The Design of an Optoelectronic Arithmetic Processor Based on Permutation Networks

Ming-Bo Lin, *Member*, *IEEE*, and A. Yavuz Oruç, *Senior Member*, *IEEE*

Abstract—This paper introduces a new concept by which it is possible to design and implement arithmetic processors using permutation networks. To demonstrate this concept, several optoelectronic arithmetic units combining optical directional coupler switches and cyclic permutation networks are designed. The designs show that addition, subtraction, and multiplication can all be performed in $O(\log n)$ time in residue code domain and using $O(n^2)$ directional coupler switches and gates, where $n = \log M$, and M is the integer range of interest. These arithmetic units also have the capability of concurrent error detection and fault-tolerance, and they can be used to construct constant time inner product processors.

Index Terms—Directional coupler switch, modulo arithmetic, optical computing, permutation network, residue codes.

1 Introduction

A major disadvantage of conventional arithmetic algorithms is that their realizations require different hardware structures. For example, addition and multiplication cannot use the same hardware unless a repeated-addition algorithm is used to perform multiplication operations. But this means a reduction in speed which, in general, is not tolerable. Another disadvantage of conventional arithmetic algorithms is that they cannot easily be parallelized due to carry propagation.

One way to alleviate these disadvantages is to transform arithmetic operations into a residue code domain and then perform the requisite computations in this domain. In the residue code domain, each binary number is represented as a set of residue digits and the arithmetic operations are performed on residue digits in parallel. This provides carry-free realizations of binary addition, multiplication, and other operations, but these realizations typically do not have any overlap, and require different hardware for each operation as in conventional algorithms [16], [27].

Another way to overcome the disadvantages of conventional arithmetic algorithms is by using optical computing [3], [5]. The motivation behind optical computing is the observation that photons in optical fibers, optical integrated circuits, and free space travel faster than electrons do in electronic circuits since they do not have to charge a capacitor. Furthermore, photons are uncharged and do not interfere with one another as readily as electrons. This implies that optical signals can be easily handled in parallel [20].

Many implementations of optical logic operations have been suggested in the literature. Among these, spatially invariant and spatially variant techniques are dominant. Spatially invariant techniques include symbolic substitution logic (SSL) which is implemented by additive methods [5], [6] or convolution techniques [7], [12], shadow-casting logic [28], programmable logic [21], [22], and the combinatorial logic-based system [10].

The basic idea behind SSL is to first detect the presence of one or more patterns in an input plane and then substitute an appropriate pattern for each detected pattern. The major disadvantage of this approach is that, for n inputs, it requires $\Omega(2^n)$ hardware [19].

In the programmable logic technique, a set of n-input variables are mapped to an input array along with their complements. A series of $n + 1$ interconnection stages consisting of one crossover network and one mask per stage is then used, along with $n + 1$ arrays of AND gates, to generate all the minterms of the desired function. Finally, the appropriate minterms are combined through a similar series of $n + 1$ interconnection stages, along with $n + 1$ arrays of OR gates, to generate the function. Thus, the total number of gates required by this technique is $O(n2^n)$.

The basic idea of the spatially variant technique proposed in [19] is to build a set of the spatially variant elementary logic functions. These elementary functions are used to construct larger functions as in conventional logic designs. As an example, an n-bit carry look-ahead adder can be implemented by this approach with $O(n^3)$ gates and $O(\log n)$ delay.

All these approaches, however, are based on full spatial optical principles, and they could not be easily realized by integrated optics [20]. Recently, a number of optical computing schemes that mix electronic and optical techniques have also been proposed [1], [2]. These mainly use directional coupler switches as building blocks. In essence, a directional coupler switch is a five-terminal gate with two inputs, two outputs, and one control input. The two outputs are logical functions of two throughgoing input signals and the control input signal.

- *M.-B. Lin is with the Electronic Engineering Department, National Taiwan Institute of Technology, 43, Keelung Road Section 4, Taipei, Taiwan. E-mail: mblin@et.ntit.edu.tw.*
- *A.Y. Oruç is with the Electrical Engineering Department, Institute of Advanced Computer Studies, University of Maryland, College Park, MD 20742.*

Manuscript received July 1994; revised Apr. 26, 1996.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96325.

Benner et al. [1], [2] have proposed a scheme that shows how to design oscillators and divide-by-N counters using optical directional coupler switches. Although such gates do not constitute truly all optical devices, the logical input-output signals are optical and the use of electronic signals in the control input provides some flexibility that is not yet available in well-developed optical switches. This also provides an opportunity for optical computing based on integrated optics [20].

Recently, Lea [14] designed photonic interconnection networks using directional coupler switches. In these networks, electronic switches are replaced by directional coupler switches, and single-mode waveguides are used to establish the connections between directional coupler switches. Historically, interconnection networks were introduced to enhance the communication bandwidth in telephone systems and parallel processors. In this paper, we explore the possibility of using interconnection networks to perform arithmetic operations. The rationale behind this is that any algebraic computation can be viewed as a sequence of permutation operations. The key problem is how to pick up an appropriate mapping function that could map all of the desirable input patterns into corresponding permutations. Thus, the underlying network must have enough permutations so that it can cover all of the input patterns.

We refine these ideas, and combine optical directional coupler switching, residue arithmetic, and permutation networks to obtain a novel optoelectronic computational model, and to develop various optoelectronic arithmetic processor modules. Optical devices reduce the signal propagation delay and therefore increase the computation speed. Permutation networks provide a way of computation which replaces the propagation delays in conventional logic circuits by the transmission delay of light through waveguides, and the residue arithmetic keeps the cost of permutation networks to an acceptable level.

The remainder of the paper is organized as follows: Section 2 summarizes the mathematical preliminaries used in this paper. Section 3 shows how to design addition, subtraction, and multiplication modules, and Section 4 shows how to design an input encoder and output decoder using directional coupler switches and cyclic permutation networks. Section 5 analyzes the performance of various arithmetic modules designed in the previous sections. The paper is concluded in Section 6.

2 Preliminaries

We begin with a review of some basic mathematical facts [8], [23].

2.1 Finite Groups

Let a, b, and n be integers with n being positive. Then the expression

$$a \equiv b (\text{mod } n)$$

called a congruence, holds if and only if $a - b = kn$ for some integer k. n is called the modulus and b is called a residue of a and vice versa. Let a be an integer and $[a]_n$ denote the set of integers q satisfying

$$q \equiv a (\text{mod } n).$$

The set $[a]_n$ is called the congruence class modulo n of a. Writing $b \in [a]_n$ is the same as writing $b \equiv a (\text{mod } n)$. The set of all such congruence classes is denoted as:

$$Z_n = \{[a]_n : 0 \le a \le n-1\} \tag{1}$$

or more simply as:

$$Z_n = \{0, 1, \ldots, n-1\}. \tag{2}$$

For a given modulus n, define the operations $+_n$ and $\times_n$ as

$$[a]_n +_n [b]_n = [c]_n \text{ if and only if } a + b \equiv c (\text{mod } n) \tag{3}$$

$$[a]_n \times_n [b]_n = [c]_n \text{ if and only if } ab \equiv c (\text{mod } n) \tag{4}$$

Then $(Z_n, +_n)$ forms an additive group of order n. Similarly, the set $Z_n^* = \{[a]_n\} \in Z_n : gcd(a, n) = 1\}$, that is, the set of all elements relatively prime to n forms a multiplicative group of $\phi(n)$ elements, where $\phi(n)$ is called the Euler ϕ function. If p is prime, then $Z_p^* = \{1, 2, \ldots, p-1\}$ and $\phi(p) = p - 1$. If n is composite, then $\phi(n) < n - 1$. In this paper, we only consider the case that p is prime.

A group G is called cyclic if there is an element g in G such that $G = \{g^n \mid n \in Z\}$. Element g is called a generator of G. The multiplicative group Z_p^* can be shown to be a cyclic group and therefore has a single generator, also called a primitive element. The least positive primitive elements for the first 35 primes are listed in Table 1.

An isomorphism f from a group $(G, \circ)$ to a group $(\overline{G}, \diamond)$ is a one-to-one and onto (i.e., bijection) mapping from G to $\overline{G}$ that preserves the group operation. That is,

$$f(a \circ b) = f(a) \diamond f(b) \quad \textit{for all } a, b \in G \tag{5}$$

If there is an isomorphism from G to $\overline{G}$, we say that G and $\overline{G}$ are isomorphic and denote it by $G \simeq \overline{G}$.

THEOREM 1. *Every finite cyclic group with order n is isomorphic to* $(Z_n, +_n)$.

THEOREM 2. $(Z_p^*, \times_p)$ *is isomorphic to* $(Z_{p-1}, +_{p-1})$, *where p is prime.*

Now we define permutations and permutation groups. A permutation of a set A is a function from A to A, that is both one-to-one and onto. A permutation group of a set A is a set of permutations of A that forms a group under function composition. The degree of a permutation group on a set A is the cardinality of A.

A convenient way to denote a permutation is by using an array notation. Let $A = \{0, 1, \ldots, n-1\}$ be a set of n elements and α be a permutation on set A. Then we write

$$\alpha = \begin{pmatrix} 0 & 1 & 2 & \ldots & n-1 \\ \alpha(0) & \alpha(1) & \alpha(2) & \ldots & \alpha(n-1) \end{pmatrix}. \tag{6}$$

The set of all permutations of n elements is called the symmetric group of degree n and is denoted by S_n.

Composition of permutations expressed in array notation is carried out from left to right by going from top to

TABLE 1
PRODUCT AND SUM OF FIRST 35 CONSECUTIVE PRIMES

order	m_i	$\lceil \log m_i \rceil$ (bits)	$\sum m_i$	$\sum \lceil \log m_i \rceil$ (bits)	$\lceil \log \prod m_i \rceil$ (bits)	least positive primitive root
1	2	1	2	1	1	1
2	3	2	5	3	2	2
3	5	3	10	6	4	2
4	7	3	17	9	7	3
5	11	4	28	13	11	2
6	13	4	41	17	14	2
7	17	5	58	22	18	3
8	19	5	77	27	23	2
9	23	5	100	32	27	5
10	29	5	129	37	32	2
11	31	5	160	42	37	3
12	37	6	197	48	42	2
13	41	6	238	54	48	6
14	43	6	281	60	53	3
15	47	6	328	66	59	5
16	53	6	381	72	64	2
17	59	6	440	78	70	2
18	61	6	501	84	76	2
19	67	7	568	91	82	2
20	71	7	639	98	88	7
21	73	7	712	105	95	5
22	79	7	791	112	101	3
23	83	7	874	119	107	2
24	89	7	963	126	114	3
25	97	7	1,060	133	120	5
26	101	7	1,161	140	127	2
27	103	7	1,264	147	134	5
28	107	7	1,371	154	140	2
29	109	7	1,480	161	147	6
30	113	7	1,593	168	154	3
31	127	7	1,720	175	161	3
32	131	8	1,851	183	168	2
33	137	8	1,988	191	175	3
34	139	8	2,127	199	182	2
35	149	8	2,276	207	189	2

bottom, then top to bottom. Let

$$\alpha = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 3 & 0 & 2 & 4 & 1 \end{pmatrix} \text{ and } \beta = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 0 \end{pmatrix} \quad (7)$$

then

$$\alpha\beta = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 3 & 0 & 2 & 4 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 0 & 2 \end{pmatrix} = (0423)(1). \quad (8)$$

The cycle notation (0 4 2 3)(1) expresses the same permutation in a more compact way.

THEOREM 3 (cyclic permutation group). *A set of permutations generated by permutation function*

$$\alpha(i) = i + 1 \bmod n, \text{ for } 0 \le i \le n - 1 \quad (9)$$

forms a cyclic group of order n under function composition.

Finite groups are related to permutation groups by the following well-known Cayley Theorem.

THEOREM 4 (Cayley Theorem [8]). *Every finite group is isomorphic to a permutation group.*

Cayley Theorem gives a constructive rule for an isomorphism, which is called the regular representation of the group. If $(G, \circ)$ is a group, Cayley Theorem states that the (right) regular representation of G is given by

$$R = \{T_a \mid a \in G\}$$

where the bijection T_a is defined as

$$T_a : b \to a \circ b \quad \forall b \in G.$$

As an example, the regular representation of $(Z_n, +_n)$ is given by $(\overline{G}, \circ)$, where

$$\overline{G_n} = \{T_i \mid 0 \le i < n\}$$

and T_i is given by

$$T_i : j \to i +_n j \quad \forall j; 0 \le j < n.$$

The next corollary immediately follows.

COROLLARY 1. *Every cyclic group is isomorphic to a cyclic permutation group.*

2.2 Realizations of Permutation Groups

In the previous section, we have established the relationship between a finite group and a permutation group through the well-known Cayley Theorem. In this section, we will show how to realize a permutation group on a permutation network.

A permutation network is an interconnection network with an input port and an output port, where each port has s lines, labeled from top to bottom as 0, 1, …, $s - 1$. An example is shown in Fig. 1. To represent the permutation map

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 3 & 4 & 0 & 1 & 2 \end{pmatrix},$$

input 0 is connected to output 3, input 1 to output 4, and so on. Fig. 1 shows the composition of two permutations. The time taken to add two numbers is basically the time taken to compose two permutations, which depends on the kind of permutation networks used. Although we show a direct connection between the inputs and outputs, this is in general not necessary. All that is required is that there be a path from each input to a distinct output as specified by the permutation map.

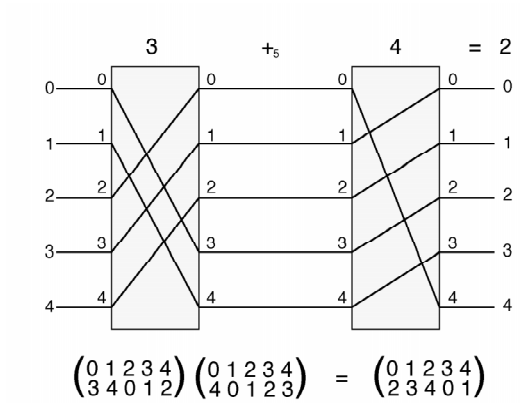


Fig. 1. Permutations for 3 $+_5$ 4 = 2.

One way to realize a permutation network such as the one shown in Fig. 1 is to use a rearrangeable interconnection network [4], [29]. However, for n inputs, these networks use $O(n \log n)$ switches, and they are not an attractive choice to realize cyclic groups of order n as the latter only requires n permutations. For cyclic groups such as those we will consider in this paper, a cyclic permutation network, performing n permutations, is sufficient as implied by Cayley theorem. However, even with n permutations, the realization can become prohibitively difficult as n gets large if we use the Cayley isomorphism. The reason is that the Cayley isomorphism of a cyclic group will exact n permutations, where n is the order of the group. This problem can be overcome by decomposing the group into a set of subgroups whose orders are relatively prime. We then apply the Cayley regular representation to each subgroup.

DEFINITION 1. *Let* $(G_1, \circ_{G_1}), (G_2, \circ_{G_2}), \ldots, (G_r, \circ_{G_r})$ *be a collection of groups. The direct product of* $G_1, G_2, \ldots, G_r$, *written as* $G_1 \times G_2 \ldots \times G_r$ *is the set of all r-tuples for which the* ith *component is an element of* G_i *and the operation is componentwise, that is,*

$$G_1 \times G_2 \times \ldots \times G_r = \{(a_1, a_2, \ldots, a_r) \mid a_i \in G_i\}$$

where $(a_1, a_2, \ldots, a_r)(b_1, b_2, \ldots, b_r)$ *is defined to be* $(a_1 \circ_{G1} b_1, a_2 \circ_{G2} b_2, \ldots, a_r \circ_{Gr} b_r)$.

In general, $G_1, G_2, \ldots, G_r$ are groups with the same operation but different orders.

An example is shown in Fig. 2. $(Z_{30}, +_{30})$ is decomposed into three smaller groups, $(Z_2, +_2)$, $(Z_3, +_3)$, and $(Z_5, +_5)$. These groups are then mapped into permutation groups by using Cayley Theorem, respectively. In terms of permutations, three disjoint cycles with order 2, 3, and 5 are used to represent the groups $(Z_2, +_2)$, $(Z_3, +_3)$, and $(Z_5, +_5)$, respectively.

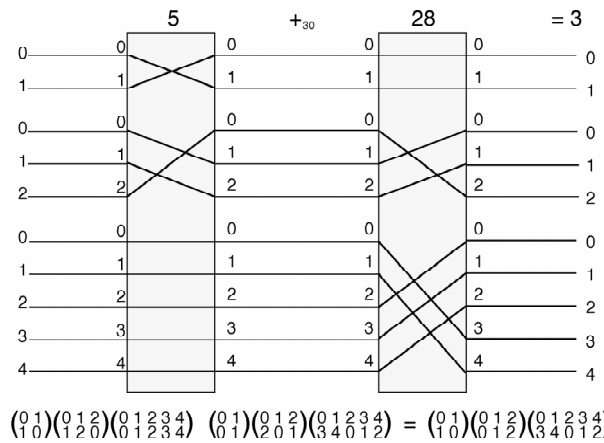


Fig. 2. Disjoint cycle permutations representing 5$+_{30}$ 28 = 3.

Now, we extend the above direct product group into a direct product of commutative rings as follows.

DEFINITION 2. *Suppose* $R_1, R_2, \ldots, R_r$ *are commutative rings with an identity element. Then the direct product* $R_1 \times R_2 \times \ldots \times R_r$ *is the set of r-tuples* $(x_1, x_2, \ldots, x_r)$ *where* $x_i \in R_i$ *for* $i = 1, 2, \ldots, r$, *with addition and multiplication defined componentwise, that is*

$$(x_1, x_2, \ldots x_r) +_{m_1, m_2, \ldots, m_r} (y_1, y_2, \ldots, y_r) = (x_1 +_{m_1} y_1, x_2 +_{m_2} y_2, \ldots, x_r +_{m_r} y_r) \quad (10)$$

$$(x_1, x_2, \ldots x_r) \times_{m_1, m_2, \ldots, m_r} (y_1, y_2, \ldots, y_r) = (x_1 \times_{m_1} y_1, x_2 \times_{m_2} y_2, \ldots, x_r \times_{m_r} y_r) \quad (11)$$

The zero is (0, 0, …, 0) *and the identity element is* (1, 1, …, 1). *It is readily seen that* $R_1 \times R_2 \times \ldots \times R_r$ *is also a commutative ring with identity element.*

2.3 Input Encoding and Output Decoding

The input encoding procedure is based on the following theorem [25].

THEOREM 5. *Let* $M = m_1 m_2 \ldots m_r$ *where* $gcd(m_i, m_j) = 1$ *if* $i \neq j$. *Then there is an isomorphism between* Z_M *and*

$$Z_{m_1} \times Z_{m_2} \times \ldots \times Z_{m_r}.$$

The isomorphism is defined as

$$X \equiv x_i (\text{mod } m_i) \quad \text{for} \quad i = 1, 2, \ldots, r. \quad (12)$$

The r-tuple $(x_1, x_2, \ldots, x_r)$ is called the residue code of X.

The relationship between X and its residue codes is bridged by the Chinese Remainder Theorem [25].

THEOREM 6 (Chinese Remainder Theorem). *Suppose the positive integers $m_1, m_2, \ldots, m_r$ are relatively prime in pairs, that is, $gcd(m_i, m_j) = 1$ for all i, j where $i \neq j$. Then the set of congruences*

$$X \equiv x_i (\bmod\ m_i)$$

for $i = 1, 2, \ldots, r$ has a unique common solution modulo M, where $M = m_1 m_2 \ldots m_r$. The solution is given by

$$X = \sum_{i=1}^{r} x_i m_i' m_i'' \bmod M, \tag{13}$$

where $m_i' = \frac{M}{m_i}$, and m_i'' are integers satisfying

$$m_i' m_i'' \equiv 1 \bmod m_i.$$

In Section 4, we shall show how to realize the isomorphism and its inverse isomorphism between Z_M and $Z_{m_1} \times Z_{m_2} \times \ldots \times Z_{m_r}$ in terms of permutation networks.

3 ARITHMETIC UNIT DESIGN

This section introduces various optoelectronic arithmetic circuits. All of these optoelectronic arithmetic circuits consist of three basic components: cyclic permutation networks, directional coupler switches, and Y-junctions.

3.1 Basic Components

3.1.1 Cyclic Permutation Networks

In this section, we will show how to realize a cyclical permutation network which will be used to realize cyclic groups. A cyclical (right shift) permutation network with degree m, denoted by $CRPN(m)$ has m inputs and m outputs and is a cascade of $\lceil \log m \rceil$ switches, numbered form left to right $\lceil \log m \rceil - 1, \ldots, 2, 1, 0$ in that order, and each having m inputs and m outputs such that the switch in stage k, for all $k = 0, 1, \ldots, \lceil \log m \rceil - 1$ has two switching states:

state 0: input i is passed directly to output i, for all $i = 0, 1, \ldots, m - 1$;

state 1: input i is connected to output j such that

$$j = i +_m 2^k \tag{14}$$

for all $i = 0, 1, \ldots, m - 1$.

It is easy to verify that a right circular shift of the sequence of elements $012 \ldots m - 1$ r times, $0 \leq r \leq m - 1$, can be realized on this network by expressing r in binary as $a_0 2^0 + a_1 2^1 + \ldots + a_{\lceil \log m \rceil - 1} 2^{\lceil \log m \rceil - 1}$ and setting all those switches for which $a_i = 0$ to state 0 and those for which $a_i = 1$ to state 1.

3.1.2 Directional Coupler Switch

The directional coupler switch can be used like a 2-by-2 switch with two states: the straight state and cross state [13], [24]. In the straight state, the signal of the upper input goes to the upper output while the signal of lower input goes to the lower output. In the cross state, the signal of the upper input goes to the lower output while the signal of lower input goes to the upper output, as shown in Fig. 3a. The logic model for the directional coupler switch is shown in Fig. 3b. By properly controlling their states, directional coupler switches can be used to implement permutations on permutation networks.

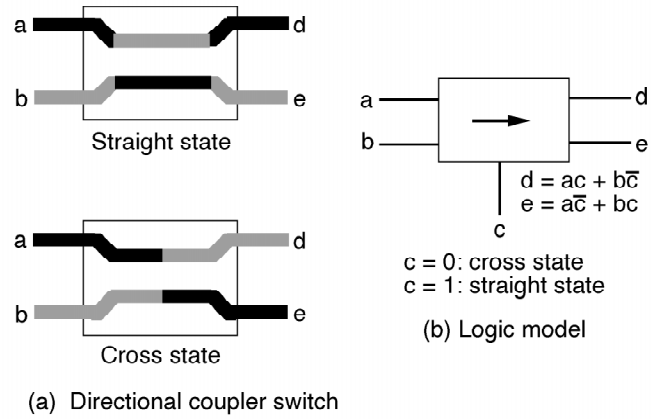


Fig. 3. Directional coupler switch and its logic model.

Two factors affect the fabric size of integration when using directional coupler switches. These are the attenuation of signals passing through the device and the cross talk inside the device [11]. However, these two factors are minimal when the underlying network topology has a logarithmic depth and if we can ensure that no two inputs on a directional coupler switch are active at the same time [11].

Here, only a subset of functions of a directional coupler switch is required to design arithmetic circuits. More precisely, we will set $b = 0$ for all times while using the directional coupler switches. Since all cyclic permutation networks used to design arithmetic circuits have a logarithmic depth and one of the inputs of each directional coupler switch is set to 0, that is, no two inputs of the same directional coupler switch in the cyclic permutation networks are active simultaneously, the signal attenuation and cross talk are minimal.

The other limits on the fabric size of integration are the large length of directional coupler switches in relation to their width and the large minimum bending radius of the diffused waveguides. All these constraints add up to a maximum integration array size of 32 by 32 [11]. However, from Table 1 we see that the product range of the first 10 primes is already about 2^{32} and the largest prime is only 29. Therefore, the arithmetic circuits proposed in this paper are feasible within the domain of current integrated optics.

Another component needed to realize a cyclic permutation network in optics is a Y-junction, which is a special kind of optical coupler, and sometimes called a combiner [1]. This device joins two signals into one that can then propagate to the next stage. Since the technology of integrated optics is still in its developing stages, the signal losses of directional coupler switches and Y-junctions are high but they are likely to be reduced to some extent in the near future. For practical purposes, if it is necessary, optical regenerators may be used to recover the signals [1].

3.1.3 Input Encoder and Output Decoder

Occasionally the operands must be converted to residue representation and the results in residue representation

must be converted to binary. We use encoder and decoder circuits to convert between binary and 1-out-of-m position codes. More specifically, the 1-out-of-m encoder in the input stage codes its $\lceil \log m \rceil$-bit binary input into a 1-out-of-m position code, and the 1-out-of-m decoder in the output stage decodes its 1-out-of-m position code into its $\lceil \log m \rceil$-bit binary equivalent. Positions are numbered 0, 1, to $m - 1$, and position i is identified with binary input i.

Since the encoder and decoder are electronic circuits, it is necessary to do the conversions between optical and electronic signals in these circuits and vice versa. We will see that optical-to-electronic and electronic-to-optical circuits are only needed in the encoder and decoder stages.

3.2 Modulo *m* Addition

As implied by Cayley theorem, the set $Z_m = \{0, 1, \ldots, m - 1\}$ under addition modulo m, is isomorphic to a cyclic permutation group of order m. Thus, it is sufficient to consider the implementation of a cyclic permutation group on the $CRPN(m)$. Here, the implementation of a cyclic group refers to composing any two elements of the group to obtain another element in the group. This is to be done by entering the elements into the network, and the network will produce as output the composition of the elements. We first note that a cyclic group $(G, \bullet)$ of order m is generated by a generator g and its elements are g^0 $(= g^m)$, $g^1, \ldots, g^{m-1}$, where $g^0 = e$ is the identity permutation, and g^i, for all i, $0 \le i \le m - 1$, is the permutation obtained by cyclically shifting right the sequence 0 1 2 … $m - 1$ i times. It is easy to verify that

$$g^x \bullet g^y = g^{x +_m y} \tag{15}$$

for all $x, y = 0, 1, \ldots, m - 1$. That is, the composition of two cycles g^x and g^y results in a cycle obtained by cyclically right shifting the sequence 0 1 2 … $m - 1$ $x + y$ times. Thus, an implementation of a cyclic group of order m can be obtained by cascading two $CRPN(m)$s. The composition of two elements g^x and g^y in the group can then be realized by cyclically shifting right the inputs of the first network x times and then cyclically shifting right the inputs of the second network y times, as shown in Fig. 1. The resulting permutation is then read from the outputs of the second network. While providing an implementation of a cyclic group, this approach requires two $CRPN(m)$s. An implementation requiring only one $CRPN(m)$ can be obtained by dropping the second network and connecting the outputs of the first network back onto its inputs. The composition of $g^x \bullet g^y$ is then computed by first cyclically right shifting the inputs x times and then copying the outputs back onto the inputs, and finally cyclically right shifting the inputs y times. This design reduces the hardware cost, but like the first design it requires a total of $x + y$ cyclical right shifts. The number of shifts can be reduced to y by noting that one of the elements, say g^x, can be entered into the network directly by coding it in terms of binary m-tuples in which exactly one entry is 1. The position of that entry is specified by x.

The central component of this implementation is the $CRPN(m)$, which is constructed as described above. Given two elements g^x and g^y, the network receives exponent x to activate its xth input. The network then cyclically shifts the "1" down y positions and outputs it at output $x +_m y$. For $(Z_n, +_n)$, the operation is addition so we use multiples instead of powers. We set its generator to "1" so that the operation $g^x \bullet g^y$ can be performed simply by using $x +_m y$.

Fig. 4 shows an example of modulo 5 addition. To illustrate how the adder operates, suppose that the operation $1 +_5 3$ is required. Then input x (in this case it is 1), activates the line 1. The other input y (in this case it is 3), is applied to the control inputs of the directional coupler switches through inverters and determines the switching state of the network. The final result, which is shown in darker line is $1 +_5 3 = 4$. We note that all inputs of stage i is shifted right (down) by $2^i \bmod 5$, $0 \le i \le 2$.

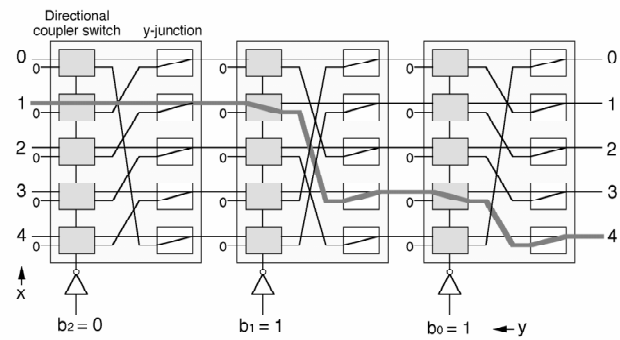


Fig. 4. Modulo 5 addition on $CRPN(5)$ ($x +_5 y = 1 +_5 3 = 4$).

3.3 Modulo *m* Subtraction

As in complement arithmetic, subtraction $x - y$ is carried out on the $CRPN(m)$ by adding the additive inverse of the subtrahend (y) to the minuend (x). In terms of the elements of a cyclic group, if g^z is the inverse of g^y then $g^y \bullet g^z = g^0 = g^m$, that is, $y + z = m$ or $z = m - y$. Thus, the inverse of y can be computed by right shifting the inputs of the $CRPN(m)$ $m - y$ times. Using this fact, subtraction $x - y$ can be performed by first inputting x into the $CRPN(m)$ and then shifting it right $m - y$ times. While this facilitates subtraction on the $CRPN(m)$, it requires an explicit computation of $m - y$ outside the $CRPN(m)$. This problem can be avoided in at least two ways. As a first solution, we note that right circular shifting $m - y$ times is the same as left circular shifting y times. Therefore, we can perform $x - y$ simply by inputting x into a cyclical left permutation network with degree m, denoted by $CLPN(m)$, and then shifting left (up) the inputs of the network y times as depicted in Fig. 5. Like the $CRPN(m)$, each stage in the $CLPN(m)$ has two states: State 0 is the same as state 0 in the $CRPN(m)$ and in its state 1, all inputs in the kth stage are shifted left circularly by 2^k. Fig. 5 is an example of computing $x -_5 y = 1 -_5 3 = 3$. The generator is assumed to be "1."

The $CLPN(m)$ eliminates the need for an explicit computation of $m - y$ but if we want to combine addition and subtraction on a single network then we must have three switching states for each stage. Instead, we can perform both addition and subtraction on the $CRPN(m)$ if we enter y to the $CRPN(m)$ from the left side of the $CRPN(m)$ and use x to determine the switching state of the network. We need one other minor modification to the network. Because we want to compute $x + (m - y)$, we must enter y to input $m - y$

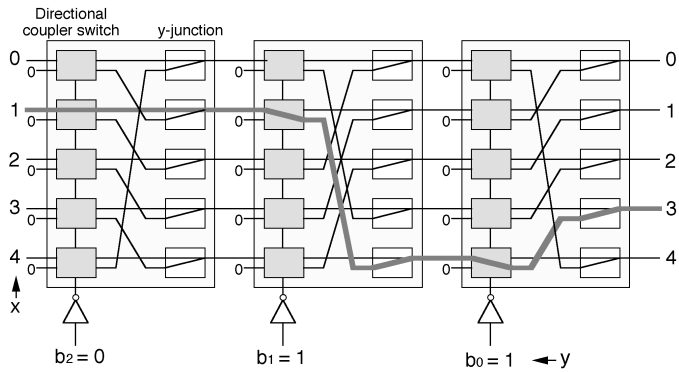


Fig. 5. Modulo 5 subtraction on *CLPN*(5) ($x -_5 y = 1 -_5 3 = 3$).

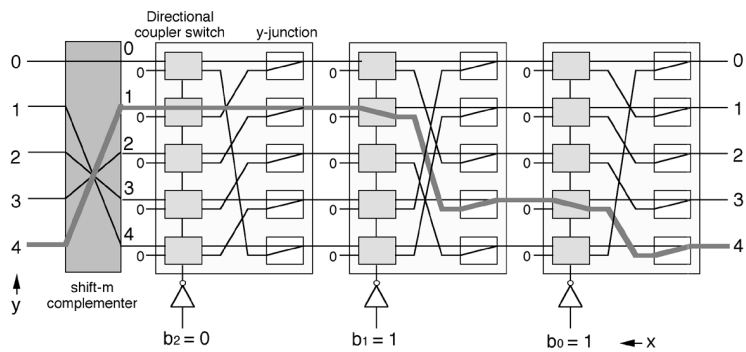


Fig. 6. Modulo 5 subtraction on *CRPN*(5) ($x -_5 y = 3 -_5 4 = 4$).

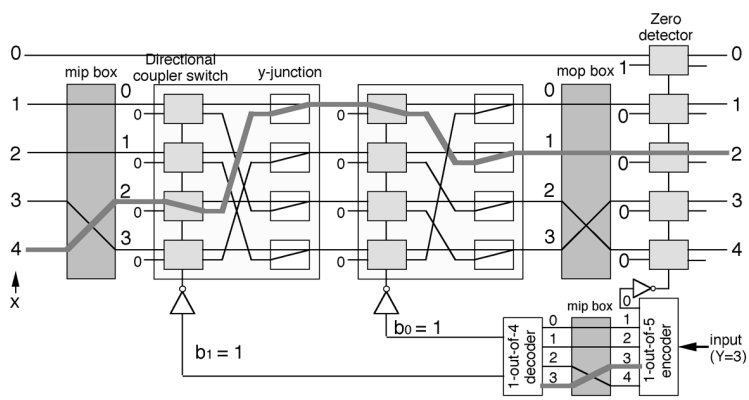


Fig. 7. Modulo 5 multiplication on *CRPN*(5) ($x \times_5 y = 4 \times_5 3 = 2$).

rather than input y of the *CRPN*(m). This is accomplished by a permutation network that maps y, $0 \le y \le m - 1$, to input $m - y$ of the *CRPN*(m). Essentially, this operation amounts to determining the inverse g^{m-y} of the permutation g^y, where g is the generator of the cyclic group. We will call g^{m-y} the shift-m complement of g^y. Fig. 6 depicts the organization of this network. The shaded box, called the *shift-m complementer* computes the shift-m complement of g^y. We must note that for a fixed m, the shift-m complementer has a fixed pattern of connections that do not change with the value of y. We must also note that addition and subtraction can be combined together by adding the 0 state to the shift-m complementer, i.e., by connecting its ith input to its ith output, $0 \le i \le m - 1$.

3.4 Modulo m Multiplication

The set $\{1, 2, \ldots, m - 1\}$ under multiplication modulo m forms a cyclic group of order $m - 1$, when m is a prime. A generator of this group is an element h such that $h^{m-1} \equiv 1 \pmod{m}$. In general, there is no straightforward method to determine the generators of a multiplicative group. However, for small primes, one can always find the generators by trial and error.

Let Z_m^* denote the multiplicative group mod m with a generator h. To realize Z_m^* on the *CRPN*(m),[1] we set up an

1. In reality, the network in this case has $m - 1$ inputs, but for notational convenience, we will refer to it as an *CRPN*(m).

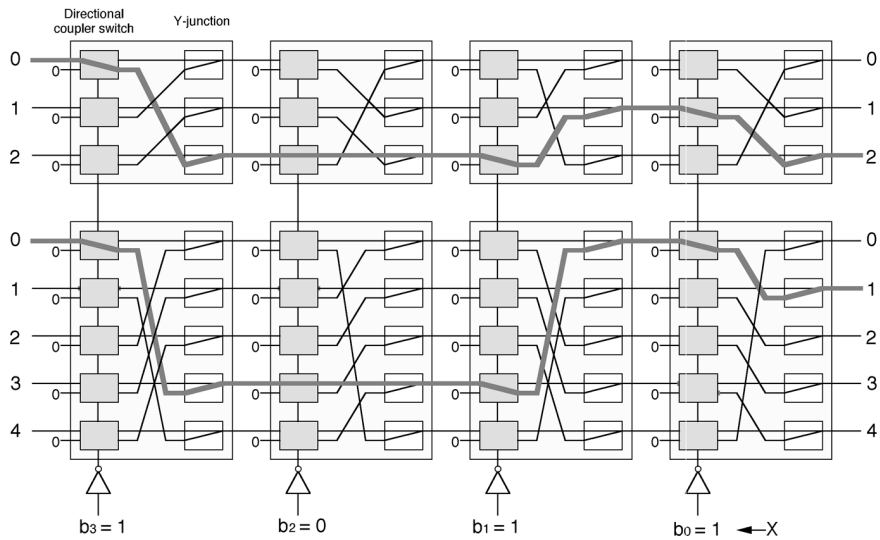


Fig. 8. Converting an unsigned number (11_{10}) into its residue codes (2, 1) on *CRPN*(15).

isomorphism f between Z_m^* and the cyclic permutation group G_{m-1} defined by $f(h^i) = g^i$; $i = 0, 1, \ldots, m-2$. Given this, what remains to be done is to express the elements of Z_m^* as a power of the generator h. For a given multiplicative group and a generator, these powers are fixed and can be predetermined. Once this is done, the realization of Z_m^* on the *CRPN*(m) proceeds much the same way as in the realization of addition mod ($m - 1$) as shown in Fig. 7. There are a few minor differences: First, if one or both operands x and y are zero, then the outputs of the *CRPN*(m) are bypassed by using a zero detection circuit. The extra stage of directional coupler switches as shown in the figure are added to handle this case. Second, the shaded boxes, called the *mip* (multiplication input permutation) and *mop* (multiplication output permutation) boxes are inserted before and after the *CRPN*(m) to convert the x operand into a power of the generator used, and reconvert the power of the generator that is obtained at the outputs of the *CRPN*(m) into the product of x and y. Another mip box along with a 1-out-of-($m - 1$) decoder and 1-out-of-m encoder are used to convert y into a power of the generator. In all these cases, the connections of the mip and mop boxes are fixed for a given m and a generator. Hence, they do not incur any additional cost. Fig. 7 illustrates the multiplication process on the *CRPN*(m) for $x = 4$, $y = 3$ and when the generator is 2.

4 Input Encoding and Output Decoding

Now that we have completed our discussion of how to implement various algebraic operations on a *CRPN*(m), we will show how to convert between residue codes and binary number systems on such a network. The problem of converting a binary number into its residue codes is called the *input translation* or *input encoding* problem. Likewise, the problem of converting a residue codes into its binary form is called the *output translation* or *output decoding* problem [15].

4.1 Input Encoding

Let $X = (b_{k-1} \ldots b_0 b_1)$ be a k-bit binary number, where $k = \lceil \log m_1 m_2 \ldots m_r \rceil$, that is, the range of X is $M = m_1 \times m_2 \times \ldots \times m_r$, and let $(x_1, x_2, \ldots, x_r)$ be the residue codes of X. The input encoding is implemented by r *CRPN*(m_i)s, where the ith network has m_i inputs and all networks have $\lceil \log m_1 m_2 \ldots m_r \rceil$ stages. The entries x_i, $1 \le i \le r$, are determined by setting the jth stage of all the networks according to the jth bit b_j in the binary representation of X. If $b_j = 0$, then the jth stages of all networks are set to state 0, and if $b_j = 1$, then they are set to state 1. The value of x_i is then obtained at the output of the last stage of the ith network. If the residue codes are desired in a binary form, a 1-out-of-m_i decoder can be connected to the output of the last stage of the ith network. The decoder identifies that output of the network which is connected to its 0 input after its stages are set. Fig. 8 illustrates this method for $m_1 = 3$, $m_2 = 5$, and $X = (01011)_2 = 11_{10} = (2, 1)$.

The rationale behind this method is Horner's evaluation of x mod m_i, i.e.,

$$x_i = X \bmod m_i = \sum_{j=0}^{k-1} b_j 2^j \left(\bmod m_i\right) \tag{16}$$

$$= \left(\ldots\left(\left(b_{k-1}2^{k-1} \bmod m_i\right) + b_{k-2}2^{k-2}\right) \bmod m_i + \ldots + b_0 2^0\right) \bmod m_i, \quad \text{for all } i,\ 1 \le i \le r. \tag{17}$$

The first stage in the ith network computes $b_{k-1}2^{k-1}$ mod m_i, the second stage computes $((b_{k-1}2^{k-1} \bmod m_i) + b_{k-2}2^{k-2}) \bmod m_i$, and inductively the last stage computes the entire expression.

4.2 Output Decoding

The output decoding circuit is based on the idea of base extension and scaling by 2^t [26]. First, we carry out the residue decoder using an extension of Garner's algorithm [9]. Let $(x_1, x_2, \ldots, x_r)$ be the residue codes of X with moduli $m_1, m_2, \ldots, m_r$, respectively. Garner's algorithm receives $x_1, x_2, \ldots, x_r$ as input and produces a single output. The algorithm given below is a modified version of Garner's algorithm and computes t bits of the binary output at a time, where t is some positive number between 1 and log M, and $M = m_1 m_2 \ldots m_r$.

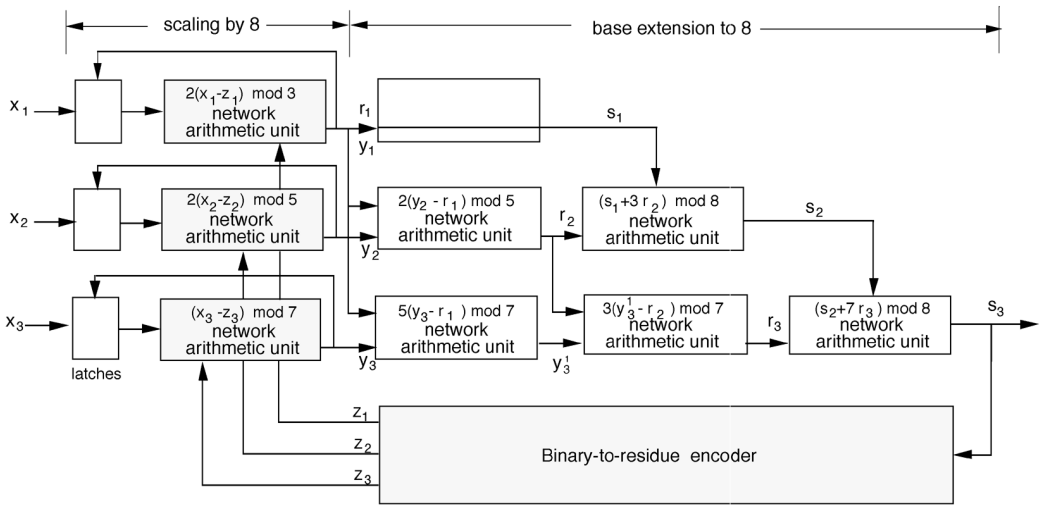


Fig. 9. Output decoding on *CRPN*(105); $r = 3$, $n = 3$, $m_1 = 3$, $m_2 = 5$, $m_3 = 7$.

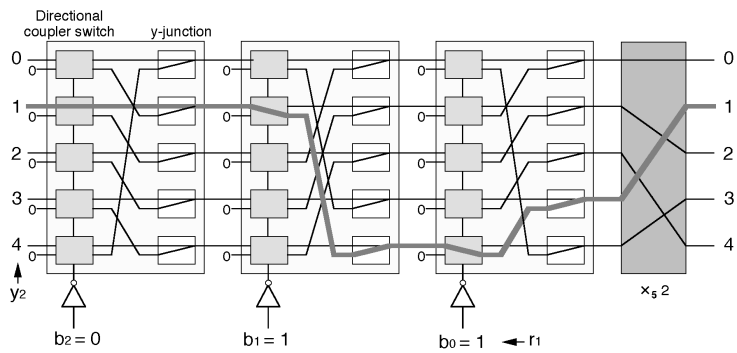


Fig 10. Network arithmetic unit to compute $2(y_2 -_5 r_1) = 2(1 -_5 3) = 1$ on *CRPN*(5).

Algorithm (Conversion residue codes into its binary form.)

Begin

Step 1: Determine constants $c_{i,j}$ satisfying $m_i c_{i,j} = 1 \bmod m_j$, for $1 \le i < j \le r$, and constants $c_{2^t,i}$ satisfying $2^t c_{2^t,i} = 1 \bmod m_i$ for $1 \le i \le r$, where t is a positive integer $\ge \max_{1 \le i \le r}\{\lceil \log m_i \rceil\}$. This step is carried out off line and is not part of the residue decoder.

Step 2: Let $y_i = x_i$, for all i, $1 \le i \le r$.

Step 3: Base extension to mod 2^t.

3.1 Compute

$$r_1 = y_1 \bmod m_1$$
$$r_2 = (y_2 - r_1)c_{12} \bmod m_2$$
$$r_3 = ((y_3 - r_1)c_{13} - r_2)c_{23} \bmod m_3$$
$$\vdots$$
$$r_r = (\ldots((y_r - r_1)c_{1r} - r_2)c_{2r} - \ldots - r_{r-1})c_{r-1,r} \bmod m_r \qquad (18)$$

3.2 Compute

$$S_1 = r_1 \bmod 2^t$$
$$S_2 = S_1 + r_2 m_1 \bmod 2^t$$
$$\vdots$$
$$S_r = S_{r-1} + r_r m_1 m_2 \ldots m_{r-1} \bmod 2^t \qquad (19)$$

Step 4: Scaling by 2^t.

Compute $y_i = (x_i - z_i)c_{2^t,i}$ and set $x_i = y_i$, for all i, $1 \le i \le r$, where $z_i = S_r \bmod m_i$.

Step 5: Repeat Steps 3 and 4 for $\left\lceil \frac{\log M}{t} \right\rceil - 1$ times.

End

The correctness of the algorithm can be found in [9], [26].

Fig. 9 depicts a network implementation of the residue decoder for $r = 3$, $m_1 = 3$, $m_2 = 5$, $m_3 = 7$, and $t = 3$. The number X (take (0, 2, 3) = 87 as an example) enters the circuit on the left in residue codes, and exits it in binary on the right in three iterations; the first iteration computes the least significant three bits (= 111), the second iteration computes the next three significant bits (= 010), and the last iteration computes the most significant bits (= 001).

Basically, a network arithmetic unit is a *CRPN*(m) modulo m subtracter cascaded with a constant modulo m

multiplier. Since one input of the modulo m multiplier is a constant, it can be implemented by a fixed set of connections as in the shift-m complementer case (Fig. 6). A network arithmetic unit that is set to compute $2(y_2 -_5 r_1)$ is shown in Fig. 10.

5 Performance Analysis

Now we consider the hardware cost and delay for the arithmetic circuits designed above. Let $C_{ADD}(m)$ and $D_{ADD}(m)$ denote the cost and the delay of mod m addition or subtraction network, respectively. Similarly, let $C_{MUL}(m)$ and $D_{MUL}(m)$ denote the cost and the delay of mod m multiplication network, respectively. The $CRPN(m)$ can be implemented with $m\lceil \log m \rceil$ directional coupler switches and Y-junctions, and with $\lceil \log m \rceil$ delay. For simplicity and without loss of generality, we assume that the cost and delay of both directional coupler switch and Y-junction are the same. The shift-m complementer box used in the subtraction circuit is hardwired, and we assume that it does not incur any additional cost or delay. With these observations, we conclude that the modulo m addition and subtraction on the $CRPN(m)$ requires

$$C_{ADD}(m) = m\lceil \log m \rceil + m\lceil \log m \rceil \tag{20}$$

cost and

$$D_{ADD}(m) = 2\lceil \log m \rceil \tag{21}$$

delay.

Now we consider the case of $M = m_1 \times m_2 \times \cdots \times m_r$. Let $C_{ADD}(M)$ and $D_{ADD}(M)$ denote the total number of optical directional coupler switches and the delay for the modulo M adder, respectively, excluding the residue encoder and decoder circuits. It is easy to deduce from the above analysis that $C_{ADD}(M)$ is the sum of all subnetworks $CRPN(m_i)$. Hence, we have

$$C_{ADD}(M) = 2\sum_{i=1}^{r} m_i\lceil \log m_i \rceil.$$

It is shown in [16] that if m_i, $1 \le i \le r$ are the first r primes, then $\sum_{i=1}^{r} m_i\lceil \log m_i \rceil = O\left(\log^2 M\right) = O\left(n^2\right)$, where $n = \log M$. Thus,

$$C_{ADD}(M) = O(\log^2 M) = O(n^2).$$

The delay through the network is given by the delay through the largest subnetwork, i.e., subnetwork $CRPN(m_r)$. Hence,

$$D_{ADD}(M) = \max_{i=1}^{r}\lceil 2\log m_i \rceil.$$

From [16], we know that if m_i, $1 \le i \le r$ are the first r primes then $\max_{i=1}^{r} m_i = O(\log M)$, so that we have

$$D_{ADD}(M) = O(\log\log M) = O(\log n).$$

These expressions also hold for the modulo M subtracter since the shift complement circuits in the subtracter are hardwired and do not incur any additional cost or delay.

Let $C_{MPY}(M)$ and $D_{MPY}(M)$ denote the hardware cost and the delay of $CRPN(M)$ modulo M multiplier, respectively, again not including the encoder and decoder circuits. In addition to r $CRPN(m_i)$s, the $CRPN(M)$ multiplier also uses a 1-out-of-m_i encoder and a 1-out-of-$(m_i - 1)$ decoder in its ith subnetwork $CRPN(m_i)$. Each 1-out-of-$(m_i - 1)$ decoder can be implemented with $(m_i - 1)\lceil \log (m_i - 1) \rceil$ two-input OR gates and $\lceil \log (m_i - 1) \rceil$ delay; each 1-out-of-m_i encoder can be implemented with $m_i\lceil \log m_i \rceil$ two-input AND gates and $\lceil \log m_i \rceil$ delay. For convenience and without loss of generality, we assume that a two-input logic gate have the same delay and cost as that of optical directional coupler switch or Y-junction. Given this assumption and the fact that all *mip* and *mop* boxes used in the subnetworks are hardwired, we have

$$\begin{aligned} C_{MPY}(M) &= \sum_{i=1}^{r}\left(5(m_i - 1)\lceil \log(m_i - 1)\rceil + m_i + m_i\lceil \log m_i \rceil\right) \\ &\le \sum_{i=1}^{r} 7m_i\lceil \log m_i \rceil. \end{aligned}$$

Again, noting that $\sum_{i=1}^{r} m_i\lceil \log m_i \rceil = O\left(\log^2 M\right) = O\left(n^2\right)$, we have

$$C_{MPY}(M) = O(\log^2 M) = O(n^2).$$

The delay through the multiplier is

$$D_{MPY}(M) = \max_{i=1}^{r}\left(\lceil \log m_i \rceil + \lceil 5\log(m_i - 1)\rceil + 1\right)$$

or, given that $\max_{i=1}^{r} m_i = O(\log M)$,

$$D_{MPY}(M) = O(\log\log M) = O(\log n).$$

Now, let $C_{INPUT}(M)$ and $D_{INPUT}(M)$ denote the cost and the delay of input encoding network, respectively. We have

$$C_{INPUT}(M) = 2n\sum_{i=1}^{r} m_i. \tag{22}$$

Recalling that $n = \lceil \log M \rceil$, and that $m_1 + m_2 + \ldots + m_r = O(\log^2 M/\log\log M)$ from [16], the cost of this network is $O(\log^3 M/\log\log M) = O(n^3/\log n)$. Its delay is

$$D_{INPUT}(M) = 2n. \tag{23}$$

Remark 1. We note that this input encoder uses 1-out-of-m_i codes. This allows it to detect all unidirectional errors concurrently [17]. If we relax this property, then a binary tree algorithm can be used to reduce the cost and delay complexities of the input encoder to $O(n^2)$ and $O(\log n)$, respectively [16].

Finally, let $C_{OUTPUT}(M)$ and $D_{OUTPUT}(M)$ denote the cost and the delay of the output decoding network, respectively. Then

$$C_{OUTPUT}(M) = \sum_{i=1}^{r} iC_{ADD}(m_i) + (r-1)C_{ADD}\left(2^t\right) + \left[2t\sum_{i=1}^{r} m_i + \sum_{i=1}^{r} m_i\lceil \log m_i \rceil\right] \tag{24}$$

where the first summand is contributed by the networks that compute r_i and y_i, $1 \le i \le r$, the second term is the cost of networks that compute S_i, $1 \le i \le r$, and the last summand is the cost of the binary-to-residue conversion network that

TABLE 2
COMPLEXITY COMPARISON OF VARIOUS DESIGN FOR AN n-BIT PARALLEL ADDER

Method	Number of Gates, Light Sources, or Detection Planes	Circuit Depth or Time Complexity
SSL [5], [19]	$\Omega(2^n)$	$O(1)$
Shadow casting [19], [28]	$O(2^n)$	$O(1)$
Programmable logic [19], [21]	$O(n2^n)$	$O(n)$
Spatially variant method [19]	$O(n^3)$	$O(\log n)$
Proposed method (Residue codes domain)	$O(n^2)$	$O(\log n)$
Proposed method (Binary number domain)	$O(n^3/\log n)$	$O(n^2/\log n)$

compute z_i, $1 \le i \le r$. Substituting (20) for $C_{ADD}(m_i)$ and $C_{ADD}(2^t)$,

$$C_{OUTPUT}(M) = \sum_{i=1}^{r} i\left(2m_i\lceil \log m_i \rceil\right) + \left[(r-1)\left(2 \times t \times 2^t\right)\right] + \left(2t\sum_{i=1}^{r} m_i + \sum_{i=1}^{r} m_i\lceil \log m_i \rceil\right) \quad (25)$$

Given that $r = O(n/\log n)$ and $m_r = O(n)$, the first summand on the right is $O(n^3/\log n)$. Likewise, the middle term is $O(t2^t n/\log n)$. Also, given that

$$\sum_{i=1}^{r} m_i = O\left(\log^2 M / \log\log M\right) = O\left(n^2 / \log n\right)$$

and

$$\sum_{i=1}^{r} m_i \log m_i = O\left(\log^2 M\right) = O\left(n^2\right),$$

the last summand is $O(tn^2/\log n + n^2)$ as established in [16].

Summing these three terms together we find

$$C_{OUTPUT}(M) = O(n^3/\log n + tn2^t/\log n + tn^2/\log n + n^2). \quad (26)$$

Now as for the delay through the network, we note that inputs are circulated through the network $\left\lceil \frac{\log M}{t} \right\rceil - 1$ times, and that the delay experienced in each iteration is $2t + \max_{1\le i\le r}\{\lceil \log m_i \rceil\} + (r+1)\max_{1\le i\le r}\{\lceil \log m_i \rceil\}$. Thus, the delay through this conversion network is

$$D_{OUTPUT} = r\max_{1\le i\le r}\left\{\lceil \log m_i \rceil\right\} + \left(\left\lceil \frac{\log M}{t} \right\rceil - 1\right)\left(2t + \max_{1\le i\le r}\left\{\lceil \log m_i \rceil\right\} + (r+1)\max_{1\le i\le r}\left\{\lceil \log m_i \rceil\right\}\right) \quad (27)$$

The first term is $O(\log M) = O(n)$, while the second term is $O(\frac{\log M}{t}(t + \log M))$, and hence

$$D_{OUTPUT}(M) = O\left(n + \frac{n}{t}(t+n)\right). \quad (28).$$

Now, if we let $t = \log\log M = \log n$, then we have

$$C_{OUTPUT}(M) = O(n^3/\log n) \quad (29)$$

$$D_{OUTPUT}(M) = O(n^2/\log n). \quad (30)$$

REMARK 2. These cost and delay complexities can be reduced to $O(n^2\log n)$ and $O(\log^2 n)$ if we use an algorithm based on Chinese Remainder Theorem [16]. However, an outstanding feature of the output decoder presented here is that, like the input encoder, it also has the capability of detecting all unidirectional errors concurrently [17]. Therefore, all modules described in this paper can be combined into a system that has the capability of detecting all unidirectional errors concurrently.

In summary, modulo addition, subtraction, and multiplication exact $O(n^2)$ hardware and $O(\log n)$ delay in the residue code domain, $O(n^3/\log n)$ hardware and $O(n^2/\log n)$ delay in the binary domain. The increase of complexity in the binary domain is due to the required conversions between binary and residue codes. Table 2 compares these complexities with the complexities of previously published designs.

6 CONCLUSION

In this paper, cyclic permutation networks are defined and used to construct various arithmetic circuits. Unlike conventional arithmetic units, these arithmetic circuits are based on coding numbers into permutation maps, and then carrying out arithmetic operations by composing permutations. The resulting permutations are then converted back into sums and products. It has been established that in order of complexity terms, optical implementations of these arithmetic circuits are more efficient and faster than the previously reported arithmetic architectures. Another added advantage of these new arithmetic circuits is that they are inherently capable of concurrent error detection [17]. Moreover, they can be used to construct inner product processors with $O(1)$ computation time [18].

One potential drawback of these new arithmetic circuits is their time overhead for converting between binary and residue codes. This may not be as critical in signal processing applications as it is in general purpose computations. Nonetheless, in the case of general purpose computations, much of the potential performance degradation due to conversion time overhead can be alleviated by pipelining the conversion steps over input encoding and output decoding circuits.

ACKNOWLEDGMENTS

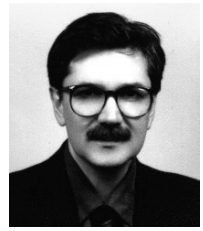
The authors would like to thank the editor and anonymous reviewers for their helpful comments. This work was supported by the National Science Council, Taiwan, Republic of China, under contract NSC 84-2215-E011-002.

REFERENCES

[1] A.F. Benner et al., "Digital Optical Counter Using Directional Coupler Switches," *Applied Optics,* vol. 30, no. 29, pp. 4,179-4,189, Oct. 1991.
[2] A.F. Benner, H.F. Jordan, and V.P. Heuring, "Digital Optical Computing with Optically Switched Directional Couplers," *Optical Eng.,* vol. 30, pp. 1,936-1,941, Dec. 1991.
[3] S. Barua, "High-Speed Multiplier for Digital Signal Processing," *Optical Eng.,* vol. 30, pp. 1,997-2,002, Dec. 1991.
[4] V.E. Benes, "On Rearrangeable Three-Stage Connecting Networks," *The Bell System Technical J.,* vol. 41, pp. 14,81-1,492, May 1962.
[5] K.H. Brenner, A. Huang, and N. Streibl, "Digital Optical Computing with Symbolic Substitution, " *Applied Optics,* vol. 25, pp. 3,054-3,060, 1986.
[6] K.H. Brenner, "New Implementation of Symbolic Substitution Logic," *Applied Optics,* vol. 25, pp. 3,061-3,064, 1986.
[7] K.H. Brenner, A.W. Lohmann, and T.M. Merklein, "Symbolic Substitution Implemented by Spatial Filtering Logic," *Optical Eng.,* vol. 28, pp. 390-396, 1989.
[8] J.A. Gallian, *Contemporary Abstract Algebra,* second edition. D.C. Heath and Company, 1990.
[9] H.L. Garner, "The Residue Number System," *IRE Trans. Electronic Computers,* vol. 8, pp. 140-147, June 1959.
[10] P.S. Guilfoyle and W.J. Wiley, "Combinatorial Logic Based Digital Optical Computing Architectures," *Applied Optics,* vol. 27, pp. 1,661-1,673, 1988.
[11] H.S. Hinton, "Switching to Photonics," *IEEE Spectrum,* vol. 29, no. 2, pp. 42-45, Feb. 1992.
[12] H. Jeon, M.A.G. Abushagur, A.A. Sawchuk, and B.K. Jenkins, "Digital Optical Processor Based on Symbolic Substitution Using Holographic Matched Filtering," *Applied Optics,* vol. 29, pp. 2,113-2,125, 1990.
[13] M. Kondo et al., "Integrated Optical Switch Matrix for Single-Mode Fiber Networks," *IEEE J. Quantum Electronics,* vol. 18, pp. 1,759-1,765, Oct. 1982.
[14] C.-T. Lea, "Crossover Minimization in Directional-Coupler-Based Photonic Switching Systems," *IEEE Trans. Comm.,* vol. 36, pp. 355-363, Mar. 1988.
[15] M.-B. Lin and A.Y. Oruç, "The Design of a Network-Based Arithmetic Processor," UMIACS-TR-91-141, CS-TR-2780, College Park, Md., Oct. 1991.
[16] M.-B. Lin, "Unified Algebraic Computations on Permutation Networks," PhD dissertation, EE Dept., Univ. of Maryland, College Park, 1992.
[17] M.-B. Lin and A.Y. Oruç, "A Fault-Tolerant Permutation Network Modulo Arithmetic Processor," *IEEE Trans. VLSI Systems,* vol. 2, pp. 312-319, Sept. 1994.
[18] M.-B. Lin and A.Y. Oruç, "Constant Time Inner Product and Matrix Computations on Permutation Network Processors," *IEEE Trans. Computers,* vol. 43, no. 12, pp. 1,429-1,434, Dec. 1994.
[19] A. Louri and A. Post, "Complexity Analysis of Optical-Computing Paradigms," *Applied Optics,* vol. 31, no. 26, pp. 5,568-5,583, Sept. 1992.
[20] A.D. McAulay, *Optical Computer Architectures: The Application of Optical Concepts to Next Generation Computers.* John Wiley and Sons, 1991.
[21] M.J. Murdocca, A. Huang, J. Jahns, and N. Streibl, "Optical Design of Programmable Logic Arrays," *Applied Optics,* vol. 27, pp. 1,651-1,660, 1988.
[22] M.J. Murdocca and T.J. Cloonan, "Optical Design of a Digital Switch," *Applied Optics,* vol. 28, pp. 2,505-2,517, 1989.
[23] W.K. Nicholson, *Introduction to Abstract Algebra.* PWS-KENT Publishing Company, 1993.
[24] J.P. Pratt and V.P. Heuring, "Designing Digital Optical Computing Systems: Power Distribution and Cross Talk," *Applied Optics,* vol. 31, pp. 4,657-4,661, Aug. 1992.
[25] H.E. Rose, *A Course in Number Theory*. New York: Oxford Univ. Press, 1988.
[26] A.S. Shenoy and R. Kumaresan, "Residue to Binary Conversion for RNS Arithmetic Using Only Modular Look-Up Tables," *IEEE Trans. Circuits and Systems,* vol. 35, no. 9, pp. 1,158-1,162, Sept. 1988.
[27] T. Stouraitis, S.W. Kim, and A. Skavantzos, "Full Adder-Based Arithmetic Units for Finite Integer Rings," *IEEE Trans. Circuits and Systems: II,* vol. 40, pp. 741-745, Nov. 1993.
[28] J. Tanida and Y. Ichioka, "Modular Components for an Optical Array Logic System," *Applied Optics,* vol. 26, pp. 3,954-3,960, 1987.
[29] A. Waksman, "A Permutation Network," *J. ACM,* vol. 15, pp. 159-163, 1968.

Ming-Bo Lin (S'90-M'93) received the BSc degree in electronic engineering from the National Taiwan Institute of Technology, Taipei, in 1981, the MSc degree in electrical engineering from the National Taiwan University, Taipei, in 1986, and the PhD degree in electrical engineering from the University of Maryland, College Park, in 1992.

Since August 1992, he has been an associate professor with the Department of Electronic Engineering at the National Taiwan Institute of Technology, Taipei. His research interests include VLSI system design, parallel algorithms, computer arithmetic, and fault-tolerant computing. He is a member of the IEEE.



A. Yavuz Oruç received his PhD degree in electrical engineering from Syracuse University, Syracuse, New York, in 1983. He is a professor in the Electrical Engineering Department at the University of Maryland, College Park, Maryland. His research interests include computer systems and networks, graph theory, and parallel processing.

Dr. Oruç is a member of the IEEE Computer, Communications, and Information Theory Societies and a senior member of the IEEE.