

DIGITAL LOGIC

Chapter 4 part2: Standard Components

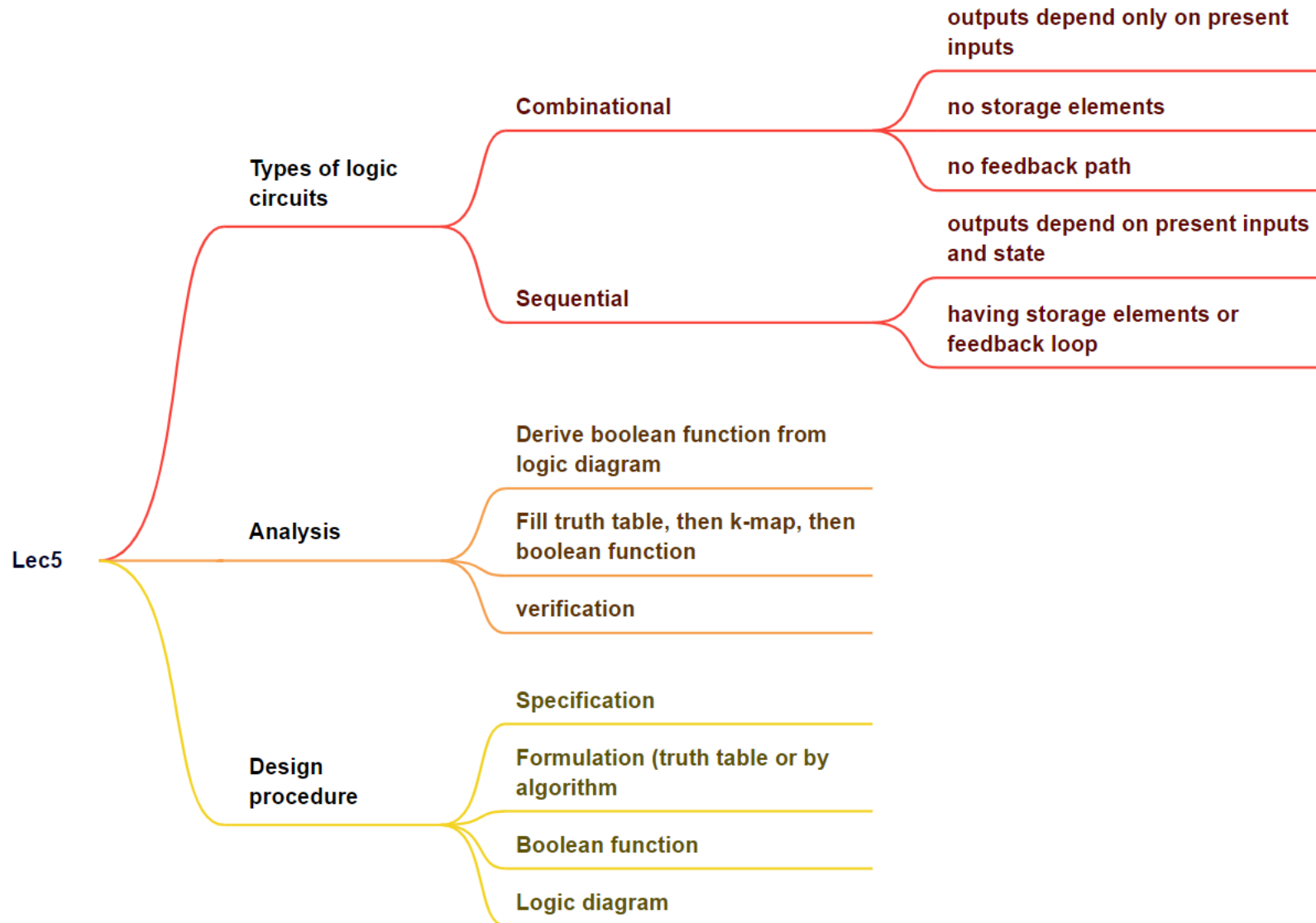
2023 Fall

Today's Agenda

- Recap
- Context
 - Decoder
 - Multiplexer
 - Encoder
- Reading: Textbook, Chapter 4.9-4.11
 - Next Lecture we continue to chapter 5
 - Arithmetic Logic will be taught later



Recap





Outline

- **Decoder**
- Multiplexer
- Encoder
- Gate Behavior

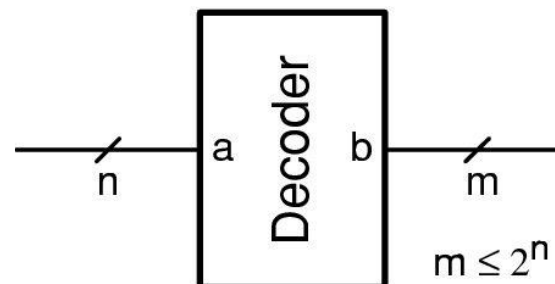
One-hot Representation

- Represent a set of N elements with N bits
- Exactly one bit is set

Binary	One-hot
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

Decoder

- A decoder is a combinational circuit that converts binary information from n input lines to m (maximum of 2^n) unique output lines
 - n -to- m -line decoder
- A binary one-hot decoder converts a symbol from binary code to a one-hot code
 - Output variables are mutually exclusive because only one output can be equal to 1 at any time (the 1-minterm)
- Example
 - binary input a to one-hot output b
 - $b[i] = 1$ if $a = i$ or $b = 1 \ll a$
 - a stands for position of 1 in b



1-to-2-Line Decoder

- Step1: Specification
- Step2: Formulation

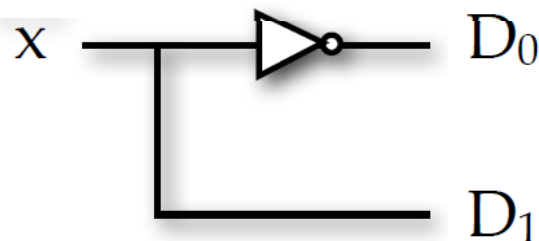
x	D ₁	D ₀
0	0	1
1	1	0

- Step3: Optimization

$$D_0 = x' \quad \swarrow \text{minterms}$$

$$D_1 = x \quad \swarrow$$

- Step4: Logic Diagram



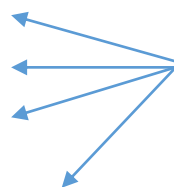
2-to-4-Line Decoder

Step 1,2

a_1	a_0	b_3	b_2	b_1	b_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

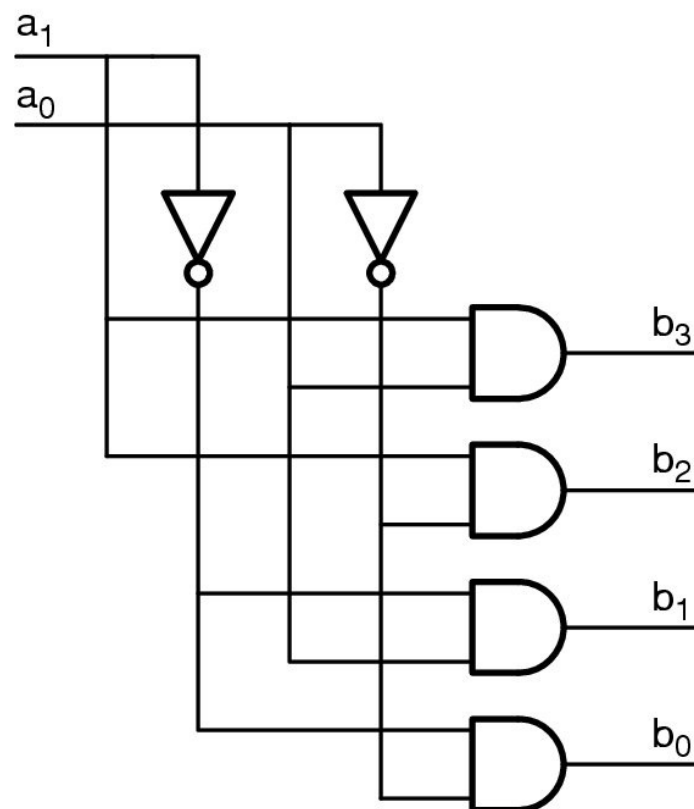
Step 3

$$\begin{aligned} b_3 &= a_1 a_0 \\ b_2 &= a_1 a_0' \\ b_1 &= a_1' a_0 \\ b_0 &= a_1' a_0' \end{aligned}$$



minterms

Step 4

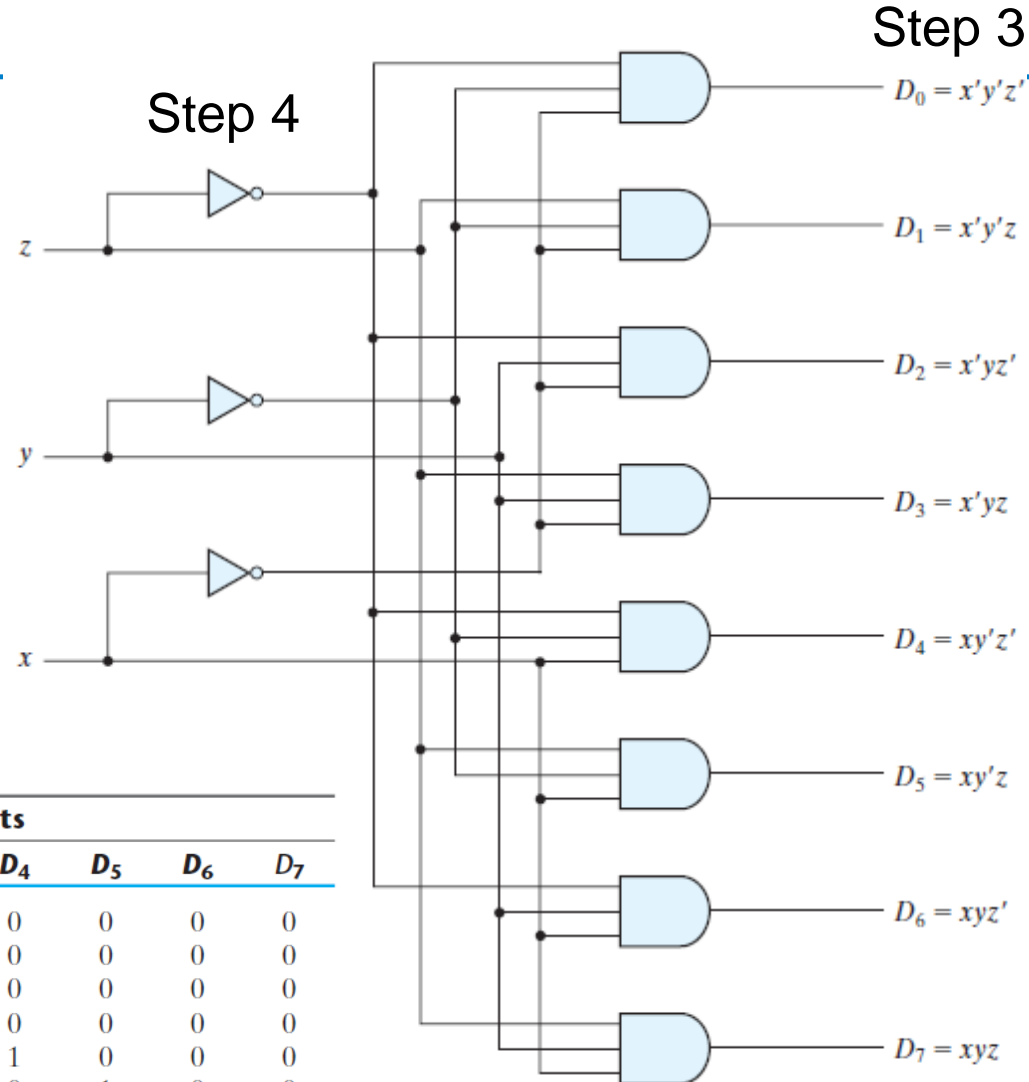


3-to-8-Line Decoder

- Each output of the decoder represents one of the eight minterms of the Boolean function

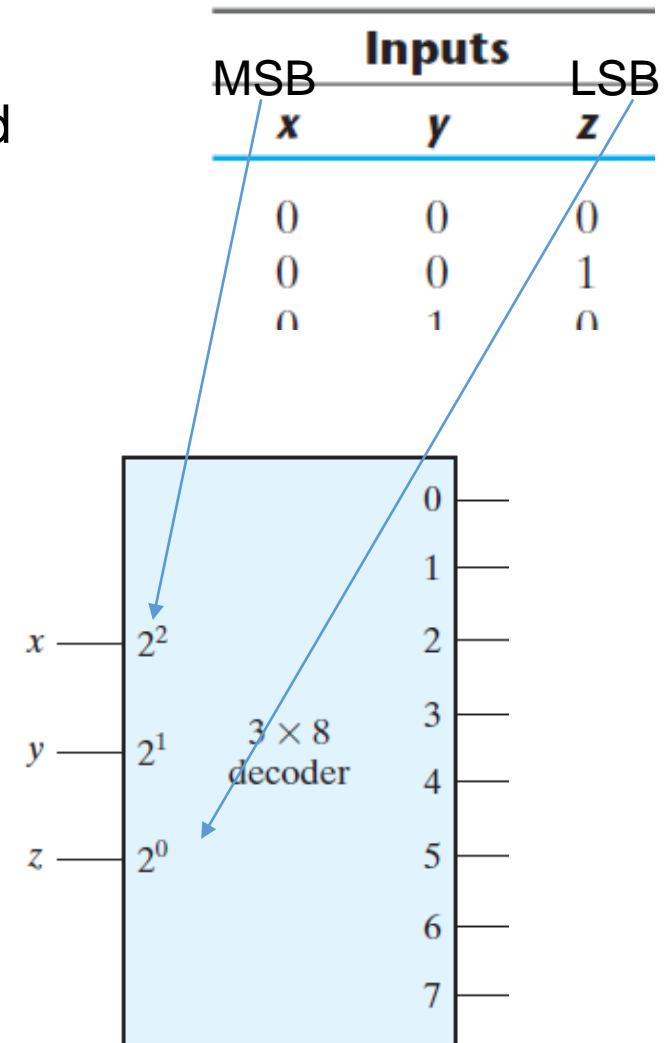
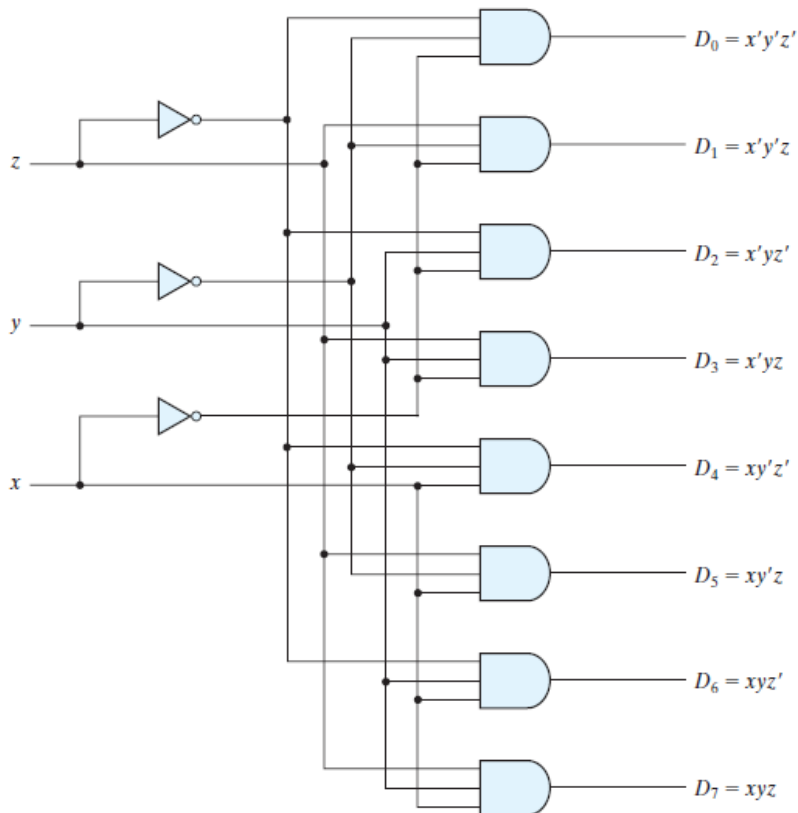
Step 1,2

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_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



Logic Diagram vs Block Diagram

- We can use block diagram
 - Clearly denoting the input position and output sequence



Main Usages of Decoders

- Minterm generator (最小项生成器):
 - Generate the 2^n (or fewer) minterms of n input variables. For example: a 3-8 line decoder
- Data demultiplexing (数据分配器):
 - A decoder with enable input can function as a **demultiplexer** – a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- Display decoding: (显示解码)
 - Decoders are used in display systems to select a specific output line based on the input code and drive the corresponding segment of the display.
- Address decoding: (地址解码):
 - Identify a memory cell, disk sector, or other memory or storage device, to ensure one device can communicate with the processor at one time.

Decoder for logic implementation

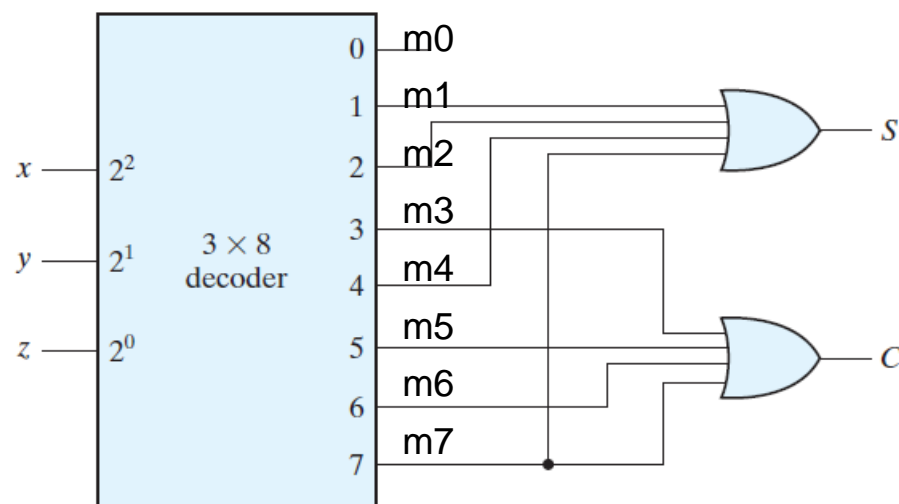
Example1

- Decoder can be used to implement the logic function by connecting the appropriate minterms to an OR gate.
 - Any combinational circuit with n inputs and m outputs can be implemented with an n -to- 2^n decoder in conjunction with m external OR gates

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

$$S(x, y, z) = \sum(1, 2, 4, 7)$$

$$C(x, y, z) = \sum(3, 5, 6, 7)$$



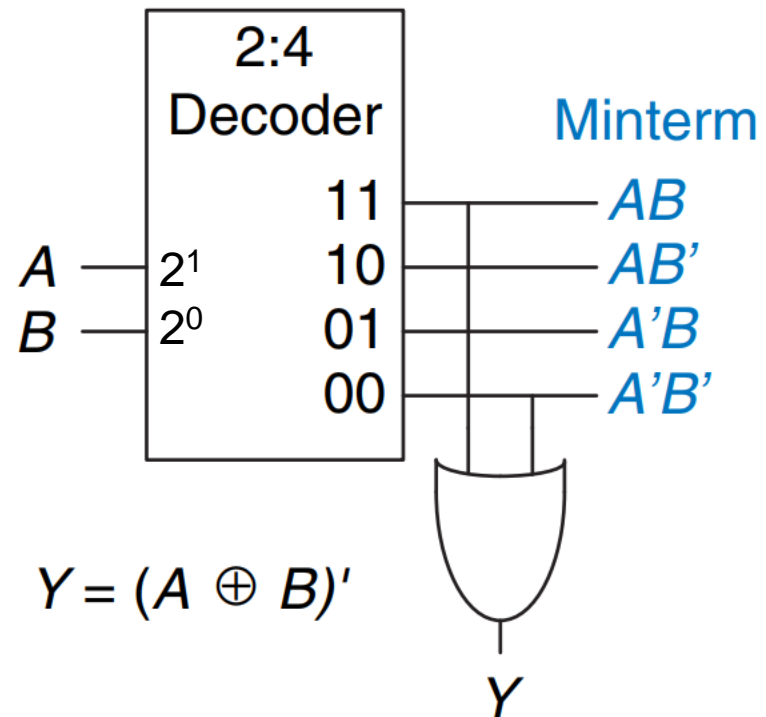
Decoder for logic implementation

Example2

- Exercise:
 - Implement $Y = A \text{ XNOR } B$ using a 2-to-4 line decoder and external OR gate, you need to clearly write down the input and output pins

$$\begin{aligned} Y &= A \text{ XNOR } B \\ &= (A \oplus B)' \\ &= A'B' + AB \\ &= \sum(0, 3) \end{aligned}$$

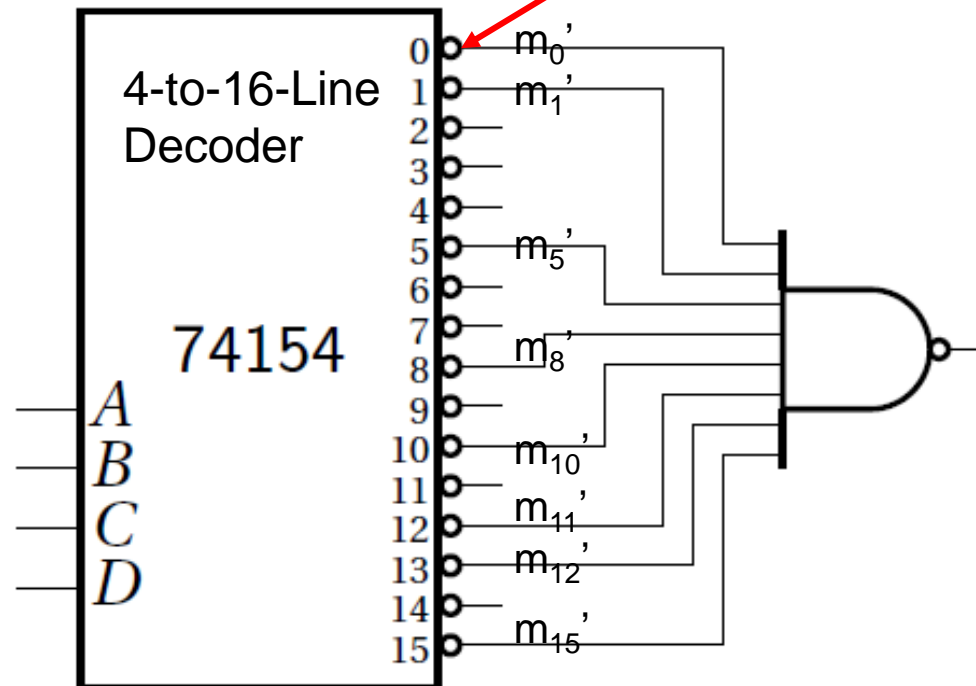
Connect output 0 and 3
to an OR gate



Decoder for logic implementation

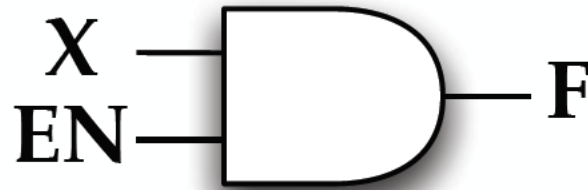
Example3

- MSI 74154: 4-to-16 line decoder
 - If $A = B = C = D = 0$ the output 0 of the decoder is **0** while all other outputs are 1. (**active low**) → generate inverse of minterms
 - Example: $F(A,B,C,D) = \sum(0, 1, 5, 8, 10, 12, 13, 15)$.
$$= [(m_0 + m_1 + m_5 + m_8 + m_{10} + m_{12} + m_{13} + m_{15})]'$$
$$= (m_0' \cdot m_1' \cdot m_5' \cdot m_8' \cdot m_{10}' \cdot m_{12}' \cdot m_{13}' \cdot m_{15}')$$



Enabling

- Enabling permits an input signal to pass through to an output.



- $F = EN \cdot X$

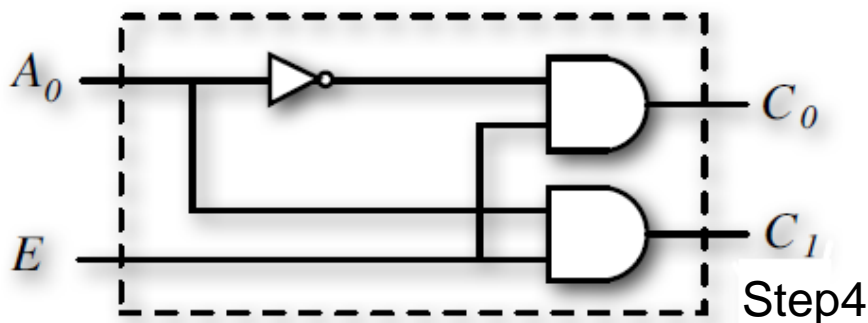
EN	X	F
0	0	0
0	1	0
1	0	0
1	1	1

Decoder with Enable Input

- Decoder with enable control (E)

Step1,2

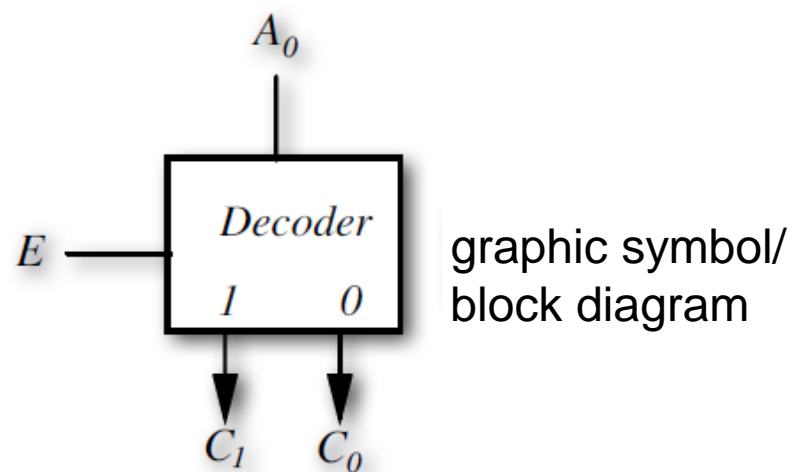
	E	A ₀	C ₁	C ₀
Active High Enable	1	0	0	1
	1	1	1	0
Low → disabled	0	X	0	0



Step3

$$C_0 = EA_0'$$

$$C_1 = EA_0$$



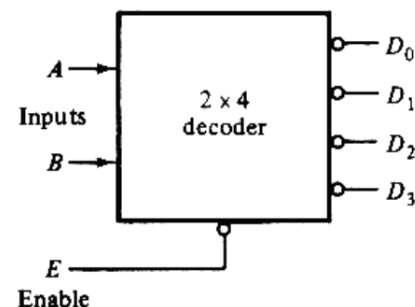
1-2 Line Decoder with Enable

Decoder with Active-Low Enable

- Constructed with NAND gates
 - decoder minterms in their complemented form (more economical)

	E	A	B	D_0	D_1	D_2	D_3
High \rightarrow disabled	1	X	X	1	1	1	1
Active Low Enable	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

Output in complement form

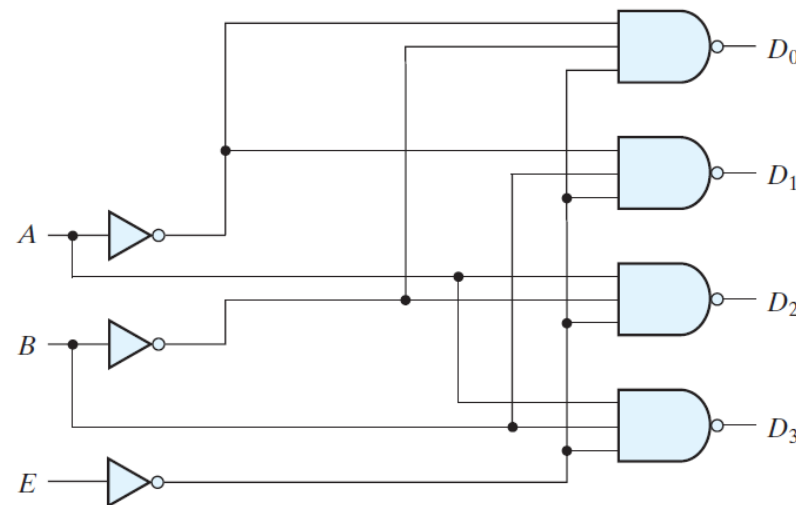


$$D_0 = (E'A'B)'$$

$$D_1 = (E'A'B)'$$

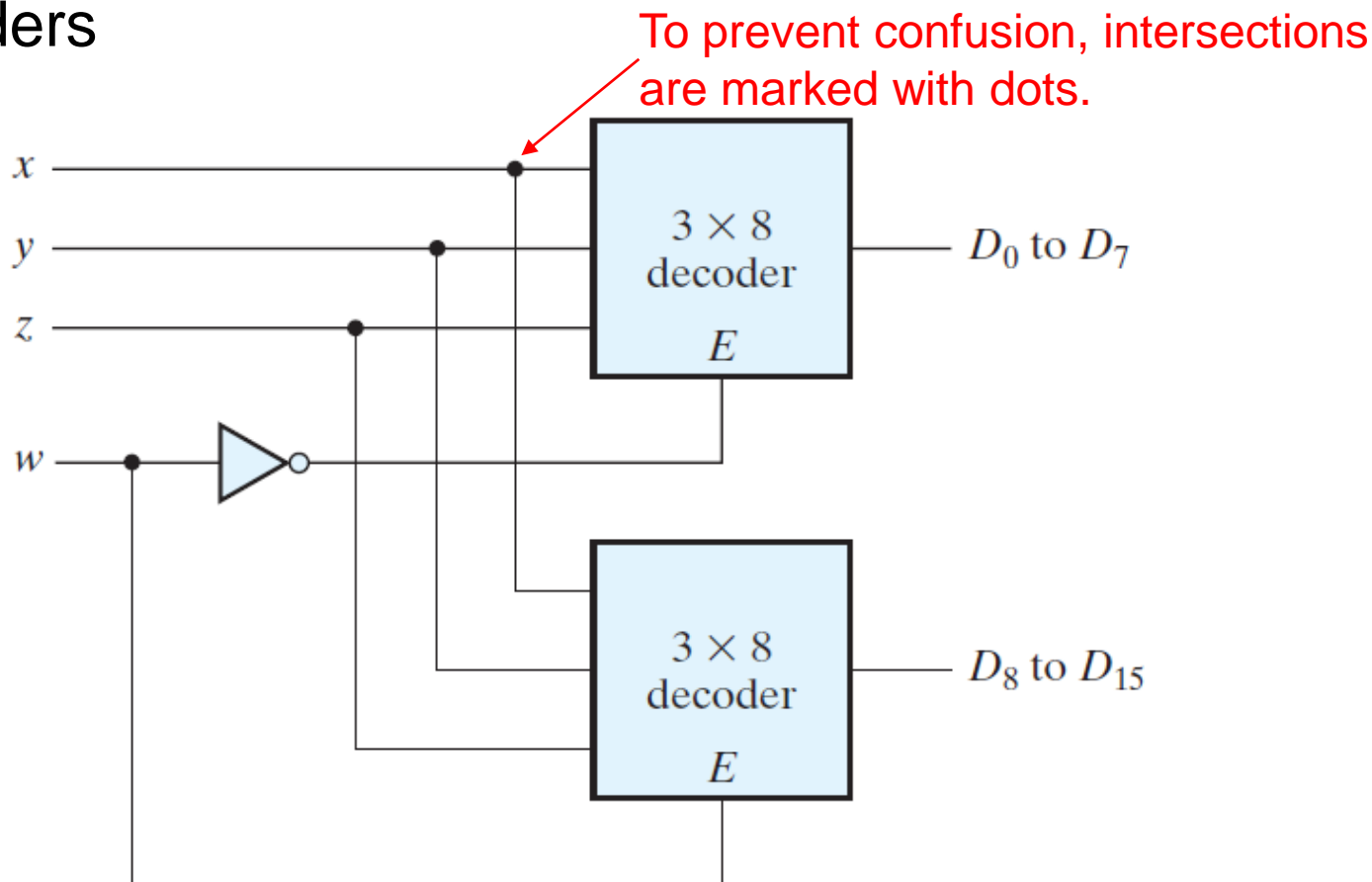
$$D_2 = (E'AB)'$$

$$D_3 = (E'AB)'$$



Decoder Expansion

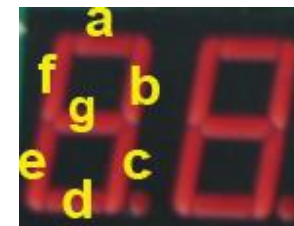
- Larger decoders can be implemented with smaller decoders



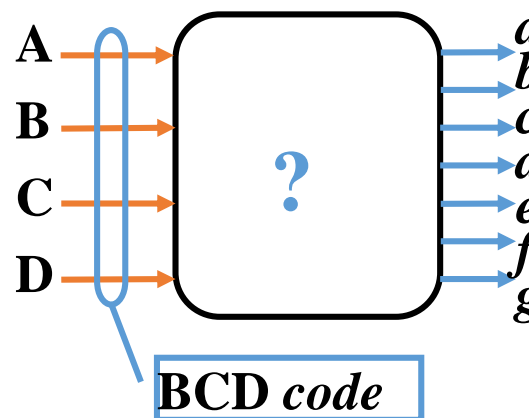
A 4-to-16-line decoder from two 3-to-8-line decoders

Other Decoders

- BCD-to-7-Segment Display Decoder
 - input (ABCD), output (abcdefg)(MSB to LSB)
 - ABCD:0000~1001(0~9)



BCD Input				7-Segment Display						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	0	0	1	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0



$$a = A'C + A'BD + B'C'D' + A'B'C'$$

$$b = A'B' + A'C'D' + A'CD + AB'C'$$

$$c = A'B + A'D + B'C'D' + AB'C'$$

$$d = A'CD' + A'B'C + B'C'D' + AB'C' + A'BC'D$$

$$e = A'CD' + B'C'D'$$

$$f = A'BC' + A'C'D' + A'BD' + AB'C'$$

$$g = A'CD' + A'B'C + A'BC' + AB'C'$$

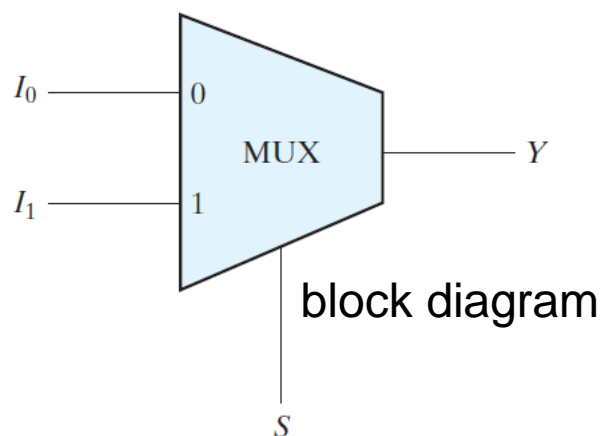
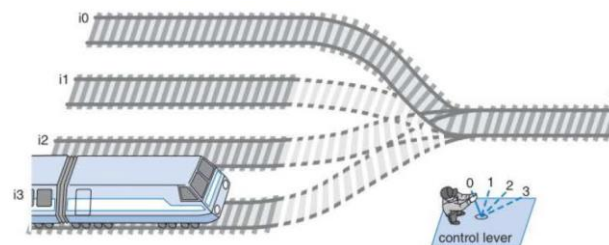


Outline

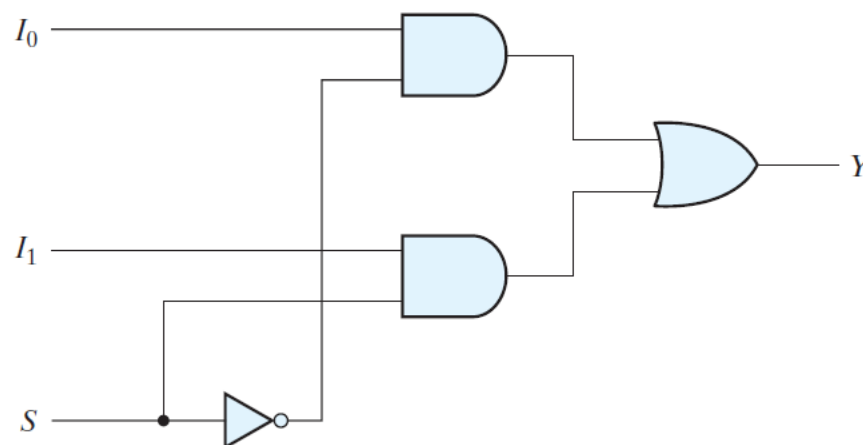
- Decoder
- **Multiplexer**
- Encoder
- Gate Behavior

Multiplexers (MUX)

- A Multiplexer selects (usually by n select lines) binary information from one of many (usually 2^n) input lines and directs it to a single output line.



2:1 multiplexer



Function table

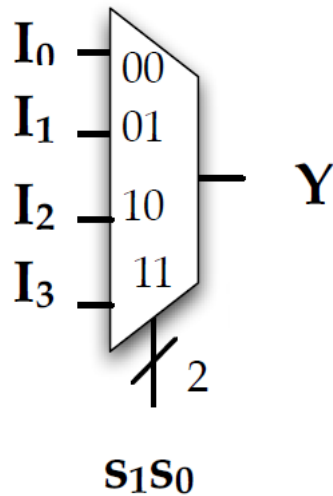
S	Y
0	I_0
1	I_1

Logic equation

$$Y = S'I_0 + SI_1$$

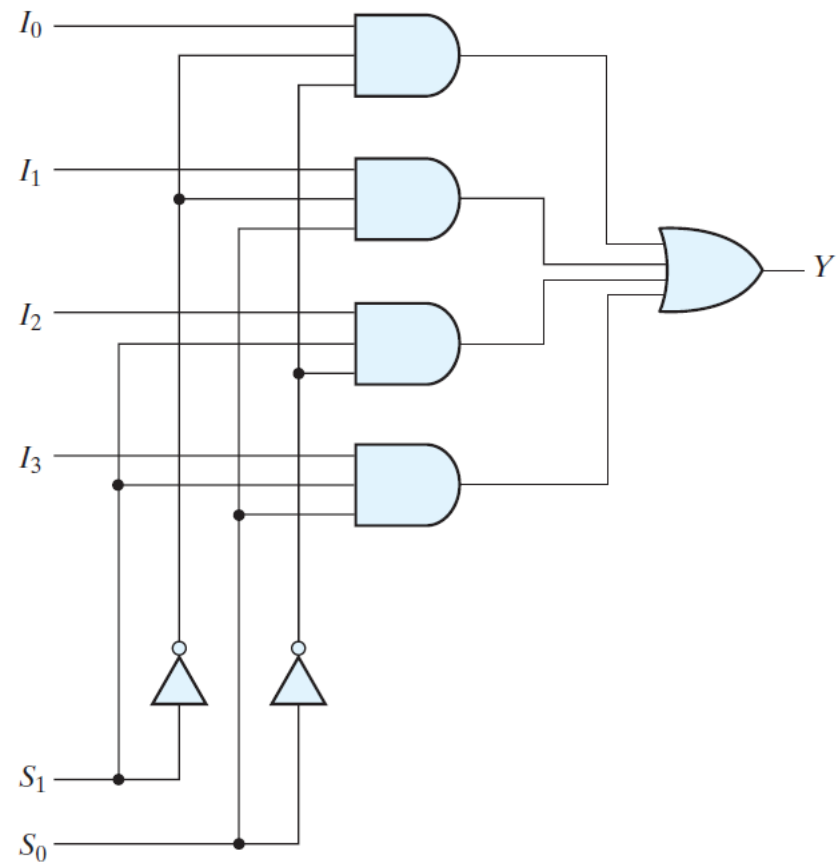
function table lists the input that is passed to the output for each combination of the binary selection values

4:1 MUX



Function table

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

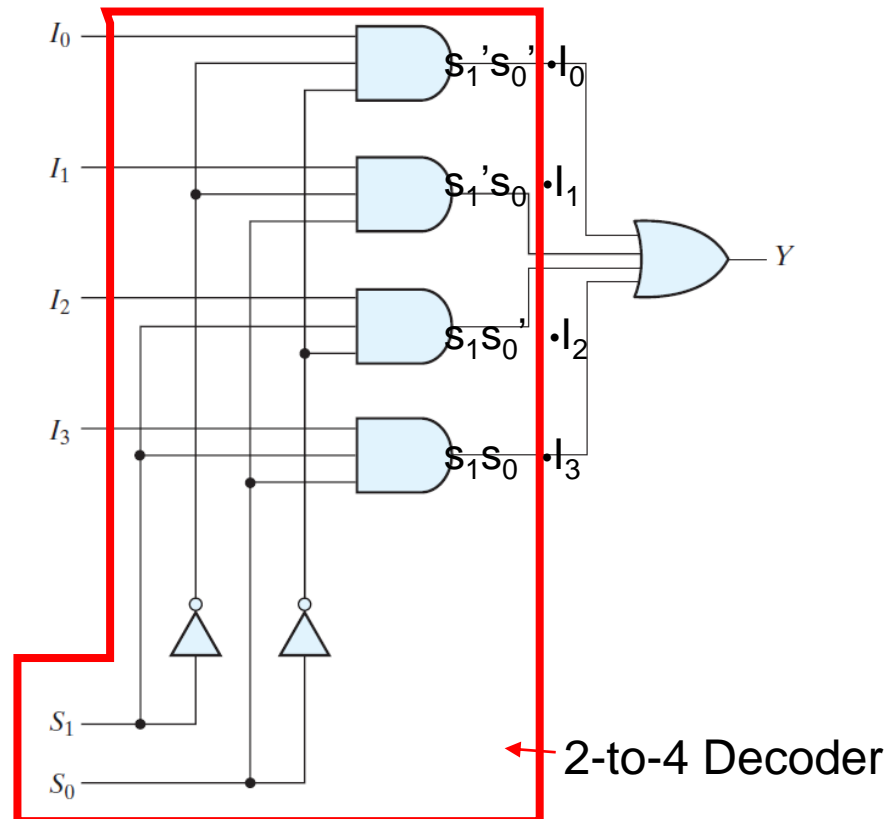


Logic equation

$$Y = s_1's_0'I_0 + s_1's_0I_1 + s_1s_0'I_2 + s_1s_0I_3$$

MUX Composition

- MUX = decoder + OR gate
 - The device has two control or selection lines S_1 and S_0 ,
 - Logic equation: $Y = s_1's_0'I_0 + s_1's_0I_1 + s_1s_0'I_2 + s_1s_0I_3$



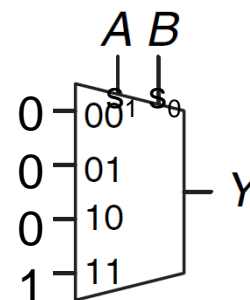
MUX for logic implementation

Example1

- Implement AND function using MUX
 - can be used as a look-up table
 - 4:1 multiplexer can be used (truth table)

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

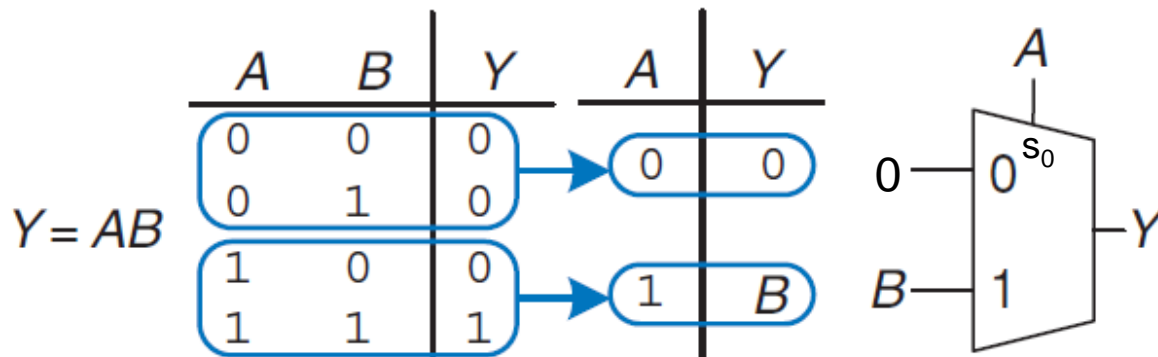
$Y = AB$



Exercise: Implement XNOR function using

- 1) a 4:1 MUX
- 2) a 2:1 MUX

- What if only 2:1 MUX is allowed to use?
 - By using variable as data inputs

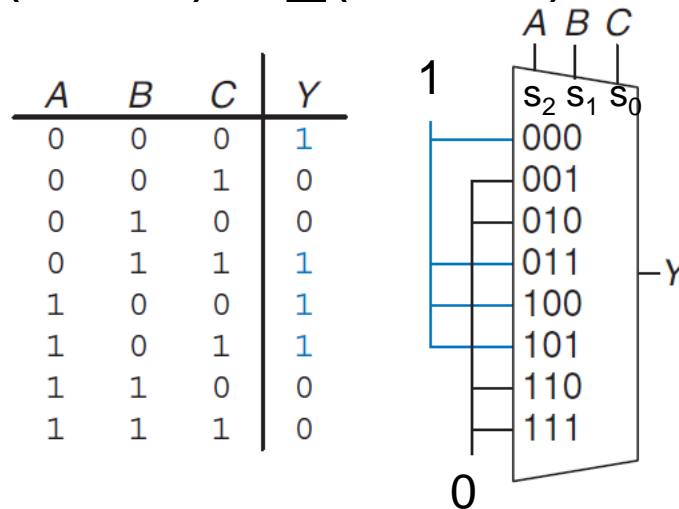


MUX for logic implementation

Example2

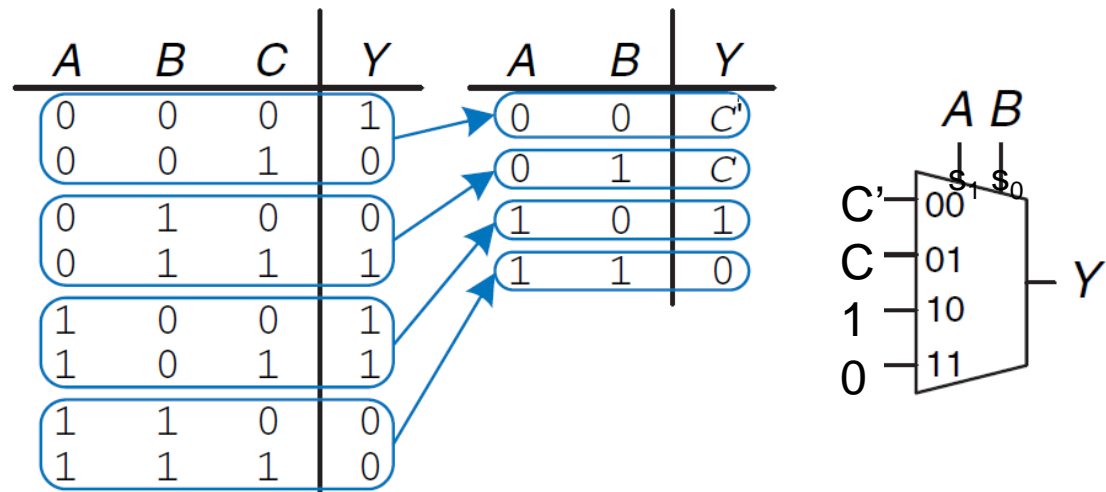
- Implement the function $Y(A,B,C) = \sum(0,3,4,5)$ with MUX

1. using 8:1 MUX



2. using 4:1 MUX

- We can use 4:1 MUX by reducing the truth table to four rows by letting A,B as select bit s_1 and s_0



MUX for logic implementation

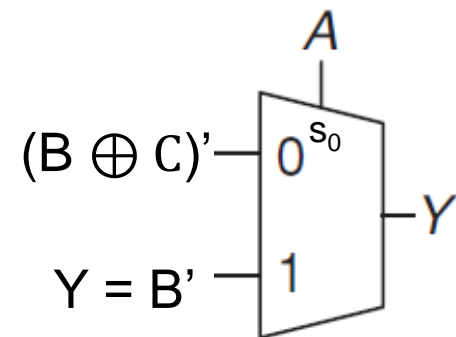
Example2

- Implement the function $Y(A,B,C) = \sum(0,3,4,5)$ with MUX
- 3. Using 2:1 MUX?

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$Y = (B \oplus C)'$

$Y = B'$

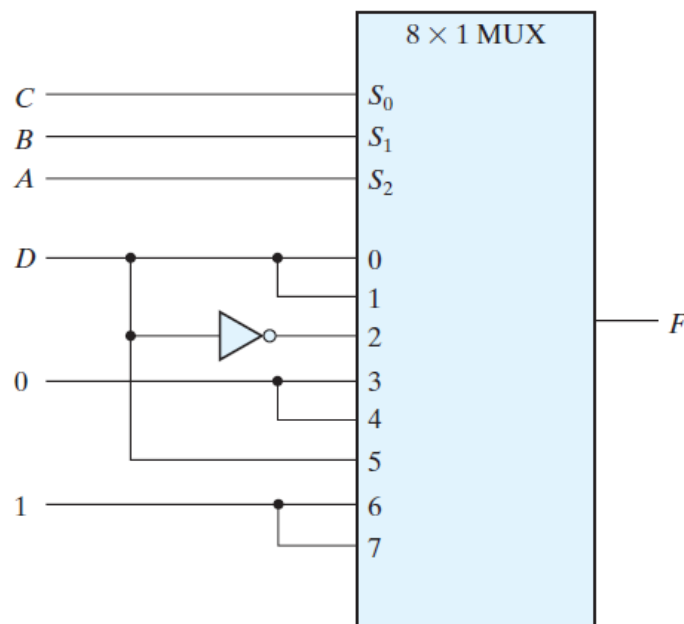


MUX for logic implementation

Example3

- Implement $F(A, B, C, D) = (1, 3, 4, 11, 12, 13, 14, 15)$ with three selection inputs Multiplexer.
 - A must be connected to selection input S_2 so that A, B, and C correspond to selection inputs S_2, S_1 , and S_0 , respectively

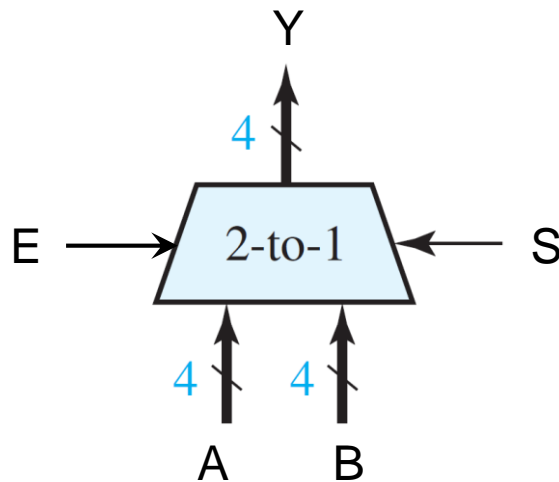
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



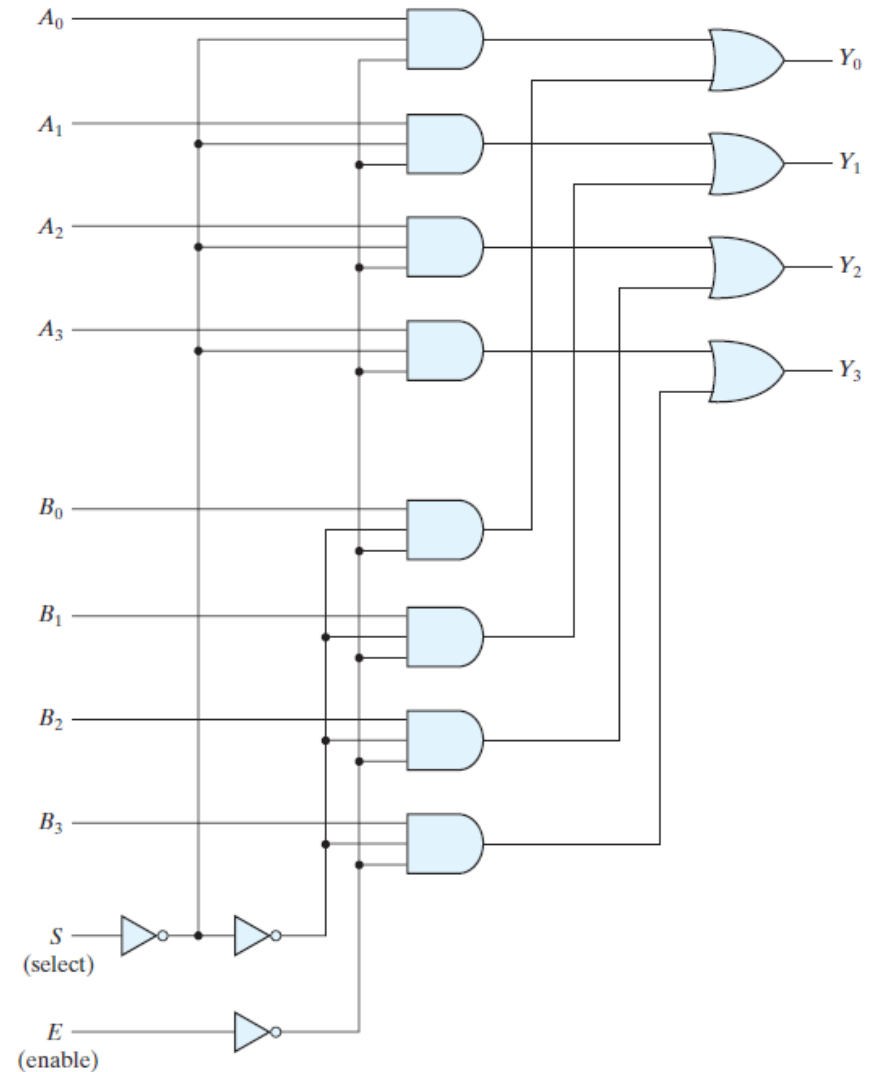
Quadruple 2:1 MUX (4-bit 2:1 MUX)

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table



four 2:1 MUX with enable



MUX Expansion

- Wider multiplexers, such as 8:1 and 16:1 multiplexers, can be built with smaller multiplexers

Function table

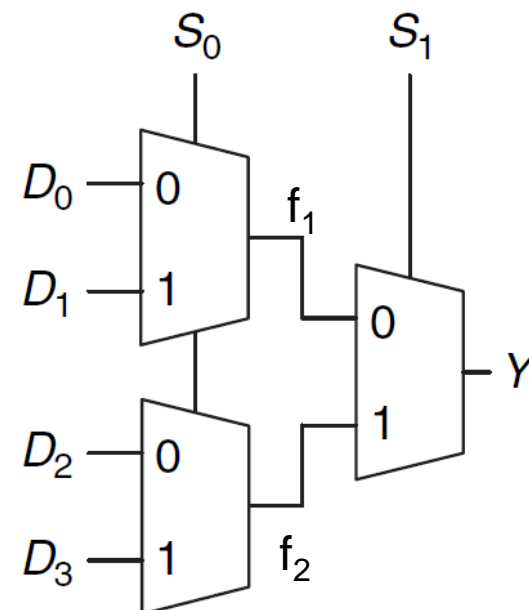
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$f_1 = S_0'D_0 + S_0D_1$
 $f_2 = S_0'D_2 + S_0D_3$

Logic equation

$$\begin{aligned}
 Y &= s_1'f_1 + s_1f_2 \\
 &= s_1'(s_0'D_0 + s_0D_1) + s_1(s_0'D_2 + s_0D_3) \\
 &= s_1's_0'D_0 + s_1's_0D_1 + s_1s_0'D_2 + s_1s_0D_3
 \end{aligned}$$

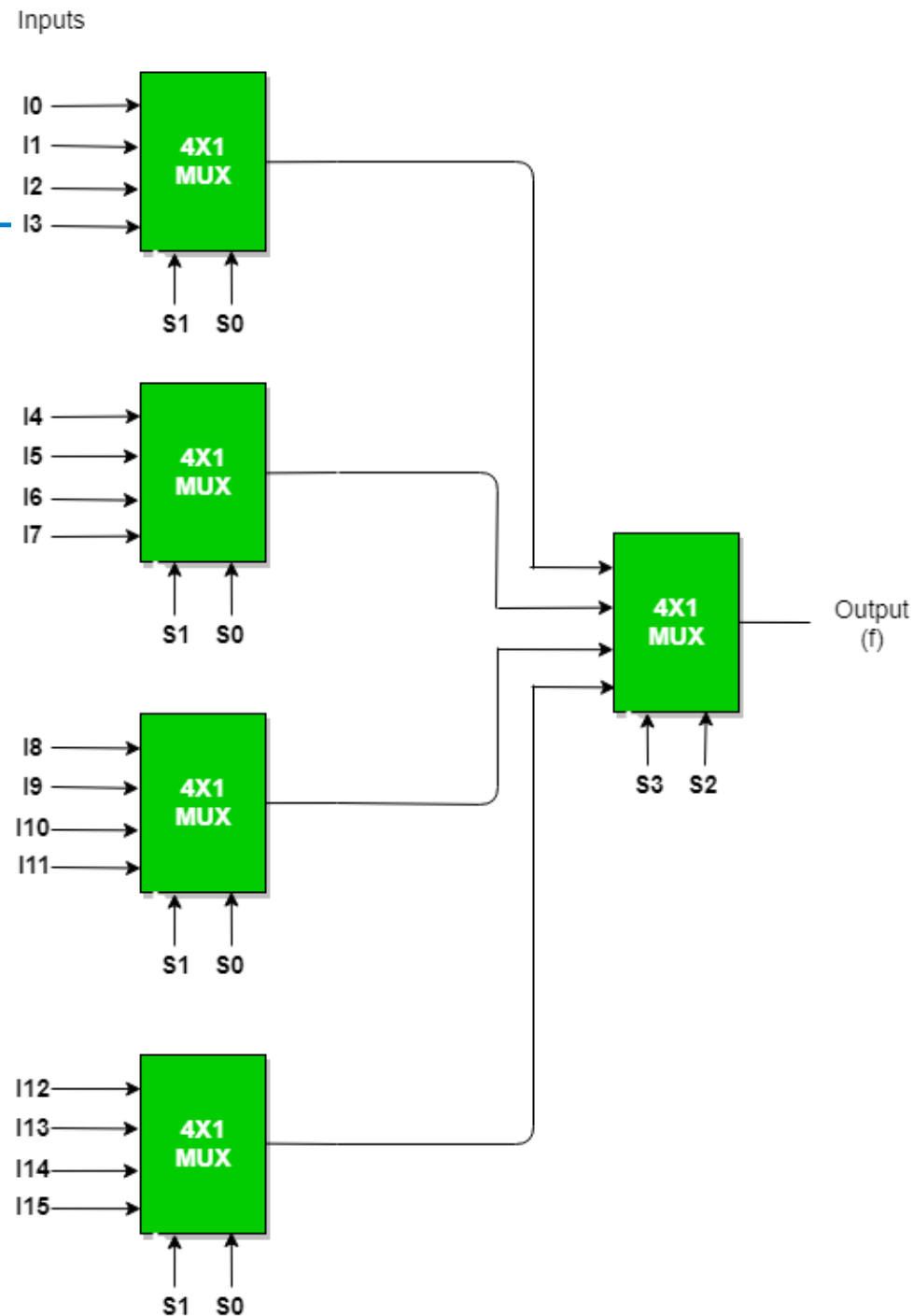
4:1 MUX with three 2:1 MUX



MUX Expansion

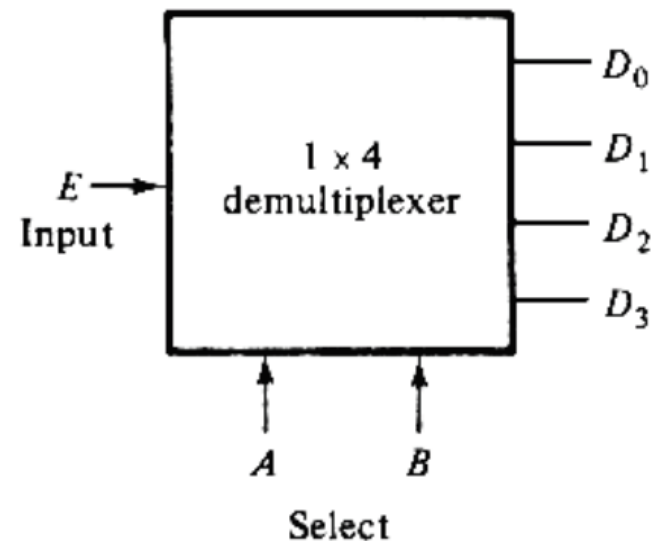
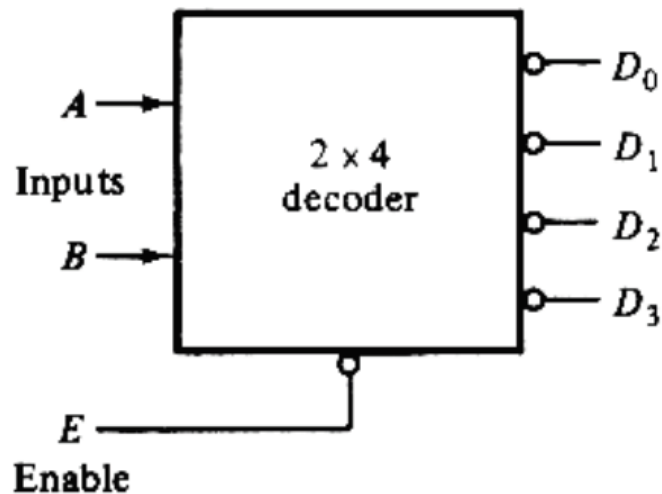
- How to build a 16-to-1 multiplexer using five 4-to-1 multiplexers?
 - $16 = 2^4$
 - 4 bits for selection

Exercise: How to build a 8-to-1 multiplexer using two 4-to-1 MUX and a 2-to-1 MUX? You must carefully connect the selection and input pins



Demultiplexer

- A decoder with enable input can function as demultiplexer
 - a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
 - Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a demultiplexer.





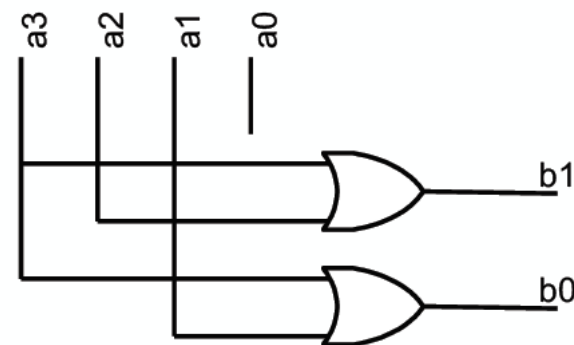
Outline

- Decoder
- Multiplexer
- **Encoder**
- Gate Behavior

Encoder

- An encoder is an inverse of a decoder
- Encoder is a logic module that converts a **one-hot** input signal to a binary-encoded output signal
- Other input patterns are **forbidden** in the truth table
- Example: a 4-→2 encoder

a_3	a_2	a_1	a_0	b_1	b_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



$$b_0 = a_3 + a_1$$

$$b_1 = a_3 + a_2$$

Encoder

- A combinational logic that performs the inverse operation of a decoder
 - Only one input has value 1 at any given time
 - Can be implemented with OR gates
- However, when both D3 and D6 goes 1, the output will be 111 (ambiguity)! **illegal inputs !Use priority encoder!**

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
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

$$x = D_4 + D_5 + D_6 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$z = D_1 + D_3 + D_5 + D_7$$

Priority Encoder

- Ensure only one of the input is encoded
- Assuming D_3 has the highest priority, while D_0 has the lowest priority.
- X is the don't care conditions, V is the valid output indicator.

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	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

$$V = D_0 + D_1 + D_2 + D_3$$

Priority Encoder

D_0D_1		D_2D_3			
		00	01	11	10
D_0	00	m_0 X	m_1 1	m_3 1	m_2 1
	01	m_4	m_5 1	m_7 1	m_6 1
	11	m_{12}	m_{13} 1	m_{15} 1	m_{14} 1
	10	m_8	m_9 1	m_{11} 1	m_{10} X

$x = D_2 + D_3$

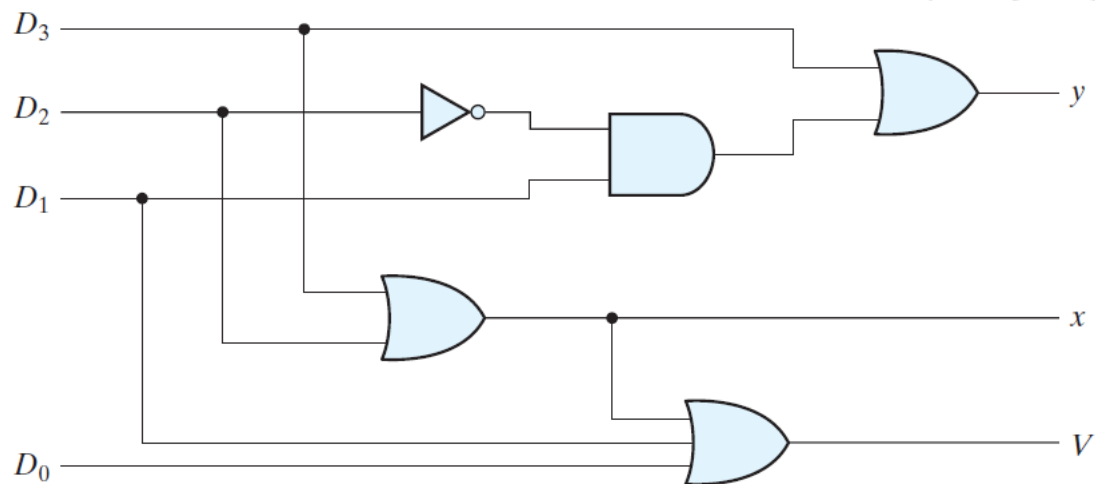
$$x = D_2 + D_3$$

$$y = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$

D_0D_1		D_2D_3			
		00	01	11	10
D_0	00	m_0 X	m_1 1	m_3 1	m_2
	01	m_4 1	m_5 1	m_7 1	m_6
	11	m_{12} 1	m_{13} 1	m_{15} 1	m_{14}
	10	m_8	m_9 1	m_{11} 1	m_{10}

$y = D_3 + D_1 D_2'$



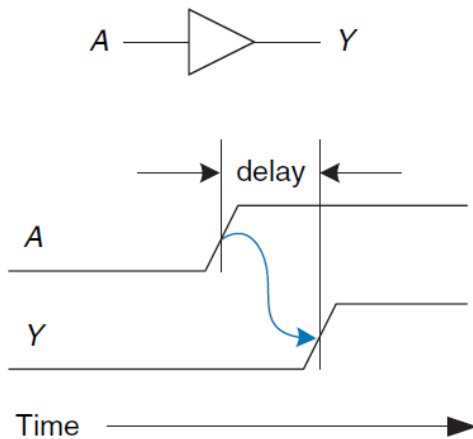
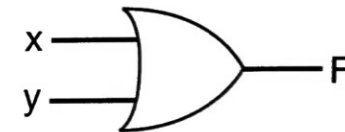


Outline

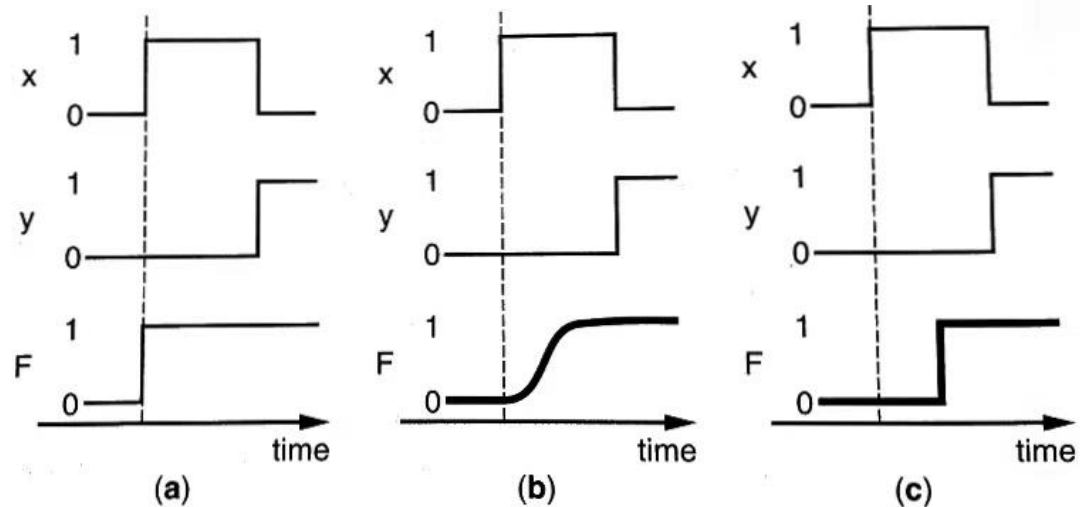
- Decoder
- Multiplexer
- Encoder
- **Gate Behavior**

Gate Delays

- When the input to a logic gate is changed, the output will not change immediately. The output of the gate experiences a **propagation delay** in response to changes in the input.



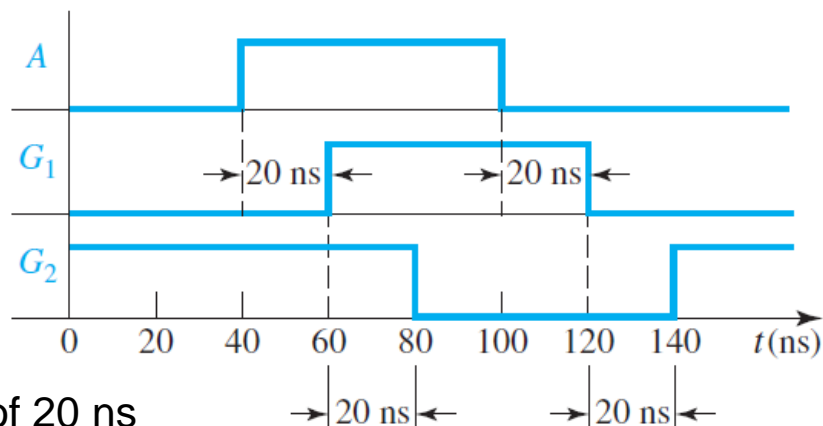
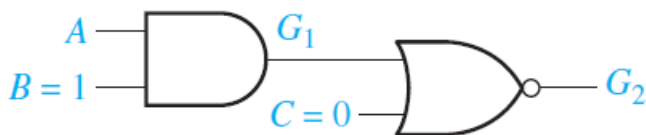
delay between an input change and the subsequent output change for a buffer



- (a) ideal behavior without gate delay
(b) a more realistic illustration
(c) switching incorporating the delay

Effect of gate delays

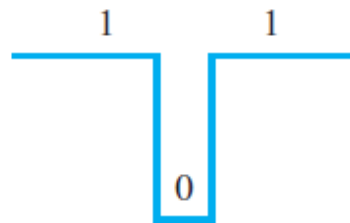
- The analysis of a combinational circuit ignoring delays can predict only its **steady-state behavior**.
- Predicts a circuit's output as a function of its inputs assuming that the inputs have been stable for a long time, relative to the delays in the circuit's electronics.
- Because of circuit delays, the **transient behavior** of a combinational logic circuit may differ from what is predicted by steady-state analysis.



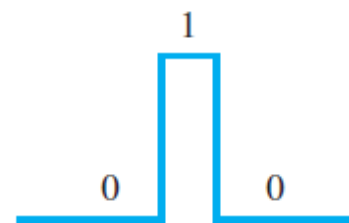
assume each gate has a propagation delay of 20 ns

Hazard (Glitches)

- **Timing hazard**: **Unwanted** switching transients (glitches) appearing in the output while the input to a combinational circuit changes.
 - **static-1 hazard** is a short **0** glitch when for a changed input, we expect (by logic theorems) the output to remain constant **1**.
 - **static-0 hazard** is a short **1** glitch when we expect the output to remain constant **0**.



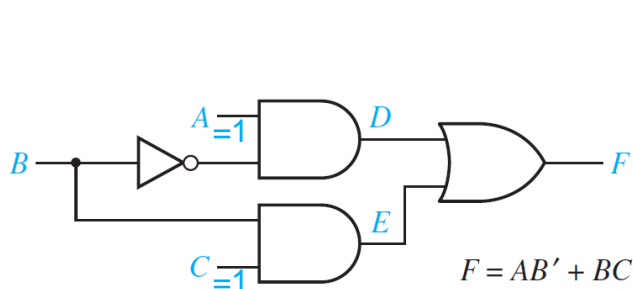
(a) Static 1-hazard



(b) Static 0-hazard

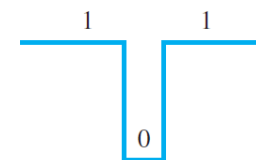
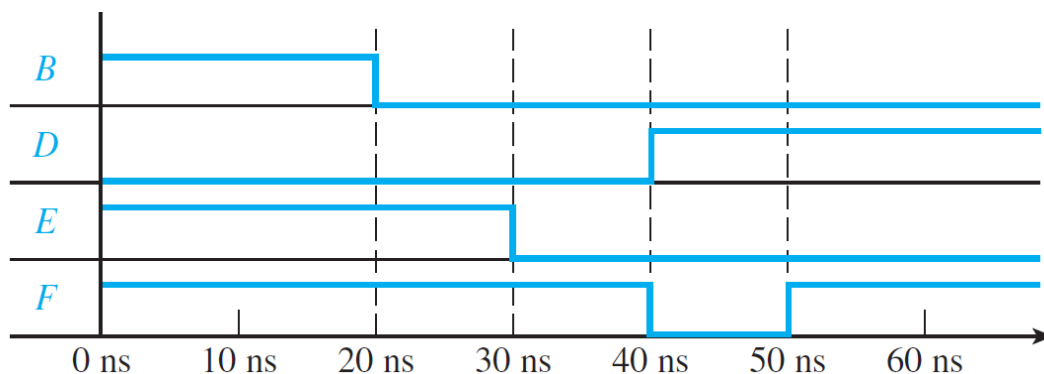
Circuit with a static 1 Hazard

- Assume each gate has a propagation delay of 10 ns
 - if $A = C = 1$ and B changes from 1 to 0, F should be a stable 1. Change propagates to output F along two paths with different delays, resulting in a glitch in F .



BC		B			
		00	01	11	10
A	0	0	0	1	0
	1	1	1	1	0
		C			

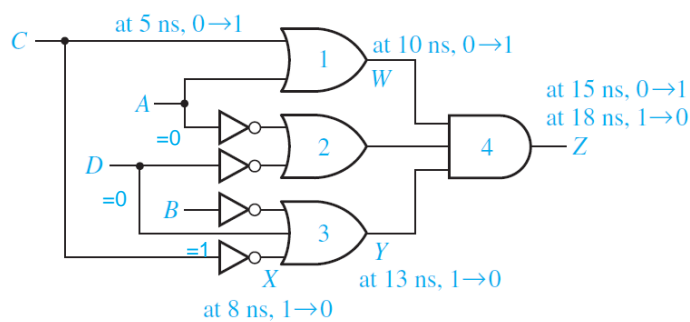
If any two adjacent 1's are not covered by the same circle, a 1-hazard exists for the transition between the two 1's.



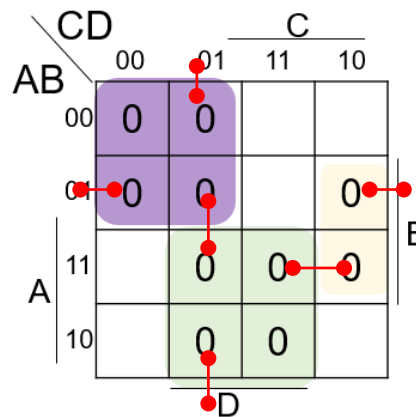
Static 1-hazard

Circuit with a static 0 Hazard

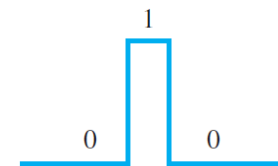
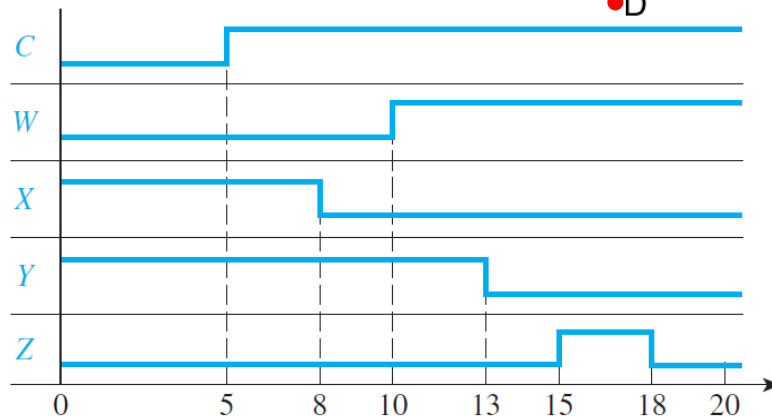
- Assume gate delay: Inverter = 3ns, AND/OR = 5ns
 - if $A = 0$, $B = 1$, $D = 0$, and C changes from 0 to 1, F should be a stable 0. Change propagates to output F along two paths with different delays., resulting in a glitch in F .



$$F = (A + C)(A' + D')(B' + C' + D)$$



If any two adjacent 0's are not covered by the same circle, a 0-hazard exists for the transition between the two 0's.



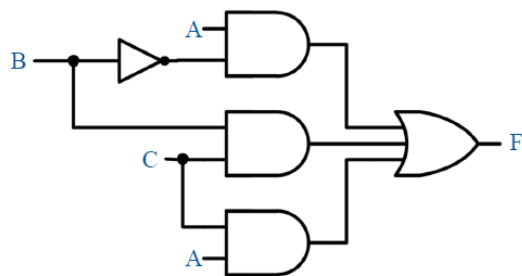
Static 0-hazard

Removing Hazard

- We can eliminate the 1-hazard or 0-hazards by adding additional circle that covers the adjacent 1's or 0's not already covered within a common circle.

		BC		B	
		00	01	11	10
A	0	0	0	1	0
	1	1	1	1	0

Term **AC** is added to eliminate 1 hazard



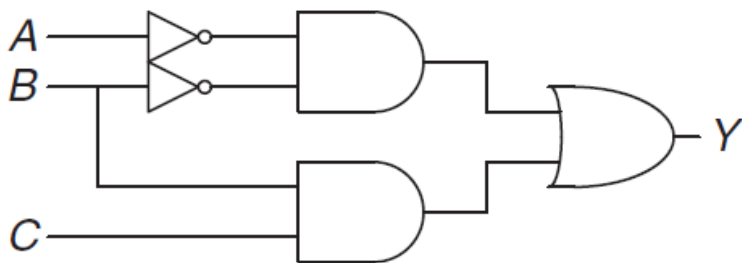
$$F = AB' + BC + AC$$

		CD		C	
		00	01	11	10
AB	00	0	0		
	01	0	0		0
A	11		0	0	0
	10		0	0	

Term **(C+D')**, **(A+B'+D)**, **(A'+B'+C')** are added to eliminate 0 hazard

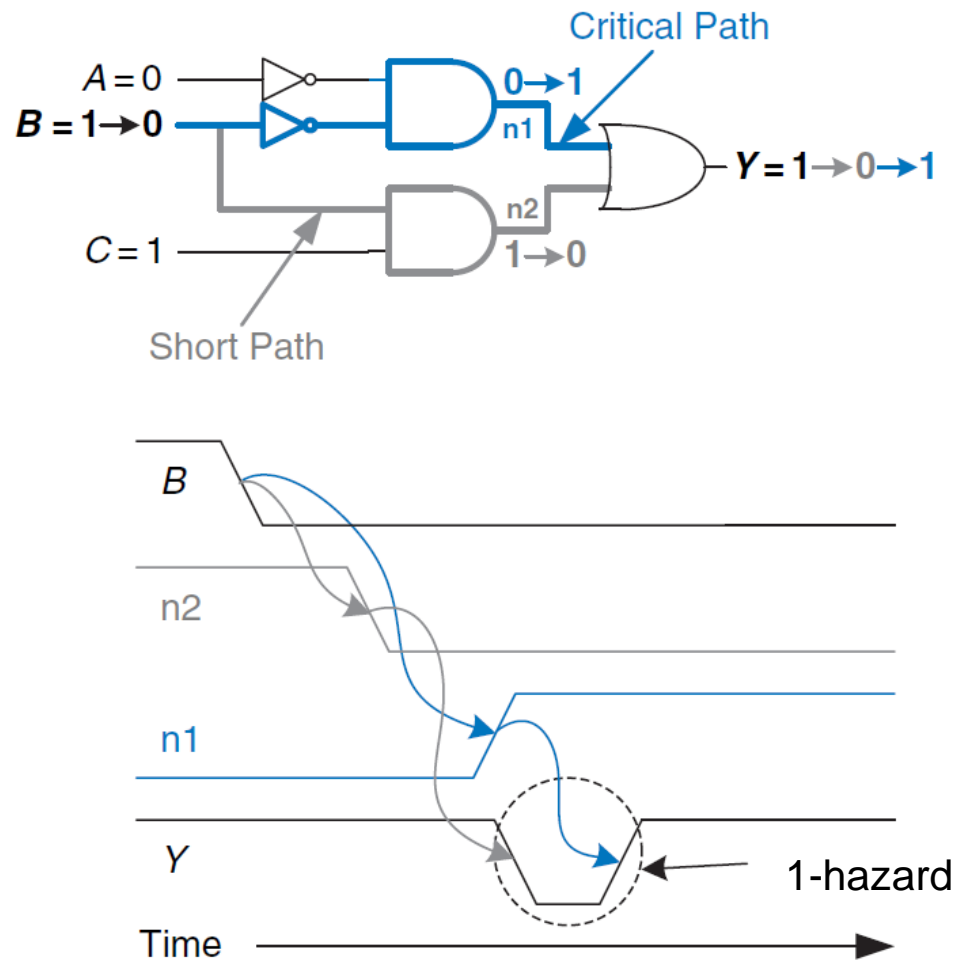
Example

- What happens when $A = 0$, $C = 1$, B falls?



		AB			
		00	01	11	10
C	0	1	0	0	0
	1	1	1	1	0

$$Y = A'B' + BC$$



Example

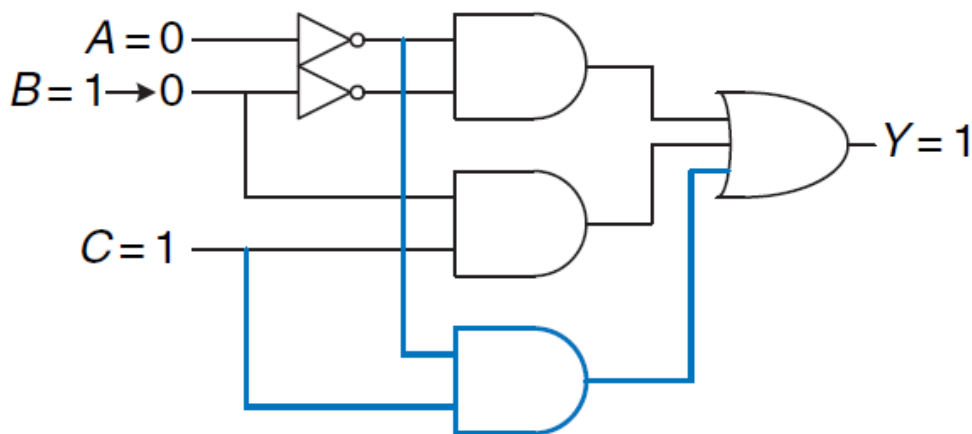
- To eliminate the 1-hazard, add an additional circles that cover the adjacent 1's not already covered by a common circle.

Y C	AB			
	00	01	11	10
0	1	0	0	0
1	1	1	1	0

Minimal cost
but with hazard:
 $Y = A'B' + BC$

Y C	AB			
	00	01	11	10
0	1	0	0	0
1	1	1	1	0

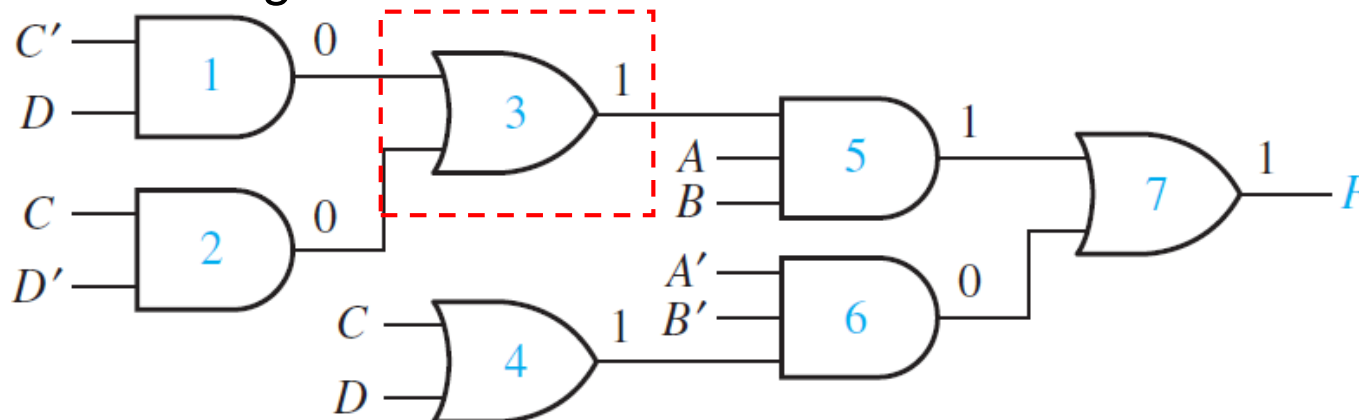
$Y = A'B' + BC + A'C$



Eliminate hazard by
adding a product term

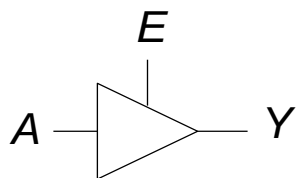
Testing of Logic Circuits

- In the logic circuit, a wrong output may be due to
 - Incorrect design
 - Gates connected wrong
 - Wrong input signals to the circuit
 - Defective gates
 - Defective connecting wires
- Logic circuit with incorrect output
 - $A = B = C = D = 1$, F should be 0 but an incorrect 1 is generated, what's wrong?

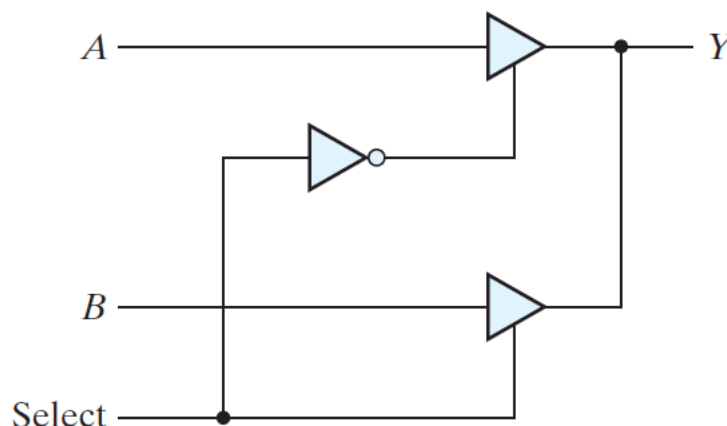
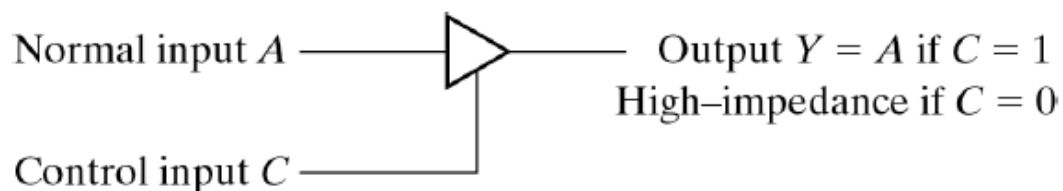


Tri-state

- Tri-state driver (buffer) has three possible output states: 0, 1, Z (high impedance).



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1



2:1 MUX using tri-state

Sel	Y
0	A
1	B