Arnold Weinberger

High-Speed Programmable Logic Array Adders

Programmable Logic Array (PLA) adders are described which perform an addition in one cycle with a single pass through a PLA and require a reasonable number of product terms for an 8-, 16-, or even a 32-bit adder. The PLA features two-bit input decoders feeding an AND array followed by an OR array whose outputs are pairwise Exclusive-ORed. Carry-look-ahead adder equations, adapted to the PLA to require relatively few product terms, are adjusted for maximum sharing of product terms. The number of unique product terms is a relative measure of one of the physical dimensions of the PLA. Equations for contiguous sum bits are grouped into strings, each using a common input carry. A procedure optimally assigns sum bits to strings to further minimize the total number of unique product terms. The methods are extended to PLAs with decoders of increased inputs and substantially reduced product terms. They can serve as dedicated macro functions on a chip, using special decoders relevant to adders. As a result, the other PLA dimension comprising the number of outputs from all input decoders increases only moderately, and can even decrease, with larger decoders. Finally, the PLA adder can be further substantially compressed by splitting the OR array into two parts such that a row of the AND array is shared between two product terms, and an OR array column is shared between two sums of product terms.

Introduction

Programmable Logic Arrays, PLAs [1, 2], have been successfully applied to the design of control logic and simple functions such as counters, small adders, etc. Large adders have usually been implemented on standard PLAs iteratively, a few bits per cycle. With previous methodology, the implementation of a large width adder in one cycle with a single pass through a PLA has generally required too many product terms to be economical. The number of product terms in the AND array is a measure of one of the dimensions of a PLA and is directly related to the silicon area on a chip as well as the signal delay through the PLA.

This paper describes one-cycle adder designs for standard PLAs as well as for PLAs dedicated to adders. The standard PLA adder is an improved version of one described elsewhere by the author [3]. These designs reduce the number of product terms to acceptable levels even for 16- and 32-bit adders. Two features of standard PLA designs are particularly useful in reducing the number of logical product terms. These are:

1. Two-bit input decoders, where a pair of inputs and their inverters are replaced by a two-input decoder, and
2. Exclusive-OR (XOR) outputs, where a pair of OR array outputs are XORed.

Two-bit decoders can, in turn, be replaced by fewer decoders having more than two inputs to further reduce the number of product terms. To avoid an uneconomical increase in the number of decoder outputs, however, the decoders are restricted to produce only outputs that are pertinent to the add function. The result is a PLA design dedicated to adders.

Adder equations are expressed in a suitable manner to take advantage of these features and of various methods of sharing product terms among sums of product terms. In particular, strings of output sum bits, each comprising one or more contiguous sums, are expressed in terms of their common carry using well-known carry-look-ahead

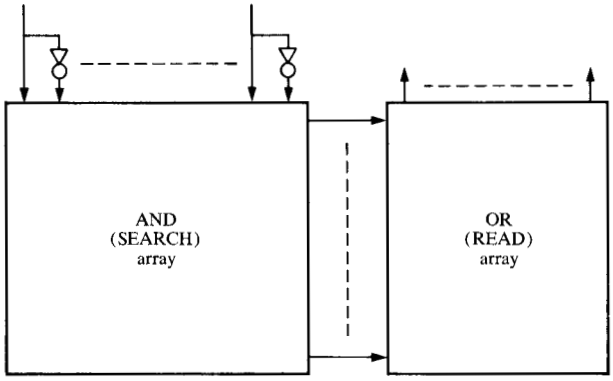

Figure 1 PLA (Programmable Logic Array).

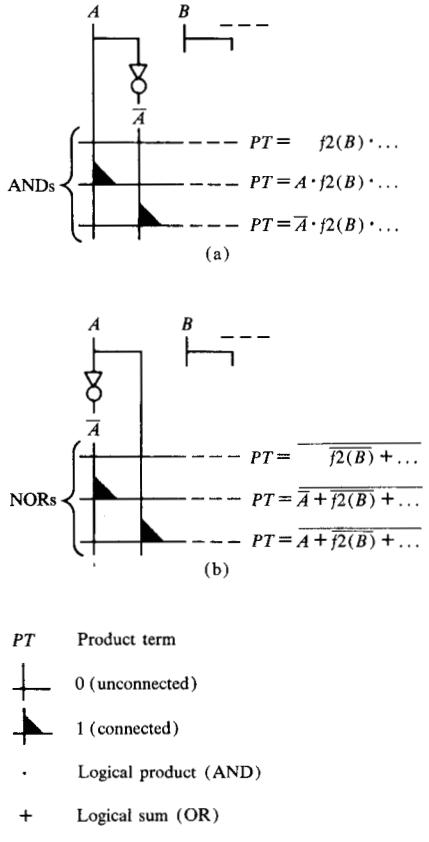

Figure 2 Personalization of AND (SEARCH) array using (a) real ANDs, or (b) NORs.

methods, and a procedure is developed to optimize string sizes to additionally reduce the number of product terms.

The AND array which contains the unique product terms can be further reduced through the sharing of a row of the array by two product terms. Similarly, the OR array, which generates logical sums of product terms, can also be reduced through the sharing of a column of the array by two sums of products. The split rows and columns are particularly effective in dedicated PLA adders, although they may also be useful for other logical functions using PLAs.

Standard PLAs

A PLA consists basically of an AND array and an OR array in series, as shown in Fig. 1. The array names, AND-OR, describe the generic logic levels of the SEARCH-READ arrays of an associative table [4]. The two arrays may be implemented with types of logic other than AND-OR; a widely-used logic, implemented with MOS technology, is NOR-NOR.

The generic AND (SEARCH) array produces an array of product terms of the inputs to the PLA. Each product term is the AND of functions of the individual inputs, $A, B, C, \ldots$, as in Eq. (1):

$$\textit{Product term} = f1(A) \cdot f2(B) \cdot \ldots . \quad (1)$$

Each input enters the AND in one of three states: true, complement, or don't care. The true and complement lines of each input intersect the AND array at two connections which are personalized for one of the three states. The personalization (illustrated for the input A) is of the forms shown in Fig. 2(a) when the generic AND (SEARCH word) is implemented with a real AND, and is of the form shown in Fig. 2(b) when implemented with a NOR. It should be noted that only one connection at most is made at the intersections of the AND array with the true and complement lines of an input. For the don't care state, no connections are made. To personalize the two connections it is sufficient to provide a single switching device to which either the true, the complement, or neither line is connected.

The generic OR (READ) array produces a generic OR of selected product terms on each array output. The array is personalized with a single bit at each intersection of a product term with an output line. A 1 selects the product term, a 0 does not. Each array output is the real OR of selected product terms if the array is comprised of real ORs, as in Fig. 3. If the array is comprised of NORs, each output is the NOR of selected product terms.

A complete set of minterms (maxterms) corresponds to the positive (negative) outputs of an n-bit decoder. Figures 2(a) and (b) can be interpreted as providing a one-bit decoder for input A: Fig. 2(a) provides the two maxterms A and $\bar{A}$, while Fig. 2(b) provides the corresponding two minterms $\bar{A}$ and A.

The personalized two-bit cell at the intersection of a product term with the one-bit decoder outputs corresponds to selecting the subset of minterms (maxterms) to comprise the desired function. Figures 4(a) and (b) illustrate the four possible functions of input A, of which three are used. Figure 4(a) shows the possible products of maxterms, each maxterm included or not according to the function to be personalized. A 1 is ORed with the maxterm if it is not included in the function, while a 0 is ORed if it is included. Similarly, Fig. 4(b) shows the possible sums of minterms, each minterm included according to the function to be personalized. A 1 is ANDed with the minterm if included, a 0 if not.

The number of product terms can be significantly reduced by substituting two-bit decoders for a pair of one-bit decoders [5]. The total number of decoder outputs remains the same. The product term now represents the AND of functions of pairs of inputs, as in Eq. (2):

$$\textit{Product term} = f1(A1, B1) \cdot f2(A2, B2) \cdot \ldots . \qquad (2)$$

Figure 5 shows the 16 possible functions of inputs A and B, of which 15 may be used. Figure 5(a) shows the possible products of maxterms, while Fig. 5(b) shows the possible sums of minterms. The latter defines functions of A and B to correspond to a NOR implementation of a product term, as in Eq. (3).

$$\textit{Product term} = \overline{\overline{f1(A1, B1)} + \overline{f2(A2, B2)} + \ldots .} \qquad (3)$$

This corresponds to Fig. 4(b) for one-bit decoders. It should be noted that only three switching devices are needed to personalize the four bits since the last function, requiring four connections, is unused [6].

Two-input decoders have already been applied to a standard PLA [2] and will be shown to be particularly useful for adders.

Another economizing PLA feature is the use of XOR outputs [7], where pairs of OR array outputs are XORed to produce a single PLA output. Figure 6 shows the PLA expanded to include two-input decoders and XOR outputs.

Adders

A typical adder adds two n-bit numbers, $A(A_0, \ldots, A_{n-1})$ and $B(B_0, \ldots, B_{n-1})$ together with an input carry C_{in} to produce a sum $S(S_0, \ldots, S_{n-1})$ and an output carry C_{out} (C_0). Using the single-bit-position functions,

$$G_i = A_i B_i, \qquad P_i = A_i + B_i, \qquad H_i = A_i \,\forall\, B_i,$$

a carry C_i from any bit position i can be expressed directly in terms of these functions and C_{in}, as in Eqs. (4) and (5):

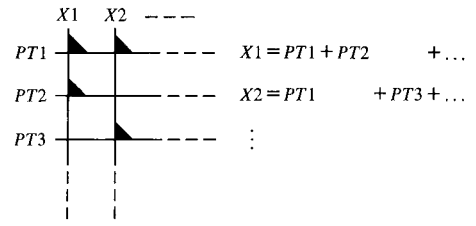

Figure 3 Personalization of OR (READ) array.

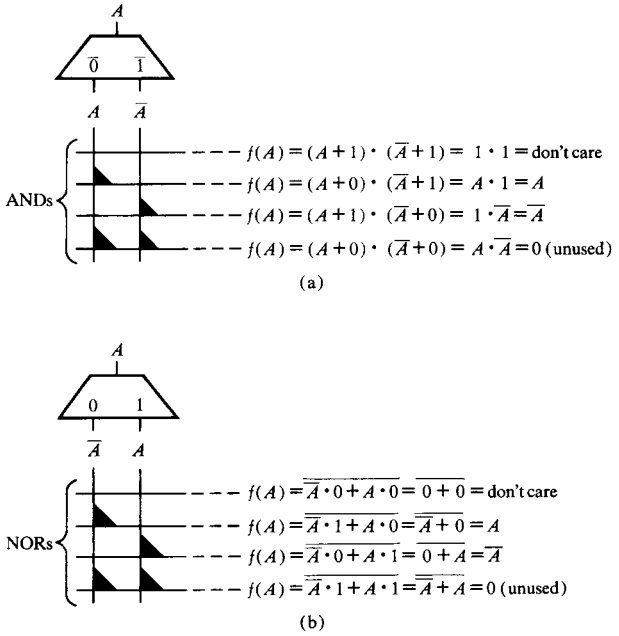

Figure 4 One-input functions using a one-input decoder and a personalized two-bit cell with (a) complement decode outputs and maxterm personalization, or (b) true decode outputs and minterm personalization.

$$C_i = \sum_{a=i}^{n} \left[\prod_{b=i}^{a-1} H_b^* \right] \cdot G_a, \qquad (4)$$

$$\bar{C}_i = \sum_{a=i}^{n} \left[\prod_{b=i}^{a-1} H_b^{**} \right] \cdot \bar{P}_a, \qquad (5)$$

where Σ and Π are symbols for OR and AND, respectively, H^* means either H or P may be used, H^{**} means either H or $\bar{G}$ may be used, $G_n \equiv C_{\text{in}} \equiv C_n$, and $\bar{P}_n \equiv \bar{C}_{\text{in}} \equiv \bar{C}_n$. (It is desirable to substitute P or $\bar{G}$ for H where possible since $P = A + B$ and $\bar{G} = \bar{A} + \bar{B}$ require but one connection while $H = A \,\forall\, B$ requires two connections in the AND array, as shown in Fig. 5.)

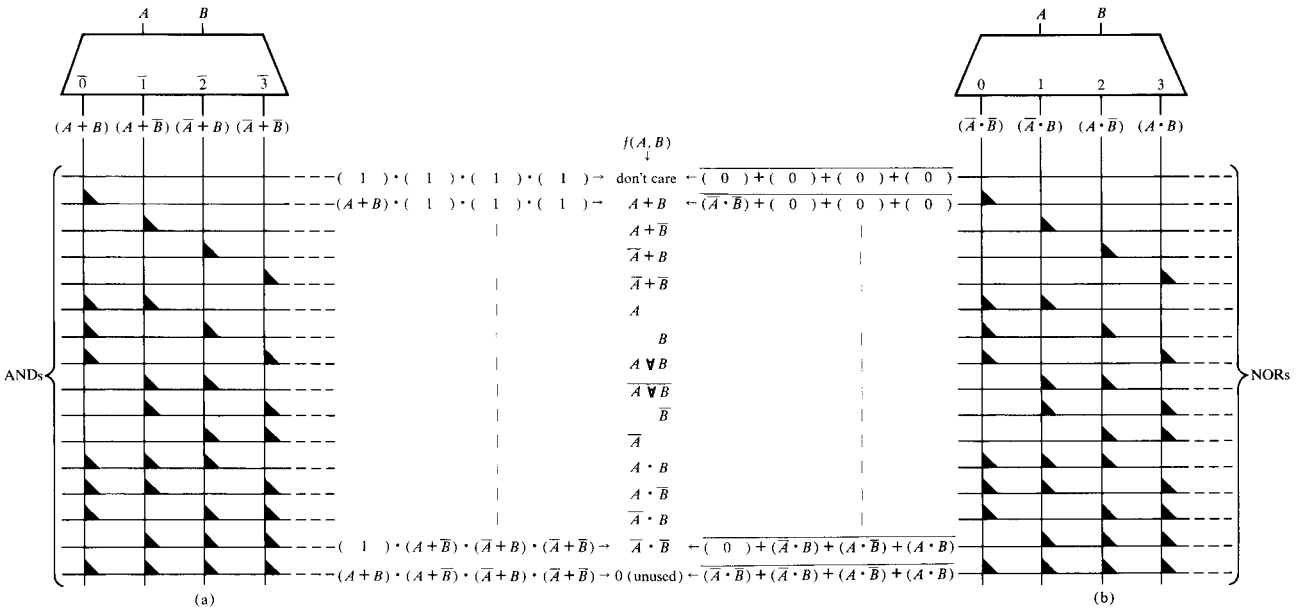

Figure 5 Two-input functions using a two-bit decoder and a personalized four-bit cell with (a) complement decode outputs and maxterm personalization, or (b) true decode outputs and minterm personalization.

Also, a sum bit can be expressed as a function of the output carry from the preceding bit position and expanded into an XOR of two entities, one of which includes a distant carry, as in Eqs. (6) and (7):

$$\begin{aligned} S_i &= H_i \forall C_{i+1} = H_i \forall (G^j_{i+1} + H^j_{i+1} \cdot C_{j+1}) \\ &= (H_i \forall G^j_{i+1}) \forall (H^j_{i+1} \cdot C_{j+1}) \\ &= (H_i \forall \bar{G}^j_{i+1}) \forall (\bar{H}^j_{i+1} + \bar{C}_{j+1}), \end{aligned} \tag{6}$$

$$\begin{aligned} \bar{S}_i &= H_i \forall \bar{C}_{i+1} = H_i \forall (\overline{GH}^j_{i+1} + H^j_{i+1} \cdot \bar{C}_{j+1}) \\ &= (H_i \forall \overline{GH}^j_{i+1}) \forall (H^j_{i+1} \cdot \bar{C}_{j+1}) \\ &= (H_i \forall GH^j_{i+1}) \forall (\bar{H}^j_{i+1} + C_{j+1}), \end{aligned} \tag{7}$$

where G^j_{i+1} = carry-generate condition for bit group $i+1$ through j (high-to-low order, $i \leq j$), H^j_{i+1} = strict carry-propagate condition (mutually exclusive with G^j_{i+1}), and $GH^j_{i+1} = G^j_{i+1} + H^j_{i+1}$ = inclusive carry-propagate condition, which can be expressed as sums of product terms, as follows:

$$G^j_{i+1} = \sum_{a=i+1}^{j} \left[\prod_{b=i+1}^{a-1} H^*_b \right] \cdot G_a,$$

$$H^j_{i+1} = \prod_{b=i+1}^{j} H_b,$$

$$GH^j_{i+1} = \left\{ \sum_{a=i+1}^{j-1} \left[\prod_{b=i+1}^{a-1} H^*_b \right] \cdot G_a \right\} + \left[\prod_{b=i+1}^{a-1} H^*_b \right] \cdot P_j,$$

$$\bar{G}^j_{i+1} = \left\{ \sum_{a=i+1}^{j-1} \left[\prod_{b=i+1}^{a-1} H^{**}_b \right] \cdot \bar{P}_a \right\} + \left[\prod_{b=i+1}^{j-1} H^{**}_b \right] \cdot \bar{G}_j,$$

$$\bar{H}^j_{i+1} = \sum_{b=i+1}^{j} \bar{H}_b,$$

$$\overline{GH}^j_{i+1} = \sum_{a=i+1}^{j} \left[\prod_{b=i+1}^{a-1} H^{**}_b \right] \cdot \bar{P}_a.$$

In a similar fashion, the output carry can be expressed as an XOR of two entities, one of which includes a distant carry, as shown in Eqs. (8) and (9):

$$\begin{aligned} C_{\text{out}} &= \overline{\overline{GH}^j_0 + H^j_0 \cdot \bar{C}_{j+1}} = \overline{\overline{GH}^j_0 \forall H^j_0 \cdot \bar{C}_{j+1}} \\ &= \{\overline{GH}^j_0\} \forall \{\bar{H}^j_0 + C_{j+1}\}, \end{aligned} \tag{8}$$

$$\begin{aligned} \bar{C}_{\text{out}} &= \overline{G^j_0 + H^j_0 \cdot C_{j+1}} = \overline{G^j_0 \forall H^j_0 \cdot C_{j+1}} \\ &= \{G^j_0\} \forall \{\bar{H}^j_0 + \bar{C}_{j+1}\}. \end{aligned} \tag{9}$$

Equations (6) through (9) can also be expressed as functions of the distant carry of opposite polarity. The selected forms of the equations provide more opportunities for sharing product terms.

PLA adder designs

The adder equations can now be applied to the PLA of Fig. 6.

Addend and augend of the same bit position, A_i and B_i, enter a common decoder, so that the intersection of an AND with the decoder outputs can produce one of the six useful adder functions of A_i and B_i, i.e., G_i, P_i, H_i, or their complements. The input carry C_{in} enters as the sole input to a decoder. (For uniformity, a two-input decoder is provided for C_{in} with one input unused.)

A string of K contiguous sum bits is generated as a function of a common carry into the string, using Eqs. (6) and (7). Positive and negative strings of sum bits are shown in Eqs. (10) and (11), respectively:

$$S_j = \{\bar{H}_j\} \forall \{\bar{C}_{j+1}\},$$

$$S_i = \left\{ \begin{array}{l} \bar{H}_i \cdot \sum_{a=i+1}^{j-1} \left[\prod_{b=i+1}^{a-1} H_b^{**} \right] \cdot \bar{P}_a \\ + \bar{H}_i \cdot \left[\prod_{b=i+1}^{j-1} H_b^{**} \right] \cdot \bar{G}_j \\ + H_i \cdot \sum_{a=i+1}^{j} \left[\prod_{b=i+1}^{a-1} H_b^{*} \right] \cdot G_a \end{array} \right\}$$

$$\forall \left\{ \begin{array}{l} \sum_{a=i+1}^{j-1} \bar{H}_a \cdot \left[\prod_{b=a+1}^{j-1} H_b^{**} \right] \cdot \bar{G}_j \\ + \bar{H}_j + \bar{C}_{j+1} \end{array} \right\}; \qquad (10)$$

$$\bar{S}_j = \{\bar{H}_j\} \forall \{C_{j+1}\},$$

$$\bar{S}_i = \left\{ \begin{array}{l} \bar{H}_i \cdot \sum_{a=i+1}^{j-1} \left[\prod_{b=i+1}^{a-1} H_b^{*} \right] \cdot G_a \\ + \bar{H}_i \cdot \left[\prod_{b=i+1}^{j-1} H_b^{*} \right] \cdot P_j \\ + H_i \cdot \sum_{a=i+1}^{j} \left[\prod_{b=i+1}^{a-1} H_b^{**} \right] \cdot \bar{P}_a \end{array} \right\}$$

$$\forall \left\{ \begin{array}{l} \sum_{a=i+1}^{j-1} \bar{H}_a \cdot \left[\prod_{b=a+1}^{j-1} H_b^{*} \right] \cdot P_j \\ + \bar{H}_j + C_{j+1} \end{array} \right\} \qquad (11)$$

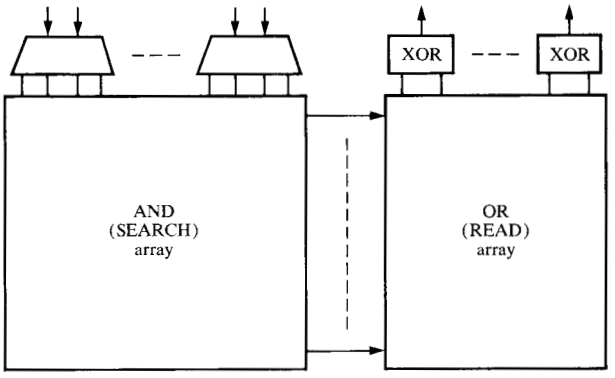

Figure 6 PLA with two-input decoders and XOR outputs.

for $i = j - K + 1, \ldots, j - 1$; high-to-low order; $i < j$. Note that $\bar{H}_{i+1}^{j} = \bar{H}_{i+1} + \ldots + \bar{H}_j$ of the bracket to the right of the Exclusive-OR is actually implemented with product terms already present in the left brackets of the string of sums. The reader can verify that the different representations of Eq. (12) are equivalent:

$$\bar{H}_{i+1}^{j} = \sum_{a=i+1}^{j} \bar{H}_a = \sum_{a=i+1}^{j-1} \bar{H}_a \cdot \left[\prod_{b=a+1}^{j-1} H_b^{**} \right] \cdot \bar{G}_j + \bar{H}_j$$

$$= \sum_{a=i+1}^{j-1} \bar{H}_a \cdot \left[\prod_{b=a+1}^{j-1} H_b^{*} \right] \cdot P_j + \bar{H}_j. \qquad (12)$$

The common carry shared by the sum bits of a string is expressed as a sum of product terms according to Eq. (4) or (5) and is generated in the AND array. Clearly, if the sum bits are grouped into few but large strings, few such common carries, and hence few product terms for these carries, would be needed. On the other hand, the number of product terms needed for a sum bit in a string increases with the distance of the sum bit from the common carry. Therefore, the total number of product terms needed for the adder is minimized by choosing an optimal grouping of sum bits to strings.

Three string types are identified: low-order, intermediate, and high-order.

A *low-order string* includes a product term representing the input carry C_{in} or $\bar{C}_{in}$, the low-order sum bits implemented according to Eq. (10) or (11), and the product terms representing the output carry of the string according to Eq. (4) or (5). The indexes $(j - 1, j)$ become $(n - 1, in)$. Note that the high-order sum of the string, S_i (for $i = j - K + 1$) of Eq. (10), shares some of its product

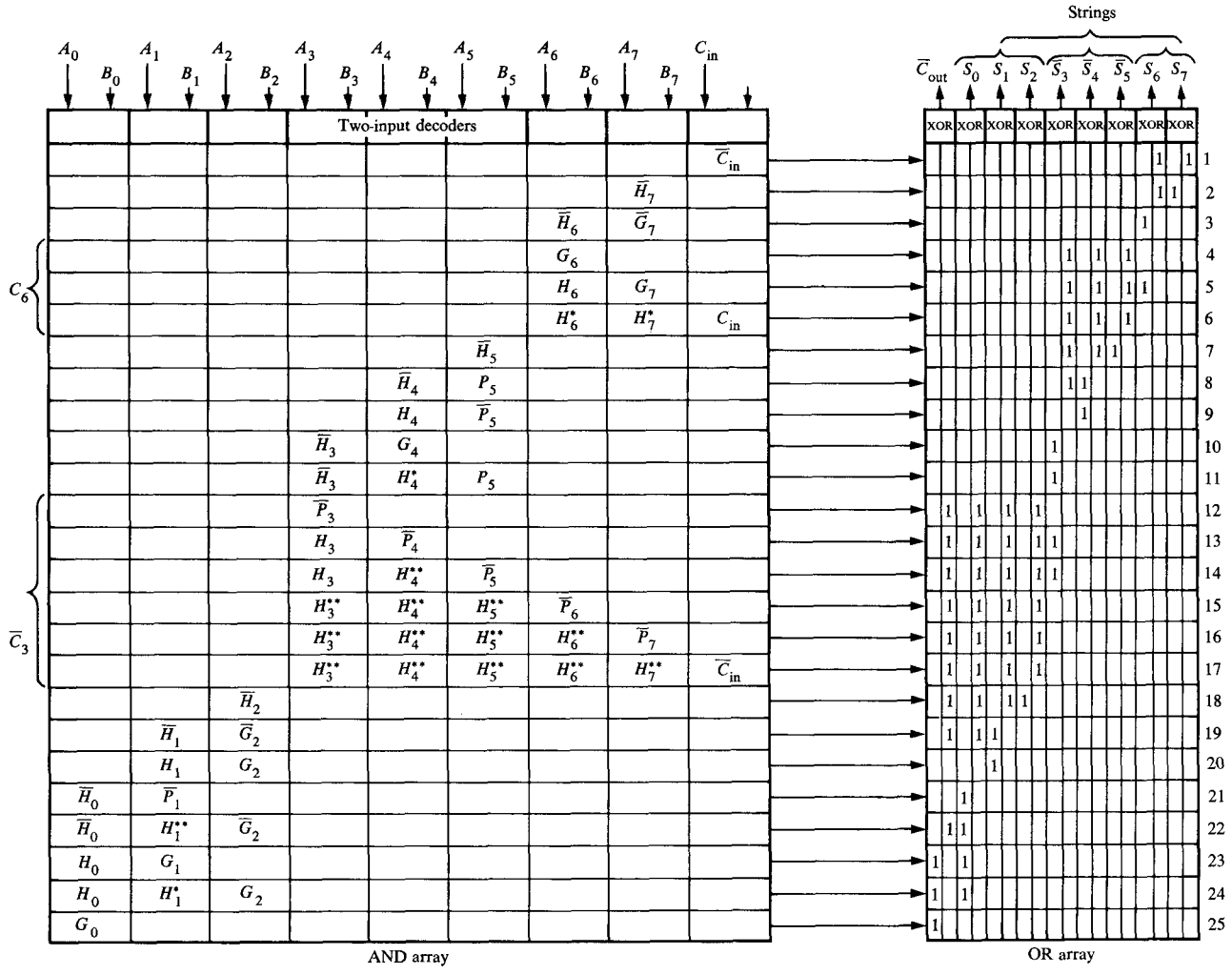

Figure 7(a) Eight-bit PLA adder: PLA format.

terms with the output carry of the strong C_i of Eq. (4), and $\bar{S}_i$ of Eq. (11) shares product terms with $\bar{C}_i$ of Eq. (5). For example, the product term $H^*_i \cdot H^*_{i+1} \cdot \ldots \cdot H^*_{j-1} \cdot G_j$ of Eq. (4) can be shared with the product term $H_i \cdot H^*_{i+1} \cdot \ldots \cdot H^*_{j-1} \cdot G_j$ of the left bracket of Eq. (10). Therefore, it is advantageous to use the same polarity output carry from the string as the sum bits. Since the sum bits are a function of the opposite polarity input carry to the string, it is also advantageous to alternate polarities of strings. It should also be noted that when sharing product terms between S_i and C_i (or $\bar{S}_i$ and $\bar{C}_i$), the common factor H_i must be used and P_i (or $\bar{G}_i$) cannot be substituted for it, i.e., H^*_i (or H^{**}_i) does not apply.

The number of unique product terms needed for a low-order string of K sum bits and its output carry is: 1 for the input carry, $1 + 2 + 4 + \ldots + 2(K-1)$ for the sum bits (noting that some product terms are shared, e.g., $\bar{H}_j$), and 2 for the additional unique (non-shared) product terms contained in the output carry of the string. Equation (13) expresses T_{low}, the number of unique product terms of the low-order string:

$$\begin{aligned} T_{\text{low}} &= 3 \qquad \text{for } K = 1, \\ &= 1 + [1 + 2 + 4 + \ldots + 2(K-1)] + 2 \\ &= K^2 - K + 4 \qquad \text{for } K > 1. \end{aligned} \tag{13}$$

For $K = 1$, the low-order sum is generated more efficiently according to Eq. (14) or (15):

$$S_{n-1} = \{H_{n-1} \cdot \bar{C}_{\text{in}}\} \forall \{\bar{H}_{n-1} \cdot C_{\text{in}}\}, \tag{14}$$

$$\bar{S}_{n-1} = \{H_{n-1} \cdot C_{\text{in}}\} \forall \{\bar{H}_{n-1} \cdot \bar{C}_{\text{in}}\}, \tag{15}$$

together with the opposite polarity output carry of this string, $\bar{C}_{n-1} = \bar{P}_{n-1} + H_{n-1} \cdot \bar{C}_{\text{in}}$, or $C_{n-1} = G_{n-1} + H_{n-1} \cdot C_{\text{in}}$, respectively. The two product terms of S_{n-1} (or $\bar{S}_{n-1}$) and the additional unique product term for $\bar{C}_{n-1}$ (or C_{n-1}) add up to three unique product terms for a low-order

$$S_7 = \{\bar{H}_7\} \; \forall \; \{\bar{C}_{in}\}$$

$$S_6 = \begin{Bmatrix} \bar{H}_6 \cdot \bar{G}_7 \\ +H_6 \cdot G_7 \end{Bmatrix} \forall \{\bar{H}_7 + \bar{C}_{in}\}$$

$$\bar{S}_5 = \{\bar{H}_5\} \; \forall \; \{C_6\}$$

$$\bar{S}_4 = \begin{Bmatrix} \bar{H}_4 \cdot P_5 \\ +H_4 \cdot \bar{P}_5 \end{Bmatrix} \forall \{\bar{H}_5 + C_6\}$$

$$\bar{S}_3 = \begin{Bmatrix} \bar{H}_3 \cdot G_4 \\ +\bar{H}_3 \cdot H_4^* \cdot P_5 \\ +H_3 \cdot \bar{P}_4 \\ +H_3 \cdot H_4^{**} \cdot \bar{P}_5 \end{Bmatrix} \forall \begin{Bmatrix} \bar{H}_4 \cdot P_5 \\ +\bar{H}_5 + C_6 \end{Bmatrix}$$

$$S_2 = \{\bar{H}_2\} \; \forall \; \{\bar{C}_3\}$$

$$S_1 = \begin{Bmatrix} \bar{H}_1 \cdot \bar{G}_2 \\ +H_1 \cdot G_2 \end{Bmatrix} \forall \{\bar{H}_2 + \bar{C}_3\}$$

$$S_0 = \begin{Bmatrix} \bar{H}_0 \cdot \bar{P}_1 \\ +\bar{H}_0 \cdot H_1^{**} \cdot \bar{G}_2 \\ +H_0 \cdot G_1 \\ +H_0 \cdot H_1^* \cdot G_2 \end{Bmatrix} \forall \begin{Bmatrix} \bar{H}_1 \cdot \bar{G}_2 \\ +\bar{H}_2 + \bar{C}_3 \end{Bmatrix}$$

$$\bar{C}_{out} = \begin{Bmatrix} G_0 \\ +H_0 \cdot G_1 \\ +H_0 \cdot H_1^* \cdot G_2 \end{Bmatrix} \forall \begin{Bmatrix} \bar{H}_0 \cdot H_1^{**} \cdot \bar{G}_2 \\ +\bar{H}_1 \cdot \bar{G}_2 \\ +\bar{H}_2 + \bar{C}_3 \end{Bmatrix}$$

$$C_6 = G_6 + H_6 \cdot G_7 + H_6^* \cdot H_7^* \cdot C_{in}$$

$$\bar{C}_3 = \bar{P}_3 + H_3 \cdot \bar{P}_4 + H_3 \cdot H_4^{**} \cdot \bar{P}_5 + H_3^{**} \cdot H_4^{**} \cdot H_5^{**} \cdot \bar{P}_6 + H_3^{**} \cdot H_4^{**} \cdot H_5^{**} \cdot H_6^{**} \cdot \bar{P}_7 + H_3^{**} \cdot H_4^{**} \cdot H_5^{**} \cdot H_6^{**} \cdot H_7^{**} \cdot \bar{C}_{in}$$

H^* H or P may be used
H^{**} H or $\bar{G}$ may be used

Figure 7(b) Eight-bit PLA adder: equations.

string of one. If a low-order string of one is used, the next string is of the same polarity as the low-order sum in order to make use of the opposite polarity output carry of the low-order string.

An *intermediate string* uses the product terms of the output carry of the preceding string to generate the sum bits according to Eq. (10) or (11). It also generates the output carry of the string according to Eq. (4) or (5), respectively.

The number of unique product terms for an intermediate string, T_i, of size $K > 1$ is one less than for a low-order string because the input carry to the intermediate string has already been counted as part of the preceding string. For $K = 1$ they are equal. However, the output carry of the string has additional product terms equal to L, the number of bit positions of lower order than the string.

$$T_i = K^2 - K + 3 + L \qquad \text{for } K \geq 1. \tag{16}$$

A *high-order string* generates the high-order sum bits as for an intermediate string. However, the output carry of the string, C_0, is needed only as an output of the adder, C_{out}, so that it can be generated according to Eq. (8) or (9) as a function of the input carry to the string, as shown in expanded form in Eq. (17) or (18):

$$C_{out} = \left\{\bar{P}_0 + H_0 \cdot \sum_{a=1}^{j} \left[\prod_{b=1}^{a-1} H_b^{**}\right] \cdot \bar{P}_a\right\} \forall \left\{\sum_{a=0}^{j-1} \bar{H}_a \cdot \left[\prod_{b=a+1}^{j-1} H_b^*\right] \cdot P_j + \bar{H}_j + C_{j+1}\right\}, \tag{17}$$

$$\bar{C}_{out} = \left\{G_0 + H_0 \cdot \sum_{a=1}^{j} \left[\prod_{b=1}^{a-1} H_b^*\right] \cdot G_a\right\} \forall \left\{\sum_{a=0}^{j-1} \bar{H}_a \cdot \left[\prod_{b=a+1}^{j-1} H_b^{**}\right] \cdot \bar{G}_j + \bar{H}_j + \bar{C}_{j+1}\right\}. \tag{18}$$

Here, product terms can be shared between C_{out} and $\bar{S}_0$ (or $\bar{C}_{out}$ and S_0), so that opposite polarities are selected. Also, Eq. (12) is used to take advantage of product terms already present in the sums of the string. Therefore, only one additional unique product term is needed for C_{out} or $\bar{C}_{out}$.

The number of unique product terms for the high-order string, T_{high}, is $L + 1$ less than for an intermediate string, since the output carry is a function of the input carry to the string:

$$T_{high} = K^2 - K + 2 \qquad \text{for } K \geq 1. \tag{19}$$

Figure 7(a) illustrates an eight-bit adder that generates the outputs in a one-cycle pass through the PLA. The out-

Table 1 Transition values for optimum intermediate string sizes.

$K \leftrightarrow (K+1)$	$2 \leftrightarrow 3$	$3 \leftrightarrow 4$	$4 \leftrightarrow 5$	- - -
L_t	3	9	17	- - -
ΔL_t	6	8	10	

Table 2 Illustration of procedure for optimal string assignment.

First-pass string assignment (Nos. are string sizes)								*Final string assignment*						
	5	4	4	3	3	2	1	}						
5	5	4	4	3	3	2	1	} no change						
5	5	5	4	4	3	3	2	1	}						
							+							
1	5	4	4	3	3	2	1		5	4	4	3	3	2	2
						+	+							
2	5	4	4	3	3	2	1		5	4	4	3	3	3	2
				+		+	+							
3	5	4	4	3	3	2	1		5	4	4	4	3	3	2
		+		+		+	+							
4	5	4	4	3	3	2	1		5	5	4	4	3	3	2

+ above numbers marks strings to be increased by one.
/ through numbers marks remainder to be absorbed.

put sum bits are divided into three strings of 3, 3, and 2 bits, high-to-low order. The strings have been optimized to further reduce the total number of product terms to 25. An entry in the AND array is noted with a function of the decoder inputs, i.e., $G_i = A_i \cdot B_i$, etc. These functions can be readily converted to personalized four-bit cells by means of Fig. 5. Figure 7(b) expresses the eight-bit adder in equation form to correspond to the PLA format used.

Optimization

An optimum string size is determined by minimizing the total number of product terms (T) averaged over the string size (K). We begin with the low-order string and proceed toward the higher-order strings.

An optimum low-order string is either one or two bits long, since

$$(T_{\text{low}}/K)\ \min = 3 \qquad \text{for } K = 1 \text{ or } 2. \tag{20}$$

For an intermediate string, the minimum number of product terms averaged over the string size,

$$(T_i/K)\ \min = [(K^2 - K + 3 + L)/K]\ \min \qquad \text{for } K \geq 1,$$

is a function of L, the number of bits of lower order than the string. Successive (higher-order) intermediate strings should therefore be increasing monotonically. We determine the transition value of L, L_t, for which string sizes K and $K + 1$ are equally efficient, i.e.,

$$(K^2 - K + 3 + L)/K = [(K+1)^2 - (K+1) + 3 + L]/(K+1),$$

$$L_t = K^2 + K - 3 \qquad \text{for } K \geq 1. \tag{21}$$

For $K = 1$, L_t is negative, which means that an intermediate string size of two is always more efficient than a string of one.

Table 1 lists various transition values as well as changes in transition values. It shows that, after three lower-order bit positions, the next string size is equally efficient at two or three; after nine lower-order bit positions, the next string size is equally efficient at three or four; etc.

The change in transition values, ΔL_t, where

$$\Delta L_t = L_t(K \leftrightarrow K+1) - L_t(K-1 \leftrightarrow K) = (K^2 + K - 3) - [(K-1)^2 + (K-1) - 3] = 2K, \tag{22}$$

shows that a pair of equal intermediate string sizes (two $K - 1$ sizes) are followed by a pair of next larger size (two K sizes) for optimum assignment of intermediate string sizes. In other words, after a low-order string of one is arbitrarily selected and followed by an intermediate string of two, pairs of next higher string sizes follow (pairs of threes, pairs of fours, etc.).

An optimum high-order string is determined in relation to the other strings. First we note that if the high-order string is greater than (or smaller than) the adjacent intermediate string by two or more, the combined number of product terms for the two strings can be reduced by reducing (or increasing) the high-order string by one and increasing (or reducing) the adjacent string by one.

This leads to the following empirical procedure for assigning string sizes: We begin with a low-order string of one (the smaller of the two optimal sizes), followed by a single string of two and pairs of strings of three, four, etc. If the bit positions of the adder are exhausted when the high-order string is equal to or one greater than the adjacent string, the first-pass string assignment is final. If the high-order string is less than the adjacent string, the latter becomes the new high-order string and the former high-order string is deemed a remainder to be absorbed by the intermediate strings as follows: First, the low-order string of one is increased to two, the next string of two is increased to three, the higher-order of the two strings of

Table 3 Number of product terms for (a) eight-bit adder, (b) 16-bit adder, and (c) 32-bit adder, using a conventional PLA. K = string size, L = number of lower-order bit positions, and T = number of product terms.

(a)

	Bit position								
	0	1	2	3	4	5	6	7	C_{in}
K	3			3			2		
L				2			—		
T	8			11			6		

(25) product terms

(b)

	Bit position																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	C_{in}
K	4				4				3			3				2	
L					8				5			2				—	
T	14				23				14			11				6	

(68) product terms

(c)

	Bit position																															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	C_{in}
K	5					5					5					4				4				3			3			2		1	
L						22					17					13				9				6			3			1		—	
T	22					45					40					28				24				15			12			6		3	

(195) product terms

three is increased to four, the higher-order of the next pair of intermediate strings is increased by one, etc., until the remainder is exhausted.

Table 2 illustrates the above procedure for assigning strings to achieve a minimum number of product terms. The assignment is not necessarily unique. For some adder sizes a different assignment can achieve the same minimum. For example, the eight-bit adder of Fig. 7 can also be implemented with 25 product terms using string sizes 2, 3, 2, and 1, high-to-low order.

Table 3 illustrates the relevant parameters for eight-bit, 16-bit, and 32-bit adders, using 25, 68, and 195 product terms, respectively.

Decoders with more than two inputs

Additional reduction in the number of product terms for an adder may be obtained using four-input or higher-input decoders while preserving the generality of use of the PLA. A product term may now be defined as the AND of functions of input groups, with an input group comprising the inputs of a decoder. With standard decoders, however, this results in a wider AND array and more costly decoding. For example, a four-input decoder replacing two two-input decoders doubles the number of decoder outputs from 8 to 16, an eight-input decoder replacing four two-input decoders increases the number of decoder outputs from 16 to 256, etc. In the limit, a single decoder accepting all adder inputs becomes a conventional ROM decoder, while each product term can represent any function of the decoder inputs without the need of an OR array. In short, the single decoder and the AND array comprise a complete ROM whose outputs are any desired logic functions of the inputs.

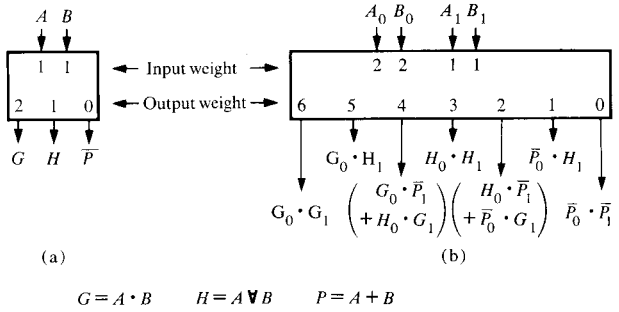

Figure 8 Special decoders generating elementary symmetric functions from (a) one pair of adder inputs, and (b) two adjacent pairs of adder inputs.

Special decoders, however, can permit more inputs per decoder without expanding the width of the AND array or, at most, only moderately expanding it. One type of special decoder produces elementary symmetric functions to take advantage of symmetry which is derived from the relative weights of the adder inputs. Thus, the adder input of bit position i, A_i or B_i, has a relative weight of 1 when the input is not zero; A_{i-1} or B_{i-1} has a relative weight of 2 when not zero; A_{i-2} or B_{i-2} a relative weight of 4 when not zero; etc. The decoder generates the unique values of the combined weights of its inputs. For example, two pairs of adder inputs of adjacent bit positions have relative weights of 2, 2, 1, and 1. They enter the special decoder

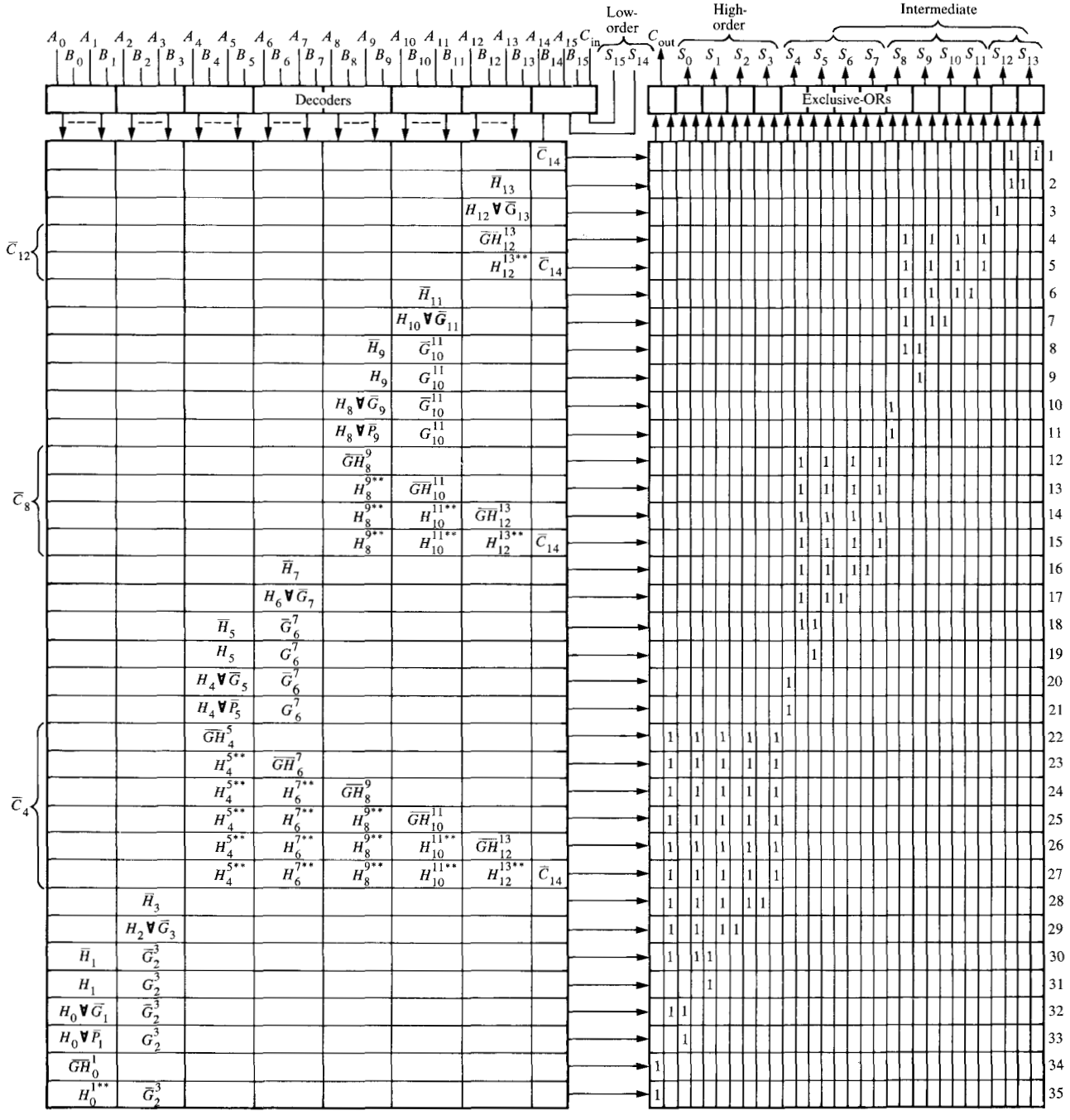

Figure 9(a) 16-bit adder using four-input and five-input special decoders: PLA format.

which generates seven elementary symmetric functions representing the combined input values ranging from 0 to 6. Any adder function of the four inputs can be generated from a combination of the seven decoder outputs. By contrast, a conventional two-input decoder assumes relative input weights of 8, 4, 2, and 1, requiring 16 outputs ranging in value from 0 to 15.

Figure 8 compares two-input and four-input special decoders showing the generated outputs. It is noted that replacing a pair of two-input special decoders with one four-input special decoder increases the number of decoder outputs from six to seven. By contrast, with conventional decoders, the number of outputs doubles—from eight to 16.

$$S_{15} = (H_{15} \forall C_{in})$$
$$S_{14} = (H_{14} \forall (G_{15} + P_{15} \cdot C_{in}))$$
$$S_{13} = \{(\bar{H}_{13})\} \forall \{\bar{C}_{14}\}$$
$$S_{12} = \{(H_{12} \forall \bar{G}_{13})\} \forall \{(\bar{H}_{13}) + \bar{C}_{14}\}$$
$$S_{11} = \{(\bar{H}_{11})\} \forall \{\bar{C}_{12}\}$$
$$S_{10} = \{(H_{10} \forall \bar{G}_{11})\} \forall \{(\bar{H}_{11}) + \bar{C}_{12}\}$$
$$S_9 = \begin{Bmatrix} (\bar{H}_9) \cdot (\bar{G}_{10}^{11}) \\ + (H_9) \cdot (G_{10}^{11}) \end{Bmatrix} \forall \begin{Bmatrix} (H_{10} \forall \bar{G}_{11}) \\ + (\bar{H}_{11}) + \bar{C}_{12} \end{Bmatrix}$$
$$S_8 = \begin{Bmatrix} (H_8 \forall \bar{G}_9) \cdot (\bar{G}_{10}^{11}) \\ + (H_8 \forall \bar{P}_9) \cdot (G_{10}^{11}) \end{Bmatrix} \forall \begin{Bmatrix} (\bar{H}_9) \cdot (\bar{G}_{10}^{11}) + (H_{10} \forall \bar{G}_{11}) \\ + (\bar{H}_{11}) + \bar{C}_{12} \end{Bmatrix}$$
$$S_7 = \{(\bar{H}_7)\} \forall \{\bar{C}_8\}$$
$$S_6 = \{(H_6 \forall \bar{G}_7)\} \forall \{(\bar{H}_7) + \bar{C}_8\}$$
$$S_5 = \begin{Bmatrix} (\bar{H}_5) \cdot (\bar{G}_6^7) \\ + (H_5) \cdot (G_6^7) \end{Bmatrix} \forall \begin{Bmatrix} (H_6 \forall \bar{G}_7) \\ + (\bar{H}_7) + \bar{C}_8 \end{Bmatrix}$$
$$S_4 = \begin{Bmatrix} (H_4 \forall \bar{G}_5) \cdot (\bar{G}_6^7) \\ + (H_4 \forall \bar{P}_5) \cdot (G_6^7) \end{Bmatrix} \forall \begin{Bmatrix} (\bar{H}_5) \cdot (\bar{G}_6^7) + (H_6 \forall \bar{G}_7) \\ + (\bar{H}_7) + \bar{C}_8 \end{Bmatrix}$$
$$S_3 = \{(\bar{H}_3)\} \forall \{\bar{C}_4\}$$
$$S_2 = \{(H_2 \forall \bar{G}_3)\} \forall \{(\bar{H}_3) + \bar{C}_4\}$$
$$S_1 = \begin{Bmatrix} (\bar{H}_1) \cdot (\bar{G}_2^3) \\ + (H_1) \cdot (G_2^3) \end{Bmatrix} \forall \begin{Bmatrix} (H_2 \forall \bar{G}_3) \\ + (\bar{H}_3) + \bar{C}_4 \end{Bmatrix}$$
$$S_0 = \begin{Bmatrix} (H_0 \forall \bar{G}_1) \cdot (\bar{G}_2^3) \\ + (H_0 \forall \bar{P}_1) \cdot (G_2^3) \end{Bmatrix} \forall \begin{Bmatrix} (\bar{H}_1) \cdot (\bar{G}_2^3) + (H_2 \forall \bar{G}_3) \\ + (\bar{H}_3) + \bar{C}_4 \end{Bmatrix}$$
$$C_{out} = \begin{Bmatrix} (\bar{G}\bar{H}_0^1) \\ + (H_0^{1**}) \cdot (\bar{G}_2^3) \end{Bmatrix} \forall \begin{Bmatrix} (H_0 \forall \bar{G}_1) \cdot (\bar{G}_2^3) \\ + (\bar{H}_1) \cdot (\bar{G}_2^3) + (H_2 \forall \bar{G}_3) \\ + (\bar{H}_3) + \bar{C}_4 \end{Bmatrix}$$

$$\bar{C}_{14} = \begin{pmatrix} \bar{P}_{14} \\ + \bar{G}_{14} \cdot \bar{P}_{15} \\ + \bar{G}_{14} \cdot \bar{G}_{15} \cdot \bar{C}_{in} \end{pmatrix}$$
$$\bar{C}_{12} = (\bar{G}\bar{H}_{12}^{13}) + (H_{12}^{13**}) \cdot (\bar{C}_{14})$$
$$\bar{C}_8 = (\bar{G}\bar{H}_8^9) + (H_8^{9**}) \cdot (\bar{G}\bar{H}_{10}^{11}) + (H_8^{9**}) \cdot (H_{10}^{11**}) \cdot (\bar{G}\bar{H}_{12}^{13}) + (H_8^{9**}) \cdot (H_{10}^{11**}) \cdot (H_{12}^{13**}) \cdot (\bar{C}_{14})$$
$$\bar{C}_4 = (\bar{G}\bar{H}_4^5) + (H_4^{5**}) \cdot (\bar{G}\bar{H}_6^7) \vdots + (H_4^{5**}) \cdot \ldots \cdot (\bar{G}\bar{H}_{12}^{13}) + (H_4^{5**}) \cdot \ldots \cdot (H_{12}^{13**}) \cdot (\bar{C}_{14})$$

Figure 9(b) 16-bit adder using four-input and five-input special decoders: equations.

The width of the AND array can be further reduced by customizing each decoder to produce only those functions that the product terms require, particularly for decoders with a large number of inputs. For example, an eight-input special decoder which accepts four adjacent pairs of adder inputs of relative weights 8, 8, 4, 4, 2, 2, 1, and 1, produces 31 elementary symmetric functions representing weights 0 through 30. However, the number of different functions of these inputs actually needed by the product terms of a 32-bit adder varies from six to ten. In other words, the width of the AND array is actually less for eight-input custom decoders than for decoders with fewer inputs. At the same time, the number of product terms is also reduced. The reduction of the AND array in both dimensions results in a set of more complex functions produced by the custom decoders.

A custom decoder is particularly useful for the low-order inputs with which the input carry may be combined in one decoder.

Adders using four- and five-input decoders

The 16-bit adder defined in Figs. 9(a) and (b) will be used to demonstrate the effect of using four- and five-input decoders for adder designs. A five-input custom decoder is used for the inputs comprising the input carry, C_{in}, and the two pairs of inputs to the low-order bit positions 14 and 15. The remaining decoders accept four inputs each, comprising pairs of inputs of adjacent bit positions.

Adder outputs are again grouped in strings of contiguous sum bits. The low-order string includes the two positive low-order sum bits, S_{15} and S_{14}. They exit di-

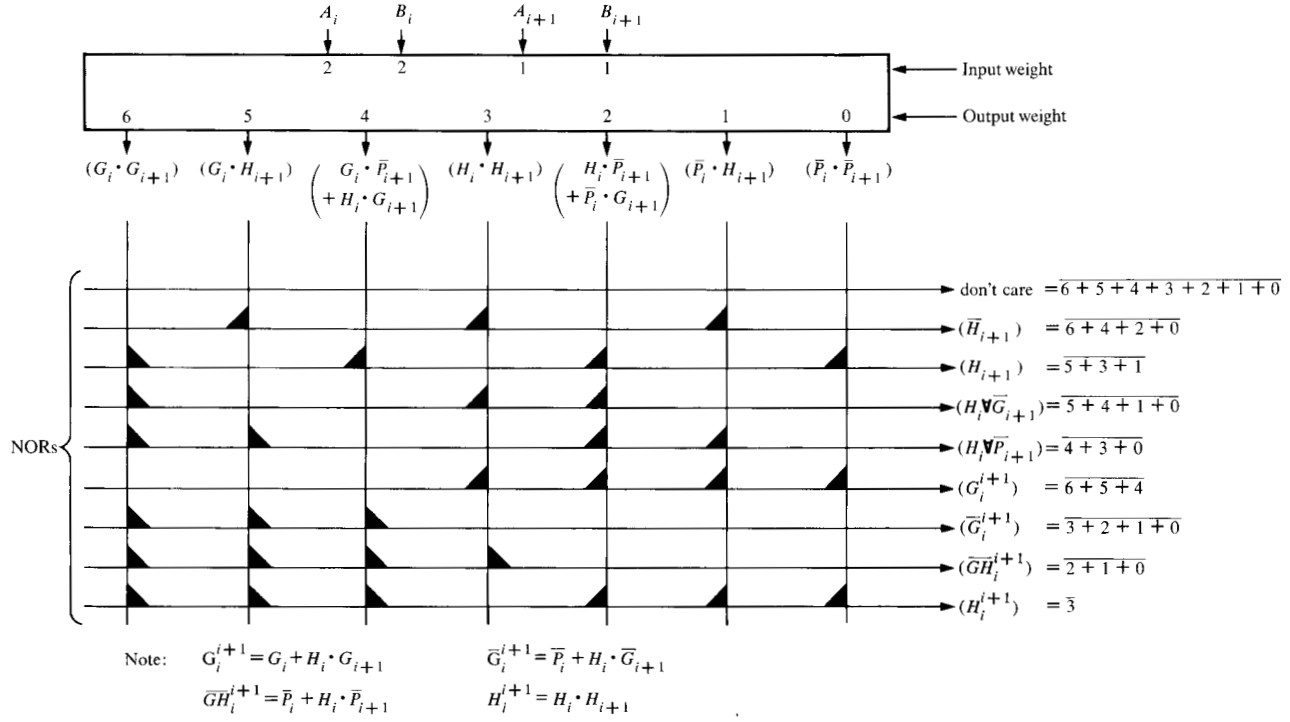

Figure 10 Personalization of AND array functions controlled by a four-input symmetric function generator.

rectly from the custom five-input decoder, together with the output carry from the string, C_{14}, which enters the AND array to help generate the carries $\bar{C}_{12}$, $\bar{C}_8$, and $\bar{C}_4$. Succeeding strings of sum bits comprise (S_{13} and S_{12}), (S_{11}, S_{10}, S_9, and S_8), (S_7, S_6, S_5, and S_4), and (S_3, S_2, S_1, and S_0).

The general equations for the sums and the carries can be derived in a manner similar to those for the adder using a PLA with two-input decoders. A few differences are noted:

1. A product term is expressed as the AND of functions of the new decoders. An entry in the AND array is a function of the respective decoder inputs. The four-input decoders may still be conventional, with an entry in the AND array readily converted to a personalized 16-bit cell. The conversion follows from an extension of Fig. 5 to a four-bit decoder. However, when special decoders are used, the conversion of an entry in the AND array to a personalized cell of fewer than 16 bits requires different rules.
2. The double asterisk attached to the strict propagate function, H_a^{a+1**}, means that $\overline{GH}_a^{a+1}$ may be used as don't-care conditions; e.g., $\bar{G}_a^{a+1}$ may be substituted for H_a^{a+1}. This simplifies personalization and may also reduce decoder outputs, as will be subsequently demonstrated. This principle was applied earlier in simpler form to single-bit propagate functions, where P or $\bar{G}$ was substituted for H, and is extendable to multi-bit propagate functions.
3. It can be noted in Fig. 9(b) that the left bracket of an equation for a high-order sum of a string, e.g., S_4, cannot share product terms with the carry from the string, either $\bar{C}_4$ or C_4. Therefore, $\bar{C}_4$ is arbitrarily selected to produce successive sum outputs of the same polarity. This is in contrast to Fig. 7(a), where such product term sharing requires alternating polarities of strings. To enable this kind of product term sharing, the left bracket of the high-order sum of a string, such as S_4, would be expressed as

$$\left\{ \begin{array}{l} (\bar{H}_4 \cdot \bar{P}_5 + H_4 \cdot G_5) \\ + (\bar{H}_4 \cdot H_5^{**}) \cdot (\bar{G}_6^7) \\ + (H_4 \cdot H_5^{*}) \cdot (G_6^7) \end{array} \right\},$$

which takes three product terms instead of two. The product term, $(H_4 \cdot H_5^*) \cdot (G_6^7)$, could then be shared between S_4 and the carry-look-ahead expression for C_4, but without any advantage in total number of product terms and with the possible disadvantage of alternating polarities of strings. However, such sharing be-

comes economical for string sizes of six or greater. The optimized strings of the 16-bit adder of Fig. 9 calls for string sizes of only four and two.

An empirical procedure for optimally assigning string sizes, similar to one described earlier, results in the following number of product terms (and string sizes): for an 8-bit adder, 13 product terms (string sizes 4, 2, and 2); for a 16-bit adder, 35 product terms (string sizes 4, 4, 4, 2, and 2); and for a 32-bit adder, 99 product terms (string sizes 6, 6, 6, 4, 4, 4, and 2).

Figure 10 illustrates the bit personalization for the various functions of a four-bit special decoder. The decoder is an elementary symmetric function generator producing positive outputs and driving an AND array consisting of NORs. Note that a maximum of only six switching devices needs to be provided for personalizing a function because the function requiring all seven columns to be connected is never used. It is assumed that a switching device is located between two adjacent columns and can be shared between the two columns. Therefore, six devices can be shared by the seven columns, with each device connected to its left column (connection pointing left), its right column (connection pointing right), or neither column (no connection shown). No devices need be provided between adjacent sets of columns. Also note that an elementary symmetric function is not connected if it is included in the desired function, corresponding to the rule for a conventional decoder with positive outputs driving an AND array consisting of NORs. If the AND array is implemented with ANDs, the decoder should produce complement outputs.

Other expressions may be substituted for some of those in the AND array of Fig. 9 to reduce the number of device connections. For example, the complement of the inclusive two-bit propagate function $\bar{G}_i^{i+1}$ may be substituted for the strict propagate function H_i^{i+1} without affecting the outputs of the adder. The substitution reduces the maximum number of connections in Fig. 10 from six to four. Rearranging the outputs of the decoder permits reducing the number of devices that need to be provided, even assuming that a device can be shared only between its two adjacent columns. As shown more explicitly in Fig. 11, only four devices are needed for bit positions 10 and 11, and five devices for bit positions 8 and 9, to personalize the respective functions.

The five-input custom decoder for inputs A_{14}, B_{14}, A_{15}, B_{15}, and C_{in} produces the two low-order sum bits directly, as well as the carry C_{14} driving the AND array, as shown in Fig. 12. The positive C_{14} is intended for the NOR implementation of the AND array in Fig. 9 where $\bar{C}_{14}$ is needed

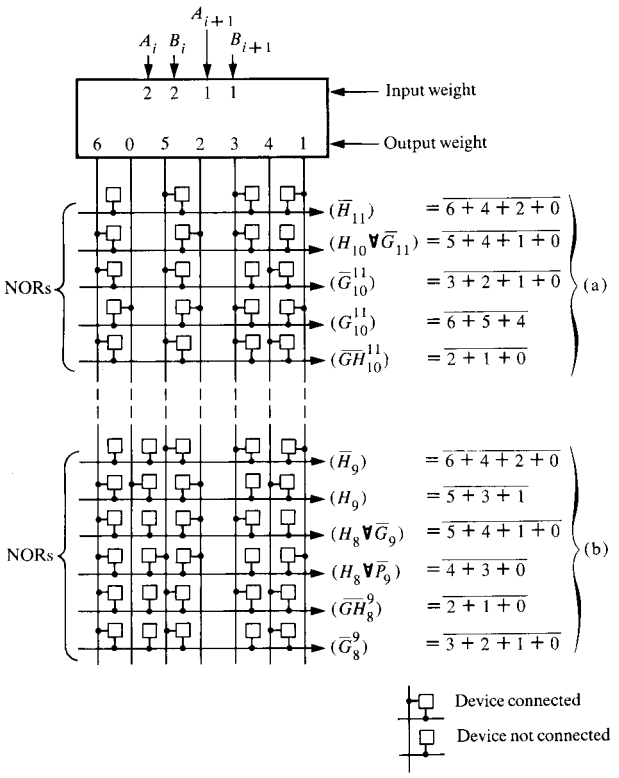

Figure 11 Personalization of AND array functions using (a) four devices with a maximum of four connections, and (b) five devices with a maximum of four connections.

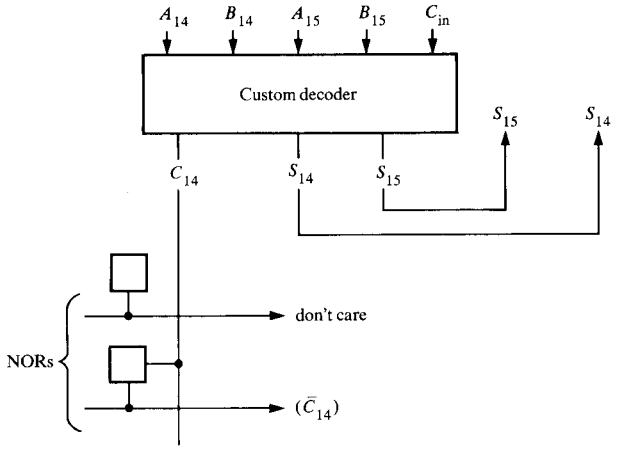

Figure 12 Custom decoder for low-order bit positions of 16-bit adder.

for several product terms. If the AND array is implemented with ANDs, the custom decoder should generate $\bar{C}_{14}$.

The width of the AND array reduces to 50 columns using the special decoders consisting of seven elementary symmetric function generators with seven columns each and the custom decoder with one column for the AND array.

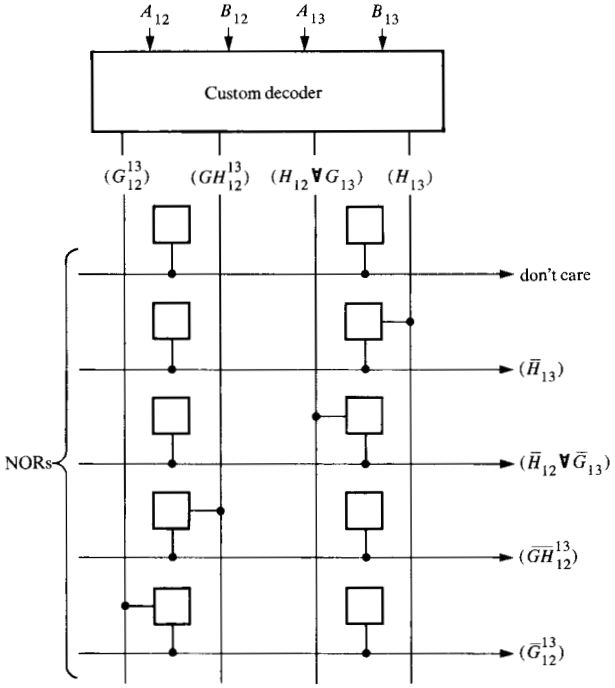

Figure 13 Custom decoder for bit positions 12 and 13 of 16-bit adder.

If custom decoders replace the elementary symmetric function generators, the width of the AND array is further reduced. Moreover, still fewer devices are needed and only one device connection is made at the intersection of an AND array row with the outputs of a custom decoder. For example, Fig. 13 shows the custom decoder outputs for bit positions 12 and 13 of the 16-bit adder of Fig. 9, as well as the AND array personalization for the five unique functions the decoder must provide. Again, the decoder generates complement functions to drive a NOR implementation of the AND array. Based on the number of unique functions needed, the total width of the AND array of the 16-bit adder is reduced to 37.

The 16-bit dedicated PLA adder can be further compressed horizontally and vertically using schemes which eliminate array sections of unconnected devices [8]. It should be noted in Fig. 9(a) that the arrays are rather sparsely populated with entries (representing connected devices). For example, the first row contains entries only in the columns of the low-order decoder, in the AND array, and of the sum bits S_{12} and S_{13}, in the OR array. A compressed 16-bit adder is illustrated in Fig. 14. First, the OR array is split into a left and a right part to permit an AND array row to be shared by two product terms. The left and right product terms sharing a row are shown separated with a heavy vertical line. Second, OR array columns are also shared between pairs of outputs, the split in the column being indicated by a heavy horizontal line. Third, the inputs and outputs are arranged to enable large sections of unused ends of rows of columns to be truncated. The number of AND array rows is thus reduced to 22, and the combined number of columns is reduced to 55. The latter are composed of ten columns for the left OR array, eight for the right OR array, and 6 + 4 + 6 + 5 + 6 + 5 + 4 + 1 for the custom decoder outputs driving the AND array. (Note that $\tilde{G}_i^{i+1}$ can be substituted for H_i^{i+1**}.

Adders using decoders with larger number of inputs

Using custom decoders, it is possible to continue the trade-off between decoder complexity and array size. For example, with four adder bit position inputs to a decoder, custom decoders of eight and nine inputs may be used. The nine-input decoder would be assigned to the low-order four-bit positions plus the input carry C_{in}. The decoder would generate the low-order four sum bits directly as well as the signal representing the carry out of the decoder inputs to drive the AND array.

When optimum string sizes are used, the number of product terms (and string sizes) needed for an eight-bit, 16-bit, and 32-bit adder is six (string sizes 4 and 4), 19 (string sizes 4, 4, 4, and 4) and 54 (string sizes 8, 8, 4, 4, 4, and 4), respectively.

If carried to the limit in which all inputs to the adder enter a single custom decoder, the "decoder" becomes a custom designed adder without the need of arrays.

Summary and conclusions

It has been demonstrated that one-cycle addition of a wide data path can be effectively implemented with one pass through a PLA. Effectiveness is measured in the number of product terms needed, since that number relates to the chip area required by a PLA as well as to the delay through the PLA AND and OR arrays. The adder is designed to take advantage of two-bit input decoders and Exclusive-OR outputs—two features which can presently be incorporated in a standard PLA.

Adder equations with carry-look-ahead have been adapted to the PLA features to use product terms sparingly and to maximize sharing of product terms among different functions of product terms. For example, a string of contiguous sum bits is expressed using a common carry of one polarity so that the product terms representing the carry are shared by the several sum bits. The development of a procedure that determines the optimum string sizes into which the adder sum bits are grouped to minimize the total number of product terms has also been demonstrated.

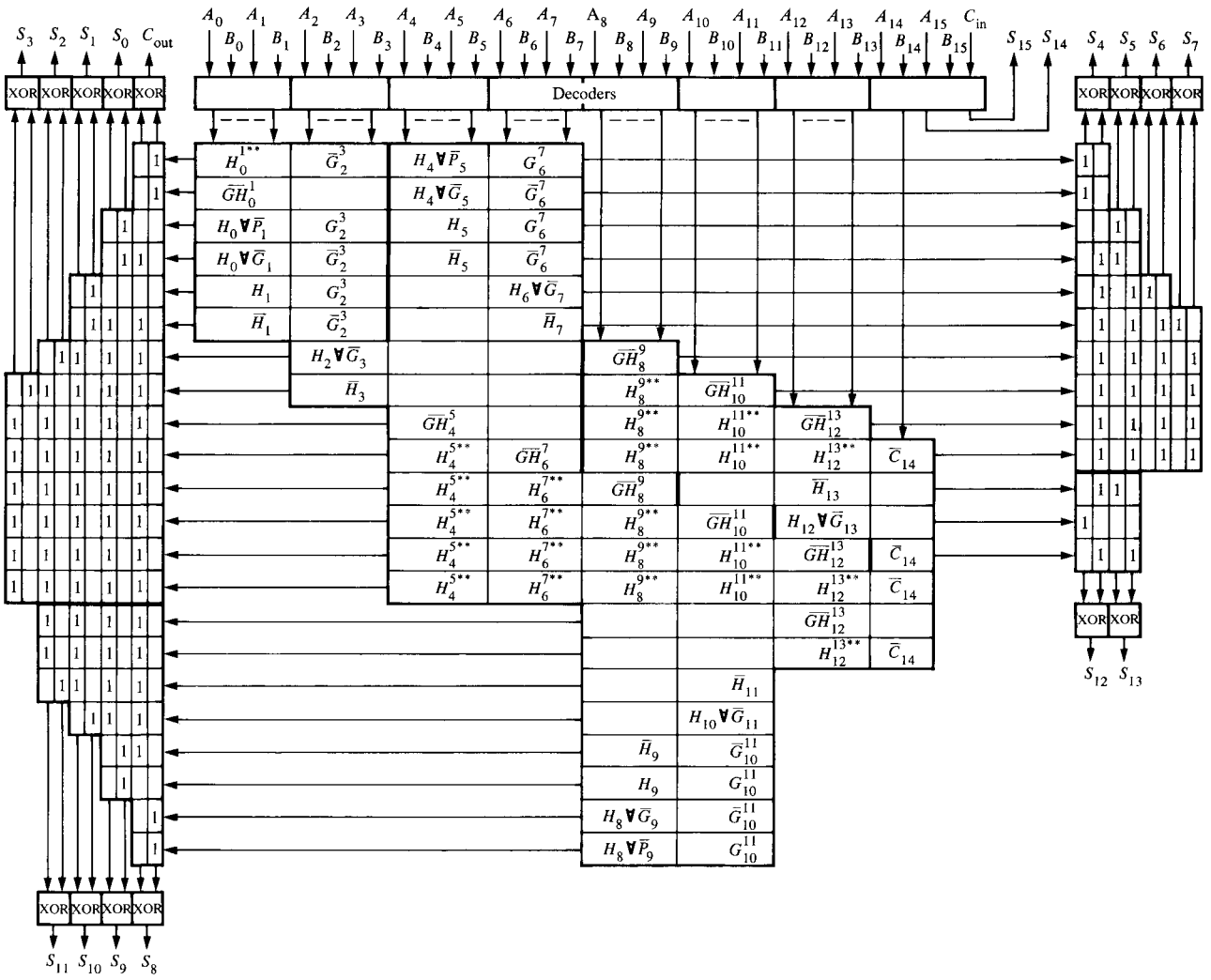

Figure 14 Compressed 16-bit PLA adder.

A standard PLA will normally implement a number of functions, one of which may be an adder. With LSI, PLAs will increasingly be used as macros on a chip, tailored to specific functional needs. If a PLA is dedicated to an adder, further efficiencies can be gained. Input decoders with more than two inputs can further reduce the number of product terms needed. At the same time, the width of the AND array of a PLA, the dimension which measures the number of decoder outputs, can be reduced by substituting special decoders to produce functions relevant to addition. As a result, both the height and the width of a PLA adder can be significantly reduced.

A dedicated PLA adder can be further compressed in size by splitting the OR array of the PLA into two parts with the single AND array between them. Many of the AND array rows, which normally contain a single product term, can thus be shared between two product terms. Also, an OR array column can be split to contain two sums of product terms, instead of one, by providing distinct outputs at the top and bottom of the column.

References

1. W. N. Carr and J. P. Mize, *MOS/LSI Design and Application*, McGraw-Hill Book Co., Inc., New York, 1972.
2. J. C. Logue, N. F. Brickman, F. Howley, J. W. Jones, and W. W. Wu, "Hardware Implementation of a Small System in Programmable Logic Arrays," *IBM J. Res. Develop.* **19,** 110 (1975).
3. A. Weinberger, "Parallel Adders Using Standard PLAs," *Proceedings of the Fourth Symposium on Computer Arithmetic*, Santa Monica, CA, October 25–27, 1978.
4. M. Flinders, P. L. Gardner, J. F. Minshull, and R. J. Llewelyn, "Functional Memory as a General Purpose System Technology," *Proceedings of the IEEE Computer Group Conference*, June 1970, pp. 314–324.
5. A. Weinberger, "Functional Memory Using Multistate Associative Cells," U.S. Patent #3,761,902, 1973.

6. A. Weinberger, "Device Sharing in Array Logic," *IBM Tech. Disc. Bull.* **19,** 1357 (1976).
7. J. W. Jones, "Array Logic Macros," *IBM J. Res. Develop.* **19,** 120 (1975).
8. A. Weinberger, "Logic Array with Multiple Read-Out Tables," U.S. Patent #3,975,623, 1976.

Received August 24, 1978; revised October 23, 1978

The author is located at the IBM Data Systems Division laboratory, Poughkeepsie, New York 12602.