where \(2d\) is descriptive of the "diameter" of the pattern space occupied by the set; that is, it is a measure of spread. The "effective volume \(v^*\) occupied by the given set of patterns in terms of points in the total binary space is

\[
m \leq v = 2^d = 2^n/2^A \leq 2^n \text{ points}.
\]

If the source consists of all possible patterns but they occur with unequal frequency then \(A\) will exceed zero and as the frequency of one pattern increases to predominate over all others the value of \(A\) increases towards \(n\).

VII. CONCLUSIONS

The matrix transformations presented here can be employed in the design of networks for analyzing binary patterns [10], [12], [14]. The three matrix transformations \(R_n\), \(U_n\), and \(U_m\) are nonsingular, so that if the observations are complete (all \(N\)-tuples) then the probability distribution of the patterns (given by \(F\)) can be obtained with the inverse matrices. The \(N\) and \(U\) transformations describe logic operations for the processing of pattern data and involve fewer operations (about one-half, direct or inverse) giving a computational advantage over the orthogonal \(R\) transformation. The latter has been used for processing two-dimensional pictures [17] and for generalized spectral analysis [2]. Moreover, the \(AND\) and \(NOR\) operations can be implemented in hardware for fast convenient processing and the results subsequently converted to the overall equivalent of the orthogonal \(R\) transformation with no net increase in computational complexity. All the transformations and inverses that exist can be calculated using fast algorithm techniques. Thus, different types of smoothing can be obtained from the different transformations by removing the higher order parameters and then applying the inverse transformations. For example, smoothing of binary pattern images occurs when higher order parameters in the orthonormal expansion are neglected [1]; but, this can now be translated into a form that is applicable when \(NOR\) gates are used for data acquisition. As an example of a quantity, descriptive of observed patterns, a measure of association has been defined for a set of binary patterns which can be derived from either the \(NOR\), \(AND\), or parity parameters.

REFERENCES

[1] N. A. Alexandridis and A. Klinger, "Real-time Walsh–Hadamard transformation," IEE Trans. Comput. (Short Notes), vol. C-21, pp. 288–292, Mar. 1972.
[2] H. C. Andrews and K. L. Caspary, "A generalized technique for spectral analysis," IEEE Trans., vol. C-19, pp. 16–25, 1970.
[3] H. Butin, "A compact definition of Walsh functions," IEEE Trans. Comput., vol. C-21, pp. 590–592, 1972.
[4] A. T. Butson, "Generalized Hadamard matrices," Proc. Amer. Math. Soc., vol. 13, pp. 894–898, 1962.
[5] P. Galingaert, "Switching function canonical forms based on commutative and associative binary operations," AIEE Trans. Commun. Electron., vol. 79, pp. 808–814, 1961.
[6] C. K. Chow, "An optimum character recognition system using decision functions," IRE Trans. Electron. Comput., vol. EC-6, pp. 247–254, 1957.
[7] P. F. Coleman, "Orthogonal functions for the logical design of switching circuits," IRE Trans. Electron. Comput., vol. EC-10, pp. 379–382, 1961.
[8] B. J. Fino and V. R. Algazli, "Slant Haar transform," Proc. IEEE (Lett.), vol. 62, pp. 653–654, May 1974.
[9] L. T. Fisher, "Unateness properties of AND-EXCLUSIVE-OR logic circuits," IEEE Trans. Comput., vol. C-23, pp. 166–172, Feb. 1974.
[10] K. Fukunaga and T. Itó, "A design theory of recognition functions in self-organizing systems," IEEE Trans. Electron. Comput., vol. EC-14, pp. 44–52, 1965.
[11] I. J. Good, "The interactive algorithm and practical Fourier analysis," J. Roy. Statist. Soc., vol. 20B, pp. 361–372, 1958.
[12] E. E. Gose, "An adaptive network for producing real functions of binary inputs," Inform. Contr., vol. 8, pp. 111–123, 1965.
[13] T. Itó, "A synthesis technique for networks consisting of logical functions feeding a linear summation element," IEEE Trans. Electron. Comput., vol. EC-14, pp. 254–256, 1965.
[14] T. Itó, "A note on a general expansion of functions of binary variables," Inform. Contr., vol. 12, pp. 206–211, 1970.
[15] R. J. Lechner, "Transformations among switching function canonical forms," IEEE Trans. Electron. Comput., vol. EC-12, pp. 129–130, 1963.
[16] J. Pearl, "Application of Walsh transform to statistical analysis," IEEE Trans. Syst., Man, Cybern., vol. SMC-1, pp. 111–119, 1971.
[17] W. K. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," Proc. IEEE, vol. 57, pp. 58–68, 1971.
[18] H. C. Ratz, "A new computer for process optimization," Int. J. Contr., vol. 1, pp. 81–99, 1965.
[19] A. M. Ulett, "The design of conditional probability computers," Inform. Contr., vol. 2, pp. 1–24, 1959.
[20] J. L. Walsh, "A closed set of normal orthogonal functions," Amer. J. Math., vol. 45, pp. 5–24, 1923.
[21] H. C. Wehrfritz, "Techniques for the transformation of logic equations," IEEE Trans. Comput., vol. C-23, pp. 477–480, May 1974.

A Note on Attrubin's Real-Time Iterative Multiplier

LAKSHMI N. GOYAL

Abstract—This correspondence presents a new multiplication algorithm for Attrubin's one-dimensional real-time iterative multiplier such that all the cells including the first cell in the array are identical in all respects, for the no-delay case.

Index Terms—Arithmetic, iterative array, multiplication, online multiplier, real-time multiplier.

I. INTRODUCTION

In his paper "A one-dimensional real-time iterative multiplier," [1] Attrubin presented the design of a real-time multiplier. His multiplier consists of a one-dimensional iterative bilateral array of cells (finite state machines) such that when the digits of two integers are presented to the cell at the extreme left end of the array a pair at a time, the same cell indicates the product digits at the rate of one per cycle. He had further shown that for the no-delay case the extreme left-hand cell or the initial cell had to be different from the rest of the cells which were all identical finite state machines in terms of the memory space needed as well as the control. In this correspondence we shall show that a no-delay real-time multiplication can be achieved by an identical set of cells including the initial cell, without increasing the complexity of the individual cell.

II. STRUCTURE OF THE MULTIPLIER

For the sake of simplicity, we shall use, wherever possible, the notation used by Attrubin.

Suppose the two integers to be multiplied are

\[
A = \sum_{i=0}^{n} a(i) 2^i, \quad a(i) \in \{0,1\}
\]

\[
B = \sum_{i=0}^{m} b(i) 2^i, \quad b(i) \in \{0,1\}
\]

where \(n\) and \(m\) are arbitrary.

Let the product be given by

Manuscript received July 17, 1974; revised January 31, 1975. This work was supported in part by the National Science Foundation under Grant US-NSP-GJ-3820.

The author is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801.
\[ P = \sum_{i=0}^{m+n} p(i)2^i, \quad p(i) \in [1,0]. \]

Without loss of generality we shall assume that \( m = n \).

The multiplier consists of a one-dimensional, iterative, bilateral array of cells. Each cell has five storage locations labeled as \( TSL, USL, LSL, PP, \) and \( K \). The top three storage locations can be either in quiescent state \( Q \) or in state \( g \) where the latter means that the location contains a pair \( g(i) = [b(i), a(i)] \) of one multiplicand and one multiplier bit.

At any time \( t = l \), the states of the storage locations \( TSL_i, USL_i, \) and \( LSL_i \) of cell \( j \) are given as follows:

\[ TSL_j = \begin{cases} 
    g(l - j + 1) & \text{if } l \geq 3j - 3 \\
    Q & \text{otherwise}
\end{cases} \quad (1) 
\]
\[ USL_j = \begin{cases} 
    g(2j - 1) & \text{if } l \geq 3j - 1 \\
    Q & \text{otherwise}
\end{cases} \quad (2) 
\]
\[ LSL_j = \begin{cases} 
    g(2j - 2) & \text{if } l \geq 3j - 2 \\
    Q & \text{otherwise}.
\end{cases} \quad (3) 
\]

The two other storage locations \( PP \) and \( K \), respectively, act as the partial product bit register and the carry store. Initially, all the storage locations for the pair of multiplicand and multiplier bits are in quiescent state \( Q \) and the partial product bit register \( PP \) and carry store \( K \) contain zero.

**III. DEFINITION OF BINARY OPERATOR \( \oplus \)**

In order to clearly show the difference between our algorithm and Atrubin’s algorithm and for the ease of expression, we define first a new binary operator \( \oplus \).

Let the binary operator \( \oplus \) have the two operands \( U \) and \( V \) and suppose

\[ W = U \oplus V \]

where either or both of the operands can be in state \( g \) or \( Q \). Clearly there are three possible cases.

**Case 1:** When both operands \( U \) and \( V \) are in state \( g \), say \( g(m) \) and \( g(n) \), respectively.

\[ U \oplus V : = a(m) \times b(n) + a(n) \times b(m) \]

where

\[ g(m) = [b(m), a(m)]; \]
\[ g(n) = [b(n), a(n)]. \quad (4) \]

**Case 2:** When one of the operands say \( U \) is in state \( g \), say \( g(m) \) and the other operand \( V \) is in state \( Q \).

\[ U \oplus V : = a(m) \times b(m). \quad (5) \]

**Case 3:** When both the operands \( U \) and \( V \) are in state \( Q \).

\[ U \oplus V : = 0. \quad (6) \]

**IV. MULTIPLICATION ALGORITHMS**

**A. Atrubin’s Algorithm**

Atrubin’s algorithm for the calculation of the contents of the partial product register and carry store of each cell can be concisely expressed in terms of our newly defined operator as follows. We must consider two cases.

**Case 1—Cell \( j \) When \( j > 1 \):** The contents of the partial product register \( PP \) and carry store \( K \) of cell \( j \) (\( j > 1 \)) at time \( t \) are given by

\[ PP_j = F(TSL_{j-1} \oplus USL_{j-1}, K_{j-1}, TSL_{j+1}, LSL, PP_{j+1}) \quad (7) \]
\[ = F_j \mod 2 \quad (8) \]

where \( TSL_{j-1} \) = state of storage location \( TSL \) of cell \( j \) at time \( t \), etc., and

\[ F_j = \begin{cases} 
    TSL_{j-1} \oplus USL_{j-1} + TSL_{j+1} \oplus LSL_{j+1} & \text{if } 1 \leq l \leq 2 \\
    TSL_{j-1} & \text{if } l > 2
\end{cases} \quad (9) \]

\[ K_j = F_j - PP_j \mod 2. \]

**Case 2—Cell \( j \) When \( j = 1 \):** Since cell 1 produces the final product digit, the calculation of the contents of product register \( PP \) and carry store \( K \) of cell 1 must take into account the states \( TSL_1 \) and \( LSL_1 \) in addition to other parameters of the function \( F \). In other words,

\[ PP_1 = H(TSL_1 \oplus USL_1, K_1, TSL_{j+1}, LSL_{j+1}, PP_{j+1}, TSL, LSL) \]
\[ = H_j \mod 2 \quad (10) \]

where

\[ H_j = F_j + TSL_1 \oplus LSL \]
\[ = [TSL_1 \oplus USL_1 - TSL_2 \oplus LSL_2 + PP_2^{-1}] + K_1^{-1} + TSL \oplus LSL. \quad (11) \]

The contents of carry store \( K_1 \) are given by

\[ K_1 = \frac{H_j - PP_j}{2}. \quad (12) \]

The final product bit \( p(i) \) is given by \( PP_i \)—the value of the partial product register of cell 1 at time \( t = i \).

It can be easily verified that relation (5) in the definition of the operator \( \oplus \) is equivalent in effect to Atrubin’s method of representing state \( Q \) as \( g(k) \) encircled, computing their contribution using relation (4) and then halving the contribution of the circled entries before being summed.

**B. Our Algorithm**

The contents of the product digit register \( PP \) and carry store \( K \) of any cell \( j \) for all \( j \) at time \( t \) are given by

\[ PP_j = G(TSL_j, LSL_j, USL_j, TSL_{j+1}, PP_{j+1}, K_j) \]
\[ = G_j \mod 2 \quad (13) \]

and

\[ K_j = \frac{G_j - PP_j}{2}. \quad (14) \]

where

\[ G_j = [TSL_j \oplus USL_j + USL_j \oplus TSL_{j+1} + PP_{j+1} + K_j^{-1}]. \quad (15) \]

Now the product digit \( p(i) \) of the final product \( P \) is given by the value of \( PP_i \), the contents of partial product register \( PP_i \) of cell 1 at time \( t = i \). In other words

\[ p(i) = PP_i = [TSL_1 \oplus USL_1 + USL_1 \oplus TSL_1 + PP_2^{-1} - K_1^{-1}] \mod 2. \quad (16) \]

A proof that the expression (16) correctly calculates the product digits \( p(i) \) can be found in [2].

**V. EXAMPLE**

As an example, we present in Fig. 1 the solution to the example given in Fig. 6 of Atrubin’s paper. The two integer operands to be multiplied are

\[ A = 00011010011101 \]
\[ B = 00000010111110. \]

The corresponding product \( P \) is given by

\[ P = 00010011101100011001111. \]
VI. CONCLUSION

We have shown that real time integer multiplication can be achieved by a one-dimensional iterative array of identical cells.

REFERENCES

[1] A. J. Atrubin, “A one-dimensional real-time iterative multiplier,” IEEE Trans. Electron. Comput., vol. EC-14, pp. 394–399, June 1965.

[2] L. N. Goyal, “A note on Atrubin’s real-time iterative multiplier,” Dep. Comput. Sci., Univ. Illinois, Urbana, Rep. UIUCDCS-R-74-646, July 1974.

Efficient Parallel Evaluation of Boolean Expressions

FRANCO P. PREPARATA AND DAVID E. MULLER

Abstract—A Boolean expression with n literals, i.e., n distinct appearances of variables, can be evaluated by a parallel processing system in at most 1.81 \( \log_2 n \) steps, or, equivalently, by a network constructed with two-input AND and OR gates and having at most 1.81 \( \log_2 n \) levels.

Index Terms—Boolean expressions, combinational networks, computational complexity, evaluation of Boolean expressions, parallel computation.

Brent et al. [1] suggested that their scheme for parallel evaluation of division-free arithmetic expressions could have applications in the area of logical design in order to obtain networks with small overall delay. Specifically, if \( n \) is the number of literals in a given expression \( E \) formed using the connectives OR and AND, they showed that no more than 2.465 \( \log_2 n \) levels of two-input gates are required for the evaluation of \( E \). Recently, Barak and Shamir [2] have obtained the tighter bound of 2 \( \log_2 n - 1 \) levels. In this correspondence we show that the bound can be further reduced to 1.81 \( \log_2 n \), by taking advantage of the dual distributivity of Boolean algebras. It must be noted that the model of two-input gates is suggested by that of two-operand processors and is somewhat unrealistic in the context of logical design. In fact, within the achievable fan-in, the gate propagation delay is essentially independent of the actual number of gate inputs; we feel, however, that the presented techniques could suggest useful approaches to the practical problem.

We assume that Boolean expressions are evaluated by two-input AND and OR gates and that complements appear only on single variables. A Boolean expression is called primitive if each variable appears just once. Any Boolean expression may be transformed into a primitive expression by replacing different appearances of any variable by distinct variables. Hence-