

## LESSON 7 ARITHMETIC CIRCUITS

### 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

$$\text{SUM } S = A.\bar{B} + \bar{A}.B \quad (7.5)$$

$$\text{CARRY } C = A.B \quad (7.6)$$

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

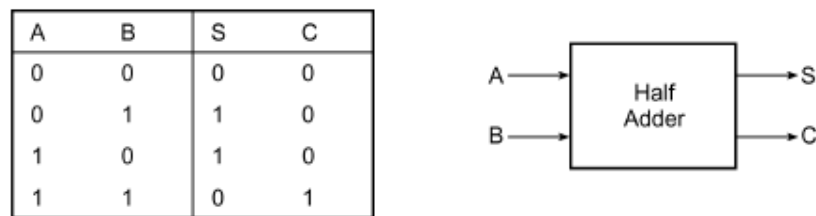


Figure 7.4 Truth table of a half-adder.

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 on Boolean algebra how various logic gates can be implemented in the form of either only NAND gates or NOR gates. 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.

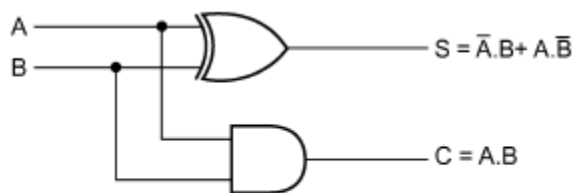


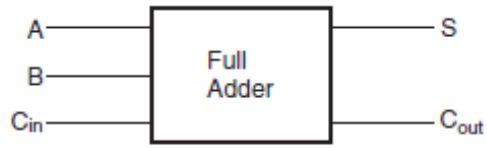
Figure 7.5 Logic implementation of a half-adder.

### 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 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.



A	B	C <sub>in</sub>	SUM (S)	C <sub>out</sub>
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.

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<sub>out</sub>):

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

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

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

$$C_{out} = B.C_{in} + A.B + A.C_{in} \quad (7.9)$$

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:

$$\begin{aligned} S &= \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(A.B + \overline{A}.\overline{B}) \\ S &= \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(\overline{\overline{A}.B + A.\overline{B}}) \end{aligned} \quad (7.10)$$

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

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

AB \ C <sub>in</sub>		
	$\overline{C}_{in}$	$C_{in}$
$\overline{A} \overline{B}$		1
$\overline{A} B$	1	
$A \overline{B}$		1
$A B$	1	

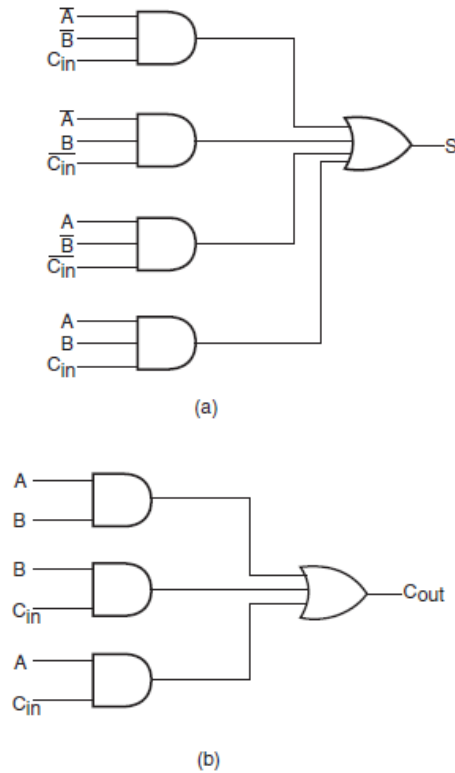
(a)

AB \ C <sub>in</sub>		
	$\overline{C}_{in}$	$C_{in}$
$\overline{A} \overline{B}$		
$\overline{A} B$		1
$A \overline{B}$	1	1
$A B$		1

(b)

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

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



**Figure 7.9** Logic circuit diagram of a full adder.

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.

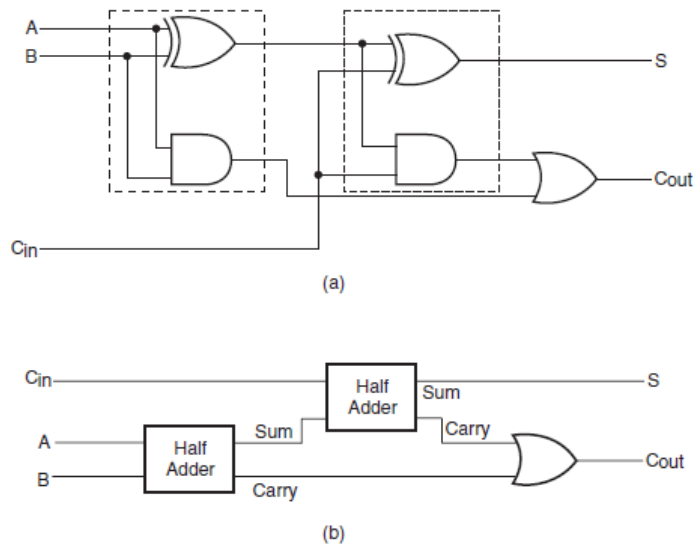


Figure 7.10 Logic implementation of a full adder with half-adders.

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.

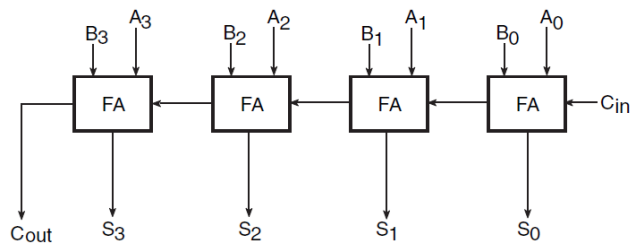


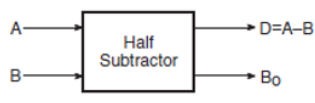
Figure 7.11 Four-bit binary adder.

### 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.



A	B	D	B <sub>0</sub>
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Figure 7.12 Half-subtractor.

The Boolean expressions for the two outputs are given by the equations

$$D = \bar{A}.B + A.\bar{B} \quad (7.12)$$

$$B_0 = \bar{A}.B \quad (7.13)$$

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 an EX-OR gate, the expression for the BORROW output (B<sub>0</sub>) 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).

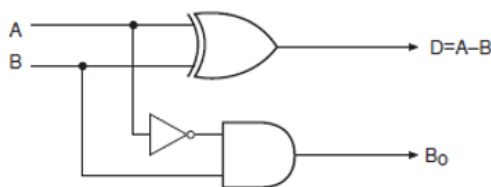
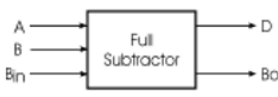


Figure 7.13 Logic diagram of a half-subtractor.

### 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<sub>in</sub>. There are two outputs, namely the DIFFERENCE output D and the BORROW output B<sub>0</sub>. 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.



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

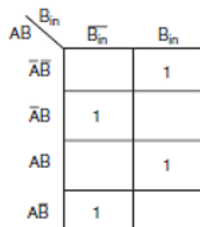
Figure 7.14 Truth table of a full subtractor.

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

$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in} \quad (7.14)$$

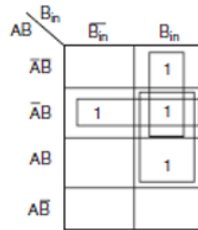
$$B_o = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in} \quad (7.15)$$

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  $B_o$ .



	$B_{in}$	$\overline{B}_{in}$	$B_{in}$
$\overline{A}\overline{B}$			1
$\overline{A}B$	1		
$AB$			1
$A\overline{B}$	1		

(a)



	$B_{in}$	$\overline{B}_{in}$	$B_{in}$
$\overline{A}\overline{B}$			1
$\overline{A}B$	1		1
$AB$			1
$A\overline{B}$			

(b)

Figure 7.15 Karnaugh maps for difference and borrow outputs.

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

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

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.

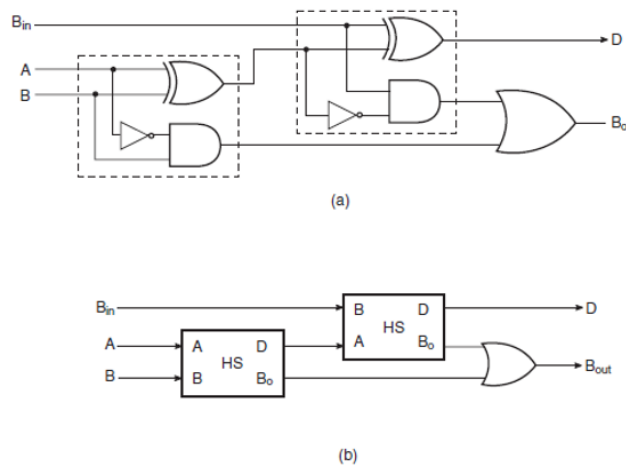


Figure 7.16 Logic implementation of a full subtractor with half-subtractors.

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.

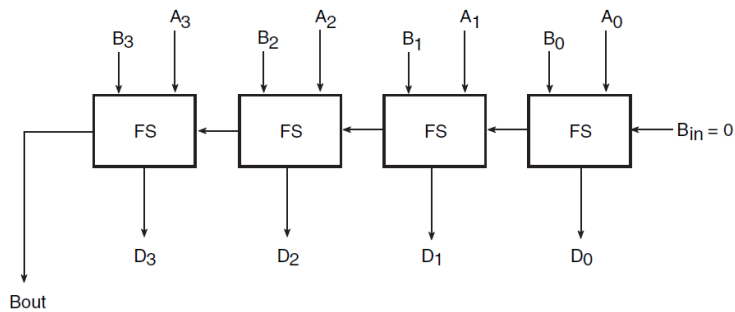


Figure 7.17 Four-bit subtractor.

### 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).

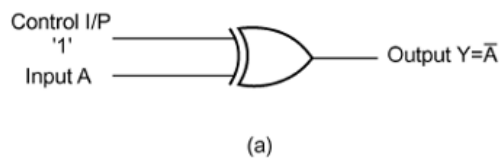
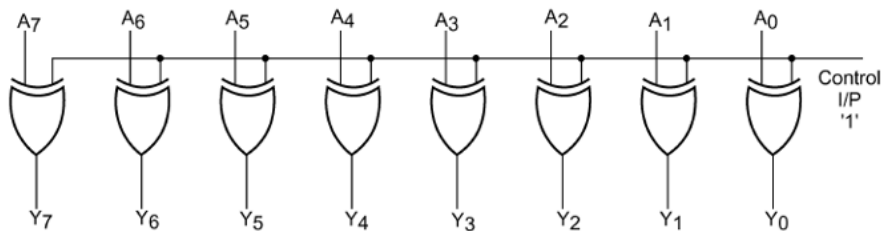


Fig. 7.18(a).



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.



(b)

Figure 7.18(b)

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 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 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.

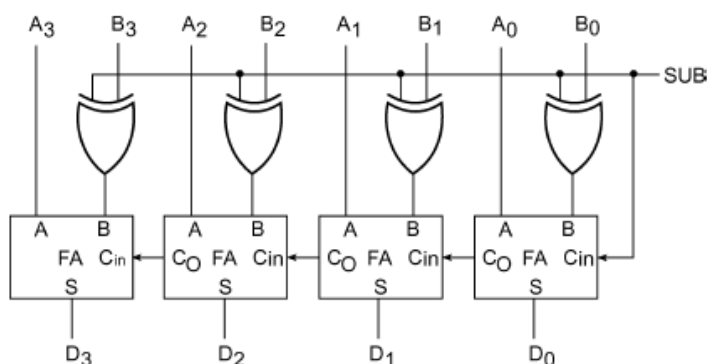


Figure 7.19 Four-bit adder-subtractor.

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.