Parallel Adders Using Standard PLAs

Arnold Weinberger

International Business Machines Corporation
B89/707, PO Box 390
Poughkeepsie, New York 12602

ABSTRACT

PLA adders are described that add in one cycle and require a reasonable number of product terms for an 8, 16, or even a 32-bit adder. A procedure is also described for minimizing the number of product terms for any size adder.

INDEX TERMS

Programmable Logic Array (PLA), input decoders, output exclusive-ORs, product term minimization, adder, carry-look-ahead.

INTRODUCTION

Programmable Logic Arrays(PLAs) [1] have been successfully applied to 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. To implement a large adder in one cycle required too many product terms to be economical.

This paper describes single-cycle adder designs for standard PLAs that minimize the number of product terms to acceptable levels even for 16 and 32-bit adders. The PLAs have two features that reduce the number of product terms:

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

The adder equations are expressed in a manner to take advantage of the two features and of various methods of sharing product terms. In particular, a string of several adjacent sum bits are expressed in terms of the carry into the string, using carry-look-ahead. A procedure is developed to optimize string sizes to further reduce the number of product terms.

PLAs

A PLA consists basically of two arrays in series, an AND array and an OR array, as shown in Figure 1. The array names, AND-OR, describe the generic logic levels of the SEARCH-READ arrays of an associative table [2]. The two arrays may be implemented with other types of logic besides AND-OR. A widely-used logic is NOR-NOR implemented with MOS technology.

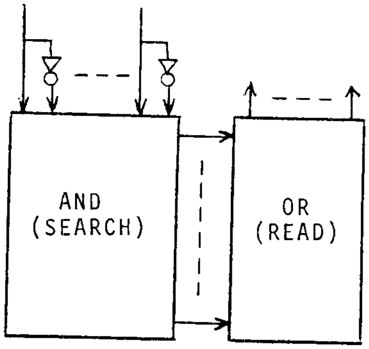

Figure 1. PLA (Programmable Logic Array)

The generic AND (SEARCH) array produces an array of product terms of the inputs to the PLA. Each product term is the AND of the inputs, as in (1). Each input enters the AND

$$\text{Product Term}= f1(A)\bullet f2(B)\bullet \ldots \qquad (1)$$

in one of three states: true, complement, or don't care. The true and complement lines of an input intersect the AND array at each AND with two bits which are personalized for one of the three states. The personalization is shown in Figure 2a when the generic AND (SEARCH word) is implemented with a real AND and in Figure 2b when implemented with a NOR.

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

CH1412-6/78/0000-0116$00.75 (c) 1978 IEEE

is the real OR of selected product terms if the array is comprised of real ORs, as in Figure 3. If the array is comprised of NORs, each output is the NOR of selected product terms.

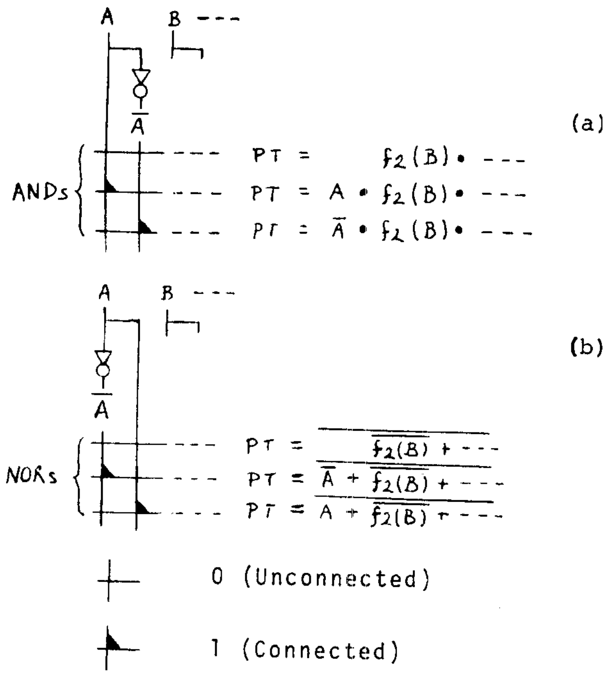
A
B ---
$\overline{A}$
(a)
ANDs
PT = f2(B)• ---
PT = A • f2(B)• ---
PT = $\overline{A}$ • f2(B)• ---
A
B ---
$\overline{A}$
(b)
NORs
PT = $\overline{\overline{f2(B)} + ---}$
PT = $\overline{\overline{A} + \overline{f2(B)} + ---}$
PT = $\overline{A + \overline{f2(B)} + ---}$
0 (Unconnected)
1 (Connected)

Figure 2. Personalization of AND (SEARCH) Array Using (a) Real ANDs or (b) NORs

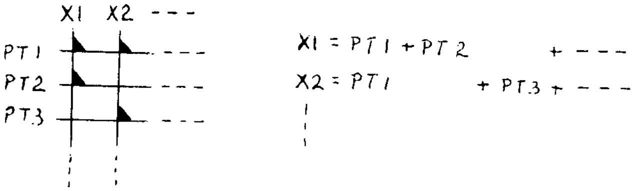

Figure 3. Personalization of OR(READ) Array

It is well known that a function of n variables can be expressed as a sum of a subset of the minterms (or as a product of a subset of maxterms). There are 2^n minterms (or maxterms) and therefore 2^{2n} possible subsets of minterms (or maxterms).

A complete set of minterms (maxterms) corresponds to the positive (negative) outputs of an n-bit decoder. Figures 2a and b can be interpreted as providing a 1-bit decoder for input A: Figure 2a provides the two maxterms A and $\overline{A}$, while Figure 2b provides the corresponding two minterms $\overline{A}$ and A.

The personalized 2-bit cell at the intersection of a product term with the 1-bit decoder outputs corresponds to selecting the subset of minterms (maxterms) to comprise the desired function. Figure 4 shows the four possible functions of input A, of which three are used. Figure 4a 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, Figure 4b 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 complement of the OR function of A is actually shown, to correspond to the true function for a generic AND when implemented with a NOR.

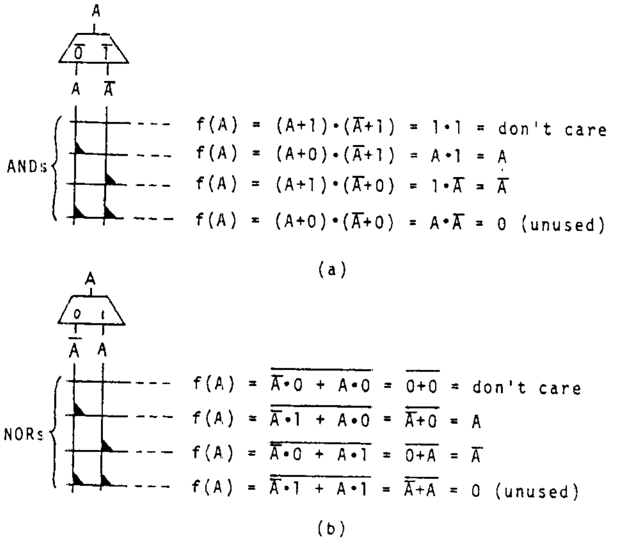
A
0 1
A $\overline{A}$
ANDs
f(A) = (A+1)•($\overline{A}$+1) = 1•1 = don't care
f(A) = (A+0)•($\overline{A}$+1) = A•1 = A
f(A) = (A+1)•($\overline{A}$+0) = 1•$\overline{A}$ = $\overline{A}$
f(A) = (A+0)•($\overline{A}$+0) = A•$\overline{A}$ = 0 (unused)
(a)
A
0 1
$\overline{A}$ A
NORs
f(A) = $\overline{\overline{A}•0 + A•0}$ = $\overline{0+0}$ = don't care
f(A) = $\overline{\overline{A}•1 + A•0}$ = $\overline{\overline{A}+0}$ = A
f(A) = $\overline{\overline{A}•0 + A•1}$ = $\overline{0+A}$ = $\overline{A}$
f(A) = $\overline{\overline{A}•1 + A•1}$ = $\overline{\overline{A}+A}$ = 0 (unused)
(b)

Figure 4. One-input functions using a 1-input decoder and a personalized 2-bit cell with (a) complement decode outputs and maxterm personalization or (b) true decode outputs and minterm personalization.

The number of product terms is significantly reduced by substituting 2-bit decoders for a pair of 1-bit decoders [3]. The total number of decoder outputs remains the same. The product term now represents the AND of functions of pairs of inputs, as in (2).

Product Term= f1(A1,B1)•f2(A2,B2)•... (2)

Figure 5 shows the 16 possible functions of inputs A and B, of which 15 may be used. Figure 5a shows the possible products of maxterms, while Figure 5b 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 (3).

$$\text{Product Term}= \overline{\overline{f1(A1,B1)}+\overline{f2(A2,B2)}+\ldots} \quad (3)$$

This corresponds to Figure 4b for 1-bit decoders.

Two-input decoders have already been applied to a standard PLA [4] and will be shown to be particularly useful for adders.

Another economizing PLA feature is using XOR outputs [5]. Pairs of OR array outputs are XORed to produce a single PLA output.

Figure 6 shows the PLA expanded to include 2-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 Cin to produce a sum $S(S_0,\ldots, S_{n-1})$ and an output carry $Cout(=C_0)$. Using single-bit-position functions:

$$G_i = A_i \bullet B_i, \quad P_i = A_i + B_i, \quad H_i = A_i \forall B_i,$$

a carry from any bit position can be expressed directly in terms of single-b bit-position functions and the Cin, as in (4) and (5).

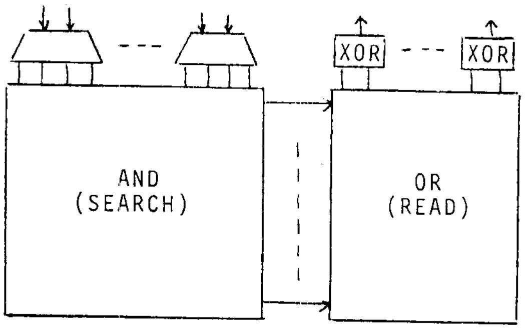

Figure 6. PLA with 2-Input Decoders and XOR Outputs

$$\begin{aligned} C_i = {} & G_i \\ & + H_i^{*} \bullet G_{i+1} \\ & \;\vdots \\ & + H_i^{*} \bullet \ldots \bullet H_{n-2}^{*} \bullet G_{n-1} \\ & + H_i^{*} \bullet \ldots \bullet H_{n-1}^{*} \bullet Cin \end{aligned} \quad (4)$$

$$\begin{aligned} \overline{C}_i = {} & \overline{P}_i \\ & + H_i^{**} \bullet \overline{P}_{i+1} \\ & + H_i^{**} \bullet \ldots \bullet H_{n-2}^{**} \bullet \overline{P}_{n-1} \\ & + H_i^{**} \bullet \ldots \bullet H_{n-1}^{**} \bullet \overline{Cin} \end{aligned} \quad (5)$$

where H^{*} means either H or P may be used
H^{**} means either H or $\overline{G}$ may be used

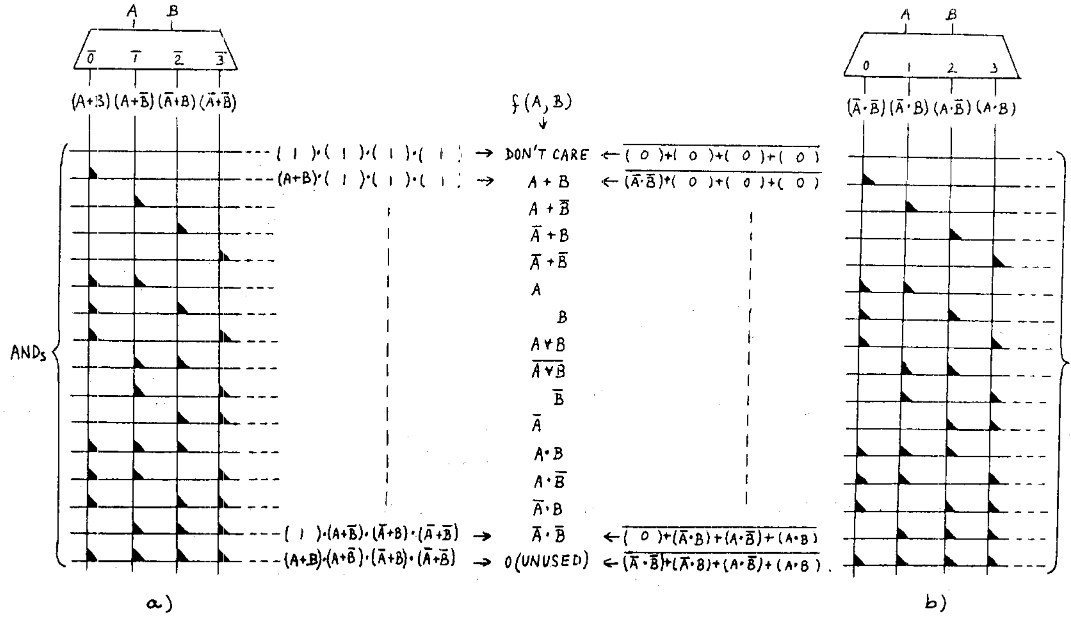
A
B
$\overline{0}$
$\overline{1}$
$\overline{2}$
$\overline{3}$
$(A+B)$ $(A+\overline{B})$ $(\overline{A}+B)$ $(\overline{A}+\overline{B})$
ANDs
$f(A,B)$
$(1)\cdot(1)\cdot(1)\cdot(1)$ → DON'T CARE ← $\overline{(0)+(0)+(0)+(0)}$
$(A+B)\cdot(1)\cdot(1)\cdot(1)$ → $A+B$ ← $\overline{(\overline{A}\cdot\overline{B})+(0)+(0)+(0)}$
$A+\overline{B}$
$\overline{A}+B$
$\overline{A}+\overline{B}$
A
B
$A \forall B$
$\overline{A \forall B}$
$\overline{B}$
$\overline{A}$
$A\cdot B$
$A\cdot\overline{B}$
$\overline{A}\cdot B$
$(1)\cdot(A+\overline{B})\cdot(\overline{A}+B)\cdot(\overline{A}+\overline{B})$ → $\overline{A}\cdot\overline{B}$ ← $\overline{(0)+(\overline{A}\cdot B)+(A\cdot\overline{B})+(A\cdot B)}$
$(A+B)\cdot(A+\overline{B})\cdot(\overline{A}+B)\cdot(\overline{A}+\overline{B})$ → 0 (UNUSED) ← $\overline{(\overline{A}\cdot\overline{B})+(\overline{A}\cdot B)+(A\cdot\overline{B})+(A\cdot B)}$
A
B
0
1
2
3
$(\overline{A}\cdot\overline{B})$ $(\overline{A}\cdot B)$ $(A\cdot\overline{B})$ $(A\cdot B)$
NORs
a)
b)

Figure 5. Two-input functions using a 2-bit decoder and a personalized 4-bit cell with (a) complement decode outputs and maxterm personalization or (b) true decode outputs and minterm minimization.

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

$$\begin{aligned} S_i &= H_i \veebar C_{i+1} = H_i \veebar (G^j_{i+1} + H^j_{i+1} \bullet C_{j+1}) \\ &= (H_i \veebar G^j_{i+1}) \veebar (H^j_{i+1} \bullet C_{j+1}) \qquad (6) \\ &= (H_i \veebar \overline{G}^j_{i+1}) \veebar (\overline{H}^j_{i+1} + \overline{C}_{j+1}) \end{aligned}$$

$$\begin{aligned} \overline{S}_i &= H_i \veebar \overline{C}_{i+1} = H_i \veebar (\overline{GH}^j_{i+1} + H^j_{i+1} \bullet \overline{C}_{j+1}) \\ &= (H_i \veebar \overline{GH}^j_{i+1}) \veebar (H^j_{i+1} \bullet \overline{C}_{j+1}) \qquad (7) \\ &= (H_i \veebar GH^j_{i+1}) \veebar (\overline{H}_{i+1} + C_{j+1}) \end{aligned}$$

where G^j_{i+1} = carry-generate condition for bit group i+1 through j (high-to-low order)

H^j_{i+1} = strict carry-propagate condition (mutually exclusive with G^j_{i+1})

$GH^j_{i+1} = G^j_{i+1} + H^j_{i+1}$ = inclusive carry-propagate condition

$$\begin{aligned} G^j_{i+1} = {} & G_{i+1} \\ & + H^{*}_{i+1} \bullet G_{i+2} \\ & \vdots \\ & + H^{*}_{i+1} \bullet \ldots \bullet H^{*}_{j-1} \bullet G_j \end{aligned}$$

$$\begin{aligned} \overline{G}^j_{i+1} = {} & \overline{P}_{i-1} \\ & + H^{**}_{i+1} \bullet \overline{P}_{i+2} \\ & \vdots \\ & + H^{**}_{i+1} \bullet \ldots \bullet \overline{P}_{j-1} \\ & + H^{**}_{i+1} \bullet \ldots \bullet \overline{H}^{**}_{j-1} \bullet \overline{G}_j \end{aligned}$$

$$H^j_{i+1} = H_{i+1} \bullet \ldots \bullet H_j$$

$$\overline{H}^j_{i+1} = \overline{H}_{i+1} + \ldots + \overline{H}_j$$

$$\begin{aligned} GH^j_{i+1} = {} & G_{i+1} \\ & + H^{*}_{i+1} \bullet G_{i+1} \\ & \vdots \\ & + H^{*}_{i+1} \bullet \ldots \bullet G_{j-1} \\ & + H^{*}_{i+1} \bullet \ldots \bullet H^{*}_{j-1} \bullet P_j \end{aligned}$$

$$\begin{aligned} \overline{GH}^j_{i+1} = {} & \overline{P}_{i+1} \\ & + H^{**}_{i+1} \bullet \overline{P}_{i+2} \\ & \vdots \\ & + H^{**}_{i+1} \bullet \ldots \bullet H^{**}_{j-1} \bullet \overline{P}_j \end{aligned}$$

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

$$\begin{aligned} \text{Cout} &= \overline{\overline{GH}^j_0 + H^j_0 \bullet \overline{C}_{j+1}} = \overline{\overline{GH}^j_0 \veebar H^j_0 \bullet \overline{C}_{j+1}} \qquad (8) \\ &= \{\overline{GH}^j_0\} \veebar \{\overline{H}^j_0 + C_{j+1}\} \end{aligned}$$

$$\begin{aligned} \overline{\text{C}}\text{out} &= \overline{G^j_0 + H^j_0 \bullet C_{j+1}} = \overline{G^j_0 \veebar H^j_0 \bullet C_{j+1}} \qquad (9) \\ &= \{G^j_0\} \veebar \{\overline{H}^j_0 + \overline{C}_{j+1}\} \end{aligned}$$

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

PLA Adder Designs

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

Addend and augend of the same bit position, Ai and Bi, enter a common decoder, so that the intersection of an AND with the decoder outputs can produce a function of Ai and Bi, i.e., Gi, Pi, Hi, or their complements. The input carry Cin enters as the sole input to a decoder. (For uniformity, a 2-input decoder will be provided for Cin with one input unused.)

Sums are generated in strings, each string as a function of a common carry into the string, using Eqs. (6) and (7). A positive string of sum bits is shown in (10), and a negative string in (11). The output carry of the string is generated as a sum of products according to (4) or (5). For the purpose of counting product terms, a string includes the sum bits and the output carry of the string. The output carry of one string serves as the input carry to the next higher string.

$$\left.\begin{aligned}
S_j &= \left\{ \overline{H}_j \right\} \veebar \left\{ \overline{C}_{j+1} \right\} \\
S_{j-1} &= \left\{ \begin{aligned} &\overline{H}_{j-1} \bullet \overline{G}_j \\ &+H_{j-1} \bullet G_j \end{aligned} \right\} \veebar \left\{ \overline{H}_j + \overline{C}_{j+1} \right\} \\
&\vdots \\
S_i &= \left\{ \begin{aligned}
&\overline{H}_i \bullet \overline{P}_{i+1} \\
&+\overline{H}_i \bullet H^{**}_{i+1} \bullet \overline{P}_{i+2} \\
&\vdots \\
&+\overline{H}_i \bullet H^{**}_{i+1} \bullet \ldots \bullet \overline{P}_{j-1} \\
&+\overline{H}_i \bullet H^{**}_{i+1} \bullet \ldots \bullet H^{**}_{j-1} \bullet \overline{G}_j \\
&+H_i \bullet G_{i+1} \\
&+H_i \bullet H^{*}_{i+1} \bullet G_{i+2} \\
&\vdots \\
&+H_i \bullet H^{*}_{i+1} \bullet \ldots \bullet H^{*}_{j-1} \bullet G_j
\end{aligned} \right\} \veebar \left\{ \overline{H}_{i+1} + \ldots + \overline{H}_j + \overline{C}_{j+1} \right\}
\end{aligned}\right] \tag{10}$$

$$\left.\begin{aligned}
\overline{S}_j &= \left\{ \overline{H}_j \right\} \veebar \left\{ C_{j+1} \right\} \\
\overline{S}_{j-1} &= \left\{ \begin{aligned} &\overline{H}_{j-1} \bullet P_j \\ &+H_{j-1} \bullet \overline{P}_j \end{aligned} \right\} \veebar \left\{ \overline{H}_j + C_{j+1} \right\} \\
&\vdots \\
\overline{S}_i &= \left\{ \begin{aligned}
&\overline{H}_i \bullet G_{i+1} \\
&\overline{H}_i \bullet H^{*}_{i+1} \bullet G_{i+2} \\
&\vdots \\
&+\overline{H}_i \bullet H^{*}_{i+1} \bullet \ldots \bullet G_{j-1} \\
&+\overline{H}_i \bullet H^{*}_{i+1} \bullet \ldots \bullet H^{*}_{j-1} \bullet P_j \\
&+H_i \bullet \overline{P}_{i+1} \\
&+H_i \bullet H^{**}_{i+1} \bullet \overline{P}_{i+2} \\
&\vdots \\
&+H_i \bullet H^{**}_{i+1} \bullet \ldots \bullet H^{**}_{j-1} \bullet \overline{P}_j
\end{aligned} \right\} \veebar \left\{ \overline{H}_{i+1} + \ldots + \overline{H}_j + C_{j+1} \right\}
\end{aligned}\right] \tag{11}$$

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

A low-order string includes a product term representing the input carry Cin or $\overline{\text{Cin}}$, the low order sum bits implemented according to (10) or (11), and the product terms representing the output carry of the string according to (4) or (5). The indexes (j-1, j) become (n-1, in). Note that the high order sum of the string Si shares some of its product terms with the output carry of the string Ci, and $\overline{\text{Si}}$ shares product terms with $\overline{\text{Ci}}$. 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 Si and Ci (or $\overline{\text{Si}}$ and $\overline{\text{Ci}}$), the common factor Hi must be used and Pi (or $\overline{\text{GI}}$) cannot be substituted for it, i.e., Hi* (or Hi**) 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+5+...+(2K-1) for the sum bits (noting that some product terms are shared, e.g., $\overline{Hj}$), and 2 for the additional unique (non-shared) product terms comprising the output carry of the string. Eq. (12) expresses T_{low}, the number of unique product terms of low-order string for K>1.

$$T_{low} = 1+[1+2+5+---+(2K-1)]+2 = K^2+2 \qquad (12)$$

Eq. (12) also holds for K=1, when the low-order sum is generated acccording to (13) or (14).

$$S_{n-1} = H_{n-1}\cdot\overline{Cin} \veebar \overline{H_{n-1}}\cdot Cin \qquad (13)$$

$$\overline{S}_{n-1} = H_{n-1}\cdot Cin \veebar \overline{H_{n-1}}\cdot\overline{Cin} \qquad (14)$$

together with the opposite polarity output carry of this string, $\overline{C_{n-1}} = \overline{P_{n-1}}+H_{n-1}\cdot\overline{Cin}$, or $C_{h-1} = G_{n-1} + H_{n-1} \bullet C_{in}$, respectively. The two product terms of S_{n-1} (or $\overline{S_{n-1}}$) and the additional unique product term for C_{n-1} (or $\overline{C_{n-1}}$) add up to three unique product terms for a low-order 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 <u>intermediate string</u> uses the product terms of the output carry of the preceding string to generate the sum bits according to Eqs. (10) or (11). It also generates the output carry of the string according to (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 string has already been counted as part of the preceding string. The output carry of the string has additional product terms equal to L, the number of bit positions of lower-order than the string.

$$T_i = K^2+1+L \text{ for } K>1 \qquad (15)$$
$$= 3+L \quad \text{for } K=1$$

A <u>high-order string</u> 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, Cout, so that it can be generated as in (8) or (9) as a function of the input carry to the string. Here, product terms can be shared between Cout and $\overline{S_0}$ or between $\overline{Cout}$ and S_0, so that opposite polarities are selected and only two additional unique product terms are needed for Cout or $\overline{Cout}$.

The number of unique product terms for the high-order string, T_{high}, is the same as for an intermediate string without the factor L, since the output carry is a function of the input carry to the string.

$$T_{high} = K^2+1 \text{ for } K>1 \qquad (16)$$

Figure 7 illustrates an 8-bit adder divided into four strings of 2 bits each. The strings have been optimized to further reduce the total number of product terms to 27. Table 1 expresses the 8-bit adder in equation form to correspond to the PLA format used.

<u>Optimizing Strings</u>

Optimum string sizes are determined differently for the different string types. For the low-order string, it is determined by minimizing the normalized number of product terms needed for a string, i.e., by determining (T_{low}/K) min.

$$(T_{low}/K)\min = [(K^2+2)/K]\min \qquad (17)$$
$$= 3 \text{ for } K=1 \text{ or } 2$$

In other words, a minimum low-order string is either 1 or 2 bits long.

For an intermediate string, the minimum normalized number of product terms,

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

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, therefore, determine the transition value of L, L_t, for which string sizes K and K+1 are equally efficient. The value of L_t for string sizes 1 and 2 is:

$$3+L_t = (5+L_t)/2, \therefore L_t = -1$$

so that it is always more efficient to have an intermediate string size of 2 than of 1. To determine larger string size transition values,

$$(K^2+1+L_t)/K = [(K+1)^2+1+L_t]/(K+1) \qquad (18)$$
$$L_t = K^2+K-1 \text{ for } K>1$$

Table 2 shows various transition values.

TABLE 1. Equations for 8-Bit Adder

$$S_7 = \left\{ \overline{H}_7 \right\} \veebar \left\{ \overline{C}\mathrm{in} \right\}$$

$$S_6 = \left\{ \begin{matrix} \overline{H}_6 \bullet \overline{G}_7 \\ +H_6 \bullet G_7 \end{matrix} \right\} \veebar \left\{ \overline{H}_7 + \overline{C}\mathrm{in} \right\} \qquad C_6 = \begin{array}{l} G_6 \\ +H_6 \bullet G_7 \\ +H_6^* \bullet H_7^* \bullet \mathrm{Cin} \end{array}$$

$$\overline{S}_5 = \left\{ \overline{H}_5 \right\} \veebar \left\{ C_6 \right\}$$

$$\overline{S}_4 = \left\{ \begin{matrix} \overline{H}_4 \bullet P_5 \\ +H_4 \bullet \overline{P}_5 \end{matrix} \right\} \veebar \left\{ \overline{H}_5 + C_6 \right\} \qquad \overline{C}_4 = \begin{array}{l} \overline{P}_4 \\ +H_4 \bullet \overline{P}_5 \\ +H_4^{**} \bullet H_5^{**} \bullet \overline{P}_6 \\ +H_4^{**} \bullet H_5^{**} \bullet H_6^{**} \bullet \overline{P}_7 \\ +H_4^{**} \bullet H_5^{**} \bullet H_6^{**} \bullet H_7^{**} \bullet \overline{C}\mathrm{in} \end{array}$$

$$S_3 = \left\{ \overline{H}_3 \right\} \veebar \left\{ \overline{C}_4 \right\}$$

$$S_2 = \left\{ \begin{matrix} \overline{H}_2 \bullet \overline{G}_3 \\ +H_2 \bullet G_3 \end{matrix} \right\} \veebar \left\{ \overline{H}_3 + \overline{C}_4 \right\} \qquad C_2 = \begin{array}{l} G_2 \\ +H_2 \bullet G_3 \\ +H_2^* \bullet H_3^* \bullet G_4 \\ +H_2^* \bullet H_3^* \bullet H_4^* \bullet G_5 \\ +H_2^* \bullet H_3^* \bullet H_4^* \bullet H_5^* \bullet G_6 \\ +H_2^* \bullet H_3^* \bullet H_4^* \bullet H_5^* \bullet H_6^* \bullet G_7 \\ +H_2^* \bullet H_3^* \bullet H_4^* \bullet H_5^* \bullet H_6^* \bullet H_7^* \bullet \mathrm{Cin} \end{array}$$

$$\overline{S}_1 \quad \left\{ \overline{H}_1 \right\} \veebar \left\{ C_2 \right\}$$

$$\overline{S}_0 = \left\{ \begin{matrix} \overline{H}_0 \bullet P_1 \\ +H_0 \bullet \overline{P}_1 \end{matrix} \right\} \veebar \left\{ \overline{H}_1 + C_2 \right\}$$

$$\mathrm{Cout} = C_0 = \left\{ \begin{matrix} \overline{P}_0 \\ +H_0 \bullet \overline{P}_1 \end{matrix} \right\} \veebar \left\{ \overline{H}_0 + \overline{H}_1 + C_2 \right\}$$

H^* H or P may be used

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

TABLE 2. Transition Values for Optimum Intermediate String Sizes

K ⟶ (K+1)	2 ⟶ 3	3 ⟶ 4	4 ⟶ 5	---
L_t	5	11	19	---

In other words, after 5 lower-order bit positions, the next string size is equally efficient at 2 or 3; after 11 lower-order bit postions, the next string size is equally efficient at 3 or 4; etc.

The change in transition values, ΔL_t, in (19), shows that

$$\Delta L_t = L_t(K \longrightarrow K+1) - L_t(K-1 \longrightarrow K) \tag{19}$$

$$= (K^2+K-1) - [(K-1)^2+(K-1)-1] = 2K$$

a pair of equal intermediate string sizes (two K-1 sizes) are followed by a pair next higher size (two K sizes) for optimum assignment of intermediate string sizes. In other words, a low-order string is followed by intermediate strings of a pair of two's, a pair of three's, 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 (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 (increasing) the high-order string by one and increasing (reducing) the adjacent intermediate string by one.

This leads to the following procedure for assigning string sizes: We begin with a low-order string of two (the larger of the two optimal sizes), followed by pairs of strings of two, three, 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 exactly one less than the adjacent string, the high-order string is increased by one and the low-order string is reduced by one to yield a more optimum assignment. If the high-order string is at least two 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: For each pair of equally-sized intermediate strings, the high order of the pair is increased by not more than one. We can arbitrarily choose the high-to-low order pairs for increasing an intermediate string by one.

Table 3 illustrates the procedure for assigning optimum strings. The assignment is not necessarily unique. For some adder sizes other optima are possible. For example, the 8-bit adder of Figure 7 can also be implemented with 27 product terms using string sizes 1, 2, 2, and 3, low-to-high order.

Tables 4a, b and c illustrate the relevant parameters for 8-bit, 16-bit and 32-bit adders, using 27, 74 and 211 product terms, respectively.

CONCLUSION

PLAs have been shown to be capable of economically implementing one-cycle multi-bit adders. Economy is achieved using a special adder algorithm which permits a large amount of product term sharing as well as efficient use of 2-input decoders and exclusive-OR outputs.

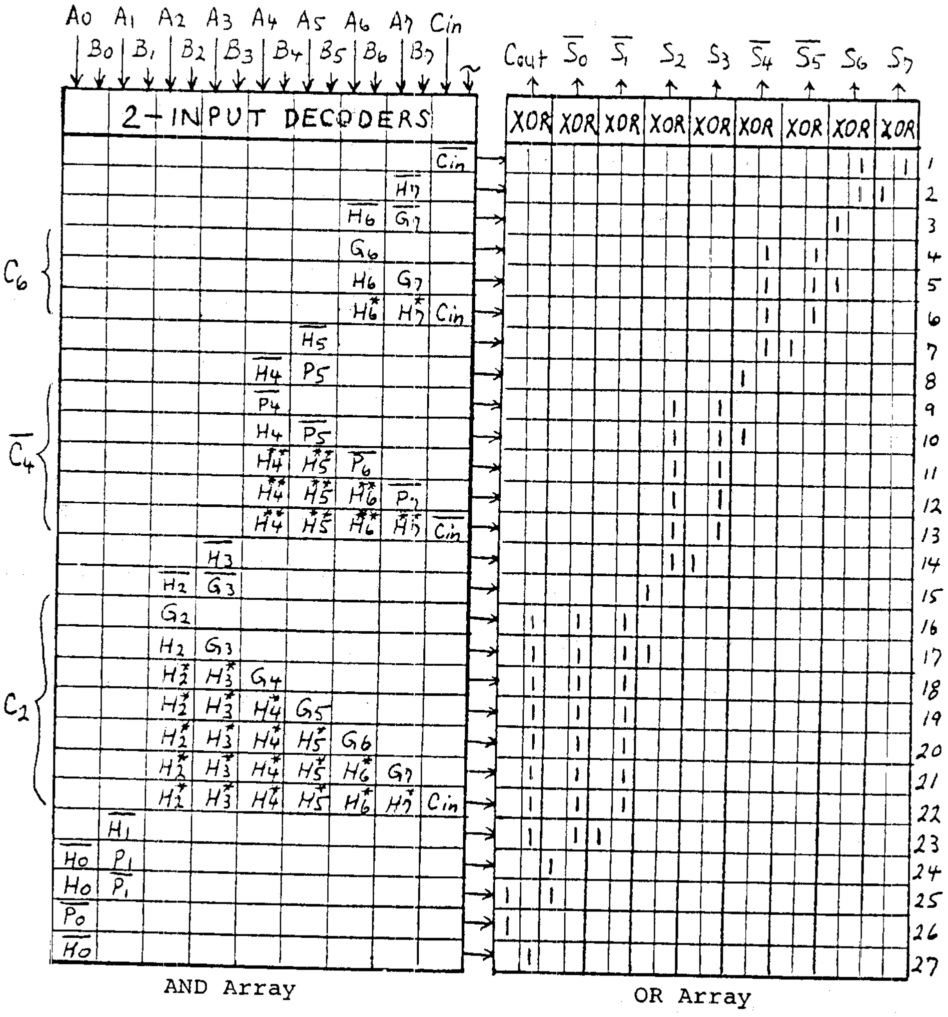

Figure 7. 8-Bit Adder

TABLE 3. Illustration of Procedure for Optimal String Assignment

First-pass string assignment (Nos. are string sizes)	Final string assignment
54433222	no change
44433222	
$\overset{+}{3}443322\overset{-}{2}$	44433221
$\overset{+}{3}43322\overset{-}{2}$	4433221
$\cancel{2}44\overset{+}{3}3\overset{+}{2}22$	4443322
$\cancel{2}4\overset{+}{3}3\overset{+}{2}22$	443322
$\cancel{1}44\overset{+}{3}3222$	4443222
$\cancel{1}4\overset{+}{3}3222$	443222

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

TABLE 4. Number of product terms for a) 8-bit, b)-16-bit and c) 32-bit PLA adders

01	23	45	67	Cin	Bit position
2	2	2	2		←K (String size)
	4	2	-		←L (Number of lower-order bit positions)
5	9	7	6		←T (Number of product terms)

27 Product terms

(a)

			11	11	11		
0123	456	789	01	23	45	Cin	
4	3	3	2	2	2		←K
	9	6	4	2	-		←L
17	19	16	9	7	6		←T

74 Product terms

(b)

		11111	1111	1222	222	22	22	33		
01234	56789	01234	5678	9012	345	67	89	01	Cin	
5	5	5	4	4	3	2	2	2		←K
	22	17	13	9	6	4	2	-		←L
26	48	43	30	26	16	9	7	6		←T

211 Product terms

(c)

REFERENCES

1. W.N. Carr and J.P. Mize, "MOS/LSI Design and Application." McGraw-Hill Book Co., INc., New York 1972.

2. M. Flinders, P.L. Gardner, J.F. Minshull, R.J. Llewelyn, "Functional Memory as a General Purpose System Technology," IEEE Computer Group Conference, June 1970, pp. 314-324.

3. A. Weinberger "Functional Memory Using Multistate Associative Cells," U.S. Patent #3761902.

4. J.C. Logue, N.F. Brickman, F. Howley, J.W. Jones, W.W. Wu, "Hardware Implementation of a Small System in Programmable Logic Arrays," IBM Journal of Research and Development, Vol. 19, No. 2, March 1975, pp. 110-119.

5. J.W. Jones, "Array Logic Macros," IBM Journal of Research and Development, Vol. 19, No. 2, March 1975, pp. 120-126.