ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 1, 2024

DIGITAL DIVISION ALGORITHMS FOR EFFICIENT EXECUTION ON INTEGRATED CIRCUITS

Anatoliy Obshta[1], Volodymyr Khoma[2], Andrii Prokopchuk[1]

[1]*Lviv Polytechnic National University, 12, Bandera Str, Lviv, 79013,Ukraine.*
[2]*Opole University of Technology, Proszkowska 76, 45-758 Opole, Poland.*
Authors' e-mails: *anatolii.f.obshta@lpnu.ua, v.khoma@po.edu.pl, andrii.prokopchuk.mkisp.2022@lpnu.ua*

https://doi.org/10.23939/acps2024.01.046

Submitted on 09.04.2024

***Abstract*: In this paper, we analyse division algorithms for use on chips and propose the implementation of an optimal divider for these chips. By "optimal", we refer to an algorithm that meets the following criteria: space efficiency – which involves minimizing resource utilization on the IC's die area; speed efficiency – the algorithm's processing time (measured in n clock cycles); power efficiency – power consumption of the divider; implementation time – time for implementation of the algorithm using HDL. The chosen algorithm should strike a balance between space efficiency and processing speed, ensuring the efficient use of hardware resources while delivering swift computational results. The ultimate goal is to create a division module that aligns seamlessly with the integrated circuit's architecture, catering to computational efficiency and resource constraints.**

Index Terms: FPGA, ASIC, Digital Divider, Digit Recurrence, Functional Iteration.

I. INTRODUCTION

Arithmetic operations, such as addition, subtraction, multiplication, and division, play a fundamental role in computational systems. Although these operations have long been studied and implemented, the performance of the division operation is one of the most complex and resource-intensive tasks. As the advancements in chip manufacturing techniques and technologies continue to push the boundaries of Moore's law, it has become increasingly essential to address the challenges of realizing efficient division methods.

One of the major challenges in implementing division is that the result is an approximate value. While other arithmetic operations can give a specific, precise result, division is performed approximately. The result of dividing two integers, in general, is a rational or even irrational number, which in some cases, when represented in the binary system, cannot be accurately expressed by a limited number of bits.

Another challenge in developing custom solutions on ICs is that most synthesizers do not support the division operation. The only permitted solutions are division by a power of 2 or if the divisor is a constant. In all other cases, the synthesizer will produce an error.

This work aims to develop a reconfigurable divider for low-power electronic devices, optimized for the implementation area.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

Due to the described issues, in Table 1, many research works are aimed at finding an optimal digital division algorithm for integrated circuits that would satisfy fundamental criteria such as space, speed, and energy efficiency.

Table 1

Supported Expressions for Verilog Language

Expression	Symbol	Status
Concatenation	{}	Supported
Replication	{{}}	Supported
Arithmetic	+,-,*,**	Supported
Division	/	Supported only if the second operand is a power of 2, or both operands are constant.
Modulus	%	Supported only if the second operand is a power of 2.
Addition	+	Supported
Subtraction	-	Supported
Multiplication	*	Supported

Furthermore, methods for optimizing existing algorithms are a sought-after and popular topic of research work.

The division operation is critical for certain fields, for example, soft processors [1], digital signal processing [2], machine learning [3], cryptographic [4], or arithmetical logical units [5].

Among the main directions of research and studies on this topic, the following can be highlighted:

1) reviews of existing division algorithms, highlighting their advantages and disadvantages;
2) enhancement of a specific division method for a specific task;
3) verification and evaluation of the efficiency of existing algorithms.

Some of the newest works on the classification and detailed review of digital division algorithms can be considered in papers [6] and [7]. Division algorithms can be divided into five classes: digit recurrence, functional iteration, very high radix, look-up table, and variable latency. Many practical division algorithms are hybrids of several of these classes. In the following work [8], clearer examples were provided for the use of specific classes of dividers depending on the specific tasks.

Significant emphasis is placed on the digit recurrence class among the works aimed at improving existing division algorithms. In most cases, this is related to the fact that the division operation is critical for low-power integrated circuits, where there are limitations on the number of logic gates, area, and energy efficiency.

As it is described below, this class of digital divisors best meets the requirements outlined above. Improvements to existing algorithms began in the last century with the development of a new division method known as SRT. The main area of application of the SRT algorithm is general-purpose processors, which are commonly used in personal computers, FPGA systems, and ASIC processors. After the discovery of this new method, an iterative process of improving various parameters began. In [9], the authors managed to build bit selection tables for SRT division and square root extraction with a given radius and redundancy. In [10], the authors managed to reduce the number of LUTs used to 36% when implementing the SRT division algorithm for fast Fourier transform. One of the ideas was also the development of a 10-bit divider, presented in [11] and [12], based on the SRT algorithm, which demonstrated improved performance in terms of both implementation area and speed. The proposed algorithm in [13] for IoT applications is based on the parallel execution of different steps to shorten the time-critical path and the use of fuzzy logic to solve the overlap problem in the selection of the coefficient; hence, the reduced one inverter can be removed by using a new quotient digit converter. Another technique is described in [15], where the author implements a 16-bit divider with a speculated quotient digit that is used to simultaneously compute the two possible partial remainders for the next step while the quotient digit is being corrected. This process does not affect the overall speed hence speeding up the division process. A different idea to speed up the division process is described in [16] related to architecture modification by pipelining the divider. In one of the most recent works [17], the author developed a new divider based on the digital recurrence class, which utilizes the Baudhayan-Pythagoras triplet method. They managed to achieve a smaller implementation area while increasing the speed of the division process. In another work [18], the author proposed to abandon the normalization of the divisor, and hence, reduce the required area-consuming leading to one (or zero) detection nor shifts of variable amount.

For the functional iteration class, in the work [19], a modified Goldschmidt algorithm was proposed, which shows a significant improvement in area utilization, by 20-40% when compared to the implementation based on the conventional Goldschmidt algorithm. In our opinion, to accelerate the division operation, the work presented in [20] can be considered, where the authors managed to increase the speed of the multiplication operation by reducing the number of non-zero digits of the multiplier.

In the case of the look-up table class, the authors propose solutions that use small lookup tables, which are well-matched with the hardware resources of an FPGA [21].

From the analysis of [22] work on evaluating the effectiveness of algorithms, it can be concluded that there are advantages in terms of the scope of implementation and energy efficiency for digital divisors belonging to the class of recurrent digits. Using simple arithmetic operations like addition, subtraction, and shifting makes it possible to utilize on-chip resources most efficiently, albeit at the cost of speed.

III. SCOPE OF WORK AND OBJECTIVES

Develop a reconfigurable divider optimised for the implementation domain, suitable for FPGAs (such as Xilinx, Altera, Lattice, etc.) and ASICs.

IV. DIVISION ALGORITHMS BACKGROUND

Based on the method of conversion, we can distinguish division algorithms in the following classes that are shown in Fig. 1. Digit Recurrence (1); Functional Iteration (2); Look-up table (3).

Based on hardware architecture, we can classify types of dividers as: serial or sequential type(1); parallel type (2); pipelined type (3). Based on performance, we can classify types of dividers as: slow type (1); and fast type (2). Based on execution, we can classify types of dividers as: iterative subtraction type (1); predictive type (2).

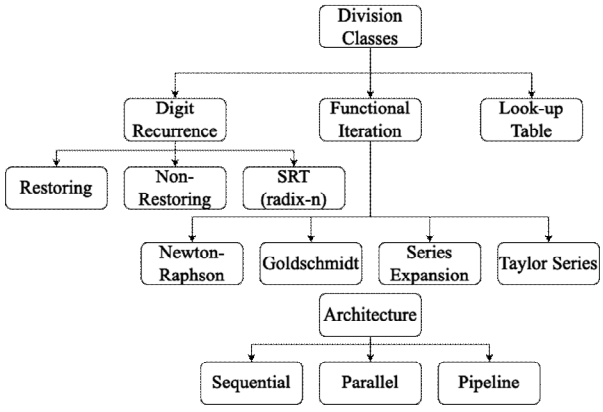

Fig. 1. The distribution of different division algorithms

V. DIGIT RECURRENCE CLASS

One of the simplest and most widely used classes of digital division is digit recurrence. In the beginning, this class became commonly used due to the limited capabilities of programmable logic devices. The main idea of this division is simply to subtract the divisor from the dividend cyclically which produces quotient bits in sequence. This is the iterative type of division since it performs repeated subtraction. This approach generates from 1 to n of quotient bits per iteration. The main reason for its widespread popularity in usage is due to its set of simple arithmetic operations such as addition, subtraction, shifting, multiplication, etc.

This class of division algorithm mainly covers three types of dividers: restoring (1); non-restoring (2); SRT (radix n) (3).

A. RESTORING ALGORITHM

The Restoring algorithm has similarities with the long-division method. The fundamental concept of this algorithm involves a repetitive process of shifting and subtracting the divisor from the dividend. The detailed flowchart of division using the Restoring algorithm is shown in Fig. 2, where: n – number of bits, MSB – most significant bit, Q0 – quotient bit calculated in each iteration, Q – final result.

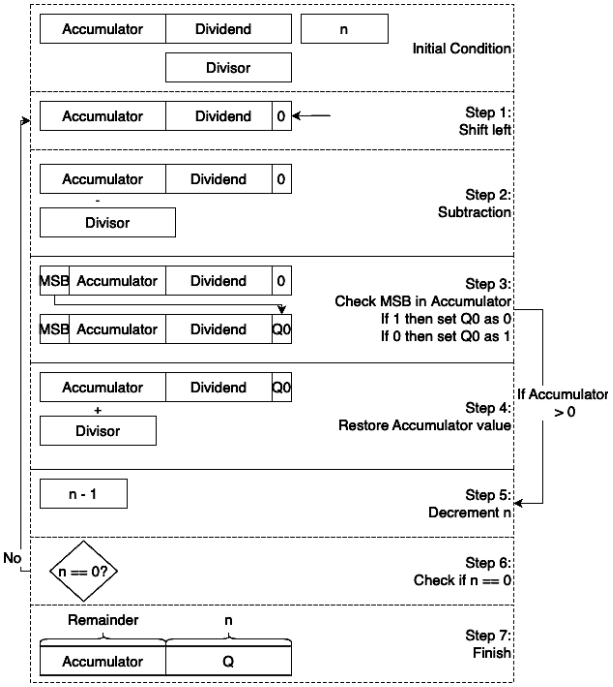

Fig. 2. Flowchart of Restoring Division Algorithm

The steps to achieve the Restoring division algorithm are: (1) select initial values for the divisor, dividend, partial remainder/accumulator, and the number of bits (n); (2) shift dividend left by 1; (3) subtract the divisor from the partial remainder/accumulator, and the result is stored in the partial remainder/accumulator; (4) check for the most significant bit of the partial remainder/accumulator; if 0, then the least significant bit of Q is set to 1; otherwise, the least significant bit of Q is set to 0, and the value of the partial remainder/accumulator is restored to the value before the subtraction; reduce the value of n by one (1); continue iterations until we get a value of n=0 (2).

The main drawback of this algorithm is that it calculates only 1 quotient bit during each cycle, which affects the overall speed of the algorithm. Additionally, a crucial requirement for its implementation is that the partial remainder must always be positive or equal to 0, which necessitates an additional addition operation in the case of a negative remainder. The name of the algorithm, "Restoring," originates from the need to restore the partial remainder in each cycle. However, this algorithm requires minimal effort in terms of implementation. Moreover, its simplicity is an advantage, as it positively impacts the utilization area of the integrated circuit.

B. NON-RESTORING ALGORITHM

This algorithm is very similar to the Restoring division algorithm as it requires an iterative process of shifting, subtracting, and adding. The main difference is that there is no longer a need to restore the partial remainder. This is achieved by performing a check on the partial remainder during the last cycle (n = 0), and if it is negative, restoring it with just one addition operation. The detailed division that utilizes the Non-Restoring algorithm is shown in Fig. 3.

t and requires more effort during implementation.

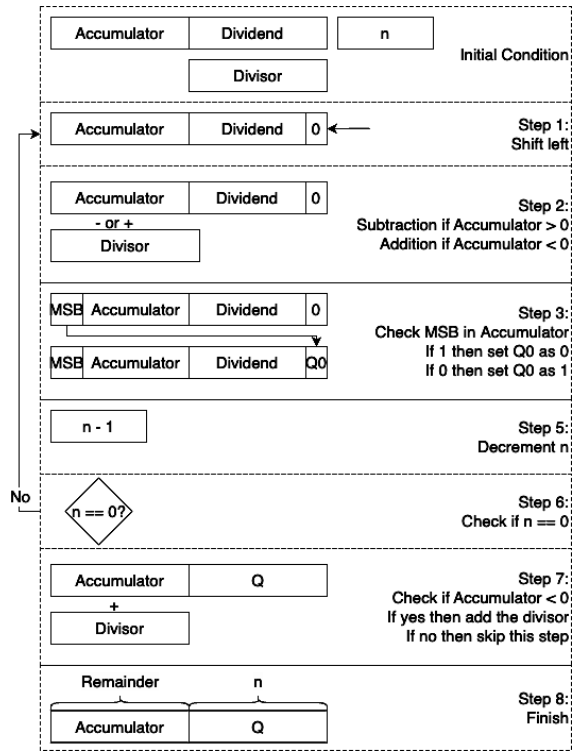

Fig. 3. Flowchart of Non-Restoring Division Algorithm

The steps to achieve a Non-Restoring division algorithm are: (1) select initial values for the divisor, dividend, partial remainder/accumulator, and the number of bits (n); (2) shift dividend left by 1; (3) subtract (at first iteration) / add divisor from/to the partial remainder/accumulator and the result is stored in the partial remainder/accumulator; (4) check for the most significant bit of the partial remainder/accumulator; if 0, then the least significant bit of Q is set to 1; otherwise, the least significant bit of Q is set to 0; (5) reduce the value of n by one; (6) continue iterations until we get a value of n=0; (7) check partial remainder/accumulator. If < 0 then restore partial remainder.

The main advantage of this algorithm is its higher speed compared to the Restoring algorithm, as well as a smaller implementation area. However, it still retains the limitation of calculating only 1 quotient bi

C. SRT ALGORITHM (RADIX-N)

The SRT algorithm is the most efficient among those mentioned above. The main advantage of this algorithm is that it calculates more than 1 quotient bit (when the radix is higher than 2) in a single cycle, making it the fastest algorithm among the digit-recurrent classes. The detailed division using the SRT algorithm is shown in Fig. 4.

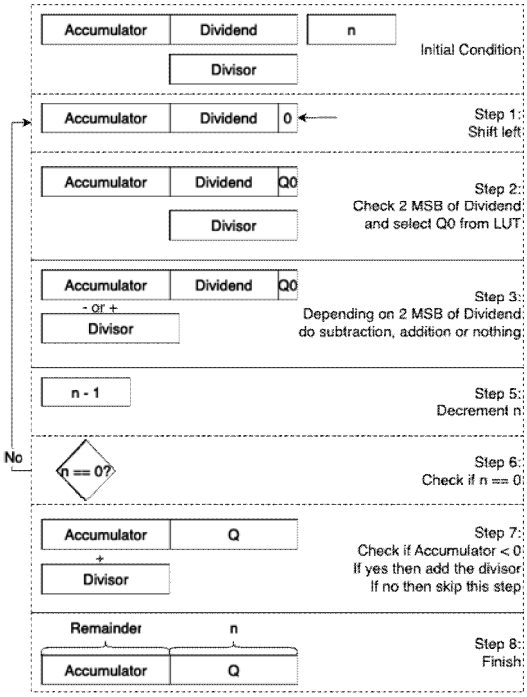

Fig. 4. Flowchart of Radix-2 SRT Division Algorithm

The steps to achieve the SRT division algorithm are: (1) select initial values for the divisor, dividend, partial remainder/accumulator, and the number of bits (n); (2) shift dividend left by radix/2; (3) check for the most significant bit of Dividend and select quotient bits from LUT; (4) check for the most significant bit of the partial remainder/accumulator; depending on that, do subtraction, addition, or skip this step; (5) reduce the value of n by one; (6) continue iterations until we get a value of n=0; (7) check partial remainder/accumulator. If < 0 then restore partial remainder.

This algorithm is the most complex in terms of development among the digit recurrent class. The complexity lies in the need to create a Look-up table (LUT) for a specific radix-n. The larger the radix, the bigger and more complex LUT is required for selecting quotient bits. However, this allows computing more bits in a single cycle. There are many nuances in implementing this algorithm, namely: (1) choice of radix; (2) choice of remainder representation; (3) choice of quotient digit set.

Depending on these choices, the speed, implementation area, and latency of the algorithm will be affected.

VI. FUNCTIONAL ITERATION CLASS

A fundamentally different class of division is the functional iteration class. This division method uses multiplication operations instead of additions and subtractions as in the digit-recurrence class. The result is computed through iterative approximations, allowing for the calculation of multiple quotient bits per iteration and significantly increasing the speed of the division process. Utilizing multiplication for the functional iterative dividers leads to the increased implementation area, prompting the use of smaller-size multipliers. Consequently, this approach becomes more intricate compared to straightforward digit recurrence dividers. This category of dividers suffers from a significant drawback - the quotient results are imprecise due to the direct rounding of approximate solution values instead of infinite precise values. The functional iteration-based algorithm is efficient in performing division but cannot guarantee exact results consistently. Functional iteration dividers work on the series expansion phenomenon.

This class of division algorithm mainly covers the following types of dividers: (1) Newton–Raphson algorithm (NRA); (2) Goldschmidt algorithm (GSA); (3) Series expansion algorithm (SEA); (4) Taylor series algorithm (TSEA).

A. NEWTON-RAPHSON ALGORITHM

It is acknowledged that the outcome of the division process can be expressed as a single term of the product between the dividend and anti-divisor (reciprocal). Computing the anti-divisor in the Newton-Raphson algorithm relies on the choice of the priming function, which determines its root at the anti-divisor, a parameter that typically has multiple values. The accuracy of the quotient's convergence depends on which root is selected, potentially leading to errors in the division process and generating additional overhead if the selected root overshoots the true quotient. Improving accuracy can be achieved by selecting the appropriate root initially.

The detailed figure of division using the Newton-Raphson algorithm is shown in Fig. 5, where: i –

number of iterations, X0 – initial value of approximation, Xi – current value of approximation, Xi+1 – next value of approximation

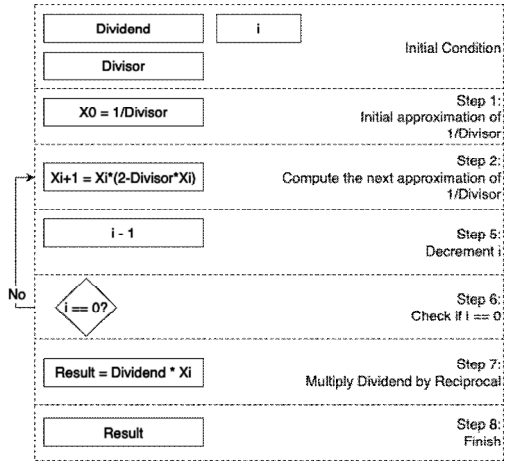

Fig. 5. Flowchart of Newton-Raphson Division Algorithm

The steps to achieve the Newton-Raphson algorithm are: (1) select initial values for divisor, dividend, and the number of iterations (i); (2) perform an initial approximation of 1/Divisor; (3) compute the next approximation using the formula; (4) reduce the value of i by one; (5) continue iterations until we get a value of i=0; (6) multiply the Dividend by the calculated reciprocal (anti-divisor).

There are also other division methods within this class, such as the Goldschmidt, Series expansion, and Taylor series algorithms. Certainly, they have specific differences and characteristics that help enhance speed and reduce delays. However, all of them share a fundamental idea - approximating the reciprocal of the divisor and subsequently replacing the division operation with multiplication.

VII. LOOK-UP TABLE

Another class of division is Look-up tables. The concept here is that the division result values can be pre-stored in this table and processed when two operands are input into the table. This method allows for an incredibly fast output of results due to the extensive implementation space. However, it typically lacks precision (as the accuracy is chosen during development and depends on the table size).

The figure of division using the Look-up table algorithm is shown in Fig. 6.

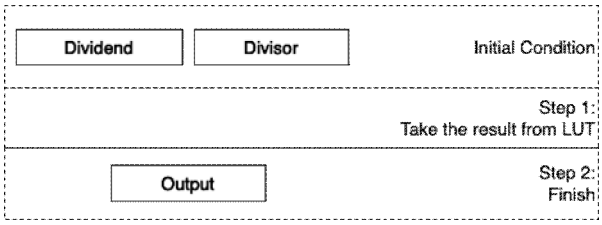

Fig. 6. Flowchart of Look-up table Division Algorithm

VIII. ANALYSIS OF RESULTS AND PROPOSAL FOR DIVIDER IMPLEMENTATION

The investigation into existing classes of dividers based on the division algorithm has yielded valuable insights into their strengths and weaknesses. Among the classes studied, the digit recurrence class emerges as the most prevalent approach for implementing digital dividers. Notably, this method can generate radix/2 digits per iteration, making it an optimal choice for dividers where space efficiency is a critical concern.

For scenarios involving the division of smaller numbers with an emphasis on speed, the lookup table method proves to be a highly effective solution. This approach excels in swiftly handling computations involving modest-sized operands. However, as the magnitude of the numbers to be divided increases, a shift towards the functional iteration class of dividers becomes more advisable.

The functional iteration class stands out as the fastest algorithm among all those considered in this study. Its remarkable computational speed, however, comes at the cost of increased space and timing requirements for proper implementation. This implies that while it may be the most efficient option for demanding computational tasks, careful consideration of resource allocation is necessary to ensure its optimal performance.

Table 2

Resource utilization of different division architecture in the Spartan-3E FPGA

Division Architecture	Resources		
	FFs	LUTs	MUXs
Restoring	21	31	1
Non-Restoring	22	25	1
SRT	31	59	1
SRT with CSA	40	100	1
Goldschmidt	22	197	1
Xilinx Divider core	224	79	1
Matlab system generator	436	143	1

Table 3

Resource utilization of different division architecture in the Spartan-6 FPGA

Division Architecture	Resources			
	FFs	LUTs	MUXs	DSP Slices
Restoring	21	31	12	0
Non-Restoring	21	25	8	0
SRT	37	38	12	0
SRT with CSA	40	84	16	0
Goldschmidt	21	22	0	2
Xilinx Divider core	224	152	104	0
Matlab system generator	436	296	192	0

By analyzing the research results presented in articles [22] and [23], it can be concluded that the non-restoring division algorithm is optimal in cases where space, timing, and power efficiency are crucial, with minimal development time. The results of the logic resource utilization of 8-bit divider architectures in the Spartan-3E and Spartan-6 FPGA are shown in Table 2 and Table 3 respectively. The critical path delay in each division architecture for both the Spartan-3E and the Spartan-6 FPGAs is shown in Table 4.

Table 4

Delay in the Spartan-3E and Spartan-6 FPGAs

Division Architecture	Device	
	Spartan-3E	Spartan-6
Restoring	4,992 ns	2,906 ns
Non-Restoring	4,719 ns	3,314 ns
SRT	6,116 ns	3,258 ns
SRT with CSA	7,112 ns	5,016 ns
Goldschmidt	13,791 ns	7,998 ns
Xilinx Divider core	3,957 ns	2,626 ns
Matlab system generator	3,692 ns	3,150 ns

After investigation of existing algorithms and implementations of dividers based on them, certain drawbacks in current implementations become apparent. The designed dividers have a fixed input data size, do not scale well, and may not be suitable for various platforms.

The proposal is to develop a reconfigurable divider based on the non-restoring division algorithm. This divider will have the ability to easily integrate into any synchronous project on IC (FPGA / ASIC) by using parameterized input data, and the selection of bit width can be determined during the synthesis phase.

Additionally, one of the advantages of this divider will be the implementation of a finite state machine, which will ensure the sequential execution of division in multiple stages. In cases where optimization of used space and low-power devices is a concern, this will reduce the number of components used, directly impacting area and power consumption.

In terms of optimizing the development time T_{st_d} of a digital divider based on parameterized inputs/outputs bit width, the following equations can be used:

$$T_{st\_d} = T_{des} + T_{ver} \quad , \tag{1}$$

$$T_{rec\_fig\_div} = \frac{T_{st\_d} + T_{mod}}{N} \quad . \tag{2}$$

Equation (1) represents the time required to implement a fixed bit width divider, where T_{des} is the time for RTL implementation using HDL, and T_{ver} is the time required to verify the divider. In case you need to change the bit width of the input data for another project, you will have to invest time in RTL development and verification again.

Equation (2) represents the time required to implement a reconfigurable divider with paameterized input/output bit width, where T_{st_d} is the time required for the implementation of a standard divider with fixed bit width, T_{mod} is the time to implement parameterized bit width feature, N is the number of divider re-usage ($N = 1$ean divider implementation stage). In this case, there is no longer a need to repeat the development and verification stages of the divider when it needs to be used with a different input data width. As evident from equation (2), implementing this feature may require additional time, but it becomes advantageous when the divider is re-used at least once (3). If we assume that X is the time required for standard divider implementation, and $X/4$ is the time required for modifications (this parameter was determined using approximate values corresponding to the number of changes in the RTL code) then it can be argued that the development time of this divider is 1.6 times less than in the standard approach (4):

$$T_{rec\_fig\_div} = \frac{X + \frac{X}{4}}{2} = 0.625X \quad , \tag{3}$$

$$T_{boost} = \frac{T_{st\_d}}{T_{rec\_fig\_div}} = \frac{X}{0.625X} = 1.6 \quad . \tag{4}$$

Our reconfigurable digital divider, which can change the bit width of input/output data, allows for flexible use according to user requirements and easy integration into designs. With this approach, we plan to speed up the development of integrated circuits. The developed digital divider should reduce the development time by at least 1.5 times.

The requirements for a digital divider are as follows: (1) the input/output bit depth must be parameterised; (2) it must be implemented using the most efficient algorithm in terms of the area of implementation; (3) must be suitable for reusability. This divider will be suitable for FPGAs (such as Xilinx, Altera, Lattice, etc.) and ASICs.

IX. CONCLUSION

Considering these findings, it is imperative to carefully weigh the specific requirements of a given application when selecting a divider implementation. For scenarios where space efficiency is paramount, the digit recurrence class should be the preferred choice.

Conversely, when dealing with smaller operands and speed is of the essence, the lookup table method may be the most suitable option.

For applications involving substantial numbers and demanding computational tasks, the functional iteration class emerges as the clear frontrunner in terms of speed, albeit with a caveat of increased resource allocation. Engineers and developers must conduct a thorough assessment of their project's needs and resources before determining the optimal divider implementation.

As a result of this research, we addressed the issue of implementing the division operation on integrated circuits. We examined existing division algorithms, highlighted their advantages and disadvantages, and

compared hardware resource utilization and algorithm complexity.

The implementation of a reconfigurable divider based on the irreducible division algorithm can be applied in many areas of CPS, where fast and efficient data processing is required, as well as adaptability to changing operating conditions.

In particular, CPS are a critical aspect of cybersecurity, as these systems combine physical processes and computer systems, and the vulnerability of one of the components can lead to serious consequences. The implementation of a reconfigurable divider based on the irreducible division algorithm for chips can be applied to improve cybersecurity in various aspects of CPS: (1) Detection and prevention of network attacks: reconfigurable dividers can be used to distribute traffic in communication networks depending on the attack variables. For example, they can redistribute traffic to prevent network congestion or reduce the impact of malicious data packets. (2) Protection against data leakage: Reconfigurable dividers can be used to encrypt and decrypt data in different parts of a network or system, reducing the risk of sensitive information leakage. (3) Strong access control: Reconfigurable dividers can be used to create dynamic access rules for resources or systems, enabling effective access control and preventing unauthorised use. (4) Vulnerability detection and remediation: reconfigurable dividers can be used to continuously analyse system health and identify potential vulnerabilities or attacks, and to quickly respond and remediate these issues. (5) Dynamic management of computing resources: reconfigurable dividers can optimise the allocation of computing resources in real time, ensuring efficient use of resources and minimising the possibility of exploiting attacks.

These aspects demonstrate that the implementation of a reconfigurable divisor based on the irreducible division algorithm for chips will contribute to improving cybersecurity in various aspects of cyber-physical systems.

References

[1] Matthews E., Lu A., Fang Z., Shannon L., (2019). Rethinking integer divider design for FPGA-based soft-processors. *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 289–297. DOI:https://doi.org/10.1109/FCCM.2019.00046

[2] Sanju Vikasini M.K., Kailath B.J., (2021). 16-bit Modified vedic paravartya divider with quotient in fractions. *IEEE Region 10 Symposium*, pp.1–5. DOI:https://doi.org/10.1109/TENSYMP52854.2021.9551013

[3] Han G., Zhang W., Niu L., Zhang C., Wang Z., (2022). Hardware implementation of approximate fixed-point divider for machine learning optimization algorithm. *IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics,* pp.22–25. DOI:https://doi.org/10.1109/PrimeAsia56064.2022.10104001

[4] Tynymbayev S., Aitkhozhayeva E., Berdibayev R., Gnatyuk S., Okhrimenko T., Namazbayev T., (2019). Development of modular reduction based on the divider by blocking negative remainders for critical cryptographic applications. *IEEE 2nd Ukraine Conference on Electrical and Computer Engineering,* pp.809–812. DOI:https://doi.org/10.1109/UKRCON.2019.8879846

[5] Purohit A.A., Ahmed M.R., Reddy R.V.S., (2020). Design of area optimized arithmetic and logical unit for microcontroller. *IEEE VLSI DEVICE CIRCUIT AND SYSTEM,* pp.335–339. DOI:https://doi.org/10.1109/VLSIDCS47293.2020.9179942

[6] Patankar U.S., Flores M.E., Koel A., (2020). Division algorithms - from past to present chance to improve area time and complexity for digital applications. *IEEE Latin America Electron Devices Conference*, pp. 1–4. DOI:https://doi.org/10.1109/LAEDC49063.2020.9073050

[7] Patankar U.S., Flores M.E., Koel A., (2021). Study of estimation based functional iteration approximation dividers. *IEEE International Conference on Consumer Electronics*, pp. 1–4. DOI: https://doi.org/10.1109/ICCE50685.2021.9427657

[8] Patankar U.S., Flores M.E., Koel A., (2021). Review of basic classes of dividers based on division algorithm. *IEEE Access*, pp. 23035–23069. DOI:https://doi.org/10.1109/ACCESS.2021.3055735

[9] Liu Z., Song X., Wang Z., Wang Y., Zhou J., (2023). Constructing high radix quotient digit selection tables for SRT division and square root. *IEEE Transactions on Computers*, pp. 2111–2119. DOI:https://doi.org/10.1109/TC.2023.3235978

[10] Chouhan M., Raghuvanshi A.S., Muchahary D., (2022). FPGA implementation of high performance and energy efficient Radix-4 based FFT. *Asian Conference on Innovation in Technology*, pp. 1–5. DOI:https://doi.org/10.1109/ASIANCON55314.2022.9908613

[11] Lang T., Nannarelli A., (2007). A radix-10 digit-recurrence division unit: algorithm and architecture. *IEEE Transactions on Computers*, pp. 727–739. DOI:https://doi.org/10.1109/TC.2007.1038

[12] Vazquez A., Antelo E., Montuschi P., (2007). A radix-10 SRT divider based on alternative BCD codings. *International Conference on Computer Design*, pp. 280–287. DOI:https://doi.org/10.1109/ICCD.2007.4601914

[13] Mehta B., Talukdar J., Gajjar S., (2017). High speed SRT divider for intelligent embedded system. International Conference on Soft Computing and Its Engineering Applications, pp. 1–5. DOI:https://doi.org/10.1109/ICSOFTCOMP.2017.8280077

[14] Jun K., Swartzlander E.E., (2012). Modified non-restoring division algorithm with improved delay profile and error correction. *Circuits, Systems and Computers*, pp. 1460–1464. DOI: https://doi.org/10.1109/ACSSC.2012.6489269

[15] Dixit S., Nadeem M., (2017). FPGA accomplishment of a 16-bit divider. *Imperial Journal of Interdisciplinary Research*, pp. 140–143. Available at: https://www.researchgate.net/publication/360588032_FPGA Accomplishment_of_a_16-Bit_Divider (Accessed: 07 November 2023).

[16] Narendra K., Ahmed S., Kumar S., Asha G.H., (2015). FPGA implementation of fixed point integer divider using iterative array structure. *International Journal of Engineering and Technical Research*, pp. 170–179. Available at: https://www.erpublication.org/published paper/JETR031914.pdf (Accessed: 07 November 2023).

[17] Patankar U.S., Flores M.E., Koel A., (2023). A. Novel data dependent divider circuit block implementation for com-

plex division and area critical applications. *Sci Rep* 13, pp.1–27. DOI: https://doi.org/10.1038/s41598-023-28343-3

[18] Takagi N., Kadowaki S., Takagi K., (2005). A hardware algorithm for integer division. *IEEE Symposium on Computer Arithmetic*, pp. 140–146. DOI:https://doi.org/10.1109/ARITH.2005.6

[19] Han K., Tenca A., Tran D., (2009). High-speed floating-point divider with reduced area. *The International Society for Optical Engineering*, pp. 1–8. Available at: https://www.researchgate.net/publication/253273653_High-speed_floating-point_divider_with_reduced_area (Accessed: 07 November 2023).

[20] Korol I., Korol I., (2019). Logical algorithms of the accelerated multiplication with minimum quantity of nonzero digits of the converted multipliers. *Advances in Cyber-Physical Systems,* pp. 25–31. DOI:https://doi.org/10.23939/acps2019.01.025

[21] Ugurdag H.F., De Dinechin F., Gener Y.S., Gören S., Didier L.S., (2017). Hardware division by small integer constants. *IEEE Transactions on Computers*, pp. 2097–2110. DOI:https://doi.org/10.1109/TC.2017.2707488

[22] Mannatungal K.S., Perera M.D.R., (2016). Performance evaluation of division algorithms in FPGA. Proceedings of the *International Research Conference "Medical, Allied Health, Basic and Applied Sciences", pp. 84*–88. Available at: http://ir.kdu.ac.lk/bitstream/handle/345/1170/FAHS017.pdf?isAllowed=y&sequence=1 (Accessed: 07 November 2023).

Anatoliy Obshta - Doctor of Technical Sciences. Speciality: devices and methods for measuring electrical and magnetic quantities. Theme: theory and methods of construction of quality control equipment for conductive materials. He is a professor, author of more than 150 scientific and methodological works. In particular, he is the author of 10 textbooks on various courses in mathematics and computer science and 5 monographs on aggregation-iterative and two-sided methods for solving operator equations

Volodymyr Khoma, received the Ph.D. and the Habilitation degrees in Electrical and Electronic Engineering from the Lviv Polytechnic National University in 1990 and 2001, respectively. He received the title of Professor in the field of Control Engineering in 2003. He is currently employed as a Professor with the Institute of Control Engineering, Opole University of Technology. His scientific in terests include AI in cybersecurity and IoT, digital measurement, and signal processing.

Andrii Prokopchuk received a Bachelor`s degree in Computer Engineering at Lviv Polytechnic National University in 2022. Currently, he is receiving a Master's degree.

He has professional experience working in IT since 2020 and currently working as a digital design engineer at Renesas Electronics (Lviv, Ukraine).