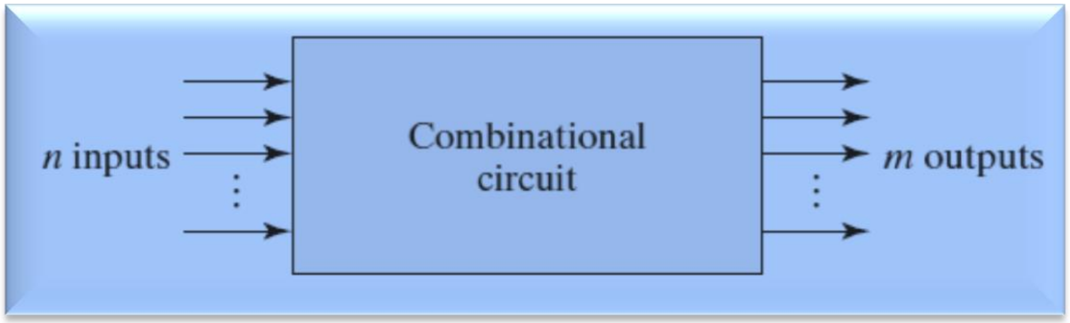
Lecture 8

Combinational Circuits

- A combinational circuit is a connected arrangement of logic gates with a set of inputs and outputs
- Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data
- In many applications, the source and destination are storage registers.
- If the registers are included with the combinational gates, then the total circuit must be considered to be a sequential circuit.
- A combinational circuit can be described by truth table showing binary relationship between n, input variables and m, output variables



Combinational Circuits

Half Adder

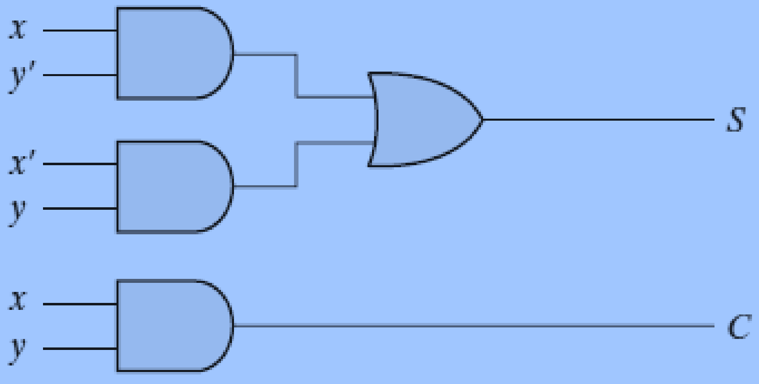
- ❖ A combinational circuit that performs the arithmetic addition of two bits is called half adder
- ❖ Two half adders make up a full adder that can perform addition of three bits
- ❖ The simplified Boolean functions for the two outputs can be obtained directly from the truth table.
- ❖ The simplified sum-of-products expressions are
- ❖ *Sum = x'y + xy'*
- ❖ *Carry = xy*

Combinational Circuits

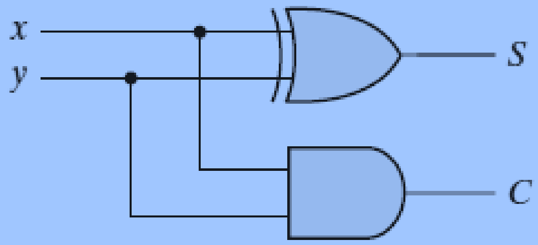
Half Adder

Table 4.3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



(a) $S = xy' + x'y$
$C = xy$

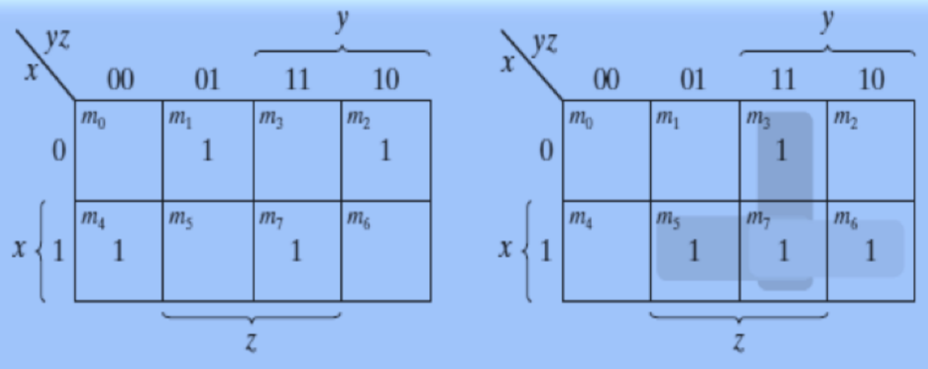


(b) $S = x \oplus y$
$C = xy$

Combinational Circuits

- **Full Adder**
- A combinational circuit that performs the arithmetic addition of three bits is called full adder
- Consists of three inputs and two outputs
- Two of the input variables, denoted by *x* and *y* , represent the two significant bits to be added.
- The third input, *z* , represents the carry from the previous lower significant position
- Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits.
- The two outputs are designated by the symbols *S* for sum and *C* for carry.
- The binary variable *S* gives the value of the least significant bit of the sum.
- The binary variable *C* gives the output carry formed by adding the input carry and the bits of the words

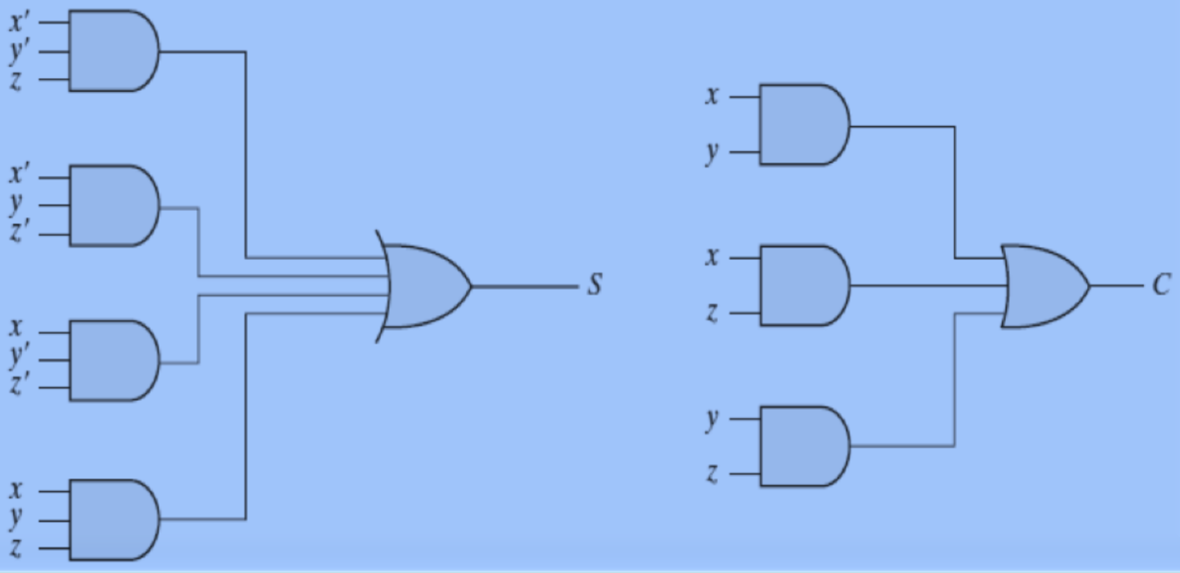
Combinational Circuits

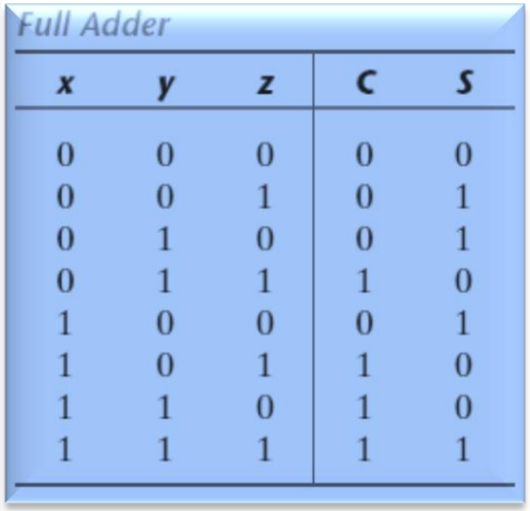


(a) $S = x'y'z + x'yz' + xy'z' + xyz$

(b) $C = xy + xz + yz$

FIGURE 4.6
K-Maps for full adder





Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Combinational Circuits

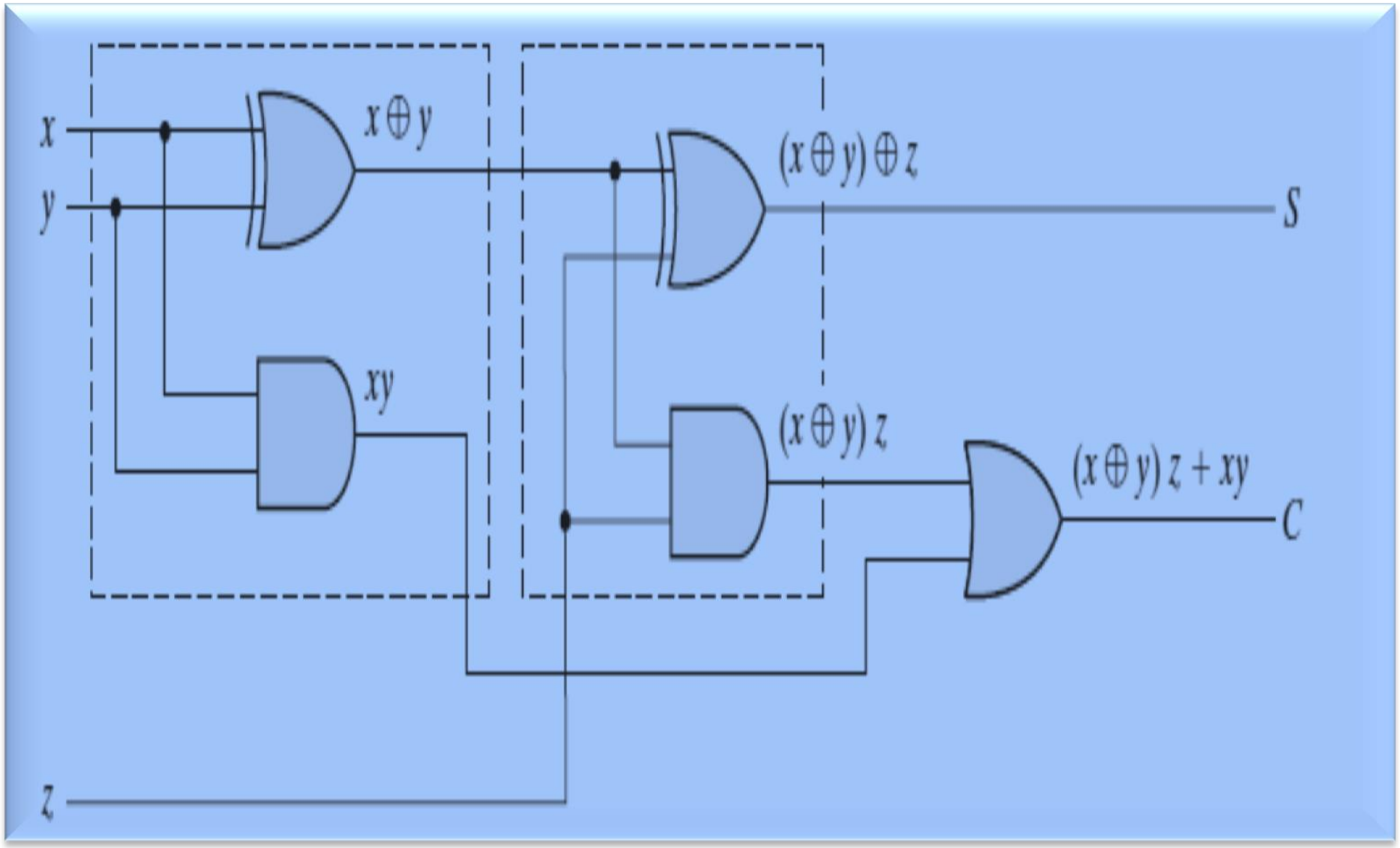
- It can also be implemented with two half adders and one OR gate
- The *S* output from the second half adder is the exclusive-OR of *z* and the output of the first half adder, giving

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

The carry output is

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

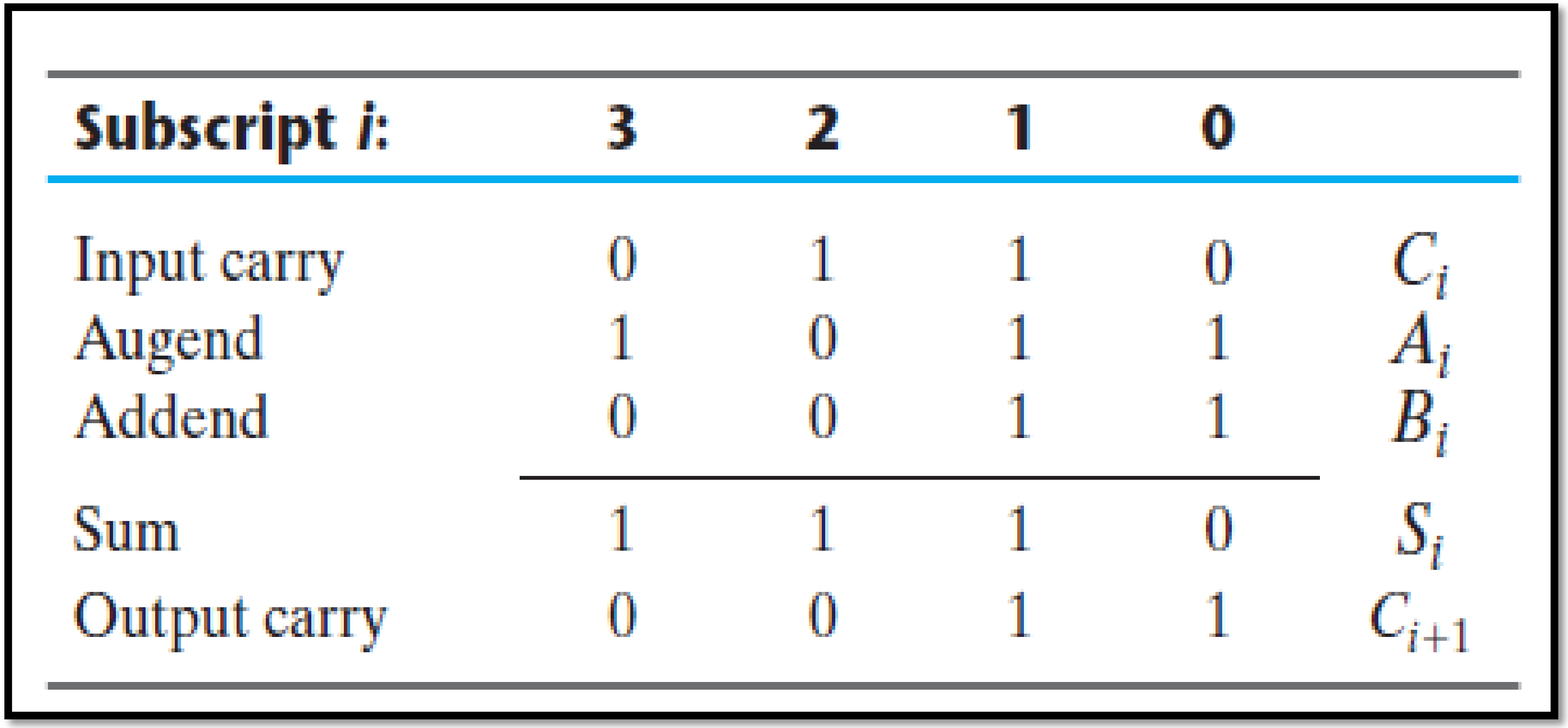
Full Adder



Binary Adder

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders.
- The input carry to the least significant position is fixed at 0.
- Figure 4.9 shows the interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.
- The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit.

Binary Adder



Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Binary Adder

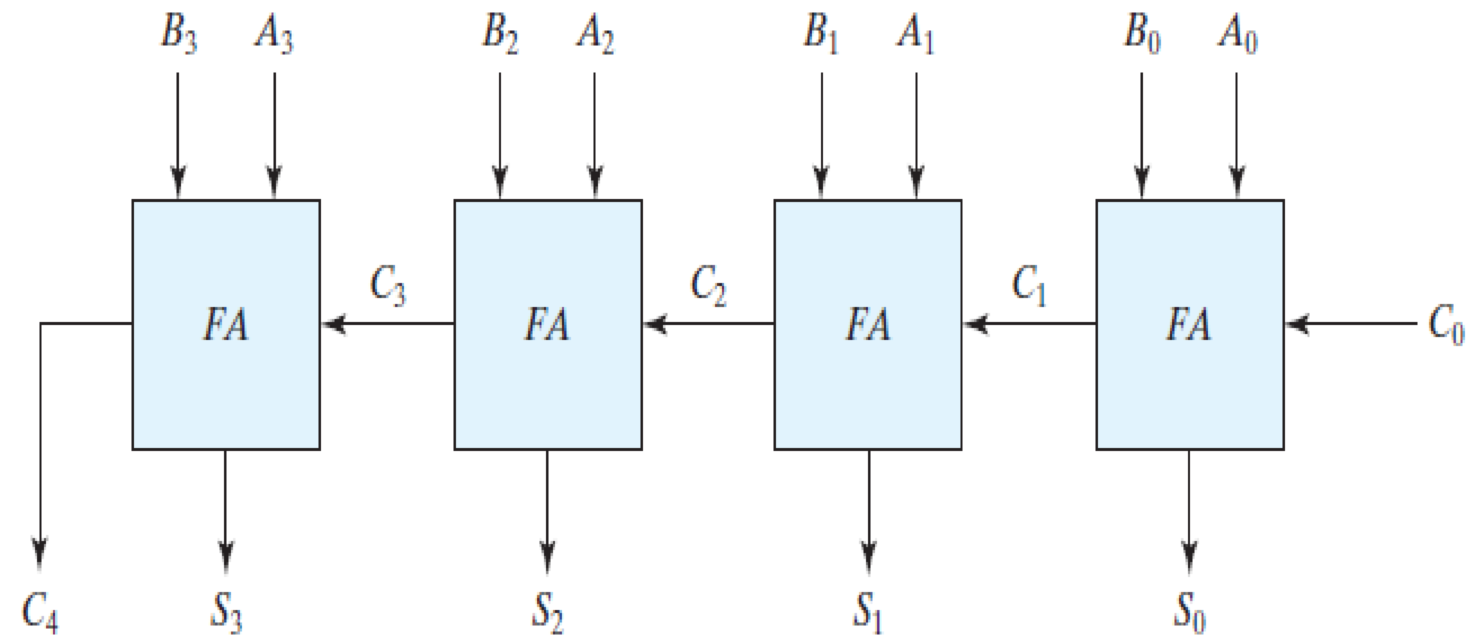


FIGURE 4.9
Four-bit adder

Carry Propagation

- The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time.
- As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals.
- The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.
- The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.
- Since each bit of the sum output depends on the value of the input carry, the value of *Si* at any given stage in the adder will be in its steady-state final value only after the input carry to that stage has been propagated.

Carry Propagation

- In this regard, consider output $S3$ in Fig. 4.9 .
- Inputs $A3$ and $B3$ are available as soon as input signals are applied to the adder.
- However, input carry $C3$ does not settle to its final value until $C2$ is available from the previous stage.
- Similarly, $C2$ has to wait for $C1$ and so on down to $C0$.
- Thus, only after the carry propagates and ripples through all stages will the last output $S3$ and carry $C4$ settle to their final correct value.
- The number of gate levels for the carry propagation can be found from the circuit of the full adder.
- The circuit is redrawn with different labels in Fig. 4.10 for convenience.
- The input and output variables use the subscript *i* to denote a typical stage of the adder.

Carry Propagation

- The signals at *Pi* and *Gi* settle to their steady-state values after they propagate through their respective gates.
- These two signals are common to all half adders and depend on only the input augend and addend bits.
- The signal from the input carry *Ci* to the output carry *Ci*+1 propagates through an AND gate and an OR gate, which constitute two gate levels.
- If there are four full adders in the adder, the output carry *C*4 would have 2 * 4 = 8 gate levels from *C*0 to *C*4.
- For an *n* -bit adder, there are 2*n* gate levels for the carry to propagate from input to output.

Carry Propagation

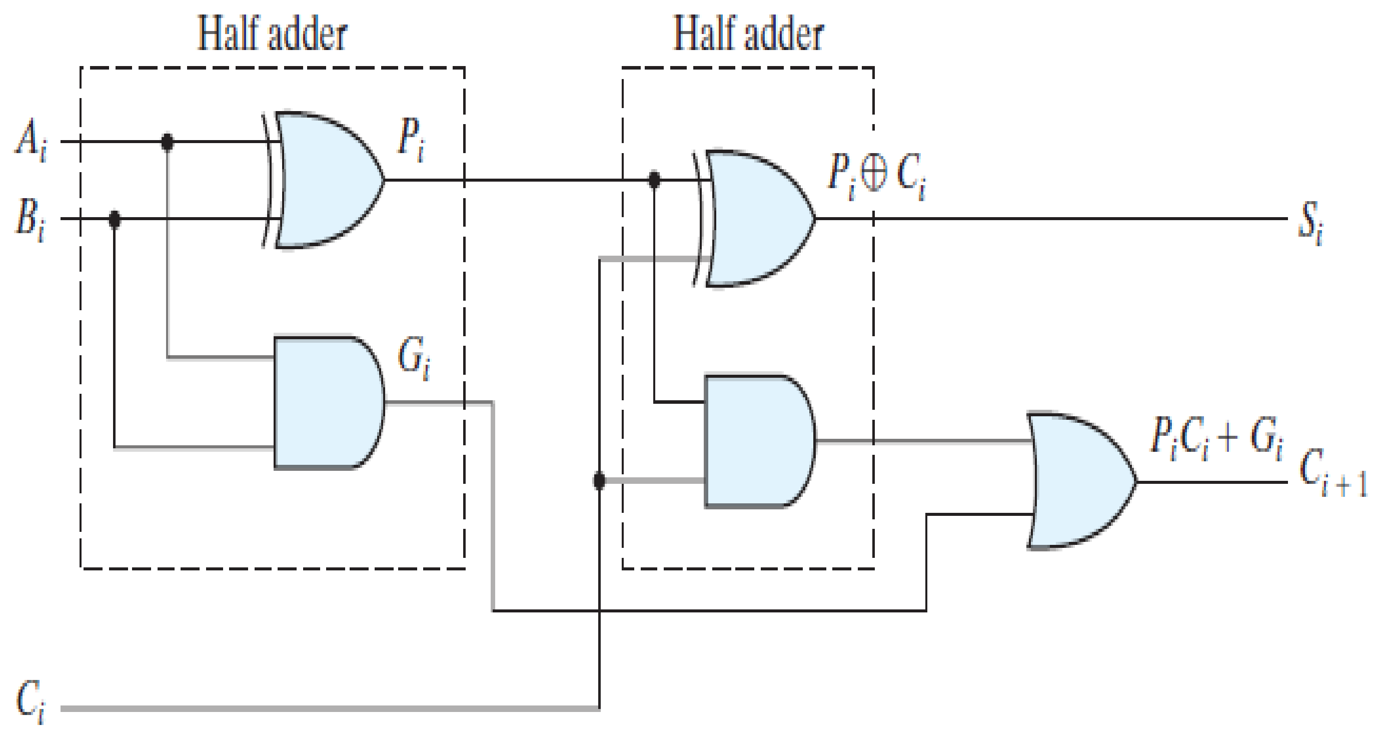


FIGURE 4.10
Full adder with *P* and *G* shown

Carry Propagation

- An obvious solution for reducing the carry propagation delay time is to employ faster gates with reduced delays.
- However, physical circuits have a limit to their capability.
- Another solution is to increase the complexity of the equipment in such a way that the carry delay time is reduced.
- There are several techniques for reducing the carry propagation time in a parallel adder.
- The most widely used technique employs the principle of *carry look-ahead logic* .
- Consider the circuit of the full adder shown in Fig. 4.10 .
- If we define two new binary variables

Carry Propagation

$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

- the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_iC_i$$

- *Gi* is called a *carry generate* , and it produces a carry of 1 when both *Ai* and *Bi* are 1, regardless of the input carry *Ci* and *Pi* is called a *carry propagate*.
- We now write the Boolean functions for the carry outputs of each stage and substitute the value of each *Ci* from the previous equations:
- *C*0 = input carry
- *C*1 = *G*0 + *P*0*C*0

Carry Propagation

- $C2 = G1 + P1C1 = G1 + P1(G0 + P0C0) = G1 + P1G0 + P1P0C0$
- $C3 = G2 + P2C2 = G2 + P2(G1 + P1G0 + P1P0C0)$
- $C3 = G2 + P2G1 + P2P1G0 + P2P1P0C0$
- The three Boolean functions for *C*1, *C*2, and *C*3 are implemented in the carry look-ahead generator shown in Fig. 4.11 .
- Note that this circuit can add in less time because *C*3 does not have to wait for *C*2 and *C*1 to propagate; in fact, *C*3 is propagated at the same time as *C*1 and *C*2.
- This gain in speed of operation is achieved at the expense of additional complexity (hardware).
- The construction of a four-bit adder with a carry look-ahead scheme is shown in Fig. 4.12 .

Carry Propagation

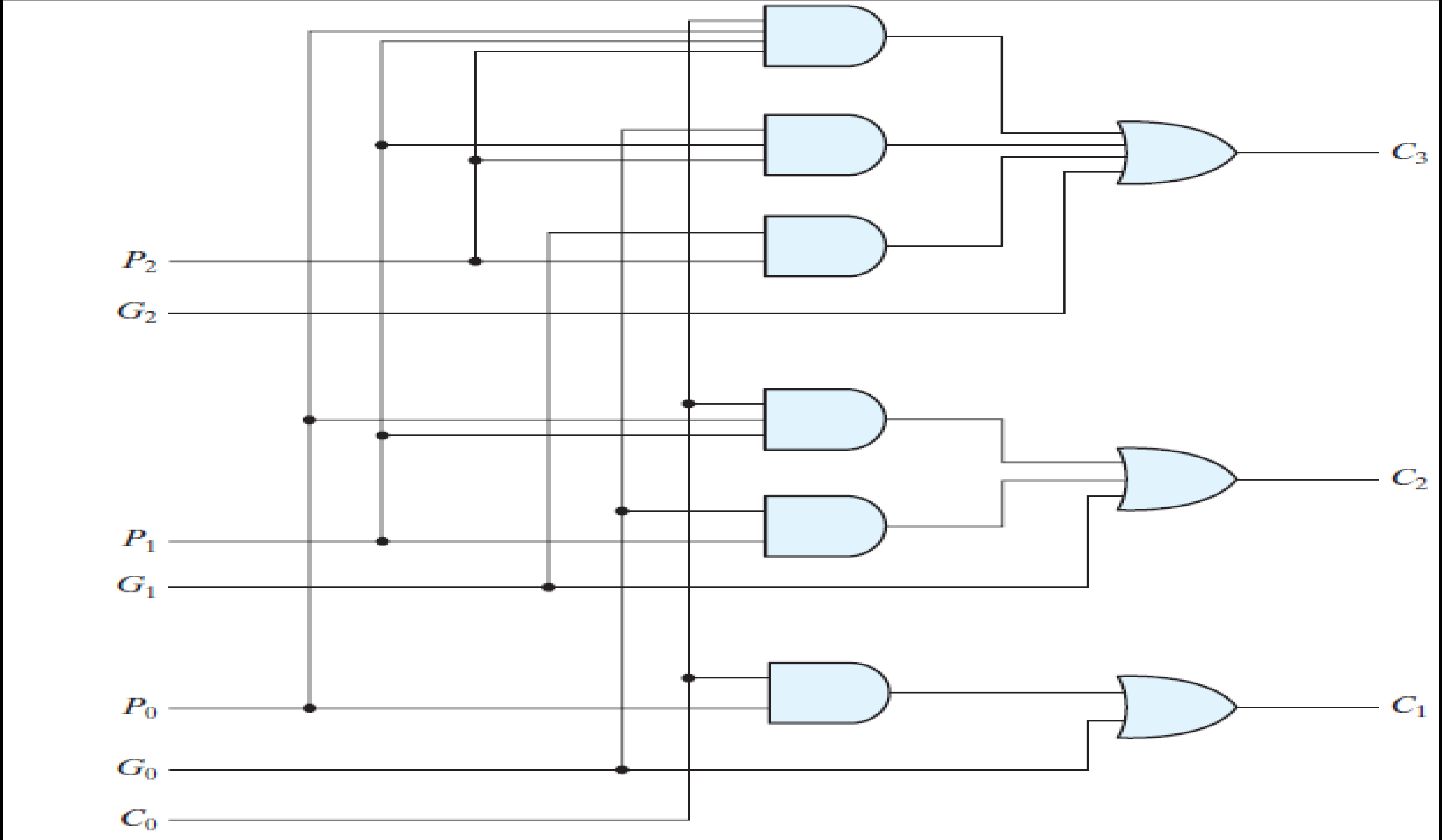


FIGURE 4.11
Logic diagram of carry lookahead generator

Carry Propagation

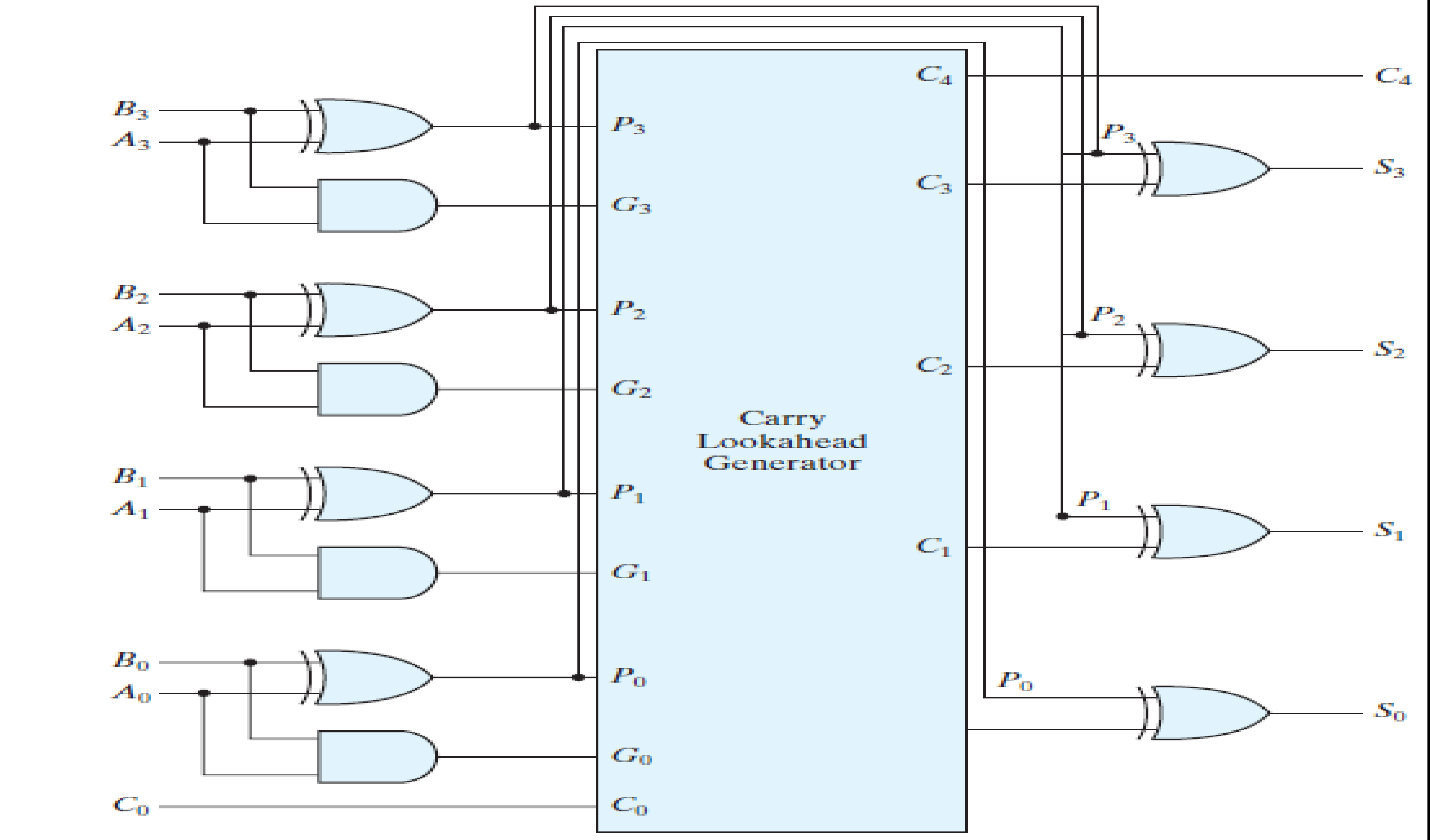


FIGURE 4.12
Four-bit adder with carry lookahead

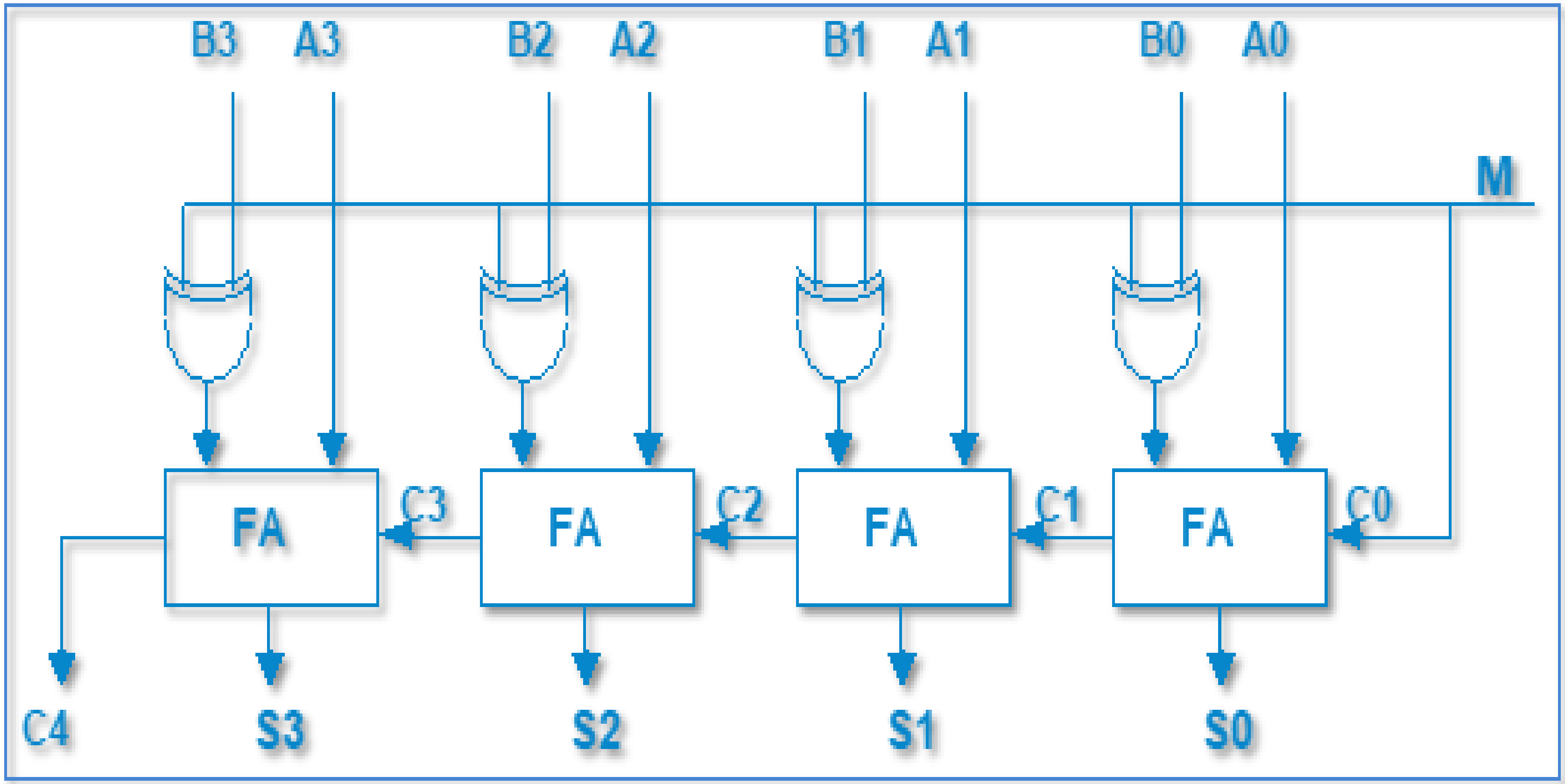
Binary Subtractor

- The subtraction of unsigned binary numbers can be done most conveniently by means of complements.
- Subtraction A - B can be done by taking the 2’s complement of B and adding it to A .
- The 2’s complement can be obtained by taking the 1’s complement and adding 1 to the least significant pair of bits.
- The 1’s complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.
- The input carry $C0$ must be equal to 1 when subtraction is performed.
- The operation thus performed becomes A , plus the 1’s complement of B , plus 1.
- This is equal to A plus the 2’s complement of B .

Binary Subtractor

- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder.
- The mode input *M* controls the operation.
- When *M* = 0, the circuit is an adder, and when *M* = 1, the circuit becomes a subtractor.
- Each exclusive-OR gate receives input *M* and one of the inputs of *B.*
- When *M* = 0, we have *B* ⊗ 0 = *B*.
- The full adders receive the value of *B* , the input carry is 0, and the circuit performs *A* plus *B* .
- When *M* = 1, we have *B* ⊗ 1 = '*B* and *C*0 = 1.
- The *B* inputs are all complemented and a 1 is added through the input carry.
- The circuit performs the operation *A* plus the 2's complement of *B* .

Binary Subtractor

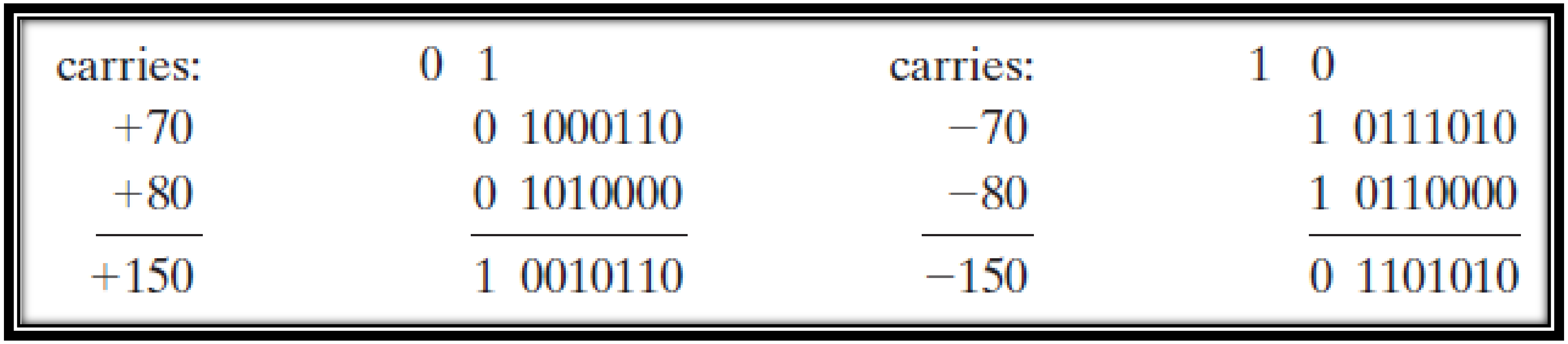


Binary Subtractor(Overflow)

- When two numbers with *n* digits each are added and the sum is a number occupying *n* + 1 digits, we say that an overflow occurred.
- The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned.
- When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position.
- In the case of signed numbers, two details are important: the leftmost bit always represents the sign, and negative numbers are in 2’s-complement form.
- When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

Binary Subtractor(Overflow)

- Two signed binary numbers, +70 and +80, are stored in two eight-bit registers.
- The range of numbers that each register can accommodate is from binary +127 to binary -128.
- Since the sum of the two numbers is +150, it exceeds the capacity of an eight-bit register.
- This is also true for -70 and -80.



carries:	0 1	carries:	1 0
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010

Binary Subtractor(Overflow)

- Note that the eight-bit result that should have been positive has a negative sign bit and the eight-bit result that should have been negative has a positive sign bit.
- If, however, the carry out of the sign bit position is taken as the sign bit of the result, then the nine-bit answer so obtained will be correct.
- But since the answer cannot be accommodated within eight bits, we say that an overflow has occurred.
- **An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.**
- If these two carries are not equal, an overflow has occurred.

Binary Subtractor(Overflow)

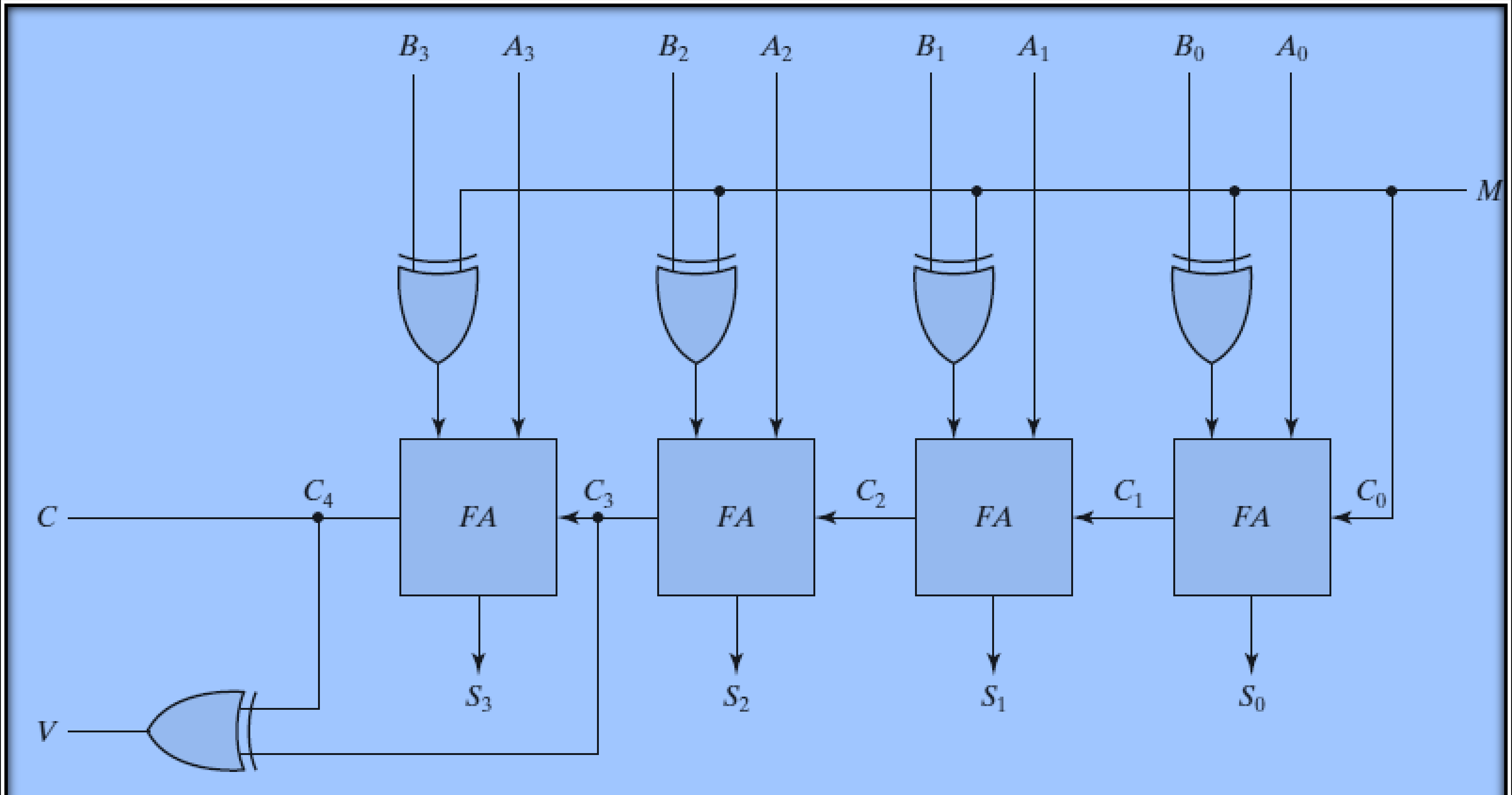


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

BINARY MULTIPLIER

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.
- Each such multiplication forms a partial product.
- Successive partial products are shifted one position to the left.
- The final product is obtained from the sum of the partial products.

BINARY MULTIPLIER

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\cline{3-4}
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

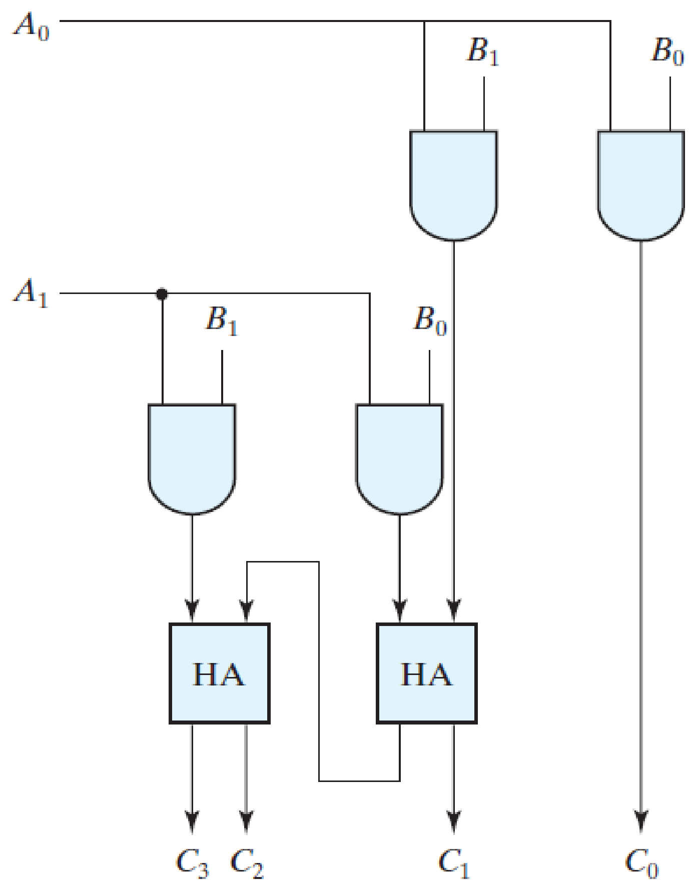


FIGURE 4.15
Two-bit by two-bit binary multiplier

BINARY MULTIPLIER

- The multiplicand bits are $B1$ and $B0$, the multiplier bits are $A1$ and $A0$, and the product is $C3C2C1C0$.
- The first partial product is formed by multiplying $B1B0$ by $A0$.
- The multiplication of two bits such as $A0$ and $B0$ produces a 1 if both bits are 1; otherwise, it produces a 0.
- This is identical to an AND operation.
- Therefore, the partial product can be implemented with AND gates as shown in the diagram.

BINARY MULTIPLIER

- The second partial product is formed by multiplying $B1B0$ by $A1$ and shifting one position to the left.
- The two partial products are added with two half-adder (HA) circuits.
- Usually, there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.
- Note that the least significant bit of the product does not have to go through an adder, since it is formed by the output of the first AND gate.

BINARY MULTIPLIER

- For *J* multiplier bits and *K* multiplicand bits, we need (*J* K)* AND gates and (*J* – 1) *K* -bit adders to produce a product of (*J* + *K*) bits.
- As a second example, consider a multiplier circuit that multiplies a binary number represented by four bits by a number represented by three bits.
- Let the multiplicand be represented by *B*3*B*2*B*1*B*0 and the multiplier by *A*2*A*1*A*0.
- Since *K* = 4 and *J* = 3, we need 12 AND gates and two 4-bit adders to produce a product of seven bits.

BINARY MULTIPLIER

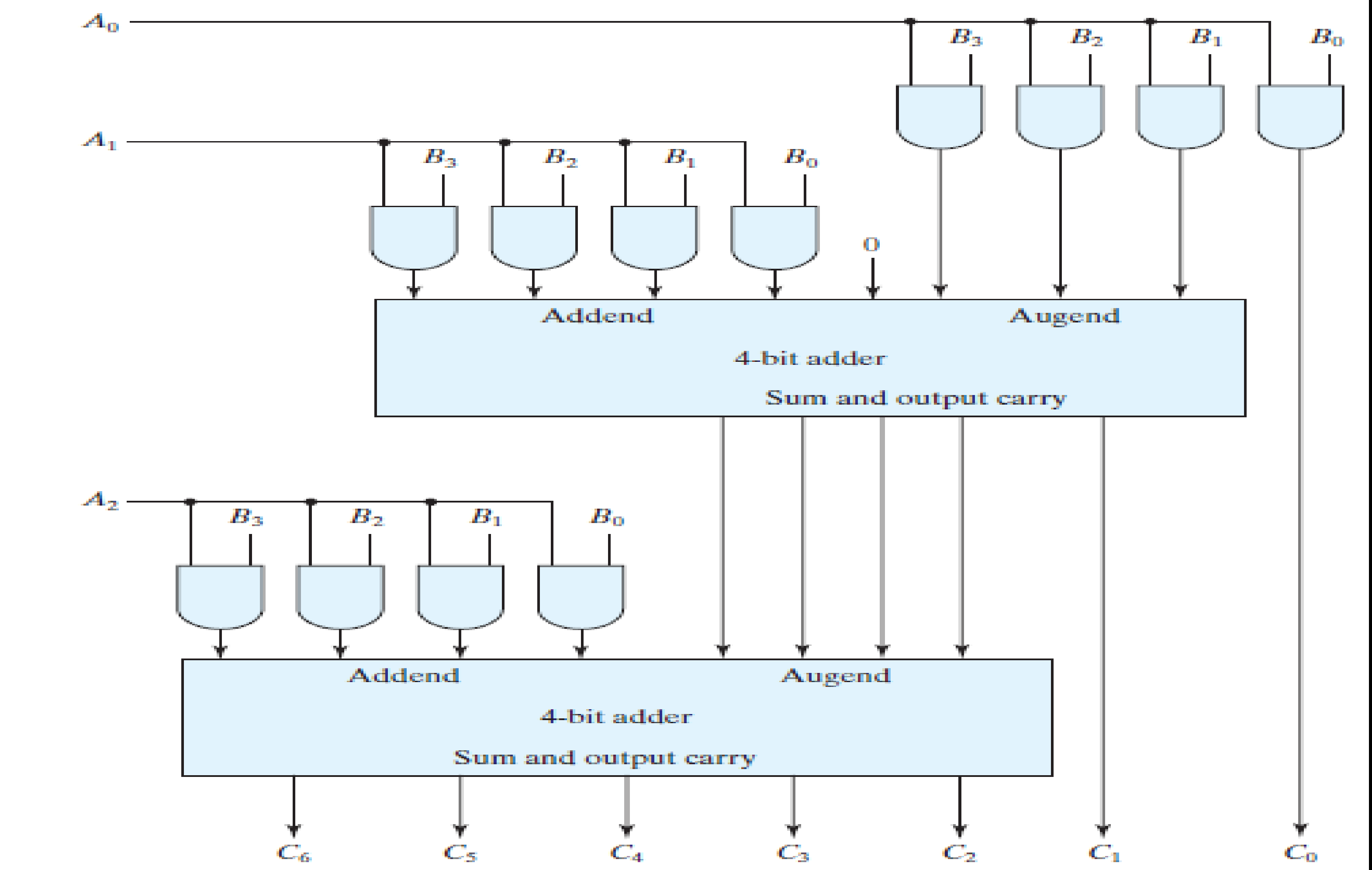


FIGURE 4.16
Four-bit by three-bit binary multiplier

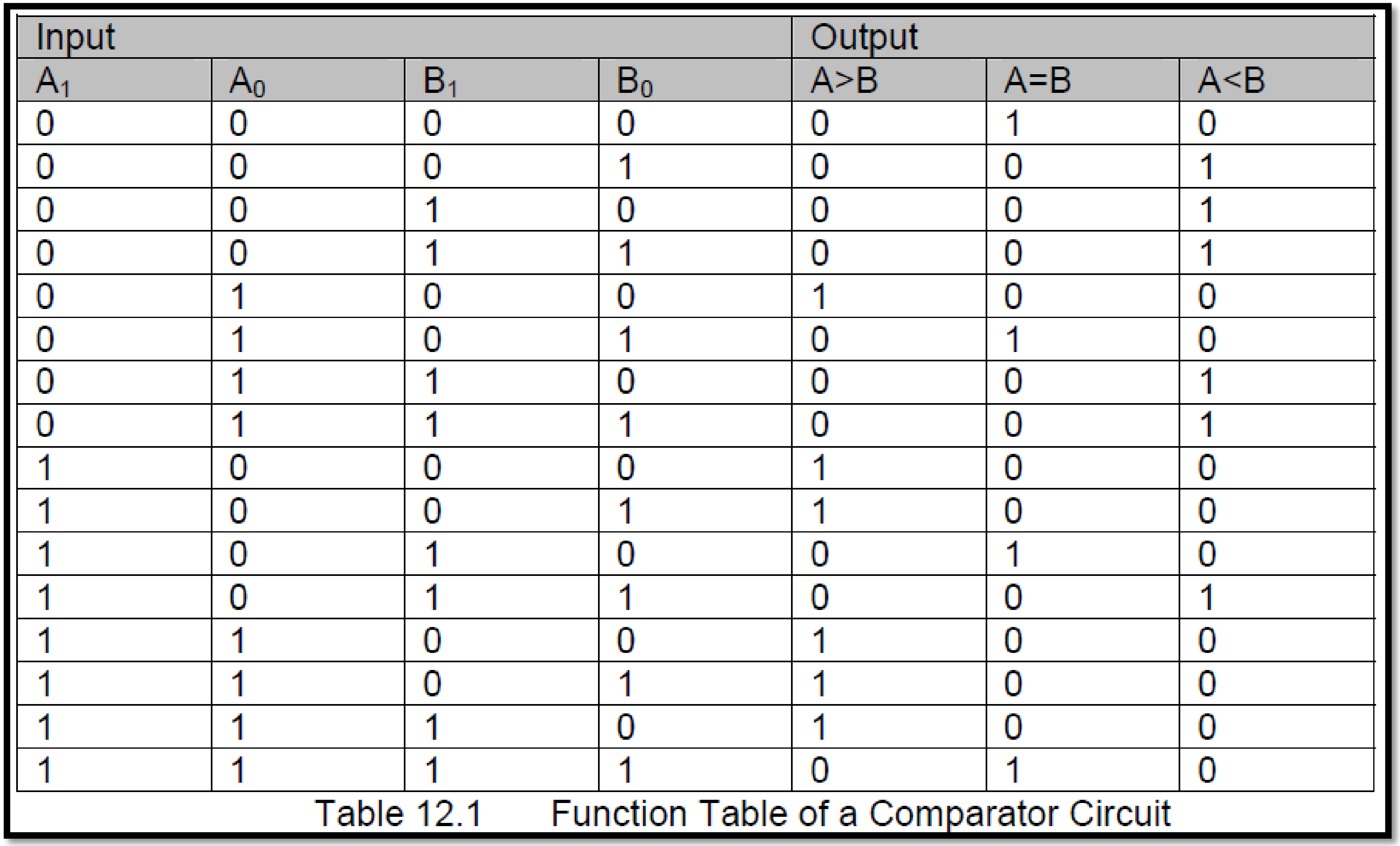
MAGNITUDE COMPARATOR

- A comparator circuit compares two numbers and sets one of its three outputs to 1 indicating the result of the comparison operation.
- A Comparator circuit has multiple inputs and three outputs.
- A 2-bit Comparator circuit compares two 2-bit numbers A and B.
- The comparator circuit has three outputs.
- It sets the A>B output to 1 if A>B.
- It sets the A=B output to 1 if A=B and sets A<B output to 1 if A < B.

MAGNITUDE COMPARATOR

- The output A>B is set to 1 when the input combinations are 01 00, 10 00, 10 01, 11 00, 11 01 and 11 10
- The output A=B is set to 1 when the input combinations are 00 00, 01 01, 10 10 and 11 11
- The output A<B is set to 1 when the input combinations are 00 01, 00 10, 00 11, 01 10, 01 11 and 10 11
- The circuit has 4-bit input, 2-bits represent A and 2-bits represent B and a 3-bit output representing A>B, A=B and A<B.

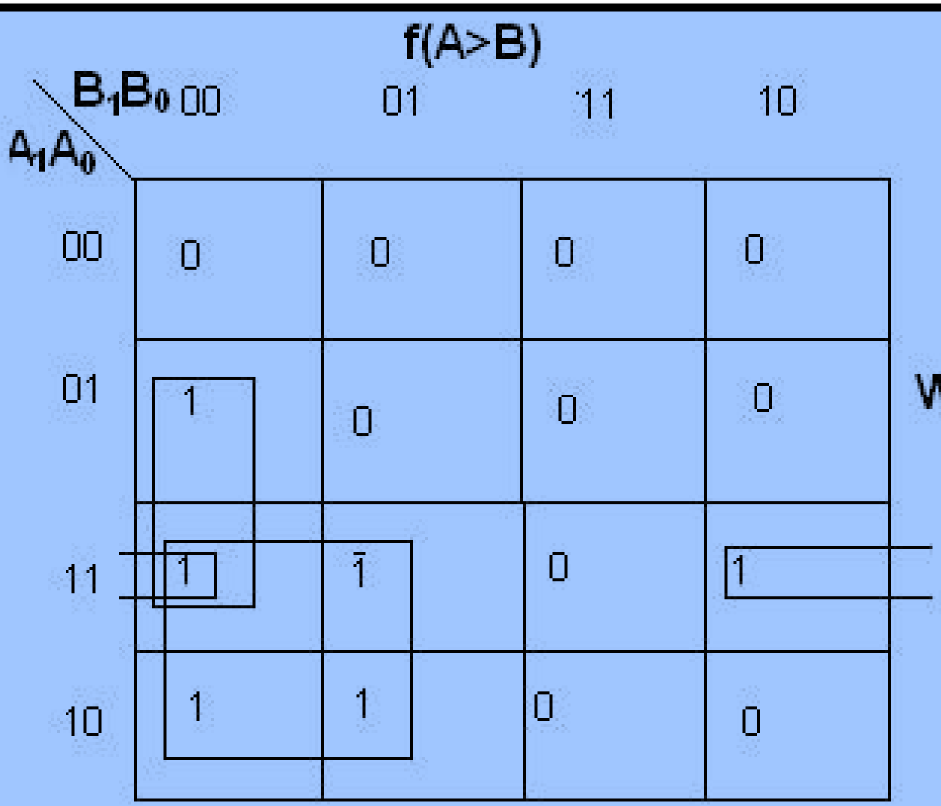
MAGNITUDE COMPARATOR



Input				Output		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Table 12.1 Function Table of a Comparator Circuit

MAGNITUDE COMPARATOR



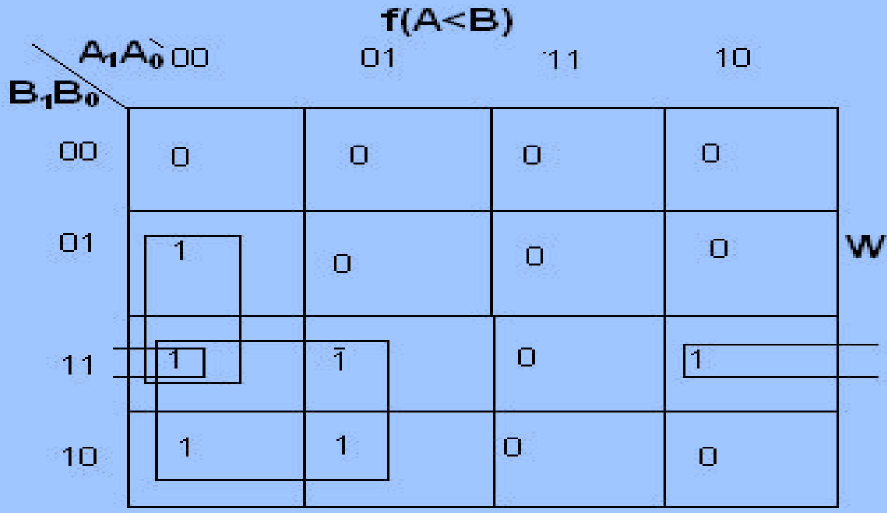
f(A>B)

A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

We get the equation as f(A>B)

$$= A_1\overline{B_1} + A_0\overline{B_1}\,\overline{B_0} + A_1A_0\overline{B_0}$$

MAGNITUDE COMPARATOR

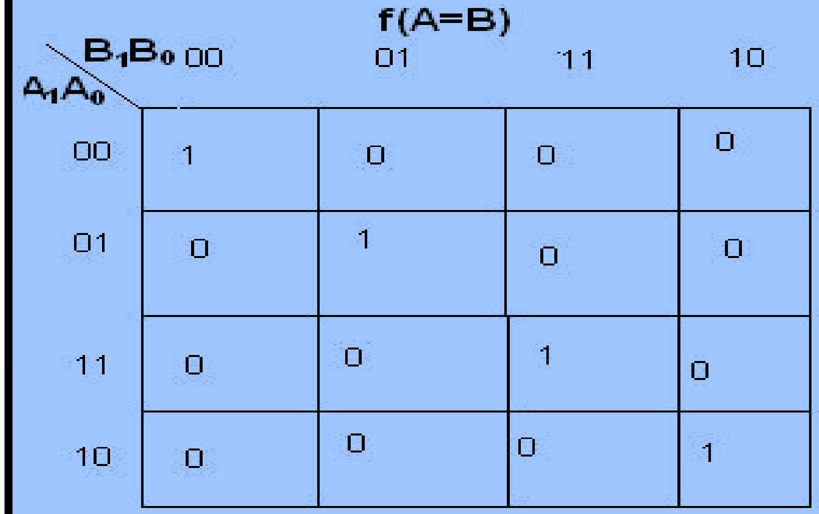


f(A<B)

$B_1B_0 \backslash A_1A_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

We get the equation as f(A<B)

$= \overline{A_1}B_1 + \overline{A_0}B_1B_0 + \overline{A_1}\,\overline{A_0}B_0$



f(A=B)

$A_1A_0 \backslash B_1B_0$	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

We get the equation as f(A=B)

$=(A_1 \text{XOR } B_1).(A_0 \text{XOR } B_0)$

MAGNITUDE COMPARATOR

- Consider two numbers, *A* and *B* , with four digits each.
- Write the coefficients of the numbers in descending order of significance:
- $A = A3\ A2\ A1\ A0$ $\qquad$ $B = B3\ B2\ B1\ B0$
- The two numbers are equal if all pairs of significant digits are equal: $A3 = B3$, $A2 = B2$, $A1 = B1$, and $A0 = B0$.
- When the numbers are binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as:
- $xi = AiBi + A'i\ B'i$ for $i = 0, 1, 2, 3$ where $xi = 1$ only if the pair of bits in position *i* are equal

MAGNITUDE COMPARATOR

- The equality of the two numbers *A* and *B* is displayed in a combinational circuit by an output binary variable that we designate by the symbol (*A* = *B)*.
- This binary variable is equal to 1 if the input numbers, *A* and *B* , are equal, and is equal to 0 otherwise.
- For equality to exist, all *xi* variables must be equal to 1, a condition that dictates an AND operation of all variables:
- $(A = B) = x3x2x1x0$
- The *binary* variable *A* = *B* is equal to 1 only if all pairs of digits of the two numbers are equal.

MAGNITUDE COMPARATOR

- To determine whether A is greater or less than B , we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position.
- If the two digits of a pair are equal, we compare the next lower significant pair of digits.
- The comparison continues until a pair of unequal digits is reached.
- If the corresponding digit of A is 1 and that of B is 0, we conclude that $A > B$.
- If the corresponding digit of A is 0 and that of B is 1, we have $A < B$.

MAGNITUDE COMPARATOR

- A>B:A3’B3+X3A2’B2+X3X2A1’B1+X3X2X1A0’B0

- A<B:’A3B3+X3’A2B2+X3X2’A1B1+X3X2X1’A0B0

- A=B: X3X2X1X0

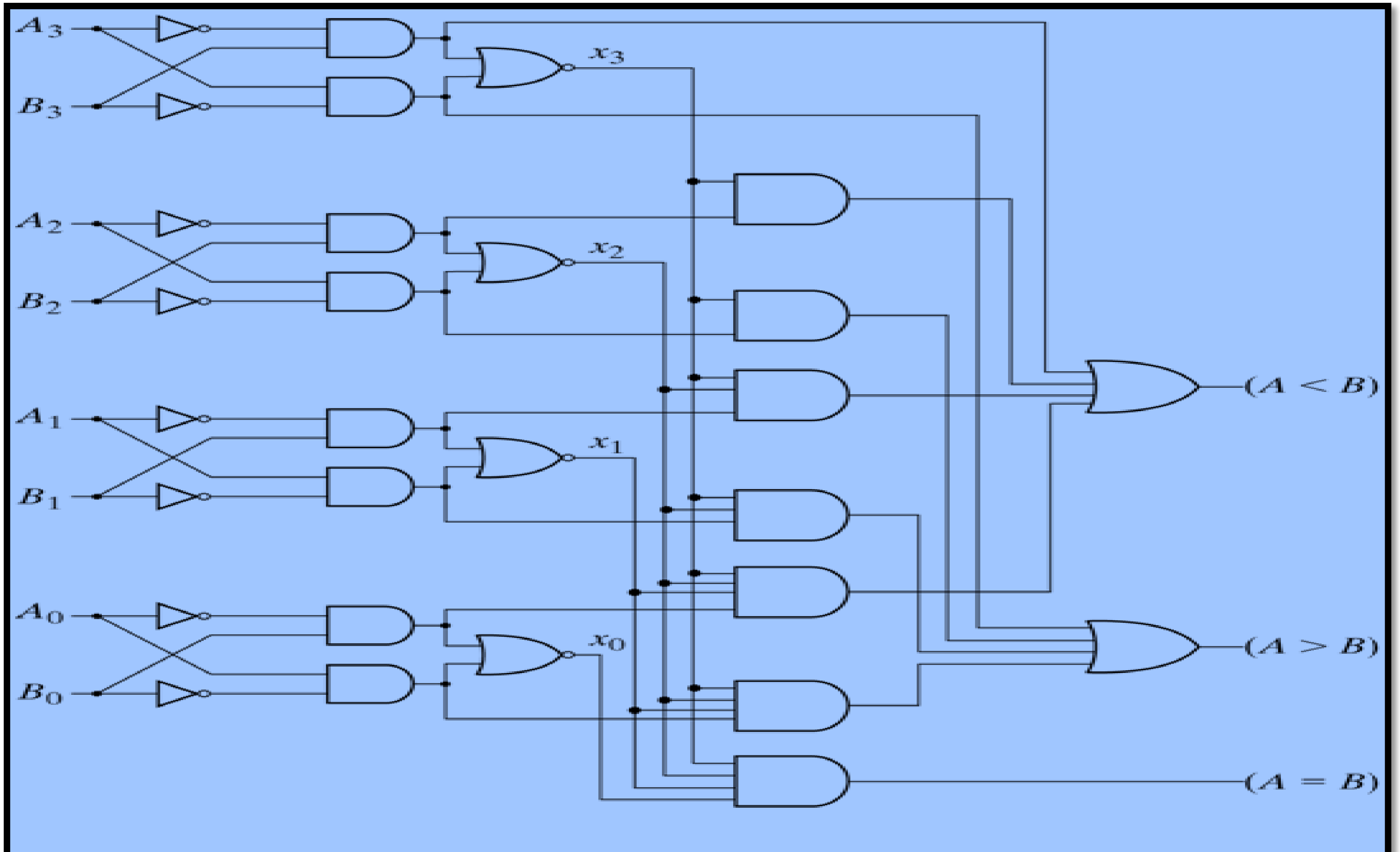


Fig. 4-17 4-Bit Magnitude Comparator

BCD Adder

- Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage.
- Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry.
- Suppose we apply two BCD digits to a four-bit binary adder.
- The adder will form the sum in *binary* and produce a result that ranges from 0 through 19.
- These binary numbers are listed in Table 4.5 and are labeled by symbols *K*, *Z*8, *Z*4, *Z*2, and *Z*1.
- *K* is the carry, and the subscripts under the letter *Z* represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

BCD Adder

- The columns under the binary sum list the binary value that appears in the outputs of the four-bit binary adder.
- The output sum of two decimal digits must be represented in BCD and should appear in the form listed in the columns under “BCD Sum.”
- The problem is to find a rule by which the binary sum is converted to the correct BCD digit representation of the number in the BCD sum.
- In examining the contents of the table, it becomes apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed.

BCD Adder

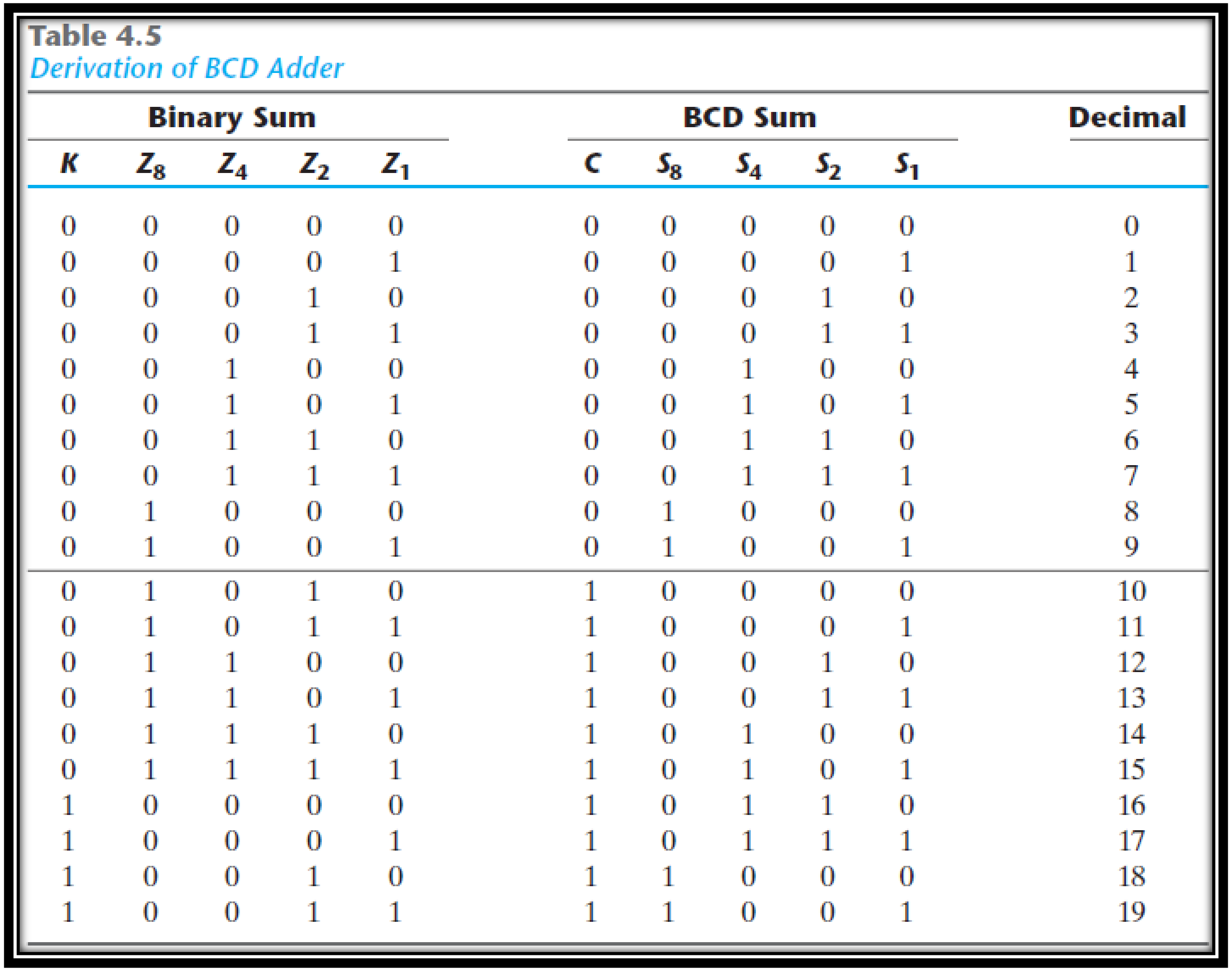


Table 4.5
Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder

- When the binary sum is greater than 1001, we obtain an invalid BCD representation.
- The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- The logic circuit that detects the necessary correction can be derived from the entries in the table.
- It is obvious that a correction is needed when the binary sum has an output carry $K = 1$.
- The other six combinations from 1010 through 1111 that need a correction have a 1 in position $Z8$.

BCD Adder

- To distinguish them from binary 1000 and 1001, which also have a 1 in position $Z8$, we specify further that either $Z4$ or $Z2$ must have a 1.
- The condition for a correction and an output carry can be expressed by the Boolean function
- $C = K + Z8Z4 + Z8Z2$
- When $C = 1$, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage

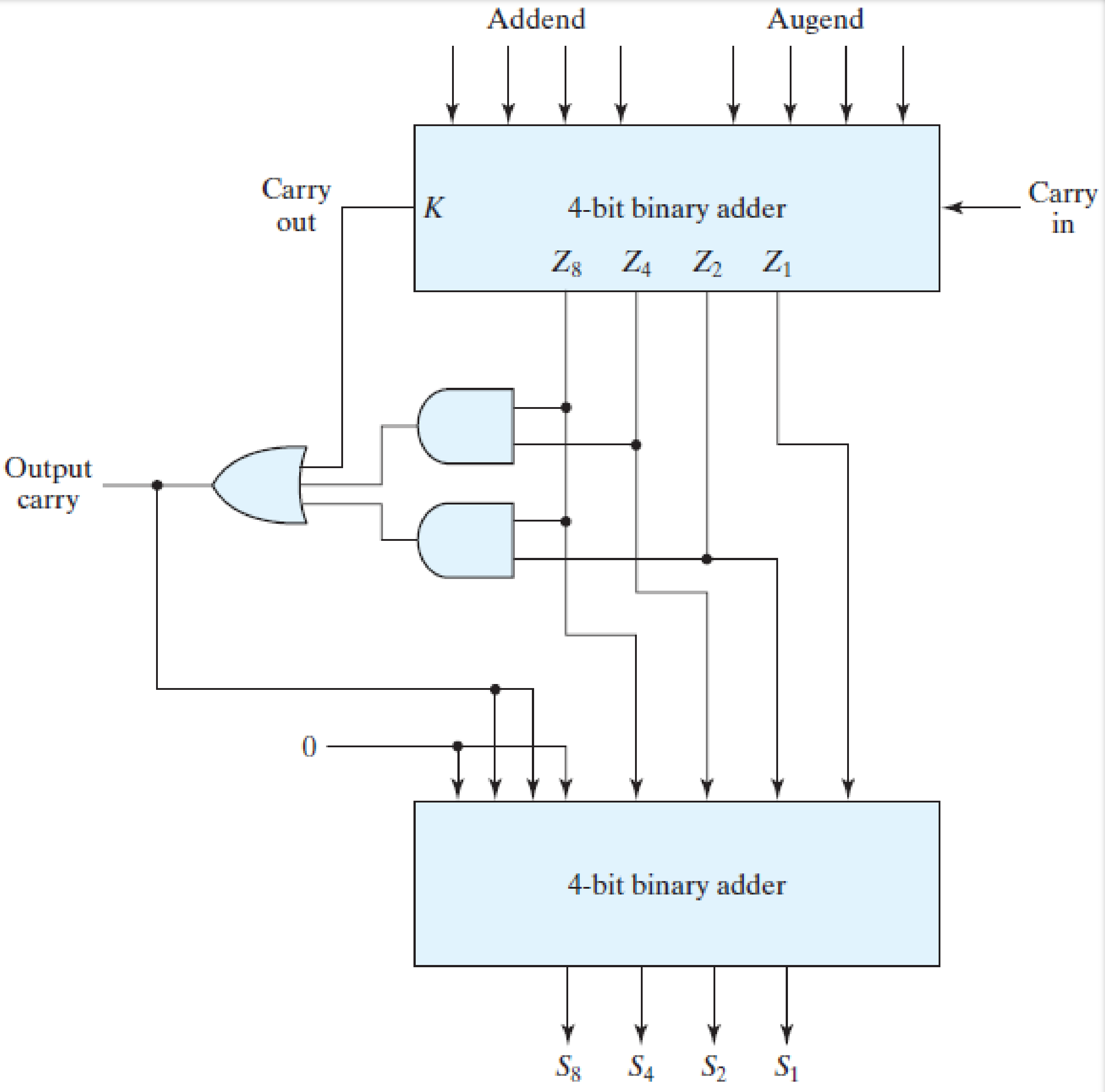


FIGURE 4.14
Block diagram of a BCD adder