7

Arithmetic Circuits

Beginning with this chapter, and in the two chapters following, we will take a comprehensive look at various building blocks used to design more complex combinational circuits. A combinational logic circuit is one where the output or outputs depend upon the present state of combination of the logic inputs. The logic gates discussed in Chapter 4 constitute the most fundamental building block of a combinational circuit. More complex combinational circuits such as adders and subtractors, multiplexers and demultiplexers, magnitude comparators, etc., can be implemented using a combination of logic gates. However, the aforesaid combinational logic functions and many more, including more complex ones, are available in monolithic IC form. A still more complex combinational circuit may be implemented using a combination of these functions available in IC form. In this chapter, we will cover devices used to perform arithmetic and other related operations. These include adders, subtractors, magnitude comparators and look-ahead carry generators. Particular emphasis is placed upon the functioning and design of these combinational circuits. The text has been adequately illustrated with the help of a large number of solved problems, the majority of which are design oriented.

7.1 Combinational Circuits

A *combinational circuit* is one where the output at any time depends only on the present combination of inputs at that point of time with total disregard to the past state of the inputs. The logic gate is the most basic building block of combinational logic. The logical function performed by a combinational circuit is fully defined by a set of Boolean expressions. The other category of logic circuits, called *sequential logic circuits*, comprises both logic gates and memory elements such as flip-flops. Owing to the presence of memory elements, the output in a sequential circuit depends upon not only the present but also the past state of inputs. Basic building blocks of sequential logic circuits are described in detail in Chapters 10 and 11.

Figure 7.1 shows the block schematic representation of a generalized combinational circuit having *n* input variables and *m* output variables or simply outputs. Since the number of input variables is

Digital Electronics: Principles, Devices and Applications Anil K. Maini
© 2007 John Wiley & Sons, Ltd. ISBN: 978-0-470-03214-5

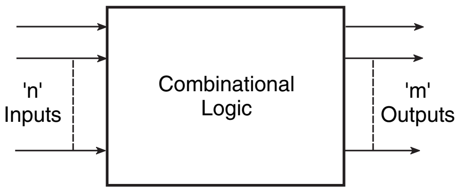


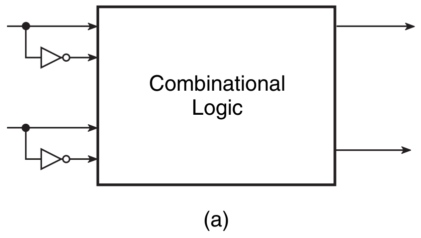
Figure 7.1 Generalized combinational circuit.

n, there are 2^n possible combinations of bits at the input. Each output can be expressed in terms of input variables by a Boolean expression, with the result that the generalized system of Fig. 7.1 can be expressed by m Boolean expressions. As an illustration, Boolean expressions describing the function of a four-input OR/NOR gate are given as

$$Y_1 \text{ (OR output)} = A+B+C+D \quad \text{and} \quad Y_2 \text{ (NOR output)} = \overline{A+B+C+D}$$

Also, each of the input variables may be available as only the normal input on the input line designated for the purpose. In that case, the complemented input, if desired, can be generated by using an inverter, as shown in Fig. 7.2(a), which illustrates the case of a four-input, two-output combinational function. Also, each of the input variables may appear in two wires, one representing the normal literal and the other representing the complemented one, as shown in Fig. 7.2(b).

In combinational circuits, input variables come from an external source and output variables feed an external destination. Both source and destination in the majority of cases are storage registers, and these



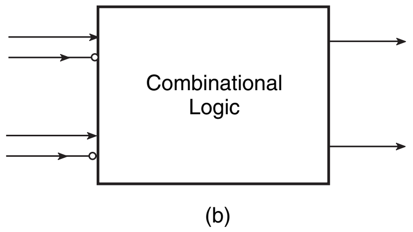


Figure 7.2 Combinational circuit with normal and complemented inputs.

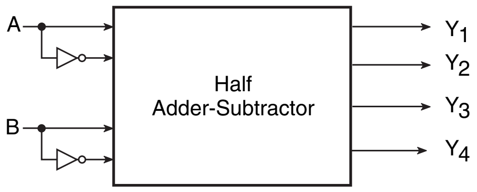


Figure 7.3 Two-input, four-output combinational circuit.

storage devices provide both normal as well as complemented outputs of the stored binary variable. As an illustration, Fig. 7.3 shows a simple two-input (A, B), four-output (Y_1, Y_2, Y_3, Y_4) combinational logic circuit described by the following Boolean expressions

$$Y_1 = A.\overline{B} + \overline{A}.B \tag{7.1}$$

$$Y_2 = A.\overline{B} + \overline{A}.B \tag{7.2}$$

$$Y_3 = A.B \tag{7.3}$$

$$Y_4 = \overline{A}.B \tag{7.4}$$

The implementation of these Boolean expressions needs both normal as well as complemented inputs. Incidentally, the combinational circuit shown is that of a half-adder–subtractor, with A and B representing the two bits to be added or subtracted and Y_1, Y_2, Y_3, Y_4 representing SUM, DIFFERENCE, CARRY and BORROW outputs respectively. Adder and subtractor circuits are discussed in Sections 7.3, 7.4 and 7.5.

7.2 Implementing Combinational Logic

The different steps involved in the design of a combinational logic circuit are as follows:

1. Statement of the problem.
2. Identification of input and output variables.
3. Expressing the relationship between the input and output variables.
4. Construction of a truth table to meet input–output requirements.
5. Writing Boolean expressions for various output variables in terms of input variables.
6. Minimization of Boolean expressions.
7. Implementation of minimized Boolean expressions.

These different steps are self-explanatory. One or two points, however, are worth mentioning here. There are various simplification techniques available for minimizing Boolean expressions, which have been discussed in the previous chapter. These include the use of theorems and identities, Karnaugh mapping, the Quinne–McCluskey tabulation method and so on. Also, there are various possible minimized forms of Boolean expressions. The following guidelines should be followed while choosing the preferred form for hardware implementation:

1. The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.
2. There should be a minimum number of interconnections, and the propagation time should be the shortest.
3. Limitation on the driving capability of the gates should not be ignored.

It is difficult to generalize as to what constitutes an acceptable simplified Boolean expression. The importance of each of the above-mentioned aspects is governed by the nature of application.

7.3 Arithmetic Circuits – Basic Building Blocks

In this section, we will discuss those combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers. Addition and subtraction are the two most commonly used arithmetic operations, as the other two, namely multiplication and division, are respectively the processes of repeated addition and repeated subtraction, as was outlined in Chapter 2 dealing with binary arithmetic. We will begin with the basic building blocks that form the basis of all hardware used to perform the aforesaid arithmetic operations on binary numbers. These include half-adder, full adder, half-subtractor, full subtractor and controlled inverter.

7.3.1 Half-Adder

A *half-adder* is an arithmetic circuit block that can be used to add two bits. Such a circuit thus has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY. Figure 7.4 shows the truth table of a half-adder, showing all possible input combinations and the corresponding outputs.

The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$\text{SUM } S = A.\overline{B} + \overline{A}.B \tag{7.5}$$

$$\text{CARRY } C = A.B \tag{7.6}$$

An examination of the two expressions tells that there is no scope for further simplification. While the first one representing the SUM output is that of an EX-OR gate, the second one representing the

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

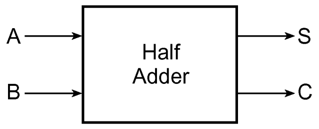


Figure 7.4 Truth table of a half-adder.

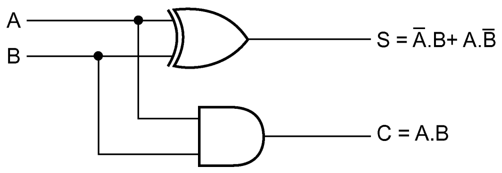


Figure 7.5 Logic implementation of a half-adder.

CARRY output is that of an AND gate. However, these two expressions can certainly be represented in different forms using various laws and theorems of Boolean algebra to illustrate the flexibility that the designer has in hardware-implementing as simple a combinational function as that of a half-adder. We have studied in Chapter 6 on Boolean algebra how various logic gates can be implemented in the form of either only NAND gates or NOR gates. Although the simplest way to hardware-implement a half-adder would be to use a two-input EX-OR gate for the SUM output and a two-input AND gate for the CARRY output, as shown in Fig. 7.5, it could also be implemented by using an appropriate arrangement of either NAND or NOR gates. Figure 7.6 shows the implementation of a half-adder with NAND gates only.

A close look at the logic diagram of Fig. 7.6 reveals that one part of the circuit implements a two-input EX-OR gate with two-input NAND gates. EX-OR implementation using NAND was discussed in the previous chapter. The AND gate required to generate CARRY output is implemented by complementing an already available NAND output of the input variables.

7.3.2 Full Adder

A *full adder* circuit is an arithmetic circuit block that can be used to add three bits to produce a SUM and a CARRY output. Such a building block becomes a necessity when it comes to adding binary numbers with a large number of bits. The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. Let us recall the procedure for adding larger binary numbers. We begin with the addition of LSBs of the two numbers. We record the sum under the LSB column and take the carry, if any, forward to the next higher column bits. As a result, when we add the next adjacent higher column bits, we would be required to add three bits if there were a carry from the previous addition. We have a similar situation for the other higher column bits

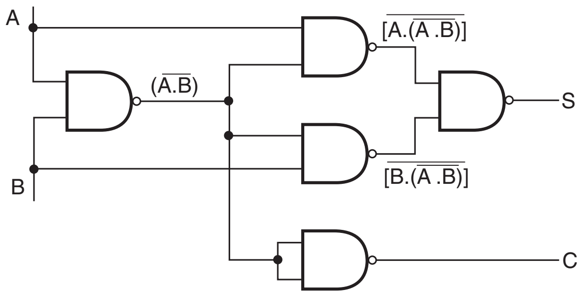
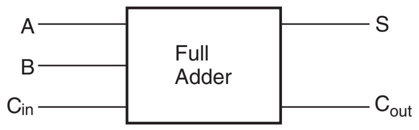

A
B
($\overline{A.B}$)
[$\overline{A.(\overline{A.B})}$]
[$\overline{B.(\overline{A.B})}$]
S
C

Figure 7.6 Half-adder implementation using NAND gates.



A	B	C_{in}	SUM (S)	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 7.7 Truth table of a full adder.

also until we reach the MSB. A full adder is therefore essential for the hardware implementation of an adder circuit capable of adding larger binary numbers. A half-adder can be used for addition of LSBs only.

Figure 7.7 shows the truth table of a full adder circuit showing all possible input combinations and corresponding outputs. In order to arrive at the logic circuit for hardware implementation of a full adder, we will firstly write the Boolean expressions for the two output variables, that is, the SUM and CARRY outputs, in terms of input variables. These expressions are then simplified by using any of the simplification techniques described in the previous chapter. The Boolean expressions for the two output variables are given in Equation (7.7) for the SUM output (S) and in Equation (6.6) for the CARRY output (C_{out}):

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in} \tag{7.7}$$

$$C_{out} = \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C}_{in} + A.B.C_{in} \tag{7.8}$$

The next step is to simplify the two expressions. We will do so with the help of the Karnaugh mapping technique. Karnaugh maps for the two expressions are given in Fig. 7.8(a) for the SUM output and Fig. 7.8(b) for the CARRY output. As is clear from the two maps, the expression for the SUM (S) output cannot be simplified any further, whereas the simplified Boolean expression for C_{out} is given by the equation

$$C_{out} = B.C_{in} + A.B + A.C_{in} \tag{7.9}$$

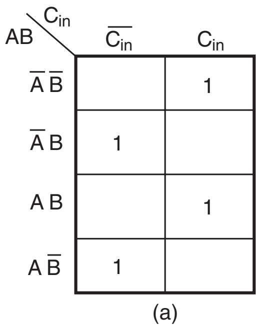
Figure 7.9 shows the logic circuit diagram of the full adder. A full adder can also be seen to comprise two half-adders and an OR gate. The expressions for SUM and CARRY outputs can be rewritten as follows:

$$S = \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(A.B + \overline{A}.\overline{B})$$

$$S = \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.\overline{(\overline{A}.B + A.\overline{B})} \tag{7.10}$$

Similarly, the expression for CARRY output can be rewritten as follows:

$$\begin{aligned} C_{out} &= B.C_{in}.(A + \overline{A}) + A.B + A.C_{in}.(B + \overline{B}) \\ &= A.B + A.B.C_{in} + \overline{A}.B.C_{in} + A.B.C_{in} + A.\overline{B}.C_{in} = A.B + A.B.C_{in} + \overline{A}.B.C_{in} + A.\overline{B}.C_{in} \\ &= A.B.(1 + C_{in}) + C_{in}.(\overline{A}.B + A.\overline{B}) \end{aligned}$$



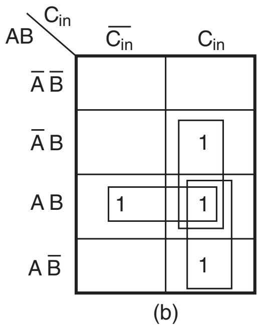
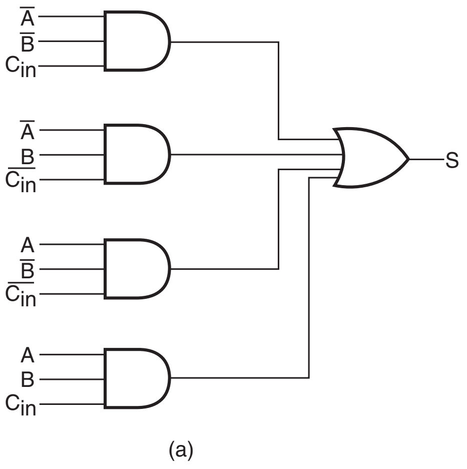


Figure 7.8 Karnaugh maps for the sum and carry-out of a full adder.

$$C_{out} = A.B + C_{in}.(\overline{A}.B + A.\overline{B}) \tag{7.11}$$

Boolean expression (7.10) can be implemented with a two-input EX-OR gate provided that one of the inputs is C_{in} and the other input is the output of another two-input EX-OR gate with A and B as its inputs. Similarly, Boolean expression (7.11) can be implemented by ORing two minterms. One of them is the AND output of A and B. The other is also the output of an AND gate whose inputs are C_{in} and the output of an EX-OR operation on A and B. The whole idea of writing the Boolean expressions in this modified form was to demonstrate the use of a half-adder circuit in building a full adder. Figure 7.10(a) shows logic implementation of Equations (7.10) and (7.11). Figure 7.10(b) is nothing but Fig. 7.10(a) redrawn with the portion of the circuit representing a half-adder replaced with a block.

The full adder of the type described above forms the basic building block of binary adders. However, a single full adder circuit can be used to add one-bit binary numbers only. A cascade arrangement of these adders can be used to construct adders capable of adding binary numbers with a larger number of bits. For example, a four-bit binary adder would require four full adders of the type shown in Fig. 7.10 to be connected in cascade. Figure 7.11 shows such an arrangement. $(A_3A_2A_1A_0)$ and $(B_3B_2B_1B_0)$ are the two binary numbers to be added, with A_0 and B_0 representing LSBs and A_3 and B_3 representing MSBs of the two numbers.



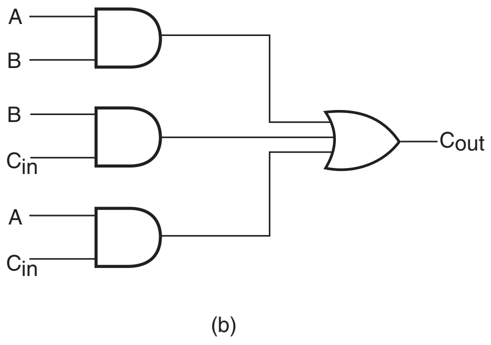


Figure 7.9 Logic circuit diagram of a full adder.

7.3.3 Half-Subtractor

We have seen in Chapter 3 on digital arithmetic how subtraction of two given binary numbers can be carried out by adding 2's complement of the subtrahend to the minuend. This allows us to do a subtraction operation with adder circuits. We will study the use of adder circuits for subtraction operations in the following pages. Before we do that, we will briefly look at the counterparts of half-adder and full adder circuits in the half-subtractor and full subtractor for direct implementation of subtraction operations using logic gates.

A *half-subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The truth table of a half-subtractor, as shown in Fig. 7.12, explains this further. The Boolean expressions for the two outputs are given by the equations

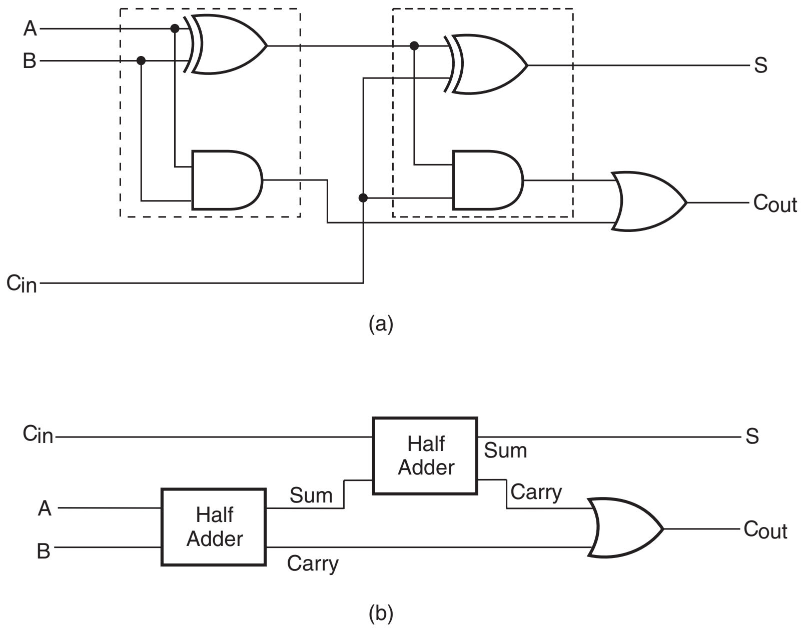


Figure 7.10 Logic implementation of a full adder with half-adders.

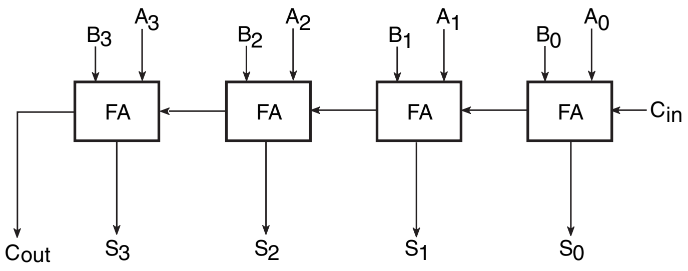
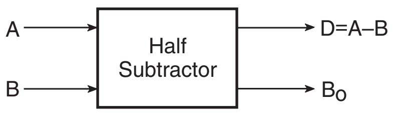


Figure 7.11 Four-bit binary adder.

$$D = \overline{A}.B + A.\overline{B} \tag{7.12}$$

$$B_o = \overline{A}.B \tag{7.13}$$

It is obvious that there is no further scope for any simplification of the Boolean expressions given by Equations (7.12) and (7.13). While the expression for the DIFFERENCE (D) output is that of



A	B	D	Bo
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Figure 7.12 Half-subtractor.

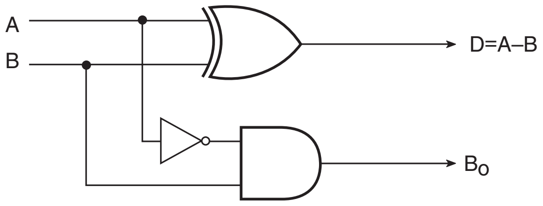


Figure 7.13 Logic diagram of a half-subtractor.

an EX-OR gate, the expression for the BORROW output (B_o) is that of an AND gate with input A complemented before it is fed to the gate. Figure 7.13 shows the logic implementation of a half-subtractor. Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, that is, the minuend, is complemented, an AND gate can be used to implement the BORROW output. Note the similarities between the logic diagrams of Fig. 7.5 (half-adder) and Fig. 7.13 (half-subtractor).

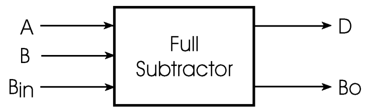
7.3.4 Full Subtractor

A *full subtractor* performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not. As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. Figure 7.14 shows the truth table of a full subtractor.

The Boolean expressions for the two output variables are given by the equations

$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in} \tag{7.14}$$

$$B_o = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in} \tag{7.15}$$



Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figure 7.14 Truth table of a full subtractor.

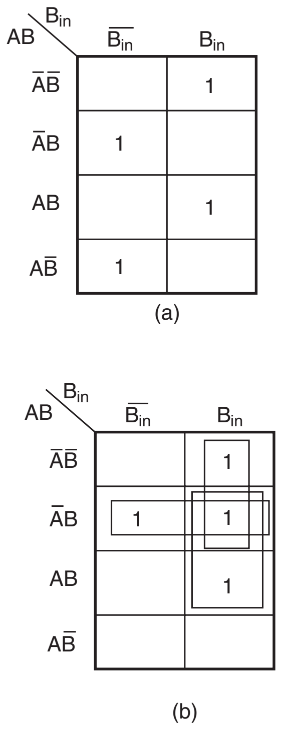
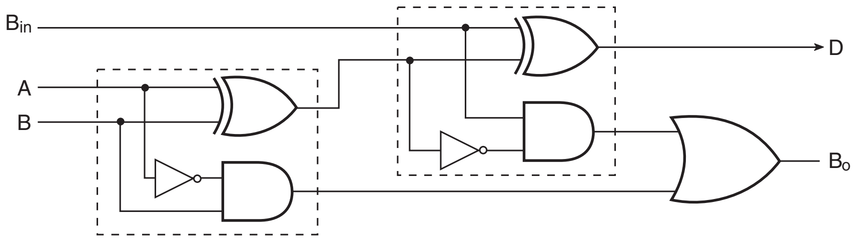


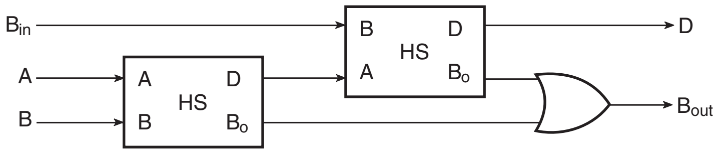
Figure 7.15 Karnaugh maps for difference and borrow outputs.

The Karnaugh maps for the two expressions are given in Fig. 7.15(a) for DIFFERENCE output D and in Fig. 7.15(b) for BORROW output B_o. As is clear from the two Karnaugh maps, no simplification is possible for the difference output D. The simplified expression for B_o is given by the equation

$$B_o = \overline{A}.B + \overline{A}.B_{in} + B.B_{in} \qquad (7.16)$$



(a)



(b)

Figure 7.16 Logic implementation of a full subtractor with half-subtractors.

If we compare these expressions with those derived earlier in the case of a full adder, we find that the expression for DIFFERENCE output D is the same as that for the SUM output. Also, the expression for BORROW output B_o is similar to the expression for CARRY-OUT C_o. In the case of a half-subtractor, the A input is complemented. By a similar analysis it can be shown that a full subtractor can be implemented with half-subtractors in the same way as a full adder was constructed using half-adders. Relevant logic diagrams are shown in Figs 7.16(a) and (b) corresponding to Figs 7.10(a) and (b) respectively for a full adder.

Again, more than one full subtractor can be connected in cascade to perform subtraction on two larger binary numbers. As an illustration, Fig. 7.17 shows a four-bit subtractor.

7.3.5 Controlled Inverter

A *controlled inverter* is needed when an adder is to be used as a subtractor. As outlined earlier, subtraction is nothing but addition of the 2's complement of the subtrahend to the minuend. Thus, the first step towards practical implementation of a subtractor is to determine the 2's complement of the subtrahend. And for this, one needs firstly to find 1's complement. A controlled inverter is used to find 1's complement. A one-bit controlled inverter is nothing but a two-input EX-OR gate with one of its inputs treated as a control input, as shown in Fig. 7.18(a). When the control input is LOW, the input bit is passed as such to the output. (Recall the truth table of an EX-OR gate.) When the control input is HIGH, the input bit gets complemented at the output. Figure 7.18(b) shows an eight-bit controlled inverter of this type. When the control input is LOW, the output ($Y_7\ Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0$) is the same as the input ($A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$). When the control input is HIGH, the output is 1's complement

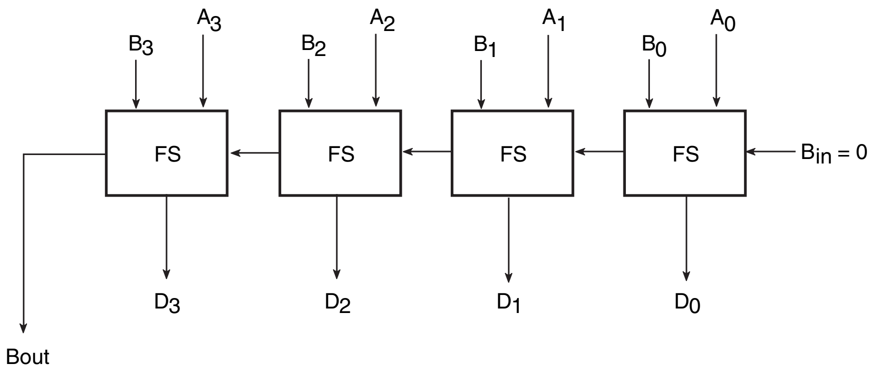


Figure 7.17 Four-bit subtractor.

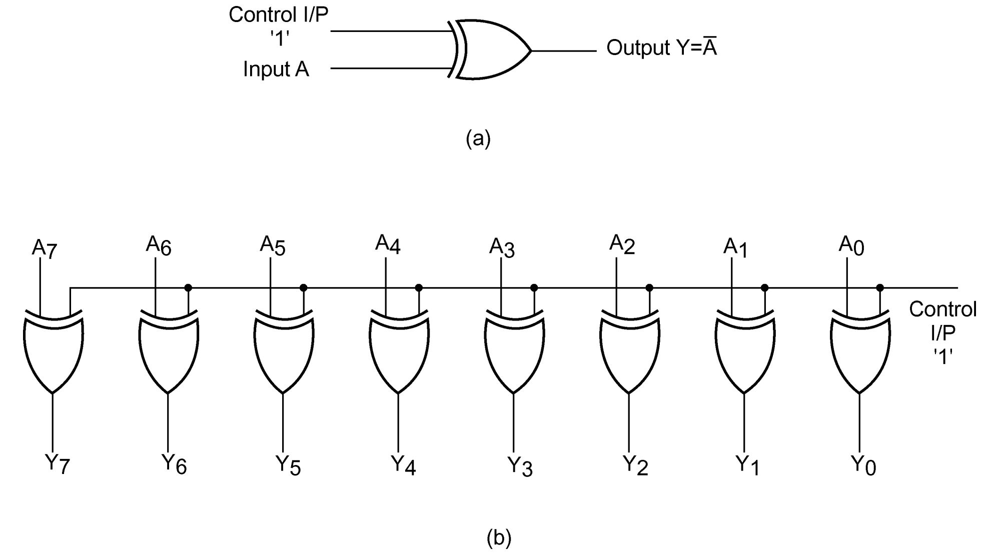


Figure 7.18 (a) One-bit controlled inverter and (b) eight-bit controlled inverter.

of the input. As an example, 11010010 at the input would produce 00101101 at the output when the control input is in a logic '1' state.

7.4 Adder–Subtractor

Subtraction of two binary numbers can be accomplished by adding 2's complement of the subtrahend to the minuend and disregarding the final carry, if any. If the MSB bit in the result of addition is

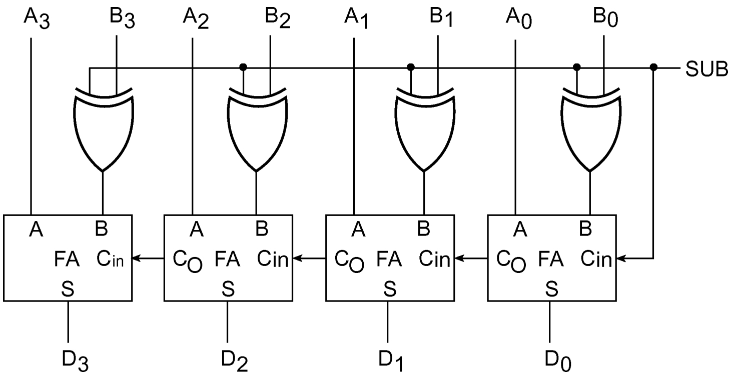


Figure 7.19 Four-bit adder-subtractor.

a '0', then the result of addition is the correct answer. If the MSB bit is a '1', this implies that the answer has a negative sign. The true magnitude in this case is given by 2's complement of the result of addition.

Full adders can be used to perform subtraction provided we have the necessary additional hardware to generate 2's complement of the subtrahend and disregard the final carry or overflow. Figure 7.19 shows one such hardware arrangement. Let us see how it can be used to perform subtraction of two four-bit binary numbers. A close look at the diagram would reveal that it is the hardware arrangement for a four-bit binary adder, with the exception that the bits of one of the binary numbers are fed through controlled inverters. The control input here is referred to as the SUB input. When the SUB input is in logic '0' state, the four bits of the binary number ($B_3\,B_2\,B_1\,B_0$) are passed on as such to the B inputs of the corresponding full adders. The outputs of the full adders in this case give the result of addition of the two numbers. When the SUB input is in logic '1' state, four bits of one of the numbers, ($B_3\,B_2\,B_1\,B_0$) in the present case, get complemented. If the same '1' is also fed to the CARRY-IN of the LSB full adder, what we finally achieve is the addition of 2's complement and not 1's complement. Thus, in the adder arrangement of Fig. 7.19, we are basically adding 2's complement of ($B_3\,B_2\,B_1\,B_0$) to ($A_3\,A_2\,A_1\,A_0$). The outputs of the full adders in this case give the result of subtraction of the two numbers. The arrangement shown achieves $A - B$. The final carry (the CARRY-OUT of the MSB full adder) is ignored if it is not displayed.

For implementing an eight-bit adder–subtractor, we will require eight full adders and eight two-input EX-OR gates. Four-bit full adders and quad two-input EX-OR gates are individually available in integrated circuit form. A commonly used four-bit adder in the TTL family is the type number 7483. Also, type number 7486 is a quad two-input EX-OR gate in the TTL family. Figure 7.20 shows a four-bit binary adder–subtractor circuit implemented with 7483 and 7486. Two each of 7483 and 7486 can be used to construct an eight-bit adder–subtractor circuit.

7.5 BCD Adder

A *BCD adder* is used to perform the addition of BCD numbers. A BCD digit can have any of the ten possible four-bit binary representations, that is, 0000, 0001, . . . , 1001, the equivalent of decimal numbers 0, 1, . . . , 9. When we set out to add two BCD digits and we assume that there is an input carry too, the highest binary number that we can get is the equivalent of decimal number 19 ($9 + 9 + 1$).

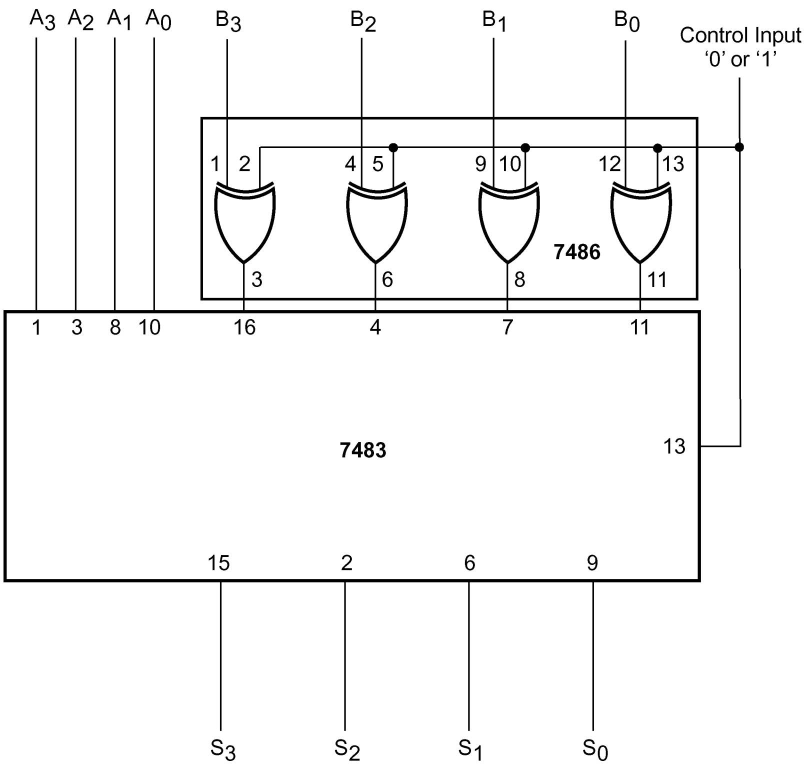


Figure 7.20 Four-bit adder-subtractor.

This binary number is going to be $(10011)_2$. On the other hand, if we do BCD addition, we would expect the answer to be $(0001\ 1001)_{\text{BCD}}$. And if we restrict the output bits to the minimum required, the answer in BCD would be $(1\ 1001)_{\text{BCD}}$. Table 7.1 lists the possible results in binary and the expected results in BCD when we use a four-bit binary adder to perform the addition of two BCD digits. It is clear from the table that, as long as the sum of the two BCD digits remains equal to or less than 9, the four-bit adder produces the correct BCD output.

The binary sum and the BCD sum in this case are the same. It is only when the sum is greater than 9 that the two results are different. It can also be seen from the table that, for a decimal sum greater than 9 (or the equivalent binary sum greater than 1001), if we add 0110 to the binary sum, we can get the correct BCD sum and the desired carry output too. The Boolean expression that can apply the necessary correction is written as

$$C = K + Z_3.Z_2 + Z_3.Z_1 \tag{7.17}$$

Equation (7.17) implies the following. A correction needs to be applied whenever $K = 1$. This takes care of the last four entries. Also, a correction needs to be applied whenever both Z_3 and Z_2 are '1'. This takes care of the next four entries from the bottom, corresponding to a decimal sum equal to

Table 7.1 Results in binary and the expected results in BCD using a four-bit binary adder to perform the addition of two BCD digits.

Decimal sum	Binary sum					BCD sum				
	K	Z_3	Z_2	Z_1	Z_0	C	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

12, 13, 14 and 15. For the remaining two entries corresponding to a decimal sum equal to 10 and 11, a correction is applied for both Z_3 and Z_1, being '1'. While hardware-implementing, 0110 can be added to the binary sum output with the help of a second four-bit binary adder. The correction logic as dictated by the Boolean expression (7.17) should ensure that (0110) gets added only when the above expression is satisfied. Otherwise, the sum output of the first binary adder should be passed on as such to the final output, which can be accomplished by adding (0000) in the second adder. Figure 7.21 shows the logic arrangement of a BCD adder capable of adding two BCD digits with the help of two four-bit binary adders and some additional combinational logic.

The BCD adder described in the preceding paragraphs can be used to add two single-digit BCD numbers only. However, a cascade arrangement of single-digit BCD adder hardware can be used to perform the addition of multiple-digit BCD numbers. For example, an n-digit BCD adder would require n such stages in cascade. As an illustration, Fig. 7.22 shows the block diagram of a circuit for the addition of two three-digit BCD numbers. The first BCD adder, labelled LSD (Least Significant Digit), handles the least significant BCD digits. It produces the sum output ($S_3\,S_2\,S_1\,S_0$), which is the BCD code for the least significant digit of the sum. It also produces an output carry that is fed as an input carry to the next higher adjacent BCD adder. This BCD adder produces the sum output ($S_7\,S_6\,S_5\,S_4$), which is the BCD code for the second digit of the sum, and a carry output. This output carry serves as an input carry for the BCD adder representing the most significant digits. The sum outputs ($S_{11}\,S_{10}\,S_9\,S_8$) represent the BCD code for the MSD of the sum.

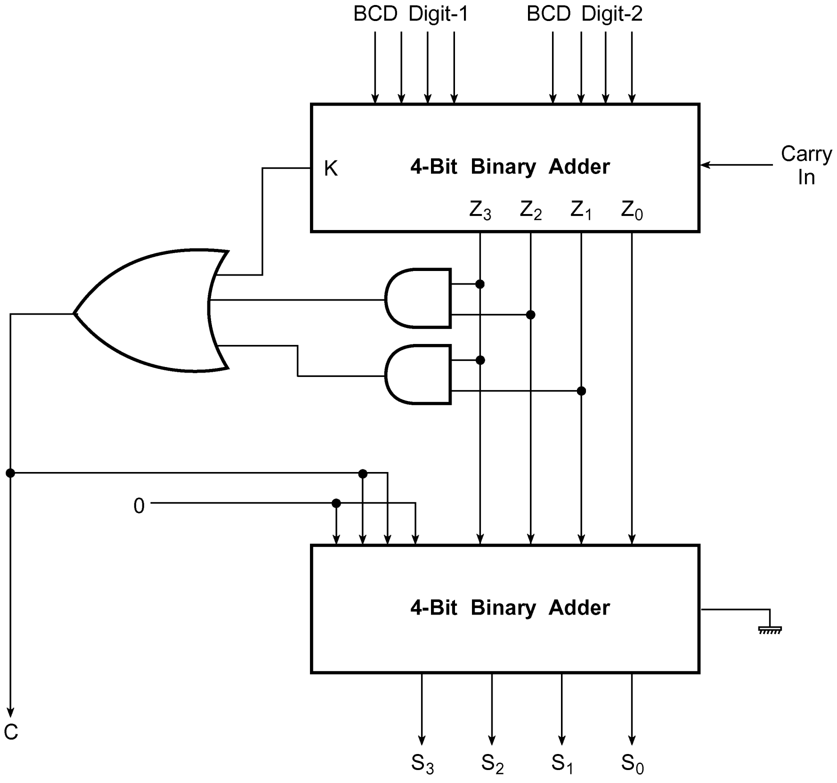


Figure 7.21 Single-digit BCD adder.

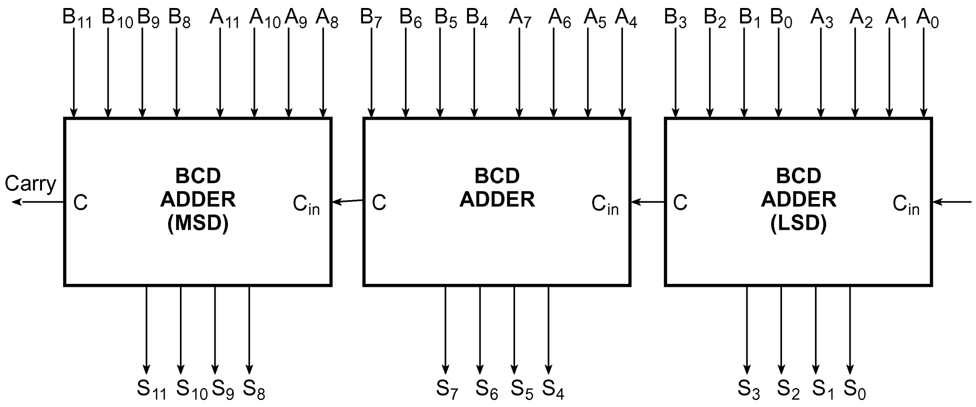

B_{11} B_{10} B_9 B_8 A_{11} A_{10} A_9 A_8
B_7 B_6 B_5 B_4 A_7 A_6 A_5 A_4
B_3 B_2 B_1 B_0 A_3 A_2 A_1 A_0
Carry
C
BCD ADDER (MSD)
C_{in}
C
BCD ADDER
C_{in}
C
BCD ADDER (LSD)
C_{in}
S_{11} S_{10} S_9 S_8
S_7 S_6 S_5 S_4
S_3 S_2 S_1 S_0

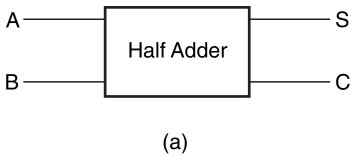
Figure 7.22 Three-digit BCD adder.

Example 7.1

For the half-adder circuit of Fig. 7.23(a), the inputs applied at A and B are as shown in Fig. 7.23(b). Plot the corresponding SUM and CARRY outputs on the same scale.

Solution

The SUM and CARRY waveforms can be plotted from our knowledge of the truth table of the half-adder. All that we need to remember to solve this problem is that $0+0$ yields a '0' as the SUM output and a '0' as the CARRY. $0+1$ or $1+0$ yield '1' as the SUM output and '0' as the CARRY. $1+1$ produces a '0' as the SUM output and a '1' as the CARRY. The output waveforms are as shown in Fig. 7.24.



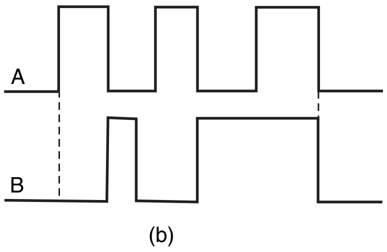


Figure 7.23 Example 7.1.

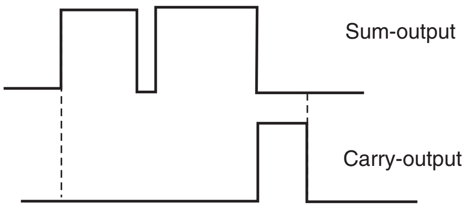


Figure 7.24 Solution to example 7.1.

Example 7.2

Given the relevant Boolean expressions for half-adder and half-subtractor circuits, design a half-adder–subtractor circuit that can be used to perform either addition or subtraction on two one-bit numbers. The desired arithmetic operation should be selectable from a control input.

Solution

Boolean expressions for the half-adder and half-subtractor are given as follows:

Half-adder

$$\text{SUM output} = \overline{A}B + A\overline{B} \quad \text{and} \quad \text{CARRY output} = AB$$

Half-subtractor

$$\text{DIFFERENCE output } = \overline{A}B + A\overline{B} \quad \text{and} \quad \text{BORROW output} = \overline{A}B$$

If we use a controlled inverter for complementing A in the case of the half-subtractor circuit, then the same hardware can also be used to add two one-bit numbers. Figure 7.25 shows the logic circuit diagram. When the control input is '0', input variable A is passed uncomplemented to the input of the NAND gate. In this case, the AND gate generates the CARRY output of the addition operation. The EX-OR gate generates the SUM output. On the other hand, when the control input is '1', the AND gate generates the BORROW output and the EX-OR gate generates the DIFFERENCE output. Thus, '0' at the control input makes it a half-adder, while '1' at the control input makes it a half-subtractor.

Example 7.3

Refer to Fig. 7.26. Write the simplified Boolean expressions for DIFFERENCE and BORROW outputs.

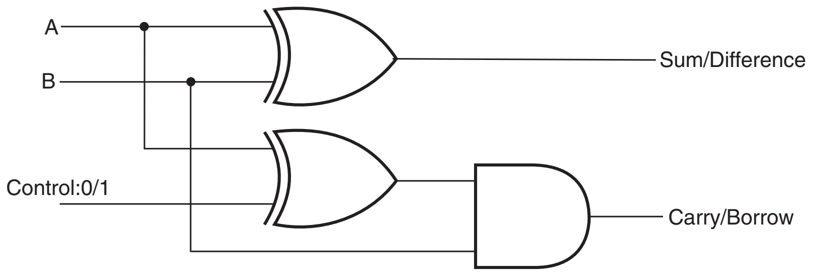


Figure 7.25 Solution to example 7.2.

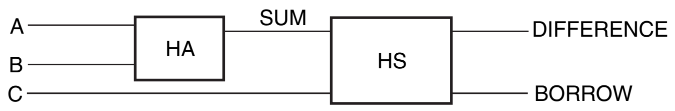


Figure 7.26 Example 7.3.

Solution

Let us assume that the two inputs to the half-subtractor circuit are X and Y, with X equal to the SUM output of the half-adder and Y equal to C. DIFFERENCE and BORROW outputs can then be expressed as follows:

$$\text{DIFFERENCE output} = X \oplus Y = \overline{X}.Y + X.\overline{Y} \quad \text{and} \quad \text{BORROW output} = \overline{X}.Y$$

Also, $X = \overline{A}.B + A.\overline{B}$ and $Y = C$.

Substituting the values of X and Y, we obtain

$$\begin{aligned}\text{DIFFERENCE output} &= (\overline{\overline{A}.B + A.\overline{B}}).C + (\overline{A}.B + A.\overline{B}).\overline{C} = (A.B + \overline{A}.\overline{B}).C + (\overline{A}.B + A.\overline{B}).\overline{C} \\ &= A.B.C + \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C}\end{aligned}$$

$$\text{BORROW output} = \overline{X}.Y = (\overline{\overline{A}.B + A.\overline{B}}).C = (A.B + \overline{A}.\overline{B}).C = A.B.C + \overline{A}.\overline{B}.C$$

Example 7.4

Design an eight-bit adder–subtractor circuit using four-bit binary adders, type number 7483, and quad two-input EX-OR gates, type number 7486. Assume that pin connection diagrams of these ICs are available to you.

Solution

IC 7483 is a four-bit binary adder, which means that it can add two four-bit binary numbers. In order to add two eight-bit numbers, we need to use two 7483s in cascade. That is, the CARRY-OUT (pin 14) of the 7483 handling less significant four bits is fed to the CARRY-IN (pin 13) of the 7483 handling more significant four bits. Also, if $(A_0 \ldots A_7)$ and $(B_0 \ldots B_7)$ are the two numbers to be operated upon, and if the objective is to compute $A - B$, bits B_0, B_1, B_2, B_3, B_4, B_5, B_6 and B_7 are complemented using EX-OR gates. One of the inputs of all EX-OR gates is tied together to form the control input. When the control input is in logic '1' state, bits B_0 to B_7 get complemented. Also, feeding this logic '1' to the CARRY-IN of lower 7483 ensures that we get 2's complement of bits $(B_0 \ldots B_7)$. Therefore, when the control input is in logic '1' state, the two's complement of $(B_0 \ldots B_7)$ is added to $(A_0 \ldots A_7)$. The output is therefore $A - B$. A logic '0' at the control input allows $(B_0 \ldots B_7)$ to pass through EX-OR gates uncomplemented, and the output in that case is $A + B$. Figure 7.27 shows the circuit diagram.

Example 7.5

The logic diagram of Fig. 7.28 performs the function of a very common arithmetic building block. Identify the logic function.

Solution

Writing Boolean expressions for X and Y,

$$X = \overline{(\overline{\overline{A}.B}).(\overline{A.\overline{B}})} = (\overline{\overline{\overline{A}.B}} + \overline{\overline{A.\overline{B}}}) = \overline{A}.B + A.\overline{B} \quad \text{and} \quad Y = (\overline{\overline{A} + \overline{B}}) = A.B$$

Boolean expressions for X and Y are those of a half-adder. X and Y respectively represent SUM and CARRY outputs.

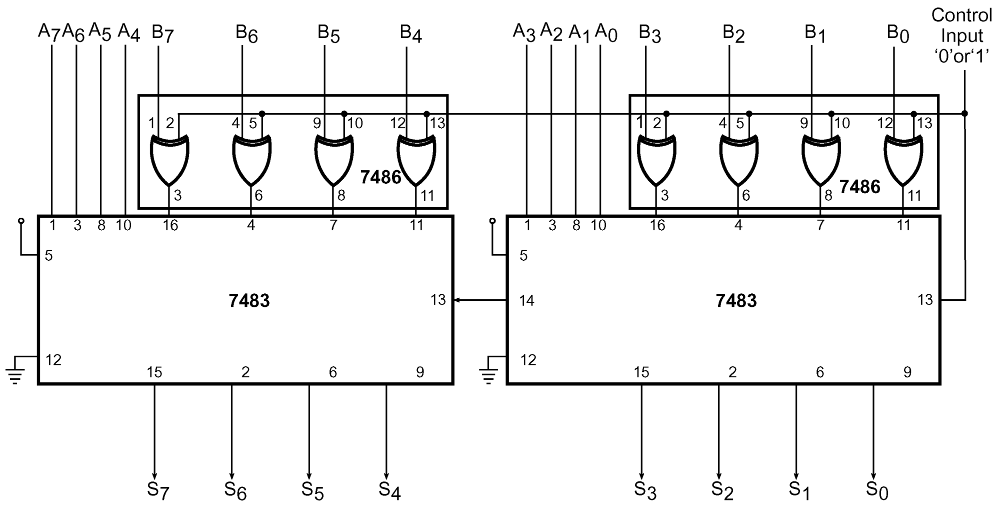


Figure 7.27 Solution to example 7.4.

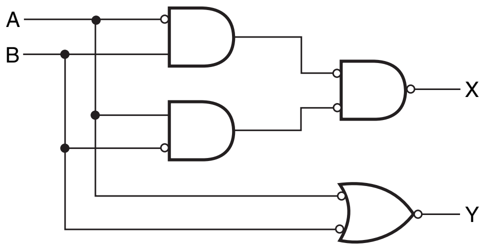


Figure 7.28 Example 7.5.

Example 7.6

Design a BCD adder circuit capable of adding BCD equivalents of two-digit decimal numbers. Indicate the IC type numbers used if the design has to be TTL logic family compatible.

Solution

The desired BCD adder is a cascaded arrangement of two stages of the type of BCD adder discussed in the previous pages. Figure 7.29 shows the logic diagram, and it follows the generalized cascaded arrangement discussed earlier and shown in Fig. 7.22 for a three-digit BCD adder. The BCD adder of Fig. 7.21 can be used to add four-bit BCD equivalents of two single-digit decimal numbers. A cascaded arrangement of two such stages, where the output *C* of Fig. 7.21 (CARRY-OUT) is fed to the CARRY-IN of the second stage, is shown in Fig. 7.29. In terms of IC type numbers, IC 7483 can be used for four-bit binary adders as shown in the diagram, IC 7408 can be used for implementing

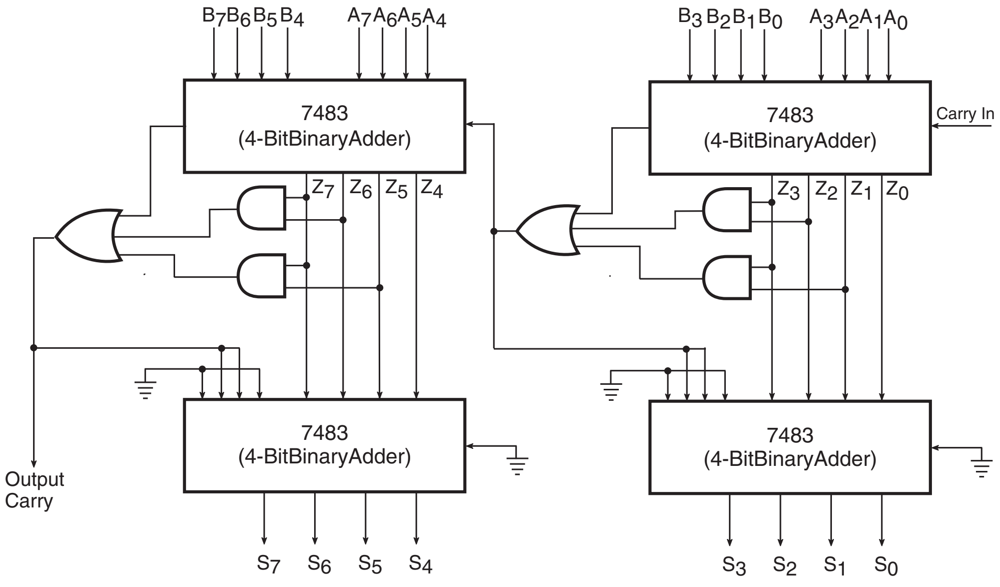


Figure 7.29 Example 7.6.

the required four two-input AND gates (IC 7408 is a quad two-input AND) and IC 7432 can be used to implement the required two three-input OR gates. IC 7432 is a quad two-input OR. Two two-input OR gates can be connected in cascade to get a three-input OR gate.

7.6 Carry Propagation–Look-Ahead Carry Generator

The four-bit binary adder described in the previous pages can be used to add two four-bit binary numbers. Multiple numbers of such adders are used to perform addition operations on larger-bit binary numbers. Each of the adders is composed of four full adders (FAs) connected in cascade. The block schematic arrangement of a four-bit adder is reproduced in Fig. 7.30(a) for reference and further discussion. This type of adder is also called a parallel binary adder because all the bits of the augend and addend are present and are fed to the full adder blocks simultaneously. Theoretically, the addition operation in various full adders takes place simultaneously. What is of importance and interest to users, more so when they are using a large number of such adders in their overall computation system, is whether the result of addition and carry-out are available to them at the same time. In other words, we need to see if this addition operation is truly parallel in nature. We will soon see that it is not. It is in fact limited by what is known as *carry propagation time*. Refer to Figs 7.30(a) and (b). Figure 7.30(b) shows the logic diagram of a full adder. Here, C_i and C_{i+1} are the input and output CARRY; P_i and G_i are two new binary variables called CARRY PROPAGATE and CARRY GENERATE and will be addressed a little later.

For $i=1$, the diagram in Fig. 7.30(b) is that of the LSB full adder of Fig. 7.30(a). We can see here that C_2, which is the output CARRY of FA (1) and the input CARRY for FA (2), will appear at the output after a minimum of two gate delays plus delay due to the half adder after application of A_i, B_i and C_i inputs.

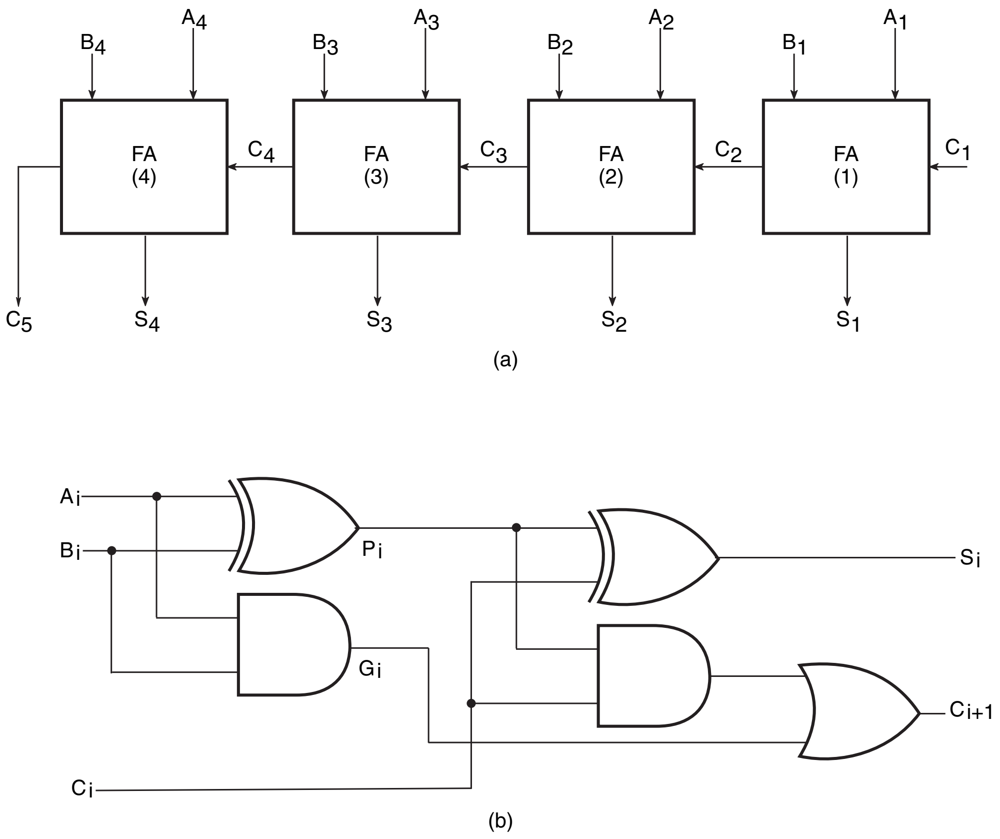


Figure 7.30 Four-bit binary adder.

The steady state of C_2 will be delayed by two gate delays after the appearance of C_1. Similarly, C_3 and C_4 steady state will be four and six gate delays respectively after C_1. And final carry C_5 will appear after eight gate delays.

Extending it a little further, let us assume that we are having a cascade arrangement of two four-bit adders to be able to handle eight-bit numbers. Now, C_5 will form the input CARRY to the second four-bit adder. The final output CARRY C_9 will now appear after 16 gate delays. This carry propagation delay limits the speed with which two numbers are added. The outputs of any such adder arrangement will be correct only if signals are given enough time to propagate through gates connected between input and output. Since subtraction is also an addition process and operations like multiplication and division are also processes involving successive addition and subtraction, the time taken by an addition process is very critical.

One of the possible methods for reducing carry propagation delay time is to use faster logic gates. But then there is a limit below which the gate delay cannot be reduced. There are other hardware-related techniques, the most widely used of which is the concept of look-ahead carry. This concept attempts to look ahead and generate the carry for a certain given addition operation that would otherwise have resulted from some previous operation. In order to explain the concept, let us define two new binary variables: P_i called CARRY PROPAGATE and G_i called CARRY GENERATE. Binary variable G_i is so called as it generates a carry whenever A_i and B_i are '1'. Binary variable P_i is called CARRY PROPAGATE as it is instrumental in propagation of C_i to C_{i+1}. CARRY, SUM, CARRY GENERATE and CARRY PROPAGATE parameters are given by the following expressions:

$$P_i = A_i \oplus B_i \tag{7.18}$$

$$G_i = A_i.B_i \tag{7.19}$$

$$S_i = P_i \oplus C_i \tag{7.20}$$

$$C_{i+1} = P_i.C_i + G_i \tag{7.21}$$

In the next step, we write Boolean expressions for the CARRY output of each full adder stage in the four-bit binary adder. We obtain the following expressions:

$$C_2 = G_1 + P_1.C_1 \tag{7.22}$$

$$C_3 = G_2 + P_2.C_2 = G_2 + P_2.(G_1 + P_1.C_1) = G_2 + P_2.G_1 + P_1.P_2.C_1 \tag{7.23}$$

$$C_4 = G_3 + P_3.C_3 = G_3 + P_3.(G_2 + P_2.G_1 + P_1.P_2.C_1)$$

$$C_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_1.P_2.P_3.C_1 \tag{7.24}$$

From the expressions for C_2, C_3 and C_4 it is clear that C_4 need not wait for C_3 and C_2 to propagate. Similarly, C_3 does not wait for C_2 to propagate. Hardware implementation of these expressions gives us a kind of look-ahead carry generator. A look-ahead carry generator that implements the above expressions using AND-OR logic is shown in Fig. 7.31.

Figure 7.32 shows the four-bit adder with the look-ahead carry concept incorporated. The block labelled *look-ahead carry generator* is similar to that shown in Fig. 7.31. The logic gates shown to the left of the block represent the input half-adder portion of various full adders constituting the four-bit adder. The EX-OR gates shown on the right are a portion of the output half-adders of various full adders.

All sum outputs in this case will be available at the output after a delay of two levels of logic gates. 74182 is a typical look-ahead carry generator IC of the TTL logic family. This IC can be used to generate relevant carry inputs for four four-bit binary adders connected in cascade to perform operation on two 16-bit numbers. Of course, the four-bit adders should be of the type so as to produce CARRY GENERATE and CARRY PROPAGATE outputs. Figure 7.33 shows the arrangement. In the figure shown, C_n is the CARRY input, G_0, G_1, G_2 and G_3 are CARRY GENERATE inputs for 74182 and P_0, P_1, P_2 and P_3 are CARRY PROPAGATE inputs for 74182. C_{n+x}, C_{n+y} and C_{n+z} are the CARRY outputs generated by 74182 for the four-bit adders. The G and P outputs of 74182 need to be cascaded. Figure 7.34 shows the arrangement needed for adding two 64-bit numbers.

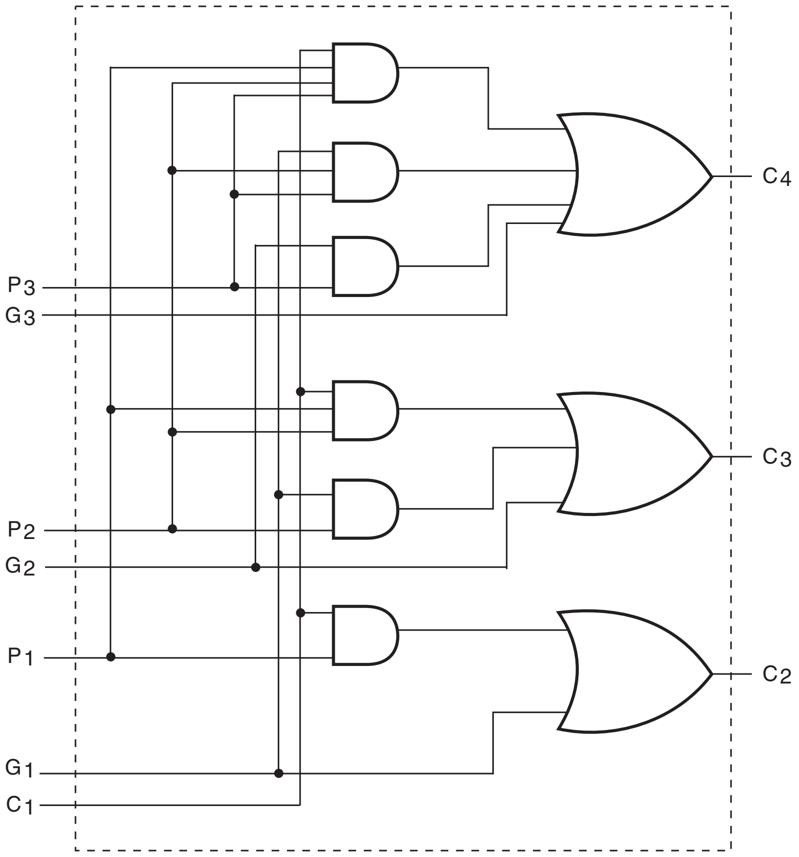


Figure 7.31 Look-ahead carry generator.

Example 7.7

If the CARRY GENERATE G_i and CARRY PROPAGATE P_i are redefined as $P_i = (A_i + B_i)$ and $G_i = A_iB_i$, show that the CARRY output C_{i+1} and the SUM output S_i of a full adder can be expressed by the following Boolean functions:

$$C_{i+1} = (\overline{\overline{C_i}.\overline{G_i} + \overline{P_i}}) = G_i + P_i.C_i \quad \text{and} \quad S_i = (P_i.\overline{G_i}) \oplus C_i$$

Solution

$$C_{i+1} = (\overline{\overline{C_i}.\overline{G_i} + \overline{P_i}}) = [\overline{\overline{C_i}.(\overline{A_{i.}B_i}) + (\overline{A_i + B_i})}]$$

$$= [\overline{\overline{C_i}.(\overline{A_i.B_i})}.(A_i + B_i)]$$

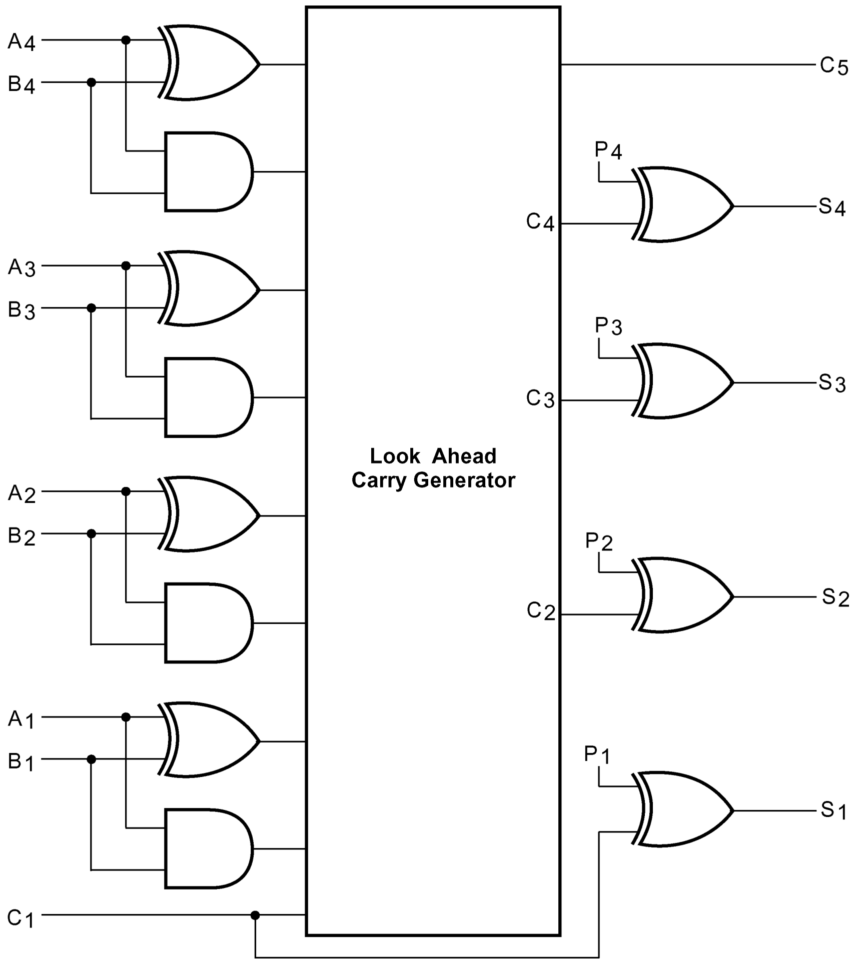


Figure 7.32 Four-bit full adder with a look-ahead carry generator.

$$= (C_i + A_i.B_i).(A_i + B_i) = C_i.(A_i + B_i) + A_i.B_i.(A_i + B_i)$$
$$= C_i.(A_i + B_i) + A_i.B_i = P_i.C_i + G_i$$

$$S_i = (A_i \oplus B_i) \oplus C_i = (\overline{A_i}.B_i + A_i.\overline{B_i}) \oplus C_i$$

Also

$$(P_i.\overline{G_i}) \oplus C_i = [(A_i + B_i).(\overline{A_i.B_i})] \oplus C_i$$
$$= [(A_i + B_i).(\overline{A_i} + \overline{B_i})] \oplus C_i = (\overline{A_i}.B_i + A_i.\overline{B_i}) \oplus C_i$$

Therefore, $S_i = (P_i.\overline{G_i}) \oplus C_i$.

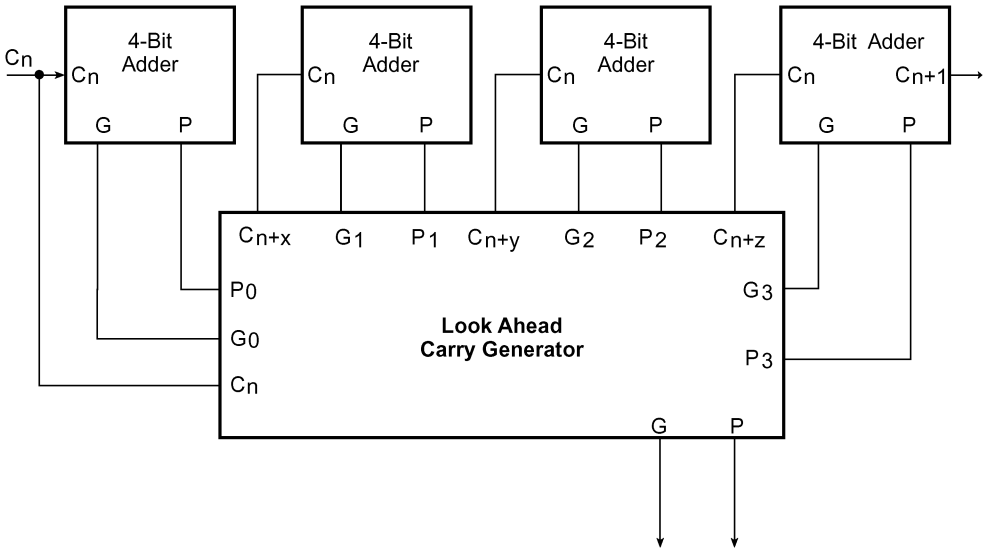


Figure 7.33 IC 74182 interfaced with four four-bit adders.

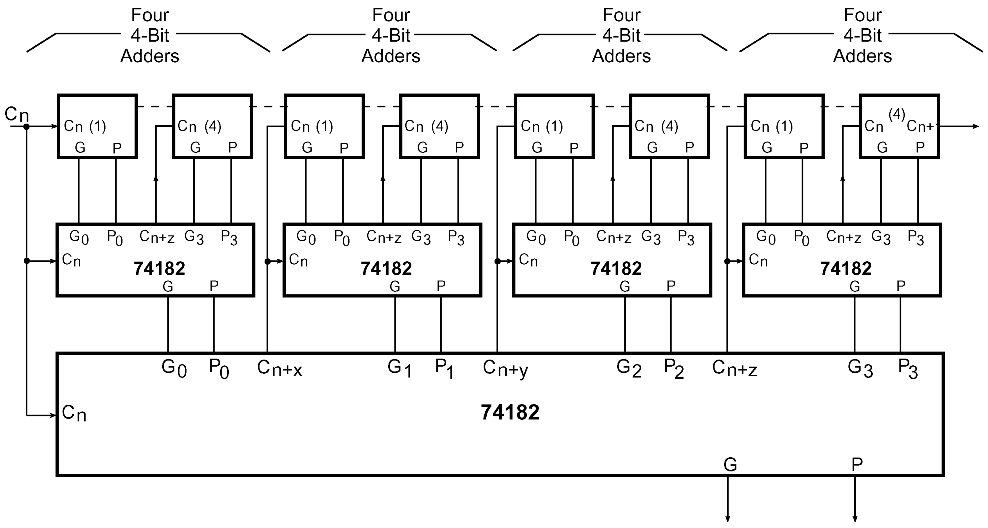


Figure 7.34 Look-ahead carry generation for adding 64-bit numbers.

7.7 Arithmetic Logic Unit (ALU)

The *arithmetic logic unit* (ALU) is a digital building block capable of performing both arithmetic as well as logic operations. Arithmetic logic units that can perform a variety of arithmetic operations such as addition, subtraction, etc., and logic functions such as ANDing, ORing, EX-ORing, etc., on two four-bit numbers are usually available in IC form. The function to be performed is selectable from *function select* pins. Some of the popular type numbers of ALU include 74181, 74381, 74382, 74582 (all from the TTL logic family) and 40181 (from the CMOS logic family). Functional details of these ICs are given in the latter part of the chapter under the heading of *Application-Relevant Information*. More than one such IC can always be connected in cascade to perform arithmetic and logic operations on larger bit numbers.

7.8 Multipliers

Multiplication of binary numbers is usually implemented in microprocessors and microcomputers by using *repeated addition and shift* operations. Since the binary adders are designed to add only two binary numbers at a time, instead of adding all the partial products at the end, they are added two at a time and their sum is accumulated in a register called the *accumulator register*. Also, when the multiplier bit is '0', that very partial product is ignored, as an all '0' line does not affect the final result. The basic hardware arrangement of such a binary multiplier would comprise shift registers for the multiplicand and multiplier bits, an accumulator register for storing partial products, a binary parallel adder and a clock pulse generator to time various operations.

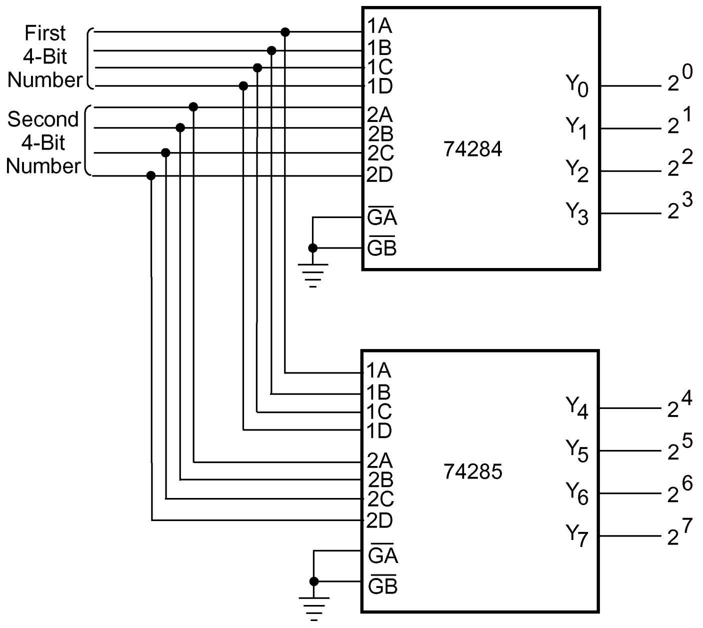


Figure 7.35 4 × 4 bit multiplier.

Binary multipliers are also available in IC form. Some of the popular type numbers in the TTL family include 74261 which is a 2 × 4 bit multiplier (a four-bit multiplicand designated as B_0,B_1,B_2, B_3 and B_4, and a two-bit multiplier designated as M_0, M_1 and M_2).

The MSBs B_4 and M_2 are used to represent signs. 74284 and 74285 are 4 × 4 bit multipliers. They can be used together to perform high-speed multiplication of two four-bit numbers. Figure 7.35 shows the arrangement. The result of multiplication is often required to be stored in a register. The size of this register (accumulator) depends upon the number of bits in the result, which at the most can be equal to the sum of the number of bits in the multiplier and multiplicand. Some multiplier ICs have an in-built register.

Many microprocessors do not have in their ALU the hardware that can perform multiplication or other complex arithmetic operations such as division, determining the square root, trigonometric functions, etc. These operations in these microprocessors are executed through software. For example, a multiplication operation may be accomplished by using a software program that does multiplication through repeated execution of addition and shift instructions. Other complex operations mentioned above can also be executed with similar programs. Although the use of software reduces the hardware needed in the microprocessor, the computation time in general is higher in the case of software-executed operations when compared with the use of hardware to perform those operations.

7.9 Magnitude Comparator

A *magnitude comparator* is a combinational circuit that compares two given numbers and determines whether one is equal to, less than or greater than the other. The output is in the form of three binary variables representing the conditions $A = B$, $A > B$ and $A < B$, if A and B are the two numbers being compared. Depending upon the relative magnitude of the two numbers, the relevant output changes state. If the two numbers, let us say, are four-bit binary numbers and are designated as ($A_3\,A_2\,A_1\,A_0$) and ($B_3\,B_2\,B_1\,B_0$), the two numbers will be equal if all pairs of significant digits are equal, that is, $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$. In order to determine whether A is greater than or less than B, we inspect the relative magnitude of pairs of significant digits, starting from the most significant position. The comparison is done by successively comparing the next adjacent lower pair of digits if the digits of the pair under examination are equal. The comparison continues until a pair of unequal digits is reached. In the pair of unequal digits, if $A_i = 1$ and $B_i = 0$, then $A > B$, and if $A_i = 0$, $B_i = 1$ then $A < B$. If X, Y and Z are three variables respectively representing the $A = B$, $A > B$ and $A < B$ conditions, then the Boolean expression representing these conditions are given by the equations

$$X = x_3.x_2.x_1.x_0 \quad \text{where } x_i = A_i.B_i + \overline{A_i}.\overline{B_i} \tag{7.25}$$

$$Y = A_3.\overline{B_3} + x_3.A_2.\overline{B_2} + x_3.x_2.A_1.\overline{B_1} + x_3.x_2.x_1.A_0.\overline{B_0} \tag{7.26}$$

$$Z = \overline{A_3}.B_3 + x_3.\overline{A_2}.B_2 + x_3.x_2.\overline{A_1}.B_1 + x_3.x_2.x_1.\overline{A_0}.B_0 \tag{7.27}$$

Let us examine equation (7.25). x_3 will be '1' only when both A_3 and B_3 are equal. Similarly, conditions for x_2, x_1 and x_0 to be '1' respectively are equal A_2 and B_2, equal A_1 and B_1 and equal A_0 and B_0. ANDing of x_3, x_2, x_1 and x_0 ensures that X will be '1' when x_3, x_2, x_1 and x_0 are in the logic '1' state. Thus, $X = 1$ means that $A = B$. On similar lines, it can be visualized that equations (7.26) and

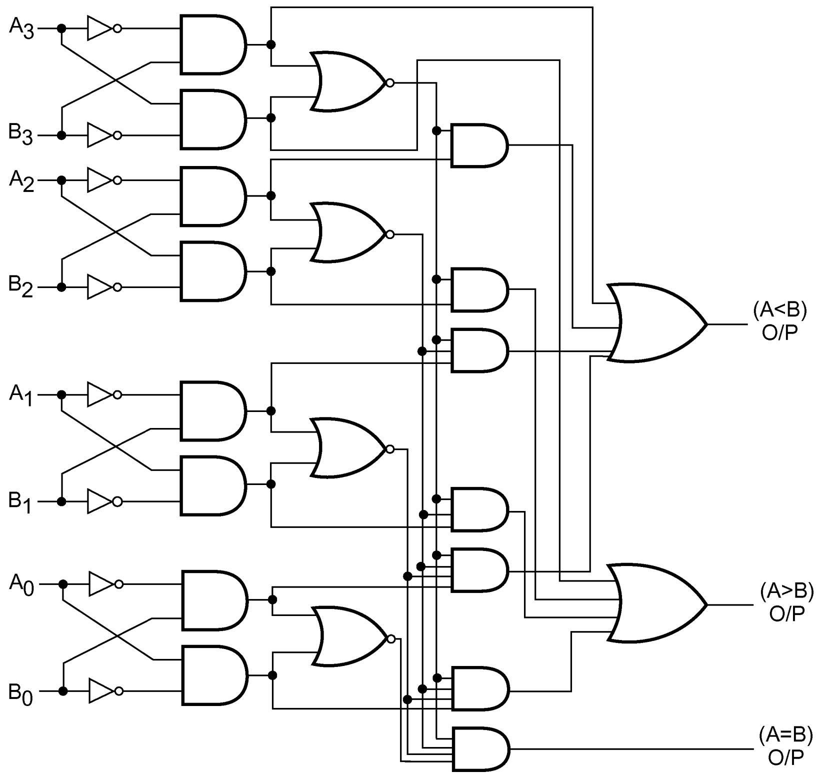


Figure 7.36 Four-bit magnitude comparator.

(7.27) respectively represent $A > B$ and $A < B$ conditions. Figure 7.36 shows the logic diagram of a four-bit magnitude comparator.

Magnitude comparators are available in IC form. For example, 7485 is a four-bit magnitude comparator of the TTL logic family. IC 4585 is a similar device in the CMOS family. 7485 and 4585 have the same pin connection diagram and functional table. The logic circuit inside these devices determines whether one four-bit number, binary or BCD, is *less than*, *equal to* or *greater than* a second four-bit number. It can perform comparison of straight binary and straight BCD (8-4-2-1) codes. These devices can be cascaded together to perform operations on larger bit numbers without the help of any external gates. This is facilitated by three additional inputs called cascading or expansion inputs available on the IC. These cascading inputs are also designated as $A = B$, $A > B$ and $A < B$ inputs. Cascading of individual magnitude comparators of the type 7485 or 4585 is discussed in the following paragraphs. IC 74AS885 is another common magnitude comparator. The device is an eight-bit magnitude comparator belonging to the advanced Schottky TTL family. It can perform high-speed arithmetic or logic comparisons on two eight-bit binary or 2's complement numbers and produces two fully decoded decisions at the output about one number being either greater than or less than the other. More than one of these devices can also be connected in a cascade arrangement to perform comparison of numbers of longer lengths.

7.9.1 Cascading Magnitude Comparators

As outlined earlier, magnitude comparators available in IC form are designed in such a way that they can be connected in a cascade arrangement to perform comparison operations on numbers of longer lengths. In cascade arrangement, the $A = B$, $A > B$ and $A < B$ outputs of a stage handling less significant bits are connected to corresponding inputs of the next adjacent stage handling more significant bits. Also, the stage handling least significant bits must have a HIGH level at the $A = B$ input. The other two cascading inputs ($A > B$ and $A < B$) may be connected to a LOW level. We will illustrate the concept by showing the arrangement of an eight-bit magnitude comparator using two four-bit magnitude comparators of the type 7485 or 4585. Figure 7.37 shows the cascaded arrangement of the two comparators. We can see the three comparison outputs of the comparator handling less significant four bits of the two numbers being connected to the corresponding cascading inputs of the comparator handling more significant four bits of the two numbers. Also, cascading inputs of the less significant comparator have been connected to a HIGH or LOW level as per the guidelines mentioned in the previous paragraph.

Operation of this circuit can be explained by considering the functional table of IC 7485 or IC 4585 as shown in Table 7.2. The two numbers being compared here are $(A_7 \ldots A_0)$ and $(B_7 \ldots B_0)$. The less significant comparator handles (A_3, A_2, A_1, A_0) and (B_3, B_2, B_1, B_0), and the more significant comparator handles (A_7, A_6, A_5, A_4) and (B_7, B_6, B_5, B_4). Let us take the example of the two numbers being such that $A_7 > B_7$. From the first-row entry of the function table it is clear that, irrespective of the status of other bits of the more significant comparator, and also regardless of the status of its cascading inputs, the final output produces a HIGH at the $A > B$ output and a LOW at the $A < B$ and $A = B$ outputs. Since the status of cascading inputs of the more significant comparator depends upon the status of comparison bits of the less significant comparator, the cascade arrangement produces the correct output for $A_7 > B_7$ regardless of the status of all other comparison bits. On similar lines, the circuit produces a valid output for any given status of comparison bits.

Example 7.8

Design a two-bit magnitude comparator. Also, write relevant Boolean expressions.

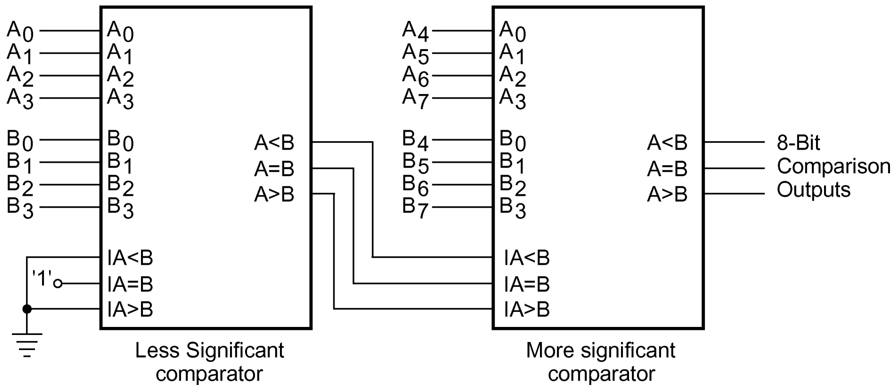


Figure 7.37 Cascading of individual magnitude comparators.

Table 7.2 Functional table of IC 7485 or IC 4585.

Comparison inputs				Cascading inputs			Outputs		
A_3,B_3	A_2,B_2	A_1,B_1	A_0,B_0	A>B	A<B	A=B	A>B	A<B	A=B
$A_3 > B_3$	X	X	X	X	X	X	HIGH	LOW	LOW
$A_3 < B_3$	X	X	X	X	X	X	LOW	HIGH	LOW
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	HIGH	LOW	LOW
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	LOW	HIGH	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	HIGH	LOW	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	LOW	HIGH	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	HIGH	LOW	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	LOW	HIGH	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	HIGH	LOW	LOW	HIGH	LOW	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	LOW	HIGH	LOW	LOW	HIGH	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	LOW	LOW	HIGH	LOW	LOW	HIGH
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	X	X	HIGH	LOW	LOW	HIGH
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	HIGH	HIGH	LOW	LOW	LOW	LOW
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	LOW	LOW	LOW	HIGH	HIGH	LOW

Solution

Let $A\,(A_1A_0)$ and $B\,(B_1B_0)$ be the two numbers. If X, Y and Z represent the conditions $A = B, A > B$ and $A < B$ respectively (that is, $X = 1$, $Y = 0$ and $Z = 0$ for A =B; X= 0, $Y = 1$ and $Z = 0$ for $A > B$; and $X = 0$, $Y = 0$ and $Z = 1$ for $A < B$), then expressions for X, Y and Z can be written as follows:

$$X = x_1.x_0 \text{ where } x_1 = A_1.B_1 + \overline{A_1}.\overline{B_1} \quad \text{and} \quad x_0 = A_0.B_0 + \overline{A_0}.\overline{B_0}$$

$$Y = A_1.\overline{B_1} + x_1.A_0.\overline{B_0}$$

$$Z = \overline{A_1}.B_1 + x_1.\overline{A_0}.B_0$$

Figure 7.38 shows the logic diagram of the two-bit comparator.

Example 7.9

Hardware-implement a three-bit magnitude comparator having one output that goes HIGH when the two three-bit numbers are equal. Use only NAND gates.

Solution

The equivalence condition of the two three-bit numbers is given by the equation $X = x_2.x_1.x_0$, where $x_2 = A_2.B_2 + \overline{A_2}.\overline{B_2}$, $x_1 = A_1.B_1 + \overline{A_1}.\overline{B_1}$ and $x_0 = A_0.B_0 + \overline{A_0}.\overline{B_0}$.

Figure 7.39 shows the logic diagram. x_2, x_1 and x_0 are respectively given by EX-NOR operation of (A_2, B_2), (A_1, B_1) and (A_0, B_0). These are then ANDed to get X.

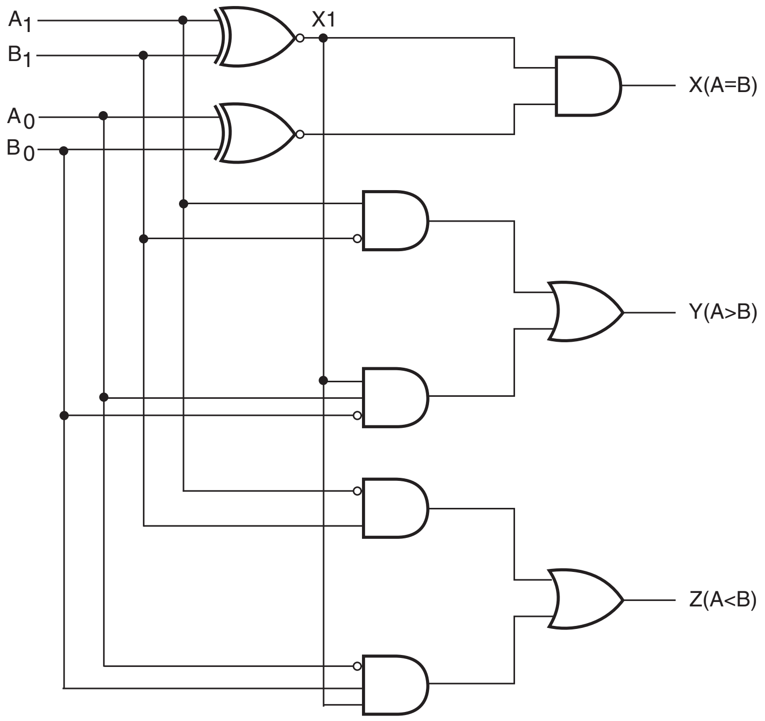


Figure 7.38 Solution to example 7.8.

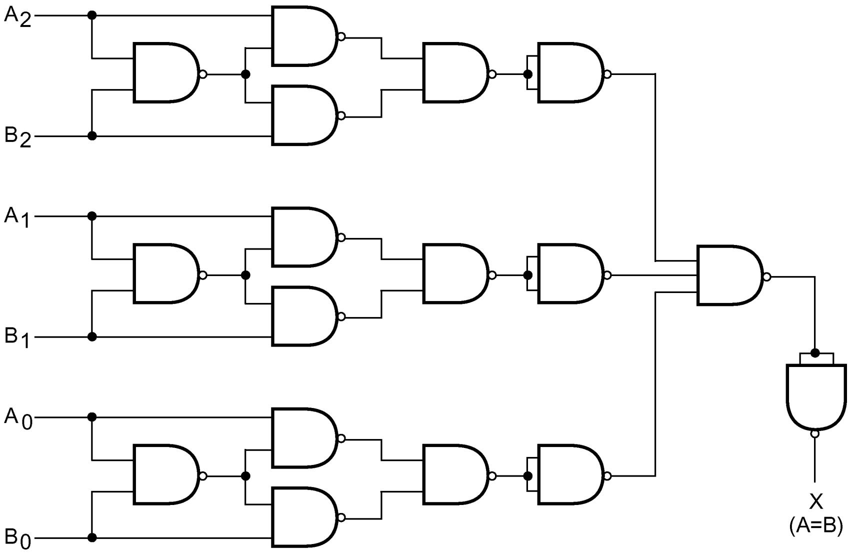


Figure 7.39 Solution to example 7.9.

Table 7.3 Commonly used IC type numbers used for arithmetic operations.

IC type number	Function	Logic family
7483	Four-bit full adder	TTL
7485	Four-bit magnitude comparator	TTL
74181	Four-Bit ALU and function generator	TTL
74182	Look-ahead carry generator	TTL
74183	Dual carry save full adder	TTL
74283	Four-bit full binary adder	TTL
74885	Eight-bit magnitude comparator	TTL
4008	Four-bit binary full adder	CMOS
4527	BCD rate multiplier	CMOS
4585	Four-bit magnitude comparator	CMOS
40181	Four-bit arithmetic logic unit	CMOS
40182	Look-ahead carry generator	CMOS
10179	Look-ahead carry block	ECL
10180	Dual high-speed two-bit adder/subtractor	ECL
10181	Four-bit arithmetic logic unit/function generator	ECL
10182	Four-bit arithmetic logic unit/function generator	ECL
10183	4×2 multiplier	ECL

7.10 Application-Relevant Information

Table 7.3 lists commonly used IC type numbers used for arithmetic operations. Application-relevant information such as pin connection diagrams, truth tables, etc., in respect of the more popular of these type numbers is given on the companion website.

Review Questions

1. How do you characterize or define a combinational circuit? How does it differ from a sequential circuit? Give two examples each of combinational and sequential logic devices.
2. Beginning with the statement of the problem, outline different steps involved in the design of a suitable combinational logic circuit to implement the hardware required to solve the given problem.
3. Write down Boolean expressions representing the SUM and CARRY outputs in terms of three input binary variables to be added. Design a suitable combinational circuit to hardware-implement the design using NAND gates only.
4. Draw the truth table of a full subtractor circuit. Write a minterm Boolean expression for DIFFERENCE and BORROW outputs in terms of minuend variable, subtrahend variable and BORROW-IN. Minimize the expressions and implement them in hardware.
5. Draw the logic diagram of a three-digit BCD adder and briefly describe its functional principle.
6. Briefly describe the concept of look-ahead carry generation with respect to its use in adder circuits. What is its significance while implementing hardware for addition of binary numbers of longer lengths?
7. With the help of a block schematic of the logic circuit, briefly describe how individual four-bit magnitude comparators can be used in a cascade arrangement to perform magnitude comparison of binary numbers of longer lengths.

Problems

1. A, B, B_{in}, D and B_{out} are respectively the minuend, the subtrahend, the BORROW-IN, the DIFFERENCE output and the BORROW-OUT in the case of a full subtractor. Determine the bit status of D and B_{out} for the following values of A, B and B_{in}:

   (a) $A = 0$, $B = 1$, $B_{in} = 1$
   (b) $A = 1$, $B = 1$, $B_{in} = 0$
   (c) $A = 1$, $B = 1$, $B_{in} = 1$
   (d) $A = 0$, $B = 0$, $B_{in} = 1$

   (a) $D = 0$, $B_{out} = 1$; (b) $D = 0$, $B_{out} = 0$; (c) $D = 1$, $B_{out} = 1$; (d) $D = 1$, $B_{out} = 1$

2. Determine the number of half and full adder circuit blocks required to construct a 64-bit binary parallel adder. Also, determine the number and type of additional logic gates needed to transform this 64-bit adder into a 64-bit adder–subtractor.

   For a 64-bit adder: HA=1, FA=63
   For a 64-bit adder–subtractor: HA = 1, FA = 63, EX-OR gates = 64

3. If the minuend, subtrahend and BORROW-IN bits are respectively applied to the Augend, Addend and the CARRY-IN inputs of a full adder, prove that the SUM output of the full adder will produce the correct DIFFERENCE output.
4. Prove that the logic diagram of Fig. 7.40 performs the function of a half-subtractor provided that Y represents the DIFFERENCE output and X represents the BORROW output.
5. Determine the number of 7483s (four-bit binary adders) and 7486s (quad two-input EX-OR gates) required to design a 16-bit adder–subtractor circuit.

   Number of 7483 = 4; number of 7486 = 4

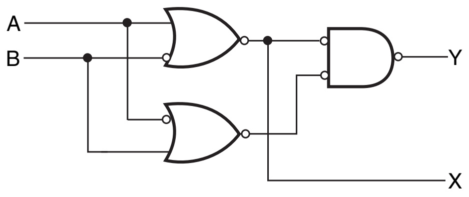


Figure 7.40 Problem 4.

6. The objective is to design a BCD adder circuit using four-bit binary adders and additional combinational logic. If the decimal numbers to be added can be anywhere in the range from 0 to 9999, determine the number of four-bit binary adder circuit blocks of type IC 7483 required to do the job.

   Number of four-bit adders = 8

Further Reading

1. Koren, I. (2001) *Computer Arithmetic Algorithms*, A. K. Peters Ltd, Natick, MA, USA.
2. Ercegovac, M. D. and Lang, T. (2003) *Digital Arithmetic*, Morgan Kaufmann Publishers, CA, USA.
3. Rafiquzzaman, M. (2005) *Fundamentals of Digital Logic and Microcomputer Design*, Wiley-Interscience, New York, USA.
4. Morris Mano, M. and Kime, C. R. (2003) *Logic and Computer Design Fundamentals*, Prentice-Hall, USA.
5. Tokheim, R. L. (1994) *Schaum's Outline Series of Digital Principles*, McGraw-Hill Companies Inc., USA.
6. Tocci, R. J. (2006) *Digital Systems – Principles and Applications*, Prentice-Hall Inc., NJ, USA.
7. Malvino, A. P. and Leach, D. P. (1994) *Digital Principles and Applications*, McGraw-Hill Book Company, USA.