COMBINATIONAL CIRCUITS

COMBINATIONAL LOGIC

- The output of this type of logic is dependent solely on its current inputs.
- When certain input values are set, a combinatorial circuit generates output values corresponding to those input values. When the input of the combinatorial logic are changed, the outputs are changing to reflect the changes in the new input values.
- Previous values of the inputs do not matter, the current outputs depend solely on the current inputs.
- It's assumed that Circuits are without time delay

DESIGN PROCEDURE

1. Design a circuit from a specification.
2. From the specifications of the circuit, determine the required number of inputs and outputs, and assign a letter symbol to each.
3. Derive the truth table that defines the required relation ship between inputs and outputs.
4. Obtain the simplified Boolean functions of each outputs as function of the input variables
5. Draw logic diagram and verify correctness

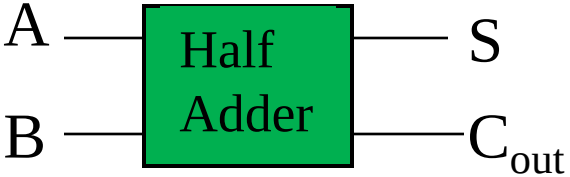
HALF ADDER

- Logic gate that perform arithmetic addition for 1-bit
- Two inputs A, B to half-adder. Resultants are Sum(S) and Carry(Cout)
- A wider than 1 bit adder can't use this circuit, because there is no way to input carry information from the previous bits

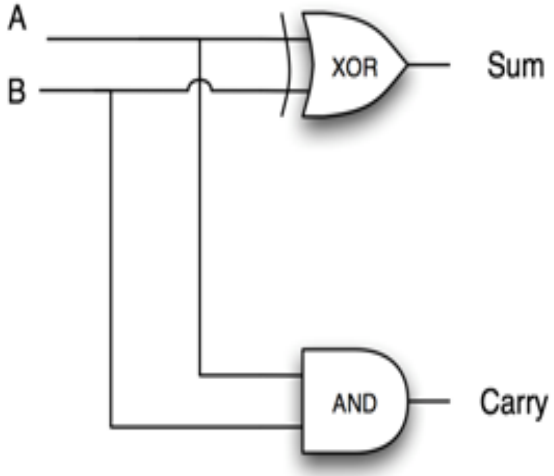
$$S = \overline{A}B + A\overline{B}$$

$$= A \oplus B$$

$$C = AB$$

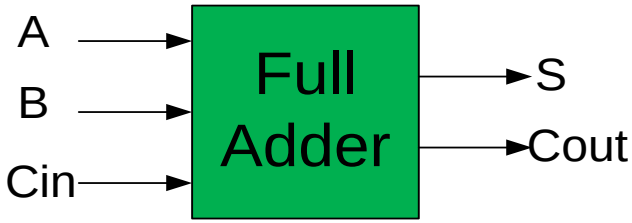


A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



FULL ADDER

- A full adder is a circuit that computes the sum of three bits and gives a two-bit answer.
- The full adder for a given column adds two bits from the input numbers together with a one-bit carry from the previous column to the right. The adder produces a two-bit answer; one of these bits is used as a carry into the next column.



FULL ADDER

- A full adder has 3 inputs and 2 outputs
- The truth table of the full-adder can be drawn with inputs A,B and Cin with outputs S and Cout.

A	B	Cin	Sum	Cout
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

FULL ADDER

- From the truth table we can write the Boolean equation for the S and Cout

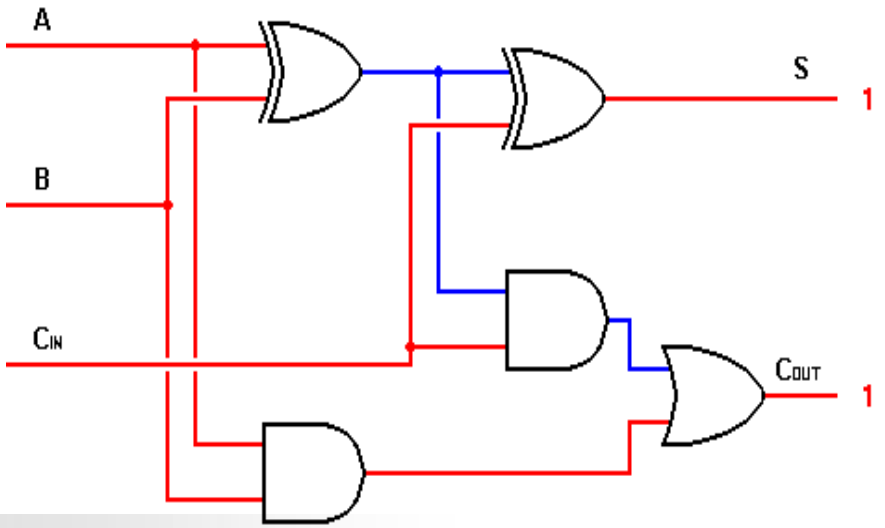
$$S = \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\overline{C}_{in} + ABC_{in}$$

$$C_{out} = \overline{A}BC_{in} + A\overline{B}C_{in} + AB\overline{C}_{in} + ABC_{in}$$

- Simplify using Boolean Algebra and K-map, we get

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

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

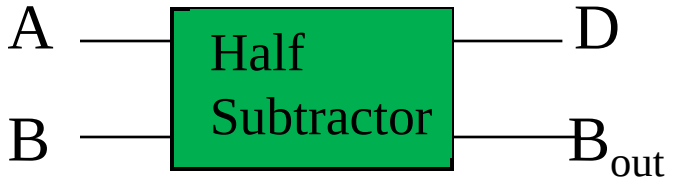


HALF SUBTRACTOR

- Logic gate that perform arithmetic subtraction for 1-bit
- Two inputs A, B to half-subtractor. Resultants are Difference(D) and Borrow(B)

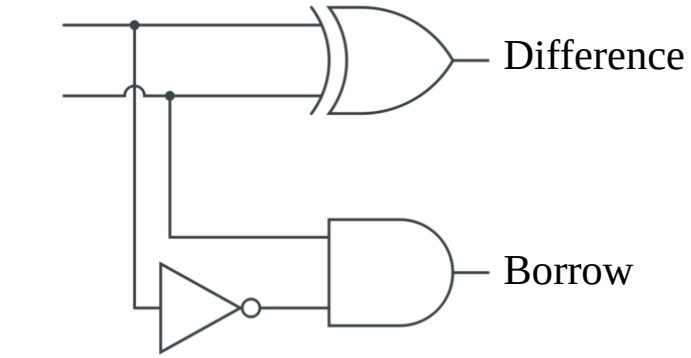
$$D = \overline{A}B + A\overline{B}$$

$$B_{out} = \overline{A}B$$



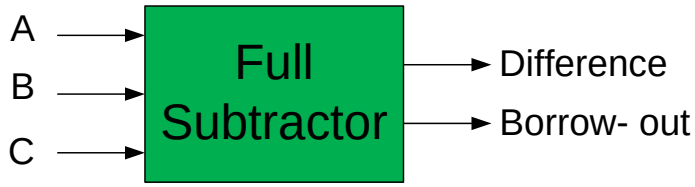
A	B	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

```
10
-1
 1
```



FULL SUBTRACTOR

- A full subtractor is a circuit that computes the difference of three bits and gives a two-bit answer.
- A full subtractor has 3 inputs and 2 outputs
- The truth table of the full- subtractor can be drawn with inputs A,B and C with outputs D and Bout.



A	B	C	D	Bout
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

FULL SUBTRACTOR

- From the truth table we can write the Boolean equation for the D and Bout

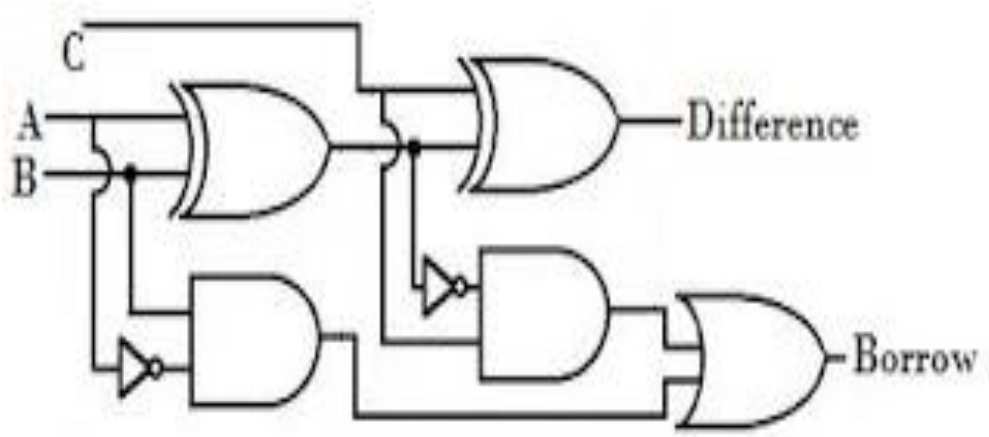
$$D = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

$$B_{out} = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + ABC$$

- Simplify using Boolean Algebra and K-map, we get

$$D = A \oplus B \oplus C$$

$$B_{out} = \overline{(A \oplus B)}C + \overline{A}B$$



COMBINED HALF ADDER/SUBTRACTOR

- X = Control signal (not involved in arithematic operation)
  - 0-add 1-subtract
- A and B inputs
- S/D and Cout/Bout outputs

	X	A	B	S/D	Cout/Bout
X=0 ADD	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	0
	0	1	1	0	1
X=1 SUBTRACT	1	0	0	0	0
	1	0	1	1	1
	1	1	0	1	0
	1	1	1	0	0

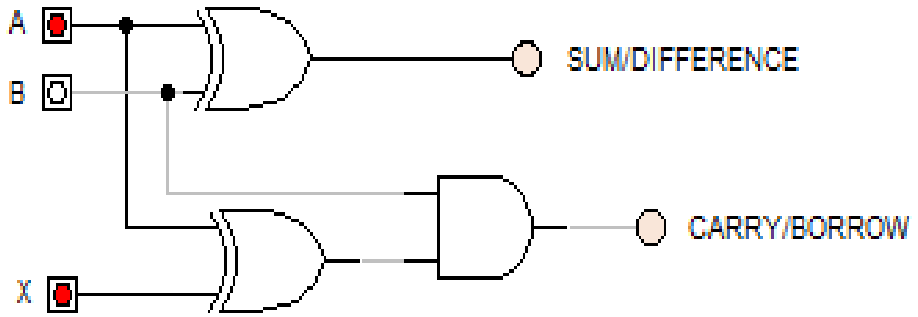
COMBINED HALF ADDER/SUBTRACTOR

$$S / D = \overline{X}\,\overline{A}B + \overline{X}A\overline{B} + X\overline{A}B + XA\overline{B}$$

$$Cout / Bout = \overline{X}AB + X\overline{A}B$$

Simplifying,

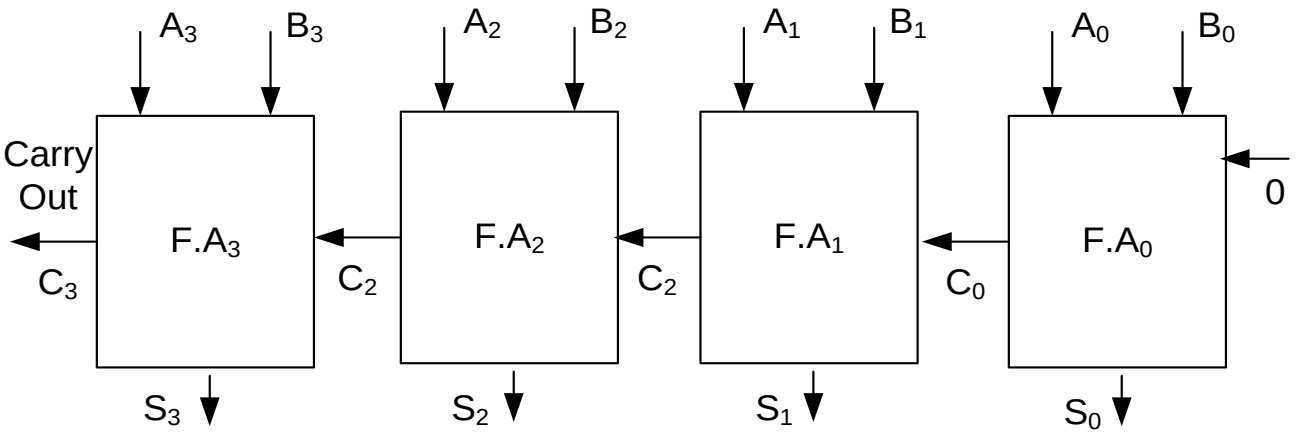
$$S / D = A \oplus B \qquad C_{out} / B_{out} = (A \oplus X)B$$



PARALLEL ADDER-4 BIT ADDER

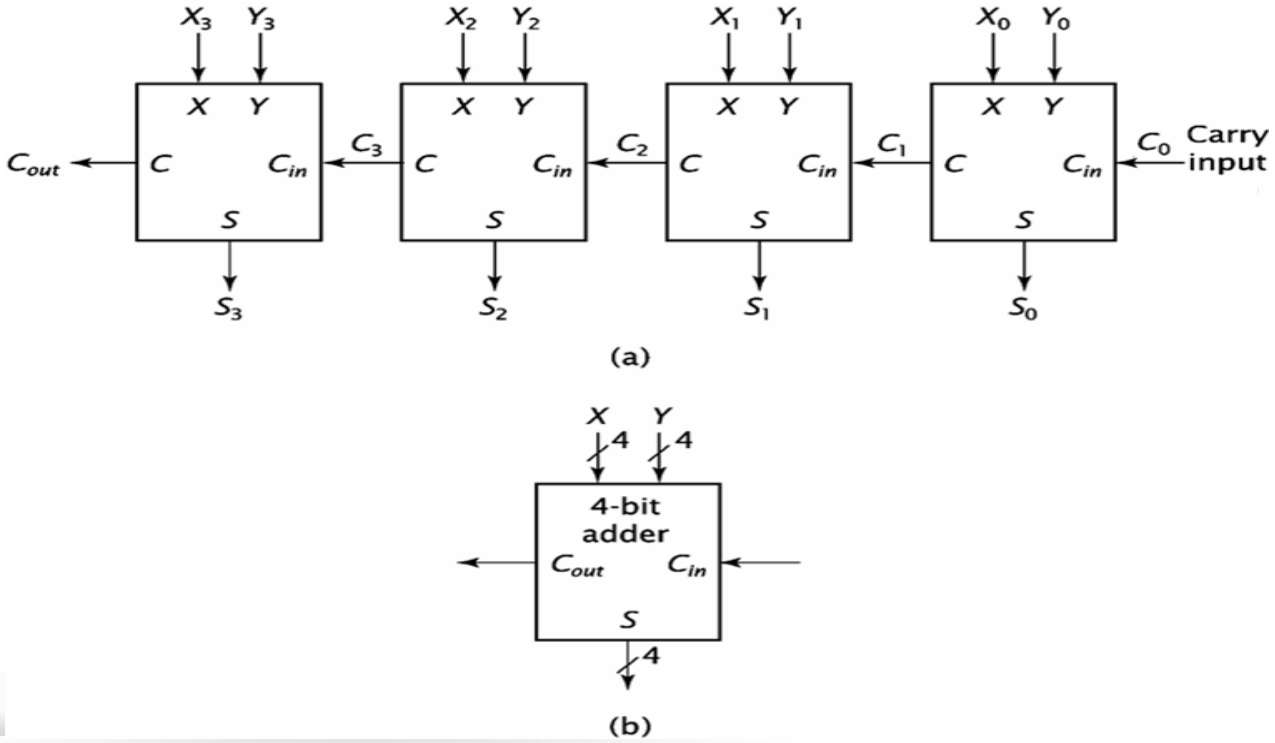
- This circuit is sometimes referred to as a ripple-through adder
- C0 ripples through four two-level logic circuits and hence the sum cannot be completed until eight gate delays
- For this kind of adder, the maximum delay is directly proportional to the number of stages n.

C	1	1	1	0
A	0	1	0	1
B	0	1	1	1
S	1	1	0	0

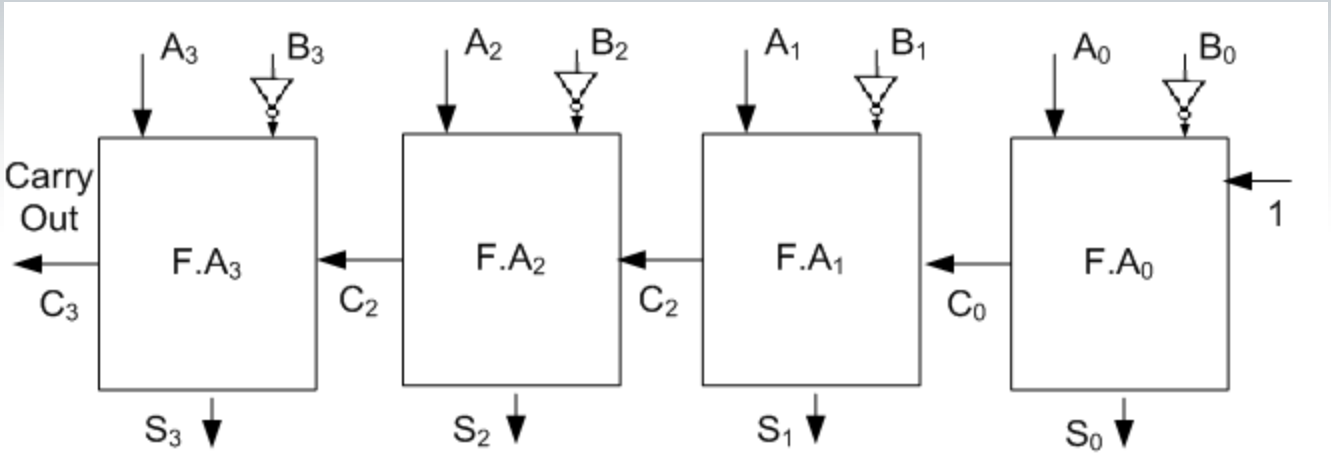


n-bit adder

- With the carry input, full adders can be cascaded to produce an n bit adder by connecting output C from one adder to input Cin of the next adder
- Such an adder is called Ripple adder (because the bits ripple through the adder).

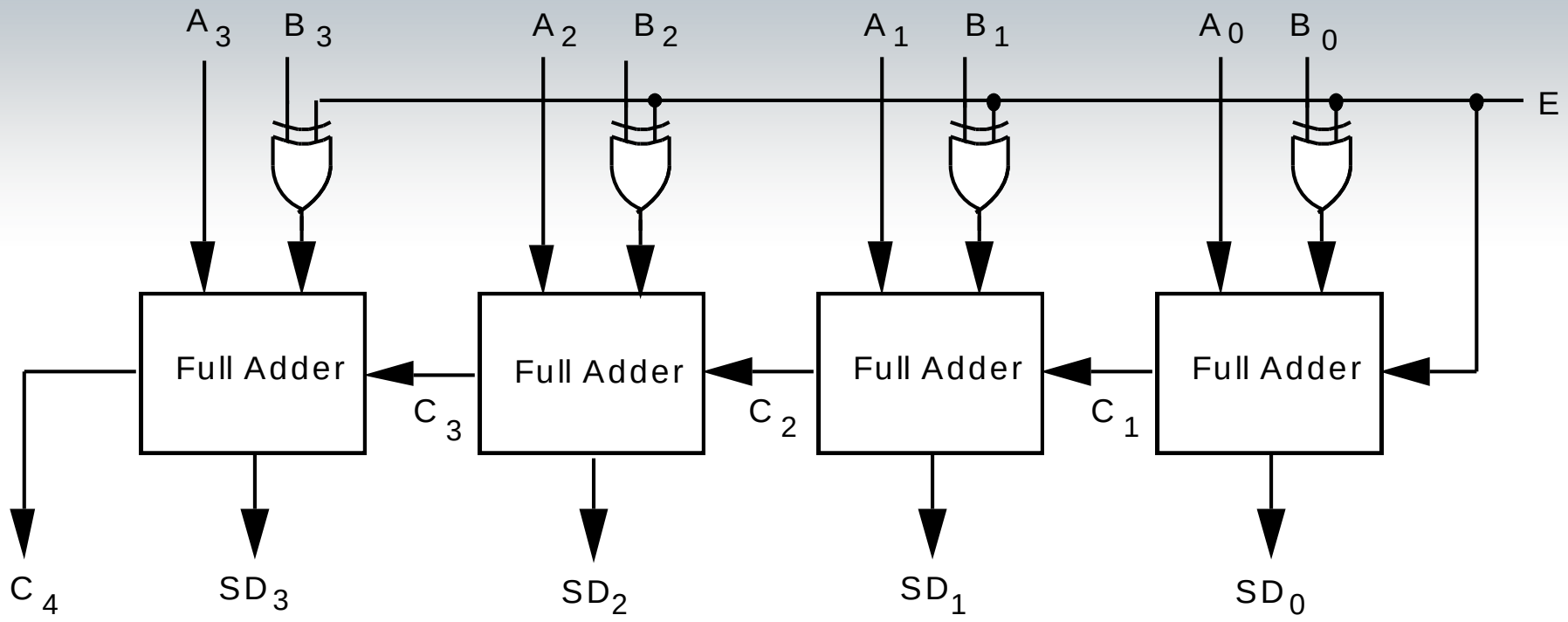


n-bit adder as a parallel subtractor



	C_2	C_1	C_0	
	A_3	A_2	A_1	A_0
	$\overline{B_3}$	$\overline{B_2}$	$\overline{B_1}$	$\overline{B_0}$
			+	1
C_3	S_3	S_2	S_1	S_0

COMBINED N-BIT ADDER/SUBTRACTOR

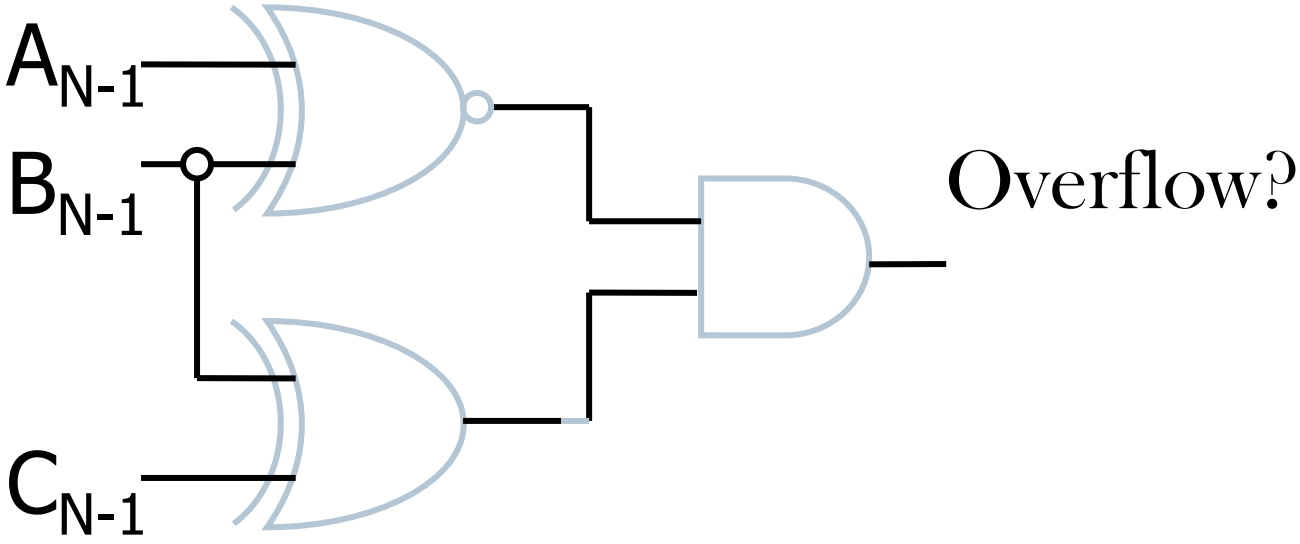


E = 0: 4-bit Adder

E = 1: 4-bit Subtractor

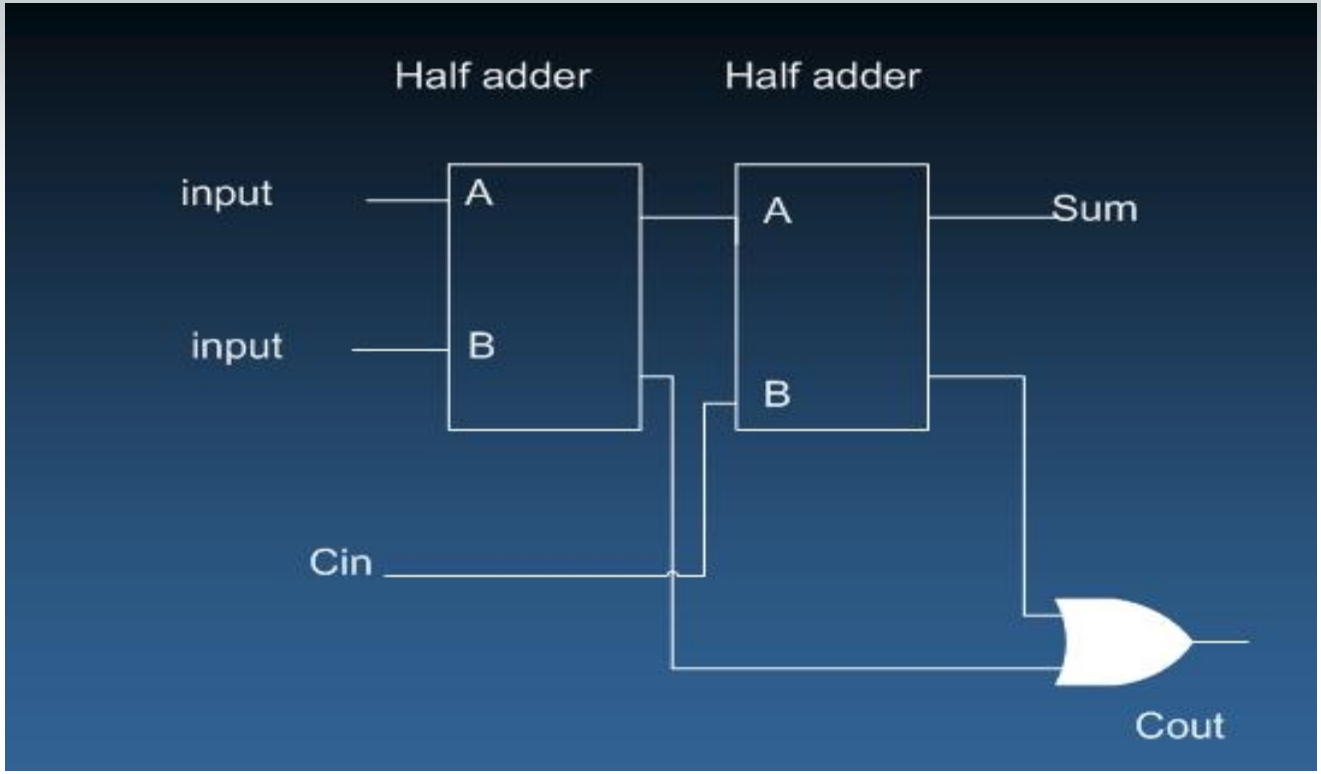
Overflow in Two's Complement Addition

- When two values of the same signs are added:
  - Result won't fit in the number of bits provided
  - Result has the opposite sign.



Assumes an N-bit adder, with bit N-1 the MSB

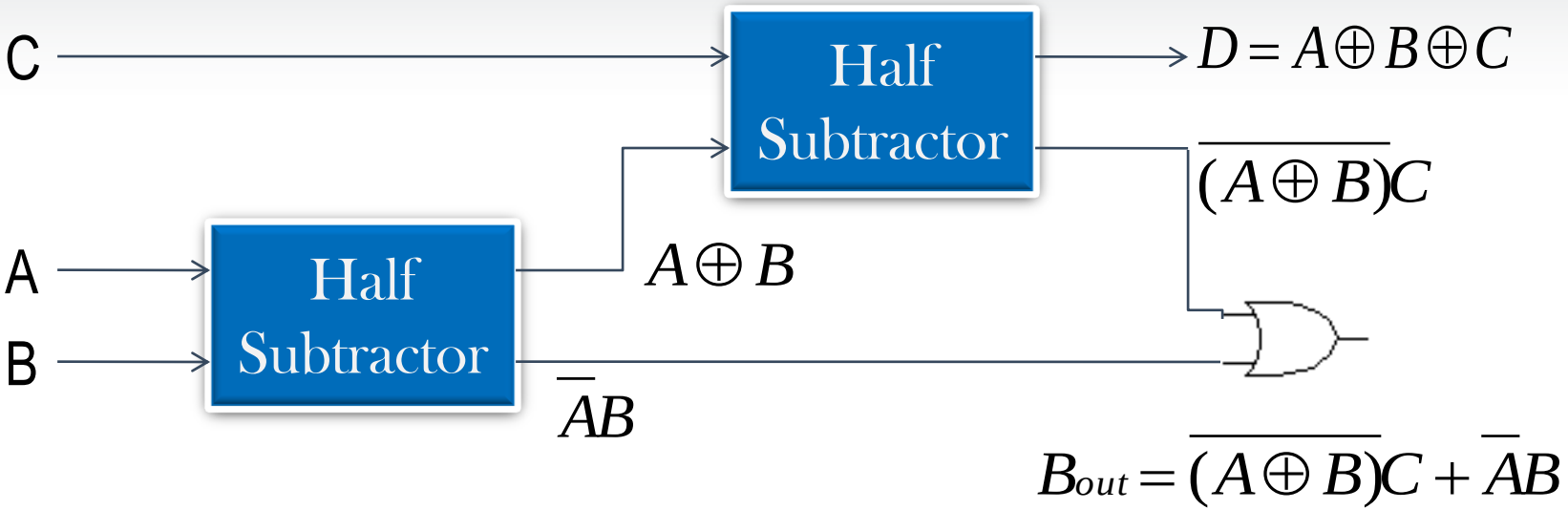
FULL ADDER USING HALF ADDERS



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

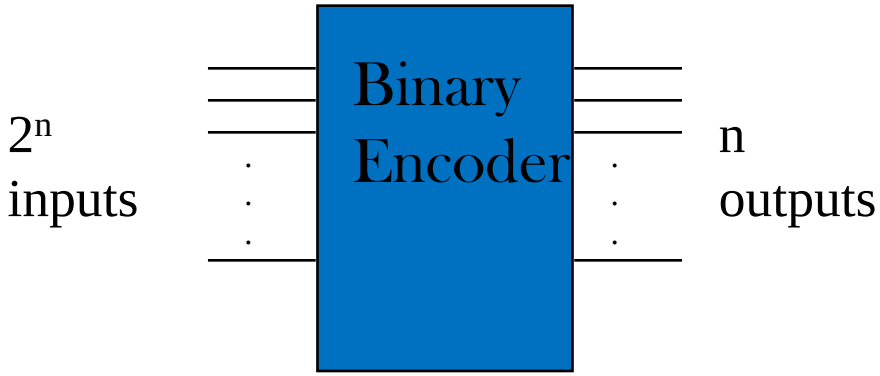
$$C_{out} = (A \oplus B)C_{in} + AB$$

FULL SUBTRACTOR USING HALF SUBTRACTORS



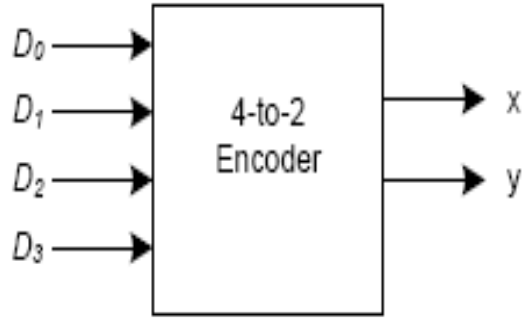
ENCODERS

- It receives 2^n inputs and outputs a n bit binary value corresponding to the one input that has a value of 1.
- Only one input will be active at a time
- Useful for compressing data
- Can be developed using AND/OR gates
- are used in various components such as keyboards



4 X 2 ENCODER

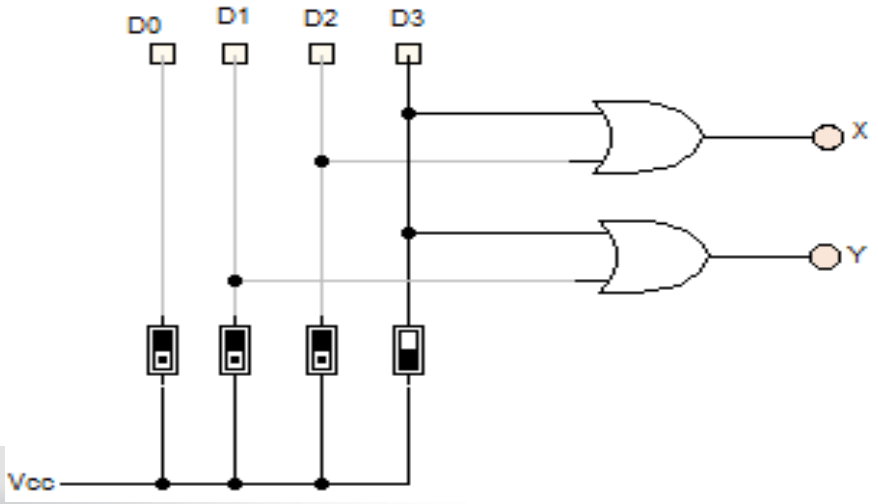
D_0	D_1	D_2	D_3	x	y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



From the truth table, we can express the outputs as:

$$x = D_2 + D_3$$

$$y = D_1 + D_3$$



8 X 3 ENCODER

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	
0	0	0	1	0	0	0	0	0	1	
0	0	0	0	1	0	0	0	1	0	
0	0	0	0	0	1	0	0	1	0	
0	0	0	0	0	0	1	0	1	1	
0	0	0	0	0	0	0	1	1	1	

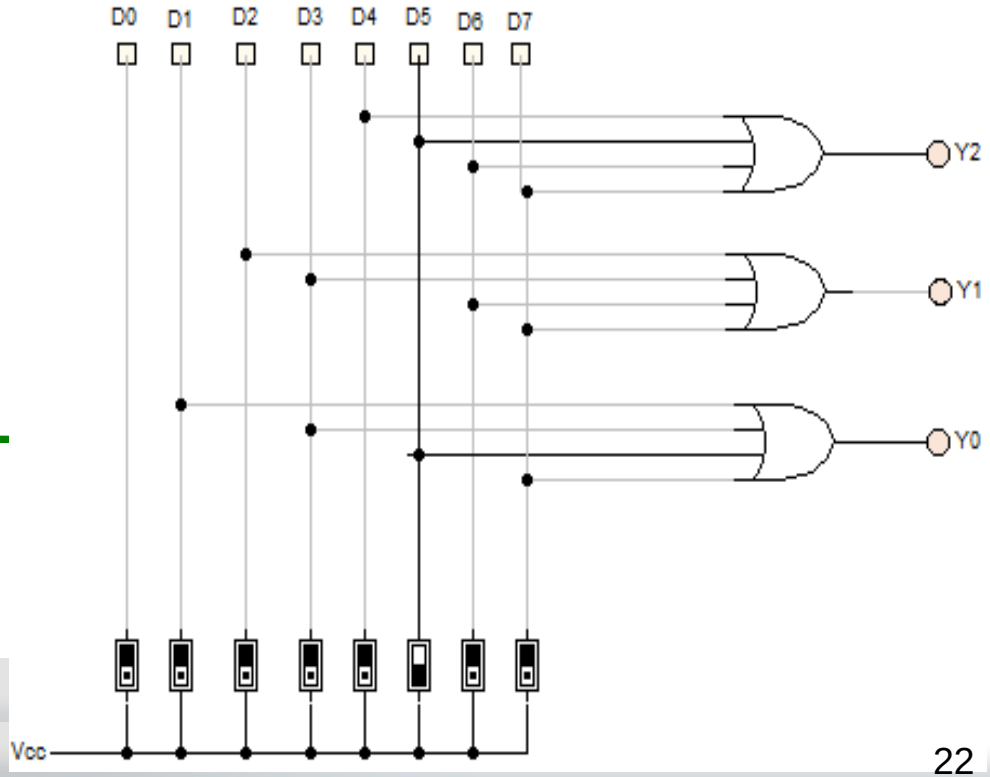
At any one time, only
one input line has a value of 1.

From the truth table, we can express the outputs as:

$$y_2 = D_4 + D_5 + D_6 + D_7$$

$$y_1 = D_2 + D_3 + D_6 + D_7$$

$$y_0 = D_1 + D_3 + D_5 + D_7$$

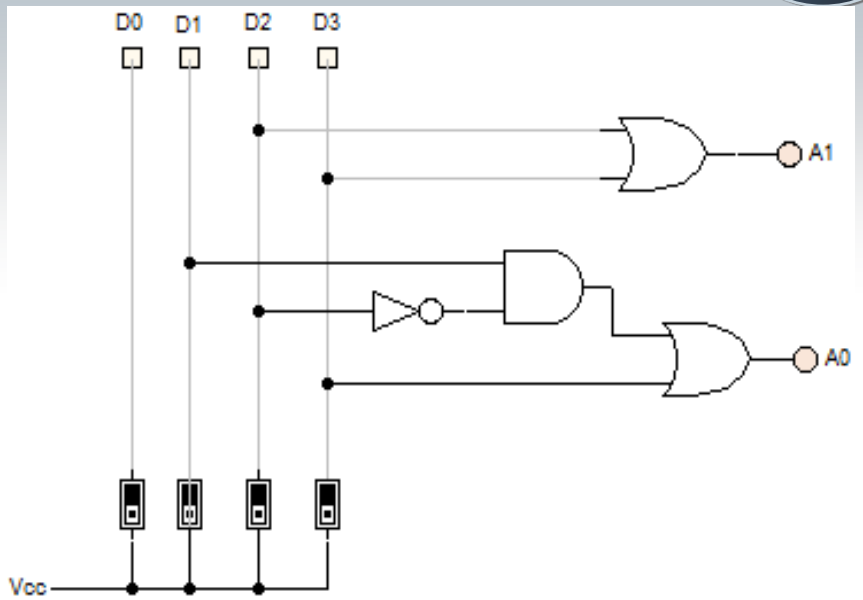


PRIORITY ENCODERS

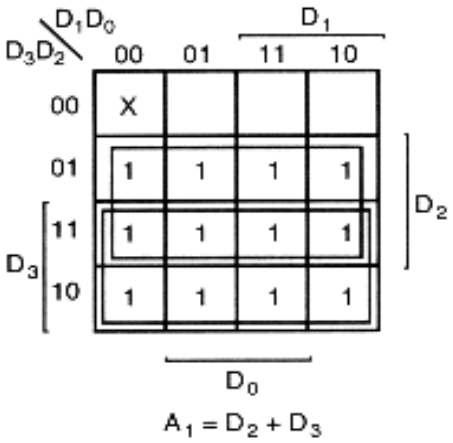
- A priority encoder works just a regular encoder, with one exception: whenever one or more input is active, the output is set to correspond to the highest active input
- Assign priorities to the inputs
- When more than one input are asserted, the output generates the code of the input with the highest priority
- For example, in a 4-to-2 encoder, if inputs 0,1 and 3 are active, then the y1 y0 = 1 1 output is set, corresponding to the input 3.

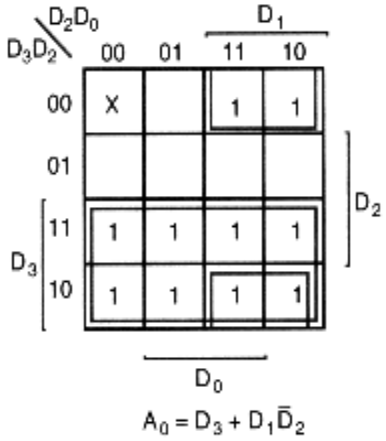
4 X 2 PRIORITY ENCODER

Inputs				Outputs	
D_3	D_2	D_1	D_0	A_1	A_0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1



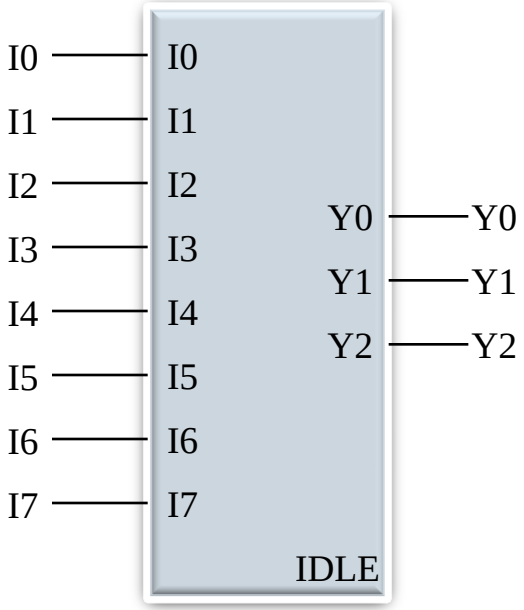
This can be minimised using the K-map as follows:





8-TO-3 PRIORITY ENCODER

- What if more than one input line has a value of 1?
- Ignore "lower priority" inputs.
- Idle indicates that no input is a 1.
- Priority encoders rank inputs and encode the highest priority input



Inputs								Outputs			
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0	Idle
0	0	0	0	0	0	0	0	x	x	x	1
1	0	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	X	X	1	1	1	1	0

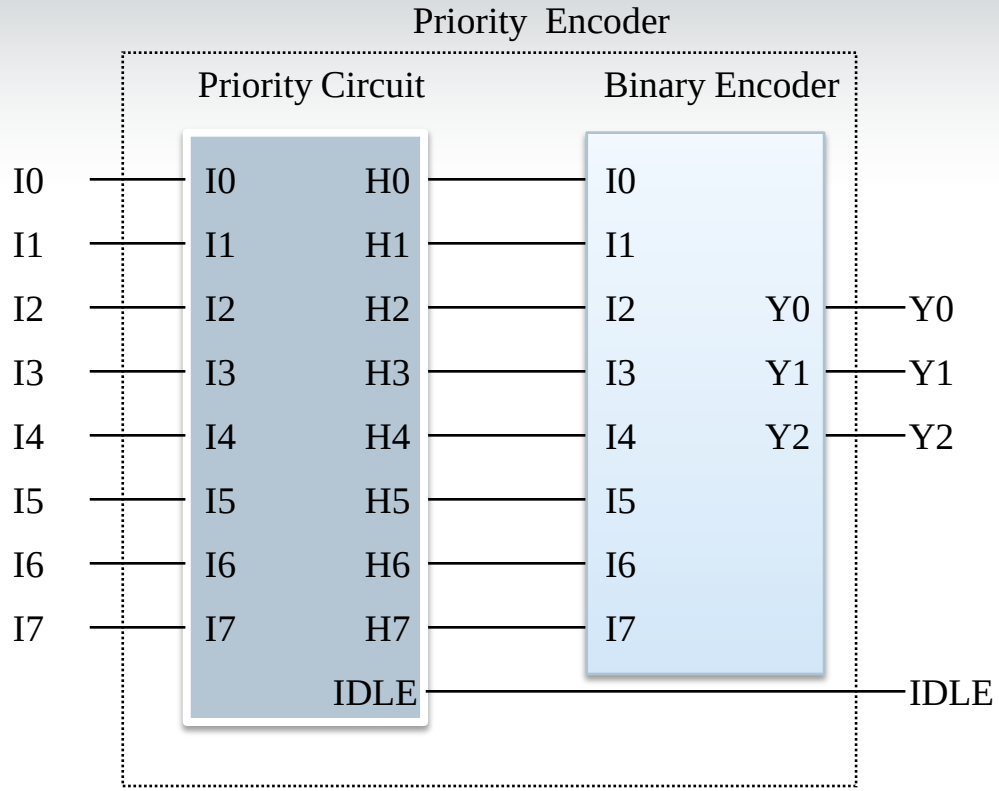
8-TO-3 PRIORITY ENCODER

Priority Encoder :

H7=I7 (Highest Priority)
H6=I6.I7'
H5=I5.I6'.I7'
H4=I4.I5'.I6'.I7'
H3=I3.I4'.I5'.I6'.I7'
H2=I2.I3'.I4'.I5'.I6'.I7'
H1=I1. I2'.I3'.I4'.I5'.I6'.I7'
H0=I0.I1'. I2'.I3'.I4'.I5'.I6'.I7'
IDLE= I0'.I1'. I2'.I3'.I4'.I5'.I6'.I7'

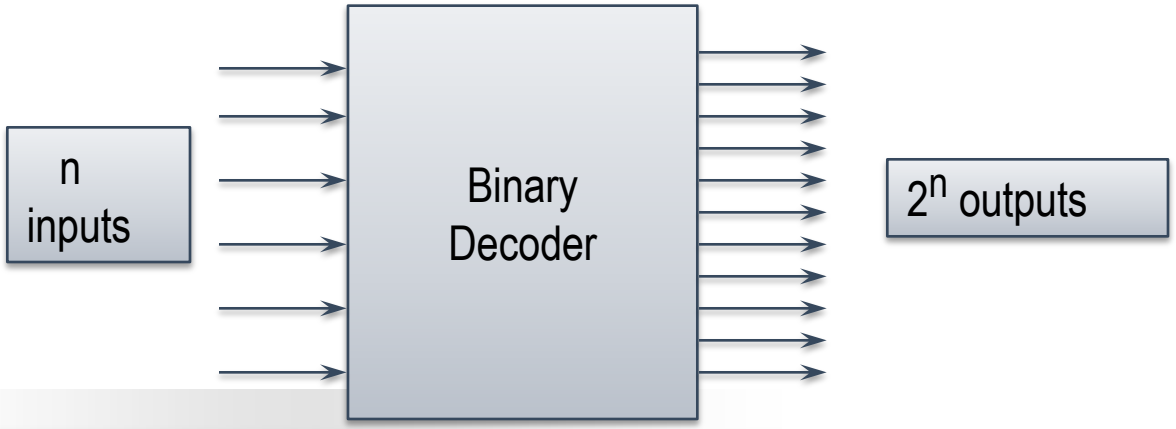
Encoder

Y0 = I1 + I3 + I5 + I7
Y1 = I2 + I3 + I6 + I7
Y2 = I4 + I5 + I6 + I7

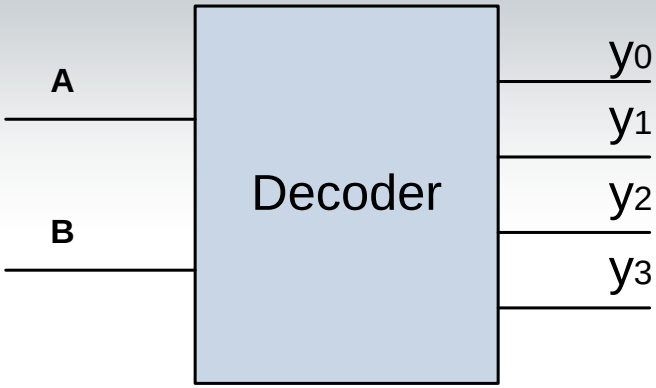


DECODER

- A decoder accepts a binary value as input and decodes it.
- It has n inputs and 2^n outputs, numbered from 0 to 2^n -1.
- Each output represents one minterm of the inputs
- The output corresponding to the value of the n inputs is activated
- Only one output is a 1 for any given input
- For example, a decoder with three inputs and eight outputs will activate output 6 whenever the input values are 110.



2 X 4 DECODER



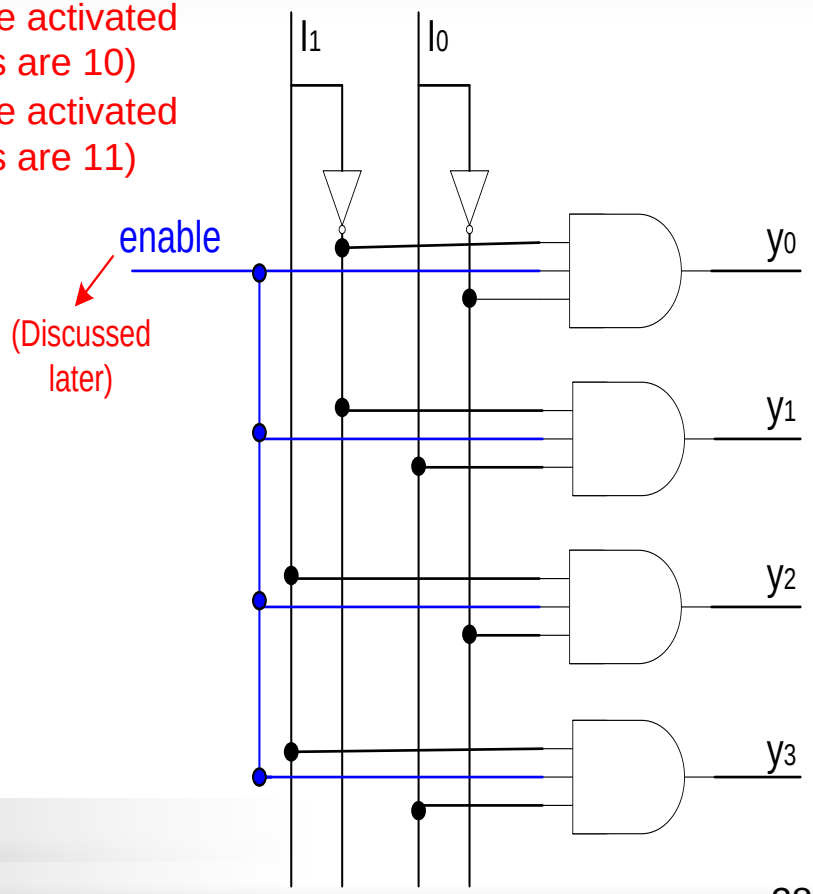
(This output will be activated when the inputs are 00)

(This output will be activated when the inputs are 01)

(This output will be activated when the inputs are 10)

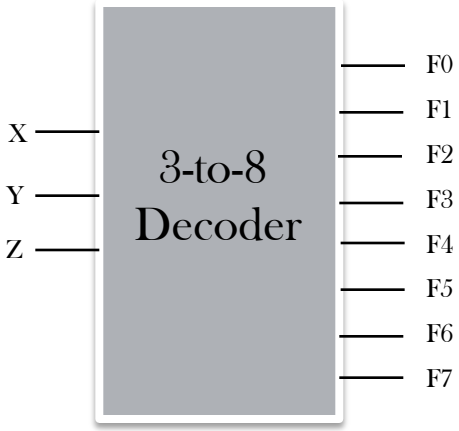
(This output will be activated when the inputs are 11)

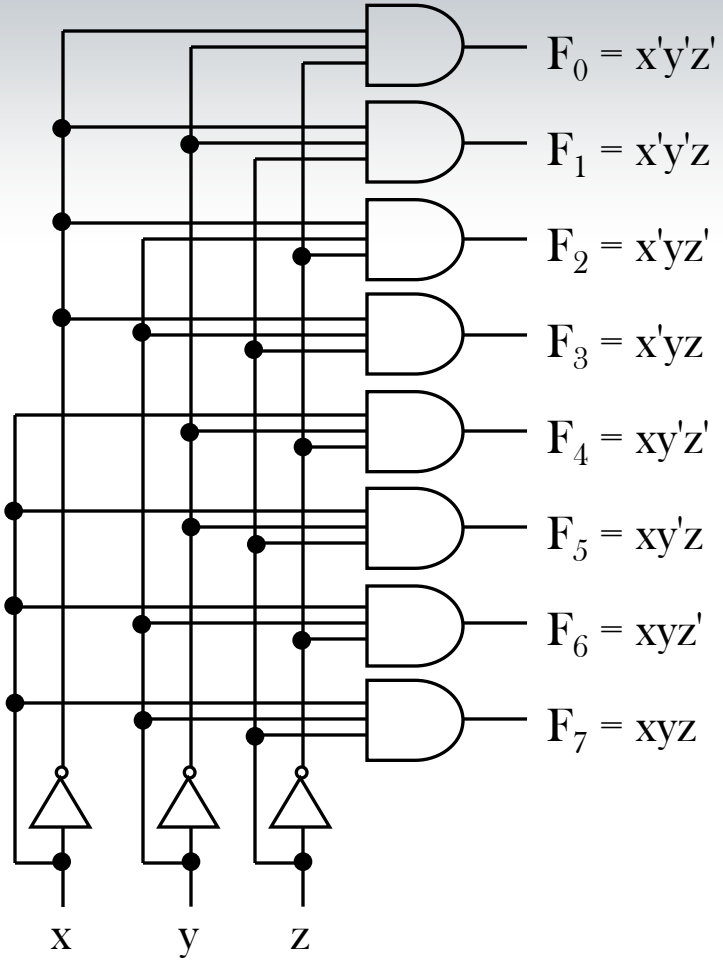
en	A	B	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	–	–	0	0	0	0



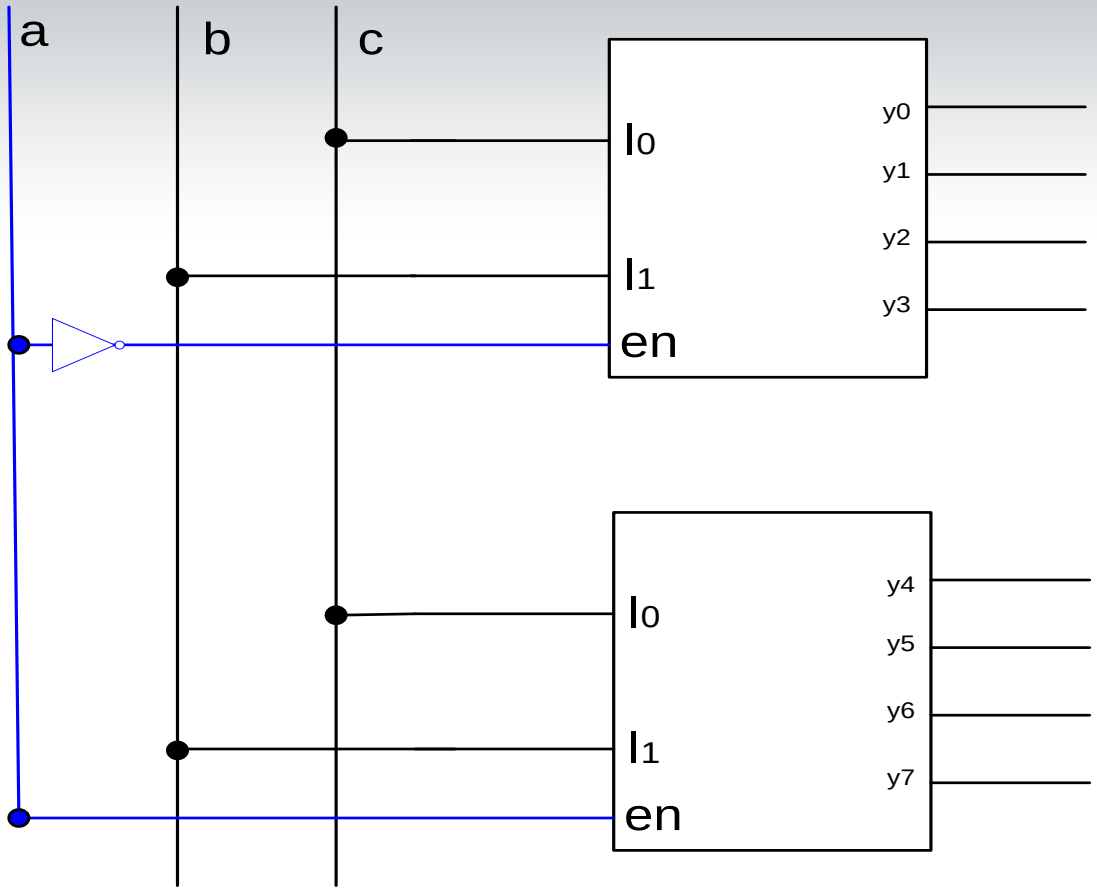
3 X 8 DECODER

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



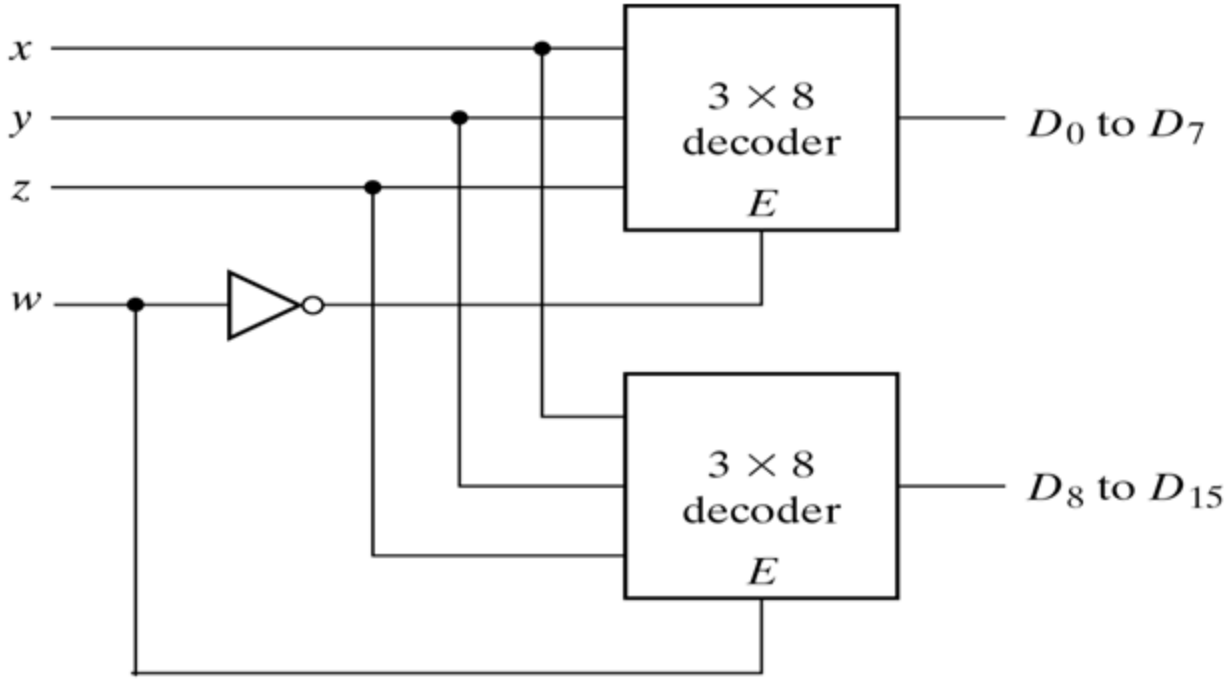


CONSTRUCTING 3 X 8 USING 2 X 4 DECODER



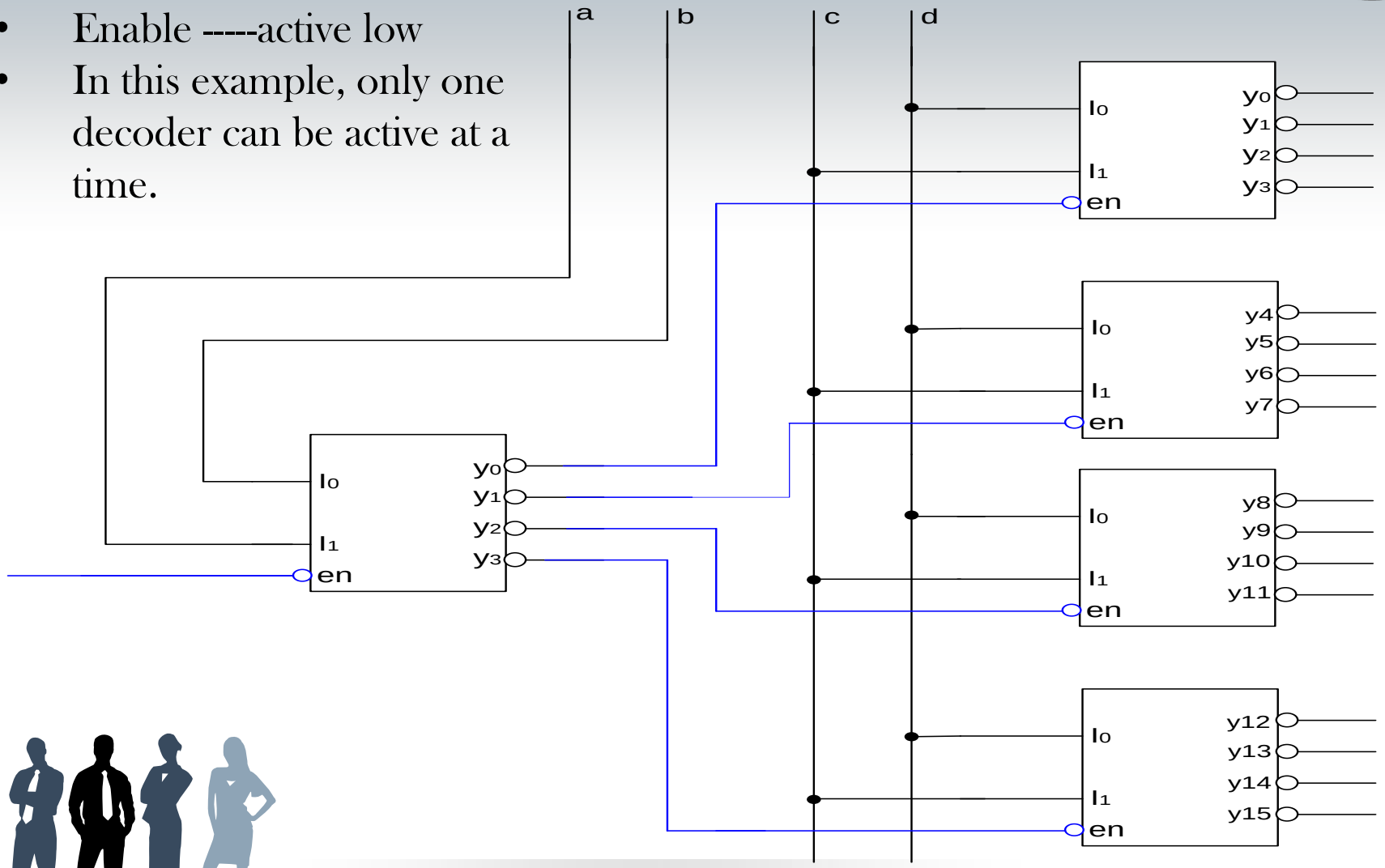
3 TO 8 DECODERS TO MAKE 4 TO 16 DECODER

- Enable -----active high
- In this example, only one decoder can be active at a time.
- x, y, z ----input
- w ----strobe or enable



CONSTRUCTING 4 X 16 USING 2 X 4 DECODER

- Enable -----active low
- In this example, only one decoder can be active at a time.



IMPLEMENTING FUNCTIONS USING DECODERS

- Any n-variable logic function can be implemented using a single n-to-2^n decoder to generate the minterms
  - OR gate forms the sum.
  - The output lines of the decoder corresponding to the minterms of the function are used as inputs to the or gate.
- Any combinational circuit with *n* inputs and *m* outputs can be implemented with an n-to-2^n decoder with *m* OR gates.
- Suitable when a circuit has many outputs, and each output function is expressed with few minterms

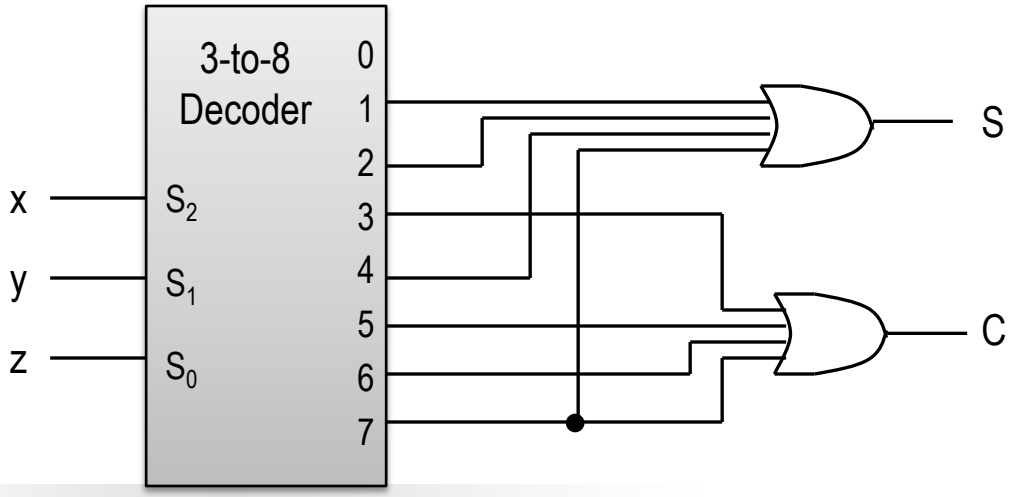
IMPLEMENTING FUNCTIONS USING DECODERS

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

Example: Full adder

$S(x, y, z) = S(1,2,4,7)$

$C(x, y, z) = S(3,5,6,7)$



BCD-TO-SEVEN-SEGMENT DECODER

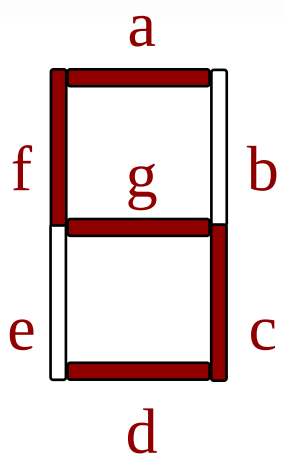
- Digital readouts on many digital products often use LED seven-segment displays.
- Each digit is created by lighting the appropriate segments. The segments are labeled a,b,c,d,e,f,g
- The decoder takes a BCD input and outputs the correct code for the seven-segment display.
- Input: A 4-bit binary value that is a BCD coded input.
- Outputs: 7 bits, a through g for each of the segments of the display.
- Operation: Decode the input to activate the correct segments

(a) Segment designation

(b) Numeric designation for display

LISTING THE SEGMENTS

0	a,b,c,d,e,f
1	b,c
2	a,b,d,e,g
3	a,b,c,d,g
4	b,c,f,g
5	a,c,d,f,g
6	a,c,d,e,f,g
7	a,b,c
8	a,b,c,d,e,f,g
9	a,b,c,d,f,g



TRUTH TABLE FOR BCD-TO-SEVEN-SEGMENT DECODER

BCD Input				Seven Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
all other inputs				x	x	x	x	x	x	x

- Fill in don't cares for *undefined* outputs.
  - Leads to a reduced implementation
- For these combinations of *undefined* inputs never to happen, fill '0'

WRITING EQUATIONS AND REDUCTIONS USING K-MAP

For segment "a" :

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_a = A + C + \overline{B}\,\overline{D} + BD$$

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a2} = A$$

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a4} = BD$$

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a1} = C$$

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a3} = \overline{B}\,\overline{D}$$

Similarly do for segment b, c, d, e, f and g.
Then draw logical diagram

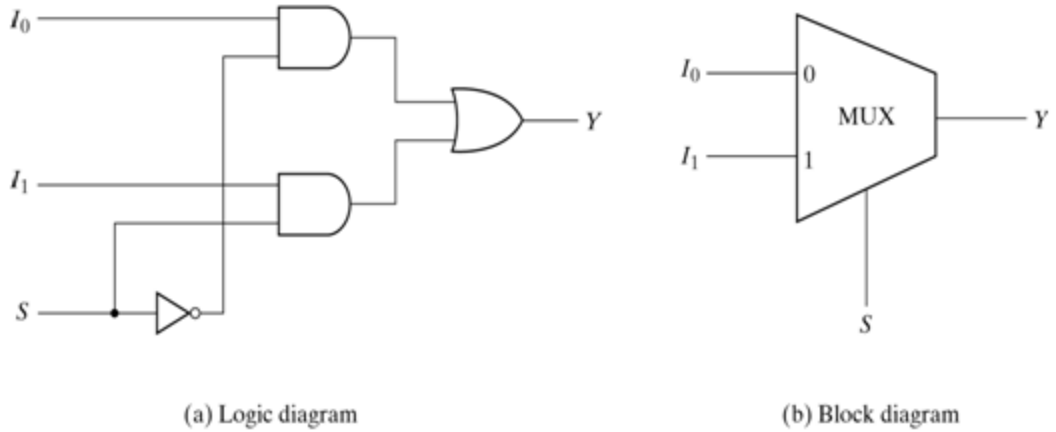
MULTIPLEXER

- It is a selector, it chooses one of its data inputs and passes it to the output according to some other selection inputs
- Select an input value with one or more select bits
- Use for transmitting data
- Allows for conditional transfer of data
- Sometimes called a mux
- Consider four binary data inputs as inputs of a multiplexer. Two select signals will determine which of the four inputs will be passed to the output.

For 2 to 1 MUX

- 2 Inputs I_0 AND I_1
- Select line S
- Output Y

S	Y
0	I_0
1	I_1

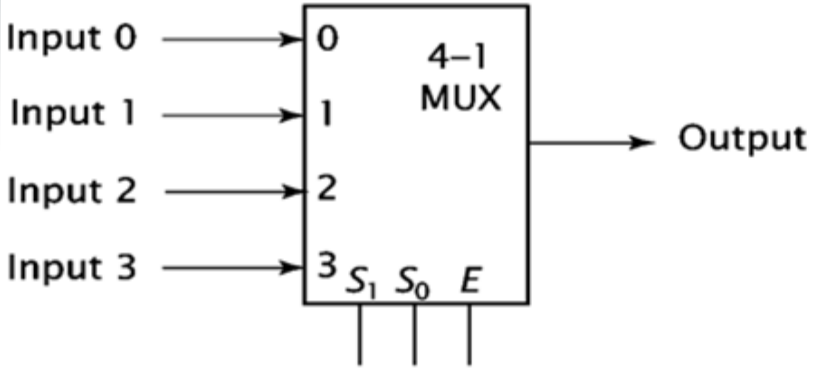


(a) Logic diagram (b) Block diagram

2-to-1-Line Multiplexer

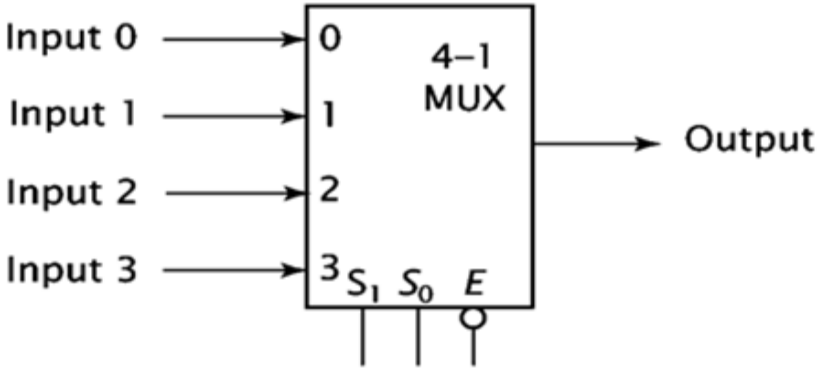
$$Y = \bar{S}I_0 + SI_1$$

4 TO 1 MULTIPLEXER



S_1	S_0	E	Output
X	X	0	Z
0	0	1	Input 0
0	1	1	Input 1
1	0	1	Input 2
1	1	1	Input 3

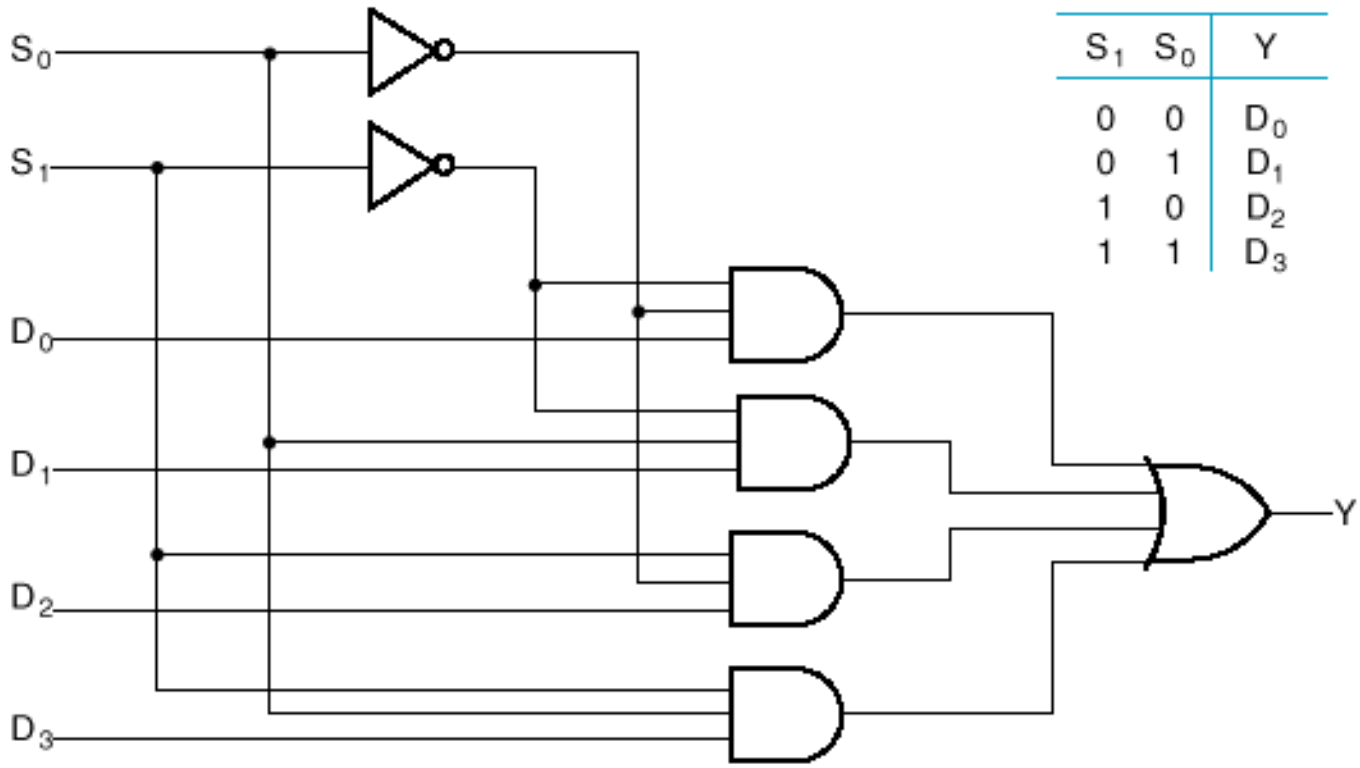
Multiplexer schematic representation with active high enable signal



S_1	S_0	E	Output
X	X	1	Z
0	0	0	Input 0
0	1	0	Input 1
1	0	0	Input 2
1	1	0	Input 3

Multiplexer schematic representation with active low enable signal

4 TO 1 MULTIPLEXER



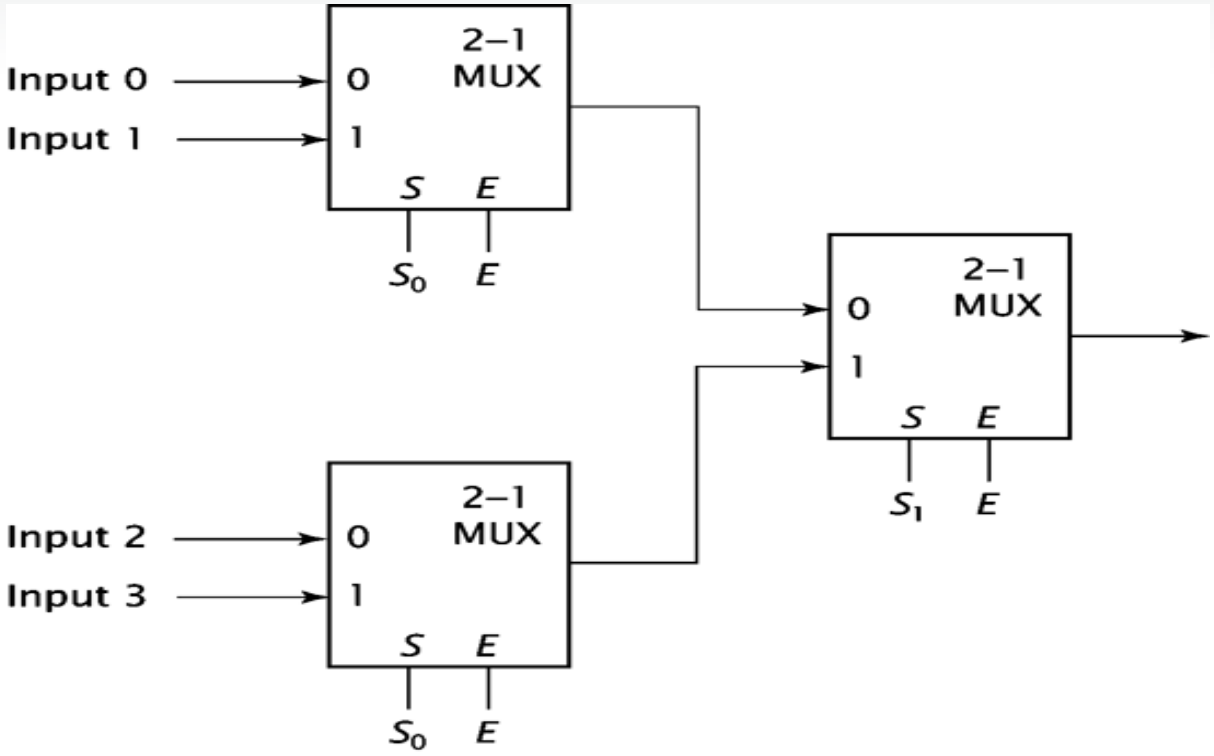
Function table

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$$Y = \overline{S_1}\,\overline{S_O} D_O + \overline{S_1} S_O D_1 + S_1 \overline{S}_o D_2 + S_1 S_O D_3$$

CONSTRUCTION OF 4 TO 1 USING 2 TO 1 MUX

- Multiplexers can be cascaded to select from a large number of inputs
- 4 to 1 multiplexer made of 2 to 1 multiplexers

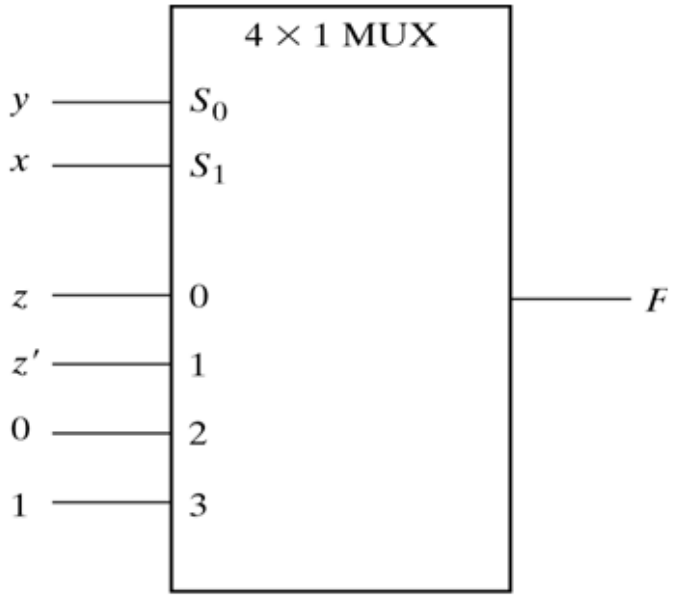


IMPLEMENTING BOOLEAN FUNCTION WITH MUX

- Connect input variables to select inputs of multiplexer *(n-1 for n variables)*
- Set data inputs to multiplexer equal to values of function for corresponding assignment of select variables
- Using a variable at data inputs reduces size of the multiplexer

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	

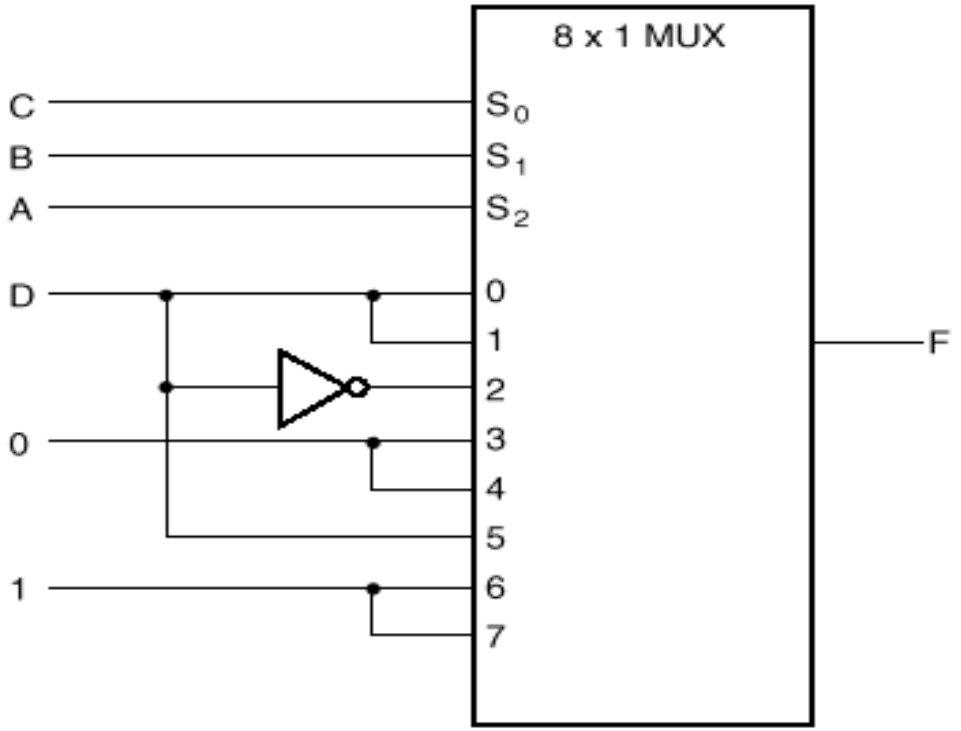
(a) Truth table



(b) Multiplexer implementation

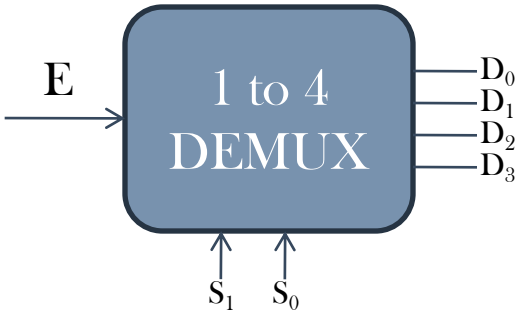
IMPLEMENTING A FOUR- INPUT FUNCTION WITH A MULTIPLEXER

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 = $\overline{D}$
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	

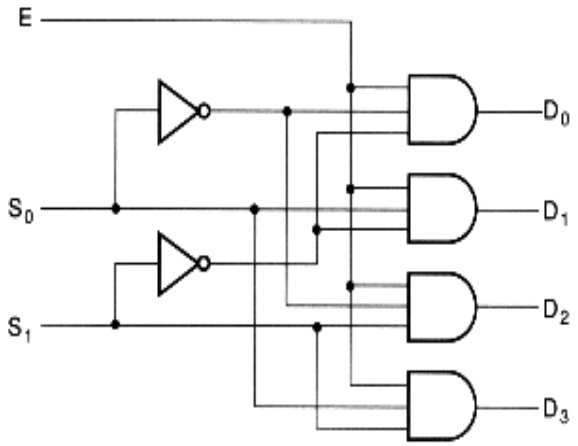


DE-MULTIPLEXER

- The de-multiplexer performs the inverse function of a multiplexer
- It receives information on one line and transmits its onto one of 2^n possible output lines.
- The selection is by *n* input select lines
- Note that a one to four multiplexer is really a two to four decoder with an additional enable input E which is the input data line.



S_1	S_0	D_0	D_1	D_2	D_3
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E



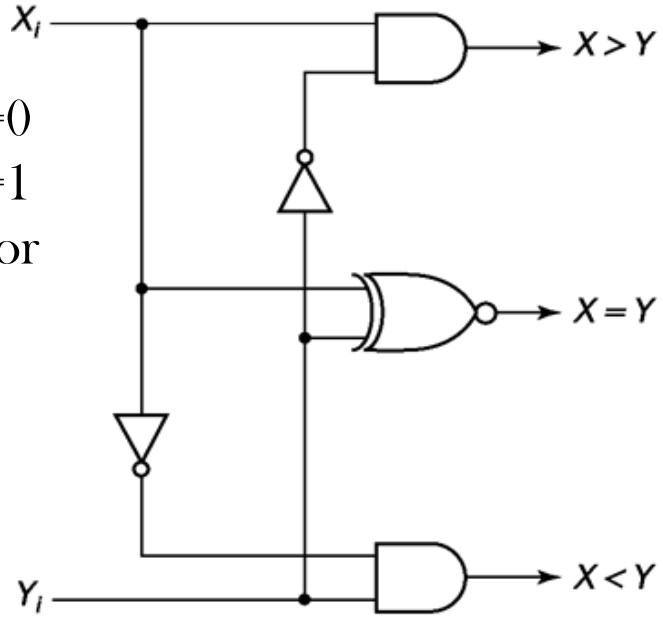
$$D_O = \overline{S_1}\,\overline{S_O}E \qquad D_2 = S_1\overline{S_O}E$$

$$D_1 = \overline{S_1}S_OE \qquad D_3 = S_1S_OE$$

COMPARATORS

- A comparator compares a two n-bit binary values to determine which is greater or if they are equal
- Consider the simple 1-bit comparator to illustrate the design
- It is possible to extend the design for multi-bit numbers

- X>Y only if Xi=1, Yi=0
- X<Y only if Xi=0, Yi=1
- X=Y only if Xi=Yi=0 or Xi=Yi=1



(a)

X_i	Y_i	X > Y	X = Y	X < Y
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

(b)

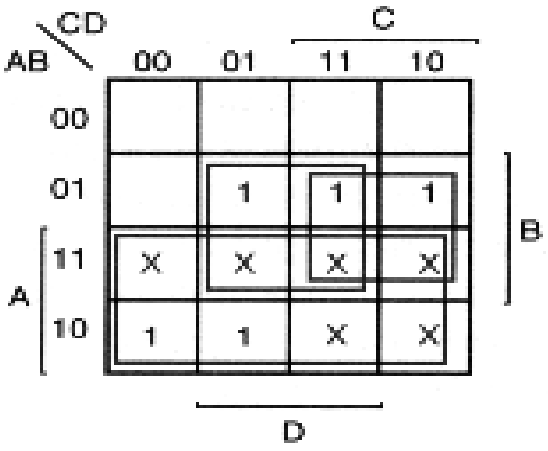
BCD TO EXCESS 3 CODE CONVERTER

- BCD is a code for the decimal digits 0-9
- Excess-3 is also a code for the decimal digits
- Inputs: a BCD input, A,B,C,D with A as the most significant bit and D as the least significant bit.
- Outputs: an Excess-3 output W,X,Y,Z that corresponds to the BCD input.
- Internal operation – circuit to do the conversion in combinational logic.
- Excess-3 code is easily formed by adding a binary 3 to the binary or BCD for the digit.
- There are 16 possible inputs for both BCD and Excess-3.
- It can be assumed that only valid BCD inputs will appear so the six combinations not used can be treated as don't cares.

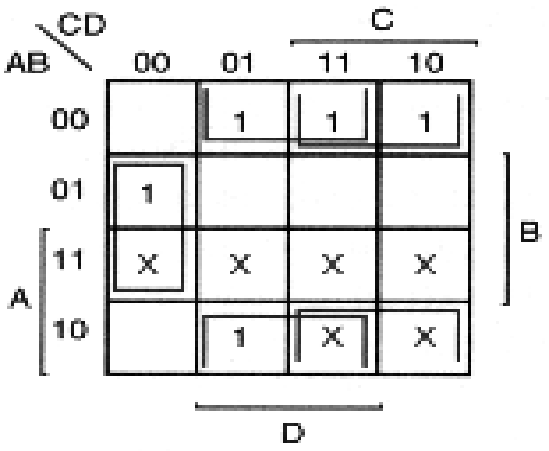
BCD TO EXCESS 3 CODE CONVERTER

Decimal	BCD input				Excess-3 output			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

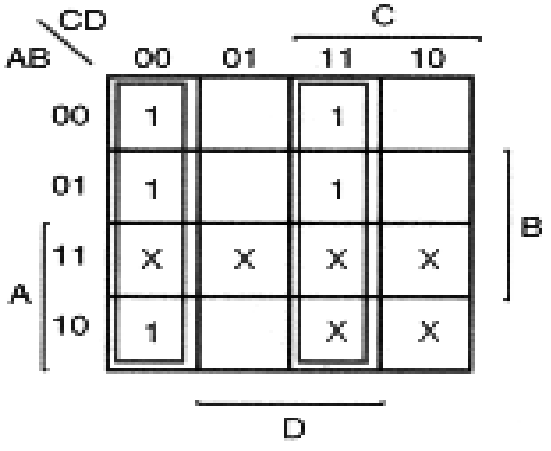
REDUCTION USING K-MAP



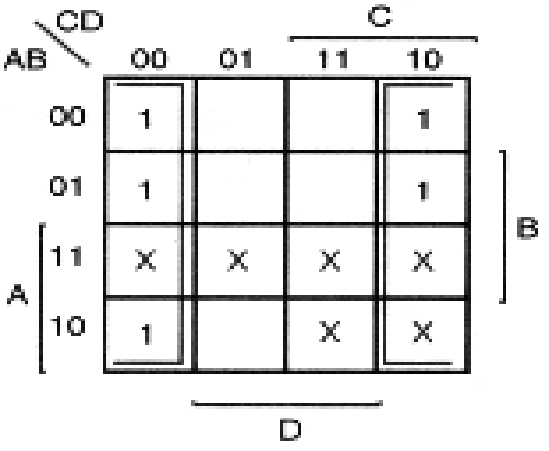
$W = A + BC + B\overline{D}$



$X = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D}$



$Y = CD + \overline{C}\,\overline{D}$



$Z = \overline{D}$

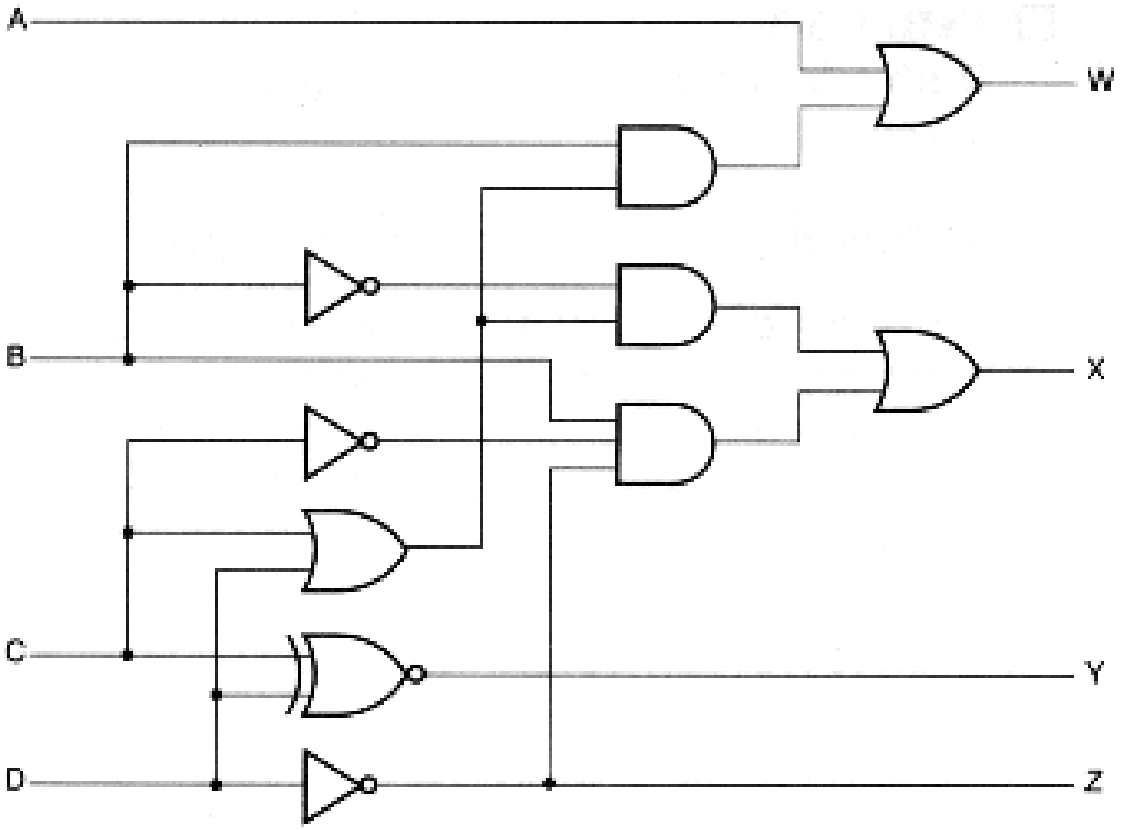
LOGICAL DIAGRAM

$$W = A + BC + BD = A + B(C + D)$$

$$X = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D} = \overline{B}(C + D) + B\overline{C}\,\overline{D}$$

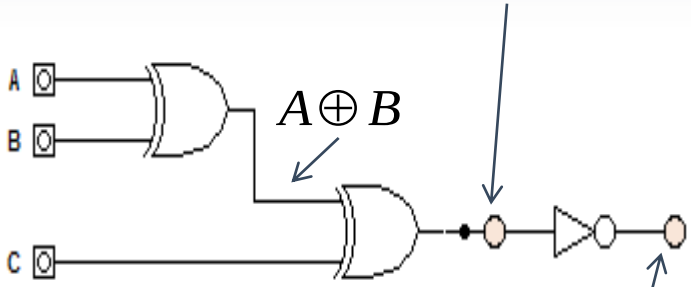
$$Y = CD + \overline{C}\,\overline{D} = \overline{C \oplus D}$$

$$Z = \overline{D}$$



EVEN & ODD PARITY GENERATOR FOR 3 BITS

$$EP = A \oplus B \oplus C$$



$$OP = \overline{A \oplus B \oplus C}$$

A	B	C	EP (Even Parity)	OP (Odd Parity)
0	0	0	0	1
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	0

BCD TO GRAY CONVERTER

Decimal	BCD Input				GRAY Output			
	A	B	C	D	W	X	Y	Z
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	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10-15	all other inputs				X	X	X	X

WRITING EQUATIONS AND REDUCTIONS USING K-MAP

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11		X	X	X
10	X 1	1	X	X

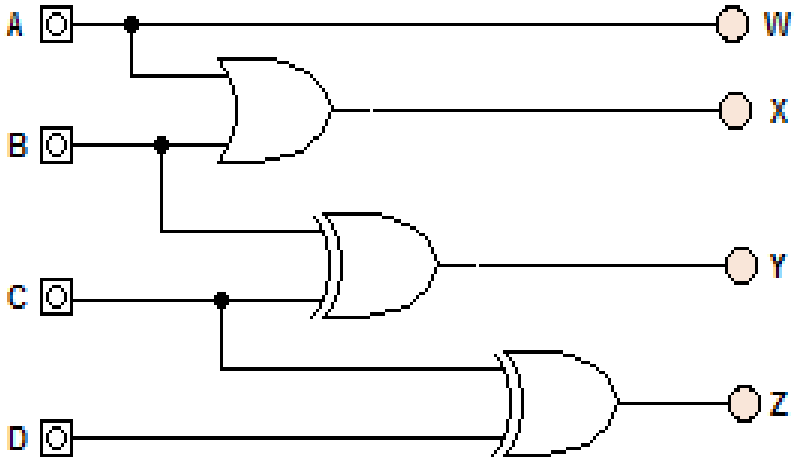
$$W = A$$

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$$X = A + B$$

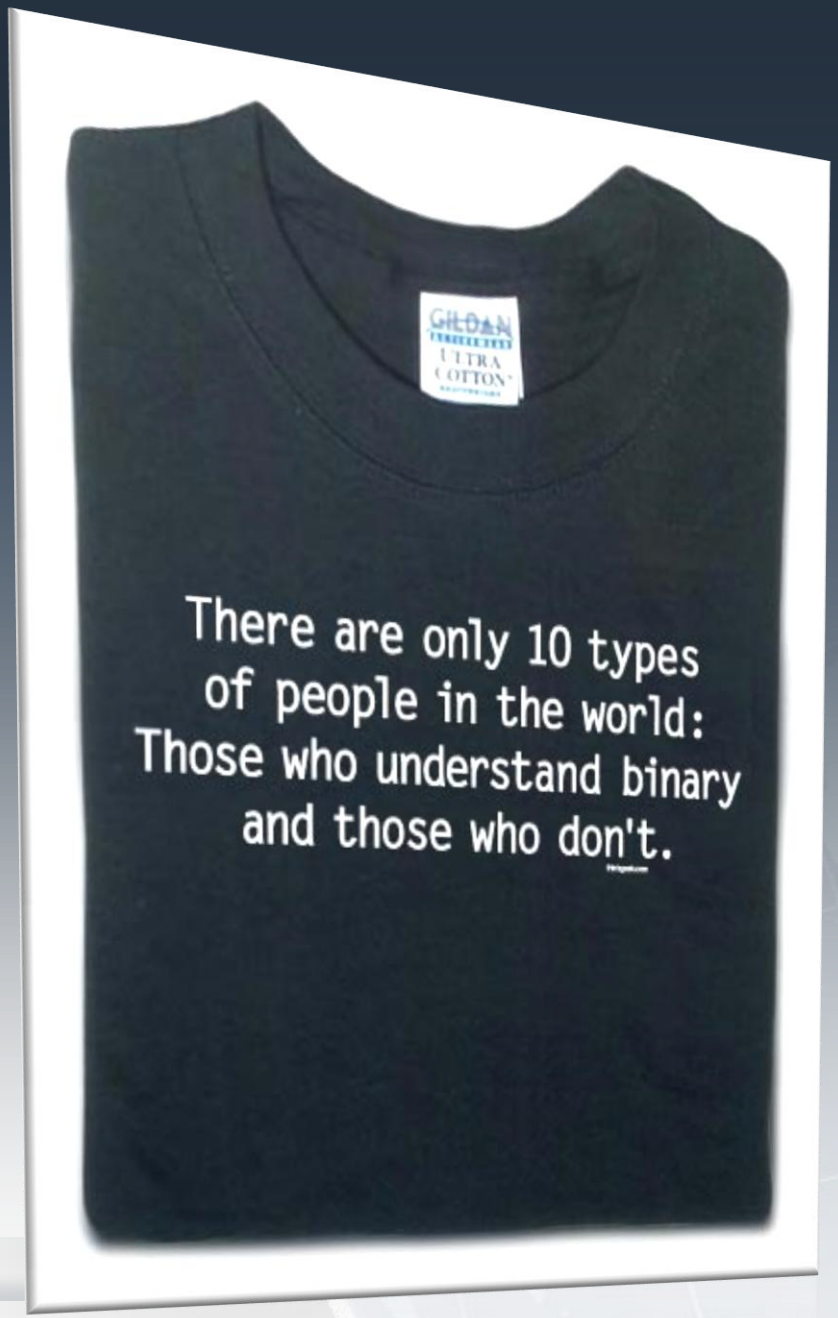
AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$$Y = B\overline{C} + \overline{B}C = B \oplus C$$



AB \ CD	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	X	X	X	X
10	0	1	X	X

$$Z = \overline{C}D + C\overline{D} = C \oplus D$$


GILDAN
ULTRA COTTON
There are only 10 types
of people in the world:
Those who understand binary
and those who don't.