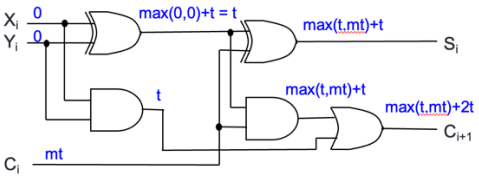
Logic Gate and Circuit Design

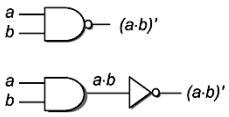
Circuit Delays: Given a logic gate with delay t. If inputs are stable at times t_1, t_2, …, t_n, then the earliest time in which the output will be stable is $\max(t_1, t_2, \ldots, t_n) + t$



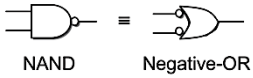
Complete Set of Logic: set {AND, OR, NOT}, since AND/OR/NOT gates are sufficient for building any Boolean function.

Universal Gates: NAND and NOR, **self-sufficient**, as any logic circuit may be built with only NAND gates (though you will probably need many of them) or NOR gates.

NAND Gate: $(a \cdot b)' = a' + b'$

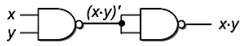


a	b	(a·b)'
0	0	1
0	1	1
1	0	1
1	1	0

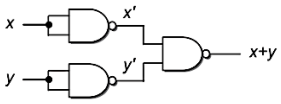


Implement NOT/AND/OR using only NAND gates

AND: $((x \cdot y)' \cdot (x \cdot y)')' = ((x \cdot y)')' = x \cdot y$

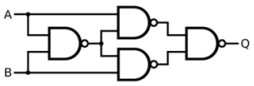


OR: $((x \cdot x)' \cdot (y \cdot y)')' = (x' \cdot y')' = x + y$

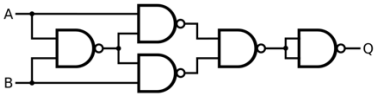


NOT: $(x \cdot x)' = x'$

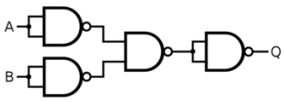
XOR:



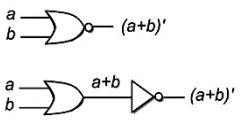
XNOR



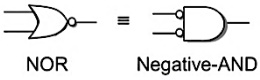
NOR



NOR Gate: $(a + b)' = a' \cdot b'$

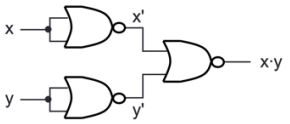


a	b	(a+b)'
0	0	1
0	1	0
1	0	0
1	1	0

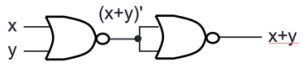


Implement NOT/AND/OR using only NOR gates

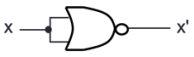
AND: $((x + x)' + (y + y)')' = (x' + y')' = x \cdot y$



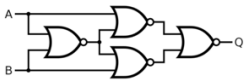
OR: $((x + y)' + (x + y)')' = ((x + y)')' = x + y$



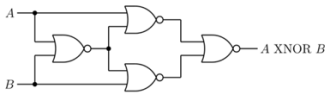
NOT: $(x + x)' = x'$



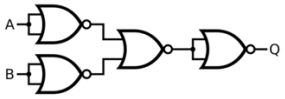
XOR:



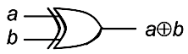
XNOR:



NOR:



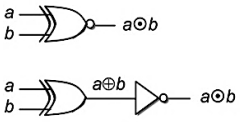
XOR Gate: $a \oplus b = a' \cdot b + a \cdot b'$



a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate:

$$a \odot b = (a \oplus b)' = (a' \cdot b + a \cdot b')' = (a + b') \cdot (a' + b)$$
$$= a \cdot b + a' \cdot b'$$

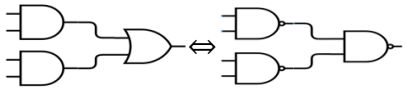


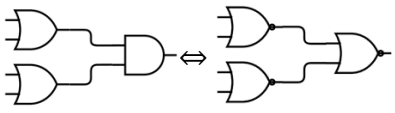
a	b	a⊙b
0	0	1
0	1	0
1	0	0
1	1	1

Substitute Idea (Replace AND-OR to NAND/OR-AND to NOR)

Basic: $\circ\!\!-\!\!-\!\!\circ \Longrightarrow (x')' = x \Longrightarrow$ —

Implementation:





SOP and POS

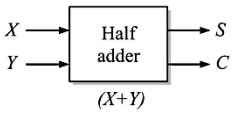
an SOP expression can be easily implemented using

- 2-level AND-OR circuit
- 2-level NAND circuit

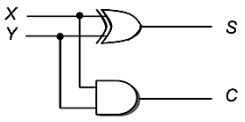
a POS expression can be easily implemented using

- 2-level OR-AND circuit
- 2-level NOR circuit

Half Adder: Only adds two bits

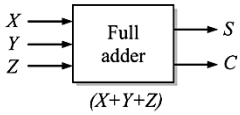


X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



- $C = X{\cdot}Y$
- $S = X'{\cdot}Y + X{\cdot}Y' = X \oplus Y$

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

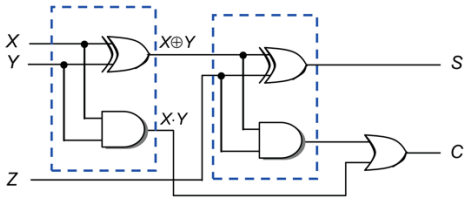
K-map for C.

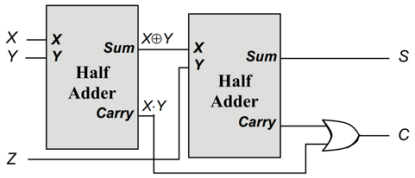
X \ YZ	00	01	11	10
0	0	0	1	0
1	0	1	1	1

K-map for S.

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

- $C = X{\cdot}Y + X{\cdot}Z + Y{\cdot}Z = X{\cdot}Y + (X \oplus Y){\cdot}Z$
- $S = X'{\cdot}Y'{\cdot}Z + X'{\cdot}Y{\cdot}Z' + X{\cdot}Y'{\cdot}Z' + X{\cdot}Y{\cdot}Z$
  $= X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$



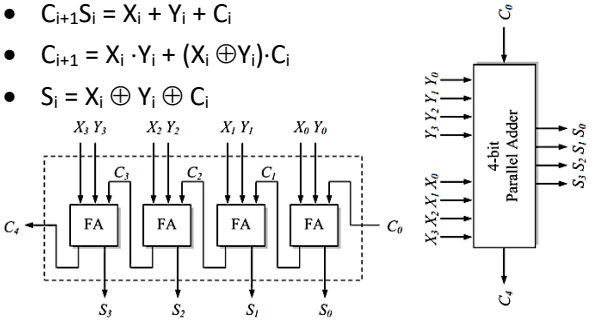


4-bit Parallel Adder:

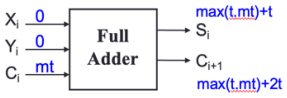
Input: two 4-bit numbers, carry-in

Output: carry-out bit C_4 and 4-bit sum $S_3S_2S_1S_0$

- $C_{i+1}S_i = X_i + Y_i + C_i$
- $C_{i+1} = X_i \cdot Y_i + (X_i \oplus Y_i){\cdot}C_i$
- $S_i = X_i \oplus Y_i \oplus C_i$



n-bit Parallel Adder Performance: Circuit Delay



$S_n = ((n-1) \times 2 + 2) \cdot t$

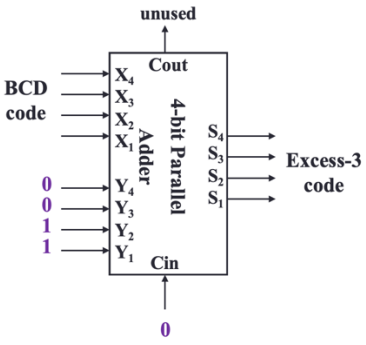
$C_{n+1} = ((n-1) \times 2 + 3) \cdot t$

Propagation delay of ripple-carry parallel adders is proportional to the number of bits it handles.

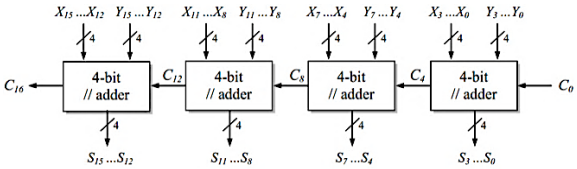
Maximum delay: $((n-1) \times 2 + 3) \cdot t$

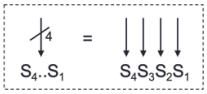
BCD to Excess-3 Converter: Excess-3 = BCD Code + 0011_2

	BCD				Excess-3			
	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
10	1	0	1	0	X	X	X	X
11	1	0	1	1	X	X	X	X
12	1	1	0	0	X	X	X	X
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X
15	1	1	1	1	X	X	X	X



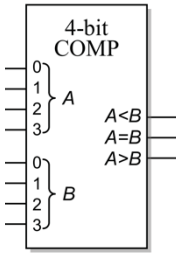
16-bit Parallel Adder





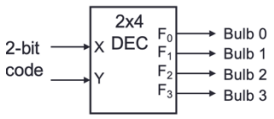
Magnitude Comparator

Compare two 4-bit number

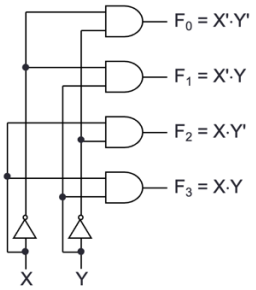


MSI Components

Decoder (2×4 decoder): n-bit code⟹2^n lines



X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Enable control signal:

E	X	Y	F_0	F_1	F_2	F_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	d	d	0	0	0	0

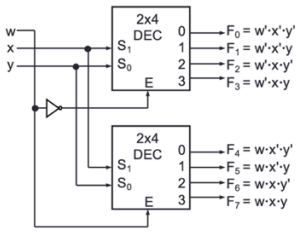
Decoder with **1-enable**

E'	X	Y	F_0	F_1	F_2	F_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	d	d	0	0	0	0

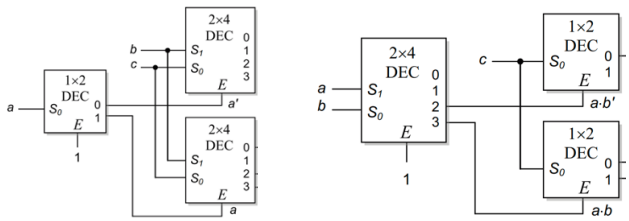
Decoder with **0-enable**

Constructing Larger Decoders (3×8 decoder)

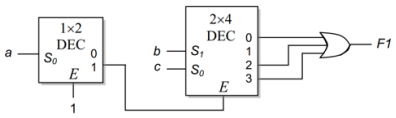
(a) Use NOT Gate and 2×4DEC



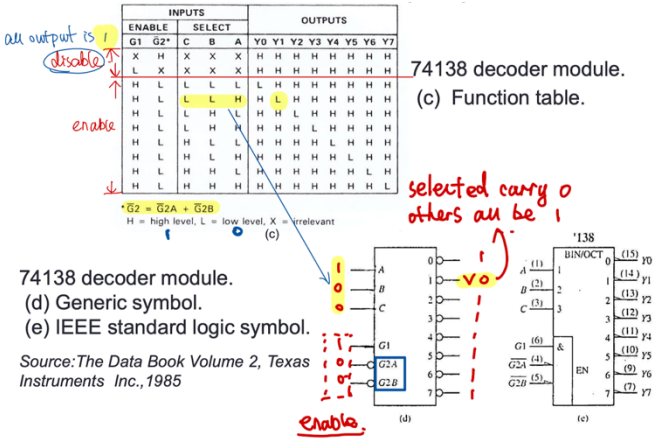
(b) Use one 1×2DEC and two 2×4DEC



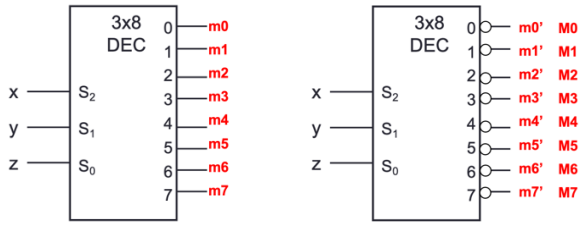
(c) Use one 1×2DEC and one 2×4DEC



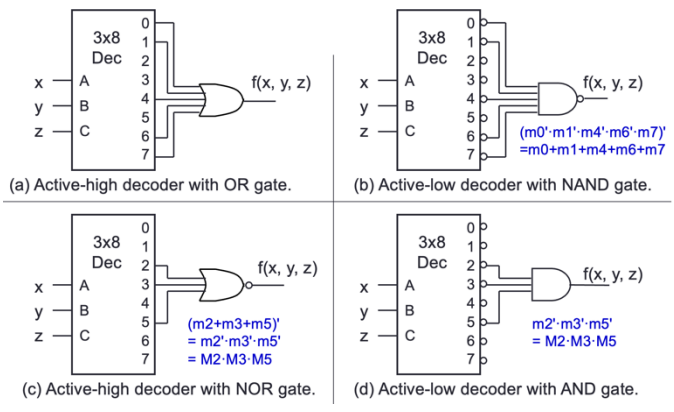
Standard Decoder



Implementing Functions

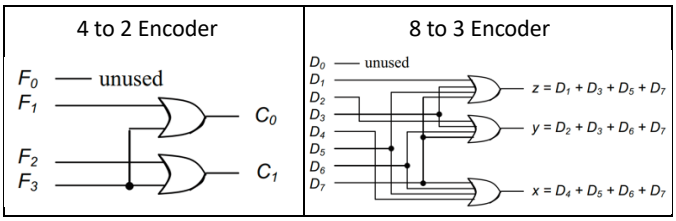


Let $f(x,y,z) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$



(a) Active-high decoder with OR gate.
(b) Active-low decoder with NAND gate.
(c) Active-high decoder with NOR gate.
(d) Active-low decoder with AND gate.

Encoder



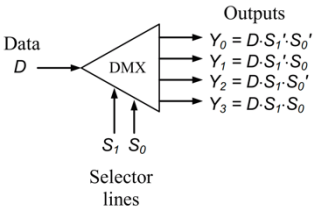
Design Truth Table: Let X be Don't care

F_0	F_1	F_2	F_3	C_1	C_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1
0	0	0	0	X	X
0	0	0	1	X	X
0	1	0	1	X	X
0	1	1	0	X	X
0	1	1	1	X	X
1	0	0	1	X	X
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X

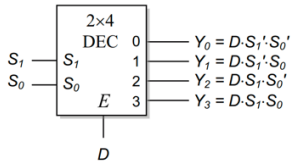
Priority Encoders

Inputs				Outputs		
D_0	D_1	D_2	D_3	f	g	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Demultiplexers: Identical to a decoder with enable



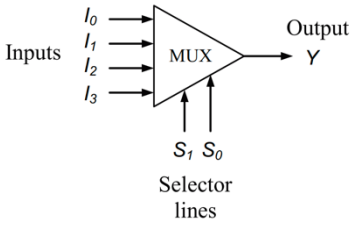
S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D



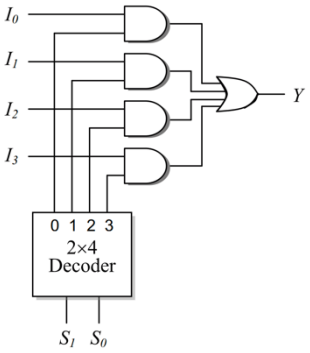
Multiplexers:

$Y = I_0\cdot(S_1'\cdot S_0') + I_1\cdot(S_1'\cdot S_0) + I_2\cdot(S_1\cdot S_0') + I_3\cdot(S_1\cdot S_0)$

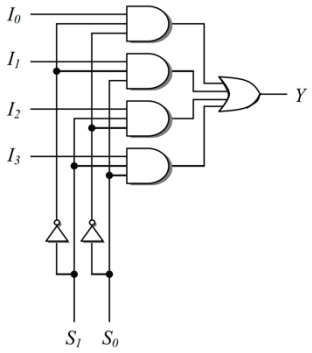
$= I_0\cdot m_0 + I_1\cdot m_1 + I_2\cdot m_2 + I_3\cdot m_3$



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

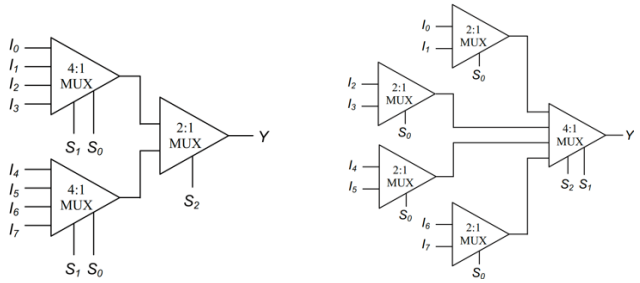


(c) The circuit (with decoder).



(d) The circuit (without decoder).

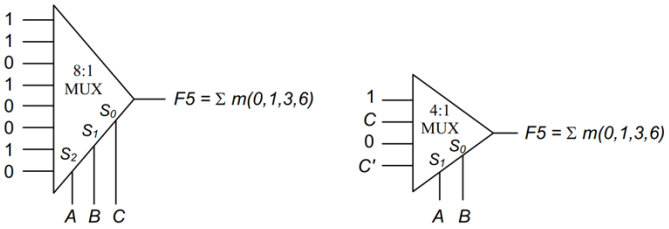
Constructing Larger Multiplexer (8:1 Multiplexer)



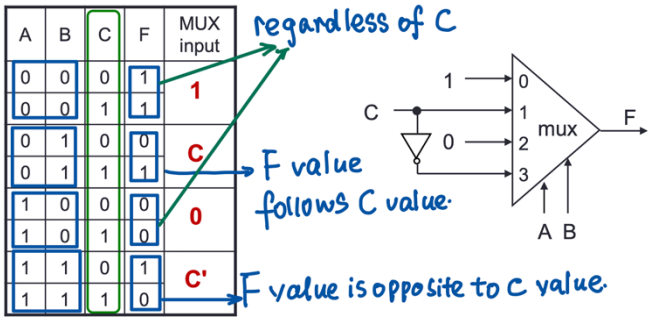
Implementing Functions

- Connect *n* variables to the *n* selection lines.
- Put a '1' on a data line if it is a min-term of the function, or '0' otherwise.

$$F(A,B,C) = \Sigma m(0,1,3,6)$$
$$= A' \cdot B' \cdot C' + A' \cdot B' \cdot C + A' \cdot B \cdot C + A \cdot B \cdot C' \quad (8:1)$$
$$= A' \cdot B' + A' \cdot B \cdot C + A \cdot B \cdot C' \quad (4:1)$$



Basic Idea of using smaller MUX: Truth Table



Logic Gate