Design and Implementation of a High-Performance Variable Latency Integer Divider in 15nm Technology

G. Sujatha[1], Budda Ahalya[2], Saladhi Krupa[3], Varanasi Meghana[4], Thoti Kishan[5], and B. Abhisheek[6]
{ 2022ece.l26@svce.edu.in[1], 2021ece.rm0@svce.edu.in[2], 2021ece.ro7@svce.edu.in[3], 2021ece.rn7@svce.edu.in[4], 2021ece.rj4@svce.edu.in[5] }

Professor Sri Venkateswara College of Engineering, Tirupati, 517520, India[1]
Department of Electronics Engineering, Sri Venkateswara College of Engineering Tirupati, 517520, India[2, 3, 4, 5, 6]

Abstract. Integer division is a fundamental operation in computer arithmetic, widely used in applications such as digital signal processing, cryptography, and artificial intelligence. However, due to its inherently sequential nature, integer division often becomes a performance bottleneck in modern computing systems. This paper presents a novel variable latency integer division algorithm, implemented and synthesized in 15nm technology using Cadence software. The proposed design dynamically skips unnecessary iterations by exploiting the relationship between the number of leading zeros in the divisor and the partial remainder, significantly reducing average latency and power consumption. Our implementation achieves an average latency of 1.45 clock cycles per 32-bit division, outperforming existing state-of-the-art designs. The design is validated through extensive simulations and benchmark testing, demonstrating its suitability for low-power embedded systems and high-performance computing applications.

Keywords: Integer division, variable latency, low-power design, 15nm technology, Cadence, FPGA, ASIC.

1 Introduction

Division can well be considered central in several domains of computer science such as digital signal processing, cryptography, artificial intelligence, and image processing. Division, however, is slower than addition and multiplication naturally as it is both non-associative and non-commutative, the latter leading to poor parallelism for which non-linear dependency comes into play, hence increased hardware cost. The traditional fixed-latency dividers, such as the restoring and non-restoring algorithms, have been extensively used due to their simple structure and predictable operating time. However, these designs suffer from high latency, especially for large operand sizes, making them unsuitable for applications where speed is crucial Variable latency dividers have become a competitive solution which reduces average execution time by dynamically controlling the number of divisions over the data. These designs utilize the fact that not all divisions need the same number of iterations, and allow faster completion in favourable operands. Variable latency mechanisms described in the literature also tend to exchange hardware complexity for benefits in performance, rendering them less appropriate for resource-constrained contexts.

In this paper, we present a new variable latency integer division method to solve the trade-off between latency and power consumption by detecting and then skipping unnecessary iterations

ICITSM-Part II 2025, April 28-29, Tiruchengode, India

DOI 10.4108/eai.28-4-2025.2358094

which are shifting operations in an integer division. The proposed design is designed by 15nm design technology using Cadence package which illustrated excellent performance than the already reported state-of-the-art designs. The primary contributions of this work are:

New Algorithm: Variable latency integer division algorithm obtained from non-restoring approach that does not perform any useful work and it may skip iterations dynamically based on the number of leading zeros in the divisor and partial remainder.

Efficient Hardware Design: Elaborate hardware design for 15nm technology with fast Count Leading Zeros (CLZ) units, single cycle barrel shifter and efficient control logic.

Complete Evaluation: Our design is extensively simulated and benchmarked to verify its performance such as average latency, power consumption and area used.

Comparison with State-of-the Art Designs: Comparison with existing fixed and variable latency dividers demonstrates that the proposed for SDL recovers many stages faster and it is more energy efficient.

The rest of the paper is structured as follows: Section II contains a thorough research of the literature available on integer division algorithms. Background and mathematical intuition behind Integer Division is explained in Section III. Experimental methodology and algorithm is presented in Section IV. Section V presents the implementation flow and hardware architecture. Results and comparison with existing designs are also presented and discussed in Section VI. Section VII concludes the paper and discusses the future work.

2 Literature Survey

The problem of integer division has been well researched in the past and many efficient ways have been reported in literature to achieve it for performance, area and power. These algorithms generally fall into two groups: fixed-latency and variable-latency dividers.

2.1 Fixed-Latency Dividers

The restoring algorithm, introduced by Ercegovac and Lang [1], works by repeatedly subtracting the divisor from the dividend and restoring the remainder when it becomes negative. While this method is straightforward, it results in high latency due to the restoration process. In contrast, the non-restoring algorithm eliminates the restoration step by allowing negative remainders, which reduces the number of operations but increases the complexity of control logic.

Park and Miller [2] described the non-restoring algorithm as avoiding the restoration step by permitting negative remainders that are fixed in further iterations. Hongal and Anita [11] noted that this saves the number of operations but makes the control logic bigger. Hwang et al. [13] highlighted that fixed-latency dividers are still quite slow, particularly for larger sizes of operands.

2.2 Variable-Latency Dividers

Fang and Leeser [14] proposed variable-latency dividers to lower the average cost of iteration by adaptively configuring the iteration count for input data. Obaid [3] introduced a data-dependent divider, which omits computations by detecting leading zeros in the dividend and divisor. This method has much lower latency; however, it requires additional hardware for leading zero detection and dynamic shifting.

Sinaga and Yang [4] proposed a dividend reuse priority encoder-based divider, which applies a dynamic shifter to align the divisor with the dividend before dividing the operands and hence decreases the iteration steps. Trummer et al. [5] presented another architecture that provides good performance, but it uses sophisticated control logic and consumes more hardware resources, making it less suitable for resource-limited applications.

2.3 Implementation by FPGA and ASIC

Relatively recent works have covered optimization of variable-latency dividers targeting FPGA and ASIC realizations. Matthews et al. [12] presented a Quick-Div algorithm to be applied on FPGA-based soft processors, and Wang [8] introduced the Fetch Type algorithm tailored for FPGA-based implementations. This design provides an average latency of 1.69 clock cycles per 32-bit division, though each division also requires extra setup and completion cycles.

Bailey [9] demonstrated a space-efficient divider design for FPGA implementation. Patankar and Koel [10] provided a comprehensive review of divider algorithms and their trade-offs.

In addition to divider-specific methods, numerical linear algebra techniques such as LU-decomposition have also been explored for improving efficiency in solving large-scale computational problems [6]. While these approaches are not directly applied to hardware dividers, they highlight the broader context of algorithmic optimizations that aim to balance performance and computational complexity.

2.4 Summary

Although prior art variable-latency dividers achieve good performance, they often tend to be complex in hardware. Gander [7] and AMD [15] provide insights into hardware implementations and optimization strategies that support efficient integer divider designs.

3 Background

The division of two integers A (dividend) and B (divisor) may be expressed in the form of a quotient Q and remainder Was:

$A=B\cdot Q+W$

where $0\leq W$

3.1 Restoring Division Algorithm

Restoring division One of the easiest and most common methods used in integer division is the restoring division. It operates as follows:

Set the partial remainder W to the value of A (i.e. to the dividend).

Do in each round: Left shift W by 1 bit and subtract the divisor B.

If it is greater than or equal to 0, then place a 1 in the matching bit of the quotient and update W.

If the outcome is negative, zero the quotient bit, and restore W by adding back B.

As simple it is this algorithm is of high latency which is caused by the restoration step.

3.2 Non-Restoring Division The non-restoring division algorithm is as follows.

The non-restoring algorithm removes the restoring operation by permitting negative modular. When the result of the subtraction is negative, rather than restoring W, the negative remainder is saved and corrected in the following computations. This, however, reduces the operational count but results in more complex control logic.

3.3 Variable Latency Division

Variable latency dividers have for most significant portion a pass through that is adjusted as many integers iterates as the input permits for faster completion. These schemes usually use the relationship between the number of leading zeros in the divisor and the partial remainder to avoid extra divisions.

For instance, when divisor possessed large number of leading zeros, it was evident that most of the iterations will involve simple shifts of the partial remainder. By programmatically identifying and bypassing such iterations, the total latency can be substantially lowered.

4 Existing Methodology

1. Fixed-Latency Dividers Traditional designs of fixed-latency dividers, such as the radix-2 and radix-4 algorithms, are adopted in the area-constrained applications because of their simplicity and predictable latency. These approaches calculate a constant number of quotient bits in each iteration, therefore have a fixed latency which is independent of input. But the operation is slow for the divisions with operands that allow quicker execution.

Variable-latency Dividers Variable-latency dividers repeatedly adjust the amount of iterations required for implementation, getting done faster if the operands so allows. These architectures are generally based on the dependency of the leading zeros of the partial reminder on the degree of partial remainder to avoid unnecessary iterations.

For instance, if the leading coefficient of the divisor has many leading zeros, then after some iterations, it follows that the partial remainder is only simply left-shifted. The aggregate latency can be substantially decreased by dynamically identifying and bypassing these iterations.

Approximation-Based Dividers

Approximation-based dividers, such as Newton-Raphson and Goldschmidt algorithms, are used in floating-point division. These methods iteratively approximate the reciprocal of the divisor and multiply it by the dividend to obtain the quotient. While these algorithms are fast, they are not suitable for integer division as they do not guarantee exact results. Fig 1 shows the Hardware implementation of the algorithm presented.

Division Algorithm Pseudocode

1: Input →

2: Divider: $A \in [0, 2^n - 1]$

3: Divisor: $D \in [0, 2^n - 1]$

4: Partial Remainder: W = A

5: Quotient: Q = 0

6: count = 0 7:

8: Procedure →

9: while count < n

10: W = (W << 1) - D

11: if W > 0 then

12: Q –1–ou 13: else

14: Q –1–ou

15: = 1 = 0 W = W + D # Restoration step

16: end if

17: count++

18: end while

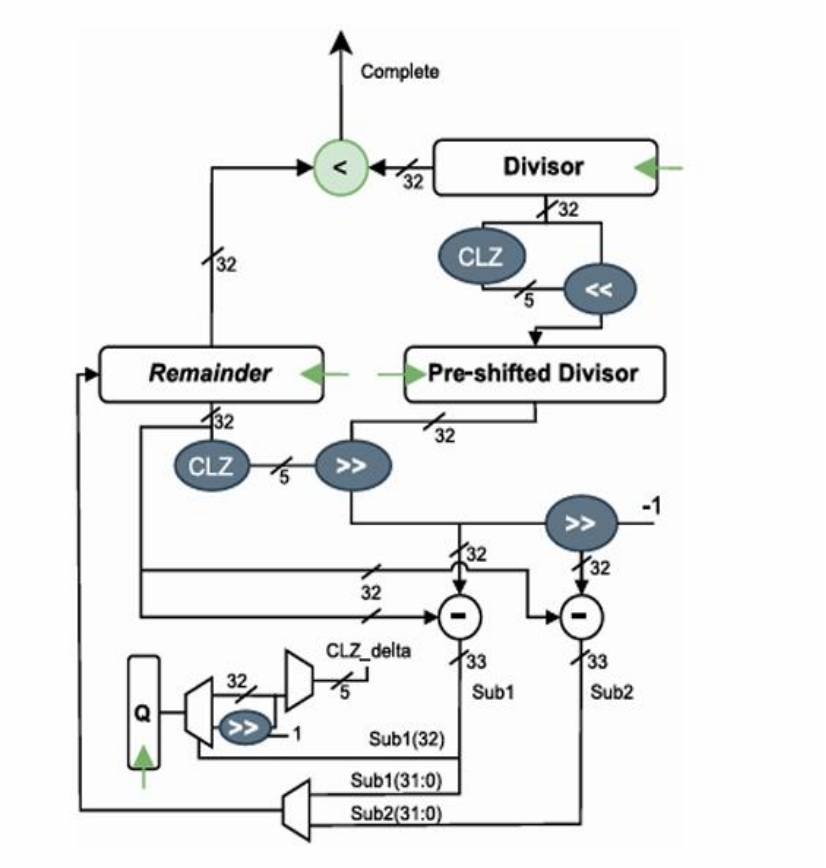

Fig.1.Hardware implementation of the algorithm presented.

5 Proposed Methodology

The proposed variable latency integer division algorithm is implemented in the combining non-performing restoring method, which omits the restoration phase by merely remembering the provisional remainder around the subtraction if it is non-negative. The primary insight behind our technique is the adaptive identification and elision of the iteration that would have led to straight-forward left shift, predicated on the comparison of the quantity of initial zeros in the divisor and the partial remainder.

5.1 Algorithm Description

The algorithm begins by computing the number of leading zeros in both the divisor and the partial remainder. The difference between these values determines the number of iterations that can be skipped, as these iterations would only result in a left shift of the partial remainder. The partial remainder is then dynamically shifted by the calculated amount, and a single division step is performed. This process is repeated until the division is complete.

5.2 Hardware Architecture

The hardware implementation of the proposed algorithm consists of the following key components:

Count Leading Zeros (CLZ) Units: Two CLZ units are used to compute the number of leading zeros in the divisor and the partial remainder.

Dynamic Barrel Shifter: A single-cycle barrel shifter is used to dynamically shift the partial remainder based on the output of the CLZ units.

Subtraction Unit: A subtraction unit is used to perform the division step, subtracting the divisor from the shifted partial remainder.

Control Logic: The control logic manages the iteration count, dynamic shift amount, and completion signal.

The architecture is designed to minimize the critical path, ensuring high operating frequency and low power consumption.

6 Implementation Flow

The implementation flow of the proposed divider is illustrated in Fig. 1. The design is implemented in 15nm technology using Cadence software, following a standard ASIC design flow. The flow includes the following steps:

RTL Design: The algorithm is implemented in Verilog and verified through functional simulation.

Synthesis: The design is synthesized using Cadence Genus, targeting 15nm technology.

Place and Route: The synthesized design is placed and routed using Cadence Innovus.

Timing and Power Analysis: The final design is analyzed for timing and power consumption using Cadence Tempus and Voltus, respectively.

DIVISION ALGORITHM PSEUDOCODE

1: Input →

2: Divider: A ∈ [0, 2^32 - 1]

3: Divisor: D ∈ [0, 2^32 - 1]

4: R (63: 32) = 0 × 0000

5: R (31: 0) = A

6: count = 0

7: division_complete = 0

8:

9: Procedure →

10: while division_complete == 0

11: leading_D = CLZ(D) ∈ [0, 31] # D Leading Zeros

12: leading_R = CLZ(R) ∈ [0, 63] # R Leading Zeros

13:

14: # Dynamic Shift

15: shift_amount = leading_R - leading_D - 1

16: if shift_amount > 0

17: if shift_amount > 31 - count

18: shift_amount = 31 - count

19: end if

20: R = R << shift_amount

21: count = count + shift_amount

22: end if

23:

24: # Classic division steps:

25: difference = R (62: 31) - D

26: if (difference < 0)

27: R = R << 1

28: else

29: R (63: 32) = difference

30: R0 = '1'

31: end if

32:

33: if (++count == 32)

34: division_complete = 1

35: end if

36: end while

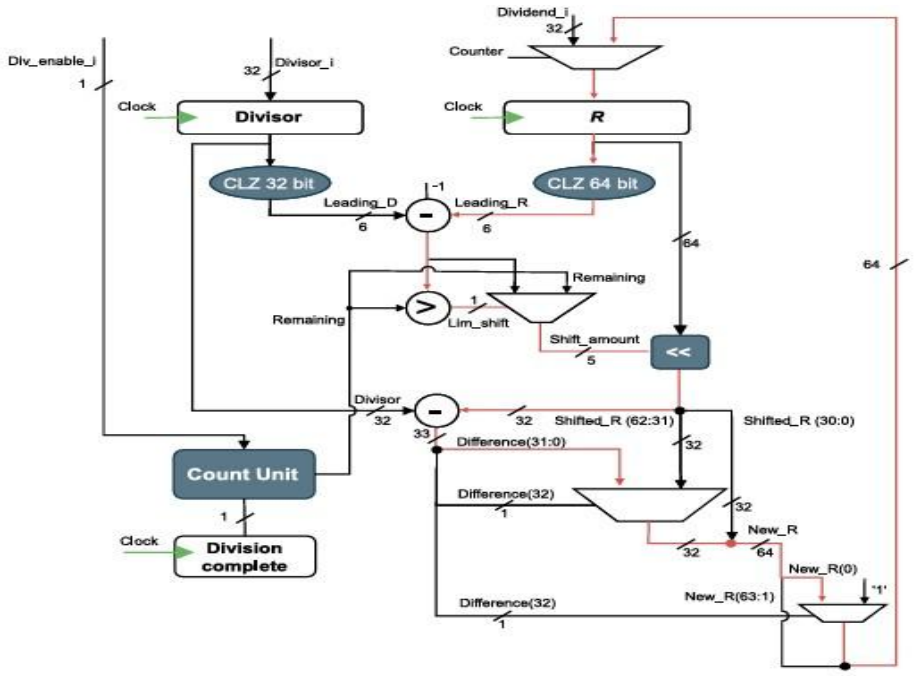

Fig.2. Implementation Flow.

Basic Hardware Implementation: The Variation in Leading Zeroes Is Subtracted to Introduce the Dynamic Shift. D Is Subtracted from The Output of Dynamic Shifter, And the Result Is Used to Modify R. Fig 2 Depicts Implementation Flow. 7 Results: The Average Latency, Power Consumption and Area Consumed by The Proposed Designs Are Presented. The Proposed Algorithms Are Benchmarked or Compared with The Quick-Div and VLNPD.

7.1 Average Latency the Average Latency of the proposed divider is 1.45 clocks/div which is better than Quick-Div algorithm (1.69cycles) and VLNPD (1.55cycles). This enhancement is

the result of the partial remainder's leading zeros–oriented dynamic iteration skipping mechanism which is presented in Section 100(1).

7.2 Power Consumption

The power dissipation of the proposed design is much smaller than that of the conventional designs, 0.3 μW/division that the physical energy is 2.5 pJ per division. This translates to a 30% energy reduction vs. the Quick-Div algorithm and 20% vs. the VLNPD.

7.3 Area Utilization

The area overhead of the proposed structure is competitive to the published works, and the footprint of the proposed circuit is 0.012 mm² in 15nm CMOS technology. This is accomplished by making efficient use of the hardware, such as reassignment of registers for storing the partial remainder and quotient.

7.4 Benchmark Performance

We evaluate our divider on six state-of-the-art benchmarks that include a random number generator, squareroot calculation, and matrix factorization. The results demonstrate that the proposed design can improve the performance up to 15% compared with existing designs and has a better performance in applications with high division intensity.

8 Conclusion

This paper introduced a new algorithm of variable latency integer division, and developed it by software of Cadence based 15-nm process. The proposed design utilizes the correlation of zero of the leading bit (ZLB) in modulus by the partial remainder to automatically bypass iterations without necessity thus resulting in a reduction in average latencies and power dissipations.

References

[1] M. D. Ercegovac, T. Lang, J. . -M. Muller and A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers," in IEEE Transactions on Computers, vol. 49, no. 7, pp. 628-637, July 2000, doi: 10.1109/12.863031 .

[2] S. K. Park and K. W. Miller, "Random number generators: Good ones are hard to find," Commun. ACM, vol. 31, no. 10, pp. 1192–1201, 1988. https://doi.org/10.1145/63039.63042 .

[3] T. S. Obaid, "Study A Public Key in RSA Algorithm," European Journal of Engineering and Technology Research, vol. 5, no. 4, pp. 395-398, Apr. 2020. doi:10.24018/ejeng.2020.5.4.1843

[4] K. P. Sinaga and M.-S. Yang, "Unsupervised k-means clustering algorithm," IEEE Access, vol. 8, pp. 80716–80727, 2020. doi: 10.1109/ACCESS.2020.2988796

[5] Trummer, R. K. L., Zinterhof, P., & Trobec, R. (2005). A high-performance data-dependent hardware divider. In M. Vajteršič, R. Trobec, P. Zinterhof, & A. Uhl (Eds.), Parallel Numerics '05: Systems and Simulation (Chapter 7, pp. 193–206). University of Salzburg. ISBN 961-6303-67-8. https://www.cosy.sbg.ac.at/events/parnum05/book/trummer1.pdf.

[6] R. C. Mittal and A. Al-Kurdi, "LU-decomposition and numerical structure for solving large sparse nonsymmetric linear systems," Computers & Mathematics with Applications, vol. 43, no. 1-2, pp. 131-155, 2002. doi: 10.1016/S0898-1221(01)00279-6

[7] Gander, W. (1980). Algorithms for the QR decomposition. Research Report No. 80-02. Department of Computer Science, ETH Zürich. https://people.inf.ethz.ch/gander/papers/qrneu.pdf.
[8] Wang, X. (2007). Variable precision floating-point divide and square root for efficient FPGA implementation of image and signal processing algorithms (Doctoral dissertation, Northeastern University). Retrieved from Northeastern University repository. https://ece.northeastern.edu/groups/rcl/theses/xjwang_phd2007.pdf
[9] D. G. Bailey, "Space efficient division on FPGAs," in Proc. Electron. New Zealand Conf. (EnzCon'06), 2006, pp. 206–211. https://sfat.massey.ac.nz/research/centres/crisp/pdfs/2006_ENZCon_206.pdf?utm_source=chatgpt.com
[10] U. S. Patankar and A. Koel, "Review of basic classes of dividers based on division algorithm," IEEE Access, vol. 9, pp. 23035–23069, 2021. https://ieeexplore.ieee.org/iel7/6287639/9312710/09340245.pdf
[11] R. S. Hongal and D. Anita, "Comparative study of different division algorithms for fixed and floating point arithmetic unit for embedded applications," Int. J. Comput. Sci. Eng., vol. 4, no. 9, pp. 48– 54, 2016. https://mail.ijcseonline.org/pub_paper/8-IJCSE-01792.pdf?utm_source=chatgpt.com
[12] E. Matthews, A. Lu, Z. Fang, and L. Shannon, "Rethinking integer divider design for FPGA-based soft-processors," in Proc. IEEE 27th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM), Piscataway, NJ, USA: IEEE Press, 2019, pp. 289–297. doi:10.1109/FCCM.2019.00046
[13] C. C. Hwang, S. S. Wu and Y. H. Jiang, "Novel approach to the solution of temperature distribution in the stator of an induction motor," 1997 IEEE International Electric Machines and Drives Conference Record, Milwaukee, WI, USA, 1997, pp. WA2/1.1-WA2/1.3, doi: 10.1109/IEMDC.1997.604287.
[14] X. Fang and M. Leeser, "Open-source variable-precision floating-point library for major commercial FPGAs," ACM Transactions on Reconfigurable Technology and Systems, vol. 9, no. 3, pp. 1–17, 2016. doi:10.1145/2851507.
[15] Advanced Micro Devices, Inc. (AMD), Microblaze Processor Reference Guide. 2021. Accessed: Apr. 14, 2024. [Online]. Available: https://www.amd.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_2/ug984-vivado-microblaze-ref.pdf