Crimson Publishers
Wings to the Research

Research Article

Research and Implementation of Carry-free Parallel Multiplication Based on Queue in Ternary Optical Computer

Liu Yong[1], Jiang Jiabao[1]*, Yan Xiaoyan[1], Li Shuang[2] and Wang Zhehe[3]

[1]Key Laboratory of Data Intelligence and Cyber Security, Chaohu University, China

[2]The College of Information, Mechanical and Electrical Engineering, SHNU, China

[3]School of Computer Science and Technology, Hainan Tropical Ocean University, China

ISSN: 2640-9739

***Corresponding author:** Jiang Jiabao, Key Laboratory of Data Intelligence and Cyber Security, Chaohu University, China

Submission: April 22, 2025
Published: May 01, 2025

Volume 3 - Issue 3

How to cite this article: Liu Yong, Jiang Jiabao*, Yan Xiaoyan, Li Shuang and Wang Zhehe. Research and Implementation of Carry-free Parallel Multiplication Based on Queue in Ternary Optical Computer. COJ Elec Communicat. 3(3). COJEC.000561.2025.
DOI: 10.31031/ COJEC.2025.03.000561

Abstract

The delay caused by carrying propagation is one of the important factors that affect the efficiency of numerical computation in computer systems. Multiplication is one of the most basic numerical operations. It is of great significance to study how to eliminate carrying delay in multiplication and how to take advantage of the characteristics of Ternary Optical Processor (TOP), such as many data bits and reconfigurable processor bits, to realize multiple parallel multipliers with different bits at the same time, in order to improve the efficiency of the TOP and to promote the application of the Ternary Optical Computer (TOC) in the field of numerical computation. To this end, in this paper, a new parallel carry-free multiplication, queue multiplication, is proposed and two queue multipliers used to parallel compute batch multiplicative data with different bits in pipelining manner are designed and implemented on TOC for the first time. Experiments show that queue multiplication occupies less hardware resources of TOP and is highly efficient compared to the previous binary iterative multiplication and the queue multiplier is easy to implement and extend.

Keywords: Carry-free parallel MSD multiplication; Pipelining multiplier; Round-robin queue; TOC; Reconfiguration

Introduction

Addition, subtraction, multiplication and division are the most basic arithmetic operations and subtraction, multiplication and division can be implemented by addition. Therefore, only hardware adders were constructed in early electronic computers. Hardware multipliers were not constructed until circuit integration reached millions of transistors. As multiplicand bits m or multiplier bits n increases, the circuit complexity of implementing a hardware multiplier rises dramatically and the construction cost, construction difficulty and computational latency also rise rapidly. Therefore, the search for efficient algorithms and new fast hardware devices to improve multiplication efficiency has attracted much attention from scholars in computer-related fields [1-2].

TOC has the characteristics as low power consumption, numerous processor bits, easily extended processor bits, per-bit allocability, per-bit reconfigurability, parallel computing and so on. It is more suitable for fast process of large and complex data than the electronic computer. In March 2017, a module of SD16, which has 192 processor bits, was developed in Optical computer Laboratory of Shanghai University. The current SD16 can be installed

with 64 modules to form ten-thousand-bit optical processor. In June 2017, a 46-bit SJ-MSD adder was reconfigured on a module of the SD16, whose computing time is independent of the number of data bits. It begins an era when the parallel MSD adder enters the practical application. SD16 lays a foundation for TOC in such fields as symbol processing, image processing, neural network and artificial intelligence, etc.

At present, the multipliers and dividers in TOC are realized by converting the operations into a series of addition (subtraction) operations in software but are not constructed with hardware directly. Previous studies related to MSD multiplication are mainly based on the principle of binary iterative summation [3-10] and the corresponding multipliers, one is called simple binary multiplier and the other is called pipelining binary multiplier, which is not yet built on TOC. In this paper, based on round-robin queue and binary iterative summation theory, a parallel carry-free MSD multiplication algorithm named queue multiplication and the corresponding multiplier named queue multiplier, are introduced. The queue multiplier consisting of one m-bit M-converter and one (m+n+1)-bit MSD adder can calculate the product of any m-bit MSD multiplicand and any n-bit MSD multiplier, and one product is produced per n machine cycles when calculates batch multiplicative data in pipelining mode. Next, experiments are presented to simultaneously build and run two queue multipliers with different operand digits on TOC, the SD16. Experiments show that the queue multiplier is efficient, easy to implement and easy to expand compared to previous binary multipliers.

Related Work on MSD Multiplication

The MSD multiplication in this paper is based on the ternary logical transformation M and MSD addition. With the completion of SJ-MSD adder on TOC [11-13], the preconditions for implementing multiplier and divider on TOC are met.

Modified signed-digit number system

In 1961, Avizienis A [14] proposed Modified Signed-Digit (MSD) number system which represents three-valued binary number system [14]. Using this number system, any decimal number $[x.y]_{10}$ can be expressed in the form of MSD as follows [12,13].

$$[x.y]_{10}=\sum_{i=0}^{w}(a_i\times 2^i)+\sum_{j=1}^{v}(b_j\times 2^{-j}) \quad (1)$$

where a_i and b_j are in {u, 0, 1}. Here u represents -1 for convenience. The weights 2^i and 2^{-j} show that the MSD number system is still a binary-based system. In TOC, the 0, 1 and u can be corresponding to the non-luminous state (W-light), the vertical polarized light state (V-light) and the horizontal polarized light state (H-light) in some definite order.

The principle of SJ-MSD adder

SJ-MSD adder includes 5 ternary logical transformations (SJ transformations) and a set of operation rules (SJ rules). The 5 transformations are in turn the transformation S_1, S_2, J_1, J_2, J_3 shown in Table 1.

Table 1: One set of logical transformations of SJ-MSD addition.

S1				S2				J1			J2			J3			J12			
b	*a*			*b*	*a*			*s1*	*s2*		*s1*	*s2*		*j2*	*j1*		*s1*	*s2*		
	0	1	u		0	1	u		0	1		0	1		0	1		0	1	H’
0	0	0	u	0	0	1	1	0	0	1	0	0	u	0	0	1	0	0	u	1
1	0	1	0	1	1	0	0	1	0	1	1	1	0	1	1	0	1	1	0	1
u	u	0	u	u	1	0	0	u	0	0	u	u	0	u	u	0	u	u	0	0

SJ rules for k-bits MSD numbers a and b are described as follows [12-13]:

Step 1: Implementing S_1 transformation on data a and b bit by bit, adding one 0 behind the transformation result s_1’ to obtain k+1 bits value s_1. At the same time, implementing S_2 transformation on data a and b bit by bit too, adding one 0 in front of the transformation result s_2’ to obtain k+1 bits value s_2.

Step 2: Because the top bit of s_2 is 0 and transformation $J_1(0, s_1) \equiv 0$, by implementing transformation J_1 on the low k bits both s_2 and s_1 bit by bit, and adding one 0 behind the result j_1’, k+1 bits value j_1 is obtained. At the same time, by implementing transformation J_2 on s_2 and s_1 bit by bit, k+1 bits value j_2 is obtained.

Step 3: By implementing transformation J_3 on j_1 and j_2 bit by bit, k+1 bits value $j_3 = a + b$ is obtained.

When constructing SJ-MSD adder in SD16, in order to save the processor bit resource of SD16 and improve the processor efficiency, one processor bit is reconfigured into the J_{12} converter in Table 1, which is used to realize the J_1 and J_2 transforms of one bit at the same time, for detailed operations, please refer to references [12-13]. In this way, only 4k+2 processor bits are needed to complete the addition of the k-bit MSD number a and b. The readers are suggested referring to the paper in details [12-13].

The principle of MSD multiplication

MSD multiplication is implemented with transformation M and SJ-MSD addition [5]. The truth table of transformation M is shown in Table 2. The transform M can be used to realize the multiplication of one bit MSD number. The multiplicand a and multiplier b are expressed as m and n bits in the paper respectively Assume $a = a_{m-1}...a_i...a_1a_0$, $b = b_{n-1}...b_j...b_1b_0$, where m, n, i and j are integer, $0 \le i \le m-1$, $0 \le j \le n-1$, a_i and b_j are in {0, 1, u}. In order to reduce the number of partial products, the shorter one of a and b is selected as the multiplier. Now $c = a \times b$ ($m \ge n$) is expressed as follows.

Table 2: The truth table of transformation M.

b	*a*		
	0	1	u
0	0	0	0
1	0	1	u
u	0	u	1

$$c = a \times b = a \times \left(\sum b_j \times 2^j\right) = \sum a \times b_j \times 2^j = \sum F_j \times 2^j = \sum P_j \qquad (2)$$

From formula (2), we know that the partial product F_j equals to a × b_j, and the operand of the sum P_j equals to $F_j \times 2^j$. The implementation principle of multiplication operation for a and b can be divided into four steps as follows:

Step 1: Generate m-bits B_j (j = m-1, ..., 1, 0) by copying the corresponding bit j of multiplier b_j m times. B_j is expressed as $B_j = b_{j(m-1)} \ldots b_{j1} b_{j0}$.

Step 2: For every B_j (j = n-1, ..., 1, 0), the transformation M is applied to a and B_j bit by bit, and n partial products (F_{n-1}, ..., F_1, F_0) can be get.

Step 3: j zeros are added to the right of each partial product F_j, so n operands of the sum (P_{n-1}, ..., P_1, P_0) are obtained.

Step 4: The sum of P_{n-1}, ..., P_1 and P_0 is obtained by using MSD addition, which is the product of a and b as follows.

$$c = P_{n-1} + \ldots + P_j + \ldots + P_1 + P_0 \qquad (3)$$

The principle of binary iterative multiplication on TOC

There are two schemes for calculating formula (3)[5]. The first summation scheme is called sequential summation, that is, first to calculate $P_1 + P_0 = P_{10}$, and then calculate $P_{10} + P_2 = P_{210}$, in this way until $P_{n-1} + P_{n-2\ldots10} = P_{n-1\ldots10}$. This scheme requires n M operations and n-1 additions. To this end, two m-bit M operators and one L-bit $(L = m + n + [\log_2^n] - 2)$ adder need to be reconfigured for building multiplier, which is named sequential multiplier. The second summation scheme is called binary iterative summation which is based on the binary iterative summation theory shown in Figure 1. Binary iterative summation principle requires $[\log_2^n]$ tiers MSD additions. Symbol Add_{ij} expresses the j-th addition of the tier i. Obviously, the number of MSD additions that tier j needs to implement is $t_i = [n/(2^i)], 1 \le i \le [\log_2 n]$. After processing, the outputs of the transformers M are the inputs of the first tier SJ-MSD adders. Similarly, the outputs of the tier i-1 SJ-MSD adders are the inputs of the tier i SJ-MSD adders (where i = 2, 3, ..., $[\log_2^n] - 1$). So, the output of the tier $[\log_2^n]$ SJ-MSD adder is the product c. If the number of MSD addition inputs in tier i-1 is odd, one MSD number with a value of 0 needs to be added to make the number of operands to be even number. Therefore, the total number of additions required to be implemented is $\sum_{i=1}^{[\log_2^n]} [n/(2^i)]$. The binary iterative summation does not reduce the total number of additions, but the additions in each tier can be executed in parallel. In Figure 1, the $K = [\log_2^n]$.

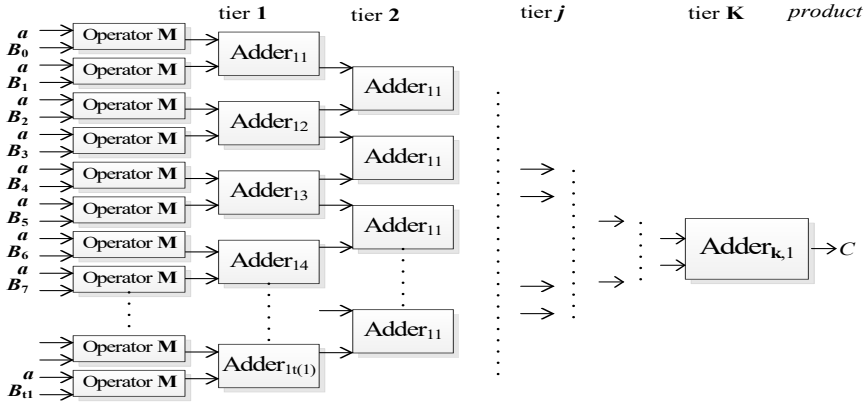


Figure 1: Schematic diagram of binary iteration summation.

In order to achieve binary iterative summation, there are two ways for building a multiplier. In method one, $2[n/2]$ m-bit operators M and $[n/2]$ L-bit adders need to be reconfigured for building multiplier, which is named simple binary multiplier. Several adders in L adders are multiplexed by the additions of each tier as shown in Figure 1. Obviously, this multiplier executes additions of each tier in parallel, but it cannot compute batch multiplicative data in pipelining manner. In method two, $2[n/2]$ m-bit operators M and $\sum_{j=1}^{[\log_2^n]} [n/(2^i)]$ L-bit adders need to be reconfigured for building multiplier, which is named pipelining binary multiplier. L adders are stratified as shown in Figure 1. Obviously, this multiplier consumes more TOP resources than simple binary multiplier and sequential multiplier, but it can compute batch multiplicative data in pipelining manner. That is to say, each tier adder can execute the corresponding tier addition of different multiplicative data in parallel.

It takes 1 clock cycle to complete an operation M and 3 machine cycles to complete one MSD addition. Obviously, to complete the operations of total multiplicative data, sequential multiplier needs T_1 = 1 + 3× (n-1) × total machine cycles, and simple binary

multiplier needs $T_2 = 1+3[\log_2^n] \times$total machine cycles, and when computing total multiplicative data in pipelining manner the pipelining binary multiplier needs $T_3 = 1 + 3[\log_2^n] +$total machine cycles. Obviously, when calculating batch multiplicative data, the efficiency of pipelining binary multiplier is obviously higher than that of simple binary multiplier and sequential multiplier.

Algorithm Design for Parallel Carry-free MSD Multiplication Based on Round-Robin Queue

The basic principle of queue multiplication, which is a new parallel carry-free MSD multiplication, is as follows.

For the m-bits multiplicand $a = a_{m-1}...a_i...a_1a_0$, n-bits multiplier $b = b_{n-1}...b_j...b_1b_0$, where m, n, i and j are integer, $0 \le i \le m-1$, $0 \le j \le n-1$, a_i and b_j are in {0, 1, u}, the queue multiplier is reconfigured on the TOP, which has only one m-bits operator M and one m+n+1 bits SJ-MSD adder. Three round-robin queues are established in working area, of which each element can store one m+n+1 bits MSD number, named queue 0, queue 1 and queue 2, respectively. The queue 0 or queue 1 has four elements for storing the additive operands output by MSD adder. The Queue 2 has two elements for storing the sum P_j output by operator M. The study shows that the MSD adder can process two multiplicative data in parallel when it computes batch multiplicative data in pipelining. The Schematic diagram of queue multiplication is shown in Figure 2.

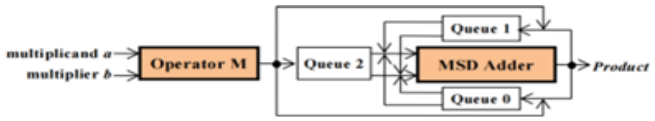


Figure 2: Schematic diagram of queue multiplication.

In Figure 2, Only if the length n of the multiplier is odd, P_{n-1} of data A goes into queue 0, P_{n-1} of data B goes into queue 1, and the rest of P_j goes into queue 2. The two inputs of the SJ-MSD adder come from either queue 0 or queue 1 or queue 2. If the output of MSD adder is a partial product, then the partial product of data A goes into the Queue 0, and the partial product of data B goes into the Queue 1; otherwise, the output of adder is stored in the product area. To control the inputs / outputs of operator M and SJ-MSD adder, a series of control variables need to be set, which is the key to queue multiplication.

Setting and initialization of control variables for queue multiplication

In queue multiplication, for implementing MSD multiplication of any m-bits multiplicand and any n-bits multiplier, the control variables are set as follows.

The variable MIO is used to control the input and output of operator M. The variable SIO is used to control the input and output of adder. The variable Mout is the identifier of queue for storing the output of operator M. Mout = 0 or 1 indicates that the output of adder should be stored in Queue 0 or Queue 1. The variable SJout is the identifier of queue for storing the output of adder. SJout = 0 or 1 indicates that the output of adder should be stored in Queue 0 or Queue 1. The variable SJin is the queue identifier for the input source of the adder. SJin = 0 or 1 indicates that the input of adder comes from Queue 0 or Queue 1. The array AlterOut[SIO] is used to judge if the variable SJout should be modified. The array AlterIn[SIO] is used to judge if the variable SJin should be modified. The array InYes[SIO] is used to judge if the current adder output is stored in Queue 0 or Queue 1. To control the end of the algorithm, the variable Products is used to record the number of products that the adder has output.

The above control variables are initialized as following C program pseudo-code.

int Begin [9] = {1,5,9,12,13,16,20,20,22};

MIO = n + 1; Mout = SJout = 0; Products =-1;

if (0 = = n%2) SJin = 1; else SJin = 0;

If n is less than 10, then SIO = Begin[n-1]; if n is an even number greater than or equal to 10, then SIO = n + 12, otherwise, SIO = n + 14.

For (i=0; i<SIO; i++) {InYes[i] = 1; AlterOut[i] = AlterIn[i] = 0;}

According to the parity of n, some elements of array InYes are modified to 0, and some elements of array AlterOut and array AlterIn are modified to 1. Take n as 8 for example as following.

If (n = = 8) {AlterIn [0] = AlterIn [8] = AlterOut [2] = AlterOut [3] = AlterOut [4] = AlterOut [5] = AlterOut [6] = 1; InYes [1] = InYes [7] = InYes [9] = InYes [11] = InYes [13] = 0; for (i = 15; i < SIO; i++) InYes[i] = 0;}

The algorithm of queue multiplication

For any m-bits MSD multiplication a and any n-bits MSD multiplier b, the flow chart of the algorithm for queue multiplication is shown in Figure 3. It is not difficult to achieve that one product is produced per n machine cycles after the first product output for any m-bits MSD multiplicand and any n-bits MSD multiplier if the queue multiplier computes batch data in pipelining.

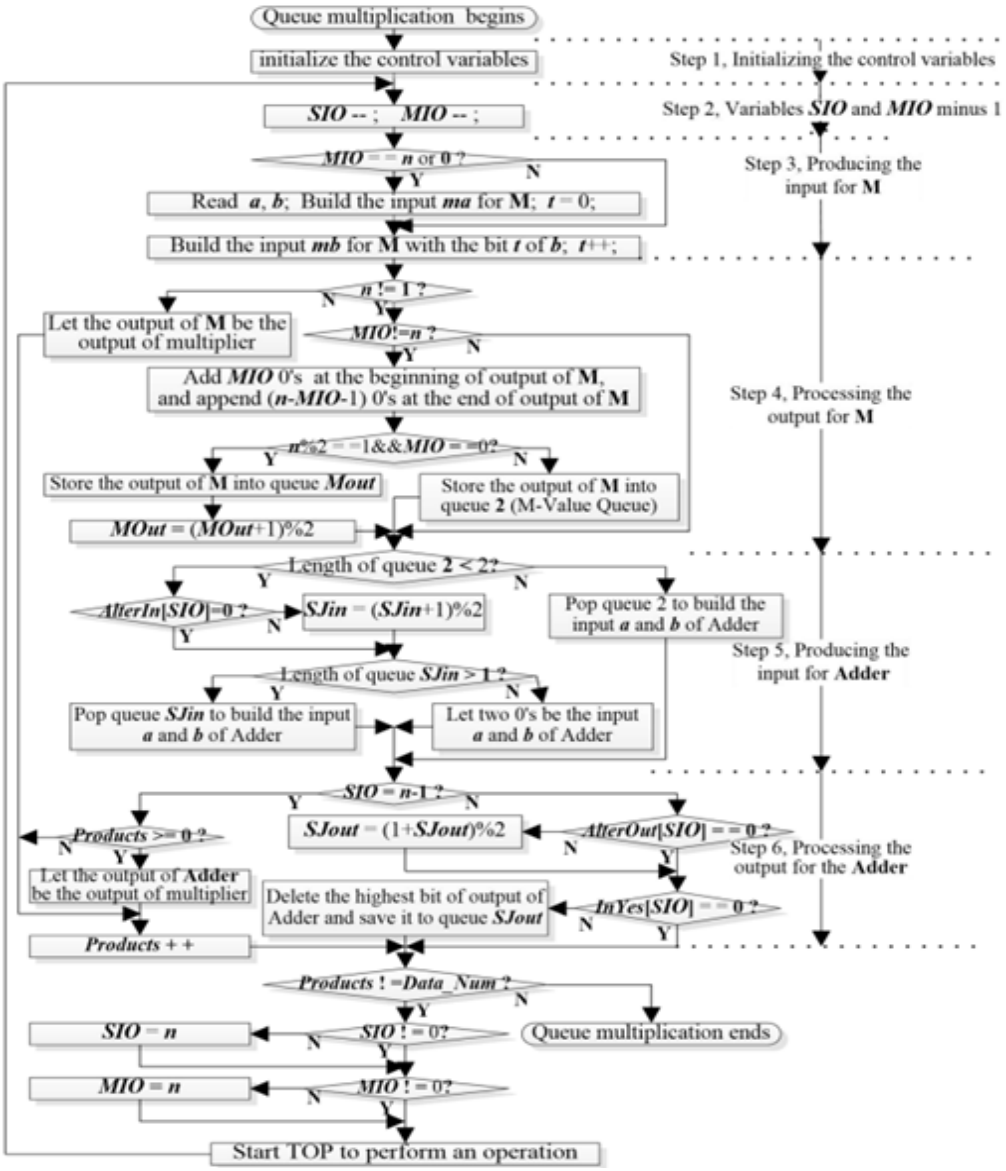


Figure 3: The flow chart of queue multiplication.

Advantages of Queue Multiplication

Now we compare queue multiplication and binary iterative multiplication from the following three aspects: the consumption of processor bit resources when performing the same computing tasks in the same time, the difficulty of implementing multiplication and the extensibility of multiplier. Firstly, an m-bits × n-bits pipelining binary multiplier occupies the processor bits T_B shown in Equation (4), and an m-bits × n-bits queue multiplier occupies m+4(m+n+1) +2 processor bits. Pipelining calculates a batch of m-bits × n-bits multiplicative data, when the pipeline fills up, the binary iterative multiplier outputs one product per clock cycle, and the queue multiplier outputs one product every n machine cycles. Obviously, n queue multipliers and 1 binary iterative multiplier have the same efficiency. It occupies processor bits T_Q is shown in Equation (5). we can see that the relation between $y = T_B - T_Q$ and n is shown in Figure 4.

Figure 4 shows that for the same efficiency, the queue multiplier occupies significantly fewer processor bits.

$$T_B = 2m[n/2] + (4L+2)\sum_{j=1}^{[\log_2 n]}\left[n/(2^j)\right] = 2m[n/2] + (4m+4n-6+4[\log_2 n])\sum_{j=1}^{[\log_2 n]}\left[n/(2^j)\right] \quad (4)$$

$$T_Q = x[m+4(m+n+1)+2] = x[5m+4n+6] = n[5m+4n+6] = 5nm+4n^2+6n; \quad (5)$$

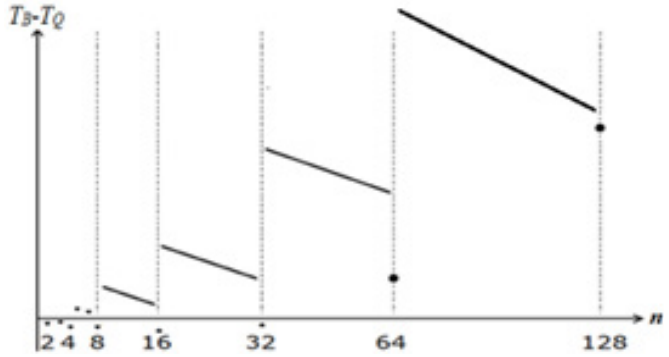


Figure 4: The diagram of relationship between y and n.

Secondly, with the increase of multiplier bits n, the number of operators M and adders for binary iterative multiplication will also increase, and the number of tiers of adders will also increase, so the processing of intermediate results and controlling for pipelining calculation will become more complicated, while the queue multiplication always requires only one operator M and an adder, and the processing of intermediate results and controlling for pipelining calculation are fixed and simple.

Thirdly, when multipliers are need to be added, adding a binary iterative multiplier requires T_B processor bits to construct n m-bits M-converters and $\sum_{i=1}^{[\log_2^n]}\left[n/(2^i)\right]$ SJ-MSD adder with the same or different bits, while adding a queue multiplier requires only (5m+4n+6) processor bits to construct an m-bit M-converter and an (m+n+1)-bit SJ-MSD adder, so the queue multiplier is easier to extend than the binary iterative multiplier. In summary, queue multipliers have a broader application prospect than binary iterative multipliers.

Design and Implementation of Queue Multiplier

Experiment device

The main experimental device is a module of SD16. Each module of SD16 has 576 pixels arranged in 24 rows and 24 columns as shown in Figure 5. Each small square in Figure 5 represents a pixel and three neighbouring pixels form a processor bit, which is numbered from 0 to 191, with pixel 0 in the middle of the lowest row. The three pixels per processor bit on the left half of TOP are, in order, w-pixels, v-pixels, and h-pixels, and the opposite is true for the right half. Each pixel can output either W-light or V-light or H-light. According to the decrease-radix design principle [14,15], any bit of a ternary logic operator can be composed of not more than six simplest basic operating units, so each processor bit of SD16 can constitute one bit of any ternary operator.

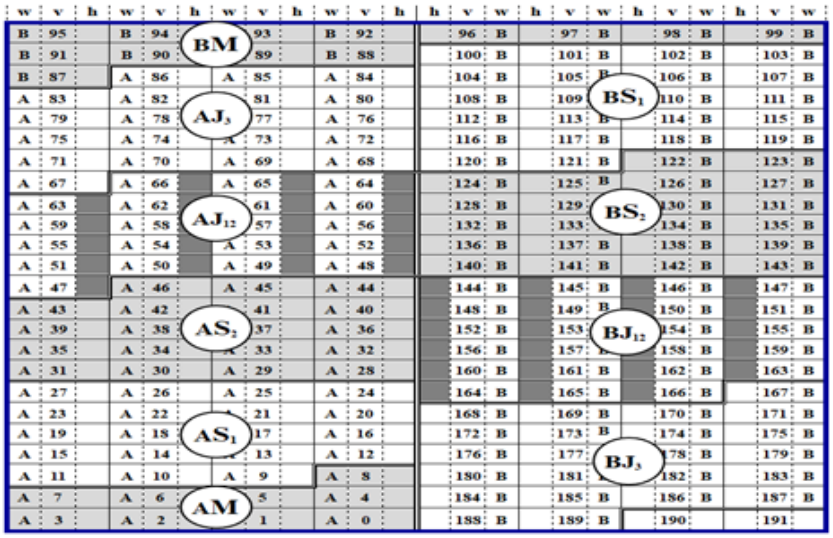


Figure 5: SD16 processor bits arrangement and multiplier distribution.

Experimental implementation

To verify the correctness of queue multiplication and that multiple queue multipliers can be constructed on the TOP to simultaneously calculate multiple batches of data with the same or different bits in pipelining manner, Both Queue Multiplier A and Queue Multiplier B are implemented on the SD16 Processor Module 1 and tested them thoroughly. The details are described as follows.

A. Processor bit allocation for the multiplier. To verify that the m-bit × n-bit multiplier outputs a product every n machine cycles after outputting the 1st product, Multiplier A is a 9-bit × 9-bit multiplier whose input data is at most a 9-bit MSD number, and the output product is a 20-bit MSD number, and Multiplier B is a 13-bit × 8-bit multiplier whose input is at most a 13-bit MSD number of the multiplicated number, the multiplication is at most an 8-bit MSD number, and the output product is a 23-bit MSD number. The processor bit allocation is shown in Figure 5 and Table 3, where S_1, S_2, J_{12} and J_3 converters form the SJ-MSD adder [12]. See reference [12] for the input/output data handling of the queue multiplier when computing batch multiplicative data in pipelining mode.

Table 3: The processor bits allocation table for two multipliers.

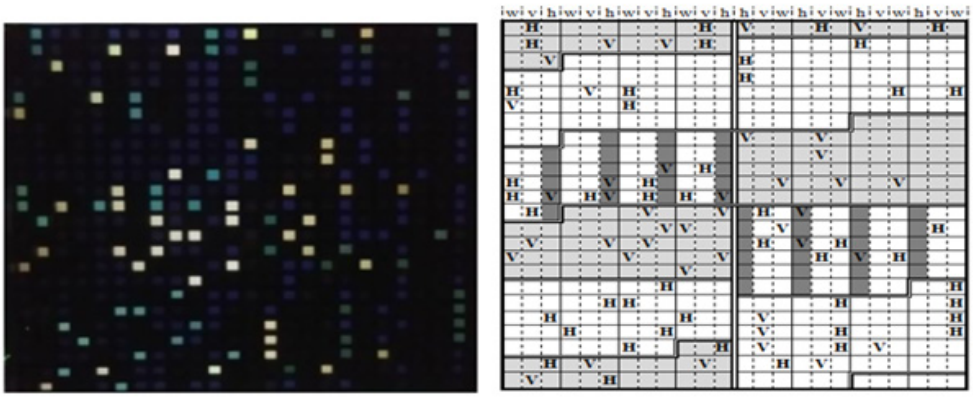
Multiplier	Operator M	Operator S1	Operator S2	Operator J12	Operator J3
A	0 ~ 8	9 ~ 27	28 ~ 46	47 ~ 66	67 ~ 86
B	87 ~ 99	100 ~ 121	122 ~ 143	144 ~ 166	167 ~ 189

B. Experimental case preparation. The experimental cases are prepared as shown in Table 4, where the inputs for case 1 to case 11 are random data whose insufficient higher bits are supplemented by 0 (0 Expressed by the symbol ϕ), while the inputs for cases 12 through 20 are boundary values. The data in the clock column in Table 4 is the clock cycle when multiplier outputs the product.

Table 4: Experimental cases table.

Multiplier A					Multiplier B			
Case	Multiplicand	Multiplier	Products	Cycle	Multiplicand	Multiplier	Products	Cycle
1	10u1u0uu1	11uuu00uu	001u0u1u10u0uu1u000u	22	10u1u01uu10u1	1u0u0u1u	00001u01u1u 0u1u001u0u0u	20
2	1uu0u0u1u	uu111100u	000000uu01u0u00u0u0u	31	uu10u1u0u1u00	u101u101	0001u0001u1uu 1uu1u01u00	28
3	uu1011u00	u0u1u0u1u	0010u0u1u0000u001u00	40	1u1u10011uu1u	ϕ 111u0uu	0001u01u0u1uu 10u1u0u01u	36
4	u101u0101	u01111uuu	00001u000u000u001u1u	49	10uu010111uu0	11uu1u00	001u0u0010u00 0u00u1u000	44
5	11101uu1u	u11uuu10u	0000uu0u001u0000u00u	58	uu0uuu01uu101	u0110110	001u000000u00 0u1u0000u0	52
6	ϕ 1111u0u u	ϕ ϕ 1011uu0	00001u1uu01u10u0u1u0	67	11u1u00uu00uu	u111100u	000000u1u01u u1u0u1u0u0u	60
7	11u011uu0	uu0u11u01	00u00000u0u1u000u1u0	76	ϕ ϕ uuu101uu10u	u0111uuu	00001u000000 0u001u00u0u	68
8	1uu1u1u00	1u11u10uu	00001u0010u000u01u00	85	u0011uu1u0u1u	u110u101	00001u000u00 000u000u01u	76
9	uu0uuu101	0u111u11u	00001uu00001uu01u01u	94	ϕ u010u1u1u001	1u11u10u	0000u10u0u0u1 u00uu01u0u	84
10	u01110110	1u0u1u001	00000u01u000u0u1u0u0	103	ϕ uu10u111u11u	11u011u0	000u1uu1u1u0 000u0u001u0	92
11	1uu1101u1	ϕ ϕ ϕ 00u1011	0000000000u00u00001u	112	1uuuu101u1u00	1u011uu0	0000001u00000 10u0u1u000	100
12	0	0	0	121	0	0	0	108
13	0	111111111	0	130	0	11111111	0	116
14	0	uuuuuuuuu	0	139	0	uuuuuuuu	0	124
15	111111111	0	0	148	1111111111111	0	0	132
16	111111111	111111111	01000u1111000000001u	157	1111111111111	11111111	01000u111011 1110000001u	140
17	111111111	uuuuuuuuu	0u001uuuuu000000000u	166	1111111111111	uuuuuuuu	0u01uuuuu0uu uuu0000000u	148
18	uuuuuuuuu	0	0	175	uuuuuuuuuuuuu	0	0	156
19	uuuuuuuuu	111111111	0u001uuuuu000000000u	184	uuuuuuuuuuuuu	11111111	0u01uuuuu0uu uuu0000000u	164
20	uuuuuuuuu	uuuuuuuuu	01000u1111000000001u	193	uuuuuuuuuuuuu	uuuuuuuu	01000u111011 1110000001u	172

C. Theoretical image preparation. The output state images of TOP at each product output are theoretically analysed and plotted. For the experimental cases given in Table 4, a total of 38 theoretical output images of TOP were plotted, and three of them are given in Figure 6, where the characters H, V and W indicate that the pixel outputs H-light, V-light and W-light in that order.

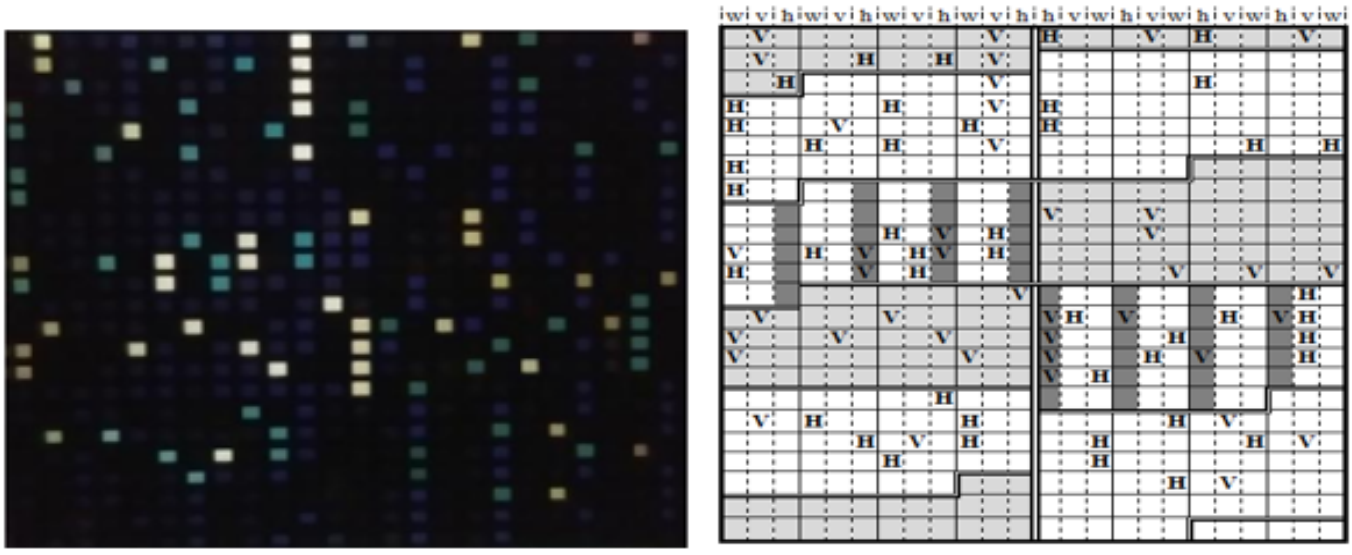


(a) Actual output (b) Theoretical output

Figure 6: The output of the 20th machine cycle.

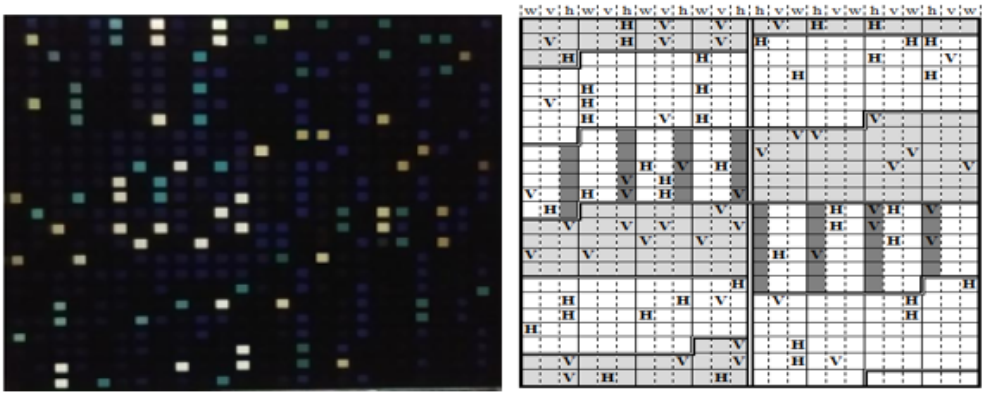
D. Experimental execution. The multipliers A and B are reconfigured as shown in Figure 5, and the input data given in Table 4 are fed into the TOP in turn, and then the "single-step operation instruction" is issued to manipulate the operation of TOP. When product outputting, the product is recorded in Table 4, the image output from TOP is captured and compared with the corresponding theoretical image and if it is different, the cause is found and processed accordingly.

As examples, Figure 6-8 give the actual output images and theoretical output images of TOP for machine cycle 20, 22 and 76, respectively. At clock cycle 20 queue multiplier B outputs the product of case 1, at clock cycle 22 queue multiplier A outputs the product of case 1, at clock cycle 76 queue multiplier A outputs the product of case 7 and queue multiplier B outputs the product of case 8. In Figure 6-8(a) is the actual output image of the processor, Figure 6-8(b) is the corresponding theoretical output image.



(a) Actual output (b) Theoretical output

Figure 7: The output of the 22th machine cycle.



(a) Actual output (b) Theoretical output

Figure 8: The output of the 76th machine cycle.

Summary

In order to fully utilize the advantages of TOC such as numerous bits, reconfigurable, distributable processor bits and processor bits easy to be expanded and so on, a new scheme of carry-free parallel MSD multiplication, queue multiplication, is proposed in this paper, which uses only one operator M and one SJ-MSD adder. At the same time, the method of implementing multiple queue multipliers on TOP is described in detail. Experiments show that the queue multiplication is correct and feasible. Compared with binary iterative multiplication, queue multiplication takes less hardware resources of TOP, and the queue multiplier is easy to be expanded and implemented. It can be seen that queue multiplication will play a positive role in promoting TOC into practical application areas such as numerical computation.

Conclusion

The research results of this paper are applied to the construction of multipliers in ternary optical computers and multi value computing electronic computers, providing a method for the construction of dividers and other arithmetic units in ternary optical computers and multi value computing electronic computers and promoting the entry of ternary optical computers and multi value computing electronic computers into the practical computing field.

Funding

Natural Science Foundation of Department of education of Anhui Province, China (KJ2020A0681), 2021 Quality Engineering Project of Anhui Province, China (2021xsxxkc191) and Hainan Provincial Natural Science Foundation of China (622MS084).

References

1. Chuang CH, Lin CL (2014) A novel synthesizing genetic logic circuit: Frequency multiplier. IEEE ACM T COMPUT BI 11(4): 702-713.
2. Sun ZW (2011) Design and verification of hardware core of configurable 18-bit multiplier based on optimized booth algorithm. Master Thesis, Xi'an Electronic Science and Technology University, Xi'an.
3. Wang X, Peng J, Li M, Shen Z, Shan O (2010) Carry-free vector-matrix multiplication on a dynamically reconfigurable optical platform. Appl Opt 49(12): 2352-2362.
4. Peng J, Wei X, Zhang X, Shen Y, Fu Y (2017) Implementation of parallel fast Fourier algorithm based on ternary optical computer. Sci China Ser F-Inf Sci 47(7): 846-862.
5. Kong S (2018) Design and implementation of third platform for MSD multiplication on ternary optical computer. Master Thesis, Shanghai University, Shanghai, China.
6. Xu Q, Wang X, Xu C (2017) Design and implementation of the modified signed digit multiplication routine on a ternary optical computer. Appl Opt 56(16): 4661-4669.
7. Shen R (2014) Research on key technologies of ternary optical computer multiplier. Master Thesis, Shanghai University, Shanghai, China.
8. Xiao-jun H, Yi J, Shan O (2014) A 40-bit multiplication routine of ternary optical computer. Journal of Shanghai University 20(5): 645-657.
9. Hu XJ (2013) Ternary optical computer integer multiplication routine. Master Thesis, Shanghai University, Shanghai, China.
10. Nie YM (2008) A dual-DMD vector matrix multiplier for coherent light. Master Thesis, China University of National Defense Science and Technology, Hunan, China.
11. Jiang JB, Chen XL (2016) Hardware implementation of converting ternary optical computer MSD into standard binary data. Chinese J Nanjing Univ Sci & Tech 40: 278-284.
12. Jiang JB, Zhang XF (2021) Design and implementation of SJ-MSD adder in ternary optical computer. Acta Electronica Sinica 49(2): 275-285.
13. Jiabao J, Yunfu S, Shan O, JunJie P, Xianchao W (2020) The application of SJ-MSD adder to mean value filtering processing. Optik-International Journal for Light and Electron Optics 206: 164271.
14. Avizienis A (1961) Signed digit number representation for fast parallel arithmetic. IRE Trans Electron Comput 10(3): 389-400.
15. Yan JY, Jin Y, Zuo K (2008) Decrease-radix design principle for carrying/borrowing free multi-valued and application in ternary optical computer. Sci China Ser F-Inf Sci 51: 1415-1426.