Delay-Insensitive Carry-Lookahead Adders *

Fu-Chiung Cheng Stephen H. Unger Michael Theobald Wen-Chung Cho

Department of Computer Science
Columbia University
New York, NY 10027

Abstract

Integer addition is one of the most important operations in digital computer systems because the performance of processors is significantly influenced by the speed of their adders. This paper proposes a delay-insensitive, carry-lookahead adder in which the logic complexity is a linear function of n, the number of inputs, and the average computation time is proportional to the logarithm of the logarithm of n. We also show an economic implementation of this adder in CMOS technology.

1. Introduction

Integer addition is one of the most important operations in digital computer systems. In addition to explicit arithmetic (such as addition, subtraction, multiplication and division) performed in a program, additions are performed to increment the program counter and to calculate the effective address [10]. Statistics presented in [10, 7] show that in a prototypical RISC machine (DLX) 72% of the instructions perform additions (or subtractions) in the data path. Thus, the performance of processors is significantly influenced by the speed of their adders.

Circuits may be classified as synchronous and asynchronous circuits. Synchronous circuits have a clock to synchronize the operations of subsystems while asynchronous circuits do not. Subsystems in asynchronous circuits usually need their *start* and *completion* mechanisms to synchronize one another. One advantage of using asynchronous circuits is that these circuits operate at average rates, while synchronous circuits are subjected to operate at the worst rates. A good example for this is that n-bit ripple-carry adders (RCA), which are synchronous, have worst case computation time $\Theta(n)$, whereas n-bit carry-completion sensing adders (CCSA), which are asynchronous, have average computation time $\Theta(\log n)$ [8, 11, 3].

Delay-insensitive (DI) circuits [5] are a subclass of asynchronous circuits. The defining property of pure DI circuits is that their correctness is insensitive to delays in both gate elements and connection wires. The class of pure DI circuits is quite limited [13]. However, extending pure DI circuits with isochronic forks is sufficient to construct any circuit of interest. (Such circuits are sometimes called quasi-DI.) For this paper we assume isochronic forks.

The CCSA is not a DI circuit. It must meet the bundling constraint [19]. One way to meet this constraint is to put a delay in the control path [4]. It is impossible to meet the bundling constraint if there is no way to control delay in control and data paths. Thus, the design of DI circuits is very interesting.

DI circuits may be implemented by using dual-rail signaling [17, 21]: For each data bit, two separate signals are used; $A_i^0 A_i^1 = 01(10)$ denotes that the data bit is one (zero). To implement DI adders, we may use dual-rail signaling for input bits, sum bits as well as carry bits.

Let $A_{n-1}A_{n-2}...A_0$ and $B_{n-1}B_{n-2}...B_0$ be two n-bit binary numbers with a sum of $S_{n-1}S_{n-2}...S_0$ and with a sequence of carry bits, $C_nC_{n-1}...C_0$. (The least significant bits are A_0 and B_0.) The DI version of the RCA (DIRCA) computes the $S_i s$ this way:

$$C_{i+1}^0 = A_i^0 B_i^0 + (A_i^0 + B_i^0) C_i^0 \quad (1)$$

$$C_{i+1}^1 = A_i^1 B_i^1 + (A_i^1 + B_i^1) C_i^1 \quad (2)$$

$$S_i^0 = A_i^0 B_i^0 C_i^0 + A_i^0 B_i^1 C_i^1 + A_i^1 B_i^0 C_i^1 + A_i^1 B_i^1 C_i^0 \quad (3)$$

$$S_i^1 = A_i^1 B_i^1 C_i^1 + A_i^1 B_i^0 C_i^0 + A_i^0 B_i^1 C_i^0 + A_i^0 B_i^0 C_i^1 \quad (4)$$

where $i = 0, 1, ..., n-1$. An n-bit DIRCA is shown in Figure 1.

Martin [14] proposed a very good design of the DIRCA adder by using CMOS technology. The transistor count per DIRCA cell is 42. Compared to the synchronous RCA cell which needs 40 transistors, it is clear that the asynchronous DIRCA adder is hardly larger than the synchronous one, in spite of the use of dual-rail signals. The logic complexity

*The research on which this paper was based was supported in part by a grant from the AT&T Foundation.

0-8186-7755-4/96 $05.00 © 1996 IEEE

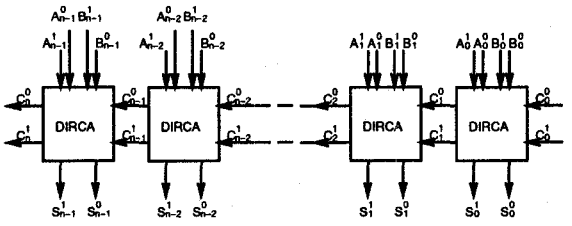

Figure 1. DI Ripple-Carry Adder

of DIRCA is a linear function of n, $\Theta(n)$, and the average computation time is proportional to the logarithm of n, $\Theta(\log n)$.

This paper proposes a delay-insensitive, carry-lookahead adder in which the logic complexity is a linear function of n, $\Theta(n)$, and the average computation time is proportional to the logarithm of the logarithm of n, $\Theta(\log \log n)$.

2. Carry-Lookahead Adders

The carry-lookahead scheme computes the $S_i s$ as follows:

$$p_i = A_i + B_i \quad \text{(carry-propagate)} \tag{5}$$
$$g_i = A_i B_i \quad \text{(carry-generate)} \tag{6}$$
$$S_i = A_i B_i C_i + A_i B_i' C_i' + A_i' B_i C_i' + A_i' B_i' C_i \tag{7}$$
$$C_{i+1} = g_i + p_i C_i \tag{8}$$

where $i = 0, 1, ..., n-1$. Equation (8) can be further expanded into

$$C_{i+1} = g_i + p_i g_{i-1} + ... + p_i p_{i-1} ... p_1 g_0 + p_i p_{i-1} ... p_0 C_0 \tag{9}$$

For large i, it is impractical to build a full carry-lookahead adder because of the practical limitations on fan-in and fan-out, irregular structure, and many long wires [16, 10]. However, the carry-lookahead scheme can be built in the form of a tree-like circuit, which has a simple, regular structure [6, 20, 2, 10], by reformulating Equation (9) into

$$P_{i,k} = P_{i,j} P_{j-1,k} \quad \text{(block-carry-propagate)} \tag{10}$$
$$G_{i,k} = G_{i,j} + P_{i,j} G_{j-1,k} \quad \text{(block-carry-generate)} \tag{11}$$
$$C_j = G_{j-1,k} + P_{j-1,k} C_k \tag{12}$$

where $i \geq j > k$, $G_{i,i} = g_i$ and $P_{i,i} = p_i$.

The tree-like circuit of the CLA with $n = 8$ is shown in Figure 2. It consists of two kinds of modules: A-modules, which compute *carry-propagate* (see Equation (5)), *carry-generate* (see Equation (6)), and sum bits (see Equation (7)), and B-modules which compute *block-carry-propagate* (see Equation (10)), *block-carry-generate* (see Equation (11)) and carry bits (see Equation (12)).

In a synchronous CPU with a CLA, the addition operation is synchronized by clock pulses. The clock period must be large enough to allow all the possible input configurations

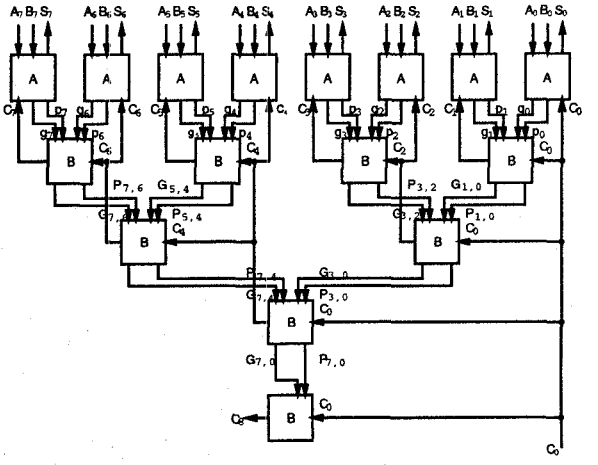

Figure 2. Carry-Lookahead Adder

to be computed. Thus, the CLA must work under the worst possible condition. The worst computation delay of the CLA is 2 unit delays [1] of the A-module plus $2 \log_2 n - 1$ unit delays of the B-module.

3. DI Carry-Lookahead Adders

Delay-Insensitive Carry-Lookahead Adders (DICLA) can be implemented by using dual-rail signaling in input bits, sum bits, and carry bits, and by using one-hot code in the internal signals. A DICLA can be built with two basic modules: C and D, connected in a tree-like structure (Figure 3). The equations of the C-module are defined as follows:

$$k_i = A_i^0 B_i^0 \quad \text{(carry-kill)} \tag{13}$$
$$g_i = A_i^1 B_i^1 \quad \text{(carry-generate)} \tag{14}$$
$$p_i = A_i^0 B_i^1 + A_i^1 B_i^0 \quad \text{(carry-propagate)} \tag{15}$$
$$S_i^0 = A_i^0 B_i^0 C_i^0 + A_i^1 B_i^1 C_i^0 + A_i^0 B_i^1 C_i^1 + A_i^1 B_i^0 C_i^1 \tag{16}$$
$$S_i^1 = A_i^1 B_i^1 C_i^1 + A_i^1 B_i^0 C_i^0 + A_i^0 B_i^1 C_i^0 + A_i^0 B_i^0 C_i^1 \tag{17}$$

where $i = 0, .., n-1$.

Note that in a DICLA the input data bits and the output data (sum) bits may be propagated at any time in any order. The completion signal, *finish*, may be generated when needed as follows:

$$finish = (C_n^0 + C_n^1) \prod_{i=0}^{n-1} (S_i^0 + S_i^1)$$

The C-module is shown in Figure 3(a). The dual-rail signals on the left-hand side of Figure 3(a) are grouped as $A_i = (A_i^0, A_i^1)$, $B_i = (B_i^0, B_i^1)$, $C_i = (C_i^0, C_i^1)$, $S_i = (S_i^0, S_i^1)$, and $I_i = (k_i, g_i, p_i)$. The resulting C-module is shown on the right-hand side of Figure 3(a).

[1] One unit delay of A- or B-modules is equal to the delay of two *AND* or *OR* gates, if they are implemented in two-level logic .

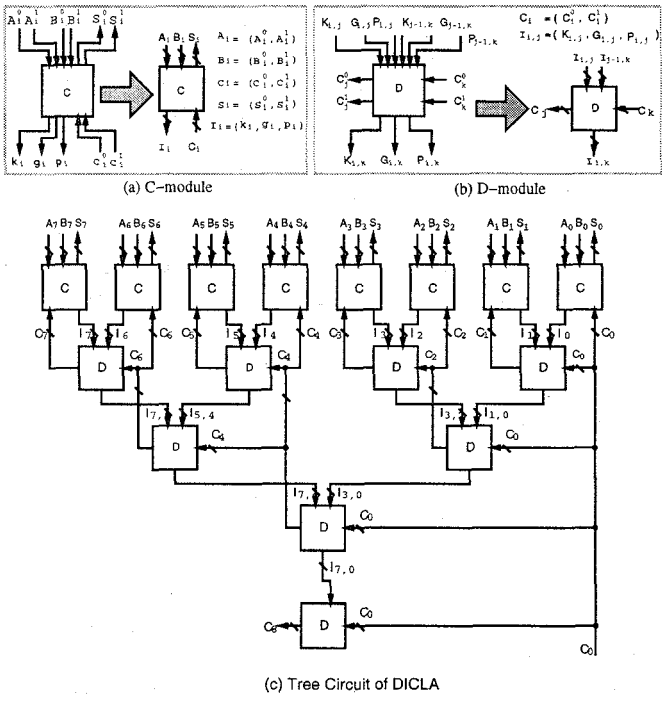

Figure 3. DI Carry-Lookahead Adder

The equations for the D-module are defined as follows:

$$P_{i,k} = P_{i,j}P_{j-1,k} \quad \text{(block-carry-propagate)} \quad (18)$$

$$K_{i,k} = K_{i,j} + P_{i,j}K_{j-1,k} \quad \text{(block-carry-kill)} \quad (19)$$

$$G_{i,k} = G_{i,j} + P_{i,j}G_{j-1,k} \quad \text{(block-carry-generate)} \quad (20)$$

$$C_j^0 = K_{j-1,k} + P_{j-1,k}C_k^0 \quad (21)$$

$$C_j^1 = G_{j-1,k} + P_{j-1,k}C_k^1 \quad (22)$$

The D-module is shown in Figure 3(b). The signals on the left-hand side of Figure 3(b) are grouped as $I_{i,j} = (K_{i,j}, G_{i,j}, P_{i,j})$, and $C_i = (C_i^0, C_i^1)$. The resulting D-module is shown on the right-hand side of Figure 3(b).

Initially, all carries (i.e. C_i^0 and C_i^1 for $i = 1, 2, .., n$) and the internal signals (i.e. $K_{i,j}, G_{i,j}$ and $P_{i,j}$) are zero, because all primary inputs (i.e. A_i^0, A_i^1, B_i^0 and B_i^1 for $i = 0, 1, .., n-1$) and input carry (i.e. C_0^0 and C_0^1) are zero. During the computation, C_i^0 and C_i^1 will not be both turned on and exact one of the internal signals will be turned on.

The performance of the DICLA may be further improved by some speed-up circuitry. It is obvious that if $A_i = B_i$ (i.e. carry-kill or carry-generate), then the carry output, C_{i+1}, is independent of the carry input, C_i. The tree-like circuit, shown in Figure 3, does not take full advantage of this feature to speed up the carry computation.

4. DICLA With Speed-Up Circuitry

The idea of speeding up carry computation of tree-like adders is to send the *carry-generate*'s and *carry-kill*'s to any possible stages which may use this information to compute carries immediately.

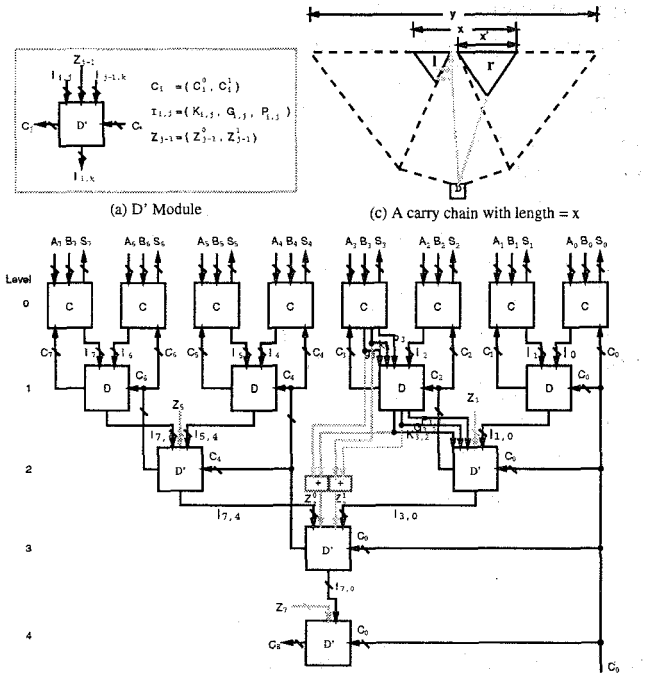

Figure 4. D'-module and DI Carry-Lookahead Adder with Speed-up Circuits

The D'-module (D-module with speed-up circuitry), shown in Figure 4(a), is designed to speed up the carry computation. It is defined in the following way:

$$P_{i,k} = P_{i,j}P_{j-1,k} \quad (23)$$

$$K_{i,k} = K_{i,j} + P_{i,j}K_{j-1,k} \quad (24)$$

$$G_{i,k} = G_{i,j} + P_{i,j}G_{j-1,k} \quad (25)$$

$$C_j^0 = Z_{j-1}^0 + K_{j-1,k} + P_{j-1,k}C_k^0 \quad (26)$$

$$C_j^1 = Z_{j-1}^1 + G_{j-1,k} + P_{j-1,k}C_k^1 \quad (27)$$

The definitions of the *block-carry-propagation*, *block-carry-kill* and *block-carry-generate* of the D'-module are the same as those of the D-module. However, each output carry (C_j^0/C_j^1) has an extra speed-up signal (Z_{j-1}^0/Z_{j-1}^1).

The general form of speed-up circuits is defined as follows:

$$Z_{j-1}^0 = \sum_{x=0}^{j-1} k_x \prod_{y=x+1}^{j-1} p_y \quad (28)$$

$$Z^1_{j-1} = \sum_{x=0}^{j-1} g_x \prod_{y=x+1}^{j-1} p_y \quad (29)$$

where $1 \leq j \leq n$. These two equations are taken from the scheme of full carry-lookahead adders.

It is impractical to implement the speed-up circuits shown in Equations (28) and (29), when n is large, because of the practical limitations on fan-in and fan-out, irregular structure, and many long wires. In addition, the logic complexity of the speed-up circuits increases more than linearly.

Fortunately, all the above problems can be solved by using the properties of a tree-like structure. That is, instead of fully employing the carry-lookahead generation to each carry, our speed-up mechanism focuses on the root nodes of a subtree. By trading time with area, the logic complexity of the adder with the speed-up circuitry is brought down to a linear function of n, the number of inputs; and, the average computation time is proportional to $\Theta(\log \log n)$. Both proofs will be given in Section 5.

The above two equations can be reformulated in a more efficient and economical way in logic as:

$$Z^0_{j-1} = k_{j-1} + \sum_{x=1}^{l-2} K_{j-1,j-2^x}$$

$$Z^1_{j-1} = g_{j-1} + \sum_{x=1}^{l-2} G_{j-1,j-2^x}$$

where $j = 2, 4, 6, ..., n$, l is the level in the tree. (The level of tree leaves is 0.) Note that only the modules at $level \geq 2$ need speed-up circuits.

In this speed-up mechanism, each root node above level 1 receives speed-up signals from the left root nodes of its right subtree. For example, the D'-module at level 3 in Figure 4(b) has the following speed-up signals: $Z^1_3 = g_3 + G_{3,2}$ and $Z^0_3 = k_3 + K_{3,2}$. The DICLA with the speed-up circuitry (DICLASP) for $n = 8$ is shown in Figure 4(b). The speedup circuitry is shown by dotted lines, and Z^1_3 and Z^0_3 are shown in detail.

Figure 4(c) is used to demonstrate the power of the speed-up circuitry. Imagine a carry chain (i.e. a sequence of carry-propagate's) with length $= x$ which is spanned by two subtrees, shown in the solid triangles of Figure 4(c). The **r** (right) subtree has a carry chain with length $= x'$ and the **l** (left) subtree has a carry chain with length $= x - x'$. Assume that x' and $x - x'$ are powers of 2 and $x' \geq x - x'$. There must exist two subtrees with the same size (say $y/2$) which contain the **r** and **l** subtrees. It takes $2\log_2 x' + 1$ module delays to compute the carries in the **r** subtree. Thanks to the speed-up circuitry, the carries can be directly propagated from the right subtree to the left subtree. The **l** subtree can start carry computation at $\log_2 x' + 1$ (the computation time from the leaves to the root node of **r** subtree) plus 1 (the delay of the D'-module which is the root of the two dotted subtrees) delays. Thus, it takes only $\log_2 x' + 1 + Max(\log_2 x', 1 + \log_2(x - x') + 1)$ delays to compute these carries. Without the speed-up circuitry, it would take $\log_2 y + 1 + \log_2(x - x') + 1$ delays.

5. Complexity Analysis

In this section we analyze the logic complexity and the average computation time of the adder designed in the previous section. We assume three things: First, the delay through each module (e.g. C-, D- and D'-modules) is d; second, the number of bit positions of input arguments, n, is a power of 2, i.e. $n = 2^k$; third, the distribution of the input configurations is a uniform distribution.

5.1. Time Complexity

The time required to perform an addition (*computation time*) in an adder is the time required for propagating the carries (*carry propagation time*) in stages plus one more delay to compute the sum bits. The computation time of an adder is sensitive to the numbers to be added. The upper and lower bound proofs of the time complexity are an extension of the proofs for CCSAs [9].

Theorem 1 *For any input configuration, the carry propagation time is proportional to the logarithm of the length of the longest carry chain.*

Proof: Consider a carry chain with length x in an input configuration where $2^{l-1} < x \leq 2^l$.

Upper bound: If the carry chain is contained in one subtree

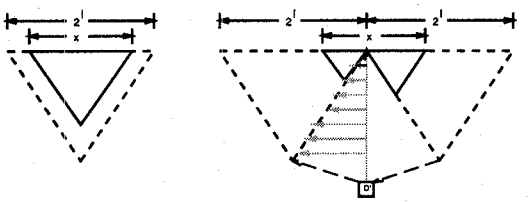

Figure 5. Spanning of the longest carry chain

with the length of leaves $= 2^l$ in DICLASP (see Figure 5(a)) then the carries in the carry chain can be computed in $(2l + 1)d$. To propagate the last carry of the carry chain to the next stage, it may need two more delays. So the carry propagation time is at most $(2l + 3)d$. Otherwise, the carry chain must span more than one subtree with the length of leaves $= 2^l$ (see Figure 5(b)). There must exist two subtrees with the length of leaves $= 2^l$ in DICLASP containing the carry chain. In this case the propagation time can be computed in at most $(2l + 3)d$. To summarize, in both cases the carry propagation time is at most $(2\log_2 x + 5)d$.

Lower bound: There must exist one subtree with the length of leaves $= 2^{l-2}$ in DICLASP such that the leaves of the subtree are contained in the carry chain. Thus, the carry propagation time is at least $(2(l-2)+1)d = (2l-3)d$, i.e. the carry propagation time is at least $(2\log_2 x - 5)d$. □

Now we shall show that the average computation time of DICLASP is $\Theta(\log\log n)$.

Theorem 2 *A lower bound of the average carry propagation time of the DICLASP is* $\Omega(\log\log n)$.

Proof: A lower bound is obtained by considering $\frac{2n}{\log_2 n}$ non-overlapping segments. Each segment has $\frac{\log_2 n}{2}$ bits.

The probability that a carry propagates across one of these segments is the probability that all $\frac{log_2 n}{2}$ stages in the segment propagate the carry. So, the probability that the carry is propagated across all $\frac{\log_2 n}{2}$ stages is $0.5^{\frac{\log_2 n}{2}} = \frac{1}{\sqrt{n}}$. The probability that a carry is not propagated across a segment is $(1-\frac{1}{\sqrt{n}})$. The probability that there is no segment over which a carry is propagated is $(1-\frac{1}{\sqrt{n}})^{\frac{2n}{\log_2 n}}$.

For $x > 1$, $(1-\frac{1}{x})^x < \frac{1}{e}$, and for $y \geq x > 1$, $(1-\frac{1}{x})^y < \frac{1}{e}$. This is satisfied when $2n \geq \sqrt{n}\log_2 n$, i.e. when $n \geq 1$. So, for $n \geq 1$, the probability that a carry is not propagated across at least one segment is less than $\frac{1}{e}$. This means that the probability that a carry is propagated across at least one segment is at least $(1-\frac{1}{e})$.

Thus, the average carry propagation time is at least $(1-\frac{1}{e})\cdot(2\log_2(\frac{\log_2 n}{2})-5)d = (1-\frac{1}{e})\cdot(2\log_2\log_2 n - 7)d$, i.e. the average carry propagation time (APT) is $\Omega(\log\log n)$. □

Theorem 3 *An upper bound of the average carry propagation time of the DICLASP is* $O(\log\log n)$.

Proof: Let $P(i)$ be the probability that the longest carry chain has length i, and let $T(i)$ be the maximal carry propagation time of chains of length i. Then, $APT = \sum_{i=0}^{n} P(i)\cdot T(i) \leq P(0)\cdot 2d + \sum_{i=1}^{2\log_2 n} P(i)\cdot(2\log_2 i + 5)d + \sum_{i=2\log_2 n+1}^{n} P(i)\cdot(2\log_2 i + 5)d$. The probability that the maximal carry chain is 0 is $\frac{1}{2^n}$. Thus, $P(0)\cdot 2d = \frac{2}{2^n}d$.

The second term can be derived as follows: $\sum_{i=1}^{2\log_2 n} P(i) < 1 \Rightarrow \sum_{i=1}^{2\log_2 n} P(i)\cdot(2\log_2 i + 5)d < (2\log_2(2\log_2 n) + 5)d = (2\log_2\log_2 n + 7)d$.

The third term can be derived by considering n overlapping segments of length $2\log_2 n$. Each carry chain of length longer than $2\log_2 n$ contains at least one segment completely. The probability that a carry propagates accross such a segment is $(0.5)^{2\log_2 n} = \frac{1}{n^2}$. The number of segments, even allowing overlapping, is n. Also, the length of any carry chain is at most n. Thus, the contribution to the expected carry propagation time of carry chains that are longer than $2\log_2 n$ is at most $n\cdot\frac{1}{n^2}\cdot(2\log_2 n + 5)d = \frac{2\log_2 n+5}{n}d$.

Since, $APT < (\frac{2}{2^n} + 2\log_2\log_2 n + 7 + \frac{2\log_2 n+5}{n})d$, the average carry propagation time is $O(\log\log n)$. □

Theorem 4 *The average computation time of the DICLASP is* $\Theta(\log\log n)$.

Proof: By Theorems 2 and 3. □

5.2. Logic Complexity

Theorem 5 *The logic complexity of DICLASP is a linear function of n.*

Proof: It is easy to show that the n-input DICLASP requires n C-modules, $\frac{n}{2}$ D-modules, $\frac{n}{2}$ D'-modules and the speed-up circuitry. We show that the speed-up circuitry is also a linear function of n by counting the total number of added inputs.

Consider the D'-modules at level i where $i \leq k$ ($k = \log_2 n$). There are 2^{k-i} D'-modules and each D'-module at level i has $2(i-1)$ inputs (i.e. $i-1$ inputs for carry generate and $i-1$ for carry kill) for the speedup circuitry. We also need the speed-up circuit, which contains $2k$ inputs, to compute the last carry. The total number of the added inputs is $N = 2(k + \sum_{i=2}^{k} 2^{k-i}(i-1)) = 2n - 2$. □

For VLSI implementation, the fan-in and fan-out for C-, D- and D'-modules used in DICLASP are no problem at all. The maximal fan-in of the speed-up circuitry is $log_2 n$. This is not a severe problem either, except for a very large n.

6. Comparisons of Adders

In this section we compare the logic and time complexity for both synchronous and asynchronous adders. They are Ripple Carry Adder (RCA), Carry-Select Adder (CSA1) [1], Conditional-Sum Adder (CSA2) [18], Carry-Skip Adder (CSA3) [12], Carry-Completion Sensing Adder (CCSA) [8], Delay-Insensitive Ripple Carry Adder (DICLA) [14], Conditional-Sum Completion-Sensing Adder (CSCSA) [15], Carry-Lookahead Adder (CLA) [2], and, our DICLA and DICLASP. The logic(area) and time complexity of the above adders are listed in Table 1. The worst case computation time is assumed for the synchronous adders and average computation time is assumed for the asynchronous adders. Area-time efficiency is computed by multiplying logic complexity and time complexity.

Type of Adder	Circuit	Logic(Area) Complexity	Time Complexity	area-time efficiency
RCA	Sync	$O(n)$	$O(n)$	$O(n^2)$
CCSA	Async	$O(n)$	$O(\log n)$	$O(n\log n)$
DIRCA	Async	$O(n)$	$O(\log n)$	$O(n\log n)$
CSA1	Sync	$O(n\log n)$	$O(\log n)$	$O(n(\log n)^2)$
CSA2	Sync	$O(n\log n)$	$O(\log n)$	$O(n(\log n)^2)$
CSA3	Sync	$O(n\log n)$	$O(\log n)$	$O(n(\log n)^2)$
CSCSA	Async	$O(n\log n)$	$O(\log\log n)$	$O(n\log n\log\log n)$
CLA	Sync	$O(n)$	$O(\log n)$	$O(n\log n)$
DICLA	Async	$O(n)$	$O(\log n)$	$O(n\log n)$
DICLASP	Async	$O(n)$	$O(\log\log n)$	$O(n\log\log n)$

Table 1. Logic and time complexity of adders:

It is worth mentioning that CSCSA also has time complexity of $O(\log \log n)$. However, the logic complexity of CSCSA is $O(n \log n)$ and the design requires circuits to detect the true sum in each level and to route the completed sum from any arbitrary level to the output latch. Our DICLASP has the best area-time efficiency among these adders.

7. CMOS Implementation

To implement economic DICLASPs in CMOS technology, Martin's method to design economic delay-insensitive datapath circuits can be applied [14]. The behaviors of gates can be represented through a set of production rules; and then they can be directly implemented in CMOS. Here we just show the CMOS circuits of DIRCA and DICLASP. For more detail, see [14, 4].

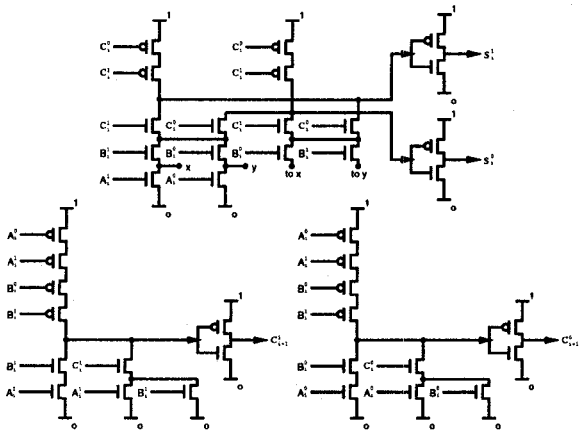

Figure 6. CMOS Implementation of DIRCA cell

The CMOS implementation of DIRCA cell is shown in Figure 6. It contains only 40 transistors. Note that in Martin's paper [14] the transistor count per DIRCA cell is 42. The improvement is due to factoring out subexpressions as much as possible. That is, some of the transistors may be shared. Compared to the 40 transistors needed for an equivalent (no pass-transistors) cell design in clocked logic, the asynchronous adder has better performance (i.e. $O(\log n)$) and uses the same amount of circuitry.

The CMOS implementation of C-, D-, D'-modules and speed-up circuitry are shown in Figure 7(a), (b), (c) and (d), respectively. Note that, in D'-module, we show only the implementation of C_j^0 and C_j^1 since the implementation of $K_{i,k}$, $G_{i,k}$ and $P_{i,k}$ is the same as that of D-module. C-, D-, D'-modules contain 36, 31, and 35 transistors, respectively.

For the D'-module in tree level 2, those g_i and k_i are directly connected to the Z (speed-up) signals. Thus, no transistors are needed. The transistor count for the speed-up circuitry can be derived in the following way: Consider the D'-modules at level i, where $i \geq 3$. There are 2^{k-i} D'-modules and each D'-module at level i contains $2(3+i)$ transistors (i.e. 3+i transistors for Z^0 and 3+i for Z^1). Also, the speed-up circuit for the last carry needs $2(4+k)$ transistors.

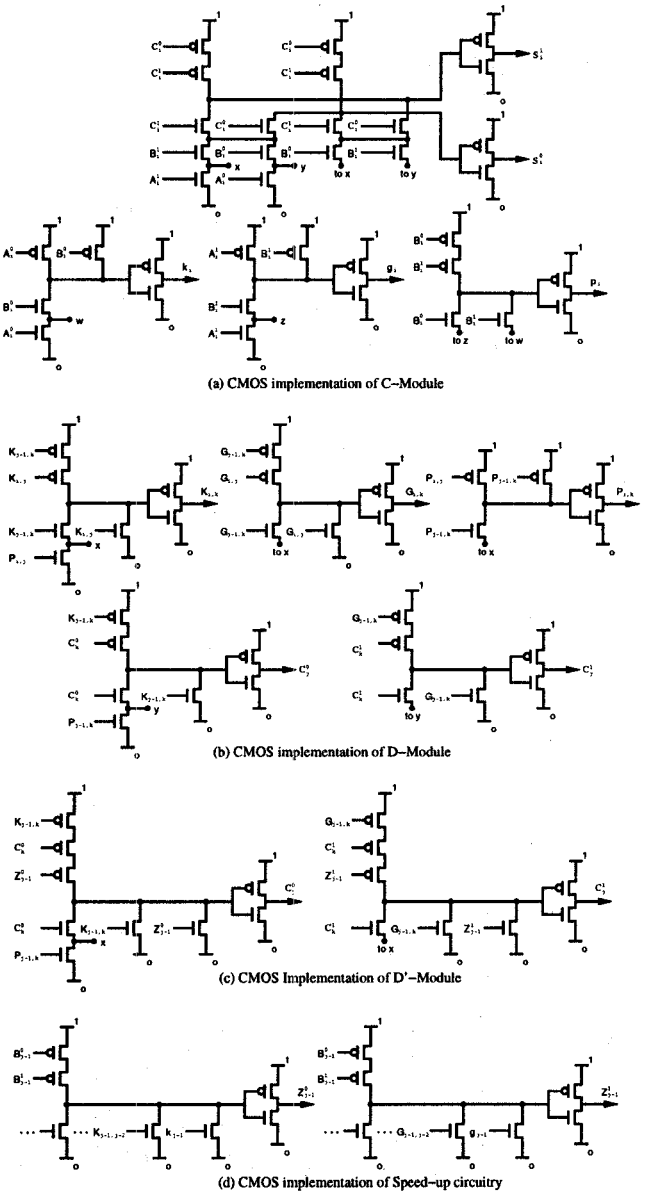

Figure 7. CMOS Implementation of DICLASP modules

For an n-bit adder ($n \geq 8$) the speed-up circuitry needs $2(4+k+(\sum_{i=3}^{k} 2^{k-i}(3+i))) = 3.5n-2$. transistors in total. Thus, an n-bit DICLASP needs $36n + 31(\frac{n}{2}) + 35(\frac{n}{2}) + 3.5n - 2 = 72.5n - 2$ transistors. It is worth mentioning that the speed-up circuitry in fact uses a very

small percentage of the CMOS area. Compared to Martin's n-bit ripple carry adder which needs $42n$ transistors, our n-bit DICLASP which needs fewer than $72.5n$ transistors is indeed practical in high speed processors.

	32-bit adder		
Adder	# of Transistors	BC Delay	WC Delay
DIRCA	1344	2.26 ns	34.96 ns
DICLASP	2318	2.83 ns	11.48 ns
	64-bit adder		
	# of Transistors	BC Delay	WC Delay
DIRCA	2688	2.30 ns	68.34 ns
DICLASP	4638	2.90 ns	13.80 ns

Table 2. Comparison of DICLASP and DIRCA:

To compare our DICLASP with the DIRCA of Martin's, we simulated both circuits using SPICE with MOSIS 2 micron CMOS parameters. The comparison of our DICLASP and Martin's DIRCA is shown is Table 2. BC delay in the table is the best case delay (i.e. the length of carry chain is zero), and WC delay is the worst case delay (i.e. the length of carry chain is equal to the number of input bits). Note that the above-mentioned delays do not include the completion-tree delay. Computing the average delay of both circuits (e.g. generate 100,000 pairs of random numbers and simulate them in SPICE) is really a formidable job.

8. Optimization of DICLASP

Adding speed-up circuitry not only enhances the performance but also makes some logic redundant, which can be removed. If the carry kill's and carry generate's are exploited in the speed-up circuitry, these signals need not to propagate through the tree. For example, since g_3 and k_3 are used by the D'-module which computes C_4, it is unnecessary to route them to the D-module which computes C_3. In general, the left carry kill and left carry generate signals of D- and D'-modules can be eliminated.

9. Conclusions

We have proposed a delay-insensitive carry-lookahead tree adder (i.e. DICLASP) in which the logic complexity is a linear function of n and the average computation time is proportional to the logarithm of the logarithm of n. To the best of our knowledge, our adder has better area-time efficiency than any other adders [16, 7, 2, 14] have.

The proposed delay-insensitive carry-lookahead tree adders are suitable for VLSI implementation because of their regular structure. We also presented an economic CMOS implementation of the proposed adders. We believe this work can be applied in the design of high speed processors.

References

[1] O. J. Bedrij. Carry-select adder. *IRE Transactions on Electronic Computers*, 11(6):340–346, June 1962.

[2] R. P. Brent and H. T. Kung. A regular layout for parallel adders. *IEEE Transactions on Computers*, 31(3):260–264, Mar. 1982.

[3] B. Briley. Some new results on average worst case carry. *IEEE Transactions on Computers*, 22(5):459–463, May 1973.

[4] F.-C. Cheng. Delay-insensitive carry-lookahead adders. Technical report, Department of Computer Science, Columbia University, 1996.

[5] J. C. Ebergen. From functional specification to a delay-insensitive circuit. Technical Report CS-89-44, University of Waterloo, Oct. 1989.

[6] I. Flores. *The Logic of Computer Arithmetic*. Prentice-Hall, 1963.

[7] M. A. Franklin and T. Pan. Performance comparison of asynchronous adders. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 117–125, Nov. 1994.

[8] B. Gilchrist, J. H. Pomerene, and S. Y. Wong. Fast carry logic for digital computers. *IRE Transactions on Electronic Computers*, EC-4(4):133–136, Dec. 1955.

[9] M. Greenstreet. *Private Communication*. 1995.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 1990.

[11] K. Hwang. *Computer Arithmetic: Principles, Architecture, and Design*. John Wiley & Sons, 1979.

[12] M. Lehman and N. Burla. Skip techniques for high-speed carry-propagation in binary arithmetic units. *IRE Transactions on Electronic Computers*, 10(12):691–698, Dec. 1961.

[13] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In W. J. Dally, editor, *Sixth MIT Conference on Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[14] A. J. Martin. Asynchronous datapaths and the design of an asynchronous adder. *Formal Methods in System Design*, 1(1):119–137, July 1992.

[15] N. M. Martin and S. P. Hufnagel. Conditional-sum early completion adder logic. *IEEE Transactions on Computers*, 29(8):753–756, Aug. 1980.

[16] T.-F. Ngai and M. J. Irwin. Regular, area-time efficient carry-lookahead adders. In *Proc. IEEE Symp. on Computer Arithmetic*, pages 9–15. IEEE Computer Society Press, 1985.

[17] C. L. Seitz. System timing. In C. A. Mead and L. A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[18] J. Sklansky. Conditional-sum addition logic. *IRE Transactions on Electronic Computers*, 9(6):226–231, June 1960.

[19] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

[20] S. H. Unger. Tree realizations of iterative circuits. *IEEE Transactions on Computers*, 26(4):365–393, Apr. 1977.

[21] C. H. K. van Berkel, C. Niessen, M. Rem, and R. W. J. J. Saeijs. VLSI programming and silicon compilation. In *Proc. International Conf. Computer Design (ICCD)*, pages 150–166, Rye Brook, New York, 1988. IEEE Computer Society Press.