19th Asilomar Conf. on Circuits, Systems & Comp Nov 85

Time-Optimal Design of a CMOS Adder

Belle W. Y. Wei
Clark Thompson
Yih-farn Chen†

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720
Telephone: (415)642-5250

ABSTRACT

In this paper, we present a systematic method of implementing a VLSI parallel adder. First, we define a family of adders, based on a modular design. Our design uses three types of component cells, which we implement in static CMOS. We then formulate the adder design as a dynamic programming problem, optimizing with respect to time. As a result, we have found the fastest 32-bit CMOS adder in our design family.

1. Introduction

Addition is the heart of computer arithmetics, and the arithmetic unit is often the work horse of a computational circuit. As a result, fast circuits for addition have been studied extensively. Most authors study addition in the context of TTL design[De84] [Ku78], for which gate count and delay are performance factors. These factors are not sufficient to evaluate a design for possible implementation in VLSI. Accordingly, in this paper, we optimize our adder circuit with respect to a refined model of VLSI delay and area.

In 1982, Brent and Kung[Br82] proposed a simple VLSI design for a fast adder. In their model, fan-out is limited to 2. This proves too restrictive. Fan-out can be traded off for shorter interconnects and a smaller area, which may in fact result in a faster circuit.

In this paper we present a realistic and systematic approach in constructing a fast parallel adder. Our approach is based on Ladner and Fischer's "parallel prefix computation"[La80], which is outlined in Section 2. In Section 3, we specify three basic circuit blocks of a parallel adder. Three types of cells are designed and analyzed in Section 4. From the timing analysis of Section 4, we optimize our adder design by solving a dynamic programming problem in Section 5. An example of a 32-bit adder is presented. In the last section, we summarize our approach to adder designs, and indicate how our method could be extended.

2. Mathematical Description

Binary addition can be transformed into a parallel computation by introducing an associative operator o[La80]. If we define the carry generate term, gIN_i, and the carry propagate term, pIN_i, for each bit position i, then the carry c_i for each bit obeys the following recurrence relation:

$$gIN_i = a_i b_i$$

$$pIN_i = a_i \oplus b_i$$

$$c_i = G_i \quad \text{for } i = 1,2,\ldots,n$$

where

$$(G_i\,,P_i) = \begin{cases}(gIN_1\,,pIN_1) & \text{if } i=1\\ (gIN_i\,,pIN_i)o(G_{i-1}\,,P_{i-1}) & \text{if } n\geq i>1\end{cases} \tag{2.1}$$

and o is a concatenation operator defined as follows:

$$(g_l\,,p_l)o(g_r\,,p_r) = (g_l + p_l g_r\,,p_l p_r) \tag{2.2}$$

Note that o is not commutative. Its left argument $(g_l\,,p_l)$ is treated differently from its right argument $(g_r\,,p_r)$. After the carry bit c_i is computed, the sum bit s_i is :

$$s_i = pIN_i \oplus c_{i-1} \quad \text{for } i=2,..,n$$

$$s_1 = p_1 \tag{2.3}$$

Given the fact that o is associative, we can choose a m such that $i\geq m>1$ and rewrite $(G_{i,1}\,,P_{i,1})$ as follows:

$$(G_{i,1}\,,P_{i,1}) = (G_{i,m}\,,P_{i,m})o(G_{m-1,1}\,,P_{m-1,1}) \tag{2.4}$$

where

$$(G_{i,m}\,,P_{i,m}) = \begin{cases}(gIN_m\,,pIN_m) & \text{if } i=m\\ (gIN_i\,,pIN_i)o(G_{i-1,m}\,,P_{i-1,m}) & \text{if } i>m\end{cases} \tag{2.5}$$

We observe that $(G_{i,m}\,,P_{i,m})$ and $(G_{i-m+1,1}\,,P_{i-m+1,1})$ have similar functional forms. Both are functions of $i-m+1$ consecutive input bits and both require $i-m$ applications of the associative operator o. As a result, both can be computed by the same circuit.

3. Implementations

To implement the functions defined in Section 2, the three circuit blocks shown in Figure 3.1 are required. The first is a combinational circuit, labeled the *pre-condition circuit*, which gates in adder inputs a_i and b_i to generate the ini-

† This work was supported in part by National Science Foundation, through its Computer Engineering program, under grant DMC 8406408.

tial pIN_i and gIN_i for each bit position i. The computed p and g terms are then fed into the *fast carry generator* which performs the operations defined in Equations 2.1 and 2.2. It is this circuit which allows accelerated carry computations and this is the focus of our study. The third block is a *sum* circuit, consisting of a row of $\oplus$gates, to combine the carry propagate bits (pIN_i) from the first block with the carry bits (c_i) from the second block, according to Equation 2.3.

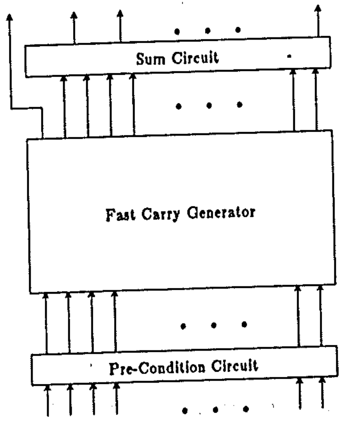


Fig.3.1 Three Functional Blocks of an Adder

diagrams.

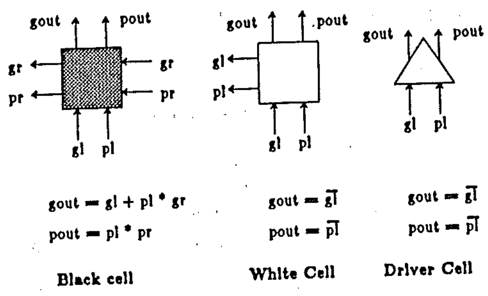


Fig.3.2 Three Basic Types of Adder Cells

We use three basic types of cells to implement the parallel carry computation: black cells, white cells, and driver cells. These are shown in Figure 3.2. Note that some of the inputs to our black and white cells "pass through" the cells. Specifically, the (g_r , p_r) inputs of the black cell are available as outputs. This convention greatly simplifies our wiring diagrams.

The black cell performs the associative concatenation defined in Equation 2.2. Based on a static CMOS implementation, the black cell is of two categories, *ba* and *bb*. The *ba* cell shown in Figure 3.3a gates in positive-true signals and produces complemented outputs. The *bb* cell, shown in Figure 3.3b, gates in complemented inputs and outputs positive-true signals.

Each *bb* or *ba* cell is further composed of *p* and *g* subcells. The *p* subcell produces a *pout* signal and the *g* subcell produces a *gout* signal.

Definition *The fan-out f is the number of subcells that a signal drives.*

For example, the fan-out for p_l (or $p_l.bar$) inside a black cell is 2, as it drives both p and g terms. However, the fan-out for g_l, g_r , p_r and their complements is 1 as they each drive only one subcell.

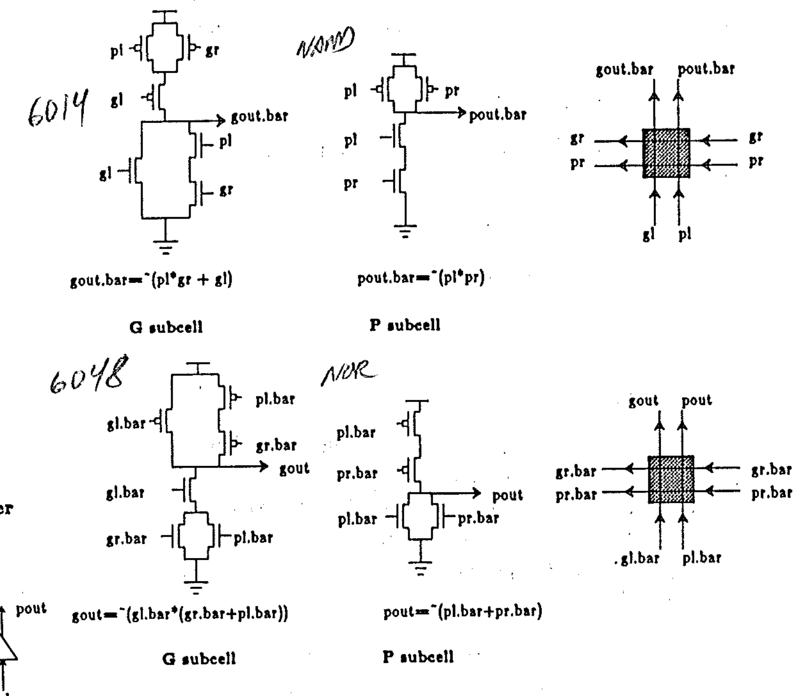


Fig.3.3 Black Cell Implementations in Static CMOS

In a static CMOS implementation, the function of a white cell is to invert the incoming signals, providing the proper input polarity for subsequent black cell operations. In the case of long wire interconnects or large fan-outs, a specially ratioed inverting driver, either in single stage or cascaded stages, is needed to optimize circuit performance.

4. Circuit Models

To construct a fast adder, we must first evaluate the signal delay associated with each type of cell. Given a static CMOS design, the circuit resistances and capacitances can be estimated. The associated signal delay is computed. Drivers are modeled explicitly and are an integral part of our circuit layout.

In implementing the black and white cells, we use minimum-length transistors for the pull-down network of each subcell. PMOS pull-up transistors are ratioed so that the maximum (over all possible input conditions) of pull-up and pull-down channel resistances are equal. We define this resistance as R_t. For simplicity, we also assume that the capacitance C_t and resistance R_t of the horizontal interconnect between neighboring cells are the same as the R and C values of the vertical interconnect. This condition depends on the specific implementation, and is discussed further in Section 6.

In a static CMOS design, a pair of a PMOS pull-up and an NMOS pull-down transistors constitutes a basic inverting unit. The input signal drives the gates of both the pull-up and pull-down transistors. Let C_g be the total gate capacitance of such an inverting unit, then we can approximate the generation time of its output signal, t_{out}, as a function of its interconnect and channel resistances (R_t and R_t, respectively), its load capacitance, its fan-out f, and its input ready time, t_{in}:

$$t_{out} = t_{in}+(R_t+R_t f)(C_t+C_g)f$$

In the case when metal is used for interconnects and $R_t > R_t f$, then t_{out} becomes:

$$t_{out} = t_{in}+R_t(C_t+C_g)f$$

Let τ be a normalized time constant, defined as

$$\tau = R_t(C_t+C_g)$$

Then

$$t_{out} = t_{in}+\tau f \tag{4.1}$$

The fan-out f of a subcell is variable, depending on the type (p or g) of subcell, and on the type and number of succeeding cells. This can be illustrated using the example of a 5-bit adder shown in Figure 4.1, in which each cell position is identified by a pair of depth and bit coordinates.

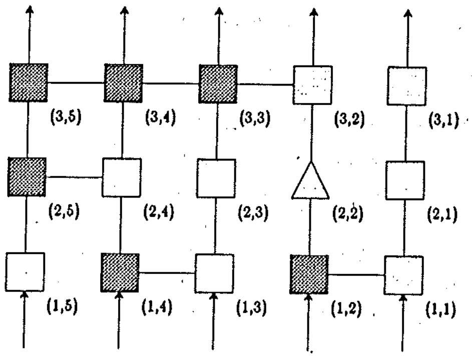
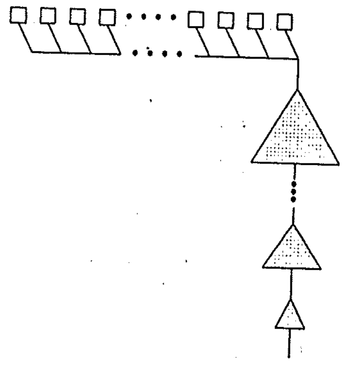


Fig.4.1 A 5-bit Adder Example

For example, Cell (2,5) refers to the second cell which lies on the vertical path of input bit number 5. Since the black cell (2,5) is followed by another black cell (3,5) in a vertical connection, the output signals of Cell (2,5), p_{out} and g_{out}, become p_l and g_l of the following black cell. As a result, the fan-out for p_{out} is 2 as it drives both the p and g subcells of Cell (3,5), and the fan-out is 1 for g_{out} as it drives only the g term of Cell (3,5). For a vertical white-cell to black-cell connection, for example Cell (2,4) to Cell (3,4), the fan-outs are the same: $f_{pout} = 2$, $f_{gout} = 1$. On the other hand, a p_{out} (or g_{out}) signal at a white or black cell may drive many p_r (or g_r) inputs. For example, g_{out} (or p_{out}) of the driver cell (2,2) has a fan-out of 4, since it drives one subcell in each of (3,2), (3,3), (3,4), and (3,5).

Signal propagation time through a cascaded driver, d, is a function of the driver's fan-out f_d, the ratio r between successive stages, and the number s of cascaded stages[Me80]. The minimum delay can be obtained if the driver has the ratio of $r = f_d^{\frac{1}{s+1}}$. This is shown in Figure 4.2.



'84 1X, 2X, 3X,

Fig.4.2 Multi-staged Driver

The corresponding minimum propagation delay, $delay(s,f_d)$, of the cascaded driver is:

$$delay(s,f_d) = (s+1)(f_d^{\frac{1}{s+1}})\tau \tag{4.2}$$

Thus, for an s-stage driver of fan-out f_d,

$$t_{dout} = t_{din}+delay(s,f_d) \tag{4.3}$$

Note that when s=0 and no driver is used, Equation 4.3 is the same as Equation 4.1.

Given the above analysis, we can evaluate the generation time for each circuit signal as the sum of its input ready time and delay factor. Define t_{gout} as the time when signal g_{out} is ready, t_{gl} as t_{gout} of the cell producing g_l , and t_{gr} as t_{gout} of the cell producing g_r. Similar definitions apply to t_{pl} and t_{pr}. We can then formulate the input ready time for the g subcircuit of a black cell, t_{gin}, as:

$$t_{gin} = \max\{t_{gl} , t_{pl} , t_{gr}\}$$

And t_{pin} for the p subcircuit of a black cell becomes:

$$t_{pin} = \max\{t_{pl} , t_{pr}\}$$

If we define f_g (resp. f_p) be the fan-out of the subcell under analysis, then

$$t_{gout} = t_{gin} + delay(s,f_g) \tag{4.4}$$

A similar formulation can written for signal p_{out}:

$$t_{pout} = t_{pin} + delay(s,f_p) \tag{4.5}$$

5. Fast Adder Architecture

Ladner and Fischer defined a family of circuits, $P_k(n)$, for solving the parallel prefix problem on n inputs. Assuming n is a power of 2, the recursive construction of $P_k(n)$ for $k \geq 1$ is shown in Figure 5.1.

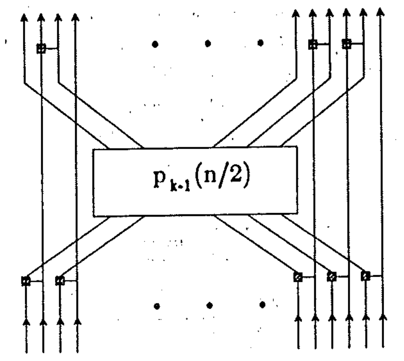


Fig.5.1 Recursive Construction of a $P_k(n)$ Adder

The basis of the recursion, the $P_0(n)$ circuit, is shown in Figure 5.2. In these figures, black boxes represent the concatenation operation, and are either *ba* or *bb* cells as appropriate.

In this paper, we limit the design space to a family of circuits, $R(n)$, which is shown in Figure 5.3. Note that $R(n)$ is a superset of $P_0(n)$, because our construction composes blocks of arbitrary sizes. Our circuit $R(n)$ has a large fan-out from the most significant bit of the right block, that is, bit m, broadcasting it to all bit positions of the left block.

Note that, in general, the depth of $R(n-m)$ may be larger than the depth of $R(m)$. This allows us to place a

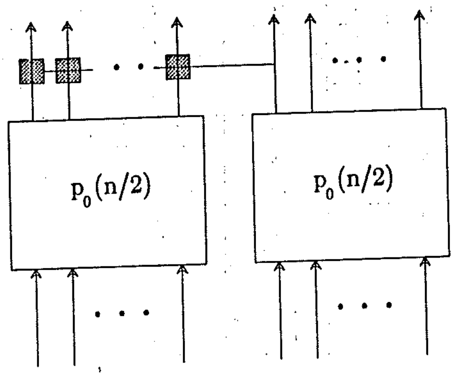


Fig.5.2 $P_0(n)$ Adder Circuit

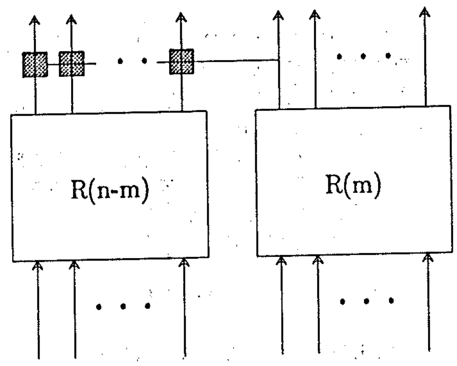


Fig.5.3 Recursive Construction of an R(n) Adder

multi-staged driver on the output of the most significant bit of $R(m)$. See Figure 5.4.

We observe that there are two possible critical paths in $R(n)$. One is the propagation through the leftmost vertical path. The other is the delay through the leftmost bit of

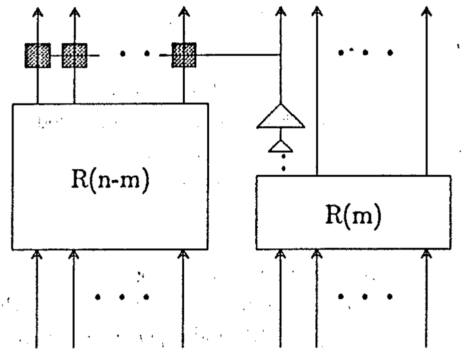


Fig.5.4 Drivers used in building an R(n) Adder

$R(m)$, which drives large fan-out and interconnect capacitances. Since the critical paths converge at the leftmost top cell, we focus our analysis on this cell. We can thus compare the adders in $R(n)$ by comparing the times at which their leftmost (p , g) outputs are produced. This "principle of optimality" validates a dynamic programming approach to choosing the best place m at which to decompose an n-bit adder into subcircuits $R(n-m)$ and $R(m)$. Our design problem, then, is to evaluate $t_{gin}(n,1)$ in the following recurrence:

$$t_{gin}(i,j) = \min_{j \le m < i}\{max[t_{gin}(i,m+1)+1\tau\ ,\ t_{pin}(i,m+1)+2\tau,\ t_{gin}(m,j)+f(i,m,j)\tau]\}$$

$$t_{pin}(i,j) = \min_{j \le m < i}\{max[t_{pin}(i,m+1)+2\tau\ ,\ t_{pin}(m,j)+f(i,m,j)\tau]\} \qquad (5.1)$$

where

$t_{gin}(i,j)$	is the input ready time for the g term of the most significant bit of an adder block of size $i-j+1$;
$t_{pin}(i,j)$	is the input ready time for the p term of the most significant bit of an adder block of size $i-j+1$;
$f(i,m,j)$	is the load function of the block of size $m-j+1$ driving that of size $i-m$.

The load function $f(i,m,j)$ is defined as:

$$f(i,m,j) = \min_{\substack{0 \le s \le u \\ \text{and } u \equiv s \bmod 2}} \{delay(s,i-m+1)\} \qquad (5.2)$$

where

$$u = \max\{0, depth(i,m+1) - depth(m,j)\}$$

In equation 5.2, s is chosen to minimize the signal delay through the driver. The depth of this s-stage driver is limited to u, which is the depth difference between two component adder blocks, $R(i-m)$ and $R(m-j+1)$. Furthermore, the output polarity of the driver must match that of the left adder block, $R(i-m)$. Thus the parity of s must be the same as that of u.

Note that in Equation 5.1 the m minimizing $t_{gin}(i,j)$ may not be the same as the m minimizing $t_{pin}(i,j)$. We will address this problem in Lemma 2, below.

Assume that the input signals to the adder, a_i and b_i, are available at time 0. Further, assume that the signals emerging out of the *fast carry generator* have unit fan-out, then the basis for the dynamic programming is:

$$t_{gin}(j,j) = t_{pin}(j,j) = r \tag{5.3}$$

Lemma 1: The optimal circuit for adding bits $j+r$ through j is identical to the optimal circuit for adding bits $r+1$ through 1. Thus, $t_{pin}(j+r,j) = t_{pin}(r+1,1)$ and $t_{gin}(j+r,j) = t_{gin}(r+1,1)$

Proof sketch: Equations 5.1, 5.2, and 5.3 are not sensitive to the absolute magnitude of i and j. Only the difference $i-j$ is important. $\square$

Lemma 2: $t_{gin}(i,j) = t_{pin}(i,j)$ for $i \geq j$

Proof: We prove the lemma by induction on i. From Equation 5.3, the basis $j = i$ is trivially true. If $i > j$, and $t_{gin}(i,j) = t_{pin}(i,j)$, then from Lemma 1:

$$t_{gin}(i+1,m+1) = t_{gin}(i,m)$$

$$t_{pin}(i+1,m+1) = t_{pin}(i,m)$$

and by inductive hyphothesis,

$$t_{gin}(i,m) = t_{pin}(i,m) \text{ for } j \leq m \leq i$$

$$t_{gin}(m,j) = t_{pin}(m,j) \text{ for } j \leq m \leq i$$

From Equation 5.1,

$$t_{gin}(i+1,j) = \min_{j \leq m < i+1} \{ max[t_{gin}(i+1,m+1)+1r,\ t_{pin}(i+1,m+1)+2r,\ t_{gin}(m,j)+f(i+1,m,j)r] \}$$

Since r is a RC time constant, it must be nonnegative. Thus

$$t_{gin}(i+1,j) = \min_{j \leq m < i+1} \{ max[t_{pin}(i+1,m+1)+2r,\ t_{gin}(m,j)+f(i+1,m,j)r] \}$$

Substituting $t_{pin}(m,j)$ for $t_{gin}(m,j)$, we have

$$t_{gin}(i+1,j) = \min_{j \leq m < i+1} \{ max[t_{pin}(i+1,m+1)+2r,\ t_{pin}(m,j)+f(i+1,m,j)r] \} = t_{pin}(i+1,j)$$

hence the lemma. $\square$

Using dynamic programming, we have calculated the optimal adder configurations for n from 1 to 32. See Table 1. Using this table to construct an optimal n-bit adder, the left adder block should be $n-m$ bits wide and the right adder block should be m bits wide. The column labelled *driver stages* indicates the number of stages in the driver connected to the output of the most significant bit of the right adder block. The *depth* entries indicate the number of layers in the optimal n-bit adder. The generation time of the most significant bit for the n-bit adder is shown in the column labelled $t_{pin(gin)}$. Figure 5.5 illustrates the optimal 32-bit adder.

n	n-m	m	$t_{pin(gin)}$	depth$(n,1)$	driver stages
1	0	1	1.00	0	0
2	1	1	2.00	1	0
3	1	2	4.00	2	0
4	2	2	5.00	2	0
5	3	2	6.00	3	1
6	4	2	7.00	3	1
7	5	2	8.00	4	2
8	5	3	8.90	4	1
9	6	3	9.20	4	1
10	7	3	10.00	5	2
11	8	3	10.90	5	2
12	8	4	11.24	5	2
13	9	4	11.46	5	2
14	10	4	12.00	6	1
15	10	5	12.67	6	2
16	11	5	12.90	6	2
17	12	5	13.24	6	2
18	13	5	13.46	6	2
19	14	5	14.00	7	1
20	15	5	14.67	7	3
21	16	5	14.90	7	3
22	16	6	15.12	7	3
23	17	6	15.24	7	3
24	18	6	15.46	7	3
25	19	6	16.00	8	2
26	19	7	16.46	8	3
27	20	7	16.67	8	3
28	21	7	16.90	8	3
29	22	7	17.12	8	3
30	23	7	17.24	8	3
31	24	7	17.46	8	3
32	24	8	17.84	8	3

Table 1: Optimal n-bit Adders, $1 \leq n \leq 32$.

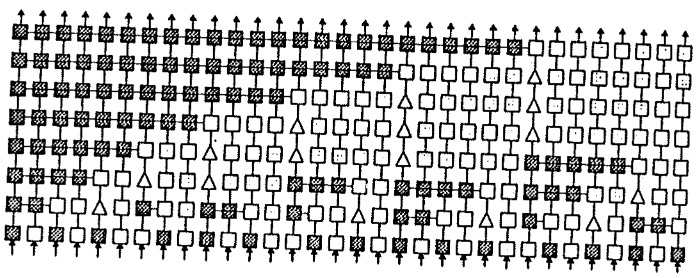


Fig.5.5 The Optimal 32-bit R(n) Adder

6. Discussion And Conclusions

To design a fast adder for a given VLSI technology, one should first implement black cells and decide on the particular layer(s) to be used for interconnects. Then, one can determine the *RC* values for the cells and interconnects, as well as the corresponding timing delay. Or for more accuracy, one could use a circuit simulator to obtain the exact timing data for the p and g subcells under various loads. These values determine the delay factors in the dynamic programming, which may be different from those given in Equation 5.1. Next, one should implement multi-stage drivers and tabulate their delays under various capacitive loads. The result should be substituted for Equation 5.2. Using this customized and more accurate model, Table 1 may then be recalculated using dynamic programming. The resulting optima will, in general, be different from the ones we calculated.

Our dynamic programming formulation does not explicitly optimize circuit area. A modification to our program would calculate the fastest $R(n)$ adder for a given depth.

In this paper, we have discussed static CMOS circuits. We are presently trying to extend our analysis to alternative circuit design styles, such as dynamic circuits.

In conclusion, we have found the optimal circuit in the $R(n)$ family of parallel adders. It is an open question to find similar optima in other classes of adder configurations. This topic is under current research.

References

De84. Despain, Alvin M., "Notes on Computer Architecture for High Performance," in *New Computer Architectures*, ed. J. Tiberghien, p. Academic Press, London, 1984.

Ku78. Kuck, David J., *The Structure of Computers and Computation*, I, pp. 54-56, 1978.

Br82. Brent, R. P. and Kung, H. T., "A Regular Layout for Parallel Adders," *IEEE Trans. on Computers*, vol. C-31, no. 3, March 1982.

La80. Ladner, Richard E. and Fischer, Michael J., "Parallel Prefix Computation," *JACM*, vol. 27, no. 4, pp. 831-838, October, 1980.

Me80. Mead, Carver and Conway, Lynn, *Introduction to VLSI Systems*, Addison-Wesley Series in Computer Science, 1980.