Logic Synthesis of Some High-Speed Digital Comparators

By M. NESENBERGS and V. O. MOWERY

(Manuscript received September 10, 1958)

Logical schemes for realizing high-speed digital comparators are derived by Boolean algebra methods. Requirements for speed and precision place serious restrictions on the switching circuits. In particular, the precision requirement makes direct subtraction by the use of analog devices undesirable; the speed requirement dictates that any carry structure should propagate from the most significant digit toward the least significant digits. Such schemes have obvious advantages when only an approximate magnitude is desired. Changing numbers in binary code introduces the common transition problem due to multiple digit changes; this problem is avoided by use of the Gray code.

Circuits satisfying the synthesis requirements and giving the sign and exact magnitude of the difference are derived first. These schemes are then modified and simplified to give the sign and approximate magnitude. Circuits giving only the sign of the difference are also derived.

I. INTRODUCTION

1.1 *Applications*

A digital comparator compares two numbers presented in digital form and obtains a measure of the difference between them. Comparison may consist of detecting only the sign of the difference or the direction of mismatch of the two numbers, or the result of the comparison may be both magnitude and sign of the difference. The comparator is essentially a subtracter suitably modified to fulfill requirements of the intended application.

An immediate need for a dependable, high-speed digital comparator is in the feedback control loop for the flying spot store of an experimental electronic switching system.[1] R. W. Ketchledge has derived several methods of implementing such comparators.[2] In this application the comparator functions as an error detector, giving an output depending on the difference between the desired input address position and the fed-

back present position. By servo action, this difference then acts to eliminate the positional error. The input address is presented in parallel digital form and the present position is digitally encoded in parallel. For proper holding action of the servo the comparator output should be a linear error signal when close to a zero difference. A similar technique could be used in applications requiring fast and accurate positioning through a precise digital servo action.

The use of reliable, high-speed comparators in a digital computer allows greater flexibility of operation. Comparison of numbers, in the sequence of a computation, can be used to determine the choice of further operations. Since the programmer cannot always know the results of a certain operation in advance, a built-in comparison scheme can initiate judgement to proceed automatically with subsequent routines. Comparison of numbers is frequently employed in sorting and determining square roots and dividends.[3]

In a broad sense, all measurements can be considered comparisons. If the quantity to be measured appears in digital form the types of comparators to be considered here could be used, especially for rapidly varying quantities. Unlike an analog device, where precision is determined largely by the components and accuracy of driving potentials, the precision of a digital device is limited only by the number of digits used.

1.2 *Synthesis Requirements*

Intended applications of the comparators discussed here place serious restrictions on the logical form of the switching circuits. For example, the extremely high access precision necessary in the flying spot store prohibits the use of analog open-loop positioning and requires the use of digital closed-loop control. This precision requirement also eliminates the possibility of direct, complete analog subtraction of the digital signals in the error detector of the servo loop. High-speed operation of the servo system implies the use of electronic combinational switching circuits with simultaneous operation on all of the digits.

In a servo operating on the magnitude of the difference between the input address and the feedback signal, the value of the feedback signal fed into the comparator may be rapidly changing for large differences. For example, in the flying spot store servo presently operating only on the sign of the error or difference, the beam position moves at a velocity of three spots per microsecond. Each spot, corresponding to an address point on the cathode ray tube face, is designated by a digital number. With the feedback signal appearing in a binary code, one cycle of the least significant digit corresponds to two spots. The frequency or rate of change of this least significant digit is therefore 1.5 mc. If the feedback

signal appears in a Gray code (discussed in Section 2.2) one cycle of the least significant digit corresponds to four spots and the rate of change of this digit is then half of that for the binary code. For a servo controlled by the magnitude of the difference, the velocity of the change of beam position is proportional to the difference. As a specific example, consider such a proportional servo operating with a Gray code and with an error of about 200 spots. The bandwidth required for the least significant digit would then be about $200 \times \frac{3}{4}$ mc $= 150$ mc.

It is apparent that in such applications the digits of lower significance can be changing at such a rate that their use in the comparison or subtraction scheme becomes impractical. The digits of higher weight will be better defined and those of lower weight will be blurred, due to band limitations. Logical operations, whenever possible, should therefore be performed on the more significant digits, and any carries necessary should propagate from digits of higher significance to digits of lower significance. This synthesis requirement prohibits the use of conventional parallel subtracters with a borrow propagating from the least significant digits.

In many applications only an approximate, or order of magnitude, difference may be required. For such applications it is also advantageous to have the carry or carries in the subtraction operation propagate from digits of higher significance to digits of lesser significance. Since the magnitude of the mismatch between digital numbers is usually determined by digits of higher significance, an approximate difference can often be obtained without the necessity of the carries propagating through all of the digits. This is not possible in a conventional subtracter. In either case, however, it is necessary to examine all digits to obtain an exact difference.

There is little loss of generality in assuming that the input address number, designated by A, appears in two-rail parallel form, as, for example, from a flip-flop register, and that the second number, which can be rapidly changing, appears in a parallel binary or Gray-code form, designated by B or G. For speed of operation, it is desirable to perform as much of the logic as possible on the digits of the fixed number A and to have the digits of the changing number B or G travel through a minimum number of series gates. This does not mean that minimal switching circuits will always be used, however, since there may be advantages to either combining or sharing functions in a slightly expanded circuit. Only functional forms of the switching circuits containing AND, OR and EXCLUSIVE-OR gates plus inverters will be derived here. For economy, flexibility and ease of replacement, an iterated logic structure is desirable. The actual electronic circuits used to realize the

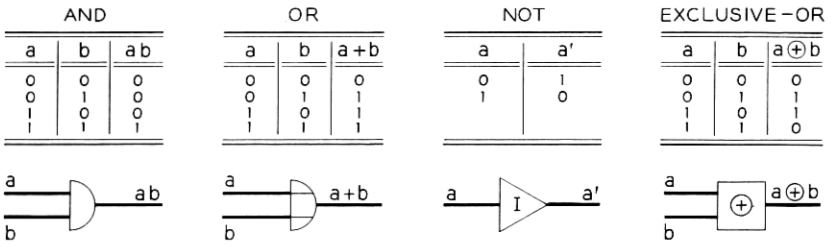

Fig. 1 — Truth tables and circuit symbols for logical operations.

various functions will not be discussed. It should be mentioned that the comparators giving both magnitude and sign of the difference are intended to drive digital-to-analog converters giving an analog output. In all cases when the difference is within ± 1 a linear indication of the difference is required if the comparator is to perform properly in a feedback control loop.

II. NOMENCLATURE

2.1 *Algebraic Operations*

Synthesis of the logic circuits to be used in the comparators is frequently simplified by the use of algebraic expressions. Boolean or switching algebra[4] has proved to be a convenient notation permitting manipulation of series-parallel two-terminal networks into a variety of equivalent circuits, often resulting in simplified or more appropriate forms. The quantities involved in Boolean algebra can be represented by letters or symbols. These symbols represent signals having discrete on-or-off values, represented by 1 or 0 respectively. This convention differs from that sometimes used (as, for example, in Ref. 4). Logical operations employed here will be briefly defined.

The AND operation is a logical conjunction or intersection resulting in 1 only when both variables are 1. The OR operation is the logical disjunction or union, indicated by $+$, resulting in 1 when either or both of the variables are 1. Truth tables and circuit symbols of these operations are shown in Fig. 1. As a consequence of these rules, both the OR and AND functions are commutative, associative and distributive. For example:

$$\begin{aligned} (a + b) + c &= (b + c) + a, \\ a(b + c) &= ab + ac, \\ (a + b)(a + c) &= a + ab + ac + bc, \\ &= a + bc. \end{aligned}$$

An additional concept is the NOT, complement, or negation indicated by a prime. The truth table and circuit symbol for this operation is also shown in Fig. 1. Since effectively the NOT function is an inversion of the signal, the symbol represents an inverting amplifier. Several important relations follow:

$$\begin{aligned} a + a' &= 1, \\ aa' &= 0, \\ (a')' &= a, \\ (ab)' &= a' + b'. \end{aligned}$$

The validity of such equations can always be verified by the method of perfect induction, or substituting for the variables the two possible values, 0 and 1, in all combinations.

A Boolean algebra is closed under the operations of negation and either the OR or AND operations; however, for convenience, we will allow both the AND and OR operations. In addition, we will find it convenient to use a fourth operation called the EXCLUSIVE-OR or "ring-sum" defined as

$$a \oplus b = ab' + a'b. \tag{1}$$

This ring sum is 1 if either a or b, but not both, are 1; it is 0 if both a and b are either 1 or 0. The ring-sum therefore detects a mismatch between the two digits and is the algebraic expression for the common half-adder used in conventional digital adders and subtracters (Ref. 3, Ch. 4). Fig. 1 also shows the truth table and circuit symbol for this operation. Note in particular that

$$a \oplus 0 = a \quad \text{and} \quad a \oplus 1 = a'. \tag{2}$$

2.2 *Codes and Translations*

The type of synthesis used in developing the comparators is dependent upon the types of codes used to represent the numbers. For this reason a brief discussion of codes employed and the translations between them is included.

One of the most convenient number systems for logical operations is the binary system. Since the number $B = b_m b_{m-1} \cdots b_1 b_0$ is represented to the base 2, and therefore each digit b_i takes on the value 0 or 1, the Boolean algebra described in the previous section can be conveniently applied to the digits. The magnitude of the integral number B is represented in this binary code by

$$B = \sum_{i=0}^{m} b_i 2^i,$$

TABLE I

Decimal	Binary			Gray		
	b_2	b_1	b_0	g_2	g_1	g_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

where m is the most significant place. In this conventional binary system multiple digit changes occur for every increase by two, and half-way through the code all of the digits change. Table I shows the three-digit binary code.

Such multiple digit changes cause difficulties whenever all of the changing digits do not change simultaneously. Nonsimultaneous changes of the digits may be due to variations of bias, gain, delay or operating levels of the individual stages or to misalignment of the coding devices. This is the familiar problem encountered, for example, in digitally encoding shaft positions or other analog-to-digital conversions, in digital positional servomechanisms,[5] and in pulse code communication.[6] To avoid the difficulty of incorrectly reading a rapidly changing number, a Gray[7] or reflected binary code[8] may be introduced, in which only one digit changes between successive numbers of the code.

In our algebraic synthesis of various comparators it is convenient first to consider both numbers in the conventional binary code and then to translate to Gray code the input number, which may be rapidly varying. The cyclic reflected binary code, which we will call simply Gray, has the convenient property of simple translation to and from the conventional binary equivalent. Table I also shows the three-digit Gray code.

The method for finding the magnitude of a number G written in the Gray code is more involved than is evaluating a number written in the binary code. Each digit g_i again has the value 0 or 1, but the weight of the ith digit is now $(2^{i+1} - 1)$ and the sign of each digit not a zero is now alternated, starting with + for the most significant digit g_m (where $g_m \neq 0$) and alternating in sign for each digit which is not zero. This could be termed the decimal translation, and can be written

$$G = \sum_{i=0}^{m} g_i(2^{i+1} - 1)(-1)^{(g_m+g_{m-1}+\cdots+g_{i+1})}.$$

The usual rule for translating the digits of a number in binary code to the digits of a number in Gray code is the following: If the binary digit b_i is preceded by a 1 ($b_{i+1} = 1$), change the ith digit ($g_i = b_i'$); if it is preceded by a 0 ($b_{i+1} = 0$), use the same digit ($g_i = b_i$). This can be written in the shorthand notation of the ring-sum operation:

$$g_i = b_i \oplus b_{i+1} . \tag{3}$$

To convert a Gray digit to its equivalent binary form the rule is to reverse those Gray digits which are preceded by an odd number of 1's in the Gray digits of higher significance. This can be expressed in terms of repeated ring-sums which, in effect, counts the number of preceding 1's:

$$b_i = g_i \oplus g_{i+1} \oplus g_{i+2} \oplus \cdots .$$

Repeated ring-sum operations effectively performs the same function as the modulo 2 notation in determining whether a set of digits is odd or even. An equivalent Gray-to-binary translation can be obtained from the previous binary-to-Gray translation by ring-summing both sides of (3) with b_{i+1} :

$$g_i \oplus b_{i+1} = b_i \oplus b_{i+1} \oplus b_{i+1} = b_i \oplus 0 = b_i . \tag{4}$$

III. EXACT PROPORTIONAL COMPARATORS

3.1 *Analog Precision Problem*

Two main requirements imposed on these comparator syntheses are speed of operation and precision. Perhaps the fastest method of comparison of digital numbers is simple and direct analog subtraction. A typical analog subtracter for obtaining the difference between two binary numbers A and B is shown in Fig. 2. An output voltage (or current) corresponding to the difference is obtained by shunting suitably weighted currents into the summing resistance R_S (or the load). Currents of proper magnitude and polarity are obtained from the AND gates controlled by the individual digits.

Although it is possible to build fast analog subtracters, it may be difficult to meet stringent precision requirements. Difficulties may arise when subtracting large numbers having a small difference. An error as small as one half of one part in the maximum value of the numbers can possibly even result in an incorrect sign for the difference. Analog methods of avoiding these difficulties are not entirely satisfactory. For accurate results, precision components and well-regulated supplies are

necessary. Further complications may result when one of the numbers is in Gray code.

Application of comparators as the error-detecting element in a digital feedback control loop usually implies an analog difference output in order to drive the control elements. What is desired is a method of avoiding direct analog subtraction of two numbers of nearly the same size. This problem is illustrated by the examples:

A	1000	A	0111
B	0111	B	1000
Diff.	+0001	Diff.	−0001

We call such a grouping of digits, where a mismatch in one direction is followed immediately by consecutive mismatches in the opposite direction, a run. Such a run ends when it is followed by a match of the digits or a mismatch in the original direction. If the two original input numbers can be operated on digitally to eliminate such runs then the precision problem can be avoided and analog subtraction can be retained.

3.2 *A One-Carry Binary-Binary Comparator*

Since runs of consecutive digits of the type described in the previous section lead to difficulties in analog subtraction, it is desirable to transform the two binary input numbers A and B into an equivalent set W and V, in which these runs are eliminated. The equivalence implied

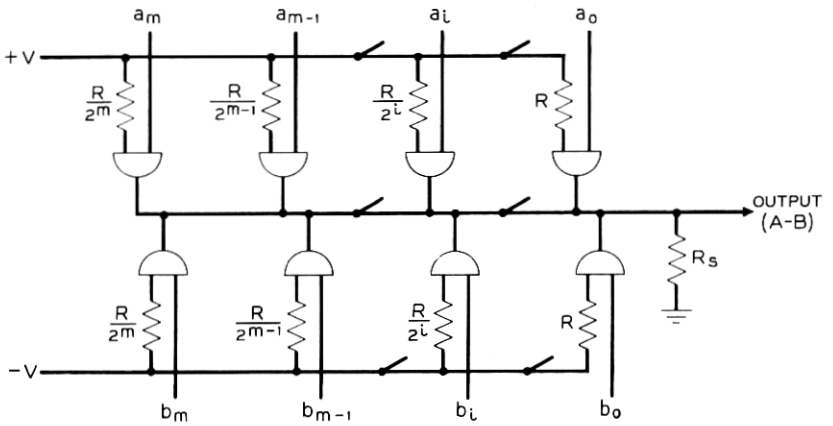

Fig. 2 — Typical analog subtracter.

TABLE II — TRUTH TABLE FOR CARRY FUNCTION OF ONE-CARRY BINARY-BINARY LOGIC

a_i	b_i	a_{i-1}	b_{i-1}	q_i	
				If $q_{i+1} = 0$	If $q_{i+1} = 1$
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	0	0

here is that of $A - B = W - V$ (see example on p. 31). Furthermore, it is also undesirable to perform analog subtraction of equal-weight digits, since a similar precision problem is likely. In this section a simple logic scheme for eliminating these runs and avoiding subtraction of equal digits is illustrated. Resulting circuits are not the most practical for this purpose but will serve to demonstrate the method. The logic will be derived in detail to further illustrate the method.

For reasons outlined in Section 1.2, comparison of the two numbers A and B should start from the most significant digit and proceed toward the lower significant digits. Whenever a run starts, i.e., a mismatch followed by a mismatch of the opposite kind, the outputs should be prohibited. To do this, we can start a carry, or inhibit, function, with the ith digit represented by q_i . This carry should then propagate through the run and stop at the last digit, so that an output can be permitted. In other words, if there is a carry coming in from the next more significant digit, i.e., $q_{i+1} = 1$, then the carry should continue only if any mismatch of the following digits a_{i-1} and b_{i-1} is of the same sign. Or, if there is no carry coming in, i.e., $q_{i+1} = 0$, then a carry should start only if a mismatch of the ith digits is followed by an opposite mismatch in the a_{i-1} and b_{i-1} digits. A carry is possible only when there is a mismatch in the digits a_i and b_i . Table II is a modified truth table giving the carry digit for the two conditions $q_{i+1} = 0$ and $q_{i+1} = 1$. The disjunctive canonical form for this truth table is obtained from the OR of the minimal polynomials for each row. The carry function can then be written:

$$q_i = q_{i+1}'(a_i'b_ia_{i-1}b_{i-1}' + a_ib_i'a_{i-1}'b_{i-1}) + q_{i+1}(a_i'b_ia_{i-1}'b_{i-1} + a_ib_i'a_{i-1}b_{i-1}').$$

TABLE III — TRUTH TABLE FOR OUTPUT FUNCTIONS OF ONE-CARRY BINARY-BINARY LOGIC

q_{i+1}	a_i	b_i	w_i	v_i
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
*1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
*1	1	1	0	1

* From (5) or the truth table shown in Table II, these conditions cannot occur and the outputs have been chosen to simplify the final expressions.

Using the ring-sum or EXCLUSIVE-OR operation introduced in Section 2.1, this carry can be rewritten as

$$q_i = (a_i \oplus b_i)(a_{i-1} \oplus b_{i-1})(q_{i+1} \oplus a_i \oplus a_{i-1}). \tag{5}$$

Outputs are permitted only when there is no carry or inhibit function; therefore, a condition for any output is $q_i = 0$. The output digits w_i carry positive weight and the digits v_i carry negative weight. If we are at the end of a run, with $q_{i+1} = 1$, then a positive output is required if $a_i = 0$ and a negative output if $a_i = 1$. These conditions are apparent from the examples given in Section 3.1. If we are not in a run, i.e., $q_{i+1} = 0$, then an output is allowed only if there is a mismatch: a positive output for $a_i = 1$ and $b_i = 0$, and a negative output for $a_i = 0$ and $b_i = 1$. Conditions for the outputs are summarized in Table III. Recalling the condition $q_i = 0$, the outputs can be obtained from the truth table in disjunctive canonical forms:

$$w_i = q_i'(q_{i+1}'a_ib_i' + q_{i+1}a_i'b_i' + q_{i+1}a_i'b_i),$$

$$v_i = q_i'(q_{i+1}'a_i'b_i + q_{i+1}a_ib_i' + q_{i+1}a_ib_i).$$

Again, using the ring-sum notation, these outputs can be manipulated to the equivalent expressions:

$$\begin{aligned} w_i &= q_i'(a_i' + b_i')(q_{i+1} \oplus a_i), \\ v_i &= q_i'(a_i + b_i)(q_{i+1} \oplus a_i)'. \end{aligned} \tag{6}$$

This form is convenient because forming the ring-sum of the ith digits by the combination $a_i \oplus b_i = (a_i + b_i)(a_i' + b_i')$ allows some of the operations in the carries and outputs to be shared.

Fig. 3 shows the ith digit of a one-carry comparator utilizing the logic of (5) and (6). Since all runs have been eliminated, it is not possible to have adjacent output digits of opposite polarity. Nor is it possible in have scheme to have both polarity outputs occurring together. These properties follow from the equations because the logical products $w_i v_i$, $w_i v_{i-1}$ and $w_{i-1} v_i$ vanish identically. Under these conditions, subtraction of the w_i and v_i digits can be performed in an analog subtracter such as shown in Fig. 2 without encountering the precision problem discussed previously.

3.3 *A Two-Carry Binary-Binary Comparator*

It should be apparent, after careful examination of the subtraction process for two binary numbers and paying particular attention to the runs described in Section 3.1, that many schemes are possible for translating the input numbers A and B into an equivalent set W and V meeting the requirement demanded by analog subtraction. In the previous section we derived a comparator logic using a single carry operating as an inhibit function. In this section we will outline the synthesis of a comparator using two carries which perform the function of permitting

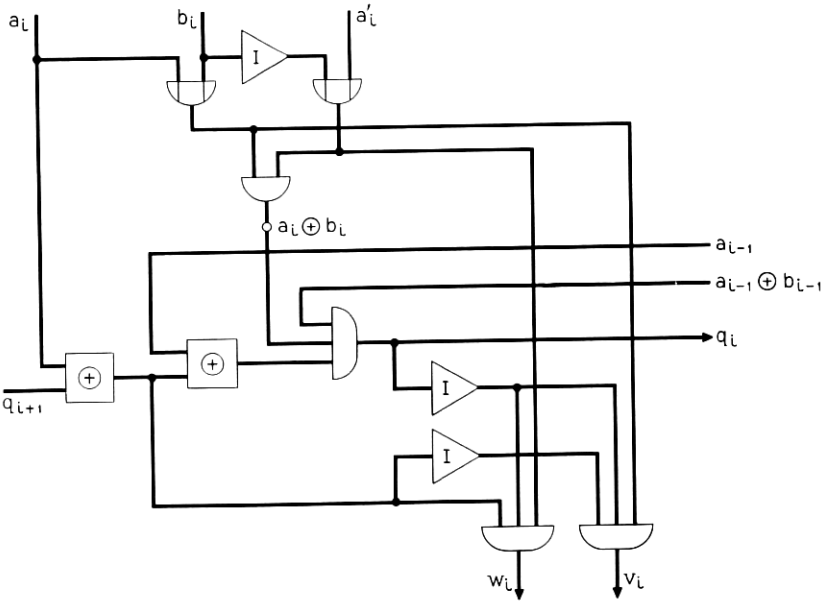

Fig. 3 — One-carry binary-binary comparator.

the outputs under proper conditions. This scheme has a somewhat simpler circuit and has the advantage that the logic can be modified to operate with one of the input numbers in the Gray code. Also, if the difference is required to be only approximately proportional to the exact difference, while still meeting the precision requirements of analog subtraction, then the logic can be further simplified.

Since an output should be permitted at the ith digit only when there is a mismatch between the digits a_i and b_i, a necessary condition for any carry which permits an output should be the ring-sum $a_i \oplus b_i = 1$. As described in Section 2.1, this logical operation detects a mismatch of either polarity. Let the ith digit of the carry which permits a positive output w_i be designated n_i and the ith digit of the carry permitting a negative output v_i be designated m_i.

If there are no carries coming into the ith digit from the next more significant digit, i.e., $n_{i+1}'m_{i+1}' = 1$, and if the mismatch is of the type $a_ib_i' = 1$, then the positive carry should be started. Similarly, if there are no carries coming in and the mismatch is of the type $a_i'b_i = 1$, then the negative carry should be started. Also, if there is a mismatch in the ith digit, i.e., $a_i \oplus b_i = 1$, with a carry coming in from the next more significant digit, then it should be propagated through this ith digit in order to permit detection of the runs discussed in Section 3.1.

These rules could again be summarized in a truth table from which the required functions could be derived. However, it should be apparent from the previous description that the carries satisfy the following functions:

Positive carry:

$$n_i = (a_i \oplus b_i)(n_{i+1}'m_{i+1}'a_ib_i' + n_{i+1}),$$

Negative carry:

$$m_i = (a_i \oplus b_i)(n_{i+1}'m_{i+1}'a_i'b_i + m_{i+1}).$$

These functions can be simplified by algebraic manipulations. In particular, using the identities $x + x'y = x + y$ and $xy'(x \oplus y) = xy'$, the carries can be rewritten as

$$\begin{aligned} n_i &= n_{i+1}(a_i \oplus b_i) + m_{i+1}'a_ib_i', \\ m_i &= m_{i+1}(a_i \oplus b_i) + n_{i+1}'a_i'b_i. \end{aligned} \tag{7}$$

Note from the development of the carry structure that both carries cannot exist together. We can show this by noting from (7) that

$$\begin{aligned} n_km_k &= n_{k+1}m_{k+1}(a_k \oplus b_k) \\ &= n_{k+2}m_{k+2}(a_k \oplus b_k)(a_{k+1} \oplus b_{k+1}) \\ &= n_mm_m(a_k \oplus b_k) \cdots (a_{m-1} \oplus b_{m-1}). \end{aligned}$$

One more substitution gives a factor $n_{m+1}m_{m+1}$ which is always zero, since the subscript m denotes the most significant digit. Therefore $n_k m_k = 0$, for all k.

Outputs of a particular polarity are possible only when the carry permitting that polarity is present. Thus, a necessary condition for a positive output w_i is $n_i = 1$ and a condition for a negative output v_i is $m_i = 1$. Whenever we are in a run, no output should occur until the run is terminated. A positive run is terminated at the ith digit when $n_{i-1} = 0$ or $a_{i-1} = 1$ and a negative run is terminated at the ith digit when $m_{i-1} = 0$ or $a_{i-1} = 0$. If we are not in a run but a carry is present, then there must also be a mismatch. When $n_i = 1$ and $a_{i-1} = 1$ a positive output w_i is needed, and when $m_i = 1$ and $a_{i-1} = 0$ a negative output v_i is needed. Forcing as much logic as possible on one of the input numbers will be to our advantage when this comparator is modified to operate with one of the input numbers changing rapidly. The output functions satisfying the previous description are

$$\begin{aligned} w_i &= n_i(n_{i-1}' + a_{i-1}), \\ v_i &= m_i(m_{i-1}' + a_{i-1}'). \end{aligned} \tag{8}$$

The following example illustrates the formation of the carries N and M and the outputs W and V:

A	1	0	0	1	1	0	0	1	1
B	0	1	1	0	1	1	0	0	0
N	1	1	1	1	0	0	0	1	1
M	0	0	0	0	0	1	0	0	0
W	0	0	1	1	0	0	0	1	1
V	0	0	0	0	0	1	0	0	0

Fig. 4 shows a typical digit of a two-carry comparator using the logic of (7) and (8).* This circuit, with carries propagating from the more significant digits, is suitable for driving an analog subtracter, since all runs have been eliminated and outputs of equal weight and opposite polarity are not possible. The circuit also has the advantage that much of the logic is performed on only one of the input numbers.

3.4 *A Two-Carry Binary-Gray Comparator*

For those applications in which one of the input numbers can be changing rapidly the multiple digit changes of the binary code lead to the

* Equations (7) and (8) can be manipulated to an equivalent scheme obtained by Ketchledge[2] from other considerations.

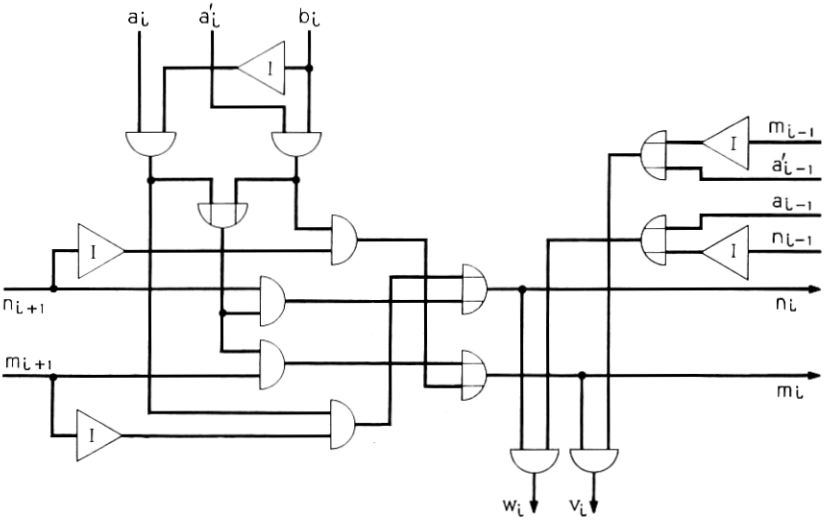

Fig. 4 — Two-carry binary-binary comparator.

difficulties discussed in Section 2.2. In this case, it is desirable to use the Gray code for the changing number and, since the cyclic reflected-binary type Gray code has the simple translation property given by (3) and (4), a translation of the logic of Section 3.3 can be easily carried out. If we make the proper translation of the binary input number B into a Gray number G, then the outputs W and V remain unaltered.

Equation (3) gives the translation from binary code to the Gray code used here. From the logic of Section 3.3, if there is a carry coming into the ith digit, i.e., $n_{i+1} = 1$ or $m_{i+1} = 1$, then there must have been a mismatch of the previous digits, i.e., $a_{i+1} \oplus b_{i+1} = 1$. Using the ring-sum properties given by (2), the ith Gray digit under these conditions can be expressed as

$$\begin{aligned} g_i' &= b_i \oplus b_{i+1} \oplus a_{i+1} \oplus b_{i+1} \\ &= b_i \oplus a_{i+1} . \end{aligned}$$

Rearranging terms gives

$$b_i = (g_i \oplus a_{i+1})' \qquad \text{if} \qquad a_{i+1} \oplus b_{i+1} = 1.$$

Similarly, if there is a match of the preceding digits, or $a_{i+1} \oplus b_{i+1} = 0$, then the carries coming in must also be zero, i.e., $n_{i+1} = 0$ and $m_{i+1} = 0$.

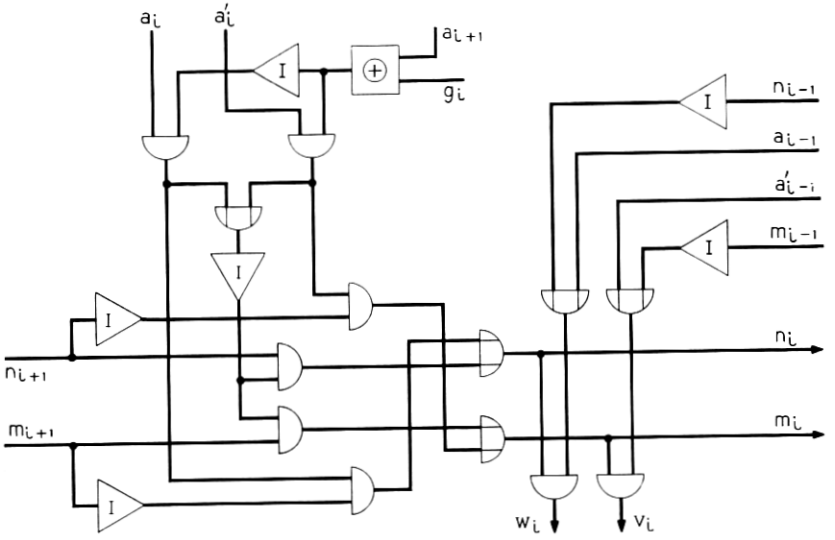

Fig. 5 — Binary-Gray comparator.

Under these conditions, the ith Gray digit can be expressed as

$$g_i = b_i \oplus a_{i+1},$$

or

$$b_i = g_i \oplus a_{i+1} \qquad \text{if} \qquad a_{i+1} \oplus b_{i+1} = 0.$$

These translations can then be substituted in the carries of (7) under the proper conditions determined by the incoming carries. The resulting logic scheme for the binary-Gray comparator is given by the following equations* and appears in Fig. 5:

$$\begin{aligned} n_i &= n_{i+1}(a_i \oplus a_{i+1} \oplus g_i)' + m_{i+1}'a_i(a_{i+1} \oplus g_i)', \\ m_i &= m_{i+1}(a_i \oplus a_{i+1} \oplus g_i)' + n_{i+1}'a_i'(a_{i+1} \oplus g_i), \\ w_i &= n_i(n_{i-1}' + a_{i-1}), \\ v_i &= m_i(m_{i-1}' + a_{i-1}'). \end{aligned} \tag{9}$$

* These equations can also be manipulated to an equivalent scheme obtained by Ketchledge[2] from other considerations.

IV. APPROXIMATE PROPORTIONAL COMPARATORS

4.1 *Approximate Binary-Binary Comparator*

In some applications of digital comparators it is not necessary to obtain the exact difference. For example, a digital servo will operate satisfactorily if the comparator or error detector supplies a control signal which increases with increasing error and decreases for decreasing error, but which is not necessarily equal to the true error difference, except for small errors. It is difficult to build circuits using the logic of the previous sections which meet very high speed requirements and it is desirable to simplify the circuits as much as possible. The logic schemes derived in this and the following section are attempts to simplify the circuitry of the exact proportional comparators synthesized previously. Again, many variations are of course possible. The particular binary-binary approximate comparator synthesized in this section meets the further requirement of simple translation to a binary-Gray approximate comparator.

Consider the subtraction of two binary numbers A and B where the most significant output occurs in the kth digit. Then $|A - B|$ will be maximum if all the following digits ($k-1, k-2, \cdots, 1, 0$) have the same polarity outputs. With the run structure determining the outputs as described in Section 3.1, it is not possible to have an uninterrupted sequence of opposite polarity outputs start in the next digit. Therefore $|A - B|$ will be minimum if all the digits starting two lower ($k-2, \cdots, 1, 0$) have the opposite polarity outputs. From these considerations,

$$|A - B|_{\max} = 2^k + \sum_{i=0}^{k-1} 2^i = 2^k + (2^k - 1) = 2^{k+1} - 1,$$

$$|A - B|_{\min} = 2^k - \sum_{i=0}^{k-2} 2^i = 2^k - (2^{k-1} - 1) = 2^{k-1} + 1,$$

and therefore

$$\tfrac{1}{2} \cdot 2^k < |A - B| < 2 \cdot 2^k \tag{10}$$

whenever the most significant difference output occurs in the kth digit. This means that, if only the most significant difference digit is allowed, the result will always be within a factor of two of the exact difference. The same result will be obtained if the output digits of one polarity, say w_i, are cancelled by the appearance of digits of the opposite polarity v_i for $i < k$. These should of course not be strict rules in the synthesis

of approximate comparators since allowing some lower-order output digits may frequently improve the approximation.

When forming carries starting from the more significant digits it is still necessary to detect combinations of digits forming runs, as discussed in Section 3.1. This restriction avoids some of the difficulties when analog subtraction is to be performed on the new output difference digits. Two carries will again be used, with the property of determining outputs of the correct polarity in the proper digits.

The logic of this scheme is simplified if all carries are allowed to propagate unaltered through digits of lower significance. One of the conditions for a positive carry n_i is then an incoming carry, or $n_{i+1} = 1$, and a condition for a negative carry m_i is $m_{i+1} = 1$. If the most significant mismatch occurs in the ith digit it must necessarily be preceded by $a_{i+1} \oplus b_{i+1} = 0$. Under these conditions, a positive carry is started if $a_i b_i' = 1$ and a negative carry is started if $a_i' b_i = 1$.

Information concerning the end of a run, and therefore the need for an output, in the exact comparators discussed previously, was contained in the carries and the address number. Examination of the conditions ending a positive-type run, e.g., (1000) − (0111), at the ith digit indicates that it is necessary to have $a_{i-1}' b_{i-1} = 0$. One of the conditions is therefore $a_{i-1} = 1$. This condition will be used in the output expressions derived later. The other necessary condition for ending a positive type run at the ith digit is $a_{i-1}' b_{i-1}' = 1$. This condition will be used to start a carry of the opposite polarity. In other words, we start a negative carry m_i if there is a mismatch of the previous digits, i.e., $a_{i+1} \oplus b_{i+1} = 1$, and $a_i' b_i' = 1$. Similarly, the conditions for ending a negative-type run, e.g., (0111) − (1000), at the ith digit is $a_{i-1} = 0$ or $a_{i-1} b_{i-1} = 1$. The first condition will also be used in forming the outputs and the second condition will be used to start a positive carry. Thus, *after* the end of a run both carries may be present. The presence of a carry at the ith digit is given by the OR of the above conditions.

$$n_i = n_{i+1} + a_i b_i'(a_{i+1} \oplus b_{i+1})' + a_i b_i (a_{i+1} \oplus b_{i+1}),$$

$$m_i = m_{i+1} + a_i' b_i (a_{i+1} \oplus b_{i+1})' + a_i' b_i' (a_{i+1} \oplus b_{i+1}).$$

And, since $x'y' + xy = (x \oplus y)'$, these expressions can be rewritten as

$$\begin{aligned} n_i &= n_{i+1} + a_i (a_{i+1} \oplus b_{i+1} \oplus b_i)', \\ m_i &= m_{i+1} + a_i' (a_{i+1} \oplus b_{i+1} \oplus b_i). \end{aligned} \tag{11}$$

Outputs are again formed only when carries are present. That is, for a positive output w_i we need n_i and for a negative output v_i we need m_i.

However, first outputs in the case of a run are never allowed until the last digit of a run. The end of a positive-type run at the ith digit, as explained above, can be detected by $a_{i-1} = 0$, or by the start of the opposite type carry in the next digit, i.e., $m_{i-1} = 1$. Similarly, the end of the first negative-type run at the ith digit can be detected by $a_i = 0$ or n_{i-1}. Outputs are therefore given by

$$\begin{aligned} w_i &= n_i(a_{i-1} + m_{i-1}), \\ v_i &= m_i(a_{i-1}' + n_{i-1}). \end{aligned} \tag{12}$$

Note that the logic of (11) and (12) also allows a proper output at the most significant mismatch which is not the start of a run. Such a scheme will give an approximate difference when the outputs are fed into an analog subtracter of the type shown in Fig. 2. This difference will always be within a factor of two of the exact difference, and the difficulties inherent in analog subtraction when a run occurs in the original input numbers, as explained in Section 3.1, do not appear. The following examples showing formation of the most significant outputs may help to clarify the process:

A	1 0 0 1 0	A	1 0 0 1 0	A	1 0 0 0 0	A	1 0 0 1 1
B	0 1 1 1 0	B	0 1 1 0 0	B	0 1 1 0 0	B	0 0 1 1 1
N	1 1 1 1 1	N	1 1 1 1 1	N	1 1 1 1 1	N	1 1 1 1 1
M	0 0 0 0 0	M	0 0 0 0 1	M	0 0 0 1 1	M	0 1 1 1 1
W	0 0 1 1 1	W	0 0 1 1 1	W	0 0 1 1 1	W	1 1 1 1 1
V	0 0 0 0 0	V	0 0 0 0 1	V	0 0 0 1 1	V	0 1 1 1 1

Fig. 6 shows the circuit for the ith digit of this approximate proportional binary-binary comparator. Note that it is possible in this scheme to have outputs of both polarities appearing together. This may also lead to difficulties in the analog subtraction of the W and V numbers. A simple way to avoid this is to allow outputs only when there is exactly one carry present and to inhibit all outputs when both carries are present. The output expressions given by (12) should then be modified as follows:

$$\begin{aligned} w_i &= n_i m_i'(a_{i-1} + m_{i-1}), \\ v_i &= m_i n_i'(a_{i-1}' + n_{i-1}). \end{aligned} \tag{13}$$

For this type of output the circuit of Fig. 6 should be modified by adding the dotted-line inputs to the output gates. This modified scheme still gives an approximate difference within a factor two as determined

by (10). It also avoids subtracting digits of equal weight in an analog subtracter. In fact, if an analog difference is not required, the numbers W and V can be used directly as the approximate digital difference.

4.2 *Approximate Binary-Gray Comparator*

Difficulties of using the binary code for an input number which is rapidly changing were pointed out in Section 2.2. These difficulties can be avoided by translating the varying number into the Gray code, as was demonstrated in the synthesis of exact comparators. Again we assume only the input B to be rapidly changing.

Examination of the logic synthesized for the binary-binary comparator of Section 4.1 shows that digits of the number B appear only in the expressions for the carries and that the only combination of these digits is $b_i \oplus b_{i+1}$. From (3), this is exactly the expression used for translating from binary to Gray. By direct substitution, the expressions for the carries therefore become

$$\begin{aligned} n_i &= n_{i+1} + a_i(a_{i+1} \oplus g_i)' \\ m_i &= m_{i+1} + a_i'(a_{i+1} \oplus g_i), \end{aligned} \tag{14}$$

and the equations for the outputs are not altered.

Using (12) for the outputs and (14) for the carries results in the approximate binary-Gray comparator circuit shown in Fig. 7. As with the

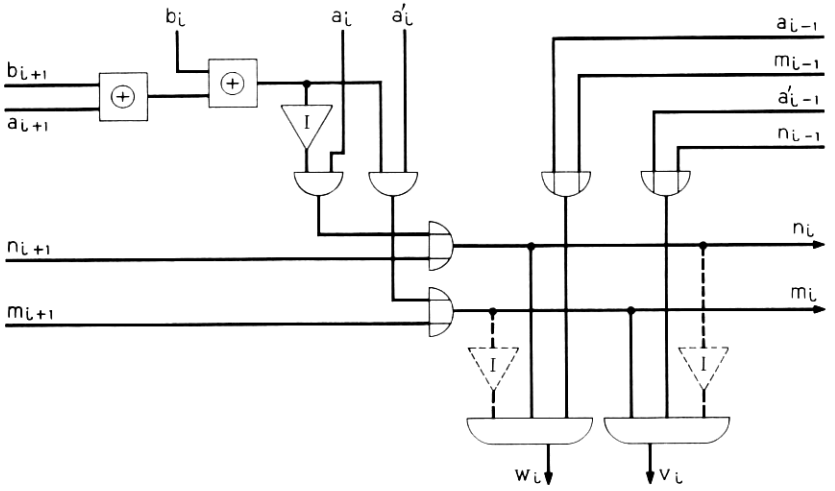

Fig. 6 — Approximate binary-binary comparator.

binary-binary comparator of the previous section, we can again inhibit all outputs during the presence of both carries by using the dotted connections shown.

V. SIGN-ONLY COMPARATORS

5.1 *Modifications of Approximate Comparators*

The approximate comparators of Section IV were largely the result of simplifying the exact comparators of Section III. In this section we will examine the approximate comparators and attempt to simplify them further. All of the previous schemes gave an output which was proportional to the difference between the two input numbers and also indicated the sign of this difference. If only the sign is required, with no measure of the magnitude of the difference, then it is possible to synthesize simple schemes giving an output on a single lead when one of the input numbers is less than or equal to the other input number, or when one number is greater than the other. No analog conversion is necessary in this case, since the output gives the desired indication directly. However, if the sign-only comparator is used as an error detector in a digital servo it is still necessary to have a linear error signal when close to zero difference.

Considering again the schemes having both inputs in the binary code, it is apparent that the sign of the difference is determined completely

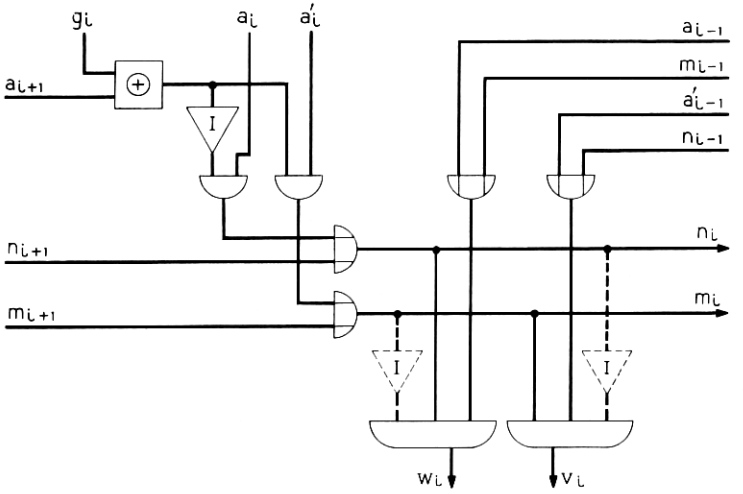

Fig. 7 — Approximate binary-Gray comparator.

by the direction of the most significant mismatch. In the proportional comparators it was desirable to examine the adjacent lower significant digits for the start of a possible run and to prohibit any output until the end of a run. Since the sign is determined only by the sign of the first mismatch it is not necessary to account for any run structure in sign-only comparators and the logic is therefore simpler. Whenever the most significant mismatch is detected, an output should be formed immediately. In addition, we should avoid any contrary action due to opposite polarity mismatches in the lower significant digits. This function can be performed by carries propagating toward the lower significant digits.

These ideas can be illustrated by an example with both inputs in the binary code. Let the output of the ith digit, designated u_i, be zero for all digits which precede the first mismatch and also be zero during and after the first mismatch if it is of the form $a_i'b_i = 1$. Let $u_i = 1$ for the first mismatch if it is of the form $a_ib_i' = 1$. Then an OR over all u_i will provide the proper output:

$$\begin{aligned} \underset{0 \leqq i \leqq m}{\text{OR}} \ (u_i) &= 1 \quad \text{if} \quad A > B \\ &= 0 \quad \text{if} \quad A \leqq B. \end{aligned} \tag{15}$$

Each u_i is determined by the carries present in that digit. A positive carry n_i should be formed for a positive mismatch, i.e., $a_ib_i' = 1$; and a negative carry m_i formed for a negative mismatch, i.e., $a_i'b_i = 1$. If these carries are allowed to propagate through lower significant digits and an output is formed in the ith digit only if a positive carry is present but not a negative carry, then the comparator output given by (15) will result. The expressions for the carries in the ith digit and the output at the ith digit are

$$\begin{aligned} u_i &= n_im_i', \\ n_i &= n_{i+1} + a_ib_i', \\ m_i &= m_{i+1} + a_i'b_i \,. \end{aligned} \tag{16}$$

The circuit for this logic is very simple, requiring three AND gates and two OR gates, each with two inputs for each digit.

A similar scheme when one of the input numbers is in the Gray code can be obtained from the same type of reasoning by using the carries of (14) in Section 4.2. Again, the output in the ith digit is $u_i = n_im_i'$. The circuit for this scheme is given in Fig. 8.

An improvement of the logic used in Fig. 8 follows from a close ex-

amination of the carries n_i and m_i. The functions of these carries in this sign-only comparator can be summarized:

i. For all initial match digits there are no carries and therefore no outputs.

ii. Only one carry is present at the *first* mismatch. This carry determines the output.

iii. If an opposite polarity mismatch occurs *after* the first mismatch a second carry is formed. This second carry inhibits all outputs.

Evidently then, one carry has been used to permit outputs and the other carry to inhibit outputs. These two operations could be performed equally well by one carry if the output function were properly chosen. When a positive mismatch occurs first at the ith digit, we require a positive output, i.e., $u_i = 1$. From the positive carry of Section 4.2, a positive mismatch at the ith digit is detected by $a_i(a_{i+1} \oplus g_i)' = 1$; however, this output should be inhibited if a previous negative mismatch has occurred. Therefore, we could use the negative carry of Section 4.2

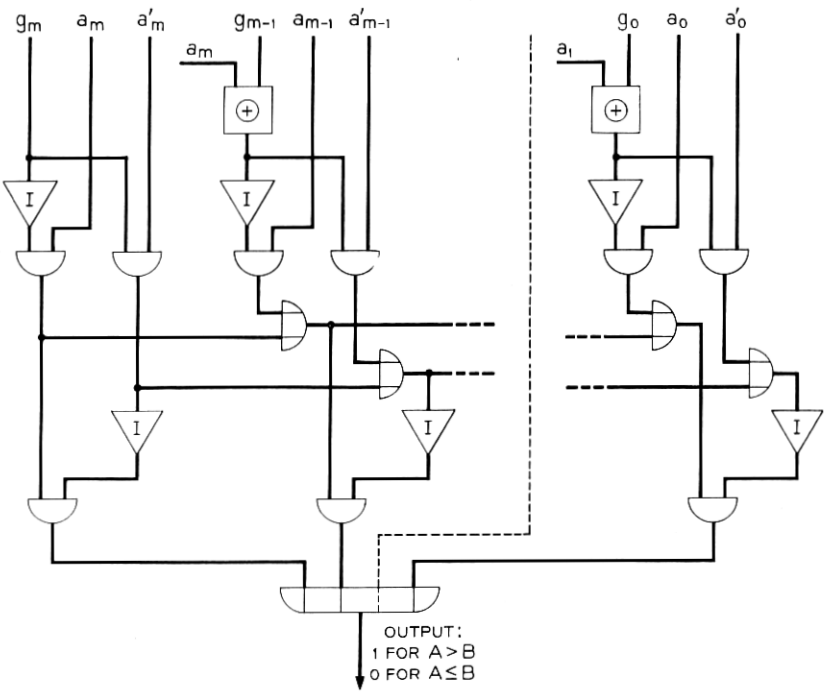

Fig. 8 — Sign-only binary-Gray comparator.

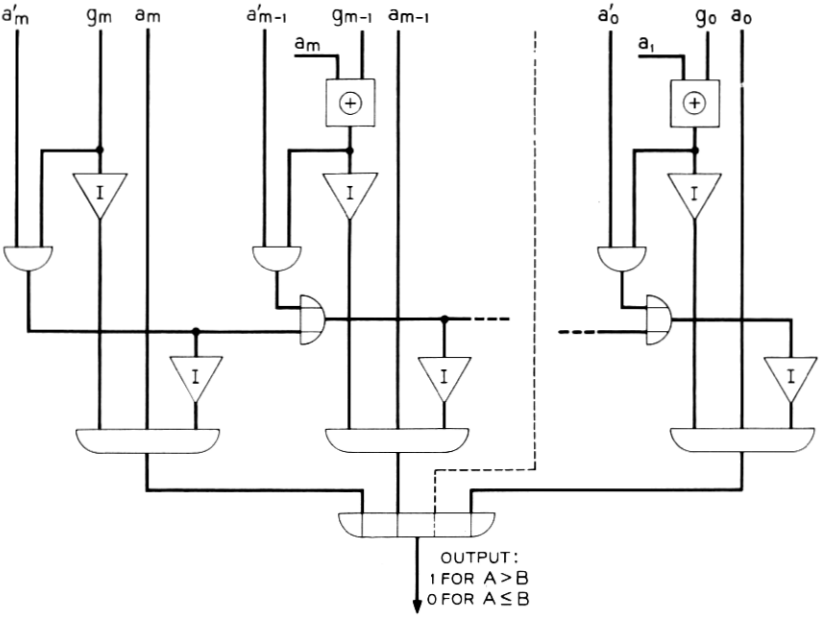

Fig. 9 — Alternate sign-only binary-Gray comparator.

as an inhibit function on u_i. The logic for this scheme then appears as

$$\begin{aligned} u_i &= a_i(a_{i+1} \oplus g_i)'m_i', \\ m_i &= m_{i+1} + a_i'(a_{i+1} \oplus g_i). \end{aligned} \tag{17}$$

These equations, together with (15), have the circuit shown in Fig. 9.

5.2 *Other Sign-Only Comparators**

Assume that the first mismatch occurs between a_j and b_j; that is, all the digits a_i and b_i for $i > j$ match in corresponding places. If $A > B$, then this first mismatch will be of the form $a_j = 1$ and $b_j = 0$. Therefore

$$a_j b_j' = 1, \quad \text{or} \quad a_j + b_j' = 1, \quad \text{if } A > B. \tag{18}$$

Similarly, if $A < B$, then this first mismatch will be of the form $a_j = 0$ and $b_j = 1$. Therefore

$$a_j b_j' = 0, \quad \text{or} \quad a_j + b_j' = 0, \quad \text{if } A < B. \tag{19}$$

* The comparators derived in this section were previously obtained by Ketchledge[2] from other considerations.

For all digits of higher significance, that is, for all $i > j$, we have

$$a_i b_i + a_i' b_i' = 1.$$

This is equivalent to

$$(a_i + b_i')(a_i b_i')' = 1. \tag{20}$$

Since (20) holds for all $i > j$ up to m, we can write

$$(a_m + b_m')(a_{m-1} + b_{m-1}') \cdots (a_{i+1} + b_{i+1}')(a_i + b_i') = 1 \tag{21}$$

and

$$a_m b_m' + a_{m-1} b_{m-1} + \cdots + a_{i+1} b_{i+1}' + a_i b_i' = 0. \tag{22}$$

Now consider the function

$$\Phi_k = (a_k + b_k') + a_m b_m' + a_{m-1} b_{m-1}' + \cdots + a_{k+1} b_{k+1}' \tag{23}$$

for all possible values of k for the conditions $A = B$, $A > B$, and $A < B$.

Then, if $A = B$, from (21), $\Phi_k = 1$ for all k. That is, the digits match in each place so that (20) is 1 for all i.

If $A > B$ from (21) and (22), $\Phi_k = 1$ for all k. That is, the first mismatch will be of the form $a_j b_j' = 1$, by (18). Therefore, $\Phi_j = 1$ and, since the digits match for all $k > j$, we have $a_k = b_k$ or $a_k + b_k' = 1$, which is the first term in Φ_k. Also, Φ_k for all $k < j$ will include the term $a_j b_j' = 1$ as an OR term and will therefore be 1.

If $A < B$ from equations (21) and (22), $\Phi_k = 0$ for $k = j$. That is,

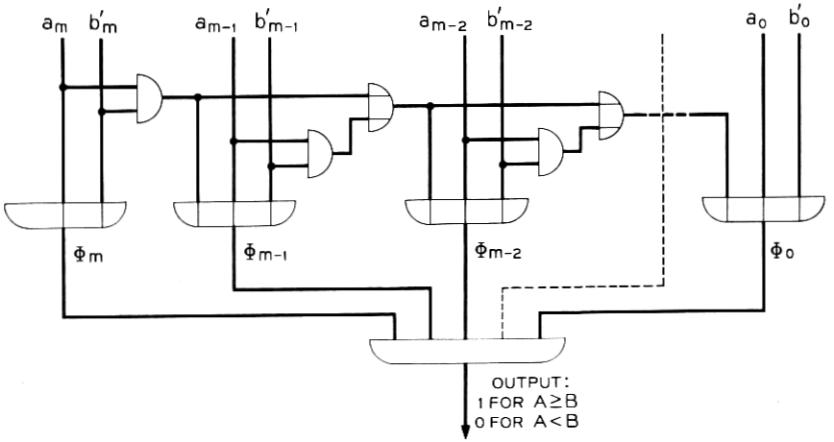

Fig. 10 — Sign-only binary-binary comparator.

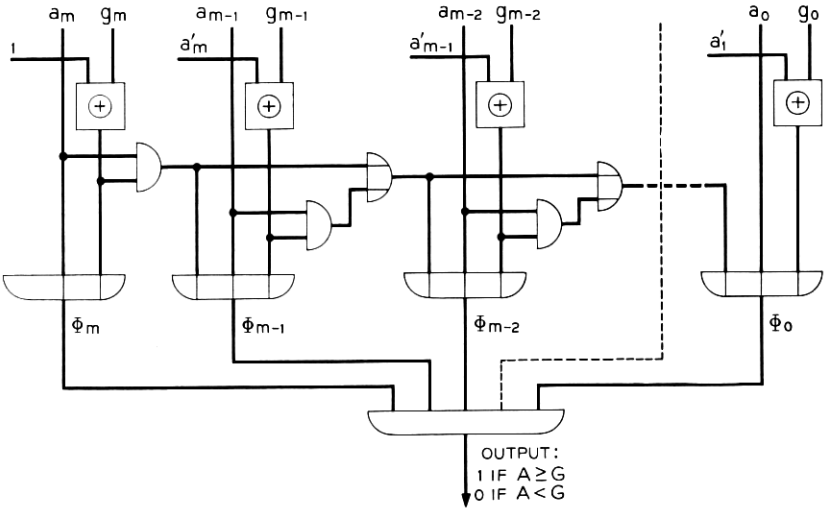

Fig. 11 — Sign-only binary-Gray comparator.

from (19), $a_i + b_j' = 0$, and, from (22), $a_m b_m' + \cdots + a_{j+1} b_{j+1}' = 0$ if the first mismatch occurs in the jth digit.

If Φ_k is generated in each digit we then have a function which is always 1 if $A \geqq B$ and which will be 0 for at least one digit if $A < B$. Therefore, the desired sign detector can be obtained by forming the AND over all the Φ_k :

$$\begin{aligned} \underset{m \geqq k \geqq 0}{\text{AND}} \ (\Phi_k) &= 1 \quad \text{for} \quad A \geqq B \\ &= 0 \quad \text{for} \quad A < B. \end{aligned} \tag{24}$$

Equations (23) and (24) are implemented in the circuit shown in Fig. 10. As in the previous developments, it is desirable to modify this scheme to operate with one of the input numbers in the Gray code. To transform b_i to g_i, we note that, if the first mismatch occurs in the jth digit, then all the preceding digits match. Thus, for $i > j$

$$b_i = g_i \oplus b_{i+1} = g_i \oplus a_{i+1}$$

and

$$b_i' = g_i \oplus a_{i+1}'. \tag{25}$$

Substituting in (23) gives

$$\begin{aligned} \Phi_k = {} & [a_k + (g_k \oplus a_{k+1}')] + a_m g_m' \\ & + a_{m-1}(g_{m-1} \oplus a_m') + \cdots + a_{k+1}(g_{k+1} \oplus a_{k+2}'). \end{aligned} \tag{26}$$

As previously, if $A = G$ in magnitude, then either $a_k = 1$ and $g_k \oplus a_{k+1}' = 0$ or $a_k = 0$ and $g_k \oplus a_{k+1}' = 1$ for $0 \leqq k \leqq m$. That is, $\Phi_k = 1$ for all k. If $A > G$ and the first mismatch occurs for $i = j$, then $\Phi_k = 1$ for $k > j$, $a_j = 1$ and $g_j + a_{j+1}' = 1$. Therefore, $\Phi_j = 1$, since the term $a_j(g_j \oplus a_{j+1}') = 1$ for all k if $A > G$. If $A < G$ and the first mismatch occurs for $i = j$, then, by transforming (19), we have $a_j + (g_j \oplus a_{j+1}') = 0$, and therefore $\Phi_j = 0$.

By forming the AND over all k of the function given in (26) we have the desired sign detection:

$$\begin{aligned} \underset{m \geqq k \geqq 0}{\text{AND}} \ (\Phi_k) &= 1, \quad \text{for} \quad A \geqq G \\ &= 0, \quad \text{for} \quad A < G. \end{aligned} \tag{27}$$

A circuit performing the operations of (26) and (27) is shown in Fig. 11. The circuits of Figs. 10 and 11 could be modified by using the duals of the above expressions and changing the output gate to an OR over all k. The individual digit functions Φ_k would then be changed to AND's of each of the dual terms.

REFERENCES

1. Hoover, C. W., Jr., Staehler, R. E. and Ketchledge, R. W., Fundamental Concepts in the Design of the Flying Spot Store, B.S.T.J., **37**, September 1958, p. 1161.
2. Ketchledge, R. W., this issue, pp. 1–17.
3. Richards, R. K., *Arithmetic Operations in Digital Computers*, D. Van Nostrand Co., New York, 1955, Ch. 9 and 10.
4. Keister, W., Ritchie, A. E. and Washburn, S. H., *The Design of Switching Circuits*, D. Van Nostrand Co., New York, 1951, Ch. 5.
5. Foss, F. A., The Use of a Reflected Code in Digital Control Systems, Trans. I.R.E., **EC-3**, December 1954, p. 1.
6. Gray, F., Patent No. 2,632,058, March 17, 1953.
7. Gilbert, E. N., Gray Codes and Paths on the n-Cube, B.S.T.J., **37**, May 1958, p. 815.
8. Flores, I., Reflected Number Systems, Trans. I.R.E., **EC-5**, June 1956, p. 79.