**DESIGN AND IMPLEMENTATION OF**

**SLOW AND FAST DIVISON ALGORITHMS IN**

**VLSI INDUSTRY PURPOSE**

|  |  |  |
| --- | --- | --- |
| Subhashini K  Electronics and communication Engineering  *Rajalakshmi Institute of Technology,*  *Chennai, India*  *subhashini.k.2021.ece@ritchennai.edu.in* | Sujitha V  Electronics and communication  *Rajalakshmi Institute of Technology*  *Chennai, India*  *sujitha.v.2021.ece@ritchennai.edu.in*  Franklin Telfer .L  *Assistant Professor*  *Department of Electronics and communication engineering*  *Rajalakshmi Institute of Technology franklintelfer@ritchennai.edu.in* | Thirisha K  Electronics and communication  *Rajalakshmi Institute of Technology*  *Chennai, India*  *thirisha.k.2021.ece@ritchennai.* |

***Abstract—******Many algorithms have been developed for implementing division in hardware. These algorithms differ in many aspects, including quotient convergence rate, fundamental hardware primitives, and mathematical formulations. This paper presents a taxonomy of division algorithms which classifies the algorithms based upon their hardware implementations and impact on system design. Division algorithms can be divided into five classes: digit recurrence, functional iteration, very high radix, table look-up, and variable latency.***

***Many practical division algorithms are hybrids of several of these classes. These algorithms are explaine*d *and compared in this work. It is found that for low-cost implementations where chip area must be +++minimized, digit recurrence algorithms are suitable. An implementation of division by functional iteration can***

***provide the lowest latency for typical multiplier latencies. Variable latency algorithms show promise for simultaneously minimizing average latency while also minimizing area.***

# I. INTRODUCTION

We all know what division is. It is a mathematical operation that gives a quotient and a remainder. We will see the hardware implementation of the Division algorithm in computer architecture.

We use registers and counters while performing division.

**Types of Division Algorithms:**

Division algorithm in computer architecture is of two categories. The first one is **Slow Division.** In the slow division, we get one digit of the quotient every iteration. The algorithms for slow division category are **Restoring, Non-Performing Restoring, Non-Restoring,** and **SRT.**

The second category is the **Fast Division.** In this method, the quotient is predicted to the closest approximation to the actual

quotient, and then the calculation starts. The algorithms for fast division category are **Newton–Raphson**, and **Goldschmidt.**Here we will discuss the restoring division algorithm in computer architecture for unsigned and signed integers.

**Restoring Algorithm For Unsigned Integer**:

The restoring division algorithm is a slow division algorithm that calculates the quotient digit by digit. This algorithm will generate a quotient and a remainder after the division algorithm. Division algorithm computer architecture uses registers for storing the numbers and calculations. The division works with the assumption that the dividend is greater than the divisor.

This division algorithm in computer architecture uses three registers. **Register A** is initialized to **0**, **register Q** stores the **dividend**, and **register M** stores the **divisor**. **N** is used as a counter and stores the number of bits present in the dividend.

Let us see the flowchart for the restoring division algorithm in computer architecture.

**Let us understand the flowchart:**

1. We initialize the variables. **Register A** is initialized with 0, **register Q** will have the dividend, and **register M** will contain the divisor. **N** is the counter and has a value equal to the number of bits present in the dividend.

1. The value of **AQ (here in this step, A and Q will be treated as a single unit)** will shift to the left.
2. In this step, subtraction occurs. M will be subtracted from A, and A will store the result.

1. In this step, we check for the most significant bit of A. Suppose the most significant bit in A is 1 after the above three stages in the restoring division algorithm in computer architecture. In that case, it will set the least significant bit of Q as 0, and the value of A will again become what it was before the subtraction operation in step 3. If the most significant bit in A is 0, then it will set the least significant bit of Q .

1. N is decreased by 1 in this

step.

1. In this step, we check the value of N. If the value of N becomes 0, we break the loop here or move back to step 2.
2. In this step, we have our answer with the quotient in Q and the remainder in A.

Let us see an example of the restoring division algorithm in computer architecture:

**Let the dividend be 0111(7) and the divisor 0011(3).**

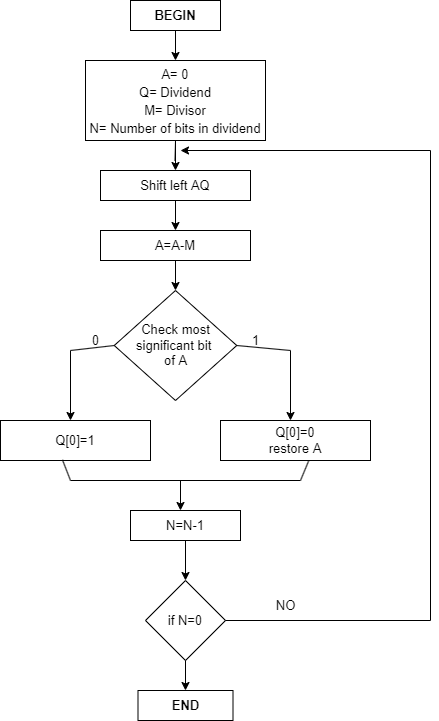


FIG 1

**II.OBJECTIVE**

The objective of fast and slow division algorithms is to efficiently compute the quotient and remainder when dividing one number (the dividend) by another (the divisor). The choice of algorithm depends on various factors, including the desired speed of computation, hardware limitations, and the level of accuracy required.

The objective of fast division algorithms is to minimize the computational time required to perform division operations. These algorithms are designed to provide rapid results, making them suitable for applications that require high-speed division, such as real-time signal processing or cryptographic calculations. Fast division algorithms often employ iterative or approximation techniques to achieve faster computation times while maintaining acceptable levels of accuracy.

On the other hand, slow division algorithms, like long division, prioritize accuracy and simplicity over speed. These algorithms are generally easier to understand and implement, making them suitable for educational purposes or situations where the division operations are not time-critical. Slow division algorithms are commonly used in manual calculations or as a basis for understanding the principles of division

It's worth noting that the classification of division algorithms into "fast" and "slow" is relative and depends on the specific context. What may be considered a slow algorithm in one scenario could be fast in another, depending on the computational resources available and the specific requirements of the application.

**III.EXISTING WORK**

The design and implementation of slow and fast division algorithms in computer architecture has been a topic of research and development for many years. Here is an overview of some prior work in this area:

**Slow Division Algorithms**:

**Restoring Division**: This algorithm performs division by repeated subtraction. It is straightforward but time-consuming since it requires multiple subtractions to find the quotient and remainder.

**Non-Restoring Division**: This algorithm improves the restoring division by guessing the quotient and then adjusting it iteratively until the correct quotient is obtained. It reduces the number of iterations but still involves multiple subtractions.

**SRT Division**: Sweeney, Robertson, and Tocher (SRT) division is an itertive algorithm that performs division using shifts and

subtracts operations. It reduces the number of iterations further by using a precomputed table of partial remainders.

**Fast Division Algorithms:**

**Radix-2 Division**: This algorithm uses shifts and compares to perform division in binary representation. It is efficient but limited to powers of two divisors.

**Newton-Raphson Division**: The Newton-Raphson iterative algorithm uses an initial guess and a series of iterations to improve the approximation of the quotient. It converges quickly but requires more hardware resources.

**Goldschmidt Division**: The Goldschmidt algorithm combines Newton-Raphson division with a multiplicative inverse. It converges rapidly but requires more complex hardware.

In terms of implementation, these algorithms have been implemented using various techniques, including:

**Software-based implementation**: These algorithms can be implemented as software routines executed by the CPU. They utilize the CPU's arithmetic and control instructions to perform the necessary operations.

**Hardware-based implementation**: Dedicated hardware units can be designed to perform division operations efficiently. These units can be integrated into the CPU or implemented as standalone coprocessors. They often use specialized arithmetic circuits and optimized control logic.

**Hybrid implementations**: Some implementations combine software and hardware approaches to leverage the strengths of both. For example, a hardware division unit can be used for most divisions, while slow division algorithms can be used as fallback routines for certain cases.

Numerous research papers and textbooks have covered the design and implementation of slow and fast division algorithms in computer architecture. It is a complex field with ongoing advancements and optimizations to improve the efficiency and speed of division operations.

**IV.PROPOSED TO WORK**

# 

Here's an outline of the approach for designing and implementing these algorithms:

**Slow Division Algorithm**: The slow division algorithm, also known as the restoring division algorithm, is a straightforward approach to division. It involves repeated subtraction and shifting to find the quotient and remainder**.**

**Fast Division Algorithm**: The fast division algorithm aims to improve the speed of the division operation. There are several fast division algorithms available, such as the non-restoring division algorithm, SRT division, and Newton-Raphson division. Choose an appropriate fast division algorithm based on your specific requirements**.**

**Implementation and Simulation**: Implement the designed hardware architecture using a hardware description language (HDL) like VHDL or Verilog. Simulate the hardware design using a simulation tool to verify its correctness and functionality.

**Performance Analysis**: Evaluate the performance of your division algorithms and hardware implementation. Compare the execution time, resource utilization, and throughput of the slow and fast division algorithms. Identify any bottlenecks or areas for further optimization.

**Iterative Refinement**: Based on the performance analysis, refine the design and implementation iterativelyto enhance efficiency.

**Testing and Verification**: Thoroughly test the division algorithms and the implemented hardware using various test cases, including corner cases and boundary conditions. Verify that the division operation produces correct results consistently.

**Documentation and Reporting**: Document the design process, implementation details, optimization techniques used, and performance analysis. Create a comprehensive report.

**V.RESULTS**

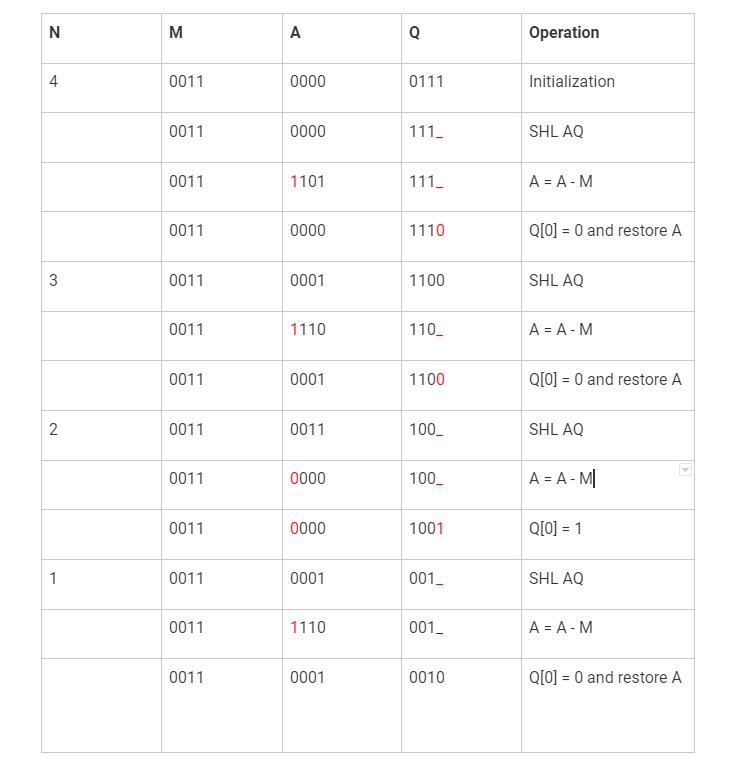


Table.1

**CONCLUSION**

This paper provides a comprehensive understanding of slow and fast division algorithms, their design considerations, and their impact on computer architecture. By analyzing the trade-offs between speed, accuracy, and hardware resources, researchers and practitioners can make informed decisions when designing and implementing division units in modern computer systems.

**REFERENCES**

1.K. Hwang, Computer Arithmetic: Principles Architectures and Design, pp. 222-223, 1979.

2.S. Kuninobu, "Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation", IEEE Proceeding of Symposium on Computer Arithmetic, pp. 80-86, 1987

3.S. Waser and M. J. Flynn, "5" in Introduction to Arithmetic for Digital Systems Designers, New York: CBS College Publishing, 1982

4.P. Markenstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor", IBM Journal of Research and Development, vol. 34, pp. 111-119, 1990

5.D. A. Patterson and J. L. Hennessy, Computer. A Quantitative Approach, CA, San Mateo: Morgan Kaufman, 1990.

6 K. S. Trivedi and M. D. Ercegovac, "On Line Algorithms for Division and Multiplication", IEEE Trans. on Computers, vol. C26, no. 7, July 1977.

7. C. V. Freiman, "Statistical Analysis of Certain Binary division algorithms", Proc. IRE, vol. 49, pp. 91-103, Jan. 1961.