Redundant binary number representation for an inherently parallel arithmetic on optical computers

Giuseppe A. De Biase and Annalisa Massini

A simple redundant binary number representation suitable for digital–optical computers is presented. By means of this representation it is possible to build an arithmetic with carry-free parallel algebraic sums carried out in constant time and parallel multiplication in log N time. This redundant number representation naturally fits the 2's complement binary number system and permits the construction of inherently parallel arithmetic units that are used in various optical technologies. Some properties of this number representation and several examples of computation are presented.

Key words: Optical computing, redundant number representation, parallel arithmetic units.

1. Introduction

To exploit the inherent parallelism of an optical computing machine fully, one needs suitable and fast arithmetic units. The problems of attaining parallel carry-free addition and parallel multiplication have been investigated by many authors, who have used two main approaches: the residue number system[1,2] and redundant number representations.[3] In fact, by means of both approaches it is possible to build totally parallel adders that operate by symbolic substitution and in constant time (the adding time is independent of the operand digit string length N). Using the residue number system, one can also perform parallel multiplication in constant time by symbolic substitution, but the number and the size of the truth tables required for both addition and multiplication increase greatly with the numerical range involved.[2] On the contrary, additions of redundant numbers can be performed in constant time by small truth tables and are independent from the digit position and from the numerical range; in this case, multiplications can be performed in log time. There are numerous studies on this subject concerning digital–optical computers in the case of free transmission of information (see, e.g., Refs. 4–11). The modified signed digit (MSD) representation is the method most widely investigated; there have been numerous studies, particularly in the digital–optical computing field (see, e.g., Refs. 9–11).

Here, a recoded version of the carry–save number representation[12] is presented. This simple redundant-binary (RB) number representation permits one to build an inherently parallel arithmetic with a two-step carry-free algebraic sum, it naturally fits the 2's complement binary number system, it presents the same advantages as other redundant number representations, and it requires only two symbols {0, 1} instead of three (such as the MSD). The RB representation permits the construction of efficient arithmetic units operating by means of symbolic substitution or other optical technologies (e.g., on the optical cellular processor, on which an arithmetic faster than that presented by Huang *et al.* can be carried out[13,14]).

2. Redundant Binary Representation

In the natural binary system an unsigned integer D is obtained by

$$D = \sum_{i=0}^{N-1} a_i 2^i, \tag{1}$$

where the digits are $a_i \in \{0, 1\}$ and the most-significant digit is the leftmost of the string. An integer D obtained by

$$D = \sum_{i=0}^{N-1} a_i 2^{i-\lceil i/2 \rceil}, \quad N \text{ even}, \tag{2}$$

is in RB representation. In Eq. (2) (where $\lceil\ \rceil$ represents the rounding to the upper integer), again $a_i \in$

The authors are with the Dipartimento di Scienze dell'Informazione, Università di Roma la Sapienza, Via Salaria 113, 00198 Roma, Italy.

Received 2 April 1992.
0003-6935/93/050659-06$05.00/0.

{0, 1} and the most-significant digit is the leftmost of the digit string. For $i = \ldots 7, 6, 5, 4, 3, 2, 1, 0$, Eq. 2 generates the following sequence of position weights:

$$\begin{array}{cccccccc} \ldots\ 8 & 8 & 4 & 4 & 2 & 2 & 1 & 1 \\ r & n & r & n & r & n & r & n. \end{array}$$

This weight sequence characterizes the RB number representation. As can be seen, all position weights are doubled. In each pair of digits of the same weight rn, the left and the right digit are called, respectively, the r (redundant) and the n (normal) digit. From Eq. (2) there follows that an RB representation of a number can be obtained from its binary representation by the use of the following recoding rules:

$$0 \rightarrow 00, \quad 1 \rightarrow 01.$$

The RB number obtained in this way is in canonical form. This coding operation is performable in parallel in constant time (one elemental logic step). Each RB number has a canonical form and several redundant representations. Some examples of unsigned RB numbers, in canonical and redundant representations, are shown in Table 1.

3. Arithmetic with Redundant Binary Numbers

A. Addition of Unsigned Numbers by Symbolic Substitution

The RB number representation permits totally parallel arithmetic operations to be performed by symbolic substitution. The addition of two RB numbers can be performed by using rules defined by truth tables that act on rn digit pairs. Table 2 (T_{rn}) shows these rules, which, applied in parallel on all pairs of two RB numbers (operands) arranged on two superposed and aligned rows, obtain the pairs of two output RB numbers, still on two superposed and aligned rows. The procedure is described below.

Procedure 1

(a) The input is two superposed and aligned RB numbers.

(b) Step 1 is the parallel application of the rules in Table 2 on all rn pairs (this step generates an intermediate sum).

Table 2. Symbolic Substitution Rule Truth Table for Redundant Binary Number Addition[a]

	u			
l	00	01	10	11
	00	10	00	10
00	00	00	01	01
	00	10	00	10
01	01	01	10	10
	00	10	00	10
10	01	01	10	10
	00	10	00	10
11	10	10	11	11

[a]This table acts on rn pairs; u and l indicate the upper and lower rows (inputs) respectively, of each cell. The lower output pair is shifted left one position.

(c) Step 2 is the parallel application of the rules in Table 2 on all rn pairs of the intermediate sum (this step generates the result).

(d) The output is the sum on the lower row and the zero is in canonical form on the upper row.

Figure 1 shows in more detail the alignment and the truncation of RB numbers for a correct application of Table 2.

Remark. The input of Table 2 is two pairs of rn digits (pairs of digits having the same weight) with digit positions $i + 1$ and i, respectively; the output is (a) an upper pair rn with digit positions $i + 1$ and i respectively, and (b) a lower pair nr with digit positions $i + 2$ and $i + 1$, respectively. The table is built in such a way that the sum of the input pairs is always equal to the sum of the output pairs.

Lemma 1. If Table 2 is applied on two superposed and aligned bit strings representing two RB numbers L_{in} and U_{in} (lower and upper operand), as shown in Fig. 1, the result is always two superposed and aligned RB numbers L_{out} and U_{out}, where $L_{\text{in}} + U_{\text{in}} = L_{\text{out}} + U_{\text{out}}$. $\square$

Proof. The application of Table 2 preserves the sum of pairs (see Remark). An RB number is a sum of rn pairs [see Eq. (2)]; then $L_{\text{in}} + U_{\text{in}} = L_{\text{out}} + U_{\text{out}}$. $\blacksquare$

Lemma 2. If Table 2 is applied twice on two superposed and aligned bit strings (upper and lower

Table 1. Example of Unsigned Redundant Binary Numbers in Canonical and Redundant Representation

		Redundant Binary Representation							
Integer	Binary	Canonical	Redundant						
0	000	000000							
1	001	000001	000010						
2	010	000100	000011	001000					
3	011	000101	001001	000110	001010				
4	100	010000	000111	001100	001011	100000			
5	101	010001	010010	001101	001110	100001	100010		
6	110	010100	010011	011000	001111	100100	100011	101000	
7	111	010101	010110	011001	100101	100110	011001	010110	101010

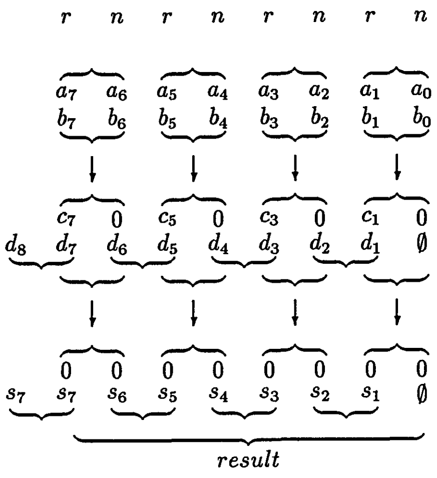
r n r n r n r n
a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
c_7 0 c_5 0 c_3 0 c_1 0
d_8 d_7 d_6 d_5 d_4 d_3 d_2 d_1 $\emptyset$
0 0 0 0 0 0 0 0
s_7 s_7 s_6 s_5 s_4 s_3 s_2 s_1 $\emptyset$
result

Fig. 1. Parallel application of the rules in Table 2 on two RB numbers to perform addition by symbolic substitution. At each step the input and the output are on two strings: a_i and b_i are the bits of the input operands, c_i and d_i are the bits of the intermediate sum, and s_i values are the bits of the result. The symbol $\emptyset$ is a padded zero.

strings) N bits wide (with N even), two resulting bit strings are obtained, and the resulting upper string is always formed by N zeros. □

Proof. The two input bit strings can be considered strings of pairs, $\ldots p_2p_1p_0$, where $p_i \in \{00, 01, 10, 11\}$. After the first application of Table 2 the resulting upper string contains only pairs $p_i \in \{00, 10\}$. In consequence, during the second application, only columns 00 and 10 of this table are used and only pairs $p_i = 00$ are generated in the upper string. ■

Theorem 1. Procedure 1 gives the RB representation of the sum of two RB numbers. □

Proof. Lemma 1 states that the application of Table 2 on two superposed and aligned RB numbers U_{in} and L_{in} (upper and lower operand) obtains two superposed and aligned RB numbers U_{out} and L_{out}, with $U_{\text{in}} + L_{\text{in}} = U_{\text{out}} + L_{\text{out}}$; this property is obviously valid also for subsequent applications. Lemma 2 states that, after two applications of this table, U_{out} is always a string of N zeros (the RB zero in canonical form) and, consequently, $U_{\text{in}} + L_{\text{in}} = L_{\text{out}}$. ■

In Fig. 2 an example of addition of two RB numbers is shown. As can be seen, the addition is completed

$(409)_{10}$ $(00001111000100011110)_{RB}$
$(576)_{10}$ $(10000000010110101011)_{RB}$

$(214)_{10}$ $(00001010001000101000)_{RB}$
$(771)_{10}$ $(10001010101010110110)_{RB}$

$(0)_{10}$ $(00000000000000000000)_{RB}$
$(985)_{10}$ $(10010100110011110010)_{RB}$

Fig. 2. Addition of two unsigned RB numbers by symbolic substitution using Procedure 1. After two steps the upper row is zeroed, and the lower row contains the sum.

in parallel in constant time (two elemental logic steps).

B. Signed Numbers and Algebraic Sum

Following the definition of RB numbers, and in analogy with the 2's complement binary system, a signed RB number is obtained by

$$D = -\sum_{i=N-2}^{N-1} a_i 2^{i-\lceil i/2\rceil} + \sum_{i=0}^{N-3} a_i 2^{i-\lceil i/2\rceil}, \quad N \text{ even.} \quad (3)$$

Equation (3) guarantees that the canonical representation of a signed RB number can be obtained from the corresponding 2's complement binary number with the same recoding used for unsigned numbers.

The same procedure of the addition of two unsigned RB numbers obtains the algebraic sum of two signed RB numbers (see Theorem 1). Hence the additive inverse of an RB number is obtained by the following Procedure 2, which is similar to that used in the 2's complement number system, taking into account the fact that the value of the negation of all RB representations of the number $(0)_{10}$ is $(-2)_{10}$ [in the 2's complement binary system it is $(-1)_{10}$].

Procedure 2

(a) The input is an RB number.
(b) In step 1 all digits of the RB number are negated.
(c) Step 2 is the algebraic sum between the RB canonical form of $(2)_{10}$ and the RB number.
(d) The output is the additive inverse of an RB number.

By using the additive inverse of an RB number, the subtraction of two signed RB numbers is then permitted. The additive inverse of an RB number can be obtained in parallel and in constant time (three logic steps). Figure 3 shows examples of the computation of the algebraic sum of two RB numbers and of the additive inverse of an RB number.

C. Decoding

In computing the value of an RB number, the use of Eq. (2) can generate values outside the range of the corresponding binary number or of the 2's complement binary number. This fact can mask the true sign of RB signed numbers.

Let RB_n and RB_r be two digit strings, $\text{RB}_n = \ldots n_6n_4n_2n_0$ and $\text{RB}_r = \ldots r_7r_5r_3r_1$, where n is the normal digit of an RB number and r is the redundant one. The decoding of RB numbers, with the correct truncation, can always be performed with the following simple procedure that derives directly from the RB number definition.

Procedure 3

(a) The input is RB_n and RB_r.
(b) Step 1 is the binary addition $\text{RB}_n + \text{RB}_r$. Only the first $N/2$ bits are considered.

$$\begin{array}{rl}
(419)_{10} & (00101000001011000110)_{RB} \\
(-452)_{10} & (10000000011000110111)_{RB} \\
\hline
(10)_{10} & (00000000000010001000)_{RB} \\
(-43)_{10} & (10101000110011001110)_{RB} \\
\hline
(0)_{10} & (00000000000000000000)_{RB} \\
(-33)_{10} & (10101001000110011010)_{RB}
\end{array}$$

(a)

$$\begin{array}{rl}
(499)_{10} & (00011001001101110110)_{RB} \\
\hline
(-501)_{10} & (11100110110010001001)_{RB} \\
(2)_{10} & (00000000000000000100)_{RB} \\
\hline
(-351)_{10} & (10001000100000000010)_{RB} \\
(-148)_{10} & (10100010100010010000)_{RB} \\
\hline
(0)_{10} & (00000000000000000000)_{RB} \\
(-499)_{10} & (00101011000010100010)_{RB}
\end{array}$$

(b)

Fig. 3. (a) Algebraic sum of two signed numbers; (b) computation of the additive inverse of an RB number by means of Procedure 2.

(c) The output is the corresponding binary or 2's complement binary number.

Namely, RB numbers can be converted into binary or 2's complement binary ones by splitting the RB number into two binary numbers: the first is obtained by the n digits and the second is obtained by the r digits. The binary sum of these two numbers obtains the expected value in natural binary representation. Using CLA adders,[15] one can perform this operation in $O(\log_2 N)$ time. Signed numbers (in 2's complement binary representation) can be encoded as (decoded from) signed RB numbers by the same rules used for unsigned numbers. An example of decoding is shown in Fig. 4. If an RB number is in canonical form, the decoding operation is trivial and performable in parallel in constant time (one elemental logic step; see Section 2). The nonconstant time complexity of the decoding operation is not a serious problem because this operation is used only when the data must be presented to the external world.

$$(\overbrace{11100110110010001001}^{N})_{RB}$$

$$\begin{array}{ll}
\mathrm{RB}_n + & (1010100001)_2 \\
\mathrm{RB}_r = & (1101101010)_2 \\
\hline
(-501)_{10} & (1\,\underbrace{1000001011}_{N/2})_2
\end{array}$$

Fig. 4. Example of decoding of a signed RB number. The least-significant $N/2$ bits of the result are the binary or 2's complement binary value.

D. Multiplication

Equation (2) guarantees that the RB numbers have shifting properties similar to those of positional number systems. In fact, multiplication by two is obtained if the RB number is shifted left by two positions. Keeping this fact in mind, one can carry out a multiplication algorithm with $O(\log_2 N)$ time complexity following the same outline as Vuillemin[12] or Takagi *et al.*[16] (redundant binary tree).

4. Properties of Redundant Binary Representation

A. Zero and Its Detection

In the case of unsigned RB numbers the $(0)_{10}$ has only the RB canonical form (see Table 1) and is hence easily detectable. In the case of signed RB numbers, $(0)_{10}$ has many RB representations. As an example, for six-digit signed RB numbers the representations of the $(0)_{10}$ are

$$(0)_{10} \leftrightarrow (000000)_{\mathrm{RB}}(101011)_{\mathrm{RB}}(101100)_{\mathrm{RB}}$$
$$(100111)_{\mathrm{RB}}(010111)_{\mathrm{RB}}(011100)_{\mathrm{RB}}(011011)_{\mathrm{RB}}.$$

This difficulty can be overcome by using the number $(-1)_{10}$ instead of $(0)_{10}$. In fact, any redundant representation of the number $(-1)_{10}$ obtains the canonical representation of the $(-1)_{10}$ if the following rules acting on *rn* pairs are applied (see Theorem 2 in Appendix A):

$$01 \to 01, \quad 10 \to 01; \tag{4}$$

then the $(-1)_{10}$ can be detected. If the result of an algebraic sum between an RB number and an RB representation of $(-1)_{10}$ is an RB representation of the number $(-1)_{10}$ again, this RB number is a representation of $(0)_{10}$. Then the procedure to detect the number $(0)_{10}$ can be as shown below.

Procedure 4

(a) Input an RB number.

(b) Step 1 is the algebraic sum between the RB canonical form of $(-1)_{10}$ and the RB number.

(c) Step 2 is the application of rules (4) to the result.

(d) The output is the RB canonical form of $(-1)_{10}$ or of another RB number.

Procedure 4 can be used to compare two RB numbers by our using the following procedure.

Procedure 5

(a) The input is the first RB number and the additive inverse of a second RB number.

$$\begin{array}{rl} & (00\,\overbrace{0000010011010010}^{(53)_{10}}\,00\,\overbrace{0101001000110010}^{(-39)_{10}})_{RB} \\ (27353)_{10} & \\ (30429)_{10} & (00\,\underbrace{0000010100110101}_{(59)_{10}}\,00\,\underbrace{0101000101001101}_{(-35)_{10}})_{RB} \end{array}$$

$$\begin{array}{rl} (22724)_{10} & (00\,\overbrace{0000100010100000}^{(44)_{10}}\,00\,\overbrace{1010000000100000}^{(-60)_{10}})_{RB} \\ (35058)_{10} & (00\,\underbrace{0000101011001100}_{(68)_{10}}\,00\,\underbrace{1010010010110100}_{(-14)_{10}})_{RB} \end{array}$$

$$\begin{array}{rl} (0)_{10} & (00\,\overbrace{0000000000000000}^{(0)_{10}}\,00\,\overbrace{0000000000000000}^{(0)_{10}})_{RB} \\ (57782)_{10} & (00\,\underbrace{0001001110110000}_{(112)_{10}}\,01\,\underbrace{0100100011101000}_{(-74)_{10}})_{RB} \end{array}$$

Fig. 5. Example of the algebraic sum of two concatenated RB numbers. Each string can be considered as a single RB number or as two RB numbers separated by a pair $p_i = 00$.

(b) Step 1 is the algebraic sum between the two RB numbers.

(c) Step 2 is the application of Procedure 4 to the result.

(d) The output is the RB canonical form of $(-1)_{10}$ or of another RB number.

B. Concatenation of Redundant Binary Numbers

The algebraic sum of RB numbers presents a property that is useful for optical computers, in which information is organized on a two-dimensional (2-D) array with rows much wider than the length of the number digit string. A digit string may be considered either as a single RB number or as the concatenation of many RB numbers; in this case, Procedure 1 obtains the respective concatenations of the sums when the RB number on both input strings has at least one pair $rn = 00$ on its left end as a separator (see Theorem 3 in Appendix A and the example in Fig. 5).

5. Concluding Remarks

The efficiency E of a redundant number representation can be obtained by

$$E = \frac{R_B}{R_R}, \tag{5}$$

where R_B is the number of bits needed in the binary number system and R_R is the number of bits needed in a redundant number representation to write the same value. The efficiency E is related to the numerical range reached by the different number representations when bit strings of the same length are used. In the MSD and carry–save representations, three symbols are needed ($\{\bar{1}, 0, 1\}$ and $\{0, 1, 2\}$, respectively), and therefore each digit requires two bits, while the RB representation needs two symbols $\{0, 1\}$, requiring only one bit. As one can see from Table 3, the efficiencies of the various redundant representations are all equal. As a consequence, the hardware complexity of computer components is the same in every case. In Table 3 the residue number system is neglected because its efficiency, computed by Eq. (5), strongly decreases when the length of the bit string N increases.

Table 3. Efficiencies of Different Redundant Number Representations

Number Representation	Digit × Position Weight	Bit × Digit	R	E
Binary	1	1	1	1
Modified signed digit	1	2	2	0.5
Carry–save	1	2	2	0.5
Redundant binary	2	1	2	0.5

If the RB number representation is used, the main arithmetic operations on signed numbers are permitted in an inherently parallel manner. The computations are made quickly and are independent of the operand digit string length. The RB number representation naturally fits the 2's complement binary number system, and for this reason, an easy manipulation of signed numbers is permitted. The decoding operation has log N time complexity (as a MSD representation), but it is needed only when the data are to be presented to the outside world. Using symbolic substitution, one can obtain a two-step algebraic sum with rules having extremely small truth tables.

Because the carry propagation is limited to the adjacent digit the RB arithmetic can also be implemented on an optical cellular processor[13] or on other processors based on a suitable 2-D algebra (see, e.g., Ref. 17). In this way an arithmetic faster than that presented by Huang *et al.*[14] and with better features is permitted; in fact, the properties of the concatenation of RB numbers can be useful for achieving a type of autoreconfigurable arithmetic unit suitable for 2-D organized data (with free format) or for row-coded images.[14]

Appendix A

Lemma 3. In the RB representations of the number $(-1)_{10}$, only pairs $rn \in \{01, 10\}$ are present. □

Proof. If an RB representation of $(-1)_{10}$ is decoded by means of Procedure 3, the result of this decoding operation has to be $(\ldots 1111)_2$. This result is possible only if $\mathrm{RB}_n = \overline{\mathrm{RB}}_r$. As a consequence, in each rn pair $a_n = \bar{a}_r$. ■

Theorem 2. Given any redundant representation of the number $(-1)_{10}$, its canonical representation can always be obtained by the following rules acting on rn pairs:

$$01 \rightarrow 01, \quad 10 \rightarrow 01. \qquad □$$

Proof: It follows immediately from Lemma 3 and Eq. (2). ■

Theorem 3. If two digit strings are a concatenation of many RB signed numbers, Procedure 1 always obtains the respective concatenation of sums when the RB number on both input strings has at least one pair $rn = 00$ on its left end (as a separator) in the same position in both upper and lower rows. □

Proof. Procedure 1 consists of two applications of Table 2 on two aligned and superposed strings of rn pairs. As can easily be seen, the application of Table 2 obtains the digits i and $i + 1$ of an rn pair on the upper row, it obtains digits $i + 1$ and $i + 2$ of a pair on the lower row, and it permits a limited carry propagation on position $i + 2$ on the lower string. The second application of the table permits a further carry propagation on position $i + 3$. If positions $i + 2$ and $i + 3$ coincide with the position of the separator and are masked by a pair 00 in the resulting string, the correct truncation is performed, and each concatenated RB number does not affect its predecessor. ■

The authors are grateful to Corrado Böhm and Renato Capocelli for their valuable suggestions.

References

1. H. L. Garner, "The residue number system," IRE Trans. Electron. Comput. **EC-8,** 140–147 (1959).
2. C. C. Guest and T. K. Gaylord, "Truth-table look-up optical processing utilizing binary and residue arithmetic," Appl. Opt. **19,** 1201–1207 (1980).
3. A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," IRE Trans. Electron. Comput. **EC-10,** 389–400 (1961).
4. K.-H. Brenner, "New implementation of symbolic substitution logic," Appl. Opt. **25,** 3061–3064 (1986).
5. K.-H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," Appl. Opt. **25,** 3054–3060 (1986).
6. M. M. Mirsalehi and T. K. Gaylord, "Truth-table look-up parallel data processing using an optical content addressable memory," Appl. Opt. **25,** 2277–2283 (1986).
7. G. Eichmann, Y. Li, and R. R. Alfano, "Optical binary coded ternary arithmetic and logic," Appl. Opt. **25,** 3113–3121 (1986).
8. M. M. Mirsalehi and T. K. Gaylord, "Logical minimization of multilevel coded functions," Appl. Opt. **25,** 3078–3088 (1986).
9. R. P. Bocker, B. L. Drake, M. E. Lasher, and T. B. Henderson, "Modified-signed addition and subtraction using optical symbolic substitution," Appl. Opt. **25,** 2456–2457 (1986).
10. Y. Li and G. Eichmann, "Conditional symbolic modified signed-digit arithmetic using optical content-addressable memory logic elements," Appl. Opt. **26,** 2382–2333 (1987).
11. A. K. Cherri and M. A. Karim, "Modified-signed digit arithmetic using an efficient symbolic substitution logic," Appl. Opt. **27,** 3824–3827 (1988).
12. J. Vuillemin, "A very fast multiplication algorithm for VLSI implementation," Integration VLSI J. **1,** 39–52 (1983).
13. K.-S. Huang, B. K. Jenkins, and A. A. Sawchuck, "Binary image algebra and optical cellular logic processor design," Comput. Vis. Graph. Image Process. **45,** 295–345 (1989).
14. K.-S. Huang, B. K. Jenkins, and A. A. Sawchuck, "Image algebra representation of parallel optical binary arithmetic," Appl. Opt. **28,** 1263–1278 (1989).
15. V. Chandran, T. F. Krile, and J. F. Walkup, "Optical techniques for real-time binary multiplication," Appl. Opt. **25,** 2272–2276 (1986).
16. N. Takagi, H. Yasuura, and S. Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree," IEEE Trans. Comput. **C-34,** 789–796 (1985).
17. M. Fukui and K.-I. Kitayama, "Image logic algebra and its optical implementations," Appl. Opt. **31,** 581–591 (1992).