Unit 9

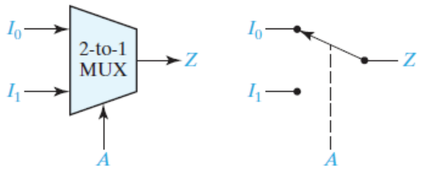
Multiplexers, Decoders, and Programmable Logic Devices

Logic Circuits (Spring 2022)

Multiplexers

- Multiplexer (MUX)
  - Data selector
  - Takes a group of data input and a group of control inputs
  - Control inputs are used to select one of the data inputs and connect it to the output terminal

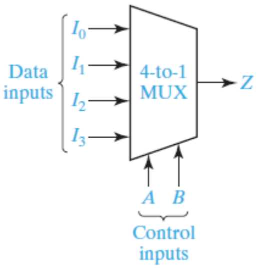
- Example: 2-to-1 Multiplexer



$$Z = A'I_0 + AI_1$$

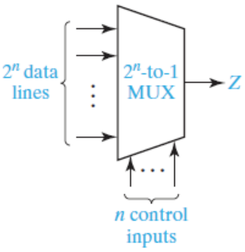
Multiplexers

4-to-1 Multiplexer



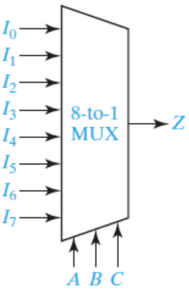
$$Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$

2ⁿ-to-1 Multiplexer

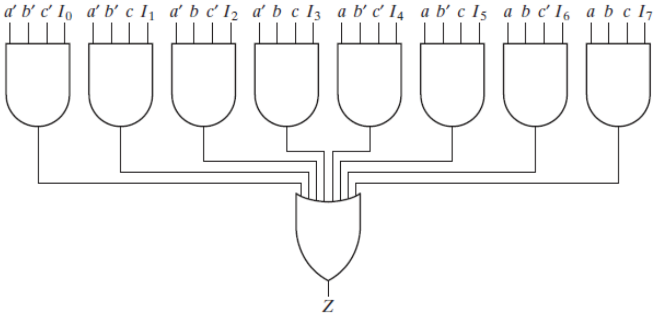


$$Z = \sum_{k=0}^{2^n-1} m_k I_k$$

8-to-1 Multiplexer



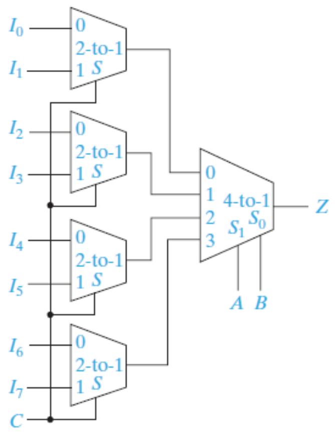
AND-OR circuit implementation



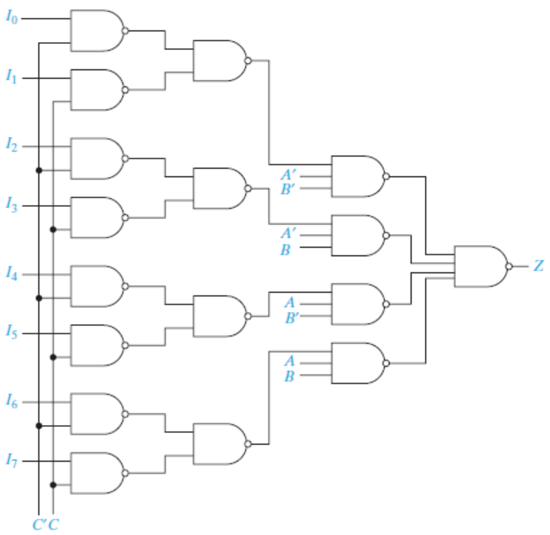
$$\begin{aligned} Z = {} & A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 \\ & + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7 \end{aligned}$$

8-to-1 Multiplexer

2-level MUX implementation

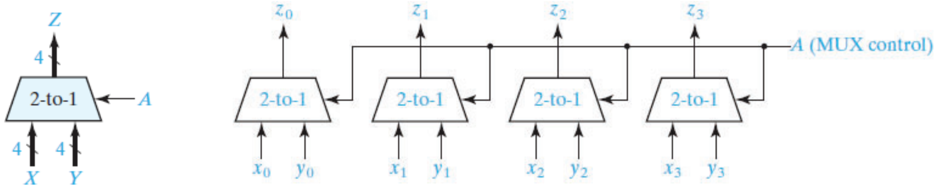


NAND implementation



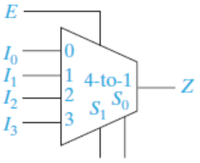
Quad 2-to-1 Multiplexer

- Quad multiplexer
  - Four multiplexer working homogeneously together
  - Selects one of “4-bit data words”
- 2-to-1 multiplexer
  - Selects one of two data

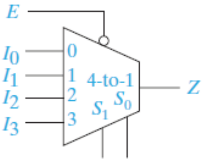


Multiplexers with Enable

- Additional “enable” signal
  - An additional input
  - If $E = 1$, the multiplexer functions as an ordinary multiplexer
  - If $E = 0$, the multiplexer does not deliver any input to the output
- Active high or active low



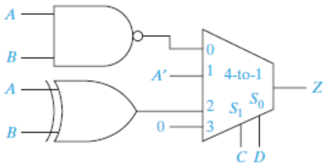
active high enable



active low enable

Implementation of a 4-Variable Function (1)

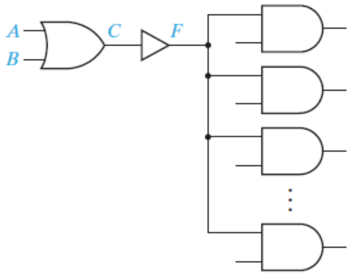
MUX implementation



$$Z = C'D'(A' + B') + C'D(A') + CD'(AB' + A'B) + CD\ (0)$$
$$= A'C' + A'BD' + AB'D'$$

Buffers

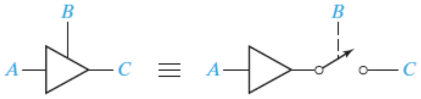
- Why buffers?
  - A gate output can only be connected to a limited number of other device inputs without degrading the digital system's performance
  - A simple buffer may be used to increase the driving capability of a gate output
- Example: a buffer between a gate output and several gate inputs



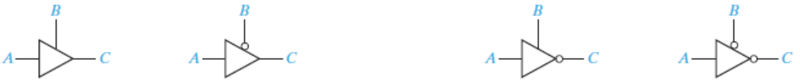
- The outputs of two or more gates cannot be connected to each other

Three-State Buffers

- Three states?
  - 0, 1
  - Hi-Z (high-impedance)



- Three-state buffer
  - B: enable input
  - When B is 0, the output C acts like an open circuit
- Four kinds of three-state buffers



B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

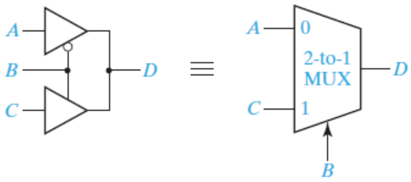
B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z

B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

B	A	C
0	0	1
0	1	0
1	0	Z
1	1	Z

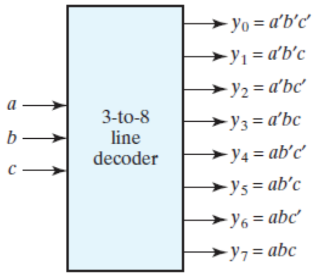
Multiplexer Using Three-State Buffers

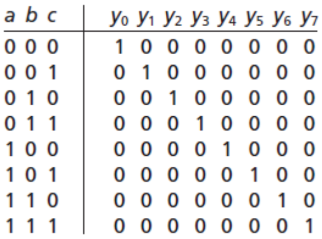
- 2-to-1 multiplexer
  - When $B = 0$, the top buffer is enabled, so $D = A$
  - When $B = 1$, the lower buffer is enabled, so $D = C$
- Two three-state buffers could be combined



3-to-8 Line Decoder

- Decoder
  - Generates all of the minterms of the input variables
  - Exactly one of the output lines is 1 for each input combination
- Example: 3-to-8 line decoder

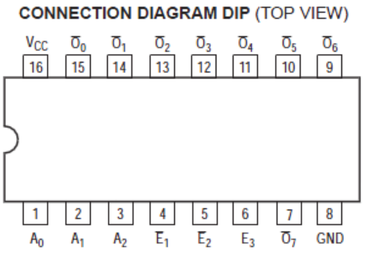


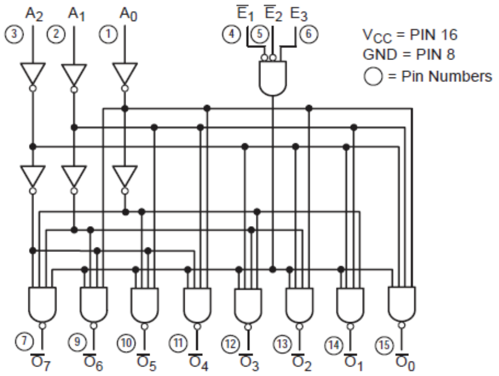


a b c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0 0 0	1	0	0	0	0	0	0	0
0 0 1	0	1	0	0	0	0	0	0
0 1 0	0	0	1	0	0	0	0	0
0 1 1	0	0	0	1	0	0	0	0
1 0 0	0	0	0	0	1	0	0	0
1 0 1	0	0	0	0	0	1	0	0
1 1 0	0	0	0	0	0	0	1	0
1 1 1	0	0	0	0	0	0	0	1

3-to-8 Line Decoder: Example

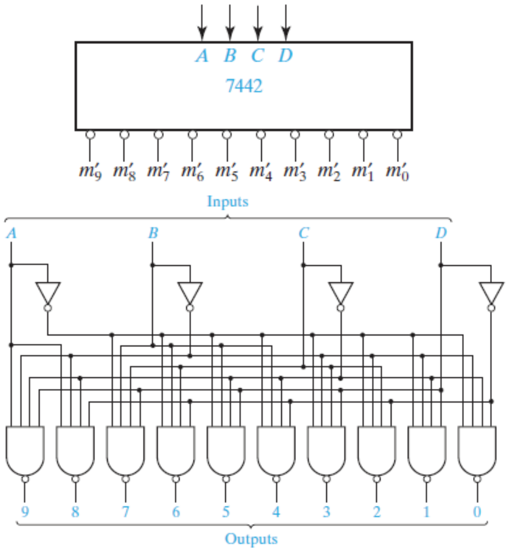
74LS138 (3-to-8 line decoder)





4-to-10 Line Decoder

- Example: 4-to-10 line decoder with inverted outputs
  - Exactly one of the output lines will be 0 for each input combination

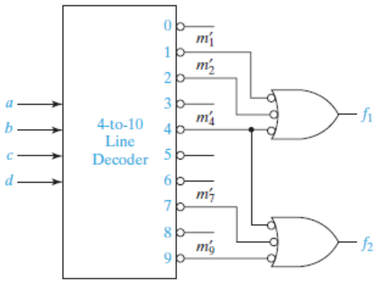

A B C D
7442
m_9' m_8' m_7' m_6' m_5' m_4' m_3' m_2' m_1' m_0'
Inputs
A
B
C
D
9 8 7 6 5 4 3 2 1 0
Outputs

BCD Input				Decimal Output									
A	*B*	*C*	*D*	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Implementation of a 4-Variable Function (2)

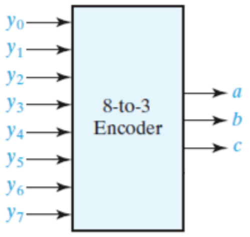
Decoder implementation

$$f_1(a, b, c, d) = m_1 + m_2 + m_4$$
$$f_2(a, b, c, d) = m_4 + m_7 + m_9$$



Encoder

- Encoder
  - Performs the inverse function of a decoder
  - If input y_i is 1 and the other inputs are 0, the *abc* outputs represent a binary number equal to *i*

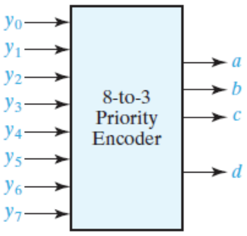


y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- Only 8 input combinations are allowed

Priority Encoder

- Priority?
  - If only an input y_i is 1, the outputs, *abc* are a binary number equal to *i*
  - If more than one inputs are 1, the outputs are defined on a priority basis
  - If any input is not 1, *d* is defined as 0, which indicates the *abc* output is not defined

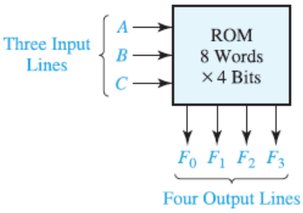


y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	*a*	*b*	*c*	*d*
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

- Output $d = 0$
  - If any input is not 1
  - Means that the *abc* output is not defined

Read-Only Memory (ROM)

- ROM
  - An array of semiconductor devices to store binary data
  - Once binary data is stored, it can be read out whenever desired
  - Stored data cannot be changed under normal operating conditions
- Example: $2^3 \times 4$ ROM
  - Realizes 4 functions of 3 variables

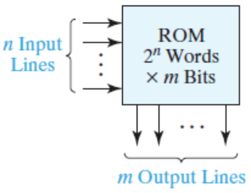


A	*B*	*C*	F_0	F_1	F_2	F_3
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1

Typical Data Stored in ROM (2^3 words of 4 bits each)

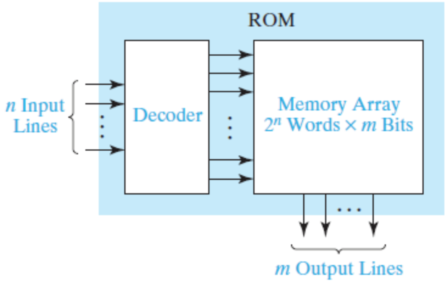
Read-Only Memory (ROM)

- $2^n \times m$ ROM
  - Can realize m functions of n variables
  - Can store a truth table with 2^n rows and m columns



n Input Variables	*m* Output Variables
00 · · · 00	100 · · · 110
00 · · · 01	010 · · · 111
00 · · · 10	101 · · · 101
00 · · · 11	110 · · · 010
⋮	⋮
11 · · · 00	001 · · · 011
11 · · · 01	110 · · · 110
11 · · · 10	011 · · · 000
11 · · · 11	111 · · · 101

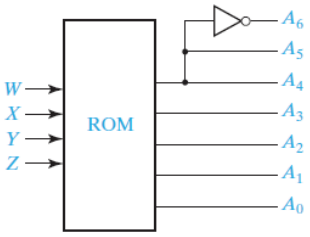
Typical Data Array Stored in ROM (2^n words of *m* bits each)



Implementation of a 4-Variable Function (3)

ROM implementation

Input				Hex	ASCII Code for Hex Digit						
W	*X*	*Y*	*Z*	Digit	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0	1
0	0	1	0	2	0	1	1	0	0	1	0
0	0	1	1	3	0	1	1	0	0	1	1
0	1	0	0	4	0	1	1	0	1	0	0
0	1	0	1	5	0	1	1	0	1	0	1
0	1	1	0	6	0	1	1	0	1	1	0
0	1	1	1	7	0	1	1	0	1	1	1
1	0	0	0	8	0	1	1	1	0	0	0
1	0	0	1	9	0	1	1	1	0	0	1
1	0	1	0	A	1	0	0	0	0	0	1
1	0	1	1	B	1	0	0	0	0	1	0
1	1	0	0	C	1	0	0	0	0	1	1
1	1	0	1	D	1	0	0	0	1	0	0
1	1	1	0	E	1	0	0	0	1	0	1
1	1	1	1	F	1	0	0	0	1	1	0



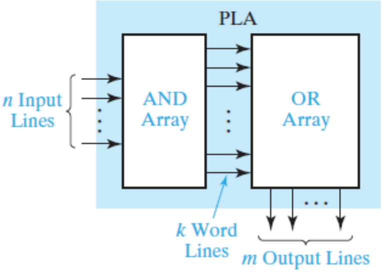
Programmable Logic Devices

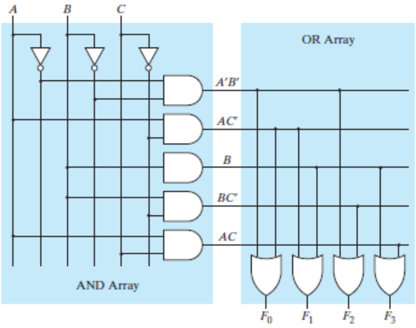
- Programmable Logic Device (PLD)
  - A digital IC that can be *programmed* to implement an arbitrary logic function
  - Undefined at the time of manufacture
  - Must be programmed before being used

- Some variants
  - PLA
  - PAL: special case of PLA
  - CPLD (Complex Programmable Logic Devices)
  - FPGA (Field Programmable Gate Array)

Programmable Logic Array (PLA)

- PLA with *n* inputs and *m* outputs

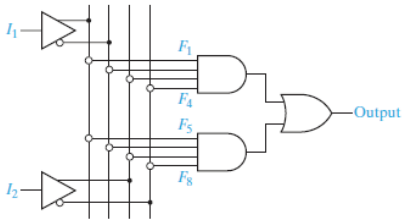


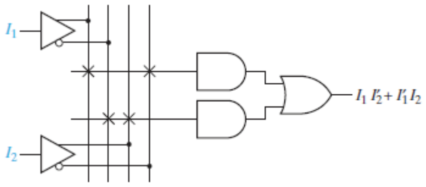


- Example: PLA with 3 inputs and 4 outputs
  - 5 product terms are *programmed* with 3 inputs
  - 4 outputs are *programmed* with the 5 product terms

Programmable Array Logic (PAL)

- PAL
  - AND array is programmable
  - OR array is fixed


I_1
F_1
F_4
Output
F_5
F_8
I_2


I_1
$I_1 I_2' + I_1' I_2$
I_2