FILE : ALU DESIGN

Huey Ling

High-Speed Binary Adder

Based on the bit pair (a_i, b_i) truth table, the carry propagate p_i and carry generate g_i have dominated the carry-look-ahead formation process for more than two decades. This paper presents a new scheme in which the new carry propagation is examined by including the neighboring pairs $(a_i, b_i; a_{i+1}, b_{i+1})$. This scheme not only reduces the component count in design, but also requires fewer logic levels in adder implementation. In addition, this new algorithm offers an astonishingly uniform loading in fan-in and fan-out nesting.

Introduction

The traditional recursive formula for carry propagation has dominated the carry handling process in the computer industry for more than two decades. Today, adder designs based on a similar technique include Amdahl V6, IBM 168, and IBM 3033.

The recursive formulation of carry is based on the bit pair (a_i, b_i) truth table. By examining the local bit pair, carry propagate p_i and carry generate g_i are formed. The high-order carries are generated by nesting the p_i and g_i together. By considering the adjacent bit pairs $(a_i, b_i; a_{i+1}, b_{i+1})$, a new recursive formula is obtained for new carry propagation. The comparison between this new scheme and the existing scheme will be discussed in the following sections. The detailed implementation, circuits, and logic level count are also included. Surprisingly, this method offers an astonishingly uniform loading in fan-in/fan-out nesting.

The formation of new carry and sum

This paper introduces a new approach to represent the new carry formation and propagation based on the concept of the complementing signal which was introduced in 1965 [1]. To examine the impact of this complementing signal in performing binary addition and complementing signal look-ahead, one should evaluate the formation of H_i and H_{i+1} as a function of neighboring bit pairs $(i, i+1)$. Let us consider adding two binary numbers A and B together, where

$$A = a_0 2^n + a_1 2^{n-1} + a_2 2^{n-2} + \cdots + a_i 2^{n-i} + \cdots + a_n 2^0 ;$$

$$B = b_0 2^n + b_1 2^{n-1} + b_2 2^{n-2} + \cdots + b_i 2^{n-i} + \cdots + b_n 2^0 .$$

The relation among the new carry (H_i, H_{i+1}) and the neighboring bit pairs $(a_i, b_i; a_{i+1}, b_{i+1})$ can be expressed as in Table 1 [1]; all of these are generated by a_i, b_i or transmitted through the low-order bits, $i+1, i+2, \cdots$, with the transmitting-enable switch ON. This signal or new carry can only be terminated when the inhibitor is ON $(a_{i+1} + b_{i+1} = 0)$. H_i plays both regular carry and complementing signal roles in performing binary addition.

By grouping all the H_i, we obtain

$$\begin{aligned} H_i &= f(1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15) \\ &= a_i b_i + H_{i+1}(\bar{a}_{i+1} b_{i+1} + a_{i+1} \bar{b}_{i+1} + a_{i+1} b_{i+1}) \\ &= a_i b_i + H_{i+1}(a_{i+1} + b_{i+1}) = k_i + H_{i+1} T_{i+1} , \end{aligned} \tag{1}$$

where k_i is the new complementing signal, H_{i+1} is the previous complementary signal, and T_{i+1} is the previous carry enable switch or the previous stage propagate.

Equation (1) shows that new carry H_i can be formed locally by k_i or produced remotely; H_{i+1} can be produced with the remote stage carry inhibitor not ON $(a_{i+1} + b_{i+1}$

Copyright 1981 by International Business Machines Corporation. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to *republish* other excerpts should be obtained from the Editor.

= 1). The formation of sum S_i can be expressed by a similar process. The truth table for S_i is shown in Table 2.

By grouping all the S_i, we obtain

$$S_i = f(1, 2, 3, 4, 5, 6, 8, 9, 10, 13, 14, 15) ;$$

$$1, 2, 3 \rightarrow \bar{a}_i\bar{b}_iH_{i+1}(\bar{a}_{i+1}b_{i+1} + a_{i+1}\bar{b}_{i+1} + a_{i+1}b_{i+1})$$
$$= \bar{a}_i\bar{b}_iH_{i+1}(a_{i+1} + b_{i+1})$$
$$= [a_ib_i + H_{i+1}(a_{i+1} + b_{i+1})]\bar{a}_i\bar{b}_i$$
$$= H_i\overline{T}_i ;$$

$$5, 6, 9, 10 \rightarrow (a_i \forall b_i)(a_{i+1} \forall b_{i+1})\overline{H}_{i+1}$$
$$= (a_i \forall b_i)(a_{i+1} \forall b_{i+1})\overline{H}_{i+1} ;$$

$$4, 8 \rightarrow (a_i \forall b_i)(\bar{a}_{i+1}\bar{b}_{i+1}) ;$$

$$4, 5, 6, 8, 9, 10 \rightarrow (a_i \forall b_i)[\overline{H}_{i+1}(a_{i+1} \forall b_{i+1}) + \bar{a}_{i+1}\bar{b}_{i+1}]$$
$$= (a_i + b_i)(\bar{a}_i + \bar{b}_i)[\overline{H}_{i+1}(a_{i+1} \forall b_{i+1})$$
$$+ \bar{a}_{i+1}\bar{b}_{i+1}\overline{H}_{i+1} + \bar{a}_{i+1}\bar{b}_{i+1}]$$
$$= (a_i + b_i)(\bar{a}_i + \bar{b}_i)(\overline{H}_{i+1} + \bar{a}_{i+1}\bar{b}_{i+1}) ;$$

$$H_i = a_ib_i + H_{i+1}(a_{i+1} + b_{i+1}) ;$$

$$\overline{H}_i = (\bar{a}_i + \bar{b}_i)(\overline{H}_{i+1} + \bar{a}_{i+1}\bar{b}_{i+1}) ;$$

$$4, 5, 6, 8, 9, 10 \rightarrow (a_i + b_i)\overline{H}_i = T_i\overline{H}_i ;$$

$$13, 14, 15 \rightarrow a_ib_iH_{i+1}(\bar{a}_{i+1}b_{i+1} + a_{i+1}\bar{b}_{i+1} + a_{i+1}b_{i+1})$$
$$= a_ib_iH_{i+1}(a_{i+1} + b_{i+1})$$
$$= k_iH_{i+1}T_{i+1} ;$$

$$S_i = f(1, 2, 3, 4, 5, 6, 8, 9, 10, 13, 14, 15)$$
$$= H_i\overline{T}_i + T_i\overline{H}_i + k_iH_{i+1}T_{i+1}$$
$$= (H_i \forall T_i) + k_iH_{i+1}T_{i+1} . \quad (2)$$

We have obtained a set of recursive formulae for both new carry H_i and sum S_i. They are different from the conventional process. Before discovering the difference, let us examine the carry-look-ahead process.

New carry-look-ahead

For ease of discussion, let us consider $i = 31$. We have

$$H_{31} = k_{31} + H_{32}T_{32} . \quad (3a)$$

By substituting $i = 30$, 29, and 28 in (3a), we obtain

$$H_{28} = k_{28} + T_{29}k_{29} + T_{29}T_{30}k_{30} + T_{29}T_{30}T_{31}k_{31}$$
$$+ T_{29}T_{30}T_{31}T_{32}k_{32} . \quad (3b)$$

By following a similar process, we obtain

$$H_{24} = k_{24} + T_{25}k_{25} + T_{25}T_{26}k_{26} + T_{25}T_{26}T_{27}k_{27}$$
$$+ T_{25}T_{26}T_{27}T_{28}H_{28} ;$$

Table 1 The relation of new carry H_i with H_{i+1} and its neighboring bit pairs $(a_i, b_i; a_{i+1}, b_{i+1})$.

i	$H_i = 1$ in relation with H_{i+1}	a_i	b_i	a_{i+1}	b_{i+1}
0	$H_i = 0$	0	0	0	0
1	$H_{i+1} = 1$	0	0	0	1
2	$H_{i+1} = 1$	0	0	1	0
3	$H_{i+1} = 1$	0	0	1	1
4	$H_i = 0$	0	1	0	0
5	$H_{i+1} = 1$	0	1	0	1
6	$H_{i+1} = 1$	0	1	1	0
7	$H_{i+1} = 1$	0	1	1	1
8	$H_i = 0$	1	0	0	0
9	$H_{i+1} = 1$	1	0	0	1
10	$H_{i+1} = 1$	1	0	1	0
11	$H_{i+1} = 1$	1	0	1	1
12	$H_{i+1} = X$	1	1	0	0
13	$H_{i+1} = X$	1	1	0	1
14	$H_{i+1} = X$	1	1	1	0
15	$H_{i+1} = X$	1	1	1	1

Table 2 Sum S_i formation.

i	S_i	a_i	b_i	a_{i+1}	b_{i+1}
0	0	0	0	0	0
1	$H_{i+1} = 1$	0	0	0	1
2	$H_{i+1} = 1$	0	0	1	0
3	$H_{i+1} = 1$	0	0	1	1
4	1	0	1	0	0
5	$H_{i+1} = 0$	0	1	0	1
6	$H_{i+1} = 0$	0	1	1	0
7	0	0	1	1	1
8	1	1	0	0	0
9	$H_{i+1} = 0$	1	0	0	1
10	$H_{i+1} = 0$	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	$H_{i+1} = 1$	1	1	0	1
14	$H_{i+1} = 1$	1	1	1	0
15	$H_{i+1} = 1$	1	1	1	1

$$H_{20} = k_{20} + T_{21}k_{21} + T_{21}T_{22}k_{22} + T_{21}T_{22}T_{23}k_{23} + T_{21}T_{22}T_{23}T_{24}H_{24}\ ; \quad (4)$$

$$H_{16} = k_{16} + T_{17}k_{17} + T_{17}T_{18}k_{18} + T_{17}T_{18}T_{19}k_{19} + T_{17}T_{18}T_{19}T_{20}H_{20}. \quad (5)$$

By substituting (3b) for (5), we obtain

$$H_{16} = H^*_{16} + I^*_{16}H_{20}, \quad (6)$$

where

$$H^*_{16} = k_{16} + T_{17}k_{17} + T_{17}T_{18}k_{18} + T_{17}T_{18}T_{19}k_{19}\ ; \quad (7)$$

$$I^*_{16} = T_{17}T_{18}T_{19}T_{20}\ . \quad (8)$$

By substituting (3a) and (3b) for (5), we obtain

$$H_{16} = H^*_{16} + I^*_{16}H^*_{20} + I^*_{16}I^*_{20}H^*_{24} + I^*_{16}I^*_{20}I^*_{24}H_{28}\ . \quad (9)$$

The asterisk of H^*_{16} represents the fact that H^*_{16} can be implemented with one level of logic. Based on current switching technology, both fan-in and fan-out are equal to four with eight-emitter dotting; H_{16} can be implemented with two levels of logic.

Comparison with the existing scheme

Based on the local bit pair (a_i, b_i), carry C_i and sum S_i can be written in the form

$$C_i = g_i + C_{i+1}p_i, \qquad g_i = a_ib_i\ ; \quad (10)$$

$$S_i = a_i \forall\ b_i \forall\ C_{i+1}, \quad p_i = a_i + b_i\ .$$

For $i = 16$, we have

$$C_{16} = g_{16} + C_{17}p_{16}\ .$$

By substituting $i = 17, 18, \cdots, 19$, C_{16} can be rewritten as

$$\begin{aligned} C_{16} &= g_{16} + p_{16}(g_{17} + p_{17}g_{18} + p_{17}p_{18}g_{19} + p_{17}p_{18}p_{19}C_{20}) \\ &= g_{16} + p_{16}g_{17} + p_{16}p_{17}g_{18} + p_{16}p_{17}p_{18}g_{19} \\ &\quad + p_{16}p_{17}p_{18}p_{19}C_{20} \\ &= G_{16p} + P_{16p}C_{20}\ , \end{aligned} \quad (11)$$

where G_{16p} and P_{16p} are the grouping of the following terms:

$$G_{16p} = g_{16} + p_{16}g_{17} + p_{16}p_{17}g_{18} + p_{16}p_{17}p_{18}g_{19}\ ; \quad (12)$$

$$P_{16p} = p_{16}p_{17}p_{18}p_{19}\ . \quad (13)$$

Similarly, C_{16} can be written in terms of C_{28}:

$$C_{16} = G_{16p} + P_{16p}G_{20p} + P_{16p}P_{20p}G_{24p} + P_{16p}P_{20p}P_{24p}G_{28p}\ .$$

Equations (6), (7), and (8) are similar to (11), (12), and (13); however, H^*_{16} can be implemented with one level of logic, whereas G_{16p} cannot. By expanding (7) and (12) we obtain

$$\begin{aligned} H^*_{16} &= a_{16}b_{16} + (a_{17} + b_{17})a_{17}\,b_{17} \\ &\quad + (a_{17} + b_{17})(a_{18} + b_{18})a_{18}b_{18} \\ &\quad + (a_{17} + b_{17})(a_{18} + b_{18})(a_{19} + b_{19})a_{19}b_{19} \\ &= a_{16}b_{16} + a_{17}b_{17} + a_{17}a_{18}b_{18} + b_{17}a_{18}b_{18} \\ &\quad + a_{17}a_{18}a_{19}b_{19} + a_{17}b_{18}a_{19}b_{19} \\ &\quad + b_{17}a_{18}a_{19}b_{19} + b_{17}b_{18}a_{19}b_{19}\ ; \end{aligned} \quad (14)$$

$$\begin{aligned} G_{16p} &= a_{16}b_{16} + (a_{16} + b_{16})a_{17}b_{17} \\ &\quad + (a_{16} + b_{16})(a_{17} + b_{17})a_{18}b_{18} \\ &\quad + (a_{16} + b_{16})(a_{17} + b_{17})(a_{18} + b_{18})a_{19}b_{19} \\ &= a_{16}b_{16} + a_{16}a_{17}b_{17} + b_{16}a_{17}b_{17} \\ &\quad + a_{16}a_{17}a_{18}b_{18} + a_{16}b_{17}a_{18}b_{18} \\ &\quad + b_{16}a_{17}a_{18}b_{18} + b_{16}b_{17}a_{18}b_{18} + a_{16}a_{17}a_{18}a_{19}b_{19} \\ &\quad + a_{16}a_{17}b_{18}a_{19}b_{19} + a_{16}b_{17}a_{18}a_{19}b_{19} + a_{16}b_{17}b_{18}a_{19}b_{19} \\ &\quad + b_{16}a_{17}a_{18}a_{19}b_{19} + b_{16}a_{17}b_{18}a_{19}b_{19} \\ &\quad + b_{16}b_{17}a_{18}a_{19}b_{19} + b_{16}b_{17}b_{18}a_{19}b_{19}\ . \end{aligned} \quad (15)$$

Equation (14) contains eight terms, whereas (15) contains fifteen. With current available technology, the former can be implemented with one level of logic (this is shown in detail in the next section); the latter can only be implemented with two levels of logic.

Let us further examine the ith-digit carry formation. For (1), the carry is generated by local complementing signal k_i, and the remote carry H_{i+1} is controlled by remote bit pair $(a_{i+1} + b_{i+1})$; whereas for (10), the carry is generated by local carry g_i, and the remote carry C_{i+1} is controlled by local bit pair $(a_i + b_i)$. From the carry-look-ahead point of view, (1) offers faster resolution, whereas the latter is one stage slower. That is why (14) contains only eight terms, and (15), fifteen.

To illustrate the step-by-step operation, two examples are given.

Example 1 Assume the contents of A and B registers to be as shown and find their sum;

A register 00000000011010101111101100011001

B register 00000000011011011010101010010111

The k_i and T_i can be implemented with one level of logic:

k_i 00000000011010001101000100010001

T_i 00000000011011111111111101011111

The complementary signals can be implemented by grouping k_i and T_i together. This process requires one level of logic:

H_i 00000000111111111111111100111111

The sum digit S_i is implemented in parallel with H_i; the result of H_i will force S_i to select one value between $H_i = 0$ and $H_i = 1$:

S_i 00000000110110001101000001110000

This example demonstrates that it is possible to implement a 32-bit adder with three levels of logic with the hardware constraints indicated in the previous section. The detailed implementation of S_i is discussed in the next section.

Example 2 Assuming that the contents of index, base, and displacement registers are as shown, compute the virtual address. (To test the generality of this scheme, odd contents are purposely chosen; in the normal mode of operation, an EXCPN will occur.)

Index register 00000000000111010111011101011011

Base register 00000000000001011100100111011101

Displacement register 101111010111

To implement the carry-save adder (CSA) requires one level of logic:

s_i 00000000000110001111010101010001

c_i 00000000000010100001011110111110

Implementation of k_i and T_i requires one additional level of logic:

k_i 00000000000010000001010100010000

T_i 00000000000110101111011111111111

Implementation of the complementary signal requires one level of logic to group k_i and T_i together:

H_i 00000000001110011111111111110000

The address digit S_i is implemented in parallel with H_i; the result of H_i will force S_i to select one value between $H_i = 0$ and 1:

S_i 00000000001000110000110100001111

This example demonstrates that the AGEN adder can be implemented with four rather than six levels of logic, as is the case in current machine organization. The detailed implementation of S_i (the address) is discussed in the next section. The logic implementation of every fourth bit (i = 31, 27, 23, 19, 16, 15, 11, 7, 3, 0) is shown in the Appendix of this paper.

Implementation

The detailed implementation can be divided into two categories: binary addition and subtraction, and address generation.

- *Addition and subtraction*

Equation (3b) is a general representation of the new carry-look-ahead process. For ADDITION, $k_{32} = 0$; therefore, the fifth term in (3b) is dropped and H_{28} can be written as

$$H_{28} = k_{28} + T_{29}k_{29} + T_{29}T_{30}k_{30} + T_{29}T_{30}T_{31}k_{31} . \quad (4a)$$

For SUBTRACTION, there is a HOT ONE carry input from bit 31; thus H_{28} can be written as

$$H_{28} = k_{28} + T_{29}k_{29} + T_{29}T_{30}k_{30} + T_{29}T_{30}T_{31}k_{31} + T_{29}T_{30}T_{31} . \quad (4b)$$

Equation (2) shows that S_i is a function of H_i and H_{i+1}. For ease of implementation, this equation is rewritten in the form

$$\begin{aligned} S_i &= (H_i \forall T_i) + k_i H_{i+1} T_{i+1} \\ &= [(k_i + H_{i+1}T_{i+1}) \forall T_i] + k_i H_{i+1} T_{i+1} \\ &= H_{i+1}(\bar{T}_i T_{i+1} + k_i T_{i+1}) \\ &\quad + \bar{H}_{i+1}\bar{k}_i T_i + k_i \bar{T}_i + \bar{k}_i T_i \bar{T}_{i+1} . \end{aligned} \quad (16)$$

Equation (16) demonstrates that S_i can be written in the conditional form

$$S_i(H_{i+1} = 0) = k_i \forall T_i ;$$

$$S_i(H_{i+1} = 1) = \bar{T}_i T_{i+1} + k_i T_{i+1} + k_i \bar{T}_i + \bar{k}_i T_i \bar{T}_{i+1} .$$

The general expression of SUM S_i can be written as

$$S_i = H_{i+1}(a_{i+1} + b_{i+1})\overline{(a_i \forall b_i)} + \bar{H}_{i+1}(a_i \forall b_i) + \overline{(a_{i+1} + b_{i+1})}(a_i \forall b_i) .$$

For $i = 31$, we have

$$S_{31} = H_{32}(a_{32} + b_{32})\overline{(a_{31} \forall b_{31})} + \bar{H}_{32}(a_{31} \forall b_{31}) + \overline{(a_{32} + b_{32})}(a_{31} \forall b_{31}) .$$

For $i = 0$, we obtain

$$S_0 = H_1(a_1 + b_1)\overline{(a_0 \forall b_0)} + \bar{H}_1(a_0 \forall b_0) + \overline{(a_1 + b_1)}(a_0 \forall b_0) ; \quad (17)$$

$$\begin{aligned} H_1 &= k_1 + T_2 k_2 + T_2 T_3 k_3 + T_2 T_3 T_4 H_4 \\ &= H_1^* + I_1^* H_4 ; \end{aligned} \quad (18)$$

$$H_4 = k_4 + T_5k_5 + T_5T_6k_6 + T_5T_6T_7k_7 + T_5T_6T_7T_8H_8$$
$$= H^*_4 + I^*_4H_8 ; \quad (19)$$

$$H_8 = H^*_8 + I^*_8H_{12} ; \quad (20)$$

$$H_{12} = H^*_{12} + I^*_{12}H_{16} . \quad (21)$$

By substituting (21) for (20), we have

$$H_8 = H^*_8 + I^*_8H^*_{12} + I^*_8I^*_{12}H_{16} . \quad (22)$$

Similarly, we obtain H_4 and H_1:

$$H_4 = H^*_4 + I^*_4H^*_8 + I^*_4I^*_8H^*_{12} + I^*_4I^*_8I^*_{12}H_{16}; \quad (23)$$

$$H_1 = H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12} + I^*_1I^*_4I^*_8I^*_{12}H_{16}. \quad (24)$$

By substituting Eqs. (22), (23), and (24) for Eqs. (18), (19), (20), and (21), Eq. (17) can be written as

$$\begin{aligned} S_0 &= (H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12} \\ &\quad + I^*_1I^*_4I^*_8I^*_{12}H_{16})(a_1 + b_1)\overline{(a_0 \forall b_0)} \\ &\quad + \overline{(H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12}} \\ &\quad + \overline{I^*_1I^*_4I^*_8I^*_{12}H_{16})}(a_0 \forall b_0) + \overline{(a_1 + b_1)}(a_0 \forall b_0) ; \\ &= (H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12})(a_1 + b_1) \\ &\quad \times \overline{(a_0 \forall b_0)} + I^*_1I^*_4I^*_8I^*_{12}H_{16}(a_1 + b_1)\overline{(a_0 \forall b_0)} \\ &\quad + \overline{(H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12})} \\ &\quad \times \overline{(I^*_1I^*_4I^*_8I^*_{12}} + \bar{H}_{16})(a_0 \forall b_0) \\ &\quad + \overline{(a_1 + b_1)}(a_0 \forall b_0) . \end{aligned} \quad (25)$$

By using the Sklansky conditional-sum method [2], (25) can be written as

$$\begin{aligned} S_0 &= (H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12}) \\ &\quad \times (a_1 + b_1)\overline{(a_0 \forall b_0)} + \overline{(a_1 + b_1)}(a_0 \forall b_0) \\ &\quad + \overline{(H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12})} \\ &\quad \times (a_0 \forall b_0) \quad [H_{16} = 0] \\ &\quad + I^*_1I^*_4I^*_8I^*_{12}(a_1 + b_1)\overline{(a_0 \forall b_0)} \quad [H_{16} = 1] \\ &\quad + \overline{(H^*_1 + I^*_1H^*_4 + I^*_1I^*_4H^*_8 + I^*_1I^*_4I^*_8H^*_{12})} \\ &\quad \times \overline{(I^*_1I^*_4I^*_8I^*_{12})}(a_0 \forall b_0) \quad [H_{16} = 1]. \end{aligned}$$

The hardware implementation of S_0 is included in the Appendix.

Equation (9) has indicated that H_{16} can be implemented with two levels of logic. Let us examine the individual terms of S_0. It is clearly pointed out that they also require only two levels of logic to be implemented. That is to say, when H_{16} is ready, S_0 can be obtained by using one additional level of logic. We have proved, by using current switching logic, that one can implement a 32-bit adder by consuming only three levels of logic.

- *Address generation*

In the address generation process, we are dealing with positive numbers only. Therefore, $k_{32} = 0$. The output of the (3, 2) carry-save adder provides the s_i and c_{i+1} corresponding to the a_i and b_i bit pairs. In addition, X_i and B_i both are 32 bits in length. However, D_i has only a 12-bit width. For i = 0–18, the output of the carry-save adder has a special pattern; s_i and c_{i+1} will not have the form

111—101—1111—11

101—111—1111—11 .

In general, the output of CSA will appear as

1111—01—001—10110

0001—01—001—10010 .

Therefore, for i = 19–31, S_i appears as usual:

$$S_i = (H_i \forall T_i) + k_iH_{i+1}T_{i+1} .$$

For i = 0–18, S_i appears as

$$S_i = (H_i \forall T_i) .$$

The detailed implementations for i from 0, 3, 7, 11, 15, 16, 19, 23, 27, and 31 are shown in the Appendix.

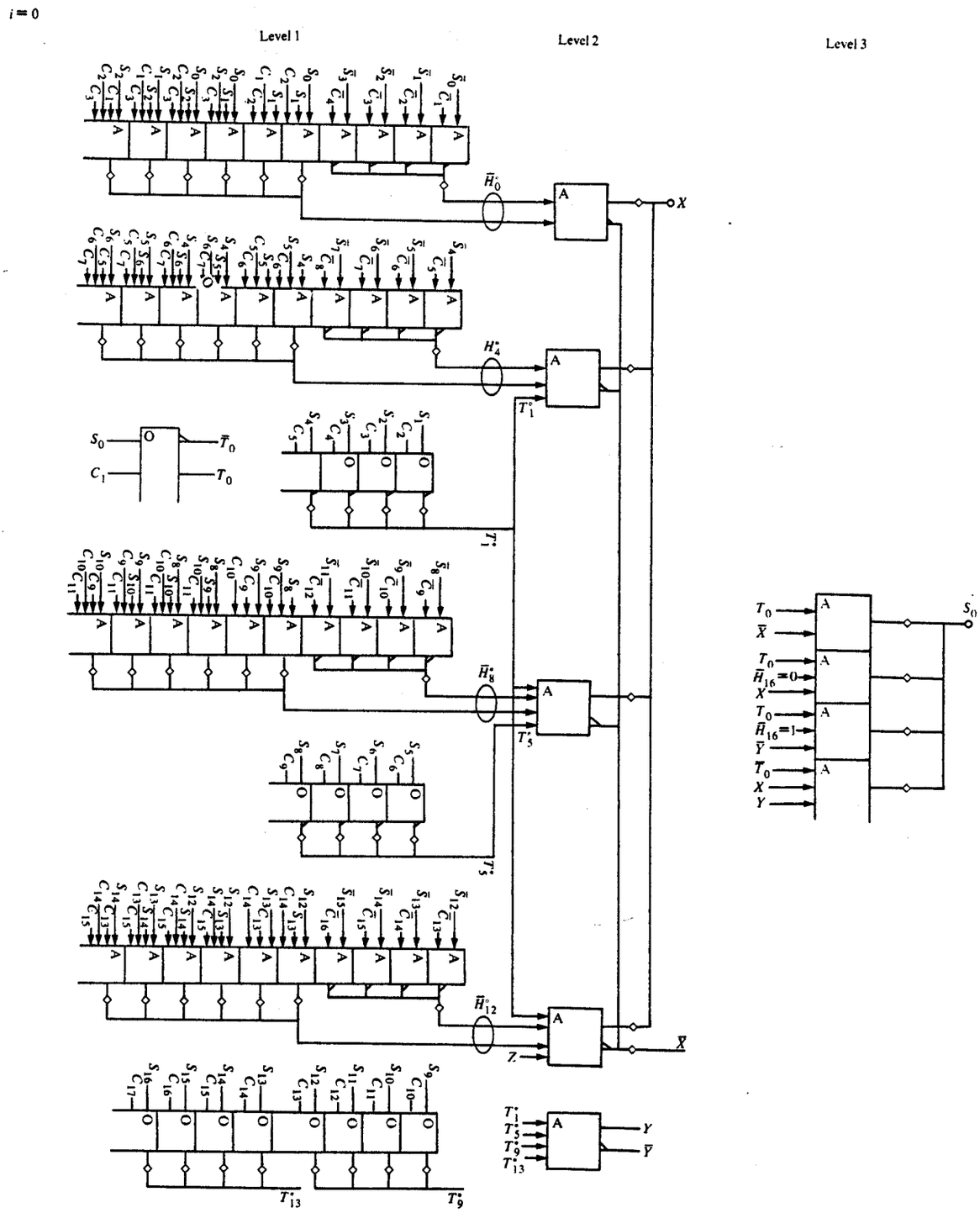
Summary

It is intended in this paper to speed up the carry propagation for examining two bit pairs. The formulation of H^*_{16} contains eight terms as compared to that of the regular carry-look-ahead process, where G_{16p} contains fifteen terms. It is possible to implement H^*_{16} with one level of logic, whereas it is not possible with G_{16p}. The formulation of sum S_i in this new process will contain slightly more terms; however, they are not in the critical path.

References

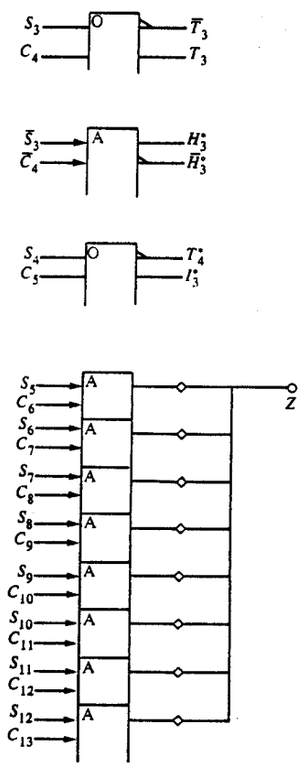
1. H. Ling, "High Speed Binary Parallel Adder," *IEEE Trans. Electron. Computers* **EC-15**, 799–802 (1966).
2. J. Sklansky, "Conditional-Sum Addition Logic," *IEEE Trans. Electron. Computers* **EC-9**, 226–231 (1960).

Appendix

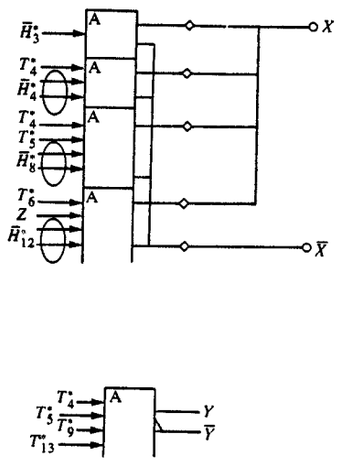


$i = 3$

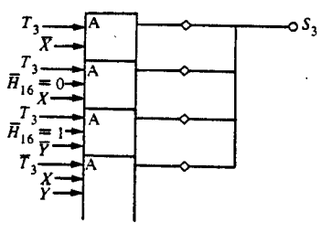
Level 1



Level 2

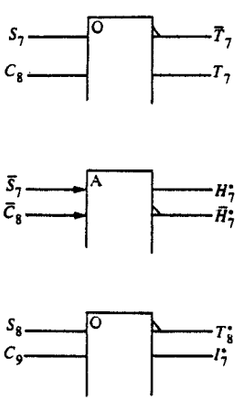


Level 3

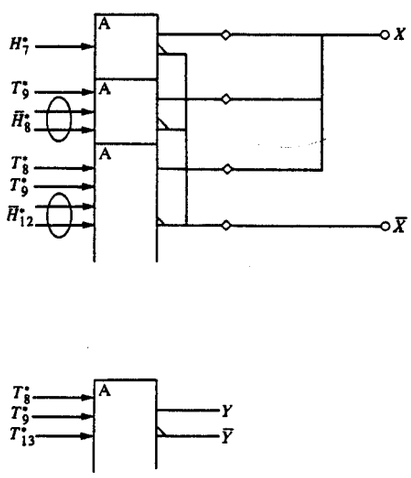


$i = 7$

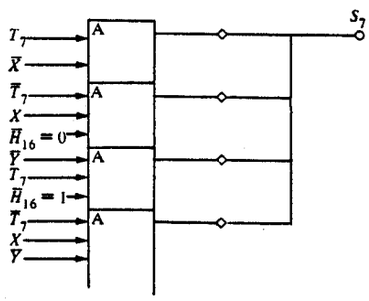
Level 1



Level 2

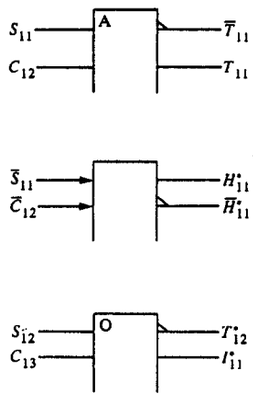


Level 3

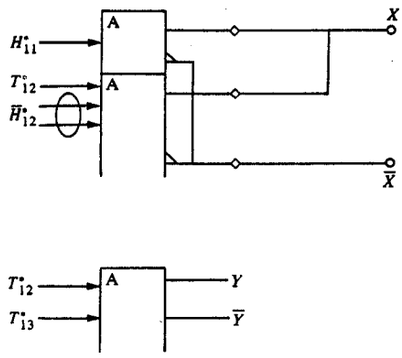


$i = 11$

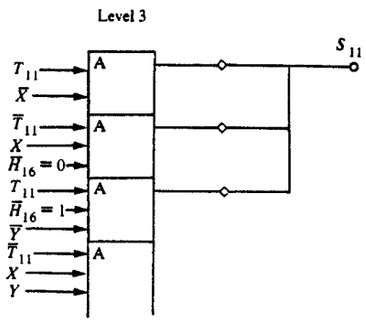
Level 1



Level 2

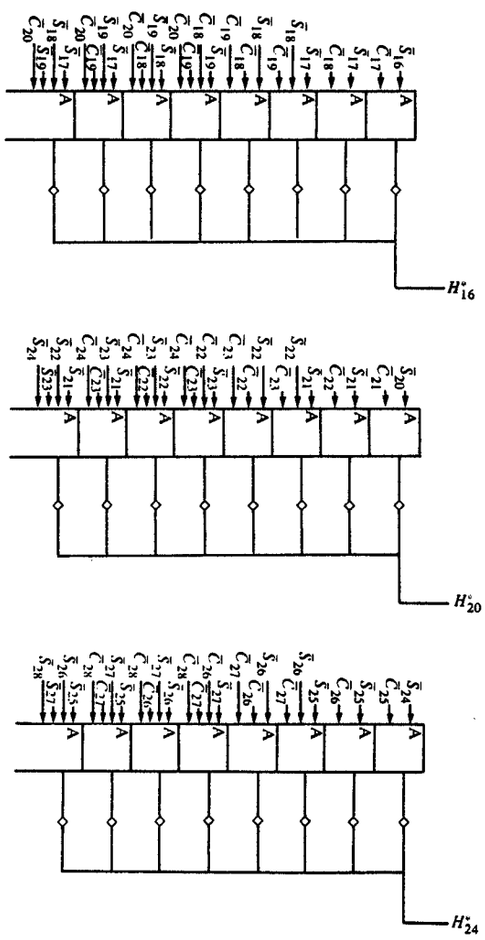


Level 3

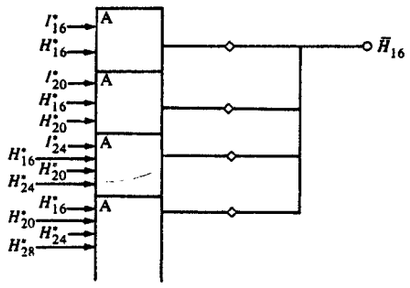


$i = 16$

Level 1



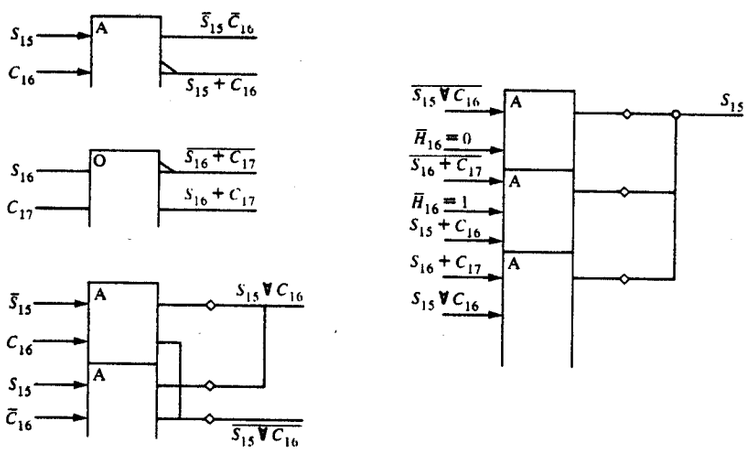
Level 2



Level 3

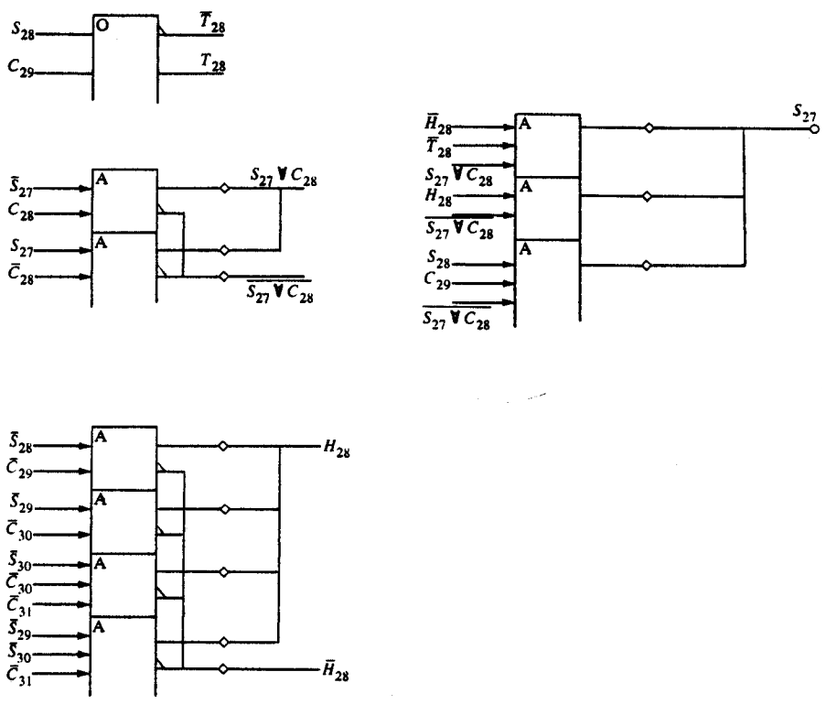
$i = 15$

Level 1 Level 2 Level 3

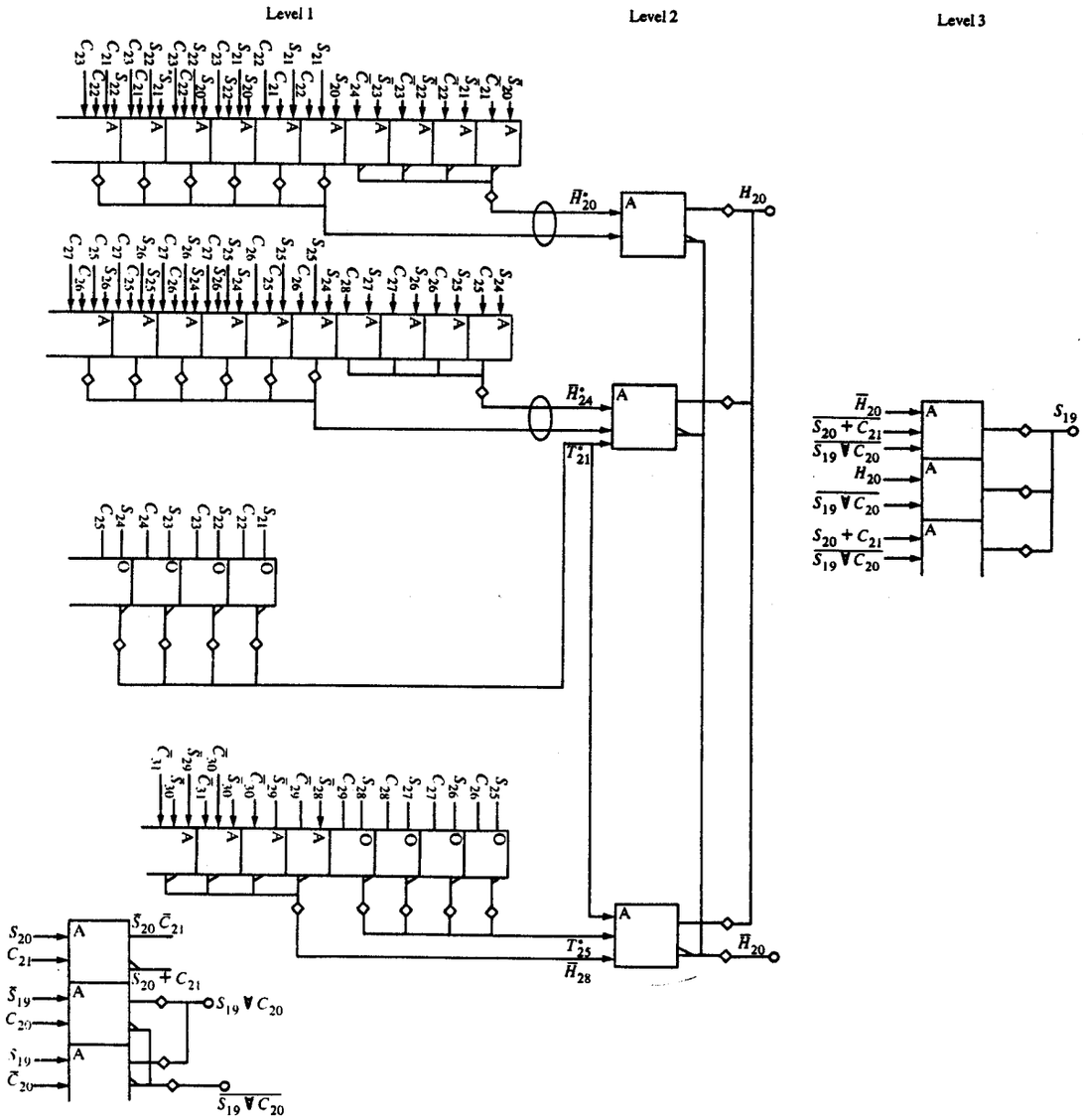


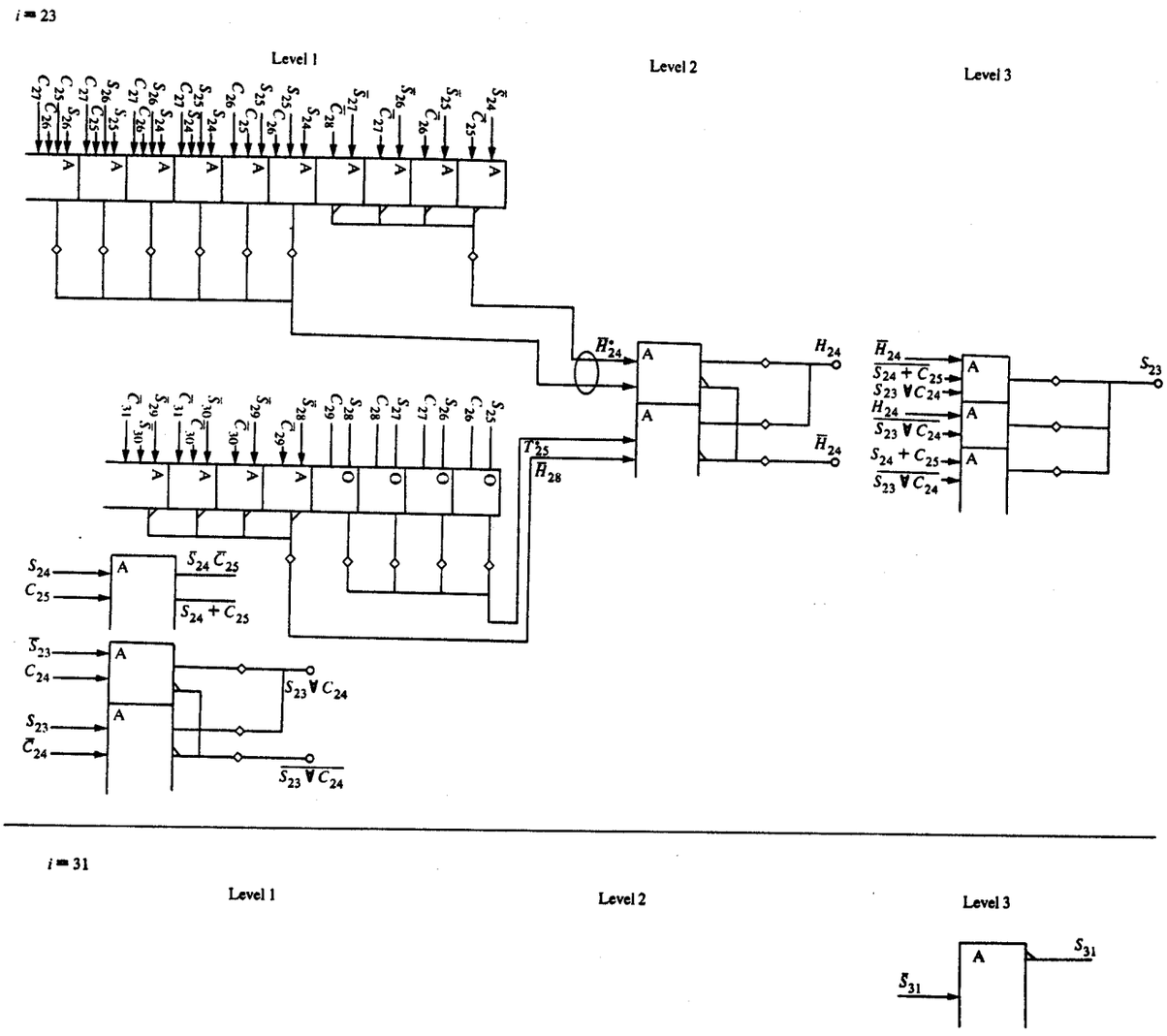
$i = 27$

Level 1 Level 2 Level 3



$i = 19$


Level 1
Level 2
Level 3
A
O
H_{20}
H_{24}
T_{21}
T_{25}
H_{28}
S_{19}
$S_{20} + C_{21}$
$S_{19} \forall C_{20}$



Received September 11, 1980; revised November 26, 1980

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.