G. G. Langdon, Jr.
C. K. Tang

Concurrent Error Detection for Group Look-ahead Binary Adders

Abstract: This paper presents an evaluation of the relative merits of two schemes for performing concurrent error detection in group look-ahead adders. One of the schemes is a residue mod 3 check and the other is a parity prediction check. The Boolean statements that define the operation of group look-ahead adders, concurrent error detection and the Boolean difference serve as background for interpreting the results of the study. The Boolean difference is a tool for calculating the "coverage" of elements in a logical network by error-checking schemes. Some weaknesses in prior studies of coverage calculation are brought to light. Tables showing the number of circuit elements in the various portions of adder and error-checking circuits are given. It is shown that the residue mod 3 check adder is not economical unless the addition operands are already provided with the mod 3 check bits. Thus, a worthwhile comparison of the checking schemes should not proceed without considering the overall data flow checking strategy. In machine organizations with three or more data transfer checks, the parity-checked adder seems to offer a cost advantage.

1. Introduction

Most medium- and high-performance computers have a parallel binary adder with either carry look-ahead or some other means of speeding up operation of the arithmetic unit. It is desirable to incorporate error detection methods in the design of this type of adder. The purpose of the study reported in this paper is to compare the merits of two methods of checking a 16-bit group look-ahead binary adder: 1) residue mod 3, and 2) parity prediction plus additional checking. Since we are considering error detection in high-speed adders, we assume that the data flow outside the adder also has means to check data transfers (such as a parity bit), and that there are no unnecessary delays from the time data enters the adder until the result reaches its destination in the CPU.

2. The carry look-ahead adder

The principle of carry look-ahead is well known.[1-3] The notation used in Chapter 6 of the book* by F. F. Sellers, Jr., M. Y. Hsiao and L. W. Bearnson[2] is followed in the discussion. Two operands A and B are added to give sum S. Sum bit S_i depends on the carry from the previous stage† C_{i-1}, as well as on a_i and b_i:

$$S_i = a_i \oplus b_i \oplus C_{i-1}, \tag{1}$$

$$C_i = a_i b_i + a_i C_{i-1} + b_i C_{i-1}. \tag{2}$$

In the Boolean equations, "$+$" denotes OR, "$\cdot$" or juxtaposition denotes AND, "$\oplus$" denotes EXCLUSIVE-OR, and the overbar denotes negation.

In look-ahead adders, the *half-adder* functions $H_i = a_i \oplus b_i$, $G_i = a_i b_i$ and $T_i = a_i + b_i$ are called the *half sum*, *generate* and *transmit* signals, respectively.

The carry-signal equations for a 16-bit adder with full look-ahead are stated as

$$C_{-1} = C_{\text{in}}, \tag{3}$$

$$C_0 = G_0 + C_{\text{in}} T_0, \tag{4}$$

$$C_1 = G_1 + G_0 T_1 + C_{\text{in}} T_0 T_1, \tag{5}$$

$$C_{15} = G_{15} + G_{14} T_{15} + G_{13} T_{14} T_{15} + G_{12} T_{13} T_{14} T_{15} + \cdots + C_{\text{in}} T_0 T_1 \cdots T_{15}. \tag{6}$$

In an adder with full look-ahead, the sum is normally generated by the equation

$$S_i = H_i \oplus C_{i-1}. \tag{7}$$

In considering the ripple-carry adder, it is well known that an error in the carry-in to position i may cause a sequence (or burst) of sum bits to be in error, while the numeric value of the error, the *error value*, will be only $\pm 2^i$ (where 2^i is the numeric value of bit position i). The following theorem is taken from Sellers.[2]

The authors are located at the IBM Systems Development Division Laboratory in Endicott, New York.

* Since this work is referenced a number of times in this paper, it will subsequently be referred to by Sellers' name alone.

† The convention in the literature is that the adder position subscripts run low-order to high. We accept this with some reluctance. Bit positions in an n-bit register are usually labeled position 0 for the high-order and position $n-1$ for the low order. As a result, in actual practice, the low-order position of a 16-bit adder is position 15; i.e., the subscripts run high-order to low. Note also that the carry subscript is one less than the adder stage into which it carries.

Theorem 1: In ripple-carry adders, if an error in C_n causes an error in C_{n+q}, then C_n also causes an error in $C_{n+1}, C_{n+2}, \cdots, C_{n+q-1}$.

In an adder with full carry look-ahead, an error in some C_{i-1} affects only bit position i. However, Sellers (Ref. 2, p. 101) has proven that a failure in either G_i or T_i may cause a sequence of sum bits to be in error. Again, the error value will be $\pm 2^i$. The following theorem, also taken from Sellers' work, is valid for full look-ahead adders under certain conditions. Gaddess[4] claims a similar result.

Theorem 2: In full look-ahead adders, if an error in T_n or G_n causes an error in C_{n+q}, then T_n or G_n will also cause errors in $C_{n+1}, C_{n+2}, \cdots, C_{n+q-1}$.

This theorem is invalid if signals T_n or G_n are duplicated to increase the fan-out capability. For example, consider adding 8-bit numbers. Suppose a fan-out capability of five is the limit. Signal T_1 must be duplicated into T_1^a and T_1^b. If T_1^a feeds bit positions 1, 2 and 3, and T_1^b feeds positions 4, 5 and 6, then the theorem obviously does not hold if T_1^b fails and causes C_5 to be in error, since C_1, C_2 and C_3 will still be correct. Interestingly enough, in this instance the error value is $\pm 2^5$; when T_1^a fails, the theorem holds, but the error value may be $\pm 7 \times 2^1$.

In situations where the adder must operate at high speed, the ripple-carry adder is unsuitable because it is too slow. As early as 1959, Lourie et al.[5] stated that a parallel accumulator without speed-up techniques is an extremely wasteful device. However, an adder with full look-ahead is also unsuitable for technical reasons. Equation (6) requires AND gates with fan-in capabilities of up to 15 inputs and an OR gate with a fan-in of 16. Furthermore, signals G_0 and T_0 require a fan-out capability of 15. It is therefore customary in practice to factor Eqs. (3) to (6) systematically. This is generally done (See Refs. 1, 3, 6) by forming groups of from 4 to 6 consecutive bit positions and forming group "generate" and "transmit" signals analogous to G_i and T_i. For the 16-bit adder divided into four-bit groups, for example, the following equations are typical:

$$G_{12-15} = G_{12}T_{13}T_{14}T_{15} + G_{13}T_{14}T_{15} + G_{14}T_{15} + G_{15}, \tag{8}$$

$$T_{12-15} = T_{12}T_{13}T_{14}T_{15}. \tag{9}$$

With the group generate and transmit signals available, the group carry-in signals may be designed by using ripple-carry techniques as derived below:

$$C_{\text{in}\,12-15} = G_{8-11} + C_{\text{in}\,8-11}T_{8-11}, \tag{10}$$

$$C_{\text{in}\,8-11} = G_{4-7} + C_{\text{in}\,4-7}T_{4-7}, \tag{11}$$

$$C_{\text{in}\,4-7} = G_{0-3} + C_{\text{in}}T_{0-3}. \tag{12}$$

However, it may be more advantageous to use look-ahead on the group signals:

$$C_{\text{in}\,12-15} = G_{8-11} + G_{4-7}T_{8-11} + G_{0-3}T_{4-7}T_{8-11} + C_{\text{in}}T_{0-3}T_{4-7}T_{8-11}, \tag{13a}$$

$$C_{\text{in}\,8-11} = G_{4-7} + G_{0-3}T_{4-7} + C_{\text{in}}T_{0-3}T_{4-7}, \tag{13b}$$

$$C_{\text{in}\,4-7} = G_{0-3} + C_{\text{in}}T_{0-3}. \tag{13c}$$

Since the high-order carry-out signal (C_{15}) is an adder output (like the sum bits), it may be obtained by a ripple as in Eqs. (10) to (12):

$$C_{15} = C_{\text{in}\,12-15}T_{12-15} + G_{12-15}. \tag{13d}$$

Similarly, once the carry-in signals *between* groups are formed using either the ripple or look-ahead technique, the bit carries *within* the group may also be designed with either the ripple principle or the look-ahead principle. As an example, for look-ahead within group 4–7, the following equations are used:

$$C_6 = C_{\text{in}\,4-7}T_4T_5T_6 + G_4T_5T_6 + G_5T_6 + G_6, \tag{14a}$$

$$C_5 = C_{\text{in}\,4-7}T_4T_5 + G_4T_5 + G_5, \tag{14b}$$

$$C_4 = C_{\text{in}\,4-7}T_4 + G_4, \tag{14c}$$

$$C_3 = C_{\text{in}\,4-7}. \tag{14d}$$

With the bit carry-in explicitly formed, the sum bit is formed as in Eq. (7). But this adds an unnecessary delay to the critical signal $C_{\text{in}\,4-7}$ for sum bits 5, 6 and 7. In the STRETCH computer adder (Ref. 6, p. 445, Fig. 8), S_7 is formed as:

$$S_7 = C_{\text{in}\,4-7}\bar{H}_7(T_4T_5T_6) + \bar{H}_7(G_4T_5T_6 + G_5T_6 + G_6) + \overline{(T_4T_5T_6)}\,\overline{(G_4T_5T_6 + G_5T_6 + G_6)}H_7 + \bar{C}_{\text{in}\,4-7}H_7\overline{(G_4T_5T_6 + G_5T_6 + G_6)}, \tag{15a}$$

$$S_6 = C_{\text{in}\,4-7}\bar{H}_6(T_4T_5) + \bar{H}_6(G_4T_5 + G_5) + \overline{(T_4T_5)}\,\overline{(G_4T_5 + G_5)}H_6 + \bar{C}_{\text{in}\,4-7}H_6\overline{(G_4T_5 + G_5)}, \tag{15b}$$

$$S_5 = C_{\text{in}\,4-7}\bar{H}_5T_4 + \bar{H}_5G_4 + \bar{T}_4\bar{G}_4H_5 + \bar{C}_{\text{in}\,4-7}H_5\bar{G}_4, \tag{15c}$$

$$S_4 = C_{\text{in}\,4-7}\bar{H}_4 + \bar{C}_{\text{in}\,4-7}H_4. \tag{15d}$$

The formation of the sum in this manner utilizes functions called the "group internal auxiliary functions." For bit 7, these functions are defined below:

$$\text{Bit 7 group internal propagate} = T_4T_5T_6, \tag{16}$$

$$\text{Bit 7 group internal generate} = G_4T_5T_6 + G_5T_6 + G_6. \tag{17}$$

The look-ahead adder considered in this study is 16 bits wide, has four groups of four bits each, and has full look-ahead between *and* within each group. This represents a reasonable compromise among speed, fan-in, and fan-out capabilities. Thus Eqs. (8), (9), (13), (15), (16) and (17) are representative of this adder. Figure 1 shows the adder block diagram.

3. Concurrent error detection

Concurrent error detection means that adder errors are detected by checking circuits during normal operations within one adder cycle of their occurrence. Gate failures will be assumed to occur as the output either "stuck-at-0" (s-a-0) or "stuck-at-1" (s-a-1). Actually, this assumption is more general than it might first appear. The coverage calculation for the error detection schemes studied here must satisfy the *concurrent check assumption;* that is, under a single gate failure of a completely covered gate, for any given input state either the network output state is correct or the checking circuit shows that a failure occurred. Thus all gate input failures and grounded interconnections, many open interconnections, and even some connections shorted together, appear under the concurrent check assumption and a particular input state as a gate s-a-0 or s-a-1. A property of the concurrent check assumption is that the coverage of a gate may be incomplete. The following definitions on coverage are an attempt to formalize the preceding intuitive concepts. We assume s-a-1 or s-a-0 gate failures may occur.

Definition 1:* A gate is (fully or completely) *covered,* if: a) under all valid input combinations or states, gate failures result in either the correct network output (i.e., the fault is *masked*) or an error indication (i.e., the fault is detected), and b) for at least one input state (not necessarily a valid input), the failure causes an error indication. (An example of an invalid input for the parity-checked adder would be an operand with incorrect parity.)

The presence of condition b) in the definition is motivated by the fact that if a gate in the checking circuit fails without ever indicating an error, the gate cannot properly be considered covered. It may then be said that a gate failure is *partially covered* if there is some valid input state for which the failure is detected, and another valid input state for which the network output is wrong with no error indication given. If a gate is neither fully nor partially covered it is *uncovered.*

In studying concurrent checking techniques it becomes apparent that the coverage calculation is a nontrivial exercise. When a gate G_i in a logic network fails, it is important to focus upon the input conditions (set of input states) for which the fault is not masked at the network outputs. In order for G_i to be covered, the concurrent error detection circuitry must signal an error for these "fault unmasking input states." To assist in the determination of the fault unmasking input states, Sellers[2] describes the Boolean difference operation $dF(X)/dx_i$ of Boolean function $F(X)$ with respect to input signal x_i, where X is a Boolean vector $x_1, x_2, \cdots, x_n$. Each value of X is an input state.

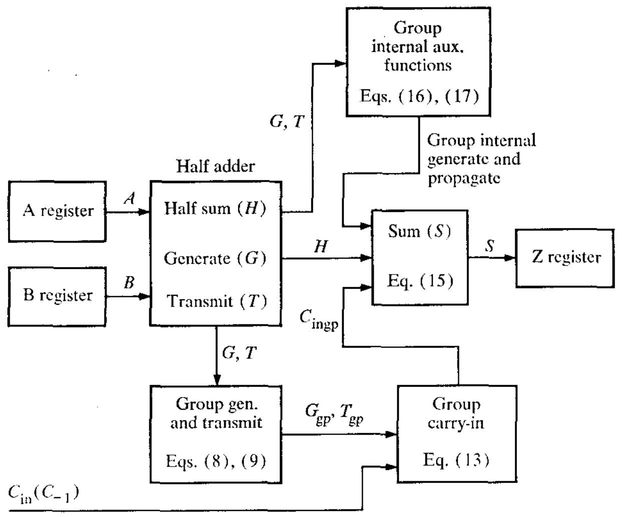


Figure 1 Block diagram of a group look-ahead adder.

Definition 2: The Boolean difference $dF(X)/dx_i$ of function F with respect to variable x_i is $F(x_1, \cdots, 1, \cdots, x_n) \oplus F(x_1, \cdots, 0, \cdots, x_n)$. That is, $dF(X)/dx_i$ is the EXCLUSIVE-OR of function F with x_i set to 1 and function F with x_i set to 0. In any case, the result $dF(X)/dx_i$ is a Boolean expression that describes the conditions under which a fault in the signal x_i changes the logic network output F.*

The Boolean difference, as defined above, concerns the effect of failed inputs on the output F of a logic network. Here, as in Chapter 6 of Sellers' work, we apply the technique to failed gates inside a multiple-output logic network. Consider the coverage calculation for gate i in Fig. 2.

In calculating the Boolean difference dF_i/dg_i, i.e., the input states for which a failure in gate i changes output F_i, the gate output signal g_i must be treated as an independent variable. The gate output is now a primary input to the circuit, and F_i becomes a function of $n + 1$ variables $F_i(x_1, \cdots, x_n, g_i)$. Now $dF_i/dg_i = F_i(x_1, \cdots, x_n, 1) \oplus F_i(x_1, \cdots, x_n, 0)$. The Boolean difference is seen to be

* One of the authors was influenced in this definition by discussions with W. C. Bouricius, W. C. Carter and D. C. Jessup, Jr., of the IBM Research Division. Another influence was Sellers (Ref. 2, p. 211).

* Input states for which the Boolean expression dF/dx_i assumes the value 1 are the fault unmasking input states.

the EXCLUSIVE-OR of a) F_j with g_i s-a-1 and b) F_j with g_i s-a-0. It is our opinion that treating gate outputs as independent variables is important as a means of addressing the problem of independent failures in the complement of a signal. Suppose, for example, that gate k inverts the output of gate i. When analyzing a s-a-1 failure in gate k, we cannot assume that the logical complement, i.e., the output of gate i, is s-a-0. This problem is avoided by treating the gate output g_k as an independent input variable.

Since g_i is itself a function of X, it is now possible to determine whether $g_i(X)$ must be s-a-0 or s-a-1 for certain fault-unmasking input states. The expression $(dF_j/dg_i)\cdot g_i(X)$ gives the conditions for which g_i s-a-0 causes F_j to be in error. Similarly, $(dF_j/dg_i)\cdot \bar{g}_i(X)$ describes the input conditions for which g_i s-a-1 affects F_j.[8] As an example, consider the circuit of Fig. 3.

In the figure $F(A, B) = A \oplus B$. Consider a failure in Gate 1. Since $F(A, B, 1) = A + B$ and $F(A, B, 0) = 0$, we have $dF/dg_1 = (A + B) \oplus 0 = A + B$. For g_1 s-a-0, input $(A + B)\cdot(\overline{AB}) = A \oplus B$ causes F to be in error; for g_1 s-a-1, input $(A + B)\cdot(\overline{\overline{AB}}) = AB$ causes F to be in error. (We note here the possibility that a gate may be covered against s-a-0 but uncovered against s-a-1.)

To calculate the coverage of a checking scheme for an adder, it is necessary to determine the effect of each gate failure on all the sum bits plus the check circuits. In the case of an independent mod 3 check, a gate i is covered if the error value of each possible fault unmasking input state for the gate is not divisible by 3. To accomplish this, the Boolean difference alone is not sufficient because no information is given about whether a failure causes an output F to go erroneously to 1 or to 0 under the given input state.

An example may illustrate this point. Suppose that, under a particular input state, a failure in gate i causes an error in sum bits S_k and S_{k+1}. If S_k and S_{k+1} are changed in the same direction, the error value is $\pm 3 \times 2^k$; if S_k and S_{k+1} are changed in opposite directions, the error value is $\pm 2^k$. In the latter case the fault is caught under a residue mod 3 check. In the former case, it goes undetected. Therefore, knowledge of the Boolean difference alone is not sufficient to calculate the coverage.

The definition of the Boolean difference may be modified to remedy this defect. In this context, let the positive Boolean difference $dF(X)^+/dx_i$ denote the fault unmasking input states for which $F(X)$ changes in the same direction as x_i; i.e., dF^+/dx_i are the conditions when x_i changes from 1 to 0 (or 0 to 1) then F must also change from 1 to 0 (or 0 to 1), respectively. Similarly, the negative Boolean difference $dF(X)^-/dx_i$ denotes the fault unmasking conditions for which an error in x_i causes $F(X)$ to change in the opposite direction. The positive and negative (polarized) Boolean differences apply, with

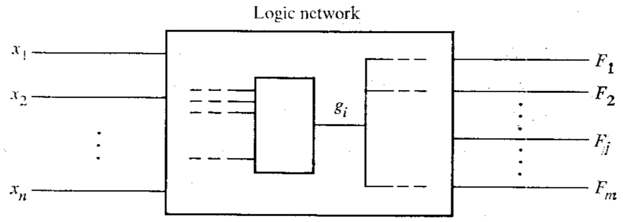
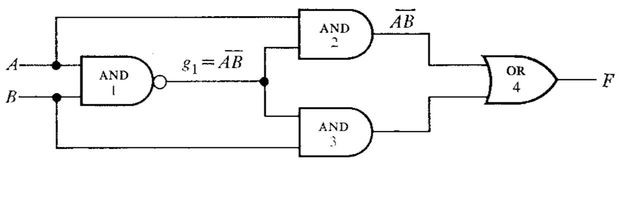


Figure 2 Coverage calculation for gate 1.

Figure 3 Example circuit, $F(A, B) = A \oplus B$.



appropriate modifications, to gate outputs g_i within a network. The Boolean expression of the polarized Boolean difference can easily be obtained by its definition. A positive Boolean difference implies that when the signal under consideration x_i is at the 0 state, the function $F(X)$ should also be at the 0 state. The input states that satisfy the above condition can be written as the Boolean expression $\bar{F}(x_1, \cdots, 0, \cdots, x_n)$. A positive Boolean difference also implies that when x_i is at the 1 state, the function $F(X)$ should stay at the 1 state. The input states that satisfy this condition are, in terms of a Boolean expression, $F(x_1, \cdots, 1, \cdots, x_n)$. The positive Boolean difference is defined as the input states that satisfy both of the above conditions. This means that when the two Boolean expressions shown above are "ANDed" together, the positive Boolean difference results. The negative Boolean difference is similarly defined.

Definition 3: The polarized Boolean differences are given by

$$\frac{dF(X)^+}{dx_i} = \bar{F}(x_1, \cdots, 0, \cdots, x_n) \cdot F(x_1, \cdots, 1, \cdots, x_n),$$

$$\frac{dF(X)^-}{dx_i} = F(x_1, \cdots, 0, \cdots, x_n) \cdot \bar{F}(x_1, \cdots, 1, \cdots, x_n).$$

This definition leads to the following identities:

$$\frac{dF(X)^+}{dx_i} \cdot \frac{dF(X)^-}{dx_i} = 0,$$

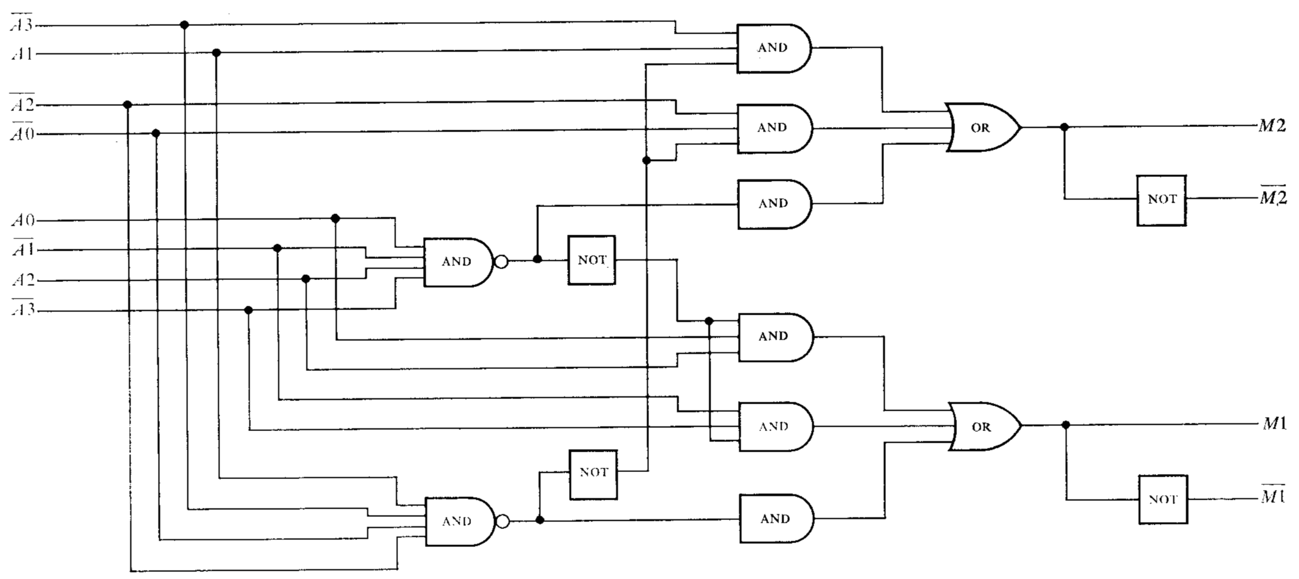


Figure 4 Residue mod 3 of a 4-bit group.

$$\frac{dF(X)^+}{dx_i} + \frac{dF(X)^-}{dx_i} = \frac{dF(X)}{dx_i},$$

$$\frac{d(A + Bx + C\bar{x})^+}{dx} = \bar{A}B\bar{C},$$

where A, B and C are independent of x,

$$\frac{d(A + Bx + C\bar{x})^-}{dx} = \bar{A}\bar{B}C,$$

$$\frac{dF(X)^+}{dx_i} = \frac{dF(X)^-}{d\bar{x}_i}.$$

In the case of an adder checked by the odd parity-check alone, a gate i is covered if each fault-unmasking condition causes the predicted parity bit to be in error. In other words, a gate is covered if each fault-unmasking input state causes an odd number of the output signals (sum-bit and predicted parity) to be affected. In this case, the polarity (positive or negative) of the Boolean difference is not a consideration.

For additional and more theoretical considerations on concurrent error detection, see Carter and Schneider.[7]

4. Mod 3 residue check

Checking addition by a residue check has been discussed by Peterson,[9] Garner[10,11] and Henderson.[12] For the residue arithmetic check using base 3, it is necessary to obtain the residue mod 3 of the operands A and B, respectively, where A mod 3 is the remainder (0, 1 or 2) of A divided by 3. The basis for the check is that the mod 3 sum of A mod 3 and B mod 3 should be equivalent to the residue mod 3 of the sum S of A and B.

The key logic function to this check is the calculation of the residue mod 3 of a binary number. This can be done by means of a "building block" approach, where the residue mod 3 is calculated 4 bits at a time. Sellers[2] presents several circuits for performing this function. Figure 4 presents another such circuit that seems to have a cost advantage.

In this circuit, the outputs M1 and M2 represent an encoding of the residues 0, 1 and 2 into $\overline{\text{M2}}$ $\overline{\text{M1}}$ (00), M2 M1 (11), $\overline{\text{M2}}$ M1 (01) and M2 $\overline{\text{M1}}$ (10), respectively. In turn, the outputs of two of these circuits may be fed to a similar circuit for the residue mod 3 of an eight-bit binary number. In this way a "mod 3 tree" can be built to form the mod 3 residue of any length binary number. This is shown in Fig. 5, where each building block (labeled mod 3) is the circuit of Fig. 4, except that the complement signal lines are not shown.

5. Mod 3 adder coverage

The mod 3 residue check detects all failures for which the difference between the correct answer and incorrect answer (error value) is not a multiple of 3.

Gates that can affect only a single sum-bit are, of course, fully covered. Consider the generate and transmit signals G_i and T_i. If the group generate and transmit signals are formed as in Eqs. (8) and (9), the group internal propagate and generate signals are formed as in Eqs. (16) and (17), Theorem 2 then holds for the adder with group look-ahead.

Gaddess[4] has studied the "parallel" adder in general terms. He makes the claim, using reasoning similar to that of Sellers[2] in Theorem 2, that faults in gates whose

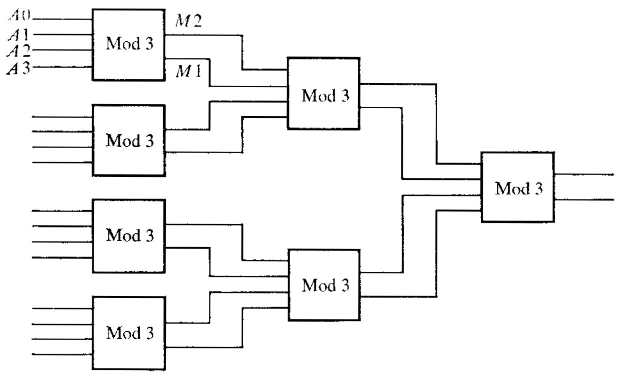


Figure 5 Residue mod 3 of a 16-bit number.

outputs affect more than one carry-in signal will cause an error burst in such a way that the error value is $\pm 2^i$ for some i. The conclusion follows that a residue mod 3 check is sufficient to catch single-gate errors. The following examples demonstrate where this conclusion fails in adders using group look-ahead.

• *Case 1*

Suppose, for example, that signals T_4, T_5 and T_6 of Eqs. (16) and (17) were replaced by H_4, H_5 and H_6 as in Eqs. (16′) and (17′):

$$\text{Bit 7 group internal propagate} = H_4H_5H_6, \qquad (16')$$

$$\text{Bit 7 group internal generate} = G_4H_5H_6+G_5H_6+G_6. \qquad (17')$$

Also suppose that a similar substitution is made for the Bit 6 internal propagate and generate signals H_4H_5 and $G_4H_5 + G_5$, respectively. Logically, the adder behaves the same as before, because, for the purpose of propagating a carry through stage i, either signal T_i or H_i does the job. However, the Boolean difference shows that if G_5 should be 1 but is s-a-0 and $G_4H_6H_7 = 1$, then S_6 and S_7 will both be 1 instead of 0. S_8 will be correct, because signal $C_{\text{in }8-11}$ will correctly be 1 due to the first term of $G_{4-7} = G_4T_5T_6T_7 + G_5T_6T_7 + G_6T_7 + G_7$. For this example, the error value is 3×2^6. The same error value results if $G_5H_7 = 1$, and H_6 should be 1 but is s-a-0. The application of the Boolean difference to this coverage calculation for gate G_5 appears in Appendix A.

• *Case 2*

If the carry is rippled within each group, there is a larger variety of possible error values. For example, if C_5 fails when $H_6H_7 = 1$, an error value of $\pm 3 \times 2^6$ will ensue; if C_4 fails when $H_5H_6H_7 = 1$, an error of $\pm 7 \times 2^5$ will ensue. The reason these error values are not of the form $\pm 2^i$ is that, with signal H_7 in the 1-state, an error in the carry-in to bit position 7 is expected to propagate up to position 8. This is not the case because $C_{\text{in }8-11}$ is independently and correctly generated.

In both Cases 1 and 2, the problem of error values not of the form $\pm 2^i$ may be circumvented by providing look-ahead within the group that utilizes the transmit signal as in Eqs. (16) and (17).

• *Case 3*

For group look-ahead adders that use look-ahead between groups [see Eq. (13)], error values not of the form 2^i are unavoidable when a group carry-in signal fails. For example, assume $H_4H_5H_6H_7 = 1$. Under this condition a fault in $C_{\text{in }4-7}$ can cause error values of $\pm\ 15 \times 2^4$. This error magnitude is not caught by a mod 3 check. Since $C_{\text{in }8-11}$ is independently generated, the carry error does not propagate to position 8 as it should if the error value is to be $\pm 2^4$.

One means of reducing this error value to $\pm 2^4$ is to employ the ripple-carry technique between groups. This technique is accomplished at the cost of increased adder cycle time. Another option is to avoid an error magnitude that is a multiple of 3 by redesigning the group look-ahead so that each group contains an odd number of bits. A third possible method is to duplicate the group carry-in signals and perform a comparison. In a group look-ahead adder that does rippling between groups and look-ahead within groups, no special problem is presented for mod 3 check; however, Cases 2 and 3 indicate combinations that may give error values not of the form $\pm 2^i$.

• *Case 4*

It is noted that in Eq. (15) the complement of the group carry-in signal $\bar{C}_{\text{in }4-7}$ is required. In the STRETCH computer indexing adder, the current-mode circuits had the capability of providing both true and complement outputs, so that both polarities of the group carry-in signals were available. If it is possible for the polarities of the group carry-in signals to fail independently, then a fault in $\bar{C}_{\text{in }4-7}$ can cause many problems. Applying the Boolean difference to $\bar{C}_{\text{in }4-7}$ shows, for example, that with the sum generated as in Eq. (15), an error value of 3×2^4 results if $H_4H_5\bar{H}_6\bar{H}_7 = 1$. One alternative here is to reduce the adder performance and obtain $\bar{C}_{\text{in }4-7}$ by inverting $C_{\text{in }4-7}$. Then, building the duplicate carry-in signal and checking against $\bar{C}_{\text{in }4-7}$ provides the additional coverage needed. A simpler solution might be to invert $C_{\text{in }4-7}$ four times, once for each sum bit, so that a single gate failure involving a $\bar{C}_{\text{in }4-7}$ signal affects only one sum bit.

Note that the types of gate failures exemplified do not give error values of the $\pm 2^i$ variety. Thus, they may also act to the detriment of product (or *An*) code schemes.[13]

6. The parity-checked adder

A description of checking addition by means of a parity check is described by Garner.[10,11] Let P_A denote the odd parity bit of an n-bit word A, and P_B denote the odd parity bit of an n-bit word B; i.e., $P_A = a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1} \oplus 1$ and $P_B = b_0 \oplus b_1 \oplus \cdots \oplus b_{n-1} \oplus 1$. In forming the sum S of binary numbers A and B, let P_C denote the even parity of the carries; i.e., $P_C = C_{\text{in}} \oplus C_0 \oplus \cdots \oplus C_{n-1}$. The basic parity check equation is

$$P_S = P_A \oplus P_B \oplus P_C \oplus 1. \tag{18}$$

The right side of this equation is called the *predicted parity*.

It is known that a failure in the carry circuit will change the sum S and hence P_S. It will, however, also affect P_C the same way so that Eq. (18) is satisfied, and no indication of failure is given. It was this absence of a check on the carry generation process that motivated Garner (Ref. 11, p. 765) to say that the parity check was "useless." It is possible, however, to remedy these deficiencies without abandoning the notion of a parity check. (In fact, Hsiao and Sellers[15] have given an elegant remedy for the ripple-carry adder.)

Let us consider the implementation of Eq. (18) in a look-ahead adder. Once the carries have been formed, only one more level of delay* is needed to calculate the sum S, whereas to calculate P_C, an EXCLUSIVE-OR tree is needed. Thus the predicted parity is available considerably later than the sum; this adversely affects adder cycle time. Ideally, the sum with its predicted parity should be available at the same time so it can be entered into the Z register, freeing the A and B registers for the next adder cycle.

Sellers,[2] and Pitkowsky and Godfrey[16] report high-speed means to perform this parity prediction. Parity prediction is done for the same groups over which look-ahead is performed by using the group carry-in ($C_{\text{in gr}}$) and the group half-adder (H_i, T_i, G_i) signals. If the group input parity signals P_A and P_B are available, the following equation (which is a corrected version of the one derived by Sellers[2]) specifies the predicted group parity PP_{sg} for group 0–3:

$$PP_{sg} = P_A \oplus P_B \oplus G_0 \oplus G_1 \oplus G_2 \oplus (G_0 H_1 \bar{H}_2 + G_1 H_2) \oplus C_{\text{in gr}}(\bar{H}_0 + H_1 \bar{H}_2). \tag{19}$$

However, by a similar derivation procedure, the following equation of Pitkowsky and Godfrey[16] was implemented in the IBM System/360 Model 50 adder and using one less EXCLUSIVE-OR gate. It also does not require the input parity bits.

$$PP_{sg} = (T_0 \oplus T_1) \oplus (T_2 \oplus H_3) \oplus (G_0 H_1 \bar{H}_2 + G_1 H_2) \oplus (C_{\text{in gr}} \bar{H}_0 + C_{\text{in gr}} H_1 \bar{H}_2). \tag{20}$$

* AND gates followed by a DOT-OR is considered "one level."

In the group look-ahead adder, Eq. (20) provides coverage against the group internal carry look-ahead and sum bit generation circuits, because only one sum bit can be affected by these faults. However, no detection of errors in the group look-ahead signals is provided, and the coverage against faults in the half-adder signals is incomplete. To increase the coverage, two supplementary checks are provided[2,16]—the half-sum check and the carry check.

- *The supplemental checks*

Let $P_H = H_0 \oplus H_1 \oplus \cdots \oplus H_{n-1}$. The *half-sum check* equation that must be satisfied is

$$P_H = P_A \oplus P_B. \tag{21}$$

The half-sum check is used to check the validity of the input parity and also to check the half-sum signals H_i.

The *duplicate-carry check* can be implemented using the high-order group internal generate and propagate signals for each group. For example, to check signal $C_{\text{in } 8-11}$ the following signal may be used:

$$\text{Duplicate } C_{\text{in } 8-11} = (\text{Bit 7 group internal generate}) \cdot H_7 + (\text{Bit 7 group internal propagate})\ H_7 \cdot C_{\text{in } 4-7} + G_7. \tag{22}$$

When compared with $C_{\text{in } 8-11}$ of the group look-ahead, the output generated by the operation in Eq. (22) detects errors in $C_{\text{in } 8-11}$, G_{4-7}, and T_{4-7}.

The remedies given in Case 4 of the mod 3 check for independent failures in C_{in} apply as well to the parity checked adder.

So far, no specific check has been provided to check the bit generate and transmit signals G_i and T_i. An idea mentioned in Sellers (Ref. 2, pp. 106–107) is to implement

$$H_i = G_i \oplus T_i. \tag{23}$$

In this way, a failure in G_i or T_i appears as a failure in H_i and is caught by the half-sum check. A problem exists in the implementation of Eq. (23). Implementation, as $\bar{G}_i T_i + G_i \bar{T}_i$, provides partial coverage if the complement signals fail independently. Another possibility is to implement H_i as $T_i \bar{G}_i$. This latter idea generally provides partial coverage. An example of applying the Boolean difference to determine the coverage of a generate signal G_5 is given in Appendix B.

7. The comparison

Whenever logic designers implement logic functions into circuitry and attempt to draw general comparisons by "circuit count" (or "can count" or "gate count"), the ancient problem of what constitutes a "logic circuit" or "gate" arises. For the adder studied here, the use of current-switch emitter-follower circuits is assumed.[14]

Table 1 Basic group look-ahead adder circuit count.

Half adders	80
Look-ahead	34
Group internal carry	24
Sum generation	56
Total	194

Table 2 Residue mod 3 checking circuits.

A 16-bit mod 3 tree	70
Mod 3 adder (A, B, C_{in}, C_{out})	20
Mod 3 comparison circuit	4
Group carry check and compare	24

Table 3 Circuit count for parity-checked adder.

16-way EXCLUSIVE-OR tree (half-sum check, sum parity, etc.)	32
High-speed parity prediction and compare	64
Parity comparison circuit	2
Coverage of G and T signals	16
Group carry check and compare	24

The current switch performs, basically, an AND function. By "dotting" (i.e., connecting) the emitters of the emitter-follower transistor to a common load resistor, an OR function (known as DOT-OR or WIRE-OR) is performed. We therefore estimate the "number of circuits" to be the number of AND gates (current switches). If the complement polarity of a dotted signal is required, another circuit is charged. In addition, a 2-circuit EXCLUSIVE-OR gate is assumed to be available.[3,9] Thus Eq. (14a) costs 4 circuits or gates and Eq. (13a) costs 5 circuits; the additional circuit is due to the fact that $\bar{C}_{\text{in }12-15}$ is also required. Equation (20) for PP_{8g} is more complicated and involves EXCLUSIVE-OR gates; it costs 14 circuits. The mod 3 circuit of Fig. 4 takes 8 AND gates and 2 inverters—a cost of 10 circuits.

Before reading the details of the comparison, the reader should be aware that "gate count" figures will vary depending on the technology used. We feel, however, that the general conclusions will still be valid if the two designs are implemented in some other technology. It is also pointed out that although both adders have been designed for complete coverage against single gate failures, the mod 3 adder should provide better coverage against multiple gate failures.

The basic group look-ahead adder may be divided into four parts for the purpose of evaluating its cost:

1) the half-adder functions (H_i, G_i, T_i),
2) the look-ahead functions [Eqs. (8), (9) and (13)],
3) the group internal carry signals [Eqs. (16) and (17)], and
4) the sum generation [Eqs. (15) and (7)].

Table 1 shows the calculated circuit count.

In the mod 3 checking scheme the group carry-in signals are partially covered. This may be remedied by going to a five-bit group or by employing the duplicate-carry check. Rather than design an adder on a five-bit group basis, we assume the mod 3 checking scheme is supplemented by group-carry check and compare circuits. Table 2 gives circuit counts for components in a residue mod 3 checking scheme.

To estimate the circuit count for the parity-checked adder, we must state how the G and T signals are covered. Since we have assumed the availability of a two-circuit EXCLUSIVE-OR gate, Eq. (23) is used to form each half-sum signal H_i. Since it could have been formed as $T_i\bar{G}_i$ at a cost of one circuit, an additional 16 circuits are charged the parity-checked adder as shown in Table 3.

From Tables 2 and 3 it can be seen that it takes more circuits to check or generate the residue mod 3 bits than the parity bit (compare line 1 of Tables 2 and 3). However, it takes more circuits in the parity-checked adder to predict the parity bits for the sum (line 2 of Tables 2 and 3). Therefore, the relative cost of mod 3 checking versus the parity check depends on the overall checking strategy for the data flow. For example, the use of the mod 3 check only to check addition, as in Fig. 6, would involve three residue mod 3 trees (210 circuits), the mod 3 adder and comparator (24 circuits) and the group-carry check (24 circuits)—for a total of 258 circuits. It would cost less to duplicate the adder and compare the outputs. Therefore, one assumes that the decision to use the mod 3 check in the adder is coupled with the decision to use it throughout the data flow, and that the adder operands are already provided with the residue mod 3 bits. The data flow should then have at least one mod 3 residue generator and one mod 3 checker. Now the cost is at least 188 circuits. Similarly, for a parity-checked adder and data flow consisting of adder checking (138 circuits), one parity generator (30 circuits) and one parity checker (32 circuits), the circuit count is at least 200 circuits.

Actually, for the purpose of better failure isolation, a data flow checking strategy may use at least four and possibly five checking trees (Ref. 2, Fig. 5.4, p. 90). Since four mod 3 checking trees cost 280 circuits, compared to 128 circuits for the same number of EXCLUSIVE-OR trees, the less expensive EXCLUSIVE-OR tree indicates why the parity check and parity-checked adder is popular in practice.

8. Additional considerations

For residues above mod 3, a circuit such as Fig. 4 becomes difficult to design. For example, a mod 15 residue calculator for an 8-bit number is an 8-input, 4-output combinational circuit. Sellers (Ref. 2, p. 79) mentions that residues mod 2^{l-1} can be derived using an n-bit adder with end-around carry. Combining this technique with carry look-ahead, the authors designed an 8-bit mod 15 residue building block that costs 40 circuits and 4 levels of delay. The mod 15 residue check for a 16-bit number using this building block costs 120 circuits and 8 levels of delay. This should be compared with 70 circuits and 9 levels of delay for the mod 3 tree. This result is consistent with Pertman,[17] whose analysis of the cost and complexity of residue generation for modulus 3, 7, 15 and 29 shows that the cost increases with the modulus when an attempt is made to maintain the speed (Ref. 17, p. 64). Thus, the use of a larger modulus will detect a larger variety of failures but will also cost a little more. At this point it should be recalled that one can always duplicate the adder unit and compare the outputs to achieve detection of a wide variety of failures, including any combination of failures within a single unit.

It may be pointed out that the arithmetic unit also performs shifting and Boolean (AND, OR or EXCLUSIVE-OR) operations, for which checking is also desired. Rao[18] discusses shifting, complementing and rotating for mod 3 residue checking. Pitkowsky and Godfrey[16] discuss shifter checking with parity bits. Sellers (Ref. 2, p. 174) comments that the use of a parity-checked adder implies a relatively inexpensive bit-by-bit AND and OR check because most of the necessary circuitry is already in the adder for checking addition. The half-sum check provides the parity for the EXCLUSIVE-OR operation. We have not extensively investigated these additional operations, although such considerations are necessary for the implementation of the over-all checking strategy. However, based upon our experience, we do not expect that deeper studies would invalidate any of our general conclusions.

9. Conclusions

This study is an attempt to evaluate the effectiveness and cost of a parity-checked group look-ahead adder similar (but 16-bits wide instead of 32-bits wide) to that used in the IBM System/360 Model 50[16] and to compare

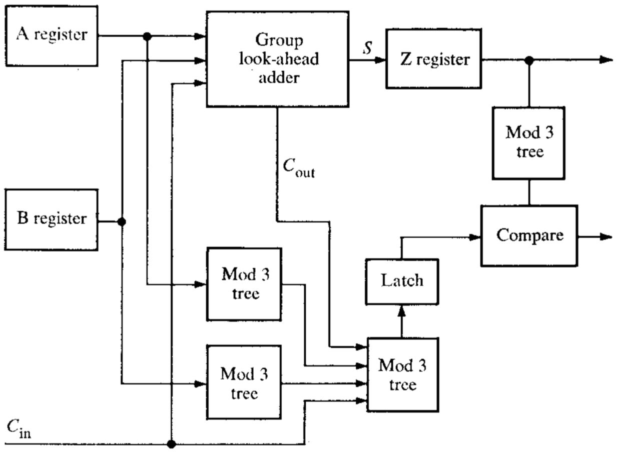


Figure 6 Basic residue mod 3 adder-checking scheme.

it with the implementation of a residue mod 3 check. The practical aspect of this study has led us to some interesting conclusions.

- The calculation of the single gate failure coverage of a concurrent error detection scheme for a look-ahead adder should, in general, not proceed independent of the adder implementation. In particular, prior studies have ignored the size of the group, the implications of gate fan-out limitations, and the fact that the complement of a signal may fail independently of the signal itself.
- The Boolean difference has shown itself to be a useful tool in demonstrating the input conditions under which a gate may be uncovered. For evaluation of the coverage of a residue-checked adder it was necessary to define the "polarized" Boolean difference.
- If the residue mod 3 of both operands must be generated in parallel by the adder, the adder cost with mod 3 checking exceeds the cost of the adder with parity checking. However, this is not the case if the operands are initially designed with the residue mod 3 bits as a part of the data flow register transfer checking. *Therefore, the selection of the adder checking scheme is not independent of the over-all system and its data transfer checking strategy.* For conventional machine organizations with three or more data transfer checks, the parity check seems to offer a cost advantage.

Acknowledgments

The authors thank their colleague D. R. Daykin for his encouragement, suggestions and critical comments. We are also grateful to M. Y. Hsiao for his comments and suggestions. In addition, we thank an anonymous referee for his suggestions, and in particular for motivating us to present the considerations given in Section 8.

Table A-1 Undetected error patterns of sum bits 4-7.

S_7	(2^3)	S_6	(2^2)	S_5	(2^1)	S_4	(2^0)	*Error value*
				+	(−)	+	(−)	$\pm 3 \times 2^4$
		+	(−)			−	(+)	
+	(−)	−	(+)			−	(+)	
+	(−)	−	(+)	−	(+)	+	(−)	
		+	(−)	+	(−)			$\pm 6 \times 2^4$
+	(−)	−	(+)	+	(−)			
+	(−)			−	(+)			
+	(−)			+	(−)	−	(+)	$\pm 9 \times 2^4$
+	(−)					+	(−)	
+	(−)	+	(−)	−	(+)	−	(+)	
+	(−)	+	(−)					$\pm 12 \times 2^4$
+	(−)	+	(−)	+	(−)	+	(−)	$\pm 15 \times 2^4$

Appendix A

When the transmit signals in the group internal look-ahead for sum generation are replaced by the half-sum signals, the adder function is unchanged. However, adder failures behave in a different manner. They will have an error value other than $\pm 2^i$ under some input states when certain single gate failures occur. The polarized Boolean difference must be used to calculate the coverage. The failure of the single gate G_5 is used to illustrate the calculation. Table A-1 is first constructed to show all the sum-bit error patterns of the group that are not detected by the mod 3 residue check. The + and − entries in the table show the polarity of the error; where the plus sign indicates the sum signals should be 0 but changes to 1, and the minus sign indicates that the sum signal should be 1 but changes to 0. Error patterns (rows) in the table are grouped together according to their error value. For example, the second row indicates that when S_4 and S_6 should be 1 (or 0) and 0 (or 1), respectively, but change to 0 (or 1) and 1 (or 0), respectively, the error value is 3×2^4 (or -3×2^4).

The error values in the table are divisible by 3, thus the error patterns shown are not detectable by the mod 3 residue check.

The polarized Boolean differences of sum signals S_4, S_5, S_6 and S_7 are calculated next. Equations (15a) and (15b) are replaced by Eqs. (15a′) and (15b′) respectively to reflect the replacement of bit transmits by bit half sum described at the beginning of Case 1.

$$\begin{aligned} S_7 = {} & C_{\text{in }4-7}\bar{H}_7(H_4H_5H_6) \\ & + \bar{H}_7(G_4H_5H_6 + G_5H_6 + G_6) \\ & + \overline{(H_4H_5H_6)}\,\overline{(G_4H_5H_6 + G_5H_6 + G_6)}H_7 \\ & + \bar{C}_{\text{in }4-7}H_7\overline{(G_4H_5H_6 + G_5H_6 + G_6)}, \end{aligned} \qquad (15a')$$

$$\begin{aligned} S_6 = {} & C_{\text{in }4-7}\bar{H}_6(H_4H_5) + \bar{H}_6(G_4H_5 + G_5) \\ & + \overline{(H_4H_5)}\,\overline{(G_4H_5 + G_5)}H_6 \\ & + \bar{C}_{\text{in }4-7}H_6\overline{(G_4H_5 + G_5)}. \end{aligned} \qquad (15b')$$

Using the above two equations and Eqs. (15c) and (15d), the following is obtained:

$$\frac{dS_7^+}{dG_5} = \bar{H}_7H_6[\bar{H}_5 + \bar{G}_4(\bar{C}_{\text{in }4-7} + \bar{H}_4)], \qquad (A1)$$

$$\frac{dS_7^-}{dG_5} = H_7H_6[\bar{H}_5 + \bar{G}_4(\bar{C}_{\text{in }4-7} + \bar{H}_4)], \qquad (A2)$$

$$\frac{dS_6^+}{dG_5} = \bar{H}_6[\bar{H}_5 + \bar{G}_4(\bar{C}_{\text{in }4-7} + \bar{H}_4)], \qquad (A3)$$

$$\frac{dS_6^-}{dG_5} = H_6[\bar{H}_5 + \bar{G}_4(\bar{C}_{\text{in }4-7} + \bar{H}_4)], \qquad (A4)$$

$$\frac{dS_5^+}{dG_5} = \frac{dS_5^-}{dG_5} = \frac{dS_4^+}{dG_5} = \frac{dS_4^-}{dG_5} = 0. \qquad (A5)$$

Since S_4 and S_5 are independent of G_5, Table A-1 is searched for error patterns in which only S_6 and S_7 are in error. It is found that the 11th row corresponds to an error value of $\pm 12 \times 2^4$. The input state that produces such an error pattern when G_5 fails is thus

$$\begin{aligned} & \frac{dS_7^+}{dG_5}\cdot\frac{dS_6^+}{dG_5} + \frac{dS_7^-}{dG_5}\cdot\frac{dS_6^-}{dG_5} \\ & = H_7H_6[\bar{H}_5 + \bar{G}_4(\bar{C}_{\text{in }4-7} + \bar{H}_4)]. \end{aligned} \qquad (A6)$$

The conjunction (AND) of the above result and $d\bar{G}_{4-7}/dG_5$ must be made to assure that the influence of the G_5 failure is confined to the adder group of bits 4–7. (If the G_5 error propagates to group 8–11 and/or group 12–15, the error value will then be $\pm 2^5$, which is detectable by the mod 3 residue check.

$$\begin{aligned} \frac{d\bar{G}_{4-7}}{dG_5} & = \frac{d}{dG_5}(G_7 + G_6H_7 + G_5T_6H_7 + G_4T_5T_6H_7) \\ & = G_6H_7 + G_4T_5T_6H_7 + \bar{T}_6 + \bar{H}_7. \end{aligned} \qquad (A7)$$

Hence the undetected input states are obtained by forming the conjunction of Eqs. (A6) and (A7) and the result is $H_7H_6G_5H_4$. This result means that when G_5 should be 1 but is s-a-0 and $G_4 = H_6 = H_7 = 1$, both S_6 and S_7 are affected in the same direction. The error value is $\pm 12 \times 2^4$, which is undetectable by the mod 3 residue check and thus G_5 is only partially covered.

Appendix B

An example is given below to illustrate how the Boolean difference can be used to determine the coverage of the gates in the parity-checked adder. The sum Eqs. (15a′), (15b′), (15c) and (15d) will be used and it will be shown that G_5 is fully covered in this situation. Note that for these equations, G_5 is not fully covered by the mod 3 residue check as shown in Appendix A.

The Boolean differences of the sum bits and the predicted group parity with respect to G_5 are first calculated.

$$\frac{dS_7}{dG_5} = H_6[\bar{H}_5 + \bar{G}_4(\bar{C}_{\text{in }4-7} + \bar{H}_4)], \tag{B1}$$

$$\frac{dS_6}{dG_5} = \bar{H}_5 + \bar{G}_4(\bar{C}_{\text{in }4-7} + \bar{H}_4), \tag{B2}$$

$$\frac{dS_5}{dG_5} = \frac{dS_4}{dG_5} = 0, \tag{B3}$$

$$\begin{aligned}\frac{dPP_{\text{sg }4-7}}{dG_5} = \frac{d}{dG_5}\\ [(T_4 \oplus T_5) \oplus (T_6 \oplus H_7) \oplus (G_4H_5\bar{H}_6 + G_5H_6)\\ \oplus (C_{\text{in }4-7}\bar{H}_4 + C_{\text{in }4-7}H_5\bar{H}_6)] = H_6.\end{aligned} \tag{B4}$$

The undetectable input states are those that change an even number of signals among the signals S_7, S_6, S_5, S_4 and $PP_{\text{sg }4-7}$ when G_5 changes. Noting that S_4 and S_5 are unaffected by G_5, the undetectable states are:

$$\frac{dS_7}{dG_5}\cdot\frac{dS_6}{dG_5}\cdot\frac{\overline{dPP}_{\text{sg }4-7}}{dG_5} + \frac{dS_7}{dG_5}\cdot\frac{d\bar{S}_6}{dG_5}\cdot\frac{\overline{dPP}_{\text{sg }4-7}}{dG_5}$$
$$+ \frac{d\bar{S}_7}{dG_5}\cdot\frac{dS_6}{dG_5}\cdot\frac{dPP_{\text{sg }4-7}}{dG_5}.$$

The reader may verify that the above Boolean expression reduces to zero. This result means that G_5 is fully covered.

References

1. O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proc. IRE* **49,** 67 (1961).
2. F. F. Sellers, Jr., M.-Y. Hsiao and L. W. Bearnson, *Error Detecting Logic for Digital Computers,* McGraw-Hill Book Co., Inc., New York, 1968.
3. I. Flores, *The Logic of Computer Arithmetic,* Prentice Hall Inc., Englewood Cliffs, N. J., 1963, Chap. 5.
4. T. G. Gaddess, "An Error Detecting Binary Adder: A Hardware-shared Implementation," *Proc. First Annual IEEE Computer Conference, September* 1967, pp. 38–41.
5. N. Lourie, H. Schrimpf, R. Reach and W. Kahn, "Arithmetic and Control Techniques in a Multi-program Computer," *Proc. EJCC,* 1959, pp. 75–81.
6. M. E. Homan, "A 4-megacycle 24-bit Checked Binary Adder," *AIEE Transactions (Communications and Electronics),* 443 (1961).
7. W. C. Carter and P. R. Schneider, "Design of Dynamically Checked Computers," *Proc. IFIPS Congress 68,* Edinburgh, Booklet E, pp. 34–38.
8. Y. T. Yen, "A Method of Automatic Fault-detection Test Generation for Four-phase MOS LSI Circuits," *Proc. SJCC, AFIPS Conf. Proc.,* Vol. 34, 1969, p. 215.
9. W. W. Peterson, "On Checking an Adder," *IBM J. Res. Develop.* **2,** 166 (1958).
10. H. L. Garner, "Generalized Parity Checking," *IRE Trans. Electronic Computers* **EC-7,** 207 (1958).
11. H. L. Garner, "Error Codes for Arithmetic Operations," *IEEE Trans. Electronic Computers* **EC-15,** 763 (1966).
12. D. S. Henderson, "Residue Class Error Checking Codes," *Proc. 16th National Meeting of the ACM,* September 1961.
13. D. T. Brown, "Error Detecting and Correcting Codes for Arithmetic Operations," *IRE Trans. Electronic Computers* **EC-15,** 333 (1960).
14. R. F. Sechler, et al., "ASLT Circuit Design," *IBM J. Res. Develop.* **11,** 74 (1967).
15. M. Y. Hsiao and F. F. Sellers, "The Carry Dependent Sum Adder," *IEEE Trans. Computers* **EC-12,** 265 (1963).
16. S. H. Pitkowsky and R. B. Godfrey, "Parity Checking and Parity Generating Means for Binary Adders," U. S. Patent 3,342,983, September 19, 1967 (Filed June 25, 1963).
17. A. E. Pertman, "Circuits for Checking Arithmetic Errors by Means of Residue Coding," U. S. Naval Ordnance Laboratory Report NOLTR 69-38, February 12, 1969, 115 pages. (Available from U. S. Government Clearinghouse as AD 687095.)
18. T. R. N. Rao, "Error-checking Logic for Arithmetic-type Operations of a Processor," *IEEE Trans. Computers* **C-17,** 845 (1968).

Received November 26, 1969