7 Variations in Fast Adders

"The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible."

CHARLES BABBAGE, ON THE MATHEMATICAL POWERS OF THE CALCULATING ENGINE

The carry-lookahead method of Chapter 6 represents the most widely used design for high-speed adders in modern computers. Certain alternative designs, however, either are quite competitive with carry-lookahead adders or offer advantages with particular hardware realizations or technology constraints. The most important of these alternative designs, and various hybrid combinations, are discussed in this chapter.

7.1 SIMPLE CARRY-SKIP ADDERS

Consider a 4-bit group or block in a ripple-carry adder, from stage i to stage $i+3$, where i is a multiple of 4 (Fig. 7.1a). A carry into stage i propagates through this group of 4 bits if and only if it propagates through all four of its stages. Thus, a *group propagate* signal is defined as $p_{[i,i+3]} = p_i p_{i+1}\, p_{i+2}\, p_{i+3}$, which is computable from individual propagate signals by a single four-input AND gate. To speed up carry propagation, one can establish bypass or skip paths around 4-bit blocks, as shown in Fig. 7.1b.

Let us assume that the delay of the skip multiplexer (mux) is equal to carry-propagation delay through one-bit position. Then, the worst-case propagation delay

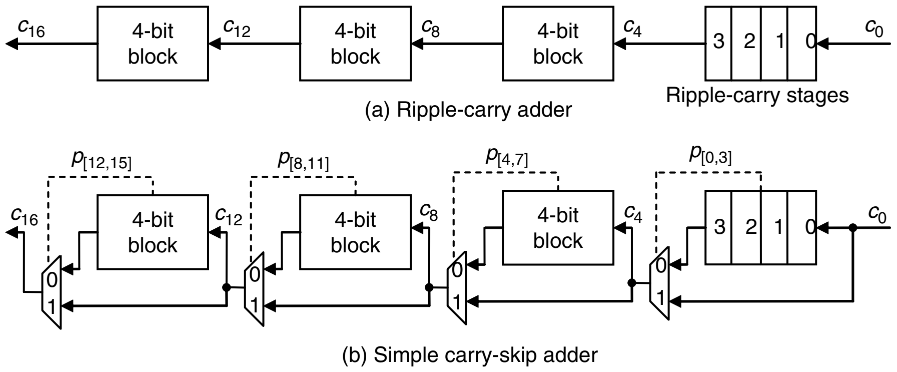

Figure 7.1 Converting a 16-bit ripple-carry adder to a simple carry-skip adder with 4-bit skip blocks.

through the carry-skip adder of Fig. 7.1b corresponds to a carry that is generated in stage 0, ripples through stages 1–3, goes through the multiplexer, skips the middle two groups, and ripples in the last group from stage 12 to stage 15. This leads to 9 stages of propagation (18 gate levels) compared to 16 stages (32 gate levels) for a 16-bit ripple-carry adder.

Generalizing from the preceding example, the worst-case carry-propagation delay in a k-bit carry-skip adder with fixed block width b, assuming that one stage of ripple has the same delay as one skip, can be derived:

$$\begin{aligned} T_{\text{fixed-skip-add}} &= \underset{\text{in block 0}}{(b-1)} + \underset{\text{mux}}{1} + \underset{\text{skips}}{(k/b-2)} + \underset{\text{in last block}}{(b-1)} \\ &\approx 2b + k/b - 3 \text{ stages} \end{aligned}$$

The optimal fixed block size can be derived by equating $dT_{\text{fixed-skip-add}}/db$ with 0:

$$\frac{dT_{\text{fixed-skip-add}}}{db} = 2 - k/b^2 = 0 \Rightarrow b^{\text{opt}} = \sqrt{k/2}$$

The adder delay with the optimal block size above is

$$T^{\text{opt}}_{\text{fixed-skip-add}} = 2\sqrt{k/2} + \frac{k}{\sqrt{k/2}} - 3 = 2\sqrt{2k} - 3$$

For example, to construct a 32-bit carry-skip adder with fixed-size blocks, we set $k = 32$ in the preceding equations to obtain $b^{\text{opt}} = 4$ bits and $T^{\text{opt}}_{\text{fixed-skip-add}} = 13$ stages (26 gate levels). By comparison, the propagation delay of a 32-bit ripple-carry adder is about 2.5 times as long.

Clearly, a carry that is generated in, or absorbed by, one of the inner blocks travels a shorter distance through the skip blocks. We can thus afford to allow more ripple stages for such a carry without increasing the overall adder delay. This leads to the idea of variable skip-block sizes.

Let there be t blocks of widths $b_0, b_1, \cdots, b_{t-1}$ going from right to left (Fig. 7.2). Consider the two carry paths (1) and (2) in Fig. 7.2, both starting in block 0, one ending

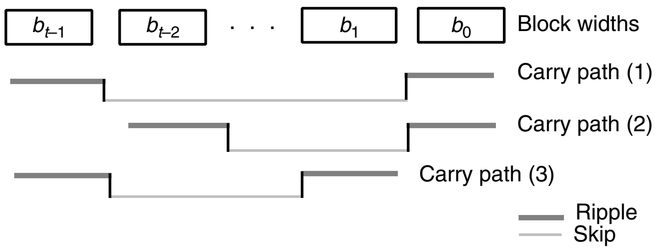

Figure 7.2 Carry-skip adder with variable-size blocks and three sample carry paths.

in block $t-1$ and the other in block $t-2$. Carry path (2) goes through one fewer skip than (1), so block $t-2$ can be 1 bit wider than block $t-1$ without increasing the total adder delay. Similarly, by comparing carry paths (1) and (3), we conclude that block 1 can be 1 bit wider than block 0. So, assuming for ease of analysis that $b_0 = b_{t-1} = b$ and that the number t of blocks is even, the optimal block widths are

$$b \qquad b+1 \qquad \cdots \qquad b+\frac{t}{2}-1 \qquad b+\frac{t}{2}-1 \qquad \cdots \qquad b+1 \qquad b$$

The first assumption ($b_0 = b_{t-1}$) is justified because the total delay is a function of $b_0 + b_{t-1}$ rather than their individual values and the second one (t even) does not affect the results significantly.

Based on the preceding block widths, the total number of bits in the t blocks is

$$2[b + (b+1) + \cdots + (b + t/2 - 1)] = t(b + t/4 - 1/2)$$

Equating the total above with k yields

$$b = k/t - t/4 + 1/2$$

The adder delay with the preceding assumptions is

$$\begin{aligned} T_{\text{var-skip-add}} &= 2(b-1) + 1 + t - 2 \\ &= \frac{2k}{t} + \frac{t}{2} - 2 \end{aligned}$$

The optimal number of blocks is thus obtained as follows:

$$\frac{dT_{\text{var-skip-add}}}{dt} = \frac{-2k}{t^2} + \frac{1}{2} = 0 \Rightarrow t^{\text{opt}} = 2\sqrt{k}$$

Note that the optimal number of blocks with variable-size blocks is $\sqrt{2}$ times that obtained with fixed-size blocks. Note also that with the optimal number of blocks, b becomes 1/2; thus we take it to be 1. The adder delay with t^{opt} blocks is

$$T^{\text{opt}}_{\text{var-skip-add}} \approx 2\sqrt{k} - 2$$

which is roughly a factor of $\sqrt{2}$ smaller than that obtained with optimal fixed-size skip-blocks.

The preceding analyses were based on a number of simplifying assumptions. For example, skip and ripple delays were assumed to be equal and ripple delay was assumed to be linearly proportional to the block width. These may not be true in practice. With complementary metal-oxide semiconductor implementation, for example, the ripple delay in a Manchester carry chain grows as the square of the block width. The analyses for obtaining the optimal fixed or variable block size carry-skip adder must be appropriately modified in such cases. A number of researchers have used various assumptions about technology-dependent parameters to deal with this optimization problem. Some of these variations are explored in the end-of-chapter problems.

7.2 MULTILEVEL CARRY-SKIP ADDERS

A (single-level) carry-skip adder of the types discussed in Section 7.1 can be represented schematically as in Fig. 7.3. In our subsequent discussions, we continue to assume that the ripple and skip delays are equal, although the analyses can be easily modified to account for different ripple and skip delays. We thus equate the carry-skip adder delay with the worst-case sum, over all possible carry paths, of the number of ripple stages and the number of skip stages.

Multilevel carry-skip adders are obtained if we allow a carry to skip over several blocks at once. Figure 7.4 depicts a two-level carry-skip adder in which second-level skip logic has been provided for the leftmost three blocks. The signal controlling this second-level skip logic is derived as the logical AND of the first-level skip signals. A carry that would need 3 time units to skip these three blocks in a single-level carry-skip adder can now do so in 1 time unit.

If the rightmost/leftmost block in a carry-skip adder is short, skipping it may not yield any advantage over allowing the carry to ripple through the block. In this case,

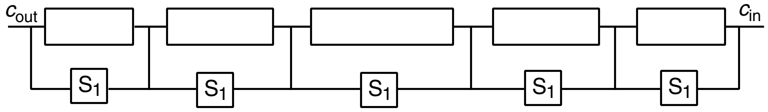

Figure 7.3 Schematic diagram of a one-level carry-skip adder.

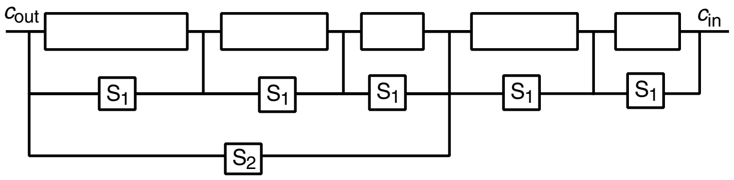

Figure 7.4 Example of a two-level carry-skip adder.

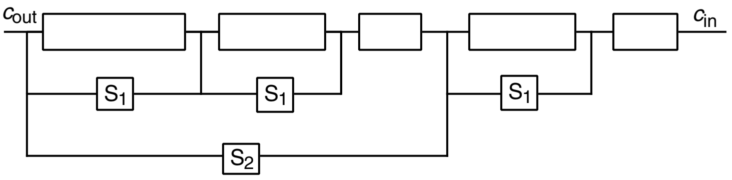

Figure 7.5 Two-level carry-skip adder optimized by removing the short-block skip circuits.

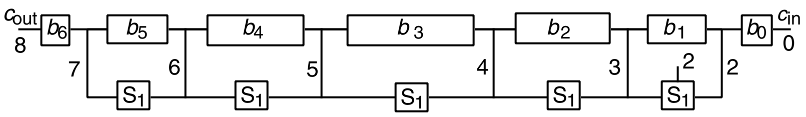

Figure 7.6 Timing constraints of a single-level carry-skip adder with a delay of 8 units.

the carry-skip adder of Fig. 7.4 can be simplified by removing such inefficient skip circuits. Figure 7.5 shows the resulting two-level carry-skip adder. With our simplifying assumption about ripple and skip delays being equal, the first-level skip circuit should be eliminated only for 1-bit, and possibly 2-bit, blocks (remember that generating the skip control signal also takes some time).

■ **EXAMPLE 7.1** Assume that each of the following operations takes 1 unit of time: generation of g_i and p_i signals, generation of a level-i skip signal from level-$(i-1)$ skip signals, ripple, skip, and computation of sum bit once the incoming carry is known. Build the widest possible single-level carry-skip adder with a total delay not exceeding 8 time units.

Let b_i be the width of block i. The numbers given on the adder diagram of Fig. 7.6 denote the time steps when the various signals stabilize, assuming that c_{in} is available at time 0. At the right end, block width is limited by the output timing requirement. For example, b_1 cannot be more than 3 bits if its output is to be available at time 3 (1 time unit is taken by g_i, p_i generation at the rightmost bit, plus 2 time units for propagation across the other 2 bits). Block 0 is an exception, because to accommodate c_{in}, its width must be reduced by 1 bit. At the left end, block width is limited by input timing. For example, b_4 cannot be more than 3 bits, given that its input becomes available at time 5 and the total adder delay is to be 8 units. Based on this analysis, the maximum possible adder width is $1+3+4+4+3+2+1 = 18$ bits.

■ **EXAMPLE 7.2** With the same assumptions as in Example 7.1, build the widest possible two-level carry-skip adder with a total delay not exceeding 8 time units.

We begin with an analysis of skip paths at level 2. In Fig. 7.7a, the notation $\{\beta, \alpha\}$ for a block means that the block's carry-out must become available no later than $T_{produce} = \beta$ and that the block's carry-in can take $T_{assimilate} = \alpha$ time units to propagate within the block without exceeding the overall time limit of 8 units. The remaining problem is to construct single-level carry-skip adders with the parameters $T_{\text{produce}} = \beta$ and $T_{\text{assimilate}} = \alpha$. Given the delay pair $\{\beta, \alpha\}$, the number of first-level blocks (subblocks) will be $\gamma = \min(\beta - 1, \alpha)$, with the width of the ith subblock, $0 \le i \le \gamma - 1$, given by $b_i = \min(\beta - \gamma + i + 1, \alpha - i)$; the only exception is subblock 0 in block A, which has 1 fewer bit (why?). So, the total width of such a block is $\sum_{i=0}^{\gamma-1} \min(\beta - \gamma + i + 1, \alpha - i)$. Table 7.1 summarizes our analyses for the second-level blocks A–F. Note that the second skip level has increased the adder width from 18 bits (in Example 7.1) to 30 bits. Figure 7.7b shows the resulting two-level carry-skip adder.

The preceding analyses of one- and two-level carry-skip adders are based on many simplifying assumptions. If these assumptions are relaxed, the problem may no longer lend itself to analytical solution. Chan et al. [Chan92] use dynamic programming to obtain optimal configurations of carry-skip adders for which the various worst-case

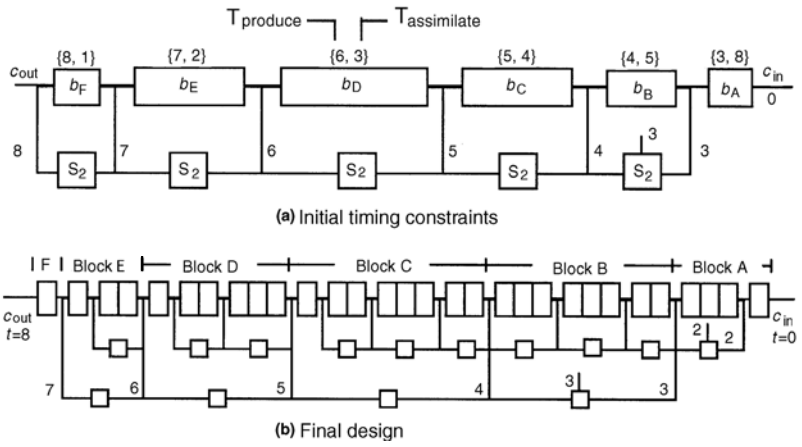

Figure 7.7 Two-level carry-skip adder with a delay of 8 units.

Table 7.1 Second-level constraints T_{produce} and $T_{\text{assimilate}}$, with associated subblock and block widths, in a two-level carry-skip adder with a total delay of 8 time units (Fig. 7.7)

Block	T_{produce}	$T_{\text{assimilate}}$	Number of subblocks	Subblock widths (bits)	Block width (bits)
A	3	8	2	1, 3	4
B	4	5	3	2, 3, 3	8
C	5	4	4	2, 3, 2, 1	8
D	6	3	3	3, 2, 1	6
E	7	2	2	2, 1	3
F	8	1	1	1	1

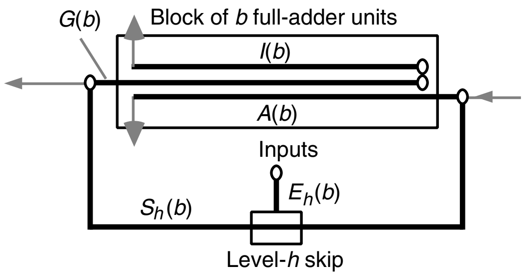

Figure 7.8 Generalized delay model for carry-skip adders.

delays in a block of b full-adder units are characterized by arbitrary given functions (Fig. 7.8). These delays include:

$I(b)$	Internal carry-propagate delay for the block
$G(b)$	Carry-generate delay for the block
$A(b)$	Carry-assimilate delay for the block

In addition, skip and enable delay functions, $S_h(b)$ and $E_h(b)$, are defined for each skip level h. In terms of this general model, our preceding analysis can be characterized as corresponding to $I(b) = b - 1, G(b) = b, A(b) = b, S_h(b) = 1$, and $E_h(b) = h + 1$. This is the model assumed by Turrini [Turr89]. Similar methods can be used to derive optimal block widths in variable-block carry-lookahead adders [Chan92].

7.3 CARRY-SELECT ADDERS

One of the earliest logarithmic time adder designs is based on the conditional-sum addition algorithm. In this scheme, blocks of bits are added in two ways: assuming an incoming carry of 0 or of 1, with the correct outputs selected later as the block's true carry-in becomes known. With each level of selection, the number of known output bits doubles, leading to a logarithmic number of levels and thus logarithmic time addition. Underlying the building of conditional-sum adders is the carry-select principle, which is described in this section.

A (single-level) carry-select adder is one that combines three $k/2$-bit adders of any design into a k-bit adder (Fig. 7.9). One $k/2$-bit adder is used to compute the lower half of the k-bit sum directly. Two $k/2$-bit adders are used to compute the upper $k/2$ bits of the sum and the carry-out under two different scenarios: $c_{k/2} = 0$ or $c_{k/2} = 1$. The correct values for the adder's carry-out signal and the sum bits in positions $k/2$ through $k - 1$ are selected when the value of $c_{k/2}$ becomes known. The delay of the resulting k-bit adder is two gate levels more than that of the $k/2$-bit adders that are used in its construction.

The following simple analysis demonstrates the cost-effectiveness of the carry-select method. Let us take the cost and delay of a single-bit 2-to-1 multiplexer as our units and assume that the cost and delay of a k-bit adder are $C_{\text{add}}(k)$ and $T_{\text{add}}(k)$, respectively.

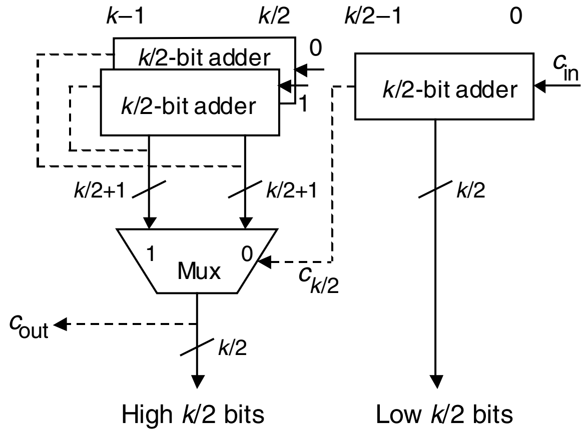

Figure 7.9 Carry-select adder for k-bit numbers built from three $k/2$-bit adders.

Then, the cost and delay of the carry-select adder of Fig. 7.9 are

$$C_{\text{select-add}}(k) = 3C_{\text{add}}(k/2) + k/2 + 1$$
$$T_{\text{select-add}}(k) = T_{\text{add}}(k/2) + 1$$

If we take the product of cost and delay as our measure of cost-effectiveness, the carry-select scheme of Fig. 7.9 is more cost-effective than the scheme used in synthesizing its component adders if and only if

$$[3C_{\text{add}}(k/2) + k/2 + 1][T_{\text{add}}(k/2) + 1] < C_{\text{add}}(k)T_{\text{add}}(k)$$

For ripple-carry adders, we have $C_{\text{add}}(k) = \alpha k$ and $T_{\text{add}}(k) = \tau k$. To simplify the analysis, assume $\tau = \alpha/2 > 1$. Then, it is easy to show that the carry-select method is more cost-effective than the ripple-carry scheme if $k > 16/(\alpha - 1)$. For $\alpha = 4$ and $\tau = 2$, say, the carry-select approach is almost always preferable to ripple-carry. Similar analyses can be carried out to compare the carry-select method against other addition schemes.

Note that in the preceding analysis, the use of three complete $k/2$-bit adders was assumed. With some adder types, the two $k/2$-bit adders at the left of Fig. 7.9 can share some hardware, thus leading to even greater cost-effectiveness. For example, if the component adders used are of the carry-lookahead variety, much of the carry network can be shared between the two adders computing the sum bits with $c_{k/2} = 0$ and $c_{k/2} = 1$ (how?).

Note that the carry-select method works just as well when the component adders have different widths. For example, Fig. 7.9 could have been drawn with one a-bit and two b-bit adders used to form an $(a + b)$-bit adder. Then c_a would be used to select the upper b bits of the sum through a $(b + 1)$-bit multiplexer. Unequal widths for the component adders is appropriate when the delay in deriving the selection signal c_a is different from that of the sum bits.

Figure 7.10 depicts how the carry-select idea can be carried one step further to obtain a two-level carry-select adder. Sum and carry-out bits are computed for each $k/4$-bit block

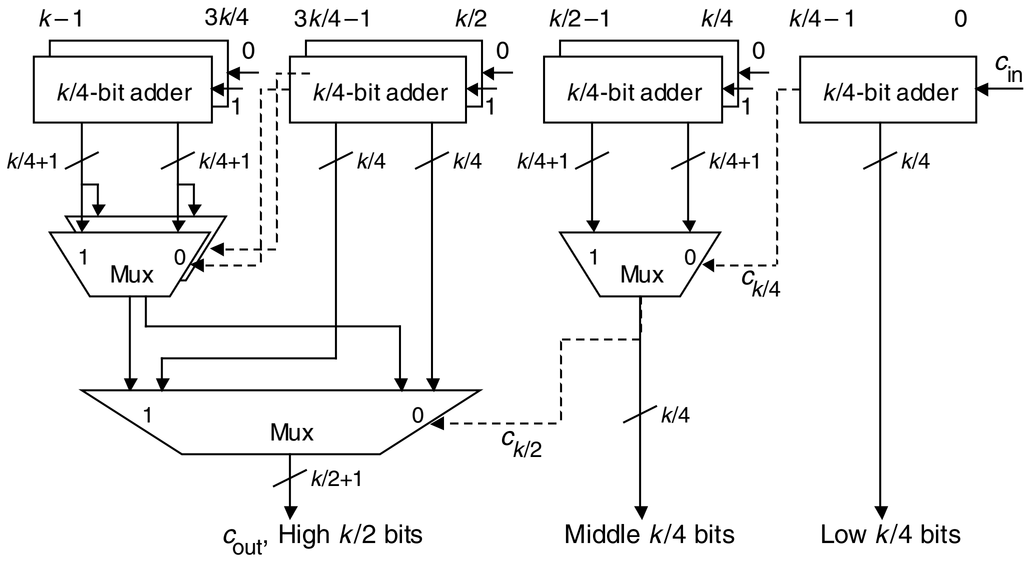

Figure 7.10 Two-level carry-select adder built of $k/4$-bit adders.

(except for the rightmost one) under two scenarios. The three first-level multiplexers, each of which is $k/4 + 1$ bits wide, merge the results of $k/4$-bit blocks into those of $k/2$-bit blocks. Note how the carry-out signals of the adders spanning bit positions $k/2$ through $3k/4 - 1$ are used to select the most-significant $k/4$ bits of the sum under the two scenarios of $c_{k/2} = 0$ or $c_{k/2} = 1$. At this stage, $k/2$ bits of the final sum are known. The second-level multiplexer, which is $k/2 + 1$ bits wide, is used to select appropriate values for the upper $k/2$ bits of the sum (positions $k/2$ through $k - 1$) and the adder's carry-out.

Comparing the two-level carry-select adder of Fig. 7.10 with a similar two-level carry-lookahead adder (Fig. 6.4, but with 2-bit, rather than 4-bit, lookahead carry generators), we note that the one-directional top-to-bottom data flow in Fig. 7.10 makes pipelining easier and more efficient. Of course, from Section 6.5 and the example in Fig. 6.13, we know that carry-lookahead adders can also be implemented to possess one-directional data flow. In such cases, comparison is somewhat more difficult, insofar as carry-select adders have a more complex upper structure (the small adders) and simpler lower structure (the multiplexers).

Which design comes out ahead for a given word width depends on the implementation technology, performance requirements, and other design constraints. Very often, the best choice is a hybrid combination of carry-select and carry-lookahead (see Section 7.5).

To understand the similarities between carry-select and carry-lookahead adders, consider a design similar to Fig. 7.9 in which only carry signals, rather than the final sum bits, are of interest. Clearly, once all carries are known, the sum bits can be generated rapidly by means of k XOR gates. Thus, the upper half of the new circuit derived from Fig. 7.9 will be responsible for generating two versions of the carries, rather than two versions of the sum bits. The carry c_{i+1} into position $i + 1 (i \geq k/2)$ is $g_{[k/2,i]}$ when

$c_{k/2} = 0$, and it is $t_{[k/2,i]}$ when $c_{k/2} = 1$. Recall that $t_{[k/2,i]} = g_{[k/2,i]} \vee p_{[k/2,i]}$. Thus, the pair of adders spanning positions $k/2$ through $k - 1$ in Fig. 7.9 become a parallel prefix carry network that uses the signal pair (g, t) instead of the usual (g, p). The entire structure of the modified design based on producing carries rather than sum bits is thus quite similar to the Ladner-Fischer carry network of Fig. 6.7. It even suffers from the same drawback of large fan-out for $c_{k/2}$, which is used as the selection signal for $k/2+1$ two-way multiplexers.

7.4 CONDITIONAL-SUM ADDER

The process that led to the two-level carry-select adder of Fig. 7.10 can be continued to derive a three-level k-bit adder built of $k/8$-bit adders, a four-level adder composed of $k/16$-bit adders, and so on. A logarithmic time conditional-sum adder results if we proceed to the extreme of having 1-bit adders at the very top. Thus, taking the cost and delay of a 1-bit 2-to-1 multiplexer as our units, the cost and delay of a conditional-sum adder are characterized by the following recurrences:

$$C(k) \approx 2C(k/2) + k + 2 \approx k(\log_2 k + 2) + kC(1)$$
$$T(k) = T(k/2) + 1 = \log_2 k + T(1)$$

where $C(1)$ and $T(1)$ are the cost and delay of the circuit of Fig. 7.11 used at the top to derive the sum and carry bits with a carry-in of 0 and 1. The term $k + 2$ in the first recurrence represents an upper bound on the number of single-bit 2-to-1 multiplexers needed for combining two $k/2$-bit adders into a k-bit adder.

The recurrence for cost is approximate, since for simplicity, we have ignored the fact that the right half of Fig. 7.10 is less complex than its left half. In other words, we have assumed that two parallel $(b + 1)$-bit multiplexers are needed to combine the outputs from b-bit adders, although in some cases, one is enough.

An exact analysis leads to a comparable count for the number of 1-bit multiplexers needed in a conditional-sum adder. Assuming that k is a power of 2, the required number

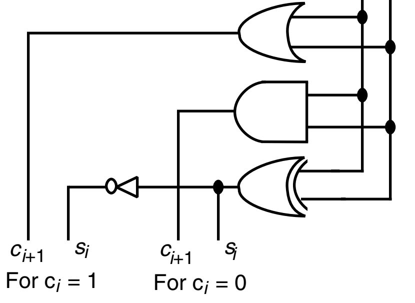

Figure 7.11 Top-level block for 1-bit addition in a conditional-sum adder.

Table 7.2 Conditional-sum addition of two 16-bit numbers: The width of the block for which the sum and carry bits are known doubles with each additional level, leading to an addition time that grows as the logarithm of the word width k

		x	0	0	1	0	0	1	1	0	1	1	1	0	1	0	1	0	
		y	0	1	0	0	1	0	1	1	0	1	0	1	1	1	0	1	
Block width	Block carry-in	Block sum and block carry-out	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	c_{in}
1	0	s	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	1	
		c	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
	1	s	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0		
		c	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1		
2	0	s	0	1	1	0	1	1	0	1	0	0	1	1	0	1	1	1	
		c	0		0		0		1		1		0		1		0		
	1	s	1	0	1	1	0	0	1	0	0	1	0	0	1	0			
		c	0		0		1		1		1		1		1				
4	0	s	0	1	1	0	0	0	0	1	0	0	1	1	0	1	1	1	
		c	0				1				1				1				
	1	s	0	1	1	1	0	0	1	0	0	1	0	0					
		c	0				1				1								
8	0	s	0	1	1	1	0	0	0	1	0	1	0	0	0	1	1	1	
		c	0								1								
	1	s	0	1	1	1	0	0	1	0									
		c	0																
16	0	s	0	1	1	1	0	0	1	0	0	1	0	0	0	1	1	1	
		c	0																
	1	s																	
		c																	
			c_{out}																

of multiplexers for a k-bit adder is

$$(k/2 + 1) + 3(k/4 + 1) + 7(k/8 + 1) + \cdots + (k - 1)2 = (k - 1)(\log_2 k + 1)$$

leading to an overall cost of $(k - 1)(\log_2 k + 1) + kC(1)$.

The conditional-sum algorithm can be visualized by the 16-bit addition example shown in Table 7.2.

Given that a conditional-sum adder is actually a $(\log_2 k)$-level carry-select adder, the comparisons and trade-offs between carry-select adders and carry-lookahead adders, as discussed at the end of Section 7.3, are relevant here as well.

7.5 HYBRID DESIGNS AND OPTIMIZATIONS

Hybrid adders are obtained by combining elements of two or more “pure” design methods to obtain adders with higher performance, greater cost-effectiveness, lower power consumption, and so on. Since any two or more pure design methods can be combined in a variety of ways, the space of possible designs for hybrid adders is immense. This leads to a great deal of flexibility in matching the design to given requirements and constraints. It also makes the designer’s search for an optimal design nontrivial. In this section, we review several possible hybrid adders as representative examples.

The one- and two-level carry-select adders of Figs. 7.9 and 7.10 are essentially hybrid adders, since the top-level $k/2$- or $k/4$-bit adders can be of any type. In fact, a common use for the carry-select scheme is in building fast adders whose width would lead to inefficient implementations with certain pure designs. For example, when 4-bit lookahead carry blocks are used, both 16-bit and 64-bit carry-lookahead adders can be synthesized quite efficiently (Fig. 6.4). A 32-bit adder, on the other hand, would require two levels of lookahead and is thus not any faster than the 64-bit adder. Using 16-bit carry-lookahead adders, plus a single carry-select level to double the width, is likely to lead to a faster 32-bit adder. The resulting adder has a hybrid carry-select/carry-lookahead design.

The reverse combination (viz., hybrid carry-lookahead/carry-select) is also possible and is in fact used quite widely. An example hybrid carry-lookahead/carry-select adder is depicted in Fig. 7.12. The small adder blocks, shown in pairs, may be based on Manchester carry chains that supply the required g and p signals to the lookahead carry generator and compute the final intermediate carries as well as the sum bits once the block carry-in signals have become known.

A wider hybrid carry-lookahead/carry-select adder will likely have a multilevel carry-lookahead network rather than a single lookahead carry generator as depicted in Fig. 7.12. If the needed block g and p signals are produced quickly, the propagation of signals in the carry-lookahead network can be completely overlapped with carry propagation in the small carry-select adders. The carry-lookahead network of Fig. 6.13 was in fact developed for use in such a hybrid scheme, with 8-bit carry-select adders based on Manchester carry chains [Lync92]. The 8-bit adders complete their computation at about

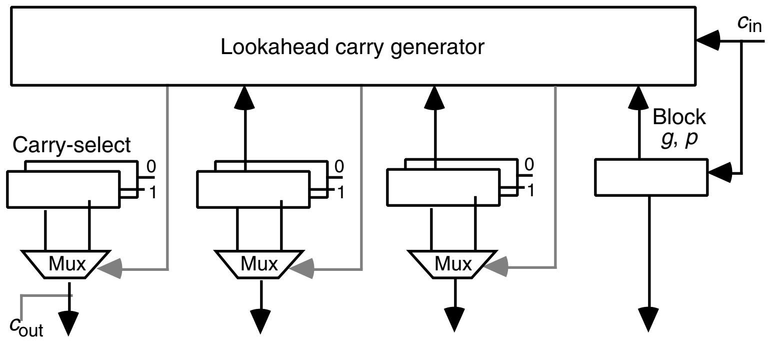

Figure 7.12 A hybrid carry-lookahead/carry-select adder.

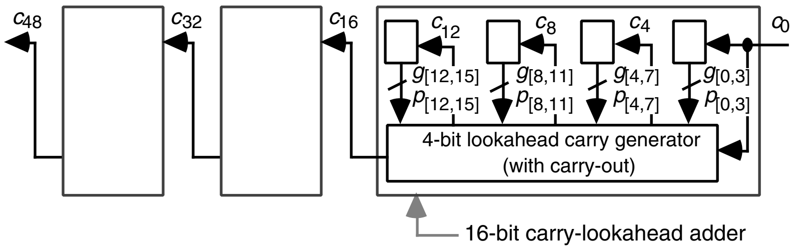
c48
c32
c16
c12
c8
c4
c0
g[12,15]
p[12,15]
g[8,11]
p[8,11]
g[4,7]
p[4,7]
g[0,3]
p[0,3]
4-bit lookahead carry generator (with carry-out)
16-bit carry-lookahead adder

Figure 7.13 Example 48-bit adder with hybrid ripple-carry/carry-lookahead design.

the same time that the carries $c_{24}, c_{32}, c_{40}, c_{48}$, and c_{56} become available (Fig. 6.13). Thus, the total adder delay is only two logic levels more than that of the carry-lookahead network.

Another interesting hybrid design is the ripple-carry/carry-lookahead adder, an example of which is depicted in Fig. 7.13. This hybrid design is somewhat slower than a pure carry-lookahead scheme, but its simplicity and greater modularity may compensate for this drawback. The analysis of cost and delay for this hybrid design relative to pure ripple-carry and carry-lookahead adders is left as an exercise, as is the development and analysis of the reverse carry-lookahead/ripple-carry hybrid combination.

Another hybrid adder example uses the hybrid carry-lookahead/conditional-sum combination. One drawback of the conditional-sum adder for wide words is the requirement of large fan-out for the signals controlling the multiplexers at the lower levels (Fig. 7.10). This problem can be alleviated by, for example, using conditional-sum addition in smaller blocks, forming the interblock carries through carry-lookahead. For detailed description of one such adder, used in Manchester University's MU5 computer, see [Omon94, pp. 104–111].

A hybrid adder may use more than two different schemes. For example, the 64-bit adder in the Compaq/DEC Alpha 21064 microprocessor is designed using four different methods [Dobb92]. At the lowest level, 8-bit Manchester carry chains are employed. The lower 32-bits of the sum are derived via carry lookahead at the second level, while conditional-sum addition is used to obtain two versions of the upper 32 bits. Carry-select is used to pick the correct version of the sum's upper 32 bits.

Clearly, it is possible to combine ideas from various designs in many different ways, giving rise to a steady stream of new implementations and theoretical proposals for the design of fast adders. Different combinations become attractive with particular technologies in view of their specific cost factors and fundamental constraints [Kant93]. In addition, application requirements, such as low power consumption, may shift the balance in favor of a particular hybrid design.

Just as optimal carry-skip adders have variable block widths, it is often possible to reduce the delay of other (pure or hybrid) adders by optimizing the block widths. For example, depending on the implementation technology, a carry-lookahead adder with fixed blocks may not yield the lowest possible delay [Niga95]. Again, the exact optimal configuration is highly technology-dependent. In fact, with modern very large-scale integration technology, gate count alone is no longer a meaningful measure of implementation cost. Designs that minimize or regularize the interconnection may actually be

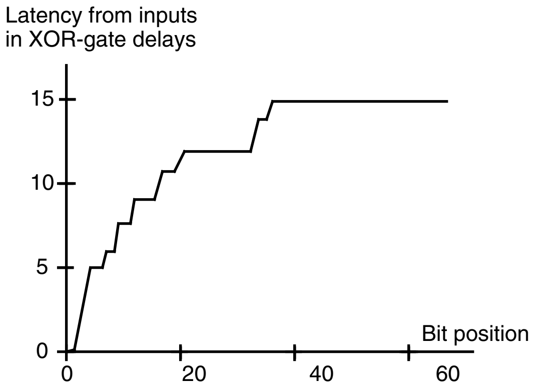

Figure 7.14 Example arrival times for operand bits in the final fast adder of a tree multiplier [Oklo96].

more cost-effective despite using more gates. The ultimate test of cost-effectiveness for a particular hybrid design or "optimal" configuration is its actual speed and cost when implemented with the target technology.

So far our discussion of adder delay has been based on the tacit assumption that all input digits are available at the outset, or at time 0, and that all output digits are computed and taken out after worst-case carries have propagated. The other extreme, where input/output digits arrive and leave serially, leads to very simple digit-serial adder designs. In between the two extremes, there are practical situations in which different arrival times are associated with the input digits or certain output digits must be produced earlier than others.

We will later see, for example, that in multiplying two binary numbers, the partial products are reduced to two binary numbers, which are then added in a fast two-operand adder to produce the final product. The individual bits of these two numbers become available at different times in view of the differing logic path depths from primary inputs. Figure 7.14 shows a typical example for the input arrival times at various bit positions of this final fast adder. This information can be used in optimizing the adder design [Oklo96]. A number of details and additional perspective can be found in [Liu03] and [Yeh00].

7.6 MODULAR TWO-OPERAND ADDERS

In some applications, with redundant number system arithmetic and cryptographic algorithms being the most notable examples, the sum of input operands x and y must be computed modulo a given constant m. In other words, we are interested in deriving $(x+y) \bmod m$, rather than $x+y$. An obvious approach would be to perform the required computation in two stages: (1) forming $x+y$, using any of the adder designs described thus far, and (2) reducing $x+y$ modulo m. Because the latency of the latter *modular reduction* step can be significant for an arbitrary value of m, direct methods similar to the carry-select approach have been used to combine the two steps.

Let us first focus on unsigned operands and certain special values of m that lead to simple modular addition. Clearly, when $m = 2^k$, the modulo-m sum is obtained by simply ignoring the carry-out. We also know that for $m = 2^k - 1$, using end-around carry allows us to reduce the sum modulo m by means of a conventional binary adder. A third special case pertains to $m = 2^k + 1$. Here, we need $k + 1$ bits to represent the $2^k + 1$ different residues modulo m. Because we can represent 2^{k+1} distinct values with $k + 1$ bits, different encodings of the $2^k + 1$ residues are possible. An interesting encoding that has been found to be quite efficient is the diminished-1 encoding. With this encoding, 0 is represented by asserting a special flag bit and setting the remaining k bits to 0, while a nonzero value x is represented by deasserting the flag bit and appending it with the representation of $x - 1$.

■ **EXAMPLE 7.3** Design a modulo-17 adder for unsigned integer operands in the range [0, 16], represented in the diminished-1 format.

Each of the input operands x and y consists of a 0-flag bit and a 4-bit binary magnitude that is one less than its true value, if nonzero. The design described in the following is based on the assumption that both operands are nonzero, as is their modulo-17 sum; that is, we assume $x \neq 0$, $y \neq 0$, $x + y \neq 17$. Augmenting the design to correctly handle these special cases is left as an exercise. The output should be $x + y - 1$, the diminished-1 representation of $x + y$, if $x + y \leq 17$, and it should be $x + y - 18$, the diminished-1 representation of $x + y - 17$, if $x + y \geq 18$. The desired results above can be rewritten as $(x - 1) + (y - 1) + 1$ if $(x - 1) + (y - 1) \leq 15$ and $(x - 1) + (y - 1) - 16$ if $(x - 1) + (y - 1) \geq 16$. These observations suggest that adding $x - 1$ and $y - 1$ with an inverted end-around carry (carry-in set to the complement of carry-out) will produce the desired result in either case. The inverted end-around carry arrangement will add 1 to the sum of $x - 1$ and $y - 1$ when $c_{\text{out}} = 0$, that is, $(x - 1) + (y - 1) \leq 15$, and will subtract 16 (by dropping the outgoing carry) when $c_{\text{out}} = 1$, corresponding to the condition $(x - 1) + (y - 1) \geq 16$.

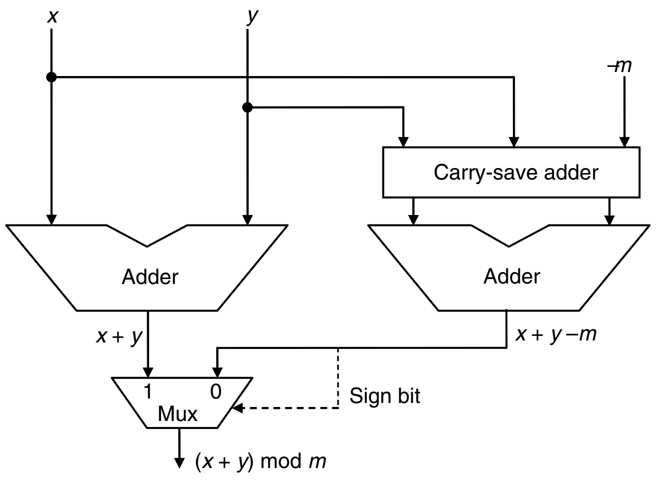

Figure 7.15 Fast modular addition.

For a general modulus m, we need to compare the sum $x + y$ to the modulus m, subtracting m from the computed sum if it equals or exceeds m. Because both the comparison and the ensuing subtraction require full carry propagation in the worst case, this approach would add a significant delay to the operation, which may be unacceptable. An alternative is to compute $x+y$ and $x+y-m$ in parallel and then use the sign of the latter value to decide whether $x + y \geq m$. If so, then $x + y - m$ is the correct result; otherwise, $x + y$ should be selected as the output. The resulting design, shown in Fig. 7.15, is quite fast, given that it only adds a full-adder level (the carry-save adder) and a multiplexer to the critical path of a conventional adder.

PROBLEMS

7.1 Optimal single-level carry-skip adders

a. Derive the optimal block width in a fixed-block carry-skip adder using the assumptions of Section 7.1, except that the carry production or assimilation delay in a block of width b is $b^2/2$ rather than b. Interpret the result.

b. Repeat part a with variable-width blocks. *Hint:* There will be several blocks of width b before the block width increases to $b + 1$.

7.2 Optimal two-level carry-skip adders

For the two-level carry-skip adder of Example 7.2, Section 7.2, verify the block sizes given in Table 7.1 and draw a complete diagram for a 24-bit adder derived by pruning the design of Fig. 7.7.

7.3 Optimal variable-block carry-skip adders

a. Build optimal single-level carry-skip adders for word widths $k = 24$ and $k = 80$.

b. Repeat part a for two-level carry-skip adders.

c. Repeat part a for three-level carry-skip adders.

7.4 Carry-skip adders with given building blocks

a. Assume the availability of 4-bit and 8-bit adders with delays of 3 and 5 ns, respectively, and of 0.5-ns logic gates. Each of our building block adders provides a “propagate” signal in addition to the normal sum and carry-out signals. Design an optimal single-level carry-skip adder for 64-bit unsigned integers.

b. Repeat part a for a two-level carry-skip adder.

c. Would we gain any advantage by going to three levels of skip for the adder of part a?

d. Outline a procedure for designing optimal single-level carry-skip adders from adders of widths $b_1 < b_2 < \cdots < b_h$ and delays $d_1 < d_2 < \cdots < d_h$, plus logic gates of delay δ.

7.5 Fixed-block, two-level carry-skip adders

Using the assumptions in our analysis of single-level carry-skip adders in Section 7.1, present an analysis for a two-level carry-skip adder in which the block widths b_1 and b_2 in levels 1 and 2, respectively, are fixed. Hence, assuming that b_1 and b_2 divide k, there are k/b_2 second-level blocks and k/b_1 first-level blocks, with each second-level block encompassing b_2/b_1 first-level blocks. Determine the optimal block widths b_1 and b_2. Note that because of the fixed block widths, skip logic must be included even for the rightmost block at each level.

7.6 Optimized multilevel carry-select adders

Consider the hierarchical synthesis of a k-bit multilevel carry-select adder where in each step of the process, an i-bit adder is subdivided into smaller j-bit and $(i - j)$-bit adders, so as to reduce the overall latency.

a. At what value of i does it not make sense to further subdivide the block?
b. When the width i of a block is odd, the two blocks derived from it will have to be of different widths. Is it better to make the right-hand or the left-hand block wider?
c. Evaluate the suggestion that, just as in carry-skip adders, blocks of different widths be used to optimize the design of carry-select adders.

7.7 Design of carry-select adders

Design 64-bit adders using ripple-carry blocks and 0, 1, 2, 3, or 4 levels of carry select.

a. Draw schematic diagrams for the three- and four-level carry-select adders, showing all components and selection signals.
b. Obtain the exact delay and cost for each design in terms of the number of gates and gate levels using two-input NAND gates throughout. Construct the ripple-carry blocks using the full-adder design derived from Figs. 5.2a and 5.1c.
c. Compare the five designs with regard to delay, cost, and the composite delay-cost figure of merit and discuss.

7.8 The conditional-sum addition algorithm

a. Modify Table 7.2 to correspond to the same addition, but with $c_{\text{in}} = 1$.
b. Using a tabular representation as in Table 7.2, show the steps of deriving the sum of 24-bit numbers 0001 0110 1100 1011 0100 1111 and 0010 0111 0000 0111 1011 0111 by means of the conditional-sum method.

7.9 Design of conditional-sum adders

Obtain the exact delay and cost for a 64-bit conditional-sum adder in terms of the number of gates and gate levels using two-input NAND gates throughout. For the topmost level, use the design given in Fig. 7.11.

7.10 Hybrid carry-completion adder

Suppose we want to design a carry-completion adder to take advantage of its good average-case delay but would like to improve on its O(k) worst-case delay. Discuss the suitability for this purpose of each of the following hybrid designs.

a. Completion-sensing blocks used in a single-level carry-skip arrangement.
b. Completion-sensing blocks used in a single-level carry-select arrangement.
c. Ripple-carry blocks with completion-sensing skip logic (separate skip circuits for 0 and 1 carries).

7.11 Hybrid ripple-carry/carry-lookahead adders

Consider the hybrid ripple-carry/carry-lookahead adder design depicted in Fig. 7.13.

a. Present a design for the modified lookahead carry generator circuit that also produces the block's carry-out (e.g., c_{16} in Fig. 7.13).
b. Develop an expression for the total delay of such an adder. State your assumptions.
c. Under what conditions, if any, is the resulting adder faster than an adder with pure carry-lookahead design?

7.12 Hybrid carry-lookahead/ripple-carry adders

Consider a hybrid adder based on ripple-carry blocks connected together with carry-lookahead logic (i.e., the reverse combination compared with the design in Fig. 7.13). Present an analysis for the delay of such an adder and state under what conditions, if any, the resulting design is preferable to a pure carry-lookahead adder or to the hybrid design of Fig. 7.13.

7.13 Hybrid carry-select/carry-lookahead adders

Show how carry-lookahead adders can be combined by a carry-select scheme to form a k-bit adder without duplicating the carry-lookahead logic in the upper $k/2$ bits.

7.14 Building fast adders from 4-bit adders

Assume the availability of fast 4-bit adders with one (two) gate delay(s) to bit (block) g and p signals and two gate delays to sum and carry-out once the bit g and p and block carry-in signals are known. Derive the cost and delay of each of the following 16-bit adders:

a. Four 4-bit adders cascaded through their carry-in and carry-out signals.
b. Single-level carry-skip design with 4-bit skip blocks.
c. Single-level carry-skip design with 8-bit skip blocks.
d. Single-level carry-select, with each of the 8-bit adders constructed by cascading two 4-bit adders.

7.15 Carry-lookahead versus hybrid adders

We want to design a 32-bit fast adder from standard building blocks such as 4-bit binary full adders, 4-bit lookahead carry circuits, and multiplexers. Compare the following adders with respect to cost and delay:

a. Adder designed with two levels of lookahead.
b. Carry-select adder built of three 16-bit single-level carry-lookahead adders.

7.16 Comparing fast two-operand adders

Assume the availability of 1-bit full adders; 1-bit, two-input multiplexers, and 4-bit lookahead carry circuits as unit-delay building blocks. Draw diagrams for, and compare the speeds and costs of, the following 16-bit adder designs.

a. Optimal variable-block carry-skip adder using a multiplexer for each skip circuit.
b. Single-level carry-select adder with 8-bit ripple-carry blocks.
c. Two-level carry-select adder with 4-bit ripple-carry blocks.
d. Hybrid carry-lookahead/carry-select adder with duplicated 4-bit ripple-carry blocks in which the carry-outs with $c_{\text{in}} = 0$ and $c_{\text{in}} = 1$ are used as the group g and p signals.

7.17 Optimal adders with input timing information

For each fast-adder type studied in Chapters 6 and 7, discuss how the availability of input bits at different times (Fig. 7.14) could be exploited to derive faster designs.

7.18 Fractional precision addition

a. We would like to design an adder that either adds two 32-bit numbers in their entirety or their lower and upper 16-bit halves independently. For each adder design discussed in Chapters 5–7, indicate how the design can be modified to allow such parallel half-precision arithmetic.
b. Propose a hybrid adder design that is particularly efficient for the design of part a.
c. Repeat part b, this time assuming two fractional precision modes: $4\times$ (8-bit) or $2\times$ (16-bit).

7.19 Design of fast adders

Assuming that 4-bit binary adders with full internal lookahead (3 gate delays overall) are available and that everything else must be built from NOT and two-input AND and OR gates, design complete 16-bit carry-skip and carry-lookahead adders and compare them with respect to speed and cost. Assume that the 4-bit adders supply c_{out} and block g and p signals.

7.20 Design of fast adders

Discuss the trade-offs involved in implementing a 64-bit fast adder in the following ways, assuming that the adder blocks used supply the block g and p signals.

a. Two-level carry-lookahead with 8-bit adder blocks and a second level 8-bit lookahead carry circuit.
b. Three-level carry-lookahead with 4-bit adder blocks and 4-bit lookahead carry circuits in two levels.
c. Two-level lookahead with 4-bit blocks and 8-bit lookahead carry circuit, followed by carry-select.
d. Two-level lookahead with 8-bit blocks and 4-bit lookahead carry circuit, followed by carry-select.

7.21 Carry-skip versus carry-select adders

Compare the costs and delays of 16-bit single-level carry-skip and carry-select adders, each constructed from 4-bit adder blocks and additional logic as needed. Which design would you consider more cost-effective and why? State all your assumptions clearly.

7.22 Comparing various adder designs

Compare the following 16-bit adders with respect to cost (gate count) and delay (gate levels on the critical path). Show your assumptions and reasoning and summarize the results in a 2×4 table.

a. Ripple-carry adder.
b. Completion sensing adder (use the average delay for comparisons).
c. Single-level carry-lookahead adder with 4-bit blocks.
d. Single-level carry-skip adder with 4-bit blocks.

7.23 Self-timed carry-skip adder

a. Apply the carry-skip idea to the self-timed adder of Fig. 5.9, illustrating the result by drawing a block diagram similar to that in Fig. 7.1 for a 16-bit carry-completion adder made of four 4-bit skip blocks.
b. Analyze the expected carry-completion time in the adder of part a, when blocks of fixed width b are used to build a k-bit self-timed adder.
c. Based on the result of part b, discuss the practicality of self-timed carry-skip adders.

7.24 Saturating adder

Study the problem of converting a carry-lookahead adder into a saturating adder, so that the added latency as a result of the saturation property is as small as possible.

7.25 Modulo-($2^k - 1$) fast adder

We know that any k-bit mod-2^k adder can be converted to a mod-($2^k - 1$) adder by connecting its carry-out to carry-in (end-around carry). This may slow down the adder because, in general, the critical path could be elongated. Show that a mod-($2^k - 1$) adder that is as fast as a mod-2^k adder can be designed by wrapping around the p (propagate) and g (generate) signals instead of carry-out [Kala00].

7.26 Combined binary/decimal fast adder

Design a 64-bit carry-lookahead adder that can be used as a 2's-complement binary adder or as a 16-digit decimal adder with binary-coded decimal encoding of each digit. *Hint:* Adding 6 to each digit of one operand allows you to share all the circuitry for carries associated with 4-bit blocks and beyond. Use carry-select within 4-bit blocks to complete the process.

7.27 Alternative formulation of carry-lookahead addition

Our discussion in Sections 6.4 and 6.5 was based on the ¢ operator that combines the carry signals for a pair of groups. In some implementation technologies, three signal pairs can be combined in a more complex circuit that may not be noticeably slower.

a. Define the three-input $¢_3$ operator in a way that it can be used in building carry networks.
b. Draw a structure similar to the Brent–Kung design of Fig. 6.9 using the new $¢_3$ operator.
c. Repeat part b for the Kogge–Stone design of Fig. 6.10.
d. Repeat part b assuming that only the carries $c_3, c_6, c_9, \ldots$ are needed because carry-select is used to determine the sum outputs in groups of 3 bits.
e. Repeat part c assuming that only the carries $c_3, c_6, c_9, \ldots$ are needed (as in part d).

7.28 Carry-skip addition

Assuming the use of ripple-carry blocks (1-unit ripple delay per bit position) to implement a 32-bit single-level carry-skip adder with 1-unit multiplexer delay for skipping, compare the following two sets of block widths in terms of the overall adder latency. Assume that forming the block propagate signal takes 1 unit of time for block widths up to 4, and 2 units of time for block widths 5–8. State all other assumptions. Block widths are listed from the most-significant end to the least-significant end.

2 3 4 5 5 5 4 3 2
3 4 5 8 5 4 3

7.29 Alternate design of carry-skip logic

The following circuit optimization may be attempted for the skip logic in Fig. 7.1. Take the multiplexer that produces $c_4 = \bar{p}_{[0,3]}d_3 \vee p_{[0,3]}c_0$, where d_3 is the carry-out of position 3. Removing $\bar{p}_{[0,3]}$ from this expression leaves $d_3 \vee p_{[0,3]}c_0$, which is logically equivalent to the original expression (why?), but simplifies the three-gate multiplexer to a two-gate circuit. Show that this simplification is ill-advised, in the sense that after the simplification, it would be possible for an addition that follows another one with all the internal carries equal to 1 to take as much time as ripple-carry addition.

REFERENCES AND FURTHER READINGS

[Bedr62] Bedrij, O. J., "Carry-Select Adder," *IRE Trans. Electronic Computers*, Vol. 11, pp. 340–346, 1962.

[Chan90] Chan, P. K., and M. D. F. Schlag, "Analysis and Design of CMOS Manchester Adders with Variable Carry Skip," *IEEE Trans. Computers*, Vol. 39, pp. 983–992, 1990.

[Chan92] Chan, P. K., M. D. F. Schlag, C. D. Thomborson, and V. G. Oklobdzija, "Delay Optimization of Carry-Skip Adders and Block Carry-Lookahead Adders Using Multidimensional Dynamic Programming," *IEEE Trans. Computers,* Vol. 41, No. 8, pp. 920–930, 1992.

[Dobb92] Dobberpuhl, D., et al., "A 200 MHz 64-b Dual-Issue CMOS Microprocessor," *IEEE J. Solid-State Circuits*, Vol. 27, No. 11, 1992.

[Guyo87] Guyot, A., and J.-M. Muller, "A Way to Build Efficient Carry-Skip Adders," *IEEE Trans. Computers*, Vol. 36, No. 10, pp. 1144–1152, 1987.

[Jabe09] Jaberipur, G. and B. Parhami, "Unified Approach to the Design of Modulo-($2^n \pm 1$) Adders Based on Signed-LSB Representation of Residues," *Proc. 19th IEEE Int'l Symp. Computer Arithmetic*, June 2009, pp. 57–64.

[Kala00] Kalampoukas, L., D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Trans. Computers*, Vol. 49, No. 7, pp. 673–680, 2000.

[Kant93] Kantabutra, V., "Designing Optimum One-Level Carry-Skip Adders," *IEEE Trans. Computers*, Vol. 42, No. 6, pp. 759–764, 1993.

[Lehm61] Lehman, M., and N. Burla, "Skip Techniques for High-Speed Carry Propagation in Binary Arithmetic Units," *IRE Trans. Electronic Computers*, Vol. 10, pp. 691–698, 1961.

[Liu03] Liu, J., S. Zhou, H. Zhu, and C.-K. Cheng, "An Algorithmic Approach for Generic Parallel Adders," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design*, November 2003, pp. 734–740.

[Lync92] Lynch, T., and E. Swartzlander, "A Spanning Tree Carry Lookahead Adder," *IEEE Trans. Computers*, Vol. 41, No. 8, pp. 931–939, 1992.

[Maje67] Majerski, S., "On Determination of Optimal Distributions of Carry Skip in Adders," *IEEE Trans. Electronic Computers*, Vol. 16, pp. 45–58, 1967.

[Niga95] Nigaglioni, R. H., and E. E. Swartzlander, "Variable Spanning Tree Adder," *Proc. Asilomar Conf. Signals, Systems, and Computers*, 1995, pp. 586–590, 1995.

[Oklo96] Oklobdzija, V. G., D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Trans. Computers*, Vol. 45, No. 3, pp. 294–306, 1996.

[Omon94] Omondi, A. R., *Computer Arithmetic Systems: Algorithms, Architecture and Implementation*, Prentice-Hall, 1994.

[Skla60] Sklansky, J., "Conditional-Sum Addition Logic," *IRE Trans. Electronic Computers*, Vol. 9, No. 2, pp. 226–231, 1960.

[Turr89] Turrini, S., "Optimal Group Distribution in Carry-Skip Adders," *Proc. 9th Symp. Computer Arithmetic*, pp. 96–103, 1989.

[Yeh00] Yeh, W.-C., and C.-W. Jen, "High-Speed Booth-Encoded Parallel Multiplier Design," *IEEE Trans. Computers*, Vol. 49, No. 7, pp. 692–701, 2000.