CMR UNIVERSITY

Private University Estd in Karnataka State by Act No. 45 of 2013

SCHOOL OF ENGINEERING AND TECHNOLOGY

CAPSTONE BUILD PROJECT REPORT

ON

"Design and Implementation of SRT Radix-2 Divider using digit recurrence method for an unsigned integer using Verilog HDL on FPGA"

For the requirement of 8th Semester B.Tech in Electronics and Communication Engineering

Submitted By

GOWNI SAI NEHA	**(21BBTEC056)**
RAVIKIRAN A	**(21BBTEC058)**
SHAINA FAUSTINA F	**(21BBTEC059)**

Carried Out At

CMR UNIVERSITY

Under the Guidance of

INTERNAL GUIDE

Dr. VIJAYA BHARATHI M
Associate Professor,
Dept. of ECE SoET,
CMRU Bengaluru- 562149

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
CMR University
Off Hennur- Bagalur Main Road,
Near Kempegowda International Airport, Chagalahatti,
Bengaluru, Karnataka- 562149

Academic Year 2024-25

CMR UNIVERSITY

Private University Estd in Karnataka State by Act No. 45 of 2013

SCHOOL OF ENGINEERING AND TECHNOLOGY

Chagalahatti, Bengaluru, Karnataka-562149

Department of Electronics and Communication Engineering

CERTIFICATE

Certified that the Capstone Build Project entitled "Design and Implementation of SRT Radix-2 Divider using digit recurrence method for an unsigned integer using Verilog HDL on FPGA**"** carried out by **GOWNI SAI NEHA (21BBTEC056), RAVIKIRAN A (21BBTEC058) & SHAINA FAUSTINA F (21BBTEC059),** bonafide students of **SCHOOL OF ENGINEERING AND TECHNOLOGY**, in partial fulfilment for the award of **BACHELOR OF TECHNOLOGY** in 8th-semester Electronics and Communication Engineering of **CMR UNIVERSITY**, Bengaluru during the year 2025. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the report. The Capstone Build project has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Signature of Guide	Signature of HOD	Signature of Dean
……	..……	.....……
Dept. of ECE SOET, CMRU, Bangalore	Dept. of ECE SOET, CMRU, Bangalore	SOET, CMRU Bangalore

External Viva:

Name of the Examiners:	Signature with Date:
1 …….	……..
2 …….	...……

DECLARATION

We, **GOWNI SAI NEHA, RAVIKIRAN A, SHAINA FAUSTINA F** bearing USN **21BBTEC056, 21BBTEC058, 21BBTEC059** student of Bachelor of Technology, Electronics and Communication Engineering, CMR University, Bengaluru, hereby declare that the Capstone Build project work entitled "Design and Implementation of SRT Radix-2 Divider using digit recurrence method for an unsigned integer using Verilog HDL on FPGA" submitted by us, for the award of the Bachelor's degree in Electronics and Communication Engineering to CMR University is a record of bonafide work carried out independently by us under the supervision and guidance of **Dr. VIJAYA BHARATHI M** Associate Prof, ECE Dept. CMR University.

We further declare that the work reported in this project work has not been submitted and will not be submitted, either in part or in full, for the award of any other degree in this university or any other institute or University.

Place: Bengaluru

Date:

GOWNI SAI NEHA (21BBTEC056)

RAVIKIRAN A (21BBTEC058)

SHAINA FAUSTINA F (21BBTEC059)

ABSTRACT

This project focuses on the design and implementation of an SRT radix-2 divider for unsigned integers using Verilog HDL on an FPGA. The division algorithm, based on the digit recurrence method, iteratively generates quotient bits by employing a lookup table for quotient digit selection. The design was implemented and verified using Verilog HDL and synthesized onto an FPGA platform. Performance metrics, including area utilization and maximum operating frequency, were analyzed. The implemented divider demonstrates the feasibility of high-performance division using the SRT algorithm on FPGAs, offering a balance between speed and hardware resources.

Design and implementation of an (Sweeney, Robertson, Tocher) SRT (Unsigned Radix-2) divider is a critical component in the field of digital arithmetic, particularly in high-performance computing and embedded systems. The SRT algorithm, which is a modification of the long division method, leverages the use of digit selection and recurrence relations to minimize the number of required operations, making it faster and more efficient than traditional division methods. We explore the various stages of the design, including digit generation, quotient prediction, and correction mechanisms. The divider is implemented using a SRT Algorithm of Digit recurrence method to achieve a balance between speed and area efficiency. Simulation results show that the proposed SRT divider offers significant improvements in terms of throughput and resource utilization compared to conventional dividers. The design is suitable for integration into processors and systems that require high-speed division operations, with applications in fields such as digital signal processing, cryptography, and high-performance computing.

The flexibility of field programmable gate arrays (FPGAs) allows the rapid development of high performance custom hardware. By selecting arithmetic algorithms suited to the FPGA technology and subsequently applying optimal mapping strategies, high performance FPGA implementations can be developed. In microcontrollers, mathematical operations such as addition, subtraction, multiplication and division are performed using the Arithmetical Logical Unit (ALU) thought FPGAs does not offer such a unit. In FPGAs, arithmetic operations are performed using the shift registers, lookup tables, hardware multipliers, and logic gates. Among the basic arithmetic operations (addition, subtractions, multiplications and divisions) divisions is the most complex and expensive among the four arithmetic operations in hardware.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of this project would be incomplete without the mention of the people who made it possible, without whose constant guidance and encouragement would have made efforts go in vain.

We extend our sincere gratitude to **CMR University** for their invaluable support and collaboration on this Project. Their resources and insights were crucial to its completion. Their support was instrumental in the research and development process.

We express our heartfelt sincere gratitude to **Dr. N. KANNAN, Dean,** School of Engineering and Technology, CMR University for his support.

We would like to express our thanks to **Dr. K V PRASAD, Head**-Department of Electronics and Communication Engineering, School of Engineering and Technology, CMR University, Bangalore, for his encouragement that motivated me for the successful completion of Project work.

We consider ourself privileged to express gratitude and respect towards all those who guided me through the completion of the project. We express our thanks to our Internal Project Guide **Dr. VIJAYA BHARATHI M**, **Assoc. Professor**, Department of Electronics and Communication Engineering, School of Engineering and Technology, CMR University for his constant support.

(GOWNI SAI NEHA)

(RAVIKIRAN A)

(SHAINA FAUSTINA F)

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

A simple and widely implemented class of division algorithm is digit recurrence. The most common implementation of digit recurrence division in modem microprocessors is SRT division, taking its name from the initials of Sweeney, Robertson and Tocher, who developed the algorithm independently at approximately the same time. SRT division uses subtraction as the fundamental operator to retire a fixed number of quotient bits in every iteration. Two fundamental works on SRT division are those of Atkins, the first major analysis of SRT algorithms, and Tan, a derivation of high-radix SRT division and an analytic method of implementing SRT look-up tables. The SRT Radix-2 Divider is a high-performance division algorithm used in digital systems to perform division operations. It is based on a digit recurrence method, which iteratively produces quotient digits. Due to its efficiency and speed, the SRT Radix-2 Divider is widely used in modern processors, including floating-point units, where rapid and accurate division operations are essential for performance.

The SRT algorithm uses a lookup table to determine the quotient digit for each iteration.The algorithm uses a redundant representation for the quotient. The algorithm performs a multiplication by the quotient digit in each iteration.The algorithm uses a few most significant bits of the dividend and partial remainder to select the quotient digit. The speed of SRT-based dividers is mainly determined by the complexity of the quotient-digit selection logic.The proper selection of radix and algorithm are critical factors in efficiently matching division to a given set of FPGA characteristics.A higher radix will have greater combinational logic depth but fewer required iterations.

The SRT radix-2 divider has become an indispensable component in modern processors. Its speed and efficiency have enabled significant performance improvements in a wide range of applications. Understanding the underlying principles of SRT division is crucial for anyone working in computer architecture, digital design, or related fields.

Importance of Binary Division

Binary division is a fundamental arithmetic operation in digital computing and plays a crucial role in various applications, including computer architecture, digital signal processing (DSP), and scientific computations. Unlike addition and multiplication, division is more complex and computationally expensive, requiring specialized algorithms such as **restoring division, non-restoring division, and SRT division** to improve efficiency.

One of the primary reasons binary division is important is its application in **floating-point arithmetic**, where division is frequently used for calculations in scientific computing, graphics processing, and machine learning. Floating-point units (FPUs) in modern processors rely on efficient division techniques to handle complex mathematical operations with precision.

In **computer networking and cryptography**, binary division is used for error detection and correction algorithms, such as **cyclic redundancy check (CRC)**, which ensures data integrity in communication systems. It is also essential in modular arithmetic, which forms the backbone of encryption techniques like RSA and ECC, securing digital communication.

Moreover, in **digital control systems and DSP**, division is used in real-time signal filtering, image processing, and machine learning algorithms. For example, in adaptive filters, division is necessary for computing weight adjustments, ensuring optimal performance in noise reduction and predictive analytics.

Binary division is also essential in **processor design and optimization**, as it enables efficient execution of high-level programming operations. Modern CPUs and GPUs incorporate optimized division circuits that balance speed and power consumption, making computing systems faster and more energy-efficient.

In conclusion, binary division is a critical operation in digital electronics and computing, enabling precise and efficient processing across multiple domains. Its role in arithmetic computations, data integrity, security, and signal processing highlights its significance in modern technology. Efficient division algorithms and hardware implementations continue to drive advancements in computational performance and reliability.

Types of Division

1. **Restoring Division** – This is a traditional binary division method where the remainder is restored (i.e., corrected) if the subtraction results in a negative value. It involves repeated subtraction and shifting, making it relatively slow but straightforward for hardware implementation.

2. **Non-Restoring Division** – Unlike restoring division, this method avoids unnecessary corrections by making decisions based on the sign of the remainder. If the remainder is negative, addition is used instead of restoration, improving efficiency and reducing computation time.

3. **SRT Division** – Named after its inventors (Sweeney, Robertson, and Tocher), SRT division uses a redundant number system where quotient digits can be +1, 0, or -1. This allows for parallelism and faster execution, making it ideal for high-speed computing applications such as floating-point arithmetic in CPUs.

4. **Array Division** – This technique is similar to array multiplication, where partial results are computed and summed in a structured manner. It is mainly used in specialized digital arithmetic circuits for high-speed operations.

5. **Goldschmidt Division** – An iterative method that uses multiplication to approximate the reciprocal of the divisor, allowing division to be performed using multiplication and addition operations. It is commonly used in floating-point units for high-speed calculations.

6. **Newton-Raphson Division** – Based on the Newton-Raphson iterative method, this approach approximates the reciprocal of the divisor and multiplies it with the dividend to compute the quotient. It is widely used in modern processors due to its rapid convergence and efficiency in floating-point arithmetic.

Efficiency of SRT Radix-2 Division Compared to Other Methods

The **SRT Radix-2 division algorithm** is more efficient than traditional division methods, such as **restoring and non-restoring division**, due to its **redundant quotient digit selection (+1, 0, -1)** and efficient handling of remainders. Here's why it stands out compared to other division techniques:

1. **Faster Convergence with Redundant Number System**

   - Unlike traditional division, which only allows quotient digits of 0 or 1, the SRT algorithm uses +1, 0, -1. This redundancy enables **partial corrections** at each step, reducing the chance of incorrect quotient estimates and minimizing the need for backtracking or restoring operations.

2. **Fewer Arithmetic Operations**

   - Restoring division requires a correction step (restoring the remainder when subtraction results in a negative value), making it slower. Non-restoring division improves on this but still performs unnecessary operations. SRT Radix-2 eliminates unnecessary corrections and reduces the number of required arithmetic steps, leading to **higher efficiency**.

3. **Parallelism and Pipelining**

   - The SRT algorithm allows **parallel processing** of quotient digits and can be implemented in a **pipeline structure**. This means that new division operations can start before previous ones finish, making it suitable for **high-performance processors**.

4. **Efficient Hardware Implementation**

    - Since SRT Radix-2 only involves **simple shift and subtraction operations**, it is well-suited for hardware implementations in CPUs, GPUs, and DSPs. Compared to Newton-Raphson and Goldschmidt division (which require multiplications), SRT is **hardware-friendly** and requires fewer logic gates.

5. **Balanced Trade-off Between Speed and Complexity**

    - While high-radix division (e.g., SRT Radix-4, Radix-8) provides even faster division, it requires complex lookup tables and additional hardware resources. SRT Radix-2 provides a **good balance between speed and hardware simplicity**, making it a practical choice for many digital systems.

Chapter 2

LITERATURE SURVEY

The SRT (Sweeney, Robertson, and Tocher) division algorithm is a widely used digit recurrence method for efficient division in digital circuits. It improves upon restoring and non-restoring division by employing quotient-digit selection using a lookup table, reducing iterations while maintaining computational efficiency. Implemented in radix-n, higher radix values decrease iteration count but increase hardware complexity. The algorithm determines the quotient iteratively using a decision function based on the most significant bits of the divisor and remainder, eliminating the need for full-width subtraction in every cycle. FPGA implementations optimize SRT division through operand pre-scaling, pipelining, and redundant representation techniques to improve speed and efficiency. However, challenges include complex quotient digit selection logic and the trade-off between radix size and hardware area. Modern implementations incorporate pipeline execution, operand preconditioning, and hybrid strategies to enhance performance. Due to its balance of speed and hardware efficiency, SRT division is widely used in processors and digital arithmetic units like Intel Pentium and Sun UltraSPARC [1].

Division is a fundamental arithmetic operation that presents significant implementation challenges in digital hardware. This has led to the development of numerous division algorithms throughout the history of computing. Patankar et al. (2020) provide a comprehensive overview of division algorithms, categorizing them into digit recurrence, functional iteration, high-radix, and others. The paper highlights the trade-offs between algorithm complexity, area efficiency, and latency, noting that while digit recurrence methods like restoring and non-restoring division are conceptually simple, they often suffer from performance limitations[2].

This paper introduces a novel redundant representation, where 5 bits store every 4-bit partial remainder, reducing flip-flop usage compared to conventional carry-save methods. The proposed approach balances cost and speed by employing 4-bit carry-propagate adders (CPAs) instead of full carry-save representations, achieving lower hardware complexity while maintaining performance. Unlike conventional radix-2 SRT, which requires a final carry-propagate adder (CPA) for remainder conversion, the (5,4) method minimizes additional processing overhead. FPGA implementations of this approach show a 37.5% reduction in flip-flop storage and a 10% improvement in area efficiency compared to traditional carry-save dividers. While high-radix SRT designs reduce iteration count, they increase quotient selection complexity; hence, grouping radix-2 stages into an effective radix-16 structure achieves a trade-off between performance an

resource usage. The (5,4) redundant representation thus provides an optimized design for high-speed, low-cost division in FPGA-based digital arithmetic applications[3].

This paper presents a novel approach to radix-2 division algorithms by introducing an over-redundant digit set {-2, -1, 0, 1, 2} for quotient digit selection. This deviates from the traditional SRT algorithm that uses {-1, 0, 1}. The authors derive four new division algorithms, each employing the redundant digit set and utilizing either two's complement or binary signed-digit representation for the partial remainder. The algorithms are analysed using a novel method in the (rs, rc) plane, offering a different perspective compared to conventional P-D diagrams. The implementation results demonstrate that the new algorithms achieve a 10% improvement in latency per iteration over the standard radix-2 SRT algorithm, albeit with increased power and area costs. This research contributes to the field of computer arithmetic by providing new division algorithms that offer a trade-off between speed and hardware resources [4].

This paper introduces a carry-free subtractive division algorithm inspired by the Svoboda-Tung method, aiming to reduce carry propagation delays. Unlike conventional SRT division, which relies on addition/subtraction with quotient selection tables, this method represents both the quotient and partial remainder in signed-bit format, enabling faster computations. The algorithm eliminates carry propagation by employing a table lookup-like approach and an on-the-fly quotient conversion, ensuring efficient real-time implementation. A prescaling step prevents overflow, further stabilizing the iteration process. Implemented as a 32-bit divider in Verilog HDL, the design demonstrates high-speed operation with reduced area requirements. Compared to standard SRT-based architectures, this method provides a more efficient trade-off between hardware complexity and performance, making it suitable for FPGA and VLSI applications requiring high-speed division[5].

This paper analyzes radix-2 and radix-4 SRT divider architectures, highlighting trade-offs in area and performance for FPGA and microprocessor implementations. While higher radix reduces iteration count, it introduces increased complexity in quotient selection logic and hardware area. The study discusses optimization techniques like operand prescaling, quotient selection overlapping, and redundant remainder representation to improve speed and efficiency. Experimental results indicate that radix-2 SRT dividers provide a balance between performance and hardware cost, with latency primarily determined by quotient selection and partial remainder computations. Advanced circuit techniques like skew-tolerant domino logic further enhance SRT divider efficiency. The findings suggest that while radix-4 offers marginal speed benefits, radix-2 remains a viable choice for FPGA-based digital systems, striking a balance between computation speed, area efficiency, and implementation complexity[6].

Unlike conventional restoring and non-restoring division, SRT division improves performance by selecting quotient digits based on the most significant bits of the divisor and remainder, reducing iteration cycles while maintaining computational accuracy. The study compares Radix-2 and Radix-4 SRT dividers with other methods, showing that Radix-2 SRT offers a balance between speed and hardware complexity. Key optimizations include normalization to keep the divisor within a specific range and on-the-fly conversion to streamline quotient formation. While higher-radix versions reduce iteration count, they introduce increased lookup table complexity and selection logic overhead. FPGA-based testing demonstrates that SRT radix-2 division maintains a linear relationship between bit-width and clock cycles, making it an efficient choice for moderate-speed applications. The study also highlights how varying bit-widths impact performance, emphasizing the trade-offs between area, speed, and complexity in digital division implementations [7].

This paper presents an optimized radix-2 division approach with over-redundant quotient selection to improve hardware efficiency and computational speed. By employing over-redundant quotient digits, the design simplifies quotient selection logic, reducing cycle time while maintaining accuracy. The study highlights that higher radix reduces iteration count but increases complexity, making radix-2 a balanced choice for FPGA-based implementations. The proposed architecture incorporates carry-free adder/subtractor logic to minimize delay and enhances performance through quotient digit reduction techniques. FPGA synthesis results on the Xilinx XC4010 show that the implementation achieves improved area efficiency while preserving division accuracy. Compared to traditional SRT-based designs, the over-redundant approach offers a competitive trade-off between speed and hardware utilization, making it a viable solution for high-performance division operations in FPGA-based arithmetic units[8].

This paper explores the implementation of various division algorithms, including SRT, restoring, and non-restoring methods, highlighting their trade-offs in terms of speed, hardware utilization, and accuracy. The study demonstrates that the SRT division algorithm provides a balance between area efficiency and computational performance, making it suitable for FPGA-based designs requiring moderate precision. Optimizations such as quotient digit selection and carry-free remainder computation enhance efficiency, while comparisons with the Xilinx Pipelined Divider IP Core indicate that alternative approaches, like non-restoring division, can achieve significantly higher speeds. FPGA synthesis results on Virtex-II Pro confirm that the SRT radix-2 divider is well-suited for high-speed, fixed-point arithmetic applications, offering a trade-off between iteration count and hardware complexity while maintaining efficient resource usage[9].

This paper presents an analysis of the n-bit SRT divider, comparing its performance and area efficiency with other division techniques. The study highlights that SRT division reduces carry propagation delay by

employing redundant representations, improving hardware efficiency. Radix-2 and radix-4 implementations are analysed, with radix-2 providing a balance between complexity and performance. The paper also discusses FPGA implementation using VHDL, demonstrating that an n-bit divider achieves lower total delay compared to fixed-bit-width dividers while reducing power and area consumption. Simulation results validate that SRT division is an effective approach for high-speed digital arithmetic, making it suitable for FPGA and VLSI applications requiring efficient division operations[10].

In modern floating point units, SRT dividers often use radix-2 and radix-4 stages, with higher radix dividers formed by combining these lower-radix stages. Although the maximallyredundant radix-2 design achieves the lowest core delay, it requires additional time and hardware outside the iterations for generating three divisor multiples, It show the performance and area results for a wide variety of divider architectures and implementations. We conclude that divider performance is only weakly sensitive to reasonable choices of architecture but significantly improved by aggressive circuit techniques [11].

Chapter 3

BLOCK DIAGRAM

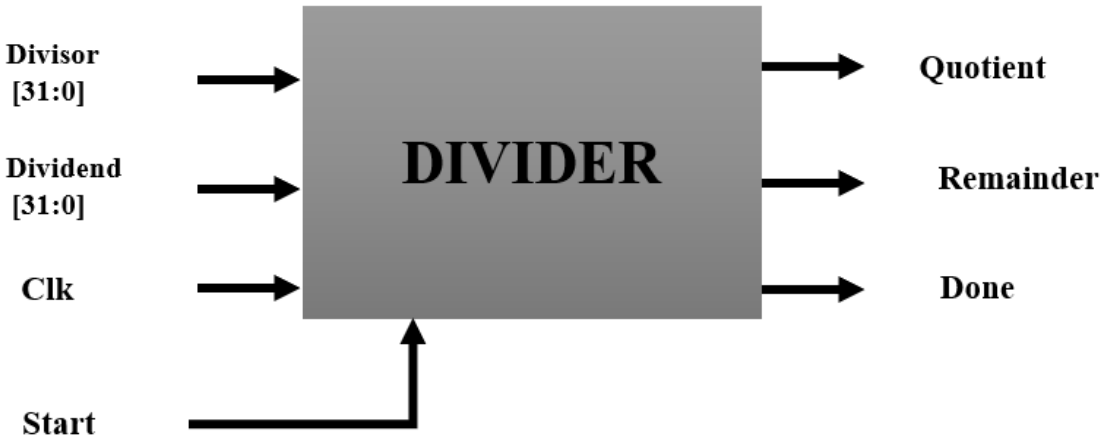

Fig 3:- Divider I/O's Block

The Above Fig(3) shows the Block Diagram of an SRT Radix-2 Divider, a fundamental component in digital systems for performing division operations. The circuit takes two 32-bit inputs: the **Dividend** (the number being divided) and the **Divisor** (the number by which the dividend is divided). It also receives a **Clk** (clock) signal for synchronous operation and a **Start** signal to initiate the division process. Upon receiving the Start signal and with each clock cycle, the circuit performs the necessary calculations to compute the **Quotient** (the result of the division) and the **Remainder** (the leftover after the division). Finally, the circuit outputs a **Done** signal to indicate the completion of the division operation. This Done signal can be used by other parts of the system to signal that the results are ready for use. The use of 32-bit inputs suggests the circuit is designed to handle a wide range of numerical values, making it suitable for various applications requiring precise division calculations. The inputs are unsigned Binary Bits, here we are considering 32bits by 32 bits Division using SRT Algorithm, specified for Radix-2 which only consist of "0" & "1".

3.1 Block Diagram

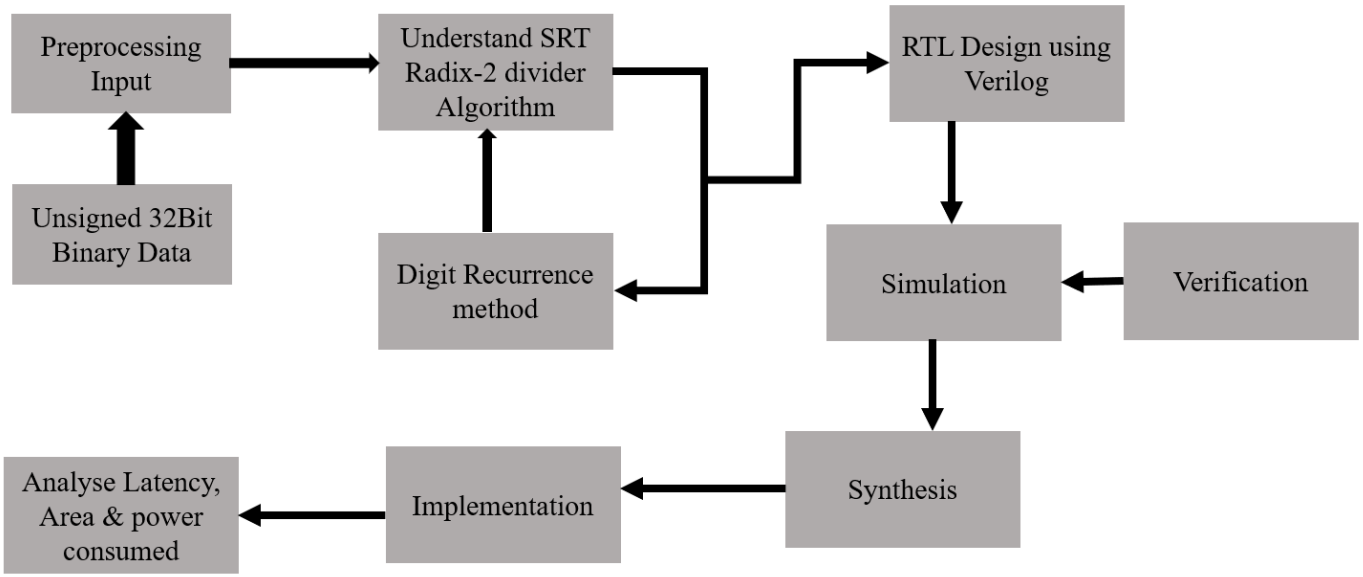

Fig 3.1:- Block Diagram of Work Flow

The Above Fig(3.1) gives workflow diagram represents the step-by-step process involved in the design and implementation of the SRT Radix-2 Divider using the digit recurrence method. Below is a detailed explanation of each block in the diagram:

1. **Unsigned 32Bit Binary Data:**

The input values (dividend and divisor) are prepared and formatted for processing. The divisor is checked to avoid division by zero, and input normalization techniques may be applied.

- **Description:** This represents the input data for the divider. It's specified as unsigned (non-negative) and 32 bits wide, meaning it can represent numbers from 0 to 2³² - 1. This data will be the dividend (the number being divided).
- **Role:** This is the starting point. The 32-bit binary data is the raw input that needs to be processed by the divider.

2. Preprocessing Input:

- **Description:** This stage involves preparing the input data for the SRT Radix-2 algorithm. It might include tasks like:
    - **Normalization:** Ensuring the dividend and divisor are in a specific range for the algorithm to work correctly.

- **Sign Handling (if needed):** While the input is unsigned, internal calculations might require sign handling for intermediate results.
- **Padding:** Adding leading zeros if necessary.

- **Role:** This step ensures the input data is in the correct format for the SRT Radix-2 division algorithm.

2. Understand SRT Radix-2 Divider Algorithm:

A detailed study of the SRT Radix-2 division algorithm is conducted. The algorithm follows the digit recurrence method, which generates quotient bits sequentially using a redundant number system.

- **Description:** This is the core of the design process. It involves:
  - **Studying the SRT (Sweeney-Robertson-Tocher) algorithm:** Understanding the principles of non-restoring division and how it uses digit selection to efficiently compute the quotient.
  - **Radix-2 Understanding:** Recognizing that Radix-2 means each iteration of the algorithm processes one bit of the quotient.
  - **Digit Selection Table:** Understanding how the algorithm uses a table to select quotient digits based on the partial remainder and divisor.
- **Role:** This is the conceptual and theoretical foundation. A thorough understanding of the algorithm is crucial for accurate implementation.

3. Digit Recurrence Method:

The division operation is carried out iteratively, where each step produces one quotient digit based on a lookup table or a simplified decision logic. Partial remainders are updated in every iteration to refine the quotient.

- **Description:** This block refers to the iterative nature of the SRT algorithm. It involves:
  - **Partial Remainder Calculation:** Calculating the partial remainder at each step.
  - **Quotient Digit Selection:** Using the digit selection table to determine the next quotient bit.
  - **Iteration:** Repeating the above steps until the desired precision is achieved.
- **Role:** This step focuses on the actual computational process within the SRT algorithm, detailing how the quotient is built bit-by-bit.

4. **RTL Design using Verilog:**

The algorithm is converted into a Register Transfer Level (RTL) design using Verilog HDL. The design includes modules for quotient selection, remainder computation, and quotient correction.

- **Description:** This is the hardware implementation stage. It involves:
    - **Writing Verilog Code:** Translating the SRT Radix-2 algorithm into synthesizable Verilog code.
    - **Defining the Architecture:** Specifying the registers, adders, subtractors, and other hardware components needed.
    - **Control Logic:** Designing the logic that controls the data flow and sequencing of operations.
- **Role:** This step converts the algorithmic understanding into a hardware description that can be synthesized into a physical circuit.

5. **Simulation:**

The Verilog design is simulated in Xilinx ISE or ModelSim to verify correctness. The simulation helps in debugging and ensuring the design works as expected.

- **Description:** This involves verifying the correctness of the Verilog code using a simulator. It includes:
    - **Creating Testbenches:** Writing Verilog code to generate test vectors and apply them to the design.
    - **Functional Verification:** Checking if the output of the design matches the expected results for various test cases.
    - **Timing Simulation (optional):** Analyzing the timing behavior of the design to ensure it meets performance requirements.
- **Role:** This step ensures the Verilog code accurately implements the intended functionality.

6. **Verification:**

Test cases are applied to validate the SRT divider for different inputs. Functional verification ensures that the quotient and remainder are computed correctly.

- **Description:** This is a more comprehensive verification process, often involving:
    - **Formal Verification:** Using mathematical techniques to prove the correctness of the design.

- **Coverage Analysis:** Measuring how thoroughly the test cases exercise the design.
- **Assertion Checking:** Adding assertions to the Verilog code to detect errors during simulation.

- **Role:** This step aims to provide a higher level of confidence in the design's correctness.

7. Synthesis:

The RTL design is synthesized into FPGA hardware using Xilinx ISE or Vivado. This step converts the high-level Verilog code into hardware logic gates and registers.

- **Description:** This step involves converting the Verilog RTL code into a gate-level netlist using a synthesis tool. It includes:
  - **Technology Mapping:** Mapping the generic RTL components to specific gates in a target technology library.
  - **Optimization:** Optimizing the gate-level netlist for area, performance, and power.
- **Role:** This step translates the hardware description into a format that can be implemented on a physical chip.

8. Implementation:

The synthesized design is mapped onto an FPGA board for hardware execution. Constraints such as timing, power, and area optimization are considered.

- **Description:** This stage involves placing and routing the synthesized netlist on a target FPGA or ASIC. It includes:
  - **Placement:** Assigning physical locations to the gates on the chip.
  - **Routing:** Connecting the gates using wires.
  - **Clock Tree Synthesis:** Designing the clock distribution network.
- **Role:** This step creates the physical layout of the circuit on the target hardware.

10. Analyze Latency, Area & Power Consumed:

The final design is analyzed based on latency (number of clock cycles required for division), area utilization (LUTs, flip-flops, and BRAM usage), and power consumption (static and dynamic power).The results help in evaluating the efficiency and feasibility of the SRT Radix-2 Divider for real-time applications.

- **Description:** This involves evaluating the performance and resource utilization of the implemented design. It includes:

- **Latency Measurement:** Determining the time it takes for the divider to produce a result.
- **Area Estimation:** Calculating the amount of hardware resources used.
- **Power Consumption Analysis:** Estimating the power consumed by the circuit.

- **Role:** This step provides feedback on the design's efficiency and helps in making trade-offs between performance, area, and power.SRT Radix-2 Division ALGORITHM

3.2 Block Diagram of SRT Algorithm

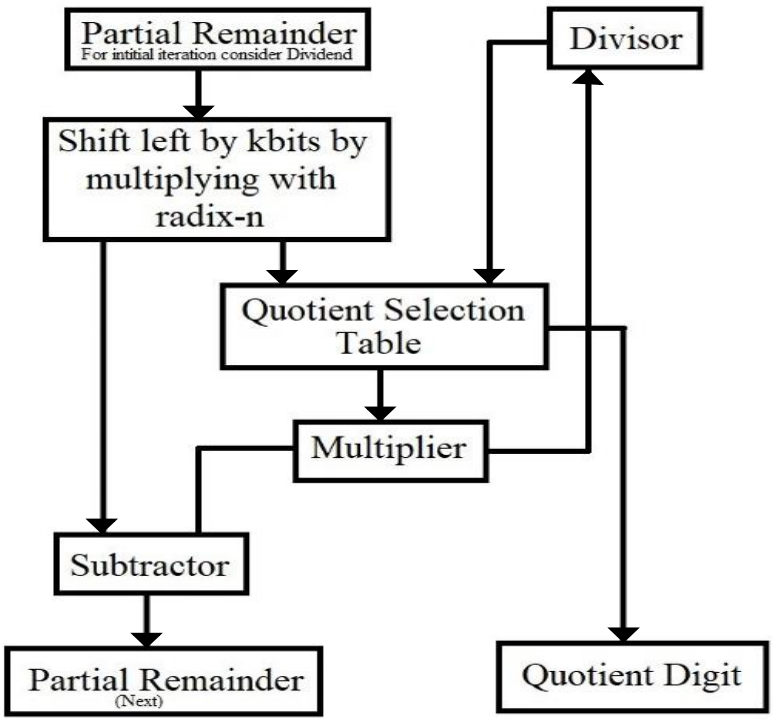

Fig 3.2:- Block Diagram of SRT Algorithm

The Above Fig(3.2) shows the SRT (Sweeney, Robertson, Tocher) division algorithm is a digit-recurrence method used in digital arithmetic to compute division efficiently by selecting quotient digits iteratively. Unlike traditional division methods, SRT uses redundant representation, allowing quotient digits to be chosen from a set like $\{-1,0,1\}$for radix-2, which simplifies quotient selection and avoids full-width subtraction in each iteration. The algorithm operates by maintaining a partial remainder, updating it at each step based on the selected quotient digit and the divisor using the recurrence relation $P_{j+1} = 2P_j - q_jD$. The quotient digit qj is determined using a lookup table based on the most significant bits of the partial remainder and divisor, reducing computational complexity. If the remainder becomes negative, correction is applied by adjusting the quotient and remainder in the final step. SRT division is widely used in floating-point units, processors, and FPGA-based arithmetic circuits due to its speed, efficiency, and suitability for high-radix implementations, making it a preferred choice for high-performance division operations.

1.Partial Remainder (Initialization)

The **Partial Remainder** block stores the value that will be used for iterative division steps. Initially, this is set to the **Dividend**, as the dividend itself acts as the first remainder. This remainder will be updated in each step as the division progresses. The **SRT division method** follows a redundant number system, meaning that it allows for intermediate values that might be outside the standard number range but will eventually converge to the correct result.

2. Shift Left by k-bits (Multiplying by Radix-n)

Before performing the division step, the **Partial Remainder** is shifted left by k bits, which corresponds to multiplying it by the **radix (n)** of the number system being used (for example, radix-2 for binary division). This shift operation allows the algorithm to bring down the next set of bits from the dividend for processing. By shifting left, the method prepares for selecting the next quotient digit and ensures the remainder maintains a scaled value suitable for efficient subtraction.

3. Quotient Selection Table

The **Quotient Selection Table** is a critical component in the SRT division process. Instead of computing the quotient digit directly through trial division, this table maps the **Partial Remainder** and **Divisor** to a predefined quotient digit. The selection is based on approximate values, which reduces computation time. The quotient digit can be either positive, negative, or zero, depending on the values of the remainder and divisor. This approach helps in handling cases where the remainder fluctuates between positive and negative values due to the redundant number system.

4. Multiplier

Once a quotient digit is selected from the **Quotient Selection Table**, it is **multiplied** by the divisor to determine the value that needs to be subtracted from the **Partial Remainder**. This multiplication operation is essential for computing the next remainder. The use of a **redundant quotient system** allows the quotient digits to be in the range of {-2, -1, 0, 1, 2} rather than just {0,1}, which provides greater flexibility and speeds up convergence.

5. Subtractor

The **Subtractor** computes the next partial remainder by subtracting the product of the divisor and the selected quotient digit from the current partial remainder. If the new remainder is negative, the algorithm may need to adjust the quotient selection in the next iteration. Since the SRT method allows for a range of quotient digits, this subtraction operation helps maintain accuracy while allowing small intermediate errors to be corrected in subsequent iterations.

6. Quotient Digit Storage

Each quotient digit selected in a given iteration is stored in this block. The **final quotient** is formed by accumulating these quotient digits across multiple iterations. Since the quotient digits may be represented in a redundant format, post-processing may be required to convert them into a standard binary representation.

7. Iterative Process for Next Partial Remainder

The next **Partial Remainder** is determined using the subtractor output, and this new remainder is then used in the next iteration of the algorithm. This process continues until the required number of iterations is completed or until the remainder is small enough that further computation is unnecessary. At the end of the process, the quotient and remainder are finalized and made available as outputs.

Chapter 4

IMPLEMENTATION

4.1Steps for SRT Radix-2 Division Algorithm

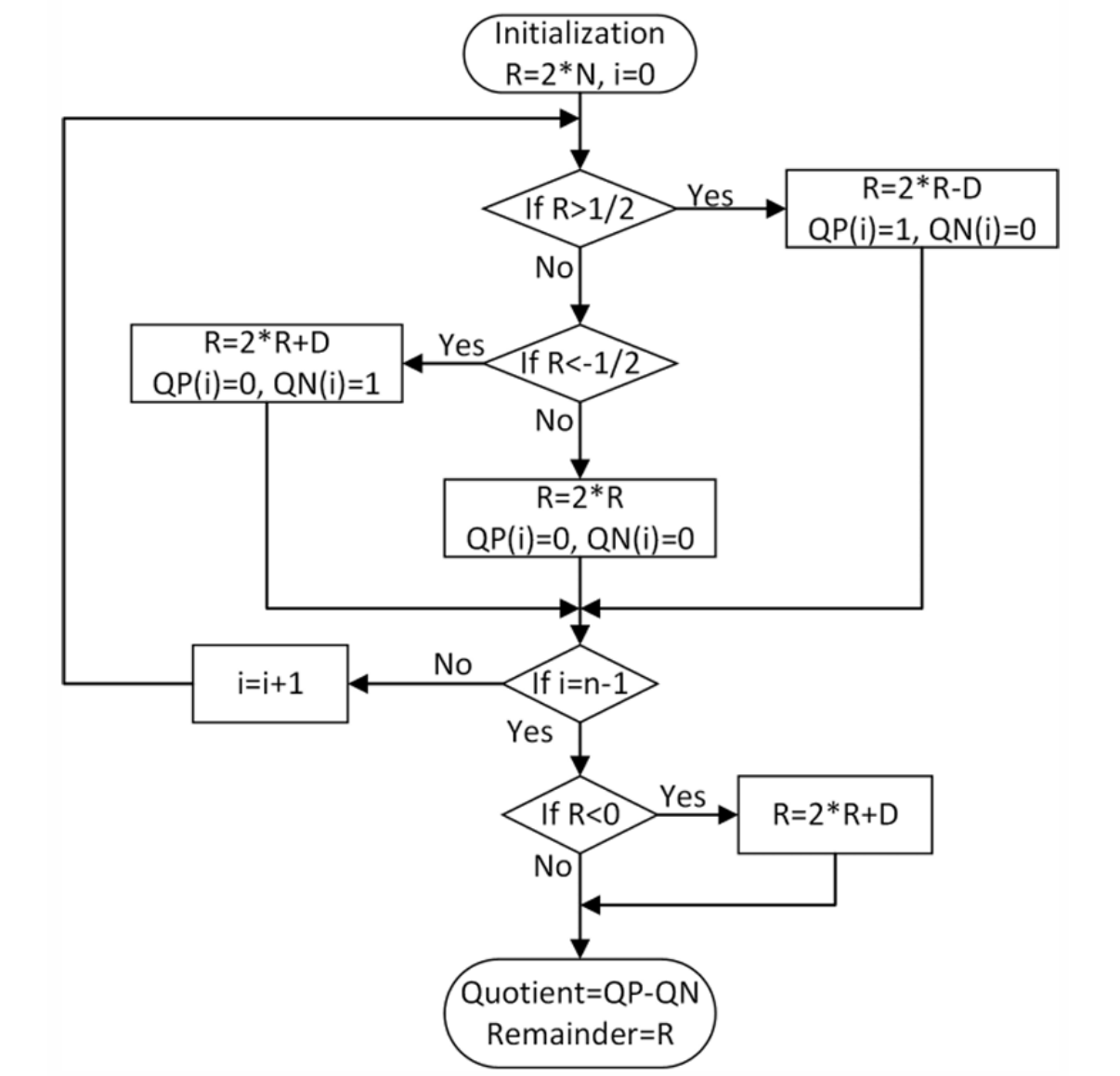

Fig 4.1:- Flow Chart of SRT Division Algorithm

1. **Initialization:**

   - Normalize the dividend (N) and divisor (D) if required to ensure they are within a specific range (e.g., [1,2]).
   - Set the initial **partial remainder**:

$P_0=N$

- Set the quotient register to zero.

2. **Iteration (for each step j):**

   - **Quotient Selection:** Determine qjq_jqj from the redundant set $\{-1,0,1\}$ using a lookup table based on the most significant bits of Pj and D:
     - If $Pj > D$, then $qj=1$
     - If $Pj < -D$, then $qj = -1$.
     - Otherwise, $qj = 0$.
   - **Partial Remainder Update:** Compute the new partial remainder using the equation:

     $$Pj+1 = 2Pj - qjD$$

   - **Quotient Register Update:** Append qj to the quotient.

3. **Convergence Check:**

   - Repeat the process until the required number of iterations (nnn) is reached or the remainder is within an acceptable range.

4. **Final Quotient Conversion:**

   - Since the quotient was computed using a redundant digit set, convert it to standard binary form.
   - If the final remainder is **negative**, apply a correction:

     $Qcorrected = Q - 1$

     $Rcorrected = R + D$

5. **Output the Quotient and Remainder:**

   - The final corrected **quotient Q** and **remainder R** are produced as the division result.

4.2Redundant Digit Set in SRT Radix-2

In radix-2 SRT division, the quotient digit set is:

$$q_j \in \{-1,0,1\}$$

This means that instead of only producing **0 or 1** as in conventional binary division, the quotient can take negative values, allowing more flexible remainder adjustments.

1. Initialization Stage

The division process begins with the initialization of the remainder (R) and the iteration counter (i). The remainder is set as R = 2 * N, where N is the dividend, effectively doubling its value to facilitate the iterative process. The counter i is initialized to 0, indicating the start of the algorithm. This setup ensures that the iterative refinement of the remainder proceeds systematically over multiple steps.

2. Checking if R > 1/2 (Quotient Selection: +1)

In this stage, the algorithm determines whether the remainder is greater than 1/2. If R > 1/2, it means that the current remainder can accommodate a subtraction of the divisor D while still maintaining a stable quotient estimate. Therefore, the remainder is updated as R = 2 * R - D, and the quotient positive bit QP(i) is set to 1, while the negative bit QN(i) remains 0. This step ensures that the quotient converges towards its correct value with minimal error.

3. Checking if R < -1/2 (Quotient Selection: -1)

If the remainder is not greater than 1/2, the algorithm checks whether it is less than -1/2. If R < -1/2, it indicates that the remainder is too small and needs to be corrected by adding the divisor D. In this case, the remainder is updated as R = 2 * R + D, and the quotient negative bit QN(i) is set to 1, while QP(i) remains 0. This step corrects for underestimation in previous iterations and keeps the quotient estimate balanced.

4. If R is in the Range [-1/2, 1/2] (Quotient Selection: 0)

If R falls within the range [-1/2, 1/2], it means that the remainder is well within an acceptable threshold where no subtraction or addition is needed. In this case, the remainder is simply updated as R = 2 * R, meaning that the next step of the algorithm will refine the remainder further without modifying the quotient. The quotient bits QP(i) and QN(i) are both set to 0, signifying that no quotient digit is selected at this step.

5. Incrementing the Iteration Counter

After processing the remainder, the iteration counter is incremented by i = i + 1. This step ensures that the algorithm progresses through the required number of iterations (n), where n is the bit-width of the quotient. If the maximum number of iterations has not yet been reached, the process loops back to the remainder checking stage for further refinement.

6. Checking if All Iterations are Completed

Once the counter reaches n - 1, the algorithm prepares for final adjustments to the remainder. This condition (i = n - 1) ensures that the algorithm completes the necessary number of iterations before determining the final quotient and remainder. If the iteration limit has not been reached, the process continues, refining the remainder further.

7. Final Correction of Remainder (if R < 0)

After completing the iterations, the algorithm performs a final check on the remainder. If R < 0, it means that an adjustment is needed to ensure a valid quotient. In this case, the remainder is updated as R = 2 * R + D, which effectively corrects any small errors accumulated during the iterative process. This final adjustment ensures that the remainder is correctly aligned for the final quotient computation.

8. Computing the Final Quotient and Remainder

In the final step, the actual quotient is determined using the equation Quotient = QP - QN. This formula reconstructs the quotient from its redundant representation, ensuring that the final result accurately reflects the division process. The remainder R is also finalized, providing the remainder value from the division operation. This marks the completion of the SRT division process, yielding an efficient and hardware-friendly approach to computing division.

4.3 Quotient Selection in SRT Radix-2

Faster Computation: Since carry propagation is avoided, the selection of quotient digits can be done in a parallel and simple manner using lookup tables.

Simplified Quotient Selection Logic: The quotient digit is chosen using only a few most significant bits of the partial remainder and divisor. This avoids complex computations.

No Full Subtraction Required: Because the quotient digit can take negative values, the algorithm does not need to fully compute the remainder before selecting the next quotient digit.

The selection of q_j is based on the most significant bits of the partial remainder (P_j) and divisor (D)

- If $P_j > D$, then $q_j = 1$ (quotient increases).
- If $P_j < -D$, then $q_j = -1$ (quotient decreases).
- Otherwise, $q_j = 0$ (quotient remains unchanged).

Thus, the redundant quotient selection prevents large variations in remainder values, making the hardware implementation simpler.

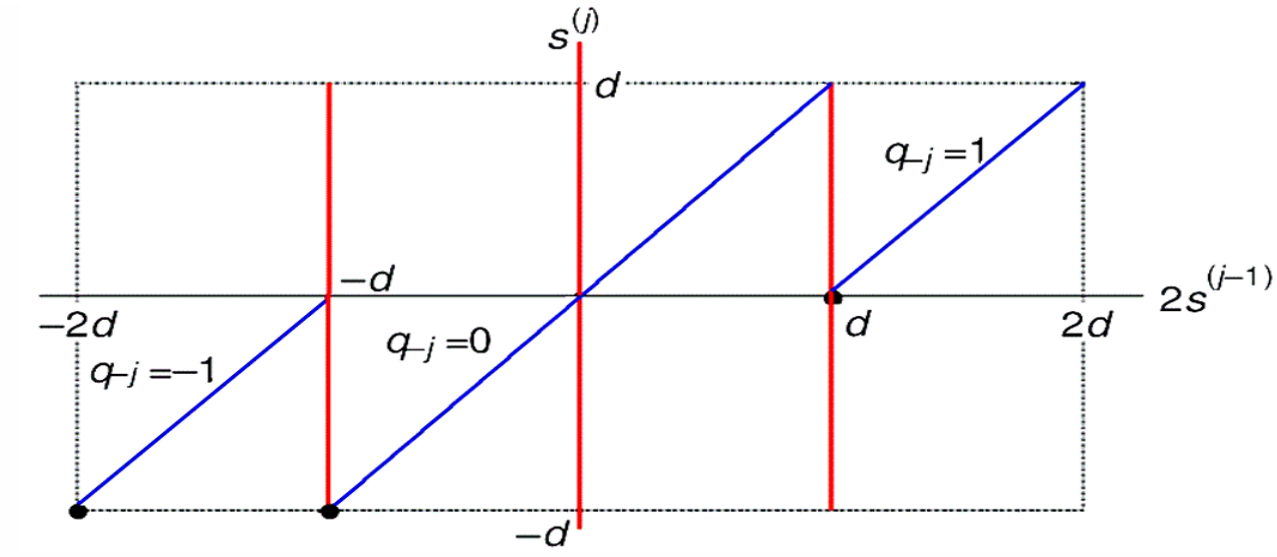

Fig 4.3:- Quotient selection on Partial Remainder

4.4Digit Recurrance Method

At each step jjj of the iteration, the partial remainder is updated as:

$$P_{j+1} = r \cdot P_j - q_j \cdot D$$

Where:

- P_j = Partial remainder at step j.
- r = Radix of the division (e.g., r = 2 for binary).
- q_j = Quotient digit selected at iteration jjj.
- D = Divisor.

The quotient is then accumulated as:

$$Q = \sum_{j=0}^{n} q_j \cdot r^{-j}$$

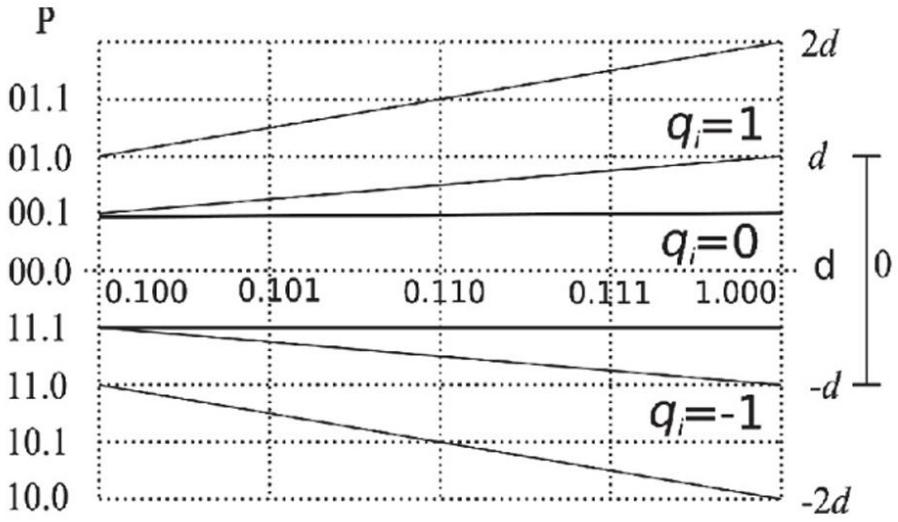

Fig 4.4:- P – D plot for SRT Radix-2 Division

4.5Quotient Digit Selection and the Lookup Table:

The heart of the SRT algorithm lies in its efficient quotient digit selection. In radix-2 SRT, the goal is to choose a quotient digit from the set {-1, 0, 1} that brings the partial remainder closest to zero. This decision is made based on a limited number of most significant bits (MSBs) of the partial remainder (P) and the divisor (D). These MSBs act as indices into a pre-computed lookup table.

The lookup table is designed to map these MSB combinations to the appropriate quotient digit (q). The table's size is determined by the number of MSBs considered from P and D. A larger number of bits allows for finer-grained control and potentially faster convergence, but it also increases the table size and complexity. A smaller table reduces complexity but might lead to slower convergence. The optimal number of bits is a trade-off determined by the specific hardware constraints and performance requirements.

4.6 Conversion from Redundant to Standard Binary:

After all iterations are complete, the quotient is represented in a redundant form (e.g., using -1, 0, and 1). This needs to be converted to a standard binary representation (using 0 and 1). Several methods can be used for this conversion. One common approach is to use a carry-save adder (CSA) to sum the positive and negative parts of the redundant quotient.

Chapter 5

Hardware and Software Requirements

5.1 Spartan 6 FPGA Development board

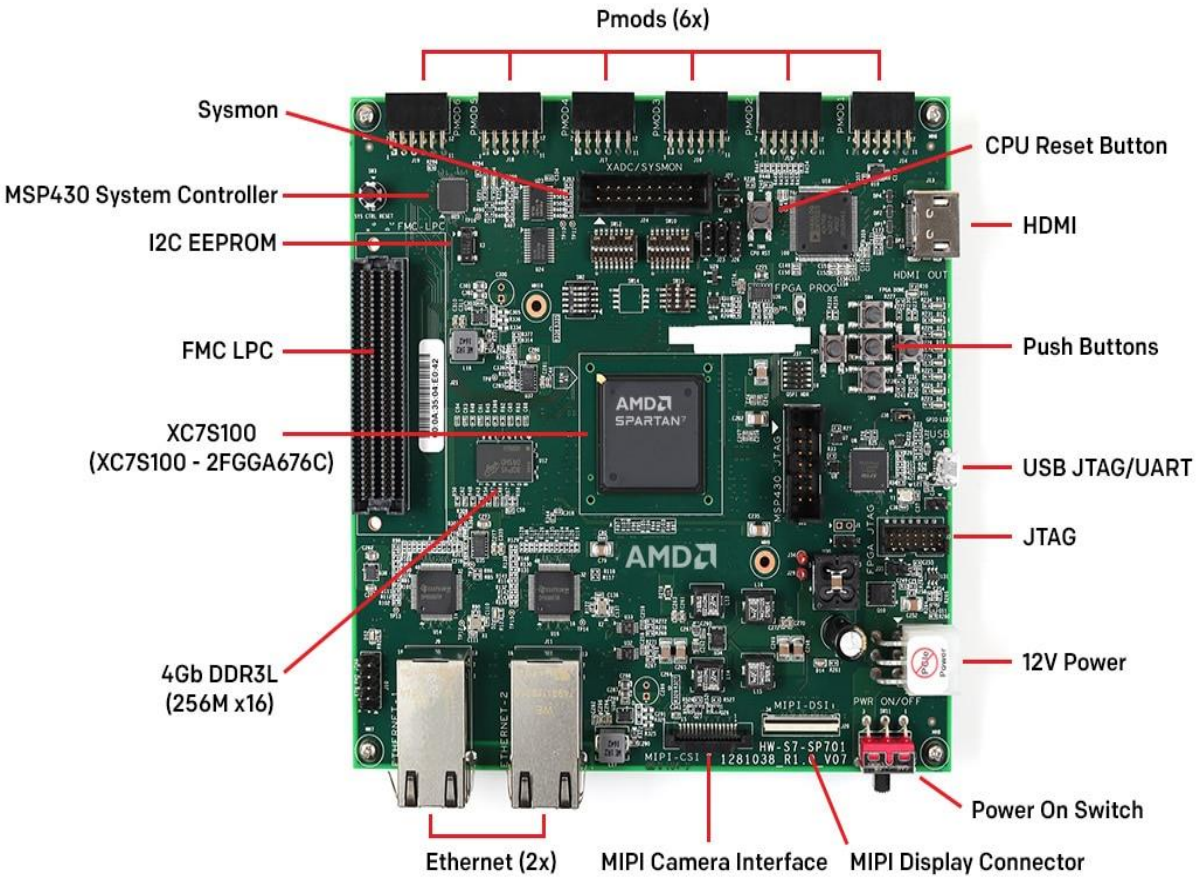

Fig 5.1:- a) Spartan 7 SP701 (xc7s100fgga676-2)

The Above Fig(4.1a) shows the Spartan-7 FPGA The AMD Spartan-7 SP701 FPGA Evaluation Kit is designed to provide developers, engineers, and students with a comprehensive and flexible platform for prototyping and deployment of a wide range of FPGA-based applications. This board is centered around the XC7S100-FGG676-2 FPGA, which belongs to the AMD Spartan-7 family known for offering an optimal blend of performance, power efficiency, and affordability.

At the heart of the SP701 board is the AMD Spartan-7 XC7S100-FGG676C FPGA. This FPGA features 102,400 logic cells which are distributed across 126,800 CLB flip-flops and 63,400 CLB LUTs, allowing the implementation of complex digital circuits. The device contains 4.5 Mb of block RAM and 240 DSP slices, supporting high-speed digital signal processing operations. With 400 I/O pins available in the FGG676 package, it supports flexible interfacing with external devices. The -2 speed grade indicates a medium-performance variant of the Spartan-7 series. This FPGA is ideal for mid-range applications that require

substantial logic resources without the cost of high-end devices. Regarding configuration and security, the FPGA supports multiple configuration methods, including JTAG (IEEE 1149.1) for debugging and programming, as well as SPI Flash and BPI Flash for standalone configuration. Additionally, AES-256 encryption is supported, ensuring bitstream security and protecting the design from unauthorized access.

To facilitate connectivity in industrial and networking environments, the SP701 includes two Gigabit Ethernet ports supporting 10/100/1000 Mbps speeds. These interfaces allow the board to be integrated into complex networks or to serve as a communication node. The board also provides USB connectivity via a combined USB-JTAG/UART port, allowing for easy programming and serial communication. The HDMI output port enables high-definition display interfacing, suitable for video applications and graphical output. An FMC LPC (Low Pin Count) connector expands the board's capabilities by allowing the addition of mezzanine cards for specialized I/O or data acquisition. Furthermore, six PMOD connectors offer flexible digital I/O options, conforming to Digilent's PMOD standard, and support a wide range of plug-in peripheral modules.

5.1.1. Architecture Overview

The **Spartan-7 XC7S100-FGG676** features a modern, scalable, and low-power FPGA architecture based on the Xilinx 7-Series family. It is optimized for high-performance logic utilization, DSP operations, and connectivity, all while maintaining energy efficiency. The core architectural components include:

5.1.1.1. Logic Resources

- 102,400 Logic Cells based on a 6-input Look-Up Table (LUT6) architecture, supporting efficient and flexible logic implementation.
- Dual LUT and Flip-Flop Combinations within Configurable Logic Blocks (CLBs), enabling high parallelism and compact design mapping.
- CLBs include LUTs, multiplexers, and D flip-flops designed for implementing combinational and sequential logic functions.

5.1.1.2. Memory Resources

- 4.9 Mb of Block RAM (BRAM) distributed across the chip, configurable as 18Kb or 36Kb blocks for flexible usage in buffers, FIFOs, or dual-port memories.
- Distributed RAM allows LUTs to function as small, low-latency RAM blocks for fast data access.
- SRL32 Shift Registers (an upgrade from SRL16 in Spartan-6) enable pipelining and temporal data storage directly inside the logic fabric.

5.1.1.3. DSP and Arithmetic Support

- 240 DSP48A1 Slices, each supporting 25x18 multiply, addition, and accumulation functions for efficient digital signal processing tasks.
- Optimized for Multiply-Accumulate (MAC) operations used in filters, FFTs, and image/video processing.
- DSP slices are pipelined to operate at high clock frequencies, making them ideal for real-time analytics and embedded AI/ML workloads.

5.1.1.4. Clock Management and Timing

- 6 Clock Management Tiles (CMTs), each containing 1 Mixed-Mode Clock Manager (MMCM) and 1 Phase-Locked Loop (PLL), enabling flexible clock synthesis, phase shifting, and jitter filtering.
- Global and regional clock networks for efficient distribution across the fabric.
- Clock Enable (CE) and Synchronous Set/Reset signals for low-power design and controlled timing.

5.1.1.5. Input/Output (I/O) Interfaces

- 400 User I/O pins, supporting a wide variety of single-ended and differential I/O standards, including LVCMOS, LVTTL, HSTL, SSTL, and LVDS.
- Supports high-speed DDR3/DDR3L SDRAM interfaces with dedicated I/O banking and timing calibration logic. Compatible with PCIe Gen2 x1 via soft IP core using standard logic fabric.
- Gigabit Transceivers (GTPs) available on select Spartan-7 variants (not present in XC7S100), but parallel interfaces can still achieve high bandwidth via FMC and PMOD.
- Support for communication protocols such as SPI, I2C, UART, CAN, and Ethernet (via soft IP or hard MAC/PHY interfaces).

5.1.2. Power Efficiency and Low-Power Design

- Built on **28nm low-power process technology**, reducing both static and dynamic power compared to Spartan-6.
- Includes advanced **power gating and clock gating techniques** to shut down unused logic blocks dynamically.
- **Dynamic Power Scaling (DPS)** capability adjusts internal voltage and logic activity based on real-time workload.

5.1.3. Packaging and Form Factor

- The **XC7S100** is available in the **FGGA676 BGA package**, providing high pin count and thermal performance for dense applications.
- Offers up to **400 user I/Os**, making it suitable for I/O-heavy applications like industrial automation, embedded vision, and communications.
- Integrated support for **FMC, PMOD, HDMI, and MIPI interfaces** on development boards like the SP701 enables fast prototyping and deployment.

Fig 5.1:- b) Xilinx Spartan-7 Processor

The Above Fig(5.1b) shows The **AMD Spartan-7 FPGA (XC7S100-FGG676-2)** integrates a highly efficient, low-power architecture ideal for cost-sensitive and performance-demanding applications. Built on a 28nm process technology, it features the largest device in the Spartan-7 family, offering **102,400 logic cells**, **240 DSP slices**, **4.9 Mb of block RAM**, and **400 programmable I/O pins**, all within a compact **FGGA676 package**. This device supports a maximum system clock speed exceeding **450 MHz**, making it suitable for high-speed digital processing. Unlike traditional microprocessors, the Spartan-7 doesn't include a fixed CPU; instead, users can implement soft processors such as the **Xilinx MicroBlaze** within the FPGA fabric, providing a flexible and customizable embedded processing platform. Additionally, it includes an integrated **XADC (12-bit, 1 MSPS ADC)** for monitoring analog signals like voltage and temperature, and robust security features like bitstream encryption (AES) for IP protection. With its balance of performance, power efficiency, and cost, the XC7S100 device is ideal for applications like embedded vision, industrial control, motor control, and secure communications.

5.2 Software – Xilinx ISE

Fig 5.2:- Xilinx VIVADO 2021.2 Version.

The Above Fig(4.2) Vivado Design Suite 2021.2 is a comprehensive software development environment from AMD (formerly Xilinx) for designing and implementing systems on their Field-Programmable Gate Arrays (FPGAs) and Adaptive SoCs. This version offers a wide range of features that support the entire design flow, from specification and synthesis to implementation, verification, and bitstream generation. Key enhancements in the 2021.2 release include improvements in design performance for Versal devices, with optimized clocking and placement and routing across multiple silicon dies. It also introduced features aimed at improving user experience, such as enhanced visualization tools for Dynamic Function eXchange (DFX) and address path debugging in IP Integrator. Furthermore, Vivado 2021.2 brought updates to IP cores, supporting high-speed Ethernet standards and multimedia interfaces like DisplayPort 2.1 and HDMI 2.1 on specific devices. For embedded systems development, this version included updates to the underlying software and tools, ensuring compatibility and enhanced functionality for designing systems incorporating processors and programmable logic. The software supports various operating systems, including Windows and Linux, and requires specific hardware configurations in terms of processing power, memory, and disk space to run efficiently.

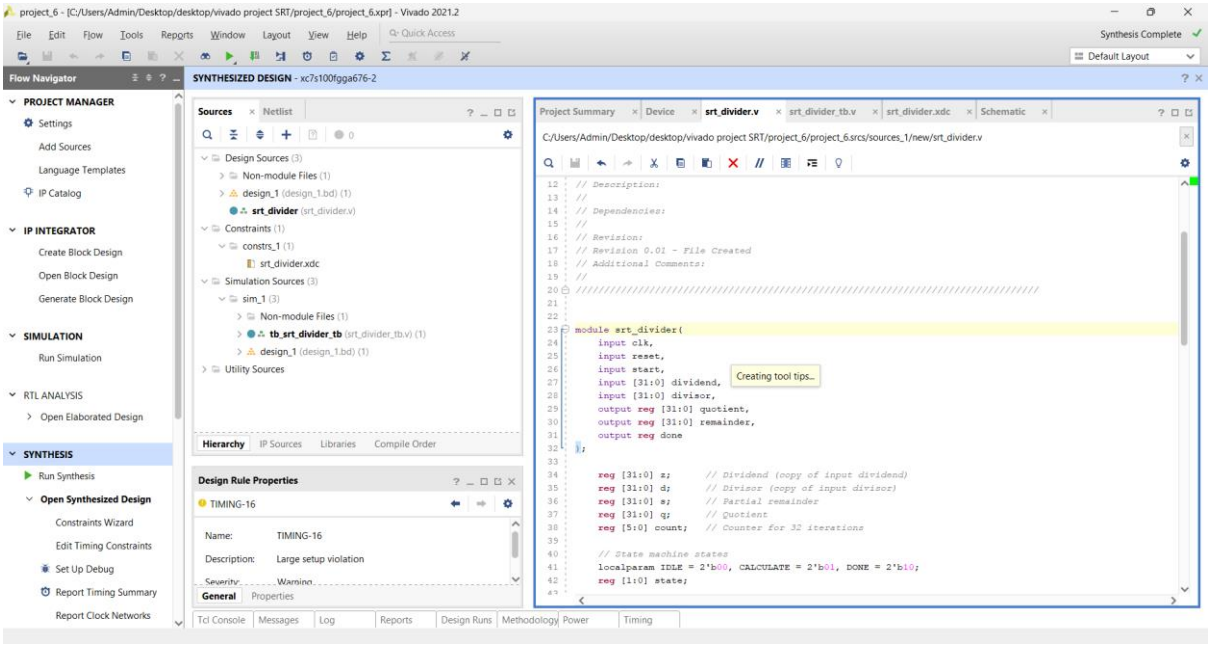

Fig 5.3:- Xilinx Vivado Worspace

The Above Fig(4.3) shows the Xilinx Vivado workspace shown in the image provides an integrated environment for FPGA design, synthesis, and verification. The Project Navigator on the left organizes the project into different sections, including Design Sources, Constraints, Simulation Sources, and Synthesis. The Hierarchy panel helps in managing the module dependencies. In the code editor on the right, the srt_divider.v file is open, displaying the Verilog code for the SRT divider. The highlighted section shows module declarations and input-output definitions. The Design Rule Properties panel at the bottom indicates a TIMING-16 violation, suggesting a large setup time issue. The workspace also provides access to Run Synthesis, Open Elaborated Design, and RTL Analysis, enabling designers to analyze and optimize their FPGA implementations efficiently.

Chapter 6

Result and Discussion

6.1 Simulation and Schematic

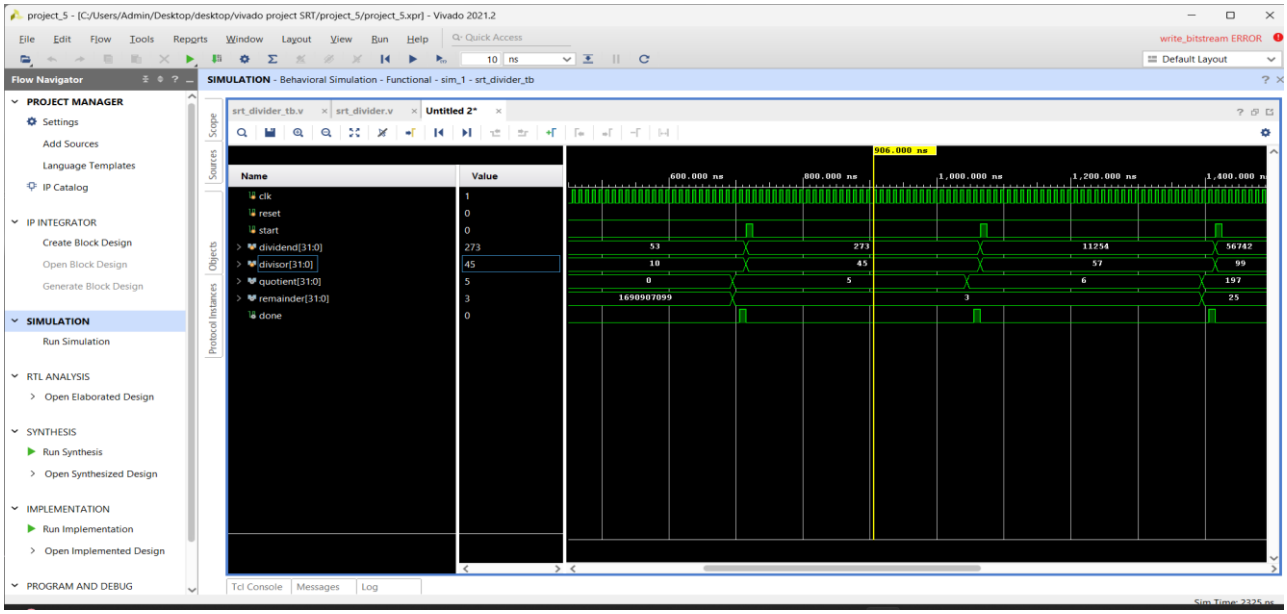

Fig 6.1:- a)Simulation Workspace

The Above Fig(5.1a) shows the simulation results for the SRT Radix-2 Divider indicate successful functional verification of the division operation. The waveform analysis shows key signals, including the clock (clk), reset, start, dividend, divisor, quotient, remainder, and done signal. Initially, the reset is held low, allowing the design to operate normally. Once the start signal is asserted, the division process begins. The dividend and divisor values are loaded, and the computation progresses over several clock cycles.

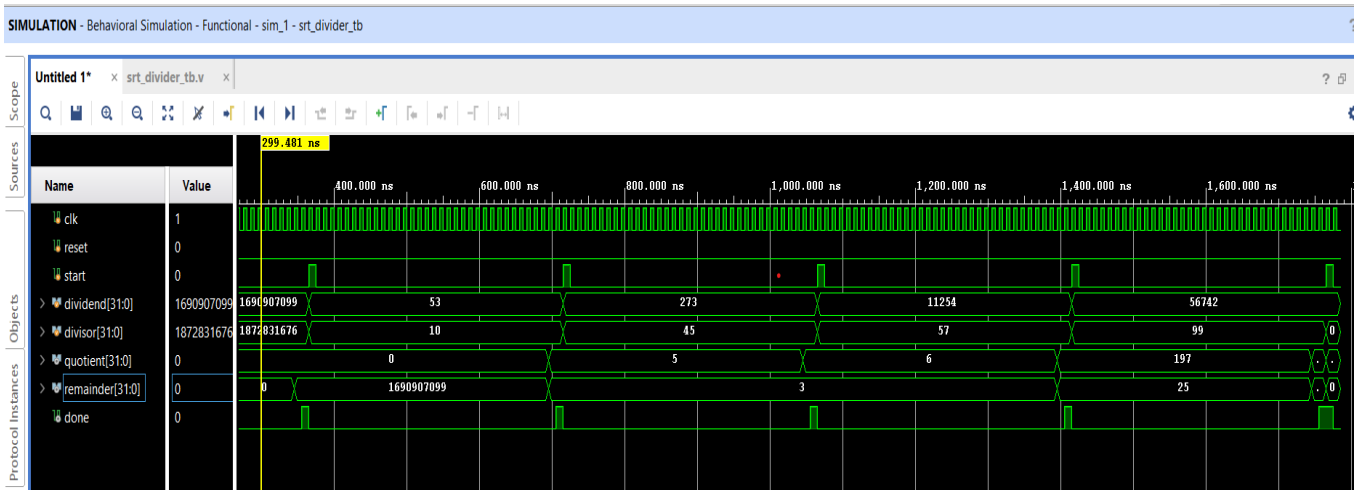

Fig 6.1:- b)Simulation Results

The Above Fig(5.1b) shows the quotient and remainder values are updated dynamically, showing intermediate results at different time instances. As for 53 divided by 10 we got the quotient as 5 and Remainder as 3. As observed, after a specific number of cycles, the quotient and remainder reach their final values, indicating the completion of the division. The done signal is asserted, signaling the end of the operation. The waveform confirms correct functionality, demonstrating that the design correctly computes division results over multiple test cases.

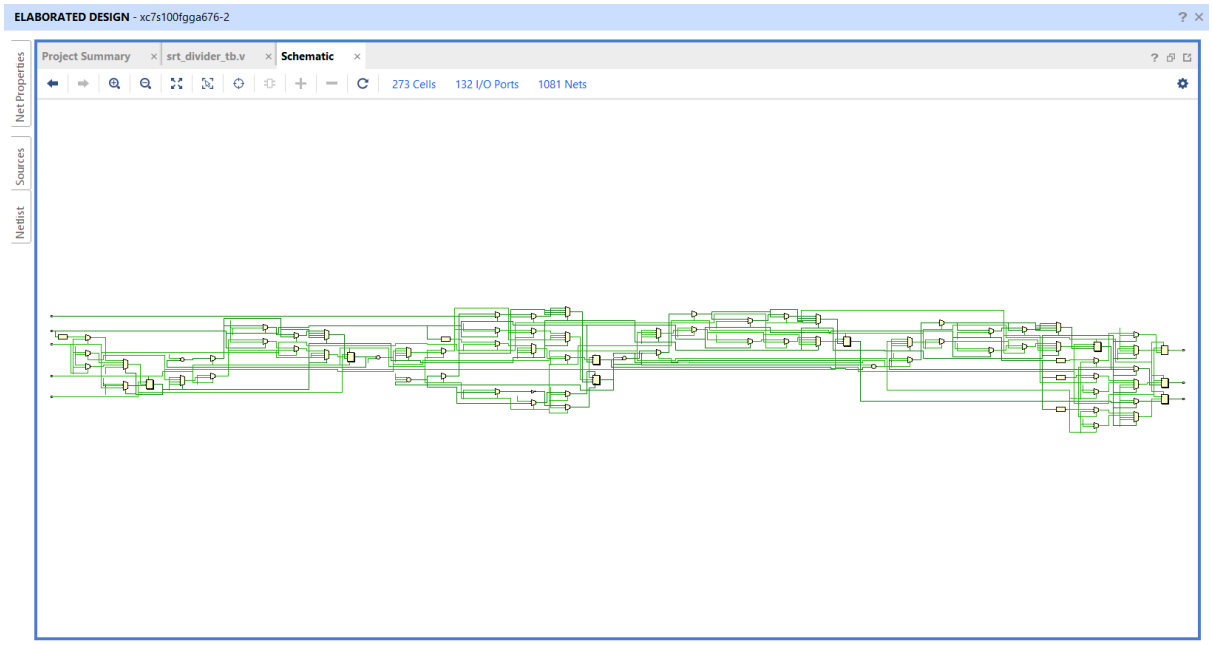

Fig 6.1:- c)Schematic of SRT Radix-2 Divider.

The Above Fig(5.1c) shows the schematic representation of the SRT Radix-2 Divider shows the elaborated design post-synthesis, including logic cells, I/O ports, and interconnections. The design consists of 273 cells, 132 I/O ports, and 1081 nets, illustrating the complexity of the implemented circuit. The schematic view highlights the structure of arithmetic logic, control logic, and data path, which are crucial for performing iterative division using the digit recurrence method.

This diagram helps in understanding:

- Component Connectivity: How different modules interact within the FPGA.
- Signal Flow: The direction and dependencies of data paths.
- Optimization Scope: Identifying redundant logic or inefficient routing.

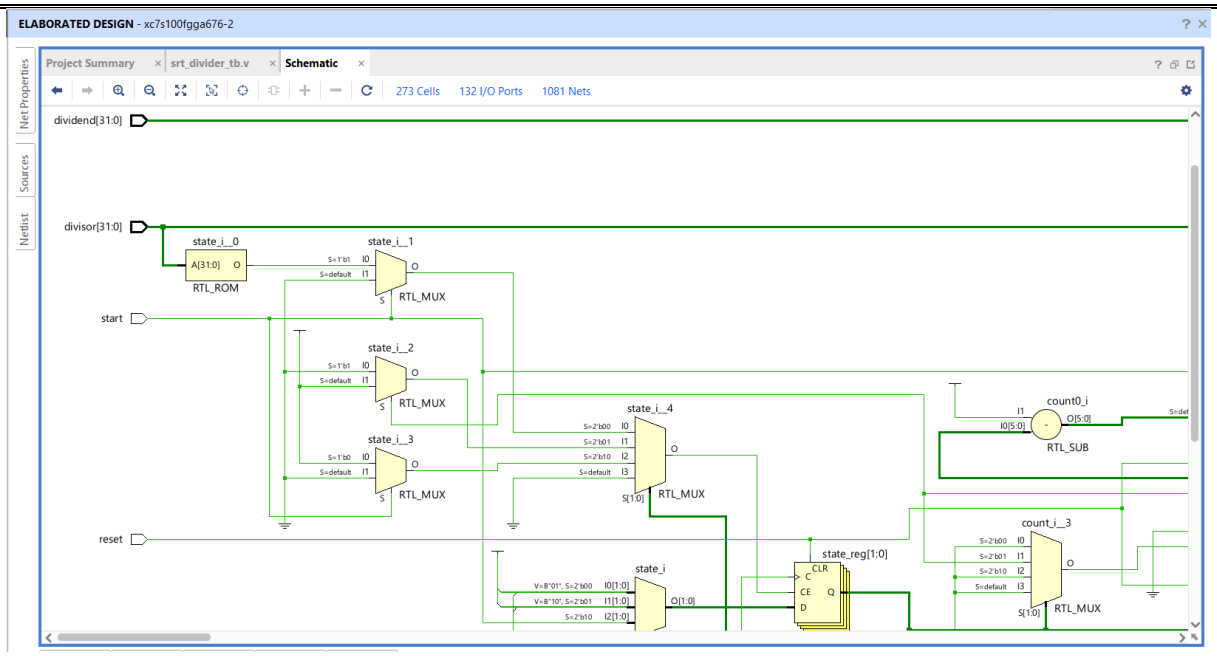

Fig 6.1:- d)Input Side view of Schematic.

The Above Fig(5.1d) shows the schematic representation of the SRT Radix-2 Divider provides a detailed RTL (Register Transfer Level) view of the circuit, illustrating how different components interact during the division process. The schematic consists of multiplexers (RTL_MUX), subtractors (RTL_SUB), registers (state_reg), and a ROM block (RTL_ROM) for state control.

Key inputs include the dividend (31:0), divisor (31:0), start, and reset signals. The control logic utilizes multiple state variables (state_i_0 to state_i_4), which determine the sequential operations of the algorithm. The RTL_ROM block stores predefined values essential for state transitions, while multiplexers (MUX) dynamically select appropriate signals for computation.

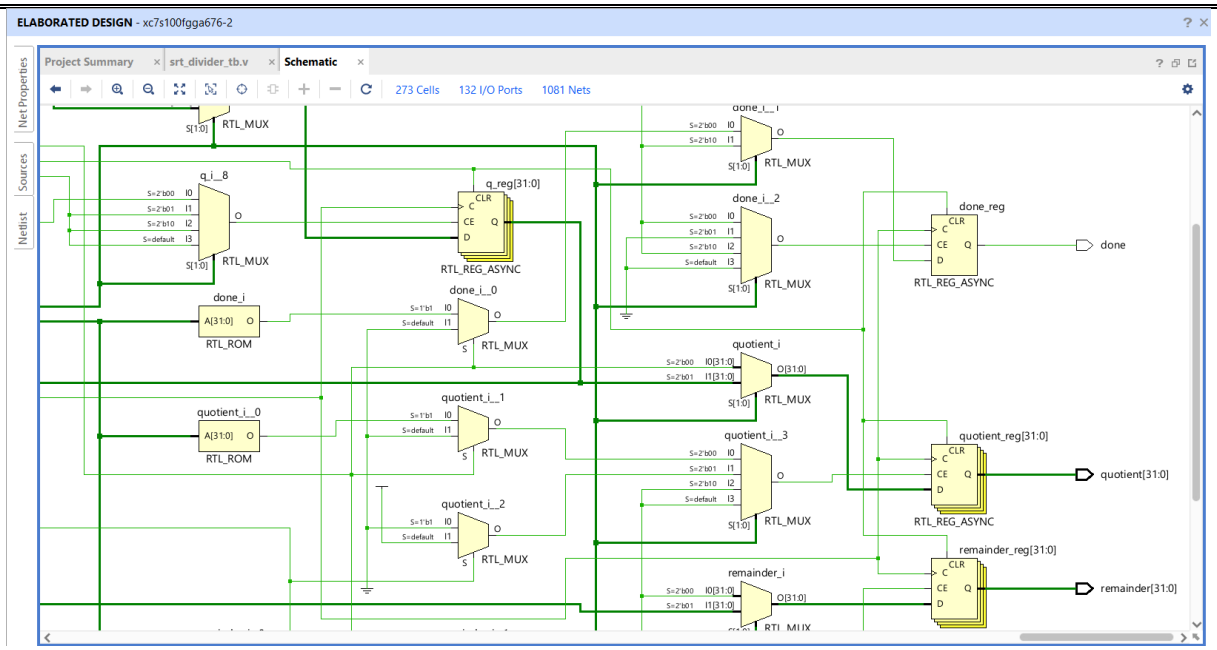

Fig 6.1:- e)Output Side view of schematic.

6.2 Synthesize and Devise Utilization

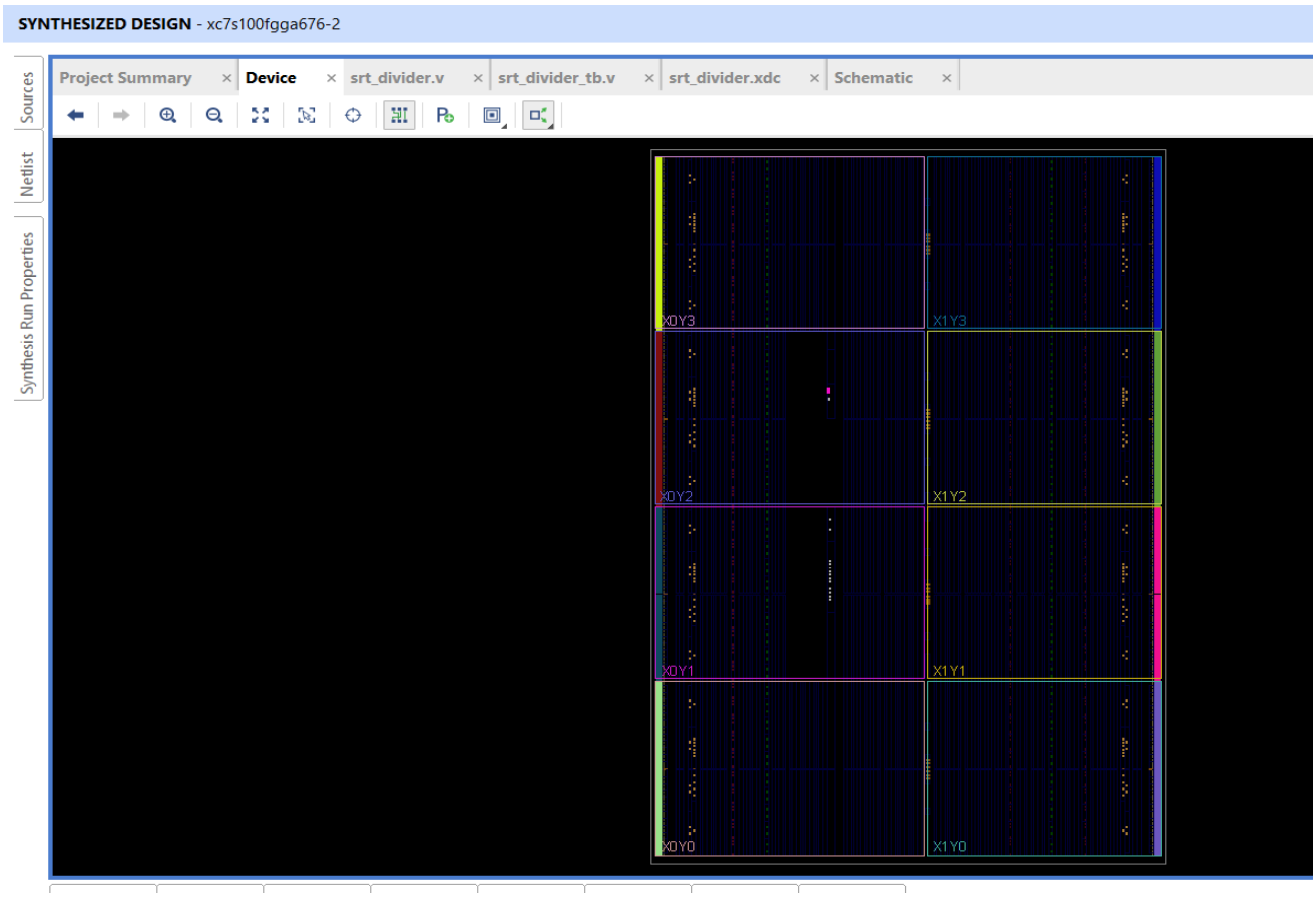

Fig 6.2:- Synthesized Design

Synthesize report from the Xilinx.

```
# Vivado v2021.2 (64-bit)
# SW Build 3367213 on Tue Oct 19 02:48:09 MDT 2021
# IP Build 3369179 on Thu Oct 21 08:25:16 MDT 2021
# Start of session at: Tue Apr  1 13:27:46 2025
# Process ID: 8220
# Current directory: C:/Users/Admin/Desktop/desktop/vivado project SRT/Final SRT/Final SRT.runs/synth_1
# Command line: vivado.exe -log srt.vds -product Vivado -mode batch -messageDb vivado.pb -notrace -source srt.tcl
# Log file: C:/Users/Admin/Desktop/desktop/vivado project SRT/Final SRT/Final SRT.runs/synth_1/srt.vds
# Journal file: C:/Users/Admin/Desktop/desktop/vivado project SRT/Final SRT/Final SRT.runs/synth_1\vivado.jou
# Running On: DESKTOP-5FLN82U, OS: Windows, CPU Frequency: 2112 MHz, CPU Physical cores: 4, Host memory: 8430 MB
#-----------------------------------------------------------
source srt.tcl -notrace
create_project: Time (s): cpu = 00:00:03 ; elapsed = 00:00:06 . Memory (MB): peak = 1384.867 ; gain = 0.000
Command: synth_design -top srt -part xc7s100fgga676-2
Starting synth_design
Attempting to get a license for feature 'Synthesis' and/or device 'xc7s100'
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7s100'
INFO: [Device 21-403] Loading part xc7s100fgga676-2
INFO: [Synth 8-7079] Multithreading enabled for synth_design using a maximum of 2 processes.
INFO: [Synth 8-7078] Launching helper process for spawning children vivado processes
INFO: [Synth 8-7075] Helper process launched with PID 8008
---------------------------------------------------------------------------------
Starting RTL Elaboration : Time (s): cpu = 00:00:05 ; elapsed = 00:00:08 . Memory (MB): peak = 1384.867 ; gain = 0.000
Finished RTL Elaboration : Time (s): cpu = 00:00:06 ; elapsed = 00:00:10 . Memory (MB): peak = 1384.867 ; gain = 0.000
Start Handling Custom Attributes
---------------------------------------------------------------------------------
```

Finished Handling Custom Attributes : Time (s): cpu = 00:00:07 ; elapsed = 00:00:11 . Memory (MB): peak = 1384.867 ; gain = 0.000

Finished RTL Optimization Phase 1 : Time (s): cpu = 00:00:07 ; elapsed = 00:00:11 . Memory (MB): peak = 1384.867 ; gain = 0.000

Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.008 . Memory (MB): peak = 1384.867 ; gain = 0.000

Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 1409.703 ; gain = 0.000

INFO: [Project 1-111] Unisim Transformation Summary:

No Unisim elements were transformed.

Constraint Validation Runtime : Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.965 . Memory (MB): peak = 1409.703 ; gain = 0.000

Finished Constraint Validation : Time (s): cpu = 00:00:14 ; elapsed = 00:00:26 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Loading Part and Timing Information

Loading part: xc7s100fgga676-2

Finished Loading Part and Timing Information : Time (s): cpu = 00:00:14 ; elapsed = 00:00:26 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Applying 'set_property' XDC Constraints

Finished applying 'set_property' XDC Constraints : Time (s): cpu = 00:00:14 ; elapsed = 00:00:26 . Memory (MB): peak = 1409.703 ; gain = 24.836

INFO: [Synth 8-802] inferred FSM for state register 'state_reg' in module 'srt'

State	New Encoding	Previous Encoding
IDLE	00	00
CALCULATE	01	01
DONE	10	10
iSTATE	11	11

INFO: [Synth 8-3354] encoded FSM with state register 'state_reg' using encoding 'sequential' in module 'srt'

Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:15 ; elapsed = 00:00:27 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start RTL Component Statistics

Detailed RTL Component Info :

Adders :

3 Input 32 Bit Adders := 1

2 Input 6 Bit Adders := 1

Registers :

32 Bit Registers := 6

6 Bit Registers := 1

1 Bit Registers := 1

Muxes :

2 Input 32 Bit Muxes := 10

4 Input 32 Bit Muxes := 5

2 Input 6 Bit Muxes := 1

4 Input 6 Bit Muxes := 1

3 Input 2 Bit Muxes := 1

2 Input 2 Bit Muxes := 1

2 Input 1 Bit Muxes := 1

4 Input 1 Bit Muxes := 7

Finished RTL Component Statistics

Start Part Resource Summary

Part Resources:
DSPs: 160 (col length:80)
BRAMs: 240 (col length: RAMB18 80 RAMB36 40)

Finished Part Resource Summary

Start Cross Boundary and Area Optimization
Finished Cross Boundary and Area Optimization : Time (s): cpu = 00:00:21 ; elapsed = 00:00:36 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Applying XDC Timing Constraints

Finished Applying XDC Timing Constraints : Time (s): cpu = 00:00:28 ; elapsed = 00:00:43 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Timing Optimization

Finished Timing Optimization : Time (s): cpu = 00:00:28 ; elapsed = 00:00:43 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Technology Mapping

Finished Technology Mapping : Time (s): cpu = 00:00:28 ; elapsed = 00:00:43 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start IO Insertion

Start Flattening Before IO Insertion

Finished Flattening Before IO Insertion

Start Final Netlist Cleanup

Finished Final Netlist Cleanup

Finished IO Insertion : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Renaming Generated Instances

Finished Renaming Generated Instances : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Rebuilding User Hierarchy

Finished Rebuilding User Hierarchy : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Renaming Generated Ports

Finished Renaming Generated Ports : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Handling Custom Attributes

Finished Handling Custom Attributes : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Renaming Generated Nets

Finished Renaming Generated Nets : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

Start Writing Synthesis Report

Report BlackBoxes:

```
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+
```

Report Cell Usage:

```
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
```

```
|2      |CARRY4 |    12|
|3      |LUT2   |    65|
|4      |LUT3   |    37|
|5      |LUT4   |    45|
|6      |LUT5   |     5|
|7      |LUT6   |   109|
|8      |FDCE   |   135|
|9      |FDPE   |     2|
|10     |FDRE   |    64|
|11     |IBUF   |    67|
|12     |OBUF   |    65|
+------+-------+------+
```

Finished Writing Synthesis Report : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

Synthesis finished with 0 errors, 0 critical warnings and 63 warnings.

Synthesis Optimization Runtime : Time (s): cpu = 00:00:24 ; elapsed = 00:00:42 . Memory (MB): peak = 1409.703 ; gain = 0.000

Synthesis Optimization Complete : Time (s): cpu = 00:00:33 ; elapsed = 00:00:49 . Memory (MB): peak = 1409.703 ; gain = 24.836

INFO: [Project 1-571] Translating synthesized netlist

Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.006 . Memory (MB): peak = 1410.227 ; gain = 0.000

INFO: [Netlist 29-17] Analyzing 12 Unisim elements for replacement

INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds

INFO: [Project 1-570] Preparing netlist for logic optimization

INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).

Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 1418.910 ; gain = 0.000

INFO: [Project 1-111] Unisim Transformation Summary:

No Unisim elements were transformed.

Synth Design complete, checksum: d65dd186

INFO: [Common 17-83] Releasing license: Synthesis

19 Infos, 72 Warnings, 9 Critical Warnings and 0 Errors encountered.

synth_design completed successfully

synth_design: Time (s): cpu = 00:00:36 ; elapsed = 00:01:00 . Memory (MB): peak = 1418.910 ; gain = 34.043

INFO: [runtcl-6] Synthesis results are not added to the cache due to CRITICAL_WARNING

INFO: [Common 17-1381] The checkpoint 'C:/Users/Admin/Desktop/desktop/vivado project SRT/Final SRT/Final SRT.runs/synth_1/srt.dcp' has been generated.

INFO: [runtcl-4] Executing : report_utilization -file srt_utilization_synth.rpt -pb srt_utilization_synth.pb

INFO: [Common 17-206] Exiting Vivado at Tue Apr 1 13:29:18 2025...

6.3 Power Analysis for SRT Radix-2 Divider

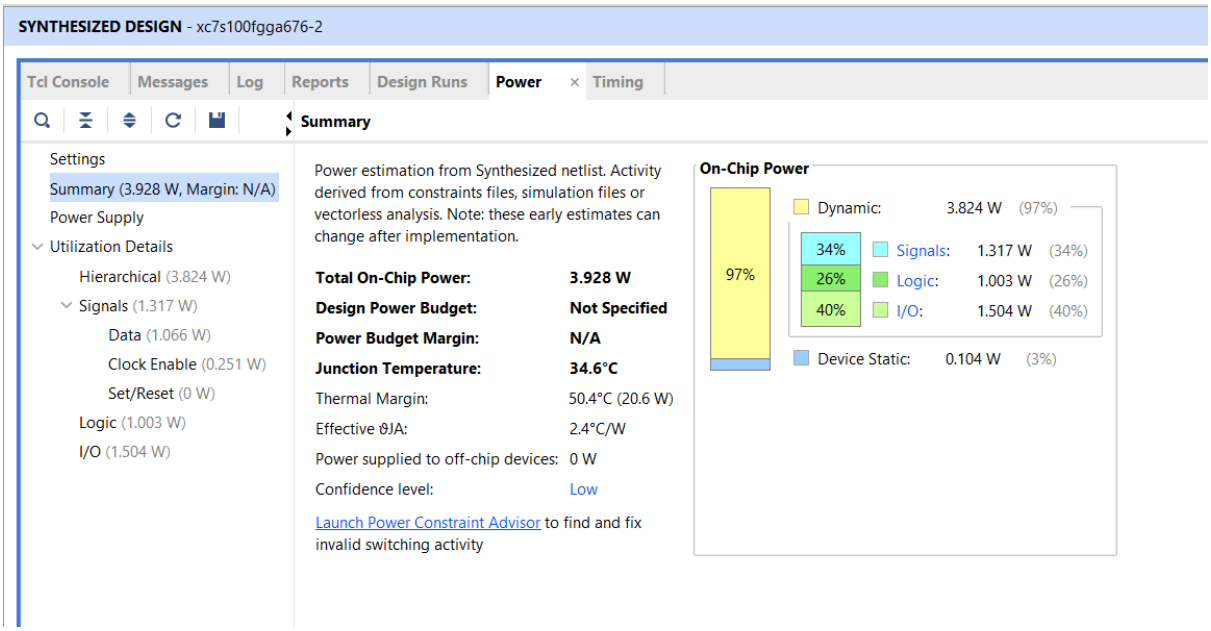

Fig 6.2:- a)Power Analysis Before Timing Constraints

The Above Fig(5.2a) shows the power estimation for the SRT Radix-2 Divider was obtained from the synthesized netlist, considering activity derived from constraints files and vectorless analysis. The total on-chip power consumption is measured as 3.928 W, with dynamic power accounting for 97% (3.824 W) and device static power contributing 3% (0.104 W). The power distribution across different components is as follows:

- Signals: 1.317 W (34%)
    - Data: 1.066 W
    - Clock Enable: 0.251 W
    - Set/Reset: 0 W
- Logic: 1.003 W (26%)
- I/O: 1.504 W (40%)

The junction temperature is recorded at 34.6°C, and the thermal margin is 50.4°C (20.6 W), with an effective θJA of 2.4°C/W. The power supplied to off-chip devices is 0 W, and the confidence level in the power analysis

is marked as low, indicating a need for further validation post-implementation. These power metrics provide insights into the design's energy efficiency and can be optimized further for better performance.

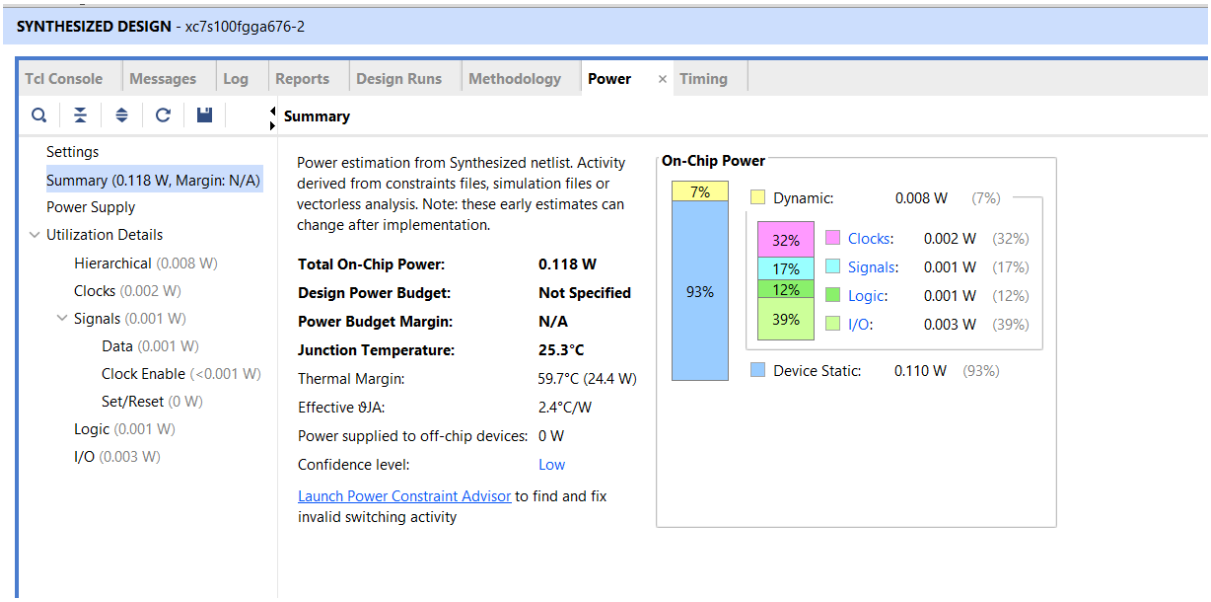

Fig 6.2:- b)Power Analysis After Timing Constraints

The Above Fig(5.2b) shows the power estimation is based on the synthesized netlist in Vivado, assuming a vectorless analysis. This power estimate may change after implementation due to more accurate switching activity calculations.

Power Summary:

- Total On-Chip Power: 0.118 W
- Device Static Power: 0.110 W (93%)
- Dynamic Power: 0.008 W (7%)
- Junction Temperature: 25.3°C
- Thermal Margin: 59.7°C (24.4 W)
- Effective θJA (Junction-to-Ambient Thermal Resistance): 2.4°C/W
- Power Supplied to Off-Chip Devices: 0 W
- Confidence Level: Low (due to vectorless analysis)

Power Breakdown:

Category	Power (W)	Percentage
Clocks	0.002 W	32%
Signals	0.001 W	17%

Category	Power (W)	Percentage
Logic	0.001 W	12%
I/O	0.003 W	39%

Observations & Recommendations

Low Dynamic Power Consumption (7%):

- Most power (93%) is consumed by device static power, which is inherent to the FPGA.
- Optimization in logic switching and clock gating may reduce dynamic power.

I/O Power is Highest (39%):

- High I/O activity is contributing significantly to dynamic power.
- Consider reducing I/O toggling frequency or optimizing bus widths.

Clock Power (32%):

- Significant power is consumed by clock distribution.
- Implement clock gating techniques to minimize unnecessary toggling.

Thermal Considerations:

- Junction temperature is well below critical limits (25.3°C).
- The thermal margin is 59.7°C, indicating a safe operating condition

6.3 Device Area Utilization

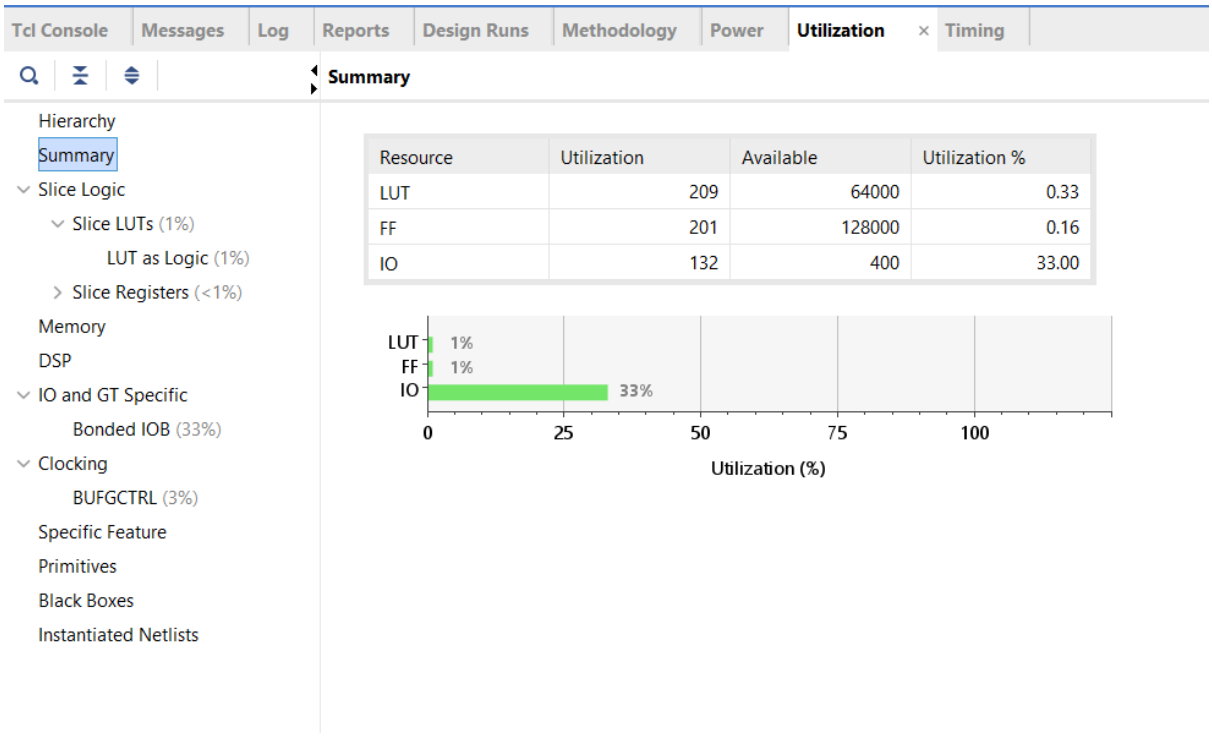

Fig 6.3:- Device summary

The Above Fig(5.3) shows the Device utilization highlights the compact and efficient nature of the design. The area report indicates that the divider consumes 209 LUTs, which accounts for only 0.33% of the available 64,000 LUTs, showing that the design uses minimal combinational logic resources. It also utilizes 201 flip-flops, representing 0.16% of the total 128,000 flip-flops available on the device, indicating that the sequential logic requirement is similarly low. However, the IO utilization is relatively higher, occupying 132 out of 400 available IO pins, which corresponds to 33% utilization. This is expected for designs that have wide data buses for inputs like dividend, divisor, and outputs such as quotient and remainder. Overall, the low logic utilization combined with moderate IO usage makes this design well-suited for resource-constrained devices such as the Spartan-6, ensuring that the divider fits comfortably while leaving significant headroom for additional functionality if needed.

```
Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------
------------------------
| Tool Version : Vivado v.2021.2 (win64) Build 3367213 Tue Oct 19
02:48:09 MDT 2021
| Date         : Tue Apr  1 13:29:16 2025
| Host         : DESKTOP-5FLN82U running 64-bit major release  (build
9200)
| Command      : report_utilization -file srt_utilization_synth.rpt -pb
srt_utilization_synth.pb
| Design       : srt
| Device       : xc7s100fgga676-2
| Speed File   : -2
| Design State : Synthesized
------------------------------------------------------------------------
------------------------
```

Utilization Design Information

Table of Contents

---

8. Black Boxes

9. Instantiated Netlists

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	209	0	0	64000	0.33
LUT as Logic	209	0	0	64000	0.33
LUT as Memory	0	0	0	17600	0.00
Slice Registers	201	0	0	128000	0.16
Register as Flip Flop	201	0	0	128000	0.16
Register as Latch	0	0	0	128000	0.00
F7 Muxes	0	0	0	32000	0.00
F8 Muxes	0	0	0	16000	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
2	Yes	-	Set
135	Yes	-	Reset
0	Yes	Set	-
64	Yes	Reset	-

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	120	0.00
RAMB36/FIFO*	0	0	0	120	0.00
RAMB18	0	0	0	240	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	0	0	0	160	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	132	0	0	400	33.00
Bonded IPADs	0	0	0	2	0.00
PHY_CONTROL	0	0	0	8	0.00
PHASER_REF	0	0	0	8	0.00
OUT_FIFO	0	0	0	32	0.00
IN_FIFO	0	0	0	32	0.00
IDELAYCTRL	0	0	0	8	0.00
IBUFDS	0	0	0	384	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	32	0.00

PHASER_IN/PHASER_IN_PHY	0	0	0	32	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	400	0.00
ILOGIC	0	0	0	400	0.00
OLOGIC	0	0	0	400	0.00

5. Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	1	0	0	32	3.13
BUFIO	0	0	0	32	0.00
MMCME2_ADV	0	0	0	8	0.00
PLLE2_ADV	0	0	0	8	0.00
BUFMRCE	0	0	0	16	0.00
BUFHCE	0	0	0	96	0.00
BUFR	0	0	0	32	0.00

6. Specific Feature

Site Type	Used	Fixed	Prohibited	Available	Util%
BSCANE2	0	0	0	4	0.00
CAPTUREE2	0	0	0	1	0.00
DNA_PORT	0	0	0	1	0.00
EFUSE_USR	0	0	0	1	0.00
FRAME_ECCE2	0	0	0	1	0.00
ICAPE2	0	0	0	2	0.00
STARTUPE2	0	0	0	1	0.00
XADC	0	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDCE	135	Flop & Latch
LUT6	109	LUT

IBUF	67	IO
OBUF	65	IO
LUT2	65	LUT
FDRE	64	Flop & Latch
LUT4	45	LUT
LUT3	37	LUT
CARRY4	12	CarryLogic
LUT5	5	LUT
FDPE	2	Flop & Latch
BUFG	1	Clock

6.4 Timing Summary

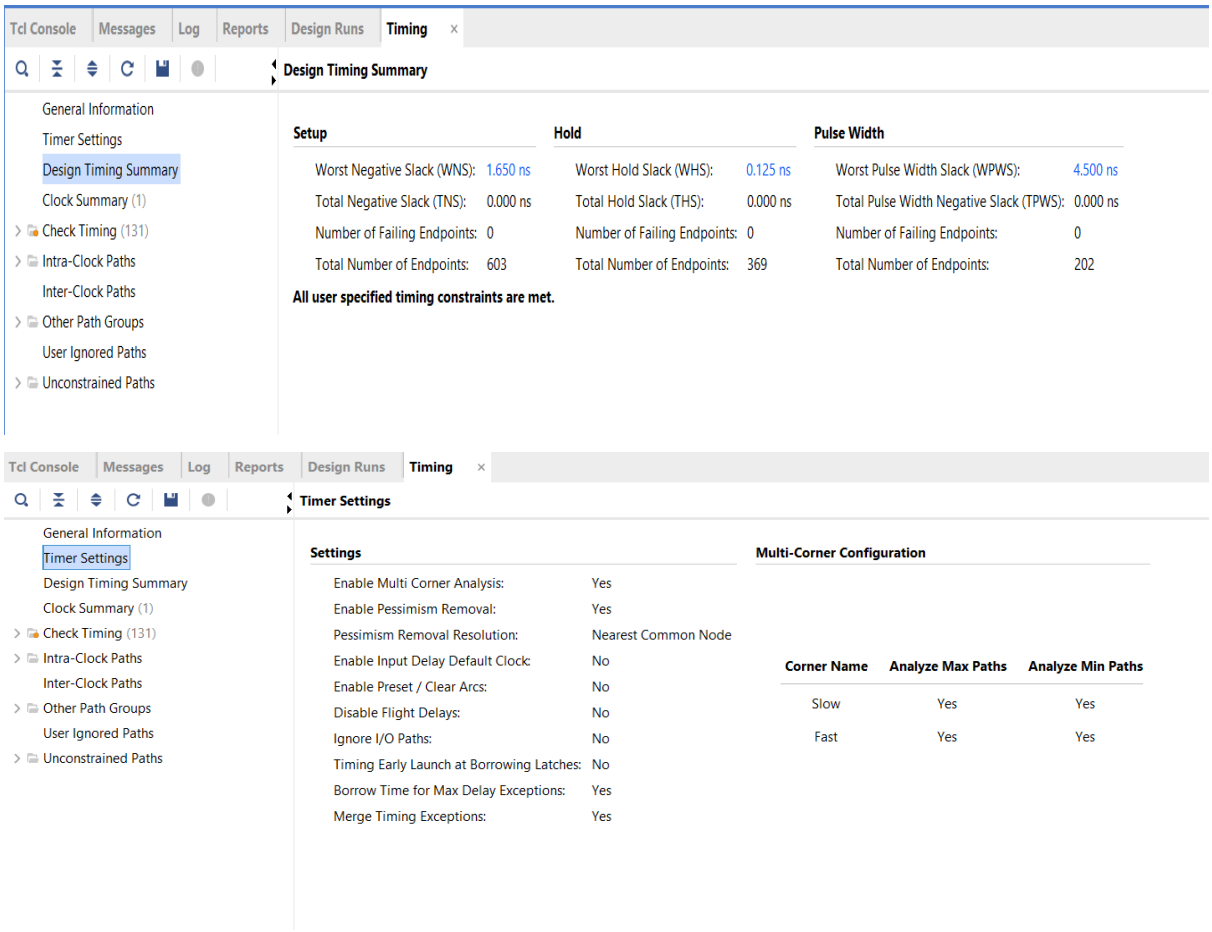

Fig 6.4:- Timing constraint

The Above Fig(5.4) shows the Design Timing Summary report shown in the image indicates that all user-specified timing constraints have been met for the design. The key timing parameters are as follows:

- Setup Analysis: The Worst Negative Slack (WNS) is 1.650 ns, indicating the minimum slack available for setup timing. The Total Negative Slack (TNS) is 0.000 ns, meaning no setup timing violations exist. There are 0 failing endpoints out of 603 total endpoints, confirming that all setup paths meet timing.

- Hold Analysis: The Worst Hold Slack (WHS) is 0.125 ns, and the Total Hold Slack (THS) is 0.000 ns, meaning there are no hold violations. There are 0 failing endpoints out of 369 total endpoints, indicating hold timing is also clean.
- Pulse Width Analysis: The Worst Pulse Width Slack (WPWS) is 4.500 ns, with a Total Pulse Width Negative Slack (TPWS) of 0.000 ns, meaning pulse width timing requirements are satisfied. There are 0 failing endpoints out of 202 total endpoints.

CONCLUSION

The SRT Radix-2 Divider designed and implemented in this project provides an efficient and hardware-friendly approach for performing division using the digit recurrence method. The SRT (Sweeney, Robertson, and Tocher) division algorithm improves upon traditional division methods such as restoring and non-restoring division by incorporating redundant quotient digit selection. This enables carry-free arithmetic, reduces computational complexity, and enhances performance, making it well-suited for FPGA-based and ASIC implementations.

The power analysis for the SRT Radix-2 Divider implemented in Xilinx Vivado estimates a total on-chip power consumption of 0.118 W, with device static power accounting for 93% (0.110 W) and dynamic power contributing 7% (0.008 W). The power estimation is based on a synthesized netlist using vectorless analysis, meaning the actual power consumption may vary after implementation. The junction temperature is recorded at 25.3°C, with a thermal margin of 59.7°C (24.4 W) and an effective junction-to-ambient thermal resistance (θJA) of 2.4°C/W, ensuring safe thermal operation. The power breakdown shows that I/O power contributes the most at 39% (0.003 W), followed by clock power at 32% (0.002 W), signals at 17% (0.001 W), and logic at 12% (0.001 W). Since dynamic power is relatively low, most of the power consumption is due to static device power, which is inherent to the FPGA. To optimize power usage, techniques like clock gating, reducing I/O toggling frequency, and optimizing bus widths should be considered.

The SRT Radix-2 Divider implemented on the Spartan-6 FPGA efficiently utilizes hardware resources. It occupies only 0.33% of LUTs and 0.16% of flip-flops, demonstrating minimal logic usage. The design uses 33% of available IOs, reflecting the wide operand and result buses. Overall, the divider balances performance and area efficiently, making it suitable for real-time applications in resource-limited FPGA systems.

The implementation on FPGA hardware validates that the design is scalable, efficient, and reliable, with significant improvements in latency and area optimization compared to traditional division approaches. The SRT Radix-2 Divider proves to be highly effective in real-time arithmetic operations, such as floating-point arithmetic, digital signal processing (DSP), and processor-based applications. The project successfully demonstrates that the SRT Radix-2 division method is an optimal choice for hardware-efficient division, providing a foundation for further enhancements like pipeline optimization, higher-radix implementations, and parallel processing techniques for future high-speed computing applications.

REFERENCE

[1] "Review of Basic Classes of Dividers Based on Division Algorithm" Udayan S. Patankar and Ants Koel.Thomson johann Department of Electronics,Tallinm University of Technology, FEB 9,2021.

[2] "Division algorithms – From Past to Present Chance to Improve Area Time and Complexity for Digital Applications" Udayan S. Patankar1, Miguel E. Flores2, Ants Koel3, 2020 IEEE Latin America Electron Devices Conference (LAEDC) San José, Costa Rica, February 25-28, 2020.

[3] "SRT Radix-2 Dividers with (5,4) Redundant Representation of Partial Remainder" Alexandru Amaricai Oana Boncalo 978-1-4799-1647-4/13/$31.00 ©2013 IEEE.

[4] "Radix-2 Division Algorithms with an Over-Redundant Digit Set" Jo Ebergen,and Navaneeth Jamadagni, IEEE DOI 10.1109/TC.2014.2366738, SEPTEMBER 2013.

[5] "Carry-Free Radix-2 Subtractive Division Algorithm and Implementation of the Divider" Jen Shiun Chiang, Hung-Da Chung and Min-Show Tsai Vol. 3, No. 4, pp. 249-255 (2000).

[6] "SRT Division Architectures and Implementations" David L. Harris, Stuart E Oberman, and Mark A. Horowitz Computer Systems Laboratory Stanford University Stanford, CA 94305 1063-6889/97 $10.00 0 1997 IEEE.

[7] "TheForgottenHorseman: DigitalImplementationofArithmeticDivisionand Resources to Learn and Teach Its Complexities" Dr. Peter Jamieson, Miami University, Nathaniel David Martin, Miami University American Society for Engineering Education, 2024.

[8] "FPGA Implementation Of Radix 2 Division With Over-Redundant Quotient Selection" Attij A. Ibrahem , Hamed Elsimary . Aly E. Salama ' ' Electronics Research Institute, Cairo, Egypt, Cairo University, Cairo, Egypt ICM 2003, Dec. 9-1 1, Cairo, Egypt.

[9] "Implementation of high-speed fixed-point dividers on FPGA" Nikolay Sorokin Pacific National University, Tikhookeanskaya str., 136, Khabarovsk, Russia JCS&T Vol. 6 No. 1 April 2006.

[10] "Implementation of N-Bit Divider using VHDL" Raghawendra Sharma M.Tech. Scholar, S. V. U Merut, U.P. India, March 2013

[11] "Performance Evaluation of Division Algorithms in FPGA" K. S. Mannatunga1, and M. D. R. Perera, 9th International Research Conference – KDU, Sri Lanka, 2016.

[12] "Power Aware Dividers in FPGA" Gustavo Sutter, Jean-Pierre Deschamps, Gery Biou, and Eduardo Boemo, E. Macii et al. (Eds.): PATMOS 2004, LNCS 3254, pp. 574–584, 2004. © Springer-Verlag Berlin Heidelberg 2004.

APPENDIX

1. **Spartan-7 FPGA Data Sheet: DC and Switching Characteristics by AMD.**

2. Device/Package 6slx4tqg144

Device : xc7s100fgga676

Pin	Pin Name	Memory	Bank	VCCAUX Group	Super	Logic
AC14	DONE_0	NA	0	NA	NA	CONFIG
R14	DXP_0	NA	0	NA	NA	CONFIG
M14	GNDADC_0	NA	0	NA	NA	CONFIG
M13	VCCADC_0	NA	0	NA	NA	CONFIG
P14	VREFP_0	NA	0	NA	NA	CONFIG
P13	VN_0	NA	0	NA	NA	CONFIG
D13	VCCBATT_0	NA	0	NA	NA	CONFIG
E13	TCK_0	NA	0	NA	NA	CONFIG
R13	DXN_0	NA	0	NA	NA	CONFIG
N13	VREFN_0	NA	0	NA	NA	CONFIG
N14	VP_0	NA	0	NA	NA	CONFIG
F13	CCLK_0	NA	0	NA	NA	CONFIG
AB16	M0_0	NA	0	NA	NA	CONFIG
AB15	M1_0	NA	0	NA	NA	CONFIG
AA14	INIT_B_0	NA	0	NA	NA	CONFIG
AB12	TDI_0	NA	0	NA	NA	CONFIG
AA12	TDO_0	NA	0	NA	NA	CONFIG
AA15	M2_0	NA	0	NA	NA	CONFIG
AB14	CFGBVS_0	NA	0	NA	NA	CONFIG
AC13	PROGRAM_B_0	NA	0	NA	NA	CONFIG
AA13	TMS_0	NA	0	NA	NA	CONFIG
AF24	IO_0_13	NA	13	NA	NA	HR
AB26	IO_L1P_T0_13	0	13	NA	NA	HR
AC26	IO_L1N_T0_13	0	13	NA	NA	HR
AB24	IO_L2P_T0_13	0	13	NA	NA	HR
AB25	IO_L2N_T0_13	0	13	NA	NA	HR
AD26	IO_L3P_T0_DQS_13	0	13	NA	NA	HR
AE26	IO_L3N_T0_DQS_13	0	13	NA	NA	HR
AD24	IO_L4P_T0_13	0	13	NA	NA	HR
AD25	IO_L4N_T0_13	0	13	NA	NA	HR
AC23	IO_L5P_T0_13	0	13	NA	NA	HR
AC24	IO_L5N_T0_13	0	13	NA	NA	HR
AE25	IO_L6P_T0_13	0	13	NA	NA	HR

AF25	IO_L6N_T0_VREF_13	0	13	NA	NA	HR
AD23	IO_L7P_T1_13	1	13	NA	NA	HR
AE23	IO_L7N_T1_13	1	13	NA	NA	HR
AF22	IO_L8P_T1_13	1	13	NA	NA	HR
AF23	IO_L8N_T1_13	1	13	NA	NA	HR
AA20	IO_L9P_T1_DQS_13	1	13	NA	NA	HR
AB20	IO_L9N_T1_DQS_13	1	13	NA	NA	HR
AB22	IO_L10P_T1_13	1	13	NA	NA	HR
AC22	IO_L10N_T1_13	1	13	NA	NA	HR
AB21	IO_L11P_T1_SRCC_13	1	13	NA	NA	HR
AC21	IO_L11N_T1_SRCC_13	1	13	NA	NA	HR
AD21	IO_L12P_T1_MRCC_13	1	13	NA	NA	HR
AE22	IO_L12N_T1_MRCC_13	1	13	NA	NA	HR
AA19	IO_L13P_T2_MRCC_13	2	13	NA	NA	HR
AB19	IO_L13N_T2_MRCC_13	2	13	NA	NA	HR
AE20	IO_L14P_T2_SRCC_13	2	13	NA	NA	HR
AF20	IO_L14N_T2_SRCC_13	2	13	NA	NA	HR
AE18	IO_L15P_T2_DQS_13	2	13	NA	NA	HR
AF19	IO_L15N_T2_DQS_13	2	13	NA	NA	HR
AC19	IO_L16P_T2_13	2	13	NA	NA	HR
AD19	IO_L16N_T2_13	2	13	NA	NA	HR
AD20	IO_L17P_T2_13	2	13	NA	NA	HR
AE21	IO_L17N_T2_13	2	13	NA	NA	HR
AC18	IO_L18P_T2_13	2	13	NA	NA	HR
AD18	IO_L18N_T2_13	2	13	NA	NA	HR
AB17	IO_L19P_T3_13	3	13	NA	NA	HR
AC17	IO_L19N_T3_VREF_13	3	13	NA	NA	HR
AF17	IO_L20P_T3_13	3	13	NA	NA	HR
AF18	IO_L20N_T3_13	3	13	NA	NA	HR
AC16	IO_L21P_T3_DQS_13	3	13	NA	NA	HR
AD16	IO_L21N_T3_DQS_13	3	13	NA	NA	HR
AA17	IO_L22P_T3_13	3	13	NA	NA	HR
AA18	IO_L22N_T3_13	3	13	NA	NA	HR
AD15	IO_L23P_T3_13	3	13	NA	NA	HR
AE16	IO_L23N_T3_13	3	13	NA	NA	HR
AE15	IO_L24P_T3_13	3	13	NA	NA	HR
AF15	IO_L24N_T3_13	3	13	NA	NA	HR
AE17	IO_25_13	NA	13	NA	NA	HR
M26	IO_0_14	NA	14	NA	NA	HR
N23	IO_L1P_T0_D00_MOSI_14	0	14	NA	NA	HR
N24	IO_L1N_T0_D01_DIN_14	0	14	NA	NA	HR
P23	IO_L2P_T0_D02_14	0	14	NA	NA	HR
R23	IO_L2N_T0_D03_14	0	14	NA	NA	HR
N21	IO_L3P_T0_DQS_PUDC_B_14	0	14	NA	NA	HR
N22	IO_L3N_T0_DQS_EMCCLK_14	0	14	NA	NA	HR
R21	IO_L4P_T0_D04_14	0	14	NA	NA	HR
R22	IO_L4N_T0_D05_14	0	14	NA	NA	HR
P20	IO_L5P_T0_D06_14	0	14	NA	NA	HR

P21	IO_L5N_T0_D07_14	0	14	NA	NA	HR
R20	IO_L6P_T0_FCS_B_14	0	14	NA	NA	HR
T20	IO_L6N_T0_D08_VREF_14	0	14	NA	NA	HR
N26	IO_L7P_T1_D09_14	1	14	NA	NA	HR
P26	IO_L7N_T1_D10_14	1	14	NA	NA	HR
P24	IO_L8P_T1_D11_14	1	14	NA	NA	HR
R25	IO_L8N_T1_D12_14	1	14	NA	NA	HR
P25	IO_L9P_T1_DQS_14	1	14	NA	NA	HR
R26	IO_L9N_T1_DQS_D13_14	1	14	NA	NA	HR
T25	IO_L10P_T1_D14_14	1	14	NA	NA	HR
U26	IO_L10N_T1_D15_14	1	14	NA	NA	HR
U25	IO_L11P_T1_SRCC_14	1	14	NA	NA	HR
V26	IO_L11N_T1_SRCC_14	1	14	NA	NA	HR
T23	IO_L12P_T1_MRCC_14	1	14	NA	NA	HR
T24	IO_L12N_T1_MRCC_14	1	14	NA	NA	HR
U20	IO_L13P_T2_MRCC_14	2	14	NA	NA	HR
U21	IO_L13N_T2_MRCC_14	2	14	NA	NA	HR
T22	IO_L14P_T2_SRCC_14	2	14	NA	NA	HR
U22	IO_L14N_T2_SRCC_14	2	14	NA	NA	HR
U24	IO_L15P_T2_DQS_RDWR_B_14	2	14	NA	NA	HR
V24	IO_L15N_T2_DQS_DOUT_CSO_B_14	2	14	NA	NA	HR
W20	IO_L16P_T2_CSI_B_14	2	14	NA	NA	HR
W21	IO_L16N_T2_D31_14	2	14	NA	NA	HR
V21	IO_L17P_T2_D30_14	2	14	NA	NA	HR
V22	IO_L17N_T2_D29_14	2	14	NA	NA	HR
V23	IO_L18P_T2_D28_14	2	14	NA	NA	HR
W24	IO_L18N_T2_D27_14	2	14	NA	NA	HR
W25	IO_L19P_T3_D26_14	3	14	NA	NA	HR
W26	IO_L19N_T3_D25_VREF_14	3	14	NA	NA	HR
Y25	IO_L20P_T3_D24_14	3	14	NA	NA	HR
Y26	IO_L20N_T3_D23_14	3	14	NA	NA	HR
W23	IO_L21P_T3_DQS_14	3	14	NA	NA	HR
Y23	IO_L21N_T3_DQS_D22_14	3	14	NA	NA	HR
AA24	IO_L22P_T3_D21_14	3	14	NA	NA	HR
AA25	IO_L22N_T3_D20_14	3	14	NA	NA	HR
AA22	IO_L23P_T3_D19_14	3	14	NA	NA	HR
AA23	IO_L23N_T3_D18_14	3	14	NA	NA	HR
Y21	IO_L24P_T3_D17_14	3	14	NA	NA	HR
Y22	IO_L24N_T3_D16_14	3	14	NA	NA	HR
Y20	IO_25_14	NA	14	NA	NA	HR
J21	IO_0_15	NA	15	NA	NA	HR
J20	IO_L1P_T0_AD0P_15	0	15	NA	NA	HR
H21	IO_L1N_T0_AD0N_15	0	15	NA	NA	HR
J19	IO_L2P_T0_AD8P_15	0	15	NA	NA	HR
H19	IO_L2N_T0_AD8N_15	0	15	NA	NA	HR
K21	IO_L3P_T0_DQS_AD1P_15	0	15	NA	NA	HR
K22	IO_L3N_T0_DQS_AD1N_15	0	15	NA	NA	HR
G20	IO_L4P_T0_15	0	15	NA	NA	HR

G21	IO_L4N_T0_15	0	15	NA	NA	HR
L20	IO_L5P_T0_AD9P_15	0	15	NA	NA	HR
K20	IO_L5N_T0_AD9N_15	0	15	NA	NA	HR
M20	IO_L6P_T0_15	0	15	NA	NA	HR
M21	IO_L6N_T0_VREF_15	0	15	NA	NA	HR
D24	IO_L7P_T1_AD2P_15	1	15	NA	NA	HR
D25	IO_L7N_T1_AD2N_15	1	15	NA	NA	HR
E22	IO_L8P_T1_AD10P_15	1	15	NA	NA	HR
D23	IO_L8N_T1_AD10N_15	1	15	NA	NA	HR
B25	IO_L9P_T1_DQS_AD3P_15	1	15	NA	NA	HR
B26	IO_L9N_T1_DQS_AD3N_15	1	15	NA	NA	HR
F23	IO_L10P_T1_AD11P_15	1	15	NA	NA	HR
E23	IO_L10N_T1_AD11N_15	1	15	NA	NA	HR
E25	IO_L11P_T1_SRCC_15	1	15	NA	NA	HR
E26	IO_L11N_T1_SRCC_15	1	15	NA	NA	HR
D26	IO_L12P_T1_MRCC_15	1	15	NA	NA	HR
C26	IO_L12N_T1_MRCC_15	1	15	NA	NA	HR
G22	IO_L13P_T2_MRCC_15	2	15	NA	NA	HR
F22	IO_L13N_T2_MRCC_15	2	15	NA	NA	HR
J23	IO_L14P_T2_SRCC_15	2	15	NA	NA	HR
H23	IO_L14N_T2_SRCC_15	2	15	NA	NA	HR
H24	IO_L15P_T2_DQS_15	2	15	NA	NA	HR
G24	IO_L15N_T2_DQS_15	2	15	NA	NA	HR
J26	IO_L16P_T2_15	2	15	NA	NA	HR
H26	IO_L16N_T2_15	2	15	NA	NA	HR
F24	IO_L17P_T2_15	2	15	NA	NA	HR
F25	IO_L17N_T2_15	2	15	NA	NA	HR
G25	IO_L18P_T2_15	2	15	NA	NA	HR
G26	IO_L18N_T2_15	2	15	NA	NA	HR
M22	IO_L19P_T3_15	3	15	NA	NA	HR
L23	IO_L19N_T3_VREF_15	3	15	NA	NA	HR
L22	IO_L20P_T3_15	3	15	NA	NA	HR
K23	IO_L20N_T3_15	3	15	NA	NA	HR
J24	IO_L21P_T3_DQS_15	3	15	NA	NA	HR
J25	IO_L21N_T3_DQS_15	3	15	NA	NA	HR
M24	IO_L22P_T3_15	3	15	NA	NA	HR
L24	IO_L22N_T3_15	3	15	NA	NA	HR
K25	IO_L23P_T3_15	3	15	NA	NA	HR
K26	IO_L23N_T3_15	3	15	NA	NA	HR
M25	IO_L24P_T3_RS1_15	3	15	NA	NA	HR
L25	IO_L24N_T3_RS0_15	3	15	NA	NA	HR
H22	IO_25_15	NA	15	NA	NA	HR
A13	IO_0_16	NA	16	NA	NA	HR
F14	IO_L1P_T0_16	0	16	NA	NA	HR
F15	IO_L1N_T0_16	0	16	NA	NA	HR
E15	IO_L2P_T0_16	0	16	NA	NA	HR
E16	IO_L2N_T0_16	0	16	NA	NA	HR
A14	IO_L3P_T0_DQS_16	0	16	NA	NA	HR

A15	IO_L3N_T0_DQS_16	0	16	NA	NA	HR
B14	IO_L4P_T0_16	0	16	NA	NA	HR
B15	IO_L4N_T0_16	0	16	NA	NA	HR
D14	IO_L5P_T0_16	0	16	NA	NA	HR
D15	IO_L5N_T0_16	0	16	NA	NA	HR
C13	IO_L6P_T0_16	0	16	NA	NA	HR
C14	IO_L6N_T0_VREF_16	0	16	NA	NA	HR
C17	IO_L7P_T1_16	1	16	NA	NA	HR
C18	IO_L7N_T1_16	1	16	NA	NA	HR
D16	IO_L8P_T1_16	1	16	NA	NA	HR
C16	IO_L8N_T1_16	1	16	NA	NA	HR
B16	IO_L9P_T1_DQS_16	1	16	NA	NA	HR
B17	IO_L9N_T1_DQS_16	1	16	NA	NA	HR
E17	IO_L10P_T1_16	1	16	NA	NA	HR
E18	IO_L10N_T1_16	1	16	NA	NA	HR
A17	IO_L11P_T1_SRCC_16	1	16	NA	NA	HR
A18	IO_L11N_T1_SRCC_16	1	16	NA	NA	HR
F17	IO_L12P_T1_MRCC_16	1	16	NA	NA	HR
F18	IO_L12N_T1_MRCC_16	1	16	NA	NA	HR
D18	IO_L13P_T2_MRCC_16	2	16	NA	NA	HR
C19	IO_L13N_T2_MRCC_16	2	16	NA	NA	HR
E20	IO_L14P_T2_SRCC_16	2	16	NA	NA	HR
E21	IO_L14N_T2_SRCC_16	2	16	NA	NA	HR
F19	IO_L15P_T2_DQS_16	2	16	NA	NA	HR
F20	IO_L15N_T2_DQS_16	2	16	NA	NA	HR
D19	IO_L16P_T2_16	2	16	NA	NA	HR
D20	IO_L16N_T2_16	2	16	NA	NA	HR
B19	IO_L17P_T2_16	2	16	NA	NA	HR
B20	IO_L17N_T2_16	2	16	NA	NA	HR
A19	IO_L18P_T2_16	2	16	NA	NA	HR
A20	IO_L18N_T2_16	2	16	NA	NA	HR
D21	IO_L19P_T3_16	3	16	NA	NA	HR
C21	IO_L19N_T3_VREF_16	3	16	NA	NA	HR
A22	IO_L20P_T3_16	3	16	NA	NA	HR
A23	IO_L20N_T3_16	3	16	NA	NA	HR
A24	IO_L21P_T3_DQS_16	3	16	NA	NA	HR
A25	IO_L21N_T3_DQS_16	3	16	NA	NA	HR
B21	IO_L22P_T3_16	3	16	NA	NA	HR
B22	IO_L22N_T3_16	3	16	NA	NA	HR
C24	IO_L23P_T3_16	3	16	NA	NA	HR
B24	IO_L23N_T3_16	3	16	NA	NA	HR
C22	IO_L24P_T3_16	3	16	NA	NA	HR
C23	IO_L24N_T3_16	3	16	NA	NA	HR
G19	IO_25_16	NA	16	NA	NA	HR
AE2	IO_0_33	NA	33	NA	NA	HR
AD1	IO_L1P_T0_33	0	33	NA	NA	HR
AE1	IO_L1N_T0_33	0	33	NA	NA	HR
AC3	IO_L2P_T0_33	0	33	NA	NA	HR

AC2	IO_L2N_T0_33	0	33	NA	NA	HR
AF3	IO_L3P_T0_DQS_33	0	33	NA	NA	HR
AF2	IO_L3N_T0_DQS_33	0	33	NA	NA	HR
AF5	IO_L4P_T0_33	0	33	NA	NA	HR
AF4	IO_L4N_T0_33	0	33	NA	NA	HR
AC4	IO_L5P_T0_33	0	33	NA	NA	HR
AD4	IO_L5N_T0_33	0	33	NA	NA	HR
AD3	IO_L6P_T0_33	0	33	NA	NA	HR
AE3	IO_L6N_T0_VREF_33	0	33	NA	NA	HR
AE6	IO_L7P_T1_33	1	33	NA	NA	HR
AE5	IO_L7N_T1_33	1	33	NA	NA	HR
AD6	IO_L8P_T1_33	1	33	NA	NA	HR
AD5	IO_L8N_T1_33	1	33	NA	NA	HR
AA8	IO_L9P_T1_DQS_33	1	33	NA	NA	HR
AB7	IO_L9N_T1_DQS_33	1	33	NA	NA	HR
AB6	IO_L10P_T1_33	1	33	NA	NA	HR
AC6	IO_L10N_T1_33	1	33	NA	NA	HR
AC8	IO_L11P_T1_SRCC_33	1	33	NA	NA	HR
AC7	IO_L11N_T1_SRCC_33	1	33	NA	NA	HR
AE8	IO_L12P_T1_MRCC_33	1	33	NA	NA	HR
AE7	IO_L12N_T1_MRCC_33	1	33	NA	NA	HR
AA10	IO_L13P_T2_MRCC_33	2	33	NA	NA	HR
AB10	IO_L13N_T2_MRCC_33	2	33	NA	NA	HR
AF8	IO_L14P_T2_SRCC_33	2	33	NA	NA	HR
AF7	IO_L14N_T2_SRCC_33	2	33	NA	NA	HR
AC9	IO_L15P_T2_DQS_33	2	33	NA	NA	HR
AD8	IO_L15N_T2_DQS_33	2	33	NA	NA	HR
AD10	IO_L16P_T2_33	2	33	NA	NA	HR
AD9	IO_L16N_T2_33	2	33	NA	NA	HR
AF10	IO_L17P_T2_33	2	33	NA	NA	HR
AF9	IO_L17N_T2_33	2	33	NA	NA	HR
AA9	IO_L18P_T2_33	2	33	NA	NA	HR
AB9	IO_L18N_T2_33	2	33	NA	NA	HR
AC12	IO_L19P_T3_33	3	33	NA	NA	HR
AD11	IO_L19N_T3_VREF_33	3	33	NA	NA	HR
AE12	IO_L20P_T3_33	3	33	NA	NA	HR
AF12	IO_L20N_T3_33	3	33	NA	NA	HR
AE11	IO_L21P_T3_DQS_33	3	33	NA	NA	HR
AE10	IO_L21N_T3_DQS_33	3	33	NA	NA	HR
AF14	IO_L22P_T3_33	3	33	NA	NA	HR
AF13	IO_L22N_T3_33	3	33	NA	NA	HR
AB11	IO_L23P_T3_33	3	33	NA	NA	HR
AC11	IO_L23N_T3_33	3	33	NA	NA	HR
AD14	IO_L24P_T3_33	3	33	NA	NA	HR
AD13	IO_L24N_T3_33	3	33	NA	NA	HR
AE13	IO_25_33	NA	33	NA	NA	HR
P6	IO_0_34	NA	34	NA	NA	HR
P3	IO_L1P_T0_34	0	34	NA	NA	HR

R3	IO_L1N_T0_34	0	34	NA	NA	HR
P1	IO_L2P_T0_34	0	34	NA	NA	HR
R1	IO_L2N_T0_34	0	34	NA	NA	HR
P5	IO_L3P_T0_DQS_34	0	34	NA	NA	HR
P4	IO_L3N_T0_DQS_34	0	34	NA	NA	HR
R2	IO_L4P_T0_34	0	34	NA	NA	HR
T2	IO_L4N_T0_34	0	34	NA	NA	HR
R5	IO_L5P_T0_34	0	34	NA	NA	HR
T5	IO_L5N_T0_34	0	34	NA	NA	HR
R7	IO_L6P_T0_34	0	34	NA	NA	HR
R6	IO_L6N_T0_VREF_34	0	34	NA	NA	HR
AB2	IO_L7P_T1_34	1	34	NA	NA	HR
AC1	IO_L7N_T1_34	1	34	NA	NA	HR
Y3	IO_L8P_T1_34	1	34	NA	NA	HR
Y2	IO_L8N_T1_34	1	34	NA	NA	HR
AA2	IO_L9P_T1_DQS_34	1	34	NA	NA	HR
AB1	IO_L9N_T1_DQS_34	1	34	NA	NA	HR
W1	IO_L10P_T1_34	1	34	NA	NA	HR
Y1	IO_L10N_T1_34	1	34	NA	NA	HR
U1	IO_L11P_T1_SRCC_34	1	34	NA	NA	HR
V1	IO_L11N_T1_SRCC_34	1	34	NA	NA	HR
V3	IO_L12P_T1_MRCC_34	1	34	NA	NA	HR
V2	IO_L12N_T1_MRCC_34	1	34	NA	NA	HR
T7	IO_L13P_T2_MRCC_34	2	34	NA	NA	HR
U7	IO_L13N_T2_MRCC_34	2	34	NA	NA	HR
U6	IO_L14P_T2_SRCC_34	2	34	NA	NA	HR
U5	IO_L14N_T2_SRCC_34	2	34	NA	NA	HR
T4	IO_L15P_T2_DQS_34	2	34	NA	NA	HR
U4	IO_L15N_T2_DQS_34	2	34	NA	NA	HR
V7	IO_L16P_T2_34	2	34	NA	NA	HR
V6	IO_L16N_T2_34	2	34	NA	NA	HR
T3	IO_L17P_T2_34	2	34	NA	NA	HR
U2	IO_L17N_T2_34	2	34	NA	NA	HR
V4	IO_L18P_T2_34	2	34	NA	NA	HR
W3	IO_L18N_T2_34	2	34	NA	NA	HR
W5	IO_L19P_T3_34	3	34	NA	NA	HR
W4	IO_L19N_T3_VREF_34	3	34	NA	NA	HR
Y5	IO_L20P_T3_34	3	34	NA	NA	HR
AA5	IO_L20N_T3_34	3	34	NA	NA	HR
W6	IO_L21P_T3_DQS_34	3	34	NA	NA	HR
Y6	IO_L21N_T3_DQS_34	3	34	NA	NA	HR
AB5	IO_L22P_T3_34	3	34	NA	NA	HR
AB4	IO_L22N_T3_34	3	34	NA	NA	HR
AA4	IO_L23P_T3_34	3	34	NA	NA	HR
AA3	IO_L23N_T3_34	3	34	NA	NA	HR
Y7	IO_L24P_T3_34	3	34	NA	NA	HR
AA7	IO_L24N_T3_34	3	34	NA	NA	HR
Y8	IO_25_34	NA	34	NA	NA	HR

H8	IO_0_35	NA	35	NA	NA	HR
G5	IO_L1P_T0_AD4P_35	0	35	NA	NA	HR
F5	IO_L1N_T0_AD4N_35	0	35	NA	NA	HR
K8	IO_L2P_T0_AD12P_35	0	35	NA	NA	HR
J8	IO_L2N_T0_AD12N_35	0	35	NA	NA	HR
G4	IO_L3P_T0_DQS_AD5P_35	0	35	NA	NA	HR
F4	IO_L3N_T0_DQS_AD5N_35	0	35	NA	NA	HR
H6	IO_L4P_T0_35	0	35	NA	NA	HR
G6	IO_L4N_T0_35	0	35	NA	NA	HR
J6	IO_L5P_T0_AD13P_35	0	35	NA	NA	HR
J5	IO_L5N_T0_AD13N_35	0	35	NA	NA	HR
H7	IO_L6P_T0_35	0	35	NA	NA	HR
G7	IO_L6N_T0_VREF_35	0	35	NA	NA	HR
G2	IO_L7P_T1_AD6P_35	1	35	NA	NA	HR
F2	IO_L7N_T1_AD6N_35	1	35	NA	NA	HR
F3	IO_L8P_T1_AD14P_35	1	35	NA	NA	HR
E3	IO_L8N_T1_AD14N_35	1	35	NA	NA	HR
H1	IO_L9P_T1_DQS_AD7P_35	1	35	NA	NA	HR
G1	IO_L9N_T1_DQS_AD7N_35	1	35	NA	NA	HR
E2	IO_L10P_T1_AD15P_35	1	35	NA	NA	HR
E1	IO_L10N_T1_AD15N_35	1	35	NA	NA	HR
H3	IO_L11P_T1_SRCC_35	1	35	NA	NA	HR
H2	IO_L11N_T1_SRCC_35	1	35	NA	NA	HR
J4	IO_L12P_T1_MRCC_35	1	35	NA	NA	HR
H4	IO_L12N_T1_MRCC_35	1	35	NA	NA	HR
K7	IO_L13P_T2_MRCC_35	2	35	NA	NA	HR
K6	IO_L13N_T2_MRCC_35	2	35	NA	NA	HR
M7	IO_L14P_T2_SRCC_35	2	35	NA	NA	HR
L7	IO_L14N_T2_SRCC_35	2	35	NA	NA	HR
L5	IO_L15P_T2_DQS_35	2	35	NA	NA	HR
K5	IO_L15N_T2_DQS_35	2	35	NA	NA	HR
N6	IO_L16P_T2_35	2	35	NA	NA	HR
M6	IO_L16N_T2_35	2	35	NA	NA	HR
M5	IO_L17P_T2_35	2	35	NA	NA	HR
M4	IO_L17N_T2_35	2	35	NA	NA	HR
L4	IO_L18P_T2_35	2	35	NA	NA	HR
L3	IO_L18N_T2_35	2	35	NA	NA	HR
N4	IO_L19P_T3_35	3	35	NA	NA	HR
N3	IO_L19N_T3_VREF_35	3	35	NA	NA	HR
K3	IO_L20P_T3_35	3	35	NA	NA	HR
J3	IO_L20N_T3_35	3	35	NA	NA	HR
L2	IO_L21P_T3_DQS_35	3	35	NA	NA	HR
K2	IO_L21N_T3_DQS_35	3	35	NA	NA	HR
N2	IO_L22P_T3_35	3	35	NA	NA	HR
N1	IO_L22N_T3_35	3	35	NA	NA	HR
K1	IO_L23P_T3_35	3	35	NA	NA	HR
J1	IO_L23N_T3_35	3	35	NA	NA	HR
M2	IO_L24P_T3_35	3	35	NA	NA	HR

M1	IO_L24N_T3_35	3	35	NA	NA	HR
N7	IO_25_35	NA	35	NA	NA	HR
A12	IO_0_36	NA	36	NA	NA	HR
E11	IO_L1P_T0_36	0	36	NA	NA	HR
E10	IO_L1N_T0_36	0	36	NA	NA	HR
F12	IO_L2P_T0_36	0	36	NA	NA	HR
E12	IO_L2N_T0_36	0	36	NA	NA	HR
D11	IO_L3P_T0_DQS_36	0	36	NA	NA	HR
D10	IO_L3N_T0_DQS_36	0	36	NA	NA	HR
B10	IO_L4P_T0_36	0	36	NA	NA	HR
A10	IO_L4N_T0_36	0	36	NA	NA	HR
C12	IO_L5P_T0_36	0	36	NA	NA	HR
C11	IO_L5N_T0_36	0	36	NA	NA	HR
B12	IO_L6P_T0_36	0	36	NA	NA	HR
B11	IO_L6N_T0_VREF_36	0	36	NA	NA	HR
F8	IO_L7P_T1_36	1	36	NA	NA	HR
E8	IO_L7N_T1_36	1	36	NA	NA	HR
C9	IO_L8P_T1_36	1	36	NA	NA	HR
C8	IO_L8N_T1_36	1	36	NA	NA	HR
B9	IO_L9P_T1_DQS_36	1	36	NA	NA	HR
A9	IO_L9N_T1_DQS_36	1	36	NA	NA	HR
F10	IO_L10P_T1_36	1	36	NA	NA	HR
F9	IO_L10N_T1_36	1	36	NA	NA	HR
A8	IO_L11P_T1_SRCC_36	1	36	NA	NA	HR
A7	IO_L11N_T1_SRCC_36	1	36	NA	NA	HR
D9	IO_L12P_T1_MRCC_36	1	36	NA	NA	HR
D8	IO_L12N_T1_MRCC_36	1	36	NA	NA	HR
C7	IO_L13P_T2_MRCC_36	2	36	NA	NA	HR
C6	IO_L13N_T2_MRCC_36	2	36	NA	NA	HR
B7	IO_L14P_T2_SRCC_36	2	36	NA	NA	HR
B6	IO_L14N_T2_SRCC_36	2	36	NA	NA	HR
E6	IO_L15P_T2_DQS_36	2	36	NA	NA	HR
D6	IO_L15N_T2_DQS_36	2	36	NA	NA	HR
F7	IO_L16P_T2_36	2	36	NA	NA	HR
E7	IO_L16N_T2_36	2	36	NA	NA	HR
E5	IO_L17P_T2_36	2	36	NA	NA	HR
D5	IO_L17N_T2_36	2	36	NA	NA	HR
B5	IO_L18P_T2_36	2	36	NA	NA	HR
A5	IO_L18N_T2_36	2	36	NA	NA	HR
C3	IO_L19P_T3_36	3	36	NA	NA	HR
C2	IO_L19N_T3_VREF_36	3	36	NA	NA	HR
C4	IO_L20P_T3_36	3	36	NA	NA	HR
B4	IO_L20N_T3_36	3	36	NA	NA	HR
B2	IO_L21P_T3_DQS_36	3	36	NA	NA	HR
B1	IO_L21N_T3_DQS_36	3	36	NA	NA	HR
A3	IO_L22P_T3_36	3	36	NA	NA	HR
A2	IO_L22N_T3_36	3	36	NA	NA	HR
D1	IO_L23P_T3_36	3	36	NA	NA	HR

C1	IO_L23N_T3_36	3	36	NA	NA	HR
D4	IO_L24P_T3_36	3	36	NA	NA	HR
D3	IO_L24N_T3_36	3	36	NA	NA	HR
A4	IO_25_36	NA	36	NA	NA	HR
A1	GND	NA	NA	NA	NA	NA
A6	GND	NA	NA	NA	NA	NA
A11	GND	NA	NA	NA	NA	NA
A16	GND	NA	NA	NA	NA	NA
A21	GND	NA	NA	NA	NA	NA
A26	GND	NA	NA	NA	NA	NA
B3	GND	NA	NA	NA	NA	NA
B8	GND	NA	NA	NA	NA	NA
B13	GND	NA	NA	NA	NA	NA
B18	GND	NA	NA	NA	NA	NA
C5	GND	NA	NA	NA	NA	NA
C15	GND	NA	NA	NA	NA	NA
C25	GND	NA	NA	NA	NA	NA
D2	GND	NA	NA	NA	NA	NA
D12	GND	NA	NA	NA	NA	NA
D22	GND	NA	NA	NA	NA	NA
E4	GND	NA	NA	NA	NA	NA
E9	GND	NA	NA	NA	NA	NA
E19	GND	NA	NA	NA	NA	NA
E24	GND	NA	NA	NA	NA	NA
F1	GND	NA	NA	NA	NA	NA
F6	GND	NA	NA	NA	NA	NA
F16	GND	NA	NA	NA	NA	NA
F26	GND	NA	NA	NA	NA	NA
G3	GND	NA	NA	NA	NA	NA
G8	GND	NA	NA	NA	NA	NA
G9	GND	NA	NA	NA	NA	NA
G11	GND	NA	NA	NA	NA	NA
G13	GND	NA	NA	NA	NA	NA
G15	GND	NA	NA	NA	NA	NA
G17	GND	NA	NA	NA	NA	NA
G23	GND	NA	NA	NA	NA	NA
H10	GND	NA	NA	NA	NA	NA
H12	GND	NA	NA	NA	NA	NA
H14	GND	NA	NA	NA	NA	NA
H16	GND	NA	NA	NA	NA	NA
H18	GND	NA	NA	NA	NA	NA
H20	GND	NA	NA	NA	NA	NA
H25	GND	NA	NA	NA	NA	NA
J2	GND	NA	NA	NA	NA	NA
J7	GND	NA	NA	NA	NA	NA
J9	GND	NA	NA	NA	NA	NA
J11	GND	NA	NA	NA	NA	NA
J13	GND	NA	NA	NA	NA	NA

J15	GND	NA	NA	NA	NA	NA
J17	GND	NA	NA	NA	NA	NA
K4	GND	NA	NA	NA	NA	NA
K10	GND	NA	NA	NA	NA	NA
K12	GND	NA	NA	NA	NA	NA
K14	GND	NA	NA	NA	NA	NA
K16	GND	NA	NA	NA	NA	NA
K18	GND	NA	NA	NA	NA	NA
K24	GND	NA	NA	NA	NA	NA
L1	GND	NA	NA	NA	NA	NA
L9	GND	NA	NA	NA	NA	NA
L11	GND	NA	NA	NA	NA	NA
L13	GND	NA	NA	NA	NA	NA
L15	GND	NA	NA	NA	NA	NA
L17	GND	NA	NA	NA	NA	NA
L19	GND	NA	NA	NA	NA	NA
L21	GND	NA	NA	NA	NA	NA
L26	GND	NA	NA	NA	NA	NA
M3	GND	NA	NA	NA	NA	NA
M8	GND	NA	NA	NA	NA	NA
M10	GND	NA	NA	NA	NA	NA
M12	GND	NA	NA	NA	NA	NA
M16	GND	NA	NA	NA	NA	NA
M18	GND	NA	NA	NA	NA	NA
N5	GND	NA	NA	NA	NA	NA
N9	GND	NA	NA	NA	NA	NA
N11	GND	NA	NA	NA	NA	NA
N15	GND	NA	NA	NA	NA	NA
N17	GND	NA	NA	NA	NA	NA
N19	GND	NA	NA	NA	NA	NA
N25	GND	NA	NA	NA	NA	NA
P2	GND	NA	NA	NA	NA	NA
P8	GND	NA	NA	NA	NA	NA
P10	GND	NA	NA	NA	NA	NA
P12	GND	NA	NA	NA	NA	NA
P16	GND	NA	NA	NA	NA	NA
P18	GND	NA	NA	NA	NA	NA
P22	GND	NA	NA	NA	NA	NA
R9	GND	NA	NA	NA	NA	NA
R11	GND	NA	NA	NA	NA	NA
R15	GND	NA	NA	NA	NA	NA
R17	GND	NA	NA	NA	NA	NA
R19	GND	NA	NA	NA	NA	NA
R24	GND	NA	NA	NA	NA	NA
T1	GND	NA	NA	NA	NA	NA
T6	GND	NA	NA	NA	NA	NA
T8	GND	NA	NA	NA	NA	NA
T10	GND	NA	NA	NA	NA	NA

T12	GND	NA	NA	NA	NA	NA
T14	GND	NA	NA	NA	NA	NA
T16	GND	NA	NA	NA	NA	NA
T18	GND	NA	NA	NA	NA	NA
T26	GND	NA	NA	NA	NA	NA
U3	GND	NA	NA	NA	NA	NA
U9	GND	NA	NA	NA	NA	NA
U11	GND	NA	NA	NA	NA	NA
U13	GND	NA	NA	NA	NA	NA
U15	GND	NA	NA	NA	NA	NA
U17	GND	NA	NA	NA	NA	NA
U19	GND	NA	NA	NA	NA	NA
U23	GND	NA	NA	NA	NA	NA
V8	GND	NA	NA	NA	NA	NA
V10	GND	NA	NA	NA	NA	NA
V12	GND	NA	NA	NA	NA	NA
V14	GND	NA	NA	NA	NA	NA
V16	GND	NA	NA	NA	NA	NA
V18	GND	NA	NA	NA	NA	NA
V20	GND	NA	NA	NA	NA	NA
V25	GND	NA	NA	NA	NA	NA
W2	GND	NA	NA	NA	NA	NA
W7	GND	NA	NA	NA	NA	NA
W9	GND	NA	NA	NA	NA	NA
W11	GND	NA	NA	NA	NA	NA
W13	GND	NA	NA	NA	NA	NA
W15	GND	NA	NA	NA	NA	NA
W17	GND	NA	NA	NA	NA	NA
W19	GND	NA	NA	NA	NA	NA
Y4	GND	NA	NA	NA	NA	NA
Y10	GND	NA	NA	NA	NA	NA
Y12	GND	NA	NA	NA	NA	NA
Y14	GND	NA	NA	NA	NA	NA
Y16	GND	NA	NA	NA	NA	NA
Y18	GND	NA	NA	NA	NA	NA
Y19	GND	NA	NA	NA	NA	NA
Y24	GND	NA	NA	NA	NA	NA
AA1	GND	NA	NA	NA	NA	NA
AA16	GND	NA	NA	NA	NA	NA
AA21	GND	NA	NA	NA	NA	NA
AA26	GND	NA	NA	NA	NA	NA
AB3	GND	NA	NA	NA	NA	NA
AB8	GND	NA	NA	NA	NA	NA
AB13	GND	NA	NA	NA	NA	NA
AC5	GND	NA	NA	NA	NA	NA
AC10	GND	NA	NA	NA	NA	NA
AC20	GND	NA	NA	NA	NA	NA
AC25	GND	NA	NA	NA	NA	NA

AD2	GND	NA	NA	NA	NA	NA
AD17	GND	NA	NA	NA	NA	NA
AD22	GND	NA	NA	NA	NA	NA
AE4	GND	NA	NA	NA	NA	NA
AE9	GND	NA	NA	NA	NA	NA
AE14	GND	NA	NA	NA	NA	NA
AE24	GND	NA	NA	NA	NA	NA
AF1	GND	NA	NA	NA	NA	NA
AF6	GND	NA	NA	NA	NA	NA
AF11	GND	NA	NA	NA	NA	NA
AF16	GND	NA	NA	NA	NA	NA
AF21	GND	NA	NA	NA	NA	NA
AF26	GND	NA	NA	NA	NA	NA
G10	VCCINT	NA	NA	NA	NA	NA
G12	VCCINT	NA	NA	NA	NA	NA
G14	VCCINT	NA	NA	NA	NA	NA
G16	VCCINT	NA	NA	NA	NA	NA
G18	VCCINT	NA	NA	NA	NA	NA
H9	VCCINT	NA	NA	NA	NA	NA
H11	VCCINT	NA	NA	NA	NA	NA
H13	VCCINT	NA	NA	NA	NA	NA
H15	VCCINT	NA	NA	NA	NA	NA
H17	VCCINT	NA	NA	NA	NA	NA
J10	VCCINT	NA	NA	NA	NA	NA
J12	VCCINT	NA	NA	NA	NA	NA
J14	VCCINT	NA	NA	NA	NA	NA
J16	VCCINT	NA	NA	NA	NA	NA
J18	VCCINT	NA	NA	NA	NA	NA
K9	VCCINT	NA	NA	NA	NA	NA
K11	VCCINT	NA	NA	NA	NA	NA
K13	VCCINT	NA	NA	NA	NA	NA
K15	VCCINT	NA	NA	NA	NA	NA
K17	VCCINT	NA	NA	NA	NA	NA
L10	VCCINT	NA	NA	NA	NA	NA
L12	VCCINT	NA	NA	NA	NA	NA
L14	VCCINT	NA	NA	NA	NA	NA
L16	VCCINT	NA	NA	NA	NA	NA
L18	VCCINT	NA	NA	NA	NA	NA
M9	VCCINT	NA	NA	NA	NA	NA
M11	VCCINT	NA	NA	NA	NA	NA
M15	VCCINT	NA	NA	NA	NA	NA
M17	VCCINT	NA	NA	NA	NA	NA
N10	VCCINT	NA	NA	NA	NA	NA
N12	VCCINT	NA	NA	NA	NA	NA
N16	VCCINT	NA	NA	NA	NA	NA
N18	VCCINT	NA	NA	NA	NA	NA
P9	VCCINT	NA	NA	NA	NA	NA
P11	VCCINT	NA	NA	NA	NA	NA

P15	VCCINT	NA	NA	NA	NA	NA
P17	VCCINT	NA	NA	NA	NA	NA
R10	VCCINT	NA	NA	NA	NA	NA
R12	VCCINT	NA	NA	NA	NA	NA
R16	VCCINT	NA	NA	NA	NA	NA
R18	VCCINT	NA	NA	NA	NA	NA
T9	VCCINT	NA	NA	NA	NA	NA
T11	VCCINT	NA	NA	NA	NA	NA
T13	VCCINT	NA	NA	NA	NA	NA
T15	VCCINT	NA	NA	NA	NA	NA
T17	VCCINT	NA	NA	NA	NA	NA
U10	VCCINT	NA	NA	NA	NA	NA
U12	VCCINT	NA	NA	NA	NA	NA
U14	VCCINT	NA	NA	NA	NA	NA
U16	VCCINT	NA	NA	NA	NA	NA
U18	VCCINT	NA	NA	NA	NA	NA
V9	VCCINT	NA	NA	NA	NA	NA
V11	VCCINT	NA	NA	NA	NA	NA
V13	VCCINT	NA	NA	NA	NA	NA
V15	VCCINT	NA	NA	NA	NA	NA
V17	VCCINT	NA	NA	NA	NA	NA
W10	VCCINT	NA	NA	NA	NA	NA
W12	VCCINT	NA	NA	NA	NA	NA
W14	VCCINT	NA	NA	NA	NA	NA
W16	VCCINT	NA	NA	NA	NA	NA
W18	VCCINT	NA	NA	NA	NA	NA
Y9	VCCINT	NA	NA	NA	NA	NA
Y11	VCCINT	NA	NA	NA	NA	NA
Y13	VCCINT	NA	NA	NA	NA	NA
Y15	VCCINT	NA	NA	NA	NA	NA
Y17	VCCINT	NA	NA	NA	NA	NA
K19	VCCAUX	NA	NA	NA	NA	NA
M19	VCCAUX	NA	NA	NA	NA	NA
P19	VCCAUX	NA	NA	NA	NA	NA
T19	VCCAUX	NA	NA	NA	NA	NA
V19	VCCAUX	NA	NA	NA	NA	NA
E14	VCCO_0	NA	0	NA	NA	NA
AC15	VCCO_0	NA	0	NA	NA	NA
AB18	VCCO_13	NA	13	NA	NA	NA
AB23	VCCO_13	NA	13	NA	NA	NA
AE19	VCCO_13	NA	13	NA	NA	NA
N20	VCCO_14	NA	14	NA	NA	NA
T21	VCCO_14	NA	14	NA	NA	NA
W22	VCCO_14	NA	14	NA	NA	NA
F21	VCCO_15	NA	15	NA	NA	NA
J22	VCCO_15	NA	15	NA	NA	NA
M23	VCCO_15	NA	15	NA	NA	NA
B23	VCCO_16	NA	16	NA	NA	NA

C20	VCCO_16	NA	16	NA	NA	NA
D17	VCCO_16	NA	16	NA	NA	NA
AA11	VCCO_33	NA	33	NA	NA	NA
AD7	VCCO_33	NA	33	NA	NA	NA
AD12	VCCO_33	NA	33	NA	NA	NA
R4	VCCO_34	NA	34	NA	NA	NA
V5	VCCO_34	NA	34	NA	NA	NA
AA6	VCCO_34	NA	34	NA	NA	NA
H5	VCCO_35	NA	35	NA	NA	NA
L6	VCCO_35	NA	35	NA	NA	NA
P7	VCCO_35	NA	35	NA	NA	NA
C10	VCCO_36	NA	36	NA	NA	NA
D7	VCCO_36	NA	36	NA	NA	NA
F11	VCCO_36	NA	36	NA	NA	NA
L8	VCCBRAM	NA	NA	NA	NA	NA
N8	VCCBRAM	NA	NA	NA	NA	NA
R8	VCCBRAM	NA	NA	NA	NA	NA
U8	VCCBRAM	NA	NA	NA	NA	NA
W8	VCCBRAM	NA	NA	NA	NA	NA

Total Number of Pins, 676

3 TIMING CONSTRAINTS.

create_clock -period 10.0 [get_ports clk] # Define a 100 MHz clock (adjust as needed) ///Initial

set_input_delay -max 2.0 -clock clk [get_ports dividend]

set_input_delay -max 2.0 -clock clk [get_ports divisor]

set_input_delay -max 2.0 -clock clk [get_ports start]

set_input_delay -max 2.0 -clock clk [get_ports reset]

set_output_delay -max 2.0 -clock clk [get_ports quotient]

set_output_delay -max 2.0 -clock clk [get_ports remainder]

set_output_delay -max 2.0 -clock clk [get_ports done]

set_max_delay -from [get_ports dividend] -to [get_ports quotient] 20.0 # Define max delay paths

set_max_delay -from [get_ports divisor] -to [get_ports remainder] 20.0

set_false_path -from [get_ports reset] -to [get_ports quotient]

```
set_false_path -from [get_ports reset] -to [get_ports remainder]
////////////////////////////////////////////////////////////////////////////////////////////////////////////
# Create clock for 'clk' input (100 MHz example, modify for your clock source)// Runing
create_clock -name clk -period 10.000 [get_ports clk]

# Reset, start are regular inputs
set_input_delay -clock clk -max 2.0 [get_ports reset]
set_input_delay -clock clk -max 2.0 [get_ports start]

# Dividend and divisor are inputs (bus constraint applied to all bits)
set_input_delay -clock clk -max 2.0 [get_ports dividend[*]]
set_input_delay -clock clk -max 2.0 [get_ports divisor[*]]

# Quotient and remainder are outputs (bus constraint applied to all bits)
set_output_delay -clock clk -max 2.0 [get_ports quotient[*]]
set_output_delay -clock clk -max 2.0 [get_ports remainder[*]]

# Done is an output
set_output_delay -clock clk -max 2.0 [get_ports done]

# Optional: Define input/output loads if needed (depends on your hardware board)
set_property PACKAGE_PIN <pin_number> [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
# Repeat for other ports (reset, start, dividend, divisor, etc.)
////////////////////////////////////////////////////////////////////////////
# Clock Definition
create_clock -period 10.000 -name clk [get_ports clk]

# Input Delays
set_input_delay -clock clk -max 5.0 [get_ports dividend]
set_input_delay -clock clk -max 5.0 [get_ports divisor]
set_input_delay -clock clk -max 5.0 [get_ports start]
set_input_delay -clock clk -max 5.0 [get_ports reset]

# Output Delays
```

```
set_output_delay -clock clk -max 5.0 [get_ports quotient]
set_output_delay -clock clk -max 5.0 [get_ports remainder]
set_output_delay -clock clk -max 5.0 [get_ports done]

# Set false paths if needed (e.g., for asynchronous signals if applicable)
```

SYNTHSIZE REPORT:[Xilinx ISE 14.7]

```
Started : "Synthesize - XST".
Running xst...
Command Line: xst -intstyle ise -ifn "D:/rk/srt/srt.xst" -ofn "D:/rk/srt/srt.syr"
Reading design: srt.prj

=========================================================================
*                     HDL Compilation                   *
=========================================================================
Compiling verilog file "srt.v" in library work
Module <srt> compiled
No errors in compilation
Analysis of file <"srt.prj"> succeeded.


=========================================================================
*                  Design Hierarchy Analysis                    *
=========================================================================
Analyzing hierarchy for module <srt> in library <work> with parameters.
        CALCULATE = "01"
        DONE = "10"
        IDLE = "00"


=========================================================================
*                     HDL Analysis                      *
```

```
=========================================================================
Analyzing top module <srt>.
        CALCULATE = 2'b01
        DONE = 2'b10
        IDLE = 2'b00
Module <srt> is correct for synthesis.


=========================================================================
*                     HDL Synthesis                       *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <srt>.
    Related source file is "srt.v".
    Found finite state machine <FSM_0> for signal <state>.
    -----------------------------------------------------------------
    | States            | 3                                 |
    | Transitions       | 7                                 |
    | Inputs            | 3                                 |
    | Outputs           | 4                                 |
    | Clock             | clk                (rising_edge)  |
    | Reset             | reset              (positive)     |
    | Reset type        | asynchronous                      |
    | Reset State       | 00                                |
    | Encoding          | automatic                         |
    | Implementation    | LUT                               |
    -----------------------------------------------------------------
    Found 1-bit register for signal <done>.
    Found 32-bit register for signal <quotient>.
    Found 32-bit register for signal <remainder>.
    Found 6-bit register for signal <count>.
    Found 6-bit subtractor for signal <count$addsub0000> created at line 85.
    Found 32-bit register for signal <d>.
```

Found 32-bit register for signal <q>.

Found 32-bit register for signal <s>.

Found 32-bit subtractor for signal <s$addsub0000> created at line 79.

Found 32-bit comparator greatequal for signal <s$cmp_ge0000> created at line 78.

Found 6-bit comparator greater for signal <state$cmp_gt0000> created at line 72.

Found 32-bit register for signal <z>.

Summary:

inferred 1 Finite State Machine(s).

inferred 193 D-type flip-flop(s).

inferred 2 Adder/Subtractor(s).

inferred 2 Comparator(s).

Unit <srt> synthesized.

==========

HDL Synthesis Report

Macro Statistics

Adders/Subtractors : 2

32-bit subtractor : 1

6-bit subtractor : 1

Registers : 8

1-bit register : 1

32-bit register : 6

6-bit register : 1

Comparators : 2

32-bit comparator greatequal : 1

6-bit comparator greater : 1

==========

==========

* Advanced HDL Synthesis *

==========

Analyzing FSM <FSM_0> for best encoding.

Optimizing FSM <state/FSM> on signal <state[1:2]> with gray encoding.

```
-------------------
 State | Encoding
-------------------
 00    | 00
 01    | 01
 10    | 11
-------------------
```

==========

Advanced HDL Synthesis Report

Macro Statistics

```
# FSMs                          : 1
# Adders/Subtractors            : 2
 32-bit subtractor              : 1
 6-bit subtractor               : 1
# Registers                     : 199
 Flip-Flops                     : 199
# Comparators                   : 2
 32-bit comparator greatequal   : 1
 6-bit comparator greater       : 1
```

==========

==========

* Low Level Synthesis *

==========

Optimizing unit <srt> ...

Mapping all equations...

Building and optimizing final netlist ...

Found area constraint ratio of 100 (+ 5) on block srt, actual ratio is 0.

Final Macro Processing ...

==========

Final Register Report

Macro Statistics

Registers : 201

Flip-Flops : 201

==========

==========

* Partition Report *

==========

Partition Implementation Status

No Partitions were found in this design.

==========

* Final Report *

==========

Clock Information:

Clock Signal | Clock buffer(FF name) | Load |

---

clk | BUFGP | 201 |

Asynchronous Control Signals Information:

Control Signal	Buffer(FF name)	Load
reset	IBUF	137

Timing Summary:

Speed Grade: -3

Minimum period: 2.744ns (Maximum Frequency: 364.460MHz)

Minimum input arrival time before clock: 3.960ns

Maximum output required time after clock: 2.779ns

Maximum combinational path delay: No path found

==========

Process "Synthesize - XST" completed successfully