EC 8392 – DIGITAL ELECTRONICS

UNIT – II : COMBINATIONAL CIRCUIT DESIGN

Binary Adder (Parallel Adder):

The 4-bit binary 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 figure below.

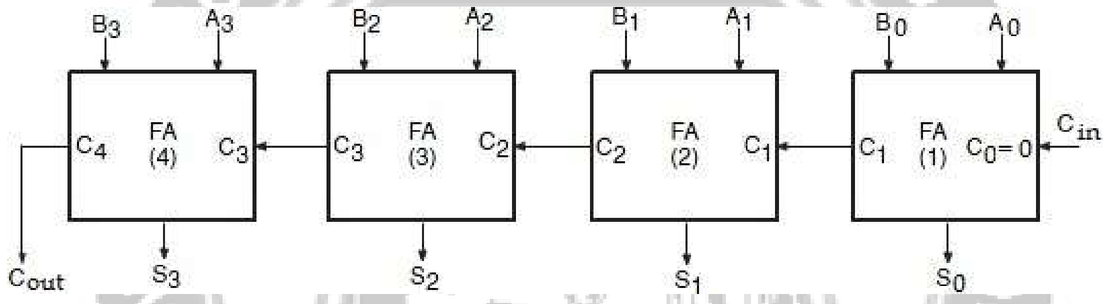
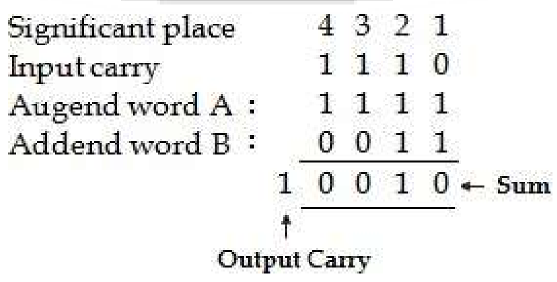


Fig : 2.11 - 4-bit binary parallel Adder

Since all the bits of 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.

Let the 4-bit words to be added be represented by, $A_3A_2A_1A_0$= 1111 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					

The bits are added with full adders, starting from the least significant position, to form the sum it and carry bit. The input carry C_0 in the least significant position must be 0.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 sum S_1 and carry $C_{2,}$ in the third stage, A_2, B_2 and C_2 are added resulting in sum S_2 and carry $C_{3,}$ in the third stage, A_3, B_3 and C_3 are added resulting in sum 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. 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.

Carry Propagation–Look-Ahead Carry Generator:

In Parallel adder, all the bits of the augend and the addend are available for computation at the same time. The carry output of each full-adder stage is connected to the carry input of the next high-order stage. Since each bit of the sum output depends on the value of the input carry, time delay occurs in the addition process. This time delay is called as **carry propagation delay.**

For example, addition of two numbers (0011+ 0101) gives the result as 1000. Addition of the LSB position produces a carry into the second position. This carry when added to the bits of the second position, produces a carry into the third position. This carry when added to bits of the third position, produces a carry into the last position. The sum bit generated in the last position (MSB) depends on the carry that was generated by the addition in the previous position. i.e., the adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in an each full-adder. For example, if each full adder is considered to have a propagation delay of

30nsec, then S_3 will not react its correct value until 90 nsec after LSB is generated. Therefore total time required to perform addition is 90+ 30 = 120nsec.

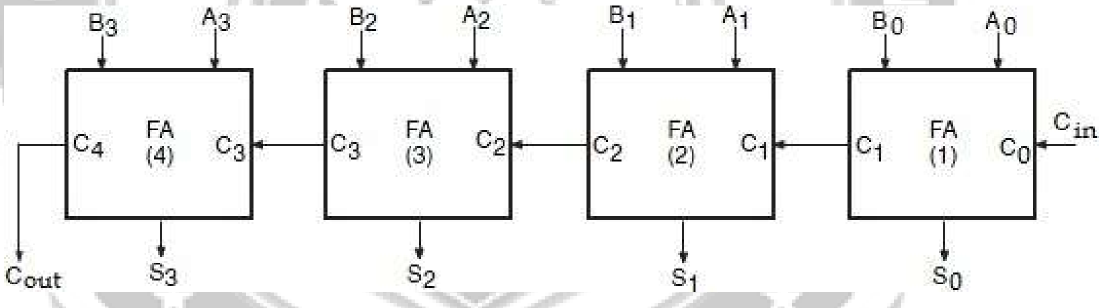


Fig : 2.12 - 4-bit Parallel Adder

The method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition**. This method utilizes logic gates to look at the lower order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions: carry generate and carry propagate.

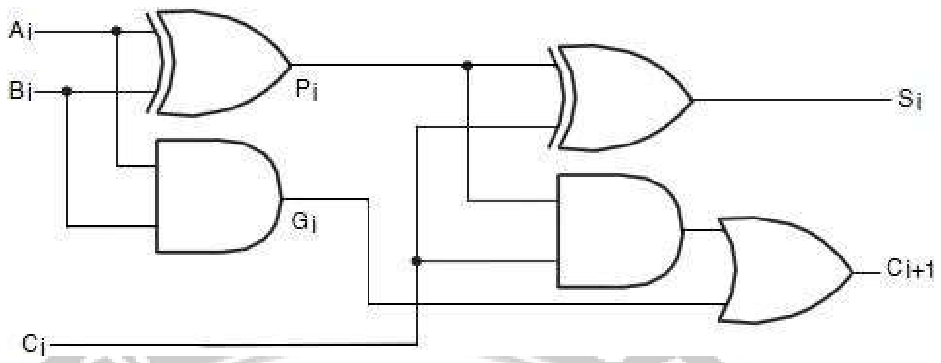


Fig : 2.13 - Full-Adder circuit

Consider the circuit of the full-adder shown above. Here we define two functions: carry generate (G_i) and carry propagate (P_i) as,

Carry generate, $\mathbf{G_i = A_i \quad B_i}$

Carry propagate, $\mathbf{P_i = A_i \quad B_i}$ the output sum and carry can be expressed as,

Si = Pi Ci

Ci+1 = Gi PiCi

G_i (carry generate), it produces a carry 1 when both Ai and Bi are 1, regardless of the input carry C_i.

Pi (carry propagate) because it is the term associated with the propagation of the carry from C_i to C_{i+1}.

The Boolean functions for the carry outputs of each stage and substitute for each C_i its value from the previous equation:

C_0= input carry

C1= G0 + P0C0

C2= G1 + P1C1 = G1 + P1 (G0 + P0C0)

= G1 + P1G0 + P1P0C0

C3= G2 + P2C2 = G2 + P2 (G1 + P1G0 + P1P0C0)

= G2 + P2G1 + P2P1G0 + P2P1P0C0

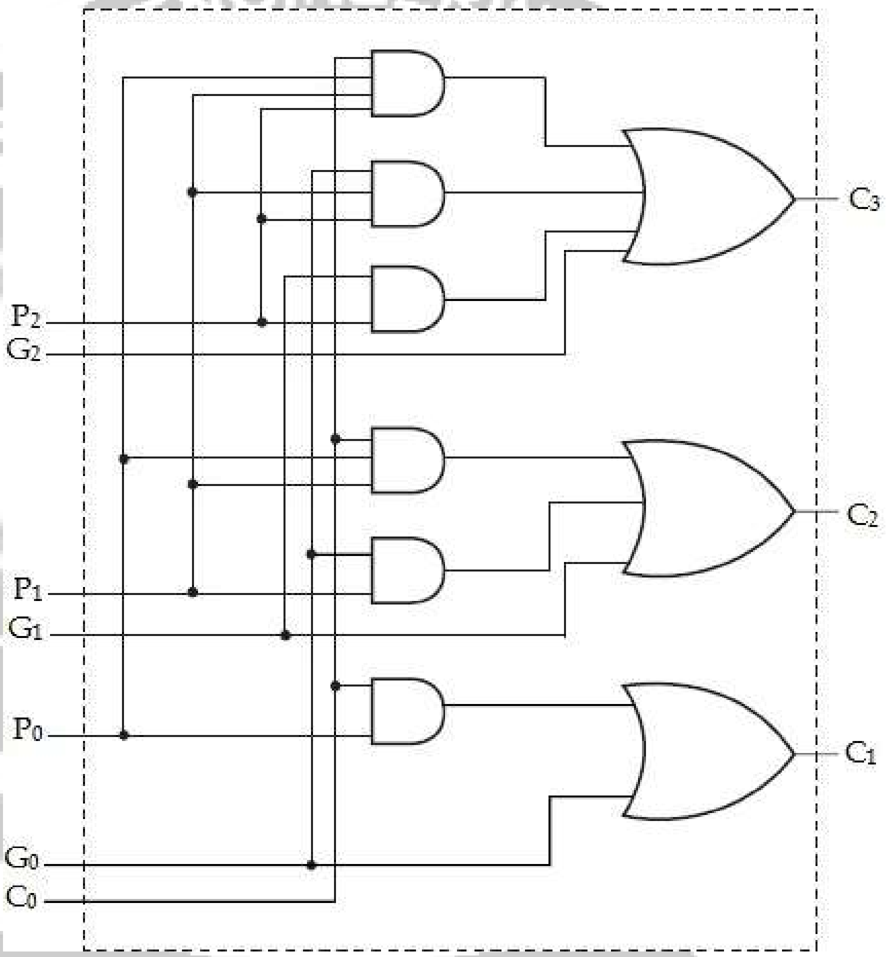


Fig : 2.14 - Logic diagram of Carry Look-ahead Generator

Since the Boolean function for each output carry is expressed in sum of products, each function can be implemented with one level of AND gates followed by an OR gate. The three Boolean functions for C_1, C_2 and C_3 are implemented in the carry look-ahead generator as shown below. Note that C_3 does not have to wait for C_2 and C_1 to propagate; in fact C_3 is propagated at the same time as C_1 and C_2.

Using a Look-ahead Generator we can easily construct a 4-bit parallel adder with a Look-ahead carry scheme. Each sum output requires two exclusive-OR gates. The output of the first exclusive-OR gate generates the P_i variable, and the AND gate generates the G_i variable. The carries are propagated through the carry look-ahead generator and applied as inputs to the second exclusive-OR gate. All output carries are generated after a delay through two levels of gates. Thus, outputs S_1 through S_3 have equal propagation delay times.

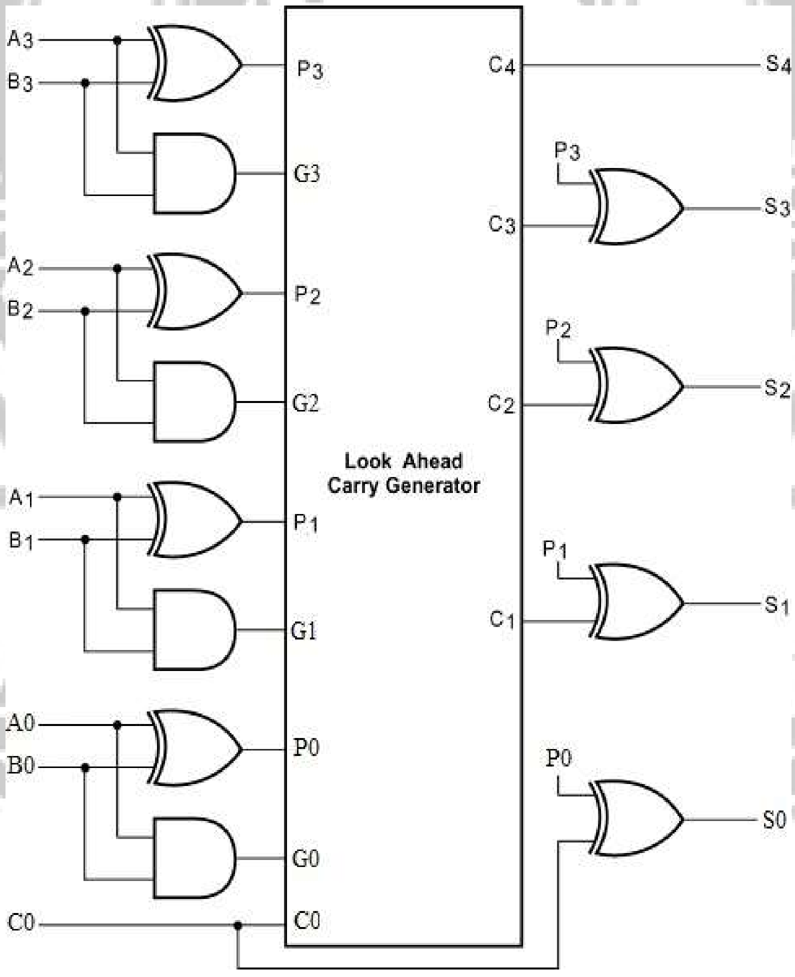


Fig : 2.15 - 4-Bit Adder with Carry Look-ahead

<u>Binary Subtractor (Parallel Subtractor):</u>

The subtraction of unsigned binary numbers can be done most conveniently by means of complements. The subtraction A-B can be done by taking the 2‘s complement of B and adding it to A. The 2‘s complement can be obtained by taking the 1‘s complement and adding 1 to the least significant pair of bits. The 1‘s complement can be implemented with inverters and a 1 can be added to the sum through the input carry.

The circuit for subtracting A-B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when performing subtraction. The operation thus performed becomes A, plus the 1‘s complement of B, plus1. This is equal to A plus the
2‘s complement of B.

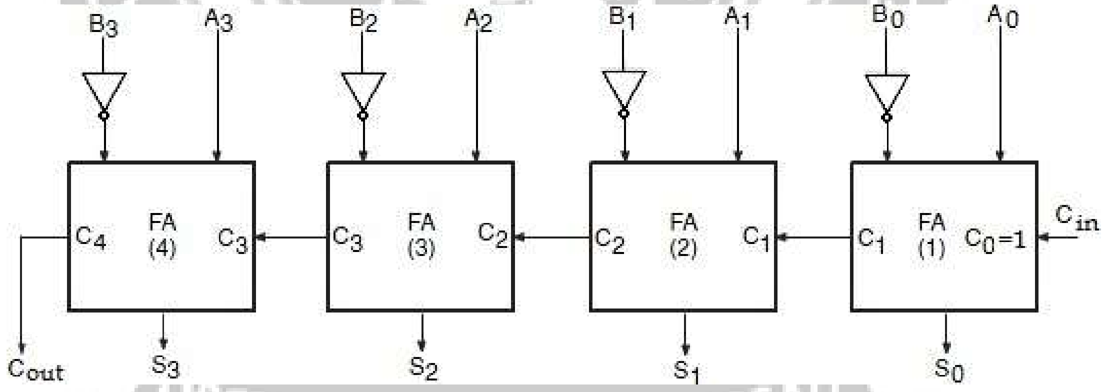


Fig : 2.16 - 4-bit Parallel Subtractor

<u>Parallel Adder/ Subtractor:</u>

The addition and subtraction operation can be combined into one circuit with one common binary adder. This is done by including an exclusive-OR gate with each full adder. A 4-bit adder Subtractor circuit is shown below.

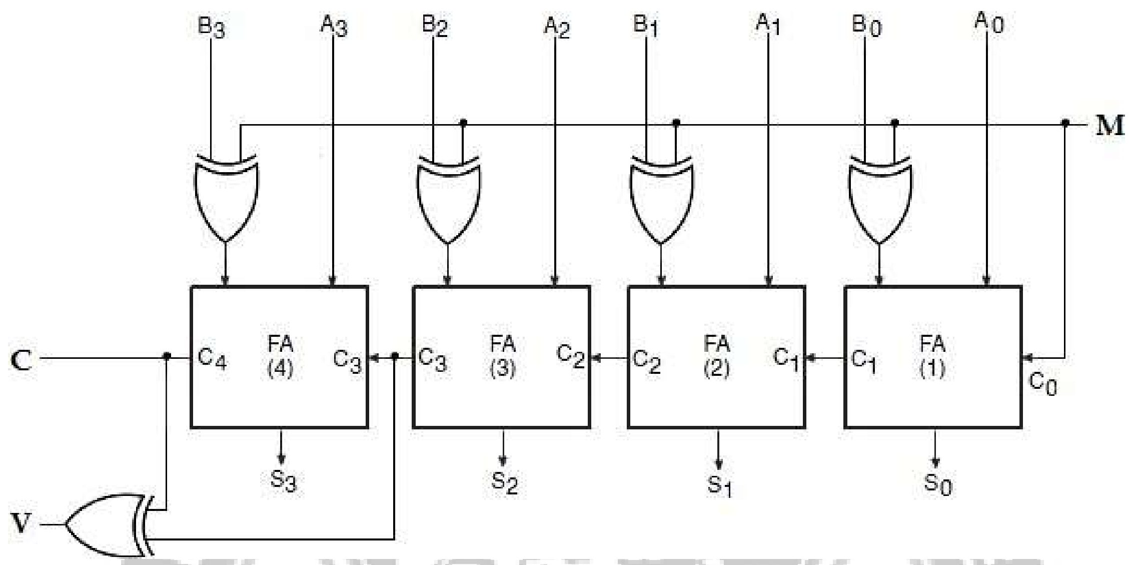


Fig : 2.17 - 4-Bit Adder Subtractor

The mode input M controls the operation. When M= 0, the circuit is an adder and when M=1, the circuit becomes a Subtractor. Each exclusive-OR gate receives input M and one of the inputs of B. When M=0, we have B 0= B. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B. When M=1, we have B 1= B‘ and C_0=1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2‘s complement of B. The exclusive-OR with output V is for detecting an overflow.

Decimal Adder (BCD Adder):

The digital system handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD bits and produces a sum digit also in BCD.

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9+ 9+1 = 19; the 1 is the sum being an input carry. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

These binary numbers are labeled by symbols K, Z_8, Z_4, Z_2, Z_1, K is the carry. The columns under the binary sum list the binary values that appear in the outputs of the 4-bit binary adder. The output sum of the two decimal digits must be represented in BCD.

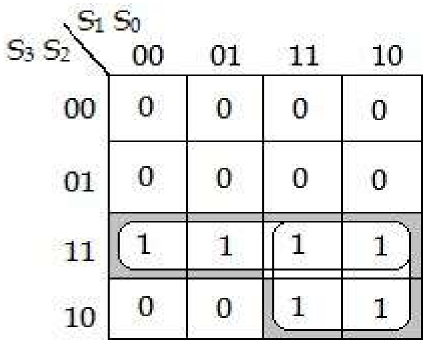
Binary Sum					**BCD Sum**					**Decimal**
K	**Z8**	**Z4**	**Z2**	**Z1**	**C**	**S8**	**S4**	**S2**	**S1**	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9

0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

In examining the contents of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 9 (1001), we obtain a non-valid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of the given truth table.

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$Y = S_3S_2 + S_3S_1$

To implement BCD adder we require:

x4-bit binary adder for initial addition xLogic circuit to detect sum greater than 9 and

xOne more 4-bit adder to add 0110_2 in the sum if the sum is greater than 9 or carry is 1.

The two decimal digits, together with the input carry, are first added in the top4-bit binary adder to provide the binary sum. When the output carry is equal to zero, nothing is added to the binary sum. When it is equal to one, binary 0110 is added to the binary sum through the bottom 4-bit adder. The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal. The output carry from one stage must be connected to the input carry of the next higher-order stage.

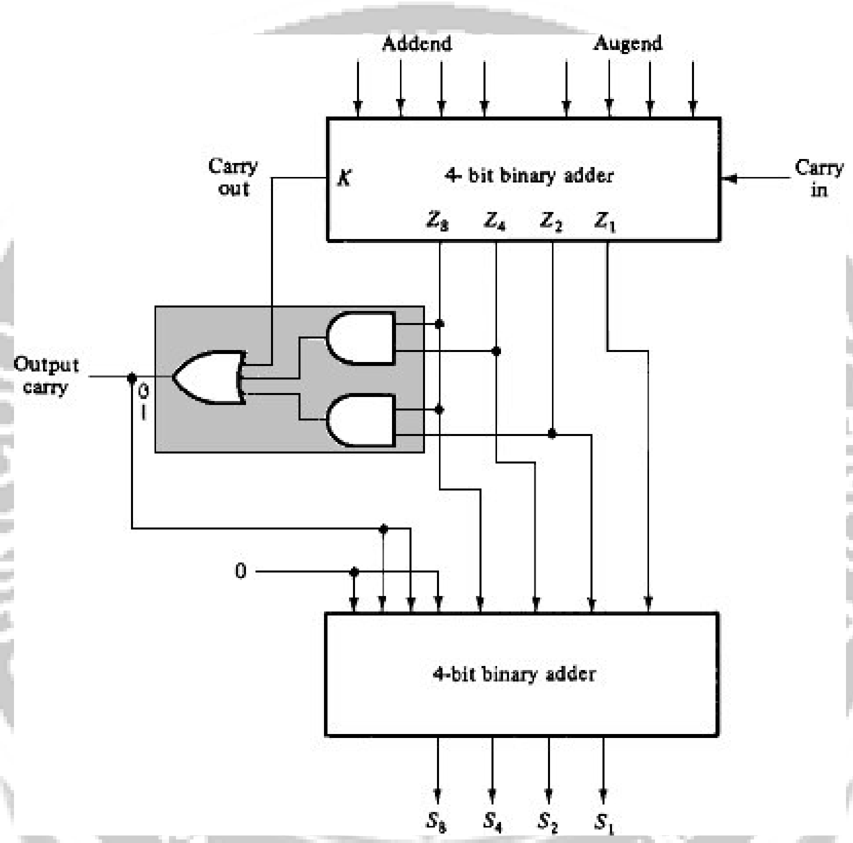


Fig : 2.18 - Block diagram of BCD adder

Binary Multiplier:

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

Consider the multiplication of two 2-bit numbers. The multiplicand bits are B_1 and B_0, the multiplier bits are A_1 and A_0, and the product is C_3, C_2, C_1 and C_0. The first partial product is formed by multiplying A_0 by B_1B_0. The multiplication of two bits such as A_0 and B_0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation. Therefore the partial product can be implemented with AND gates as shown in the diagram below.

The second partial product is formed by multiplying A_1 by B_1B_0 and shifted one position to the left. The two partial products are added with two half adder (HA) circuits.

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

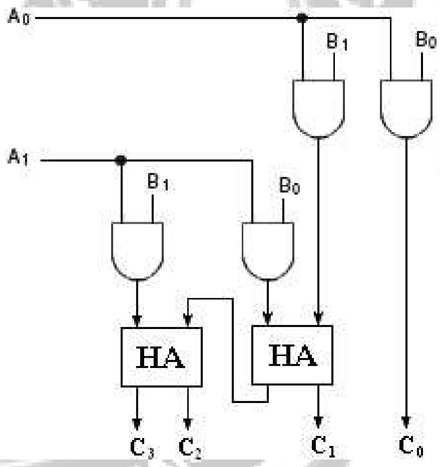


Fig : 2.19 - 2-bit by 2-bit Binary multiplier

Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products. The least significant bit of the product does not have to go through an adder since it is formed by the output of the first AND gate.

A combinational circuit binary multiplier with more bits can be constructed in a similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier. The binary output in each level of AND gates are added with the partial product of the previous level to form a new partial product. The last level produces the product. For J multiplier bits and K multiplicand bits we need (J x K) AND gates and (J-1) k-bit adders to produce a product of J+K bits.

Consider a multiplier circuit that multiplies a binary number of four bits by a

number of three bits. Let the multiplicand be represented by B3, B2, B1, B0 and the multiplier by A_2, A_1, and A_0. Since K= 4 and J= 3, we need 12 AND gates and two 4-bit adders to produce a product of seven bits. The logic diagram of the multiplier is shown below.

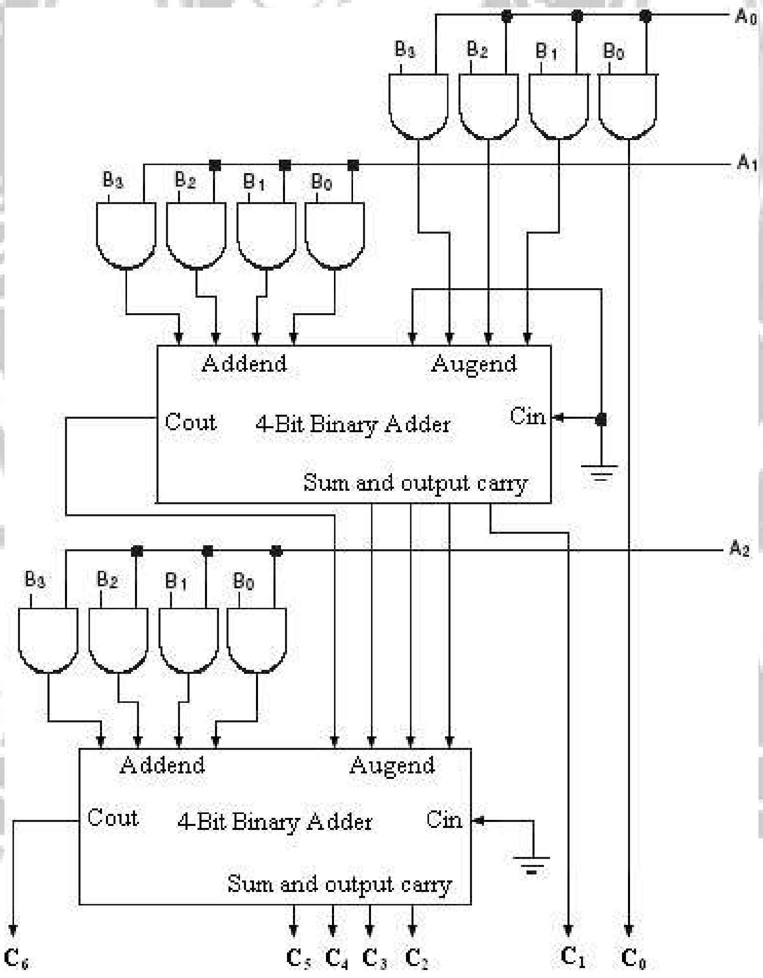


Fig : 2.20 - 4-bit by 3-bit Binary multiplier