Martin S. Schmookler

Design of Large ALUs Using Multiple PLA Macros

This paper describes methods of designing large Arithmetic and Logical Units (ALUs) using multiple Programmable Logic Array (PLA) macros in which the outputs are obtained in one cycle corresponding to one pass through any PLA. The design is based on the well-known technique of providing conditional sums and group carries in parallel and selecting the proper sum using gating circuits. The PLA for each group of bits uses an adder design published by Weinberger in which each bit of the sum is formed from the EXCLUSIVE-OR *of two outputs of the* OR *array. By placing the gating circuits in front of the* EXCLUSIVE-OR *circuits, the sums can be obtained using two* OR *array outputs for each bit and one additional* OR *array output for each internal string of bits. Also discussed are how ALUs containing more than two groups can obtain the group carries using a separate carry-look-ahead PLA macro and how this macro can be compressed by using special decoders and special physical design layout techniques. Additionally, the paper demonstrates how the PLAs can be used to provide detection of overflow and of zero results, and to also provide Boolean operations.*

Introduction

Early forms of Programmable Logic Arrays (PLAs) [1] consisted of single chips with input phase splitters, an AND array, and an OR array. Both arrays were essentially Read-Only Memory (ROM) matrices which could be personalized by changing a single mask. These chips were useful for implementing combinational logic. Through the addition of flip-flops at the outputs and the feeding of the flip-flop outputs back to the phase splitters, these PLAs could also be used for sequential logic. The advantage of these devices was that unique functions could be developed by creating a single mask which defined the data content of the arrays; therefore, even the mask itself could be created using a simple automated process. These chips were useful for the design of control logic and counters; but for the addition function only small adders were implemented, and these were used iteratively since large adders required too many product terms, as will be shown.

More recently, PLA macros have been used for implementing control functions in microprocessor chips [2–4], often in place of conventional ROMs for holding microprograms. In these applications, PLA macros, like ROM macros, permit a high degree of engineering change (EC) capability as well as automated physical design, thereby reducing design time and development costs. Additionally, PLA macros can be tailored to specific functions without compromising too much EC capability. For example, one can use additional logic circuits at the inputs and outputs of the PLA to enhance its functional capability. Such enhancements, recently described, permit larger adders to be implemented with only one pass through a PLA. This paper will describe several techniques for optimizing one-pass PLA adders and for extending previously published techniques to larger-width adders. This paper will also discuss how additional functions may be included in the PLA so that it may be used as a general-purpose Arithmetic and Logical Unit (ALU). The designs to be described herein are most useful in a custom chip environment, since additional circuits are employed in conjunction with the PLAs, and since the PLAs themselves are compressed to eliminate large unused portions of the arrays.

The design technique to be described in this paper, as well as other techniques which are continuing to evolve, will permit the design of large PLA ALUs which will compare favorably in silicon area and in performance with custom-designed ALUs. Their advantage will result mainly from the use of automated physical design pro-

cesses and from their regular structure which can be easily implemented with dynamic circuitry, thereby reducing the power-delay product.

Figure 1 illustrates a straightforward method of designing a one-pass PLA adder for eight bits (A_0, B_0; A_1, B_1; . . . ; A_7, B_7) and input carry (C_{in}). For each bit position i, the logical functions $G_i = A_i \cdot B_i$, $P_i = A_i + B_i$, and $H_i = A_i \oplus B_i = A_i \cdot \bar{B}_i + \bar{A}_i \cdot B_i$ are defined. The number of product terms is minimized by producing the complement output carry ($\bar{C}_{out}$), enabling it to share product terms with the high-order sum bit (S_0). The adder design of Fig. 1 uses two-input decoders on corresponding bits from the two operands and requires 73 product terms. For an N-bit adder, the number of product terms N_p is given by

$$N_p = N^2 + N + 1, \tag{1}$$

while the number of bits per word, or the number of columns, is given by

$$N_c = N_c(\text{AND}) + N_c(\text{OR}) = (4N + 2) + (N + 1) = 5N + 3. \tag{2}$$

An eight-bit adder would thus have 73 product terms and 43 columns, for a total array size requirement of 3139 bits.

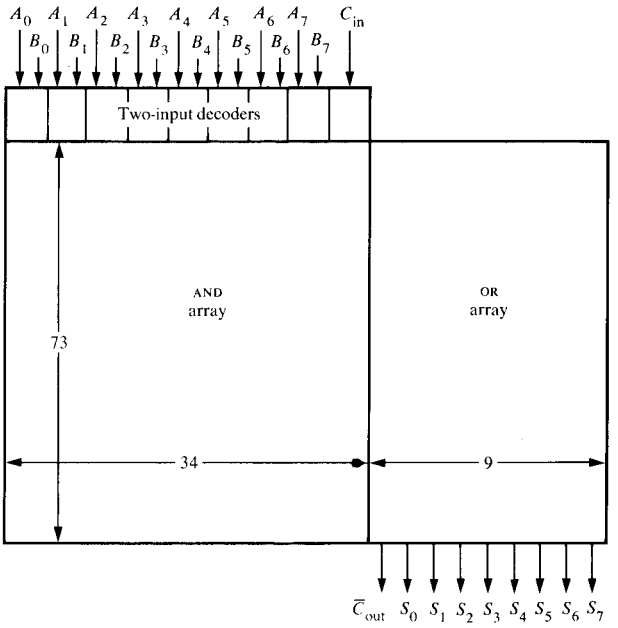
Without two-input decoders, an eight-bit one-pass adder would be impractical because with one-input decoders the number of product terms would grow exponentially, as shown by

$$N_p = 2^{N+3} - 4N - 7, \tag{3}$$

which would require 2009 product terms and 86 387 total array bits. A four-bit adder, which would appear to be the practical limit for the one-input partitioned design, would require 105 product terms and 2415 array bits.

Even with two-input decoders, however, this approach becomes impractical for N much greater than eight; and even for eight-bit adders it is not very efficient, as it requires too much chip area, and the large number of product terms required increases the delay through the PLA.

A recent paper by A. Weinberger [5] demonstrates an efficient way to use the EXCLUSIVE-OR (XOR) functions available at the outputs of some PLAs. Weinberger shows that the eight-bit adder can be reduced to 25 product terms. However, since two output columns are needed for each XOR output, the number of columns in the array is $N_c = (4N + 2) + (2N + 2) = 6N + 4$. Therefore, an eight-bit adder would have 1300 array bits. Larger adders would also be practical. A 16-bit adder would have 68 product terms, 100 columns, and 6800 array bits.



(a)

Number of product terms	Expressions for sum bits (S_i) and output carry ($\bar{C}_{out}$)
2	$S_7 = \bar{H}_7 \cdot C_{in} + H_7 \cdot \bar{C}_{in}$
4	$S_6 = \bar{H}_6 \cdot G_7 + \bar{H}_6 \cdot P_7 \cdot C_{in} + H_6 \cdot \bar{P}_7 + H_6 \cdot \bar{G}_7 \cdot \bar{C}_{in}$
	⋮
16	$S_0 = \bar{H}_0 \cdot G_1 + \bar{H}_0 \cdot P_1 \cdot G_2 + \ldots + \bar{H}_0 \cdot P_1 \cdot \ldots \cdot P_7 \cdot C_{in} + [H_0 \cdot \bar{P}_1 + H_0 \cdot G_1 \cdot \bar{P}_2 + \ldots + H_0 \cdot \bar{G}_1 \cdot \ldots \cdot \bar{G}_7 \cdot \bar{C}_{in}]$
1	$\bar{C}_{out} = \bar{P}_0 + [H_0 \cdot \bar{P}_1 + H_0 \cdot G_1 \cdot \bar{P}_2 + \ldots + H_0 \cdot \bar{G}_1 \cdot \ldots \cdot \bar{G}_7 \cdot \bar{C}_{in}]$
where	$P_i = A_i + B_i$, $G_i = A_i \cdot B_i$, and $H_i = A_i \oplus B_i = A_i \cdot \bar{B}_i + \bar{A}_i \cdot B_i$

(b)

Figure 1 Eight-bit PLA adder: (a) PLA arrangement, and (b) equations. Note: · is the logical product (AND), + is the logical sum (OR), and ⊕ is the logical exclusive-or (XOR) function.

An alternative approach which has been suggested by Peter Cook of the IBM Research Laboratory in Yorktown Heights, New York also relies on using external circuits at the array outputs. A separate PLA adder is used for each group of, say, eight bits (Fig. 2). The lowest-order group could provide the carry to the next-higher group. If there were more than two groups, a separate carry-look-ahead (CLA) PLA could be used to provide carries to the higher-order groups. Each group PLA adder, except the low-order PLA, would be modified to provide two separate conditional sums, one assuming an input carry to the group, the other assuming no input carry. The carry to each group would then select the proper sum using external AND gates. The same number of product terms are

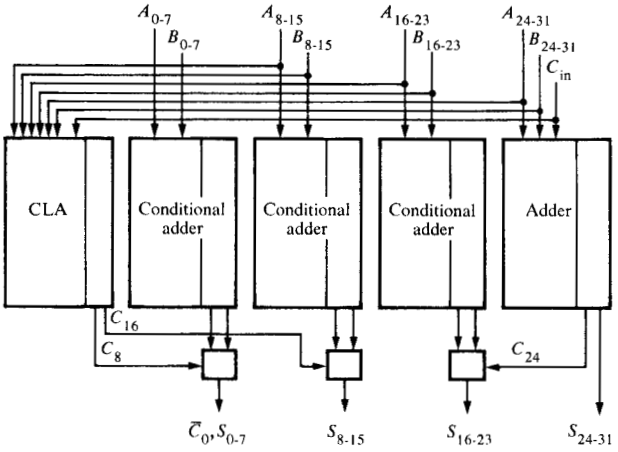


Figure 2 32-bit adder using carry-look-ahead (CLA) PLA and conditional sum PLAs.

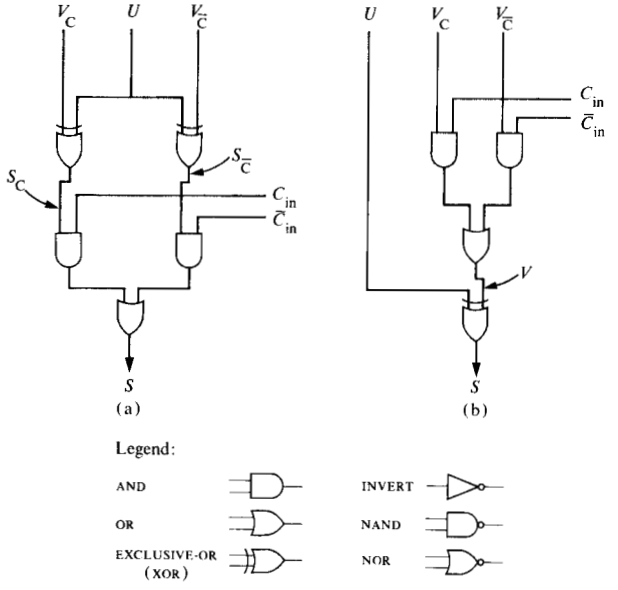


Figure 3 External circuit for Weinberger adder with conditional sums. $S = U \oplus V$ where only V is dependent upon C_{in}. $S_C = U \oplus V_C$, $S_{\bar{C}} = U \oplus V_{\bar{C}}$.

needed to produce two conditional sums as to produce an unconditional sum. Therefore, an eight-bit adder could consist of two four-bit adders, each having only 21 product terms, while a 16-bit adder could have two eight-bit adders with 73 product terms in each.

Table 1 compares the sizes of PLA adders for the four different types of designs heretofore discussed. These are:

1. Single PLA with one-input decoders (phase splitters).
2. Single PLA with two-input decoders.
3. Weinberger PLA adder with two-input decoders and XOR output circuits.
4. Cook adder with two PLAs having $N/2$ bit positions each. Each PLA has two-input decoders, and the high-order PLA produces conditional sums which are selected by the output carry from the low-order PLA.

The expressions for the total number of bits N_b in Table 1 show that the Weinberger adder exhibits the slowest growth with respect to N, but the Cook adder is more economical for up to 16-bit adders. However, for adders greater than eight bits, neither design as shown so far compares well with adders designed using conventional logic circuits, for which the growth is approximately proportional to N. Some additional techniques described in Weinberger's paper, such as the use of custom decoders and compressed arrays, permit practical PLA adders beyond eight bits.

In the next section a scheme will be presented which combines Cook's idea of using multiple PLAs and conditional sum outputs with Weinberger's methods for implementing each PLA. The result will be a much more economical design than either design by itself. Furthermore, as will be demonstrated, this scheme can also take advantage of custom decoders and compression in order to reduce the silicon area even further. It will also be shown that further benefits can be derived from the use of multiple PLAs when designing a complete ALU containing Boolean functions as well.

Design of large adders using multiple PLAs

• *Direct implementation of Weinberger adder with conditional sums*

For adders of 16 bits or more, it is more efficient to combine the two approaches. The group PLA adders shown in Fig. 2 could be replaced by Weinberger eight-bit adders. The low-order PLA need not be modified. The other group adders must be modified to provide two conditional sums. As before, no additional product terms are needed, and the product term corresponding to the input carry is eliminated.

Although each bit of an unconditional sum requires two output columns for the XOR circuit, no more than three columns are needed for each pair of conditional sum bits. The reason for this is that one column, which is not dependent on the input carry, may be common to both conditional sums. The low-order bit of a modified PLA adder has only two columns, one for each conditional sum, and no output XOR circuits are needed.

Table 1 Number of product terms N_p, number of columns N_c, and total number of bits N_b for four types of PLA adders.

	Number of adder bits					
	4	*8*	*12*	*16*	*32*	*N*
Type 1						
N_p	105	2,009	32,713	5.2×10^5	3.4×10^{10}	$2^{N+3} - 4N - 7$
N_c	23	43	63	83	163	$5N + 3$
N_b	2,415	86,387	2.1×10^6	4.3×10^7	5.6×10^{12}	$\cong 40N \times 2^N$
Type 2						
N_p	21	73	157	273	1,057	$N^2 + N + 1$
N_c	23	43	63	83	163	$5N + 3$
N_b	483	3,139	9,891	22,659	1.7×10^5	$\cong 5(N + 0.5)^3$
Type 3						
N_p	10	25	44	68	195	$\cong 1.07N^{1.5}$
N_c	28	52	76	100	196	$6N + 4$
N_b	280	1,300	3,344	6,800	38,220	$\cong 6.6N^{2.5}$
Type 4						
N_p	7	21	43	73	273	$(N/2)^2 + (N/2) + 1$
N_c	27	49	71	93	181	$5.5N + 5$
N_b	189	1,029	3,053	6,789	49,413	$\cong 1.4(N + 1)^3$

The output circuits for each of the other bits can consist of two XOR circuits plus the gating circuits for selecting the appropriate sum, as shown in Fig. 3(a). However, by placing the gating circuits before the XOR, as shown in Fig. 3(b), one of the XORs can be removed. This simplification is based on the following identity:

$$C \cdot f(U, V_1) + \bar{C} \cdot f(U, V_0) = f(U, C \cdot V_1 + \bar{C} \cdot V_0),$$

where the function f, in this case, is the XOR function. The circuit shown in Fig. 3(b) should require fewer devices, but may increase the delay from the input carry.

It will be shown in the next section that the modified PLA adder can be simplified even further, resulting in fewer output columns and fewer output gates.

• *Simplified Weinberger adder with modified conditional sums*

Figure 4 illustrates Weinberger's latest version of an eight-bit adder with two-input decoders. The bits are subdivided into strings of consecutive bits. For each string, there is a set of product terms which, when ORed, form either the true or the complement of the input carry to that string. These product terms are included in one of the two output columns of each bit in the string. Each true sum (*i.e.*, S_0, S_1, S_2, S_6, or S_7) has the form

$$S_i = \{F_i\} \oplus \{D_i + \bar{C}_s\}, \tag{4}$$

where $\bar{C}_s$ represents the set of product terms for the complement of the string input carry, and F_i and D_i represent groups of product terms unique to S_i and dependent only on inputs contained within the string. For the low-order strings, $\bar{C}_s$ equals $\bar{C}_{in}$. For the other strings, $\bar{C}_s$ has the form

$$\bar{C}_s = \bar{P}_s + H_s \cdot \bar{P}_{s+1} + H_s \cdot H_{s+1} \cdot \bar{P}_{s+2} + \ldots + H_s \cdot H_{s+1} \cdot \ldots \cdot H_7 \cdot \bar{C}_{in}, \tag{5}$$

which can be rewritten as

$$\bar{C}_s = \{K_s\} + \{H_s^7\} \cdot \bar{C}_{in}, \tag{6}$$

where K_s represents a group of product terms common to all of the sums in the string. Therefore, for true sums above the low-order string,

$$S_i = \{F_i\} \oplus [\{D_i + K_s\} + \{H_s^7\} \cdot \bar{C}_{in}]. \tag{7}$$

Letting each group of terms enclosed in { } correspond to an output column of the OR array, each S_i can be formed externally from three columns and C_{in}. Since C_{in} is used only in the external circuit, it does not incur the delay of the PLA. Therefore, it may come from the PLA adder of a lower-order group or from a CLA PLA.

The product $\{H_s^7\} \cdot C_{in}$ is common to all of the bits in the string. Thus, only two columns are needed for each bit, plus one for each string above the low-order string.

The complement sums $\bar{S}_3$, $\bar{S}_4$, and $\bar{S}_5$ are formed in a similar way, except that C_{in} is used in place of $\bar{C}_{in}$.

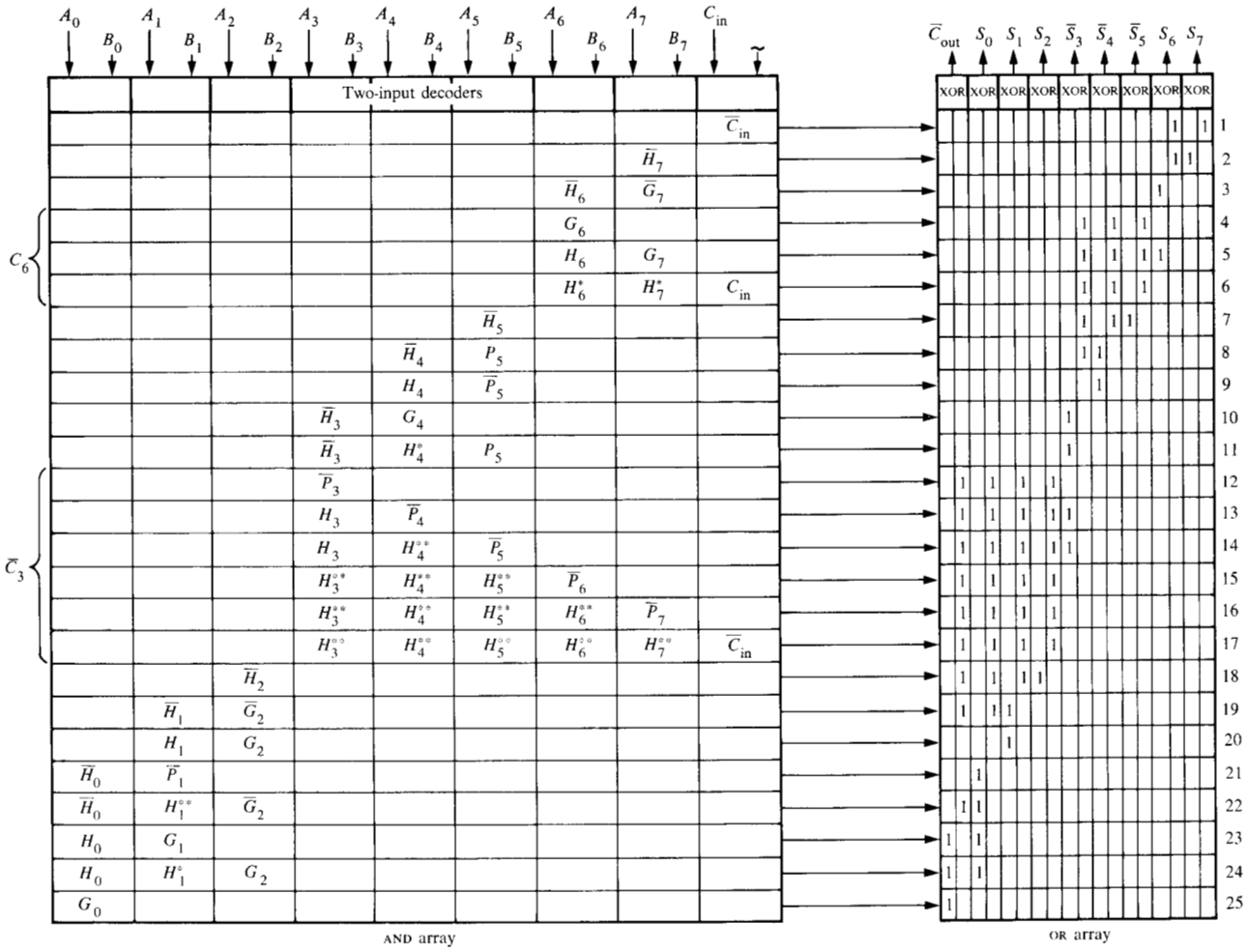


Figure 4 Weinberger's eight-bit PLA adder. Note: for H^*, H or P may be used; and for H^{**}, H or $\bar{G}$ may be used.

Only two columns are needed for the low-order group consisting of S_6 and S_7:

$$S_7 = \{\bar{H}_7\} \oplus \bar{C}_{in} \tag{8}$$

and

$$S_6 = \{\bar{H}_6 \cdot \bar{G}_7 + H_6 \cdot G_7\} \oplus [\{\bar{H}_7\} + \bar{C}_{in}], \tag{9}$$

where $\{\bar{H}_7\}$ is common to both sums.

If needed, the output group carry $\bar{C}_{out}$ has the same form as the sum bits of the high-order string. However, the output circuit can be slightly simplified, which will also reduce the delay. Designating the two inputs to the XOR circuit as C_L and C_R, respectively, then

$$\bar{C}_{out} = C_L \oplus C_R = C_L \cdot \bar{C}_R + \bar{C}_L \cdot C_R.$$

But $C_L = G_0^2$ is the carry-generate condition for the high-order string, and $\bar{C}_R = H_0^2 \cdot C_3$, where H_0^2 is the strict carry-propagate condition for the string and is therefore mutually exclusive with G_0^2. Therefore, $C_L \cdot \bar{C}_R = 0$, and

$$\bar{C}_{out} = \bar{C}_L \cdot C_R. \tag{10}$$

Thus, we can replace the XOR circuit with the simpler and faster AND circuit if the left column output is complemented.

Figure 5 shows the modified PLA adder. It has one less product term and two less columns in the AND array than the unmodified PLA adder. For an eight-bit group, they each have the same number of columns in the OR array. If the output carries are not needed from the intermediate groups of the adder, then one product term and two OR array columns can be removed from each.

Additional product terms can be removed by using external NOR circuits in place of the AND circuits. The out-

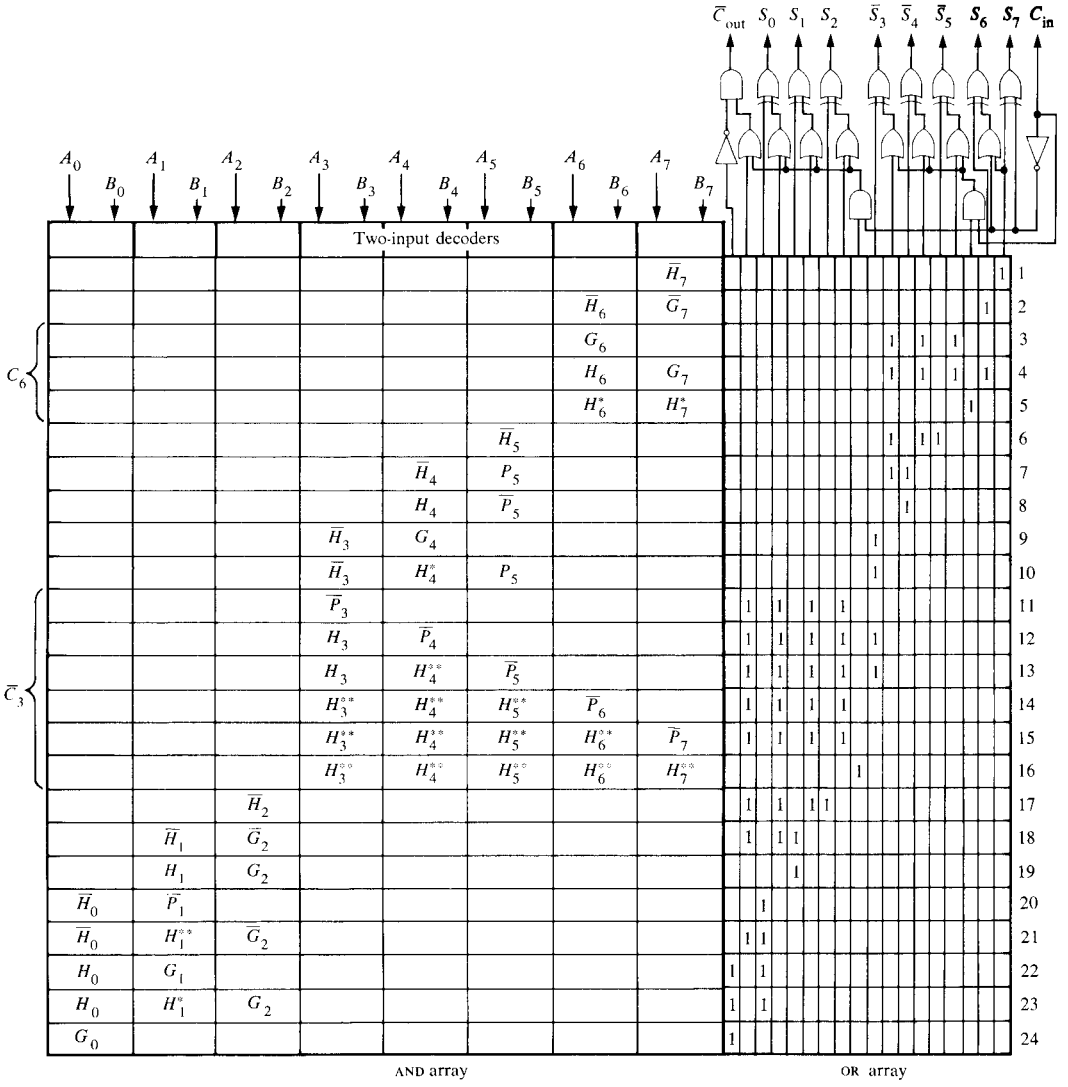


Figure 5 "Modified conditional sum" eight-bit Weinberger PLA adder.

put functions H_6^7 and H_3^7, which correspond to product terms 5 and 16, respectively, can be replaced by their complements. The complement of H_6^7 is given by

$$\bar{H}_6^7 = \bar{H}_6 + \bar{H}_7 = \bar{H}_6 \cdot \bar{G}_7 + \bar{H}_7 \quad (\text{since } \bar{H}_7 = G_7 + \bar{P}_7),$$

which can be obtained by ORing product terms 1 and 2. Similarly, the complement of H_3^7 is the OR of product terms 1, 2, 6, 7, and 10. In general, one less product term is needed for each internal string carry.

The expressions given by Weinberger for determining the number of unique product terms required for each string are easily modified. The low-order string has two less product terms than before, since one is removed for the input carry and one is removed for the string output carry. Therefore, $1 + 2 + 4 + \ldots + 2(K - 1)$ product terms are required for a string of K sum bits, and one additional unique product term is required for the string output carry. The resulting expression for T_{low} is given by

$$T_{low} = [1 + 2 + 4 + \ldots + 2(K - 1)] + 1$$
$$= K^2 - K + 2 \qquad \text{for } K \geq 1. \tag{11}$$

(Here, as in the Weinberger definitions, T is the number of product terms and K is the string size.)

For intermediate strings, the output string carry has L additional product terms, where L is the number of bit

Table 2 Number of product terms N_p, number of columns N_c, and total number of bits N_b for two Weinberger PLA adders with $N/2$ modified conditional sums each.

	Number of adder bits					
	4	*8*	*12*	*16*	*32*	*N*
N_p	4	8	14	22	63	$\cong 0.34N^{1.5}$
N_c	24	50	76	100	200	$\cong 6.25N$
N_b	96	400	1,064	2,200	12,600	$\cong 2.2N^{2.5}$

positions of lower order than the string. Therefore,

$$T_i = K^2 - K + 2 + L \qquad \text{for } K \geq 1. \tag{12}$$

The number of product terms needed for the high-order string is unchanged, since no internal string carry is formed, and is therefore given by

$$T_{high} = K^2 - K + 2 \qquad \text{for } K \geq 1. \tag{13}$$

Although the use of modified conditional sums has been illustrated for an eight-bit adder using standard PLAs with two-input decoders, this approach is applicable also to larger adders, including those having custom decoders with more than two inputs, which are also described in the Weinberger paper.

Complete adder

• *Adders with two or more groups*

Since the input carry is usually available at the same time as the operands which are to be added, the PLA used for the low-order group does not have to produce conditional sums. However, an adder which provides modified conditional sums requires almost the same area as one which provides unconditional sums. Therefore, the PLAs for all groups can be identical, with no additional cost in area.

The output carry from the low-order group may be used directly as the input carry by the output circuits of the adjacent group. For adders consisting of only two groups, no additional circuits or PLAs are needed for providing input carries to each group. For comparison with the PLA adders listed in Table 1, the sizes of several PLA adders are given in Table 2, with each adder consisting of two Weinberger adders with modified conditional sums. Each PLA has $N/2$ bit positions and uses two-input decoders. Fewer than one-third as many bits are needed when compared with a Weinberger adder using a single PLA.

If the adder consists of several groups, there are several alternatives available. The simplest is to connect the output carry from each group PLA adder to the external output circuits of the next group. Thus, the carry from the low-order group will ripple through the output circuits which form the carries for each of the other groups. If these circuits are considerably faster than a PLA, the total delay should be adequate.

The ripple delay can be reduced if each modified PLA produces carry-generate and carry-propagate functions for the group instead of the output carry. These functions, combined in a carry-look-ahead circuit, can be obtained with minor changes in the PLA arrays or with changes in the external circuit. As shown in Fig. 5, $\bar{C}_{out}$ is obtained from three array outputs and C_{in}. Using the same notation as in Eq. (7), it has the form

$$\bar{C}_{out} = \{\bar{F}\} \cdot [\{D + K\} + \{H_3^7\} \cdot \bar{C}_{in}].$$

Then

$$\bar{G} = \{\bar{F}\} \cdot [\{D + K\} + \{H_3^7\}]$$

and

$$\bar{P} = \{D + K\},$$

which can be obtained from the same array outputs, may be used as the carry-generate and carry-propagate functions of the group, respectively.

Another alternative which eliminates the delay of the external carry-look-ahead circuit is the use of a separate carry-look-ahead (CLA) PLA as shown in Fig. 2 for providing the carries C_8 and C_{16} to the two high-order groups. In a custom macro environment there are several techniques available which will reduce the size of this PLA to a fairly small area.

• *Description of the carry-look-ahead PLA*

The CLA PLA for Fig. 2 has 42 product terms; however, as shown in Fig. 6, it can be compressed into two separate but interlocking PLAs having only 25 rows. The separation of the two PLAs is indicated by the heavy lines in the AND array. Both rows and columns are shared by the two arrays, and separate sets of decoders are needed, one at the top and one at the bottom. However, we will see that only a very simple decoder is needed for each bit, and therefore the bottom set of decoders should fit within the rectangular boundary defined by the 25 rows. The OR arrays, which consist of one column each, are placed one on each side.

Since the PLA has 49 inputs at the top, it might appear that it needs 100 columns, including two output columns. However, the only logic functions used are the bit-propagate and bit-generate functions. We can use a special decoder which provides only two columns per adder bit,

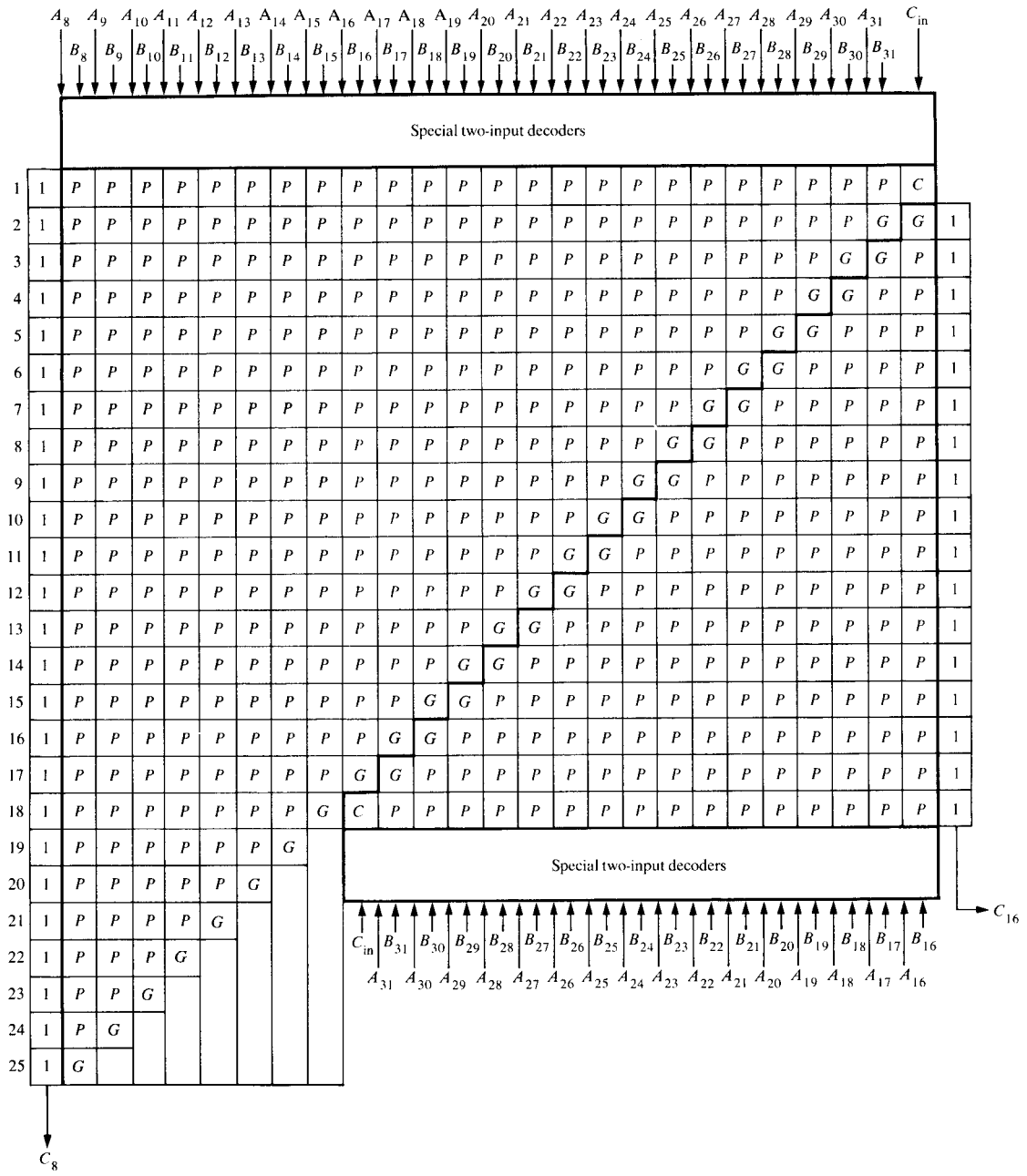


Figure 6 Carry-look-ahead PLA with split OR array.

resulting in 52 columns for the entire PLA. The special decoder itself is much simpler than the decoder normally used with two-input partitions, consisting of just an AND and an OR circuit. Furthermore, since the two columns are never both selected in the same product term, they may share a single device, as with single-input partitions; and for some circuit layouts, this can reduce the spacing between columns. Figure 7 illustrates a decoder and one pair of columns with shared devices. With NMOS devices the AND array consists of NOR circuits, and each decoder consists of a NAND and a NOR circuit.

Techniques for reducing PLA delay

Although the speed of a PLA is mostly dependent on device and circuit design, there are ways for the logic designer to reduce the delay of a PLA macro. For the pur-

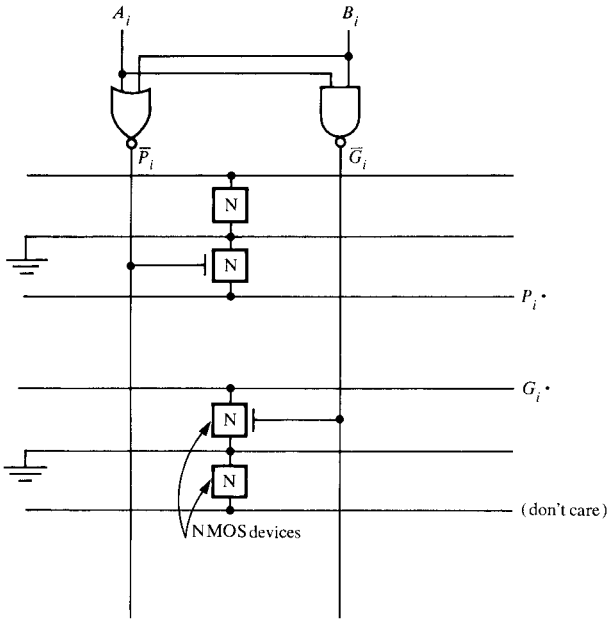


Figure 7 Special decoder for CLA PLA.

pose of illustration, FET devices will be assumed in the following discussions. The total delay typically consists of the input decoder delay, the AND array delay, and the OR array delay. Generally, the delays for each are dependent on device sizes and output capacitance. Since there is a relatively small number of decoder circuits, large devices can be used without greatly increasing the total area. The capacitance of the AND array columns, which affects the decoder delay, can be reduced by minimizing both the total number of rows and the number of personalized rows along any column.

The OR array can be split, with some outputs on one side of the AND and some on the other. This allows two product terms, in some cases, to share the same row of the AND array, and results in a compressed PLA. It reduces both the size of the PLA and its delay. Figure 8 illustrates the 22 product terms compressed into 13 rows. (Product terms 5 and 16 have been deleted using the method previously described.) Compression reduces the column capacitance, therefore reducing both the decoder and the OR array delays.

Figure 8 also demonstrates that the propagate function $H = A \oplus B$ can be replaced by the OR function $P = A + B$ for propagating true carries, and by $\bar{G} = \bar{A} + \bar{B}$ for propagating complement carries. Since the H function is formed by ANDing columns corresponding to both P and $\bar{G}$, the replacement reduces the maximum number of personalized bits both in a column and in a product term. For Fig. 8, the maximum column personalization in the AND array is reduced from six to five ($\bar{A}_4 + \bar{B}_4$ column), and the maximum product term personalization is reduced from eleven to seven (product term 15). Therefore, both the decoder delay and the AND array delay are reduced.

Column personalization in the AND array can also be reduced by duplication of the columns. This can be especially useful for the function select lines of an ALU which contains both an adder and the Boolean functions.

The AND array delay is also improved when special decoders are used, such as previously described for the CLA PLA. The special decoders reduce the number of columns in the AND array and may also reduce the number of personalized bits in a product term.

If all of the personalized bits in a product term are adjacent to unpersonalized bits, it may be feasible to use larger devices for the personalized bits and for the load device too, if necessary, therefore reducing the AND array delay.

The OR array is frequently quite sparse. If such is the case, it may be possible to order the rows as shown in Fig. 8, where every personalized bit in the OR array has an adjacent unpersonalized bit in the same column. This makes device sharing possible using two devices for every three rows. If one blank row were added to Fig. 8, the rows could be arranged to permit one device for every pair of rows. Either way, device sharing could reduce the spacing between rows for the entire PLA. Alternatively, it could permit the use of larger devices in the OR array, which might possibly reduce the OR array delay.

PLA adder as part of an ALU

• *Other ALU functions*

A general-purpose Arithmetic and Logical Unit (ALU) should provide capability for other arithmetic and logical operations in addition to addition. It should also provide detection of specific conditions such as overflow or zero result. Some of these functions can easily be included in the PLAs used for the adder. Others require additional circuits external to the PLAs. Some other functions, such as multiply and divide, require use of algorithms which use the ALU iteratively. The least important functions can be provided by software subroutines. The choice of how each function should be provided will depend on the application of the ALU as well as consideration of cost and performance trade-offs.

For discussions of arithmetic operations which follow, only fixed point binary operands with negative numbers

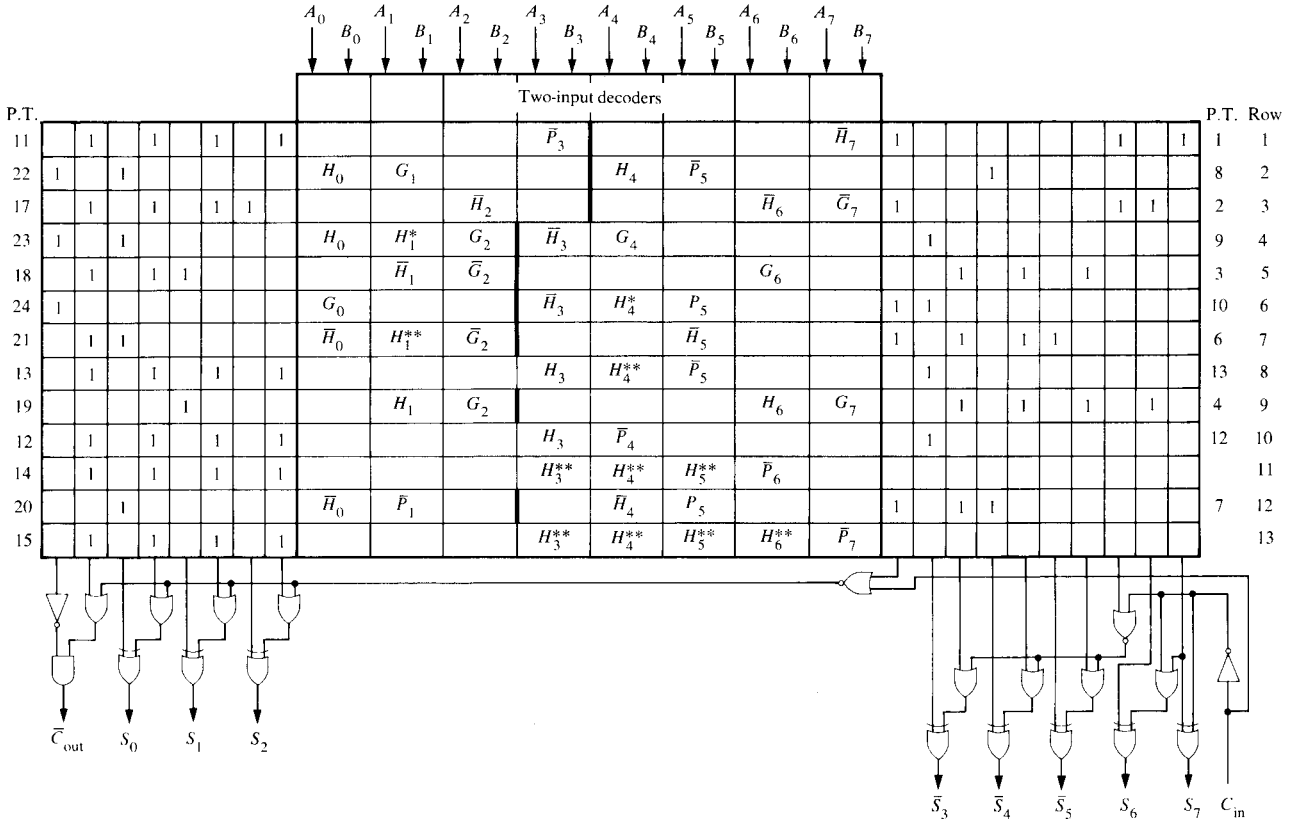


Figure 8 Compressed eight-bit adder. Notes: (1) Product term (P.T.) numbers reference corresponding terms in Fig. 5. (2) Product terms 5 and 16 have been eliminated by replacing external ANDs with NORs. (3) For H^*, H or P may be used; and for H^{**}, H or $\bar{G}$ may be used.

represented in two's-complement notation will be considered. Consideration of other number representations would require discussion of possible algorithms which are outside the scope of this paper. Operations with other number representations are usually provided by microprograms or by software subroutines.

The ALU must provide some means of doing subtraction. If this function is included directly in the PLAs with the adder, it will require as many product terms as are needed for addition, with no common terms between the two functions. Instead, one operand can first be complemented and then added to the other operand. If two machine cycles are permitted, the operand can be complemented during the first cycle by using some additional product terms in the PLAs; if subtraction needs to be completed in one cycle, additional gates can be provided at the inputs to the PLAs for complementing one operand.

Multiplication and division are usually implemented by iterative addition or subtraction. However, the ALU must provide some means of shifting the partial result after each iteration. Again, shifting functions can be provided in the PLAs with additional product terms. However, it will take fewer machine cycles and probably less space if shift gates external to the PLAs are used.

During the addition cycle the ALU must also detect specific conditions which may result. It must detect the sign of the result and the occurrence of an output carry. For two's-complement representation, the sign bit is the sum S_0, so both of these are available as direct outputs from the adder. Other conditions which are usually detected are overflow and zero result. In the following sections, we will show how each of these can be detected using a combination of external circuits and additional product terms in each PLA.

• *Overflow detection*

Overflow occurs when two numbers having the same sign are added and the result has an opposite sign. This can be expressed as

$$OV = A_0 \cdot B_0 \cdot \bar{S}_0 + \bar{A}_0 \cdot \bar{B}_0 \cdot S_0$$
$$= G_0 \cdot \bar{S}_0 + \bar{P}_0 \cdot S_0. \quad (14)$$

It is probably desirable that the overflow signal should be

no slower than the other adder outputs, and therefore should not be developed from the S_0 output. Using

$$S_0 = (H_0 \oplus G_1^{K-1}) \oplus (H_1^{K-1} \cdot C_K), \tag{15}$$

where K is the number of bits in the high-order string, and using the fact that G_0, H_0, and $\bar{P}_0$ are all mutually exclusive, we obtain

$$\begin{aligned} OV &= G_0 \cdot \bar{G}_1^{K-1} \oplus G_0 \cdot H_1^{K-1} \cdot C_K \\ &\quad \oplus \bar{P}_0 \cdot G_1^{K-1} \oplus \bar{P}_0 \cdot H_1^{K-1} \cdot C_K \\ &= (G_0 \cdot \bar{G}_1^{K-1} \oplus \bar{P}_0 \cdot G_1^{K-1}) \oplus (G_0 \oplus \bar{P}_0)(H_1^{K-1} \cdot C_K) \\ &= (G_0 \cdot \bar{G}_1^{K-1} + \bar{P}_0 \cdot G_1^{K-1}) \oplus \bar{H}_0 \cdot H_1^{K-1} \cdot C_K \\ &= \{\bar{G}_0 \cdot \bar{G}_1^{K-1} + P_0 \cdot G_1^{K-1}\} \oplus \{H_0 + \bar{H}_1^{K-1} + \bar{C}_K\}. \end{aligned} \tag{16}$$

The expression in the left braces expands into $2(K - 1)$ product terms. Except for the product term H_0, the expression in the right braces is identical to the right-hand output for S_0. Therefore, a total of $(2K - 1)$ additional unique product terms are needed. The adder shown in Fig. 8 would require five additional rows in order to provide overflow detection.

• *Zero sum detection*

The simplest way to detect a result of all zeros is to OR all of the output sums and then complement the result. This results in a detection signal which is slower than the other outputs. This method, however, provides valid detection regardless of the operation performed by the ALU.

For some operations, detection of a zero result can be provided directly as an output from the PLA adder, using just one product term for each operation. In some applications of the adder, it may only be necessary to detect a zero result when one operand is being subtracted from the other, such as when comparing or decrementing the operands. For such cases, a zero result can only occur when the two operands are identical. Since the subtrahend is complemented before it enters the adder, the XOR of each pair of adder inputs must be true. Then the zero result can easily be detected using a single product term based on the following equation:

$$\begin{aligned} Zeros &= H_0 \cdot H_1 \cdot H_2 \cdot \ldots \cdot H_{N-1} \\ &= H_0^{N-1}. \end{aligned} \tag{17}$$

This method could also be used for addition in systems which use sign and magnitude representation, since addition of numbers with opposite signs would be done by subtraction.

We will show subsequently that if Boolean operations are also included in the ALU, then a zero result can be detected with just one product term for each operation.

For general applications, zero detection must also include addition. A patent by Weinberger [6] shows that a zero result for an eight-bit adder can be obtained using the equation

$$\begin{aligned} \overline{Zeros} &= [H_0 \cdot \bar{G}_1 \cdot \bar{G}_2 \cdot \ldots \cdot \bar{G}_7] \cdot \bar{C}_{\text{in}} \\ &\quad + [\bar{H}_0 \cdot P_1 + \bar{H}_1 \cdot P_2 + \ldots + \bar{H}_6 \cdot P_7] \\ &\quad + [\bar{H}_7] \cdot C_{\text{in}}, \end{aligned} \tag{18}$$

which can be rewritten as

$$\begin{aligned} \overline{Zeros} &= [\bar{Z}_1] \cdot \bar{C}_{\text{in}} + [\bar{Z}_2] + [\bar{Z}_3] \cdot C_{\text{in}} \\ &= (\{\bar{Z}_1 + \bar{Z}_2\} + C_{\text{in}}) \cdot (\{\bar{Z}_2 + \bar{Z}_3\} + \bar{C}_{\text{in}}). \end{aligned} \tag{19}$$

If each set of braces represents an output from the PLA, these outputs can be combined with the input carry using external circuits. For an adder containing several PLAs, the outputs from all of the PLAs can be combined to obtain the zero result signal.

Equation (18) contains nine product terms, or $(N + 1)$ product terms for an N-bit adder. However, the product term $\bar{H}_{N-1}$ is already contained in the adder, and each string of complement sums contains one common product term. For the eight-bit adder of Fig. 8, product term $\bar{H}_4 \cdot P_5$ is common, so that only seven additional unique product terms are needed. If the outputs are taken from the right-hand OR array, five of those product terms can share the five additional rows that were needed for overflow detection. Therefore, this method of zero sum detection requires two additional rows for either an eight-bit adder or for an adder composed of several PLAs of eight bits each.

Since the number of additional product terms is nearly equal to the number of bits being added in each PLA, we can see another advantage of using several small PLAs for a large adder.

• *Boolean operations*

The ALUs contained in most general-purpose microprocessors are also capable of doing Boolean operations on corresponding bits of two operands. They usually contain instructions for three Boolean functions: AND, OR, and XOR. With two-input decoders, any Boolean function can be obtained with a single product term for each bit. Separate product terms are needed for each Boolean function. However, the AND, OR, and XOR operations can be provided with only two product terms per bit, since the OR function can be provided by selecting both the AND and XOR operations. For those bit positions whose output is complemented, we can take advantage of the XORs at the PLA outputs, since forcing a logical *one* into one input of the XOR will complement the function at the other input.

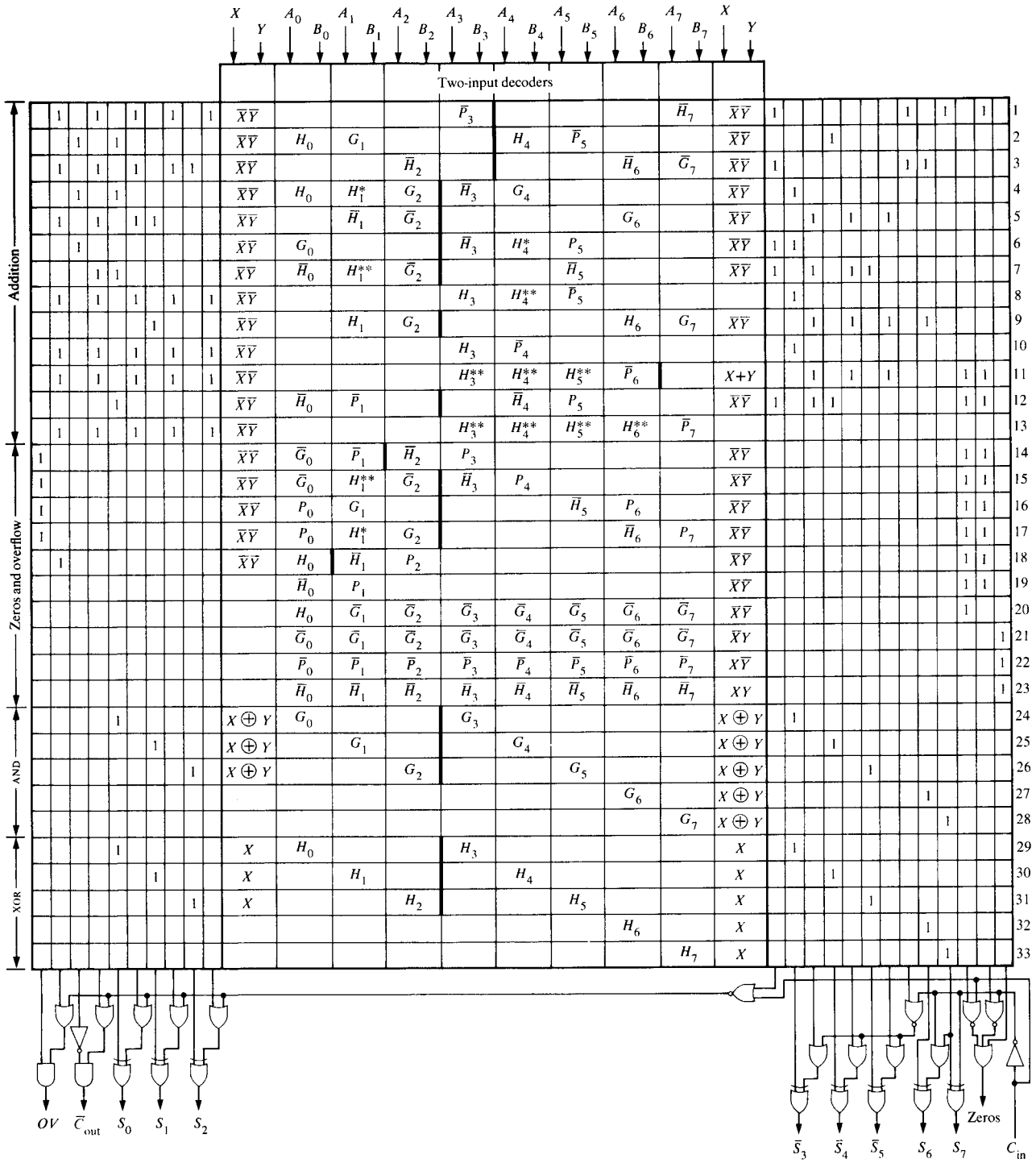


Figure 9 Complete eight-bit ALU.

A zero result for any Boolean function can be detected using a single product term containing the product of the complement of the Boolean function for each bit position. For example, a result of zeros for an OR operation can be obtained using the product term $\bar{P}_0 \cdot \bar{P}_1 \cdot \bar{P}_2 \cdot \ldots \cdot \bar{P}_{N-1}$, since each P_i is the OR of A_i and B_i. The zero result for the

Boolean functions must be combined externally with the zero result for addition, since the latter, as given by Eq. (19), is in complement form.

In order to provide both arithmetic and Boolean functions, each PLA must have additional columns of inputs for the function selection bits. These bits must be encoded in a way that permits some product terms to be selected by more than one function.

All of the 16 possible two-variable Boolean functions can be obtained using only four product terms per bit corresponding to the four minterms $A \cdot B$, $A \cdot \bar{B}$, $\bar{A} \cdot B$, and $\bar{A} \cdot \bar{B}$. With four corresponding function selection bits, any combination of the minterms can be selected.

Inclusion of the Boolean functions provides another good reason for breaking the ALU into small groups of bits, each with a separate PLA. If the AND, OR, and XOR functions are provided for a 32-bit ALU using only one PLA, it will need two product terms per bit, or 64 total. By sharing rows between corresponding terms of high-order and low-order bits, approximately 32 rows will be needed. If four eight-bit PLAs are used, each will need only 16 product terms for these functions.

• *PLA for complete ALU*

Figure 9 shows a PLA for an eight-bit group for an ALU containing overflow, zero detection, and the three Boolean functions. It contains a total of 54 product terms which are compressed into 33 rows: 22 for the basic add function, 12 additional for overflow and zero detection, 16 more for the Boolean functions, three for zero detection of the Boolean functions, and one for forcing *ones* to external XORs for the complement sum outputs.

Two function selection bits, X and Y, are used to select any one of four functions. They are encoded as follows: 00, 01, 10, and 11 for ADD, AND, OR, and XOR, respectively. There are two function select decoders, one for each side of the AND array. Each provides a complete two-input decode of X and Y. This permits any product term to be selected by any combination of the four functions.

Conclusions

A practical method has been presented for designing large adders in which the outputs are obtained in one pass through any PLA. The adders are broken into small groups of bits, each using a separate PLA. The input carry to each group enables the correct sums to be formed using additional circuits at the outputs of the PLAs. As improvements continue to evolve in the design of PLA adders, this technique should still permit significant reduction of silicon area for large adders.

The use of separate PLAs for each small group of bits also allows other common ALU functions, such as detection of a zero sum, and Boolean operations, to be efficiently included in the PLAs. Use of smaller PLAs, along with other techniques described herein, will also help to reduce the delay of the ALU and thus reduce the cycle time of the system.

While PLA macros have already appeared in commercially available microprocessors for use in control logic, it is now becoming practical to use them for the ALU as well. This will permit use of common automated design processes for both control logic and data paths. Use of dynamically clocked logic will also become more practical and will result in lower power levels and higher levels of integration.

References

1. W. N. Carr and J. P. Mize, *MOS/LSI Design and Application*, McGraw-Hill Book Co., Inc., New York, 1972, Ch. 8.
2. L. Dang, P. B. Ashkin, R. Yee, and M. O'Brien, "A CMOS/SOS 16-Bit Parallel Microprocessor," *Digest of Technical Papers*, IEEE International Solid State Circuits Conference, Philadelphia, February 1977, pp. 134-135.
3. C. Erickson, H. K. Hingarh, R. Moeckel, and D. Wilnai, "A 16-Bit Monolithic I^3L Processor," *Digest of Technical Papers*, IEEE International Solid State Circuits Conference, Philadelphia, February 1977, pp. 140-141, 247. (I^3L is a registered trademark of Fairchild Camera and Instrument Corp., Mountain View, CA. It stands for isoplanar integrated injection logic.)
4. D. Stamm, D. Budde, and W. Morgan, "A Single Chip, Highly Integrated, User Programmable Microcomputer," *Digest of Technical Papers*, IEEE International Solid State Circuits Conference, Philadelphia, February 1977, pp. 142-143.
5. A. Weinberger, "High-Speed Programmable Logic Array Adders," *IBM J. Res. Develop.* **23,** 163-178 (1979).
6. A. Weinberger, "Adder With Fast Detection of Sum Equal to Zeroes or Radix Minus One," U.S. Patent 3,983,382, 1976.

Received April 17, 1979; revised June 18, 1979

The author is located at the IBM General Systems Division laboratory, 11400 Burnett Road, Austin, Texas 78758.