

Lofgren [12] treats a problem related to the one discussed in this section: namely, the problem of the existence of a contact net realizing a truth function $f(p_1, \dots, p_n)$ and having, for each i , at most one contact labeled p_i and at most one labeled $\bar{p}_i$.

BIBLIOGRAPHY

- [1] W. V. Quine, "Two theorems about truth functions," *Bol. de la soc. mat. mex.*, vol. 10, pp. 64-70; 1953.
- [2] R. McNaughton and B. Mitchell, "The minimality of rectifier nets with multiple outputs and unspecified cases," *J. Franklin Inst.*, vol. 264, pp. 457-480; 1957.
- [3] E. N. Gilbert, "Lattice-theoretic properties of frontal switching functions," *J. of Math. Phys.*, vol. 33, pp. 57-67; 1954.
- [4] M. C. Paull and E. J. McCluskey, Jr., "Boolean functions realizable with single threshold devices," *PROC. IRE*, vol. 48, pp. 1335-1337; July, 1960.
- [5] E. F. Moore, "Counter-example to a conjecture of McCluskey and Paull," unpublished; 1957.
- [6] R. C. Minnick, "Synthesis of linear input logic by the simplex method," presented at the Sixth Annual Symp. on Computers and Data Processing of the Denver Res. Inst., Denver, Colo.; July 30, 1959. Also "Linear-input logic," this issue p. 6.
- [7] R. Mattson, "A self-organizing binary system," *Proc. EJCC*, pp. 212-217; December 1-3, 1959.
- [8] D. G. Willis, "Plastic neurons as memory elements," 1959 IRE WESCON CONVENTION RECORD, pt. 4, pp. 55-65; also in, *Proc. Internatl. Conf. on Information Processing*, UNESCO House, Paris, France, pp. 290-298; 1959.
- [9] S. Muroga, "The principle of majority decision logical elements and the complexity of their circuits," *ibid.*, pp. 400-407.
- [10] R. O. Winder, "Single-Stage Threshold Logic," presented at the NEC-AIEE Joint Symp. on Switching Circuit Theory and Logical Design, Chicago, Ill.; October 11-12, 1960.
- [11] M. Karnaugh, "Pulse-switching circuits using magnetic cores," *PROC. IRE*, vol. 43, pp. 570-584; May, 1955.
- [12] I. Lofgren, "Irredundant and redundant Boolean branch-networks," *IRE TRANS. ON INFORMATION THEORY*, vol. IT-5, special suppl., pp. 158-175; May, 1959.

Linear-Input Logic*

ROBERT C. MINNICK†, MEMBER, IRE

Summary—Techniques are developed for the logical design of magnetic core circuits to produce arbitrary single-output combinational switching functions. The approach is based on the relationship of a single magnetic core circuit to a linearly separable switching function. A synthesis procedure is developed which uses a pair of logical primitives, AND with NOT and OR with NOT, which are similar to the STROKE primitive and its inverse. Procedures are developed for the synthesis of symmetric functions which require no more than the integral part of $(n+3)/2$ cores, approximately half the number used in previously published procedures. The synthesis of arbitrary switching circuits is treated as a linear programming problem, and a table of all four-variable circuits is presented in which no circuit requires more than three cores.

INTRODUCTION AND NOTATION

LINEARLY separable switching functions [1]-[8] have been studied under different names, such as *linearly separable* logic, *linear-input* logic, *threshold* logic, *majority* logic, and *voting* logic. The relationships of these switching functions with unate¹ functions have been studied [2], [8]-[12]; and other papers have been published showing uses of linearly separable functions in self-organizing systems [13], [14].

* Received by the PGEC, August 17, 1960; revised manuscript received, October 3, 1960. This paper is the result of work done while the author was employed by Burroughs Corp., ElectroData Div., Pasadena, Calif. Portions of this paper were delivered at the Sixth Annual Symp. on Computers and Data Processing of the Denver Res. Inst., Denver, Colo.; July, 1959.

† Stanford Res. Inst., Menlo Park, Calif.

¹ A unate switching function is one which can be represented as a normal form in which no variable appears both negated and unnegated.

In the present paper the problem of synthesizing arbitrary combinational switching circuits using linearly separable functions is considered. This problem has been partially treated by a number of writers [1], [3]-[7], and somewhat more completely by Muroga [5]. In this form of logical design, which in the present paper is termed *linear-input* logic, the binary inputs to a circuit are combined in a weighted linear sense, and the resulting sum is applied to a device which has threshold and amplifying properties.

Examples of circuits to which linear-input logic can apply are:

- 1) *Magnetic core circuits* [1], [23]—The inputs are the presence or absence of currents on several wires, each of which is associated with a given number of turns on the core. The resulting magnetomotive force is the weighted sum of the input currents, with the turns representing the weights.
- 2) *Resistor-transistor circuits* [15]—The inputs are one of the two voltage levels connected to a Kirchhoff resistor-adder. The resulting voltage applied to the base of a transistor is the weighted sum of the input voltages, with the resistor conductances representing the weights.
- 3) *Parametron circuits* [16]—The inputs are sinusoidal signals having one of two standard phases. These are applied to a linear weighted summing network, such as a magnetic core, and the resulting sum is amplified.

- 4) *Resistor tunnel-diode circuits* [17], [25]—The inputs are one of two voltage levels connected to a Kirchhoff resistor-adder in a manner very similar to that of circuit 2). The resulting voltage is applied to a network consisting essentially of two biased tunnel diodes.
- 5) *Multiple coil relay circuits* [18]—The inputs are one of two current levels which are connected to several coils on one relay core. The resulting magnetomotive force on the relay is the weighted sum of the input currents with the weights represented as the number of turns of the coils.

Simplified drawings of circuits for these five examples appear as Figs. 1 through 5. From the figures it is evident that all of these circuits, as well as others, have similar logical properties; however, the exact application details differ in each case. For the remainder of this paper, magnetic core circuitry is assumed; however, with appropriate modifications of the constraints, the methods may be made to apply for other circuit types. No consideration is given in the present paper of such engineering problems as transient behavior and current tolerances. This should not be construed as an indication that such problems are solved or are trivial.

Let there be $n+1$ input variables, $x_0, x_1, \dots, x_n$, to a circuit for the production of a combinational switching function, F , of n variables. Each of these inputs is connected to a winding of N_i turns of wire, where the sign of N_i indicates the polarity of that winding. One winding of N_0 turns is associated with an input x_0 which is always true; that is, with a bias generator. Let it be assumed that a true input ($x_i=1$) corresponds to a current of 1, while a false input ($x_i=0$) corresponds to a current of 0. With these assumptions the resulting magnetomotive force of the $n+1$ inputs is

$$M = \sum_{i=0}^n N_i x_i, \quad \text{with } x_0 = 1. \quad (1)$$

Since each term in the summation is an integer, M may take on only integral values. If M exceeds some threshold value, the magnetic core is set and the function $F=1$; that is, unit current is assumed to be available at the output. If M is less than this threshold, $F=0$ and no current is delivered at the output. Let it be assumed that

$$M \geq 1 \text{ corresponds to } F = 1,$$

$$M \leq 0 \text{ corresponds to } F = 0. \quad (2)$$

Let each N_i be the difference of two non-negative integers.

$$N_i = N_{i1} - N_{i0}, \quad N_{i1} \geq 0, \quad N_{i0} \geq 0, \quad (3)$$

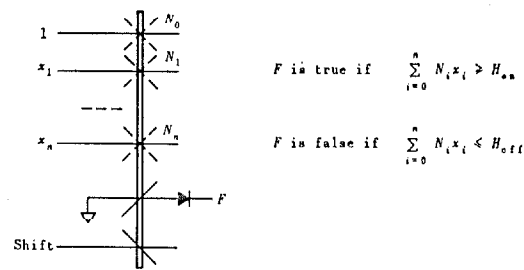


Fig. 1—Magnetic-core circuit.

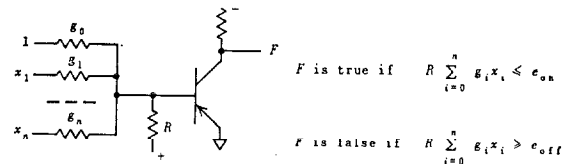


Fig. 2—Resistor-transistor circuit.

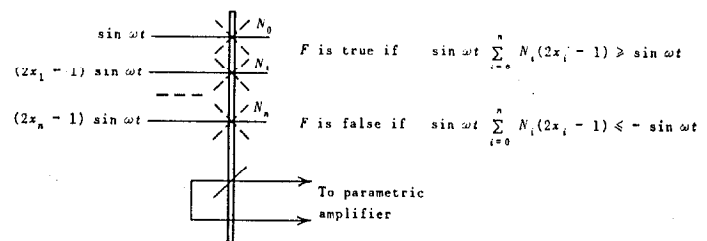


Fig. 3—Parametron circuit.

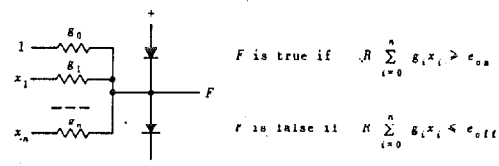


Fig. 4—Resistor tunnel-diode circuit.

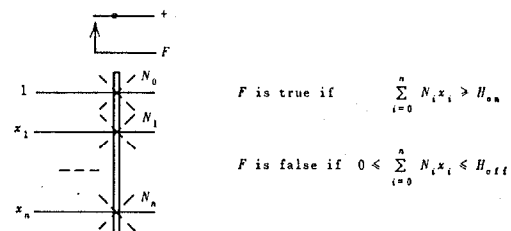


Fig. 5—Multiple coil relay circuit.

where it is assumed that only one of N_{i1} and N_{i0} is not zero. Then a linear-input circuit may be represented in a form which clearly displays the input connections:

$$F = (N_{00}, N_{10}x_1, N_{20}x_2, \dots, N_{n0}x_n, N_{01}, N_{11}x_1, N_{21}x_2, \dots, N_{n1}x_n), \quad (4)$$

where for a specific circuit the terms involving an $N_{ij}=0$ are understood to be omitted. In the form indicated by (4) all nonzero inputs to the left of the asterisks are connected negatively, while those to the right are connected positively.

To illustrate these definitions, consider the switching function shown in Table I. F is true for rows 0, 1, and 3 of this truth table. The switching function may therefore be represented in a notation similar to that used by Caldwell [19] as

$$F: \sum(0, 1, 3).$$

TABLE I
THREE-VARIABLE EXAMPLE FOR WHICH $F: \sum(0, 1, 3)$

State	x_3	x_2	x_1	F
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Using methods developed later in this paper, a linear-input circuit for producing this switching function consists of a single core with $N_0=N_1=1$, $N_2=-1$, and $N_3=-2$. The magnetomotive force equation is

$$M = 1 + x_1 - x_2 - 2x_3,$$

while the linear-input circuit representation is

$$F-(x_2, 2x_3 ** x_1, 1).$$

This magnetic-core circuit is given as Fig. 6, in which it is understood that the input and the shift pulses alternate.

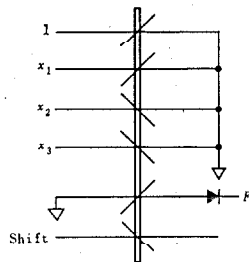


Fig. 6—Circuit for the function of Table I; $F: \sum(0, 1, 3)$.

Implicit in (1) and (2) are normalizing assumptions on the input and output thresholds; namely,

$$\begin{aligned} \text{true input: } x_i &= 1, & \text{false input: } x_i &= 0, \\ \text{true output: } M &\geq 1, & \text{false output: } M &\leq 0. \end{aligned} \quad (5)$$

A more general set of input and output thresholds is,

$$\begin{aligned} \text{true input: } x_i &= I_b, & \text{false input: } x_i &= I_a, \\ \text{true output: } M &\geq M_b, & \text{false output: } M &\leq M_a. \end{aligned} \quad (6)$$

Let the symbol $\hat{\cdot}$ correspond to a parameter for a circuit having the general thresholds in (6). Then (5) and (6) may be related by

$$\hat{x}_i = I_a + (I_b - I_a)x_i, \quad \hat{M} = M_a + (M_b - M_a)M.$$

Substituting these in (1),

$$\frac{\hat{M} - M_a}{M_b - M_a} = \sum_{i=0}^n N_i \left[\frac{\hat{x}_i - I_a}{I_b - I_a} \right].$$

Rearranging and noting that $\hat{x}_0 = I_b$,

$$\hat{M} = M_a + AI_b N_0 - AI_a \sum_{i=0}^n N_i + A \sum_{i=1}^n N_i \hat{x}_i.$$

But (1) must hold for the general thresholds; viz.,

$$\hat{M} = \sum_{i=0}^n \hat{N}_i \hat{x}_i = \hat{N}_0 I_b + \sum_{i=1}^n \hat{N}_i \hat{x}_i.$$

It follows that any circuit using normalized thresholds, (5), may be transformed into one having general thresholds as follows:

$$\begin{aligned} \hat{N}_0 &= AN_0 + \frac{M_a - I_a A \sum_{i=0}^n N_i}{I_b} \\ \hat{N}_i &= AN_i, \quad i = 1, 2, \dots, n, \end{aligned} \quad (7)$$

where

$$A = \frac{M_b - M_a}{I_b - I_a} \text{ is required to be integral.}$$

In particular, the thresholds customarily used for parametron circuits are $I_a = M_a = -1$, $I_b = M_b = 1$. For this case (7) becomes

$$\hat{N}_0 = N_0 - 1 + \sum_{i=0}^n N_i, \quad \hat{N}_i = N_i, \quad i = 1, 2, \dots, n. \quad (8)$$

It is interesting to note that for parametron circuits the algebraic sum of the turns is, from (8),

$$\sum_{i=0}^n \hat{N}_i = 2 \sum_{i=0}^n N_i - 1$$

which clearly is a positive or negative odd integer. Therefore, a parametron circuit produced by transforming a linear-input circuit with the thresholds of the present paper is guaranteed not to violate the known restriction that the magnetomotive force never be zero for any combination of the inputs.

Another transformation of interest is that which converts a linear-input circuit for a given function into the circuit for the *inverse* or Boolean complement function. This corresponds to choosing the thresholds for the inverse circuit as $I_a = M_b = 0$, $I_b = M_a = 1$. For this case (7) becomes

$$\hat{N}_0 = 1 - N_0, \quad \hat{N}_i = -N_i, \quad i = 1, 2, \dots, n. \quad (9)$$

To illustrate, the parametron circuit for the example of Table I is obtained from (8) as

$$F \text{ (parametron)} = (x_2, 2x_3, 1 ** x_1).$$

For the same example, the inverse function with normalized thresholds, (5), is obtained from (9),

$$F' = (x_1 ** x_2, 2x_3).$$

PRIMITIVE SYNTHESIS

The problem of synthesizing linear-input circuits is: given a statement of the desired combinational switching function, a set of windings, N_i , is to be found which produces the function. There is no assurance that an arbitrary function is producible with one core; indeed, the function $F = x_1x_2' + x_1'x_2$ requires two cores. Therefore, the question naturally arises of the possibility of producing an arbitrary combinational switching function with linear-input circuits. This question may be answered affirmatively; in fact, any function may be generated in one of a number of ways. Four such techniques will now be developed.

It is well known [20] that an arbitrary combinational switching function of n variables may be specified by choosing values for the 2^n binary constants f_i in four equivalent forms,

$$\begin{aligned} F &= \sum_{i=0}^{2^n-1} f_i p_i = \prod_{i=0}^{2^n-1} (f_i + p_i') = \sum_{i=0}^{2^n-1} (f_i' + p_i')' \\ &= \prod_{i=0}^{2^n-1} (f_i' p_i)', \end{aligned} \quad (10)$$

where

$$\begin{aligned} p_0 &= x_n' x_{n-1}' \cdots x_2' x_1', \\ p_0' &= x_n + x_{n-1} + \cdots + x_2 + x_1, \\ p_1 &= x_n' x_{n-1}' \cdots x_2' x_1, \\ p_1' &= x_n + x_{n-1} + \cdots + x_2 + x_1', \\ &\dots \\ p_{2^n-1} &= x_n x_{n-1} \cdots x_2 x_1, \\ p_{2^n-1}' &= x_n' + x_{n-1}' + \cdots + x_2' + x_1'. \end{aligned}$$

Each of the four alternative forms for a switching function, (10), involves the application of one or both of the

following fundamental functions, termed *primitive functions*:

$$\text{AND with NOT } F_a = v_1' v_2' \cdots v_p' v_{p+1} v_{p+2} \cdots v_{p+q},$$

$$\text{OR with NOT } F_b = u_1 + u_2 + \cdots + u_q + u_{q+1}'$$

$$+ u_{q+2}' + \cdots + u_{q+p}'. \quad (11)$$

But it is easy to verify that linear-input circuits to produce these two primitive functions are

$$F_a = (v_1, v_2, \dots, v_p, q - 1 ** v_{p+1}, v_{p+2}, \dots, v_{p+q}),$$

$$F_b = (u_{q+1}, u_{q+2}, \dots, u_{q+p} ** u_1, u_2, \dots, u_q, p). \quad (12)$$

In order to illustrate the ease of logical design using the primitive circuits of (12), consider again the example of Table I. By appropriate manipulation, this may be put into each of the forms of (10) as follows:

$$F = x_3' x_2' + x_3' x_1, \quad F = (x_3')(x_2' + x_1),$$

$$F = (x_3 + x_2)' + (x_3 + x_1)', \quad F = (x_3)'(x_2 x_1)'. \quad (12)$$

Using (12), four equivalent linear-input circuits may be constructed for this switching function:²

$$F = [** (x_3, x_2 * 1), (x_3 * x_1)],$$

$$F = [1 ** (x_3 * 1), (x_2 * x_1, 1)],$$

$$F = [(x_3, x_2), (x_1 * x_3, 1) ** 2],$$

$$F = [x_3, (x_1 * x_2) ** 1].$$

It is seen that the first three of these circuits require three cores each, while the fourth requires two cores. Since a one-core solution is given as Fig. 6, it is clear that the most economical circuits do not necessarily result from this method of logical design. On the other hand, the use of this method is reasonably straightforward.

SYMMETRIC FUNCTION SYNTHESIS

A *symmetric function* is a switching function which remains invariant to all permutations of the input variables. A well-known result of this definition is that a symmetric function of n variables may be described uniquely by stating its truth value for k true inputs with $k=0, 1, \dots, n$. For instance, the three-variable function $F: \sum(0, 3, 5, 6)$ is a symmetric function; and it may be described equivalently by stating that it is true for 0 or 2 true inputs.

It follows that an arbitrary symmetric function may be specified by stating the *ranges* of the number of true

² Although it is not strictly necessary, distinction is made between the first and second levels of a multiple-core circuit by the use of a single asterisk in the first level of the circuit notation, and a double asterisk in the second level.

inputs for which the function is true. Thus,

$$F=1 \text{ for } k \text{ inputs true, with } q_i \leq k \leq Q_i, i=1, 2, \dots, r \quad (13)$$

where q_i and Q_i , $i=1, 2, \dots, r$, are known constants for any specific function. As k can assume only the $n+1$ values $0, 1, \dots, n$, it may be ascertained that

$$r \leq [(n+1)/2] \quad (14)$$

where $[\]$ denotes "the integral part of." Procedures are given below for finding $(1+r)$ -core linear-input circuits for r -range symmetric functions. It follows from (14) that the maximum number of cores required to produce a linear-input circuit for an arbitrary symmetric function is $1 + [(n+1)/2]$. This is essentially half the number of cores required by Muroga [5].

It is claimed that the arbitrary symmetric function specified in (13) and (14) may be produced by either one of the following two circuits, where the symbol I stands for the inputs $x_1, x_2, \dots, x_n$:

$$F_c = [c_0, c_1(Q_1 * I), c_2(Q_2 * I), \dots, c_r(Q_r * I) ** I] \quad (15)$$

where

$$c_0 = q_1 - 1, \quad c_i = q_{i+1} - q_i, \quad i = 1, 2, \dots, r-1$$

$$c_r = \begin{cases} n+1 - q_r, & \text{if } Q_r \neq n \\ 0 & \text{if } Q_r = n \end{cases}$$

$$F_d = [d_1(I * q_1), d_2(I * q_2), \dots, d_r(I * q_r), I ** d_0]$$

where

$$d_0 = Q_r + 1, \quad d_1 = \begin{cases} Q_1 + 1 & \text{if } q_1 \neq 0 \\ 0 & \text{if } q_1 = 0 \end{cases}$$

$$d_i = Q_i - Q_{i-1}, \quad i = 2, 3, \dots, r. \quad (16)$$

To verify (15), it is necessary to show that the function is true for $q_j \leq k \leq Q_j$. As $Q_{j-1} < q_j$, it follows that the subsidiary circuits $(Q_i * I)$ are true only for $0 \leq i \leq j-1$. Therefore, the left side of (15) becomes

$$\sum_{i=0}^{j-1} c_i = q_j - 1;$$

and as $k \geq q_j$ of the inputs is true, the switching function is true as required. Next, it must be shown that the function is false for $Q_{j-1} < k < q_j$. But this follows immediately in that the left side of (15) is still $q_j - 1$, while only $k < q_j$ of the inputs is true. In a very similar manner, the alternative form, (16), may be verified.

Two additional circuits for arbitrary symmetric functions result by obtaining the circuits of (15) and (16) for the symmetric function which is the inverse of the one desired; the resulting circuits are inverted by (9).

It is interesting to note that a number of the examples treated in the previously cited references are symmetric

functions, and more particularly alternating symmetric (or parity) functions. Therefore, it makes a useful comparison to develop a linear-input circuit which is true for (say) an odd number of true inputs. Of the several alternative circuits which result from (15) and (16) and from the application of the inversion algorithm, (9), to the even alternating symmetric function, the following is among the most economical:

$$F = [2(1 * I), 2(3 * I), \dots, 2(2[n/2] - 1 * I) ** I]. \quad (17)$$

In particular for $n=3$,

$$F = [2(1 * x, y, z) ** x, y, z]. \quad (18)$$

This circuit is given as Fig. 7. In this figure subscripts

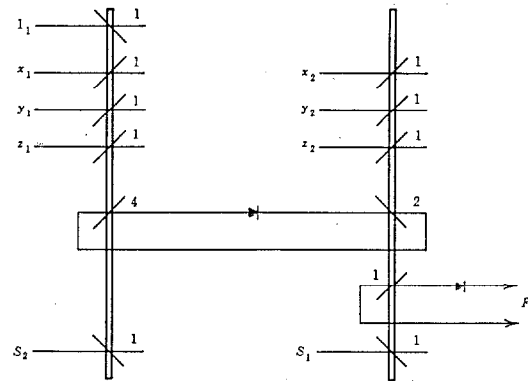


Fig. 7—Circuit for the three-variable odd alternating symmetric function.

indicate phases, while S_1 and S_2 are shift pulses. For $n=4$,

$$F = [2(1 * w, x, y, z), 2(3 * w, x, y, z) ** w, x, y, z]. \quad (19)$$

The three-variable case, (18), is treated by Wigington [4] and Lindaman [7]. The most economical result obtained by these authors, in the notation of the present paper, is

$$F = [1 ** x, (x * y, z), (x, y, z * 2)],$$

which requires three cores as compared with two for (18).

Muroga [5] treats symmetric functions more systematically. His example for the odd alternating symmetric function for $n=4$, after applying the inverse of the transformation given by (8), to account for the different thresholds, is

$$F = [(1 * w, x, y, z), (3 * w, x, y, z) ** (* w, x, y, z), (2 * w, x, y, z)],$$

which requires five cores as compared with three for (19).

LINEAR PROGRAMMING SYNTHESIS

The primitive synthesis techniques developed in the second section are attractive in that any of the four methods may be readily used to synthesize an arbitrary switching function in terms of one or two primitive circuits. However, the use of primitives in some cases appears to lead to more expensive circuits than are necessary; therefore, a general synthesis procedure will now be developed which does not depend on primitive operations. This method, at least in some cases, leads to less expensive circuits than do the previous techniques.

A switching function is completely specified if its binary value is given for each of the 2^n combinations of the n variables x_i . From (2) it is clear that for each combination of the variables, an inequality is established on M ; thus, there are 2^n inequalities in the N_{ij} . To illustrate, consider again the example of Table I. From (1) and (3)

$$\begin{aligned} N_{01} - N_{00} &\geq 1 \\ N_{01} + N_{11} - N_{00} - N_{10} &\geq 1 \\ N_{01} + N_{21} - N_{00} - N_{20} &\leq 0 \\ N_{01} + N_{11} + N_{21} - N_{00} - N_{10} - N_{20} &\geq 1 \\ N_{01} + N_{21} - N_{00} - N_{30} &\leq 0 \\ N_{01} + N_{11} + N_{31} - N_{00} - N_{10} - N_{30} &\leq 0 \\ N_{01} + N_{21} + N_{31} - N_{00} - N_{20} - N_{30} &\leq 0 \\ N_{01} + N_{11} + N_{21} + N_{31} - N_{00} - N_{10} \\ &\quad - N_{20} - N_{30} \leq 0. \end{aligned} \quad (20)$$

In general, each of the 2^n inequalities in the non-negative integers N_{ij} is one of two types,

$$\sum_{i=0}^n N_{i1}x_i - \sum_{i=0}^n N_{i0}x_i \geq 1 \text{ corresponds to } f_n = 1, \quad (21)$$

$$\sum_{i=0}^n N_{i1}x_i - \sum_{i=0}^n N_{i0}x_i \leq 0 \text{ corresponds to } f_q = 0. \quad (22)$$

Let a set of 2^n slack variables, $N_{sk} \geq 0$, for $k=0, 1, \dots, 2^n-1$, be defined. For equations of the type of (21), let the slack variable corresponding to f_p be subtracted from the left, and for equations of the type of (22), let the slack variable corresponding to f_q be added to the left. Then values of the non-negative slack variables may always be chosen which convert (21) and (22) into equalities. For reasons which will become evident, two additional sets, each of 2^n a - and b -variables, $N_{ak} \geq 0$ and $N_{bk} \geq 0$, for $k=0, 1, \dots, 2^n-1$, are defined. Let the a -variable corresponding to f_p be added to the left of

equations of the type of (21), and let the b -variable corresponding to f_q be subtracted from the left of equations of the type of (22). Thus, (21) and (22) become

$$\sum_{i=0}^n N_{i1}x_i - \sum_{i=0}^n N_{i0}x_i - N_{sp} + N_{ap} = 1$$

corresponds to $f_p = 1$, (23)

$$\sum_{i=0}^n N_{i1}x_i - \sum_{i=0}^n N_{i0}x_i + N_{sq} - N_{bq} = 0$$

corresponds to $f_q = 0$. (24)

Non-negative values of the a - and b -variables may always be chosen which do not violate the equalities of (23) and (24), providing that

$$N_{ap} \leq N_{sp} \text{ and } N_{bq} \leq N_{sq}. \quad (25)$$

While 2^{n+1} a - and b -variables are defined, only 2^n of them are used in any one problem. Therefore, for any function of n variables, (23) and (24) may be used to write 2^n equalities in $2(n+1+2^n)$ variables. Since there are more unknowns than equations, a unique solution is generally not possible, and it is of some importance to establish a criterion for choosing the best solution if any solutions exist. For this purpose, a *cost function* is defined as

$$C = \sum_{i=0}^n (N_{i1} + N_{i0}) + A \sum_{i=0}^{2^n-1} N_{ai} + B \sum_{i=0}^{2^n-1} N_{bi}, \quad (26)$$

where A and B are non-negative constants. The first summation of (26) represents the total number of input turns (disregarding the polarity) on the magnetic core; clearly it is desirable to have this quantity be a minimum. Since in the process to be described at least one of N_{ap} and N_{sp} will be zero, it follows from (25) that any nonzero N_{ak} indicates that the original set of inequalities has not been completely satisfied. Similarly, any nonzero N_{bk} indicates that the original set of inequalities has not been completely satisfied. Conversely, nonzero values of N_{sk} may occur without violating (21) or (22). Thus it is desirable to minimize, and if possible to reduce to zero, the second and third summations of (26).

It is clear, therefore, that the 2^n equations of the form of (23) and (24) must be solved for the $2(n+1+2^n)$ non-negative variables in such a way as to minimize (26). Stated in this manner, the problem evidently is one of linear programming [21]. The a -variables are the *artificial variables* of linear programming necessary for obtaining a *basic feasible solution*. That is, for equations of the type of (23), $N_{ap} - 1$, $N_{sp} - 0$, and the N_{ij} which occur

are chosen as zero. So, for the basic feasible solution, (26) becomes

$$C = A \sum_{i=0}^{2^n-1} N_{ai}.$$

If $A \gg n+1$ is chosen, the cost is reduced by causing one or more of the N_{ai} to go to zero and by simultaneously causing one or more of the N_{ij} to take on nonzero values. Therefore, by following one of the known methods for solving linear programming problems, such as the *simplex method*,³ it may be possible to find a solution for which all a -variables vanish. If this is the case, the specified function may be produced with one core; the N_{ij} determine the wiring.

The details of the simplex algorithm are not given here; it is sufficient to state that the process consists of finding successive solutions, beginning with the basic feasible solution, in such a manner that in proceeding from one solution to the next, the cost cannot be increased, and may be reduced. To illustrate the operation of this algorithm, consider the example treated before and stated initially in Table I. The set of inequalities in the form of (21) and (22) is given as (20). The introduction of slack variables and a - and b -variables in the form of (23) and (24) converts (20) into

$$\begin{aligned} N_{01} - N_{00} - N_{s0} + N_{a0} &= 1 \\ N_{01} + N_{11} - N_{00} - N_{10} - N_{s1} + N_{a1} &= 1 \\ N_{01} + N_{21} - N_{00} - N_{20} + N_{s2} - N_{b2} &= 0 \\ N_{01} + N_{11} + N_{21} - N_{00} - N_{10} - N_{20} - N_{s3} + N_{a3} &= 1 \\ N_{01} + N_{31} - N_{00} - N_{30} + N_{s4} - N_{b4} &= 0 \\ N_{01} + N_{11} + N_{31} - N_{00} - N_{10} - N_{30} + N_{s5} - N_{b5} &= 0 \\ N_{01} + N_{21} + N_{31} - N_{00} - N_{20} - N_{30} + N_{s6} - N_{b6} &= 0 \\ N_{01} + N_{11} + N_{21} + N_{31} - N_{00} - N_{10} - N_{20} - N_{30} \\ &\quad + N_{s7} - N_{b7} = 0. \end{aligned} \quad (27)$$

A basic feasible solution is obtained as described previously, and is

$$N_{a0} = N_{a1} = N_{a3} = 1, \text{ all other variables zero.}$$

Letting $A=100$ and $B=0$ in (26), the initial cost is $C=300$. If the simplex algorithm is applied for the present in such a way that all $N_{bk}=0$, it is found that the first two applications do not change the basic feasible solution or the cost. On the third application the nonzero variables and the cost are found to be

$$N_{a0} = N_{30} = N_{s4} = N_{11} = N_{s6} = 1, C = 102.$$

³ Although linear programming techniques exist for obtaining integer solutions [24], the simplex method was chosen for its relative simplicity. It is interesting to note that except for an insignificant number of unusual cases, the simplex method produced integer solutions.

That this solution is valid may be verified by substitution in (27). The fourth application results in no change in C , while the fifth produces

$$N_{30} = N_{s6} = 2, N_{20} = N_{01} = N_{s1} = N_{s4} = N_{11} = N_{s7} = 1, \\ C = 5.$$

At this point all a -variables have been reduced to zero, and a one-core solution has been found, namely,

$$F = (x_2, 2x_3 ** x_1, 1).$$

Situations exist for which a solution devoid of a -variables or b -variables, or both, cannot be obtained. For such cases, let an a -residue function, G , be defined as a switching function which is true for those code states for which there exist nonzero a -variables at the termination of the simplex process. Similarly, a b -residue function, H , is defined for the remaining b -variables. If G and H are different from the original function F , the residues may be treated as separate functions by the same linear programming algorithm.

Two examples should illustrate this. First, consider the function $F: \sum(2, 3, 4, 5)$. At the termination of the simplex process applied to this example, the nonzero variables are $N_{30} = N_{21} = 1$, $N_{a4} = N_{a5} = 2$. Therefore, the a -residue function is different, and is defined as

$$G: \sum(4, 5).$$

Re-entering this into the simplex algorithm leads to the circuit

$$G = (x_2 ** x_3).$$

Therefore, the entire circuit is

$$F = [x_3 ** 2(x_2 * x_3), x_2].$$

In a similar manner, the function $F: \sum(0, 3)$ leads to $N_{10} = N_{01} = N_{21} = 1$, $N_{b2} = N_{30} = 2$. Therefore, a b -residue function is defined and re-entered into the simplex process. The resulting circuit is

$$F = [2x_3, x_1, 2(x_3, x_1 * x_2) ** x_2, 1].$$

A program has been written for the Burroughs 220 computer for applying the simplex method to the synthesis of linear-input circuits for combinational switching functions of five and fewer variables. This program has been used to compute all distinct linear-input circuits of four variables under the transformations of inversion and permutation of the input variables, and inversion of the switching function. Following this the circuits were further simplified by a number of heuristic techniques which as yet cannot be considered to represent formal procedures.⁴ The list of the 237 distinct lin-

⁴ Credit is due to E. L. Glaser, now of the Burroughs Res. Center, who contributed extensively to the initial simplification, and to Dr. E. F. Moore, of Bell Telephone Labs., who, after the best efforts of E. L. Glaser and the author, was able to reduce some sixty three-core circuits to two-core circuits.

ear-input circuits is included as Table II. In that similar tables have been published for vacuum tube [20] and for relay [22] circuits, the notation of [20] has been followed. The four input variables are chosen as w, x, y , and z . The column labeled m indicates the number of true states in the truth table, while the column labeled s contains an index number for reference. The third column of Table II, labeled f_i , indicates the true states of the truth table, and a linear-input circuit is given in the fourth column.

An examination of the table indicates that of the 237 circuits which are listed, 14 may be produced with one core, 185 with two cores, and 38 with three cores. Taking into account the number of the 65,536 functions of four variables which map into each listed function, it is found that 1880 (3 per cent) of the functions of four variables may be produced with one core, 50,284 (77 per cent) with two cores, and 13,370 (20 per cent) with three cores. An upper bound on the number of cores required to produce any switching function of four variables by more conventional techniques is known to be $2^{n-1} + 1 = 9$ cores. In that the simplex algorithm leads to a one-core solution if it exists, it may be inferred that, for the assumed conditions, all one-core and two-core solutions require a minimum number of cores. The three-core circuits are not proved to be minimum.

In order to find the linear-input circuit for a function not listed in Table I, the transformation procedure given in Appendix I of [20] may be used, with the following modification: if $m > 8$, the circuit for the inverse function is first obtained. The combined transformation is determined as explained in the reference, and this transformation is applied to the circuit found in Table II by renaming the variables in the circuit in accordance with the transformation. If a variable is inverted in this transformation, the corresponding variable together with its coefficient is moved to the opposite side of the asterisks, both in the main and in the subsidiary circuits. If a variable is moved from the left side to the right side of the asterisks, the bias is corrected by the addition of a bias on the left side having the same value as the coefficient of the moved variable. Similarly, if a variable is moved from the right side to the left side, the correction bias is added on the right side of the asterisks. If initially $m > 8$, and the circuit for the inverse function is obtained, the resulting circuit after application of the combined transformation is inverted by (9).

To illustrate the use of Table II, suppose a linear-input circuit is desired for the switching function,

$$F = w(y'z + yz' + x'y') + w'y(x' + z) + x'y'z.$$

By conventional methods, it is easily discovered that $m = 9$ and $F: \sum(1, 2, 3, 7, 8, 9, 10, 13, 14)$. As $m > 8$, the inverse function is investigated, namely $\sum(0, 4,$

5, 6, 11, 12, 15). Application of this function to Table I. 1 of [20] leads to $s = 129$ and the combined transformation $(w, x, y, z) \rightarrow (x', w, y, z)$. The listed linear-input circuit for $s = 129$ is

$$F - [w, x, y, z ** 3(2 * x, y, z), 2].$$

Applying this combined transformation,

$$F - [w, y, z ** x, 3(2 * w, y, z), 1].$$

But this is the circuit for the inverse function, and therefore it must be inverted by (9). The resulting circuit is

$$F - [x, 3(2 * w, y, z) ** w, y, z].$$

Consider finally an example which is treated by Muroga [5],⁵

$$F = w'xyz + wx'y'z + wxy'z' + w'x'y'z'. \quad (28)$$

The linear-input circuit given by Muroga to produce this switching function, in the present notation and thresholds, is

$$F - \{3 ** [y, z ** (w * x), 2], [z ** (x * w), y, 1], \\ ** (1 * w, x), (w, x * 1), y, z], [y ** z, 1]\},$$

requiring nine cores.

From (28), it is evident that $m = 4$ and $F: \sum(0, 7, 9, 12)$. Applying this function to Tables I.1 and I.2 of [20], it is found that the combined transformation $(w, x, y, z) \rightarrow (y, x, z, w)$ to (28) produces a function which is listed in Table II; namely $s = 28$. Thus after applying the combined transformation, a two-core linear-input circuit to produce this function is found to be

$$F - [w, 2 ** x, y, z, 3(x, 2y, z * w, 1)].$$

A magnetic core circuit for this function is given in Fig. 8.

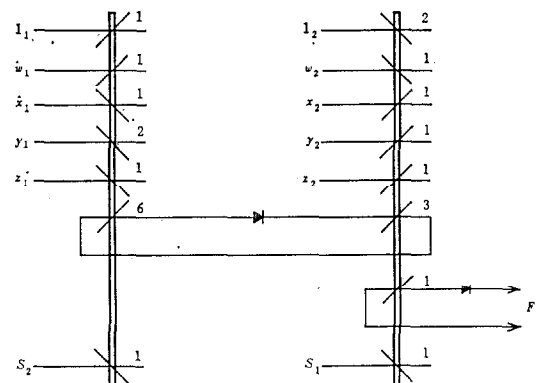


Fig. 8—Circuit for the four-variable function, $F: \sum(0, 7, 9, 12)$.

⁵ This example is given as (8) of [5], and the resulting circuit is given as Fig. 6(c). The variables have been renamed $(x_4, x_3, x_2, x_1) \rightarrow (w, x, y, z)$, and all biases have been modified by (9) of the present paper; furthermore, a presumed mistake has been corrected.

TABLE II
ALL FOUR-VARIABLE COMBINATIONAL LINEAR-INPUT SWITCHING CIRCUITS

m	s	f_s	Linear-Input Circuit	m	s	f_s	Linear-Input Circuit
1	001	0	$w, x, y, z \oplus 1$	6	061	0, 1, 2, 3, 4, 9	$3w, 2x, 2y, 2(1 \oplus x, z) \oplus z, 3$
2	002	0, 1	$w, x, y \oplus 1$	6	062	0, 1, 2, 3, 4, 11	$3w, 2x, y, z \oplus 3(x, 1 \oplus y, z), 3$
2	003	0, 3	$2w, 2x, y, 2(z \oplus y) \oplus y, 1$	6	063	0, 1, 2, 3, 4, 12	$3w, 2x, y, z \oplus 3(y, z \oplus x), 3$
2	004	0, 7	$w, x, y, z \oplus 3(w, 2 \oplus x, y, z), 1$	6	064	0, 1, 2, 3, 4, 13	$3w, 2x, y, z \oplus 4(y, 2 \oplus w, x, z), 3$
2	005	0, 15	$w, x, y, z \oplus 4(3 \oplus w, x, y, z), 1$	6	065	0, 1, 2, 3, 4, 13	$3w, 2x, y, z \oplus 5(3 \oplus w, x, y, z), 3$
3	006	0, 1, 2	$2w, 2x, y, z \oplus 2$	6	066	0, 1, 2, 3, 12, 13	$w, x \oplus 2(y, 1 \oplus w, x), 1$
3	007	0, 1, 6	$w, x, y \oplus 2(w, z, 1 \oplus x, y), 1$	6	067	0, 1, 2, 3, 12, 15	$3w, z, 5(2w, z, 1 \oplus 3x, y) \oplus 2x, y, 2$
3	008	0, 1, 14	$w, x, y \oplus 3(z, 2 \oplus w, x, y), 1$	6	068	0, 1, 2, 4, 7, 8	$w, x, y, z \oplus 2(2w, 2 \oplus x, y, z), 2$
3	009	0, 3, 5	$2w, x, y, 2(x, y \oplus z) \oplus z, 1$	6	069	0, 1, 2, 4, 7, 9	$2w, x, y, z \oplus 2(4 \oplus 2w, x, y, 3z), 2$
3	010	0, 3, 12	$x, 2z, 4(x, 2z, 1 \oplus 2w, 3y) \oplus w, 2y, 1$	6	070	0, 1, 2, 4, 7, 11	$2w, x, y, z \oplus 3(4 \oplus w, x, 2y, 2z), 2$
3	011	0, 3, 13	$2x, y, 3(3x, y \oplus 2w, z) \oplus w, z, 1$	6	071	0, 1, 2, 4, 7, 15	$2w, x, y, z \oplus 4(2 \oplus x, y, z), 2$
4	012	0, 1, 2, 3	$w, x \oplus 1$	6	072	0, 1, 2, 4, 8, 15	$w, x, y, z \oplus 3(3 \oplus w, x, y, z), 2$
4	013	0, 1, 2, 4	$2w, x, y, z \oplus 2$	6	073	0, 1, 2, 4, 9, 10	$2w, x, y, z \oplus 2(2x, 2 \oplus 2w, y, z), 2$
4	014	0, 1, 2, 5	$3w, x, 2y, z, 2(z \oplus x) \oplus 3$	6	074	0, 1, 2, 4, 9, 11	$2w, x, y, z \oplus 3(x, 1 \oplus w, z), 2$
4	015	0, 1, 2, 7	$3w, 2x, 3(x, 1 \oplus y, z) \oplus y, z, 1$	6	075	0, 1, 2, 4, 9, 14	$2w, 5(w, 3 \oplus 2x, 2y, 3z) \oplus x, y, 2z, 1$
4	016	0, 1, 2, 12	$2w, 2x, y, z \oplus 3(y, z, 1 \oplus w, x), 2$	6	076	0, 1, 2, 4, 9, 15	$2w, x, y, z \oplus 2(1 \oplus w, z), 2(3 \oplus w, x, y, z), 2$
4	017	0, 1, 2, 13	$2w, 2x, y, z \oplus 4(y, 2 \oplus w, x, z), 2$	6	077	0, 1, 2, 4, 11, 13	$x, y, z, 3(x, y, 2z \oplus 3w) \oplus w, 2$
4	018	0, 1, 2, 15	$2w, 2x, y, z \oplus 5(3 \oplus w, x, y, z), 2$	6	078	0, 1, 2, 4, 11, 15	$2w, x, y, z \oplus 4(2 \oplus w, y, z), 2$
4	019	0, 1, 6, 7	$w, 1 \oplus x, y, 2(w, x, y \oplus 1)$	6	079	0, 1, 2, 5, 6, 7	$3w, y, z, 3(y, z \oplus x) \oplus x, 2$
4	020	0, 1, 6, 10	$2w, 2x, 2y, z \oplus 3(2z, 2 \oplus w, x, 2y), 2$	6	080	0, 1, 2, 5, 6, 11	$2w, x, 3(w, 1 \oplus y, z) \oplus y, z, 1$
4	021	0, 1, 6, 11	$2w, 3x, z, 4(w, 2x, z \oplus 2y) \oplus 2y, 2$	6	081	0, 1, 2, 5, 6, 12	$3w, 2y, 2z, 4(w, y, z \oplus x) \oplus x, 3$
4	022	0, 1, 6, 14	$w, 1 \oplus x, y, 3(w, x, y \oplus 1)$	6	082	0, 1, 2, 5, 6, 13	$w, 2y, 1 \oplus 4(2w, x, 2z \oplus y, 1), x, 2z$
4	023	0, 1, 6, 15	$3x, z, 5(3x, z \oplus 2w, 2y) \oplus w, 2y, 2$	6	083	0, 1, 2, 5, 6, 15	$2y, 2z, 5(2y, 2z \oplus 3w, x) \oplus w, x, 3$
4	024	0, 1, 14, 15	$w, x, y \oplus 3(2 \oplus w, x, y), 1$	6	084	0, 1, 2, 5, 10, 12	$2w, x, 2y \oplus 3(y, z, 1 \oplus w, x), 4(x, z \oplus y), 1$
4	025	0, 3, 5, 6	$w, x, y, z \oplus 2(1 \oplus x, y, z), 1$	6	085	0, 1, 2, 5, 10, 13	$w, 2x, 2z \oplus 5(w, 2y, 1 \oplus x, 2z), y, 1$
4	026	0, 3, 5, 9	$w, x, y, 2(w, x, y \oplus z) \oplus z, 1$	6	086	0, 1, 2, 5, 10, 15	$2w, 2x, y, z \oplus 2(1 \oplus x, z), 3(z, 3 \oplus 2w, x, 2y), 2$
4	027	0, 3, 5, 10	$2x, 2y, 4(x, 2y \oplus 2w, z) \oplus w, z, 2$	6	087	0, 1, 2, 5, 11, 12	$2y, 2z, 2 \oplus 5(3w, 2x \oplus y, 2z, 1), 2w, x$
4	028	0, 3, 5, 14	$z, 2 \oplus w, x, y, 3(2w, x, y \oplus z, 1)$	6	088	0, 1, 2, 5, 11, 14	$2w, 2x, y, z \oplus 2(w, 1 \oplus x, z), 4(4 \oplus 2w, x, 2y, z), 2$
4	029	0, 3, 12, 15	$w, 2y, 4(w, 2y \oplus x, 2z) \oplus x, 2z, 1$	6	089	0, 1, 2, 5, 11, 15	$x, 2y, z, 5(2y, 2z \oplus 3w, x) \oplus 2w, 3$
4	030	0, 3, 13, 14	$2y, 3 \oplus 4(w, x, 2z \oplus 2y, 1), w, x, 2z$	6	090	0, 1, 2, 5, 14, 15	$2w, 2x, y, z \oplus 2(1 \oplus x, z), 4(2 \oplus w, x, y), 2$
5	031	0, 1, 2, 3, 4	$3w, 2x, y, z \oplus 3$	6	091	0, 1, 2, 7, 11, 12	$y, z, 1 \oplus 3(2w, 2x \oplus y, z, 1), w, x$
5	032	0, 1, 2, 3, 12	$w, x \oplus 2(y, z, 1 \oplus w, v), 1$	6	092	0, 1, 2, 7, 11, 13	$2w, 2x, y, z \oplus 4(2 \oplus w, x, y, 2z), 2$
5	033	0, 1, 2, 4, 7	$2w, x, y, z \oplus 2(w, 2 \oplus x, y, z), 2$	6	093	0, 1, 2, 7, 11, 15	$2w, 2x, y, z \oplus 5(4 \oplus w, x, 2y, 2z), 2$
5	034	0, 1, 2, 4, 8	$w, x, y, z \oplus 2$	6	094	0, 1, 2, 7, 12, 13	$3w, 2y, z, 5(2w, y, z \oplus 2x) \oplus 2x, 3$
5	035	0, 1, 2, 4, 9	$2w, x, y, z \oplus 2(x, y \oplus z), 2$	6	095	0, 1, 2, 7, 12, 15	$2w, 2x, y, z \oplus 3(2 \oplus x, y, z), 3(1 \oplus w, x), 2$
5	036	0, 1, 2, 4, 11	$2w, x, y, z \oplus 3(x, 2 \oplus w, y, z), 2$	6	096	0, 1, 2, 7, 13, 14	$2w, y, z, 4(w, y, z \oplus 2x) \oplus 2x, 2$
5	037	0, 1, 2, 4, 15	$2w, x, y, z \oplus 4(3 \oplus w, x, y, z), 2$	6	097	0, 1, 2, 7, 13, 15	$4 \oplus w, 2x, y, 2z, 5(2w, 2x, y, z \oplus 2)$
5	038	0, 1, 2, 5, 6	$2w, y, z, 2(y, z \oplus x) \oplus 2$	6	098	0, 1, 2, 12, 13, 14	$y, z, 2 \oplus 2w, 2x, 4(2w, 2x, y, z \oplus 2)$
5	039	0, 1, 2, 5, 10	$x, z \oplus 3(2w, x, 2y \oplus z, 1), y$	6	099	0, 1, 2, 12, 13, 15	$2w, 2x, y, z \oplus 5(y, 3 \oplus 2w, 2x, z), 2$
5	040	0, 1, 2, 5, 11	$2w, x, 3(w, 2 \oplus x, 2y, z) \oplus y, z, 1$	6	100	0, 1, 2, 13, 14, 15	$2w, 2x, y, z \oplus 5(4 \oplus 2w, 2x, y, z), 2$
5	041	0, 1, 2, 5, 14	$2w, 2x, y, z \oplus 4(z, 2 \oplus w, x, y), 2(w, y, 1 \oplus x, z), 2$	6	101	0, 1, 6, 7, 10, 11	$w, x, 2(w, x \oplus y) \oplus y, 1$
5	042	0, 1, 2, 7, 11	$2w, 2x, y, z \oplus 3(4 \oplus w, x, 2y, 2z), 2$	6	102	0, 1, 6, 7, 10, 12	$w, x, y \oplus 2(z, 2 \oplus w, 2x, 2y), 1$
5	043	0, 1, 2, 7, 12	$2w, y, z, 3(2w, y, z \oplus 2x) \oplus x, 2$	6	103	0, 1, 6, 7, 10, 13	$2y, 4 \oplus 5(3w, 2x, z \oplus 2y, 2), 2w, 2x, z$
5	044	0, 1, 2, 7, 13	$2w, 2x, y, z \oplus 4(4 \oplus w, 2x, y, 2z), 2$	6	104	0, 1, 6, 10, 12, 15	$2w, z, 1 \oplus 3(2x, 2y \oplus 3w, z, 1), x, y$
5	045	0, 1, 2, 7, 15	$2w, 2x, y, z \oplus 5(2 \oplus x, y, z), 2$	6	105	0, 1, 6, 10, 13, 15	$2w, 2x, 2y, z \oplus 3(2 \oplus w, x, 2y), 4(2 \oplus w, x, z), 2$
5	046	0, 1, 2, 12, 13	$2w, 2x, y, z \oplus 4(y, 1 \oplus w, x), 2$	6	106	0, 3, 5, 6, 9, 10	$2w, 2x, y, z \oplus 3(1 \oplus w, x, y, z), 1$
5	047	0, 1, 2, 12, 15	$2w, 2x, y, z \oplus 5(3 \oplus w, x, y, z), 3(y, z, 1 \oplus w, x), 2$	6	107	0, 3, 5, 10, 12, 15	$x, y, 3(x, y \oplus w, z) \oplus w, z, 1$
5	048	0, 1, 2, 13, 14	$2w, 2x, y, z \oplus 4(4 \oplus 2w, 2x, y, z), 2$	7	108	0, 1, 2, 3, 4, 5, 6	$3w, x, y, z \oplus 3$
5	049	0, 1, 2, 13, 15	$2w, 2x, y, z \oplus 5(2 \oplus w, x, z), 2$	7	109	0, 1, 2, 3, 4, 5, 8	$3w, 2x, 2y, z \oplus 4$
5	050	0, 1, 6, 7, 10	$3w, 2x, z, 4(w, x \oplus y) \oplus 2y, 2$	7	110	0, 1, 2, 3, 4, 5, 10	$2w, x, y \oplus 2(x, z \oplus y), 2$
5	051	0, 1, 6, 10, 12	$w, x, y \oplus 2(2z, 1 \oplus w, x, y), 1$	7	111	0, 1, 2, 3, 4, 5, 14	$2w, x, y \oplus 3(z, 2 \oplus w, x, y), 2$
5	052	0, 1, 6, 10, 13	$2x, z, 2 \oplus 4(3w, 2y \oplus 2x, z, 1), 2w, y$	7	112	0, 1, 2, 3, 4, 7, 8	$2w, 2x, y, z \oplus 2(w, 1 \oplus y, z), 3$
5	053	0, 1, 6, 10, 15	$2y, z, 4(3y, z \oplus 2w, 2x) \oplus w, x, 2$	7	113	0, 1, 2, 3, 4, 7, 9	$2w, x, y, 2(y, 1 \oplus x, z) \oplus z, 2$
5	054	0, 1, 6, 11, 14	$w, x, y \oplus 3(z, 1 \oplus x, y), 2(x, 2 \oplus w, y, z), 1$	7	114	0, 1, 2, 3, 4, 7, 12	$3w, 2x \oplus y, z, 5(y, z \oplus x), 1$
5	055	0, 3, 5, 6, 9	$w, x, y, z \oplus 2(3 \oplus w, 2x, 2y, 3z), 1$	7	115	0, 1, 2, 3, 4, 7, 13	$3w, 2x, y, z \oplus 4(4 \oplus w, 2x, y, 2z), 3$
5	056	0, 3, 5, 9, 14	$x, 3(z \oplus w, x, y) \oplus w, x, y, 1$	7	116	0, 1, 2, 3, 4, 8, 12	$2w, 2x, y, z \oplus 2(y, z \oplus w), 3$
5	057	0, 3, 5, 10, 12	$w, z, 3(w, z, 1 \oplus 2x, 2y) \oplus x, y, 1$	7	117	0, 1, 2, 3, 4, 8, 13	$2w, 2x, y, z \oplus 3(y, 2 \oplus w, x, z), 3$
6	058	0, 1, 2, 3, 4, 5	$2w, x, y \oplus 2$	7	118	0, 1, 2, 3, 4, 8, 15	$2w, 2x, y, z \oplus 4(3 \oplus w, x, y, z), 3$
6	059	0, 1, 2, 3, 4, 7	$3w, x, v, 2(v, 1 \oplus x, z) \oplus z, 2$	7	119	0, 1, 2, 3, 4, 9, 12	$3w, 2x, v, z \oplus 3(2y, 2 \oplus w, x, z), 3$
6	060	0, 1, 2, 3, 4, 8	$2w, 2x, y, z \oplus 3$	7	120	0, 1, 2, 3, 4, 9, 14	$3w, 2x, y, z \oplus 2(x, y, 1 \oplus w, z), 4(z, 2 \oplus w, x, y), 3$

TABLE II (Cont'd.)

m	s	f	Linear-Input Circuit	m	s	f _i	Linear-Input Circuit
121	0, 1, 2, 3, 4, 11, 12		$2w, y, z, 3(2w, y, z * 2w) ** 3$	181	0, 1, 2, 3, 4, 7, 9, 12		$2w, 2x, y, z ** 2(4 * 2w, 3x, y, z), 3$
122	0, 1, 2, 3, 4, 11, 13		$3w, 2x, y, z ** 4(4 * 2w, x, y, 2z), 3$	182	0, 1, 2, 3, 4, 7, 8, 13		$2w, 2x, y, z ** 3(4 * w, 2x, y, 2z), 3$
123	0, 1, 2, 3, 4, 12, 13		$3w, 2x, y, z ** 4(y, 1 * w, x), 3$	183	0, 1, 2, 3, 4, 7, 8, 15		$2w, 2x, y, z ** 4(2 * x, y, z), 3$
124	0, 1, 2, 3, 4, 12, 15		$3w, 2x, y, z ** 5(3 * w, x, y, z), 3(y, z, 1 * w, x), 3$	184	0, 1, 2, 3, 4, 7, 9, 10		$w, y, z ** 2(x * y, z), 1$
125	0, 1, 2, 3, 4, 13, 14		$2x, y, z, 4(2x, y, z * 3w) ** w, 3$	185	0, 1, 2, 3, 4, 7, 9, 12		$w, 2y, 2z ** 5(2w, 3x * y, 2z, 1), 2x$
126	0, 1, 2, 3, 4, 13, 15		$3w, 2x, y, z ** 5(2 * w, x, z), 3$	186	0, 1, 2, 3, 4, 7, 9, 13		$3w, x, 2y, 4(w, y, 1 * x, z) ** 2z, 3$
127	0, 1, 2, 3, 12, 13, 14		$w, x ** 2(y, z, 2 * 2w, 2x), 1$	187	0, 1, 2, 3, 4, 7, 9, 14		$2x, 2y, 1 ** 5(3w, 2z * x, 2y, 1), w, 2z$
128	0, 1, 2, 4, 7, 8, 11		$w, x, y, z ** 2(4 * w, x, 2y, 2z), 2$	188	0, 1, 2, 3, 4, 7, 12, 13		$4w, 3x, 2y, z ** 5(4 * 2w, 3x, y, z), 4$
129	0, 1, 2, 4, 7, 8, 15		$w, x, y, z ** 3(2 * x, y, z), 2$	189	0, 1, 2, 3, 4, 7, 12, 15		$w, x, 2z, 4(x, 2z * w, 2y) ** 2y, 3$
130	0, 1, 2, 4, 7, 9, 10		$x, z ** 2(2w, 3y * x, 2z, 1), y$	190	0, 1, 2, 3, 4, 7, 13, 14		$3w, 2x, y, z ** 4(3 * w, 2x, y, z), 3$
131	0, 1, 2, 4, 7, 9, 11		$2w, 3x, 2z ** 4(2y, 1 * w, 2x, 3z), y, 1$	191	0, 1, 2, 3, 4, 8, 12, 15		$w, x, 2y, 2z ** 2(w, x, 1 * y, z), 4(3 * w, x, y, z), 3$
132	0, 1, 2, 4, 7, 9, 14		$2w, 3y, 2z ** 4(2x, 1 * w, 3y, 2z), x, 1$	192	0, 1, 2, 3, 4, 8, 13, 14		$2w, 2x, y, z ** 3(4 * 2w, 2x, y, z), 3$
133	0, 1, 2, 4, 7, 9, 15		$w, 2y, z, 1 ** 4(w, 3x * 2y, 2z, 1), 2x$	193	0, 1, 2, 3, 4, 8, 13, 15		$2w, 2x, y, z ** 4(2 * w, x, z), 3$
134	0, 1, 2, 4, 7, 11, 13		$3w, 3 ** 4(x, y, z * w, 2), x, y, 2z$	194	0, 1, 2, 3, 4, 9, 12, 13		$3w, 2x, y, z ** 4(2y, 2 * 2w, x, z), 3$
135	0, 1, 2, 4, 7, 11, 15		$2w, x, y, z ** 4(4 * w, x, 2y, 2z), 2$	195	0, 1, 2, 3, 4, 9, 12, 14		$y, 2z, 1 ** 5(2w, 3x, y * w, 2), w, 2x$
136	0, 1, 2, 4, 9, 10, 11		$2w, x, y, z ** 3(2x, 2 * 2w, y, z), 2$	196	0, 1, 2, 3, 4, 9, 12, 15		$3w, 2x, y, z ** 2(1 * w, z), 3(z, 4 * 3w, 2x, y), 3$
137	0, 1, 2, 4, 9, 10, 12		$x, y, z, 2(x, y, z * w) ** 2$	197	0, 1, 2, 3, 4, 9, 14, 15		$3w, 2x, y, z ** 2(1 * w, z), 4(2 * w, x, y), 3$
138	0, 1, 2, 4, 9, 10, 13		$2x, 3y, 2z, 5(y, z * w) ** w, 4$	198	0, 1, 2, 3, 4, 11, 12, 13		$2y, z, 1 ** 5(2w, 3x * y, z, 1), w, 2x$
139	0, 1, 2, 4, 9, 10, 15		$2w, x, y, z ** 2(2 * 2w, y, z), 2(3 * w, x, y, z), 1$	199	0, 1, 2, 3, 4, 11, 12, 15		$3w, 2x, y, z ** 3(2 * w, y, z), 3(1 * w, x), 3$
140	0, 1, 2, 4, 9, 11, 13		$2w, x, y, z ** 3(1 * w, z), 2$	200	0, 1, 2, 3, 4, 11, 13, 14		$3w, 2x, y, z ** 4(3 * 2w, x, y, z), 3$
141	0, 1, 2, 4, 9, 11, 14		$2x, y, 2z, 4(x, y, 2z * 2w) ** w, 3$	201	0, 1, 2, 3, 4, 12, 13, 14		$3w, 2x, y, z ** 4(y, z, 1 * w, 2x), 3$
142	0, 1, 2, 4, 9, 11, 15		$2w, x, y, z ** 4(x, y, z * 2w, y, 2z), 2$	202	0, 1, 2, 3, 4, 12, 13, 15		$3w, 2x, y, z ** 5(4 * 2w, 2x, y, z), 3$
143	0, 1, 2, 4, 9, 14, 15		$2w, x, y, z ** 2(1 * w, z), 3(2 * w, x, y), 2$	203	0, 1, 2, 3, 4, 13, 14, 15		$w, x ** 2(1 * w, x), 1$
144	0, 1, 2, 4, 11, 13, 14		$2w, x, y, z ** 3(3 * 2w, x, y, z), 2$	204	0, 1, 2, 3, 12, 13, 14, 15		$w, x, y, z ** 2(3 * w, x, y, 2z), 2$
145	0, 1, 2, 4, 11, 13, 15		$2w, x, y, z ** 4(4 * 2w, x, y, 2z), 2$	205	0, 1, 2, 4, 7, 8, 11, 13		$w, x, y, z ** 3(4 * w, x, 2y, 2z), 2$
146	0, 1, 2, 5, 6, 7, 11		$2w, x, 3(w, x, 1 * y, z) ** y, z, 1$	206	0, 1, 2, 4, 7, 8, 11, 15		$2w, 5(w, 3 * 3x, 2y, 2z) ** x, 2y, 2z, 1$
147	0, 1, 2, 5, 6, 11, 12		$x, 2 ** 3(3w, 2y, 2z * x, 3), w, y, z$	207	0, 1, 2, 4, 7, 9, 10, 11		$x, y ** 2(w, 2z * x, y, 1), z$
148	0, 1, 2, 5, 6, 11, 13		$2w, 2x, y, z ** 2(2w, 2 * 2x, y, z), 4(4 * 2w, x, y, 2z), 2$	208	0, 1, 2, 4, 7, 9, 10, 12		$2x, 2z, 1 ** 5(2w, 3y * x, 2z, 1), w, 2y$
149	0, 1, 2, 5, 6, 11, 15		$3w, x, 5(w, 1 * y, z) ** 2y, 2z, 1$	209	0, 1, 2, 4, 7, 9, 10, 13		$w, 2y, 2z, 1 ** 5(w, 3x * 2y, 2z, 1), 2x$
150	0, 1, 2, 5, 6, 12, 13		$2w, 2x, y, z ** 2(w, z, 1 * x, y), 4(2y, 2 * w, 2x, z), 2$	210	0, 1, 2, 4, 7, 9, 10, 15		$2w, x, y, z ** 3(4 * 2w, x, y, 3z), 2$
151	0, 1, 2, 5, 6, 12, 15		$2y, 2z, 1 ** 4(3w, x * 2y, 2z, 1), w, x$	211	0, 1, 2, 4, 7, 9, 11, 13		$2x, y, 1 ** 4(2w, 2z * x, y, 1), w, 2z$
152	0, 1, 2, 5, 6, 13, 14		$w, 2y, 2z, 4(2y, 2z * w, x) ** x, 3$	212	0, 1, 2, 4, 7, 9, 11, 14		$2w, x, y, z ** 4(x, 3 * 2w, y, 2z), 2(w, 2 * x, y, z), 2$
153	0, 1, 2, 5, 6, 13, 15		$2w, 2x, y, z ** 2(w, z, 1 * x, y), 5(y, 3 * w, 2x, 2z), 2$	213	0, 1, 2, 4, 7, 9, 11, 15		$2w, x, y, z ** 3(4 * w, 2x, 2y, z), 2(1 * w, z), 2$
154	0, 1, 2, 5, 10, 12, 15		$2w, 2x, y, z ** 2(1 * x, z), 3(z, 2 * 2w, x, y), 2$	214	0, 1, 2, 4, 7, 9, 14, 15		$2w, x, y, z ** 3(2 * w, x, y, z), 2$
155	0, 1, 2, 5, 10, 13, 14		$2w, 2x, y, z ** 4(y, 1 * x, z), 4(z, 1 * w, y), 2$	215	0, 1, 2, 4, 7, 11, 13, 14		$3w, 4(2w, 1 * x, y, z) ** x, y, 2z, 1$
156	0, 1, 2, 5, 11, 12, 14		$2w, 2x, y, z ** 3(2y, 2 * w, 2x, z), 4(4 * 2w, x, 2y, z), 2$	216	0, 1, 2, 4, 7, 11, 13, 15		$3x, 2y, 2z, 5(x, y, z * w) ** w, 4$
157	0, 1, 2, 5, 11, 14, 15		$2w, 2x, y, z ** 2(w, y, 1 * x, z), 5(4 * 2w, x, 2y, z), 2$	217	0, 1, 2, 4, 9, 10, 11, 12		$2x, 2y, z, 4(y, z * w) ** w, 3$
158	0, 1, 2, 7, 11, 12, 13		$2y, z, 2 ** 5(2w, 2x * y, z, 1), 2w, 2x$	218	0, 1, 2, 4, 9, 10, 11, 13		$2w, x, y, z ** 4(3 * 2w, y, z), 2(2w, 2 * 2x, y, z), 2$
159	0, 1, 2, 7, 11, 12, 15		$2w, 2x, y, z ** 3(4 * w, x, 2y, 2z), 3(1 * w, x), 2$	219	0, 1, 2, 4, 9, 10, 11, 15		$2w, 4y, 4z ** 7(3x, 1 * w, y, 2z), 3x, 1$
160	0, 1, 2, 7, 11, 13, 14		$2w, 2x, y, z ** 4(2 * w, x, y, z), 2$	220	0, 1, 2, 4, 9, 10, 12, 15		$x, 2y, 2z, 4(y, z * w) ** w, 3$
161	0, 1, 2, 7, 12, 13, 14		$2w, 2x, y, z ** 4(4 * 2w, 3x, y, z), 2$	221	0, 1, 2, 4, 9, 10, 13, 14		$2w, x, y, z ** 2(1 * w, y), 4(y, 3 * 2w, x, 2z), 2$
162	0, 1, 6, 7, 10, 11, 12		$w, x, y ** 2(3 * 2w, 2x, 3y), 1$	222	0, 1, 2, 4, 9, 11, 13, 14		$2w, x, y, z ** 3(4 * 3w, x, y, 2z), 2$
163	0, 3, 5, 6, 9, 10, 12		$w, x, y, z ** 2(1 * w, x, y, z), 1$	223	0, 1, 2, 4, 9, 11, 14, 15		$2w, x, y, z ** 3(x, 1 * w, z), 4(2 * w, x, y), 2$
164	0, 1, 2, 3, 4, 5, 6, 7		$w ** 1$	224	0, 1, 2, 4, 9, 11, 14, 15		$2w, x, y, z ** 4(3 * 2w, x, y, z), 2$
165	0, 1, 2, 3, 4, 5, 6, 8		$2w, x, y, z ** 3$	225	0, 1, 2, 4, 11, 13, 14, 15		$2w, 2x, y, z ** 3(2 * w, 2x, y, z), 2$
166	0, 1, 2, 3, 4, 5, 6, 9		$3w, x, y, 2(2 * x, y, z) ** z, 3$	226	0, 1, 2, 5, 6, 7, 11, 12		$2w, 2x, y, z, 2(4 * w, 2x, 2y, z) ** 4(2 * w, 2x, y, z), 2$
167	0, 1, 2, 3, 4, 5, 6, 11		$3w, x, 3(2 * x, y, z) ** y, z, 2$	227	0, 1, 2, 5, 6, 11, 12, 13		$x, 4 ** 5(3w, 2y, 2z * x, 3), 2w, 2y, 2z$
168	0, 1, 2, 3, 4, 5, 6, 15		$3w, 4(w, 2 * x, y, z) ** x, y, z, 1$	228	0, 1, 2, 5, 6, 11, 12, 15		$2w, 2x, y, z ** 4(3 * 2w, x, y, z), 2(2w, 2 * 2x, y, z), 2$
169	0, 1, 2, 3, 4, 5, 8, 9		$w, x, y ** 2$	229	0, 1, 2, 5, 6, 11, 13, 14		$2w, 2x, y, z ** 5(4 * 2w, x, y, 2z), 2(2w, 2 * 2x, y, z), 2$
170	0, 1, 2, 3, 4, 5, 8, 10		$2w, 3x, y, 2z ** 2(w, y * x), 4$	230	0, 1, 2, 5, 6, 12, 13, 14		$2w, 2x, y, z ** 4(y, z, 2 * 2w, 2x), 2(2w, 2 * 2x, y, z), 2$
171	0, 1, 2, 3, 4, 5, 8, 11		$2w, 3x, 2y, 3(y, 1 * w, z) ** z, 4$	231	0, 1, 2, 5, 6, 12, 13, 14		$2w, 2x, y, z ** 2(2 * 2x, y, z), 3(4 * 2w, 2x, y, z), 2$
172	0, 1, 2, 3, 4, 5, 8, 14		$3w, 2x, 2y, z ** 4(z, 2 * w, x, y), 4$	232	0, 1, 2, 5, 6, 13, 14, 15		$2w, 2x, y, z ** 4(y, 3 * w, 2x, 2z), 4(z, 3 * 2w, x, 2y), 2$
173	0, 1, 2, 3, 4, 5, 8, 15		$3w, 2x, 2y, z ** 5(3 * w, x, y, z), 4$	233	0, 1, 2, 5, 10, 13, 14, 15		$2w, 2x, y, z ** 5(4 * 2w, x, 2y, z), 3(2y, 2 * w, 2x, z), 2$
174	0, 1, 2, 3, 4, 5, 10, 11		$w, 2x ** y, 2(w, y * x), 1$	234	0, 1, 2, 5, 11, 12, 14, 15		$2w, 2x, y, z ** 4(3 * 2w, 2x, y, z), 2$
175	0, 1, 2, 3, 4, 5, 10, 12		$2w, x, y ** 2(2z, 2 * 2w, x, y), 2$	235	0, 1, 2, 7, 11, 12, 13, 14		$w, x, y ** 2(1 * w, x, y), 1$
176	0, 1, 2, 3, 4, 5, 10, 13		$w, 2x, 3y, z, 4(x, 2y, z * 2w) ** 5$	236	0, 1, 6, 7, 10, 11, 12, 13		$2(2 * w, x, y, z), 2(2 * w, x, y, z) ** w, x, y, z, 1$
177	0, 1, 2, 3, 4, 5, 10, 14		$2w, z, 3(w, 1 * x, y) ** y, 2$	237	0, 3, 5, 6, 9, 10, 12, 15		
178	0, 1, 2, 3, 4, 5, 10, 15		$4w, 2x, 5(2w, x, 1 * 3y, z) ** 3y, z, 3$				
179	0, 1, 2, 3, 4, 5, 14, 15		$2w, 3(w, 1 * x, y) ** x, y, 1$				
180	0, 1, 2, 3, 4, 7, 8, 11		$2w, 2x ** y, z, 2(w, x, y, z * 2), 1$				

CONCLUSIONS

On the assumption that there are no restrictions on the total number of input turns to a magnetic core, linear-input circuits for the production of n variable symmetric functions have been developed which require no more than $1 + [(n+1)/2]$ cores. Logical design techniques for arbitrary switching functions have been developed which are based on primitive circuits. These are easy to use but they do not necessarily lead to the most economical circuits. A synthesis procedure based on linear programming plus inspection has been developed which leads to very economical circuits; but at present it is difficult to use. A table of all linear-input circuits of four variables verifies by exhaustion that the known bound on the number of cores required for n variable symmetric functions is met for any switching function for which $n \leq 4$.

ACKNOWLEDGMENT

In addition to E. L. Glaser and E. F. Moore, whose contributions have been mentioned, the author is greatly indebted to D. C. Condon, G. C. Arndt, Jr. and K. M. Beilstein. This paper is based on work done while the author was in the employ of the Burroughs Corporation, ElectroData Division, Pasadena, Calif.; and acknowledgment is made for permission to publish this work. Facilities and editorial assistance for the preparation of this paper were supplied by Stanford Research Institute, Menlo Park, Calif.

BIBLIOGRAPHY

- [1] M. Karnaugh, "Pulse-switching circuits using magnetic cores," *Proc. IRE*, vol. 43, pp. 570-584; May, 1955.
- [2] R. McNaughton, "Unate truth functions," this issue, p. 1.
- [3] "Project Lightning, Second Phase, First Quarterly Progress Report," Remington Rand UNIVAC, St. Paul, Minn., p. 57; February, 1959.
- [4] R. L. Wiegington, "A new concept in computing," *Proc. IRE*, vol. 47, pp. 516-523; April, 1959.
- [5] S. Muroga, "Logical Elements on Majority Decision Principle and Complexity of Their Circuit," presented at Internatl. Conf. on Information Processing, Paris, France; June 15-20, 1959.
- [6] R. C. Minnick, "Synthesis of Linear-Input Logic by the Simplex Method," presented at the Sixth Annual Symp. on Computers and Data Processing of the Denver Res. Inst., Denver, Colo.; July 30, 1959.
- [7] R. Lindaman, "A new concept in computing," *Proc. IRE*, vol. 48, p. 257; February, 1960.
- [8] M. C. Paull and E. J. McCluskey, Jr., "Boolean functions realizable with single threshold devices," *Proc. IRE*, vol. 48, pp. 1335-1337; July, 1960.
- [9] E. N. Gilbert, "Lattice theoretic properties of frontal switching functions," *J. Math. Phys.*, vol. 33, pp. 57-67; April, 1954.
- [10] W. L. Semon, "Notes on the theory of series-parallel circuits," in "Theory of Switching," Harvard Computation Lab., Bell Labs. Rept. 10, Sect. V; March, 1955.
- [11] W. L. Semon and R. Gould, "A chart of unage functions," in "Theory of Switching," Harvard Computation Lab., Bell Labs. Rept. 13, Sect. III; November, 1955.
- [12] E. F. Moore, "Counter-Example to a Conjecture of McCluskey and Paull," 1957 (unpublished).
- [13] R. L. Mattson, "A self-organizing binary system," *Proc. EJCC*, pp. 212-220; December, 1959.
- [14] D. G. Willis, "Plastic neurons as memory elements," 1959 IRE WESCON CONVENTION RECORD, pt. 4, pp. 55-65.
- [15] W. D. Rowe, "The transistor NOR circuit," 1957 IRE WESCON CONVENTION RECORD, pt. 4, pp. 231-245.
- [16] E. Goto, "The parametron, a digital computer element which utilizes parametric oscillation," *Proc. IRE*, vol. 47, pp. 1304-1316; August, 1959.
- [17] E. Goto, K. Murata, *et al.*, "Esaki diode high-speed logic circuits," *IRE TRANS. ON ELECTRONIC COMPUTERS*, vol. EC-9, pp. 25-29; March, 1960.
- [18] R. Tevonian, "Dry reed logic and memory unit," *Western Electric Engr.*, pp. 43-47; April, 1960.
- [19] S. H. Caldwell, "Switching Circuits and Logical Design," John Wiley and Sons, Inc., New York, N. Y.; 1958.
- [20] Staff of the Harvard Computation Laboratory, "Synthesis of electronic computing and control circuits," in "Annals of the Harvard Computation Laboratory," Harvard University Press, Cambridge, Mass., vol. 27; 1951.
- [21] S. Vajda, "The Theory of Games and Linear Programming," Methuen and Co., Ltd., London, Eng., Monograph 4060/u; 1956.
- [22] R. A. Higonnet and R. A. Grea, "Logical Design of Electrical Circuits," McGraw-Hill Book Co., Inc., New York, N. Y.; 1958. The table was produced by E. F. Moore.
- [23] J. L. Rosenfield, "Magnetic core pulse-switching circuits for standard packages," *IRE TRANS. ON ELECTRONIC COMPUTERS*, vol. EC-7, pp. 223-228; September, 1958.
- [24] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," *Bull. Amer. Math. Soc.*, vol. 64, pp. 275-278; September, 1958.
- [25] M. H. Lewin, "Negative-resistance elements as digital computer components," *Proc. EJCC*, pp. 15-27; December 1-3, 1959.