



Lecture Outline

Reminders to self:

- ☐ Turn on lecture recording to Cloud
- ☐ Turn on Zoom microphone

- Last Lecture

- Continued Multiplexers
- Brief look at CMOS gates at transistor level
- Started Three-state buffers / Tri-state outputs

- Today's Lecture

- Continue Three-state buffers / Tri-state outputs
- Decoders
- Encoders
- ROM



Handouts and Announcements

- Announcements

- Homework Problems 9-1 and 9-2

- Posted on Carmen yesterday afternoon (2/16)
- HW 9-1 due: 11:25am Wednesday 2/22
- HW 9-2 due: 11:59pm Thursday 2/23

- Homework Reminder:

- HW 8-1 and 8-2 Now Past-Due: 11:59pm Thursday 2/16

- Participation Quiz 8 available starting 12:25pm today

- Due 12:25pm Saturday 2/18
- Available until 12:25pm Sunday 2/19 with late penalty

- Read for Monday: pages 275-280, 331, 336-342



Handouts and Announcements

- Announcements

- Mini-Exam 3 Reminder

- Available 5pm Monday 2/20 through 5:00pm Tuesday 2/21
- Due in Carmen PROMPTLY at 5:00pm on 2/21
- Designed to be completed in ~36 min, but you may use more
- When planning your schedule:
 - I recommend building in 10-15 min extra
 - To allow for downloading exam, signing and dating honor pledge, saving solution as pdf, and uploading to Carmen
- I also recommend not procrastinating

- Exam review topics available on Carmen

- Sample Mini-Exams 3 and 4 from Au20 also available



Adder/Subtractor

Here is a 3-bit version, implemented using multiplexers

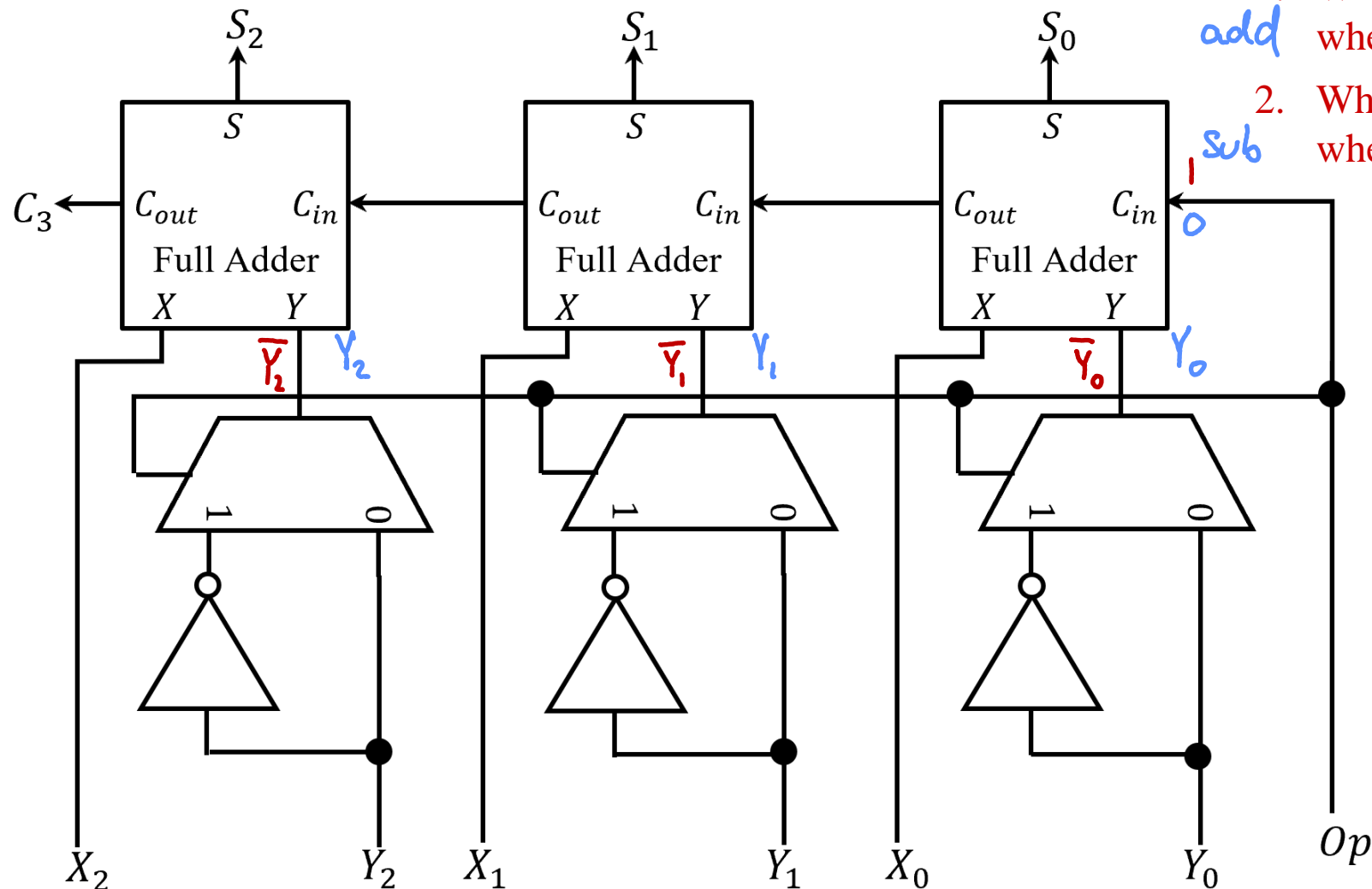
Quiz 10 review

1. What operation
when $Op = 0$?

add

2. What operation
when $Op = 1$?

sub





HW 7-3 Review

Draw a logic circuit that uses only 2-input NAND gates to implement the following function. Your circuit **must use the minimum number of 2-input NAND gates.**

Since you are limited to only 2-input NAND gates you may need to use more than two levels of logic.

$$F(A, B, C, D) = ABCD' + ABC' + AB'C'D + A'BC' + AB'C$$

- **Start by minimizing the expression**

- We taught how to use K-Maps for that: when algorithms properly followed you know you have a minimum SOP expression
- Some students tried Boolean algebra instead – all of the ones I graded stopped algebra too soon - before they had a minimum expression → too many gates
- Some students didn't minimize at all → saw a case with 21 gates! – can be done with 6

- **Then try to factor expression to reach fan-in constraint**

- Brute force implementation likely needs extra gates – often 2-NAND used as inverters
- Optimal solution of this problem didn't need any inverters

- **Then convert circuit/expression of 2-AND and 2-OR gates to 2-NAND only**

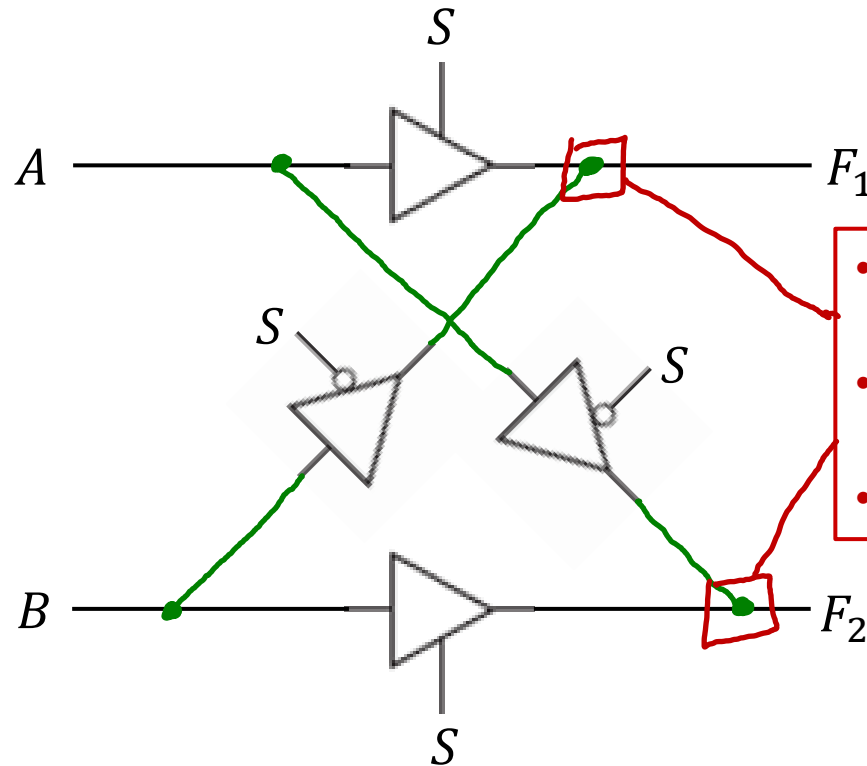
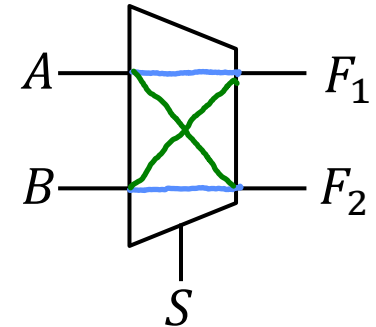
- Graphical approach with inversion bubbles, or
- Boolean algebra with liberal use of DeMorgan's



Three-State Buffers

• Straight-through/Cross-through MUX

- $S=1$ $A \rightarrow F_1$ $B \rightarrow F_2$
- $S=0$ $A \rightarrow F_2$ $B \rightarrow F_1$
- $F_1 = AS + B\bar{S}$
- $F_2 = A\bar{S} + BS$



- Two outputs tied together
- But only one active at a time
- Other in high-Z state



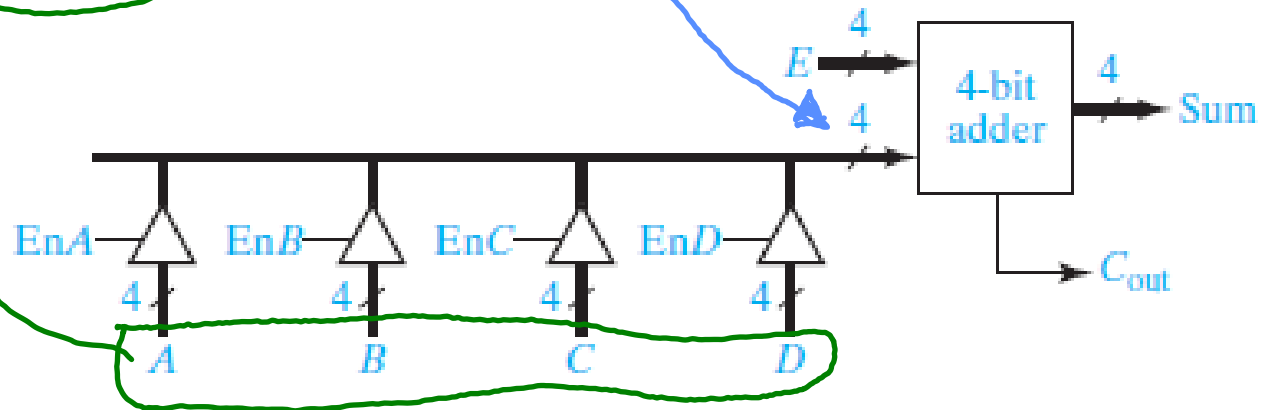
Three-State Buffers

- Three-state bus as alternative to MUX data selector
- Example: 4-bit adder with one operand sourced from four different locations

FIGURE 9-15

4-Bit Adder with
Four Sources for
One Operand

© Cengage Learning 2014



- A bus controller is needed to ensure that only one of EnA , EnB , EnC , and EnD are true at any time
- Outputs of the other buffers must be in *high-Z state*
- Device known as *decoder* can be used for bus control



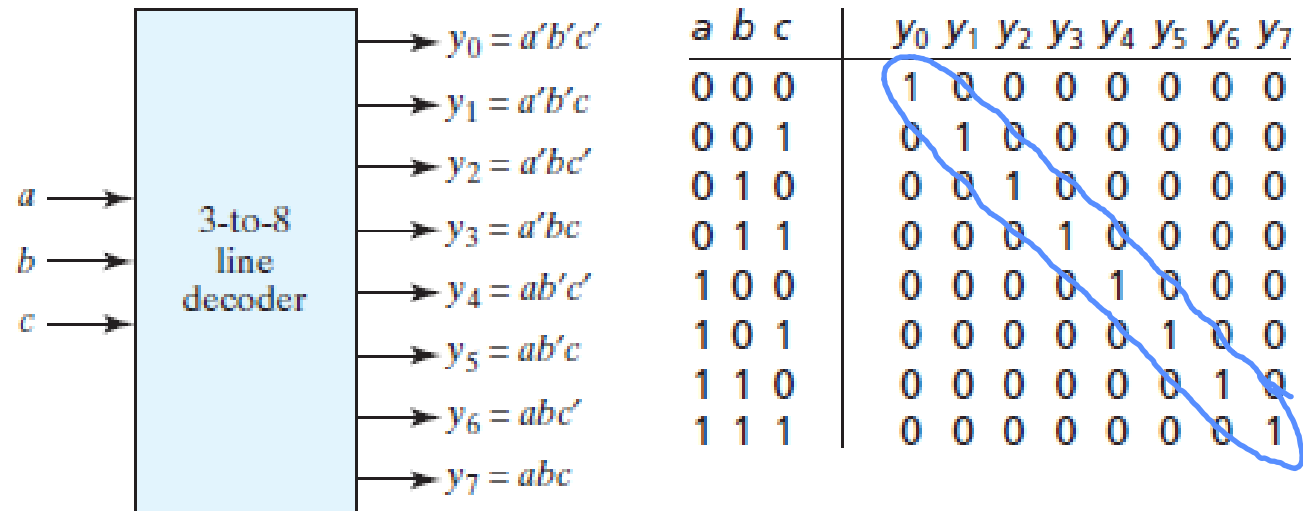
Decoder

- A decoder is another commonly used type of IC
- Example: 3-8 Decoder Block Diagram and Truth Table:

FIGURE 9-17

A 3-to-8 Line Decoder

© Cengage Learning 2014



- Input binary #
- Activates the line corresponding to that number
 - $000 \rightarrow m_0$
 - $101 \rightarrow m_5$



Three-State Buffers

- Device known as **decoder** can be used for bus control

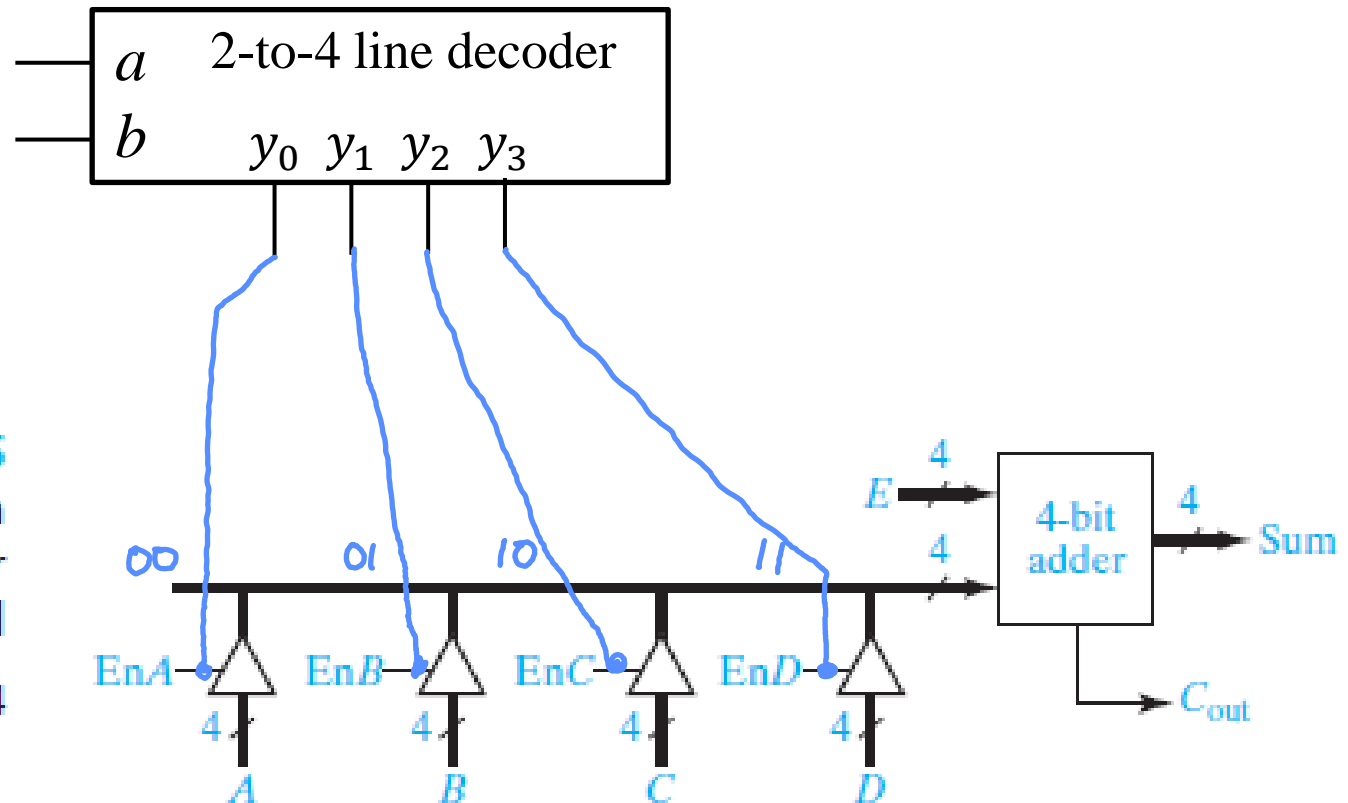


FIGURE 9-15

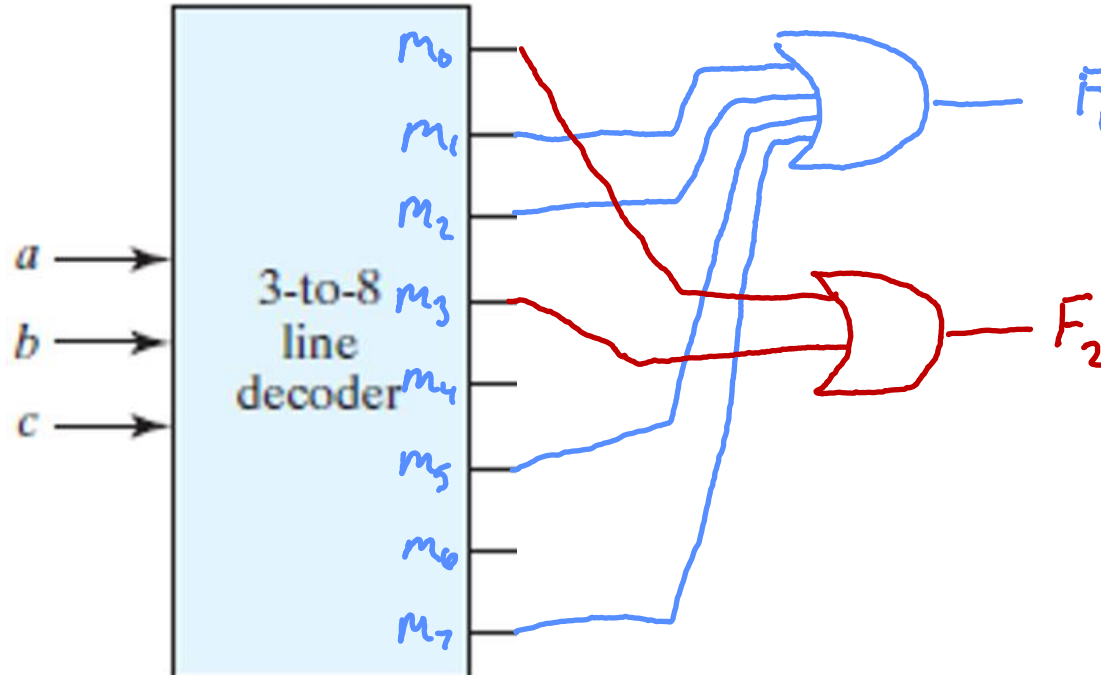
4-Bit Adder with
Four Sources for
One Operand

© Cengage Learning 2014



Decoder

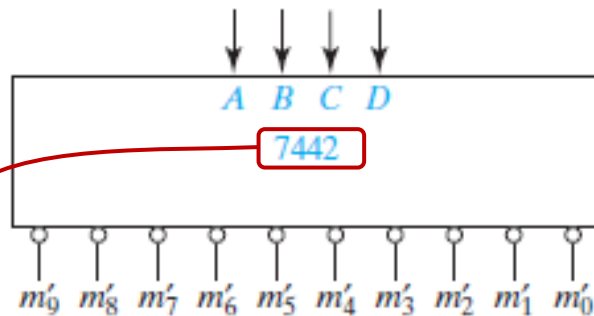
- In general, n -to- 2^n line decoder generates all 2^n minterms of the n input variables
- Example: Use to implement
 - $f_1 = \sum m(1,2,5,7)$
 - $f_2 = \sum m(0,3)$





Decoder

- Decoder may also have inverting outputs
- And may also only have a subset of the lines



(b) Block diagram

7400-series TTL part number
4-to-10 line decoder

- Generates
or
- Use inverted outputs
with gates to
generate functions

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	0	1	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

(c) Truth Table



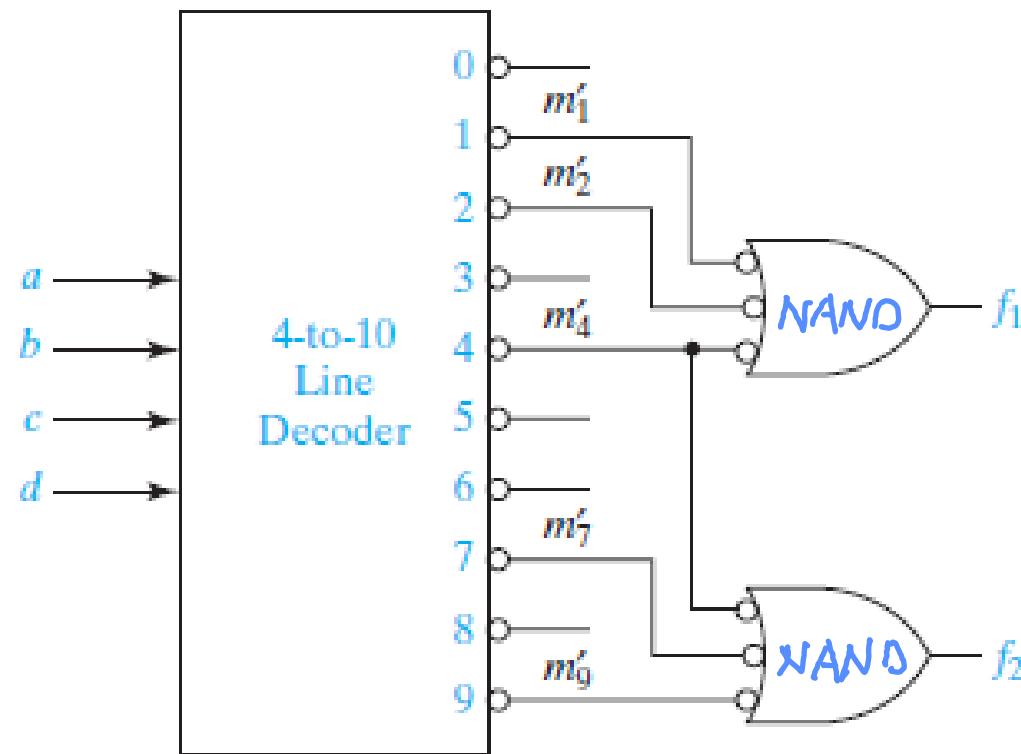
Decoder

- Use inverted outputs with **NAND** gates to generate functions

FIGURE 9-19

Realization of a
Multiple-Output
Circuit Using a
Decoder

© Cengage Learning 2014



$$f_1(a, b, c, d) = m_1 + m_2 + m_4 \text{ and } f_2(a, b, c, d) = m_4 + m_7 + m_9$$



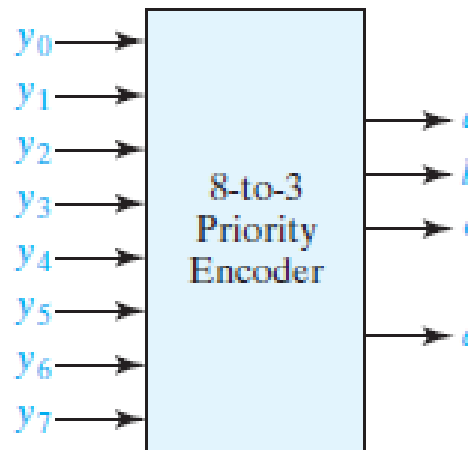
Priority Line Encoder

- Encoders have the inverse function of decoders
- If more than one input = 1, then the biggest input wins
 - Example: y_7 & $y_3 = 1$
 - Output = 111 because priority
 - The don't cares implement that
- Output $d = 1$ if any input = 1. Not all encoders have this.
 - Distinguish between all 0 and $y_0 = 1$

FIGURE 9-20

An 8-to-3 Priority Encoder

© Cengage Learning 2014

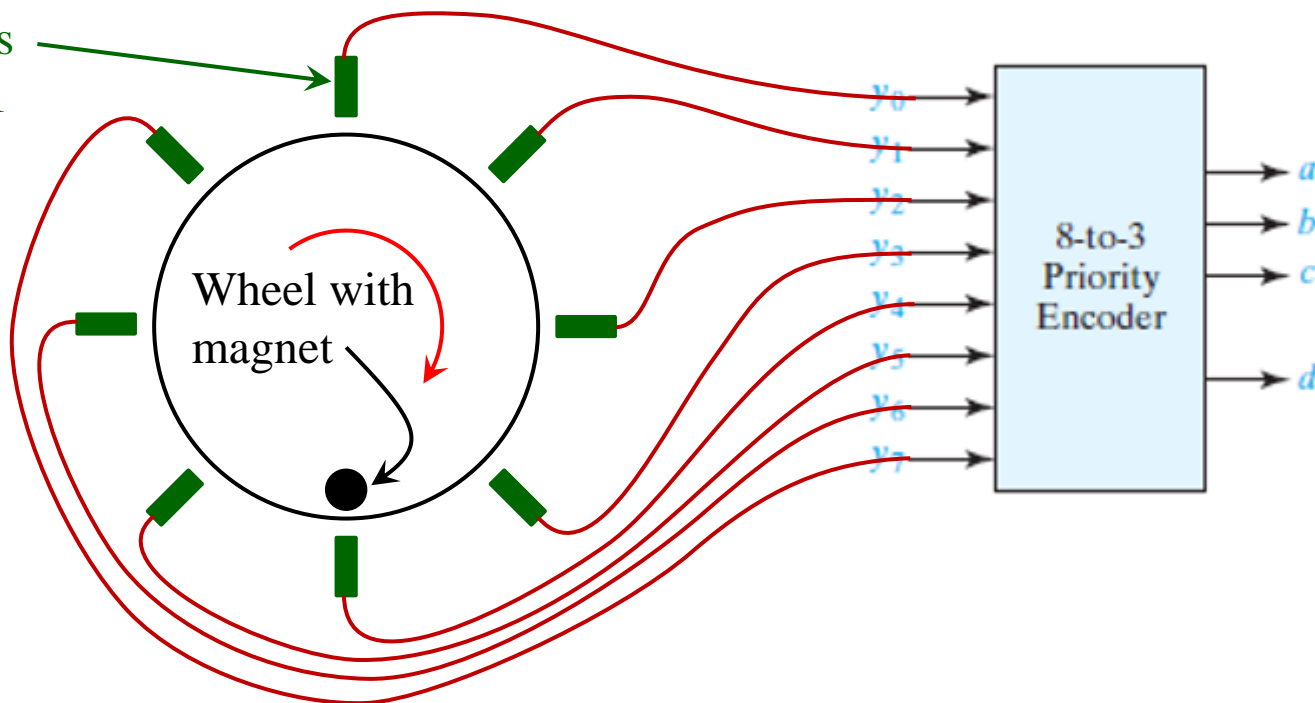


y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1



Basic Encoder Example

Magnetic reed switches
wired to deliver logic 1
when activated by
magnet



As drawn: pos = south

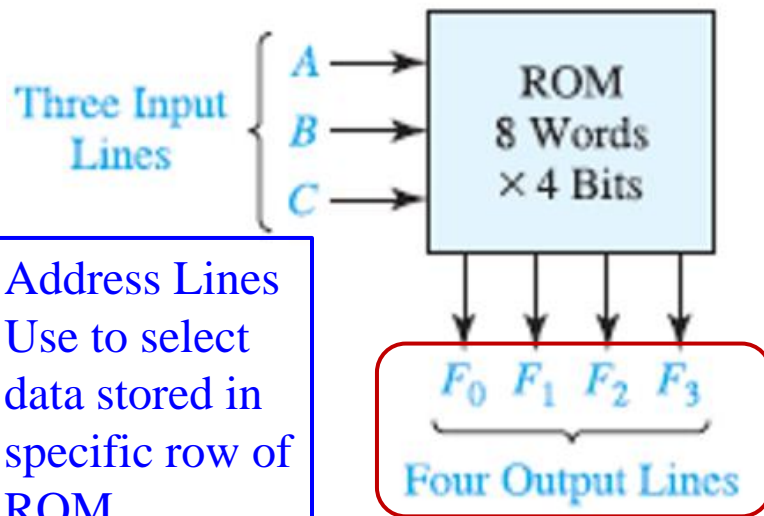
Wheel position
encoded into 3-bit
binary word: *abc*

<i>a</i>	<i>b</i>	<i>c</i>	Wheel Position
0	0	0	North
0	0	1	North-East
0	1	0	East
0	1	1	South-East
1	0	0	South
1	0	1	South-West
1	1	0	West
1	1	1	North-West



Read-Only Memory

- Read-only memory (ROM) consists of an array of semiconductor devices interconnected to store an array of binary data
- Once binary data is stored in the ROM:
 - Data can be read out whenever desired
 - But data that is stored cannot be changed during normal operation
 - Lookup Table (LUT)
- Example: An 8-Word, 4-bit ROM



(a) Block diagram

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

(b) Truth table for ROM

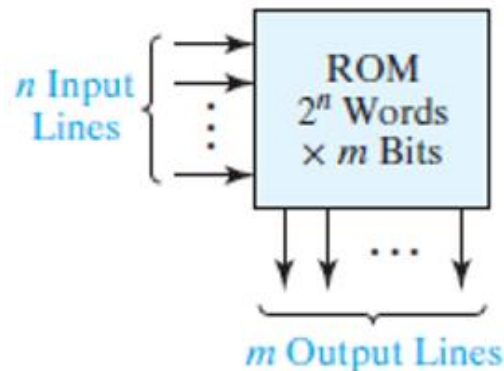
Typical (Example)
Actual data will be
for programmed
application

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



Read-Only Memory

- A ROM with n input lines and m output lines contains
 - an array of 2^n words
 - each word is m bits long.
- Input lines serve as address to select one of the 2^n words
- When an input combination is applied to the ROM, the pattern of 0's and 1's stored in the corresponding word appears at the output lines
- A $2^n \times m$ ROM can realize
 - m functions
 - of n variables
 - because it 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)

- 64k \times 8-bit ($n = 16, m = 8$)
- 512k \times 16-bit ($n = ?, m = ?$)
- 32 word, 4-bit ROM ($n = ?, m = ?$)



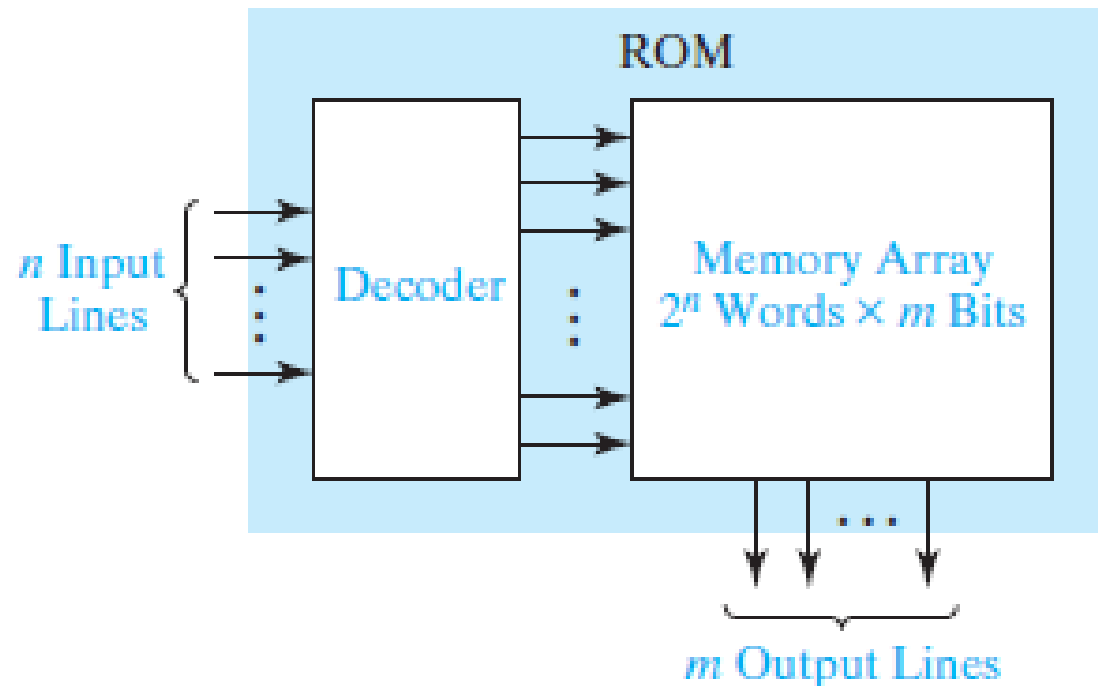
Read-Only Memory

• Basic ROM Structure:

- A ROM essentially consists of a decoder and a memory array
- When a pattern of n 0's and 1's is applied to the decoder inputs
 - Exactly one of the 2^n decoder outputs is 1
 - This decoder output line selects one of the words in the memory array
 - The bit pattern stored in this word is transferred to the memory output lines

FIGURE 9-23
Basic ROM
Structure

© Cengage Learning 2014

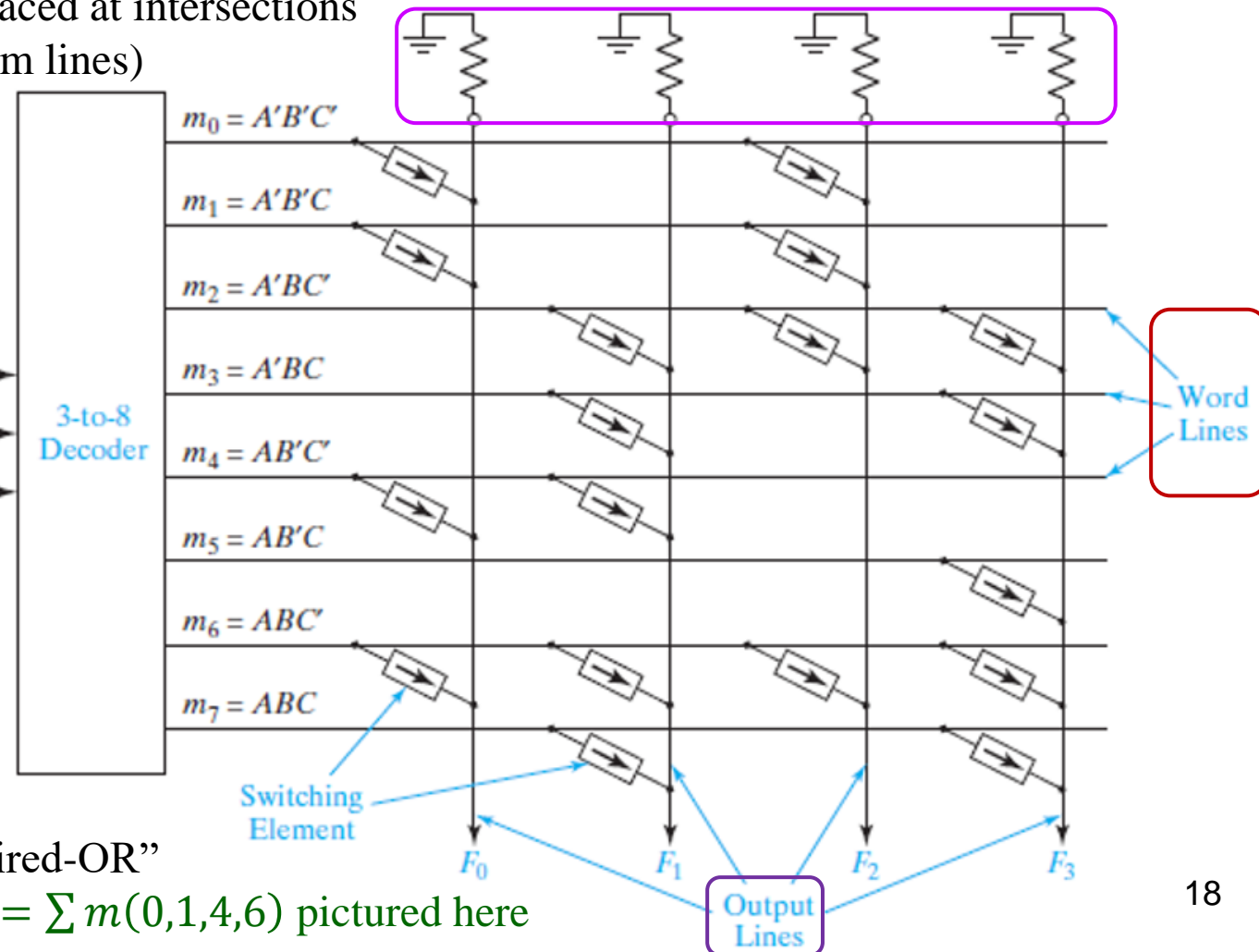




Read-Only Memory

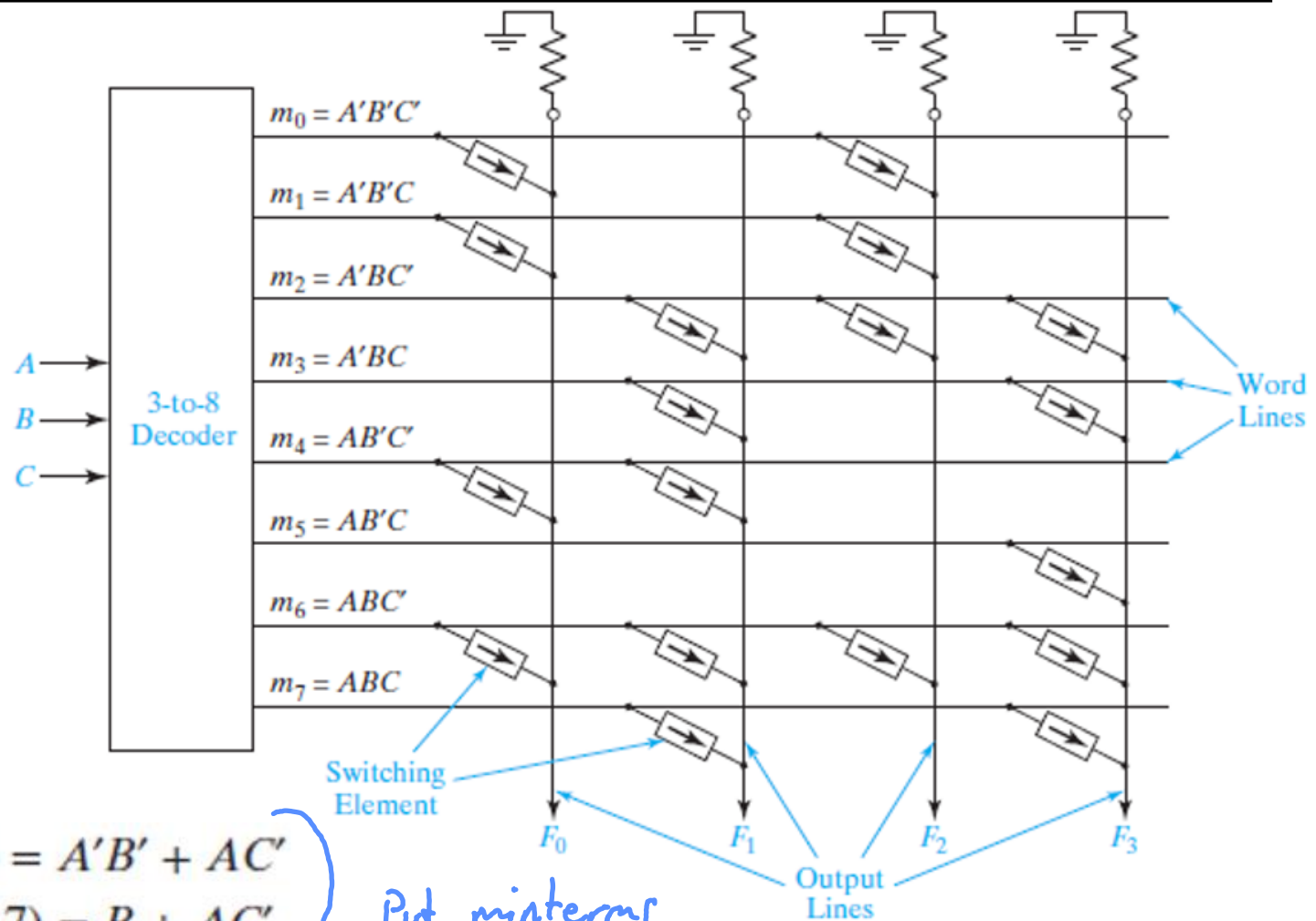
• One possible internal structure of an 8 Word \times 4-bit ROM:

- Decoder generates minterms and selects based on input address
- Switching elements placed at intersections of **Word Lines** (minterm lines) and **Output Lines** where a 1 is stored
- When any minterm that has a switching element attached is set to 1 a voltage A high is applied to that output line B C
- If no 1 is applied to an Output Line the **pull-down resistors** at top of array pull that Output Line to ground (**logic 0**)
- This implements a “wired-OR” gate. For example $F_0 = \sum m(0,1,4,6)$ pictured here





Read-Only Memory



- For this example, the four implemented functions are:

$$F_0 = \sum m(0, 1, 4, 6) = A'B' + AC'$$

$$F_1 = \sum m(2, 3, 4, 6, 7) = B + AC'$$

$$F_2 = \sum m(0, 1, 2, 6) = A'B' + BC'$$

$$F_3 = \sum m(2, 3, 5, 6, 7) = AC + B$$

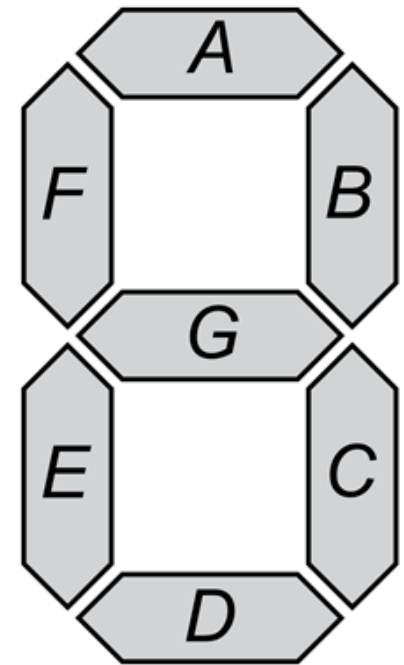
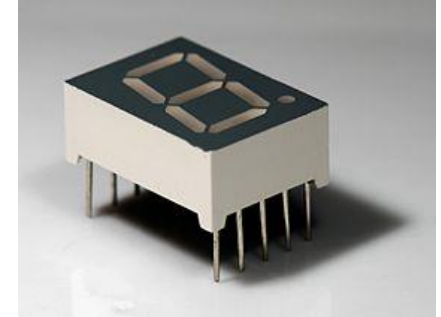
Put minterms
into four k-maps
and reduce to get these



Read-Only Memory

Example: ROM for seven-segment display driver

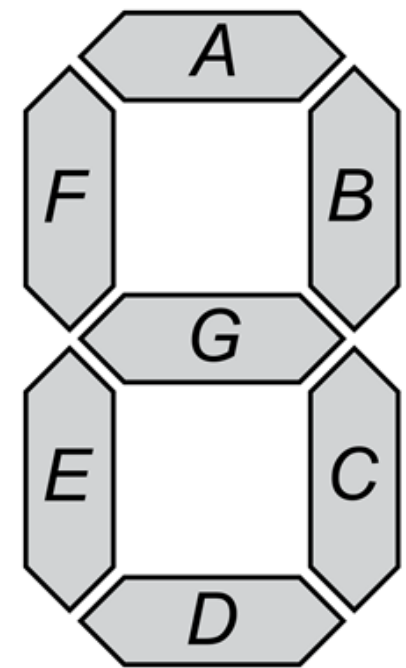
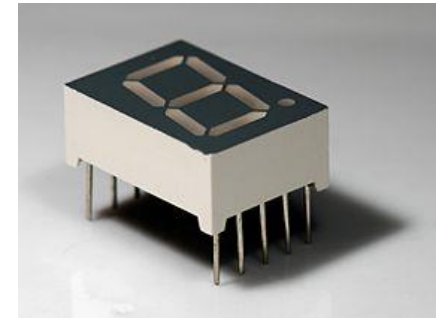
- Seven-segment display: an electronic display device for displaying decimal numerals
- Alternative to the more complex dot matrix displays
- Used in things such as digital clocks, electronic meters, basic calculators, etc.
 - Light up B and C → Displays a 1
 - Light up B, C, F and G → Displays a 4
 - Etc.
- Goal: ROM LUT for BCD to seven-segment driver





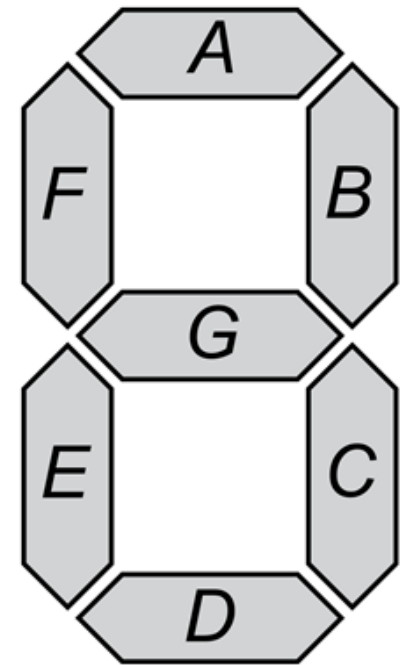
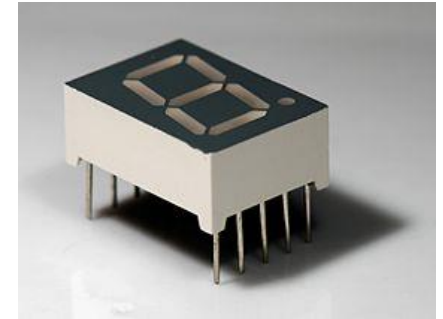
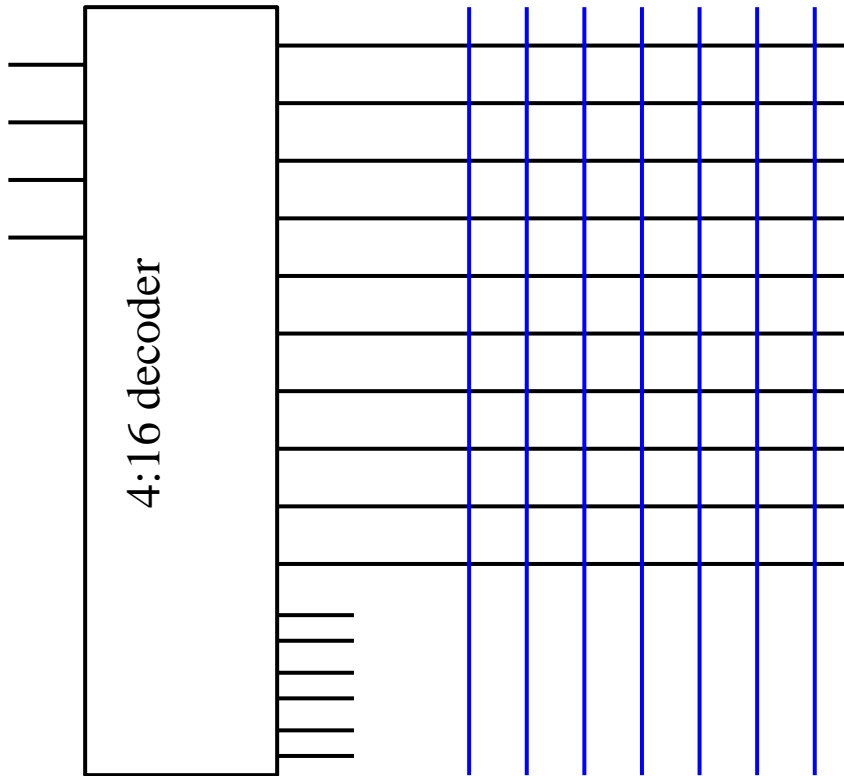
Read-Only Memory

<i>W</i>	<i>X</i>	<i>Y</i>	<i>Z</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							





Read-Only Memory





Types of ROM (Non-volatile memory)

- **Standard ROM**

- Permanent
- Switches between word and output lines hardwired in
- Either the switch is there, or it is not
- Done at “Mask-level” during fabrication of IC

- **Programmable ROM (PROM)**

- Can electronically set the locations of the “on” switches
- **Once**

- **Electronically Erasable PROM (EEPROM)**

- Can electronically reprogram
- On order of 100 to 1000 times
- Uses a special type of FET for the switches
- Takes large voltages to erase/reprogram
- Limited number of times before FET fails