


Article

Research on multiplication routine based on reconfigurable four-valued logic processor

Shanchuan Liao ¹ , Shuang Li ^{1*}, Xingquan Gu ², Luqun Li ¹, Sulan Zhang³ and Xiaofeng Li ^{4*}

¹ College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai 200234, China; 1000527100@smail.shnu.edu.cn (S.L.); lishuang@shnu.edu.cn (S.L.); success@shnu.edu.cn;

² School of Computer Science and Technology, DongHua University, Shanghai, 201620, China 2232838@mail.dhu.edu.cn

³ School of Information Science and Engineering, Jiaying University, Jiaying, Zhejiang, 314000, China zhangsl000111@163.com

⁴ Shanghai Normal University TianHua College School of Artificial Intelligence Shanghai 201815, China; lxf2812@sthnu.edu.cn

* Correspondence: lishuang@shnu.edu.cn (S.L.), lxf2812@sthnu.edu.cn (X.L.)

Abstract: Despite the indispensable role of traditional electronic computers in modern society, their limitations in parallel processing capabilities, bit-width constraints, and processor bit-width are increasingly evident, especially when handling large-scale datasets and complex computational tasks. Although hardware technology and algorithm optimization continue to advance, the arithmetic units of traditional computers—adders—remain constrained by carry delay and bit-width limitations. This bottleneck is particularly pronounced in multiplication operations, especially when adders are used for partial product accumulation. However, since 2018, the emergence of a new type of Reconfigurable four-valued logic electronic processor (RFLEP) has provided a potential solution to these traditional limitations. This processor, with its large processor bit-width, flexible bit grouping capabilities, and dynamic hardware function reconfiguration features, has brought revolutionary changes to the field of computing. In this context, this paper proposes and implements a Reconfigurable four-valued logic multiplication routine (RFLMR) tailored explicitly for the RFLEP. The RFLMR utilizes the Modified Signed-Digit (MSD) representation method in multi-valued logic, combined with the M transformation in four-valued logic to generate partial products. These partial products are then efficiently summed in parallel using the JW-MSD parallel adder, achieving rapid execution of multiplication operations. Experimental results demonstrate that the multiplication routine based on the RFLEP not only performs multiplication operations accurately but also exceeds theoretical expectations in terms of implementation efficiency and performance, showcasing great potential in the field of high-speed computing. This research not only provides new ideas for developing next-generation high-performance computing systems but also paves the way for exploring more efficient and powerful computing models, heralding a profound transformation in future computing technology.

Keywords: Reconfigurable four-valued logic electronic processor; Reconfigurable four-valued logic multiplication routine; Parallel computing; Dynamic hardware function reconfiguration; Modified Signed-Digit

Citation: Liao, S.; Li, S.; Gu, X.; Li, L.; Zhang, S. Research on multiplication routine based on reconfigurable four-valued logic processor. *Journal Not Specified* **2024**, *1*, 0. <https://doi.org/>

Received:

Revised:

Accepted:

Published:

Copyright: © 2024 by the authors. Submitted to *Journal Not Specified* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. introduction

In the early stages of electronic computing, hardware design primarily revolved around adders, with subtraction achieved by applying the complement to the subtrahend and performing the operation in the adder. Software initially handled Multiplication and division, converting them into a series of addition or subtraction operations. It wasn't until integrated circuit technology advanced to accommodate millions of transistors that it became feasible to implement multipliers in hardware, significantly accelerating computer

computation speeds. Despite hardware multipliers being core components for performing multiplication operations in electronic computers and greatly enhancing computational performance, they come with certain limitations. These include high manufacturing costs, large chip area occupation, high power consumption, and limited flexibility during operation. Designing and producing complex multiplier hardware requires substantial resources and technical investment, directly increasing their cost. Additionally, multiplier circuits often occupy considerable chip space, potentially increasing the complexity of chip design. Furthermore, the design of hardware multipliers is typically fixed, making it difficult to alter the method or precision of multiplication operations during runtime, thereby limiting the computer's adaptability to different application scenarios.

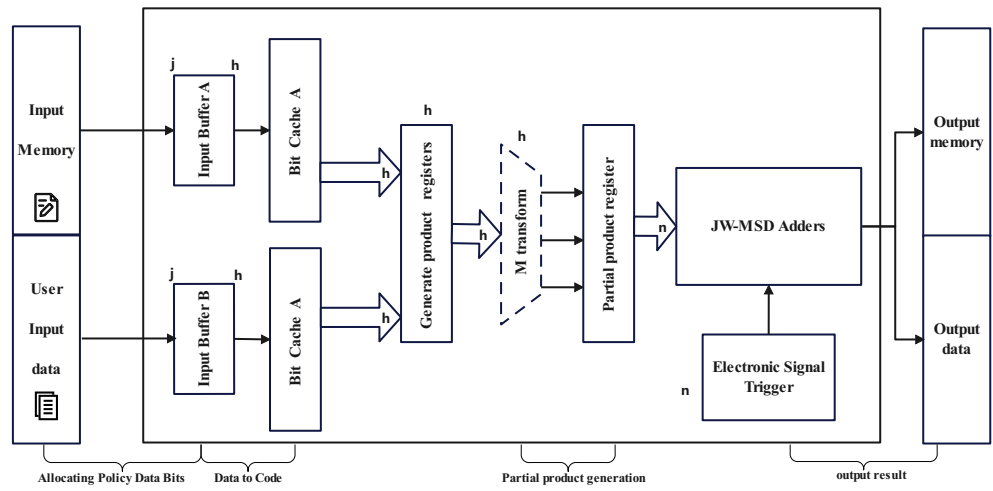


Figure 1. RFLMR structure diagram

To overcome these challenges, reconfigurable multi-valued logic electronic processors have emerged, drawing on the research theories of ternary optical computers[1, 2, 3, 4, 5, 6, 7, 8], particularly the structure and characteristics of the SD16 ternary optical processor[9, 10, 11, 12]. A patent filed by Professor Jin Yi and others in January 2019, titled "A Multi-Valued Logic Electronic Processor with Numerous Bits, Grouping Capability, and Reconfigurability, and Method," laid the foundation for the development of such processors[13]. In 2021, Wang et al. implemented a reconfigurable four-valued logic processor on an FPGA and tested it with various four-valued logic operations[14, 15, 16, 17], verifying the effectiveness and reconfigurability of the processor architecture and circuits. Based on the theory of reduced-value design[18, 19], the reconfigurable multi-valued logic electronic processor can dynamically determine the current computational function of the processor by inputting different reconfiguration instructions, meaning that the computational function of each processor bit can be reconfigured independently. By combining these bits, a multi-valued logic processor structure with numerous bits and its reconfigurable circuits can be constructed. This paper utilizes multiple computational bits of the reconfigurable four-valued logic processor to build a JW-MSD adder. The characteristic of this adder is that there is no carry relationship between data bits, allowing all data bit addition operations to be completed synchronously without considering carry dependencies between bits. On this basis, we designed and implemented a reconfigurable quaternary logic multiplication routine, as shown in Figure 1. During the reconfigurable quaternary logic multiplication process, input and output memory serve as crucial channels for data interaction between the user and the system. Input data is stored in the Input Memory and converted into MSD numbers through a bit allocation strategy. Then, auxiliary data is generated from the input MSD numbers via Bit Cache A and Bit Cache B, followed by an M Transform to obtain partial products. Finally, the electronic signal trigger activates the JW-MSD adder to per-

form accumulation operations, thus completing the multiplication routine calculation. The proposed reconfigurable quaternary logic multiplication routine provides a novel approach for ternary logic operations, laying a solid foundation for designing more complex ternary logic computing systems in the future.

2. MSD Digital System

The Modified Signed-Digit (MSD) numeral system represents a pivotal innovation in digital computation[20, 21, 22, 23], primarily due to its ability to perform non-carry addition, thereby significantly enhancing computational efficiency. The concept of MSD can be traced back to the 1960s when computer scientists actively sought new methods to improve computational speed and performance. Traditional binary numeral systems faced substantial challenges with carry mechanisms when processing large volumes of data. To address this core issue, the concept of MSD was introduced and gradually evolved into an effective solution. The introduction and development of MSD have profoundly impacted modern computer science. Firstly, it cleverly circumvents the carry problem inherent in traditional binary systems, leading to a leap in computational speed. Secondly, the application of MSD in optical computing has driven innovation and progress in optical computing technology. More importantly, the non-carry characteristic of MSD demonstrates significant advantages in large-scale parallel computing scenarios, providing robust support for high-performance computing. Therefore, MSD not only enriches the theoretical methods of numerical representation but also promotes optimising and upgrading computer hardware and software systems in practical applications, laying a solid foundation for efficient computing in the era of big data.

2.1. MSD Number

Traditional electronic computers use binary (0 and 1) for computation, where any decimal number has a unique representation. However, the ternary optical computer (TOC) adopts the MSD number system, which represents a signed ternary binary number. The same decimal number can have multiple representations, providing a theoretical basis for carry-free addition. This system was first proposed by A. Avizienis et al. in 1961 and introduced into the field of optical computing by Drake et al. in 1986. The symbol set for this system is $u, 0, 1$, where u represents the value -1. Any decimal number D can be represented using the MSD number system, with its expression shown in formula 1:

$$A = \sum_i a_i \times 2^i \quad (1)$$

In the above formula, the value of a_i is $\{u, 0, 1\}$, where i is a natural number, and 2^i indicates that the number is still a binary number.

2.2. MSD Number Characteristics

Due to redundant notation, MSD numbers have the following characteristics compared to regular binary number representations.

1. In addition to the data 0 value, any value has an infinite number of MSD number expressions, such as: $[-32]_{10} = [u00000]_M = [u100000]$, $[5.5]_{10} = [101.1]_M = [10uu.1]_M$
2. The MSD number of a value is inverted bit by bit (0 is unchanged, and one and u are converted to each other) to obtain the MSD number of the opposite of the value: $[-15]_{10} = [uuuuu]_M = -[1111]_M$, $[-7]_{10} = [uuu]_M = -[111]_M$

2.3. MSD Addition

MSD addition differs from conventional binary addition in that it is accomplished through four transformations: T , W , T' , and W' (as shown in Table 1). These four transformations are ternary logic operations. Due to the redundancy of MSD, the corresponding addition can be completed in just three clock cycles, regardless of the number of bits in

the addends[24]. Suppose there are MSD numbers $A = \sum a_i \times 2^i$ ($i=0,1,\dots,n-1$), and MSD number $B = \sum b_j \times 2^j$ ($j=0,1,\dots,m-1$).

The symbol sets for a_i and b_j are both $u, 0, 1$, where n and m are positive integers, and i and j are natural numbers, ($0 \leq i \leq n-1$), ($0 \leq j \leq m-1$). The MSD addition process is as follows:

- 1). Perform T and W operations bitwise on the inputs A and B . (The result of the T operation is padded with 0 on the lower bit, and the result of the W operation is padded with 0 on the higher bit).
- 2). Perform T' and W' operations bitwise on the results of the first step. (The result of the T' operation is padded with 0 on the lower bit, and the W' operation is padded with 0 on the higher bit).
- 3). Perform T_2 operations bitwise on the results of the second step to obtain the sum of A and B finally.

Table 1. T , W , T' and W' , T_2 transformations

T				W				T'				W'				T_2			
b	$a(t')$			b	$a(t)$			b	$w(t')$			b	$w(t)$			b	$a(t')$		
	u	0	1		u	0	1		u	0	1		u	0	1		u	0	1
u	u	u	0	u	0	1	0	u	u	0	0	u	0	u	0	u	u	u	0
0	u	0	1	0	1	0	u	0	0	0	0	0	u	0	1	0	u	0	1
1	0	1	1	1	0	u	0	1	0	0	1	1	0	1	0	1	0	1	1

The MSD number system's unique redundancy property enables the feasibility of parallel addition operations. This characteristic stems from the inherent attribute of the MSD number representation, where the computation of each digit does not need to wait for a carry signal, thereby breaking the limitation of sequential digit computation in traditional serial adders. Specifically, during the MSD addition process, the result of each digit can be generated independently, unaffected by the operations of other digits. This provides the theoretical foundation for achieving fully parallel logic transformations within a single cycle. Therefore, the parallel addition mechanism under the MSD architecture not only enhances computation speed but also optimizes the utilization efficiency of hardware resources, demonstrating potential advantages in the field of high-performance computing.

3. Implementation Mechanism of the RFLMR

3.1. Working Principle of the RFLEP

The circuit structure of the RFLEP is shown in Figure 2. This processor contains multiple processor bits (m bits). In the figure, label ① represents the 0th bit of the processor, with the output signal being $C_0(n \text{ value})$; label ② represents the $(m-1)$ th bit of the processor, with the output signal being $C_{m-1}(n \text{ value})$. The structure of each processor bit is identical. As shown in ①, each processor bit includes n column operators (such as ③ and ④) and a potential combiner, as illustrated in Figure 2. Among them, ③ is the 0th column operator of the 0th operator bit, with the output being $Co^0(n \text{ value})$; ④ is the $(n-1)$ th column operator of the 0th operator bit, with the output being $Co^{n-1}(n \text{ value})$. The structure of the column operators in each processor bit is the same. As shown in ③, each column operator includes a high-impedance gate ⑤, a level multiplexer ⑥, an A signal selector ⑦, a work enabler ⑧, a reconfiguration latch ⑨, and a reconfiguration circuit ⑩.

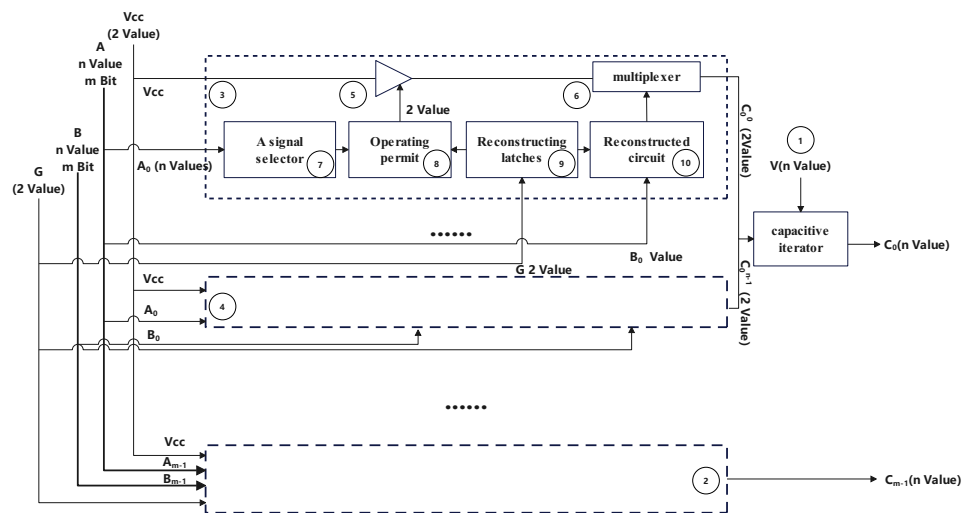


Figure 2. Schematic diagram of RMLEP architecture

In Figure 2, the RFLEP writes reconfiguration instructions into the reconfiguration latch via the G signal line. These reconfiguration instructions determine the operational function of each column operator, enabling one of the $n(n \times n)$ types of logical operations. The reconfiguration circuit has p branches, and since each n -value signal requires p binary signals to represent it, p is equal to the square root of n rounded up. Each branch of the reconfiguration circuit selects and simply transforms the i -th bit of the data B input to the branch based on the signal state of the branch control end and sends the selection and transformation result to the input end of the output generator through the output end of the branch. The input end of the output generator also has p input lines. The potential combiner of the i -th bit of the constructed operator combines the output signals of all column operators of that operator bit to form the physical output signal of that operator bit. Since any value of the i -th bit of data A meets the selection requirements of the A signal selector of a column operator, the operator bit can perform logical operations on any value of the i -th bit of data A and the i -th bit of data B. Therefore, by sending different reconfiguration instructions to the reconfiguration circuit of the general structure of the RFLEP, the current logical operation function of the processor can be dynamically configured, forming specific logical operators and achieving reconfigurable logical operation functions for each processor.

3.2. Reconfigurable four-valued logic JW-MSD adder principle

Based on in-depth research on MSD addition theory, our research team has designed and implemented an innovative parallel adder circuit architecture—the JW-MSD adder. The uniqueness of this adder lies in its specially customized ternary logic operation unit, which closely aligns with the logical requirements of the MSD addition algorithm. Streamlining the number of logic gates significantly enhances the efficiency and speed of dedicated addition operations. The ternary logic operator plays a crucial role in the core design of the JW-MSD parallel adder. Compared to traditional binary adders, which often require complex circuit layouts to handle carry propagation, the MSD adder leverages its ternary logic (u, 0, 1) characteristics to avoid the complexity of carry operations, thereby significantly accelerating computation speed and enhancing operational efficiency. Building on this, the JW-MSD adder further optimizes the design of the ternary logic operator, greatly simplifying the overall circuit structure and effectively reducing the number of logic gates used, achieving a dual optimization of performance and cost-effectiveness. Figure 3 shows the circuit schematic of our developed specialized JW-MSD parallel adder, providing readers

with a clear visual reference to facilitate a deeper understanding of the working mechanism of the JW-MSD adder.

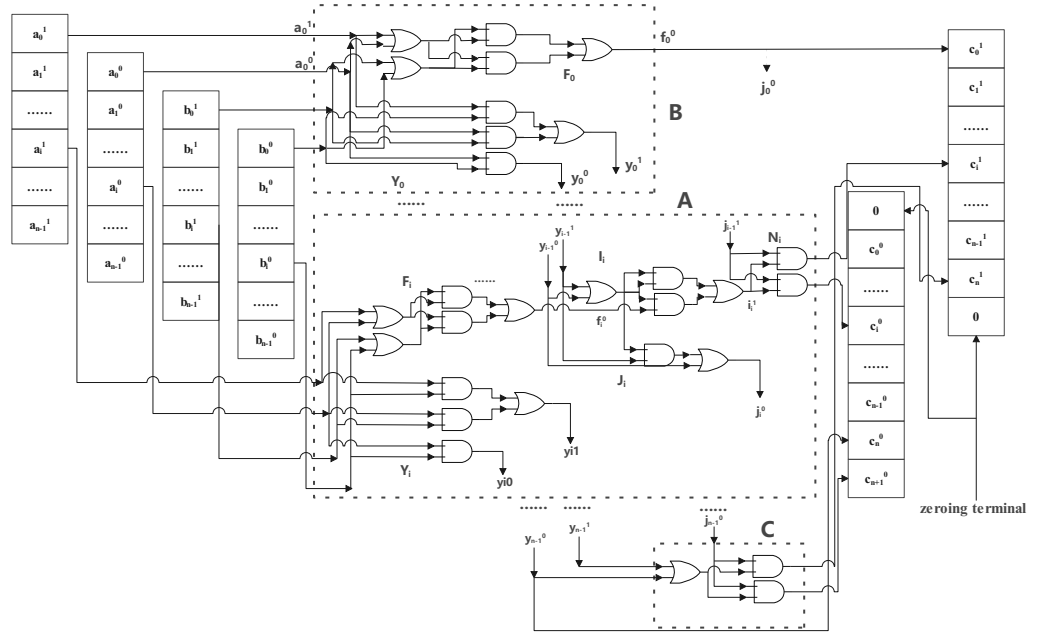


Figure 3. JW-MSD adder structure schematic diagram

The n -bit specialized JW-MSD parallel adder consists of five ternary logic operators, namely the Y operator, F operator, J operator, I operator, and N operator, as shown in Table 2. They use three two-bit binary symbols “00”, “01”, and “10” to represent the MSD numbers “0”, “u”, and “1”. This n -bit specialized JW-MSD parallel adder is composed of three operation modules, as well as input and output registers. The three operation modules are the A operation module, B operation module, and C operation module. For this n -bit specialized JW-MSD parallel adder, the input operands of the adder are represented by $a_{n-1}a_{n-2}...a_1a_0$ and $b_{n-1}b_{n-2}...b_1b_0$, and the output is represented by $c_{n-1}c_{n-2}...c_1c_0$. Each bit is represented by two binary bits, so a_i^0 represents the low bit input of the i -th bit, a_i^1 represents the high bit input of the i -th bit; b_i^0 represents the low bit input of the i -th bit, b_i^1 represents the high bit input of the i -th bit; c_i^0 represents the low bit output of the i -th bit, and c_i^1 represents the high bit output of the i -th bit. The operation cycle of this adder can be divided into three steps:

- 1). First, the n -bit input operands a and b are written into the high and low bit registers of input register a and the high and low bit registers of input register b , respectively, while the low bit signal c_0^0 of the 0th bit and the high bit signal c_{n+1}^1 of the $(n+1)$ th bit of the output register is cleared to 0.
- 2). Second, after the calculations of each operation module, the final output result is obtained. This step involves a delay of 6 layers of binary logic gates, and the calculation result is finally stored in the output register.
- 3). Third, data is read from the output register to construct the n -bit specialized JW-MSD parallel adder, which requires $7 \times n + 5$ two-input OR gates, $10 \times n + 7$ two-input AND gates, and $9 \times n + 6$ single-input NOT gates. Therefore, the n -bit specialized JW-MSD parallel adder will use fewer logic gate devices.

Table 2. Five ternary logic operators of JW-MSD parallel adder

Y operator				F operator				J operator				I operator				N operator			
$a_i \backslash b_i$	00	01	10	$a_i \backslash b_i$	00	01	10	$y_i \backslash f_i$	00	01		$y_i \backslash f_i$	00	01		$j_i \backslash i_i$	00	01	10
00	00	00	10	00	00	01	01	00	00	01		00	00	10		00	00	10	
01	00	01	00	01	01	00	00	01	01	01		01	10	00		01	01	00	
10	10	00	10	10	01	00	00	10	00	00		10	10	00					

3.3. RFLMR implementation

3.3.1. Partial Product Generation and Summation Key Techniques

For implementing the MSD multiplication routine, the main core points include the generation of partial products and the summation of partial products. This paper delves into the implementation strategies for these two aspects. Suppose the multiplicand A has n bits, represented as $A = a_{n-1}a_{n-2}\dots a_1a_0$, with its MSD expression being $A = \sum a_i \times 2^i$, where i ranges from 0 to $n-1$. Similarly, the multiplier B has m bits, represented as $B = b_{m-1}b_{m-2}\dots b_1b_0$, with its MSD expression being $B = \sum b_j \times 2^j$, where j ranges from 0 to $m-1$. The symbols a_i and b_j are both from the set $u, 0, 1$, where n and m are positive integers, and i and j are natural numbers, satisfying $0 \leq i \leq n-1, 0 \leq j \leq m-1$. Thus, the MSD multiplication routine can be expressed by formula 2.

$$C = A \times B = A \times \sum b_j \times 2^j = \sum A \times b_j \times 2^j = \sum S_j \times 2^j = \sum P_j \quad (2)$$

Based on the above formula 2, the MSD multiplication process is as follows:

- 1). In $C = A \times B$, A is the multiplicand, and B is the multiplier. B is generated as $B = \sum b_j \times 2^j$. Each bit b_j of B generates auxiliary data with p bits of b_j values. b_j and A undergo M transformation bitwise to obtain j partial products S_1, S_2, \dots, S_j , the M transformation table is shown in Table 3.
- 2). Each partial product S_j is shifted left by $j-1$ bits, $j-1$ zeros are appended to each S_j , resulting in j sum terms P_1, P_2, \dots, P_j .
- 3). The j sum terms P_1, P_2, \dots, P_j are summed using an MSD adder, and the accumulated result is the product C .

Table 3. M transformation

M			
b	a		
	u	0	1
u	1	0	u
0	0	0	0
1	u	0	1

3.3.2. RFLMR design

The RFLMR is implemented through programming on the Alinx7020, as shown in Figure 4. It consists of six operators and 199 processor bits. Since electronic transistors can only represent binary, we use two binary bits to represent MSD numbers. Specifically, the three MSD states (such as $u, 0, 1$) can be represented by binary 00, 01, and 10, respectively. In this representation method, each bit combination is realized through the on-and-off states of electronic transistors. This design not only maintains the characteristics of ternary logic but is also compatible with the binary representation of electronic transistors, making it possible to implement the RFLMR on the Alinx7020 through the Processor System (PS) and Programmable Logic (PL) parts. The design of the multiplication routine is mainly divided into four parts:

- 1). **MSD Number Conversion:** First, the input binary number must be converted into an MSD number representation. This step is crucial as it lays the foundation for subsequent partial product generation and addition operations. Our design implements MSD number conversion through the MSD_transform function on the PS side. The core objective of the MSD_transform function is to convert a given decimal number (D) into its corresponding MSD representation. Assuming we have a decimal number (D), we first convert its absolute value ($|D|$) into a binary representation. If ($0 < D$), the resulting binary number is inverted; otherwise, it remains unchanged. Then, the obtained binary number is converted into the corresponding MSD number, where 00, 01, 10, etc., represent different digits.
- 2). **Partial Product Generation:** During partial product generation, the multiplier (B) first generates auxiliary data for each bit, which is stored in registers through the generate_auxiliary_data function. Then, the auxiliary data corresponding to each bit (b_j) in the register is sent to the arithmetic unit (S) along with each bit of the multiplicand (A) for computation. Specifically, for each bit (b_j), an M-logic operation with the multiplicand (A) is performed to generate the partial product (S_j). To improve computational efficiency, M-transformation uses parallel computing technology. By implementing parallel operations in hardware, multiple partial products can be computed simultaneously, reducing computation time. The computed partial products (S_j) are stored in registers for subsequent shifting and summation operations.
- 3). **Partial Product Shifting:** After generating the partial products, each partial product needs to be shifted for subsequent summation calculations. Each partial product (S_j) needs to be left-shifted by ($j-1$) positions, with ($j-1$) zeros appended to the end of each (S_j) through the shift_operation function, resulting in partial products (P_1, P_2, \dots, P_j). This operation is implemented using hardware shift registers to ensure the efficiency of the shifting operation.
- 4). **Partial Product Summation:** Partial product summation is the crucial step in generating the final product (C), achieved through the JW-MSD adder, which uses ternary logic to sum different partial products quickly. On the PL side, partial products are sent to the JW-MSD adder for computation through the MSD_33_mWriteReg function. Given the abundant hardware logic units in the Alinx7020, multiple JW-MSD adders can be configured on the PL side. Due to the redundancy characteristics of MSD numbers (see 2.3 MSD Number Characteristics), the summation of partial products (P_1, P_2, \dots, P_j) adopts parallel computing methods. Rapid accumulation is achieved by cascading and pipelining multiple levels of MSD adders, resulting in the final MSD product. This result is then transmitted to the platform for display and further processing. The entire process is highly parallel and flexible, enabling efficient digital computation.

3.3.3. Pipelined design of RFLMR

In this study, we delve into the multiplication operation ($A \times B = C$), where (A) and (B) have (m) and (n) MSD digits respectively, satisfying ($0 < m < 33$) and ($0 < n < 33$). For the multiplication routine processor architecture, we propose a novel and efficient computational method that combines M transformation with JW-MSD addition technology, where (B) acts as the multiplier and (A) as the multiplicand. This process involves (n) M transformations and their associated partial cumulative addition operations. We fully utilize the inherent redundancy of MSD numbers to optimize multiplication efficiency. Therefore, we introduce a multiplication routine based on reconfigurable four-valued logic and implement a pipeline design strategy to facilitate smooth and fast data processing. The core idea of the pipeline design is to break down complex operations into several independent stages, allowing each stage to process different data segments in parallel or sequentially. This architecture is particularly suitable for the Alinx7020 platform, where

precise management of clock cycles and operation segmentation ensures the efficiency and accuracy of the data processing flow.

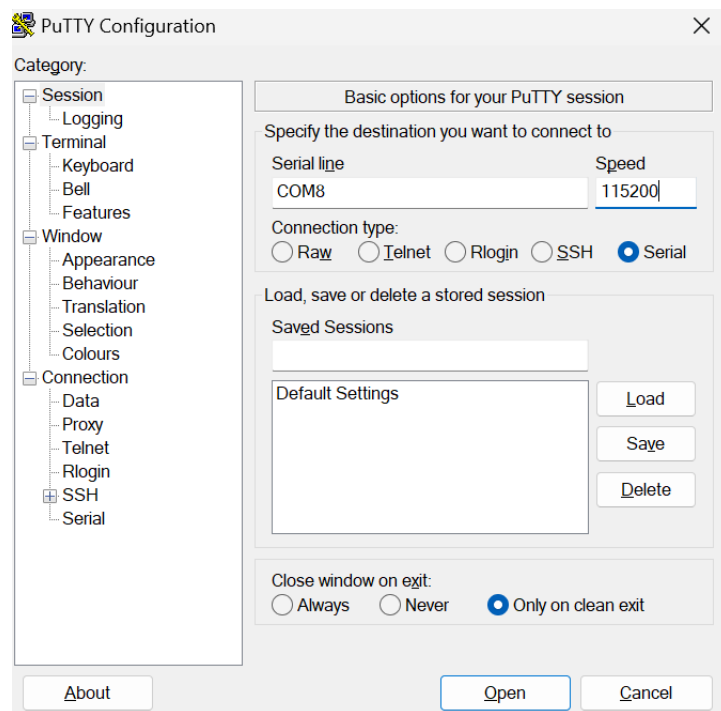


Figure 4. Putty configuration diagram

As shown in Figure 5, our pipeline design comprehensively covers the entire processing flow from input to output, illustrating the correspondence between the time sequence (measured in clock cycles) and each processing stage. Each periodic operation module—whether it is M transformation, T/W transformation, T'/W' transformation, or T2 transformation—represents a specific processing step or operational unit in the pipeline. The parallel execution of these modules significantly enhances the overall processing speed.

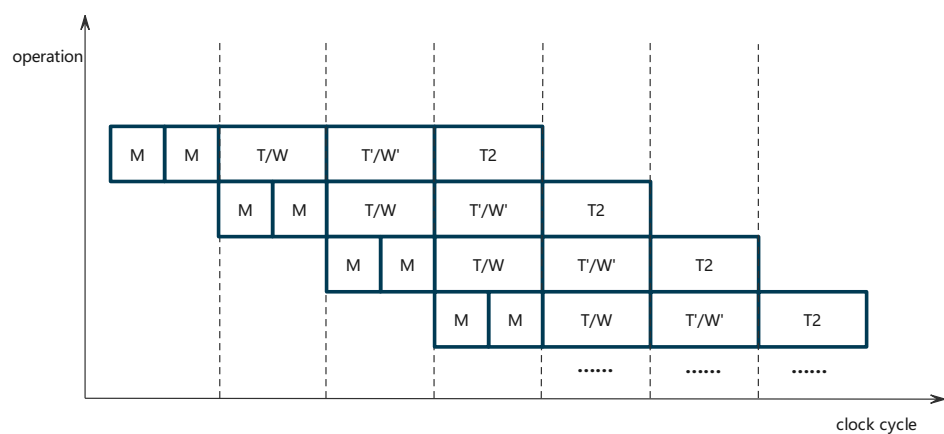


Figure 5. RFLMR pipeline diagrams

4. Efficient experiments with RFLMR

4.1. Experimental preparation

4.1.1. Experimental equipment

In this experiment, we selected the Zynq7000 series chips from Xilinx as the core components. Specifically, we used the Alinx7020 core development board equipped with the XC7Z020-2CLG400I chip. This chip is renowned for its excellent pin compatibility and the 400-pin FBGA packaging technology it employs, ensuring design flexibility and efficiency. The unique feature of the Zynq7000 chip is its integration of the Processor System (PS) and Programmable Logic (PL) cores, achieving a perfect blend of hardware and software. The PS part on the Alinx7020 core board provides wealthy external interfaces and devices, greatly facilitating user operations and function verification processes. Meanwhile, each IO interface on the PL side is designed and routed to 2.54mm connectors on the board edge, offering users unlimited expansion possibilities to meet diverse needs from prototype design to product development. Notably, the core board is equipped with a 7 x 2 JTAG connector, allowing users to easily download and debug programs on the core board using Xilinx's dedicated USB Cable downloader, significantly simplifying the development process. Figure 6 vividly illustrates the overall architecture of the Alinx7020 system. With such a hardware layout and flexible interface configuration, the Alinx7020 provides stable and reliable hardware support, making it an ideal choice for exploring advanced computing technologies and implementing complex algorithms.

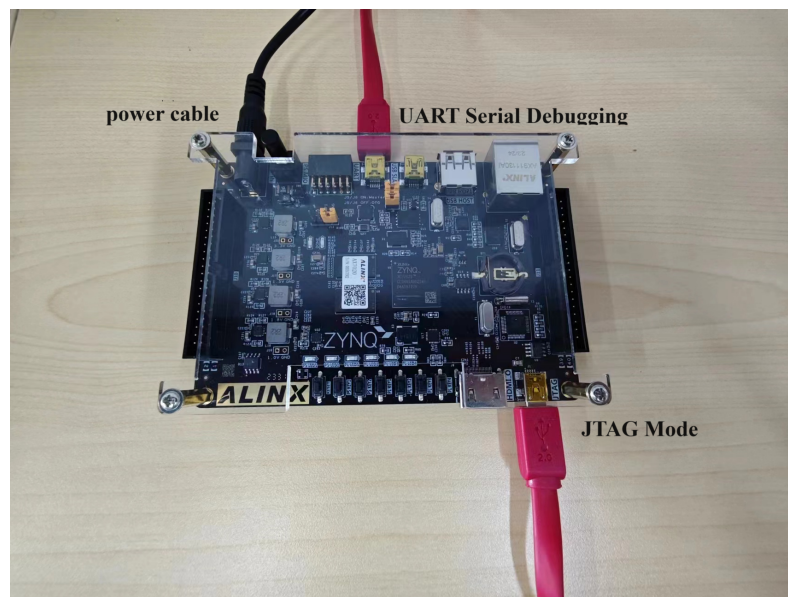


Figure 6. Alinx7020 development board

4.1.2. Processor Bit Allocation Strategy for RFLMR

On the Alinx7020 development board, we implemented the M transformation (S arithmetic unit) and JW-MSD adder in parallel within the Processor System (PS) and Programmable Logic (PL), as shown in Figure 7. Specifically, processor bits 0 to 32 are allocated for implementing the M transformation, while bits 33 to 198 are dedicated to constructing the JW-MSD adder. Each part of the design follows the bit allocation principles to ensure the efficient operation of the entire system. The layout of a 199-bit RFLMR processor is shown in Figure 7.

00	01	10	00	01	10	00	01	10	00	01	10	00	01	10	00	01	10	00	01	10
198				197				196					195				194			193
192				191				190					189				188			187
186				185				184					183				182			181
180				179				178					176				175			174
173				172				171					170				169			168
167				166				165					164				163			162
161				160				159					158				157			156
155				154				153					152				151			150
149				148				147					146				145			144
143				142				141					140				139			138
137				136				135					134				133			132
131				130				129					128				127			126
125				124				123					122				121			120
119				118				117					116				115			114
113				112				111					110				109			108
107				106				105					104				103			102
101				100				99					98				97			96
95				94				93					92				91			90
89				88				87					86				85			84
83				82				81					80				79			78
77				76				75					74				73			72
71				70				69					68				67			66
65				64				63					62				61			60
59				58				57					56				55			54
53				52				51					50				49			48
47				46				45					44				43			42
41				40				39					38				37			36
35				34				33					32				31			30
29				28				27					26				25			24
23				22				21					20				19			18
17				16				15					14				13			12
11				10				9					8				7			6
5				4				3					2				1			0

Figure 7. RFLMR processor bit allocation chart

Specifically, for a 199-bit RFLMR processor, the allocation of processor bits is shown in Table 4.

Table 4. RFLMR processor bit allocation table

M	T	W	T'	W'	T_2
0~32(S operator)	33~65(Y operator)	66~98(F operator)	99~131(J operator)	132~164(I operator)	165~198(N operator)

The rich hardware resources of the Alinx7020 development board provide a solid foundation for this multi-module parallel design. By executing each module in parallel without interference, we significantly improve the efficiency of the four-valued logic multiplication routine processor in handling complex computational tasks and greatly enhance its flexibility and reconfigurability. This design strategy allows for dynamic adjustment and optimization of the system based on different computational needs, demonstrating adaptability and high performance.

4.2. Experimental process of RFLMR

4.2.1. Test Cases

To rigorously evaluate the accuracy of the RFLMR in handling arbitrary decimal values and ensure the correctness of outputs at each computational stage, this experiment designed a series of verification steps using registers as an essential aspect. We defined three core registers:

- Register k is used to store values converted from decimals to MSD.
- Register m is responsible for recording partial products obtained after M transformation.
- Register s are specifically used to retain the complete MSD calculation results.

Through this data management strategy, we can strictly monitor and ensure each computation's accuracy and the data's integrity.

Given that the processor bit width of the reconfigurable four-valued logic routine is set to 33 bits, its numerical expression range spans from (-3^{33}) to $(3^{33} - 1)$, naturally constraining the bit width of the multiplicand and multiplier. To comprehensively examine the performance of this multiplication routine processor across a broad numerical domain,

we conducted large-scale experiments, testing over ten thousand data samples. Although space limitations prevent a detailed discussion of each test, the selected cases are highly representative, and all test results consistently confirm the algorithm’s effectiveness and accuracy. The experimental data (refer to Table 5) robustly demonstrate the computational capability and precision of the RFLMR processor across different numerical ranges.

Table 5. ERLMR test cases

Item	multiplication	Multiplier	Multiplicand
01	2×3	$A(000000000000000000000000000010)_M$	$B(000000000000000000000000000011)_M$
02	4×5	$A(0000000000000000000000000000100)_M$	$B(0000000000000000000000000000101)_M$
03	12×15	$A(00000000000000000000000000001100)_M$	$B(00000000000000000000000000001111)_M$
04	-23×34	$A(0000000000000000000000000000u0uuu)_M$	$B(0000000000000000000000000000100010)_M$
05	-67×-78	$A(0000000000000000000000000000u0000uu)_M$	$B(0000000000000000000000000000u00uuu0)_M$
06	123×234	$A(0000000000000000000000000000001111011)_M$	$B(00000000000000000000000000000011101010)_M$
07	-345×456	$A(000000000000000000000000u0u0uu00u)_M$	$B(000000000000000000000000111001000)_M$
08	1234×2345	$A(0000000000000000000000100110101010)_M$	$B(0000000000000000000000100100101001)_M$
09	-3456×4567	$A(00000000000000000000uu0uu0000000)_M$	$B(0000000000000000001000111010111)_M$

4.2.2. RFLMR experimental steps

In constructing the RFLMR, we retained the input data interface on the PS side, connecting to the Alinx7020 core board via the Putty serial debugging tool to facilitate the input of test cases and the verification of results. The specific experimental implementation steps are as follows:

1). Hardware Connection and Configuration:

- ① Use Xilinx's USB Cable to Download Files: As shown in Figure 4, connect one end of the USB Cable to the Alinx7020 board and the other to the host computer. Then, download the designed hardware configuration file to the Alinx7020 core board to ensure the core board correctly loads and runs our design.
- ② Connect the JTAG Connector for Debugging: Download the required files to the Alinx7020 via the USB Cable, and then prepare the debugging program through the 7 x 2 JTAG connection.

2). Software Environment Preparation:

- ① **Install and Configure the Putty Serial Debugging Tool:** Download and install the Putty software on the debugging computer, ensuring the installation process is completed smoothly.
- ② **Set Serial Port Parameters:** Open the Putty software and set the serial port parameters to a baud rate of 115200 and port number COM8. The detailed parameters and Putty graphical interface are shown in Figure 4 to match the communication requirements of the Alinx7020 core board.
- ③ **Confirm Connection:** In the Putty software, confirm that the serial port of the Alinx7020 core board is correctly recognized and connected, ensuring a smooth data transmission channel.

3). Input Test Cases:

- ① **Initialize Registers:** Before inputting each test case, initialize registers k , m , and s to 0 to ensure that the results of the previous operation do not affect the current calculation.
- ② **Input Test Cases:** Sequentially input test cases 1 to 17 to the Alinx7020 core board through the Putty serial tool according to the designed order. Each test case includes two operands, and the multiplication routine processor will calculate and output the results based on the input.
- ③ **Read Results:** Read the output results of the multiplication routine processor for each test case. The results are shown in Table 6.

- 4). **Display Experimental Results:** After inputting the test cases, observe the output of the Putty serial debugging tool and record the calculation results of each case. Compare the output results with the expected correct results. The calculation results are shown in Figure 9, verifying the performance and accuracy of the reconfigurable four-valued logic multiplier across different numerical ranges.

Table 6. RFLMR calculation results[illegible]

4.2.3. Experimental results and analysis

In this study, we constructed a RFLMR, with its execution process illustrated in Figure 8. The core innovation lies in the generation and merging mechanism of partial products, which starkly contrasts the binary multiplication based on shifting and accumulation used in traditional computers. Traditional algorithms are limited by the propagation delay of low-order carry, a bottleneck directly stemming from the inherent properties of the binary logic system. However, thanks to the redundancy characteristics of the MSD number system and the rich hardware logic resources of the ALinx 7020 platform, our design achieves bit-level parallel processing, completely overcoming this limitation.

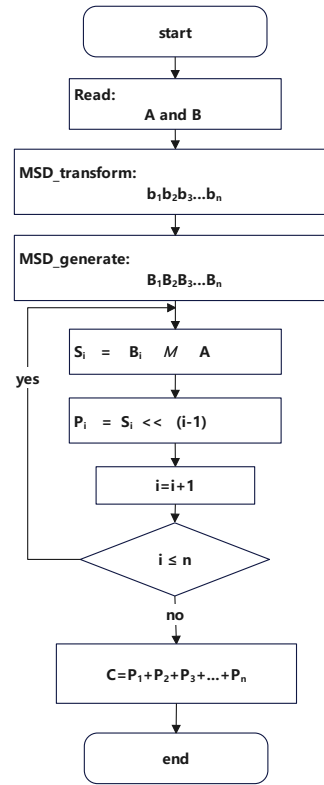


Figure 8. RFLMR flowchart

In the design process of the RFLMR, we adopted a modular strategy and deeply optimized the transformations of M , T , W , T' , and W' . Each module undertakes specific encoding conversion tasks, including but not limited to the M transformation module (S operator), T transformation module (Y operator), W transformation module (F operator), T' transformation module (J operator), W' transformation module (I operator), and an additional T_2 transformation module (N operator). The extensive library of electronic logic units on the ALinx 7020 platform (up to thousands) is sufficient to support the parallel deployment of dozens of such essential processors, meaning that only a few operational instructions are needed to complete complex computational tasks efficiently. For instance, a basic module with a processing capacity of 1024 bits in application-oriented scenarios is sufficient to construct five similar architecture processors. Therefore, the RFLMR processor can handle multiple computational tasks simultaneously with just one multiplication instruction call, significantly reducing the repeated execution of operational instructions and saving valuable computation time and hardware resources. This design approach substantially improves computational efficiency and efficiently utilizes hardware resources, bringing unprecedented speed and performance improvements to multiplication operations. The four-valued logic multiplication routine processor, with its unique parallel processing capability and high adaptability, demonstrates advantages in diverse application fields, indicating that it will play a more critical role in the future development of computing technology.

```

COM8 - PuTTY
--Reconfigurable Four-Valued Logic Multiplication Routine--

2 * 3 = 6

4 * 5 = 20

12 * 15 = 180

-23 * 34 = -782

-67 * -78 = 5226

123 * 234 = 28782

-345 * 456 = -157320

1234 * 2345 = 2893730

-3456 * 4567 = -15783552

```

Figure 9. Putty output results of RFLMR

Given that the bit-width of the processor for the RFLMR is set to 33 bits, and each of the three-valued states stored in the register—namely 00, 01, and 10—requires two bits per state, the total register capacity must accommodate 66 bits. To simplify the visual representation, we converted the three-valued states in the register into a more compact MSD numerical representation. Table 7 and Figure 9 show that the experimental results align with the expected outcomes, validating the method's effectiveness. To further confirm the accuracy of the results obtained during partial cumulative addition operations, we introduced the Digital Direct Synthesis (DDS) waveform as a verification tool. As an advanced waveform synthesis method, DDS technology can generate signals of arbitrary frequencies, providing significant advantages in verifying the correctness of computational logic. Specifically, taking Examples 1 and 2 as cases, their theoretical expected values are detailed in Table 6. The DDS waveform displays the MSD numbers formed by the conversion. In Example 1 (as shown in Figure 10(1), the values in registers *m1* and *m2* match the partial product values transformed by the *M*(*S* operator), and the *s* register outputs the result. In Example 2 (as shown in Figure 10(2), the values in registers *m1*, *m2*, and *m3* match the three partial products transformed by the *M*(*S* operator), thus confirming the correctness of the results.

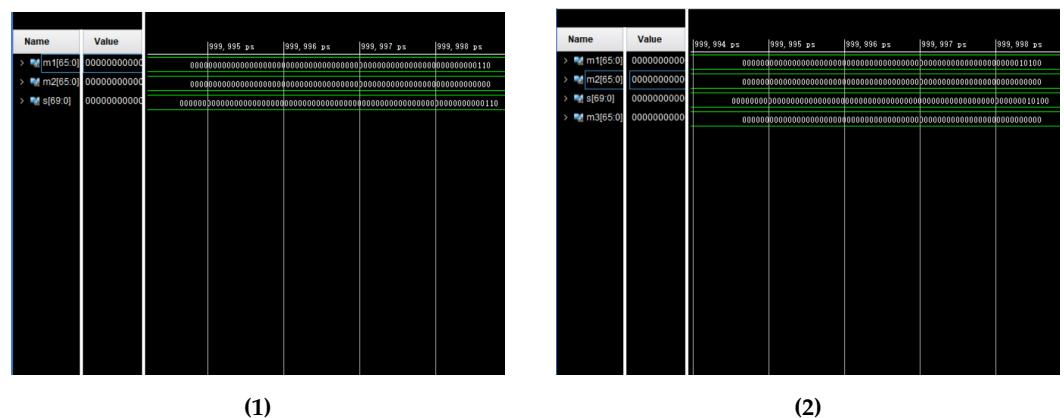


Figure 10. RFLMR partial product summation DDS waveforms

5. Conclusions

This paper establishes a multiplication routine for a four-valued electronic logic processor. The routine generates partial products through *M* transformation in four-valued logic. Then, it uses the JW-MSD parallel adder to sum the partial products, ultimately completing the design of the multiplication routine. The effectiveness of the multiplication routine was verified through experiments. The constructed multiplication routine lays the

foundation for the transition of the four-valued logic electronic processor from laboratory research to applied research. It provides a new approach and method for handling large-number multiplications. The experiments also demonstrated the flexibility and scalability of the reconfigurable four-valued logic processor. By leveraging its reconfigurable characteristics, the RFLMR processor can dynamically adjust its computational functions to improve efficiency when handling complex operations. In the experiments, we rigorously verified each step of the computation of the RFLMR processor through register analysis and serial port debugging tools, ensuring the accuracy of data storage and processing. Particularly in operations involving large numerical ranges, the RFLMR can still be handled efficiently, and the output results meet theoretical expectations. This further demonstrates the potential of the RFLMR in high-performance computing, which can meet the needs of various complex applications and provide strong support for developing future efficient computing technologies.

Author Contributions: Proposing Reconfigurable Four-Valued Logic Multiplication Routine, S.L.; Determining Computational Data and Validating Experimental Results, X.G. and L.L.; Designing and Constructing Multiplication Routine, S.L. and X.L; Writing-Preliminary Draft Preparation, S.L. and X.L; Writing-Reviewing and Editing, S.L. and S.L.; Supervision, S.Z.; All authors have read and agree to the published version of the manuscript.

Funding: This work was supported by the National Science Foundation of China (62302307, 62105207, 61866006), the Open Research Fund of Guangxi Key Lab of Human-machine Interaction and Intelligent Decision (GXHIID2209), and University- level general research project of Shanghai Normal University (SK202121), Hainan Provincial Natural Science Foundation of China under Grant (622MS084)the Education Department of Hainan Province of China (Hnjg2022-90), the General scientific research project of Zhejiang Education Department(Y202352358).

Data Availability Statement: The data used to support the findings of this study are available from the corresponding author upon request.

Acknowledgments: I want to express my gratitude to Prof. Luqun Li, Mr. Li and all the members of the Reconfigurable Four-Valued Logic Operator Team at Donghua University for their valuable comments and suggestions on this paper.

Conflicts of Interest: The authors declare that we have no competing interests.

Conflicts of Interest: The authors state that we have no conflicts of interest. We confirm that there are no financial, personal, or professional affiliations that could be perceived as influencing the research presented in this paper.

References

- [1] Sulan Zhang et al. "Key theories and technologies and implementation mechanism of parallel computing for ternary optical computer". In: *PloS one* 18.5 **2023**. e0284700.
- [2] Shuxin Wang, Jiabao Jiang, and Zhehe Wang. "Research of Tri-State Optical Signal Detectors for Ternary Optical Computers". In: *Applied Sciences* 13 **2023**. P. 98.
- [3] Zhehe Wang and Yunfu Shen. "Design and implementation of bitwise parallel MSD square rooting in ternary optical computer". In: *Optical Engineering* 60.8 **2021**. P. 085107.
- [4] Wang Zhehe et al. "Design and Implementation of a Ternary Optical Computer Simulator". In: *Computer Application Research* 40.4 **2023**. Pp. 1137–1141.
- [5] Li Shuang et al. "Basic theory and key technology of programming platform of ternary optical computer". In: *Optik* 178 **2019**. Pp. 327–336.
- [6] Jin Yi. "Management strategy of data bits in ternary optical computer". In: *Journal of Shanghai University (Natural Science Edition)* 13.5 **2007**. Pp. 519–523.
- [7] Zhang Sulan et al. "Programming model and implementation mechanism for ternary optical computer". In: *Optics Communications* 428 **2018**. Pp. 26–34.
- [8] Hongjian Wang et al. "Ternary optical computer: an overview and recent developments". In: *2021 12th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*. IEEE. **2021**, pp. 82–87.
- [9] Yi Jin et al. "Theory and Structure of the Ternary Logic Optical Processor SD16". In: *Journal of Electronics* 51.5 **2023**. Pp. 1154–1162.
- [10] Jiabao Jiang et al. "Design and Implementation of the SJ-MSD Adder in Ternary Optical Computers". In: *Journal of Electronics* 49.2 **2021**. P. 275.
- [11] Yi Jin et al. "Ternary Optical Computers". In: *Journal of Nature* 41.3 **2019**. Pp. 207–218.
- [12] Zhehe Wang et al. "Design and Implementation of the Ternary Optical Computer Simulator". In: *Application Research of Computers* 40.4 **2023**.
- [13] Yi Jin et al. "A Multi-Digit, Groupable, and Reconfigurable Multi-Valued Electronic Calculator and Method". Patent CN201811567284.7 Shanghai University. July **2022**.
- [14] Yao Lu et al. "Demonstration system of reconfigurable multi-valued logic electronic processor". In: *Second International Conference on Electronic Information Technology (EIT 2023)*. Vol. 12719. SPIE. **2023**, pp. 190–196.
- [15] Tsutomu Sasao and Hiroki Nakahara. "Implementations of reconfigurable logic arrays on FPGAs". In: *2007 International Conference on Field-Programmable Technology*. IEEE. **2007**, pp. 217–223.
- [16] Hongjian Wang et al. "The design and implementation of reconfigurable quaternary logic processor". In: *International Conference on Parallel and Distributed Computing: Applications and Technologies*. Springer. **2021**, pp. 142–149.
- [17] Yao Lu et al. "Demonstration system of reconfigurable multi-valued logic electronic processor". In: *Second International Conference on Electronic Information Technology (EIT 2023)*. Vol. 12719. SPIE. **2023**, pp. 190–196.
- [18] Junyong Yan. "Design Theory for Devaluation". PhD thesis. Shanghai University, **2010**.
- [19] Junyong Yan, Yi Jin, and Kaizhong Zuo. "Design Theory for Devaluation of No-Carry (Borrow) Calculator and Its Application in Ternary Optical Computers". In: *Science in China Series E: Technological Sciences* 38.12 **2008**. P. 11.
- [20] Richard P Bocker et al. "Modified signed-digit addition and subtraction using optical symbolic substitution". In: *Applied Optics* 25.15 **1986**. Pp. 2456–2457.
- [21] Junjie Peng et al. "Design and implementation of modified signed-digit adder". In: *IEEE Transactions on Computers* 63.5 **2012**. Pp. 1134–1143.
- [22] Barry L Drake et al. "Photonic computing using the modified signed-digit number representation". In: *Optical Engineering* 25.1 **1986**. Pp. 38–43.
- [23] Baofeng Qi et al. "Algorithm-based Study on Transformation Combination for Carry-free Modified Signed Digit (MSD) Addition". In: *2023 IEEE 10th International Conference on Cyber Security and Cloud Computing (CSCloud)/2023 IEEE 9th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE. **2023**, pp. 298–304.
- [24] Yi Jin et al. "Theory and Structure of the MSD Adder in Ternary Optical Computers". In: *Science China Information Sciences* 41.5 **2011**. Pp. 541–551.