

Parallel multipliers :-

High speed parallel multipliers are becoming one of the keys in RISCs (Reduced instruction set computers), DSPs (Digital signal processors), graphics accelerators and so on. Parallel multipliers are used in data processors as well as digital signal processors.

For example, for multi-media applications 16×16 fast multipliers are needed. For floating point unit using double-precision multiplication (IEEE-754 standard), the mantissa data has 52-bit. Then 54×54 are required for such an operation. The two added bits are the sign bit and the guard bit. In this section we discuss several parallel multiplier algorithms which have been used in ULSI. The reader can consult reference [7, 8] for more details on array multiplication algorithms.

Braun multiplier :-

Consider two unsigned numbers $X = X_{n-1} \dots X_1 X_0$ and $Y = Y_{n-1} \dots Y_1 Y_0$

$$X = \sum_{i=0}^{i=n-1} X_i 2^i \rightarrow \textcircled{1}$$

$$Y = \sum_{i=0}^{i=n-1} Y_i 2^i \rightarrow \textcircled{2}$$

The product $P = P_{2n-1} \dots P_1 P_0$, which results from multiplying the multiplicand X by the multiplier Y , can be written in the following form,

$$P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (x_i y_j) 2^{i+j} \rightarrow (3)$$

→ Each of the partial product terms $P_k = x_i y_j$ is called summand.

→ The summands are generated in parallel with AND gates.

→ The Braun's array multiplier, such as multiplier of $n \times n$, requires $n(n-1)$ adders and n^2 AND gates.

→ The adder can be implemented efficiently by arranging the array for a regular layout.

* The routing lines are also illustrated in these cells. The last cell represents a full-adder composing the final carry propagate added. The multiplier array is using what is called carry-save adders.

* The delay of such a multiplier is dependent on the delay of the full adder cell and the final adder in the last row. In the multiplier array, an adder with balanced carry and sum delays is desirable because sum and carry signals are both on the critical path. This is different than the case of a parallel adder where the carry path should be optimized and speed up compared to the sum path. For large arrays, the speed and power of the full-adder are very important.

→ The final adder in the last row can use the techniques presented in section.

$$X_3 \ X_2 \ X_1 \ X_0 = X$$

$$Y_3 \ Y_2 \ Y_1 \ Y_0 = Y$$

$$X_3Y_0 \ X_2Y_0 \ X_1Y_0 \ X_0Y_0$$

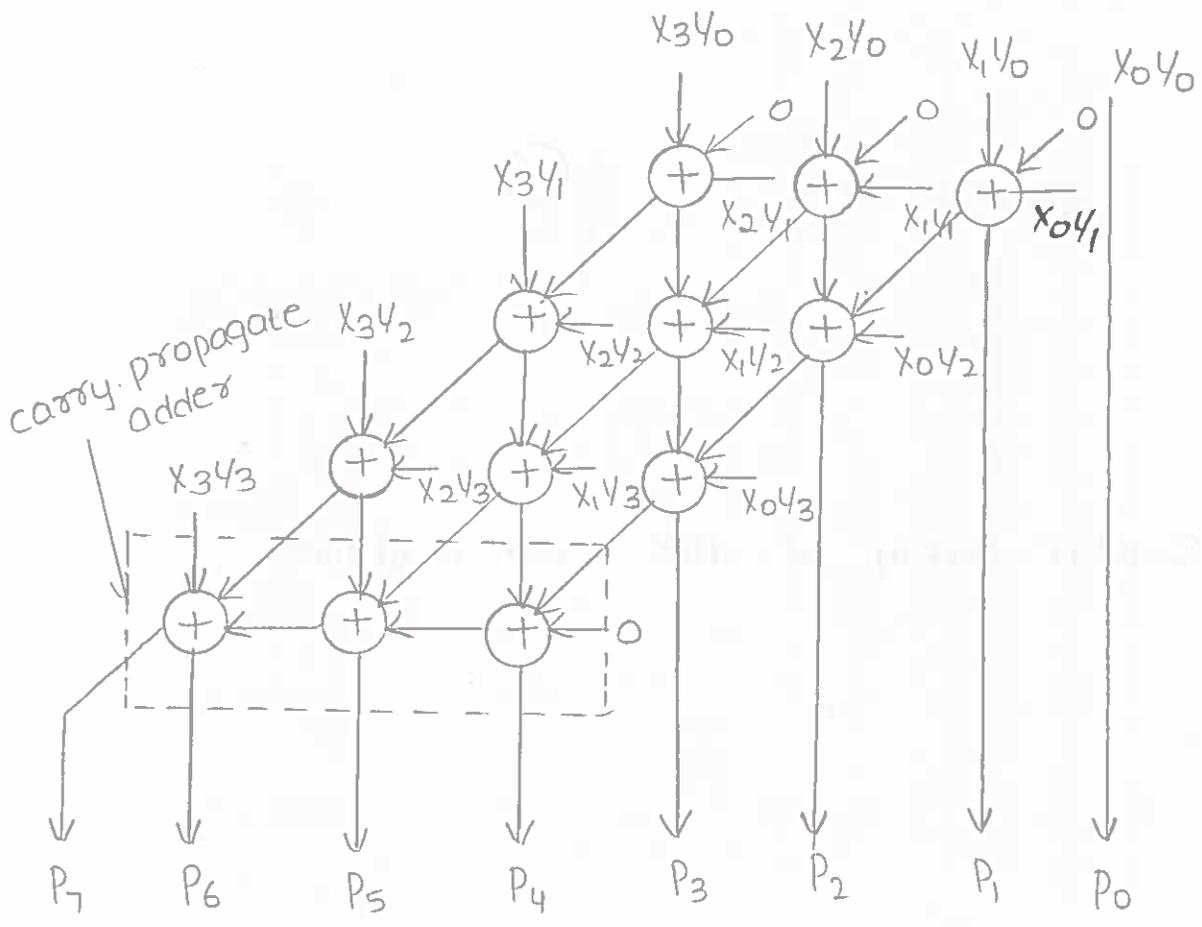
$$X_3Y_1 \ X_2Y_1 \ X_1Y_1 \ X_0Y_1$$

$$X_3Y_2 \ X_2Y_2 \ X_1Y_2 \ X_0Y_2$$

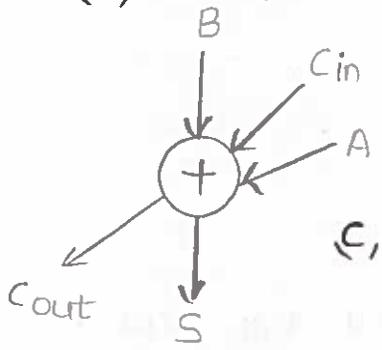
$$X_3Y_3 \ X_2Y_3 \ X_1Y_3 \ X_0Y_3$$

$$P_7 \ P_6 \ P_5 \ P_4 \ P_3 \ P_2 \ P_1 \ P_0 = P = X * Y$$

a, partial products of a 4x4 unsigned integer multiplication.

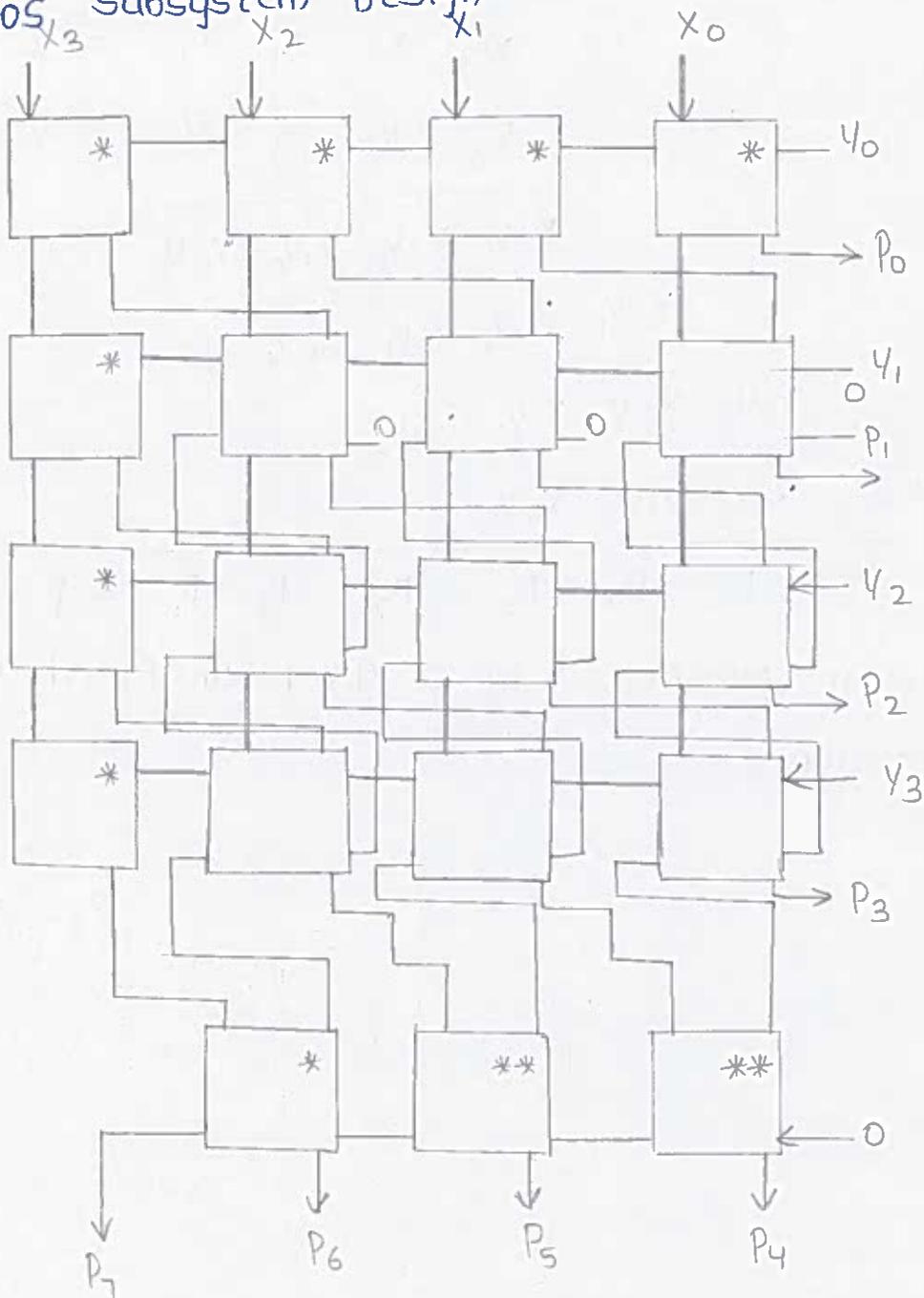


b, Multiplier array.

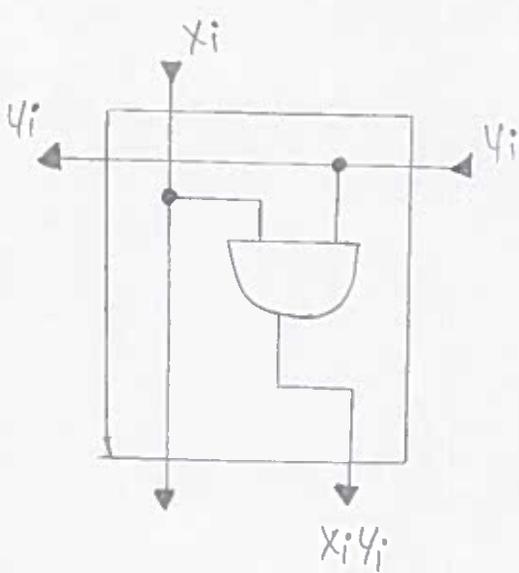


c, Full-adder schematic

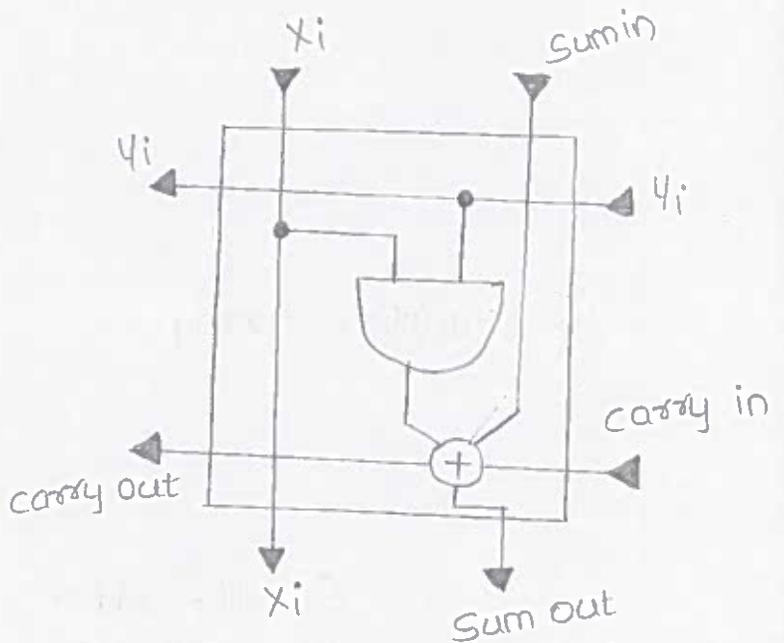
VLSI CMOS Subsystem Design



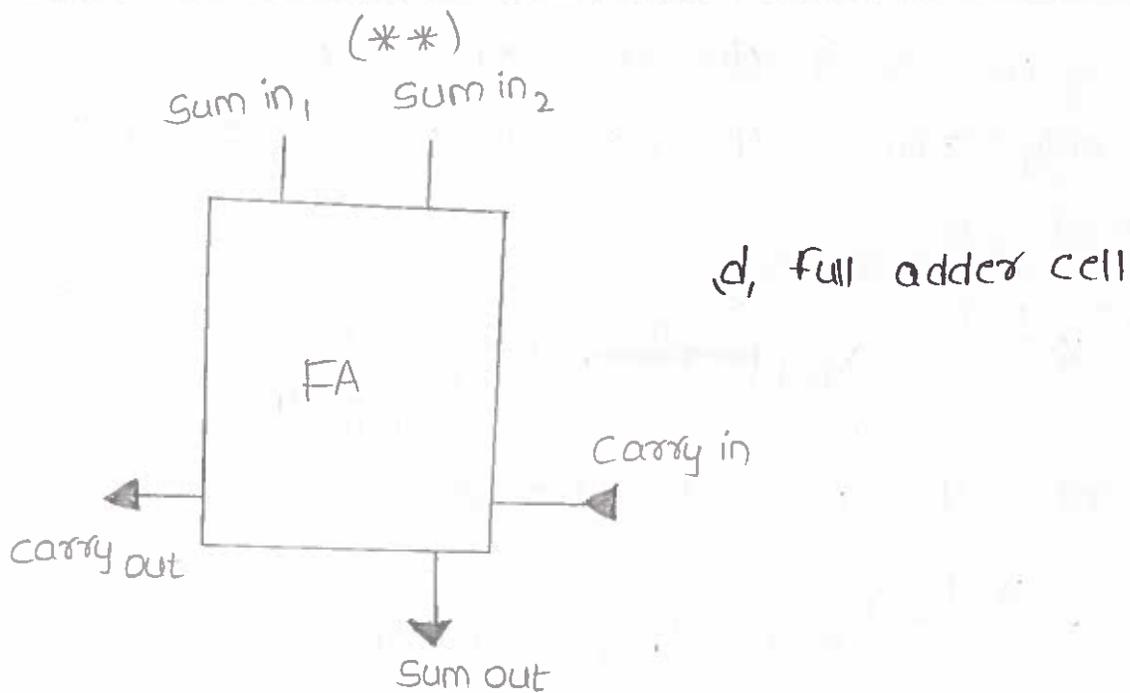
a, Regular array of the 4x4 multipliers



b, AND cell



c, AND with full-added cell



* Baugh - wooley multiplier:-

The Baugh wooley technique was developed to design regular direct multipliers for two's complement numbers. This direct approach does not need any two's complementing operations prior to multiplication.

Let us consider two numbers X and Y with the following form,

$$X = -X_{n-1}2^{n-1} + \sum_{i=0}^{i=n-2} X_i 2^i \rightarrow \textcircled{1}$$

$$Y = -Y_{n-1}2^{n-1} + \sum_{i=0}^{i=n-2} Y_i 2^i \rightarrow \textcircled{2}$$

The product $P = XY$ is given by the following equation,

$$P = XY = X_{n-1}Y_{n-1}2^{2n-2} + \sum_{i=0}^{i=n-2} \sum_{j=0}^{j=n-2} X_i Y_j 2^{i+j}$$

$$= -X_{n-1} \sum_{i=0}^{i=n-2} Y_i 2^{n+i-1} - Y_{n-1} \sum_{i=0}^{i=n-2} X_i 2^{n+i-1} \rightarrow \textcircled{3}$$

In order to avoid the use of subtractor cells and use only adders, the negative terms should be transformed. So,

$$-X_{n-1} \sum_{i=0}^{i=n-2} Y_i 2^{n+i-1} = X_{n-1} \left(-2^{2n-2} + 2^{n-1} + \sum_{i=0}^{i=n-2} \bar{Y}_i 2^{n+i-1} \right) \rightarrow (4)$$

using eq'n (2), the product 'p' becomes,

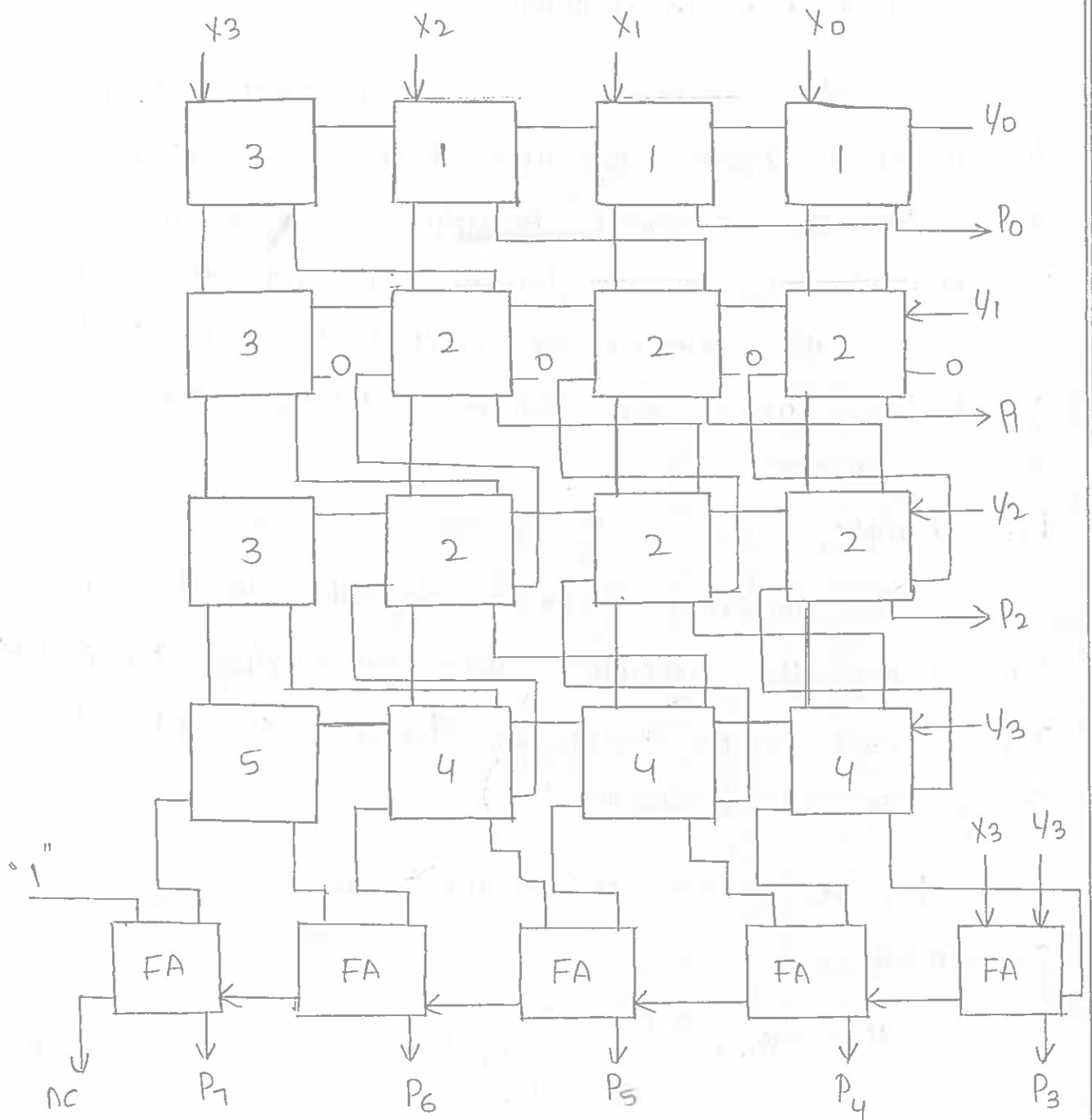
$$p = XY = -2^{2n-1} + (\bar{X}_{n-1} + \bar{Y}_{n-1} + X_{n-1} Y_{n-1}) \cdot 2^{2n-2} + \sum_{i=0}^{i=n-2} \sum_{j=0}^{j=n-2} X_i Y_j 2^{i+j} + (X_{n-1} + Y_{n-1}) \cdot 2^{n-1} + X_{n-1} \sum_{i=0}^{i=n-2} \bar{Y}_i 2^{n+i-1} + Y_{n-1} \sum_{i=0}^{i=n-2} \bar{X}_i 2^{n+i-1} \rightarrow (5)$$

using the above relation an $n \times n$ multiplier, using only adders, can be implemented. The schematic circuit diagram of a 4×4 two's complement multiplier based on Baugh-wooley's algorithm:

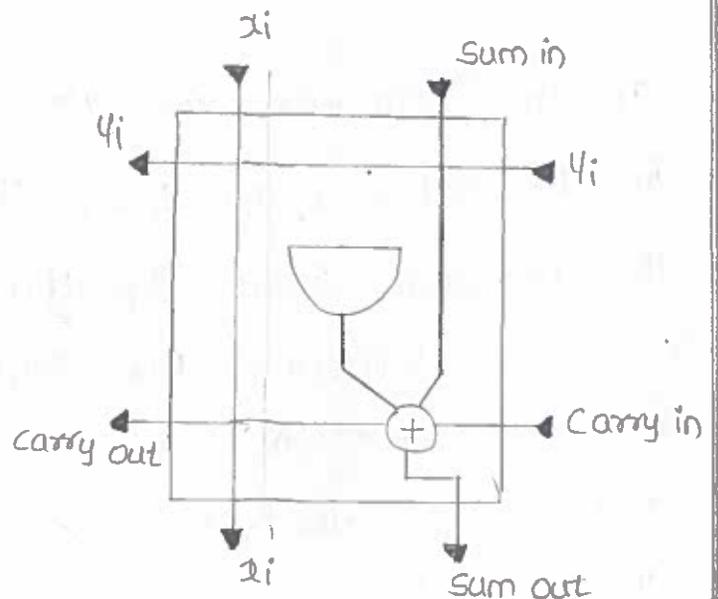
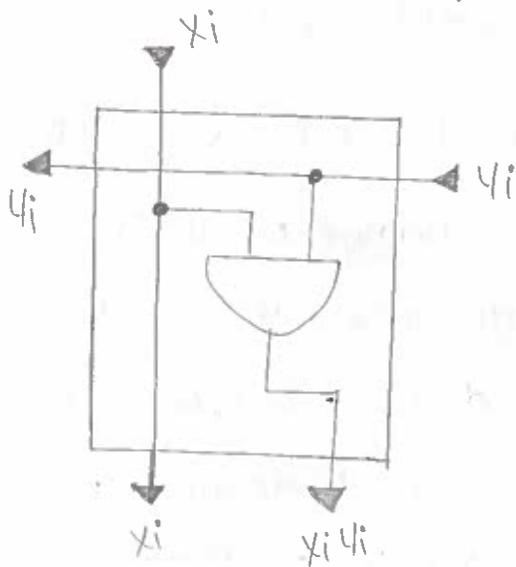
⇒ In this scheme $n(n+1)+3$ full adders are required. So, for the case of $n=4$ the array needs 15 adders.

⇒ This type of multiplier is suitable for applications where operands with less than 16 bits are to be processed. Applications for such a multiplier are, for example, for digital filters where small operands are used.

→ For operands equal (or) greater than 16-bit, the Baugh-wooley scheme becomes too area-consuming and slow.



a, 4x4 Baugh - wooley two's complement regular array (FA: Full-adder)



* The modified booth multiplier:-

For operands equal (or) greater than 16 bits the modified booth algorithm have been used in almost all the designed multipliers, It is based on recoding the two's complement operand in order to reduce the number of partial products to be added. This makes the multiplier faster and uses less hardware,

For example,

The modified radix-2 algorithm is based on partitioning the multiplier into overlapping groups of 3-bits and each group is decoded to generate the correct partial products.

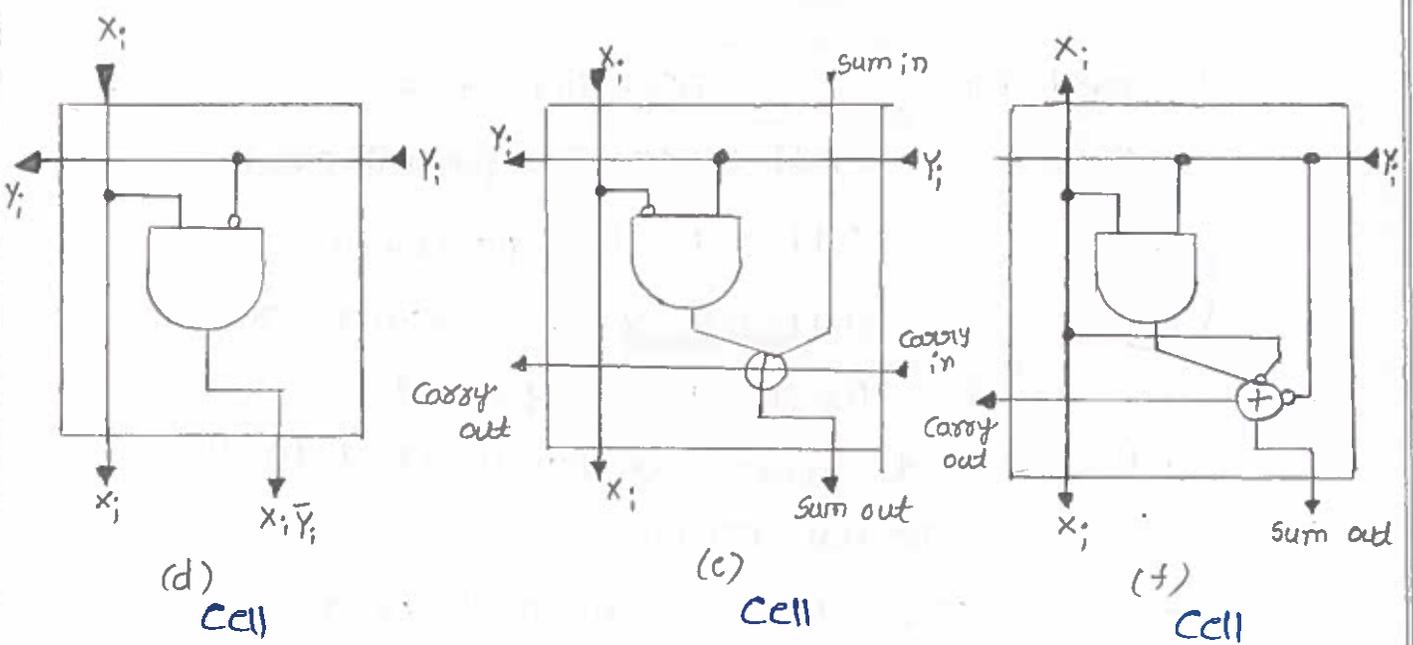
Let us write the multiplier y , in two's complement,

$$y = -y_{n-1}2^{n-1} + \sum_{i=0}^{i=n-2} y_i 2^i$$

It can be written as follows

$$y = \sum_{i=0}^{i=n/2-1} (y_{2i-1} + y_{2i} - 2y_{2i+1}) \cdot 2^{2i} \quad \text{with } y_{-1} = 0$$

In this equation, the terms in brackets have value in the set $\{-2, -1, 0, 1, 2\}$. The recoding of y , using the modified booth algorithm, generates another number with the following five signed digits, $-2, -1, 0, +1, +2$. Each recoded digit in the multiplier performs a certain operation on the multiplicand x , as illustrated in table 7.1



Y_{2i+1}	Y_{2i}	Y_{2i-1}	Recoded digit	operation on x
0	0	0	0	$0 \times x$
0	0	1	+1	$+1 \times x$
0	1	0	+1	$+1 \times x$
0	1	1	+2	$+2 \times x$
1	0	0	-2	$-2 \times x$
1	0	1	-1	$-1 \times x$
1	1	0	-1	$-1 \times x$
1	1	1	0	$0 \times x$

So the bits of the multiplier are partitioned into groups of overlapped 3-bits, each group permits generation of a certain partial product. The five possible multiples of the multiplicand are relatively easy to generate following the explanation given in table 7.2

The generated partial product is related to the multiplicand for each recoded digit by the relationships presented in table 7.3. PP_i is the partial product and PP_n is the sign bit of the partial product with $P_n = P_{n-1}$. When no shifting of the partial product is performed Note that the partial product is represented on $n+1$ bits

Recoded digit	operation on x
0	Add 0 to the partial product
+1	Add x to the partial product
+2	Shift left x one position and add it to the partial product.
-1	Add two's complement of x to the partial product.
-2	Take two's complement of x and shift left one position.

Table 7-2 :- partial product generation process.

Recoded digit	operation on x	Added to LSB
0	$PP_i = 0$ for $i=0, \dots, n$	0
+1	$PP_i = x_i$ for $i=0, \dots, n$	0
+2	$PP_i = x_{i-1}$ for $i=0, \dots, n$	0
-1	$PP_i = \bar{x}_i$ for $i=0, \dots, n$	1
-2	$PP_i = \bar{x}_{i-1}$ for $i=0, \dots, n$	1

To clarify this algorithm, an example is presented in fig 7-23. Let $x = 10010101$ and $y = 01101001$. The recoded digits of y are,

$$01101001 \rightarrow +2 -1 -2 +1$$

The bits are grouped into 3-bit groups overlapped by one bit and a bit with a value of zero is added on the right side of y as y_{-1} . So the multiplication of two 8-bit numbers generates only 4 partial products. The number is then reduced by half. The partial product in this example is represented on 9 bits. For a correct partial product's addition, the signs are

extended as shown in fig. 7.23. The shape of the multiplier is then trapezoidal due to the sign extension.

$$\begin{array}{r}
 (-107) \quad 10010101 = X \\
 (+105) \quad 01101001 = Y \\
 \hline
 \begin{array}{r}
 111111110010101 \\
 00000011010110 \\
 00000101011 \\
 1100101010 \\
 \hline
 1101010000011101 = P(-11235)
 \end{array}
 \end{array}$$

Operation	Bits-recorded
+1	010
-2	100
-1	101
+2	011

Fig :- 8 bit booth multiplication example using two complement's number.

→ In order to make the array rectangular, and then more regular for VLSI implementation, the problem of sign extension must be addressed. This problem is more crucial when the operand lengths are wide, where each partial product must be sign-extended to the length of the product. In this section we will not deal with the techniques to solve the problem of the sign extension.

The basic idea is to use two extra bits in the partial product. For the first partial product, the two additional bits, PP_{n+1} and PP_{n+2} are equal to the sign bit of the partial product.

$$PP_{n+2} = PP_{n+1} = PP_n$$

For the second partial product, if the first partial product was positive, then the two additional bits for this second partial product are given by the expression above, otherwise we have two cases,

$$PP_{n+1} = PP_{n+2} = 1 \quad \text{if } PP_n = 0$$

$$PP_{n+2} = \overline{PP_{n+1}} = 1 \text{ if } PP_n = 1$$

So, it is more interesting to use a third bit, F , as a flag to indicate whether there is, from the previous partial a negative sign bit to be propagated. F_i is the flag generated by the first partial product to the next one. For the example $F_0 = 0$ (no pp before the first one), and $F_1 = F_2 = F_3 = 1$. So for the first partial product there is a sign propagation to all the others. This flag is expressed by the following boolean equation,

$$F_{j+1} = F_j + PP_{nj}$$

$$(-107) \quad 10010101 = X$$

$$(+105) \quad 01101001 = Y$$

$\begin{array}{r} \boxed{\boxed{1}} \boxed{\boxed{1}} \boxed{1} \boxed{0} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \\ \boxed{\boxed{0}} \boxed{\boxed{1}} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \\ \boxed{\boxed{0}} \boxed{\boxed{0}} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \\ \boxed{\boxed{0}} \boxed{\boxed{0}} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \end{array}$	<table border="1"> <thead> <tr> <th>Operation</th> <th>Bits recorded</th> </tr> </thead> <tbody> <tr> <td>+1</td> <td>010</td> </tr> <tr> <td>-2</td> <td>100</td> </tr> <tr> <td>-1</td> <td>101</td> </tr> <tr> <td>+2</td> <td>011</td> </tr> </tbody> </table>	Operation	Bits recorded	+1	010	-2	100	-1	101	+2	011
Operation	Bits recorded										
+1	010										
-2	100										
-1	101										
+2	011										

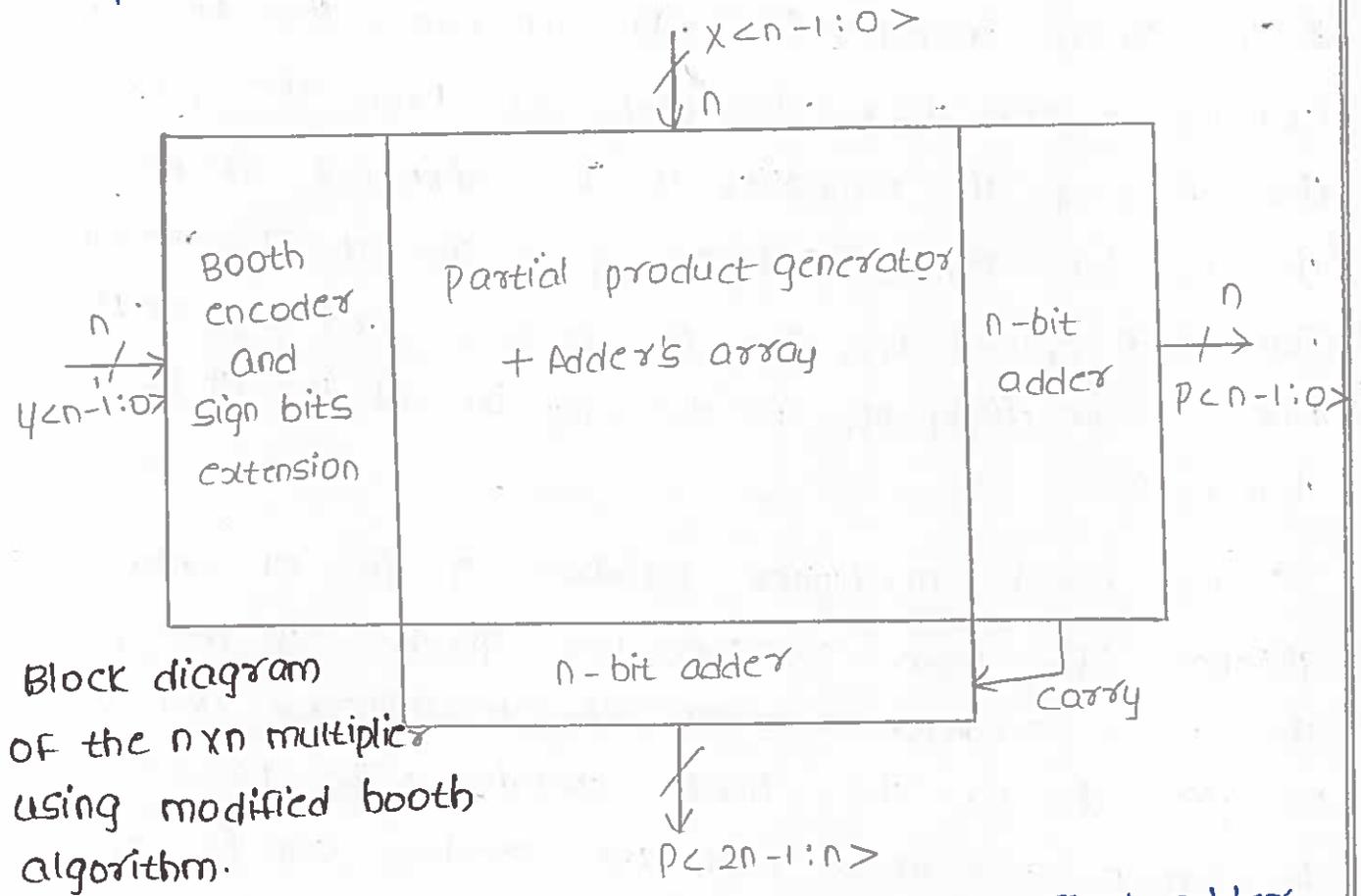
$$1101010000011101 = P (-11235)$$

- Additional bits to be generated [sign extension]
- Additional bits generated from the previous sign and the present sign.

→ where PP_{nj} is the sign bit of the j th partial product.

Let us now see the implementation of the $N \times N$ modified booth multiplier. Also it gives an idea about the floorplan of this subsystem. It is composed of the following blocks :-

- * The multiplier array containing partial product's generators and 1-bit adders;
 - * The Booth encoder and the sign extension bits (PP_{n+2}, PP_{n+1}, F). The Booth encoder generates the five signals (0, +1X, +2X, -1X, and -2X) for each group of 3-bit of 4; and
 - * The final stage adder performs 2n bits addition.
- ⇒ For the sake of simplicity, we treat the case of a 6x6 multiplier. All the cells described in this example are the basic cells of any multiplier size.



- * Four types of cells are used plus the final adder. These cells are,
- The ADD cell which generates 0 (or) 1. The schematic circuit of this cell is shown. Two implementations are possible: one using pass-transistors controlled by the five signals defining the recoded digit code, and the

another one is an AND₂ gate of the two signals $-1X$ and $-2X$.

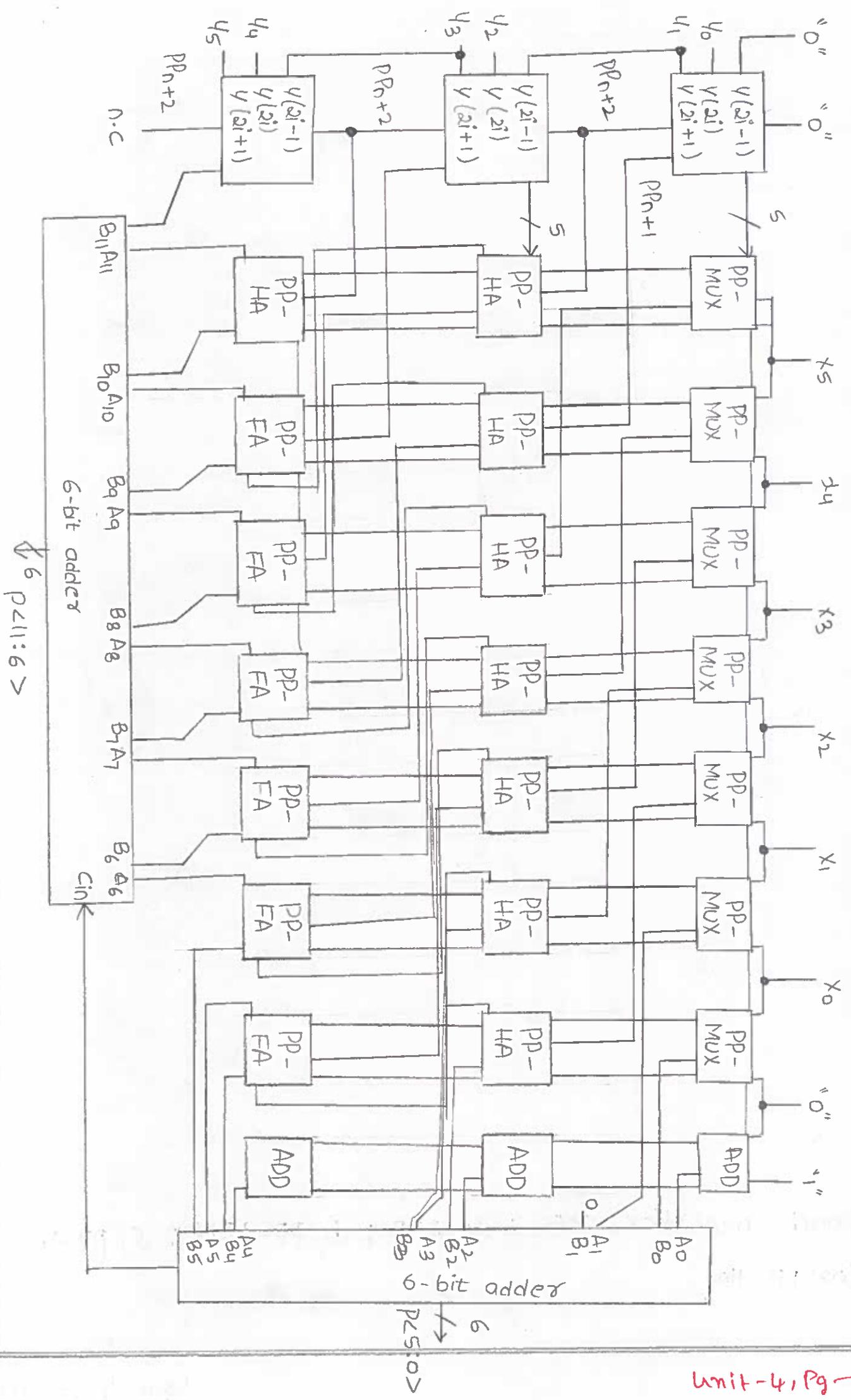
* The partial product MUX (PP-MUX) which generates the partial product. The feedback PMOS, P_F in this figure (on in the one of fig. are used to restore the high level to eliminate any DC current. This implementation permits fast operation and low-power operation.

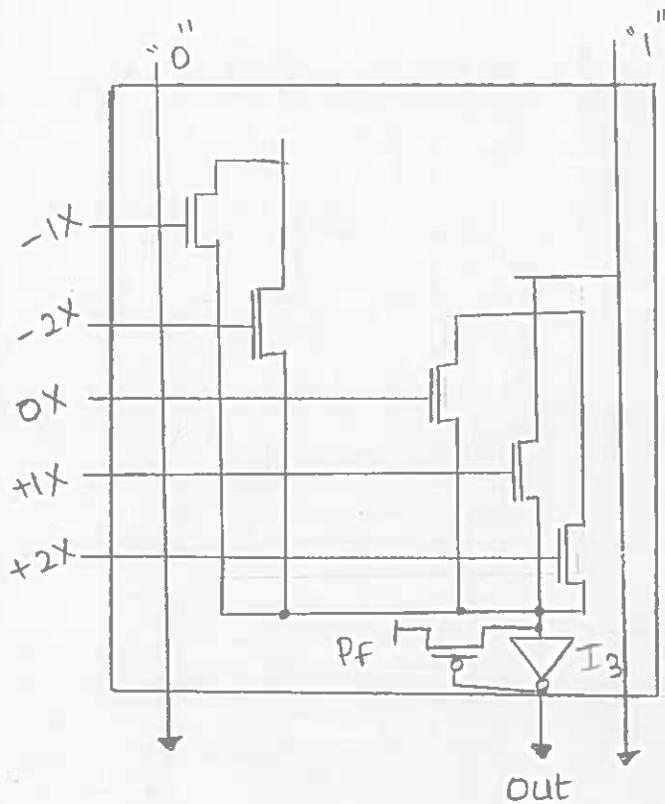
* The PP-FA (PP-HA) cells. They merge the PP-MUX circuit and a full-adder (half-adder) respectively. CPL-like adder can be utilized for fast operation and low power.

* The Booth encoder (BE). It generates the five control signals $0X$, $+1X$, $+2X$, $-1X$, and $-2X$. From the group of three bits of the multiplier Y . The additional circuits of the two bits $PP_{n+1,j}$ and $PP_{n+2,j}$ of the j th PP are also illustrated. F_j and F_{j+1} are the previous and the next flags, respectively. $PP_{n,j}$ is the sign bit of the j th PP. note that F_0 is 0.

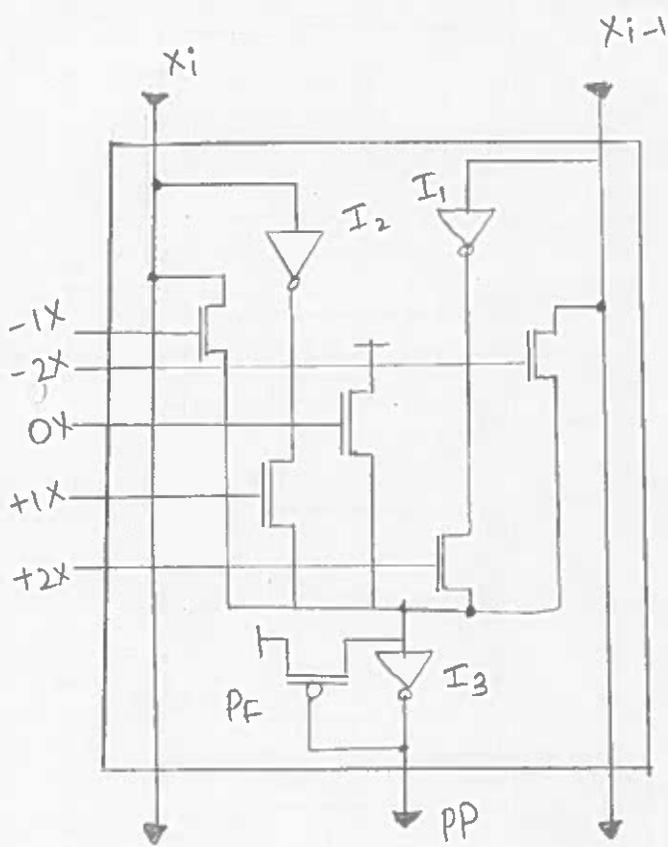
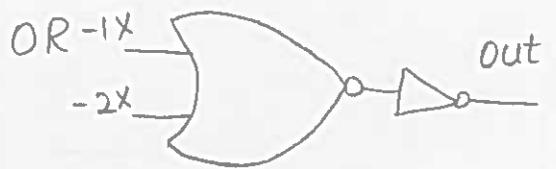
→ The booth multiplier exhibits a lot of unnecessary glitches. The main reason for glitches is due to the race condition between the multiplicand and the multiplier due to the booth encoder. The power dissipation associated with the glitches can be an important portion of the total power and hence it needs to be reduced by some techniques of signal synchronization.

A schematic of a 6-bit booth multiplier.

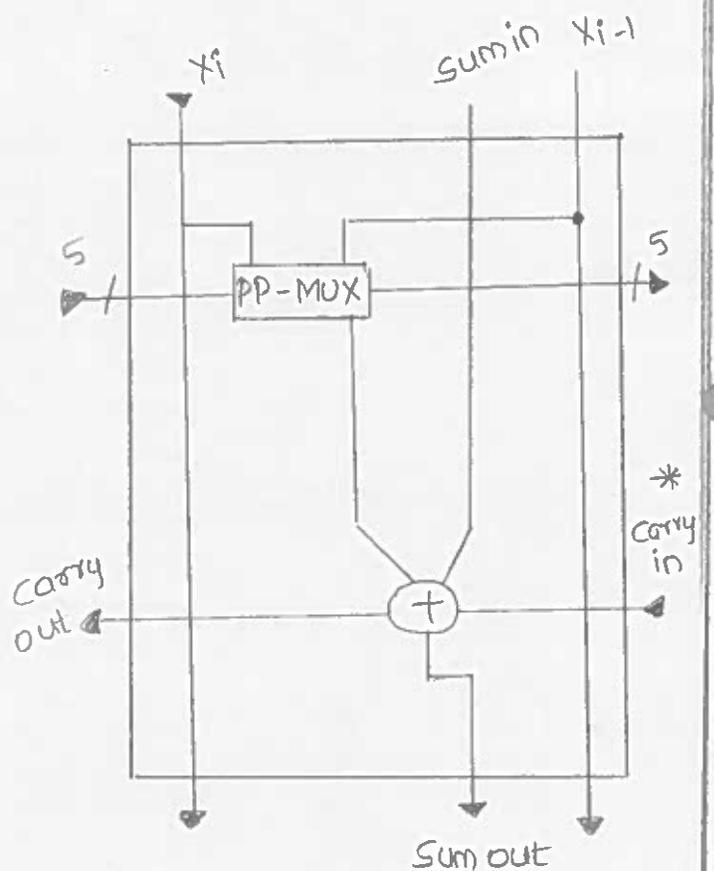




(a)



(b)



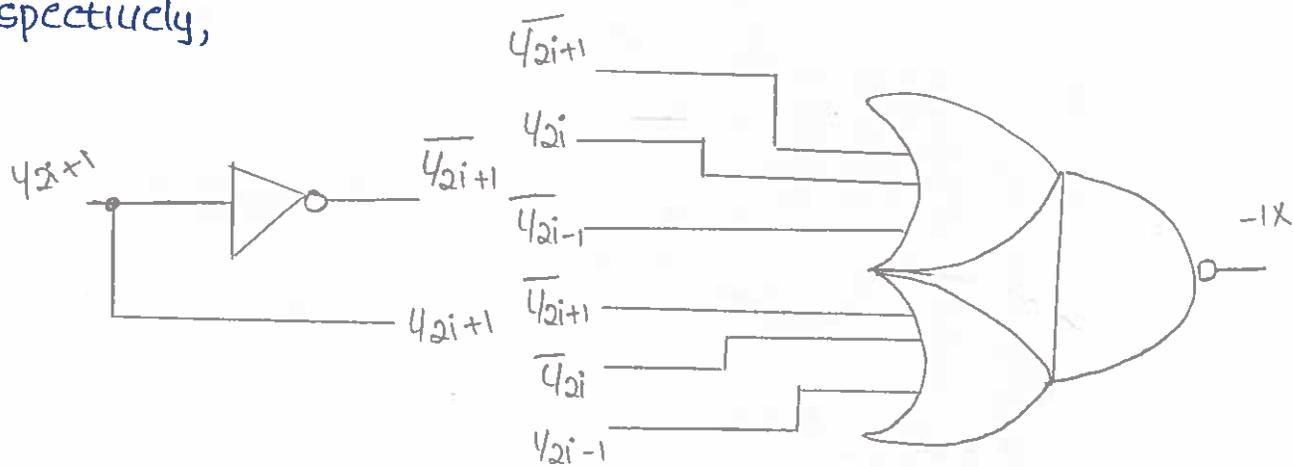
(*) not connected for pp-HA

(c)

Booth multiplier cells : (a), ADD ; (b), PP-MUX ; (c), PP-FA
(or) pp-HA.

* Wallace tree :-

By applying the booth algorithm, the number of partial products is halved. However for large multipliers, 32-bits and over, the number of the partial products is over 16-bit. In this case, the performance of the modified Booth algorithm is limited. One technique to improve the performance of these multipliers, is to adopt the wallace tree using 4-2 compressors. A 4-2 compressor accepts 4 numbers and a carry in, and sums them to produce 2 numbers and carry out. Fig 7.29(a) shows an example of such a tree on partial products of an 8x8 multiplier. Eight partial products are produced. Using 4-2 compressors, two levels of additions (stages) are needed. The final two summands are added using a fast 16-bit adder. Some zeros are added to the array. This example shows that the bits which are not used in the last 1st stage jump to the next one to be combined with the ones produced by the compressors. Fig. 7.29(b) shows the architecture of the 8x8 multiplier. For the first stage of the tree, two blocks, A and B, are required. The block A(B) of compressors group the first four partial products, respectively,



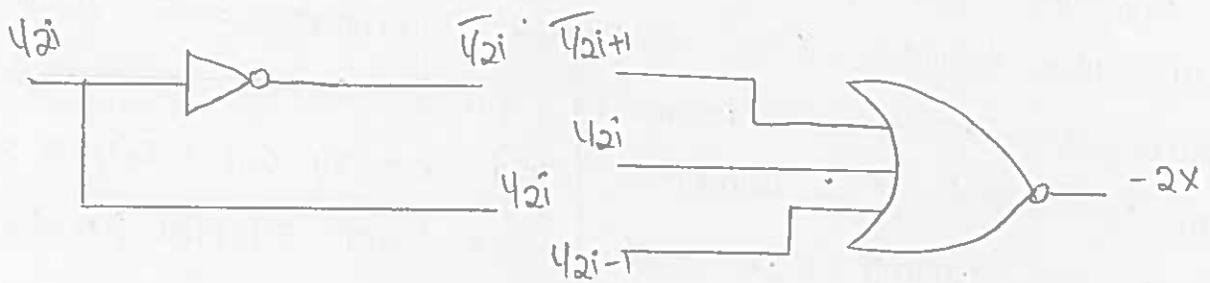
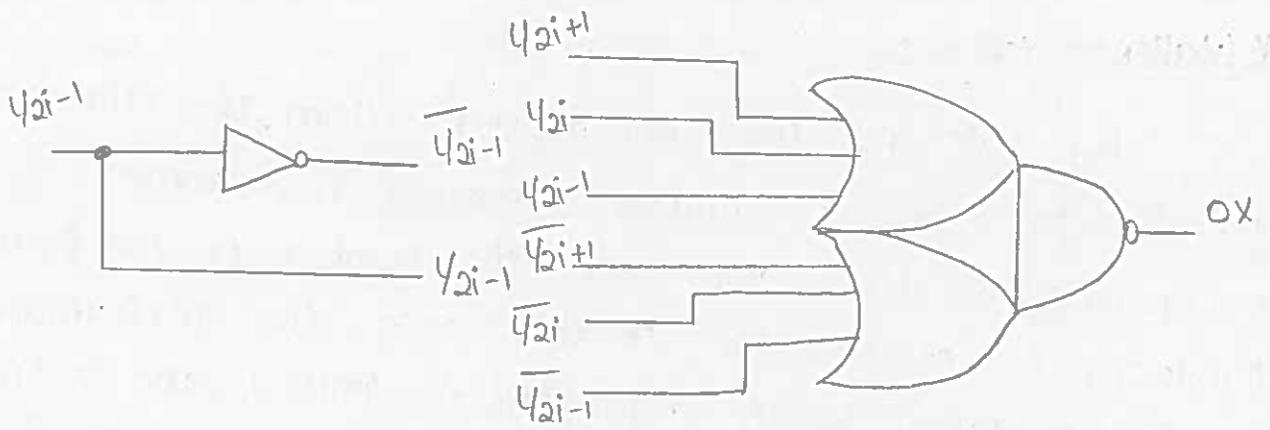


Fig:-7.28, logic schematic of the booth encoder including the sign extension logic.

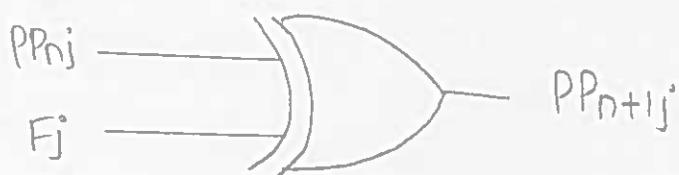
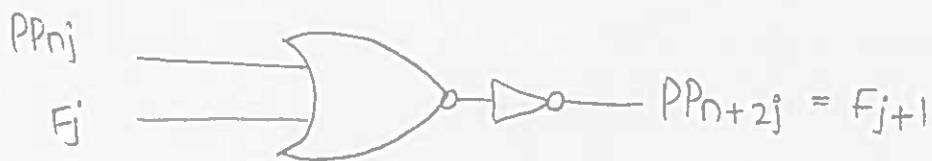
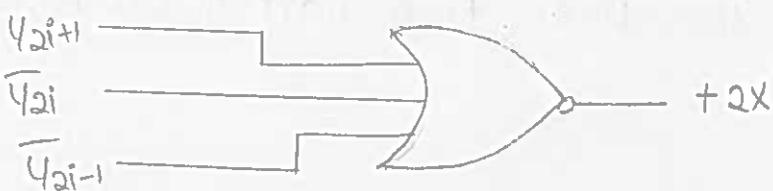
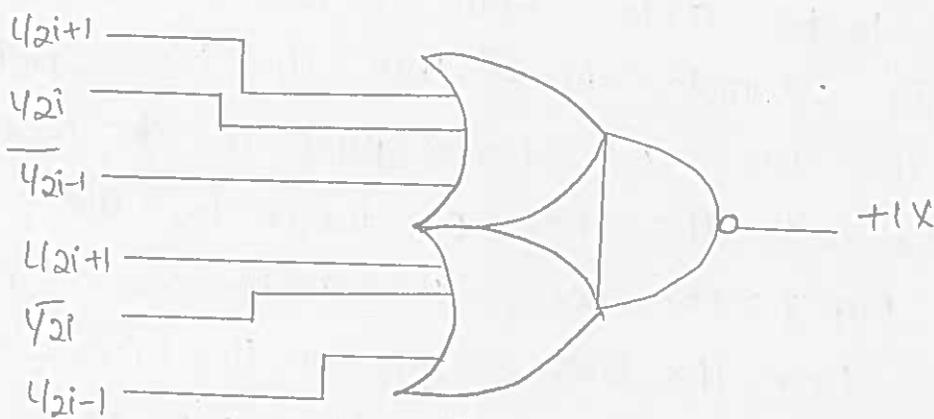
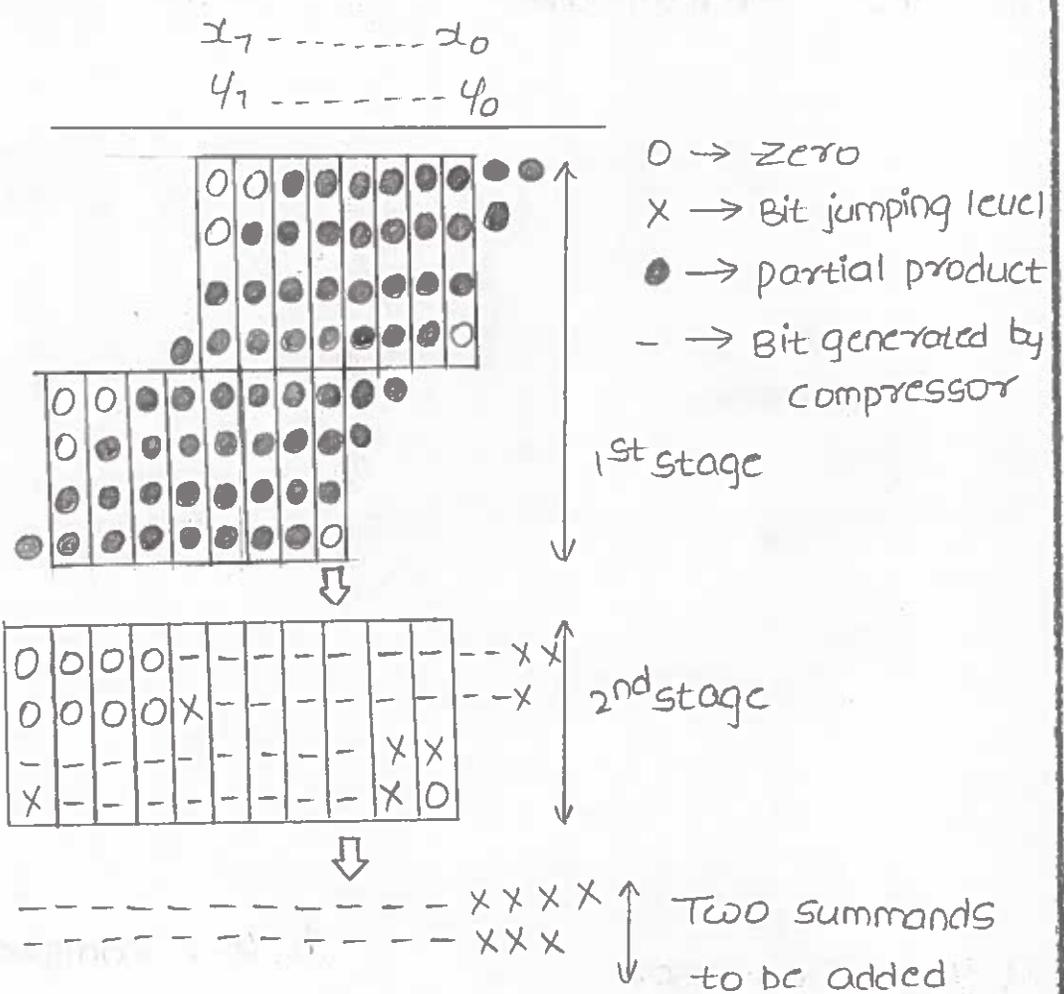
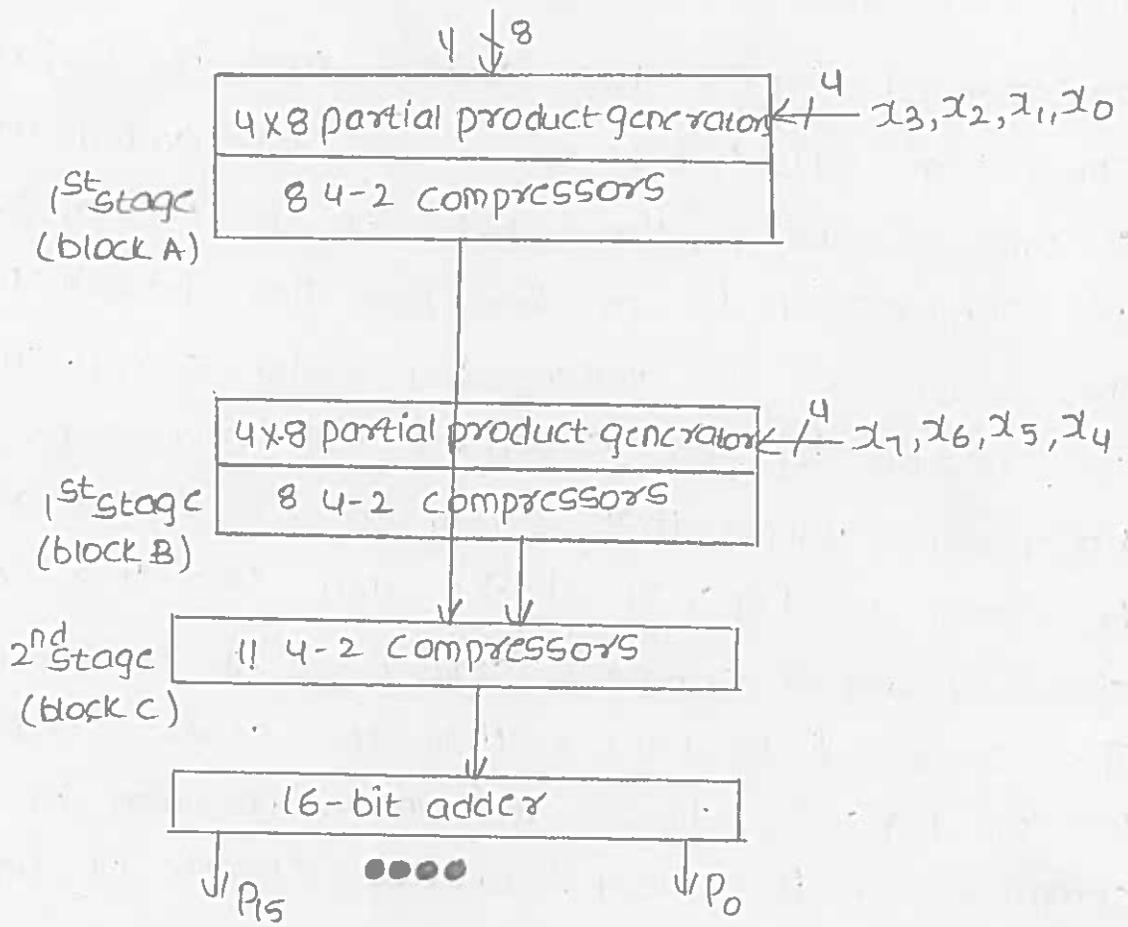


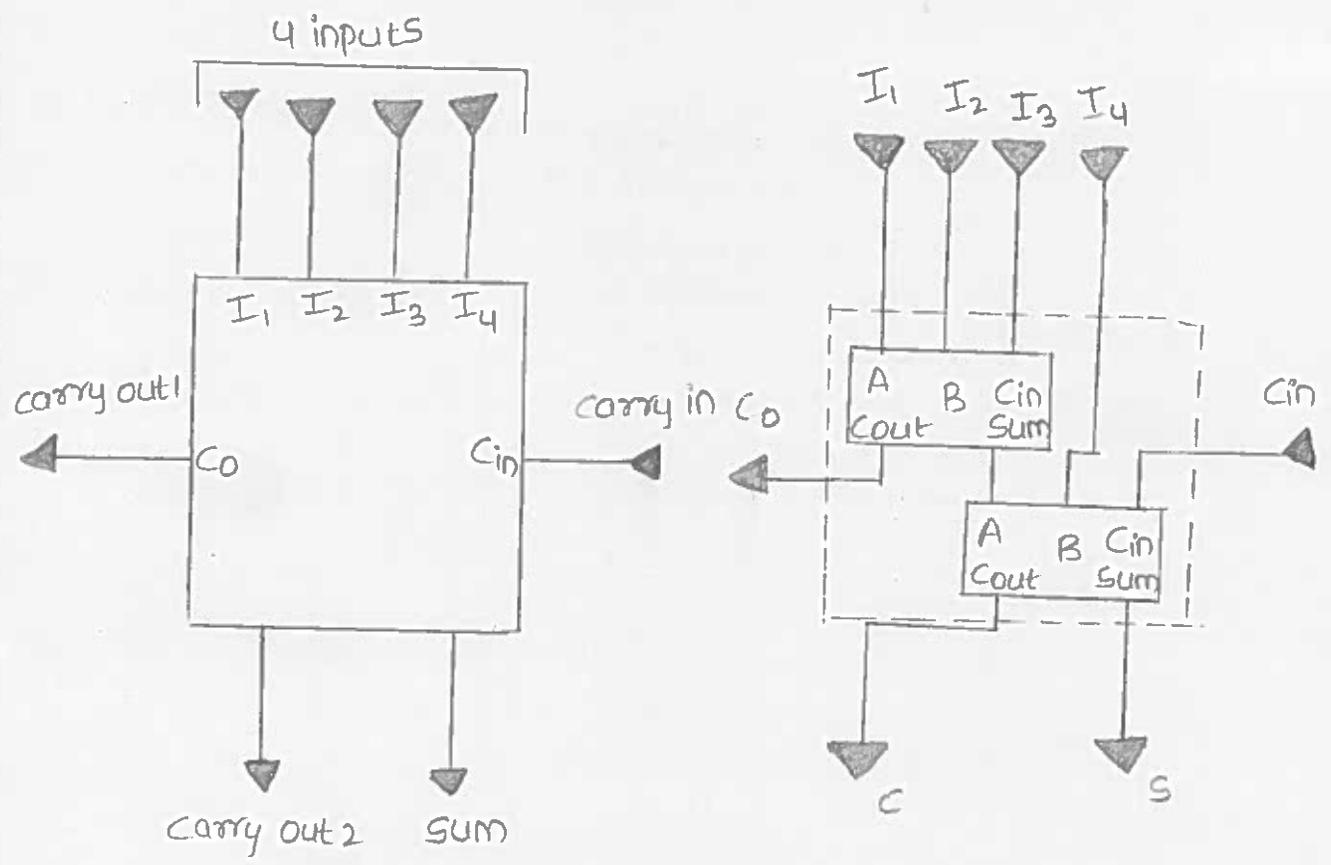
Fig 7.30 shows how the 4-2 compressor can be implemented by 2 full-adders (or) by custom static CMOS logic [9]. 4-bit I_1, \dots, I_4 are added to produce 2 sums S and C . Hence, 4-bit of the partial product are compressed to produce two new partial products. The compressor is implemented using carry-save adder construction, by two cascaded full-adders as shown in fig. 7.30(b). Notice that carry out 2 is never generated by carry in. Fig 7.31 shows that the 4-2 compressor circuit using a compact structure of multiplexers. This structure is faster than the static complementary version. Fig 7.32 shows the interconnection of the 4-2 compressors for block A of the example of fig 7.29 C_0 is connected.



a, construction of wallace's tree for 8x8 multiplier reduction of the 8 partial products with 4-2 compressors

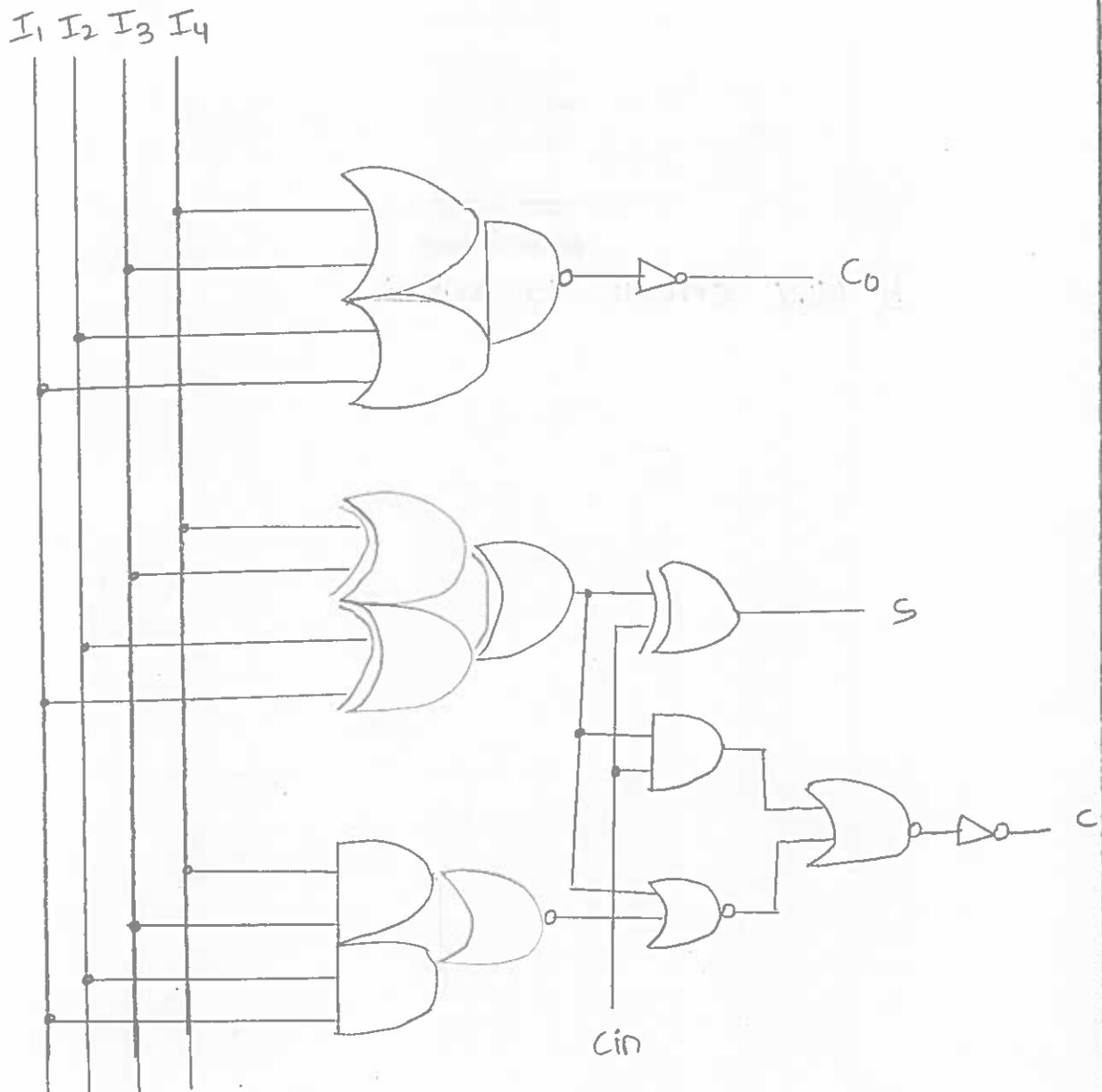


b, The architecture.

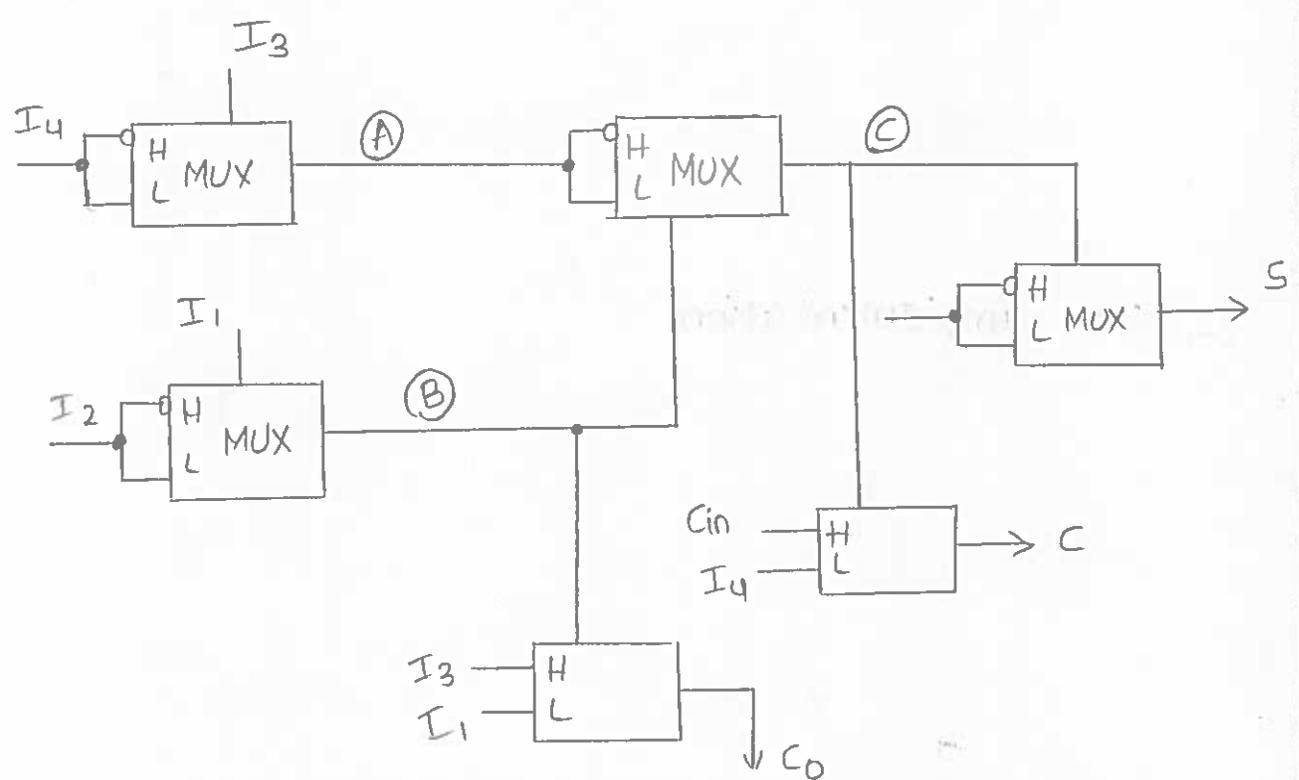


a, 4-2 compressor schematic

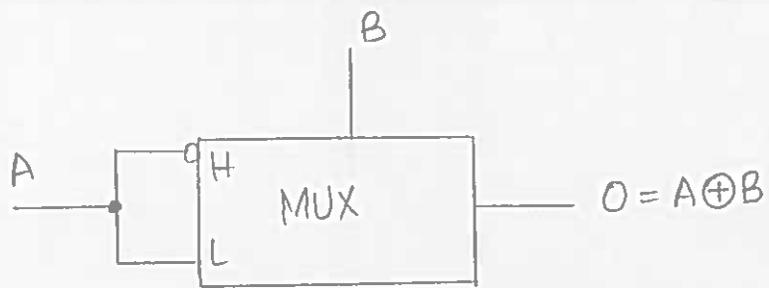
b, 4-2 compressor equivalent circuit by two full-adders



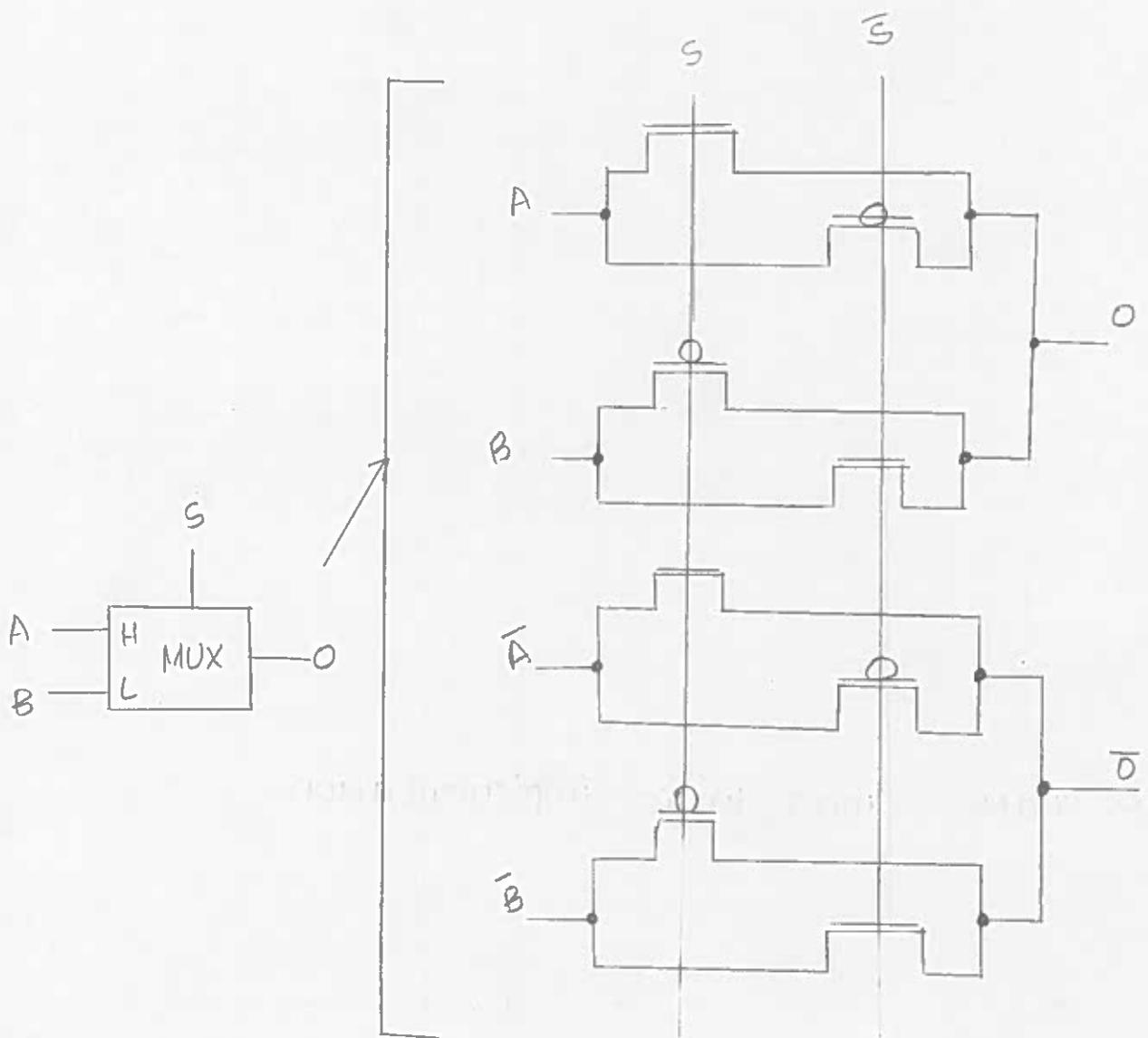
c, static CMOS logic implementation.



a, 4-2 compressor circuit using TG's



b, MUX circuit as XOR



c, MUX implementation

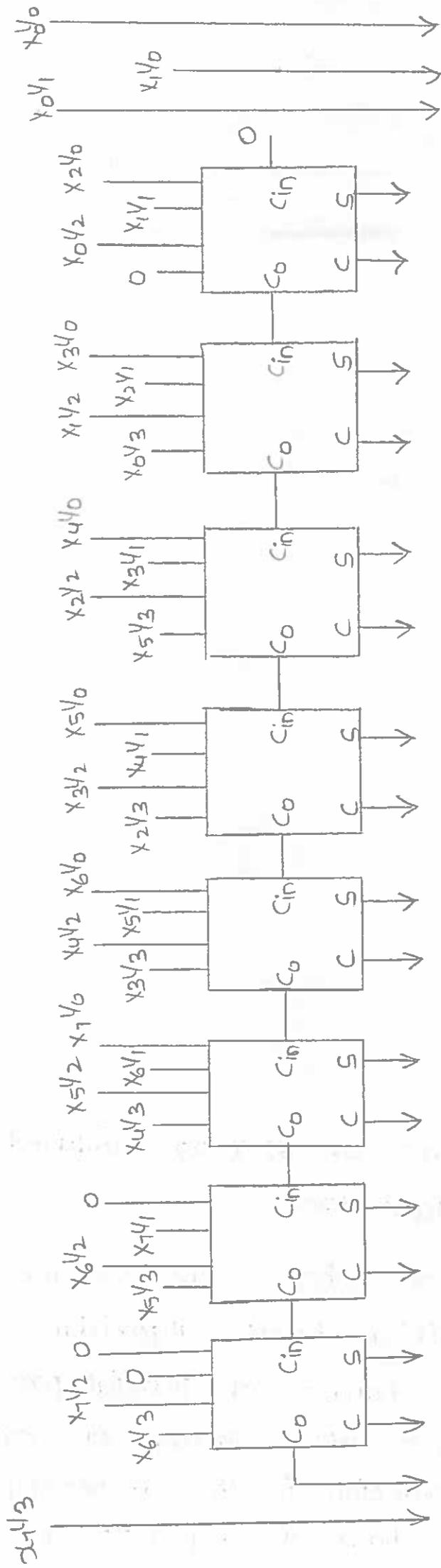


Fig. Interconnections of 4-2 compressors in block A of the 1st-stage in the 8x8 multipliers.

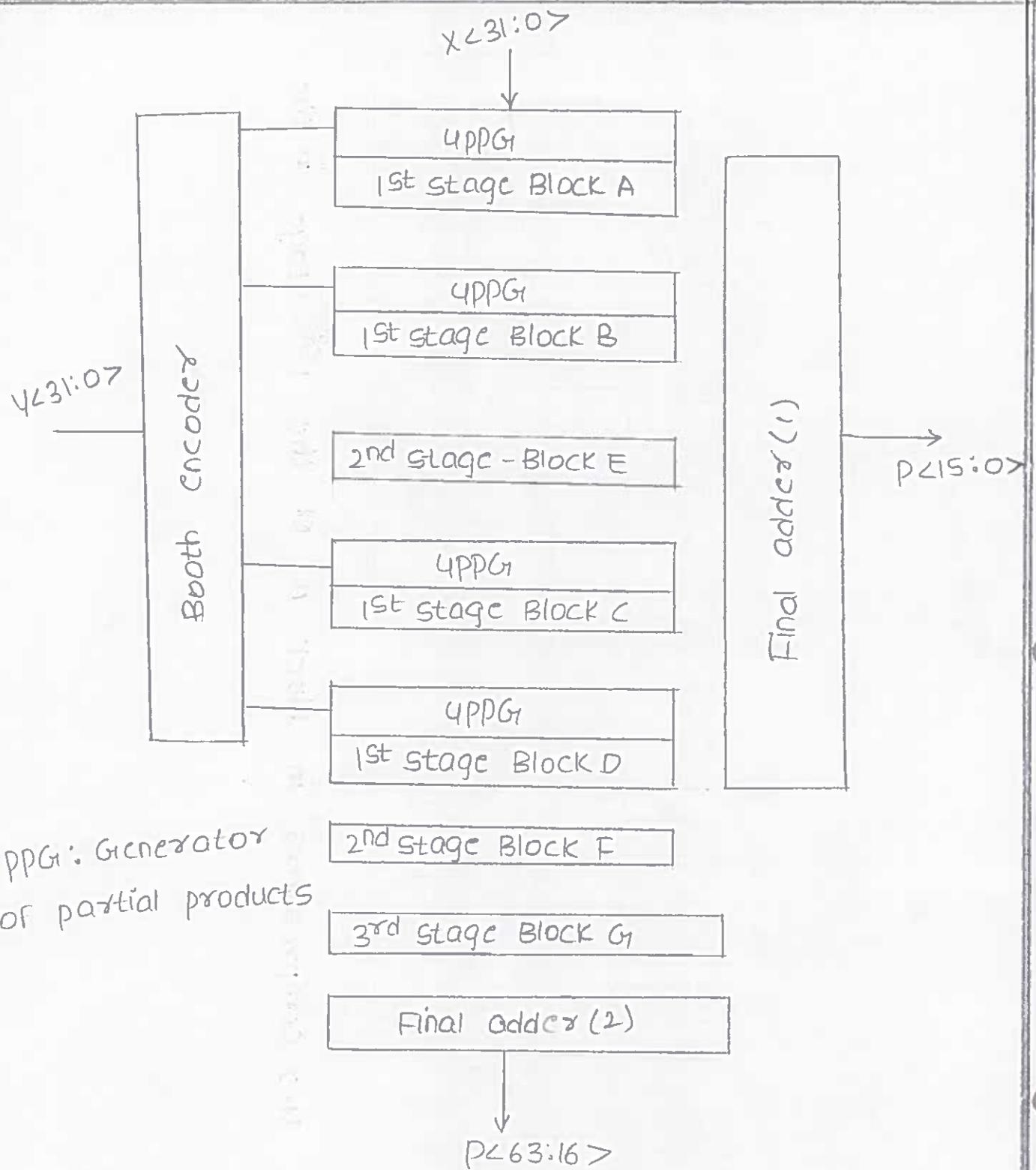


Fig:- The architecture of 32 X 32 modified booth multiplier with wallace tree.

To further enhance the wallace tree multiplier, the modified booth algorithm can be used to reduce the number of partial products by half in a carry-save adder array. One example of such combined construction is the architecture of the 32 X 32 multiplier shown in fig. 7.33. It consists of four functions :-

The booth encoder, the partial product's generator, the compressor blocks, and the final 64-bit adder. The wallace tree is constructed with 3 stages (levels). The first stage has 4 blocks (A to D), with each block summing up 4 partial products among 16. The second stage sums up the 8 new generated partial products from the first stage. Hence, two blocks are needed, E and F. Finally block G of the third stage of the tree generates two other new partial products to the final adder. This architecture exhibits some irregularities in the layout since it has a complicated interconnection scheme. Hence, the interconnection scheme. Hence, the interconnection wires affect the speed and power dissipation of the adder.

Multiplier's Comparison :-

The basic array multipliers, like Baugh-wooley scheme, consume low-power and have relatively good performance. However, their use can be limited to process operands with less than 16-bit. For operands of 16-bit and over, the modified booth algorithm reduces the partial product's numbers by half. Therefore, the speed of the multiplier is reduced. Its power dissipation is comparable to the Baugh-wooley multiplier is reduced. Its power dissipation is comparable to the Baugh wooley multiplier due to the circuitry overhead in the booth algorithm. However, circuit techniques can cause this multiplier to have low-power characteristics. The fastest multipliers adopt the wallace tree with modified booth encoding. A wallace tree would lead in general, to larger power dissipation and area, due to the interconnect wires.

Hence, it is not recommended for low-power consumption applications. Dynamic multipliers are not discussed in this section since they introduce problems of control and timing. Hence extra area and power dissipation are added to the design.

Data path:-

A ULSI chip can be partitioned in two parts the data path (or execution unit) and the control unit. Data paths are often used in digital signal processors, microprocessors and application specific ICs (ASICs). The data path consists of a combination of an Arithmetic Logic Unit (ALU), a shifter, a file register, I/O ports, a multiplier, an adder, a magnitude comparator, and data busses, etc. It performs many operations on the data in the register file, to which the results are sent back. The data busses are the communication means for the data transfer between the ALU, shifter, and file register, etc. These busses have a heavy load (few pF). In CMOS design, dynamic techniques are used to allow fast operation. One way to reduce the power dissipation, due to the precharging transistors, is to use static busses.