Chapter 2: Arithmetic Circuits

In this chapter, 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, 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.

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 2.1 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

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

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

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

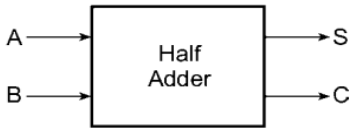


Figure 2.1 Truth table of a half-adder.

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 CARRY output is that of an AND gate.

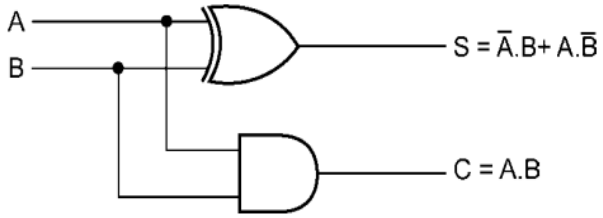
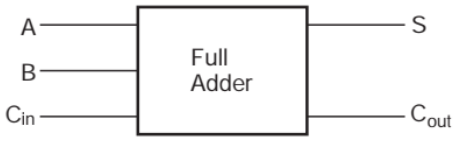


Figure 2.2 Logic implementation of a half-adder.

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. 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 2.3 shows the truth table of a full adder circuit showing all possible input combinations and corresponding outputs.



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

LSBs only.

Figure 2.3 Truth table of a full adder.

The Boolean expressions for the two output variables, that is, the SUM and CARRY outputs, in terms of input variables.

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

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

These expressions are then simplified by using the help of the Karnaugh mapping technique. Where it is obtained the S cannot be simplified any further more while

$$C_{out} = B.C_{in} + A.B + A.C_{in}$$

Figure 2.4 shows the logic circuit diagram of the full adder

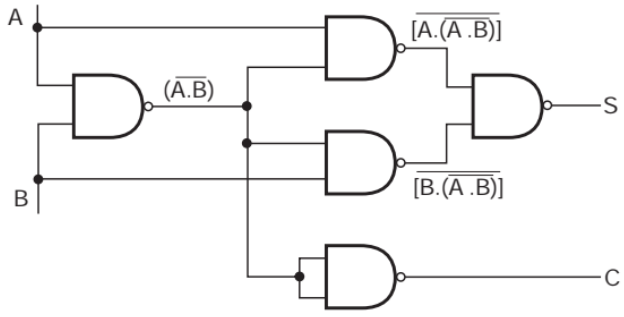


Figure 2.4 Half-adder implementation using NAND gates

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:

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

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

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

And there are circuit diagram are given by

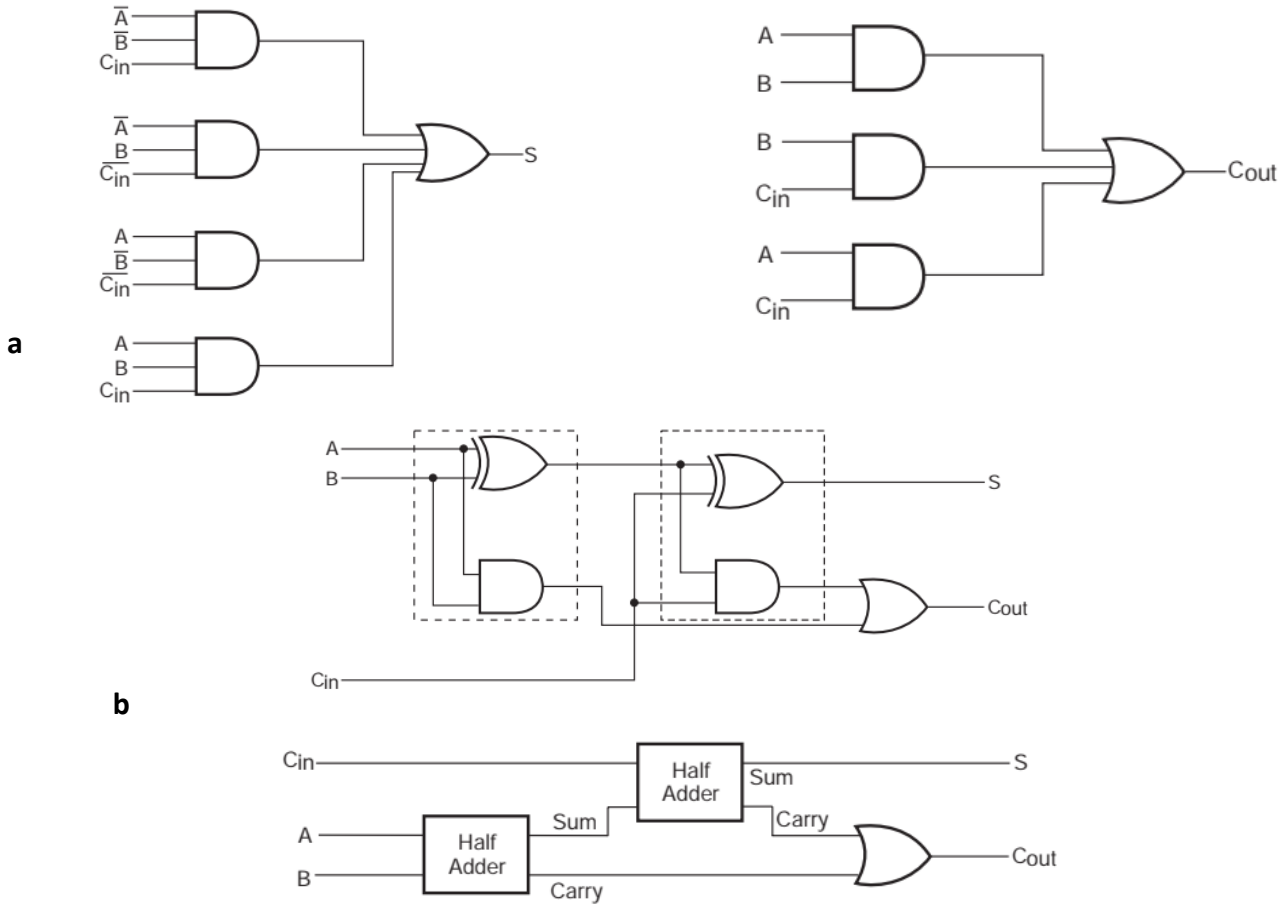


Figure 2.5 Logic circuit diagram of a full adder.

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. 2.5b to be connected in cascade. Figure 2.6 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.

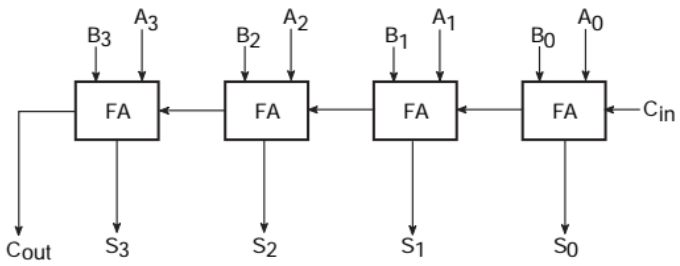


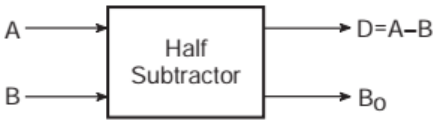
Figure 7.11 Four-bit binary adder.

Half-Subtractor

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. 2.7, explains this further. The Boolean expressions for the two outputs are given by the equations

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

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



A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Figure 7.12 Half-subtractor.

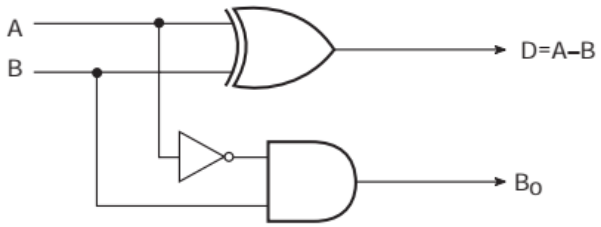


Figure 2.6 Logic implementation of a full adder with half-adders

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

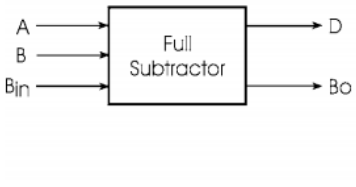
the half-subtractor is also similar to what we have for CARRY in the case of the half-adder.

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 Bin. There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. Figure 2.7 shows the truth table of a full subtractor.
The Boolean expressions for the two output variables are given by the equations

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

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



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 2.7 Truth table of a full subtractor.

From the two Karnaugh maps, it can be observed that no simplification is possible for the difference output D. The simplified expression for Bo is given by the equation:

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

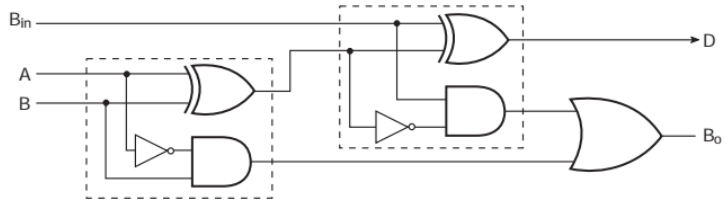
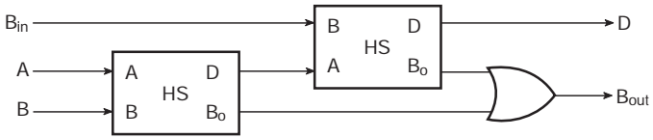


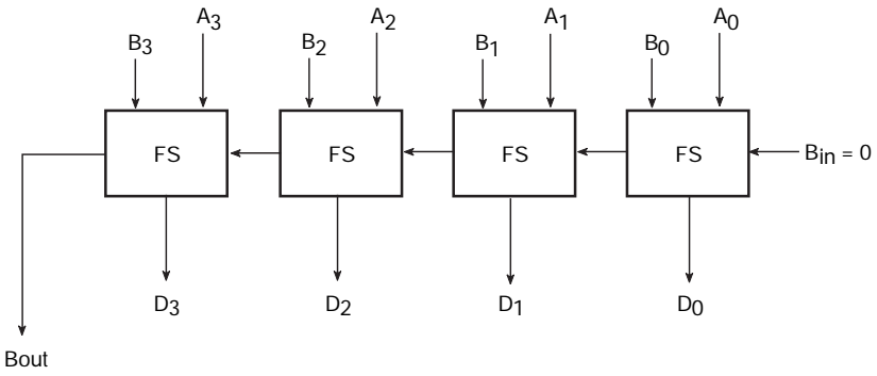
Figure 2.8 Logic implementation of a full subtractor with half-subtractors.

It is found that the expression for DIFFERENCE output D is the same as that for the SUM output. Also, the expression for BORROW output Bo is similar to the expression for CARRY-OUT Co.

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.



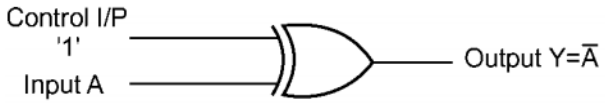
Again, more than one full subtractor can be connected in cascade to perform subtraction on two larger binary numbers as shown in Fig. 2.9. for a four-bit subtractor.



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. 2.10 (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 7.11 (b) shows an eight-bit controlled inverter of this type. When the control input is LOW, the output (Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0 is the same as the input (A7 A6 A5 A4 A3 A2 A1 A0). 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.

a)



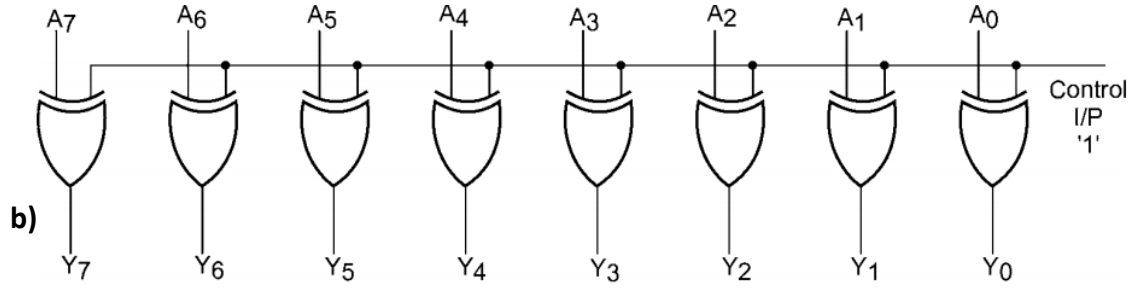
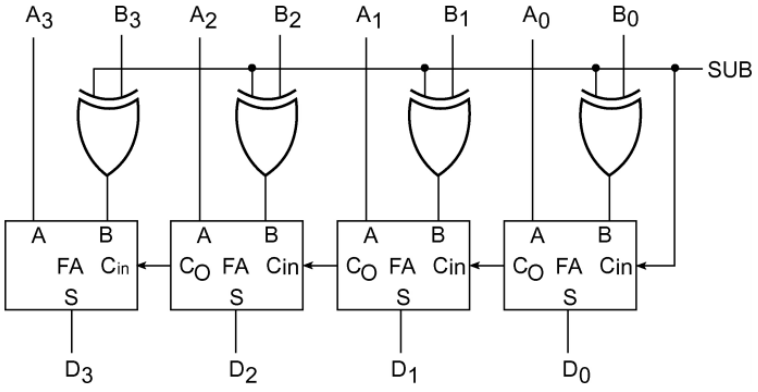


Figure 2.10 (a) One-bit controlled inverter and (b) eight-bit controlled inverter.

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 2.11 shows one such hardware arrangement.



When SUB=0 it operates as a four bit adder

When SUB=1 it operates a four bits substractor

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 2.12 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.

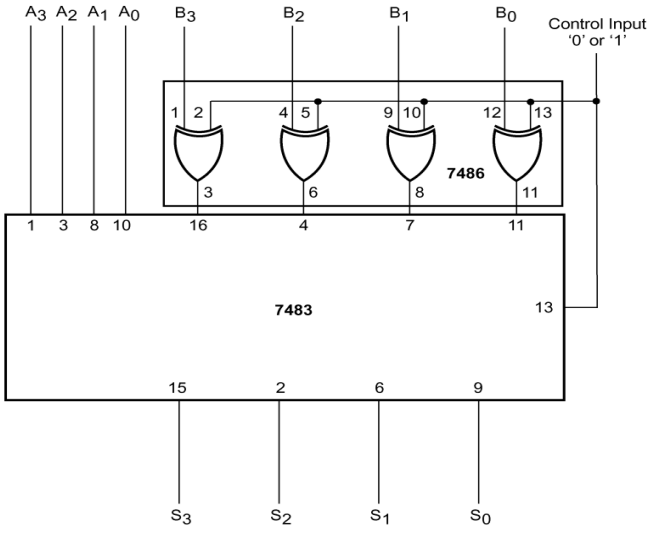


Figure 7.20 Four-bit adder-subtractor

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). 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 $(0001\ 1001)_{BCD}$. And if we restrict the output bits to the minimum required, the answer in BCD would be $(1\ 1001)_{BCD}$. Table 2.1 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

$$C = K + Z_3.Z_2 + Z_3.Z_1$$

Table 1.1 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

The four digit BCD adder logic circuit is given by:

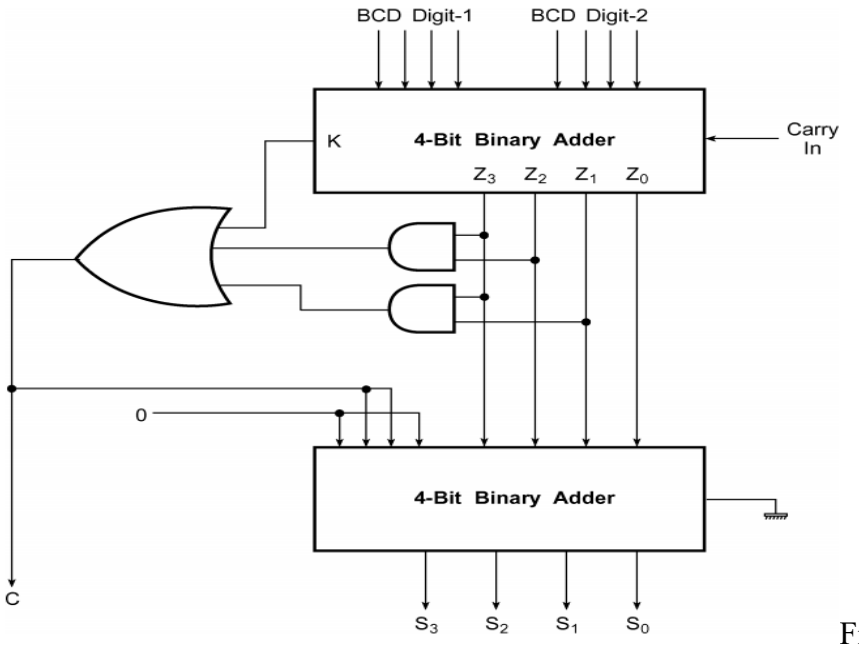


Fig 1.13

A cascade arrangement of single-digit BCD adder hardware can be used to perform the addition of multiple-digit BCD numbers. For example, a 3-digit BCD adder would require 3 such stages in cascade (Fig 2.14).

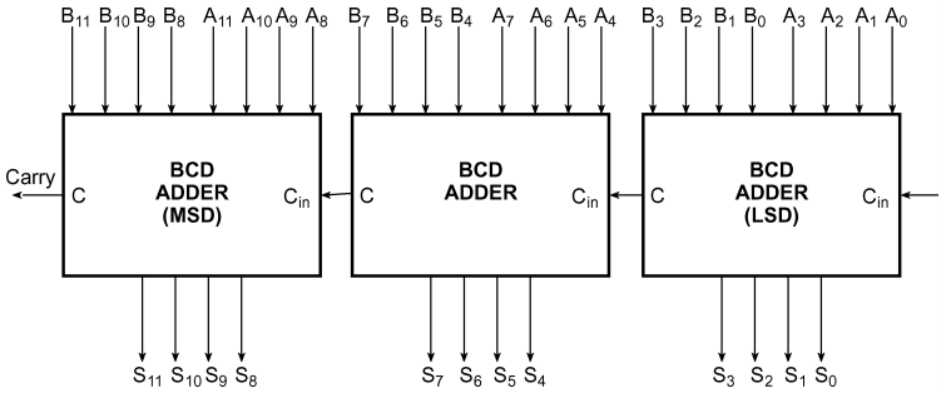
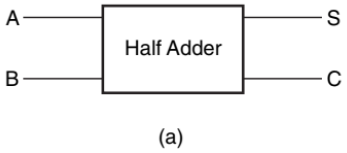


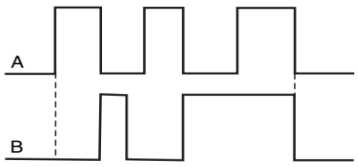
Fig 2.14

Exercice 1

For the half-adder circuit of Fig. 1(a), the inputs applied at A and B are as shown in Fig. 2(b). Plot the corresponding SUM and CARRY outputs on the same scale.



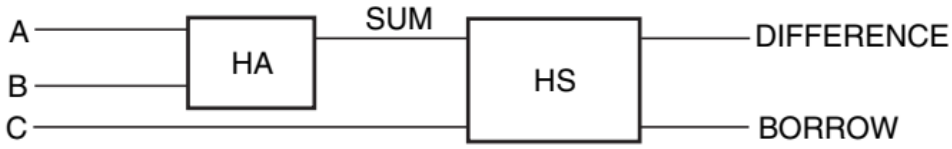
(a)



(b)

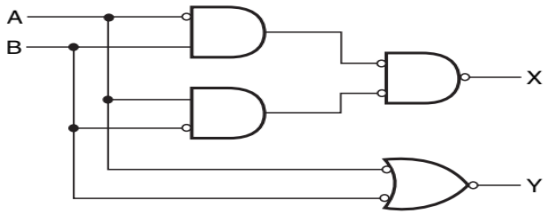
Exercice 2 Given the relevant Boolean expressions for half-adder and half-subtractor circuits, design a halfadder–subtractor circuit that can be used to perform either addition or subtraction on two one-bit numbers. The desired arithmetic operation should be selectable from a control input

Exercice 3 Refer to Fig. 2. Write the simplified Boolean expressions for DIFFERENCE and BORROW outputs.



Exercice 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.

Exercice 5 The logic diagram of Fig. 3 performs the function of a very common arithmetic building block. Identify the logic function.



Exercice 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.