Generalized Earliest-First Fast Addition Algorithm

Wen-Chang Yeh and Chein-Wei Jen, *Member, IEEE*

Abstract—This paper presents a generalized earliest-first (GEF) addition algorithm to accelerate carry propagation addition (CPA). A set of operators and notations have been developed to describe and analyze traditional carry-lookahead or conditional-sum-based algorithms. The proposed GEF algorithm schedules bit-level operations of CPA in an earliest-first manner to reduce the overall latency. With the aid of the developed operators and notations, the algorithm can be generalized to any algorithm based on carry-lookahead or conditional-sum rule. An adder generated by using the GEF algorithm outperforms traditional algorithms when inputs do not arrive simultaneously.

Index Terms—Carry-propagation adder, final adder, carry-lookahead, conditional-sum.

1 INTRODUCTION

Fast carry-propagation addition (CPA) is one of the most fundamental operations in modern digital computers and has been used ubiquitously in various applications. Although it has been studied extensively in the literature, there are still new algorithms being proposed to further improve the performance.

Conventionally, fast addition is achieved by using parallel prefix parallel adders, which can be categorized into two classes: carry-lookahead and conditional-sum-based algorithms. However, there are two major problems for the traditional algorithms. At first, it is hard to compare or evaluate their performance and determine the circuit as well as the routing complexity for different adder algorithms at algorithm-level. Therefore, different optimization techniques have been developed for each algorithm. For a specific application, a designer or a circuit optimization program has to evaluate various adder schemes to determine which of them is the best one. The second problem is that traditional algorithms only consider the special case where all the inputs arrive simultaneously. When some of the inputs arrive earlier than the others, they cannot benefit from the unequal delay profile.

In this paper, we will propose a generalized addition algorithm to solve the two problems. For the first problem, we present a set of operators and notations to describe several popular fast addition algorithms, including carry-lookahead adder (CLA), conditional-sum adder (CSMA), conditional-carry adder (CCA), and ELM adder (ELMA) [1], [1], [2], [3], [4], [5], [6], [7]. By using the algorithm-level representation, we can show that the performance of each adder algorithm can be determined and be compared by the operators. We can also compare the topology and the required routing resource at algorithm level.

The second problem has been identified in the research work for the final adder of parallel multiplier, and several techniques have been developed to exploit the unequal delay profile property [8], [9], [10]. The Left-to-Right-Carry-Free algorithm proposed in [8] requires n-level conversions to generate n-bit MSB products. It was improved in [9] by reducing the levels required. However, this approach still cannot fully exploit the unequal delay profile because it applies to the MSB-part only. In [10], a hybrid adder structure, which consists of ripple-carry adder, carry-skip adder, and conditional-sum adder blocks, was proposed. However, their empirical methodology is not general enough and requires many trials to determine the final adder partition boundary for different sizes of multiplier, thus increasing design effort. More importantly, these designs were developed specific to the final addition of multiplier and, therefore, they are not general enough to handle other different unequal delay profiles.

To take the advantage of unequal delay profile, two algorithms based on the developed representation are presented herein. The first one, dual-bit forward prediction (DFP) algorithm, constructs two-bit block in each iteration and builds the adder from LSB to MSB. It is a generalized version of the MLCSMA algorithm presented in [11]. In order to fully exploit the unequal delay profile, a generalized earliest-first (GEF) algorithm is proposed to improve DFP algorithm. It processes two or more terms at each iteration and builds the adder from the earliest arriving signals without having to process less significant bits first.

The organization of this paper is as follows: Section 2 reviews some important previous works and Section 3 derives the operators and the notations to describe conventional addition algorithms. Section 4 presents the proposed DFP and GEF algorithms. The relationships between the previously proposed MLCSMA algorithm and the DFP algorithm are demonstrated in Section 4.1. The GEF algorithm and its hardware complexity analysis are presented in

- *W.-C. Yeh is with the ZyDAS Technology Corporation, Hsinchu, Taiwan, ROC E-mail: wcyeh@zydas.com.tw.*
- *C.-W. Jen is with the Department of Electronics Engineering and Institute of Electronics, National Chiao-Tung University, 1001 Ta Hsueh Rd., Hsinchu 30050, Taiwan, ROC E-mail: cwjen@twins.ee.nctu.edu.tw.*

Manuscript received 11 Oct. 2001; revised 22 Aug. 2002; accepted 21 Oct. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 115180.

Section 4.2 and in Section 4.3, respectively. The conclusions are presented in Section 5.

2 Previous Work

Among various addition algorithms, carry-lookahead adder (CLA) and conditional-sum adder (CSMA) are the two most popular schemes. The major difference of the two algorithms results from the generation of carry signal. They can be formulated as follows: Let A and B be the inputs for an L-bit parallel adder in two's complement format. Sum and carry signals can be computed using the following equations:

$$s_i = \quad p_i \oplus c_{i-1} \tag{1}$$

$$c_i = \quad c_{i-1} \cdot r_i + \overline{c_{i-1}} \cdot g_i \tag{2}$$

$$c_i = \quad g_i + p_i \cdot c_{i-1} \tag{3}$$

$$c_i = \quad g_i + r_i \cdot c_{i-1}, \tag{4}$$

where the p_i, g_i, and r_i are defined as $p_i = a_i \oplus b_i$, $g_i = a_i \cdot b_i$, and $r_i = a_i + b_i$, respectively. Note that $c_0 = g_0$ and the subscript "i" denotes the bit position starting from LSB. Equation (1) is used to generate sum and (2) and (3) (or (4)) are used to generate carry for CSMA and CLA, respectively. Both (3) and (4) can be used to generate carry for CLA. To distinguish the two CLA schemes, we use "CLA_I" if (3) is used and "CLA_II" if (4) is used. "CLA" is used to refer to both of them.

Apart from the two algorithms, there are two alternative adder structures proposed to improve the speed or the area-time performance [4], [12]. The conditional-carry adder (CCA) proposed in [4] is based on (2) and the ELMA proposed in [12] is based on (3). Though the developers of new algorithms always claim that the proposed algorithms are superior to old ones via some benchmarks, no proof on the optimality at algorithm level is given. On the other hand, all of the mentioned algorithms were developed assuming that all bits of the two operands will arrive simultaneously. The assumption is invalid in some cases, e.g., the final addition of parallel multiplication. The resulting latency of the addition will be as long as if all of the inputs arrived at the same time as the slowest one.

To clarify the relationships among these adders, we develop a set of notations and operators to obtain the equations for them in Section 3. Then, the GEF algorithm is proposed in Section 4 to generate faster adder for unequal delay profile.

3 Analysis of Traditional Algorithms

It is important to develop algorithm level representations for carry-generation, which is the major difference among the traditional algorithms. This section presents algorithm representations for CLA, ELMA, CSMA, and CCA. The definitions of CCA and ELMA can be found in [4] and [12], respectively.

3.1 Algorithm Formulation

To simplify the analysis, three ternary operators, "$\otimes$," "$\bigtriangledown$," and "$\bigtriangleup$," are defined:

$$a \cdot b + \overline{a} \cdot c = \quad a \otimes (b, c) \tag{5}$$

$$b + (c \cdot a) = \quad a \bigtriangledown (b, c) = c \bigtriangledown (b, a) \tag{6}$$

$$b \oplus (c \cdot a) = \quad a \bigtriangleup (b, c) = c \bigtriangleup (b, a). \tag{7}$$

Note that the "$\bigtriangledown$" operator and the "$\bigtriangleup$" operator correspond to the "G" cell and the "S" cell, respectively, in [12] and the "$\bigtriangledown$" operator is equivalent to the "o" operator defined in [14]. Moreover, if the ternary operator operates on two pairs of Boolean variables, the following rules are applied:

$$\begin{aligned}
(a,b) = (\alpha,\beta) \otimes (\gamma,\delta) &\Rightarrow \begin{cases} a &= \alpha \otimes (\gamma,\delta) \\ b &= \beta \otimes (\gamma,\delta) \end{cases} \\
(a,b) = (\alpha,\beta) \bigtriangledown (\gamma,\delta) &\Rightarrow \begin{cases} a &= \alpha \bigtriangledown (\gamma,\delta) \\ b &= \beta \cdot \delta \end{cases} \\
(a,b) = (\alpha,\beta) \bigtriangleup (\gamma,\delta) &\Rightarrow \begin{cases} a &= \alpha \bigtriangleup (\gamma,\delta) \\ b &= \beta \cdot \delta. \end{cases}
\end{aligned} \tag{8}$$

All of the ternary operators defined above obey the following two important properties:

1. **Associativity**
   $[(a,b)\ \boldsymbol{op}\ (c,d)]\ \boldsymbol{op}\ (e,f) = (a,b)\ \boldsymbol{op}\ [(c,d)\ \boldsymbol{op}\ (e,f)]$
2. **Noncommutative**
   (a,b) ***op*** (c,d) $\neq$ (c,d) ***op*** (a,b),

where the ***op*** represents any one of the ternary operators. The associative property of the "$\bigtriangledown$" operator was proven in [14] and the property of the "$\otimes$" operator in [6]. The proof for the "$\bigtriangleup$" operator is omitted for conciseness. The proof for the properties of the "$\bigtriangleup$" operator is as follows.

Proof of Associativity. To prove this property, the following equation must hold:

$$[(a,b) \bigtriangleup (c,d)] \bigtriangleup (e,f) = (a,b) \bigtriangleup [(c,d) \bigtriangleup (e,f)].$$

Expand the left-hand side (L.H.S.) using (8) to obtain

$$\begin{aligned} L.H.S. &= (a \bigtriangleup (c,d), bd) \bigtriangleup (e,f) \\ &= (a \bigtriangleup (c,d) \bigtriangleup (e,f), bdf). \end{aligned}$$

Similarly, the right-hand side (R.H.S.) can be rewritten as

$$\begin{aligned} R.H.S. &= (a,b) \bigtriangleup (c \bigtriangleup (e,f), df) \\ &= (a \bigtriangleup (c \bigtriangleup (e,f), df), bdf). \end{aligned}$$

The second term of both L.H.S. and R.H.S. is equivalent. Thus, the property holds if the two first terms are equivalent, i.e.,

$$a \bigtriangleup (c,d) \bigtriangleup (e,f) = a \bigtriangleup (c \bigtriangleup (e,f), df).$$

The L.H.S. can be expanded using (7):

$$\begin{aligned} L.H.S. &= c \oplus (a \cdot d) \bigtriangleup (e,f) \\ &= e \oplus ((c \oplus ad) \cdot f) \\ &= e \oplus [f(c \oplus ad)]. \end{aligned}$$

Similarly, for the R.H.S.,

$$\begin{aligned} R.H.S. &= (c \bigtriangleup (e,f)) \oplus (a \cdot df) \\ &= [e \oplus (c \cdot f)] \oplus (adf) \\ &= e \oplus [(cf) \oplus (adf)] \\ &= e \oplus [f(c \oplus ad)]. \end{aligned}$$

Since L.H.S. = R.H.S., $[(a,b) \bigtriangleup (c,d)] \bigtriangleup (e,f) = (a,b) \bigtriangleup [(c,d) \bigtriangleup (e,f)]$ holds. □

Proof of Noncommutative. To prove this property, the following equation must hold:

$$(a,b) \bigtriangleup (c,d) \neq (c,d) \bigtriangleup (a,b).$$

The L.H.S. can be simplified to

$$L.H.S. = (a \bigtriangleup (c,d), bd) = (c \oplus (ad), bd).$$

Similarly, the R.H.S. can be rewritten as

$$R.H.S. = (c \bigtriangleup (a,b), bd) = (a \oplus (cb), bd).$$

The two first terms of L.H.S. and R.H.S. are obviously not equivalent. □

With the aid of the defined operators and the two important properties, the algorithm level representations for the algorithms can be obtained. For CSMA or CCA, the carry signal generation can be written as:

$$c_i = c_0 \otimes (r_1, g_1) \otimes \cdots \otimes (r_{i-1}, g_{i-1}) \otimes (r_i, g_i). \qquad (9)$$

Similarly, for CLA or ELMA, the carry signal generation can be rewritten as:

$$c_i = c_0 \bigtriangledown (g_1, p_1) \bigtriangledown \cdots \bigtriangledown (g_{i-1}, p_{i-1}) \bigtriangledown (g_i, p_i). \qquad (10)$$

Equations (9) and (10) illustrate that any structure used to generate carry signal for CSMA or CCA can also be used to generate carry signals for CLA or ELMA and vice versa.

Now, we consider the generation of sum signals. For CCA, the equation can be obtained by using (1) and (9).

$$s_i = p_i \oplus [c_0 \otimes (r_1, g_1) \otimes \cdots \otimes (r_{i-2}, g_{i-2}) \otimes (r_{i-1}, g_{i-1})]. \quad (11)$$

Alternatively, this equation can be rewritten as (12) for the CSMA structure.

$$s_i = c_0 \otimes (r_1, g_1) \otimes \cdots \otimes (r_{i-2}, g_{i-2}) \otimes [(p_i \oplus r_{i-1}, p_i \oplus g_{i-1})]. \quad (12)$$

Similarly, the equation for CLA can be written as:

$$s_i = p_i \oplus [c_0 \bigtriangledown (g_1, p_1) \bigtriangledown \cdots \bigtriangledown (g_{i-2}, p_{i-2}) \bigtriangledown (g_{i-1}, p_{i-1})]. \quad (13)$$

To obtain the equation for ELMA, we have to use the following equation found in [5]:

$$c_i = g_i + p_i \cdot c_{i-1} = g_i \oplus (p_i \cdot c_{i-1}) = c_{i-1} \bigtriangleup (g_i, p_i). \qquad (14)$$

Note that (14) holds only if the "g_i" and "p_i" are defined as $a_i \cdot b_i$ and $a_i \oplus b_i$, respectively. With the aid of (14), (13) can be rewritten as

$$s_i = [c_0 \bigtriangledown (g_1, p_1) \bigtriangledown \cdots \bigtriangledown (g_{i-2}, p_{i-2})] \bigtriangleup ((p_i \oplus g_{i-1}), p_{i-1}). \quad (15)$$

This is the equation for ELMA. To remove or to change the position of the square brackets in (15), the following rule has to be applied.

$$\begin{aligned} &[c_{i-k} \bigtriangledown \cdots \bigtriangledown (g_{i-1}, p_{i-1})] \bigtriangleup (a,b) \\ =&[c_{i-k} \bigtriangledown \cdots \bigtriangledown (g_{i-2}, p_{i-2})] \bigtriangleup (g_{i-1}, p_{i-1}) \bigtriangleup (a,b) \\ =&c_{i-k} \bigtriangleup (g_{i-k+1}, p_{i-k+1}) \bigtriangleup \cdots \bigtriangleup (g_{i-1}, p_{i-1}) \bigtriangleup (a,b). \end{aligned} \qquad (16)$$

To further simplify the equation, a pair of subscript indices is defined to denote the r, g, and p terms generated from primitive r, g, and p terms. For example, the (r, g) pair is defined as:

$$(r_{j,k}, g_{j,k}) = \begin{cases} (r_j, g_j) & \text{if k = j}, \\ (r_j, g_j) \otimes (r_k, g_k) & \text{if k = j + 1}, \\ (r_{j,m-1}, g_{j,m-1}) \otimes (r_{m,k}, g_{m,k}) & \text{if k} \geq \text{m} > \text{j}, \end{cases} \qquad (17)$$

where the indices, j, m, and k, are all nonnegative integers. The defined operators and notations can express the adder algorithms concisely. For example, the equations for the s_7 of the algorithms can be expressed as

$$\text{CLA\_I: } s_7 = p_7 \oplus [c_1 \bigtriangledown (g_{2,3}, p_{2,3}) \bigtriangledown (g_{4,5}, p_{4,5}) \bigtriangledown (g_6, p_6)] \qquad (18a)$$

$$\text{CLA\_II: } s_7 = p_7 \oplus [c_1 \bigtriangledown (g_{2,3}, r_{2,3}) \bigtriangledown (g_{4,5}, r_{4,5}) \bigtriangledown (g_6, r_6)] \qquad (18b)$$

$$\text{CCA: } s_7 = p_7 \oplus [c_1 \otimes (r_{2,3}, g_{2,3}) \otimes (r_{4,5}, g_{4,5}) \otimes (r_6, g_6)] \qquad (18c)$$

$$\text{ELMA: } s_7 = c_1 \bigtriangleup (g_{2,3}, p_{2,3}) \bigtriangleup (g_{4,5}, p_{4,5}) \bigtriangleup ((p_7 \oplus g_6), p_6) \qquad (18d)$$

$$\text{CSMA: } s_7 = c_1 \otimes (r_{2,3}, g_{2,3}) \otimes (r_{4,5}, g_{4,5}) \otimes ((p_7 \oplus r_6), (p_7 \oplus g_6)) \qquad (18e)$$

Equations (9) and (10) show the similarity of carry-generation between conditional-sum-based and carry-lookahead-based algorithms. By using different operators and exchanging the order of combining the p_i term for s_i as shown in (18), these popular addition algorithms can be derived. Taking CLA_I as an example, the derivation of (18a) is as follows:

$$\begin{aligned} s_7 &= p_7 \oplus c_6 \\ &= p_7 \oplus [c_0 \bigtriangledown (g_1, p_1) \bigtriangledown (g_2, p_2) \cdots \bigtriangledown (g_6, p_6)], \end{aligned}$$

according to (10). By combining two adjacent terms we can obtain

$$s_7 = p_7 \oplus [c_1 \bigtriangledown (g_{2,3}, p_{2,3}) \bigtriangledown (g_{4,5}, p_{4,5}) \bigtriangledown (g_6, p_6)].$$

The derivation for the other four equations is quite similar, and detailed derivation steps for these algorithms can be found in [13].

Equation (18) demonstrates the relationships among the five adder schemes. CLA has the same configuration as CCA and ELMA has the same configuration as CSMA. By comparing (11), (12), (13), and (15), we can see that, for CCA and CLA, the "p_i" is combined with "c_{i-1}" to generate sum at the last step; for CSMA and ELMA, "p_i" is combined with the other terms at the beginning of addition. These results are summarized in Table 1.

3.2 Performance Analysis

According to these equations, we depict the generation of the last sum bit for four of the adder schemes in Fig. 1. The dashed lines indicate the estimated critical paths. The real critical path depends on the real implementation. Note that

TABLE 1
Summary of the Four Addition Algorithms

Carry generation rule	Order of combining p_i	
	p_i first	p_i last
Conditional-sum	CSMA	CCA
Carry-lookahead	ELMA	CLA

the operators used at the last step of generating carry signal are slightly faster than the others because they operate on three operands only.

For CLA and CCA adder types, the p_i is always merged at the last step. When the wordlength, L, is even, the (g_{L-2}, r_{L-2}) term can only be combined with the other terms at the second level, which causes inefficiency. In contrast, for CSMA and the ELMA, the (g_{L-2}, r_{L-2}) term can always be combined with p_i at the beginning. Hence, if L does not equal (2^n+1), CSMA and ELMA can save one operator on the critical path compared to CLA or CCA. Based on the equations and the graphs, the latency required for these adder types is derived and summarized in Table 2. The **r** denotes the fan-in which refers to the number of bits that can be processed in one complex logic gate. Table 3 lists the properties of the related logic gates and the three ternary operators for the same driving strength with standard output loading according to the $0.35\mu m$ cell library [15]. When the subscript of the timing parameter begins with a lowercase "c," it means that it is the critical path for a logic gate or an operator. Note that $t_{cAO} < t_{MUX} < t_{cMUX} < t_{ELM} < t_{cELM}$.

According to the equations, CSMA always has the fewest operators on the critical path. However, when L is greater than eight, the number of ternary operators will be greater than three on the critical path. The CLA_II will be faster than CSMA because $5 \cdot t_{cAO} < 4 \cdot t_{cMUX}$ according to Table 3. Therefore, we conclude that the CLA_II adder will be the fastest one when $L > 8$ according to our cell library and the efficiency of the $\bigtriangledown$ operator.

To summarize, with the aid of the defined operators, notations, and the revealed properties, we can compare and evaluate the addition algorithms at algorithm level. For example, (18d) and (18e) show that ELMA and CSMA can have exactly the same structure for sum-generation and they can also use same structure to generate carry signals according to (9) and (10). Therefore, a designer can easily choose the right addition algorithm according to the cost and the performance of the $\bigtriangleup$, the $\bigtriangledown$, and the $\otimes$ operators. Furthermore, the designer may predict the cost of routing according to the properties of the operators. For example, when the two addition algorithms have the same structure, the $\otimes$ operator will consume more routing resources than either $\bigtriangleup$ or $\bigtriangledown$ operator according to the definitions given in (5)-(8).

4 Proposed Algorithms

We have demonstrated that the traditional algorithms can be expressed using the defined operators and notations. In

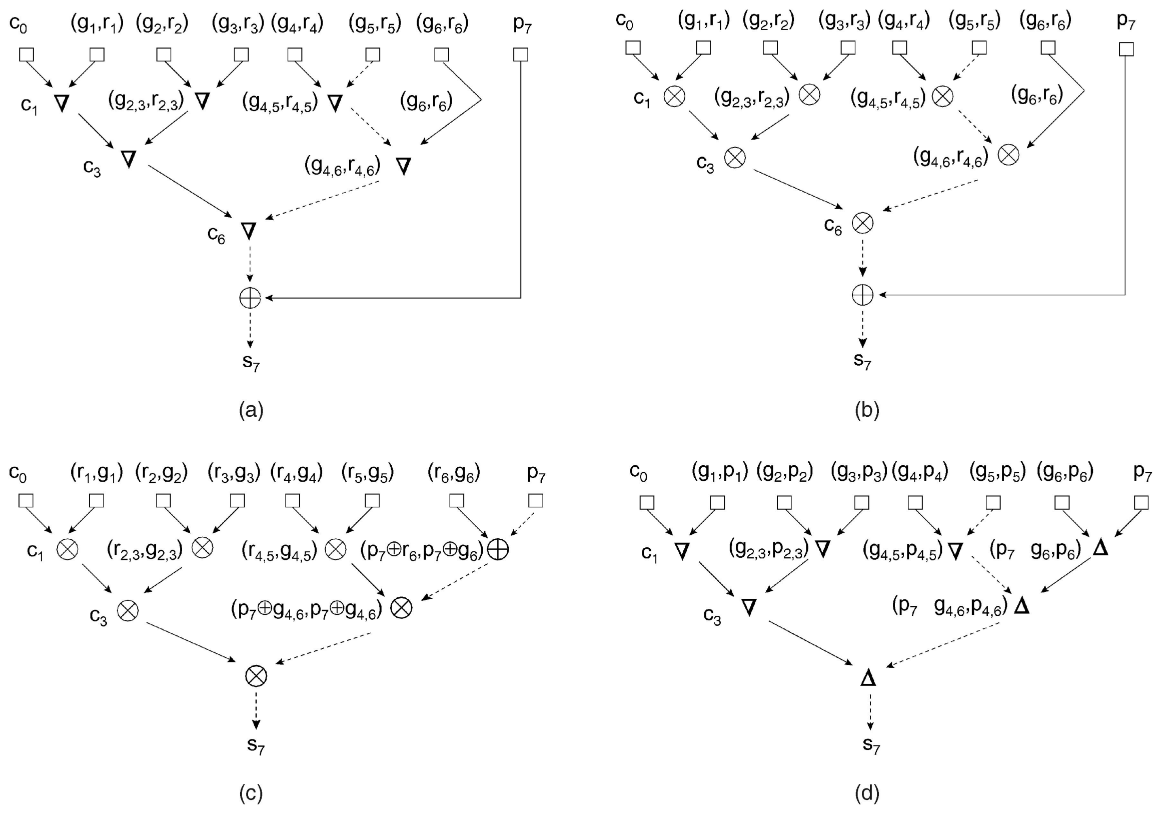

Fig. 1. The generation of sum signals. The dashed-line shows the critical path.

TABLE 2
Latency of Each Adder Type for Wordlength L with Fan-In r

Adder Type	Latency	Example I, L=16, r=2	Example II, L=17, r=2
CLA_I	$t_P + \lceil log_r(L-1) \rceil \cdot t_{cAO} + t_P$	$t_P + 4 \cdot t_{cAO} + t_P$	$t_P + 4 \cdot t_{cAO} + t_P$
CLA_II	$t_R + \lceil log_r(L-1) \rceil \cdot t_{cAO} + t_P$	$t_R + 4 \cdot t_{cAO} + t_P$	$t_R + 4 \cdot t_{cAO} + t_P$
CCA	$t_R + \lceil log_r(L-1) \rceil \cdot t_{cMUX} + t_P$	$t_R + 4 \cdot t_{cMUX} + t_P$	$t_R + 4 \cdot t_{cMUX} + t_P$
ELMA	$2t_P + t_R + (\lceil log_r L \rceil - 1) \cdot t_P$	$2t_P + t_R + 3 \cdot t_P$	$2t_P + t_R + 4 \cdot t_P$
CSMA	$2t_P + (\lceil log_r L \rceil - 1) \cdot t_{cMUX}$	$2t_P + 3 \cdot t_{cMUX}$	$2t_P + 4 \cdot t_{cMUX}$

this section, we want to move one step further to show that the unequal delay profile problem can be solved through the manipulation of the operators and the equations.

This section describes two optimization algorithms based on the developed equations and operators in Section 3. We will demonstrate that lower area-time complexity than the previously mentioned five adder algorithms can be achieved without performance loss. More importantly, the two algorithms are both capable of generating new adder structures for unequal delay profile.

To facilitate the discussion, we define *Delay Profile* as follows:

Delay Profile (DP). *Given two L-bit operands, A and B, the input delay profile, DP, is an array consisting of L elements in total. The value of each element is given by*

$$DP[i] = max(t(a_i), t(b_i)) \quad for \quad i = 0, \cdots, L-1.$$

That is, the value of the i-th element is determined by the latest arrived signal of a_i and b_i. The function max $(value1, value2)$ returns the larger one of the two values and the $t(\cdot)$ function returns the arrival timing of the given argument. "Equal delay profile" refers to the case when all bits of the two operands arrive simultaneously. In contrast, "unequal delay profile" refers to the condition when any bit of the two operands arrives earlier or later compared with the other bits. The inputs may be either primitive terms, i.e., r, g, and p terms, or operands in two's complement format, e.g., a_i and b_i. The time unit is normalized with respect to the latency of a ternary operator.

TABLE 3
Latency of the Operators

Operator	function	parameter	latency(ns)
$+$	$z = a + b$	t_R	0.378
$\cdot$	$z = a \cdot b$	t_G	0.303
$\oplus$	$z = a \oplus b$	t_P	0.498
$\bigtriangledown$	$z = c \bigtriangledown (b, a)$	t_{cAO}	0.367
(r=2)		t_{AO}	0.296
$\bigtriangledown^2$	$z = \overline{f + d \cdot e + a \cdot b \cdot c}$	t_{cAO^2}	0.533
(r=3)		t_{AO^2}	0.477
$\otimes$	$z = b \cdot a + \overline{b} \cdot c$	t_{cMUX}	0.468
(r=2)		t_{MUX}	0.403
$\triangle$	$z = c \triangle (b, a)$	t_{cELM}	0.17+0.498(=$t_G + t_P$)
(r=2)		$t_{ELM} = t_P$	0.498

4.1 Dual-Bit Forward Prediction Algorithm

The dual-bit forward prediction (DFP) algorithm optimizes the adder starting from the LSB and considers at most two bits in each iteration. At each iteration, it examines the arrival timings of the next two bit positions to determine whether it is advantageous or not to combine the next two bits before combining the signal at current bit position with the next bit. We will use $\otimes$ operator to illustrate the algorithm. However, any one of the three operators can be used for this algorithm. We assume that both the latency of a two-input AND gate and a two-input OR gate are 0.5 unit delays and the latency of a two-input XOR gate is one unit delay.

To illustrate the algorithm, let us consider the following three cases. Similar examples can be found in [11]. The first one is shown in Fig. 2. The intermediate r and g terms are generated according to the equations mentioned in Section 2. From the figure, we can see that when the incoming carry signal arrives almost concurrently as the next two bits, it will not be beneficial to use the ternary operator. Instead, a full-adder will be the best choice.

The next case is shown in Fig. 3. Because the next four bits arrive two time units earlier than the incoming carry signal, they can be combined using (8). Only two time units are required to complete the eight-bit addition.

The last case is shown in Fig. 4. In this case, the next two bits arrive earlier than the incoming carry by two and half time units. After ($r_{26,27}$, $g_{26,27}$) has been generated, there is still one unit delay between the generated ($r_{26,27}$, $g_{26,27}$) terms and the incoming carry. Therefore, ($r_{26,27}$, $g_{26,27}$) combines with ($r_{28,31}$, $g_{28,31}$), generated by the next four bits, to generate ($r_{26,31}$, $g_{26,31}$). Finally, ($r_{26,31}$, $g_{26,31}$) combines with the incoming carry to generate c_{31}. Again, an 8-bit block can be computed within two time units.

The three figures demonstrate that the latency of addition can be reduced by arranging the order of bit-level operations carefully. The ternary operator can be replaced by the other ternary operators, as we have discussed in Section 3. That is, we can apply similar scheduling algorithms to optimize the five discussed addition algorithms. The MLCSMA algorithm presented in [11] is just a special case which employs $\otimes$ operator. Therefore, the DFP algorithm can also employ the hybrid adder scheme, which uses CCA and CSMA at the same time to save area without

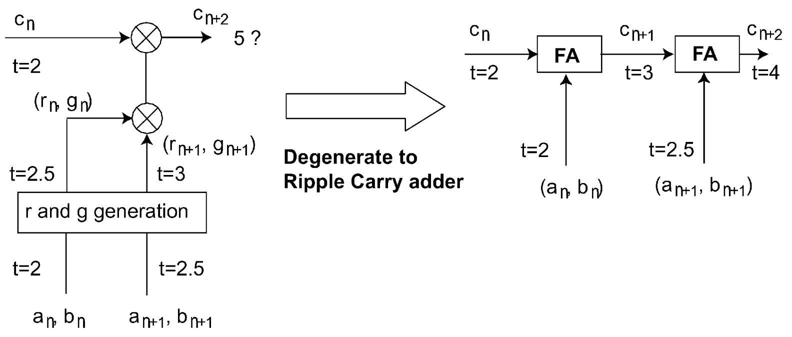

Fig. 2. c_{n+2} will be generated at t = 5 if ternary operator is used; full adders can be used instead to generate c_{n+2} at t = 4.

causing performance degradation as discussed in [11]. To summarize, the DFP algorithm employs the associative property of the ternary operator and examines two bits at a time to determine the order of bit-level computation.

4.2 Generalized Earliest-First Algorithm

Although the DFP algorithm improves the adder performance in terms of area and speed, it does not guarantee the optimality of the performance in anyway. In addition, it only considers the next two bits at each iteration. However, sometimes it may be beneficial to combine the bits other than the next two bits first to produce better DP in the next iteration. In this section, we present a novel algorithm which considers the order of combination at all possible bit positions in each iteration.

The basic idea of the generalized earliest-first (GEF) algorithm are based on two facts. At first, when some bits arrive earlier than the rest of the other bits, we can combine them first to improve the performance and the generated intermediate terms, e.g., (r, g), can still be treated as primitive terms. Second, the computation does not have to start from LSB as long as it does not violate associative and noncommutative properties.

The new algorithm consists of five steps for an L-bit addition:

1. **Setup:** Initialize an array with L elements, P_list. For $0 \leq i \leq L-1$, let P_list[i] = DP[i]. Each entry in P_list consists of the arrival timing and the bit position. Initialize T_list with zero element and the structure of each entry is the same as the P_list.
2. **Generate new T_list:** Sort P_list in ascending order according to the arrival timing. Move the elements that are equivalent to P_list[0] from P_list to T_list. Sort T_list in ascending order according to the bit position. Retain the bit position information.
3. **Combine adjacent bits:** In T_list, if there are signals adjacent to each other, combine them with ternary operator. Insert the generated new terms with bit positions and timings back into P_list.
4. Repeat. Repeat Steps 2 and 3 until only one element left in P_list.
5. Generate sum signals and finish

The algorithm guarantees the optimality in each step and, hence, the generated adder structure is optimized in terms of speed. Note that two terms are considered as adjacent to each other when there is no other term between them. They can have inconsecutive bit positions. When there are more than two adjacent terms in T_list in Step 2, fan-in greater than two is allowed as long as the generated structure is optimal based on the available technology. For example, if there are three adjacent bits in the T_list, we can cascade two ternary operators to compute the 3-bit block, i.e., r = 3. An r = 4 example can be found in [3]. However, greater fan-in does not always result in better performance. Also, if the LSB of the adjacent bits is a carry signal, it is possible to use ripple carry adder instead when the

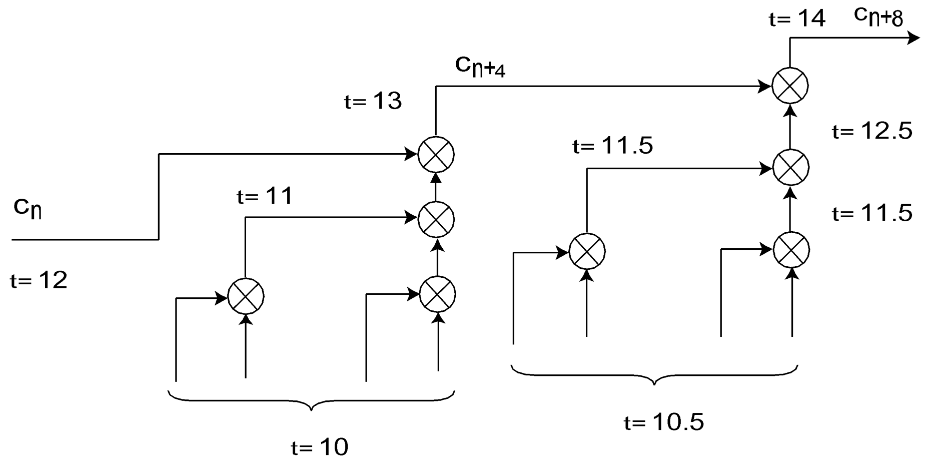

Fig. 3. c_{n+8} is generated at t = 14. An eight-bit addition is completed within two time units.

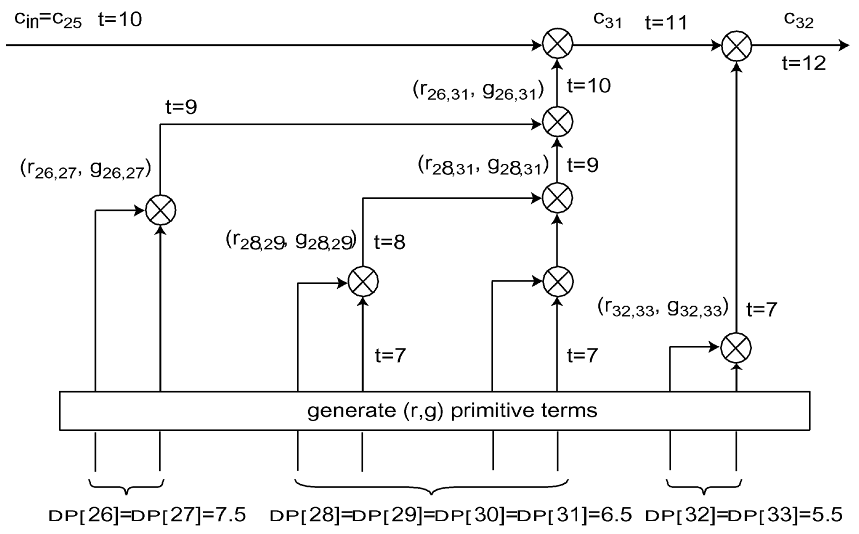
$c_{in}=c_{25}$ t=10
c_{31} t=11
c_{32}
t=12
$(r_{26,31}, g_{26,31})$ t=10
t=9
$(r_{26,27}, g_{26,27})$
$(r_{28,31}, g_{28,31})$ t=9
$(r_{28,29}, g_{28,29})$ t=8
$(r_{32,33}, g_{32,33})$ t=7
t=7
t=7
generate (r,g) primitive terms
DP[26]=DP[27]=7.5
DP[28]=DP[29]=DP[30]=DP[31]=6.5
DP[32]=DP[33]=5.5

Fig. 4. Determine whether the next two bits should be combined prior to combining them with incoming carry. Apply the rule recursively to determine the order of combination.

conditions of using ripple carry adder are met as discussed in Section 4.1.

Besides, when the r has been determined and the number of candidates to be merged is greater than r but is not a multiple of r, which of them should be merged first to generate better DP? This happens only when no two adjacent terms of the candidates arrive earlier than the others. Otherwise, the two adjacent terms must have been merged in earlier iterations. Therefore, we can just treat these adjacent signals as normal equal DP and use traditional algorithms to construct this part of the adder.

A simple delay profile is used to demonstrate the advantages of the GEF algorithm over the DFP algorithm. Assume that the delay profile DP = $\{0, 1, 2, 2, 3, 3, 4, 5, 4, 3, 2, 1\}$ and fan-in is always two. Note that the inputs are already primitive terms and the first term is the incoming carry signal. Therefore, all we have to do is to find the optimal order of computation using ternary operators.

If traditional algorithm is applied without considering the unequal delay profile, the last carry signal will be available at t = 9 ($= 5 + \lceil log_2 12 \rceil$). For the DFP algorithm, Table 4 shows the status in each iteration. The two selected terms to be combined in current iteration are in bold-face type. The term generated in the previous iteration is underlined. Eleven iterations are required and the final carry is generated at t = 8.

Similarly, Table 5 shows the status of the GEF algorithm in each iteration. We repeat Step 2 and Step 3 until T_list has

TABLE 4
Example of Dual-Bit Prediction Algorithm

	Bit Position											
Iteration	0	1	2	3	4	5	6	7	8	9	10	11
0	**0**	**1**	2	2	3	3	4	5	4	3	2	1
1		$\underline{\mathbf{2}}$	**2**	2	3	3	4	5	4	3	2	1
2			$\underline{\mathbf{3}}$	**2**	3	3	4	5	4	3	2	1
3				$\underline{4}$	**3**	**3**	4	5	4	3	2	1
4				**4**		$\underline{\mathbf{4}}$	4	5	4	3	2	1
5						$\underline{\mathbf{5}}$	**4**	5	4	3	2	1
6							$\underline{6}$	**5**	**4**	3	2	1
7							**6**		$\underline{\mathbf{6}}$	3	2	1
8									$\underline{7}$	**3**	**2**	1
9									7		$\underline{\mathbf{4}}$	**1**
10									**7**			$\underline{\mathbf{5}}$
11												$\underline{8}$

TABLE 5
Example of GEF Algorithm

		Bit Position											
Iteration	List	0	1	2	3	4	5	6	7	8	9	10	11
0	P			2	2	3	3	4	5	4	3	2	
	T	**0**	**1**										1
1	P					3	3	4	5	4	3		
	T		$\underline{\mathbf{2}}$	**2**	2							**2**	**1**
2	P							4	5	4			
	T			$\underline{\mathbf{3}}$	**2**	**3**	**3**				**3**		$\underline{\mathbf{3}}$
3	P								5				
	T				$\underline{\mathbf{4}}$		$\underline{\mathbf{4}}$	4		**4**			$\underline{\mathbf{4}}$
4	P												
	T						$\underline{\mathbf{5}}$	**4**	**5**				$\underline{\mathbf{5}}$
5	P												
	T							$\underline{\mathbf{6}}$					$\underline{\mathbf{6}}$
6	P												$\underline{7}$
	T												

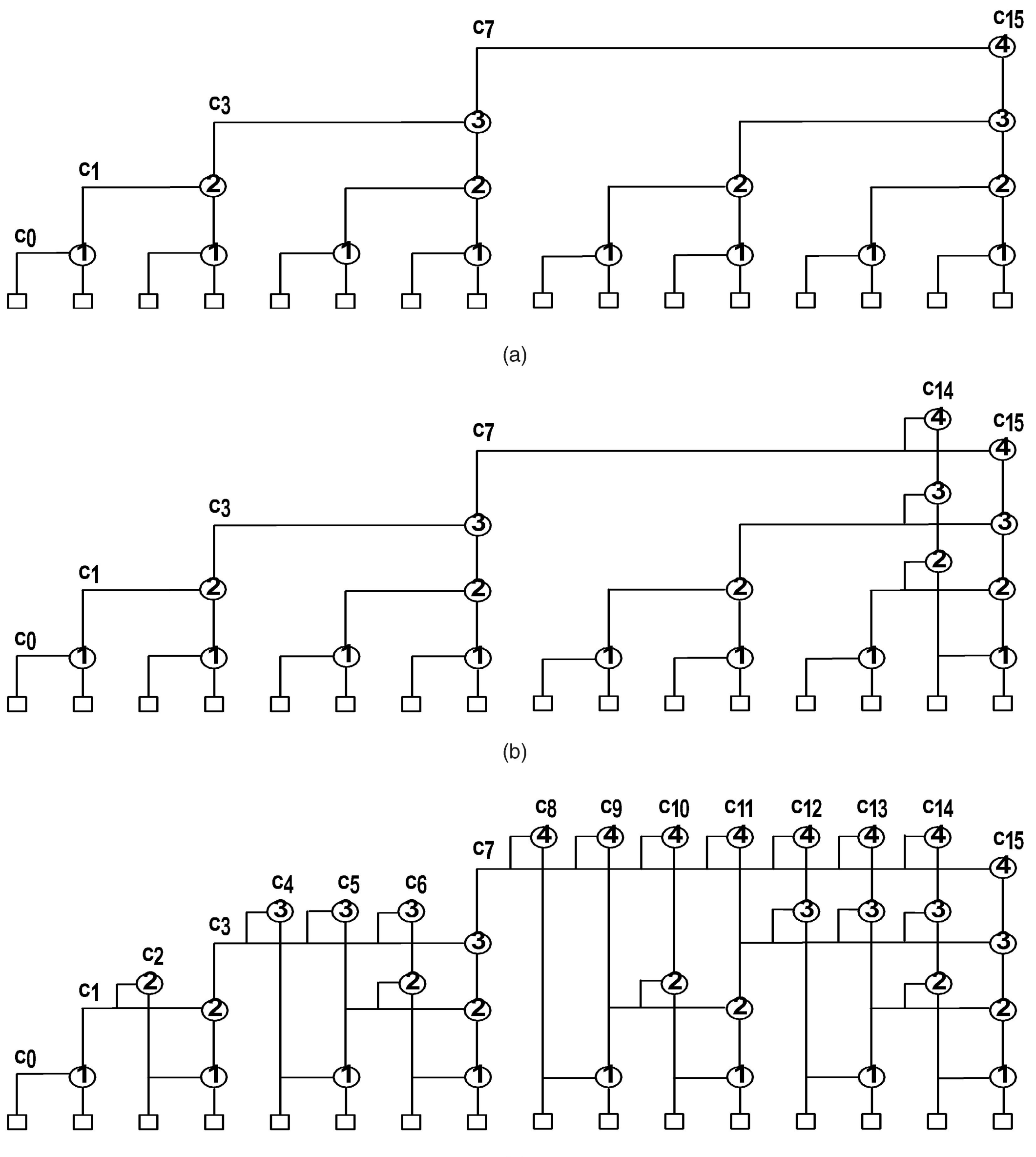

Fig. 5. The generation of carry signals for the GEF-based adder. Each circle represents a ternary operator and the number inside the circle shows the timing normalized with respect to the operator.

enough adjacent bits for computation. For example, in iteration 0, only DP[0] has been moved from P_list to T_list after the first pass. However, since there are no adjacent terms for computation, DP[0] is kept in the T_list. After the second pass, DP[1] and DP[11] are moved into T_list. In each iteration, we store the generated terms back into P_list directly without further combining the generated terms using another ternary operator, because the timing of the generated terms may be equal to that of the terms in the P_list.

The DFP algorithm generates the last carry signal at t = 8 while the GEF algorithm is at t = 7. The DFP algorithm is unable to combine the early arrived terms at higher bit positions to reduce the latency. If the signals from bit position 8 to 11 were combined first, the DFP algorithm would have been able to generate the last carry signal at t = 7. It is interesting to note that, if the slope of DP is negative one, we can perform the addition backward from MSB to LSB and the structure of the generated adder will be similar to a ripple-carry adder.

The associativity of the ternary operators enables us to perform addition from both directions and at any bit position. When dealing with unequal DP, GEF algorithm can combine all of the early arrived signals using any one of the ternary operators. The generated profile can be treated as an equal DP case. When fan-in greater than two is allowed, the overall performance may be further improved for both algorithms. However, the GEF algorithm will still outperform the DFP algorithm due to the same reasons. Only the generation of carry signal has been discussed because the generation of sum signals becomes trivial when all carry signals have been generated, as we will show in the next section.

4.3 Hardware Complexity Analysis

We have shown that the proposed algorithm can reduce the critical path for unequal DP. In this section, we present a simple methodology to generate CPA based on the GEF algorithm. The hardware complexity is then analyzed according to the generated adder.

The following steps are used to generate the GEF-based adder.

1. Choose the best ternary operator according to the available process and cell library.
2. Generate the MSB carry signal using the GEF algorithm. Store the information of the intermediate terms and carry signals generated from any ternary operator.
3. Generate the other carry signals that have not been generated from MSB to LSB. Use previously generated intermediate terms to save area and also store the information of the new intermediate terms.
4. Generate sum signals if the current adder architecture is based on CCA or CLA rules.

We use a 17-bit adder based on CLA or CCA to illustrate the generation of carry signals for equal DP. The generation of sum signals is trivial when all carry signals are ready.

Fig. 5a shows the generation of the MSB carry; c_{15}. c_0, c_1, c_3, and c_7 are also generated as the inputs show equal DP. In Fig. 5b, the second MSB carry signal is generated and three additional operators are used. The common resource sharing can be achieved easily by decomposing the equation into intermediate terms according to the subscript notation. For example, if "$\bigtriangledown$" operator is chosen, the decomposition of c_{14} is as follows:

$$c_{14} = c_{13} \bigtriangledown (g_{14}, r_{14}) = c_7 \bigtriangledown (g_{8,14}, r_{8,14}).$$

c_7 is the nearest generated carry from lower bit positions. The second term can be further decomposed into $(g_{8,11}, r_{8,11})$, $(g_{12,13}, r_{12,13})$, and (g_{14}, r_{14}) because these terms have been generated for the MSB.

$$\begin{aligned}&c_7 \bigtriangledown (g_{8,14}, r_{8,14}) = \\ &c_7 \bigtriangledown (g_{8,11}, r_{8,11}) \bigtriangledown (g_{12,13}, r_{12,13}) \bigtriangledown (g_{14}, r_{14}).\end{aligned}$$

After the carry has been represented using the previously generated terms, the GEF algorithm is used again to schedule the order of operation. The structure after all carry signals have been generated is shown in Fig. 5c.

The number of ternary operators required to construct an L + 1 bit adder for equal DP can be deduced from the figures.

$$N_{top} = \frac{L}{r} \cdot log_r L, \tag{19}$$

where $L = 2^n$, n is a positive integer, and r is the fan-in of ternary operator. Among the $\frac{L}{r} log_r L$ operators, L - 1 are used to generate carry signals at the last step. Take L = 2 as an example. Only one ternary operator is used as $\frac{2}{2} log_2 2 = 1$. c_0 and c_1 are generated and the last sum signal can be generated as $s_2 = c_1 \otimes p_2$, according to (1). Although the above analysis holds for CLA and CCA only, the case for CSMA and ELMA is quite similar.

The above analysis shows the equal DP case for the GEF algorithm and the generated adder shows the same complexity as a traditional adder. In the unequal DP case, the number of ternary operators can be reduced because the required intermediate terms can be generated sequentially and can be shared to reduce the cost. The routing may become less regular, but the required routing resource will be less than that of traditional adders.

5 Conclusion

In this paper, we have explored the design space for fast addition at algorithm level. We have developed a set of operators and notations to formulate the five popular addition algorithms in Section 3. The derived equations are useful when comparing and evaluating performance and hardware cost. By eliminating the effects of using different connection or topology, we can evaluate each addition algorithm at algorithm level. Hardware cost and timing performance can be estimated by counting the number of primitive terms and the ternary operators used as demonstrated in Section 3 and Section 4.

Unlike the traditional addition algorithms assuming that all the inputs arrive simultaneously, the two proposed algorithms are capable of generating faster structure according to the incoming delay profile. The first algorithm is generalized from MLCSMA algorithm. It examines two bits at a time and constructs the adder from LSB. Consequently, it cannot fully exploit the unequal timing profile to reduce the latency of addition. The GEF algorithm is developed to solve this problem by considering global timing properties and multiple bits in each iteration and optimal performance can be achieved. The proposed GEF algorithm can be implemented by using any one of the ternary operators and is suitable for both equal and unequal DP. The example demonstrated in Section 4.2 shows 22.2 percent performance improvement when the GEF algorithm is applied.

Acknowledgments

The authors would like to acknowledge the valuable suggestions from the reviewers and the associative editor. The work was supported by the National Science Council, Taiwan Republic of China under Grant NSC91-2218-E009-011.

References

[1] J. Sklansky, "Conditional Sum Addition Logic," *IRE Trans. Electronic Computers,* vol. 9, no. 2 pp. 226-231, June 1960.
[2] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design* p. 81. John Wiley & Sons, 1976.
[3] N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective,* second ed. pp. 526-536, Addison Wesley, 1993.
[4] K.-H Cheng, S.-M. Chiang, and S.-W. Cheng, "The Improvement of Conditional Sum Adder for Low Power Applications," *Proc. 11th Ann. IEEE Int'l Application Specific Integrated Circuits Conf.,* pp. 131-134, 1998.
[5] P. Kelliher, R.M. Owens, M.J. Irwin, and T.-T. Hwang, "ELM—A Fast Addition Algorithm Discovered by a Program," *IEEE Trans. Computers,* vol. 41, no. 9, Sept. 1992.

[6] H. Lindkvist and P. Andersson, "Techniques for Fast CMOS-Based Conditional Sum Adders," *Proc. IEEE Int'l Conf. Computer Design: VLSI in Computers and Processors,* pp. 626-635, Oct. 1994.

[7] H. Kunz and R. Zimmermann, "High-Performance Adder Circuit Generators in Parameterized Structural VHDL," Technical Report No. 96/7, Integrated Systems Laboratory, ETH Zurich Aug. 1996.

[8] M.D. Ercegovac et al., "Fast Multiplication without Carry-Propagate Addition", *IEEE Trans. Computers,* vol. 39, no. 11, Nov. 1990.

[9] R.K. Kolagotla et al., "VLSI Implementation of a 200-Mhz 16$\times$16 Left-to-Right Carry-Free Multiplier in 0.35μm CMOS Technology for Next-Generation DSPs", *Proc. IEEE 1997 Custom Integrated Circuits Conf.,* pp. 469-472, 1997.

[10] P.F. Stelling and V.G. Oklobdzija, "Optimal Designs for Multipliers and Multiply-Accumulators," *Proc. 15th IMACS World Congress on Scientific Computation, Modeling, and Applied Mathematics,* vol. 4, pp. 739-744, Aug. 1997.

[11] W.-C. Yeh and C.-W. Jen, "A High-Speed Booth Encoded Parallel Multiplier Design," *IEEE Trans. Computers,* vol. 49, no. 7, pp. 692-701, July 2000.

[12] C.N. Nagendra, M.J. Irwin, and R.M. Owens, "Area-Time-Power Tradeoffs in Parallel Adders," *IEEE Trans. Circuits and Systems XII: Analog and Digital Signal Processing,* vol. 43, no. 10, Oct. 1996.

[13] W.-C. Yeh and C.-W. Jen, "On the Study of Logarithmic Time Parallel Adders," *Proc. IEEE Workshop Signal Processing Systems,* pp. 459-466, 2000.

[14] R.P. Brent and H.-T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers,* vol. 31, no. 3, Mar. 1982.

[15] Passport 0.35 micron, 3.3 volt, Optimum Silicon SC Library, CB35OS142, Avant! Corp., Mar. 1998.

Wen-Chang Yeh received the BS degree and PhD degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, Republic of China, in 1997 and 2001, respectively. He is now working for ZyDAS Technology Corporation, Hsinchu, Taiwan, Republic of China His current research interests include computer arithmetic, digital signal processing for communication systems, computer architecture, and system level design.

Chein-Wei Jen received the BS degree from National Chiao Tung University, Hsinchu, Taiwan, Republic of China, in 1970, the MS degree from Stanford University, Stanford, California, in 1977, and the PhD degree from National Chiao Tung University in 1983. He is currently with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, as a professor. During 1985-1986, he was with the University of Southern California, Los Angeles, as a visiting researcher. His current research interests include VLSI design, digital signal processing, processor architecture, and design automation. He holds six patents and has published more than 40 journal papers and 90 conference papers in these areas. He is a mameber of the IEEE.