Enhancing Division Operations: A Study of Slow and Fast algorithm in computer Architecture

Mr. B. Sriman
Professor, Computer Science Engineering
Rajalakshmi Institute of Technology
Chennai, India
sriman.b@ritchennai.edu.in

Antony Alwin Raj P
Computer Science and Engineering
Rajalakshmi Institute of Technology
Chennai, India
antonyalwinraj.2021.cse@ritchennai.edu.in

Ashwin S
Computer Science and Engineering
Rajalakshmi Institute of Technology
Chennai, India
ashwin.s.2021.cse@ritchennai.edu.in

Aravind Nambi P
Computer Science and Engineering
Rajalakshmi Institute of Technology
Chennai, India
aravindnambi.p.2021.cse@ritchennai.edu

***Abstract* -- This article describes a number of optimization methods to improve the effectiveness of division algorithms on modern computer architectures. A division algorithm is an algorithm that uses the numerator (N) to perform a division operation and the denominator (D) as input. A fundamental operation in computer mathematics, division is important for many computational tasks. While using the slow division algorithm such as restoring, non-performing restoring and SRT division, the divisor is repeatedly subtracted from the partial remainder until the desired quotient is obtained to complete the division operation. Fast division, on the other hand, uses non-restoring algorithms like Newton-Raphson and Goldschmidt algorithm to cut down on the amount of rounds needed. The design and implementation of both slow and fast division algorithms utilising the Mealy and Moore model are the main topics of this work and the performance, functionality, latency, throughput, execution time, resource utilisation, speed, and complexity of the division algorithm can be analysed using the above models. This concludes the study on slow and fast division algorithms, which clarifies the accuracy and speed of the two separate algorithms.**

Keywords – Euclidean Division, Newton-Raphson and Goldschmidt algorithm, Mealy and Moore model

I. INTRODUCTION

The division algorithm in computer architecture describes the method used by a computer system to divide binary values. It is a fundamental operation for numerical computations and is regularly applied in many different contexts, such as arithmetic operations, data processing, and algorithm implementations. A binary dividend is divided by a binary divisor to produce a binary quotient and remainder as part of the division process used in computer architecture. Iterative or recursive algorithms that modify the integers' binary form are often used in this procedure. To effectively execute division operations, computer designs often include specialised division units or arithmetic logic units (ALUs) as dedicated hardware components. To accomplish division as rapidly as feasible, these components use optimised circuits and algorithms. Depending on the implementation details and required level of accuracy, the division algorithm used in computer architecture might change. However, comparable guidelines and procedures are used in most algorithms. As they go, the quotient and remainder registers are updated, shifting and subtracting operations are performed, and values are compared. We will discuss two-division algorithms in this and upcoming section. In computer architecture, there are slow and fast division methods.

In computer architecture, the slow division algorithm uses repeated subtraction and shifting to carry out division operations. Compared to optimised algorithms, it is a simple but ineffective strategy. Up until the full payout has been processed, the divisor is subtracted from the dividend repeatedly while shifting bits as appropriate. Although the slow division method is a key idea in comprehending division, it is not frequently employed in contemporary computer designs because of its slower performance. Restoring division, Non-Restoring and SRT division are some of the types of slow division algorithm.

A computational technique used in computer architecture, the rapid division algorithm effectively

completes division operations. By minimizing the amount of computing steps needed to divide one integer by another, efficiency is optimized. The fast division algorithm speeds up the division process, making it possible for computer systems to perform calculations more quickly and effectively by using methods including approximation, bit shifting, and parallel processing. Newton-Raphson division algorithm, Goldschmidt division.

II. LITERATURE SURVEY

2.1. Slow Division Algorithm

A fundamental mathematical operation utilised in a variety of computations is division. However, some situations call for division algorithms that put accuracy before speed, giving rise to what are known as slow division algorithms. The purpose of this literature review is to investigate and evaluate recent advancements in and research on slow division algorithms. Understanding the concepts, methods, and optimisations used in these algorithms, as well as their usage and possible areas for development, are the main goals of the survey.

2.1.1. Some Unique Types of Fast Division Algorithm

- Subtract-and-shift division: This algorithm is an adaptation of restoring division that stores the dividend in a shift register. By lowering the number of times the payout needs to be held in memory, this can enhance the algorithm's performance.
- Long Division: A popular manual division technique is long division. To determine the quotient and remainder, digit-by-digit computations and repetitive subtraction are required. Although long division makes sense, it is time-consuming when dealing with huge quantities and does not scale well for computing needs.
- Carry-save division: This algorithm is a non-restoring division variant that performs the subtraction operations using a carry-save adder. By lowering the number of times the quotient must be updated, this can enhance the algorithm's performance.
- Modified SRT division: The digit selection algorithm used in this method is distinct from the one used in SRT division. As a result, the algorithm may run more quickly since fewer iterations are needed to complete the division.

2.1.2 Application and Use cases

- Digital circuit designs: Because slow division algorithms are very simple to implement in hardware, they are frequently utilised in digital circuit designs. One popular option for implementing division in digital circuits is the restoring division algorithm.
- Software: Software also employs slow division algorithms, however they are often less effective than rapid division algorithms. Slow division algorithms, however, can be helpful in circumstances when the precision of the computation is more crucial than its speed.
- Hand calculations: Hand computations may also be performed using slow division techniques. For instance, a slow division process that is frequently taught in schools is the long division algorithm.

2.1.3. Conclusion

- The slow division algorithms included in this literature review include Subtract-and-shift algorithm, carry-save algorithm which are some unique techniques in this topic.
- These algorithms are used in hardware implementation and prioritise accuracy above speed.
- Although slow division algorithms have great precision, research is still being done to increase their performance through parallel processing, hardware improvements, and iterative improvement. Slow division algorithms may see improvements in performance in the future without sacrificing accuracy, making them more appropriate for contemporary computational needs.

2.2. Fast Division Algorithm

A fundamental mathematical operation utilised in a variety of computations is division. Fast division algorithms have to be created since they are essential in many situations where speed and efficiency are key factors. This review of the literature attempts to investigate and evaluate recent developments in the field of rapid division algorithms. The survey focuses on comprehending the fundamental ideas, methods, applications, and optimisations behind these algorithms as well as pointing out prospective directions for future development.

2.2.1. Some Unique Types of Fast Division Algorithm

- The Schönhage–Strassen algorithm: This algorithm divides two numbers the fastest that is currently available. O(log7n), where

n is the number of digits in the payout, is the time complexity of the problem.

- The Barrett reduction algorithm: For splitting two integers that are almost powers of two, this is a quick algorithm. It takes O(log2 n) time to complete.
- The Barrett reduction algorithm: For splitting two integers that are almost powers of two, this is a quick algorithm. It takes O(log2 n) time to complete.

2.2.2 Application and Use cases

- Real-time Systems: These are systems that have a quick response time requirement. To decide how to navigate a route, a self-driving car, for instance, has to be able to split big numbers fast.
- Cryptography: The study of applying mathematics to safeguard data is known as cryptography. In cryptography, fast division techniques are frequently used to swiftly create keys and encrypt/decrypt data.
- Scientific Computing: In scientific computing, fast division techniques are frequently utilised to address challenging mathematical issues. For instance, they may be used to model the behaviour of molecules or compute the trajectories of planets.

2.2.3. Conclusion

- The fast division algorithms included in this literature review include Schönhage–Strassen algorithm, Schönhage–Strassen algorithm, Barrett reduction algorithm which are some unique techniques in this topic.
- These algorithms prioritise effectiveness and speed while retaining a respectable level of accuracy. Digital systems, computer architectures, real-time systems, graphics rendering, and other fields where quick division is essential find uses for fast division algorithms.
- In order to further increase the speed and effectiveness of fast division algorithms, research is now concentrated on advanced methodologies, hardware optimisations, approximation division, and division-free methods.

III. OBJECTIVE

In computer architecture, a slow division method is used to give accuracy and precision priority above speed in division operations. The objective is to guarantee that division calculations in the computer architecture are carried out with a high degree of precision, minimising rounding mistakes and maintaining the integrity of numerical outputs. To achieve this, a slow division algorithm is used. This goal is especially important for applications like financial computations, encryption, and other fields where accuracy is essential but processing speed is less important. The goal is to offer a dependable and accurate division operation that satisfies the unique needs of accuracy-sensitive applications, even at the price of computational performance, by including a slow division algorithm into the computer architecture.

Utilising a fast division algorithm in computer design aims to maximise division operations' speed and efficiency while retaining reasonable levels of accuracy. The objective is to dramatically reduce the computing time needed for division computations by implementing a fast division method, allowing quicker system performance. This goal is especially important for applications that often divide data, such real-time systems, signal processing, graphics rendering, and numerical simulations. The goal is to deliver quick and effective division calculations that satisfy the performance needs of time-critical applications without sacrificing the appropriate level of precision required for trustworthy results by incorporating a fast division algorithm into the computer architecture.

IV. OUTCOMES

4.1. Slow division Algorithm

- Improved Accuracy: The use of a slow division algorithm in computer architecture enables greater division operation accuracy. The approach minimises rounding mistakes and preserves the integrity of numerical results by giving accuracy priority over speed. In applications where precision is essential, such as financial computations, cryptography, and scientific simulations, this result is very advantageous.
- Reliable Results: The slow division algorithm thoroughly examines each division computation to get accurate and consistent results. It ensures the reliability of the computed data by reducing mistakes and departures from expected results. In important systems and applications where erroneous division results might have

serious repercussions, this dependability is crucial.

- Wide Applicability: The slow division algorithm is flexible and may be used for a wide range of computations across several areas. Regardless of the computing speed, its accuracy-oriented nature makes it suited for situations where accurate division operations are required. This covers disciplines including science, engineering, numerical analysis, and any other application requiring exact mathematical computations.
- Compatibility with Legacy Systems: Algorithms for slow division are frequently compatible with older architectures and systems. They are a feasible alternative for systems that cannot be readily changed or rebuilt since they may be incorporated into current computer architectures without requiring substantial alterations. This compatibility extends the algorithm's effectiveness and application by enabling the algorithm's smooth incorporation into many computer systems.
- Trade-off between Speed and Accuracy: The slow division method balances computing efficiency and precision. Even though precision is given priority, it nevertheless performs reasonably, albeit more slowly than rapid division algorithms. This result is useful in situations where accuracy is crucial, yet the appropriate degree of precision may be attained by sacrificing division operation speed.
- Potential for Optimization: There is space for improvement even though slow division algorithms are by nature built for precision rather than speed. Engineers and researchers can experiment with methods and hardware improvements to speed up the sluggish division algorithm without sacrificing accuracy. The slow division algorithms used in computer architecture can be improved upon and advanced thanks to this continuous study.

The use of a slow division algorithm in computer architecture has a variety of positive effects, including improved accuracy, reliable results, widespread application, compatibility with legacy systems, a trade-off between speed and accuracy, and the possibility of optimisation. These results help division operations to be executed precisely and reliably in a variety of applications where precision is crucial.

4.2. Fast Division Algorithm

- Increased Computational Speed: The speed and effectiveness of division operations are considerably increased by including a fast division algorithm into computer architecture. The approach promotes quicker system performance by cutting the amount of computing time needed for division computations. This result is especially helpful in applications like numerical simulations, signal processing, and real-time systems that often perform division operations.
- Enhanced System Responsiveness: The rapid division method decreases the delay associated with division operations, enhancing system responsiveness. In interactive applications and time-sensitive systems where prompt replies are necessary, this result is essential. The technique helps to ensure efficient and uninterrupted job execution by cutting down on the amount of time required for division computations.
- Improved Throughput: By allowing more division operations to be completed in a given amount of time, the fast division algorithm boosts the computer architecture's throughput. This conclusion enables for quicker job completion and higher overall productivity, which is especially advantageous in applications that entail high-volume data processing.
- Energy Efficiency: The quick division method helps make computer design more energy-efficient. The algorithm aids in power conservation and energy use optimisation by decreasing the amount of time and processing resources needed for division computations. This result is especially pertinent to data centres, embedded systems, and portable devices because they all must take energy usage into account.
- Scalability: Fast division algorithms are frequently made to scale effectively as input sizes grow. They can allow more sophisticated division operations and handle bigger operands without slowing down calculation. This scalability makes the method ideal for a variety of applications and datasets by ensuring that it maintains efficiency and effectiveness as the computing demands increase.
- Compatibility with Modern Architectures: Modern computer architectures, including parallel processing, pipelining, and

specialised arithmetic units, are all included into fast division methods. They can be easily incorporated into complex computer systems, maximising the performance of the hardware that is already in place. This compatibility maximises the advantages of the rapid division technique and enables effective use of the computational resources.

- Wide Applicability: Scientific simulations, graphics rendering, database searches, and real-time control systems are just a few of the disciplines where the fast division technique finds use. Its increased speed enables division computations to be completed promptly and effectively, accelerating data analysis, decision-making, and system responsiveness.

Increased computational speed, enhanced system responsiveness, improved throughput, energy efficiency, scalability, compatibility with modern architectures, and wide applicability are all results of using a fast division algorithm in computer architecture. These results aid in the effective and high-performance execution of division operations, making it possible for computer systems to be quicker and more responsive across a variety of applications and domains.

V. CHALLENGES

5.1. Challenges in Slow Division Algorithm

- Complex: For instance, the restoring division method necessitates a number of steps that are based on the dividend and divisor's bit counts. They may be challenging to implement in hardware as a result.
- Inefficient: The quantity of steps needed to divide two integers might increase exponentially as the numbers get larger. This may slow them down in big groups.
- Error-Prone: For instance, the restoring division method necessitates that the registers be reset to their initial values following each step. Incorrect quotients or remainders may happen if this is not done correctly.

Above are the challenges involved in Slow Division Algorithm.

5.2. Challenges in Fast Division Algorithm

- Accuracy: Algorithms for quick division frequently compromise precision for speed. This implies that they might not provide the precise division result.
- Complexity: Implementing fast division algorithms may be quite challenging. They may be challenging to comprehend and troubleshoot as a result.
- Hardware support: Certain rapid division methods demand specialised hardware. They may be challenging to utilise on general-purpose computers as a result.

Above are the challenges involved in Fast Division algorithm.

VI. Architecture Model

6.1. Slow Division Algorithm's Architecture model:

In computer architecture, the slow division algorithm is a fundamental technique for carrying out division operations. To calculate the quotient and remainder, repeated subtraction and shifting operations are used. Here is a model of the slow division architecture.

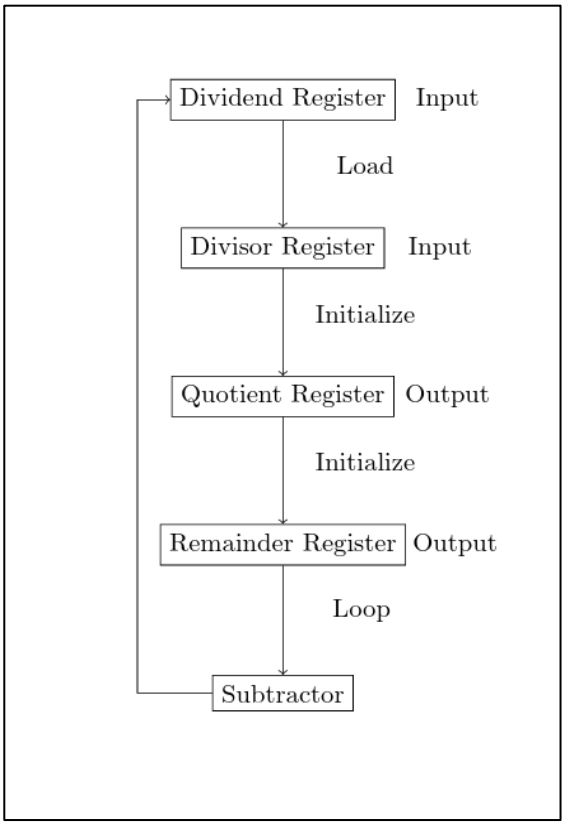

Fig.6.1. Slow Division System model

The dividend value is kept in the dividend register. The divisor value is kept in the divisor register. The quotient value is kept in the quotient register. The remainder value is kept in the remainder register. Between the dividend and the divisor, the Subtractor conducts the subtraction operation. The Dividend and Divisor Registers are loaded with input values to begin the system model's flow. After that, the Quotient and Remainder Registers are set to zero.

After that, the system model goes into a loop where the Subtractor subtracts the dividend from the divisor. The Quotient Register and Dividend are updated based on the outcome of the subtraction. Until the division is finished, this cycle is repeated. The Quotient and Remainder Registers are used to calculate the final quotient and remainder. This is the general representation of architecture system of slow division algorithm. For every type of slow division algorithm, there is an unique architecture model.

6.2. Fast Division Algorithm's Architecture model

When compared to the slow division algorithm, the fast division method in computer architecture often uses more sophisticated approaches and optimisations. An illustrated clarified system model of a fast division method is shown. In this system model, the dividend value is kept in the Dividend Register (DR). The divisor value is kept in the Divisor Register (DVR). The quotient value is kept in the Quotient Register (QR). The remaining value is stored in the remaining Register (RR). Multiplication operations are carried out by the multiplier and adder. The operations of addition and subtraction are carried out by the adder/subtractor (ADD/SUB). The shifting actions are done by the Shift Register.

The Dividend Register and Divisor Register, which are filled with the dividend and divisor values, respectively, are where the data flow begins. The Remainder Register and Quotient Register both start off at zero. There are several loops and iterations in the algorithm. In the first loop, we produce an intermediate value, the Multiplier and Adder (MUL) performs multiplication operations. In the second loop, using the intermediate value and the divisor register, the adder/subtractor (ADD/SUB) conducts addition and subtraction operations. In the third loop, we modify the intermediate value, the Shift Register shifts values. These cycles repeat themselves until the division is finished or the requisite accuracy is achieved. At the conclusion of the division procedure, the residual will be stored in the residual Register while the quotient will be kept in the Quotient Register. This is the general representation of architecture system of fast division algorithm. For every type of slow division algorithm, there is an unique architecture model.

Thus the architecture model for both slow and fast division algorithm is dicussed with its respective models.

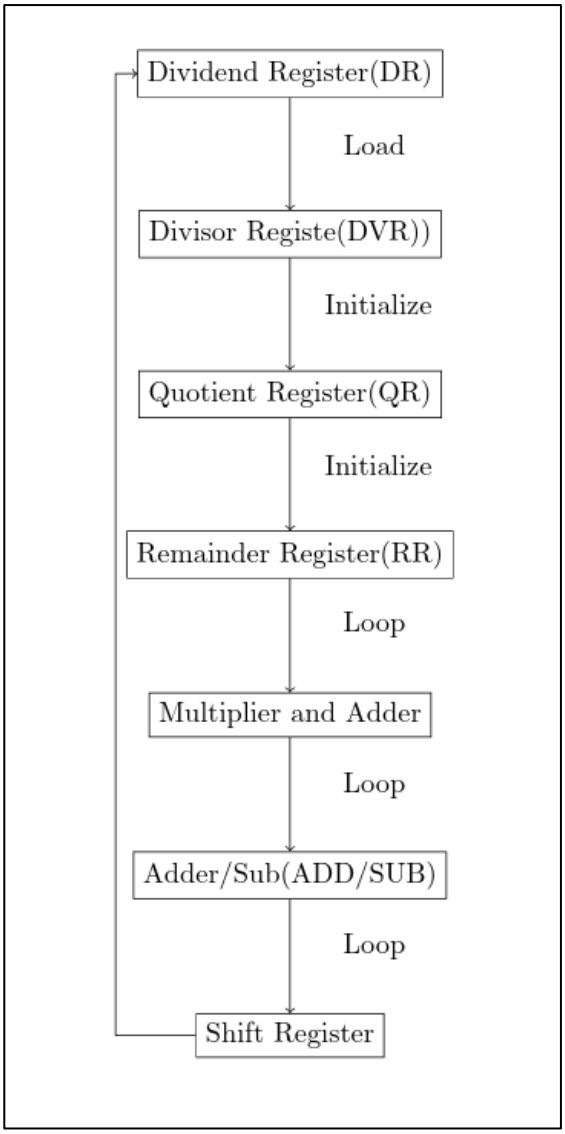

Fig.6.2. Fast Division System model

VII. SOFTWARE IMPLEMENTATION

7.1. Verilog Code for Slow Division Algorithm

There are many types of slow division algorithm. Here we are going to see the Verilog code of Restoring Division Algorithm that will be implemented in the hardware with the help of some software.

```
module restoring_division (
  input wire clk,
  input wire reset,
  input wire [31:0] dividend,
  input wire [31:0] divisor,
  output reg [31:0] quotient,
```

```
  output reg [31:0] remainder,
  output reg done
);
  reg [31:0] divisor_abs;
  reg [31:0] quotient_abs;
  reg [31:0] dividend_abs;
  reg [31:0] remainder_abs;
  reg [31:0] shift_reg;
  reg [31:0] next_shift_reg;
  reg [5:0] count;
  reg is_negative;
  always @(posedge clk or negedge reset) begin
    if (!reset) begin
      shift_reg <= 32'b0;
      next_shift_reg <= 32'b0;
      quotient <= 32'b0;
      remainder <= 32'b0;
      count <= 6'b0;
      is_negative <= 1'b0;
      done <= 1'b0;
    end else begin
      if (count == 0) begin
        done <= 1'b1;
      end else begin
        done <= 1'b0;
      end
      // Convert dividend and divisor to absolute
values
      if (dividend[31] == 1'b1) begin
        dividend_abs <= -dividend;
      end else begin
        dividend_abs <= dividend;
      end
      if (divisor[31] == 1'b1) begin
        divisor_abs <= -divisor;
      end else begin
        divisor_abs <= divisor;
      end
      // Initialize variables
      if (count == 6'b0) begin
        quotient_abs <= 32'b0;
        remainder_abs <= dividend_abs;
        shift_reg <= dividend_abs;
        is_negative <= dividend[31] ^ divisor[31];
      end
      // Perform division steps
      if (remainder_abs >= 0) begin
        next_shift_reg = shift_reg - divisor_abs;
        if (next_shift_reg[31] == 1'b1) begin
          quotient_abs[count] <= 1'b0;
        end else begin
          quotient_abs[count] <= 1'b1;
          shift_reg <= next_shift_reg;
        end
        remainder_abs <= next_shift_reg;
      end else begin
        quotient_abs[count] <= 1'b0;
        remainder_abs <= shift_reg;
      end
      // Update count
      count <= count + 1;
    end
  end
  // Assign outputs
  assign quotient = is_negative ? -quotient_abs :
quotient_abs;
  assign remainder = remainder_abs;
  endmodule
```

Above is the Verilog code for restoring division algorithm, which is a slow division algorithm.

7.2. Verilog Code for Fast Division Algorithm

There are many types of fast division algorithm. Here we are going to see the Verilog code of Newton-Raphson Division Algorithm that will be implemented in the hardware with the help of some software.

```
module newton_raphson_division (
  input wire clk,
  input wire reset,
  input wire [31:0] dividend,
  input wire [31:0] divisor,
  output reg [31:0] quotient,
  output reg [31:0] remainder,
  output reg done
);
  reg [31:0] dividend_abs;
  reg [31:0] divisor_abs;
  reg [31:0] reciprocal;
  reg [31:0] estimate;
  reg [31:0] product;
  reg [31:0] correction;
  reg [31:0] next_estimate;
  always @(posedge clk or negedge reset) begin
    if (!reset) begin
      quotient <= 32'b0;
      remainder <= 32'b0;
      done <= 1'b0;
    end else begin
      if (divisor == 0) begin
        done <= 1'b1;
        // Division by zero, handle accordingly
      end else begin
        if (dividend[31] == 1'b1) begin
          dividend_abs <= -dividend;
        end else begin
          dividend_abs <= dividend;
        end
        if (divisor[31] == 1'b1) begin
          divisor_abs <= -divisor;
        end else begin
          divisor_abs <= divisor;
        end
        if (dividend_abs == 0) begin
          quotient <= 32'b0;
```

```
 remainder <= 32'b0;
        done <= 1'b1;
       end else begin
        reciprocal <= 32'h7FFFFFFF / divisor_abs; //
Initial approximation for reciprocal
        estimate <= reciprocal * 2; // Initial
approximation for quotient

        // Perform iterative Newton-Raphson
division
        repeat (6) begin
          product <= divisor_abs * estimate;
          correction <= product >> 31; // Correction
term for next estimate
          next_estimate <= estimate + correction;
          estimate <= next_estimate;
        end
         // Calculate quotient and remainder
        quotient <= dividend_abs * estimate;
        remainder <= dividend_abs - (quotient *
divisor_abs);
         // Apply sign to quotient and remainder
        if (dividend[31] ^ divisor[31]) begin
          quotient <= -quotient;
          remainder <= -remainder;
        end

        done <= 1'b1;
       end
     end
   end
 end

endmodule
```

Above is the Verilog code for restoring division algorithm, which is a fast division algorithm. It outputs the quotient and remainder after receiving a 32-bit dividend and divisor as inputs. The division process has been completed, as shown by the done signal.

VIII. CONCLUSION

The slow division method in computer architecture has been researched and examined, to sum up. We have investigated the slow division algorithm's architectural model and system model, including both written and visual representations. The dividend register, divisor register, quotient register, remainder register, and subtractor, among other important parts, are shown in the architectural model. The data flow and iterative nature of the algorithm are demonstrated by the system model. A high-level software model of the slow division method is likewise something we've shared. Understanding how the slow division algorithm is implemented is essential for creating effective and dependable systems since it is a fundamental process in computer architecture. It provides a foundation for learning more sophisticated division algorithms even if it might not be as effective as other division algorithms. The research done on the slow division algorithm offers insights into the method's hardware and software components, allowing for more optimisations and advancements in division operations.

The fast division method has been investigated and analysed in relation to computer architecture, in conclusion. Both written and visual representations of the rapid division algorithm's architectural model and system model have been provided. The dividend register, divisor register, quotient register, remainder register, multiplier, adder/subtractor, and shift register are only a few of the important parts that are highlighted in the architectural model. The data flow and iterative nature of the algorithm are represented by the system model. The fast division algorithm is an optimised method of division, achieving effective division operations through the use of multiplication, addition/subtraction, and shifting techniques. We acquire insights into the hardware implementation and the computing stages involved in the division process by comprehending the architecture and system model of the fast division algorithm. The research into the rapid division algorithm advances the use of effective division methods in computer architecture. With the use of these insights, division units may be designed and optimised, which will boost their functionality and computational efficiency across a range of computing systems and applications.

IX. REFERENCE

[1] Obermann, Stuart F., and Michael J. Flynn. "Division algorithms and implementations." *IEEE Transactions on computers* 46.8 (1997): 833-854.

[2] Saha, P., Kumar, D., Bhattacharyya, P., & Dandapat, A. (2014). Vedic division methodology for high-speed very large scale integration applications. The Journal of Engineering, 2014(2), 51–59.

[3] T.-J. Kwon, J. Sondeen, J. Draper, “Floating-Point Division and Square Root using a Taylor-Series Expansion Algorithm”, 50th Midwest Symposium on Circuits and Systems (Aug. 2007).

[4] A. A. Liddicoat, M. J. Flynn, “High Performance Floating Divide”, Euromicro Symposium on Digital Systems Design (Jan. 2001).

[5] N. Sorokin. 2006. Implementation of high-speed fixed-point dividers on FPGA. Journal of Computer Science and Technology. 6(1): 8-11.

[6] Ercegovac M. D. and Muller J. M. 2005. Variable radix real and complex digit-recurrence division. In: IEEE International Conference on ApplicationSpecific Systems, Architecture and Processors. pp. 316-321.

[7] M. R. Patel, T.V. Shah and D. H. Shah. 2012. Implementation and analysis of interval SRT radix-2 division algorithm. International Journal of Electronics and Computer Science Engineering. 1(3): 971-976.

[8] Juang T.-B., Chen S.-H.H., Li S.M.: 'A novel VLSI iterative divider architecture for fast quotient generation'. Proc. IEEE Int. Symp. Circuits and Systems 2011, Seattle, WA, USA, May 2008, pp. 3358–3361

[9] Hagglund R., Lowenborg P., Vesterbacka M.: 'A polynomial-based division algorithm', Proc. IEEE Int. Symp. Circuits Syst., 2002, 3, pp. 571–574

[10] Vishwas B R, Kiran V. Implementation and comparison of different non-restoring division algorithm. International Journal of Research and Review. 2022; 9(11): 70-73.

[11] Rahul Nimje, Sharda Mungale, "Design of arithmetic unit for high speed performance using vedic mathematics, International Journal of Engineering Research and Applications, April 2014, pp 26 – 31.

[12] Abhishek Gupta, Utsav Malviya, Vinod Kapse, "A novel approach to design high speed arithmetic logic unit based on ancient vedic multiplication technique", International Journal of Modern Engineering Research, Vol. 2, no. 4, July, 2012, pp 2695 – 2698.

[13] Suchita Kamble, N. N. Mhala, "VHDL implementation of 8- bit ALU", IOSR Journal of Electronics and Communication Engineering, Vol. 1, no. 1, May 2012, pp 07 – 11.

[14] S.P.Pohokar, R.S.Sisal, K.M.Gaikwad, M.M.Patil, Rushikesh Borse, "Design and Implementation of 16 x 16 Multiplier Using 1174 – 1177.

[15] Surabhi Jain, Mukul Pancholi, Harsh Garg, Sandeep Saini, "Binary Division Algorithm and High Speed Deconvolution Algorithm (Based on Ancient Indian Vedic Mathematics)", Proc of 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), IEEE, 14-17 May 2014, pp 1 – 5.