Digital Circuits and Design

FIFTH EDITION

S. Salivahanan
Principal, SSN College of Engineering
Chennai

S. Arivazhagan
Principal, Mepco Schlenk Engineering College
Sivakasi

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trade mark of
Oxford University Press in the UK and in certain other countries.

Published in India by
Oxford University Press
Ground Floor, 2/11, Ansari Road, Daryaganj, New Delhi 110002, India

Previous editions published by Vikas Publishing House Pvt. Ltd

Fifth Edition published in 2018

ISBN-13: 978-0-19-948868-1
ISBN-10: 0-19-948868-1

Typeset in Times
by P.N. Computers, New Delhi
Printed in India by Magic International (P) Ltd., Greater Noida

Cover image: deepadesigns/Shutterstock

Preface to the Fifth Edition

Digital electronic circuits are mainly based on digital design. Today, digital circuits form the workhorses of the mobile phone, smart TV, digital camera, computer, GPS, and many other applications that demand digital electronic circuitry. Ever since the invention of transistors in 1947, there has been a growing dependency on digital electronic devices in our day-to-day life. The usage of system-based design tools from the mid-1980s has revolutionized the electronic industry worldwide. The functionality of any digital circuit can be written using a hardware description language (HDL) such as Verilog or VHDL and it can be synthesized into hardware using FPGA or CMOS technology. Every digital device in future will be smart enough to automatically communicate with other devices and also do work without human intervention. Moreover, any technological advancement in industry finds its way to engineering curriculum. This book provides an exposition of the fundamental concepts for the design of digital circuits and furnishes suitable methods and procedures for a variety of digital design applications. The enhanced usage of digital circuits in all disciplines of engineering has created an urge among students to have an in-depth knowledge about digital circuits and design.

About the Book

A single textbook dealing with the basics of digital technology including the design aspects of digital circuits is the need of the day. We present this fifth edition to fulfil the requirements of the students of various B.E./B.Tech. degree courses, including Electronics and Communication Engineering, Electrical and Electronics Engineering, Information Technology, Computer Science and Engineering, and Electronics and Instrumentation Engineering offered in all Indian universities. It will also serve as textbook for students of B.Sc. and M.Sc. degree courses in Electronics, Information Technology, Computer Science, Applied Physics and Computer Software, and MCA, AMIE, Grad. IETE, and Diploma courses, and as a reference book for competitive examinations. All the topics have been illustrated with clear diagrams. A variety of examples is given to enable students to design digital circuits efficiently.

Key Features

- Provides simple and clear explanation of digital electronic concepts in a lucid language.
- Includes numerous examples—each solved step-by-step in chapters.
- Numerous review questions and additional problems are given at the end of each chapter to help reader apply and practice the concepts learnt.

New to this Edition

- Introduces newer topics such as SDRAM, DDR RAMs, Flash memories, and GAL.
- Includes more solved problems using Boolean algebra, six-variable K-map, Quine-McClusky method, more logic function implementation using multiplexers, minimization of state diagram using merger graph, and also additional problems for analysis of sequence cells.
- Presents Verilog HDL programs in addition to VHDL programs.

Organization of the Book

This book is divided into 16 chapters. Each chapter begins with an introduction and ends with review questions and problems.

Chapter 1 provides an introduction to the number system and binary arithmetic and codes.

Chapter 2 deals with Boolean algebra, simplification using Boolean theorems, K-map method, and Quine-McCluskey method.

Chapter 3 discusses logic gates and implementation of switching functions using basic and universal gates.

Chapter 4 deals with various logic families such as TTL and CMOS logic circuits.

Chapters 5 and **6** give a brief description on combinational circuits like arithmetic and data processing.

Chapter 7 describes flip-flops and realization using flip-flops.

Chapter 8 discusses synchronous and asynchronous counters and the design of synchronous counters in detail.

Chapter 9 presents shift registers, shift counters, and ring counters and their design.

Chapter 10 elaborates upon memory devices, which include ROM, RAM, PLA, PAL, and FPGA.

Chapters 11 and **12** are devoted to the design of synchronous and asynchronous sequential circuits, respectively.

Chapter 13 explains some of the most common types of digital to analog and analog to digital converters.

Chapters 14 and **15** deal with clock generators and applications of digital circuits, respectively.

Chapter 16 describes hardware description language (HDL) for digital circuits.

An appendix provides a table of 74XX series TTL gates.

Acknowledgements

We sincerely thank the managements of SSN College of Engineering, Chennai and Mepco Schlenk Engineering College, Sivakasi, for their constant encouragement and providing the necessary facilities for completing this project. We express our deep gratitude to Prof. M. Salihu, Former Vice-Chancellor, Madurai Kamaraj University, for writing Foreword to the first edition of this book. Many of our colleagues have reviewed the additional material and we thank them for their useful comments, which have improved the book considerably over the previous edition. Our thanks are due to Mr. R. Gopalakrishnan, Mr. K. Rajan, and Mr. A. Chakkarapani for word processing the additional script.

We specially thank Oxford University Press for their initiation to bring out this revised edition in a short span of time.

Dr Salivahanan is greatly thankful to his wife, Kalavathy and sons, Santhosh Kanna and Subadesh Kanna. Dr Arivazhagan expresses his heartfelt thanks to his wife Rosilin Glory and children Sri Madhu Mitha and Selva Yokesh.

We welcome suggestions for the improvement of the book.

S. Salivahanan
S. Arivazhagan

Contents

CHAPTER 5

Arithmetic Circuits

5.1 Introduction

Digital computers and calculators consist of arithmetic and logic circuits, which contain logic gates and flip-flops that add, subtract, multiply and divide binary numbers. The basic building blocks of the arithmetic unit in a digital computer are *adders*. These circuits perform operations at speeds less than 1μs.

A digital system consists of two types of circuits, namely

(i) Combinational logic circuit,

(ii) Sequential logic circuit.

In a combinational circuit, the output at any time depends only on the input values at that time. In a sequential circuit, the output at any time depends on the present input values as well as the past output values. The basic building blocks of an arithmetic unit such as half-adder and full-adder are combinational circuits.

5.2 Procedure for the Design of Combinational Circuits

Any combinational circuit can be designed by following the design procedure given below:

1. From the word description of the problem, identify the inputs and outputs and draw a block diagram.
2. Draw a truth table such that it completely describes the operation of the circuit for different combinations of inputs.
3. Write down the switching expression(s) for the output(s).
4. Simplify the switching expression using either algebraic or K-map method.
5. Implement the simplified expression using logic gates.

5.3 Half-adder

The simplest combinational circuit which performs the arithmetic addition of two binary digits is called a *half-adder*. As shown in Fig. 5.1(a), the half-adder has two inputs and two outputs. The two inputs are the two 1-bit numbers A and B, and the two outputs are the sum (S) of A and B and the carry bit denoted by C. From the truth table of the half-adder shown in Table 5.1, one can understand that the Sum output is 1 when either of the inputs (A or B) is 1, and the Carry output is 1 when both the inputs (A and B) are 1.

Table 5.1 Truth table of half-adder

Inputs		Outputs	
Augend ***A***	**Addend** ***B***	**Sum** ***S***	**Carry** ***C***
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

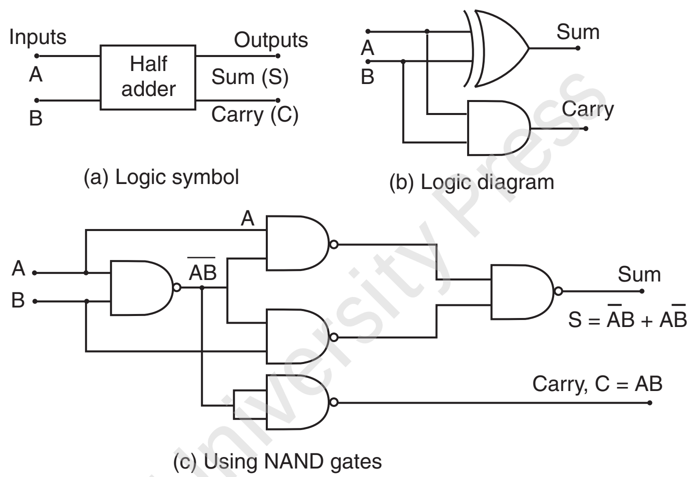


Fig. 5.1 Half-adder

From Table 5.1, the logic expression for the Sum output can be written as a Sum of Product expression by summing up the input combinations for which the sum is equal to 1.

In the truth table, the sum output is 1 when $AB = 01$ and $AB = 10$. Therefore, the expression for sum is

$$S = \overline{A}B + A\overline{B}$$

Now, this expression can be simplified as

$$S = A \oplus B$$

Similarly, the logic expression for Carry output can be expressed as a Sum of Product expression by summing up the input combinations for which the carry is equal to 1. In the truth table, the carry is 1 when $AB = 11$. Therefore,

$$C = AB$$

This expression for C cannot be simplified. The sum output corresponds to a logic Ex-OR function while the carry output corresponds to an AND function. So, the half-adder circuit can be implemented using Ex-OR and AND gates as shown in Fig. 5.1(b). Fig. 5.1(c) gives the realisation of the half-adder using minimum number of NAND gates. The implementation of the half-adder circuit using basic gates AND, OR and NOT is shown in Fig. 5.2.

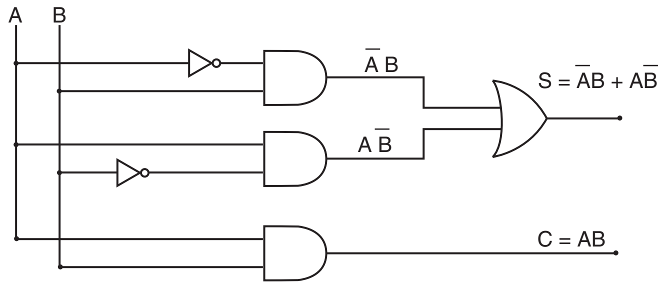
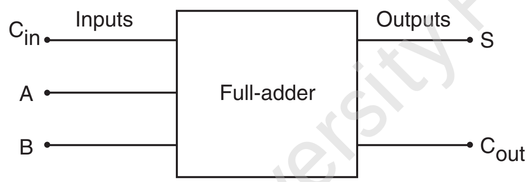


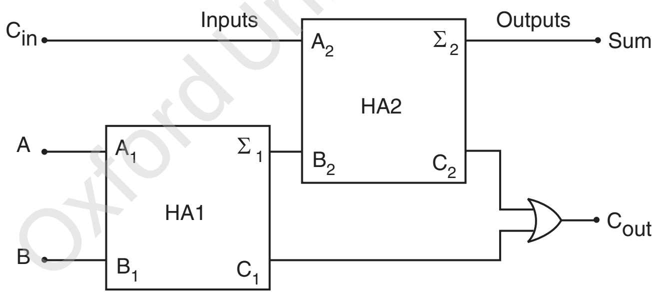
Fig. 5.2 Half-adder using basic AND, OR and NOT gates

5.4 Full-adder

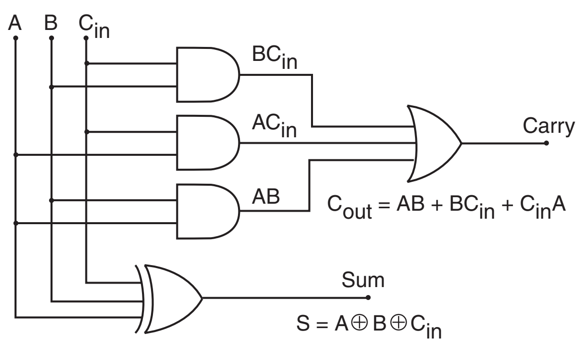
A half-adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multibit addition is performed. For this purpose, a full-adder is designed. A *full-adder* is a combinational circuit that performs the arithmetic sum of three input bits and produces a sum output and a carry.



(a) Logic symbol



(b) Symbol using 2 half-adders



(c) Logic diagram

Fig. 5.3 Full-adder

The logic symbol of the full adder is shown in Fig.5.3(a). It consists of three inputs and two outputs. The two input variables denoted by A (Augend bit) and B (Addend bit) represent the two significant bits to be added. The third input, C_{in}, represents the carry from the previous lower significant position. The outputs are designated by the symbols S (for sum) and C_{out} (for carry). The truth table for the full-adder circuit is shown in Table 5.2. The binary variable S gives the value of the LSB of the sum, and the binary variable C_{out}, gives the output carry. A full-adder can be formed using two half-adder circuits and an OR gate as shown in Fig. 5.3 (b).

Table 5.2 Truth table of full-adder

Inputs			**Outputs**	
Augend bit A	**Addend bit** B	**Carry input** C_{in}	**Sum** S	**Carry output** 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

As shown in Table 5.2, there are eight possible input combinations for the three inputs and for each case the S and C_{out} values are listed. From the truth table, the logic expression for S can be written by summing up the input combinations for which the sum output is 1 as:

$$S = \bar{A}\bar{B}C_{\text{in}} + \bar{A}B\bar{C}_{\text{in}} + A\bar{B}\bar{C}_{\text{in}} + ABC_{\text{in}}$$

Simplifying the above expression, we get

$$\begin{aligned} S &= \bar{A}(\bar{B}C_{\text{in}} + B\bar{C}_{\text{in}}) + A(\bar{B}\bar{C}_{\text{in}} + BC_{\text{in}}) \\ &= \bar{A}(B \oplus C_{\text{in}}) + A(\overline{B \oplus C_{\text{in}}}) \end{aligned}$$

Let $B \oplus C_{\text{in}} = X$

Now, $S = \bar{A}X + A\bar{X} = A \oplus X$

Replacing X by $B \oplus C_{\text{in}}$ in the above expression, we have

$$S = A \oplus B \oplus C_{\text{in}}$$

Similarly, the logic expression for C_{out} can be written by summing up the input combinations for which C_{out} is 1, as given below:

$$\begin{aligned} C_{\text{out}} &= \bar{A}BC_{\text{in}} + A\bar{B}C_{\text{in}} + AB\bar{C}_{\text{in}} + ABC_{\text{in}} \\ &= BC_{\text{in}}(A + \bar{A}) + A\bar{B}C_{\text{in}} + AB\bar{C}_{\text{in}} \\ &= BC_{\text{in}} + A\bar{B}C_{\text{in}} + AB\bar{C}_{\text{in}} \end{aligned}$$

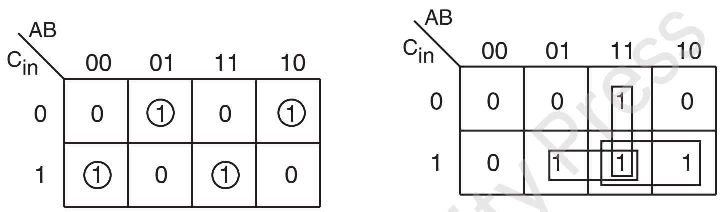
Now, the ABC_{in} term is added twice for simplification.

$$
\begin{aligned}
C_{out} &= BC_{in} + A\overline{B}C_{in} + AB\overline{C}_{in} + ABC_{in} + ABC_{in} \\
&= BC_{in} + AC_{in}(B + \overline{B}) + AB(C_{in} + \overline{C}_{in}) \\
&= BC_{in} + AC_{in} + AB
\end{aligned}
$$

From the simplified expressions of S and C_{out}, the full-adder circuit can be implemented using one 3-input Ex-OR gate, three 2-input AND gates and one 3-input OR gate as shown in Fig. 5.3(c).

5.5 K-map Simplification

K-map method can also be used for simplifying the logic expressions for S and C_{out}. The *K*-maps for the outputs S and C_{out} are given in Fig. 5.4.



(a) K-map for sum

(b) K-map for C_{out}

Fig. 5.4 Karnaugh maps

From the *K*-maps, the simplified expressions for S and C_{out} can be written as follows:

$$
\begin{aligned}
S &= \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\overline{C}_{in} + ABC_{in} \\
C_{out} &= AB + BC_{in} + C_{in}A
\end{aligned}
$$

Using the above expressions, the full-adder can be implemented using basic gates as shown in Fig. 5.5.

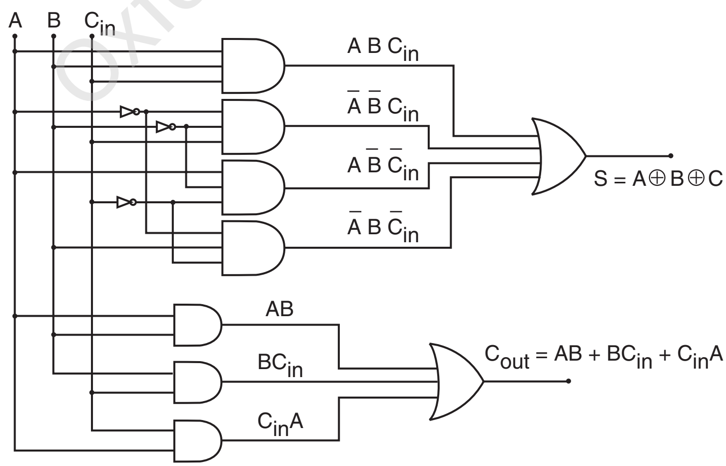


Fig. 5.5 Full-adder using basic gates

5.6 Half-subtractor

The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and B_{out} (borrow out). The logic symbol for a half-subtractor is shown in Fig. 5.6(a). The truth table for half-subtractor is shown in Table 5.3. From the truth table, it is clear that the difference output is 0 if $X = Y$ and 1 if $X \neq Y$; the borrow output B_{out} is 1 whenever $X < Y$. If X is less than Y, then subtraction is done by borrowing 1 from the next higher order bit.

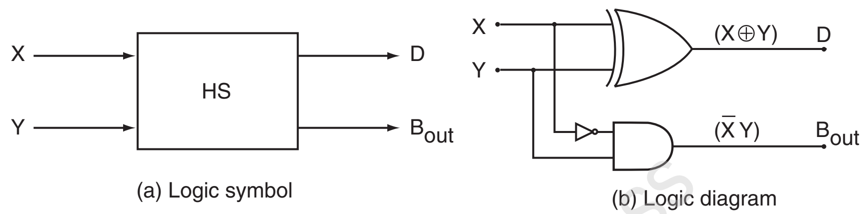


Fig. 5.6 Half-subtractor

Table 5.3 Truth table of half-subtractor

Inputs		**Outputs**	
Minuend X	**Subtrahend** Y	**Difference** D	**Borrow** B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From Table 5.3, as discussed earlier, the Boolean expressions for difference (D) and Borrow out (B_{out}) can be written as follows:

$$D = \overline{X}Y + X\overline{Y} = X \oplus Y$$

$$B_{out} = \overline{X}Y$$

From the above equations, the half-subtractor can be implemented using an Ex-OR gate, a NOT gate and an AND gate as shown in Fig. 5.6(b).

5.7 Full-subtractor

A full-subtractor is a combinational circuit that performs subtraction involving three bits, namely minuend bit, subtrahend bit and the borrow from the previous stage. The logic symbol for full-subtractor is shown in Fig. 5.7(a).

It has three inputs, X (minuend), Y (subtrahend) and B_{in} (borrow from previous stage), and two outputs D (difference) and B_{out} (borrow out). The truth table for the full-subtractor is given in Table 5.4. The full-subtractor can be implemented using two half-subtractors and an OR gate as shown in Fig. 5.7(b).

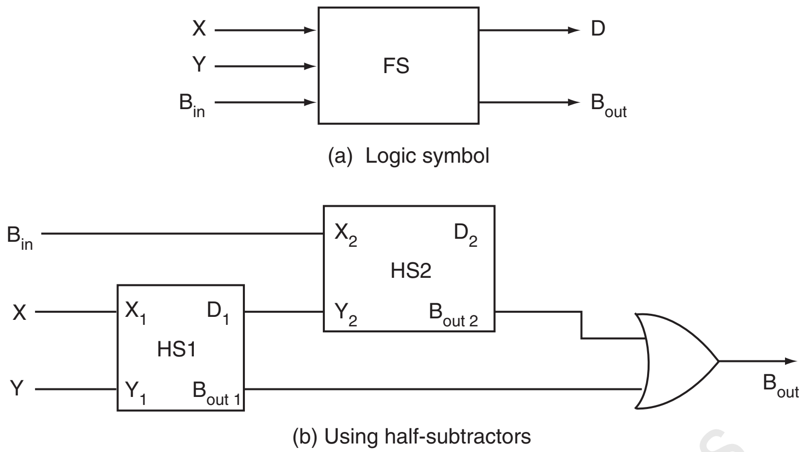


Fig. 5.7 Full-subtractor

Table 5.4 Truth table of full-subtractor

Inputs			Outputs	
Minuend bit ***X***	**Subtrahend bit** ***Y***	**Borrow in** $\boldsymbol{B_{in}}$	**Difference** ***D***	**Borrow out** **out**
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

From Table 5.4, the Sum of Product expression for the difference (D) output can be written as:

$$D = \overline{X}\,\overline{Y}B_{in} + \overline{X}Y\overline{B_{in}} + X\overline{Y}\overline{B_{in}} + XYB_{in}$$

Simplifying the above expression,

$$D = (\overline{X}\,\overline{Y} + XY)B_{in} + (\overline{X}Y + X\overline{Y})\overline{B_{in}}$$

$$= (\overline{X \oplus Y})B_{in} + (X \oplus Y)\overline{B_{in}}$$

$$\mathrm{D} = X \oplus Y \oplus B_{in}$$

Similarly, the sum of product expression for B_{out} can be written from the truth table as:

$$B_{out} = \overline{X}\,\overline{Y}B_{in} + \overline{X}Y\overline{B_{in}} + \overline{X}YB_{in} + XYB_{in}$$

The equation for B_{out} can be simplified using Karnaugh map as shown in Fig. 5.8.

B_{in} \ XY	00	01	11	10
0	0	1	0	0
1	1	1	1	0

Fig. 5.8 K-map for B_{out}

Now, $B_{out} = \overline{X}Y + \overline{X}B_{in} + YB_{in}$

Using the above simplified expressions, the full-subtractor can be realised as shown in Fig. 5.9.

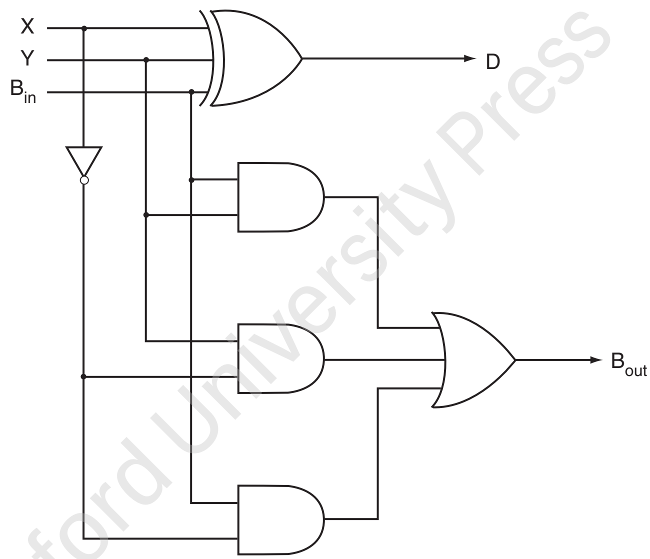


Fig. 5.9 Realisation of full-subtractor

One can notice that the equation for D is the same as the sum output for a full-adder, and the borrow output B_{out} resembles the carry output for full-adder except that one of the inputs is complemented. From these similarities, it is possible to convert a full-adder into a full-subtractor by merely complementing that input prior to its application to the input of gates which form the borrow output.

5.8 Parallel Binary Adder

In most logic circuits, addition of more than 1-bit is carried out. For example, modern computers and calculators use numbers ranging from 8 to 64-bits. The addition of multibit numbers can be accomplished using several full-adders. The 4-bit adder using full-adder circuits is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output as shown in Fig. 5.10. Since all the bits of the augend and addend are fed into the adder circuits simultaneously and the additions in each position are taking place at the same time, this circuit is known as *parallel adder*.

The addition operation is illustrated in the following example: Let the 4-bit words to be added be represented by $A_3A_2\mathrm{A}_1A_0 = 111$ and $B_3B_2B_1B_0 = 0011$.

Significant place		4 3 2 1
Input carry		1 1 1 0
Augend word A:		1 1 1 1
Addend word B:		0 0 1 1
	1	0 0 1 0 ← Sum
	↑ Output carry	

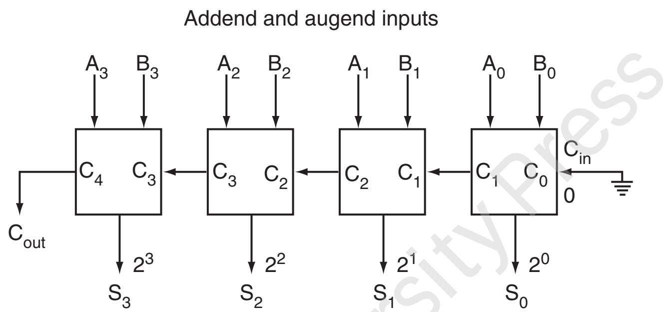


Fig. 5.10 4-bit binary parallel adder

In a 4-bit parallel binary adder circuit, the input to each full-adder will be A_i, B_i and C_i, and the outputs will be S_i and $C_{i+1,}$ where 'i' varies from 0 to 3. Also, the carry output of the lower order stage is connected to the carry input of the next higher order stage. Hence, this type of adder is called *ripple-carry adder*.

In the least significant stage, A_0, B_0 and C_0 (which is 0) are added resulting in Sum S_0 and Carry C_1. This carry C_1 becomes the carry input to the second stage. Similarly, in the second stage, A_1, B_1 and C_1 are added resulting in S_1 and C_2; in the third stage, A_2, B_2 and C_2 are added resulting in S_2 and C_3; in the fourth stage, A_3, B_3 and C_3 are added resulting in S_3 and C_4 which is the output carry. Thus, the circuit results in a sum ($S_3S_2S_1S_0$) and a carry output (C_{out}).

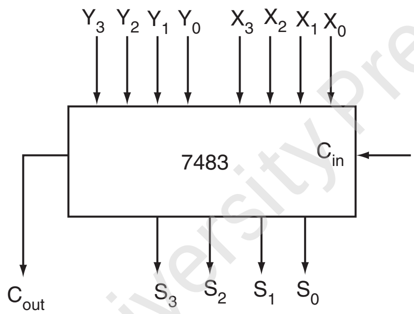
Though the parallel binary adder is said to generate its output immediately after the inputs are applied, its speed of operation is limited by the carry propagation delay through all stages. In each full-adder, the carry input has to be generated from the previous full-adder which has an inherent propagation delay. The propagation delay (t_p) of a full-adder is the time difference between the instant at which the inputs (A_i, B_i and C_i) are applied and the instant at which its outputs (S_i and C_{i+1}) are generated. Therefore, in a 4-bit binary adder, the output in LSB stage is generated only after t_p seconds. Similarly, the output in the second stage will be generated only after t_p seconds from the time the outputs of the first stage are generated, i.e., after $2t_p$ seconds from the time the inputs are applied; the third stage will generate outputs only after $3t_p$ seconds and the fourth stage will generate outputs only after $4t_p$ seconds. Thus, in a 4-bit binary adder, where each full adder has a propagation delay of 50ns, the output in the fourth stage will be generated only after $4t_p = 4 \times 50$ ns $= 200$ ns. The magnitude of such delay is prohibitive for high-speed computers. However, there are several methods to reduce this delay.

One of the methods of speeding up this process is look-ahead carry addition which eliminates the ripple-carry delay. This method is based on the carry generate and the carry propagate functions of the full-adder.

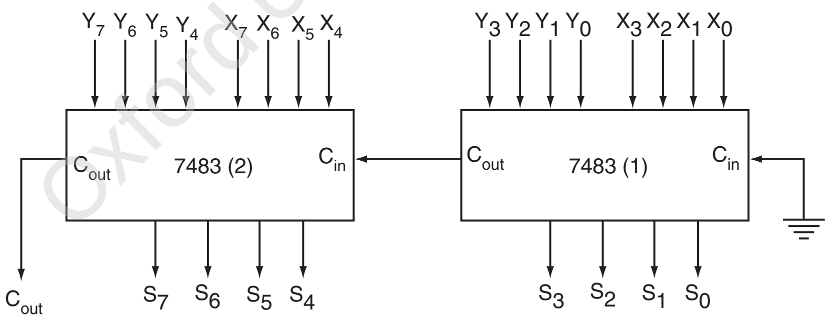
This scheme utilises logic gates to look at the lower order bits of the augend and addend if a higher-order carry is to be generated. This requires extra circuitry for getting high speed adders. This is not a significant consideration with the present day availability of integrated circuits.

5.8.1 IC 7483—4-bit Parallel Binary Adder

The IC 7483 is a commonly available TTL 4-bit parallel binary adder chip. It contains four interconnected full-adders and a look-ahead carry circuitry for its operation. The logic symbol of IC 7483 is shown in Fig. 5.11(a). It has two 4-bit inputs, $X_3 X_2 X_1 X_0$ and $Y_3 Y_2 Y_1 Y_0$, and, a carry input C_{in} in the LSB stage. The outputs are a 4-bit sum $S_3 S_2 S_1 S_0$ and a carry output C_{out} from the most significant bit stage.



(a) Logic symbol of IC 7483—4-bit parallel binary adder



(b) Cascading of two 7483 ICs

Fig. 5.11

Two or more parallel adder blocks can be connected in cascade to perform the addition operation on larger binary numbers. Fig. 5.11(b) shows the cascading connection of two 7483 adders. The four least significant bits of the numbers are added in the first adder. The carry output of this adder is given as the carry input to the second adder, which adds four most significant bits of the numbers. The output carry of the second adder is the final carry output.

5.8.2 4-bit Parallel Binary Subtractor

Just as a parallel binary adder can be implemented by cascading several full-adders, a parallel binary subtractor can also be implemented by cascading several full-subtractors. A 4-bit parallel binary subtractor that subtracts a 4-bit number $Y_3 Y_2 Y_1 Y_0$ from another 4-bit number $X_3 X_2 X_1 X_0$ is shown in Fig. 5.12. It has 4 difference outputs ($D_3 D_2 D_1 D_0$) and a borrow output (B_{out}). Note that the B_{in} of the LSB full-subtractor is connected to 0 and B_{out} of ith full-subtractor is connected to B_{in} of $(i + 1)^{th}$ full-subtractor.

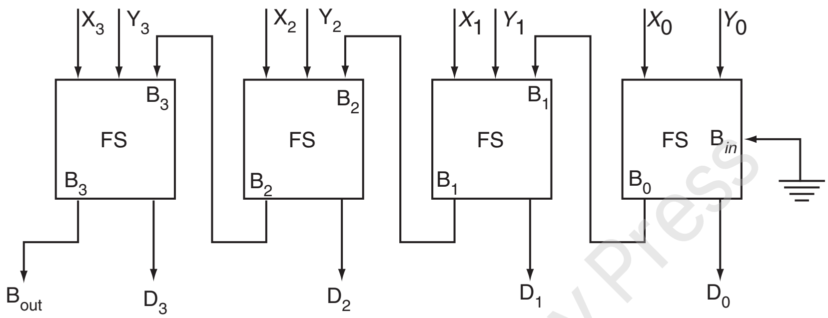


Fig. 5.12 4-bit parallel binary subtractor

5.9 Controlled Inverter

The subtraction of two binary numbers may be accomplished by taking the 2's complement of the subtrahend and adding to the minuend. By this procedure, the subtraction becomes an addition operation. The 2's complement of the subtrahend can be obtained by adding a 1 to the 1's complement of the subtrahend. From the Ex-OR gate truth table, we know that when one of the inputs is LOW the output is the true value of the other input and when one of the inputs is HIGH the output is the complement of the other input. Therefore, the complement of a binary digit can be obtained using an Ex-OR gate as shown in Fig. 5.13.

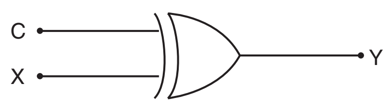


Fig. 5.13 Ex-OR gate functioning as an inverter

In Fig. 5.13, X is the input, C is the control input and Y is the output. From the figure, it is clear that if $C = 0$, then $Y = X$, i.e., input X is available at Y in uncomplemented form; if $C = 1$, then $Y = \overline{X}$, i.e., input X is available at Y in complemented form. Thus, if $C = 1$, the Ex-OR gate can function as an inverter. Similarly, a group of Ex-OR gates can be used to invert a group of bits. Fig. 5.14 shows the complementing process of a 4-bit binary number ($Y_3 Y_2 Y_1 Y_0$) using a controlled inverter.

When the control input is low, the output will be the input, i.e. $Y_3 Y_2 Y_1 Y_0$. When the control input is high, the output will be the complement of the input $\overline{Y}_3 \overline{Y}_2 \overline{Y}_1 \overline{Y}_0$.

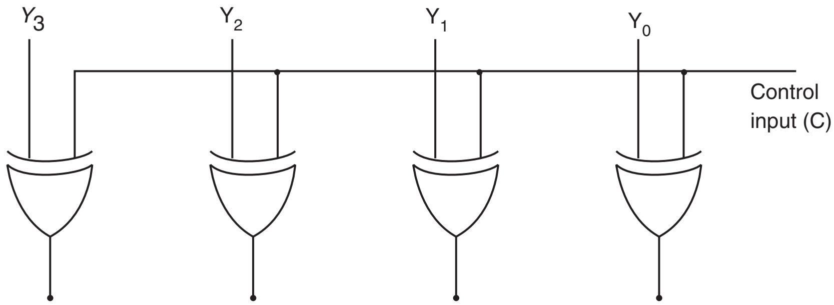


Fig. 5.14 Controlled inverter

5.10 4-bit Parallel Adder/Subtractor

The 4-bit parallel binary adder/subtractor circuit shown in Fig. 5.15 performs the operations of both addition and subtraction. It has two 4-bit inputs $X_3X_2X_1X_0$ and $Y_3Y_2Y_1Y_0$. The $\overline{\text{ADD}}$/SUB control line, connected with C_{in} of the least significant bit of the full-adder, is used to perform the operations of addition and subtraction. The Ex-OR gates are used as controlled inverters.

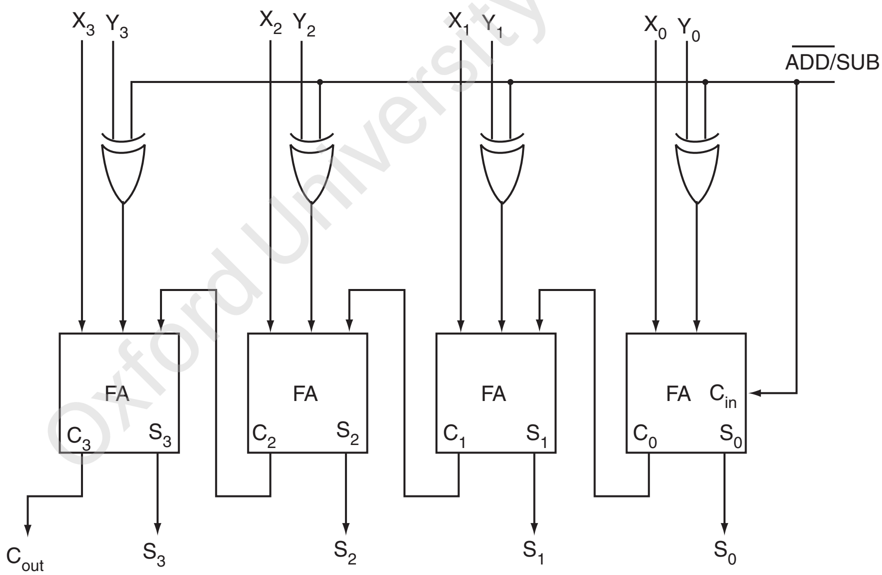


Fig. 5.15 4-bit parallel adder/subtractor

To perform subtraction, the $\overline{\text{ADD}}$/SUB control input is kept high. Now, the controlled inverter produces the 1's complement of the addend ($\overline{Y}_3\overline{Y}_2\overline{Y}_1\overline{Y}_0$). Since 1 is given to C_{in} of the least significant bit of the adder, it is added to the complemented addend producing 2's complement of the addend before addition. Now, the data $X_3X_2X_1X_0$ will be added to the 2's complement of $Y_3Y_2Y_1Y_0$ to produce the Sum, i.e., the difference between the addend and the augend, and C_{out}, i.e., the borrow output of 4-bit subtractor. Also, it has $S_3S_2S_1S_0$ as sum output and C_{out} as carry output. When

$\overline{\text{ADD}}/\text{SUB}$ input is LOW, the controlled inverter allows the addend ($Y_3\,Y_2\,Y_1\,Y_0$) without any change to the input of the full-adder, and the carry input C_{in} of least significant bit of full-adder, becomes zero. Now, the augend ($X_3\,X_2\,X_1\,X_0$) and addend ($Y_3\,Y_2\,Y_1\,Y_0$) are added with $C_{in} = 0$. Hence, the circuit functions as a 4-bit adder resulting in sum $S_3\,S_2\,S_1\,S_0$ and carry C_{out}.

Example 5.1 Construct a 4-bit parallel binary adder/subtractor using IC 7483.

Solution IC7483 is a 4-bit parallel binary adder. A 4-bit parallel binary adder/subtractor can be implemented using IC7483, controlled inverter IC7486 and a control line $\overline{\text{ADD}}/\text{SUB}$ as shown in Fig. E5.1.

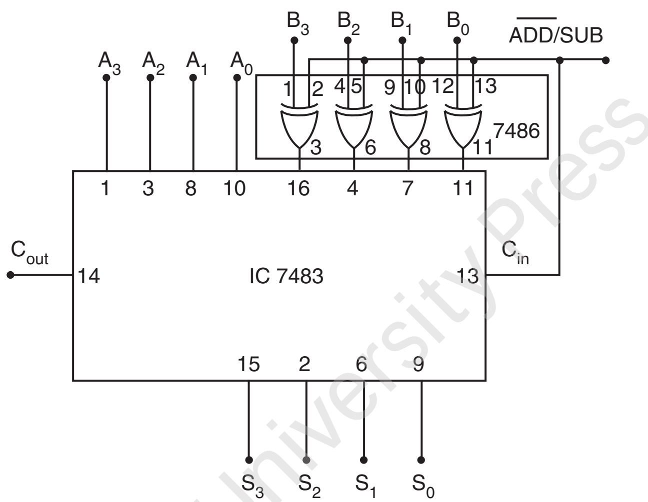


Fig. E5.1 4-bit parallel binary adder/subtractor using IC 7483

5.11 Fast Adder

In the parallel binary adder discussed in section 5.7, the carry generated by the i^{th} adder is fed as carry input to the $(i + 1)^{th}$ adder. In this adder, the output ($C_{out}\,S_3\,S_2\,S_1\,S_0$) is available only after the carry is propagated through each of the adders i.e., from LSB to MSB adder through intermediate adders. Hence, the addition process can be considered to be complete only after the above carry propagation delay through adders, which is proportional to number of stages in it. One of the methods of making this process faster is look ahead carry addition, which eliminates the ripple carry delay. This method is based on the carry generating and the carry propagating functions of the full adder.

5.11.1 4-bit Carry Look Ahead Adder

The carry look ahead adder is based on the principle of looking at the lower order bits of the augend and addend if a high order carry is generated. This adder reduces the carry delay by reducing the number of gates through which a carry signal must propagate. To explain its operation, consider the truth table of full adder, shown in Table 5.5.

Table 5.5 Truth table of Full adder, emphasizing the conditions at which carry generation occurs

Row	A	B	C_{in}	S	C_{out}
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Rows 0–1 $\rightarrow$ No carry generation i.e. $C_{out} = 0$

Rows 2–5 $\rightarrow$ Carry propagation i.e. $C_{out} = C_{in}$

Rows 6–7 $\rightarrow$ Carry generation i.e. $C_{out} = 1$

In rows 0 and 1, the carry output (C_{out}) is always 'zero' and independent of carry input (C_{in}), while in rows 6 and 7, the C_{out} is always 'one' and independent of C_{in}. These are known as carry generate combinations. In rows 2, 3, 4 and 5 the carry output is equal to the carry input i.e. $C_{out} = 1$ only when $C_{in} = 1$. These are carry propagate combinations. Let G_i represent the unity carry (i.e. $C_{out} = 1$) generate condition and P_i represent the carry propagate condition of the i^{th} stage of a parallel adder.

From the Truth Table 5.5 G_i is obtained by summing up the combinations corresponding to 6^{th} and 7^{th} rows as follows:

$$\begin{aligned} G_i &= A_iB_iC_{in} + A\ B_i\overline{C_{in}} \\ &= A_iB_i(C_{in} + \overline{C_{in}}) \\ G_i &= A_iB_i \end{aligned}$$

Similarly the carry propagation condition (P_i) occurs when either $A_i = 1$ and $B_i = 0$ or vice versa as shown in Truth Table 5.5. Now P_i is given by

$$P_i = A_i\overline{B}_i + \overline{A}_iB_i = A_i \oplus B_i$$

Consider the addition of two 4-bit binary numbers A ($A_3A_2A_1A_0$) and B ($B_3B_2B_1B_0$). The unity carry output of the i^{th} stage can be expressed in terms of G_i, P_i and C_{i-1} which is the unity carry output of the $(i-1)^{th}$ stage as follows:

$$C_i(C_{out}) = G_i + P_iC_{i-1}$$

where C_{i-1} for LSB stage is C_{in} which is assumed to be zero. In a 4-bit binary adder, four stages of addition are required to add A_0B_0, A_1B_1, A_2B_2 and A_3B_3. Therefore, for i = 0, 1, 2 and 3, the C_i's are given by

$$C_0 = G_0 + P_1C_{in} \text{ .... where } G_0 = A_0B_0;\ P_0 = A_0 \oplus B_0 \text{ and } C_{in} = 0$$

$$\begin{aligned} C_1 &= G_1 + P_1C_0 \\ &= G_1 + P_1(G_0 + P_0C_{in}) \\ &= G_1 + P_1G_0 + P_1P_0C_{in} \text{ .... where } G_1 = A_1B_1 \text{ and } P_1 = A_1 \oplus B_1 \end{aligned}$$

$$\begin{aligned} C_2 &= G_2 + P_2C_1 \\ &= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_{in}) \\ &= G_2 + P_2G_1 + P_2P_1G_0 + P_1P_0C_{in} \text{ ...where } G_2 = A_2B_2 \text{ and } P_2 = A_2 \oplus B_2 \end{aligned}$$

$$C_3 = G_3 + P_3C_2$$

$$= G_3 + P_3(G_2 + P_2G_1 + P_2P_1P_0C_{in})$$

$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{in} \text{ ...... where } G_3 = A_3\,B_3 \text{ and}$$

$$P_3 = A_3 \oplus B_3$$

The sum of A and B is given by

$$S = C_3S_3S_2S_1S_0$$

where $S_i = A_i \oplus B_i \oplus C_{i-1}$ for $i = 0, 1, 2, 3$

i.e.,

$$S_0 = A_0 \oplus B_0 \oplus C_{in}$$

$$S_1 = A_1 \oplus B_1 \oplus C_0$$

$$S_2 = A_2 \oplus B_2 \oplus C_1$$

$$S_3 = A_3 \oplus B_3 \oplus C_2$$

Using the above equation, a 4-bit carry look ahead adder can be realized as shown in Fig. 5.16. From the diagram, one can easily understand that the addition of two 4 bit numbers can be done by a carry look ahead adder in a four gate propagation time. Also, it is important to note that the addition of n-bit binary numbers takes the same four gate propagation delay.

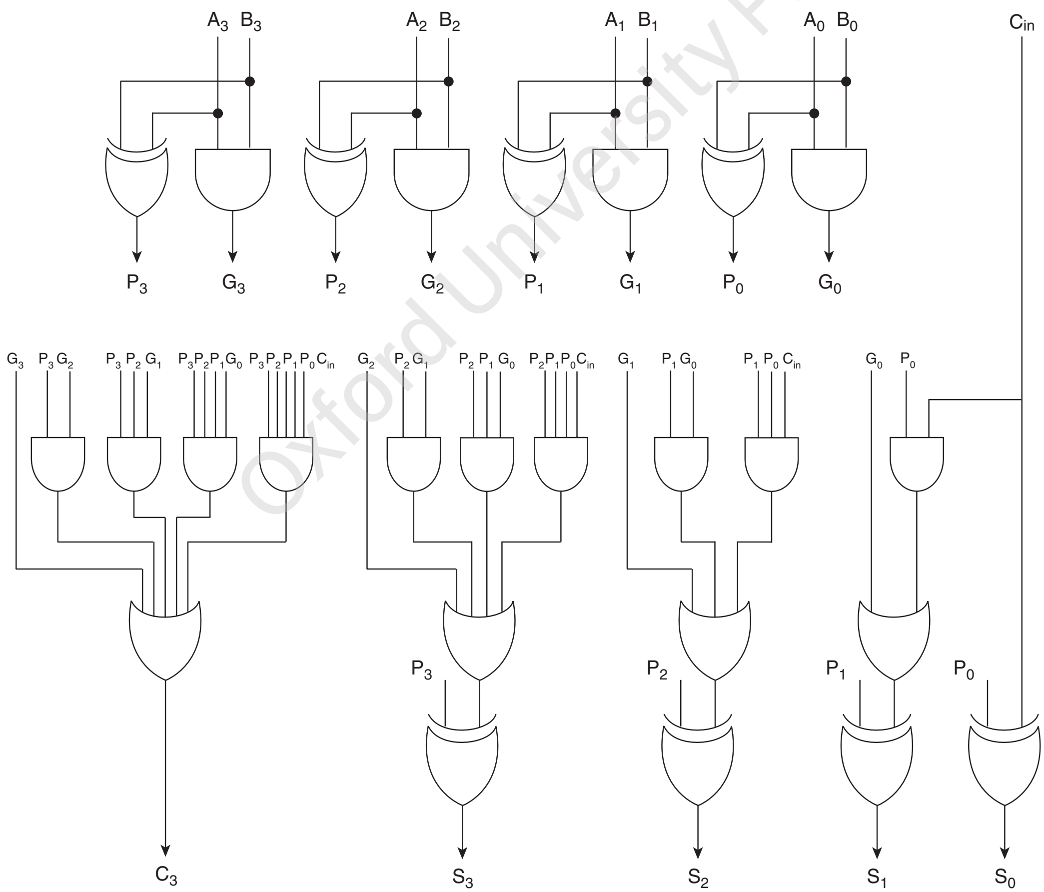


Fig. 5.16 Logic diagram of 4-bit carry look ahead adder

5.12 Serial Adder

Though the parallel adder performs the addition of two binary numbers at a relatively faster rate, the disadvantage of the parallel addition is that it requires a large amount of logic circuitry. This increases in direct proportion with the number of bits in the numbers being added. On the other hand, in *serial addition*, the addition operation is carried out bit-by-bit. Therefore, the serial adder requires simpler circuitry than a parallel adder but results in a low speed of operation.

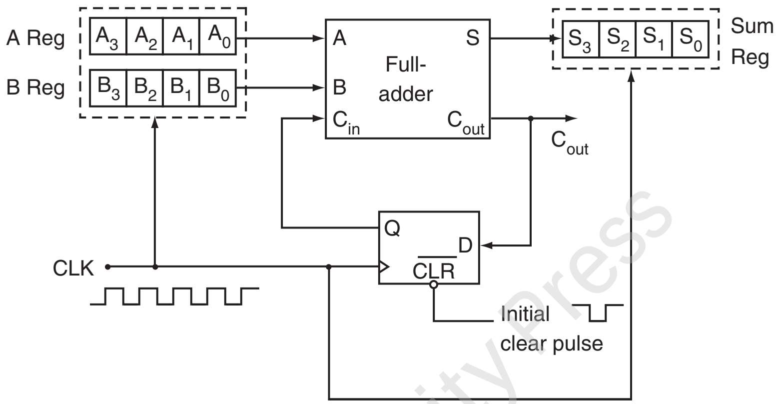


Fig. 5.17 4-bit serial adder

The diagram of a 4-bit serial adder is shown in Fig. 5.17. The two shift registers A and B are used to store the numbers to be added serially. A single full-adder is used to add one pair of bits at a time along with the carry. The D flip-flop, i.e. carry flip-flop, is used to store the carry output of the full-adder so that it can be added to the next significant position of the numbers in the registers. The contents of the shift registers shift from left to right and their outputs starting from A_0 and B_0 are fed into a single full-adder along with the output of the carry flip-flop upon application of each clock pulse.

The sum output of the full-adder is fed to the Most Significant Bit (S_3) of the sum register. For each succeeding clock pulse, the contents of both the shift registers are shifted once to the right, and new sum bit and new carry bit are transferred to sum register and carry flip-flop respectively. This process continues until all the pairs of bits are added.

The following example explains the operation of a serial adder. Let the augend ($A_3A_2A_1A_0$) be 0111 and the addend ($B_3B_2B_1B_0$) be 0010, stored in shift registers A and B respectively. Also, the carry flip-flop has been initially cleared to the 0 state, so that $C_{in} = 0$.

First clock pulse Before the first clock pulse occurs, as the inputs to the Full-adder are $A_0 = 1$, $B_0 = 0$ and $C_{in} = 0$. The full-adder outputs will be $S = 1$ and $C_{out} = 0$. When the first clock pulse occurs, the value in the A and B registers shift from left to right by one bit. In addition, the sum (S) is transferred to S_3 of sum register, and the C_{out} is transferred to the carry flip-flop, whose output becomes 0, which is the carry input of the full-adder.

Second clock pulse Now, $A_0 = 1$, B_0 and $C_{in} = 0$, at the full-adder input. Therefore, $S = 0$ and $C_{out} = 1$ When the second clock pulse occurs, A, B and sum registers again shift right; $S = 0$ is transferred to S_3, and $C_{out} = 1$ is transferred to the carry flip-flop.

Third clock pulse The values of A_0 and B_0 are now 1 and 0 respectively, and C_{in} is 1. These values produce $S = 0$ and $C_{out} = 1$ at the full-adder outputs. On the occurrence of the third clock pulse, the A, B and sum registers shift right, sum = 0 goes to S_3, and $C_{out} = 1$ goes to the carry flip-flop.

Fourth clock pulse Both A_0 and B_0 are now zero, and $C_{in} = 1$. Therefore, the full-adder produces $S = 1$ and $C_{in} = 0$. The fourth clock pulse transfers $S = 1$ to S_3 and initiates all the other transfers. At the end of this fourth clock pulse, the sum value (1001) will be available in the sum register, and C_{out} is 0.

5.13 Serial Subtraction Using 2's Complement

A serial subtractor can be obtained by converting the serial adder of Fig. 5.17 using the 2's complement system. For subtraction, the subtrahend is stored in the B register and must be 2's-complemented before it is added to the minuend stored in the A register. One simple way to accomplish this is to complement the B register and to make the initial $C_{in} = 1$ instead of 0 prior to the first clock pulse. This can be easily accomplished by feeding the inverted output $\overline{B_0}$ into the full-adder instead of B_0 and initially setting the carry flip-flop to 1 instead of clearing it. The remaining circuitry is the same as serial adder.

The circuit for 4-bit serial subtractor using full-adder is shown in Fig. 5.18.

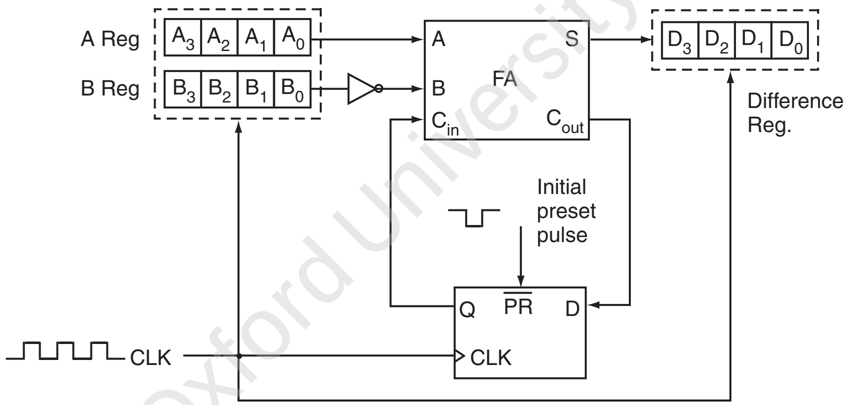


Fig. 5.18 4-bit serial subtractor using full-adder

5.14 4-bit Serial Adder/Subtractor

A 4-bit serial adder/subtractor can be constructed in a manner similar to its parallel counterpart as shown in Fig. 5.19.

In the circuit shown in Fig. 5.19, when $\overline{\text{ADD}}$/SUB = 0, the uncomplemented $B_3B_2B_1B_0$ will be applied to the full-adder. The carry flip-flop is initially cleared by applying a low pulse at $\overline{\text{CLR}}$ input and the circuit thus functions as a 4-bit serial adder. When $\overline{\text{ADD}}$/SUB = 1, the complemented output of B register ($\overline{B}_3\overline{B}_2\overline{B}_1\overline{B}_0$) will be applied to the full-adder. The carry flip-flop is initially set to 1 so as to get the 2's complement of the subtrahend by applying a low pulse at $\overline{PR}$ input and thus the circuit functions as a 4-bit serial subtractor. The comparison of serial adder with parallel adder is shown in Table 5.5.

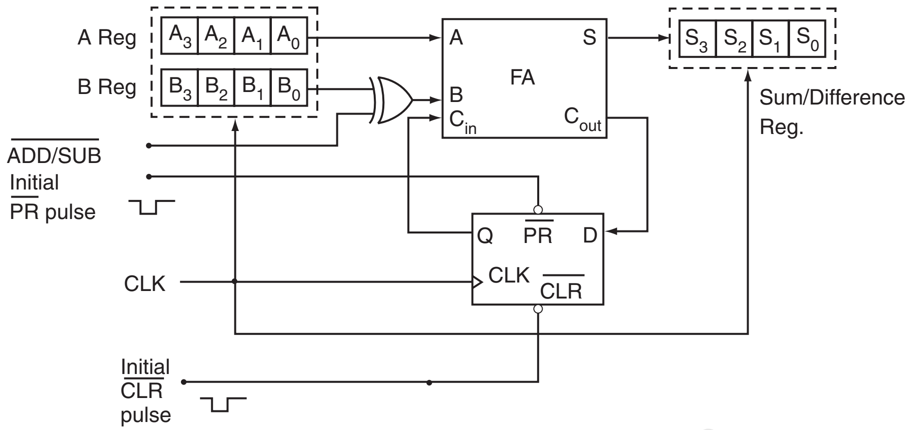


Fig. 5.19 4-bit serial adder/subtractor

Table 5.5 Serial adder Vs parallel adder

Serial adder	Parallel adder
Serial adder is less fast	Parallel adder is generally faster
It requires fewer components	It requires more components compared to serial adder
Addition is performed bit by bit starting from the LSB	All the bits are added simultaneously

5.15 BCD Adder

A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit which is also in BCD. A BCD adder must include the correction logic in its internal construction. A block diagram for the BCD adder is shown in Fig. 5.20. This adder has two 4-bit BCD inputs $X_3\ X_2\ X_1\ X_0$ $Y_3\,Y_2\ Y_1\,Y_0$ and a carry input (C_{in}). It also has a 4-bit sum output ($\Sigma_3\,\Sigma_2\,\Sigma_1\,\Sigma_0$) and a carry output ($C_{out}$). Here, the sum output is also in BCD form.

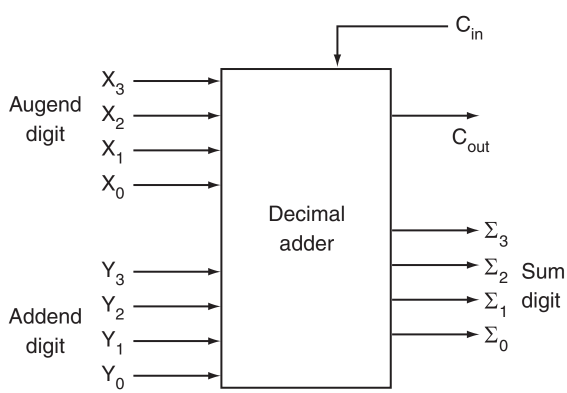


Fig. 5.20 Block diagram of a BCD adder

A BCD adder circuit must be able to do the following:

1. Add two 4-bit BCD numbers using straight binary addition.
2. If the four-bit sum is equal to or less than 9, the sum is in proper BCD form and no correction is needed.
3. If the four-bit sum is greater than 9 or if a carry is generated from the sum, the sum is not in the BCD form. Then, the digit 6 (0110) should be added to the sum to produce the BCD results. The carry may be produced due to this addition and it is added to the next decimal position.

Table 5.6 shows the results of BCD addition with corrections indicated.

Table 5.6 Results of BCD addition with corrections indicated

Decimal digit	Uncorrected BCD Sum					Corrected BCD Sum					
	C_3	S_3	S_2	S_1	S_0	C_{out}	S_3	S_2	S_1	S_0	
0		0	0	0	0		0	0	0	0	
1		0	0	0	1		0	0	0	1	
2		0	0	1	0		0	0	1	0	
3		0	0	1	1		0	0	1	1	No
4		0	1	0	0		0	1	0	0	correction
5		0	1	0	1		0	1	0	1	required
6		0	1	1	0		0	1	1	0	
7		0	1	1	1		0	1	1	1	
8		1	0	0	0		1	0	0	0	
9		1	0	0	1		1	0	0	1	
10		1	0	1	0	1	0	0	0	0	
11		1	0	1	1	1	0	0	0	1	
12		1	1	0	0	1	0	0	1	0	
13		1	1	0	1	1	0	0	1	1	
14		1	1	1	0	1	0	1	0	0	Correction
15		1	1	1	1	1	0	1	0	1	required
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	

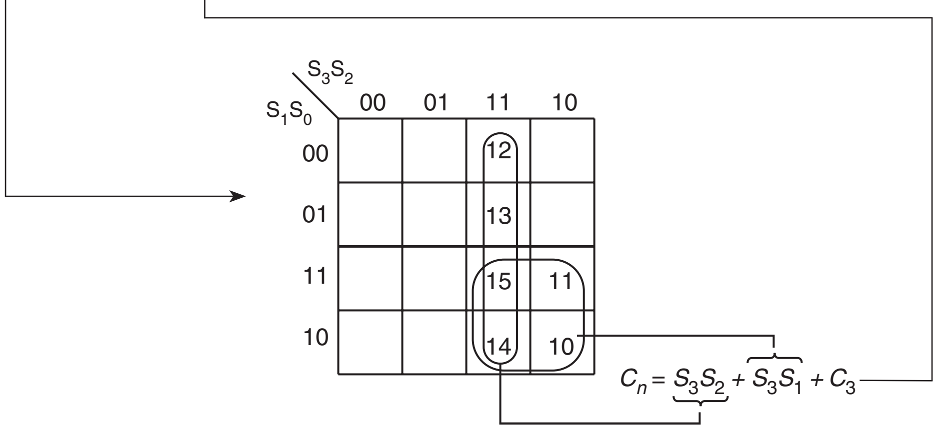


Fig. 5.21 K-map simplification

From the Table 5.6, it is clear that correction is required. When the sum output ($S_3 S_2 S_1 S_0$) is greater than *9*, i.e., when $C_3 = 1$ (OR) $S_3 = 1$ (AND) [$S_2 = 1$ (OR) $S_1 = 1$], add 0110 to get the BCD result. Note that when $C_3 = 1$, the result is 16 and above; when $S_3 = 1$ (AND) $S_2 = 1$, the result is 12 and above; when $S_3 = 1$ (AND) $S_1 = 1$, the result is 10 and above; when $S_3 = 1$ (AND) [$S_2 = 1$ (OR) $S_1 = 1$], the result is 14 and above. Therefore, the condition for correction can be written as an expression as follows:

$$C_n = C_3 + S_3(S_2 + S_1)$$

Alternatively, the above condition for correction can also be obtained by *K*-map method as shown in Fig. 5.21.

As discussed above, a BCD adder must be capable of adding two 4-bit BCD numbers, and the result has to be corrected to BCD, if the above condition is satisfied, by adding 0110. The circuit diagram for BCD adder using full-adders is shown in Fig. 5.22.

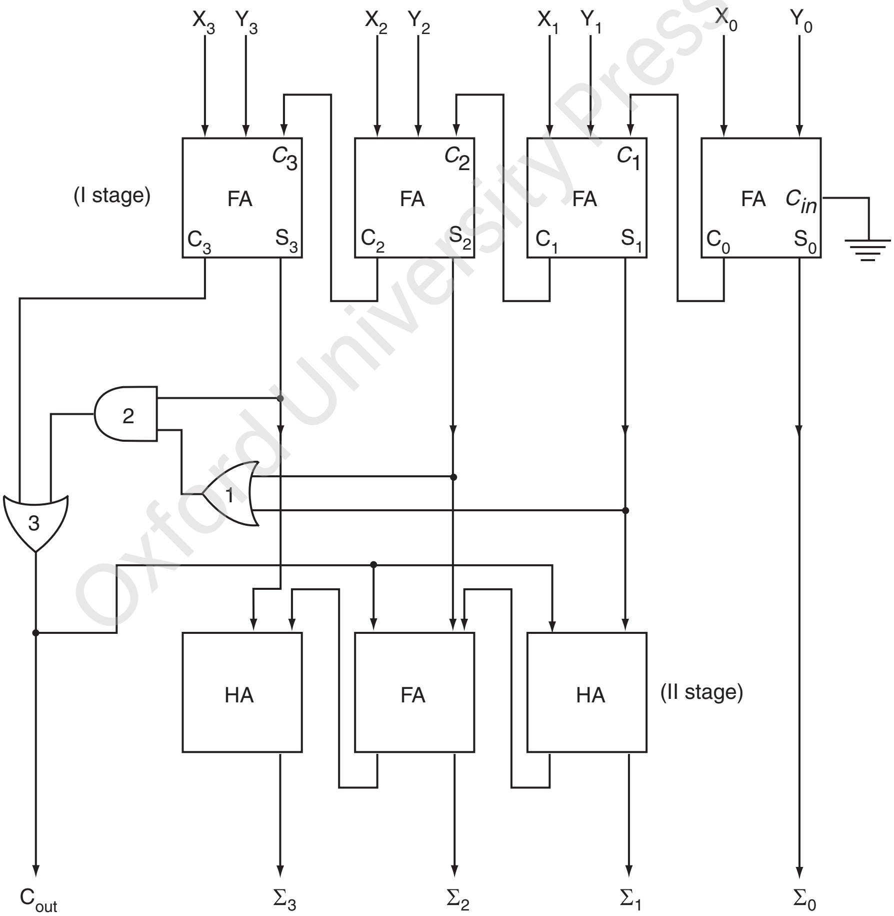


Fig. 5.22 BCD adder using full-adders

The operation of BCD adder shown in Fig. 5.22 is explained as follows: the first stage of full-adders adds two 4-bit BCD numbers, and its sum ($S_3 S_2 S_1 S_0$) and carry (C_3) are checked to ascertain whether the result exceeds 9 by AND-OR gate combinations. If the output of OR gate (3) is equal to 1, then correction is required and this is accomplished by adding 0110 in the second stage of adders as shown in Fig. 5.22. Now, the BCD result is $\Sigma_3\,\Sigma_2\,\Sigma_1\,\Sigma_0$ with carry output (C_{out}).

The BCD adder can also be implemented using two 7483 ICs as shown in Fig. 5.23. Here, $X_3X_2X_1X_0$ and $Y_3Y_2Y_1Y_0$ are the BCD inputs. The outputs of adder 1 ($S_3S_2S_1S_0$ and C_{out}) are checked to ascertain whether the output is greater than 9 by AND-OR gate combinations. If correction is required, then a 0110 is added with the output of adder 1. Now, the adder 2 output forms the BCD result ($\Sigma_3\,\Sigma_2\,\Sigma_1\,\Sigma_0$) with carry output ($C_{out}$).

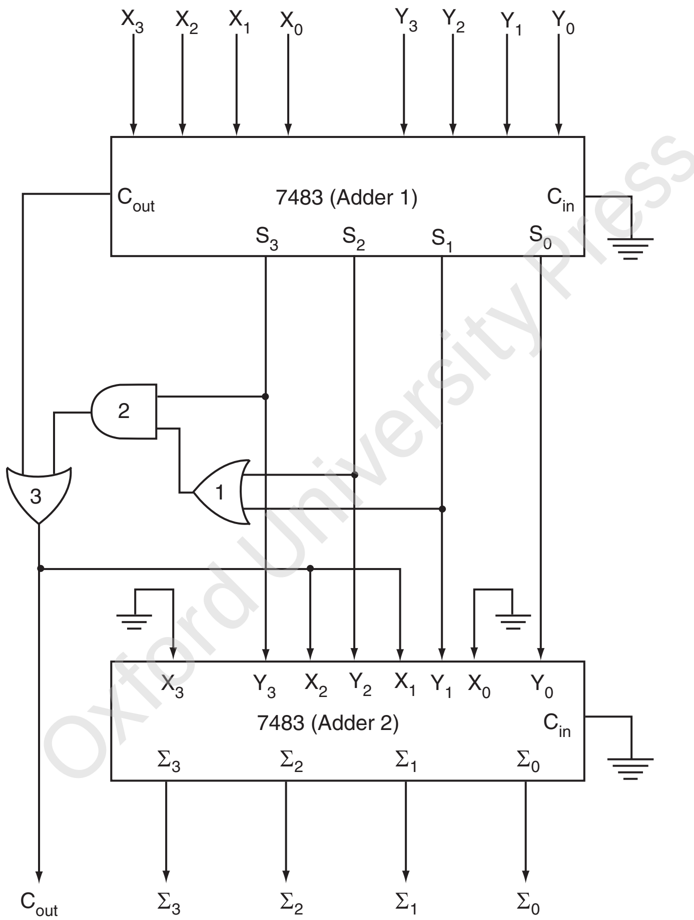


Fig. 5.23 BCD adder using 7483 ICs

5.16 Binary Multiplier

Multiplication operation can be carried out by (i) Multipliers using partial product addition and shifting and (ii) Parallel multipliers.

5.16.1 Multiplier using Shift Method

To understand the multiplication process using shift method, consider the multiplication of two 4-bit binary numbers 1010 and 1011, as an example.

$$
\begin{array}{r l}
1010 & \rightarrow \text{Multiplicand} \\
\times\ 1011 & \rightarrow \text{Multiplier} \\
\hline
1010 & \rightarrow \text{Partial product 1} \\
1010 & \rightarrow \text{Partial product 2} \\
0000 & \rightarrow \text{Partial product 3} \\
1010 & \rightarrow \text{Partial product 4} \\
\hline
1101110 & \\
\hline
\end{array}
$$

From the above multiplication process, one can easily understand that if the multiplier bit is 1, then the multiplicand is simply copied as a partial product; if the multiplier bit is 0, then the partial product is 0. Whenever a partial product is obtained, it is shifted one bit to the left of the previous partial product. This process is continued until all the multiplier bits are checked, and then the partial products are added. This multiplication process, i.e. multiplication by partial product addition and shifting, can be implemented using the block diagram shown in Fig. 5.24.

In the diagram shown in Fig. 5.24, the 4-bit multiplier is stored in register $Y(Y_3 Y_2 Y_1 Y_0)$; the 4-bit multiplicand is stored in register $M(M_3 M_2 M_1 M_0)$, and the X register $(X_4 X_3 X_2 X_1 X_0)$ is initially cleared to 00000. Here, to perform multiplication, the least significant bit of the multiplier bit (Y_0) is checked whether it is 0 or 1. If $Y_0 = 1$, the number in the multiplicand register (M) is added with the least significant 4-bits of X register ($X_3 X_2 X_1 X_0$; X_4 is to store carry in addition process) and the combined X and Y register is shifted to the right by 1 bit. If $Y_0 = 0$, the combined X and Y register is shifted to the right by 1 bit without performing any addition. This process has to be repeated four times to perform 4-bit multiplication. Now, the multiplication result $(R_7 R_6 R_5 R_4 R_3 R_2 R_1 R_0)$ will be available in X and Y registers $(X_3 X_2 X_1 X_0 Y_3 Y_2 Y_1 Y_0)$.

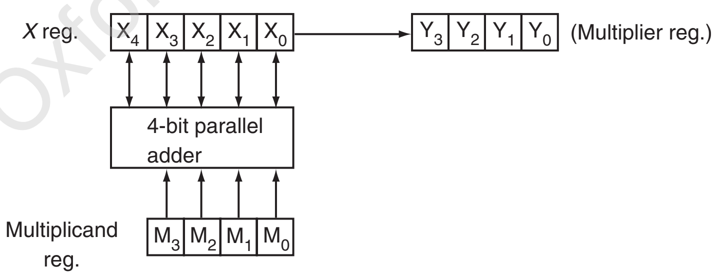


Fig. 5.24 4-bit binary multiplier using shift method

5.16.2 Parallel Multiplier

The 4-bit multiplier using shift method requires 4 cycles of addition and shifting operations, but it requires only a single 4-bit parallel adder. The speed of multiplication process can be increased considerably in parallel multiplier at the extra cost of increased hardware. The circuit diagram for a 4-bit parallel multiplier is shown in Fig. 5.25.

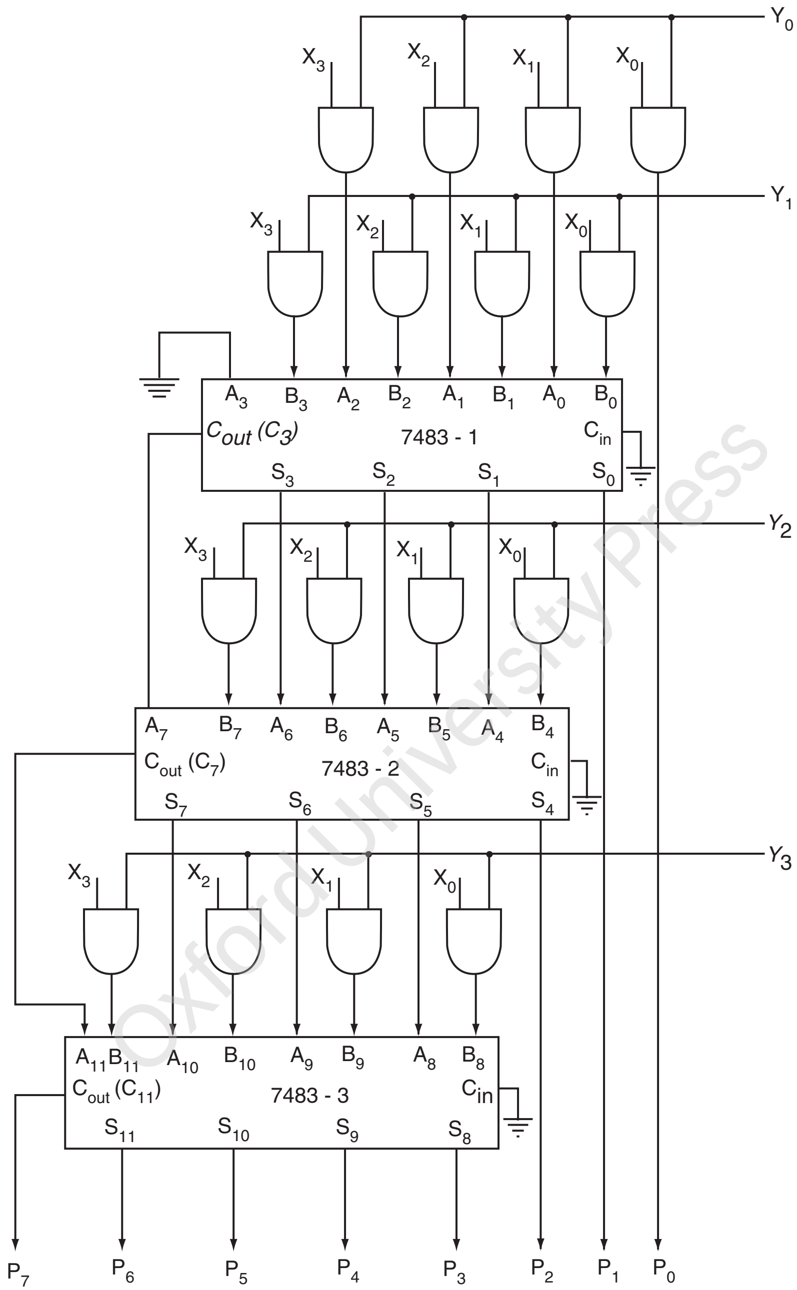


Fig. 5.25 4-bit parallel multiplier

It requires three 4-bit parallel binary adders and 16 numbers of 2-input AND gates. Here, each group of 4 AND gates is used to obtain partial products while 4-bit parallel adders are used to add the partial products. Since the generation of partial products and their additions are performed in parallel in the group of AND gates and 4-bit adders respectively, the multiplication result

($P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0$) will be available at the output immediately after the propagation delay in the multiplier circuit.

The operation of the parallel multiplier can be understood in a better manner from the symbolic form of binary multiplication process shown in Fig. 5.26.

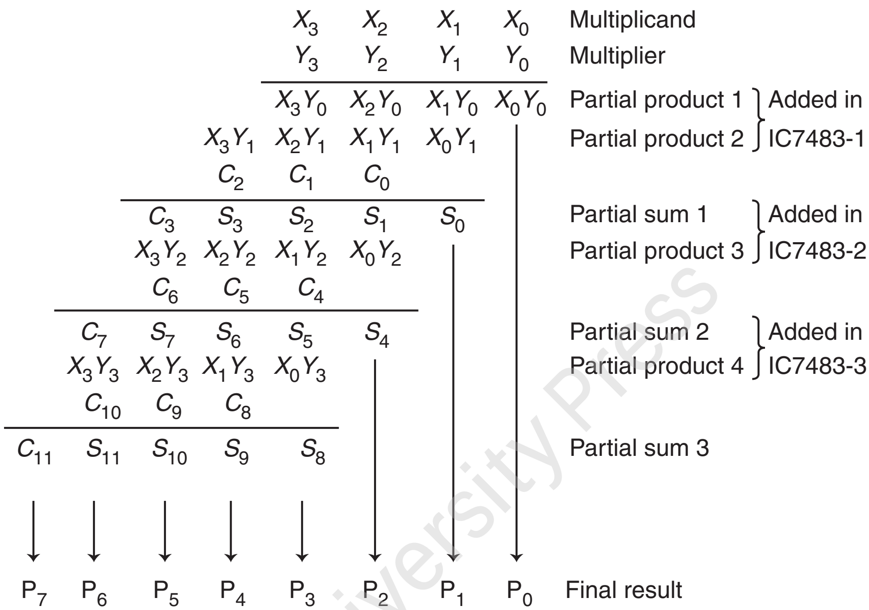


Fig. 5.26 Symbolic form of binary multiplication process

5.17 Binary Divider

Division is the most difficult and time-consuming operation for a general purpose computer. A block diagram of a binary divider unit using restoring technique for division is shown in Fig. 5.27.

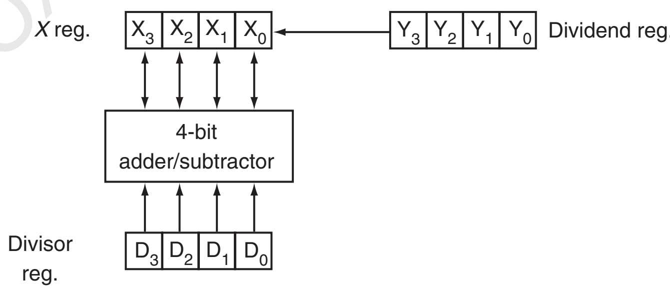


Fig. 5.27 Block diagram of a 4-bit binary divider

Here, the dividend is stored in the dividend register $Y(Y_3 Y_2 Y_1 Y_0)$, the divisor is stored in the divisor register $D(D_3 D_2 D_1 D_0)$ and initially the X register ($X_3 X_2 X_1 X_0$) is cleared.

The procedure for division operation using the circuit shown in Fig. 5.27 is explained as follows:

1. Shift the combined content of X and Y registers to the left by one bit.
2. Perform trial subtraction by subtracting the content of D register from the content of X register.
3. If there is no borrow in the previous subtraction, put 1 in the LSB of Y register, else restore the original content of X register by adding the contents of D register with the contents of X register.
4. Repeat steps 1 to 3 for n times, where n is the number of bits in the dividend. For a 4-bit division, $n = 4$.

Now, the quotient will be available in the Y register and the remainder will be in the X register. The above procedure can be understood in a better manner with the following example.

Consider the division of 1011(11) by 0011(3). The dividend and divisor are stored in Y and D registers respectively, and the X register is initially cleared to 0. Therefore,

$$Y_3 Y_2 Y_1 Y_0 = 1011$$
$$X_3 X_2 X_1 X_0 = 0000$$
$$D_3 D_2 D_1 D_0 = 0011$$

Here, both divisor and dividend are 4-bit numbers. Therefore, steps 1–3 have to be repeated four times.

I cycle

Step 1 Shift the combined contents of X and Y to the left by one bit. Therefore,

$$XY = 0001 \quad 0110$$

Step 2 Subtract dividend 0011 from register X resulting in

$$X = 1110 \quad \text{with borrow} = 1$$

Step 3 Since borrow = 1, restore the original content in X register by adding dividend 0011 with the content of X register 1110. Now,

$$XY = 0001 \quad 0110$$

II cycle

Step 1 Shift the combined contents of X and Y to the left by one bit. Therefore,

$$XY = 0010 \quad 1100$$

Step 2 Subtract dividend 0011 from X register resulting in

$$X = 1111 \quad \text{with borrow} = 1$$

Step 3 Since borrow = 1, restore the original content in X register by adding dividend 0011 with the content of X register 1111. Now,

$$XY = 0010 \quad 1100$$

III cycle

Step 1 Shift the combined contents of X and Y to the left by one bit. Therefore,

$$XY = 0101 \quad 1000$$

Step 2 Subtract dividend 0011 from register X which results in

$$X = 0010 \text{ with borrow} = 0$$

Step 3 Since borrow = 0, put 1 in the LSB of Y register (Y_0). Therefore,

$$XY = 0010 \quad 1001$$

IV cycle

Step 1 Shift the combined contents of X and Y to the left by one bit. Therefore,

$$XY = 0101 \quad 0010$$

Step 2 Subtract dividend 0011 from X register resulting in

$$X = 0010 \text{ with borrow} = 0$$

Step 3 Since borrow = 0, put 1 in the LSB of Y register (Y_0). Therefore,

$$XY = 0010 \quad 0011$$

Now, the quotient 0011(3) is available in the Y register and the remainder 0010(2) is available in the X register.

REVIEW QUESTIONS

1. What is the need of arithmetic circuits?
2. What is a half-adder? Write its truth table.
3. Design a half-adder using only NOR gates.
4. Describe the working of a half-adder.
5. What is a full-adder?
6. Draw the full-adder circuit using NAND gates only. Explain the functioning of the circuit and show that the output is that of a full-adder.
7. Design a full-adder circuit using only NOR gates. What relation has it to the half-adder circuit?
8. Design a full-subtractor using only NAND gates.
9. Design a full-subtractor using only basic gates.
10. Design a half-subtractor using only NAND gates.
11. Design a half-subtractor using only basic gates.
12. Design a half-subtractor using only NOR gates.
13. Design a full-subtractor using only NOR gates.
14. What is the difference between a full-adder and a full-subtractor?
15. Design the logic diagram of a circuit for addition/subtraction. Use a control variable w and a circuit that functions as a full-adder when $w = 0$, as a full-subtractor when $w = 1$.
16. Show how a full-adder can be converted to a full-subtractor with the addition of an inverter circuit.
17. What are the advantages of complement arithmetic?
18. Design a parallel binary multiplier that multiplies a 4-bit number $B = B_3B_2B_1B_0$ by a 3-bit number $A = A_2A_1A_0$ to form the product $C = C_6C_5C_4C_3C_2C_1C_0$.
    [**Hint**: This can be done with 12 gates and two 4-bit parallel adders. The AND gates are used to form the products of pairs of bits. The partial products formed by the AND gates are added with the parallel adder.]