B. Dunham
D. Middleton
J. H. North
J. A. Sliter
J. W. Weltzien

The Multipurpose Bias Device*
Part II
The Efficiency of Logical Elements

Abstract: The efficiency of a logical element can be equated with the set of subfunctions it realizes upon biasing or duplication of inputs. Various classes of elements are considered, and optimum or near-optimum examples are presented. Some related areas of study are suggested.

A test of logical efficiency

Part I of this paper dealt with the *Rutz commutator* as a typical multipurpose element.[1] Such elements should provide economy in both number and assortment of logical building blocks. A way of evaluating them, however, is needed. Suppose we find, for two comparable devices, that fewer units of A are generally required than of B in performing logical operations. We judge A more efficient, but seek an easy test of this supremacy. The class of subfunctions a multipurpose device realizes upon biasing or duplication of inputs is easily determined. The larger the class, the more versatile we expect the device to be, since it can be used in a greater variety of ways. Versatility, in turn, should lead to efficiency, since given logical operations can generally be performed by fewer units of a more versatile device. Here, then, is the basis of the test we need.

The technique of *biasing* inputs was fully explained in Part I of this paper.[2] If extra *signal loads*[3] are permitted, subfunctions can also be obtained by *duplicating* inputs. Consider the 3-variable function indicated in the following table:

Row	p	q	r	*Output*
1	1	1	1	1
2	0	1	1	1
3	1	0	1	0
4	0	0	1	1
5	1	1	0	0
6	0	1	0	0
7	1	0	0	1
8	0	0	0	0

The input states are shown (as in Part I) by the three columns under 'p','q', and 'r'. Biasing of individual inputs yields the 2-variable functions EXCLUSIVE-OR, IF-AND-ONLY-IF, IF-THEN, and NOT-IF-THEN. Suppose now we duplicate the inputs indicated by 'p' and 'r', that is, we introduce the same signal at both locations. We can determine the effect of this by looking at those rows of the table in which 'p' and 'r' are assigned like values, namely, rows 1, 3, 6, and 8. The function AND is obtained. If the inputs denoted by 'q' and 'r' are duplicated (rows 1, 2, 7, and 8), the function INCLUSIVE-OR is generated.

A convenient way of labeling functions is required. If values are assigned input variables in the familiar truth-table manner,[4] the table of inputs need not be shown. Every function of n variables is uniquely designated by a 2^n bit (binary) number representing an output. By this

*Portions of this paper were presented to the International Symposium on the Theory of Switching at Harvard University, April 4, 1957, and as a lecture at Oxford University on May 5, 1958.

scheme, for example, 1 0 0 1 0 1 1 0 and 1 1 1 0 1 0 0 0 would denote the sum and carry, respectively, of a full adder.

Further refinement is possible. Four bits can easily be correlated with a single octal digit, primed or unprimed. The three leftmost bits determine which octal digit is selected. The rightmost bit causes the prime to be added or not. The sixteen possible cases are shown below:

Binary	*Primed octal*	*Binary*	*Primed octal*
0 0 0 0	0	1 0 0 0	4
0 0 0 1	0′	1 0 0 1	4′
0 0 1 0	1	1 0 1 0	5
0 0 1 1	1′	1 0 1 1	5′
0 1 0 0	2	1 1 0 0	6
0 1 0 1	2′	1 1 0 1	6′
0 1 1 0	3	1 1 1 0	7
0 1 1 1	3′	1 1 1 1	7′

The primed octal numbers are thus abbreviations for binary numbers which, in turn, designate logical functions. Octal labels will be bracketed to avoid confusion. Some of the more familiar functions are as follows:

Function	*Label*
AND	[4]
INCLUSIVE-OR	[7]
EXCLUSIVE-OR	[3]
IF-AND-ONLY-IF	[4′]
NEITHER-NOR	[0′]
NOT-BOTH	[3′]
SUM (FULL ADDER)	[4′ 3]
CARRY (FULL ADDER)	[7 4]

In Part I, functions like [5′] and [6′], equivalent under permutation of input variables, were grouped into *interchange classes.*[5] For ease of reference, such classes will often be represented by the *precursor,* the member with the smallest label numerically.

We now return to the scheme of evaluation mentioned above. Assume three kinds of three-input, one-output device: *F, G,* and *H* (see Table 1). We fabricate from each (without time or feedback), those of the 68 full 3-variable precursors which require no more than two units. Table 1 is the box score of our effort. Device *H* is more efficient than *G,* which in turn surpasses *F.*

We now apply the direct test suggested earlier. *Only subfunctions that are full (that is, without vacuous variables) are counted and but one member from each interchange class.* In terms of 2-variable subfunctions through biasing alone, *F* has a rating of *2; G, 4;* and *H, 4.* When duplication is also considered, *F* and *G* have the same rating as before; but *H* scores *6.*

One further comment is pertinent. Because *F* (the carry of a full adder) is commutative, the results obtained from biasing its different inputs are identical. Such is not the case, however, for *H.* The latter's non-commutativity makes the choice of input for biasing significant. As a result, a greater number of subfunctions is produced. Commutative devices of a higher order should prove even less satisfactory. Such elements achieve only commutative subfunctions upon biasing: yet as shown in Table 2, the proportion of commutative functions varies inversely with the number of variables.

The test in operation

If we count subfunctions of two variables, *4* is the highest rating a 3-variable function obtains from biasing only, and *7,* two such functions jointly. These "high-scoring" functions, shown in Table 3, define superior three-input devices. Only the relevant precursors are indicated.

On duplication of inputs, the scores of [1 6′] and [4′ 5] are increased to *6;* the others remain *4.* In fact, *6* is the highest score possible, since any given choice of the two extreme "end bits" will eliminate two of the eight possible subfunctions. A choice of 1 . . . 1 prevents [1] and [3]; 0 . . . 1, [4] and [7]; 1 . . . 0, [0′] and [3′]; 0 . . . 0, [4′] and [5′]. Hence we regard these two functions as optimum for the three-input, one-output case. The two functions are related, however, since the denial of [1 6′] is interchange equivalent to [4′ 5]. Obviously, a function and its denial will have identical scores; also, a function and its "mirror image." The mirror image is the function obtained when the order of bits is reversed—

Table 1 **Full 3-variable precursors which can be obtained in each case from two or less units.**[6]

Device	*Output*	*Precursors obtained*
F	[7 4]	5
G	[5 6]	26
H	[1 6′]	44

Table 2 **Number of commutative and non-commutative functions.**[7]

Full functions	*Commutative*	*Non-commutative*
2-variable	6	2
3-variable	14	54
4-variable	30	3874

Table 3 **High-scoring three-input logical elements.**

3-variable function	*2-variable functions obtained*			
[0′ 5′]	[0′]	[1]	[3′]	[5′]
[1 4′]	[0′]	[1]	[3]	[4′]
[1 5′]	[0′]	[1]	[3′]	[5′]
[1 6′]	[1]	[3]	[4′]	[5′]
[3 5′]	[3]	[3′]	[4′]	[5′]
[4 3]	[1]	[3]	[4]	[4′]
[4 7]	[1]	[4]	[5′]	[7]
[4′ 5]	[1]	[3]	[4′]	[5′]
[4′ 7]	[3]	[4′]	[5′]	[7]
[5 6]	[1]	[4]	[5′]	[7]

in other words, the most significant bit is replaced by the least significant bit, and so on. The function 1 1 0 1 1 0 1 0, for example, is the mirror image of 0 1 0 1 1 0 1 1. In the particular case, the mirror image of [1 6′] is interchange equivalent to the denial of [1 6′]; but this is not always so. Hence, given a score for one interchange class of functions, we know the score for other related interchange classes: that is, denials, mirror images, and denials of mirror images. This collection of at most four interchange classes we shall call a *cycle*. The precursor with the smallest label numerically for a given cycle is the *chief precursor*. The function [1 6′] is thus chief precursor for the one cycle of three variables which has a *first* score (biasing only) of *4* and a *second* score (biasing plus duplication) of *6*.

As noted above, *7* is the highest first score obtained by a three-input, two-output device. Several such devices also have a maximal second score of *8*. It should be emphasized, however, that the immediate count of subfunctions is a less telling measure of efficiency when multi-output elements are considered. Individual high-scoring functions are helpful, but they must be aptly suited to one another—they must "team" well together. A good estimate can be made, however, as to how well suited different elements are by *team scoring*. The latter technique, which, as we shall see, is ultimately based upon the method of counting subfunctions, can also be used as an added test in one-output cases. Suppose, for example, we must choose between the high-scoring one-output functions [4 7] and [5 6]. *Both* have first and second scores of *4*. To resolve this difficulty, we examine teams of two, —that is, *all possible hookups (non-feedback) involving two or less units are tested and a single team score produced.* From [47] (by second scoring), *17* 4-variable and *19* 3-variable functions are obtained; from [56], *24* 4-variable and *26* 3-variable functions. Hence we judge [5 6] the more powerful device.

When all 80 3-variable precursors are examined by team-of-two scoring, [1 6′] and [4′ 5] (the only two precursors scoring *6* in 2-variable subfunctions) score far better than any other precursor. [1 6′] has a team-of-two second score of *44* 3-variable subfunctions and [4′ 5], *41*. No other precursor has a larger score than *32*. It is of note, however, that [1 6′] and [4′ 5], interchange denials of one another, team score differently. There is apparent advantage in a 0 . . . 1 as against a 1 . . . 0 end-bit situation; and, indeed, we see by reflection that this is the case. With 1 . . . 0 end-bits, all two-unit hookups of a given device are equivalent when the five variable inputs are alike, that is, all *on* or all *off*. Such is not the case, however, when the end bits are 0 . . . 1. As it turns out, every full non-commutative 3-variable precursor with 0 . . . 1 end bits team scores higher than its interchange denial.

Three-input, two-output elements may best be examined by team scoring. Again, we count only 3-variable subfunctions obtained from biasing plus duplication. As it turns out, the device with outputs [0′ 5] and [4′ 5′] has a highest team-of-two score, although it individually generates but *6* 2-variable subfunctions. Of the eighty precursors of three variables, *79* can be obtained with two or less units. Table 4 indicates the appropriate hookups.

A binary representation ot this device with its table of inputs is revealing:

Row	*Inputs*			*Outputs*	
	p	*q*	*r*	[0′ 5]	[4′ 5′]
1	1	1	1	0	1
2	0	1	1	0	0
3	1	0	1	0	0
4	0	0	1	1	1
5	1	1	0	1	1
6	0	1	0	0	0
7	1	0	0	1	1
8	0	0	0	0	1

Both [0′ 5] and [4′ 5′] are members of the same cycle. They differ only in their two extreme "end bits." This difference is fundamental, however, since functions which are interchange equivalent always agree in this respect. If the end bits were not thus opposite, a sizable part of the 80 3-variable precursors would probably be missed, since the latter are subdivided into four equal classes by end bits. It can be seen also that, if the two functions must differ in both end bits, there is advantage in the fact that the first function has two ZEROs and the second two ONEs. In this way, [0′ 5] has a total of three ONEs and [4′ 5′] five, instead of both having four. This lack of balance is preferable, since an output with fewer ONEs is more likely to produce functions with fewer than average ONEs; and an output with more ONEs, those with more than average ONEs. A check of Table 4 shows that the great majority of precursors with fewer than four ONEs are derived via [0′ 5] and of those with more than four ONEs, via [4′ 5′]. The remaining precursors are more or less evenly split between the two outputs. By this analysis, we see why none of the high-scoring individual functions shown in Table 3 was used. None of these has like end bits. There remains the question as to why ONEs and ZEROs are so arranged in the six inner rows of the table. Clearly, for each function they must be three in number. Since non-commutativity is desired, the ONEs must be parceled among the input rows with two ONEs (rows 2, 3, and 5) and one ONE (rows 4, 6, and 7). They must also be uniform with respect to no input variable. For example, ONEs in rows 2, 4, and 6 would coincide with assignment of ZERO to '*p*'. But why should the two functions exactly agree in rows 2 to 7? Undue orientation generally restricts the variety of operation; but it must be remembered the critical end bits provide an opposite orientation. Hence, the like assignment of rows 2 to 7 compensates for this.

It is of some note that the two-input, two-output device which seems most versatile is quite similar to the three-input device just described. By team-of-two scoring, an element with outputs [1] and [5′] has a highest rating of *8* 2-variable and *8* 3-variable functions. The reader

Table 4 **Generation of eighty 3-variable precursors from three-input, two-output element.**

	Precursor	*Appropriate hookup*
1.	[0 0]	*A* 0 0 0
2.	[0 0′]	*A r q p A p B*
3.	[0 1]	*A r q p A p r*
4.	[0 1′]	*A q r* 1
5.	[0 3]	*A q q p A B r*
6.	[0 3′]	*B r q p A p B*
7.	[0 4]	*A q p* 1 *A r B*
8.	[0 4′]	*A q p p A r B*
9.	[0 5]	*A p* 1 *r*
10.	[0 5′]	*A q r p B A q*
11.	[0 7]	*A q p* 1 *A r* 1
12.	[0 7′]	*A* 1 1 *r*
13.	[0′ 3]	*A r q p A p q*
14.	[0′ 3′]	*B r q p A p q*
15.	[0′ 4]	*A r q q A B p*
16.	[0′ 4′]	*A* 1 *r p A B q*
17.	[0′ 5]	*A p q r*
18.	[0′ 5′]	*A p q* 1 *A r p*
19.	[0′ 7]	*A r q p A B q*
20.	[0′ 7′]	*A r q p B A q*
21.	[1 4]	*A r q r A B p*
22.	[1 4′]	*A q r* 1 *A B p*
23.	[1 5]	*A p q r A q r*
24.	[1 5′]	*B p q r A q r*
25.	[1 6]	*A r q p B A* 1
26.	[1 6′]	*A q p p B A r*
27.	[1 7]	*A p q q A r q*
28.	[1 7′]	*B q p r A q B*
29.	[1′ 6]	*A q* 0 *r*
30.	[1′ 6′]	*B* 0 *q p B r A*
31.	[1′ 7]	*B q p* 1 *r A q*
32.	[1′ 7′]	*B q* 0 *r*
33.	[3 4]	*A q p* 1 *r A B*
34.	[3 4′]	*A p q p B A r*
35.	[3 5]	*A p q r A B r*
36.	[3 5′]	*A p q r B A r*
37.	[3 7]	*B* 1 *r q p A B*
38.	[3 7′]	*B q p p B A r*
39.	[3′ 7]	*B r q p B A* 1
40.	[3′ 7′]	*B r q p B A p*
41.	[4 0]	*A q* 0 *r* 0 *B p*
42.	[4 0′]	*A r q p B p A*
43.	[4 1]	*A q q r p B A*
44.	[4 1′]	*A p* 0 *q r q B*
45.	[4 3]	*A q* 0 *p A B r*
46.	[4 3′]	*A q* 0 *p B A r*
47.	[4 4]	*A q* 0 1 0 *A p*
48.	[4 4′]	*A* 0 *q r p A B*
49.	[4 5]	*A r q q p B A*
50.	[4 5′]	*B p q q B q r*
51.	[4 7]	*B r q p p B r*
52.	[4 7′]	*B q* 0 *p B* 0 *r*
53.	[4′ 3]	*A q q p A* 0 *r*
54.	[4′ 3′]	*B r q p B p q*
55.	[4′ 4]	*A* 0 *q r A B p*
56.	[4′ 4′]	*B p q* 1
57.	[4′ 5]	*A r q q A* 0 *p*
58.	[4′ 5′]	*B p q r*
59.	[4′ 7]	*B q p* 1 *B r A*
60.	[4′ 7′]	*B p q p B r A*
61.	[5 4]	*A r q p p B A*
62.	[5 4′]	*B p q r B q A*
63.	[5 5]	*B p* 1 1
64.	[5 5′]	*B p q r B q r*
65.	[5 6]	*B p q r q B r*
66.	[5 6′]	*B p* 1 *q B r A*
67.	[5 7]	*B r q q p B A*
68.	[5 7′]	*B p r* 0
69.	[5′ 6]	*B* 1 *p q B r A*
70.	[5′ 6′]	*B p q q B r A*
71.	[5′ 7]	*B r q r p B A*
72.	[5′ 7′]	*B q p p A* 0 *r*
73.	[7 4]	(three units required)
74.	[7 4′]	*B q* 0 *p r A B*
75.	[7 5]	*B r p q p B A*
76.	[7 5′]	*B q q r B p A*
77.	[7 7]	*A p q* 1 1 *B A*
78.	[7 7′]	*B q p* 0 *B A r*
79.	[7′ 7]	*B q p* 1 *r A* 0
80.	[7′ 7′]	*B* 0 0 0

can readily observe, from the following table, the various properties in common with the three-input device.

Row	*Inputs*		*Outputs*	
	p	*q*	[1]	[5′]
1	1	1	0	1
2	0	1	0	0
3	1	0	1	1
4	0	0	0	1

The notation of Table 4 requires some explanation. The letter '*A*' is used to denote the output obtained through [0′ 5]; '*B*', through [4′ 5′]. The first '*A*' or '*B*' designates which function serves as the final output. The indicated hookup requires one or two units insofar as three or six characters follow the initial '*A*' or '*B*'. The first three of these characters indicate the inputs to the first unit; and the next three, to the second. ONEs and ZEROs indicate positive and negative biases, respectively. '*p*', '*q*', and '*r*' are used in the familiar manner to designate variable inputs. Outputs from the first unit may serve as inputs to the second. The precursors are arranged in numerical order, as against the alphabetic ordering scheme used throughout Part I.

With functions of more than three variables, a computer is generally required, and the IBM 704 Data Processing Machine has been used. The 65,536 functions of four variables break down into 3984 interchange classes. Of the latter, 436 have a maximal first score of *8* 3-variable subfunctions. When we differentiate among the 436 by second scores, eight have a maximal rating of *14*. These eight compose two cycles which have the following chief precursors:

[0′ 5 7 1′] [1 5′ 6 4′]

These represent the most versatile four-input, one-output devices according to the present criteria. Part III will include a study of 4-variable functions by team scoring.

As our requirements vary, we may attach more or less weight to first and second scores. If extra signal loads are to be avoided, first scores are of greater consequence; otherwise, second scores. Hence, it is important that we evaluate functions in both ways.

From the 436 functions maximal by first scoring, four-input devices of six outputs at most are *irredundant* in that no 3-variable subfunction is repeated. The outputs of one such are shown immediately below. Irredundant devices with less outputs are (obviously) included.

[4 1′ 7 6′] [3′ 6′ 5 1′] [3′ 3 2 4]
[4 4′ 5 2] [3′ 2′ 0′ 3′] [7′ 6′ 3 1]

When second-scoring is followed, no four-input device of more than one output is irredundant, if all 3-variable subfunctions are full. Two-output devices which achieve as many as 27 such subfunctions are possible, however. The outputs for one of these are as follows:

[7 2′ 0′ 6] [3′ 4′ 0′ 2′]

It is simpler to judge functions at hand than to identify unknown optimum cases. Specific elements are easily compared to one another, and we can estimate the scores of maximal functions without producing them. Reflection shows, for example, that by first scoring, a top 5-variable element should fall somewhat short of the 3-variable limit *40*. Some other probable scores are shown in Table 5.

Table 5 **Estimated highest first scores of functions.**

6-variable	*60* (4-var.)	*68* (3-var.)	
7-variable	*84* (5-var.)	*280* (4-var.)	
8-variable	*112* (6-var.)	*448* (5-var.)	*1120* (4-var.)

To obtain higher-scoring functions beyond four variables, simple exhaustion of cases is unfeasible. Several strategies can be adopted. We may decide, in advance, probable characteristics of good functions and sample accordingly. Or, without preconception, we may let the computer *search* as follows. A randomly derived function is changed slightly. Scores are compared and the loser dropped. The process continues until a dead end is reached. Then the computer starts over with a new random element. Both strategies have been used with some success.

In terms of 3-variable subfunctions, *36* is the highest first score we have obtained for 5-variable elements. Four such are as follows:

[1 4 4′ 7′ 7′ 4′ 0′ 2] [1 4 4′ 7′ 6′ 0′ 6′ 2]
[1 4 4′ 7′ 6 6′ 2′ 2] [1 4 4′ 7′ 6 2′ 6′ 2]

If the first of these is written as a binary number,

0 0 1 0 1 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0

the two halves are mirror images of one another. Indeed, all of the *36's* found show balance. ONEs and ZEROs are of equal number in every case. Also, the arrangement of ONEs is such that the subfunctions obtained are distributed as evenly as possible into the four subclasses fixed by the two end bits of the available subfunctions. Although many *36's* were independently produced, they all fall into but four cycles, representatives of which are shown above.

The best first-scoring five-input, two-output elements we have found generate *57* 3-variable subfunctions. Three-output elements that first-score a near maximal *67* and have no redundant 4-variable subfunctions have also been attained. Table 6 provides an example of each.

In the actual fabrication of devices, it is not difficult to obtain the denial of a given function. Hence there is possible economy in two-output elements with one output the denial of the other. For five-input, two-output devices of this kind, *56* is the highest first score we have

Table 6 **High-scoring five-input logical elements.**

Two-output	[4′ 1 0 1 5 6 4 4′]
	[0′ 3′ 3 1 7′ 1′ 5′ 6′]
Three-output	[5 4′ 6 4 6 3 2 0′]
	[2 6′ 1 3′ 3 7 7 6′]
	[0′ 3 0 4 6 5′ 5′ 2′]

Table 7 **Generation of eighty 3-variable precursors from six-input, one-output device by biasing only.**

	Precursor	*Code Equivalent*		*Precursor*	*Code Equivalent*
1.	[0 0]	1 0 0 0 0 0	41.	[4 0]	*p* 1 1 *q r* 1
2.	[0 0′]	*p* 1 *q r* 0 1	42.	[4 0′]	1 1 0 *p q r*
3.	[0 1]	0 *q r* 1 0 *p*	43.	[4 1]	*p q r* 1 0 0
4.	[0 1′]	0 0 1 *q* 0 *r*	44.	[4 1′]	1 *q* 1 *p r* 1
5.	[0 3]	1 *p* 0 *q* 0 *r*	45.	[4 3]	*p r* 1 1 *q* 1
6.	[0 3′]	0 1 *p r* 0 *q*	46.	[4 3′]	*p q* 1 *r* 0 0
7.	[0 4]	*p* 1 *q* 1 0 *r*	47.	[4 4]	1 1 0 *p q* 1
8.	[0 4′]	1 1 *p q* 0 *r*	48.	[4 4′]	1 *r* 0 1 *q p*
9.	[0 5]	*p r* 1 0 0 1	49.	[4 5]	*q* 0 *p* 0 0 *r*
10.	[0 5′]	0 1 *r q p* 1	50.	[4 5′]	*p* 0 1 *r* 0 *q*
11.	[0 7]	0 *p q r* 0 0	51.	[4 7]	*p* 1 1 *q* 1 *r*
12.	[0 7′]	1 *r* 0 1 0 0	52.	[4 7′]	1 *p* 1 *q* 1 *r*
13.	[0′ 3]	1 *p* 0 *q r* 0	53.	[4′ 3]	1 *p q r* 0 0
14.	[0′ 3′]	*p q r* 0 1 1	54.	[4′ 3′]	*p* 1 *q* 0 *r* 0
15.	[0′ 4]	*p r q* 0 0 1	55.	[4′ 4]	*p* 0 *q r* 0 1
16.	[0′ 4′]	*p* 1 *q* 0 1 *r*	56.	[4′ 4′]	0 1 1 *p q* 0
17.	[0′ 5]	1 *r p* 0 0 *q*	57.	[4′ 5]	1 0 *p r* 0 *q*
18.	[0′ 5′]	*r* 1 0 *q p* 0	58.	[4′ 5′]	0 *r* 1 *q p* 0
19.	[0′ 7]	*r p* 0 *q* 1 0	59.	[4′ 7]	1 *p* 1 1 *q r*
20.	[0′ 7′]	0 1 *r* 0 *p q*	60.	[4′ 7′]	*p q r* 1 1 1
21.	[1 4]	*p q* 1 1 0 *r*	61.	[5 4]	*q* 0 0 0 *p r*
22.	[1 4′]	1 *q p* 1 0 *r*	62.	[5 4′]	1 1 *p q r* 0
23.	[1 5]	*q p* 0 0 0 *r*	63.	[5 5]	0 0 0 1 *p* 0
24.	[1 5′]	0 *q r* 1 *p* 1	64.	[5 5′]	1 *q p r* 1 0
25.	[1 6]	*p q* 0 *r* 0 0	65.	[5 6]	1 1 *q p* 1 *r*
26.	[1 6′]	1 *r* 1 *p* 0 *q*	66.	[5 6′]	*r* 1 *p q* 1 0
27.	[1 7]	0 *p q* 0 0 *r*	67.	[5 7]	0 *r* 0 1 *p q*
28.	[1 7′]	0 *r* 1 *p* 1 *q*	68.	[5 7′]	0 1 0 *r p* 0
29.	[1′ 6]	1 0 *q* 0 *r* 1	69.	[5′ 6]	1 *p q* 1 1 *r*
30.	[1′ 6′]	1 0 1 *p q r*	70.	[5′ 6′]	*p* 1 *q* 1 1 *r*
31.	[1′ 7]	0 *q r* 0 1 *p*	71.	[5′ 7]	*q* 0 *p r* 1 0
32.	[1′ 7′]	*q* 0 *r* 0 1 1	72.	[5′ 7′]	*q r* 0 *p* 1 1
33.	[3 4]	*p* 0 0 *q r* 0	73.	[7 4]	0 0 0 *p q r*
34.	[3 4′]	1 0 *p* 1 *q r*	74.	[7 4′]	*p* 1 1 *q r* 0
35.	[3 5]	*q p* 0 0 *r* 0	75.	[7 5]	0 *p* 0 0 *q r*
36.	[3 5′]	*q* 0 *r* 1 *p* 1	76.	[7 5′]	1 *q* 1 *r p* 0
37.	[3 7]	0 *p q* 0 *r* 0	77.	[7 7]	0 0 0 *p* 1 *q*
38.	[3 7′]	*p* 0 1 0 *q r*	78.	[7 7′]	*p r* 1 *q* 1 0
39.	[3′ 7]	*p* 0 *q* 0 1 *r*	79.	[7′ 7]	0 *p* 0 *q* 1 *r*
40.	[3′ 7′]	0 *p q* 1 1 *r*	80.	[7′ 7′]	0 1 0 0 0 0

found. The element obtained from [0 3 3′ 5′ 4 5′ 7′ 0′] and its denial is an example.

When second-scoring is considered, five-input, one-output functions have been found which produce *54* 3-variable subfunctions. None of these has a first score higher than *34,* however, so that we must consider the probable application in choosing an optimum function. Two of these high-scoring *54's* are as follows:

[1 4 4′ 6′ 6 4′ 7′ 2] [1 4 6 2′ 7 5′ 2′ 6]

The element on the left has a first score of *33;* the one on the right, *34.* When 4-variable subfunctions are counted by second-scoring, however, the leftmost function achieves a maximum *20;* but the other, only *18.* Because the particular choice of end bits (1 . . . 1 or 0 . . . 1 or 1 . . . 0 or 0 . . . 0) require that no new subfunctions will be gotten by duplication after biasing in one of the four exclusive classes of subfunctions determined by end bits, we see that a 3-variable second score in the *mid-50's* is the highest possible for 5-variable functions.

No 5-variable functions with opposite end bits have been found with a higher second score than *53* 3-variable subfunctions. Because the best three-input and four-input functions have opposite end bits, special interest attaches to these *53's.* All of those found fall into two cycles, whose chief precursors are as follows:

[0′ 4′ 5 1 7 5′ 3 2′] [1 4′ 6 4 7 3 5 6′]

Five-input, two-output devices are available with a maximum second score of *68* 3-variable and *40* 4-variable subfunctions. Indeed, such an element is accomplished from [0′ 3′ 4 1 4′ 5 1 1′] and its denial. Thus by a single added inversion, a 5-variable function can be extended so as to realize all possible functions of three variables.

As indicated earlier, a top 6-variable function should first-score *68* and *60* in subfunctions of three and four variables, respectively. Two such are shown below:

[6′ 0 1 3 2′ 0′ 3′ 6 3′ 2′ 7 4 6 4′ 6′ 3]
[4′ 7′ 0′ 3′ 1 0′ 1 2 7′ 2′ 5′ 3 4 1 3′ 6]

Either of these will, of course, realize all possible functions of three variables upon biasing. In Table 7, we have shown how the latter of the two generates the eighty precursors of three variables. As in Part I, the column under *Code Equivalent* indicates an input selection necessary to obtain the desired function.[8]

The two high-scoring 6-variable functions just shown have second scores of *234* and *231,* respectively, in subfunctions of four variables out of 245 possible cases. The score *244,* obtained by the function [2 6 4′ 0 5′ 0′ 3′ 1′ 7 4 4′ 3 5′ 4 5 4′], is the highest we have found thus far.

To illustrate the suggested method of testing further, it may be helpful to consider a more familiar logical element. Binary multipliers are typical "black boxes" found in computers. To multiply three-bit numbers, six 6-variable functions are required, of which four are full. These four (in order of significance) and their *individual* first and second scores are shown in Table 8. Combined first scores of *33* and *38* subfunctions of three and four variables, respectively, are obtained. Combined second scores of *39* and *100,* respectively, are also produced. Such scores for a six-input, four-output device are poor even by comparison with six-input, one-output logical elements.

Some related questions

The present study can be extended in various ways. First of all, a greater use can be made of the computer in analyzing data as against obtaining it. What is required are more direct methods of determining optimum cases. To do the latter, we must express in more formal terms the characteristics of high-scoring functions. It is felt the data already gathered are sufficient for a good start in this endeaver.

Questions of optimum *size* of basic elements need also be considered. Suppose we grant that for reasons of economy and reliability it is desirable to build our machine out of relatively standardized blocks. The present test enables us to compare elements of comparable inputs and outputs, but we have not yet examined the problem as to how large our building blocks should be for given purposes. Suppose, for example, avoiding duplication of inputs, we must choose between one-output elements of four or six inputs. The latter will obviously cost more to fabricate; but fewer units will be required in assembling circuits because of the very much greater versatility. As we have seen, a four-input, one-output device has a highest first score of *8* 3-variable subfunctions; a six-input element can achieve *68* such subfunctions.

Table 8 **Significant outputs of three-bit multipliers.**

Logical element	*First scores (3-var., 4-var., resp.)*	*Second scores (3-var., 4-var., resp.)*
[5 4 0 0 4 0 0 0 4 4 0 0 0 0 0 0]	4	4
	5	5
[6 1 4 4 1 5 0 0 3 1 0 0 5 5 0 0]	19	20
	13	29
[0 7 3 1 3 6 0 0 5 3 5 5 6 6 0 0]	21	33
	21	61
[2′ 3 5 3 2′ 5 5 5 1′ 6 6 6 7′ 0 0 0]	14	29
	12	31

Table 9 **Partial team first-scores (est.) of elements.**

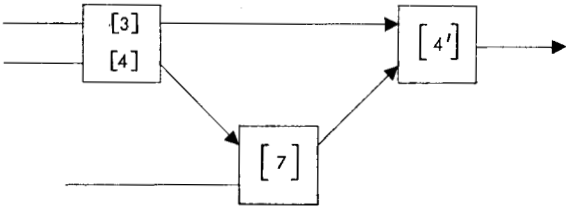
Element	*One unit*	*Two units*	*Three units*
Two-input	*1* (2-var.)	*2* (3-var.)	*5* (4-var.)
Three-input	*1* (3-var.)	*3* (5-var.)	*12* (7-var.)
Four-input	*1* (4-var.)	*4* (7-var.)	*21* (10-var.)

Team scoring may prove helpful on this problem. Suppose we wish to compare one-output devices of two, three, or four inputs. Initial proportions of *cost* and *versatility* are determined. Then progressively larger teams are examined. The former relationship remains constant; but, as Table 9 indicates, *a block initially favored gains in versatility*.

It appears advantageous to have blocks large, but not too large. Optimum size may well vary with the objective. Paradoxically, one must use enough elements for economy. Otherwise, team versatility is lost. Boxes assembled from lesser elements may in turn serve as building blocks for larger boxes, and so on. Because team membership need not be restricted to devices all from the same class, team scoring can also help determine which assortment of elements is best in a given case.

Methods for assembling economical circuits from optimum sets of elements are needed. As argued in Part I, techniques which minimize expressions formed from one set of functions may not prove relevant if another is assumed.[9] With the use of new basic elements, new ways of synthesizing and simplifying circuits may be required.

Questions of time can be introduced. In the hookup:



if all blocks operate instantaneously, the 3-variable function [3 0′] is realized. Assume, on the other hand, all three elements have some equal inherent delay and input signals follow one another at corresponding intervals. Because the lower path requires more time, two of the input terminals are used twice. Hence, the 5-variable function [0 3′ 7′ 4 7′ 4 0 3′] is obtained. It is well known that we may trade time for hardware, and *vice versa,* but the precise relationship of the two is not understood. In terms of the present study, a more systematic analysis can be undertaken.

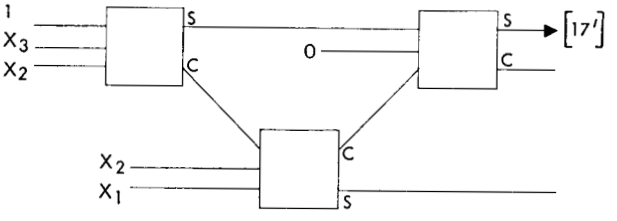
An associated study can be made of circuit reliability. Clearly, building from uniform blocks simplifies the description of hardware. Isolation of defective parts is more readily accomplished and replacement is more immediate. The redundancy intrinsic to multipurpose elements might also be used to insure reliability. Beyond this, a more general approach to the redundancy-reliability question may be possible, since existing proposals are often oriented to specific kinds of devices.[10]

The interplay of order codes and hardware need also be examined. A modern computing machine can be thought of as an extended multipurpose bias device. Alternative logical operations are performed, depending upon the presence or absence of certain signals. In general, we desire the widest range of orders possible from the least hardware. The present study is pertinent. The class of subfunctions generated by a logical element will determine the variety of orders it can execute.

We have thus only scratched the surface in our investigation of multipurpose elements. Part III, to be published at a later time, will carry the argument further.[11]

References

1. B. Dunham, "The Multipurpose Bias Device, Part I: The Commutator Transistor," *IBM Journal of Research and Development,* Vol. 1, 117-129 (April, 1957). See also R. F. Rutz, "Two-Collector Transistor for Binary Full Addition," *IBM Journal,* Vol. 1, 212-222 (July, 1957).
2. *Ibid.,* p. 118.
3. *Ibid.,* p. 120.
4. *Ibid.,* p. 119. For a more comprehensive discussion, see W. V. Quine, *Mathematical Logic* (Revised Edition, Harvard University Press, 1955), 42-43.
5. *Ibid.,* 120. The word "class" seems preferable to "group" in this context.
6. The fact that four full adders are not required in two of the 3-variable cases, as suggested on p. 123, Pt. I, was first pointed out to us by J. R. Logan, Senior Scientist, Litton Industries. He comments in a private correspondence: "You state tentatively that four full adders might be used to accomplish either [0 5′] or [1 7′]. In the following logical hookup



[1 7′] will emerge on the sum output and reversing the indicated biases will cause the same inputs to deliver [0 5′] on the same line." The results shown in Table 1 were obtained by computational means.
7. The number of interchange classes of four variables was calculated by A. Cobham of these laboratories before we obtained it by computational means.
8. Dunham, *op. cit.,* pp. 126-127.
9. *Ibid.,* p. 117.
10. See, for example, J. von Newmann's treatment in terms of *majority organs,* "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies,* Princeton, 1956, pp. 43-98.
11. Errata for Part I are as follows: p. 116—case (7-22-*BCE*), omit dot from second box; case (17-46—*BCFH*), omit bar over '*r*'; p. 121—case (35) *ABDF,* insert parenthesis before first '*p*' and insert bar over second '*p*'; case (58) *ABDFH,* add '|(*q*|*r*)' after second '*p*'; case (66) *BDFGH,* change second '*p*' to '*q*'; p. 123—first column, sixteenth line from bottom, insert '(See p. 116)' after 'Schematic Diagram 1'; p. 124—insert dot in fourth box from top; p. 129—first column, second line from bottom, change '1948' to '1938'.

Received March 26, 1958