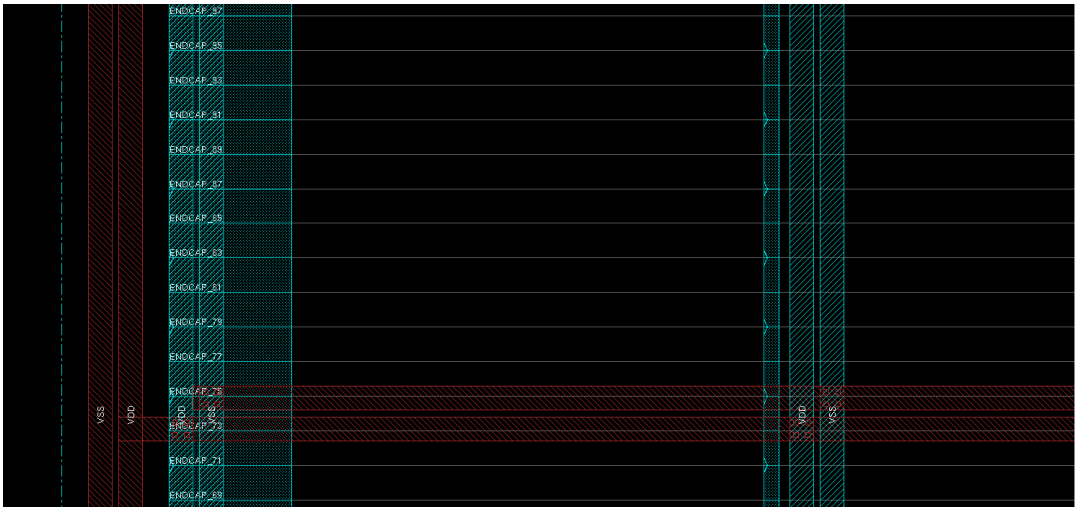


Figure 19: Enlarged view of End cap And Well taps

Placement

Placement is the process of physically arranging standard cells and I/O pins within the designated core area of the chip. The goal is to minimize wirelength between cells and avoid congestion, which in turn improves timing and reduces power consumption. During placement, each standard cell is assigned a specific location, and pins are placed at the boundary or within the core, depending on the design requirements.

This is typically done using algorithms that attempt to place cells based on their connectivity and interaction with other cells, ensuring that communicating cells are placed as close as possible to reduce the length of routing wires. This step is critical for achieving both timing and physical constraints, such as the area usage and the spacing between different components.

Once the cells are placed, the tool verifies that they meet all constraints, such as minimum area, spacing, and alignment with the power grid. The placement phase also takes into account factors like signal integrity and clock tree distribution, with certain cells requiring placement near each other to minimize delay or prevent interference. After placement is complete, the tool typically runs a verification process to check for any violations related to cell overlap, congestion, or other physical design rules. The placement phase is iterative, with optimizations made in each round to improve the overall design, ensuring that power, area, and timing are all balanced appropriately.

In complex designs, the placement tool might run multiple iterations to fine-tune the positioning of cells based on various factors such as congestion maps, timing paths, and power delivery. Once the placement is finalized and verified, it is sent to the routing stage, where actual metal connections are made. Proper placement is crucial as it directly impacts the effectiveness of the routing phase. Poor placement can lead to long wire lengths, increased power consumption, and timing issues, making the placement stage one of the most important steps in the physical design flow.

-The placement stage deals with placing of standard cells as well as Pins.

-Select place -> Place Standard Cell -> Run full placement -> Mode -> Enable 'Place I/O Pins' -> OK -> OK.

-All the standard cells and Pins are placed as placed as per the communication between them i.e, two communicating cells are placed as close as possible so that shorter net lengths can be used for connections as shorter net lengths enable better timing results.

-You can toggle the layer visibility from the list on the right. The list of layers available are shown on the right under "Layer" Tab with color coding.

Place standard cells and IO pins based on connectivity to minimize delays. This is achieved using timing-driven placement techniques.

Figure 20: Placement window

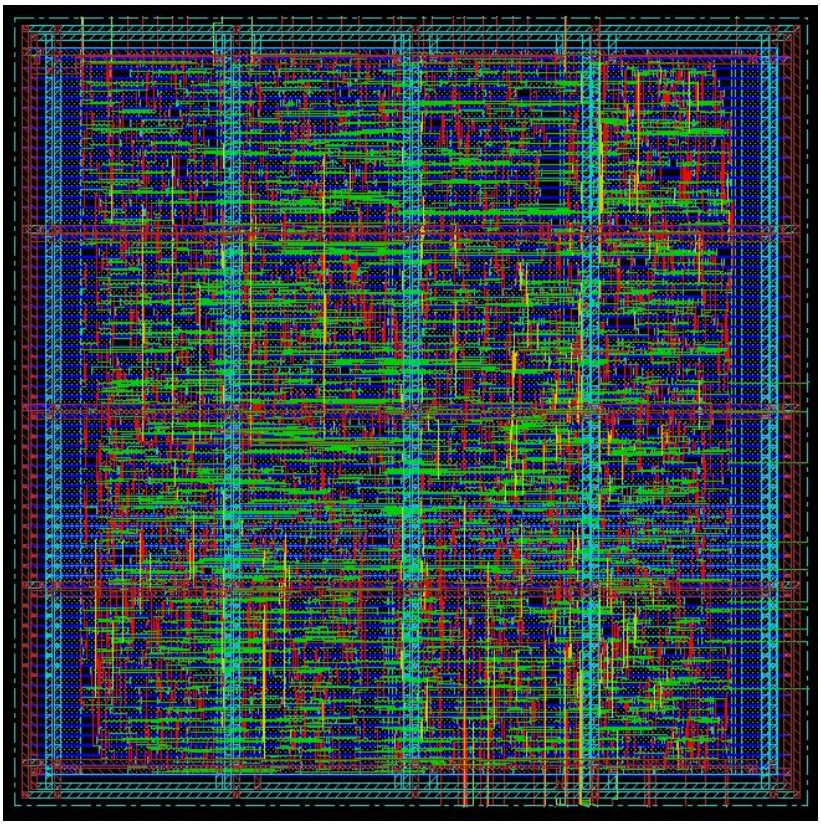


Figure 21: Post placement View of the Die

Clock Tree Synthesis (CTS)

CTS builds a clock distribution network to ensure that clock signals arrive at all flip-flops simultaneously. Clock buffers and inverters are added for signal strength.

-The CTS stage is meant to build a clock distribution network such that every register {Flip Flop} acquires clock at the same time (At least approximately).

to keep them in proper communication.

-A script can be used to build the Clock Tree as follows.

-Source the script as shown through the terminal and then Select Clock -> CCOpt Clock Tree Debugger -> OK. To build and view Clock Tree.

-The Red boxes are the Clock Pins of various Flip Flops in the design while yellow Pentagon the top represents Clock Source.

-The Clock Tree is built with Clock buffers and Clock Invertors added to boost up the clock signal

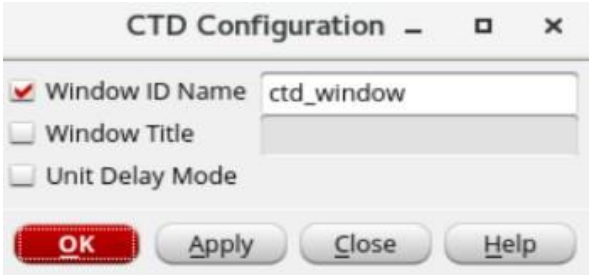


Figure 22: CTD configuration window

1. Timing Report:

-To generate timing report, Timing -> Report Timing -> Design Stage -Pre-CTS

-Analysis Type -Setup -> OK.

-Timing Report Summary can be seen on the terminal.

2. Area Report:

-cmd: report_area

3. Power report:

-cmd: report_power

-In case of any violating paths, the design could be optimised in the following way.

-To optimise the design, Select ECO -> Optimise Design -> Design Stage [Pre CTS] -> Optimisation Type -Setup -> OK

-After you run the Optimisation, The terminal displays the latest timing report and updated area and power reports can be checked.

-This step optimises your design in terms of timing, area and power. You can generate the Timing, Area, Power in similar as above report post-Optimisation to compare the reports.

```
------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):| -0.228  | -0.054  | -0.228  |   N/A   |   N/A   |   N/A   |
|           TNS (ns):| -5.355  | -0.597  | -4.758  |   N/A   |   N/A   |   N/A   |
|    Violating Paths:|   62    |   14    |   48    |   N/A   |   N/A   |   N/A   |
|          All Paths:|  1782   |   999   |   783   |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

Density: 91.090%
Routing Overflow: 0.00% H and 0.35% V
```

Figure 23: Time design report with violations

```
------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+--------
-+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate
 |
+--------------------+---------+---------+---------+---------+---------+--------
-+
|           WNS (ns):| -61.455 |  4.396  | -61.455 |   N/A   |   N/A   |   N/A
 |
|           TNS (ns):|-32831.4 |  0.000  |-32831.4 |   N/A   |   N/A   |   N/A
 |
|    Violating Paths:|   735   |    0    |   735   |   N/A   |   N/A   |   N/A
 |
|          All Paths:|  1782   |   999   |   783   |   N/A   |   N/A   |   N/A
 |
+--------------------+---------+---------+---------+---------+---------+--------
-+

+---------------+-------------------------------+------------------+
|               |              Real             |       Total      |
|    DRVs       +------------------+------------+------------------|
|               |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+---------------+------------------+------------+------------------+
|   max_cap     |      1 (1)       |   -3.527   |      1 (1)       |
|   max_tran    |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout  |      0 (0)       |     0      |      0 (0)       |
|   max_length  |      0 (0)       |     0      |      0 (0)       |
+---------------+------------------+------------+------------------+

Density: 91.090%
Routing Overflow: 0.00% H and 0.35% V
```

Figure 24: Time design report with further violations

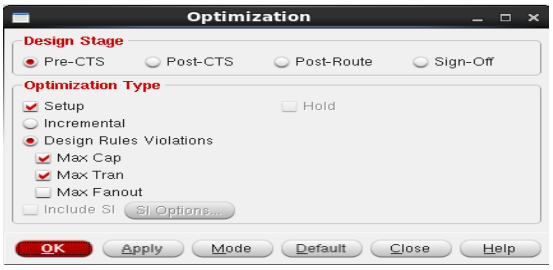


Figure 25: Optimization window

```
------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  4.400  |  4.400  |  7.680  |   N/A   |   N/A   |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |   N/A   |   N/A   |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |   N/A   |   N/A   |   N/A   |
|          All Paths:|  1782   |   999   |   783   |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+

Density: 91.314%
Routing Overflow: 0.00% H and 0.31% V
```

Figure 26: Timing Report with no violations

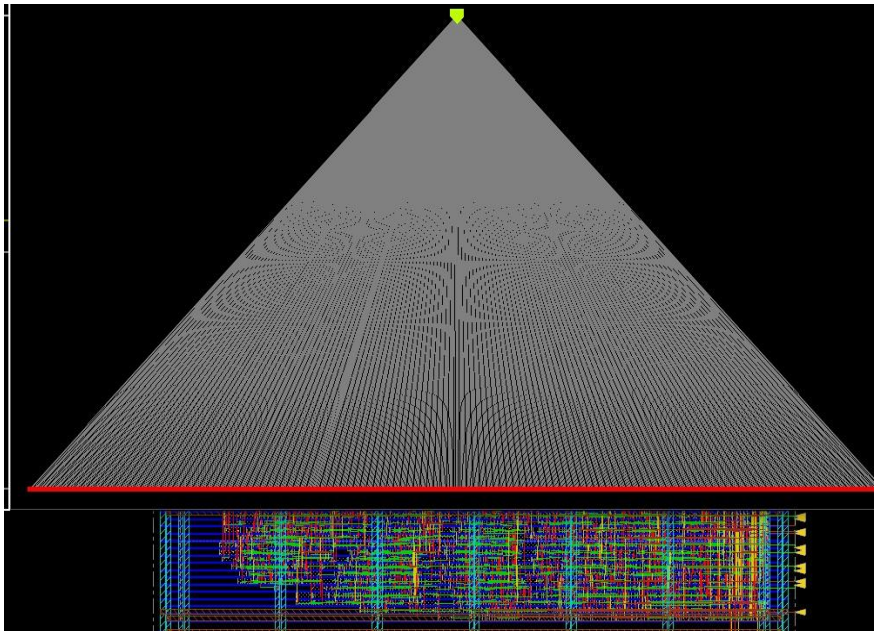


Figure 27: Clock tree Synthesis

Post CTS Analysis

```
# Analysis View: WORSTCASE
********** Clock clk Post-CTS Timing Analysis **********
Nr. of Subtrees                : 1
Nr. of Sinks                   : 735
Nr. of Buffer                  : 17
Nr. of Level (including gates) : 2
Root Rise Input Tran           : 0.1(ps)
Root Fall Input Tran           : 0.1(ps)
No Driving Cell Specified!
Max trig. edge delay at sink(R): x_reg[15][10]/CK 353(ps)
Min trig. edge delay at sink(R): y_reg[13][7]/CK 325.8(ps)


                                 (Actual)              (Required)
Rise Phase Delay               : 325.8~353(ps)         0~10(ps)
Fall Phase Delay               : 327.5~354.7(ps)       0~10(ps)
Trig. Edge Skew                : 27.2(ps)              400(ps)
Rise Skew                      : 27.2(ps)
Fall Skew                      : 27.2(ps)
Max. Rise Buffer Tran.         : 120.5(ps)             200(ps)
Max. Fall Buffer Tran.         : 109.8(ps)             200(ps)
Max. Rise Sink Tran.           : 101.2(ps)             200(ps)
Max. Fall Sink Tran.           : 93(ps)                200(ps)
Min. Rise Buffer Tran.         : 119.4(ps)             0(ps)
Min. Fall Buffer Tran.         : 108.5(ps)             0(ps)
Min. Rise Sink Tran.           : 73.7(ps)              0(ps)
Min. Fall Sink Tran.           : 70.1(ps)              0(ps)

view WORSTCASE : skew = 27.2ps (required = 400ps)
view BESTCASE : skew = 24.1ps (required = 400ps)
```

Figure 28: Post CTS Report

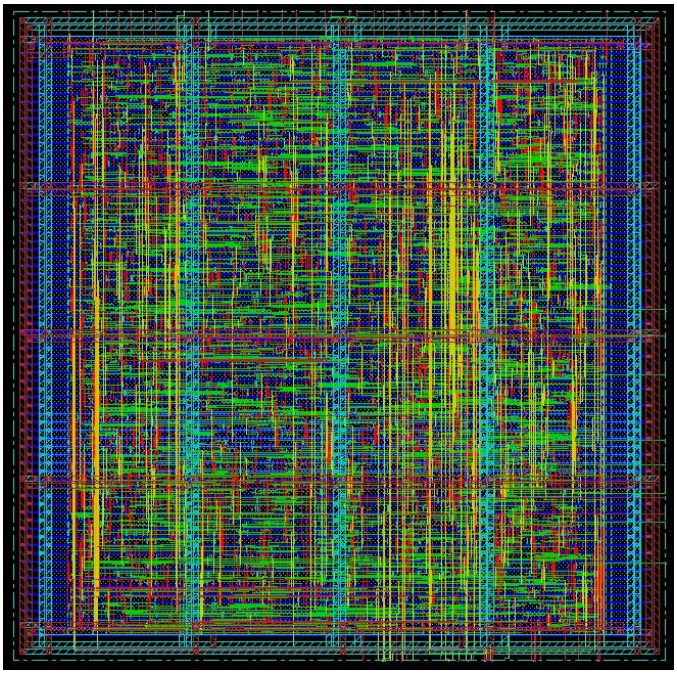


Figure 29: Post CTC view of die

Routing

Routing is the process of physically connecting the placed cells and pins using metal layers to create the required electrical paths for the design. During this phase, the tool generates the physical interconnections between standard cells, ensuring that all the logical nets from the netlist are correctly implemented in the layout. The routing stage involves replacing logical connectivity with real metal layers, and it must adhere to design rules to avoid issues like signal integrity violations, crosstalk, and manufacturing defects. Routing algorithms use the available metal layers in the design to determine the most efficient paths for each connection, taking into account factors such as congestion, signal delay, and power dissipation.

There are various types of routing performed during this phase, including global routing and detailed routing. Global routing is the initial step where the tool determines the general path of the routing wires across the chip, considering factors like congestion and available routing channels. Detailed routing follows, where the exact placement of metal wires, vias, and other components is determined.

Detailed routing ensures that all the wires fit within the available metal layers without violating design rules such as spacing and width constraints. The routing tool also checks for potential issues like signal integrity problems, which could cause delays or noise in the circuit, and it applies optimizations to improve the quality of the final design.

Once the routing is completed, the design is checked for violations such as antenna effects, shorts, or opens, which could affect the reliability of the circuit. The tool also verifies that the routing meets the timing requirements set in earlier stages of the design flow. In many cases, the routing phase is followed by post-routing optimization, where the routing is adjusted to further reduce wirelength, power consumption, or delay. The final routed design is then prepared for physical verification, which includes design rule checking (DRC) and layout versus schematic (LVS) checks to ensure that the physical design matches the intended functionality and adheres to the manufacturing constraints.

Routing replaces logical connections with physical metal wires:

- Ensure no opens, shorts, or signal integrity violations.

```
# Analysis View: WORSTCASE
********** Clock clk Post-CTS Timing Analysis **********
Nr. of Subtrees                : 1
Nr. of Sinks                   : 735
Nr. of Buffer                  : 17
Nr. of Level (including gates) : 2
Root Rise Input Tran           : 0.1(ps)
Root Fall Input Tran           : 0.1(ps)
No Driving Cell Specified!
Max trig. edge delay at sink(R): x_reg[15][10]/CK 353(ps)
Min trig. edge delay at sink(R): y_reg[13][7]/CK 325.8(ps)


                                 (Actual)              (Required)
Rise Phase Delay               : 325.8~353(ps)         0~10(ps)
Fall Phase Delay               : 327.5~354.7(ps)       0~10(ps)
Trig. Edge Skew                : 27.2(ps)              400(ps)
Rise Skew                      : 27.2(ps)
Fall Skew                      : 27.2(ps)
Max. Rise Buffer Tran.         : 120.5(ps)             200(ps)
Max. Fall Buffer Tran.         : 109.8(ps)             200(ps)
Max. Rise Sink Tran.           : 101.2(ps)             200(ps)
Max. Fall Sink Tran.           : 93(ps)                200(ps)
Min. Rise Buffer Tran.         : 119.4(ps)             0(ps)
Min. Fall Buffer Tran.         : 108.5(ps)             0(ps)
Min. Rise Sink Tran.           : 73.7(ps)              0(ps)
Min. Fall Sink Tran.           : 70.1(ps)              0(ps)

view WORSTCASE : skew = 27.2ps (required = 400ps)
view BESTCASE : skew = 24.1ps (required = 400ps)
```

Figure 30: Post Routing report

- Perform Timing-Driven and Signal Integrity (SI)-Driven Routing for better design reliability.

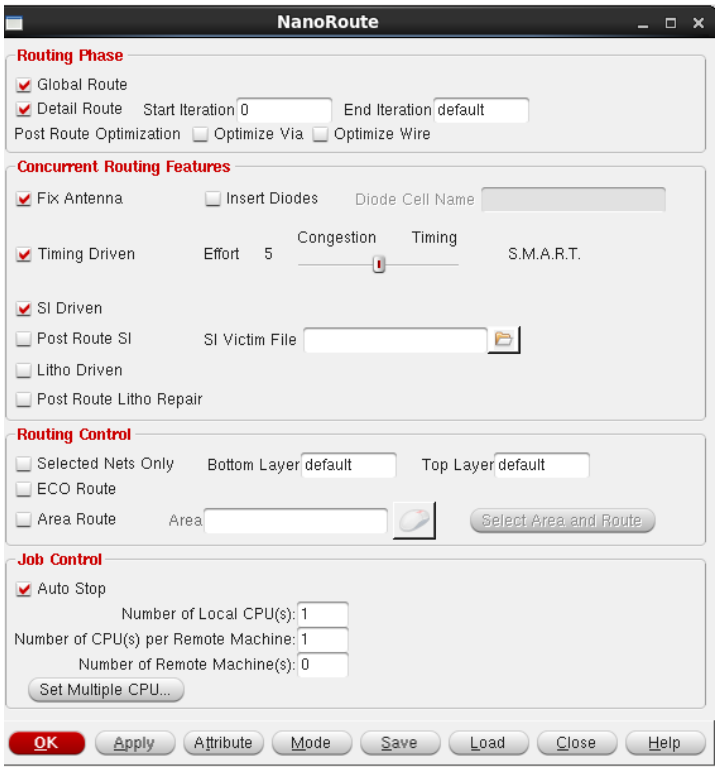


Figure 31: Post Nano Routing Report

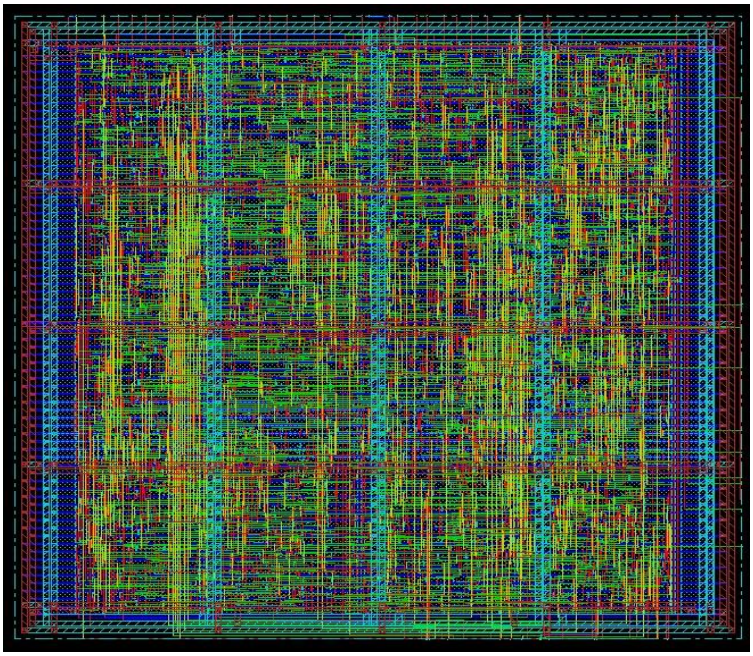


Figure 32: Post Routing View of the Die

Saving the Database

1. Save the design database: File -> Save Design -> Innovus -> <DesignName>.enc.

2. Save the netlist: File -> Save -> Netlist -> <NetlistName>.v.

3. Save the GDS II file: File -> Save -> GDS/OASIS -> <FileName>.gds.

Physical Verification

Verify the physical design using DRC and LVS:

1. **Design Rule Check (DRC)**: Ensure the design meets fabrication rules.

- Inputs: Tech library, rule sets, and GDS files.

- Output: DRC violation report.

2. **Layout vs. Schematic (LVS):** Check for consistency between layout and netlist.

- Inputs: Tech library, GDS files, and spice netlist.

- Output: LVS match/mismatch report.

-From the Innovus GUI, Select PVS -> Run DRC to open the "DRC Submission Form".

-The DRC Run Submission Form begins with mentioning the Run Directory. The Run Directory is the location where all the logs, reports and other files concerned with PVS are saved.

Procedure:

The technology Library is to be loaded under "Rules tab"

-The Technology Library is specific for PVS Tools and technology mode on which the design is created.

-On reading the tech lin, the rule set is loaded and the corresponding fabrication rules are read in to be against the design.

-The GDS format files of all standard cells available with the corresponding technology node are also provided by the vendor. Select all of them to add.

-The output report can be named and saved as shown.

-Hit "Submit" to run the DRC and the following windows appear.

-All the list of DRC Errors can be seen in the above window of the which the location of the DRC and Violation occurring can be highlighted dealing one to one.

-For example, in the above snapshot, the errors associated with N-implant can be seen (Select a error occurrence and click on the right arrow below to highlight/zoom in the location).

-You can save the DRC RUn as a "Preset" file to rerun the DRC if required as a later point of time.

5. RESULTS AND DISCUSSION

Simulation Output from Xilinix Vivado

Figure 33: Simulation Output of Xilinx Vivado

Simulation Output from Cadence Incisive

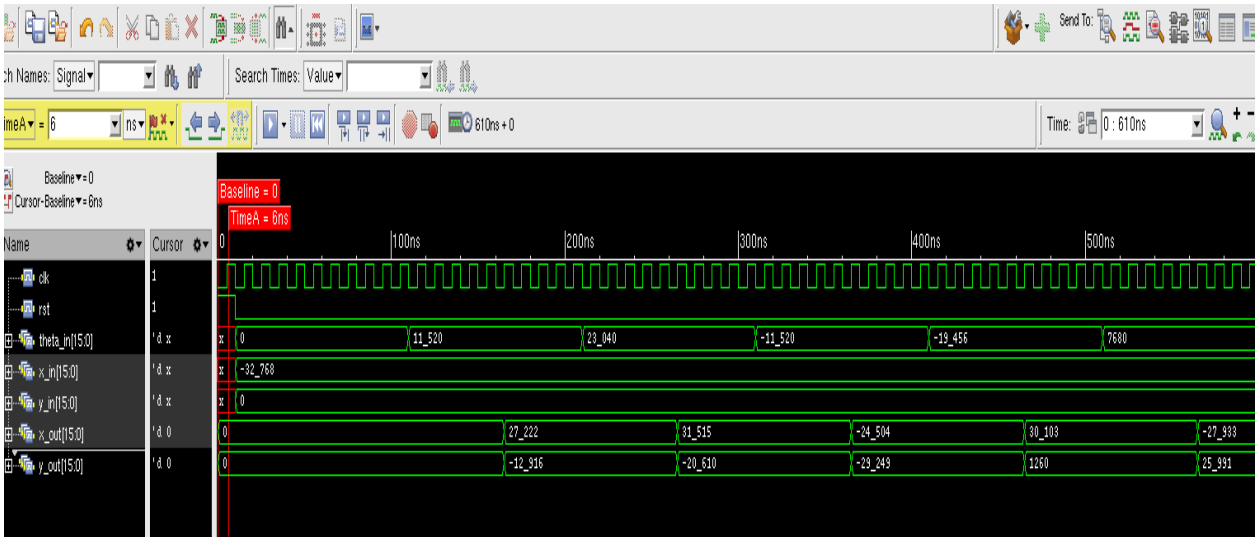


Figure 34: Simulation from Cadence Incisive

The waveforms generated in Vivado and Incisive are essential for verifying the behavior of the design. In Vivado, after running a simulation, the waveform output provides a detailed graphical representation of signal transitions, allowing the designer to observe the timing and interaction of signals within the design. This is crucial for ensuring the design's functionality and checking for any discrepancies.

Netlist generated from Cadence Genus

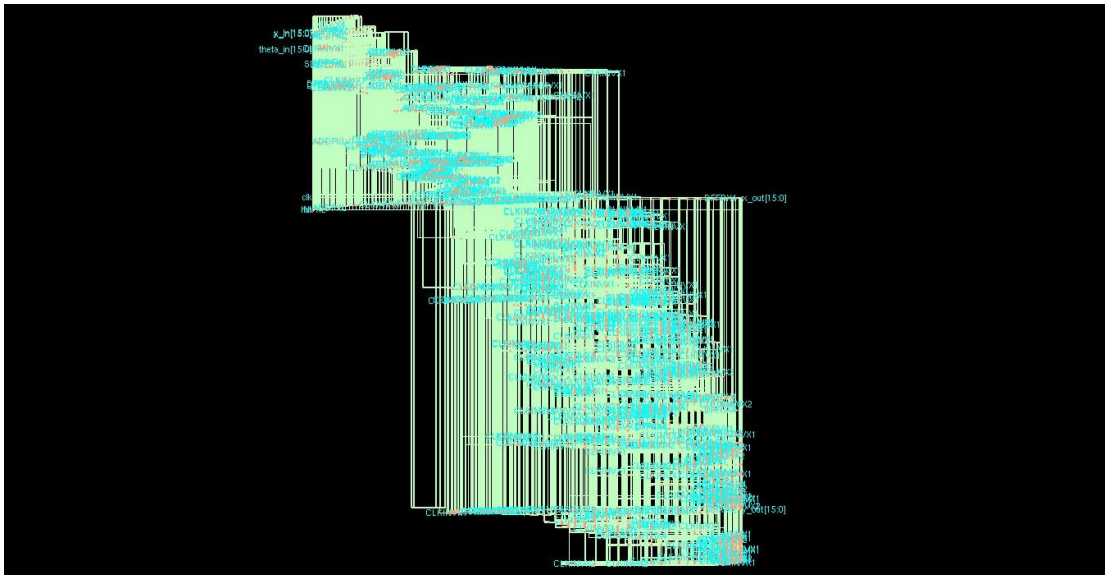


Figure 35: Netlist generated from Genus

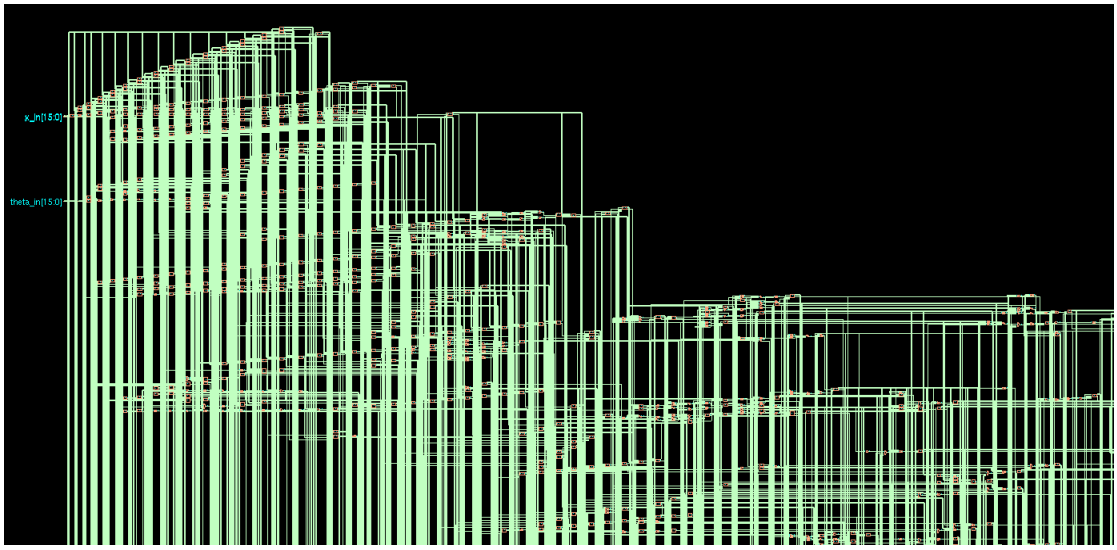


Figure 36: Enlarged view of Netlist

The netlist generated from Cadence Genus represents the synthesized design at the gate level, which includes the logical connections between standard cells, pins, and interconnections. After synthesis, the netlist contains all the components required for physical design, including the gates, flip-flops, and their interconnections, along with timing and power constraints. This netlist is crucial as it serves as the input for further stages like physical design and verification.

Pre placement reports – Genus

C: > Users > User > Downloads > cordic_area.rep

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 21.14-s082_1
  Generated on:           Oct 09 2024  11:45:43 am
  Module:                 cordic_pipeline
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

    Instance    Module  Cell Count  Cell Area  Net Area   Total Area   Wireload
-----------------------------------------------------------------------------------
cordic_pipeline              3880      0.000     0.000        0.000 <none> (D)
  (D) = wireload is default in technology library
```

Figure 37: Pre-placement Area report from Genus

Jsers > User > Downloads > cordic_power.rep

```
Instance: /cordic_pipeline
Power Unit: W
PDB Frames: /stim#0/frame#0
-------------------------------------------------------------------------
  Category         Leakage     Internal    Switching        Total    Row%
-------------------------------------------------------------------------
    memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  register     4.27861e-07  1.32398e-03  2.03773e-04  1.52818e-03  50.80%
     latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
     logic     5.24745e-07  1.23349e-03  1.99395e-04  1.43341e-03  47.65%
      bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
     clock     0.00000e+00  0.00000e+00  4.67610e-05  4.67610e-05   1.55%
       pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
-------------------------------------------------------------------------
  Subtotal     9.52607e-07  2.55747e-03  4.49929e-04  3.00835e-03 100.00%
Percentage           0.03%       85.01%       14.96%      100.00% 100.00%
-------------------------------------------------------------------------
```

Figure 38: Pre-placement power report from Genus

dit Selection View Go Run Terminal Help cordic_qor.rep - Visual Studio Code [Administrator]

ordic.sdc | cordic_area.rep | cordic_power.rep | cordic_qor.rep

Users > User > Downloads > cordic_qor.rep

```
  Cost      Critical          Violating
 Group    Path Slack  TNS      Paths
-------------------------------------
clk           6230.4   0.0          0
default    No paths    0.0
-------------------------------------
Total                  0.0          0

Instance Count
--------------
Leaf Instance Count              3880
Physical Instance count             0
Sequential Instance Count         735
Combinational Instance Count     3145
Hierarchical Instance Count         0

Area
----
Cell Area                           0.000
Physical Cell Area                  0.000
Total Cell Area (Cell+Physical)     0.000
Net Area                            0.000
Total Area (Cell+Physical+Net)      0.000

Max Fanout                          735 (clk)
Min Fanout                          0 (n_172)
Average Fanout                      2.1
Terms to net ratio                  3.2661
Terms to instance ratio             4.0843
Runtime                             148.57596 seconds
Elapsed Runtime                     166 seconds
Genus peak memory usage             1840.25
```

Figure 39: Pre-placement QOR report from Genus

Floorplaning

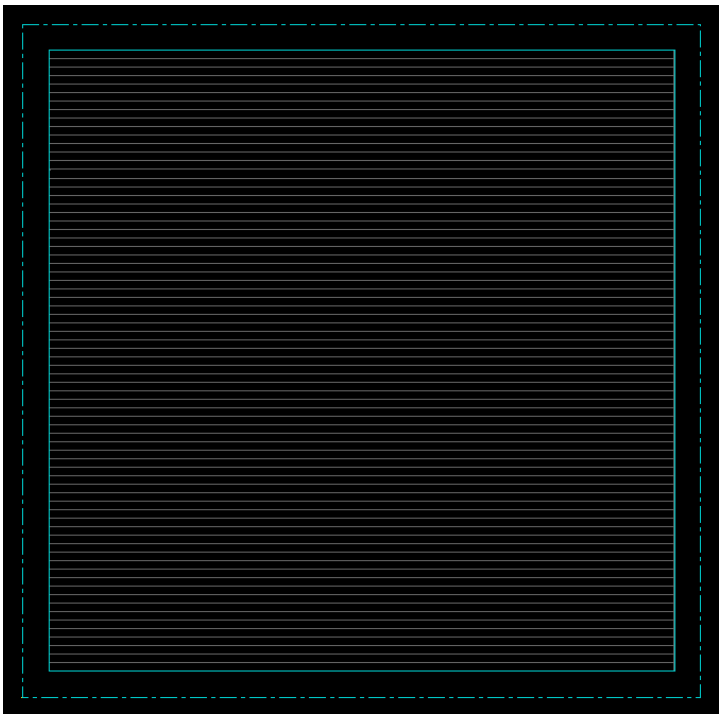


Figure 40: Die after Floor Planning

Placement

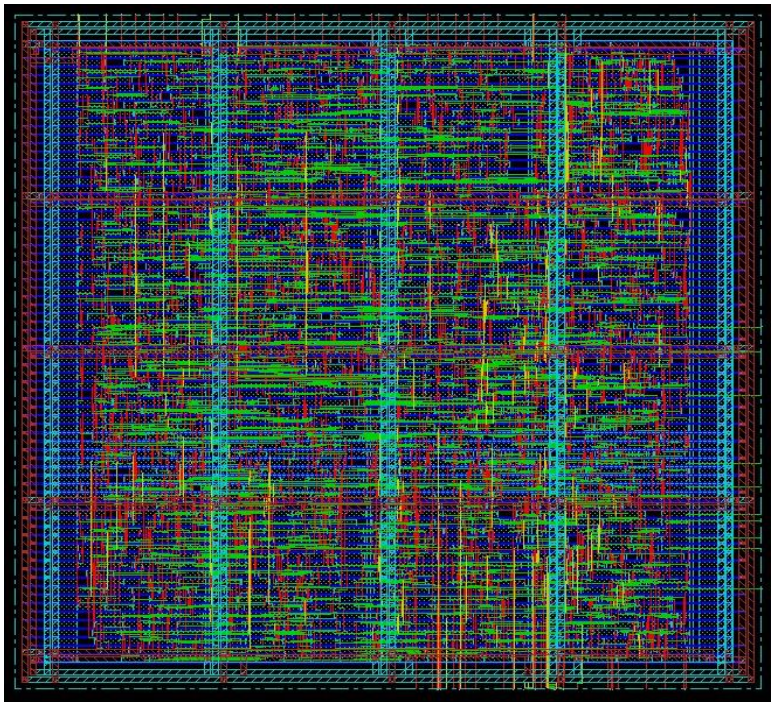


Figure 41: Die after Placement

Clock tree synthesis

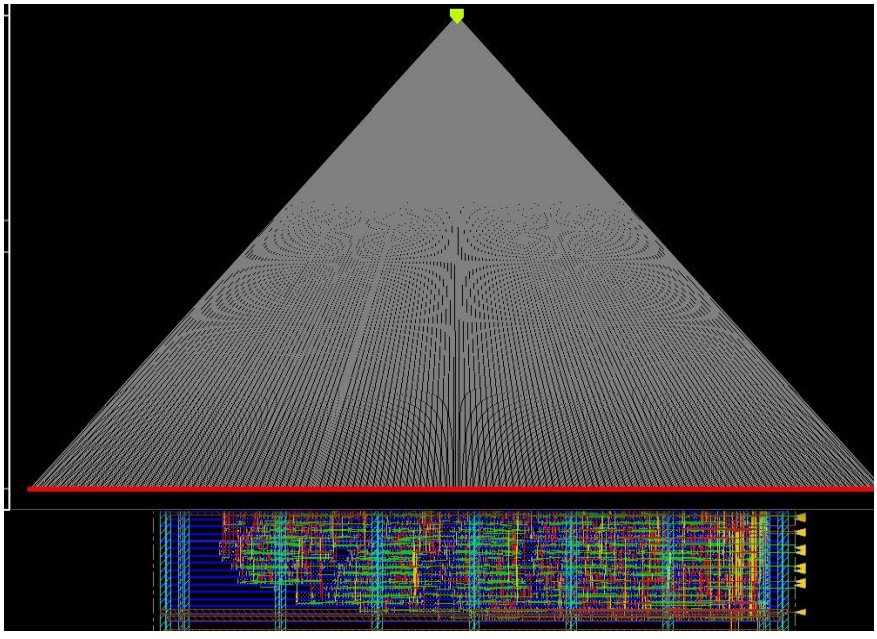


Figure 42: Clock tree Synthesis view

Routing

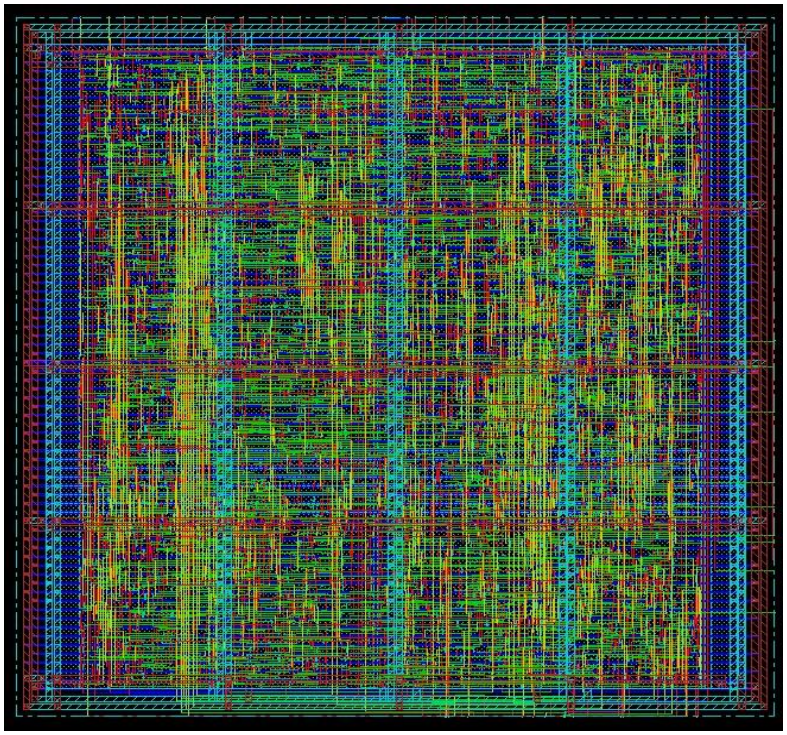


Figure 43: Final Routing image

Final Analysis reports

Power report:

```
Group                          Internal   Switching     Leakage       Total  Percentage
                               Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                        2.757      0.3065     0.08288       3.147        57.7
Macro                                 0           0           0           0           0
IO                                    0           0           0           0           0
Combinational                     1.207      0.6209     0.06764       1.896       34.76
Clock (Combinational)            0.2476      0.1599    0.003529       0.411       7.537
Clock (Sequential)                    0           0           0           0           0
-----------------------------------------------------------------------------------------
Total                             4.212       1.087       0.154       5.453         100
-----------------------------------------------------------------------------------------


Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
VDD                       0.9      4.212       1.087       0.154       5.453         100


Clock                          Internal   Switching     Leakage       Total  Percentage
                               Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
clk                              0.2476      0.1599    0.003529       0.411       7.537
-----------------------------------------------------------------------------------------
Total                            0.2476      0.1599    0.003529       0.411       7.537
-----------------------------------------------------------------------------------------


-----------------------------------------------------------------------------------------
*       Power Distribution Summary:
*              Highest Average Power:               clk__L1_I0 (CLKBUFX20):      0.03199
*              Highest Leakage Power:               clk__L1_I0 (CLKBUFX20):     0.000367
*              Total Cap:      2.73638e-11 F
*              Total instances in design:  2294
*              Total instances in design with no power:     0
*               Total instances in design with no activty:     0

*              Total Fillers and Decap:     0
-----------------------------------------------------------------------------------------
```

Figure 44: Final Power Analysis Report

Area port:

```
Gate area 2.2707 um^2
Level 0 Module cordic_pipeline  Gates= 22719 Cells= 3942 Area= 51588.8 um^2
```

Figure 45: Final Area report

DRC / LVS Checks:

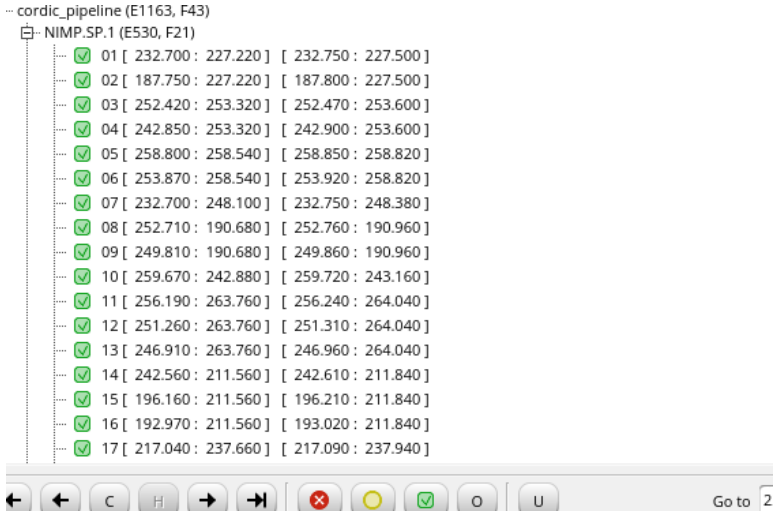


Figure 46: DRC Report

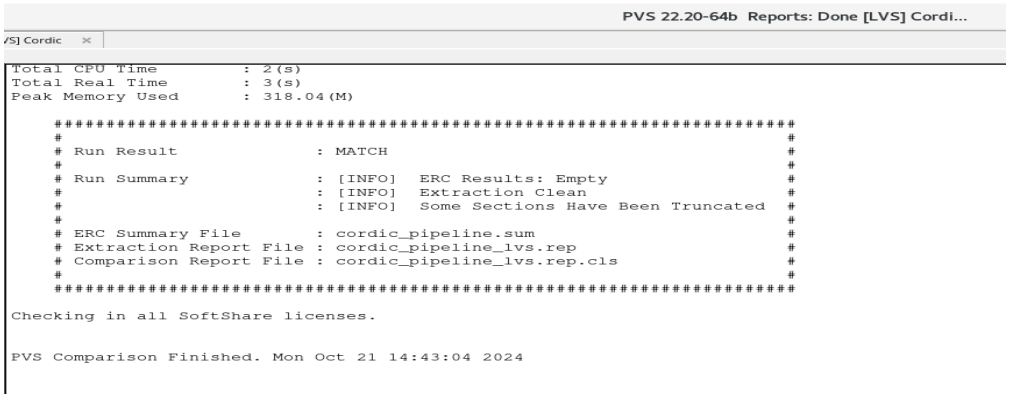


Figure 47: PVS Report

Figure 48: LVS Report

Final GDSII layout

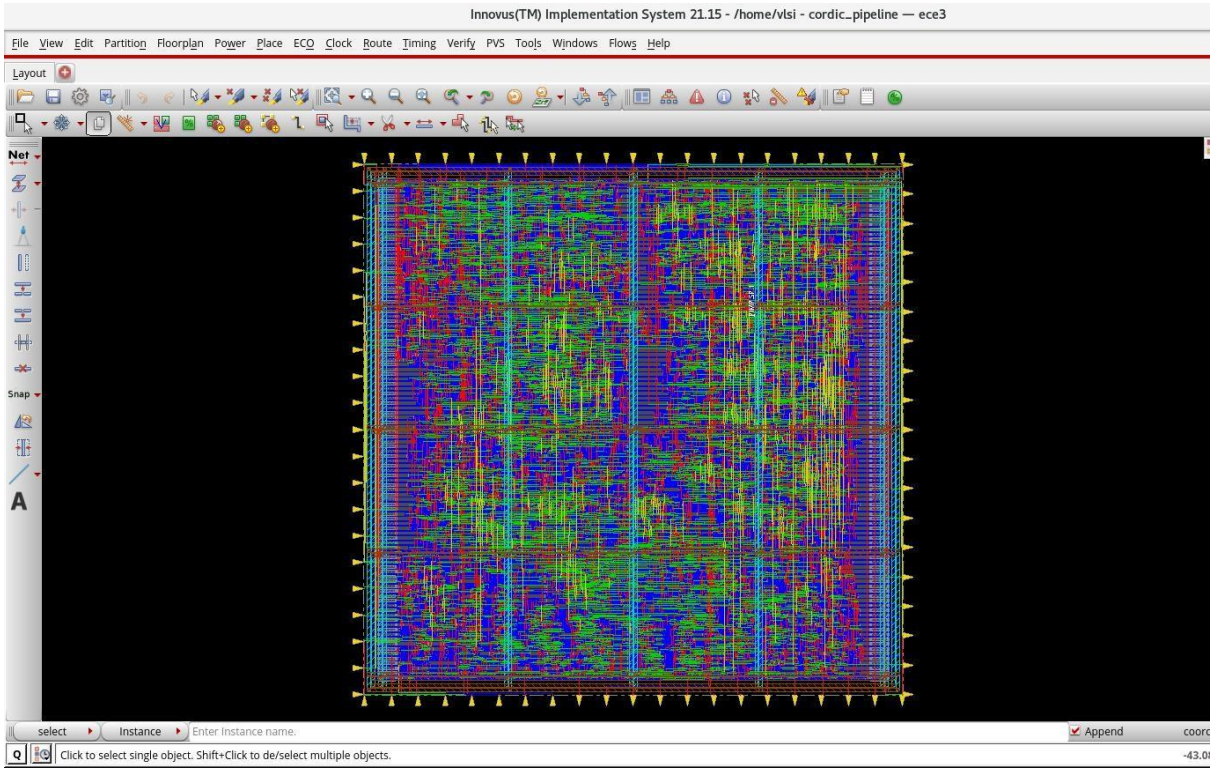


Figure 49: Final GDS image

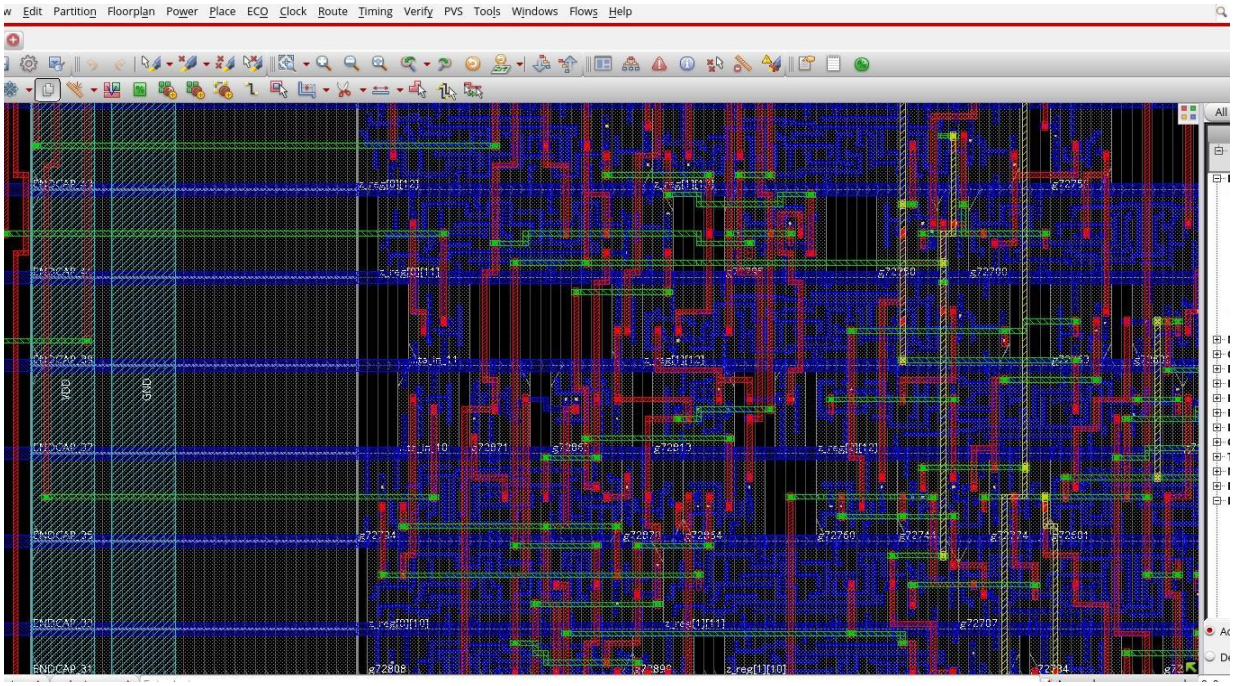


Figure 50: Enlarged view of GDS image

3D view of GDSII

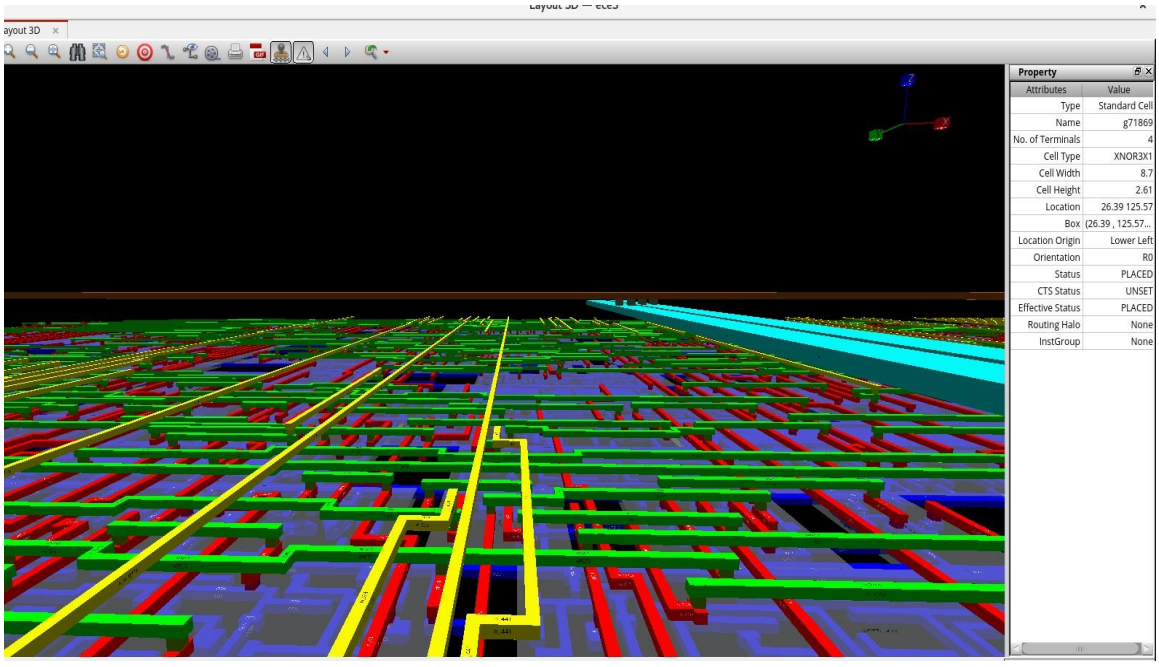


Figure 51: 3D View of GDS image

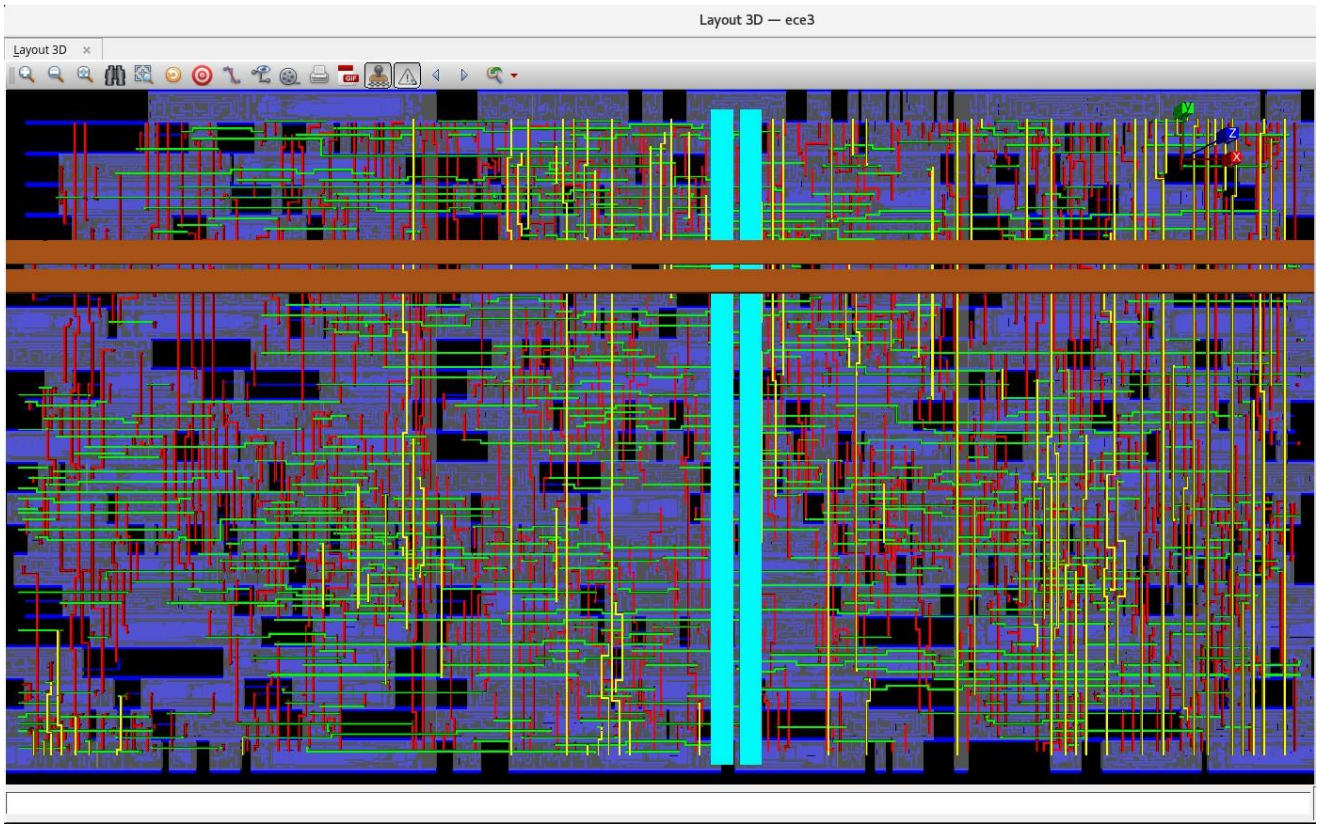


Figure 52: Final GDS File

6. APPLICATION, ADVANTAGES& LIMITATION

6.1 Application

The CORDIC algorithm has widespread use across various technical domains, leveraging its iterative approach to perform computationally intensive tasks efficiently. Key applications include:

1. **Digital Signal Processing (DSP):**

- Enables efficient computation of FFT for spectral analysis and filter design.

- Used in adaptive filtering and signal modulation processes.

2. **Graphics and Image Processing:**

- Facilitates 2D/3D geometric transformations such as rotation, scaling, and translation.

- Integral in rendering pipelines for real-time graphics and animation.

3. **Communication Systems:**

- Supports modulation/demodulation schemes like QAM and PSK by calculating phase shifts.

- Crucial in software-defined radios and baseband signal processing.

4. **Navigation and Control Systems:**

- Implements angle calculations in GPS and inertial navigation systems.

- Used for real-time trajectory computation and motion control in robotics.

5. **Embedded Systems:**

- Optimized for microcontrollers and DSP processors to execute trigonometric, logarithmic, and exponential functions.

- Widely used in resource-constrained systems lacking hardware multipliers.

6. **Medical Imaging:**

- Applied in iterative reconstruction algorithms for CT and MRI image processing.

6.2 Advantages

The CORDIC algorithm offers several technical benefits:

1. **Hardware Efficiency:**

- Eliminates the need for multipliers, relying only on shift-and-add operations.

- Ideal for FPGA and ASIC implementations in low-resource environments.

2. **Computational Versatility:**

- Capable of evaluating trigonometric, hyperbolic, and exponential functions.

- Supports various fixed-point and real-time applications.

3. **Scalability and Adaptability:**

- Easily configured for varying word lengths and precision requirements.

- Performs robustly across low and high-performance computational environments.

4. **Low Power and Cost Efficiency:**

- Reduces power consumption by minimizing hardware complexity.

- Ideal for embedded systems and consumer electronics with constrained budgets.

5. **Real-Time Performance:**

- High-speed convergence supports latency-sensitive applications such as DSP and communication systems.

6.3 Limitation

Despite its strengths, the CORDIC algorithm exhibits certain limitations:

1. **Precision Constraints:**

- Limited accuracy due to finite iterations, with truncation and rounding errors affecting fixed-point implementations.

2. **Convergence for Small Angles:**

- Slower convergence for small-angle approximations compared to direct computation methods.

3. **Scaling Factors:**

- Requires additional pre-scaling and post-scaling operations, which increase computation overhead.

4. **Resource Utilization on Modern Processors:**

- May not outperform multiplier-based algorithms in systems equipped with advanced floating-point units.

5. **Algorithmic Range Restrictions:**

- Input values must be confined to specific ranges without additional modifications.

- Limited support for non-standard functions like square roots or division without auxiliary steps.

These technical applications, advantages, and limitations underscore the algorithm's utility in high-performance systems while addressing trade-offs in modern computational environments.

7. CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

In this project, the CORDIC algorithm was successfully implemented and synthesized, demonstrating its effectiveness in performing trigonometric calculations with minimal hardware requirements. Through the use of industry-standard tools such as Cadence Genus for synthesis and Innovus for physical design, we were able to complete the flow from RTL design to physical design. The GDS file, which is crucial for fabrication, was successfully generated, marking a significant milestone in the chip design process. This achievement proves that the CORDIC algorithm can be effectively utilized in real-world hardware design, especially in applications requiring efficient arithmetic operations.

The project also highlighted the importance of leveraging professional EDA tools for effective design and verification, which ensured the accuracy and reliability of the implementation. The successful generation of the GDS file signifies that the design is ready for further stages, such as fabrication and testing, making it a viable candidate for integration into various hardware systems. This accomplishment reinforces the relevance of the CORDIC algorithm in modern semiconductor design, with its broad applications and potential for future enhancements.

The implementation of the CORDIC algorithm in this project demonstrates its efficiency in performing complex mathematical computations without using multipliers. The algorithm's simplicity and iterative nature make it a reliable choice for hardware designs, especially in low-power and resource-constrained environments. This work successfully synthesized and optimized the design using Cadence tools, ensuring compliance with industry standards. The CORDIC-based design offers flexibility for a wide range of applications, including signal processing, robotics, and telecommunications. With optimized performance in timing, area, and power, this project establishes a robust foundation for further exploration and innovation in computational hardware systems.

7.2 Future Scope

The CORDIC algorithm, due to its multiplier-less architecture, has immense potential for scaling into advanced computing systems. Future research can focus on integrating the algorithm with modern hardware architectures such as FPGAs and ASICs optimized for specific applications like 5G communication, advanced robotics, and autonomous vehicles. Enhancements in the algorithm to support higher precision while maintaining low latency will further expand its usability in critical applications like medical imaging and aerospace navigation systems.

As technology nodes continue to shrink, the CORDIC algorithm can benefit from reduced power consumption and area optimization. Research can also aim at adapting the design for parallel processing systems to improve computational speed significantly. Additionally, incorporating AI and machine learning techniques for dynamic optimization of iteration control can make the algorithm even more efficient and adaptive to varying workloads.

The future scope of the CORDIC algorithm lies in its potential integration with emerging technologies. Enhancements can focus on:

1. **Algorithm Optimization:**
   Improving the convergence rate and precision for small angles and complex functions by exploring hybrid algorithms or pre-computation methods.
2. **Hardware Acceleration:**
   Implementing the CORDIC algorithm in novel hardware architectures like RISC-V processors, GPU accelerators, and neuromorphic computing units for specialized tasks.
3. **Integration with AI/ML:**
   Leveraging the algorithm for efficient matrix operations, feature extraction, and embedded AI solutions, particularly for IoT and edge computing devices.
4. **FPGA and ASIC Applications:**
   Broadening its application in dynamic reconfigurable systems and industry-grade ASIC designs for high-throughput environments like automotive and aerospace.

REFERENCES

[1] Volder, J. E., "The CORDIC Trigonometric Computing Technique," IRE Transactions on Electronic Computers, vol. EC-8, no. 3, pp. 330–334, 1959.

[2] Andraka, R., "A Survey of CORDIC Algorithms for FPGA-Based Computers," in Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, pp. 191–200, 1998.

[3] Parhami, B., Computer Arithmetic: Algorithms and Hardware Designs, 2nd ed., Oxford University Press, 2010.

[4] NPTEL, "Digital Signal Processing," [Online].

Available: https://nptel.ac.in/courses/108/105/108105113/

[5] Smith, S. W., The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing, 1997.

[6] Baker, R. J., CMOS: Circuit Design, Layout, and Simulation, 3rd ed., Wiley-IEEE Press, 2010.

[7] Cadence Design Systems, "Innovus User Guide," [Online]. Available: https://www.cadence.com.

[8] IEEE Standard 1076-1993, IEEE Standard VHDL Language Reference Manual, IEEE, 1993.

ACHIEVEMENTS

- **Best project award 2024 at OPEN DAY 2024 on 29th November 2024 at Dayananda Sagar College of Engineering Bangalore.**

Figure 53 – RECEIVED PRIZE FOR BEST PROJECT

Figure 54 – A GROUP PHOTO WITH THE WINNERS

DrillBit

The Report is Generated by DrillBit Plagiarism Detection Software

Submission Information

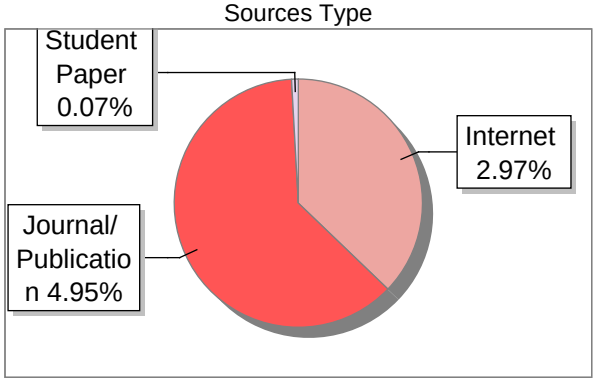
Author Name	RSK
Title	V13
Paper/Submission ID	2704137
Submitted by	hod-ece@dayanandasagar.edu
Submission Date	2024-12-05 10:34:49
Total Pages, Total Words	102, 17481
Document type	Project Work

Result Information

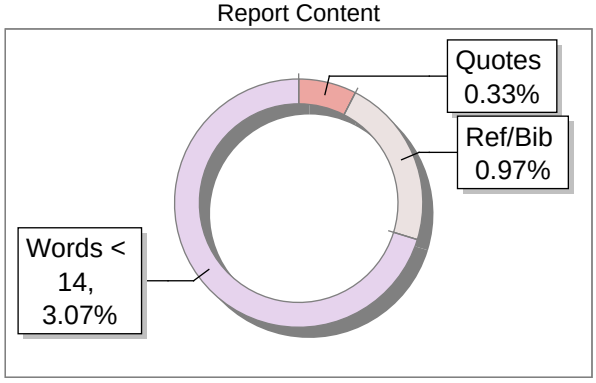
Similarity **8 %**

1 10 20 30 40 50 60 70 80 90

Sources Type



Report Content



Exclude Information

Quotes	Not Excluded
References/Bibliography	Not Excluded
Source: Excluded < 14 Words	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

8	72	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	www.slideshare.net	1	Internet Data
2	drttit.gvet.edu.in	<1	Publication
3	jamiahamdard.edu	<1	Publication
4	assets.sjbit.edu.in	<1	Publication
5	theses.hal.science	<1	Publication
6	Thesis Submitted to Shodhganga Repository	<1	Publication
7	Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedde by Cherupalli-2016	<1	Publication
8	alltimedesign.com	<1	Internet Data
9	www2.eecs.berkeley.edu	<1	Internet Data
10	information-science-engineering.newhorizoncollegeofengineering.in	<1	Publication
11	A Hardware-Efficient Algorithm for Real-Time Computation of ZadoffChu Sequences by Mohamma-2012	<1	Publication
12	www.presiuniv.ac.in	<1	Publication
13	Architecture level optimization of 3-dimensional tree-based FPGA by Pangracious-2014	<1	Publication

14	ejurnal.seminar-id.com	<1	Publication
15	technodocbox.com	<1	Internet Data
16	digitalya.co	<1	Internet Data
17	dsbs.edu.in	<1	Publication
18	www.isteonline.in	<1	Publication
19	cmjournal.net	<1	Internet Data
20	docview.dlib.vn	<1	Publication
21	eprints.hud.ac.uk	<1	Publication
22	Towards a Comprehensive City Emission Function (CCEF), by KOCIFAJ, Miroslav- 2017	<1	Publication
23	www.ncbi.nlm.nih.gov	<1	Internet Data
24	Study on robust aerial docking mechanism with deep learning based drogue detecti by Choi-2021	<1	Publication
25	www.managedserver.eu	<1	Internet Data
26	www.ncbi.nlm.nih.gov	<1	Internet Data
27	journal.unair.ac.id	<1	Internet Data
28	journaljesbs.com	<1	Publication
29	moam.info	<1	Internet Data
30	moam.info	<1	Internet Data
31	moam.info	<1	Internet Data
32	Submitted to Visvesvaraya Technological University, Belagavi	<1	Student Paper

33	www.arxiv.org	<1	Publication
34	Lecture Notes in Computer Science Advances in Cryptology EUROCRYP, by Johansson, Thomas - 2013	<1	Publication
35	moam.info	<1	Internet Data
36	moam.info	<1	Internet Data
37	Stacked modulation formats enabling highest-sensitivity optical free-space links by Ludwig-2015	<1	Publication
38	www.freepatentsonline.com	<1	Internet Data
39	www.intechopen.com	<1	Publication
40	sadefensejournal.com	<1	Internet Data
41	Signature Verification and Mail Ballots Guaranteeing Access While Preserving In by Janover-2020	<1	Publication
42	www.falmouth.ac.uk	<1	Internet Data
43	A Real-Time Ship Manoeuvring Simulation Study for the Strait of Istanbul (Bospor by Sarz-1999	<1	Publication
44	Broadband dynamic spectrum characterization based on gating-assisted electro-opt by Wei-2019	<1	Publication
45	Dealing with Market Dynamism The Role of Reconfiguration in Global Account Mana by Lind-2011	<1	Publication
46	IEEE 2014 International Topical Meeting on Microwave Photonics (MWP) by	<1	Publication
47	moam.info	<1	Internet Data
48	pdfcookie.com	<1	Internet Data

49	Site-dependent stability and electronic structure of single vacancy point defect by Shi-2013	<1	Publication
50	Thesis Submitted to Shodhganga, shodhganga.inflibnet.ac.in	<1	Publication
51	vlsics.blogspot.com	<1	Internet Data
52	www.diva-portal.org	<1	Publication
53	www.dx.doi.org	<1	Publication
54	Chemical composition of fennel seed extract and determination of fench by Alam-2019	<1	Publication
55	citeseerx.ist.psu.edu	<1	Internet Data
56	docplayer.net	<1	Internet Data
57	docplayer.net	<1	Internet Data
58	docs.replit.com	<1	Internet Data
59	documents.mx	<1	Internet Data
60	electrochemsci.org	<1	Publication
61	eprints.kingston.ac.uk	<1	Publication
62	IEEE 2018 Fourth International Conference on Computing Communicatio, by Burud, Anand Bhask- 2018	<1	Publication
63	Pathways towards enhanced techno-economic performance of flow battery by Zheng-2018	<1	Publication
64	pdfcookie.com	<1	Internet Data
65	Self-healing and phase behavior of liquid crystalline elastomer based on a block by Yan-2015	<1	Publication

66	Thesis Submitted to Shodhganga Repository	<1	Publication
67	Ultra-low power modulators using MOS depletion in a high-Q SiO2-clad silicon 2- by Anderson-2010	<1	Publication
68	www.dx.doi.org	<1	Publication
69	www.eajournals.org	<1	Internet Data
70	www.freepatentsonline.com	<1	Internet Data
71	www.ijssbt.org	<1	Publication
72	www.scribd.com	<1	Internet Data