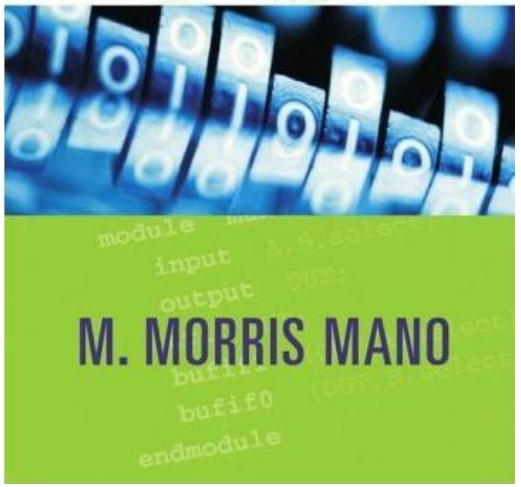


Digital Design 3e, Morris Mano

Chapter 4 – Combinational Logic

MODULAR DESIGN OF COMBINATIONAL CIRCUITS

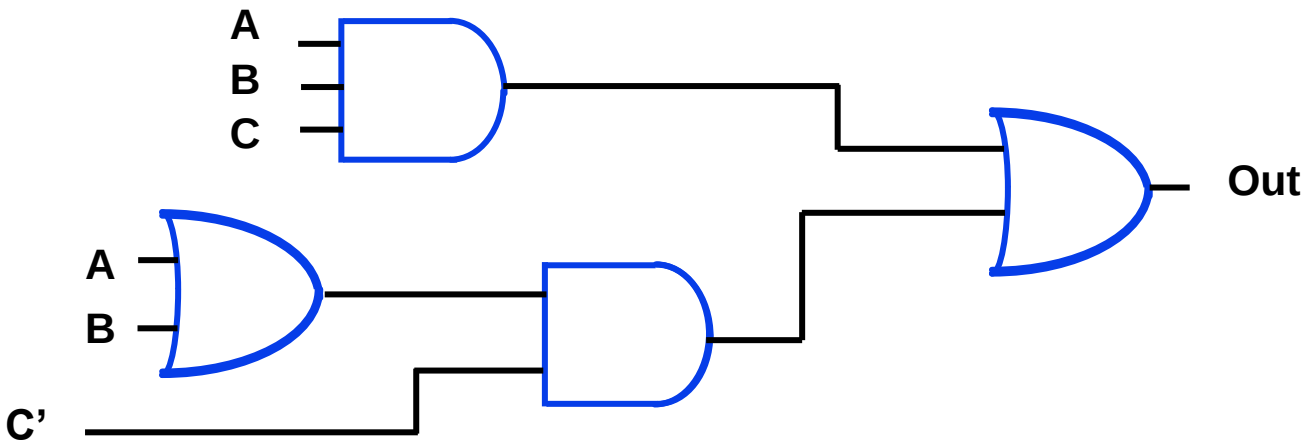
Circuit Analysis Procedure

Overview

- Important concept – analyze digital circuits
  - Given a circuit
    - Create a truth table
    - Create a minimized circuit
- Approaches
  - Boolean expression approach
  - Truth table approach
- Leads to minimized hardware
- Provides insights on how to design hardware
  - Tie in with K-maps (next time)

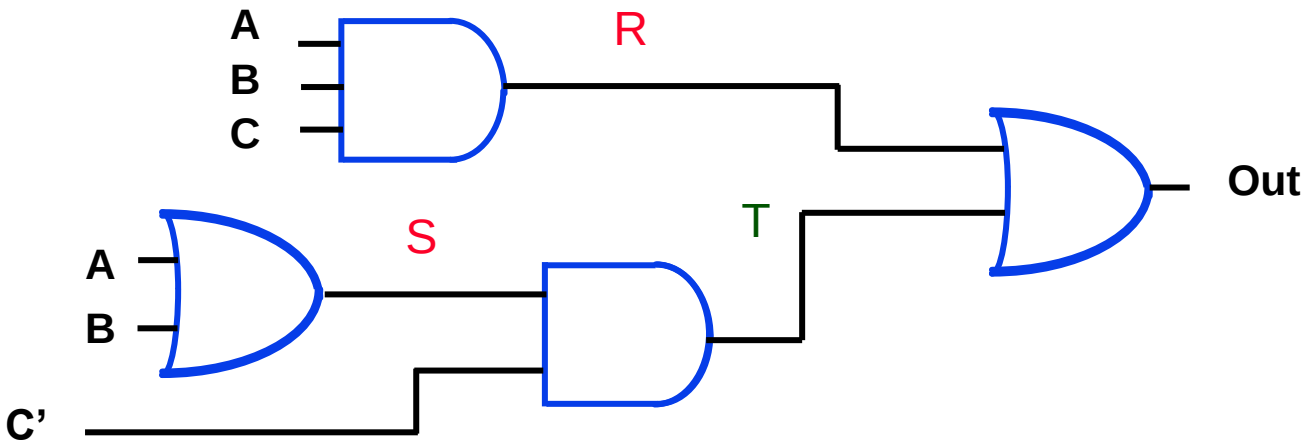
The Problem

- How can we convert from a circuit drawing to an equation or truth table?
- Two approaches
  - Create intermediate equations
  - Create intermediate truth tables



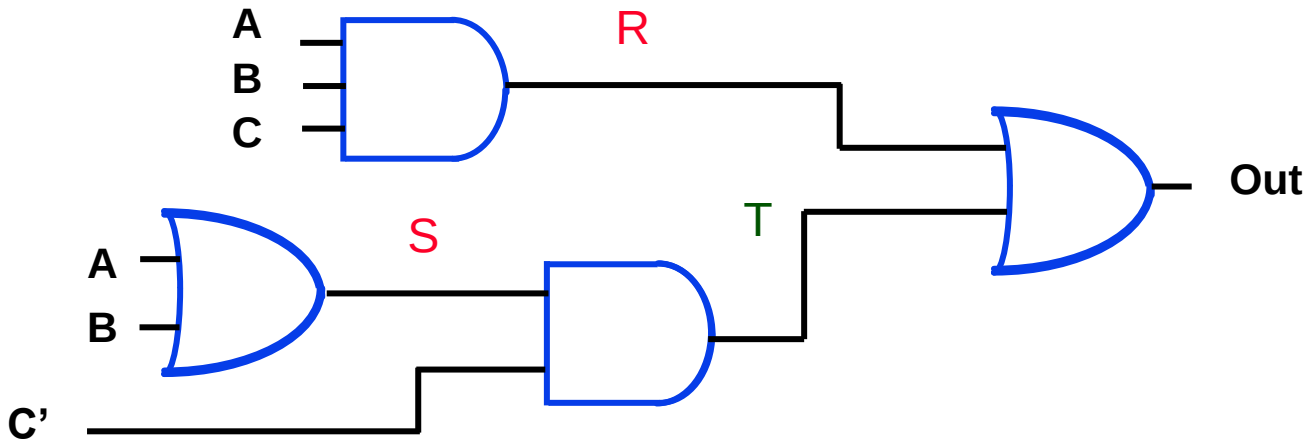
Label Gate Outputs

1. **Label all gate outputs that are a function of input variables.**
2. **Label gates that are a function of input variables and previously labeled gates.**
3. **Repeat process until all outputs are labelled.**



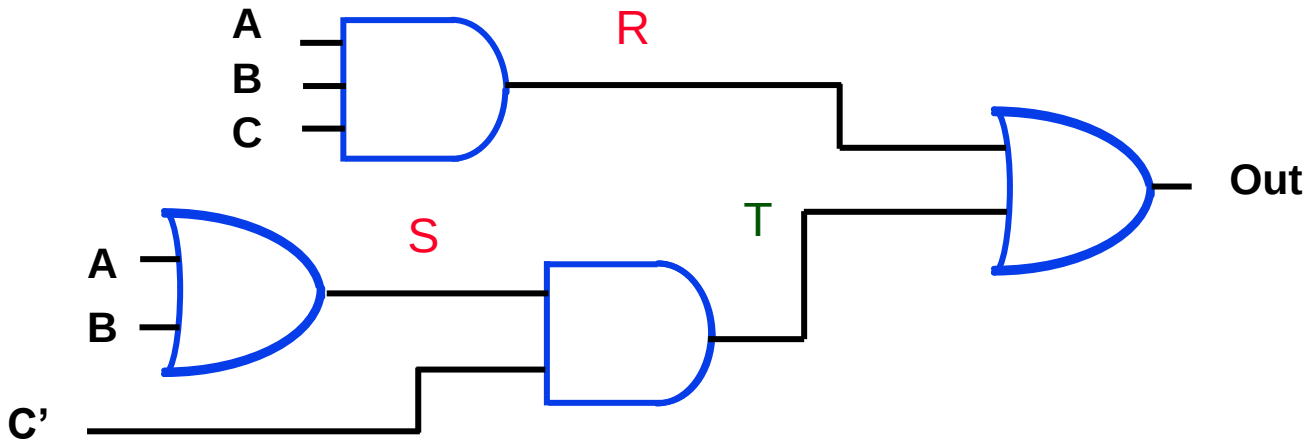
Approach 1: Create Intermediate Equations

- **Step 1: Create an equation for each gate output based on its input.**
  - **R = ABC**
  - **S = A + B**
  - **T = C'S**
  - **Out = R + T**



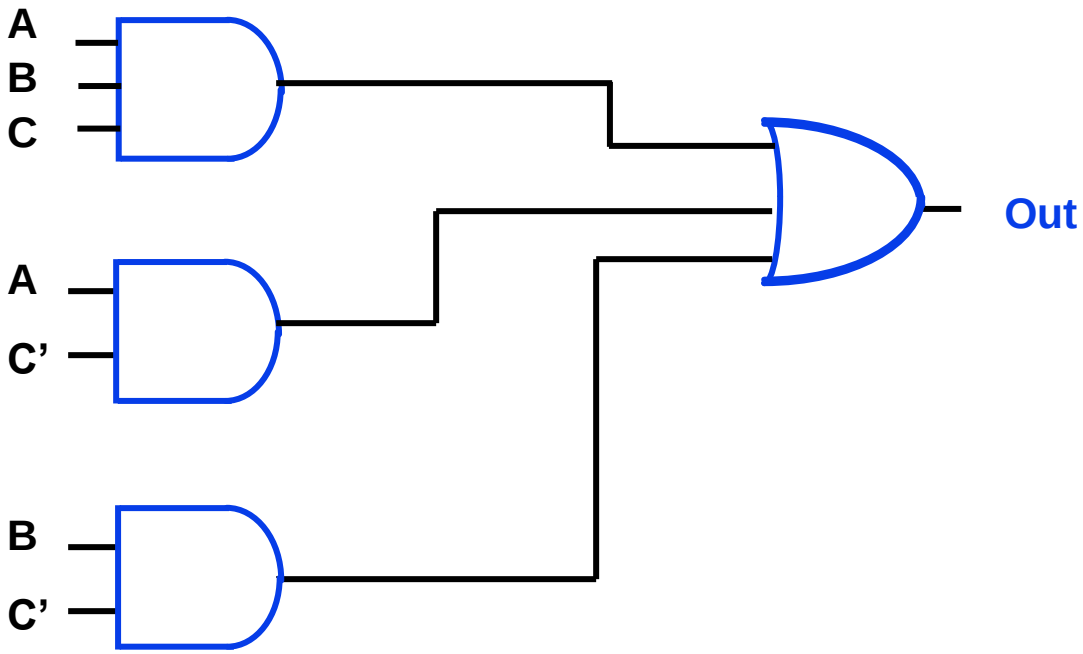
Approach 1: Substitute in subexpressions

- **Step 2: Form a relationship based on input variables (A, B, C)**
  - **R = ABC**
  - **S = A + B**
  - **T = C'S = C'(A + B)**
  - **Out = R+T = ABC + C'(A+B)**



Approach 1: Substitute in subexpressions

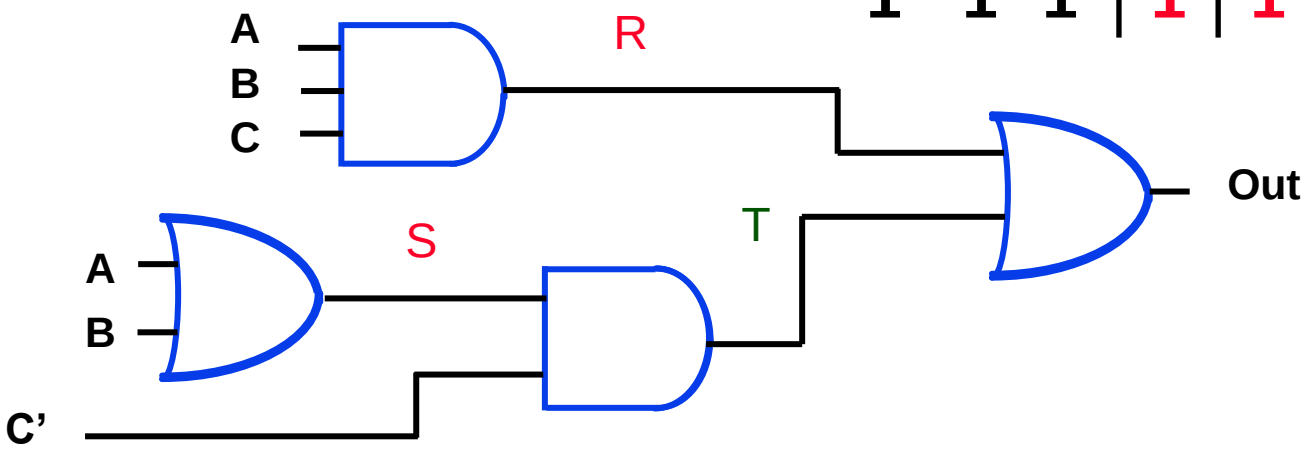
- **Step 3: Expand equation to SOP final result**
  - **Out = ABC + C'(A+B) = ABC + AC' + BC'**



Approach 2: Truth Table

- **Step 1: Determine outputs for functions of input variables.**

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

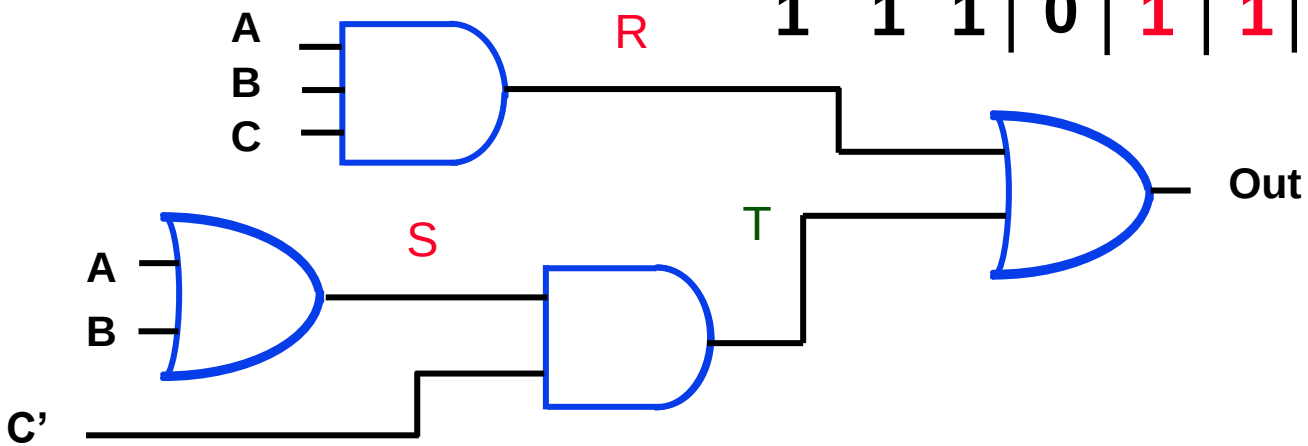


Approach 2: Truth Table

- **Step 2: Determine outputs for functions of intermediate variables.**

T = S * C’

A	B	C	C’	R	S	T
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	1
0	1	1	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	0	1	0
1	1	0	1	0	1	1
1	1	1	0	1	1	0

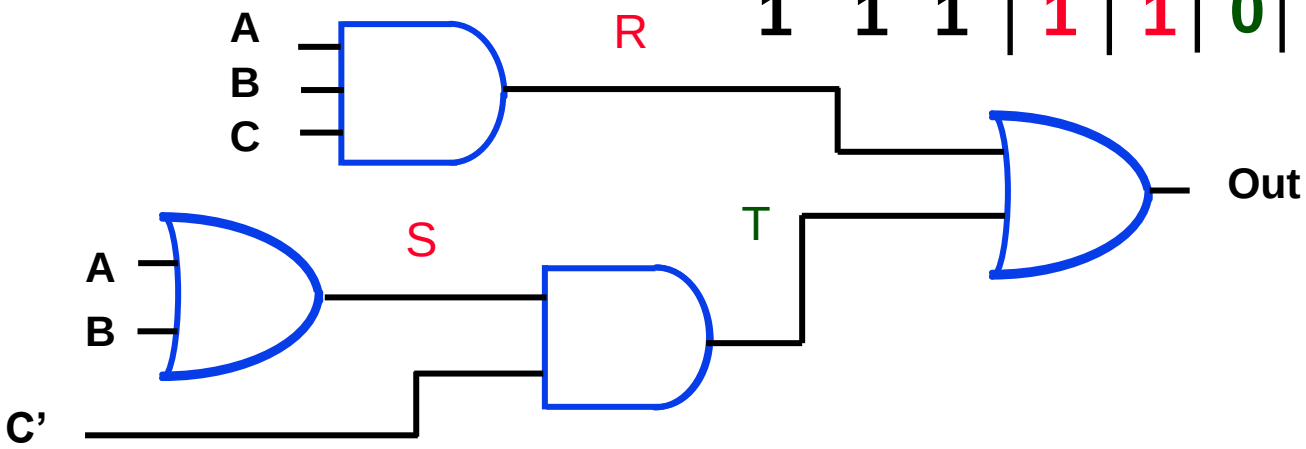


Approach 2: Truth Table

- **Step 3: Determine outputs for function.**

R + T = Out

A	B	C	R	S	T	Out
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	1	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	0	1	1	1
1	1	1	1	1	0	1



More Difficult Example

- **Step 3: Note labels on interior nodes**

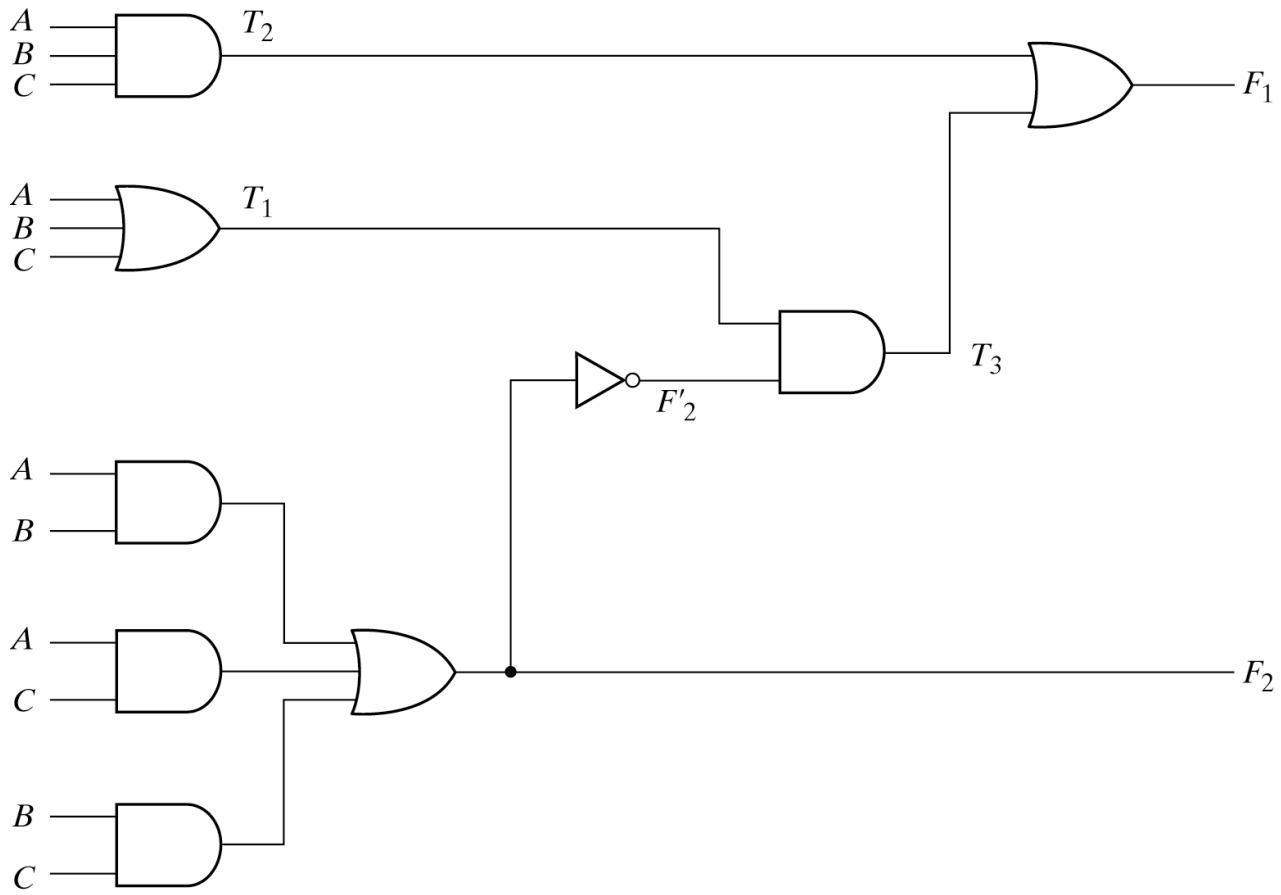


Fig. 4-2 Logic Diagram for Analysis Example

More Difficult Example: Truth Table

- Remember to determine intermediate variables starting from the inputs.
- When all inputs determined for a gate, determine output.
- The truth table can be reduced using K-maps.

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Summary

- Important to be able to convert circuits into truth table and equation form
  - WHY? ---- leads to minimized sum of product representation
- Two approaches illustrated
  - Approach 1: Create an equation with circuit output dependent on circuit inputs
  - Approach 2: Create a truth table which shows relationship between circuit inputs and circuit outputs
- Both results can then be minimized using K-maps.
- Next time: develop a minimized SOP representation from a high level description

Combinational Design Procedure

Overview

- Design digital circuit from specification
- Digital inputs and outputs known
  - Need to determine logic that can *transform* data
- Start in truth table form
- Create K-map for each output based on function of inputs
- Determine minimized sum-of-product representation
- Draw circuit diagram

Design Procedure (Mano)

Design a circuit from a specification.

1. **Determine number of required inputs and outputs.**
2. **Derive truth table**
3. **Obtain simplified Boolean functions**
4. **Draw logic diagram and verify correctness**

S = A + B + C
R = ABC

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

Previously, we have learned…

- Boolean algebra can be used to simplify expressions, but not obvious:
  - how to proceed at each step, or
  - if solution reached is minimal.
- Have seen five ways to represent a function:
  - Boolean expression
  - truth table
  - logic circuit
  - minterms/maxterms
  - Karnaugh map

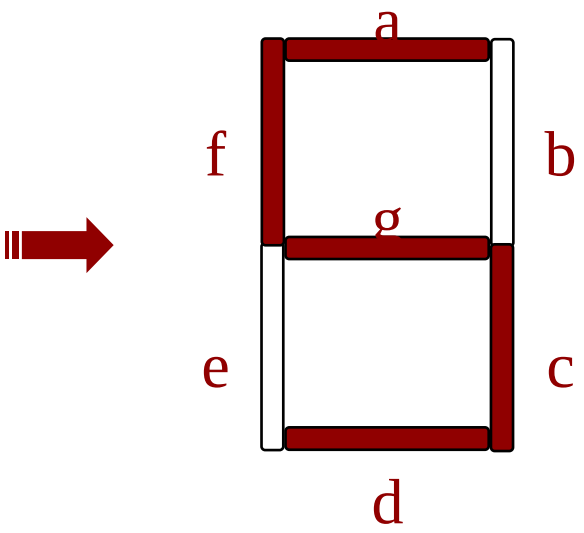
Combinational logic design

- Use multiple representations of logic functions
- Use graphical representation to assist in simplification of function.
- Use concept of “don’t care” conditions.
- Example - encoding BCD to seven segment display.
- Similar to approach used by designers in the field.

BCD to Seven Segment Display

- **Used to display binary coded decimal (BCD) numbers using seven illuminated segments.**
- **BCD uses 0's and 1's to represent decimal digits 0 - 9. Need four bits to represent required 10 digits.**
- **Binary coded decimal (BCD) represents each decimal digit with four bits**

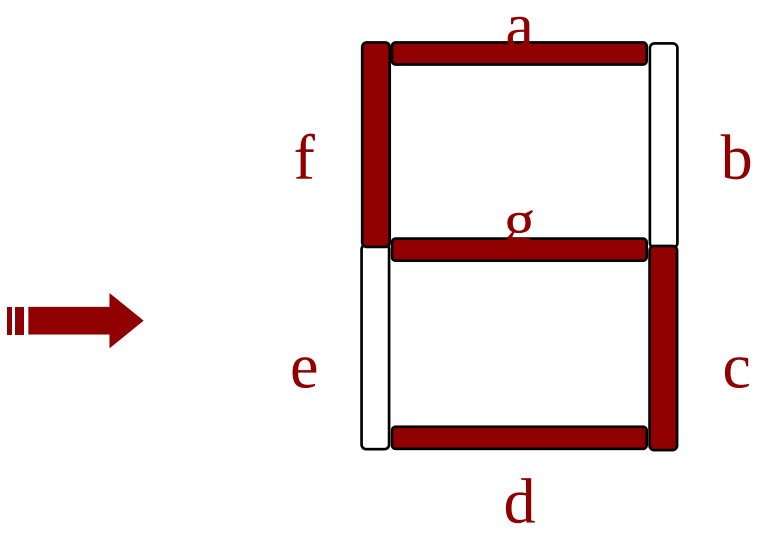
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
.	.	.	.	.
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



BCD to seven segment display

- **List the segments that should be illuminated for each digit.**

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



BCD to seven segment display

- **Derive the truth table for the circuit.**
- **Each output column in one circuit.**

	Inputs				Outputs					
Dec	w	x	y	z	a	b	c	d	e	.
0	0	0	0	0	1	1	1	1	1	.
1	0	0	0	1	0	1	1	0	0	.
2	0	0	1	0	1	1	0	1	1	.
.	.	.	.	.	.	.	.	.	.	.
7	0	1	1	1	1	1	1	0	0	.
8	1	0	0	0	1	1	1	1	1	.
9	1	0	0	1	1	1	1	1	0	.

BCD to seven segment display

- **Find minimal sum-of-products representation for each output**

For segment “a” :

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11				
10	1	1		

Note: Have only filled in ten squares, corresponding to the ten numerical digits we wish to represent.

Don’t care conditions (BCD display) ...

- **Fill in don’t cares for undefined outputs.**
  - **Note that these combinations of inputs should never happen.**
- **Leads to a reduced implementation**

For segment “a” :

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

Put in “X” (don’t care), and interpret as either 1 or 0 as desired ….

Don’t care conditions (BCD display) ...

- Circle biggest group of 1’s and Don’t Cares.
- Leads to a reduced implementation

For segment “a” :

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a1} = y$$

Don’t care conditions (BCD display)

- **Circle biggest group of 1’s and Don’t Cares.**
- **Leads to a reduced implementation**

For segment “a” :

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a2} = w$$

Don’t care conditions (BCD display) ...

- **Circle biggest group of 1’s and Don’t Cares.**
- **All 1’s should be covered by at least one implicant**

For segment “a” :

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a3} = \bar{x}\bar{z}$$

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F_{a4} = xz$$

Don’t care conditions (BCD display) ...

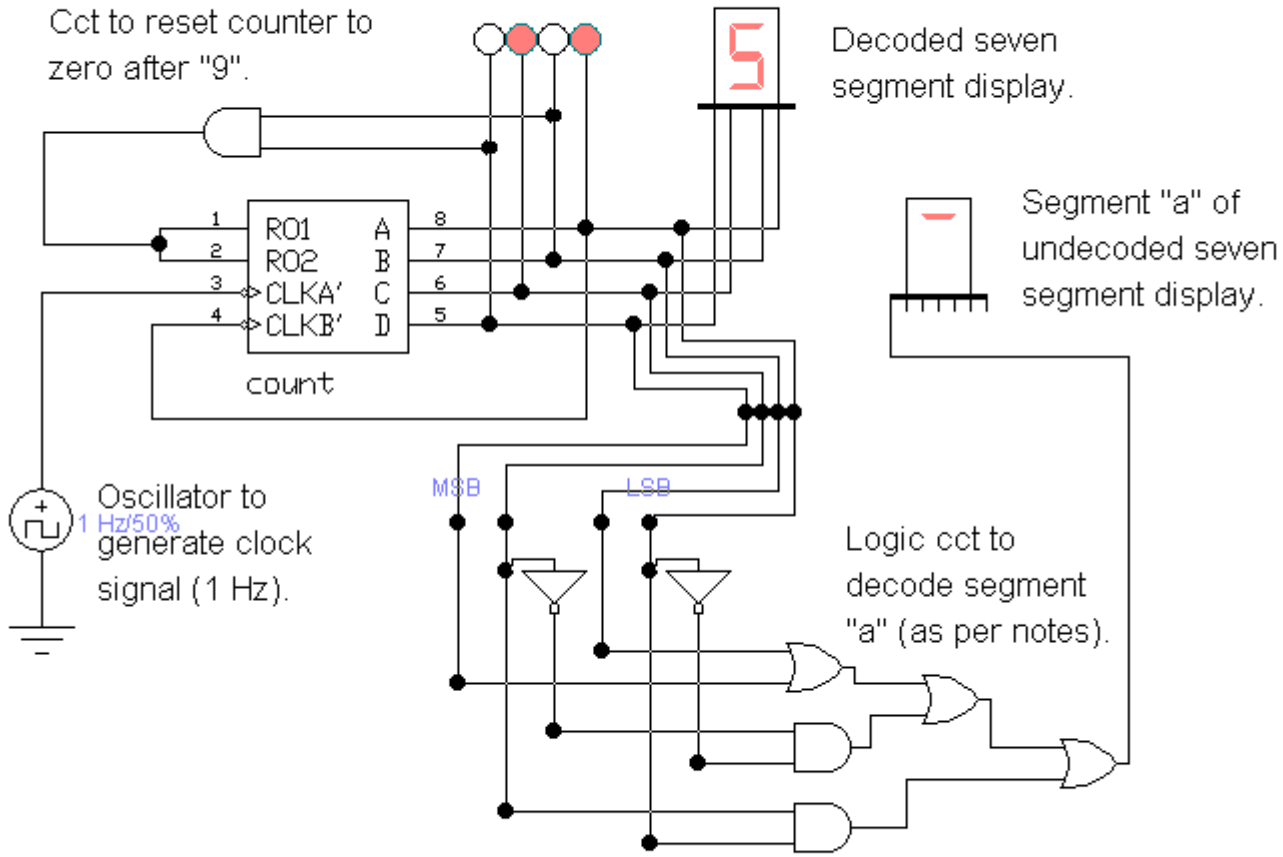
- **Put all the terms together**
- **Generate the circuit**

For segment “a” :

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$F = y + w + \bar{x}\bar{z} + xz$$

Example of seven segment display decoding.



Hint: Select a component and then push "?" from main menu bar to get info on what that component does and how it works.

BCD to seven segment display

- **Derive the truth table for the circuit.**
- **Each output column in one circuit.**

	Inputs				Outputs					
Dec	w	x	y	z	a	b	c	d	e	.
0	0	0	0	0	1	1	1	1	1	.
1	0	0	0	1	0	1	1	0	0	.
2	0	0	1	0	1	1	0	1	1	.
.	.	.	.	.	.	.	.	.	.	.
7	0	1	1	1	1	1	1	0	0	.
8	1	0	0	0	1	1	1	1	1	.
9	1	0	0	1	1	1	1	1	0	.

BCD to seven segment display

- **Find minimal sum-of-products representation for each output**

For segment "b" :

wx \ yz	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11				
10	1	1		

See if you complete this example.

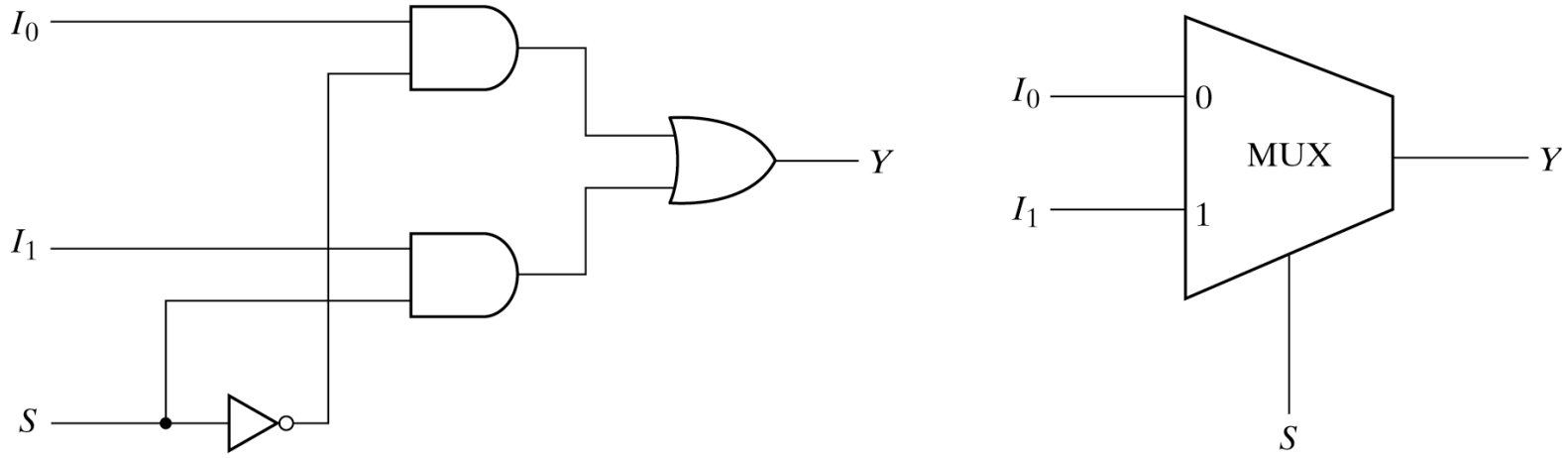
Summary

- **Need to formulate circuits from problem descriptions**
  1. **Determine number of inputs and outputs**
  2. **Determine truth table format**
  3. **Determine K-map**
  4. **Determine minimal SOP**
- **There may be multiple outputs per design**
  - **Solve each output separately**
- **Current approach doesn't have memory.**
  - **This will be covered next week.**

Multiplexers

Multiplexers

- Select an input value with one or more select bits
- Use for transmitting data
- Allows for conditional transfer of data
- Sometimes called a mux

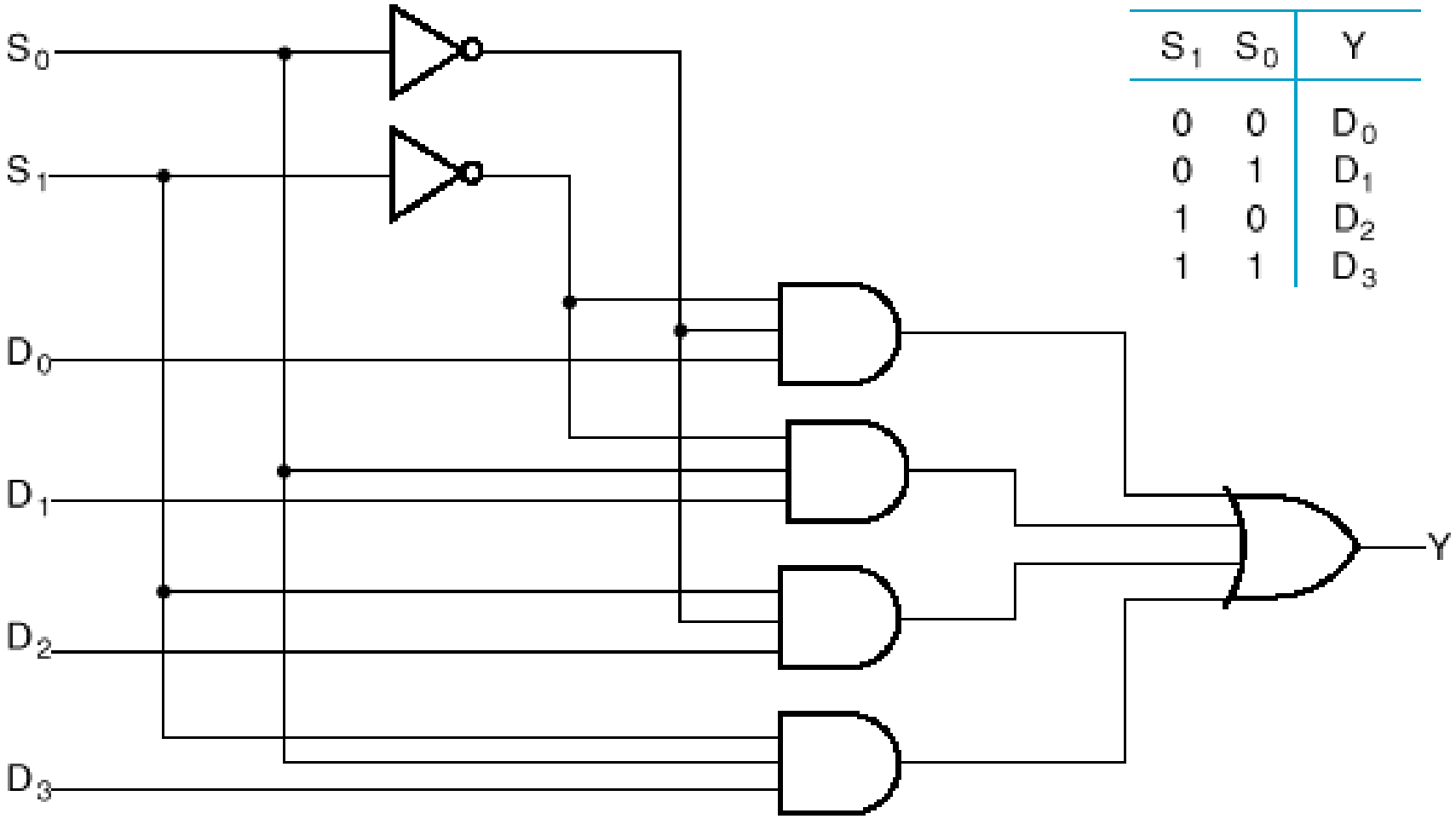


(a) Logic diagram

(b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer

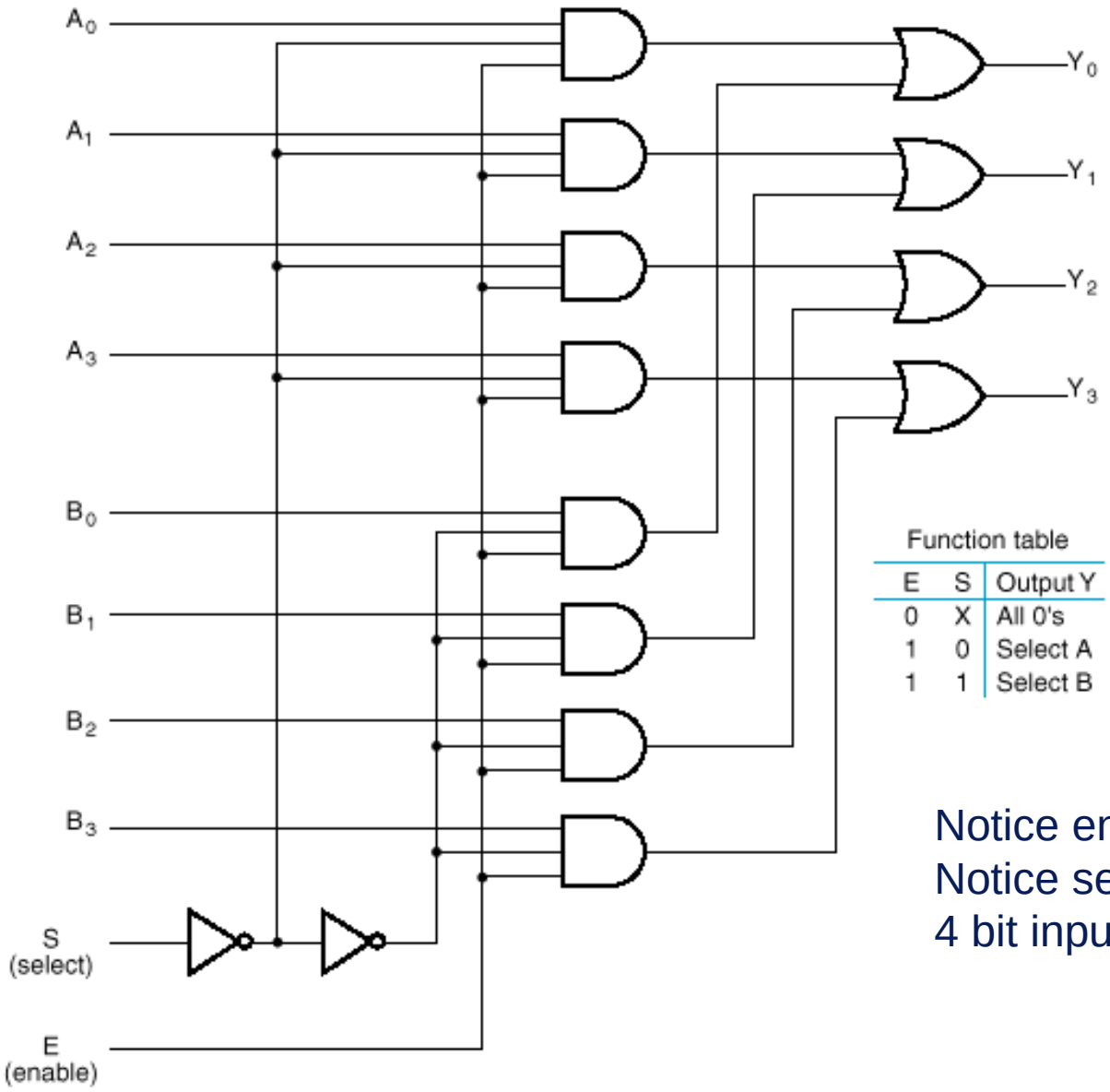
4– to– 1- Line Multiplexer



Function table

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

Quadruple 2–to–1-Line Multiplexer



Function table

E	S	Output Y
0	X	All 0's
1	0	Select A
1	1	Select B

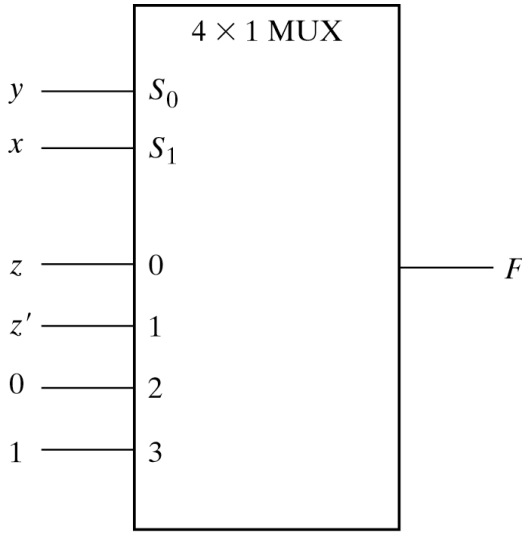
Notice enable bit
Notice select bit
4 bit inputs

Multiplexer as combinational modules

- **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

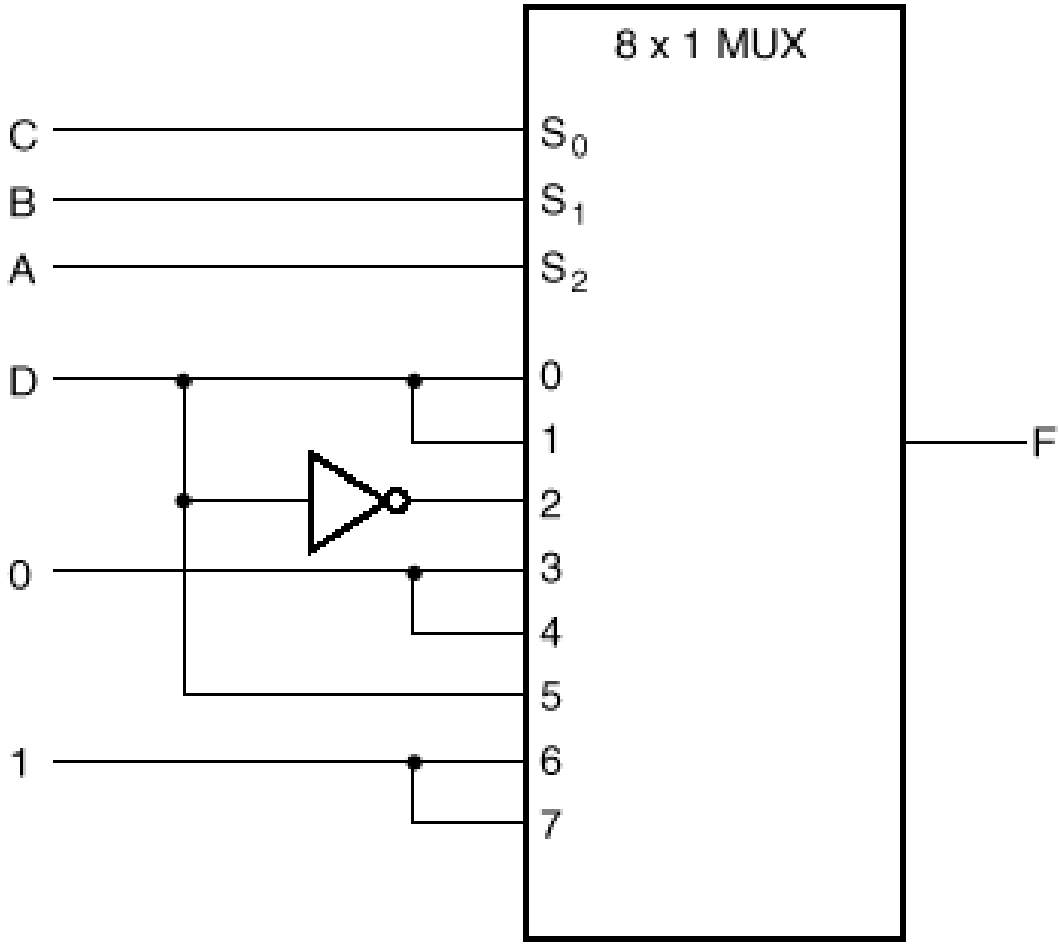


(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

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	



Typical multiplexer uses

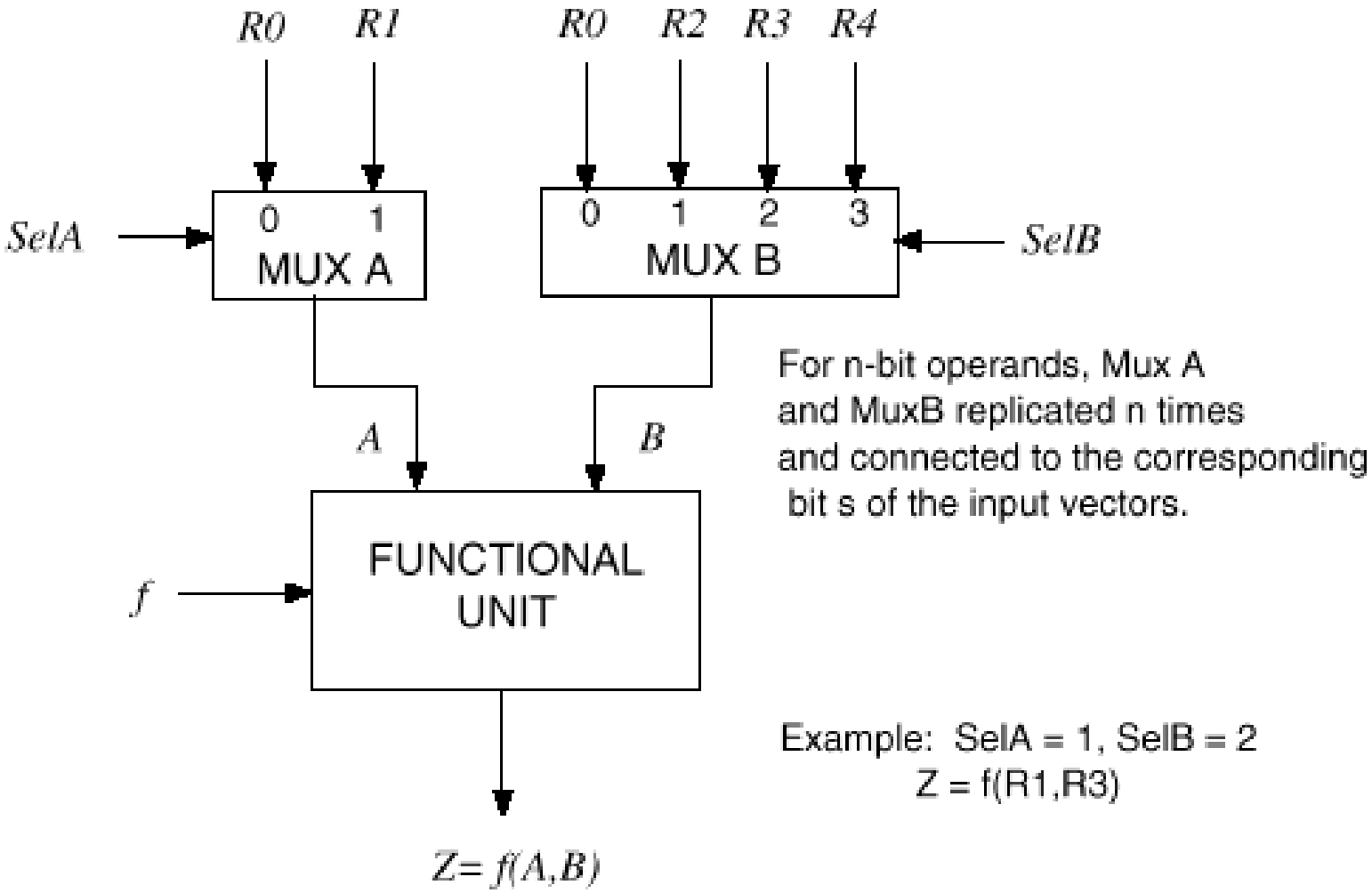
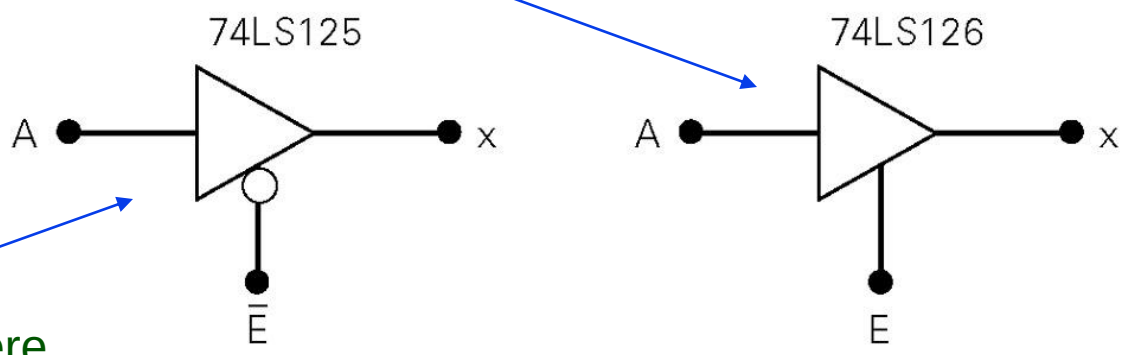


Figure 9.21: Multiplexer example of use.

Three-state gates

- **A multiplexer can be constructed with three-state gates**
- **Output state: 0, 1, and high-impedance (open ckts)**
- **If the select input (E) is 0, the three-state gate has no output**



Opposite true here,
No output if $\overline{E}$ is 1

$\overline{E}$	x
0	A
1	Hi-Z

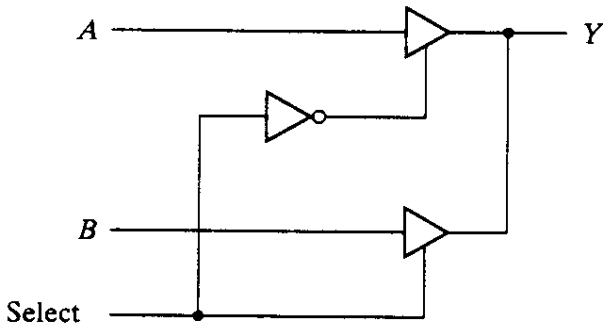
(a)

E	x
0	Hi-Z
1	A

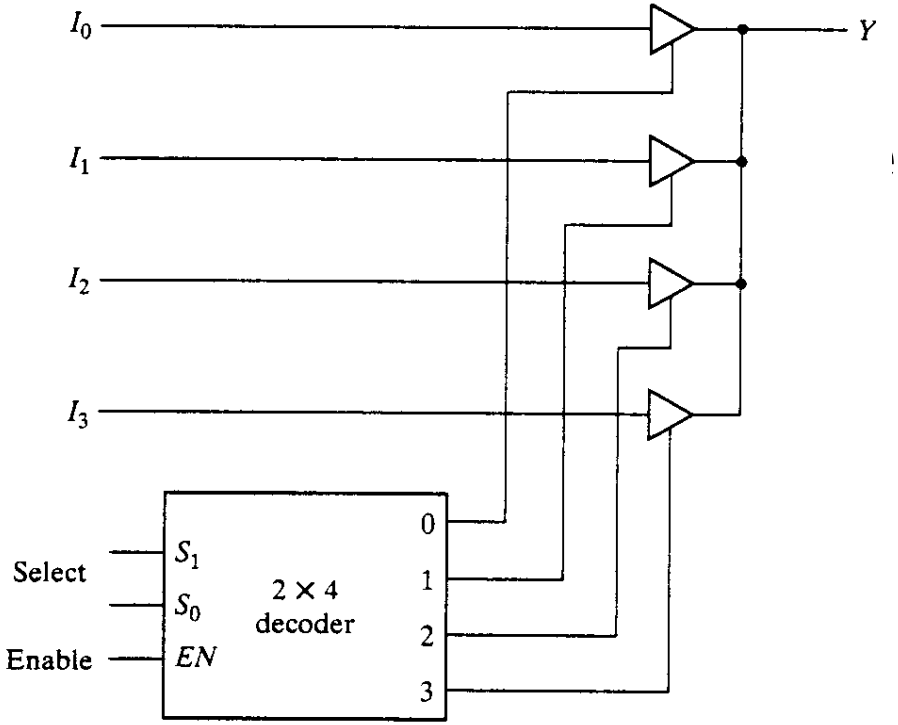
(b)

Three-state gates

- **A multiplexer can be constructed with three-state gates**
- **Output state: 0, 1, and high-impedance**



(a) 2-to-1- line mux



(b) 4 - to - 1 line mux

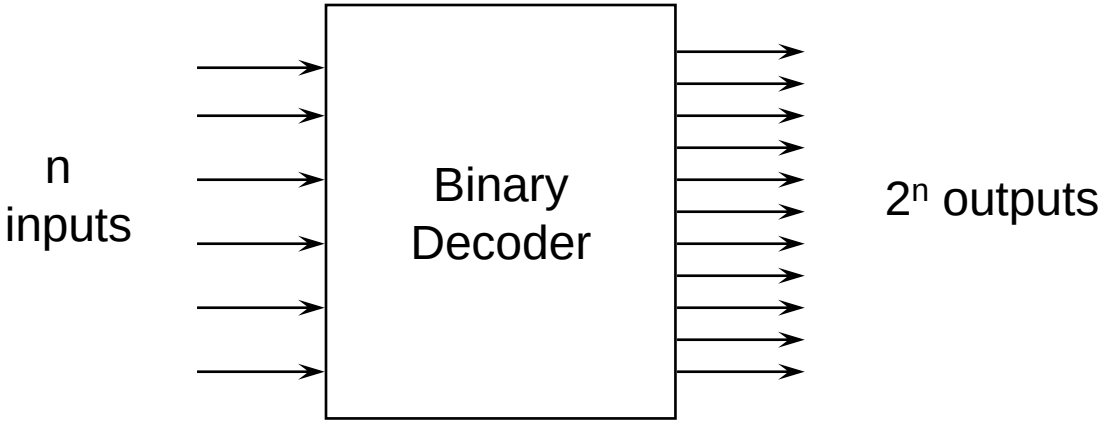
Encoders and Decoders

Overview

- Binary decoders
  - Converts an n-bit code to a single active output
  - Can be developed using AND/OR gates
  - Can be used to implement logic circuits.
- Binary encoders
  - Converts one of 2^n inputs to an n-bit output
  - Useful for compressing data
  - Can be developed using AND/OR gates
- Both encoders and decoders are extensively used in digital systems

Binary Decoder

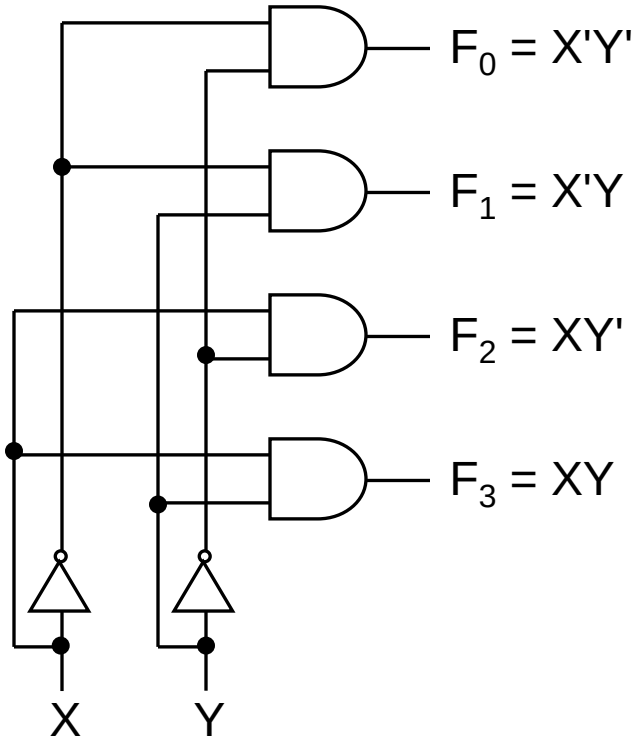
- **Black box with n input lines and 2^n output lines**
- **Only one output is a 1 for any given input**



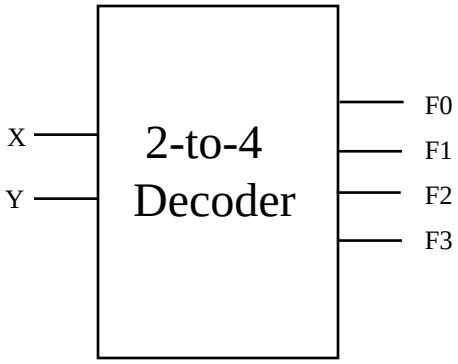
2-to-4 Binary Decoder

Truth Table:

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



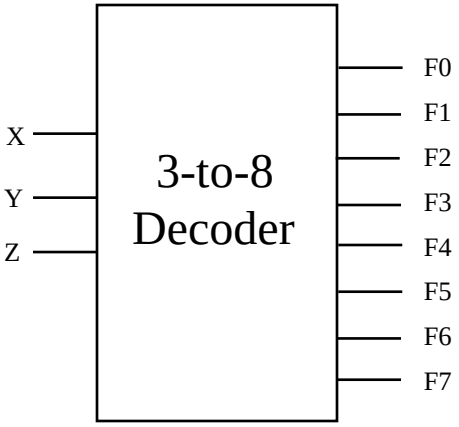
- **From truth table, circuit for 2x4 decoder is:**
- **Note: Each output is a 2-variable minterm (**X'Y', X'Y, XY' **or** XY**)**

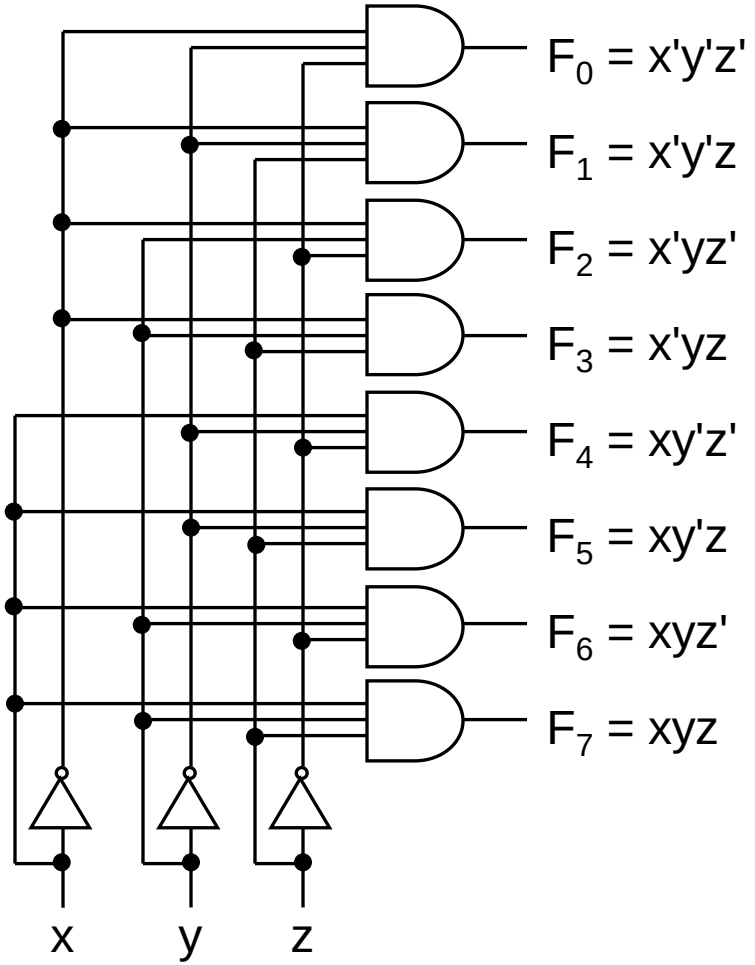


3-to-8 Binary Decoder

Truth Table:

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





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.**

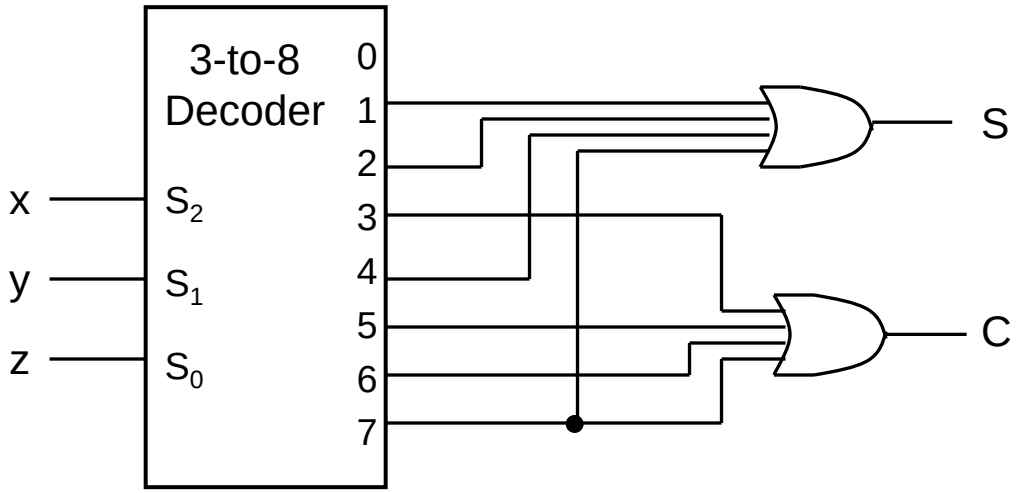
Implementing Functions Using Decoders

° **Example: Full adder**

S(x, y, z) = Σ (1,2,4,7)

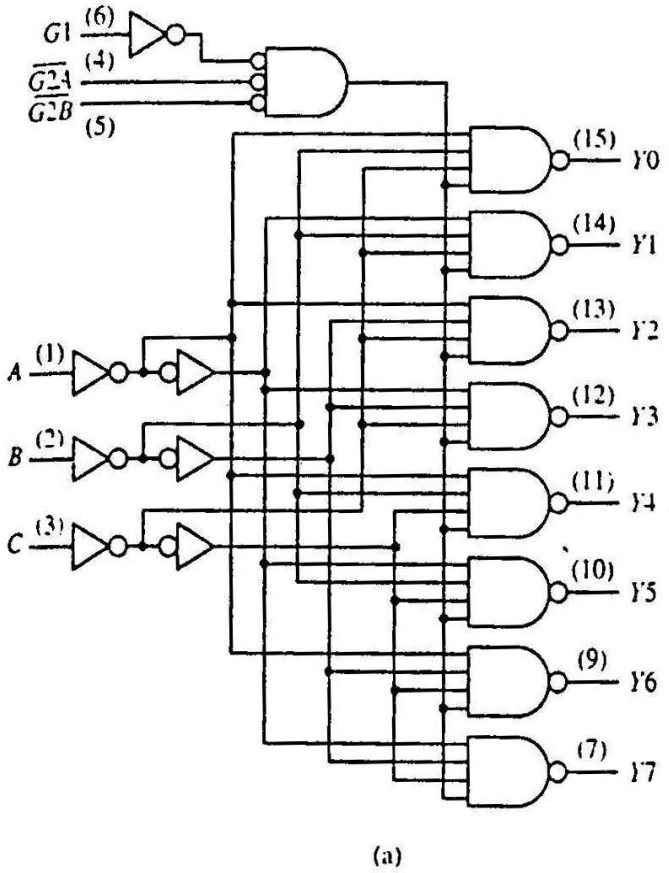
C(x, y, z) = Σ (3,5,6,7)

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

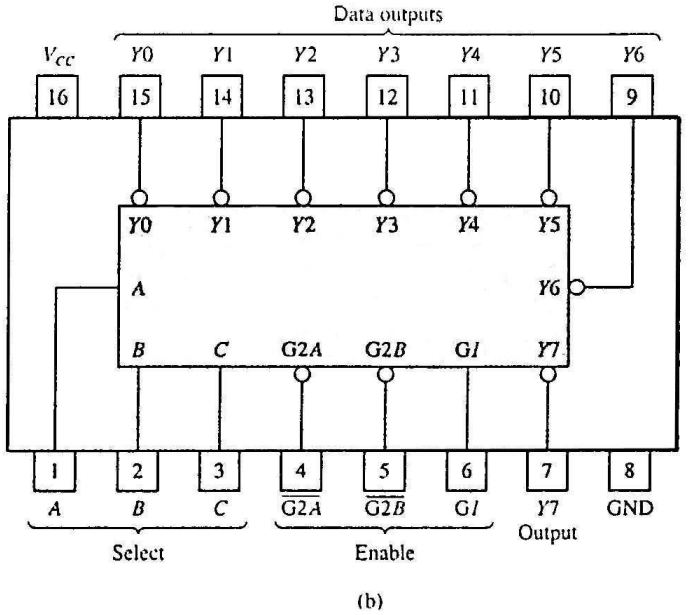


Standard MSI Binary Decoders Example

74138 (3-to-8 decoder)



(a)



(b)

Inputs					Outputs							
Enable		Select										
G1	$\overline{G2^*}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L
×	H	×	×	×	H	H	H	H	H	H	H	H
L	×	×	×	×	H	H	H	H	H	H	H	H

$\overline{G2^*} = \overline{G2A} + \overline{G2B}$

(c)

(a) Logic circuit.
(b) Package pin configuration.
(c) Function table.

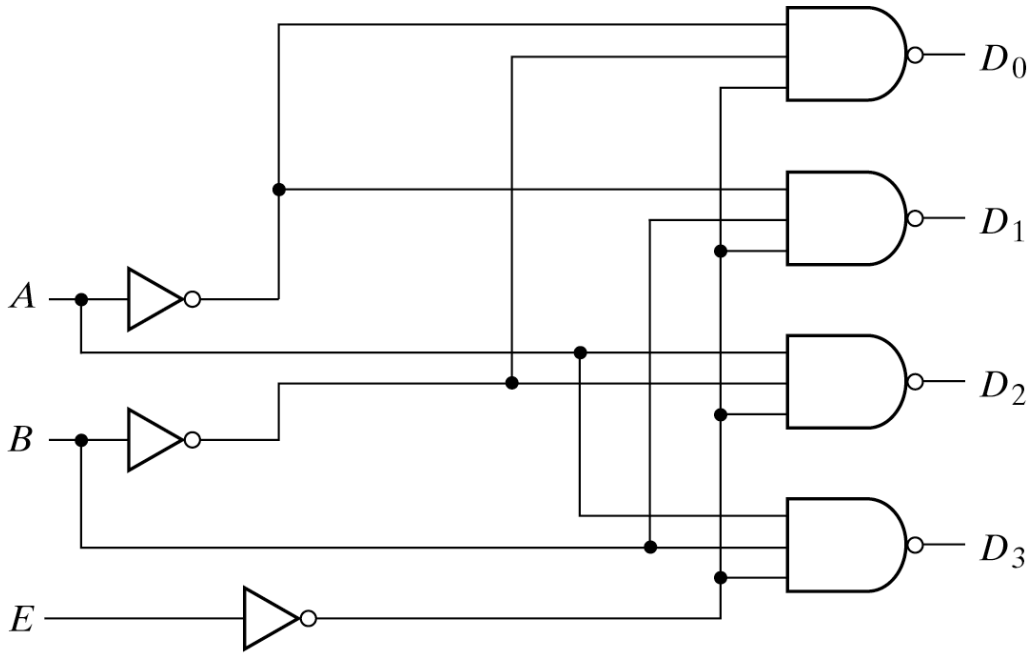
Building a Binary Decoder with NAND Gates

- **Start with a 2-bit decoder**
  - **Add an enable signal (E)**

Note: use of NANDs

only one 0 active!

if E = 0



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_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

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

Use two 3 to 8 decoders to make 4 to 16 decoder

- Enable can also be active high
- In this example, only one decoder can be active at a time.
- x, y, z effectively select output line for w

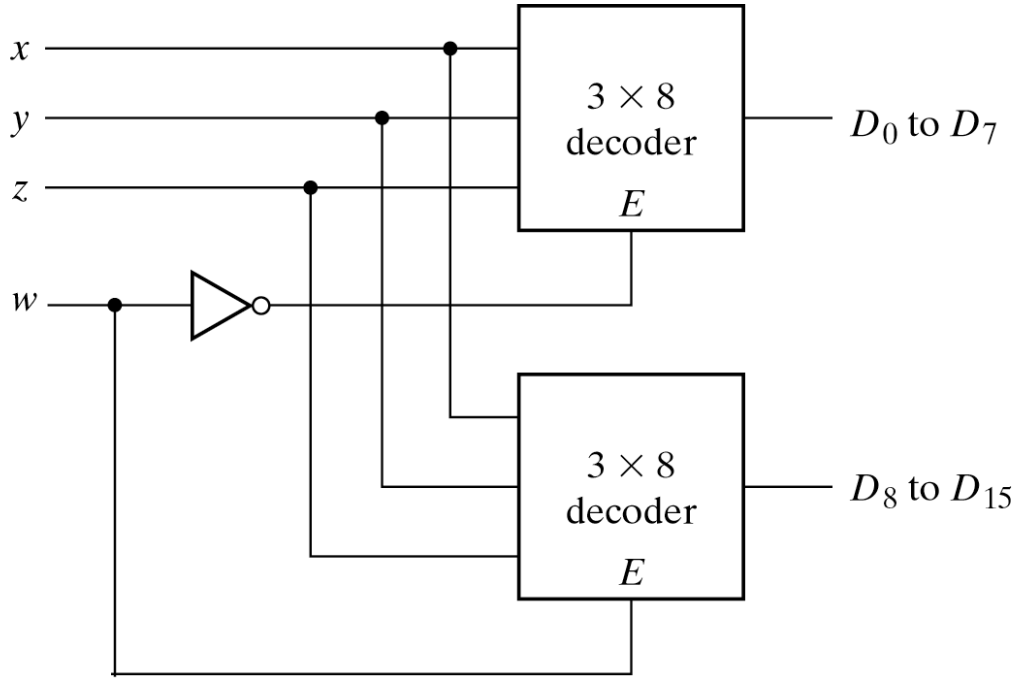


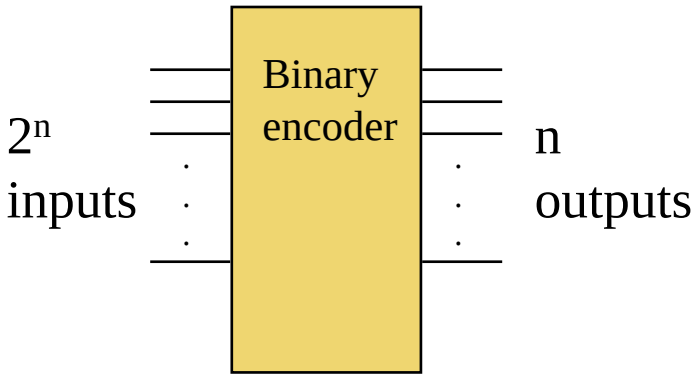
Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

Encoders

- **If the a decoder's output code has fewer bits than the input code, the device is usually called an encoder.**

  e.g. 2^n-to-n

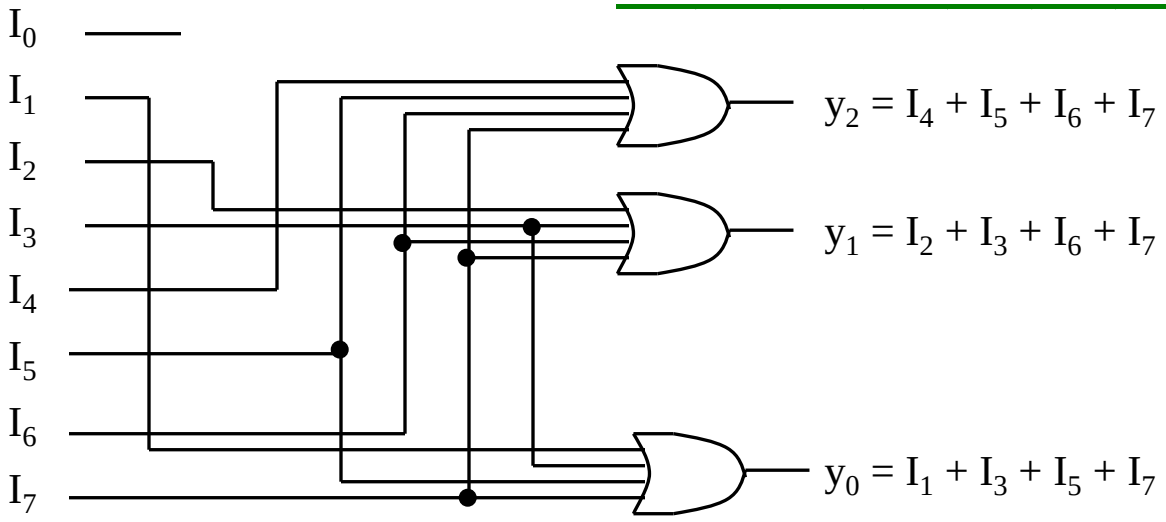
- **The simplest encoder is a 2^n-to-n binary encoder**
  - **One of 2^n inputs = 1**
  - **Output is an n-bit binary number**



8-to-3 Binary Encoder

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

Inputs								Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_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	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



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.
- Note that polarity of Idle is opposite from Table 4-8 in Mano

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

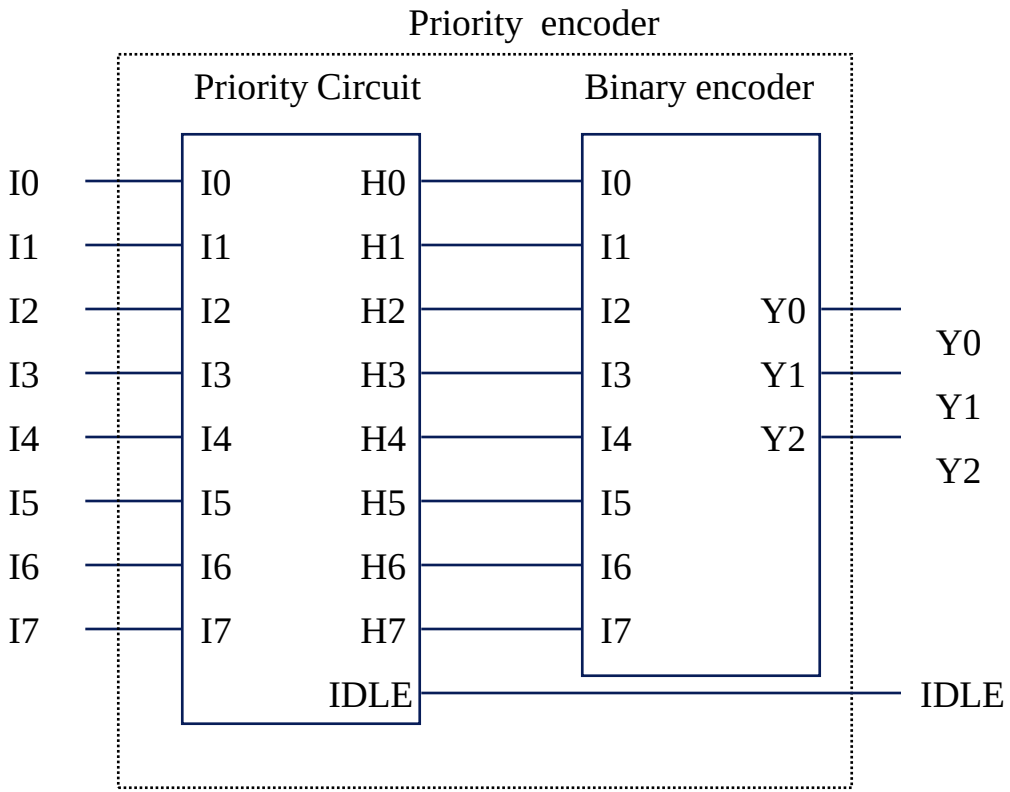
Priority Encoder (8 to 3 encoder)

- **Assign priorities to the inputs**
- **When more than one input are asserted, the output generates the code of the input with the highest priority**
- **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'

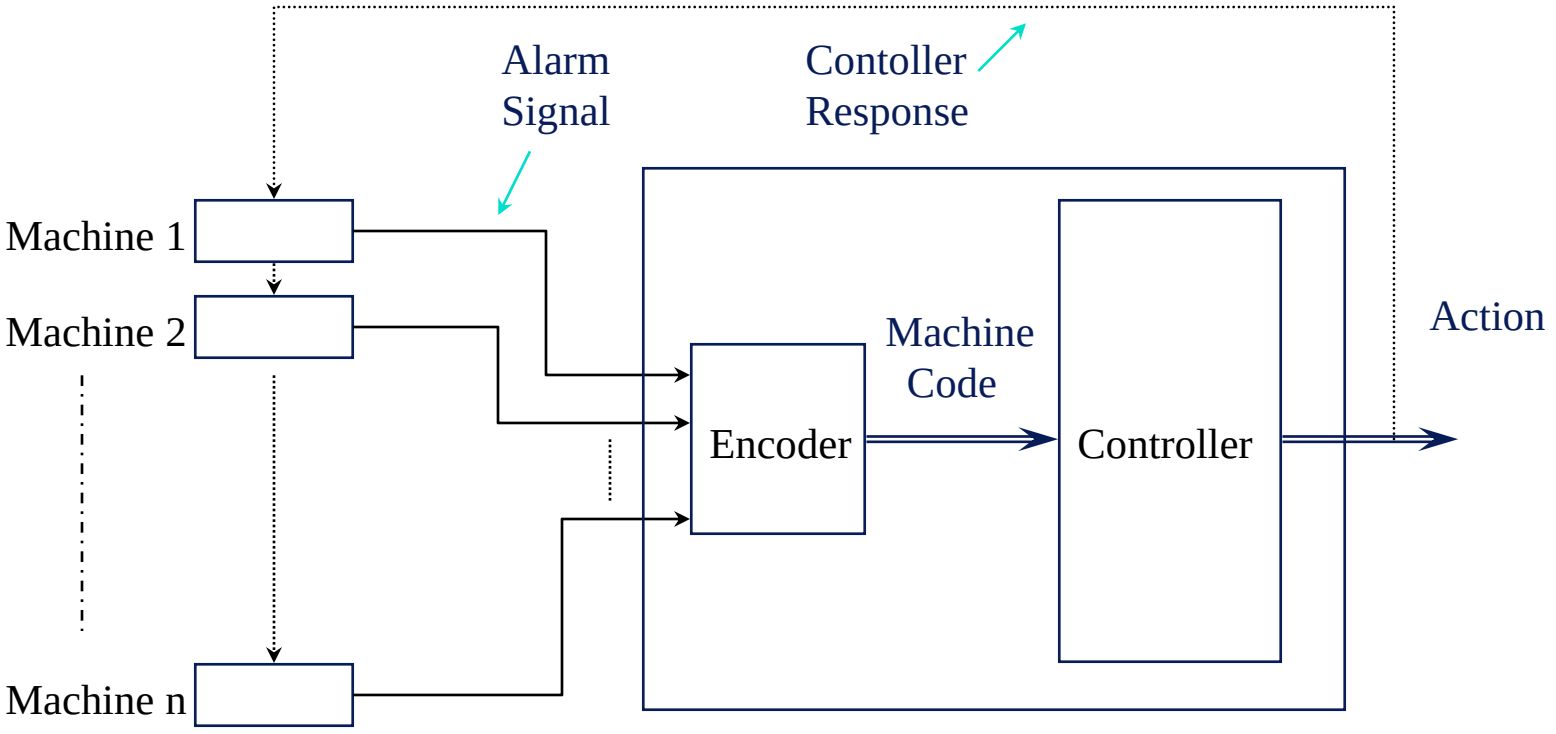
- **Encoder**

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



Encoder Application (Monitoring Unit)

Encoder identifies the requester and encodes the value
Controller accepts digital inputs.



Summary

- Decoder allows for generation of a single binary output from an input binary code
  - For an n-input binary decoder there are 2^n outputs
- Decoders are widely used in storage devices (e.g. memories)
  - We will discuss these in a few weeks
- Encoders all for data compression
- Priority encoders rank inputs and encode the highest priority input
- Next time: storage elements!