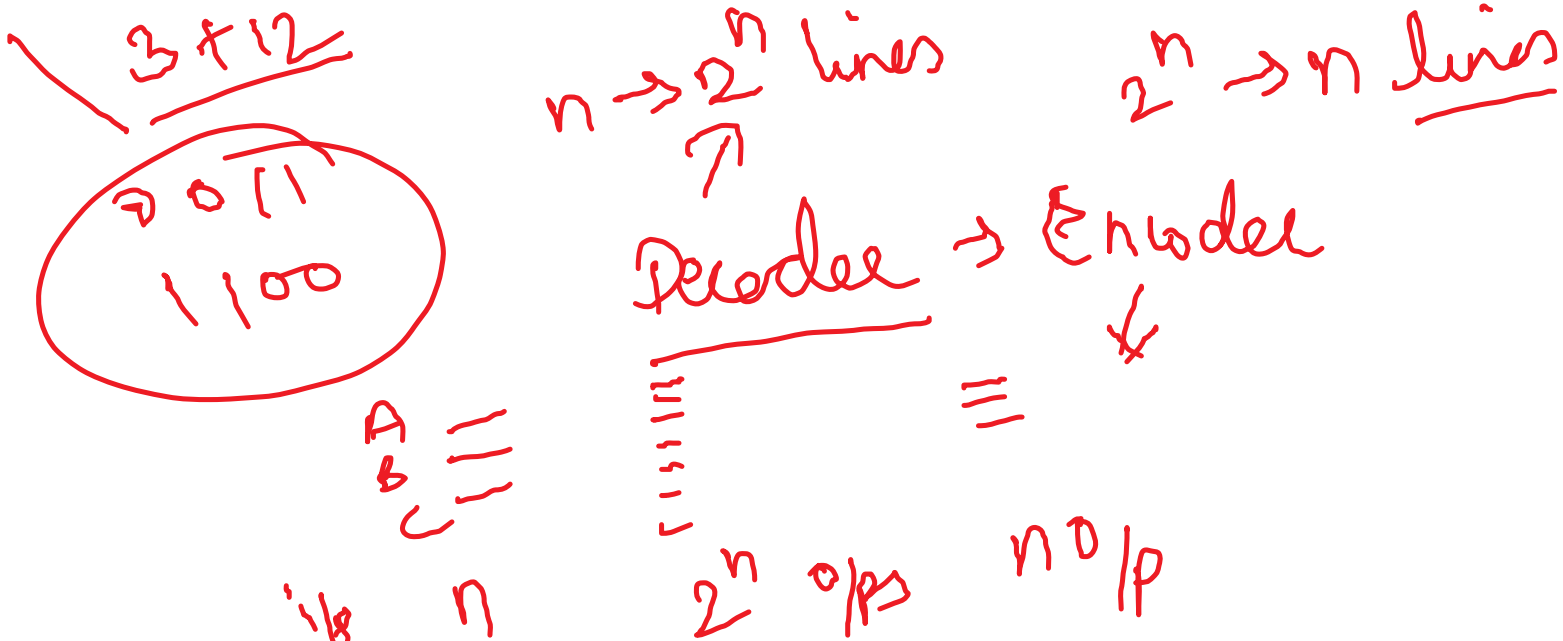
ADDER AND SUBTRACTOR



Combinational Circuits (3 module)

logic

When logic gates are connected together to produce a specific o/p for certain specific combinations of i/p variables.

→ no storage

→ i/p → o/p at all times.

Binary Adder

- A combinational circuit that performs the addition of two bits is called a half adder.
- The truth table for the half adder is listed below:

Table 4-3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S: Sum
C: Carry

$S = x'y + xy'$

$C = xy$

Ex-OR

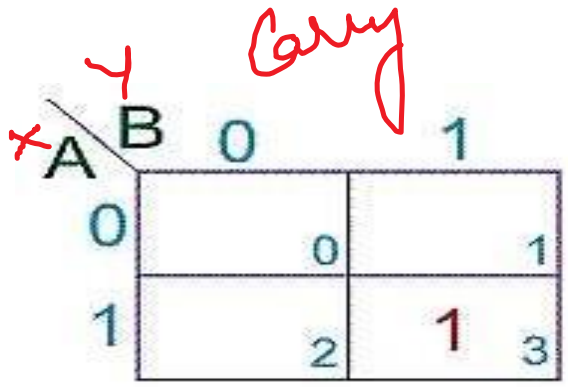
$$x'y + xy' = x \oplus y$$

Ex-NOR

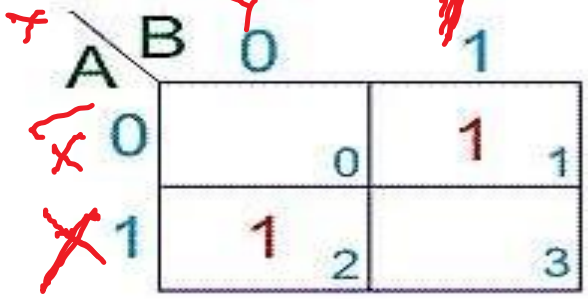
$$x'y' + xy = x \odot y$$

$$\overline{x \oplus y} = x \odot y$$

K-map for half adder



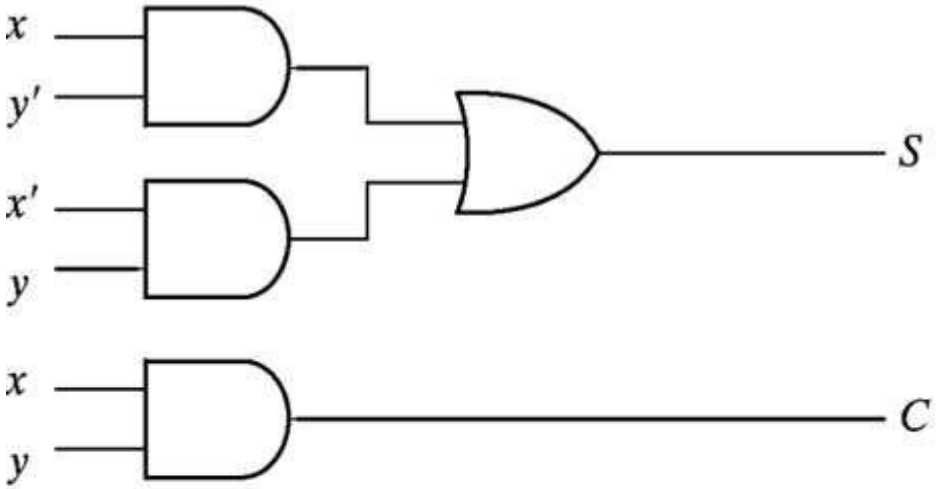
K-map for Carry



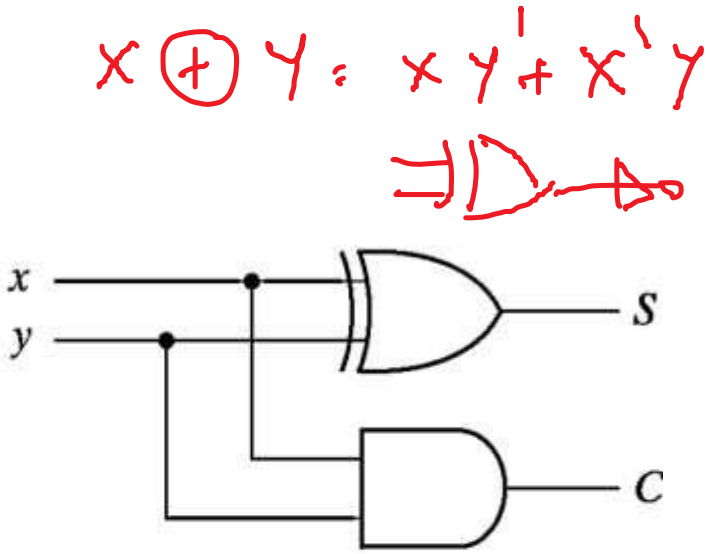
K-map for Sum

$$Carry\ C = AB \text{ and } Sum\ C = A\overline{B} + \overline{A}B.$$

Implementation of Half-Adder



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

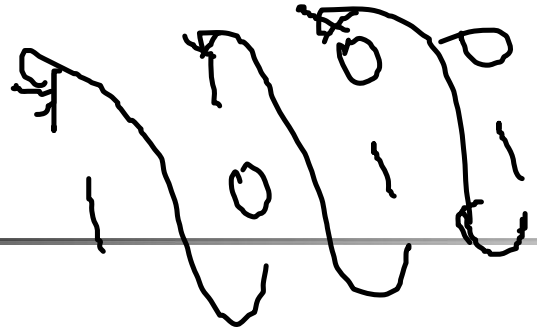
Fig. 4-5 Implementation of Half-Adder

Full-Adder

- One that performs the addition of three bits(two significant bits and a previous carry) is a full adder.

Table 4-4
Full Adder

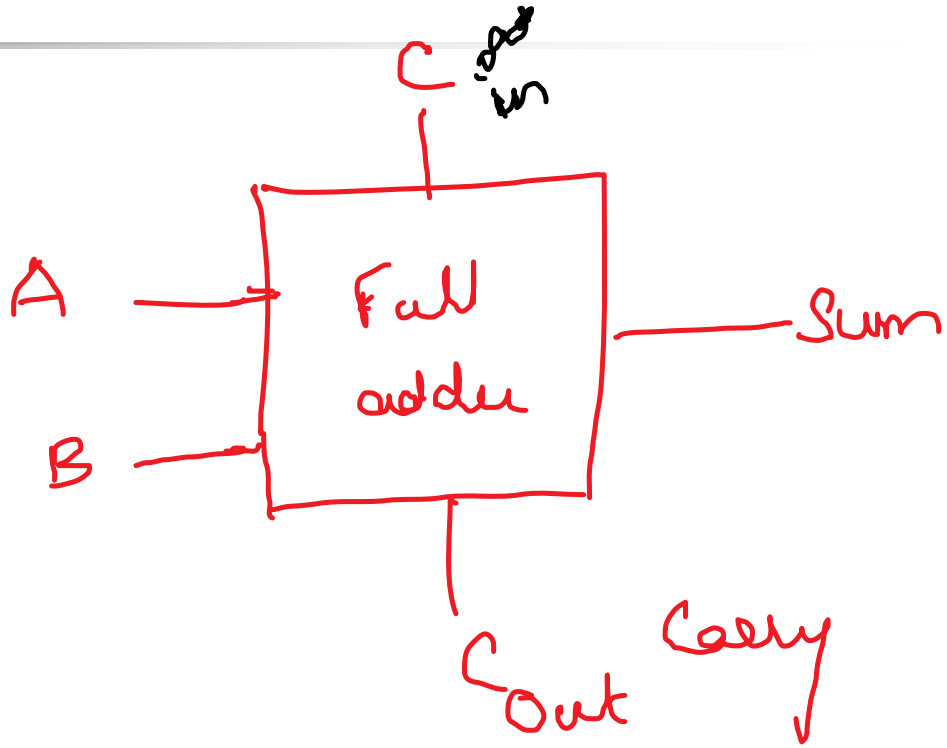
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Question

3 i/ps , 2 o/ps

Design a full adder circuits



→ T.T

→ K.map

→ Simplification of Sum and Carry.

Full Adder

- A combinational circuit that adds 3 input bits to generate a Sum bit and a Carry bit

	X	Y	Z	C	S
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

Sum

X \ YZ	00	01	11	10
0	0	1	0	1
1	1	0	1	0

S = X'Y'Z + X'YZ' + XY'Z' + XYZ

= X ⊕ Y ⊕ Z

Carry

X \ YZ	00	01	11	10
0	0	0	1	0
1	0	1	1	1

C = XY + YZ + XZ

$$X \odot Z = \overline{X \oplus Z}$$

$$S = \bar{X}\bar{Y}Z + X\bar{Y}\bar{Z} + \bar{X}Y\bar{Z} + XYZ$$

$$S = \bar{Y}\left(\bar{X}Z + X\bar{Z}\right) + Y\left(\bar{X}\bar{Z} + XZ\right) \rightarrow X \oplus Z$$

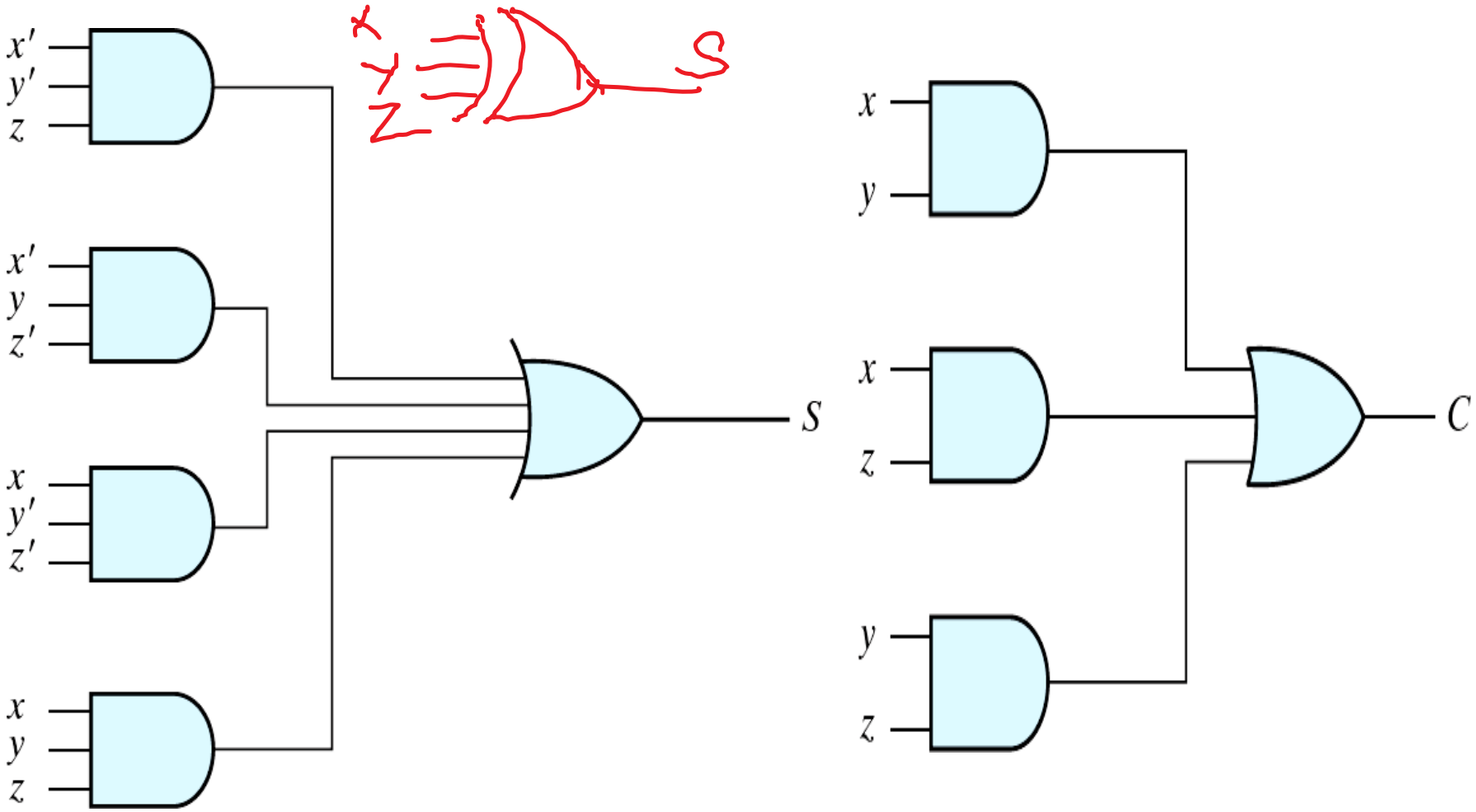
$$S = \bar{Y}\left(X \oplus Z\right) + Y\left(X \odot Z\right)$$

$$S = \bar{Y}\left(X \oplus Z\right) + Y\left(\overline{X \oplus Z}\right)$$

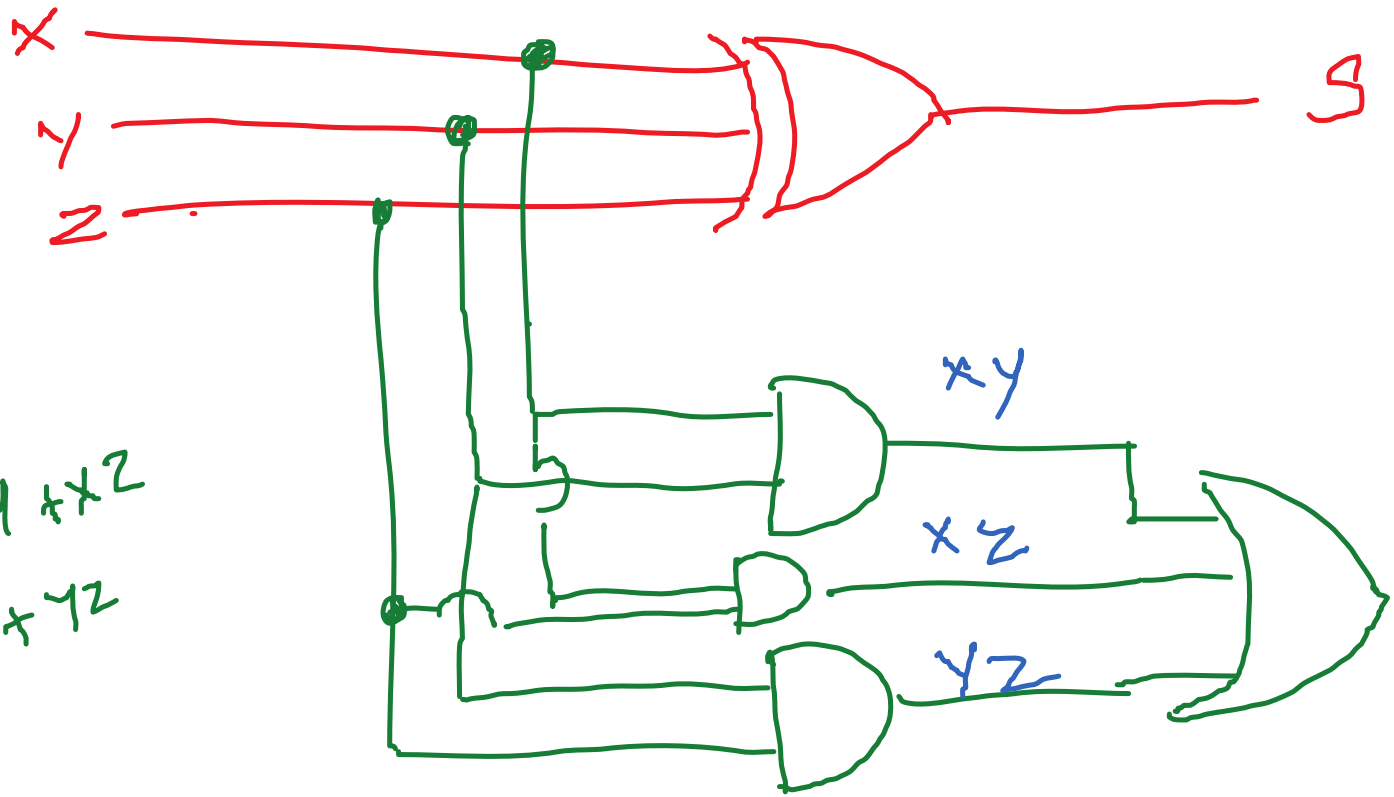
$$A'B + AB' = A \oplus B$$

$$S = Y \oplus X \oplus Z \quad \text{con } S = X \oplus Y \oplus Z$$

Logic Diagram of Full Adder



Simplified full adder circuit.



$C = xy + xz + yz$

plot full adder using two half adder. and one or gate?.

2 half adder sum half adder sum

$$\text{Sum} = C \oplus (A \oplus B)$$

$$= C_{in} \oplus (A'B + AB')$$

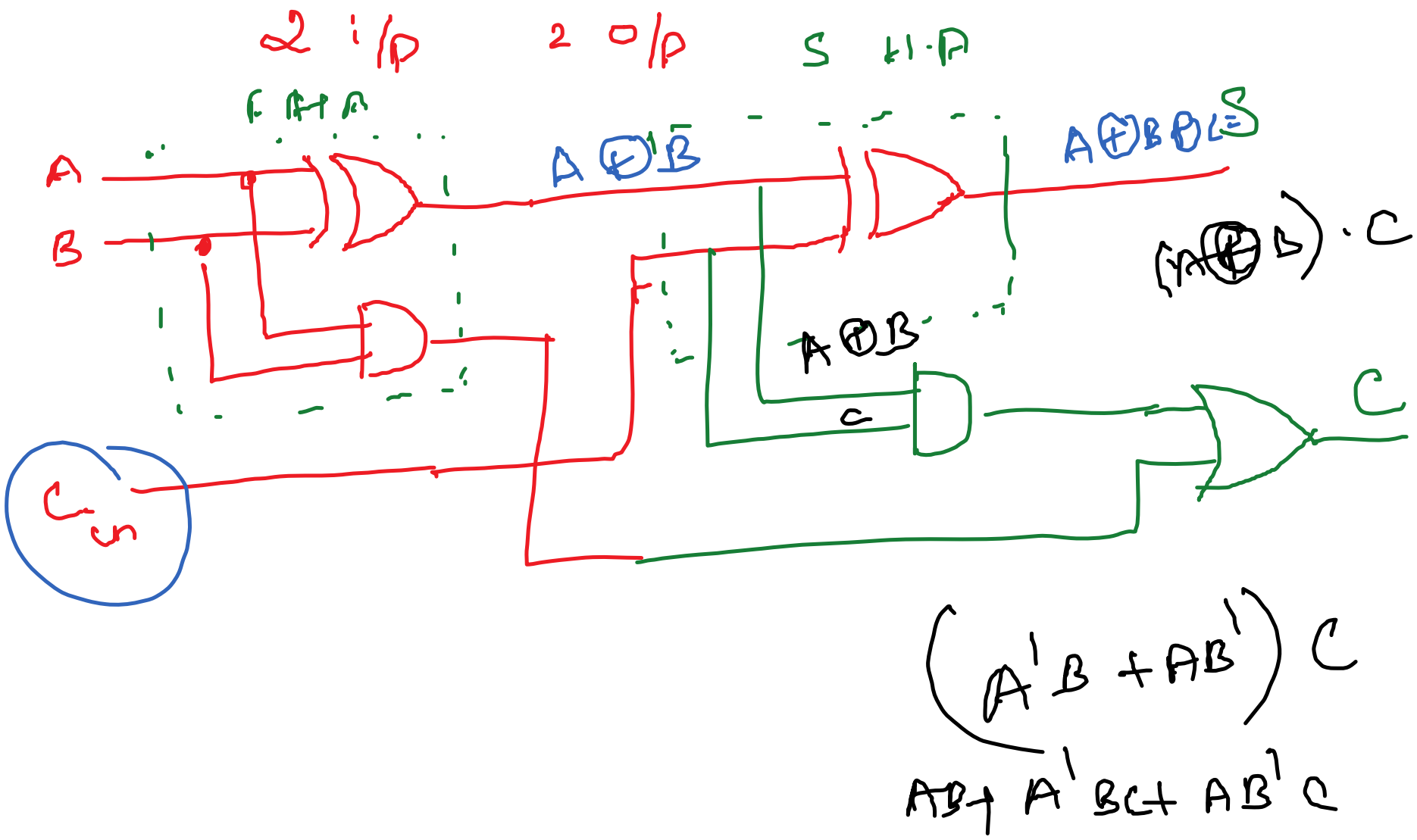
$$= C\overline{(A'B + AB')} + \bar{C}(A'B + AB')$$

Actual values from full adder T?.

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

$$C = AB + AC_{in} + BC_{in}$$

half adder.

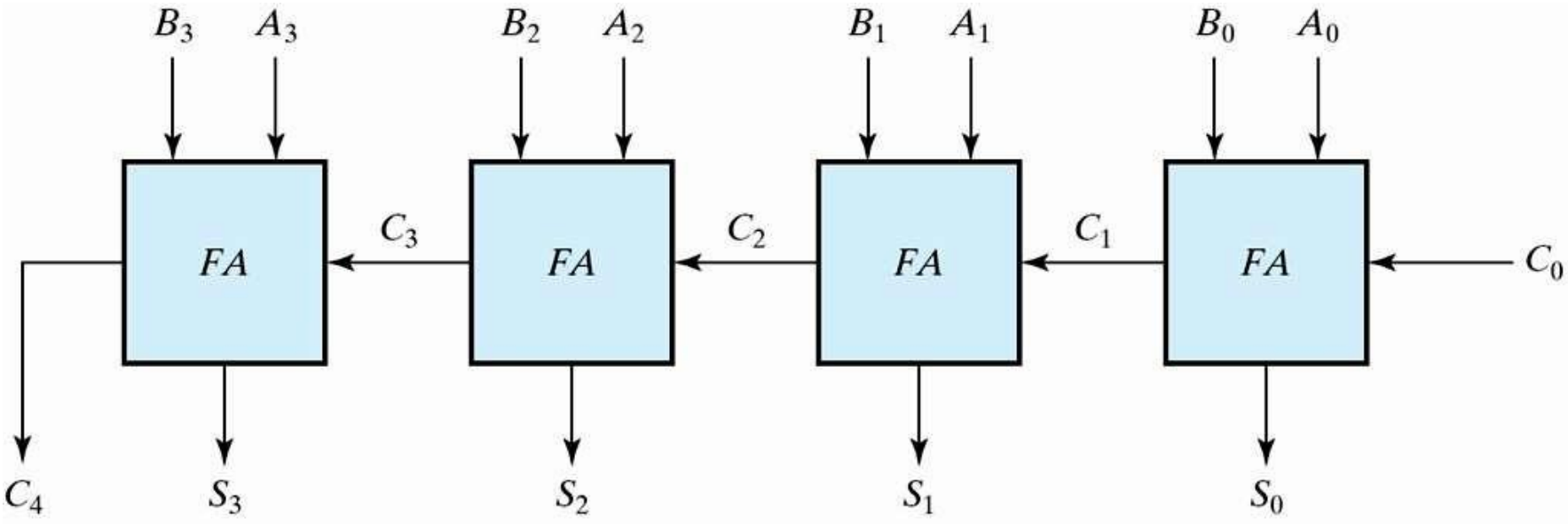

2 i/p
2 o/p
S H.A
F.H.A
A
B
A⊕B
A⊕B=S
(A⊕B)·C
A⊕B
C
C
C_in
(A'B + AB')C
AB + A'BC + AB'C

// Verilog code for full adder
// Behavioral code for full adder

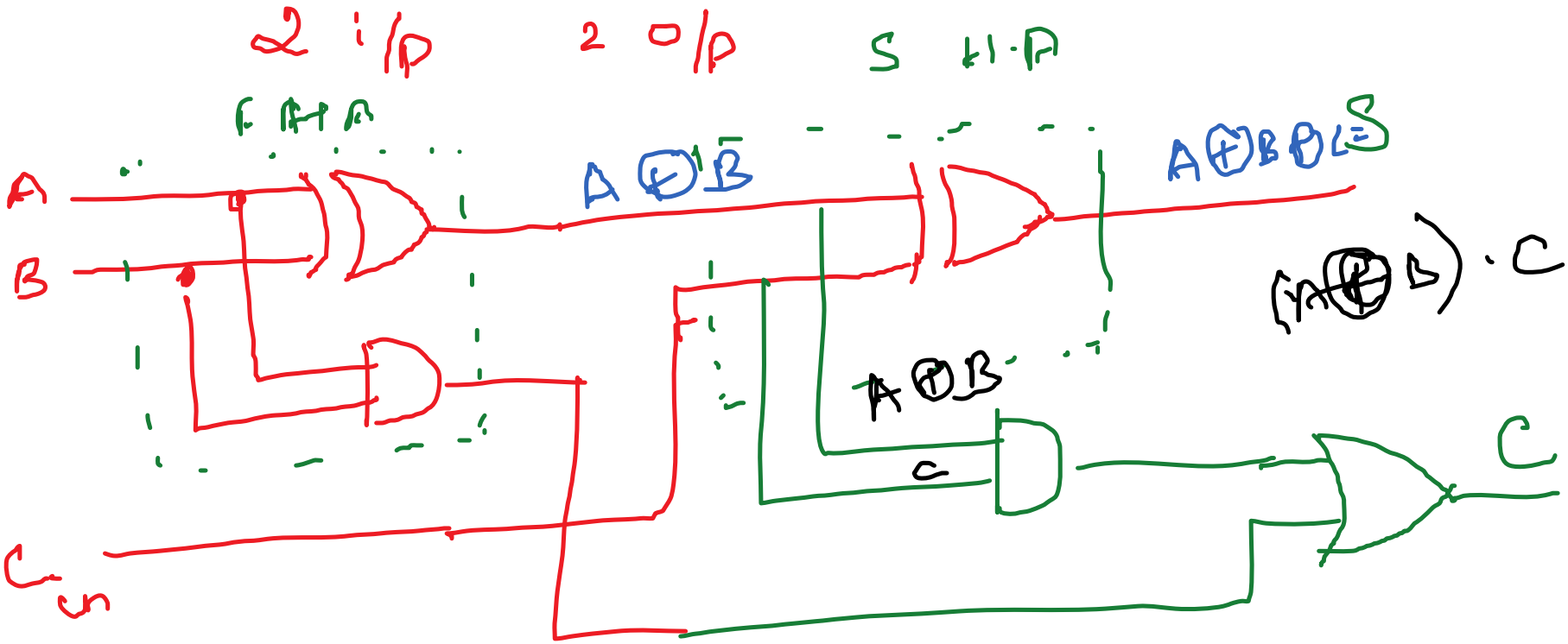
```
module
Full_Adder_Behavioral_Verilog(input X1,
X2, Cin,
  output S, Cout);
    reg[1:0] temp;
   always @(*)
   begin
   temp = {1'b0,X1} + {1'b0,X2}+{1'b0,Cin};
   end
   assign S = temp[0];
   assign Cout = temp[1];
endmodule
```

// Testbench code of the behavioral code for full adder

```
`timescale 10ns/ 10ps;
module Testbench_Behavioral_adder();
 reg A,B,Cin;
 wire S,Cout;
Full_Adder_Behavioral_Verilog Behavioral_adder
(.X1(A), .X2(B),  .Cin(Cin), .S(S), .Cout(Cout)
   );
 initial begin
   A = 0; B = 0; Cin = 0;
   #5;  A = 0; B = 0; Cin = 1;
   #5;  A = 0; B = 1; Cin = 0;
   #5; A = 0; B = 1; Cin = 1;
   #5; A = 1; B = 0; Cin = 0;
   #5; A = 1; B = 0; Cin = 1;
   #5; A = 1; B = 1; Cin = 0;
   #5; A = 1; B = 1; Cin = 1;
   #5;
  end
endmodule
```


B_3
A_3
B_2
A_2
B_1
A_1
B_0
A_0
FA
FA
FA
FA
C_3
C_2
C_1
C_0
C_4
S_3
S_2
S_1
S_0

```
// Structural code for full adder
module Full_Adder_Structural_Verilog(
  input X1, X2, Cin,
  output S, Cout );
    wire a1, a2, a3;
    xor u1(a1,X1,X2);
  and u2(a2,X1,X2);
  and u3(a3,a1,Cin);
  or u4(Cout,a2,a3);
    xor u5(S,a1,Cin);
endmodule
```



```
// Structural code for full adder
module Full_Adder_Structural_Verilog(
 input X1, X2, Cin,
 output S, Cout );
   wire a1, a2, a3;
   xor u1(a1,X1,X2);
 and u2(a2,X1,X2);
 and u3(a3,a1,Cin);
 or u4(Cout,a2,a3);
   xor u5(S,a1,Cin);
endmodule
```

```
// Testbench code of the structural code for
full adder
//timescale 10ns/ 10ps;
module Testbench_structural_adder();
 reg A,B,Cin;
 wire S,Cout;
 //Verilog code for the structural full adder
 Full_Adder_Structural_Verilog
structural_adder G1( .X1(A), .X2(B), .Cin(Cin),
    .S(S), .Cout(Cout) );
 initial begin
   A = 0;   B = 0;   Cin = 0;
   #10;   A = 0;   B = 0;   Cin = 1;
   #10;   A = 0;   B = 1;   Cin = 0;
   #10;   A = 0;   B = 1;   Cin = 1;
   #10;   A = 1;   B = 0;   Cin = 0;
   #10;   A = 1;   B = 0;   Cin = 1;
   #10;   A = 1;   B = 1;   Cin = 0;
   #10;  A = 1;B = 1;   Cin = 1;
   #10;
  end
endmodule
```

FULL ADDER

```
Module module fa(a, b, c, sum, carry);
input a, b, c;
output sum, carry;
wire d,e,f;
xor(sum,a,b,c);
and(d,a,b);
and(e,b,c);
and(f,a,c);
or(carry,d,e,f);
endmodule
```

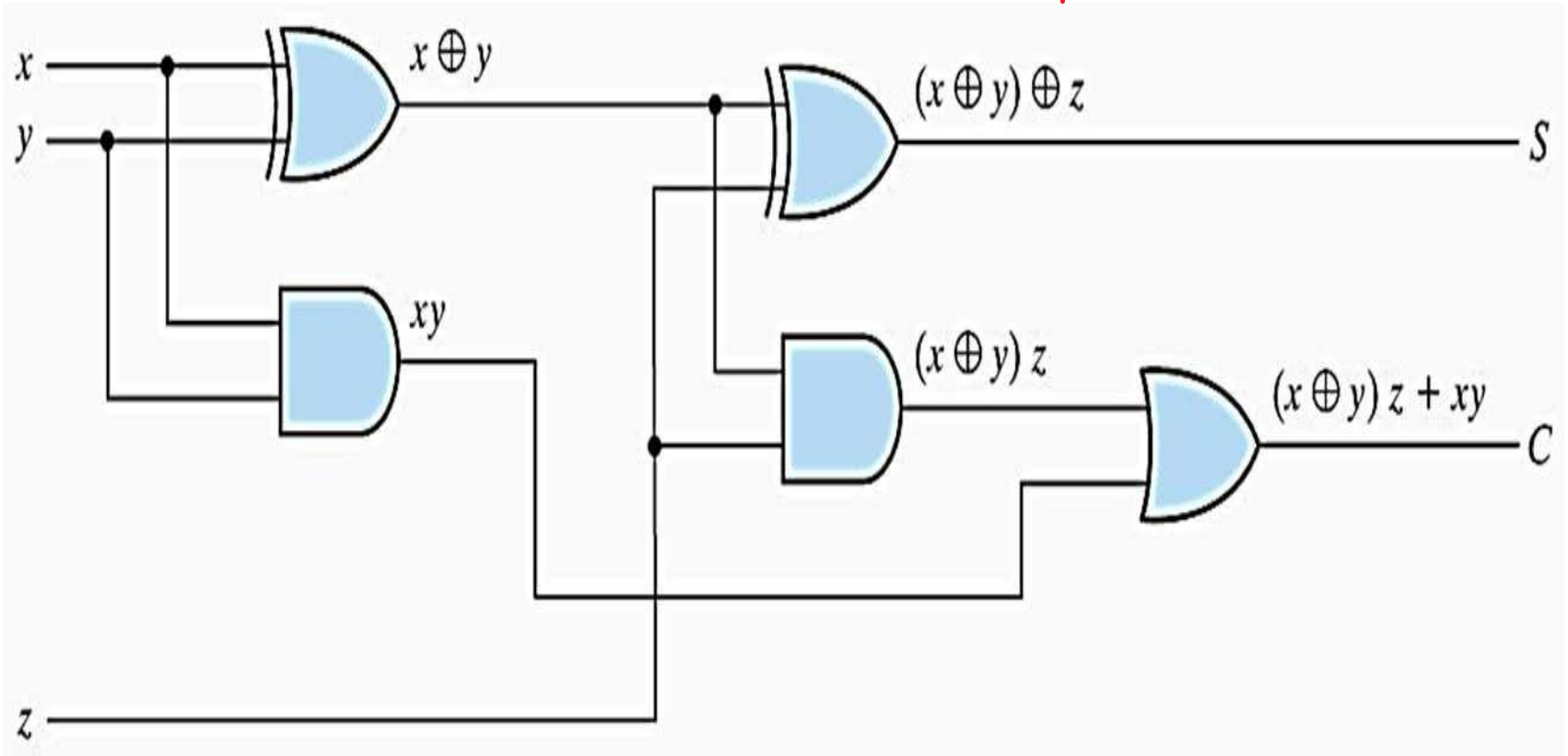
```
TEST BENCH module fulladdt_b;
reg a, b, c;
wire sum, carry;
fa uut (  .a(a),   .b(b),.c(c),.sum(sum),.carry(carry)  );
initial begin
 a=1'b0;b=1'b0;c=1'b0;
#10 a=1'b0;b=1'b0;c=1'b1;
#10 a=1'b0;b=1'b1;c=1'b0;
#10 a=1'b0;b=1'b1;c=1'b1;
#10 a=1'b1;b=1'b0;c=1'b0;
#10 a=1'b1;b=1'b0;c=1'b1;
#10 a=1'b1;b=1'b1;c=1'b0;
#10 a=1'b1;b=1'b1;c=1'b1;
#10$stop;
end
endmodule
```

Logic Diagram of Full Adder

$$S = \Sigma m(1,2,4,7)$$
$$= X' Y' C_{in} + X' Y C_{in}' + X Y' C_{in}' + X Y C_{in}$$
$$= X' (Y' C_{in} + Y C_{in}') + X (Y' C_{in}' + Y C_{in})$$
$$= X' (Y \oplus C_{in}) + X (Y \oplus C_{in})'$$
$$= X \oplus Y \oplus C_{in}$$

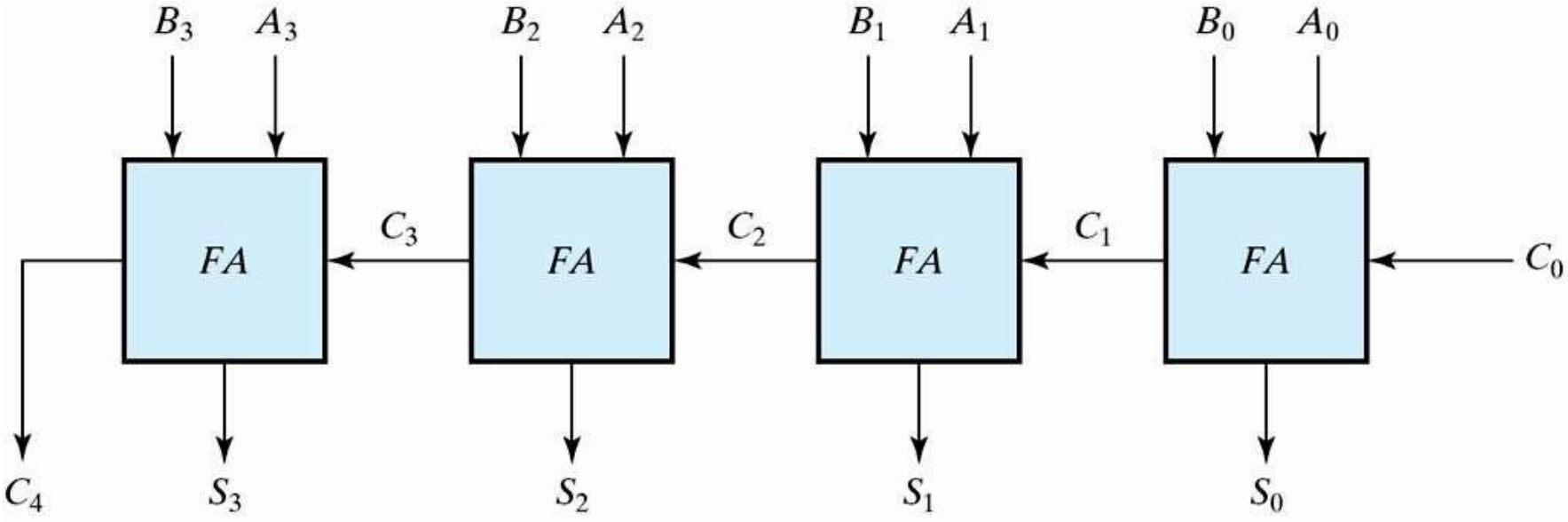
$$C_{out} = \Sigma m(3,5,6,7)$$
$$= X' Y C_{in} + X Y' C_{in} + X Y C_{in}' + X Y C_{in}$$
$$= (X' Y + X Y') C_{in} + XY(C_{in}' + C_{in})$$
$$= (X \oplus Y) C_{in} + XY$$

2 half adder can helps to represent full adder.



4-Bit Full Adder

- Binary adder



Subtractor is an electronic logic circuit for calculating the difference between two binary numbers which provides the difference and borrow as output.

➢ Halfsubtractor

Half Subtractor is used for subtracting one single bit binary number from another single bit binary number.

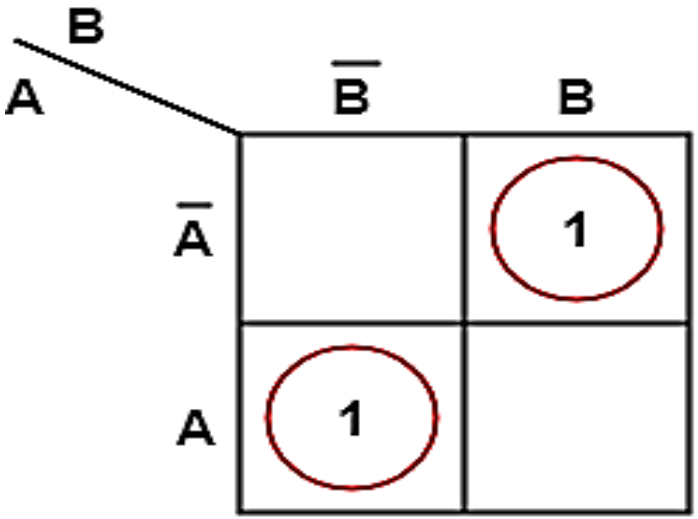
It has two inputs; Minuend (A) and Subtrahend (B) and two outputs; Difference (D) and Borrow (B_{out}).

❑ TruthTable

Input		Output	
A	B	Difference (D)	Borrow (B_{out})
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

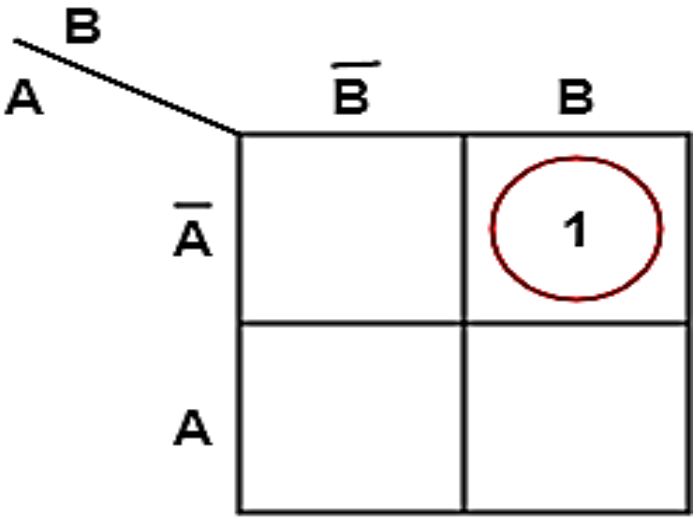
❑ Solving truth table using K-map

For D:



$D = A \oplus B$

For b:

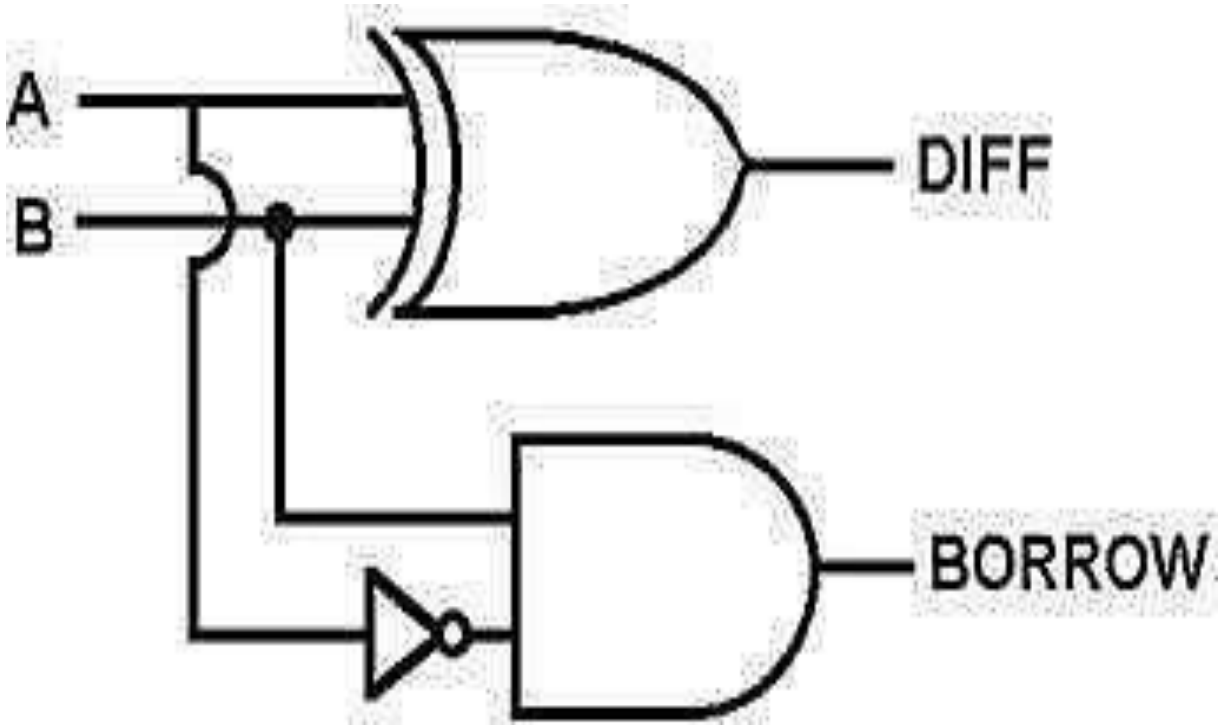


$b = \bar{A}\,B$

Borrow = $\bar{A}$.B

Difference = A $\oplus$ B

❑ LogicalCircuit



fULL subtractor

A logic Circuit Which is used for subtracting three single bit binary numbers is known as Full Subtractor.

- It has three inputs;

1. Minuend (A),
2. Subtrahend(B)
3. Subtrahend(C)

- two outputs;

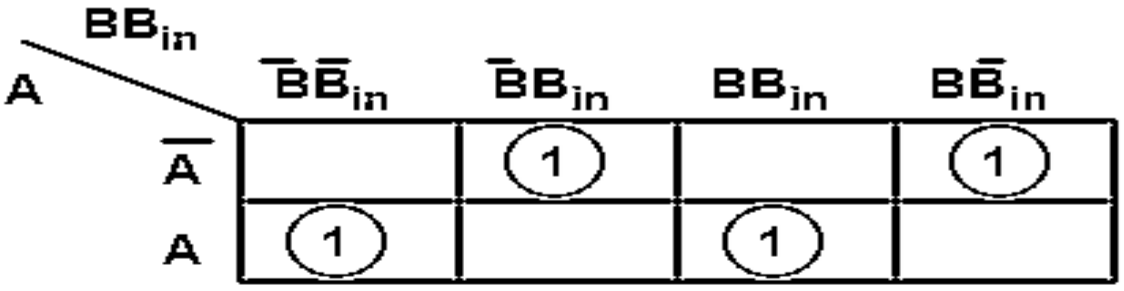
1. Difference (D)
2. Borrow (B_{out}).

❑ TruthTable

Input			Output	
A	B	C	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

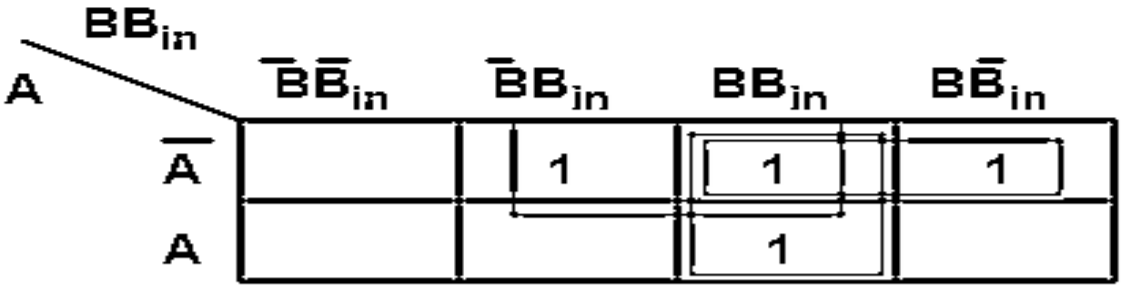
❑ Solving Truth Table using K-Map

For D:



$$D = A \oplus B \oplus B_{in}$$

For B_{in}:



$$B_{out} = \bar{A}\,B + (\bar{A} + B)\,B_{in}$$

❑ K-MapMinimization

From the Truth Table The Difference and Borrow will written as,

Difference=A'B'C+A'BC'+AB'C'+ABC

Difference=A ⊕B⊕C

Borrow=A'B'C+A'BC'+A'BC+ABC

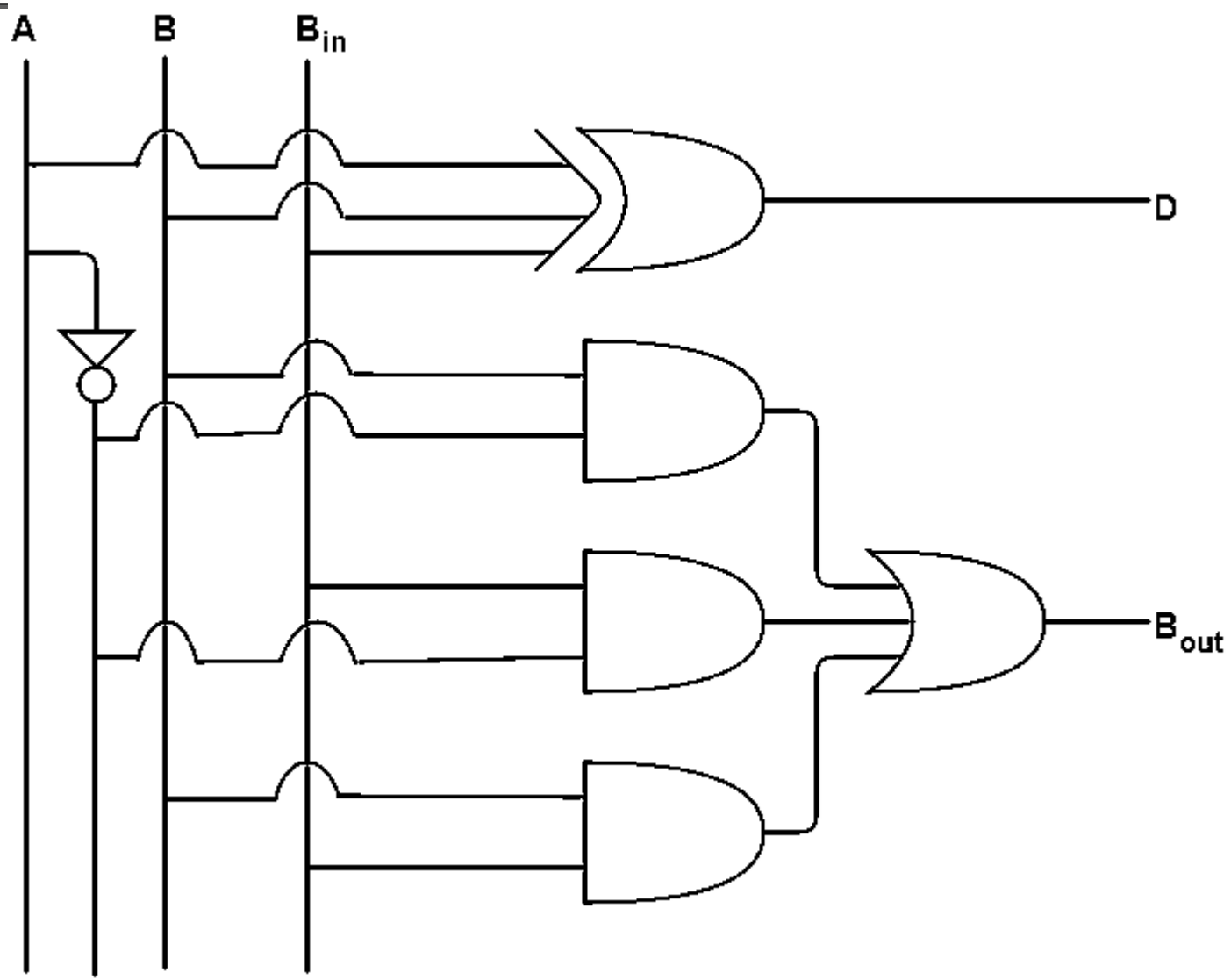
=A'B'C+A'BC'+A'BC+A'BC+A'BC+ABC

=A'C(B'+B)+A'B(C'+C)+BC(A'+A)

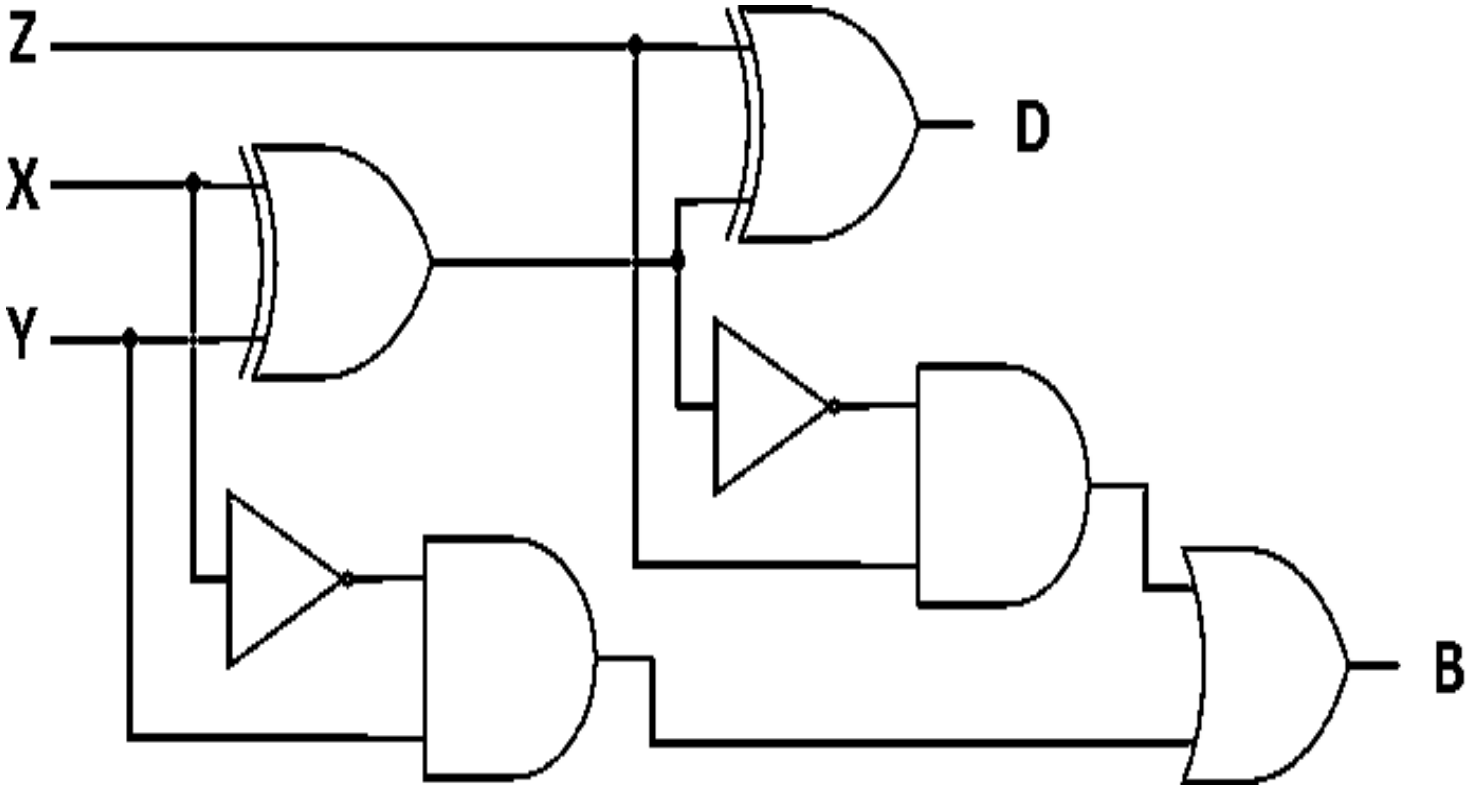
Borrow=A'C+A'B+BC

B(out) = BC + (B ⊕ C) A

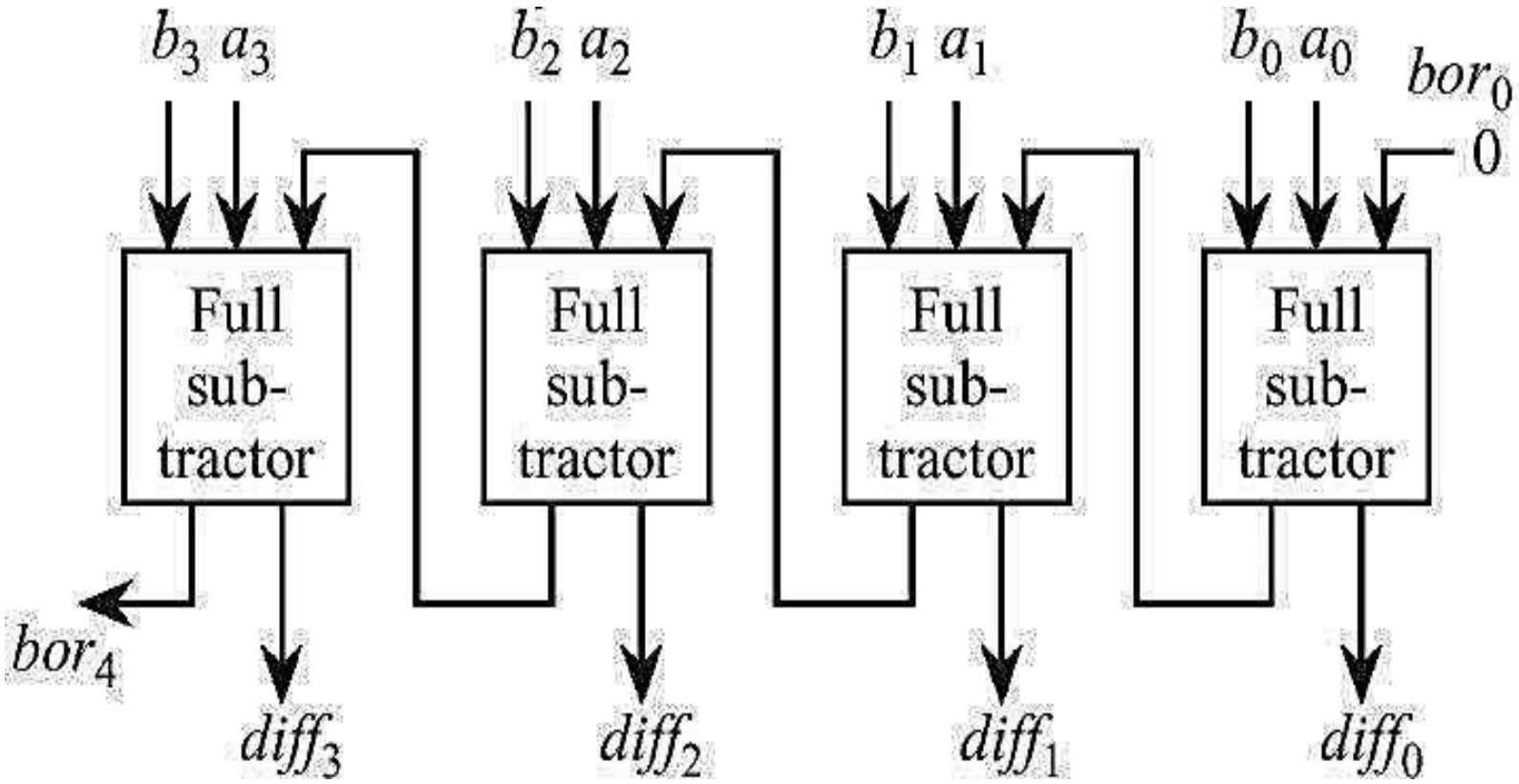
❑ LogicalCircuit



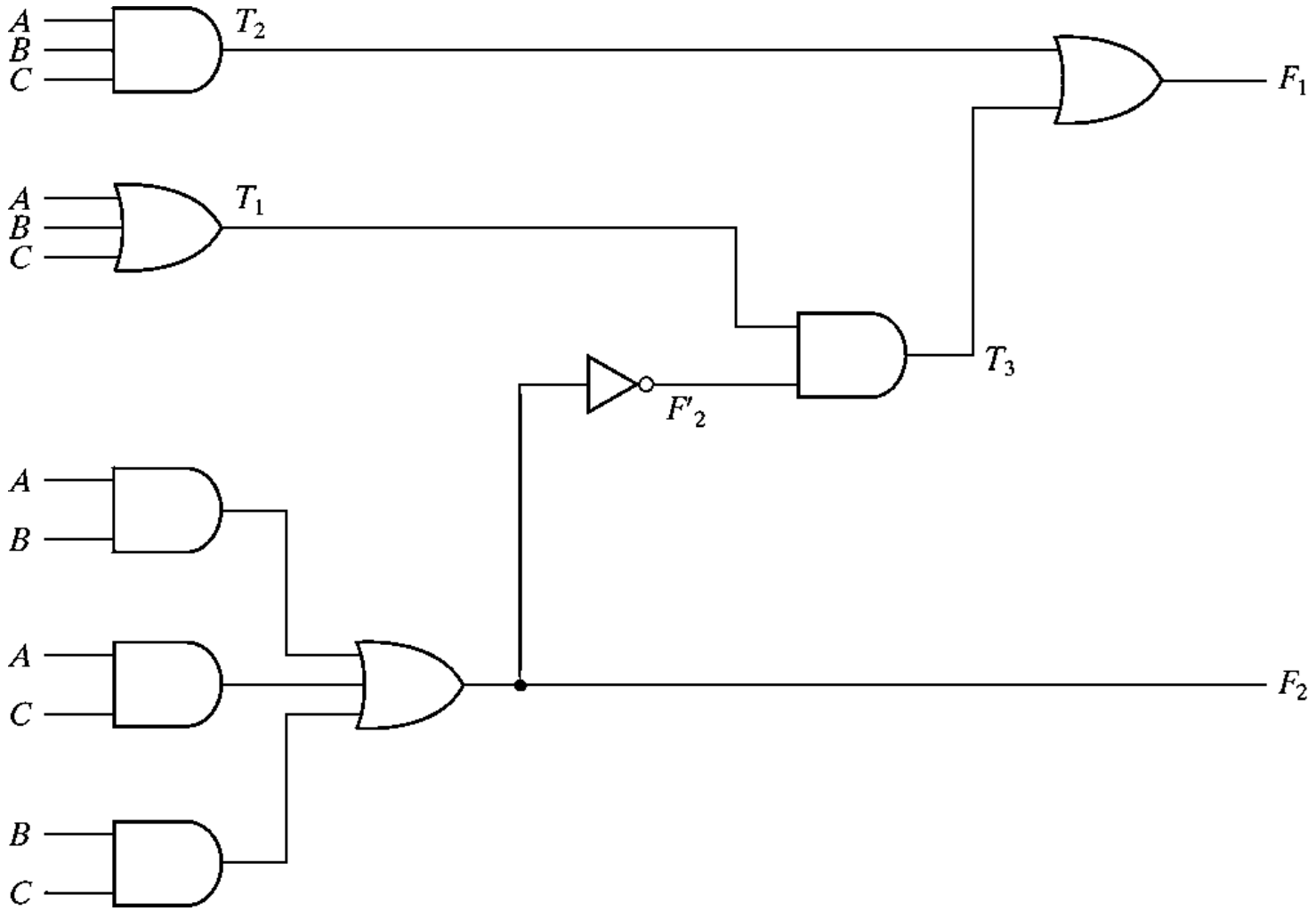
❑ LogicalCircuit



❑ Diagram


b_3 a_3
b_2 a_2
b_1 a_1
b_0 a_0
bor_0
0
Full sub-tractor
Full sub-tractor
Full sub-tractor
Full sub-tractor
bor_4
$diff_3$
$diff_2$
$diff_1$
$diff_0$

Example: derive truth table from logic diagram

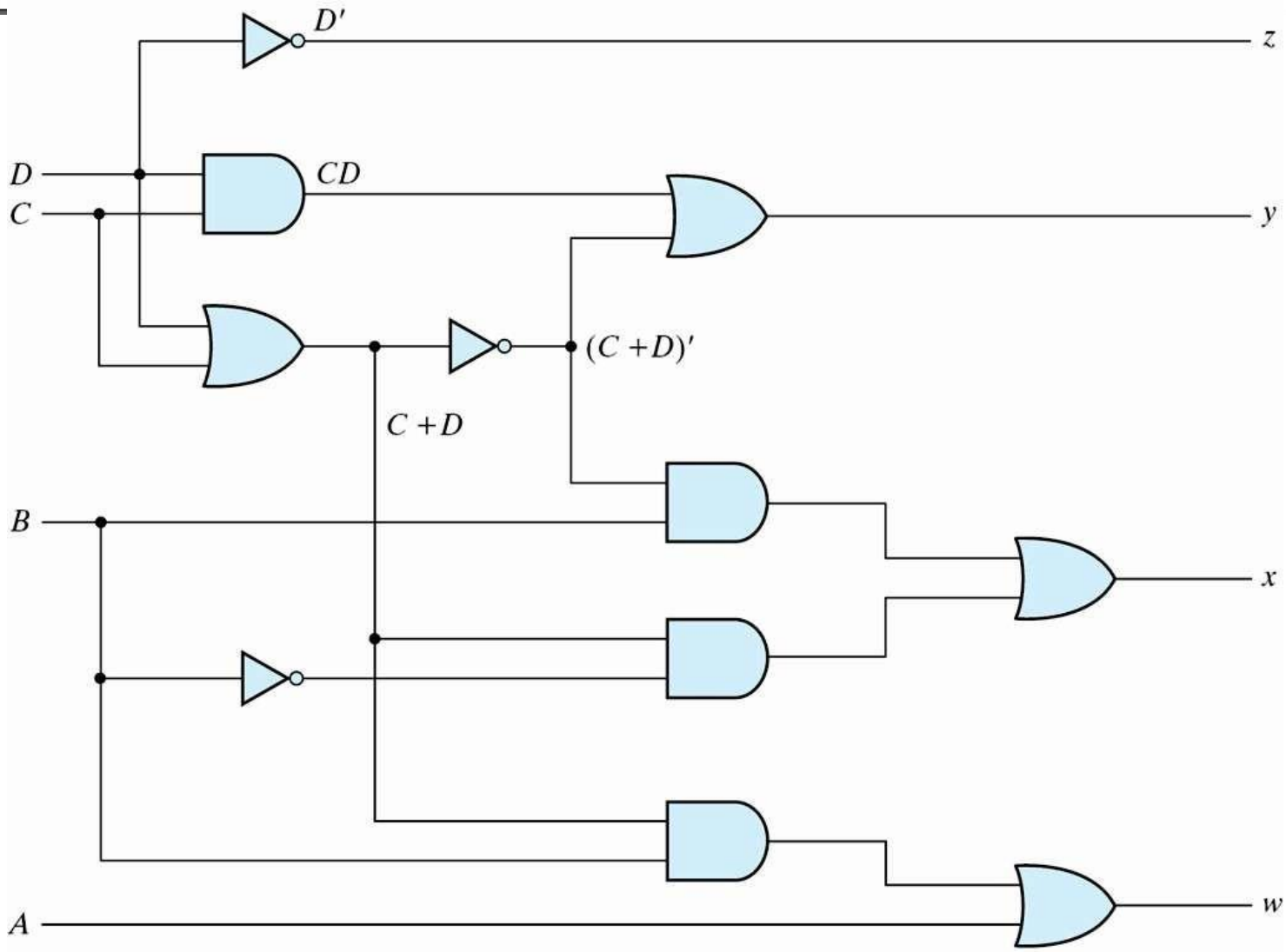


Truth Table

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Circuit implementation



Design 3-to-8 decoder with enable implement the 4-to-16 decoder

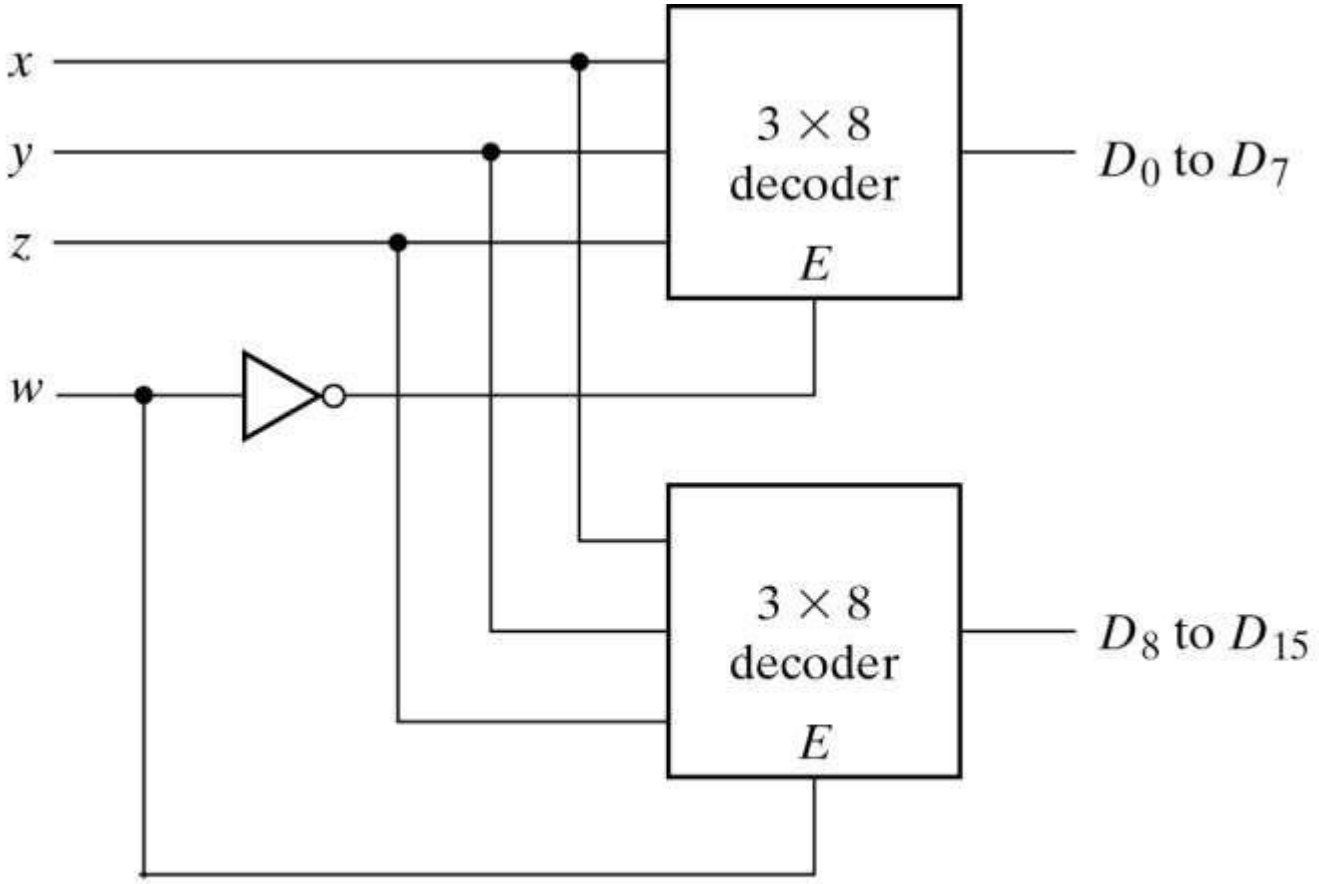


Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

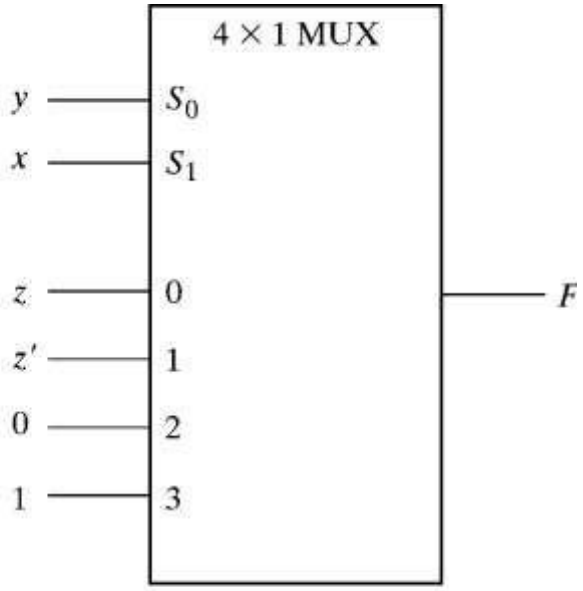
Boolean function implementation

- A more efficient method for implementing a Boolean function of n variables with a **multiplexer**.

$$F(x, y, z) = \Sigma(1,2,6,7)$$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

4-input function with a multiplexer

$F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

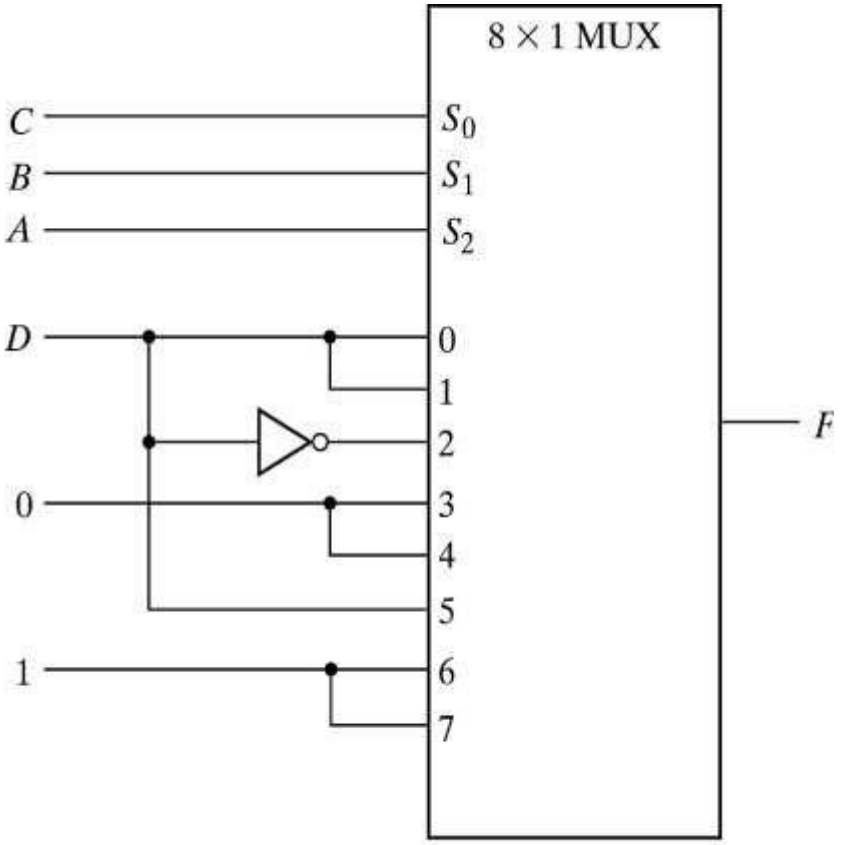


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer