Combinational Arithmetic Circuits

- **Addition:**
  - Half Adder (HA).
  - Full Adder (FA).
  - Carry Ripple Adders.
  - Carry Look-Ahead Adders.
- **Subtraction:**
  - Half Subtractor.
  - Full Subtractor.
  - Borrow Ripple Subtractors.
  - Subtraction using adders.
- **Multiplication:**
  - Combinational Array Multipliers.

Half Adder

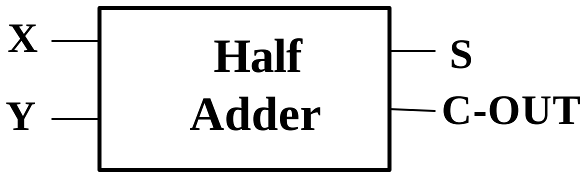
- Adding two single-bit binary values, X, Y produces a sum S bit and a carry out C-out bit.
- This operation is called half addition and the circuit to realize it is called a half adder.

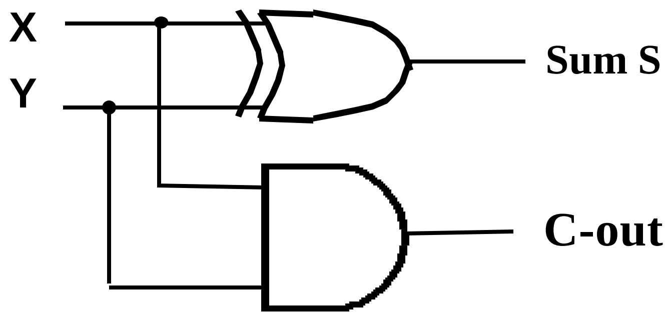
Half Adder Truth Table

Inputs		Outputs	
X	Y	S	C-out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S(X,Y) = **S** (1,2)
S = X’Y + XY’
S = X **Å** Y

C-out(x, y, C-in) = **S** (3)
C-out = XY





Full Adder

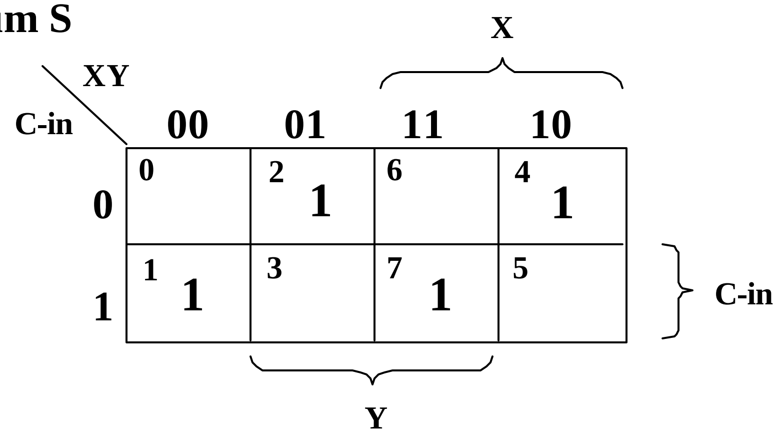
- Adding two single-bit binary values, X, Y with a carry input bit C-in produces a sum bit S and a carry out C-out bit.

Full Adder Truth Table

Inputs			Outputs	
X	Y	C-in	S	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

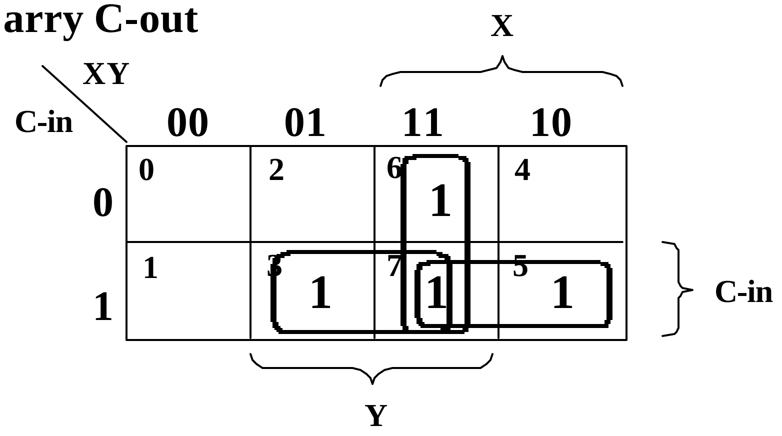
S(X,Y, C-in) = **S** (1,2,4,7)
C-out(x, y, C-in) = **S** (3,5,6,7)

Sum S



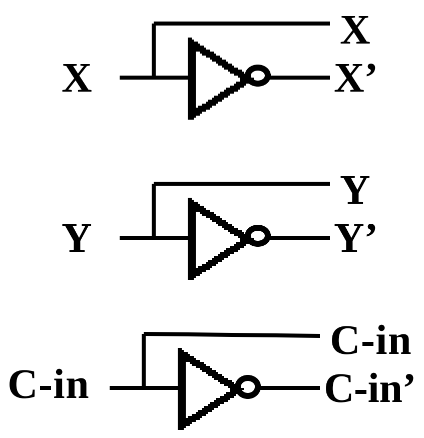
S = X'Y'(C-in) + XY'(C-in)' + XY'(C-in)' + XY(C-in)
S = X ⊕ Y ⊕ (C-in)

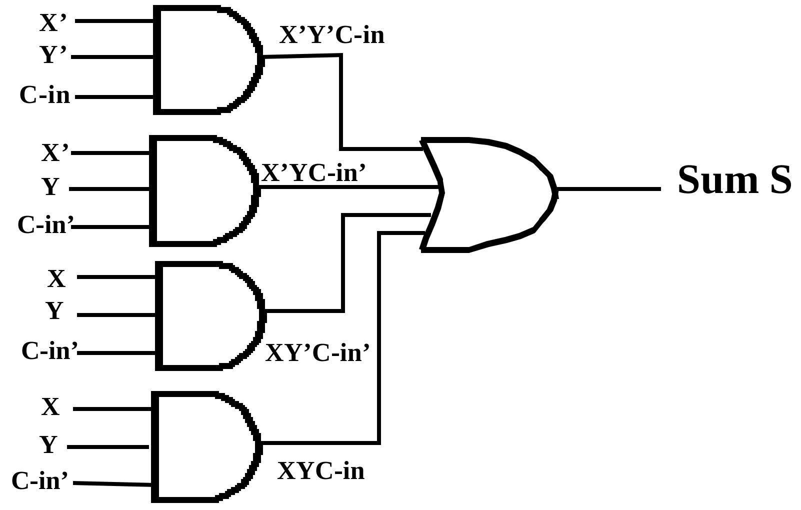
Carry C-out

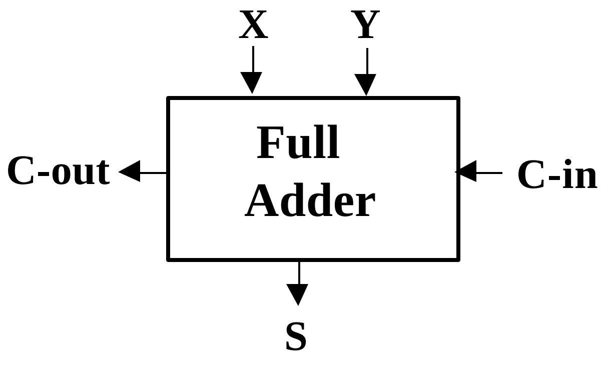


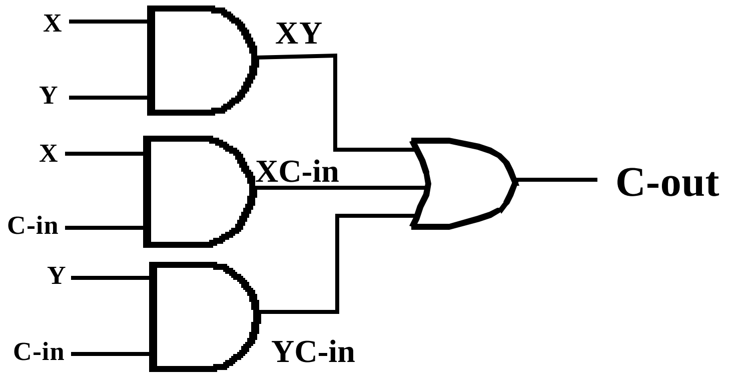
C-out = XY + X(C-in) + Y(C-in)

Full Adder Circuit Using AND-OR

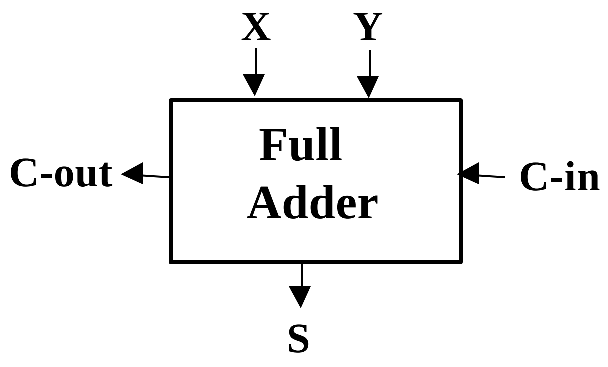


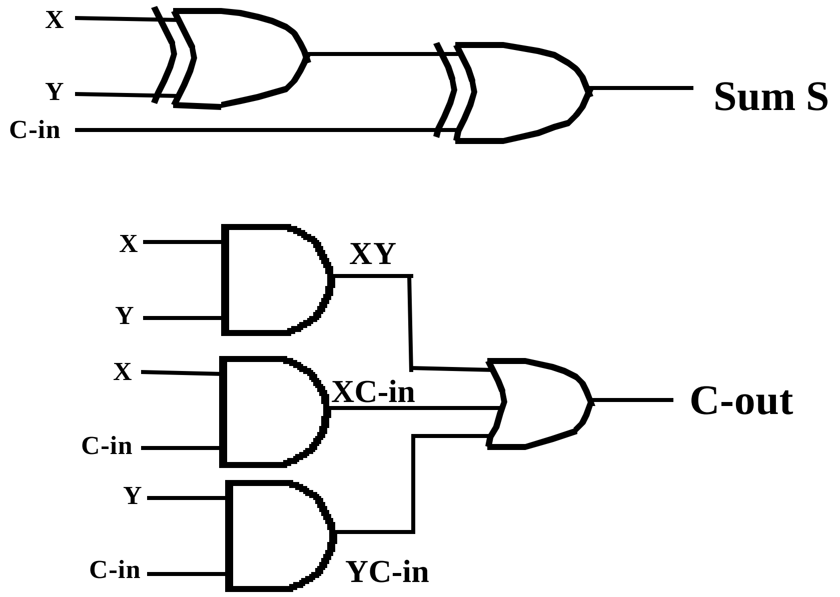






Full Adder Circuit Using XOR





n-bit Carry Ripple Adders

- **An n-bit adder used to add two n-bit binary numbers can built by connecting in series n full adders.**
  - **Each full adder represents a bit position j (from 0 to n-1).**
  - **Each carry out C-out from a full adder at position j is connected to the carry in C-in of the full adder at the higher position j+1.**
- **The output of a full adder at position j is given by:**

$$\mathbf{S_j = X_j \text{ Å } Y_j \text{ Å } C_j}$$

$$\mathbf{C_{j+1} = X_j \,.\, Y_j + X_j \,.\, C_j + Y \,.\, C_j}$$

- **In the expression of the sum C_j must be generated by the full adder at the lower position j-1.**
- **The propagation delay in each full adder to produce the carry is equal to two gate delays = 2 D**
- **Since the generation of the sum requires the propagation of the carry from the lowest position to the highest position , the total propagation delay of the adder is approximately:**

Total Propagation delay = 2 nD

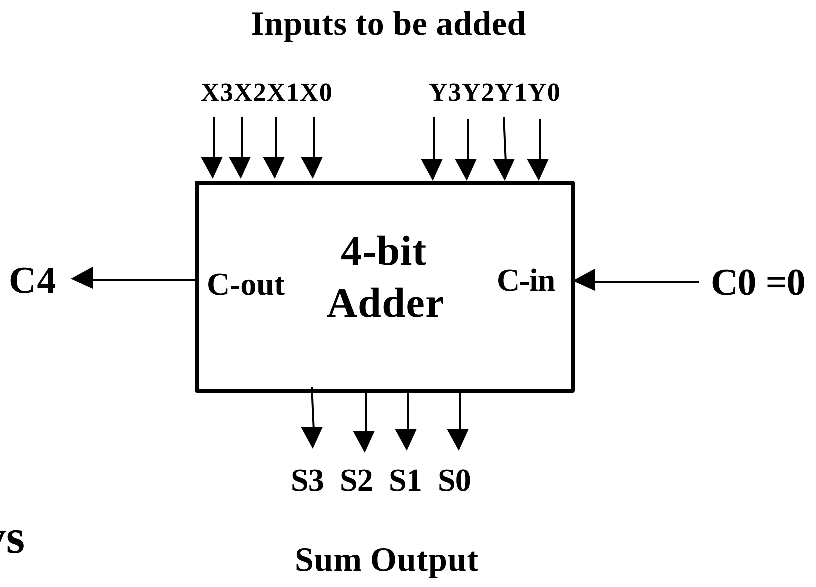
4-bit Carry Ripple Adder

Adds two 4-bit numbers:

X = X3 X2 X1 X0

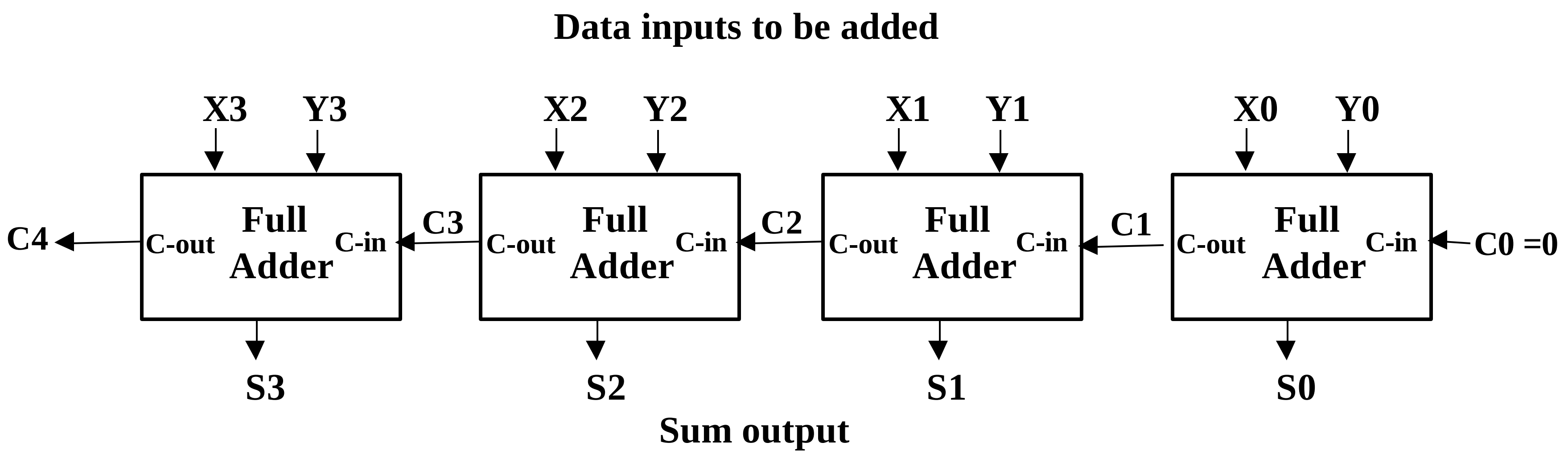
Y = Y3 Y2 Y1 Y0

producing the sum S = S3 S2 S1 S0 , C-out = C4 from the most significant position j=3



Total Propagation delay = 2 nD = 8D

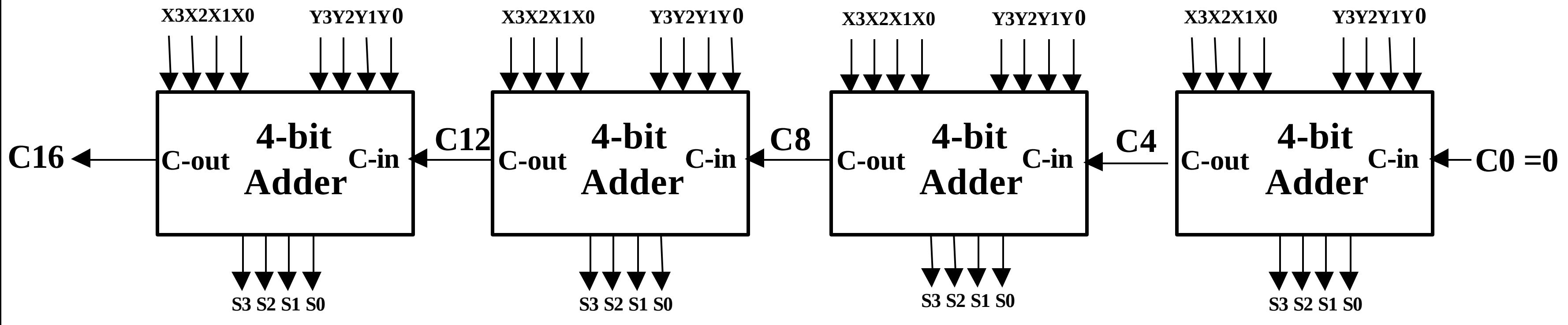
or 8 gate delays



Larger Adders

- **Example: 16-bit adder using 4, 4-bit adders**
- **Adds two 16-bit inputs X (bits X0 to X15), Y (bits Y0 to Y15) producing a 16-bit Sum S (bits S0 to S15) and a carry out C16 from most significant position.**

Data inputs to be added X (X0 to X15) , Y (Y0-Y15)



Sum output S (S0 to S15)

Propagation delay for 16-bit adder = 4 x propagation delay of 4-bit adder

= 4 x 2 nD = 4 x 8D = 32 D

or 32 gate delays

Carry Look-Ahead Adders

- The disadvantage of the ripple carry adder is that the propagation delay of adder (2 nD) increases as the size of the adder, n is increased due to the carry ripple through all the full adders.
- Carry look-ahead adders use a different method to create the needed carry bits for each full adder with a lower constant delay equal to three gate delays.
- The carry out C-out from the full adder at position i or C_{j+1} is given by:

$$\text{C-out} = C_{i+1} = X_i \, . \, Y_i + (X_i + Y_i) \, . \, C_i$$

- By defining:
  - $G_i = X_i \, . \, Y_i$ as the carry generate function for position i (one gate delay)
    (If $G_i = 1$ C_{i+1} will be generated regardless of the value C_i)
  - $P_i = X_i + Y_i$ as the carry propagate function for position i (one gate delay)
    (If $P_i = 1$ C_i will be propagated to C_{i+1})
- By using the carry generate function G_i and carry propagate function P_i, then C_{i+1} can be written as:

$$\text{C-out} = C_{i+1} = G_i + P_i \, . \, C_i$$

- To eliminate carry ripple the term C_i is recursively expanded and by multiplying out, we obtain a 2-level AND-OR expression for each C_{i+1}

Carry Look-Ahead Adders

- For a 4-bit carry look-ahead adder the expanded expressions for all carry bits are given by:

  $C_1 = G_0 + P_0.C_0$

  $C_2 = G_1 + P_1.C_1 = G_1 + P1.G_0 + P_1.P_0.C_0$

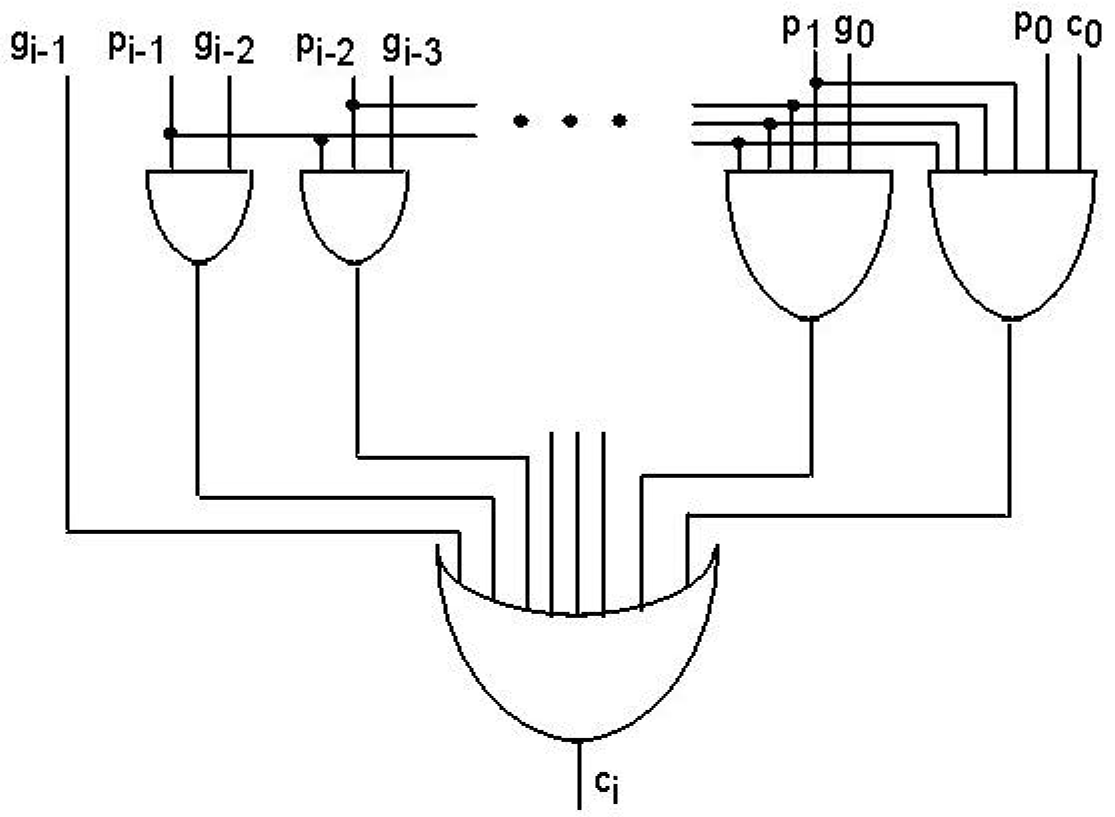
  $C_3 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_0$

  $C_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3 . P_2.P_1.G_0 + P_3.P_2.P_1.P_0.C_0$

  where $G_i = X_i . Y_i$ $P_i = X_i + Y_i$

- The additional circuits needed to realize the expressions are usually referred to as the carry look-ahead logic.
- Using carry-ahead logic all carry bits are available after three gate delays regardless of the size of the adder.

Carry Look-Ahead Circuit



$$C_i = G_{i-1} + P_{i-1} . G_{i-2} + \ldots + P_{i-1} . P_{i-2} . \ldots P_1 . G_0 + P_{i-1} . P_{i-2} . \ldots P_0 . C_0$$

Binary Arithmetic Operations Subtraction

- Two binary numbers are subtracted by subtracting each pair of bits together with borrowing, where needed.
- Subtraction Example:

```
                             0  0  1  1  1  1  1  0  0   Borrow
X       229                     1  1  1  0  0  1  0  1
Y  -     46          -          0  0  1  0  1  1  1  0
     ---------            ----------------------------
        183                     1  0  1  1  0  1  1  1
```

Half Subtractor

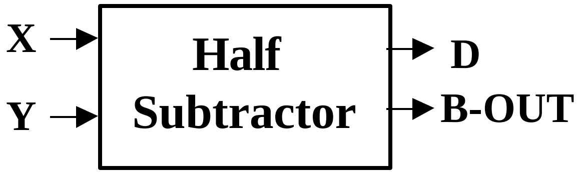
- Subtracting a single-bit binary value Y from anther X (I.e. X -Y) produces a difference bit D and a borrow out bit B-out.
- This operation is called half subtraction and the circuit to realize it is called a half subtractor.

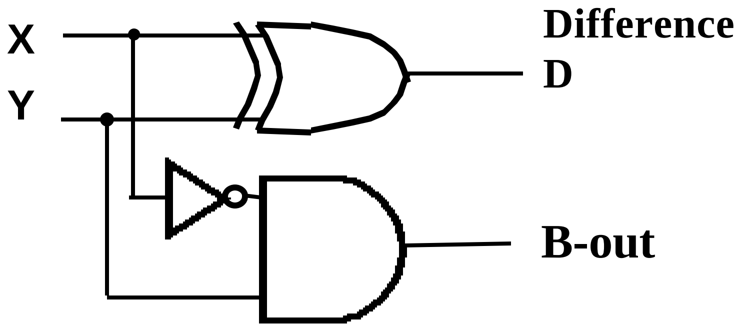
Half Subtractor Truth Table

Inputs		Outputs	
X	Y	D	B-out
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

D(X,Y) = **S** (1,2)
D = X'Y + XY'
D = X **Å** Y

B-out(x, y, C-in) = **S** (1)
B-out = X'Y





Full Subtractor

- Subtracting two single-bit binary values, Y, B-in from a single-bit value X produces a difference bit D and a borrow out B-out bit. This is called full subtraction.

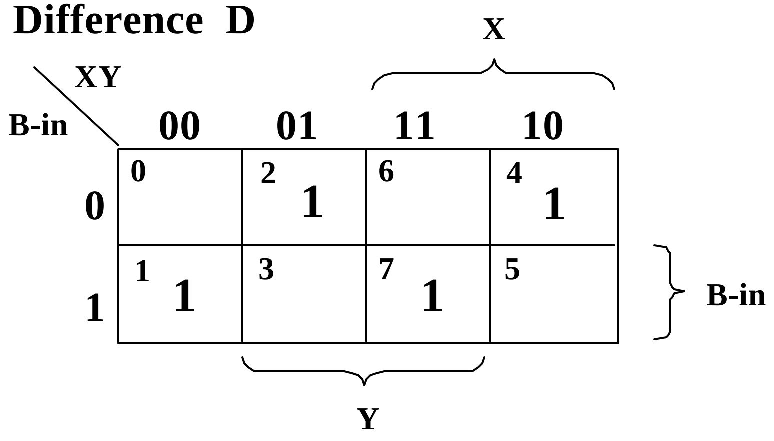
Full Subtractor Truth Table

Inputs			Outputs	
X	Y	B-in	D	B-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

S(X,Y, C-in) = **S** (1,2,4,7)

C-out(x, y, C-in) = **S** (1,2,3,7)

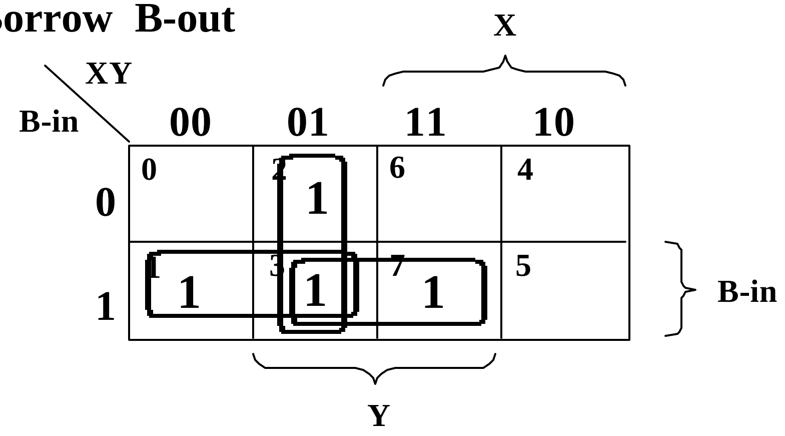
Difference D



S = X'Y'(B-in) + XY'(B-in)' + XY'(B-in)' + XY(B-in)

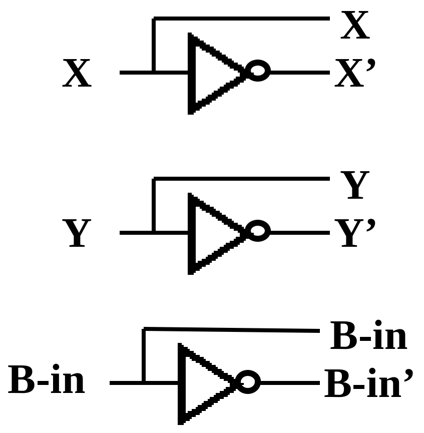
S = X ⊕ Y ⊕ (C-in)

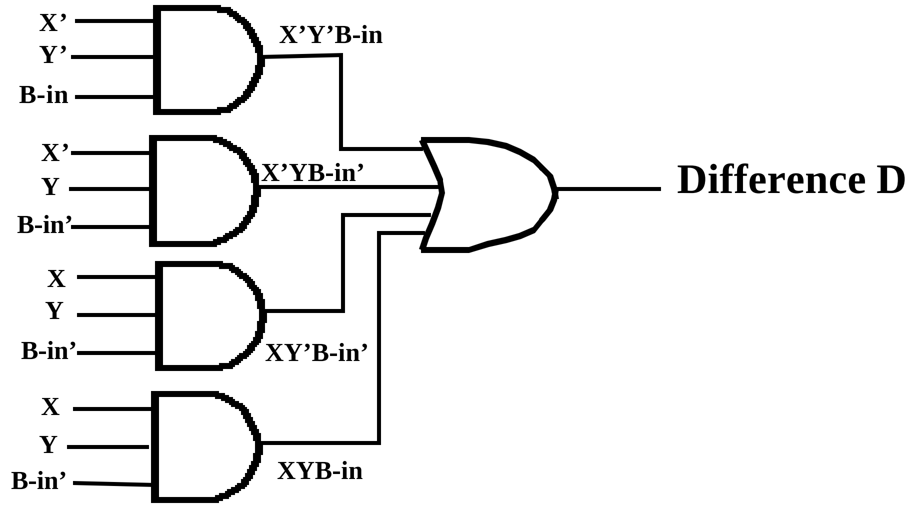
Borrow B-out

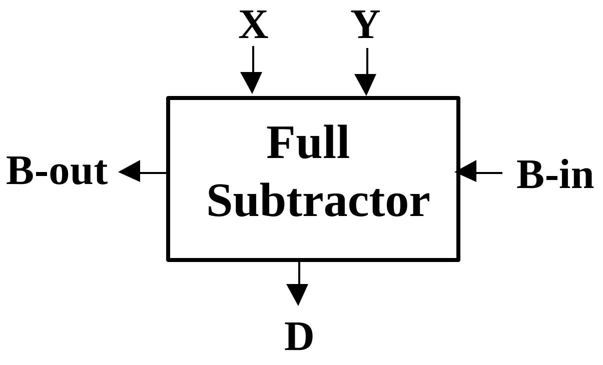


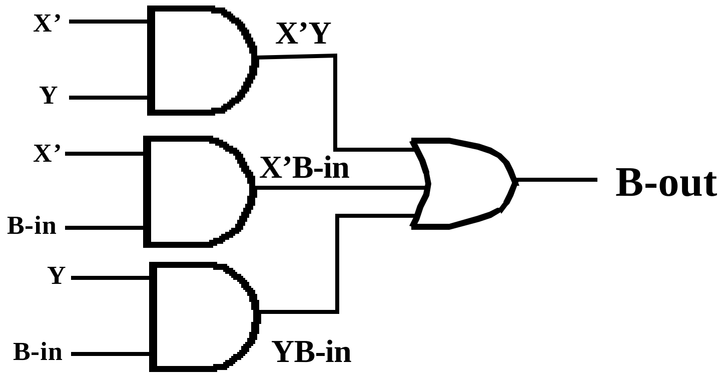
B-out = X'Y + X'(B-in) + Y(B-in)

Full Subtractor Circuit Using AND-OR

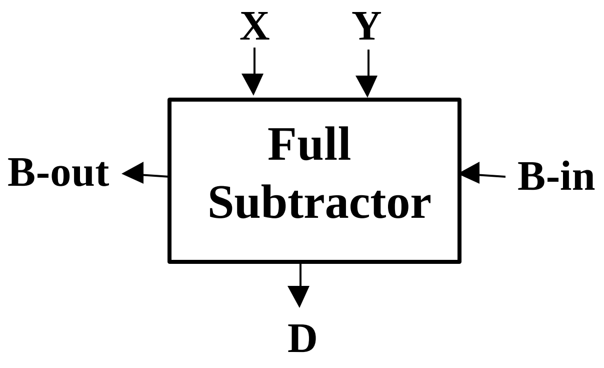


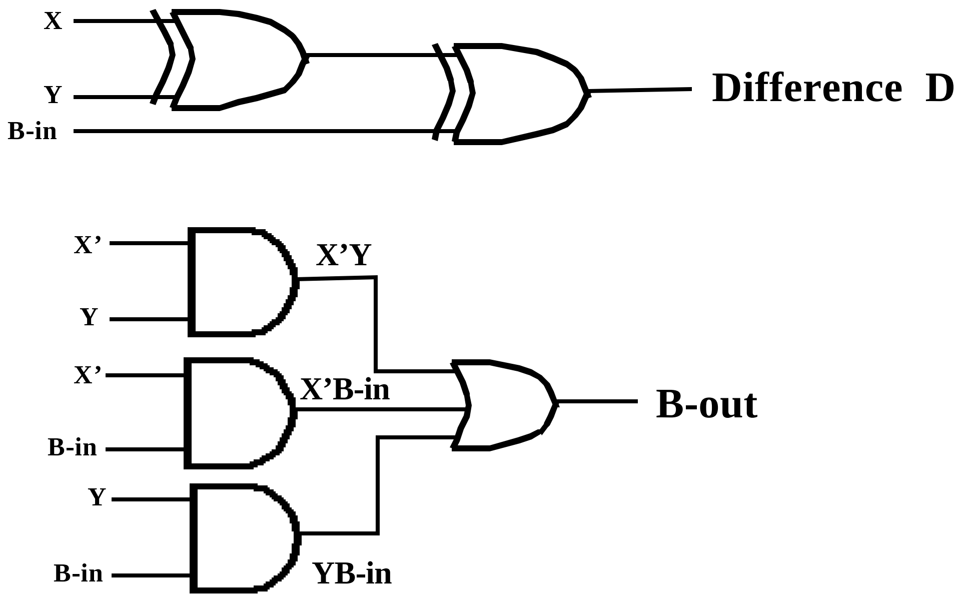






Full Subtractor Circuit Using XOR



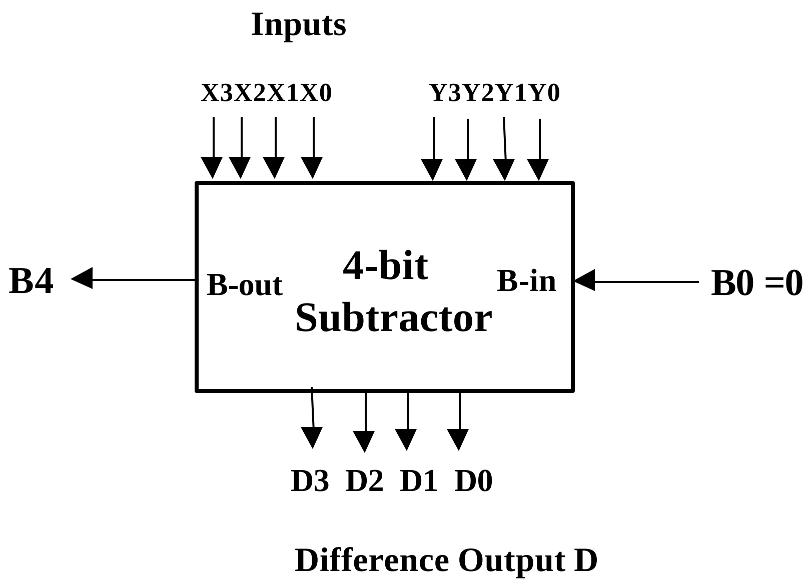


n-bit Subtractors

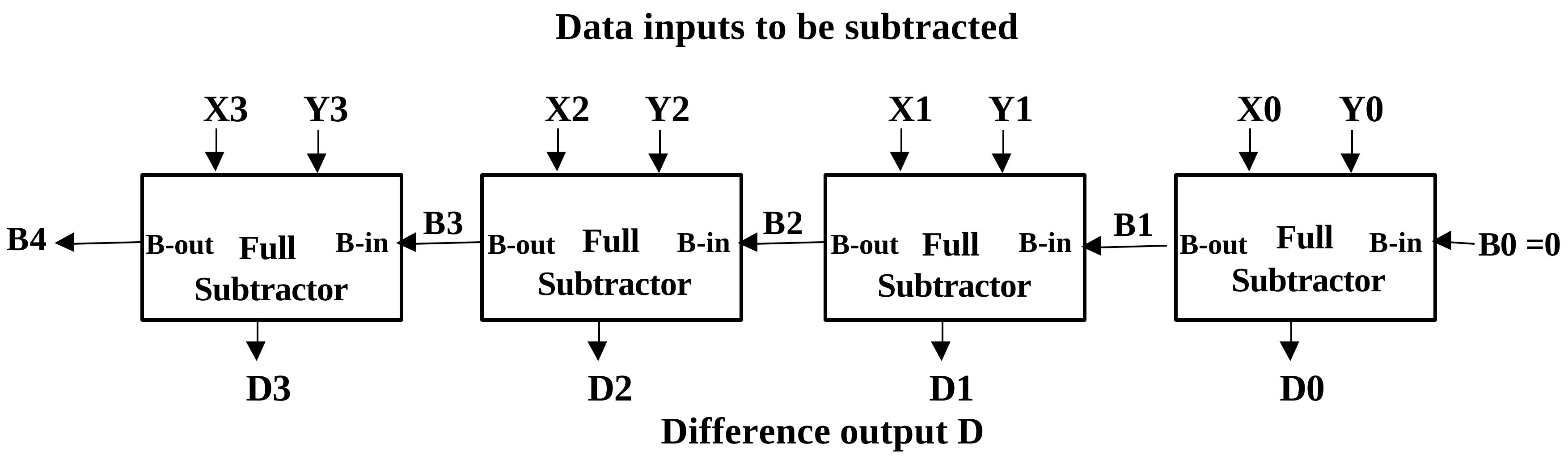
An n-bit subtracor used to subtract an n-bit number Y from another n-bit number X (i.e X-Y) can be built in one of two ways:

- **By using n full subtractors and connecting them in series, creating a borrow ripple subtractor:**
  - **Each borrow out B-out from a full subtractor at position j is connected to the borrow in B-in of the full subtracor at the higher position j+1.**
- **By using an n-bit adder and n inverters:**
  - **Find two's complement of Y by:**
    - **Inverting all the bits of Y using the n inverters.**
    - **Adding 1 by setting the carry in of the least significant position to 1**
  - **The original subtraction (X - Y) now becomes an addition of X to two's complement of Y using the n-bit adder.**

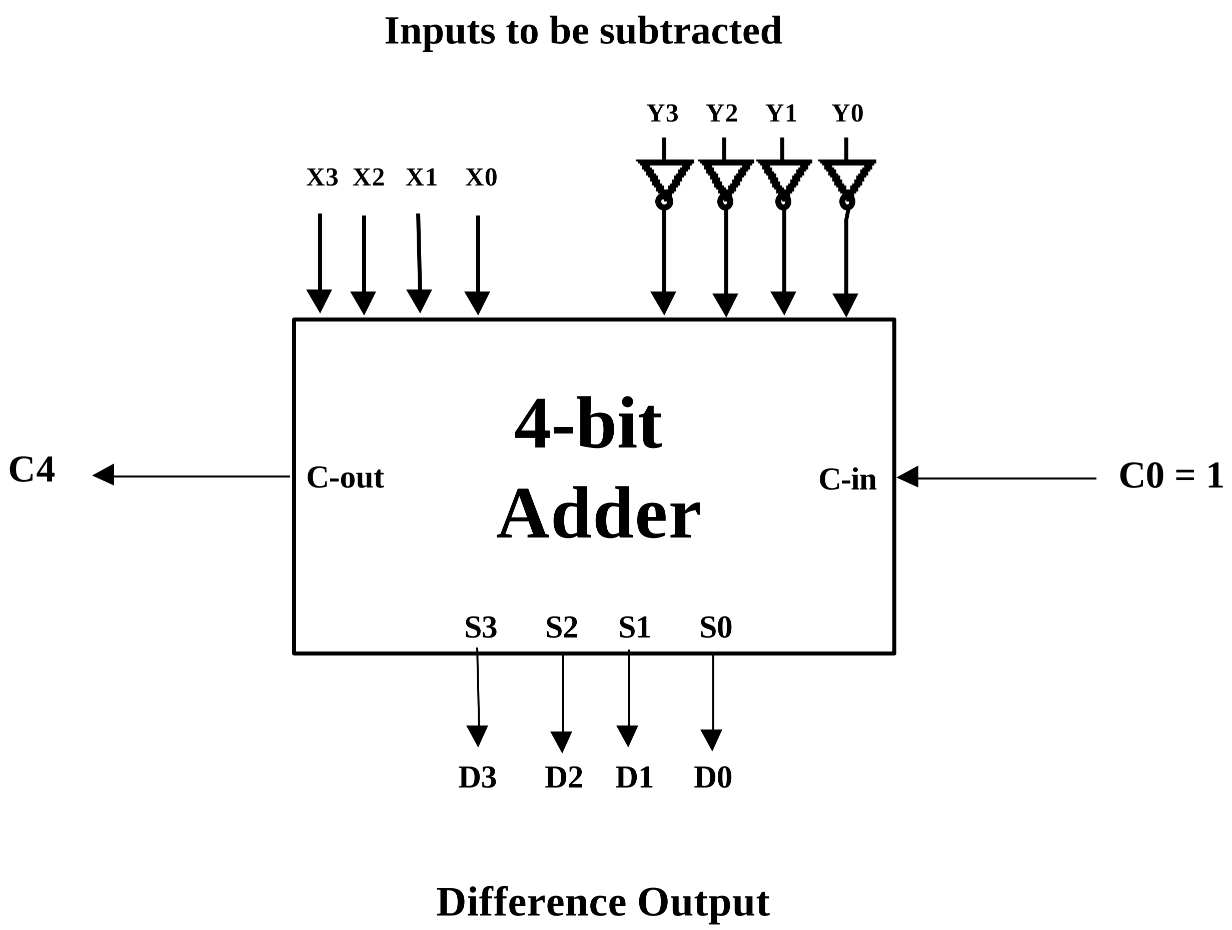
4-bit Borrow Ripple Subtractor



Subtracts two 4-bit numbers:
Y = Y3 Y2 Y1 Y0 from
X = X3 X2 X1 X0
Y = Y3 Y2 Y1 Y0
producing the difference D = D3 D2 D1 D0 ,
B-out = B4 from the most significant
position j=3



4-bit Subtractor Using 4-bit Adder



Binary Multiplication

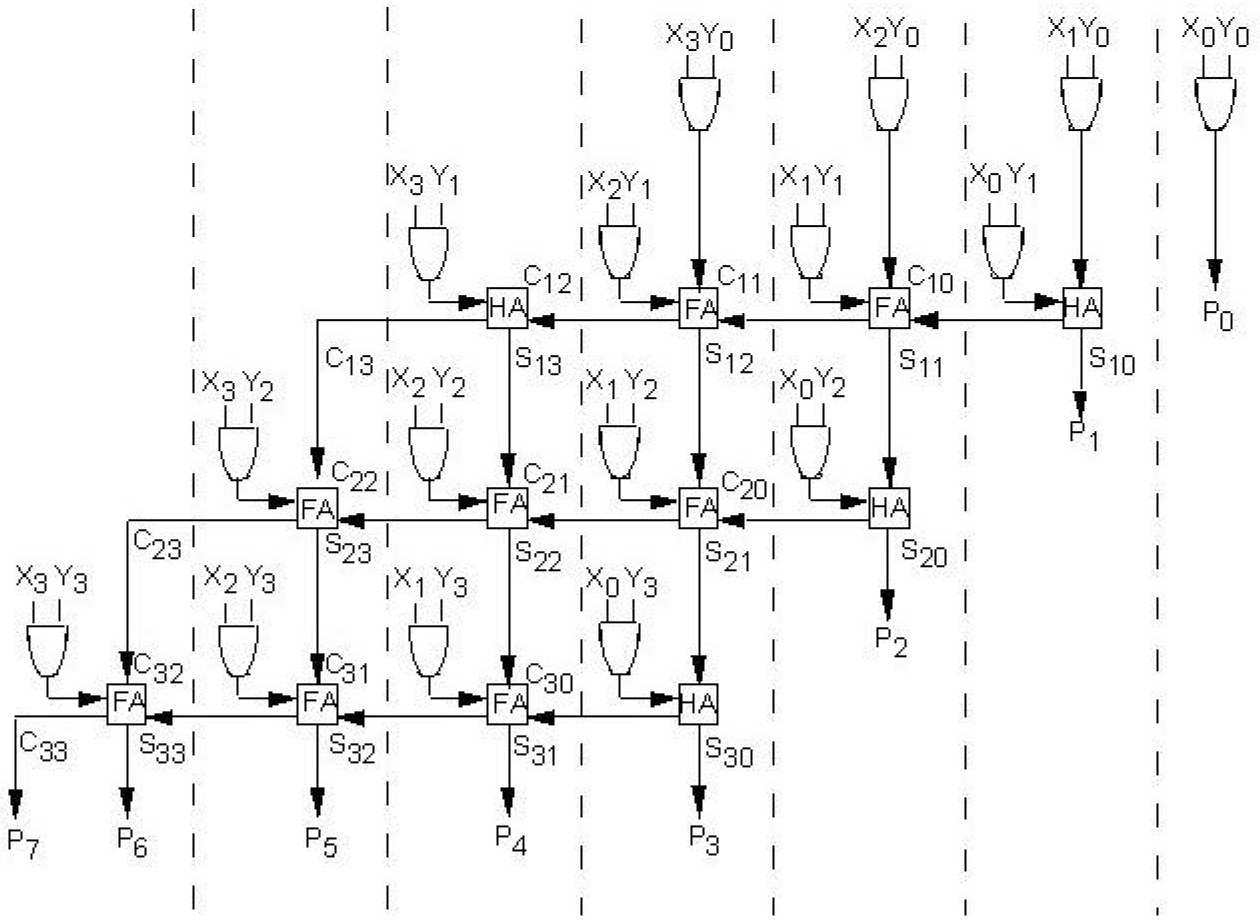
- **Multiplication is achieved by adding a list of shifted multiplicands according to the digits of the multiplier.**
- **Ex. (unsigned)**

```
  11              1 0 1 1     multiplicand (4 bits)
X 13         X    1 1 0 1     multiplier  (4 bits)
--------     -------------------
   33             1 0 1  1
  11            0 0 0 0
______         1 0 1 1
  143        1 0 1 1
            ---------------------
            1 0 0 0 1 1 1 1     Product (8 bits)
```

```
                                          X3     X2     X1     X0
                                      x   Y3     Y2     Y1     Y0
                                    ------------------------------
                                  X3.Y0  X2.Y0  X1.Y0  X0.Y0
                           X3.Y1  X2.Y1  X1.Y1  X0.Y1
                    X3.Y2  X2.Y2  X1.Y2  X0.Y2
             X3.Y3  X2.Y3  X1.Y3  X0.Y3
-----------------------------------------------------------------
P7    P6     P5     P4     P3     P2     P1     P0
```

- **An n-bit X n-bit multiplier can be realized in combinational circuitry by using an array of n-1 n-bit adders where is adder is shifted by one position.**
- **For each adder one input is the multiplied by 0 or 1 (using AND gates) depending on the multiplier bit, the other input is n partial product bits.**

4x4 Array Multiplier



X_3Y_0
X_2Y_0
X_1Y_0
X_0Y_0
X_3Y_1
X_2Y_1
X_1Y_1
X_0Y_1
HA
FA
FA
HA
C_{12}
C_{11}
C_{10}
P_0
C_{13}
S_{13}
S_{12}
S_{11}
S_{10}
P_1
X_3Y_2
X_2Y_2
X_1Y_2
X_0Y_2
FA
FA
FA
HA
C_{22}
C_{21}
C_{20}
C_{23}
S_{23}
S_{22}
S_{21}
S_{20}
P_2
X_3Y_3
X_2Y_3
X_1Y_3
X_0Y_3
FA
FA
FA
HA
C_{32}
C_{31}
C_{30}
C_{33}
S_{33}
S_{32}
S_{31}
S_{30}
P_7
P_6
P_5
P_4
P_3