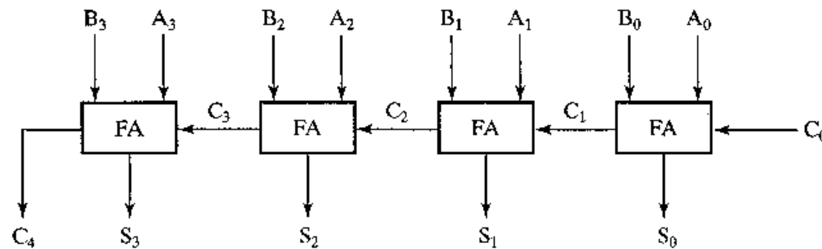
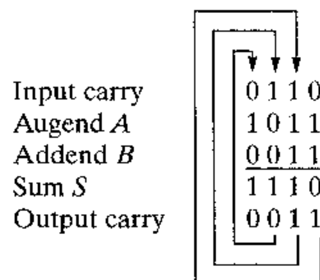


□ **FIGURE 5-5**
4-Bit Ripple Carry Adder

full adder. For example, consider the two binary numbers $A = 1011$ and $B = 0011$. Their sum, $S = 1110$, is formed with a 4-bit ripple carry adder as follows:



The input carry in the least significant position is 0. Each full adder receives the corresponding bits of A and B and the input carry and generates the sum bit for S and the output carry. The output carry in each position is the input carry of the next-higher-order position, as indicated by the blue lines.

The 4-bit adder is a typical example of a digital component that can be used as a building block. It can be used in many applications involving arithmetic operations. Observe that the design of this circuit by the usual method would require a truth table with 512 entries, since there are nine inputs to the circuit. By cascading the four instances of the known full adders, it is possible to obtain a simple and straightforward implementation without directly solving this larger problem. This is an example of the power of iterative circuits and circuit reuse in design.

Carry Lookahead Adder

The ripple carry adder, although simple in concept, has a long circuit delay due to the many gates in the carry path from the least significant bit to the most significant bit. For a typical design, the longest delay path through an n-bit ripple carry adder is $2n + 2$ gate delays. Thus, for a 16-bit ripple carry adder, the delay is 34 gate delays. This delay tends to be one of the largest in a typical computer design. Accordingly, we find an alternative design, the *carry lookahead adder*, attractive. This adder is a practical design with reduced delay at the price of more complex

hardware. The carry lookahead design can be obtained by a transformation of the ripple carry design in which the carry logic over fixed groups of bits of the adder is reduced to two-level logic. The transformation is shown for a 4-bit adder group in Figure 5-6.

First, we construct a new logic hierarchy, separating the parts of the full adders not involving the carry propagation path from those containing the path. We call the first part of each full adder a *partial full adder* (PFA). This separation is shown in Figure 5-6(a), which presents a diagram of a PFA and a diagram of four PFAs connected to the carry path. We have removed the OR gate and one of the AND gates from each of the full adders to form the ripple carry path.

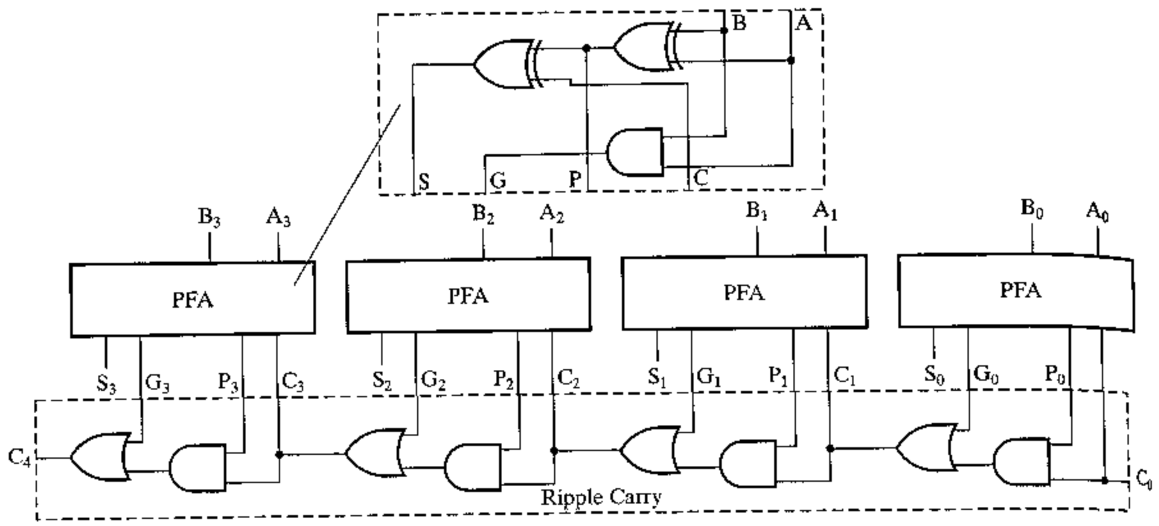
There are two outputs, P_i and G_i, from each PFA to the ripple carry path and one input C_i, the carry input, from the carry path to each PFA. The function $P_i = A_i \oplus B_i$ is called the *propagate* function. Whenever P_i is equal to 1, an incoming carry is propagated through the bit position from C_i to C_{i+1}. For P_i equal to 0, carry propagation through the bit position is blocked. The function $G_i = A_i \cdot B_i$ and is called the *generate* function. Whenever G_i is equal to 1, the carry output from the position is 1, regardless of the value of P_i, so a carry has been generated in the position. When G_i is 0, a carry is not generated, so that C_{i+1} is 0 if the carry propagated through the position from C_i is also 0. The generate and propagate functions correspond exactly to the half adder and are essential in controlling the values in the ripple carry path. Also, as in the full adder, the PFA generates the sum function by the exclusive-OR of the incoming carry C_i and the propagate function P_i.

The carry path remaining in the 4-bit ripple carry adder has a total of eight gates in cascade, so the circuit has a delay of eight gate delays. Since only AND and OR gates are involved in the carry path, ideally, the delay for each of the four carry signals produced, C_1 through C_4, would be just two gate delays. The basic carry lookahead circuit is simply a circuit in which functions C_1 through C_3 have a delay of only two gate delays. The implementation of C_4 is more complicated in order to allow the 4-bit carry lookahead adder to be extended to multiples of 4 bits, such as 16 bits. The 4-bit carry lookahead circuit is shown in Figure 5-6(b). It is designed to directly replace the ripple carry path in Figure 5-6(a). Since the logic generating C_1 is already two-level, it remains unchanged. The logic for C_2, however, has four levels. So to find the carry lookahead logic for C_2, we must reduce the logic to two levels. The equation for C_2 is found from Figure 5-6(a), and the distributive law is applied to obtain

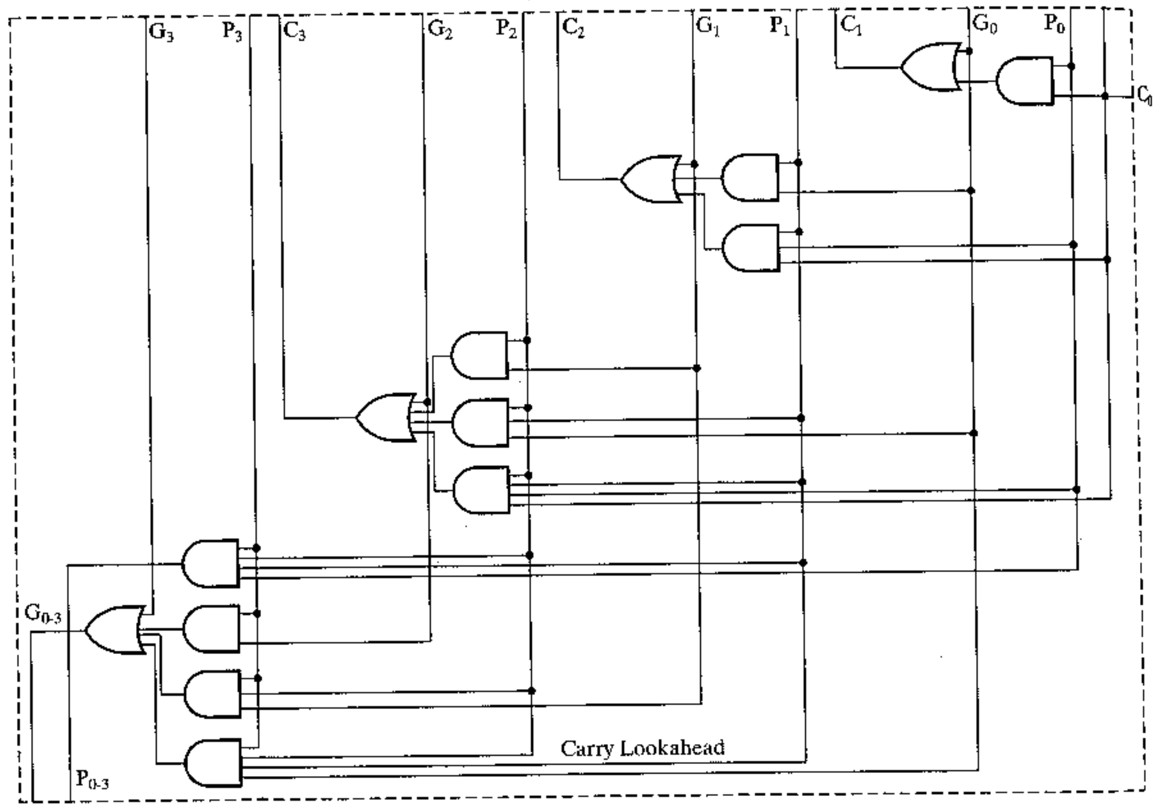
$$\begin{aligned} C_2 &= G_1 + P_1(G_0 + P_0C_0) \\ &= G_1 + P_1G_0 + P_1P_0C_0 \end{aligned}$$

This equation is implemented by the logic with output C_2 in Figure 5-6(b). We obtain the two-level logic for C_3 by finding its equation from the carry path in Figure 5-6(a) and applying the distributive law:

$$C_3 = G_2 + P_2(G_1 + P_1(G_0 + P_0C_0))$$



(a)



(b)

□ **FIGURE 5-6**
Development of a Carry Lookahead Adder

$$= G_2 + P_2(G_1 + P_1G_0 + P_0C_0)$$

$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

The two-level logic with output C_3 in Figure 5-6(b) implements this function.

We could implement C_4 using the same method. But some of the gates would have a fan-in of five, which may increase the delay. Also, we are interested in reusing this same circuit for higher numbered bits (e.g., 4 through 7, 8 through 11, and 12 through 15 of a 16-bit adder). For this adder, in positions 4, 8, and 12 we would like the carry to be produced as fast as possible without using excessive fan-in. Accordingly, we want to repeat the same carry lookahead trick for *4-bit groups* that we used to handle the 4 bits. This will allow us to reuse the carry lookahead circuit for each group of 4 bits, and also to use the same circuit for four 4-bit groups as if they were individual bits. So instead of generating C_4, we produce generate and propagate functions that apply to 4-bit groups instead of a single bit to act as the inputs for the group carry lookahead circuit. To propagate a carry from C_0 to C_4, we need to have all four of the propagate functions equal to 1, giving the *group propagate* function

$$P_{0-3} = P_3P_2P_1P_0$$

To represent the generation of a carry in positions 0, 1, 2, and 3, and its propagation to C_4, we need to consider the generation of a carry in each of the positions, as represented by G_0 through G_3, and the propagation of each of these four generated carries to position 4. This gives the *group generate* function

$$G_{0-3} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

The group propagate and group generate equations are implemented by the logic in the lower part of Figure 5-6(b). If there are only 4 bits in the adder, then the logic circuit used for C_1 can be used to generate C_4 from these two outputs. In a longer adder, a carry lookahead circuit identical to that in the figure, except for labeling, is placed at the second level to generate C_4, C_8, and C_{12}. This concept can be extended with more carry lookahead circuits in the second level and with one carry lookahead circuit in the third level to generate carries for positions 16, 32, and 48 in a 64-bit adder.

Assuming that an exclusive OR contributes 2 gate delays, the longest delay in the 4-bit carry lookahead adder is 6 gate delays, compared with 10 gate delays in the ripple carry adder. The improvement is very modest and perhaps not worth all the extra logic. But applying the carry lookahead circuit to a 16-bit adder using five copies in two levels of lookahead reduces the delay from 34 to just 10 gate delays, improving the performance of the adder by a factor of close to three. In a 64-bit adder, with the use of 21 carry lookahead circuits in three levels of lookahead, the delay is reduced from 130 gate delays to 14 gate delays, giving more than a factor of 8 in improved performance. In general, for the implementation we have shown, the delay of a carry lookahead adder designed for the best performance is $4L + 2$ gate delays, where L is the number of lookahead levels in the design.