CALIFORNIA STATE UNIVERSITY
FULLERTON™

EGEC 180 – Digital Logic and Computer Structures

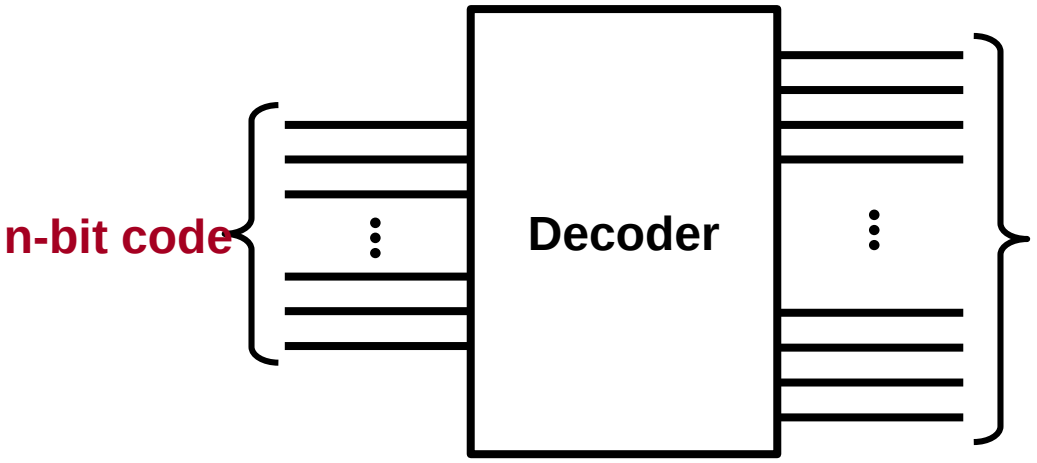
Spring 2024

Lecture 10: Decoder (2.8.2)

Rakesh Mahto, Ph.D.
***Office**: E 314, California State University, Fullerton*
***Office Hour**: Monday and Wednesday 2:00 - 3:30 pm*
Or by appointment
***Office Hour Zoom Meeting ID**: 891 2907 5346*
***Email**: ramahto@fullerton.edu*
***Phone No**: 657-278-7274*

Decoders

Converts n input lines to one of the 2^n different output lines: n-to-2^n Decoder.



Active High if outputs are 1s (minterms)

Active Low if outputs are 0s (maxterms)

The decoder generates all of the minterms (maxterms) of the n input variables. This means only one of the 2^n output lines will be 1 (or 0) all others will be 0 (or 1) for each combination of the values of the input variables.

If n-bit coded information has "don't cares", then the Decoder will output less than 2^n outputs. That is $m \leq 2^n$.

Device Design

We have 4 devices to address, this implies we need only two select lines. This implies we need to design a 2 to 4 decoder

Devices are active LOW

- $\overline{m_0}$ is the CPU
- $\overline{m_1}$ is the Data Memory
- $\overline{m_2}$ is the Video Memory
- $\overline{m_3}$ is the USB Port

E is for the enable line. It is also active Low

Select Lines			Device			
E	A	B	m_0	m_1	m_2	m_3
1	x	x	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Design Procedure for Decoders

Step 1: Let k = n where we have n inputs and x outputs

Step 2:

if k is even

- **Use 2^k AND (NAND) gates driven by two decoders of output size $2^{k/2}$**

if k is odd

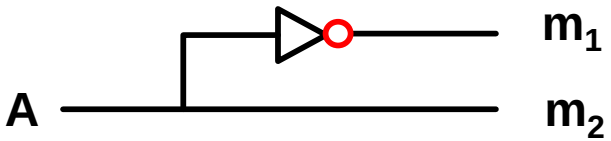
- **Use 2^k AND (NAND) gates driven by two decoders of output size $2^{(k+1)/2}$ and a decoder of output size $2^{(k-1)/2}$**

Step 3

Step 3: For each decoder resulting from step 2 repeat step 2 with k-1 equal to the values obtained in step 2 until k = 1

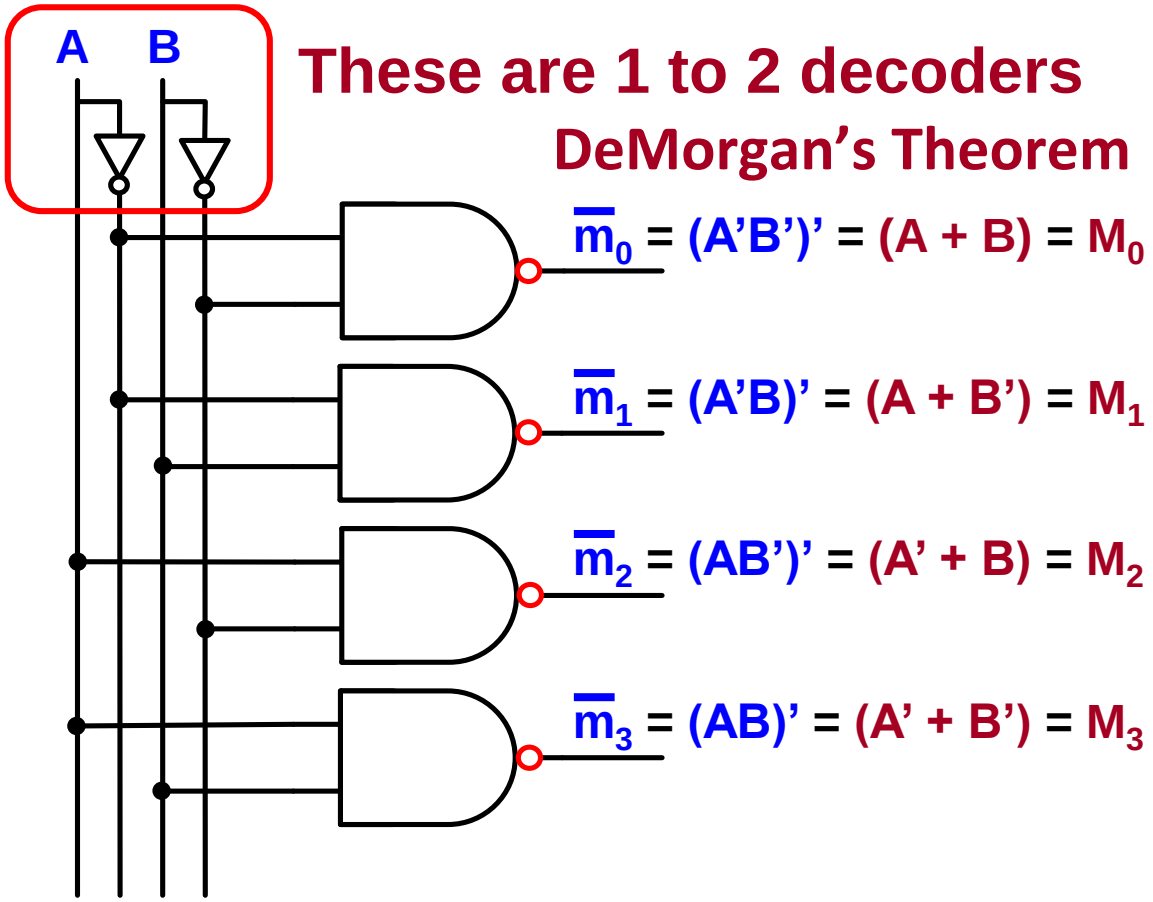
(Note when k = 1 use a 1-to-2 decoder)

A	m_1 (A')	m_2 (A)
0	1	0
1	0	1



2-to-4 Decoder Implementation

Step 2: k is even (k=2) so we will use 2^2 NAND gates.



These are 1 to 2 decoders

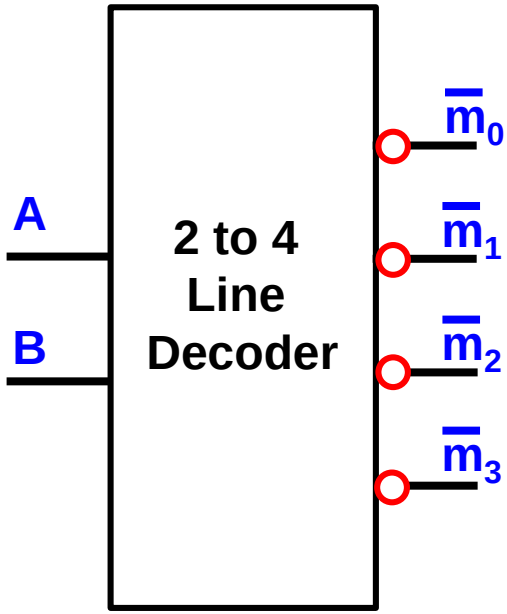
DeMorgan's Theorem

$\overline{m}_0 = (A'B')' = (A + B) = M_0$

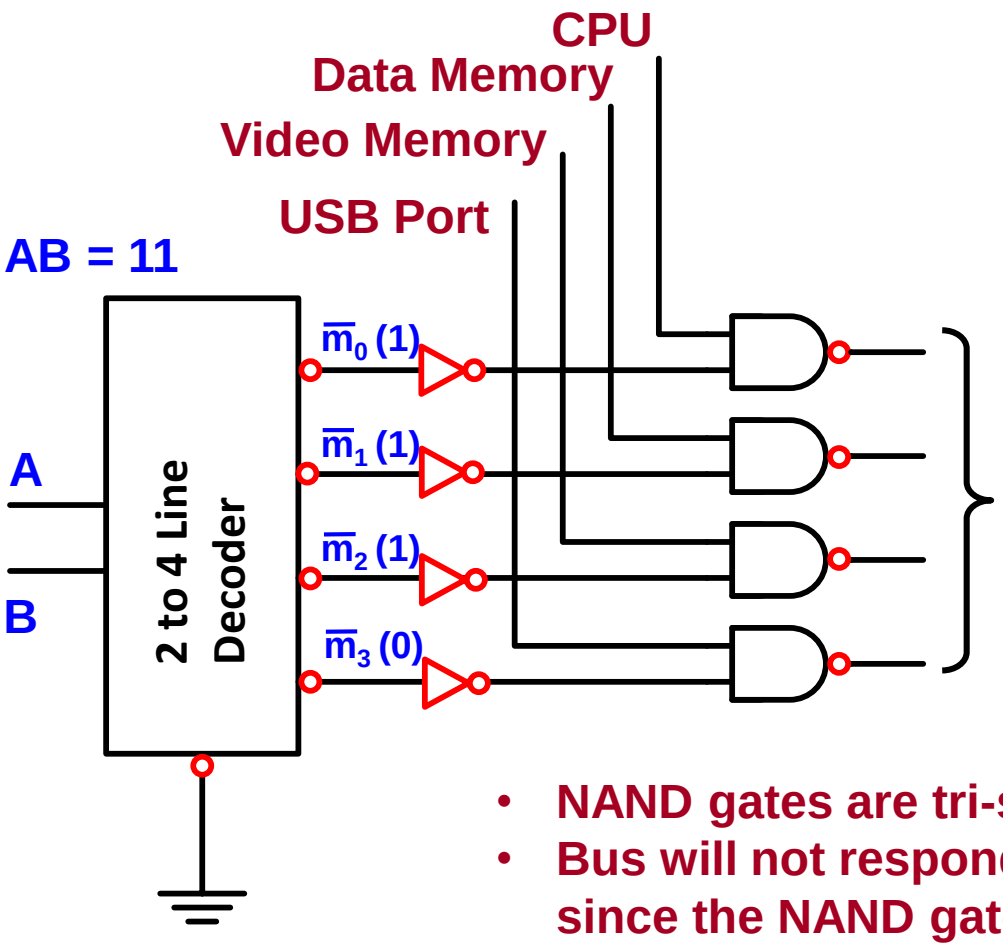
$\overline{m}_1 = (A'B)' = (A + B') = M_1$

$\overline{m}_2 = (AB')' = (A' + B) = M_2$

$\overline{m}_3 = (AB)' = (A' + B') = M_3$



Transmitting End



X	Y	NAND Multiplexer
0	0	1
0	1	1
1	0	1
1	1	0

An input of 0 forces the output to be 1 regardless of the other input

this implies Output is always inverted

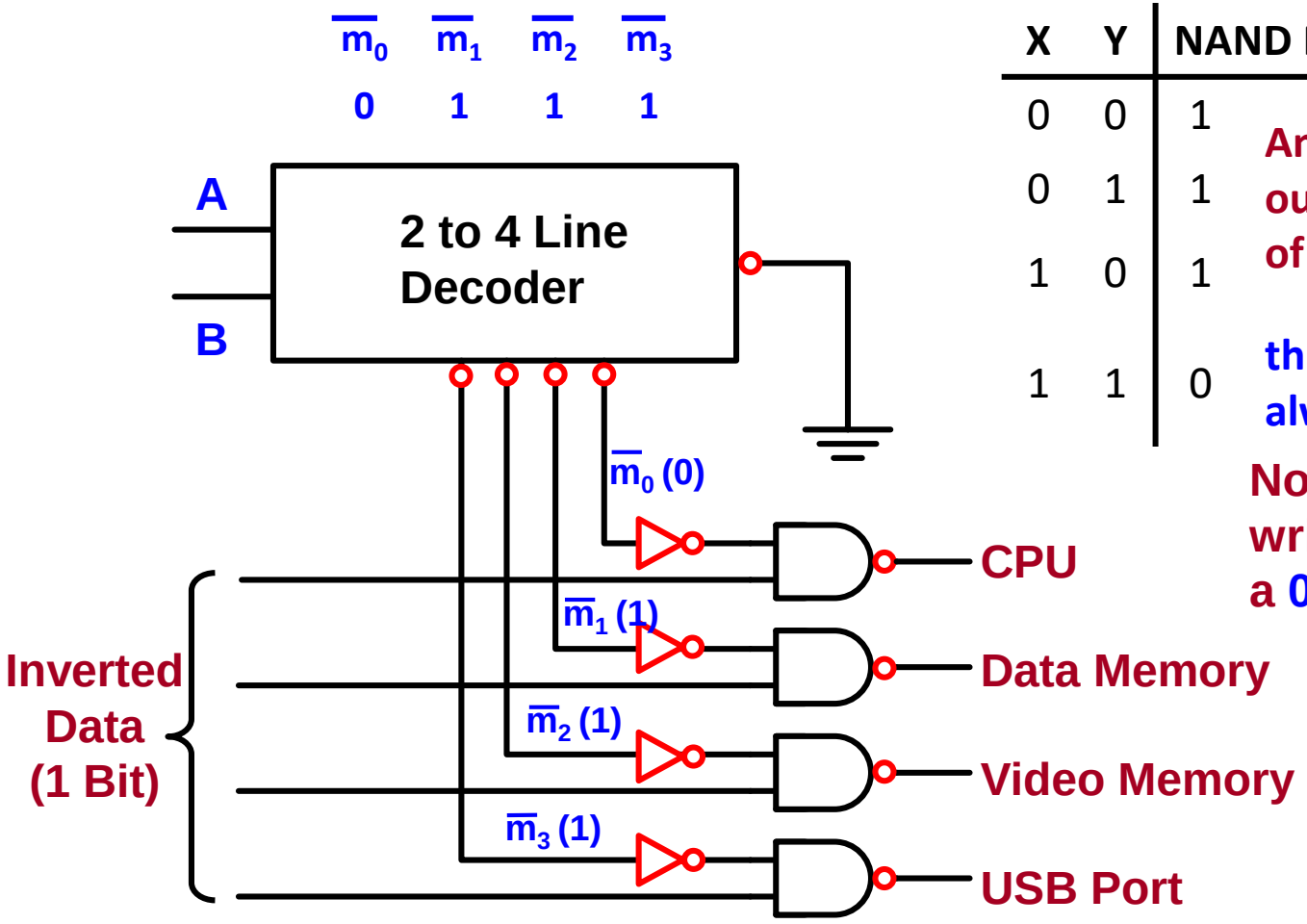
Inverted Data (1 Bit)

Note: If the USB port writes a 1 the bus gets a 0 and vice versa

- NAND gates are tri-state {0, 1, OPEN}
- Bus will not respond to non-selected devices since the NAND gate has a 0 for one of the inputs

Receiving End

Want the CPU (AB = 00) to receive data



X	Y	NAND Multiplexer
0	0	1
0	1	1
1	0	1
1	1	0

An input of 0 forces the output to be 1 regardless of the other input

this implies Output is always inverted

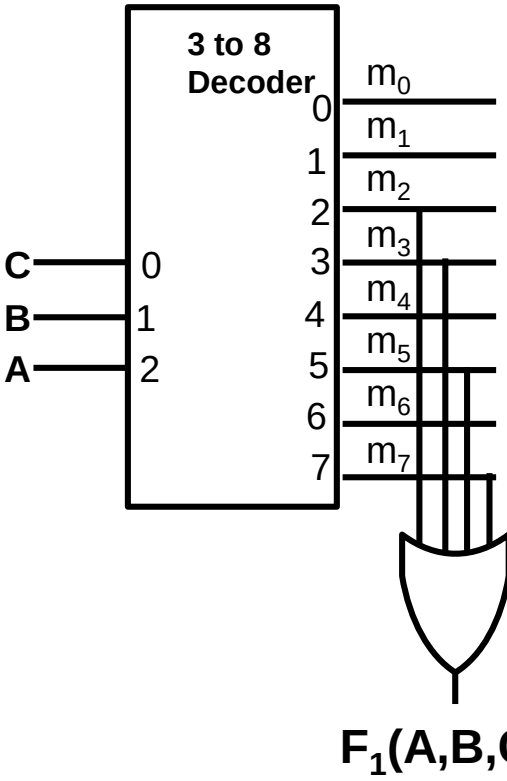
Note: If the USB port writes a 0 the CPU gets a 0 and vice versa

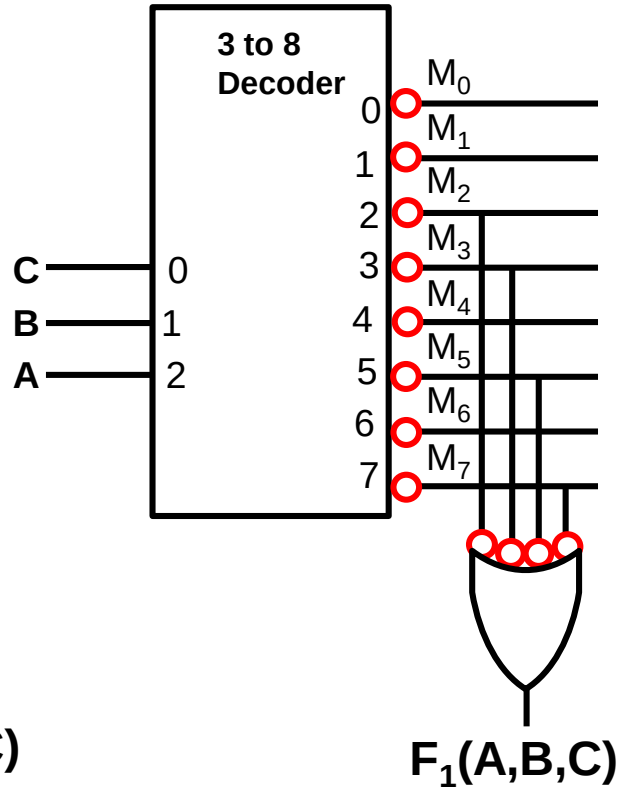
Circuit Design with Decoders

Example 1 - SOM with Decoder

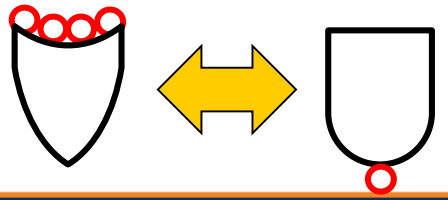
Given $F_1(A,B,C) = \Sigma\, m(2,3,5,7)$ – SOM for a) active High outputs, b) active Low outputs (demulitplexer)

A	B	C	F_1	
0	0	0	0	m_0
0	0	1	0	m_1
0	1	0	1	m_2
0	1	1	1	m_3
1	0	0	0	m_4
1	0	1	1	m_5
1	1	0	0	m_6
1	1	1	1	m_7





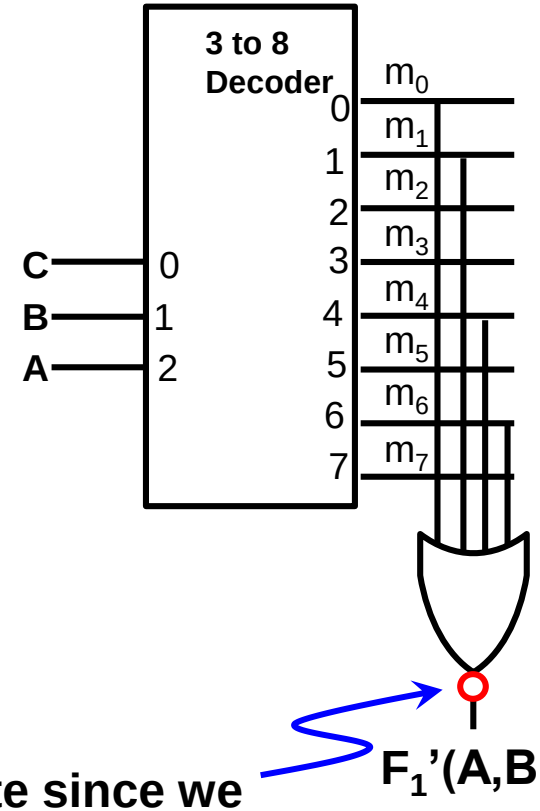
DeMorgan's Theorem

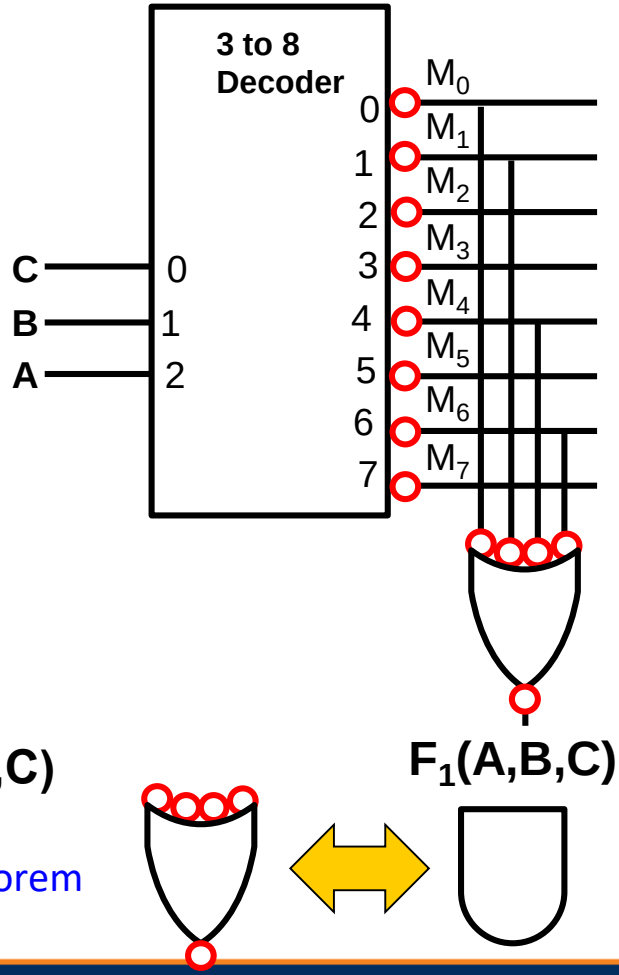


Example 1 - SOM with Decoder

Design of $F_1'(A,B,C) = \Sigma m(0,1,4,6)$ – SOM for a) active High outputs, b) active Low outputs (demulitplexer)

A	B	C	F_1	
0	0	0	0	m_0
0	0	1	0	m_1
0	1	0	1	m_2
0	1	1	1	m_3
1	0	0	0	m_4
1	0	1	1	m_5
1	1	0	0	m_6
1	1	1	1	m_7





Using NOR gate since we want the complement F_1

DeMorgan's Theorem

Excess-3 to Decimal Decoder

Exess 3 to BCD Decoder Example

Input Excess 3					Output BCD				
A	B	C	D		X	Y	Z	W	
0	0	0	0	m_0	d	d	d	d	
0	0	0	1	m_1	d	d	d	d	
0	0	1	0	m_2	d	d	d	d	
0	0	1	1	m_3	0	0	0	0	0
0	1	0	0	m_4	0	0	0	1	1
0	1	0	1	m_5	0	0	1	0	2
0	1	1	0	m_6	0	0	1	1	3
0	1	1	1	m_7	0	1	0	0	4
1	0	0	0	m_8	0	1	0	1	5
1	0	0	1	m_9	0	1	1	0	6
1	0	1	0	m_{10}	0	1	1	1	7
1	0	1	1	m_{11}	1	0	0	0	8
1	1	0	0	m_{12}	1	0	0	1	9
1	1	0	1	m_{13}	d	d	d	d	
1	1	1	0	m_{14}	d	d	d	d	
1	1	1	1	m_{15}	d	d	d	d	

Digits

Note You need 4 decoders one for X, Y, Z, and W

These functions can either implemented using 4 logic circuits or with 4 decoders

$X = \Sigma\, m(11,12) + \Sigma\, d(0,1,2,13,14,15)$

$Y = \Sigma\, m(7,8,9,10) + \Sigma\, d(0,1,2,13,14,15)$

$Z = \Sigma\, m(5,6,9,10) + \Sigma\, d(0,1,2,13,14,15)$

$W = \Sigma\, m(4,6,8,10,12) + \Sigma\, d(0,1,2,13,14,15)$

Designing the Logic Circuits

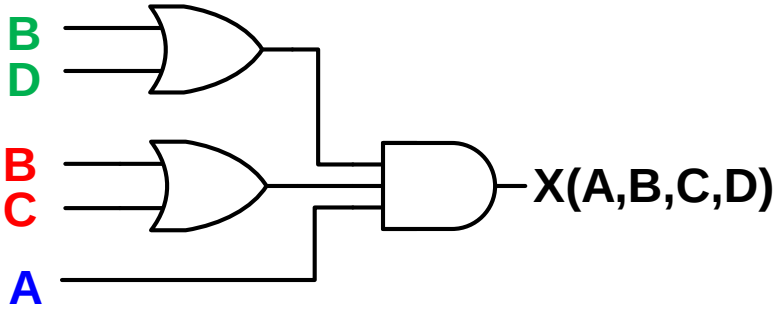
$X = \Sigma\, m(11,12) + \Sigma\, d(0,1,2,13,14,15)$

$Y = \Sigma\, m(7,8,9,10) + \Sigma\, d(0,1,2,13,14,15)$

Using maxterms

AB \ CD	C'D' (00)	C'D (01)	CD (11)	CD' (10)
A'B' (00)	X	X	0	X
A'B (01)	0	0	0	0
AB (11)	1	X	X	X
AB' (10)	0	0	1	0

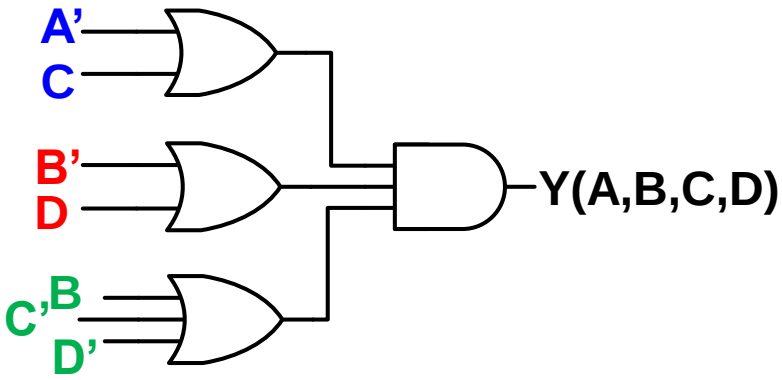
$X(A,B,C,D) = A(B+C)(B+D)$



Using maxterms

AB \ CD	C'D' (00)	C'D (01)	CD (11)	CD' (10)
A'B' (00)	X	X	0	X
A'B (01)	0	0	1	0
AB (11)	0	X	X	X
AB' (10)	1	1	0	1

$Y(A,B,C,D) = (A'+C)(B'+D)(B+C'+D')$



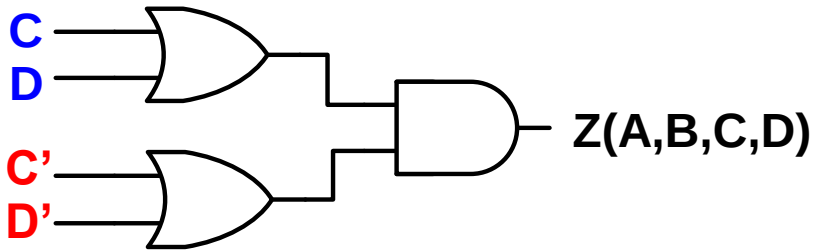
Designing the Logic Circuits

$Z = \Sigma\, m(5,6,9,10) + \Sigma\, d(0,1,2,13,14,15)$

Using maxterms

AB \ CD	C'D' 00	C'D 01	CD 11	CD' 10
A'B' 00	X	X	0	X
A'B 01	0	1	0	1
AB 11	0	X	X	X
AB' 10	0	1	0	1

$X(A,B,C,D) = (C+D)(C'+D')$



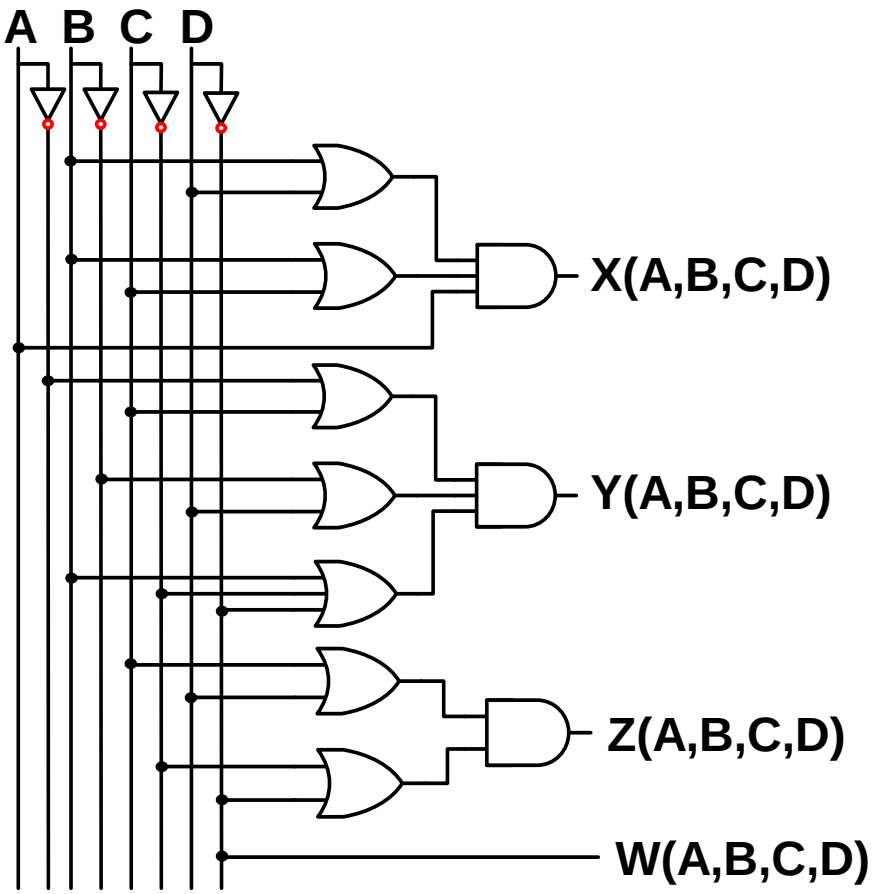
$W = \Sigma\, m(4,6,8,10,12) + \Sigma\, d(0,1,2,13,14,15)$

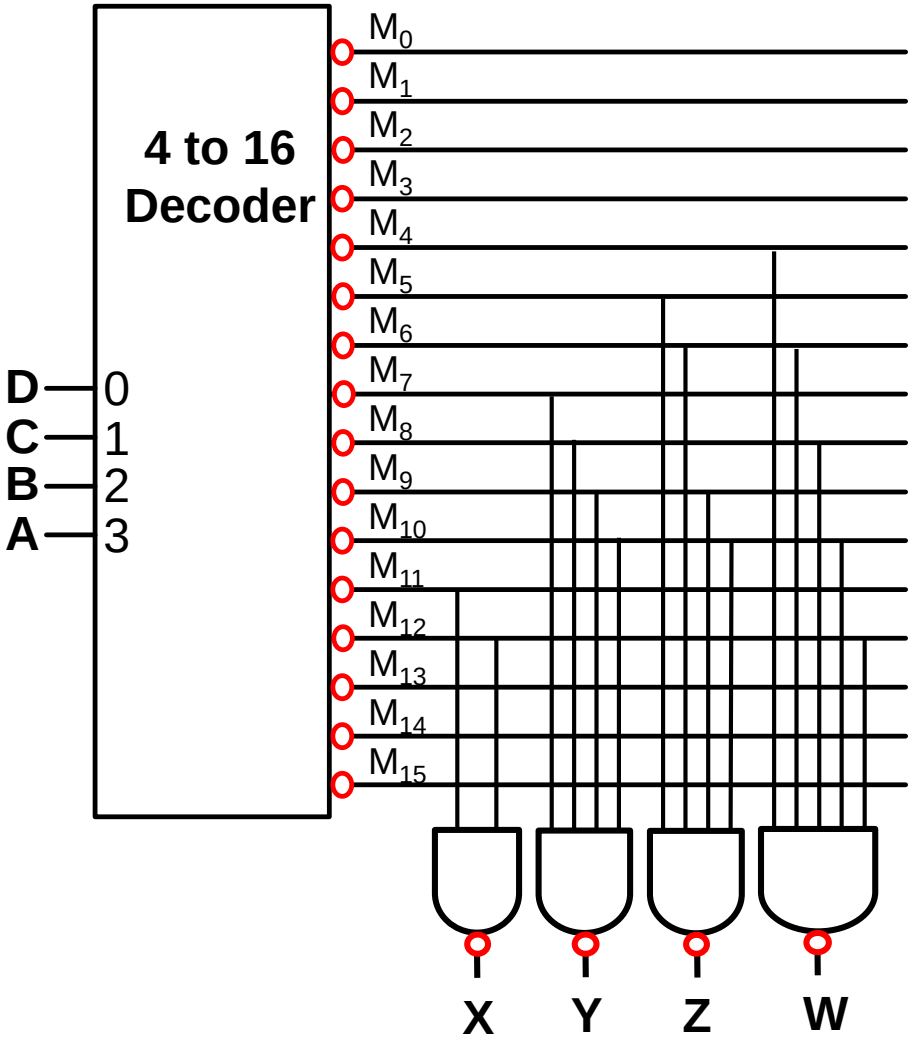
Using maxterms

AB \ CD	C'D' 00	C'D 01	CD 11	CD' 10
A'B' 00	X	X	0	X
A'B 01	1	0	0	1
AB 11	1	X	X	X
AB' 10	1	0	0	1

$W(A,B,C,D) = D'$

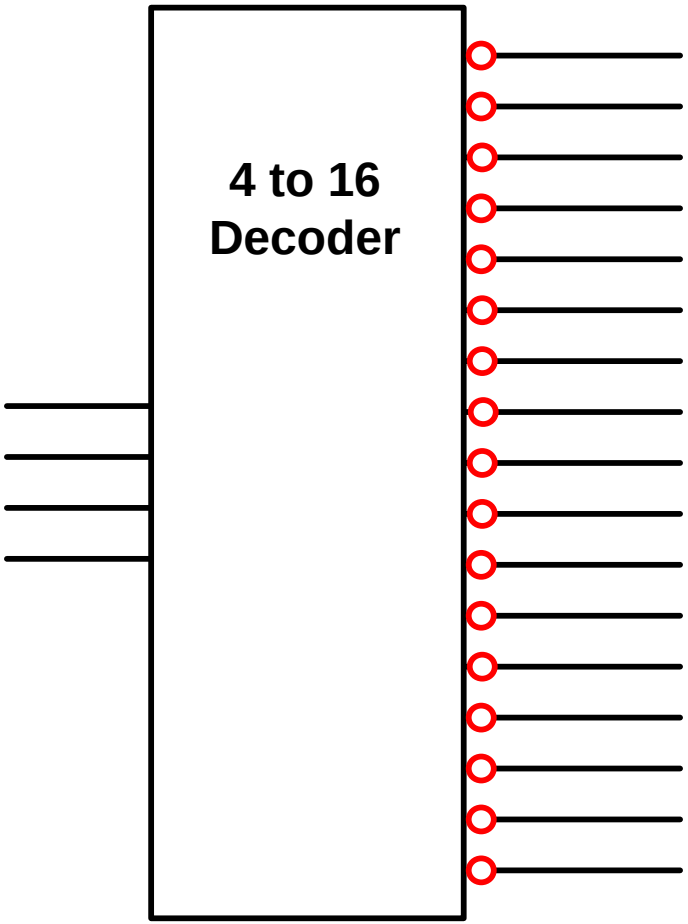
The Finally

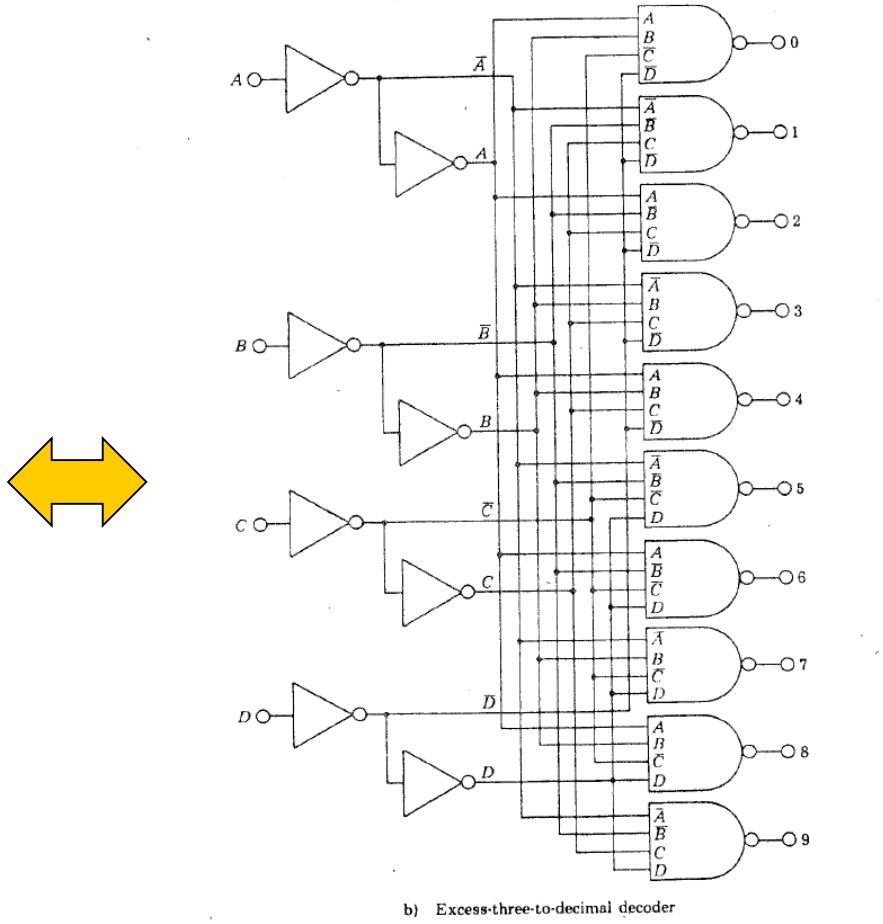

A B C D
X(A,B,C,D)
Y(A,B,C,D)
Z(A,B,C,D)
W(A,B,C,D)


4 to 16
Decoder
D
C
B
A
0
1
2
3
M0
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
M13
M14
M15
X
Y
Z
W

Food For Thought

Although this may be simpler. It implies a lot of circuitry (a lot of power consumed)





b) Excess-three-to-decimal decoder

Q&A