RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

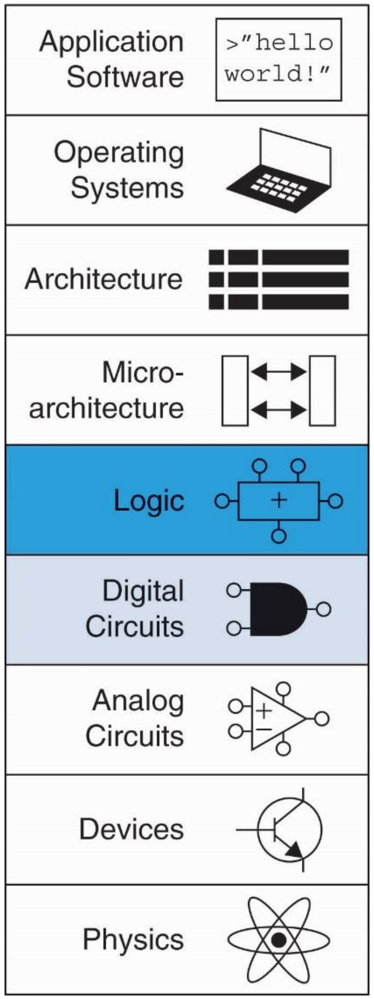
ECE-332:437
DIGITAL SYSTEMS DESIGN (DSD)

Fall 2016 – Lecture 4

Nagi Naganathan
September 15, 2016

Chapter 2 :: Topics

- Introduction
- Boolean Equations
- Boolean Algebra
- From Logic to Gates
- Multilevel Combinational Logic
- X's and Z's, Oh My
- Karnaugh Maps
- Combinational Building Blocks
- Timing



Combinatorial Building Blocks

Combinational Building Blocks

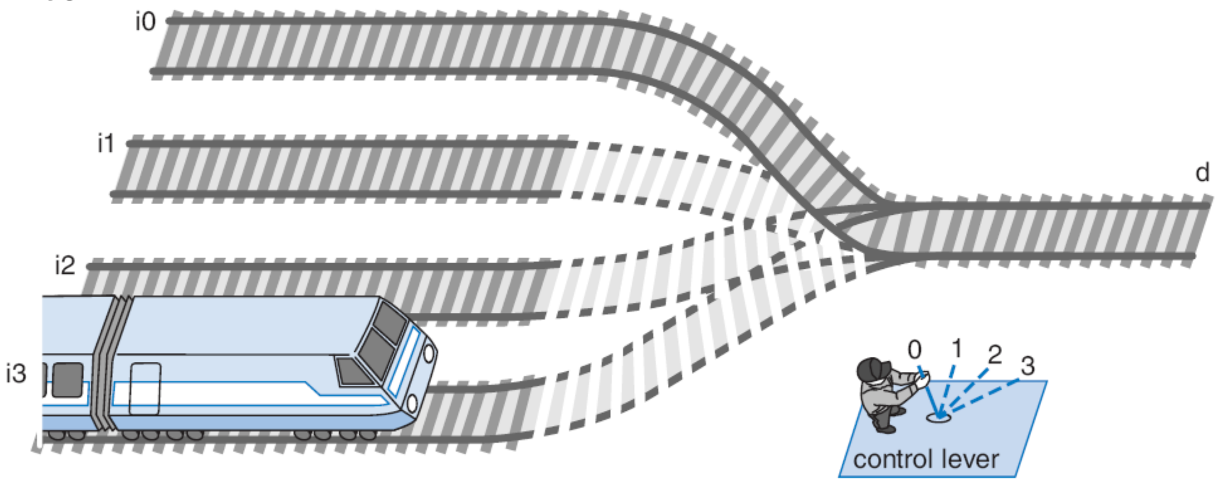
- Multiplexers
- De-Multiplexers
- Decoders
- Encoders

Selecting

- **Selecting of data or information is a critical function in digital systems and computers**
- **Circuits that perform selecting have:**
  - **A set of information inputs from which the selection is made**
  - **A single output**
  - **A set of control lines for making the selection**
- **Logic circuits that perform selecting are called *multiplexers***
- **Selecting can also be done by three-state logic or transmission gates**

Multiplexor (Mux)

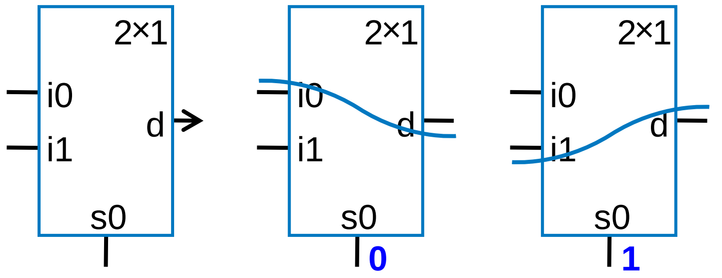
- Mux: Another popular combinational building block
  - Routes one of its N data inputs to its one output, based on binary value of select inputs
    - 4 input mux → needs 2 select inputs to indicate which input to route through
    - 8 input mux → 3 select inputs
    - N inputs → $\log_2(N)$ selects
  - Like a rail yard switch



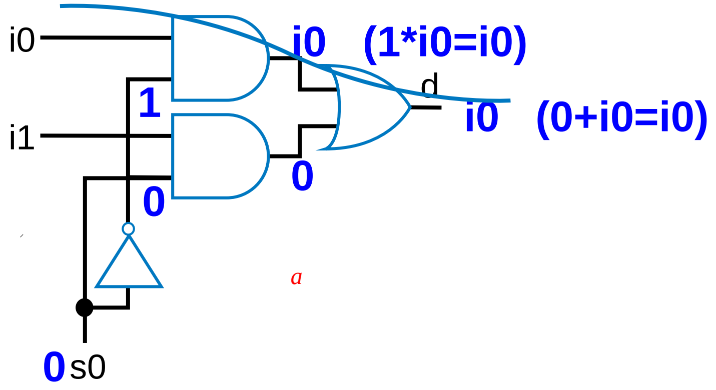
Multiplexers

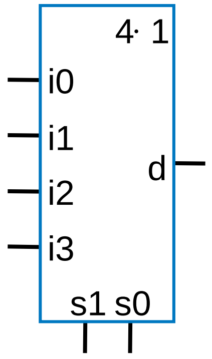
- **A multiplexer selects information from an input line and directs the information to an output line**
- **A typical multiplexer has *n* control inputs (S_{n-1}, … S_0) called *selection inputs*, 2^n information inputs ($I_{2^n - 1}$, … I_0), and one output Y**
- **A multiplexer can be designed to have *m* information inputs with $m < 2^n$ as well as *n* selection inputs**

Mux Internal Design

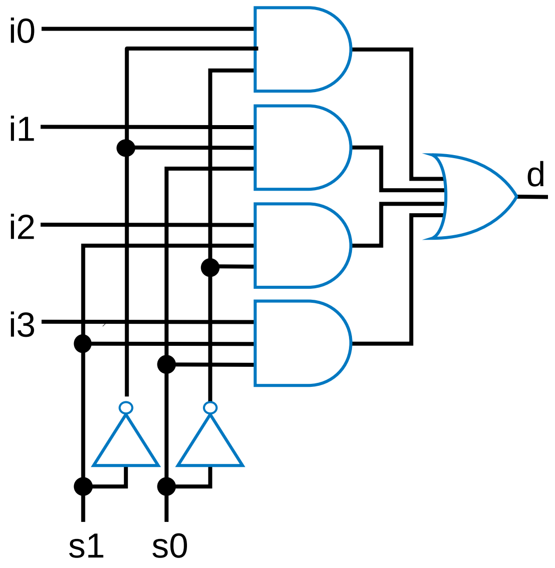


2x1 mux



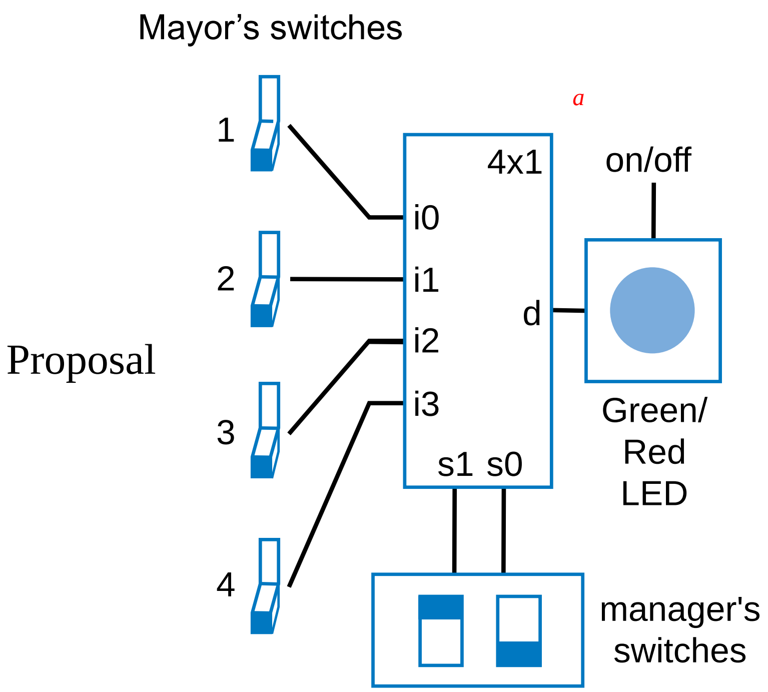


4x1 mux

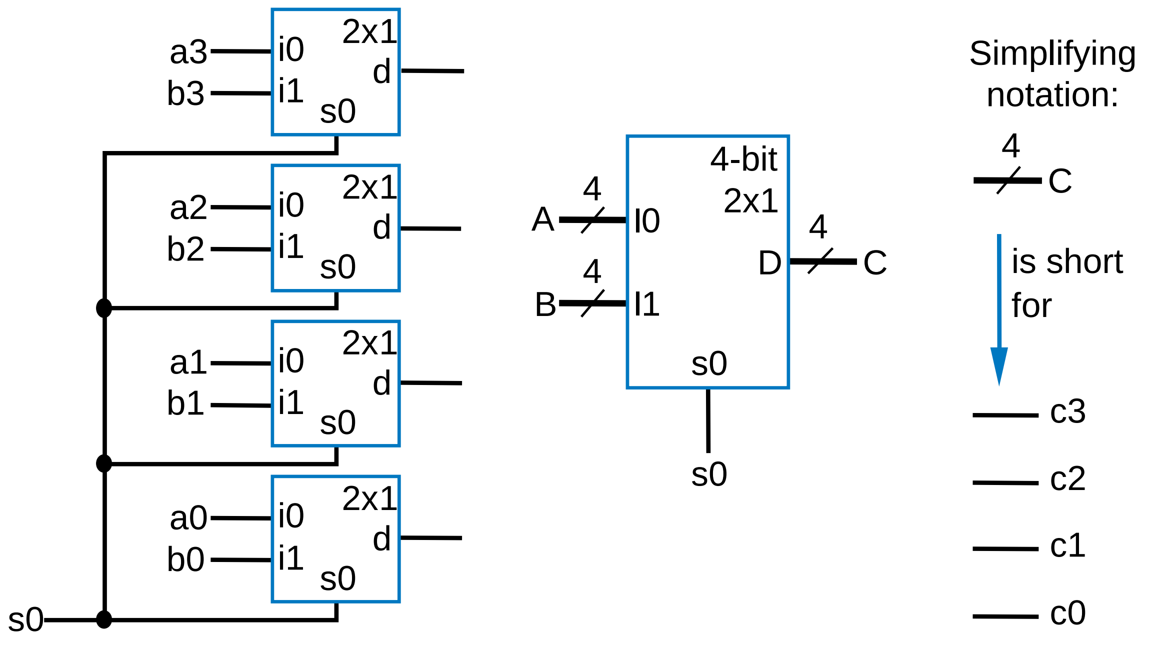


Mux Example

- City mayor can set four switches up or down, representing his/her vote on each of four proposals, numbered 0, 1, 2, 3
- City manager can display any such vote on large green/red LED (light) by setting two switches to represent binary 0, 1, 2, or 3
- Use 4x1 mux

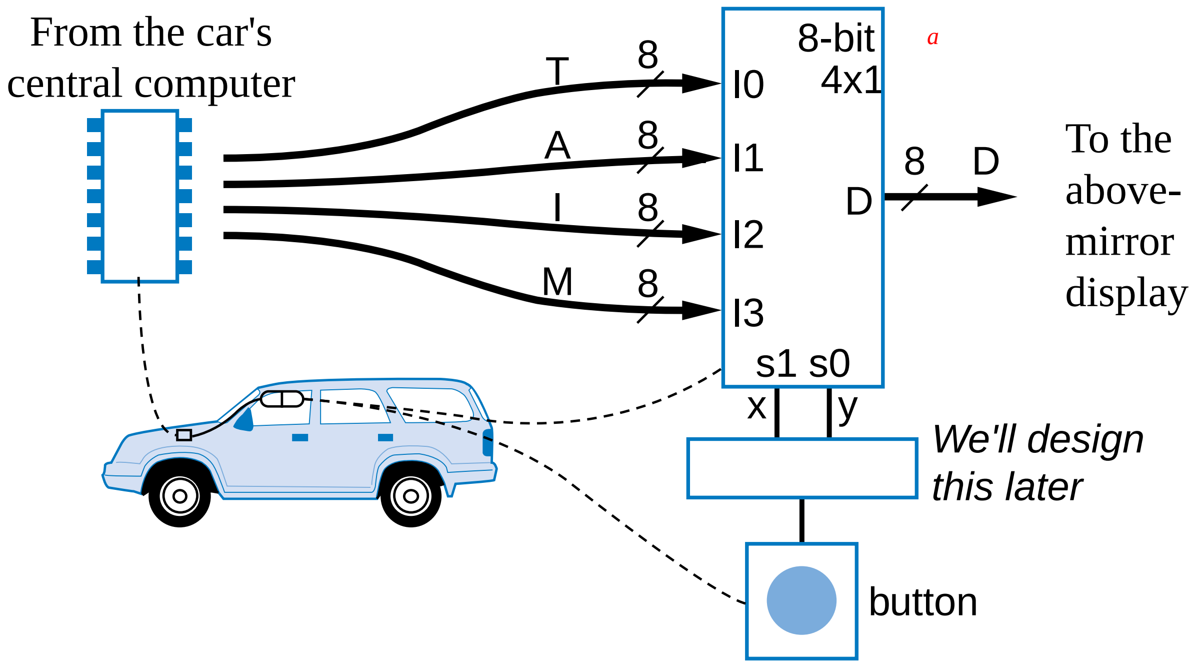


Muxes Commonly Together – N-bit Mux



- Ex: Two 4-bit inputs, A (a3 a2 a1 a0), and B (b3 b2 b1 b0)
  – 4-bit 2x1 mux (just four 2x1 muxes sharing a select line) can select between A or B

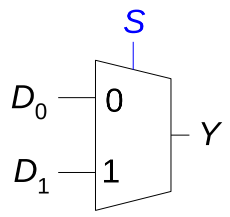
N-bit Mux Example



- Four possible display items
  - Temperature (T), Average miles-per-gallon (A), Instantaneous mpg (I), and Miles remaining (M) – each is 8-bits wide
  - Choose which to display on D using two inputs x and y
    - Pushing button sequences to the next item
  - Use 8-bit 4x1 mux

Multiplexer (Mux)

- Selects between one of *N* inputs to connect to output
- $\log_2 N$-bit select input – control input
- **Example:** **2:1 Mux**

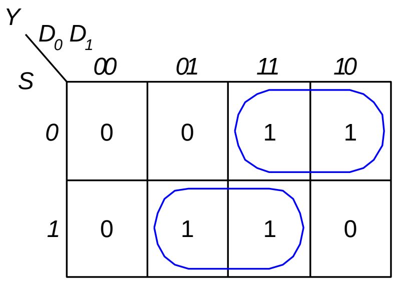


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

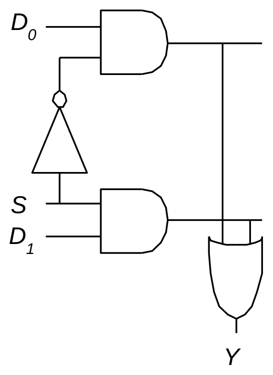
Multiplexer Implementations

- **Logic gates**
  - Sum-of-products form

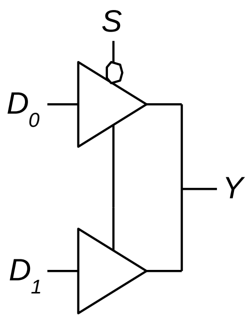


Y / S \ D_0D_1	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$Y = D_0\overline{S} + D_1S$



- **Tristates**
  - For an N-input mux, use N tristates
  - Turn on exactly one to select the appropriate input

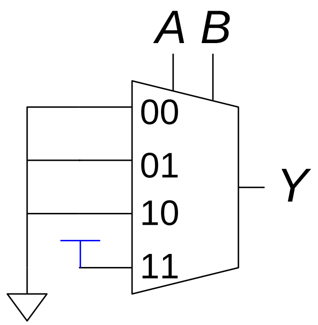


Logic using Multiplexers

- Using the mux as a lookup table

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

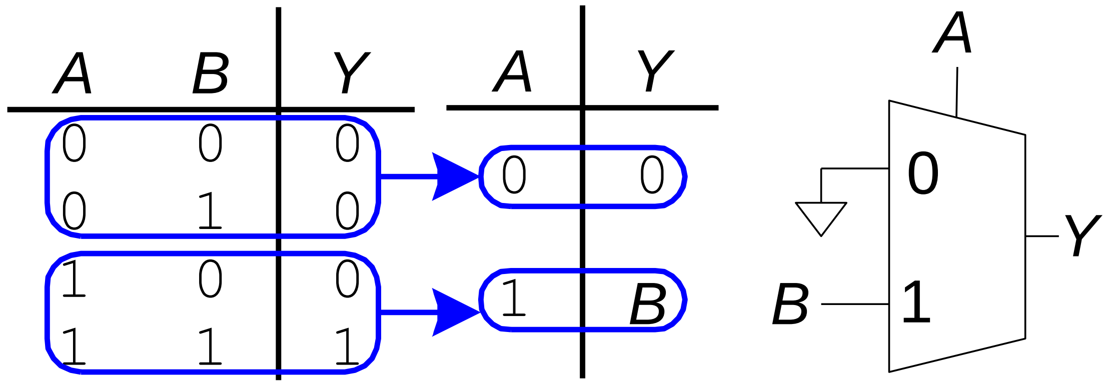
$$Y = AB$$



Logic using Multiplexers

- Reducing the size of the mux

$Y = AB$



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

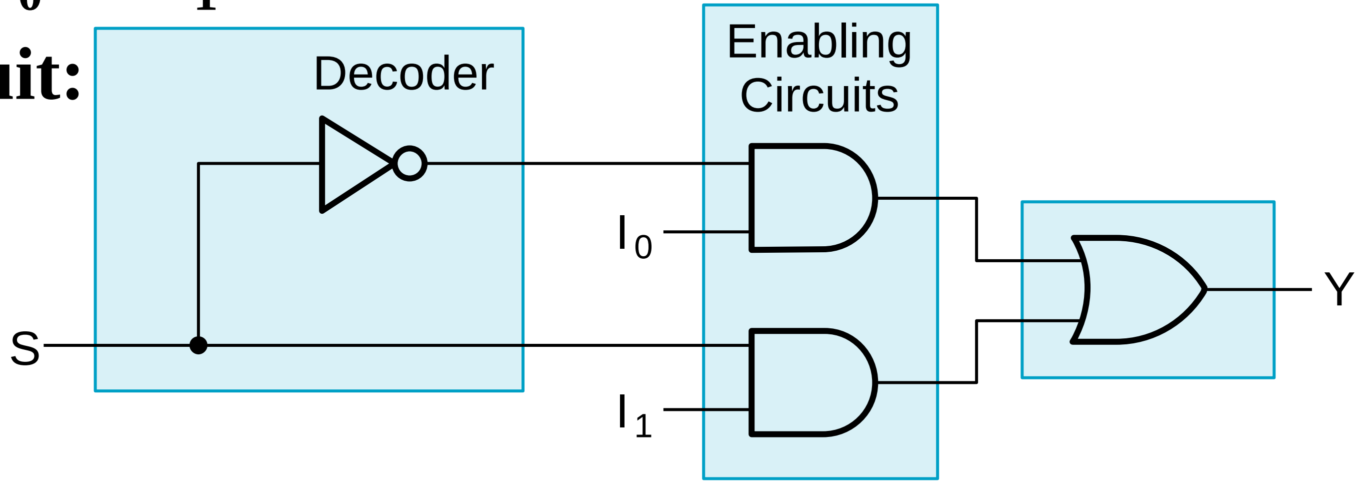
A	Y
0	0
1	B

2-to-1-Line Multiplexer

- **Since $2 = 2^1$, $n = 1$**
- **The single selection variable S has two values:**
  - **S = 0 selects input I_0**
  - **S = 1 selects input I_1**
- **The equation:**

$$Y = \overline{S}I_0 + SI_1$$

- **The circuit:**

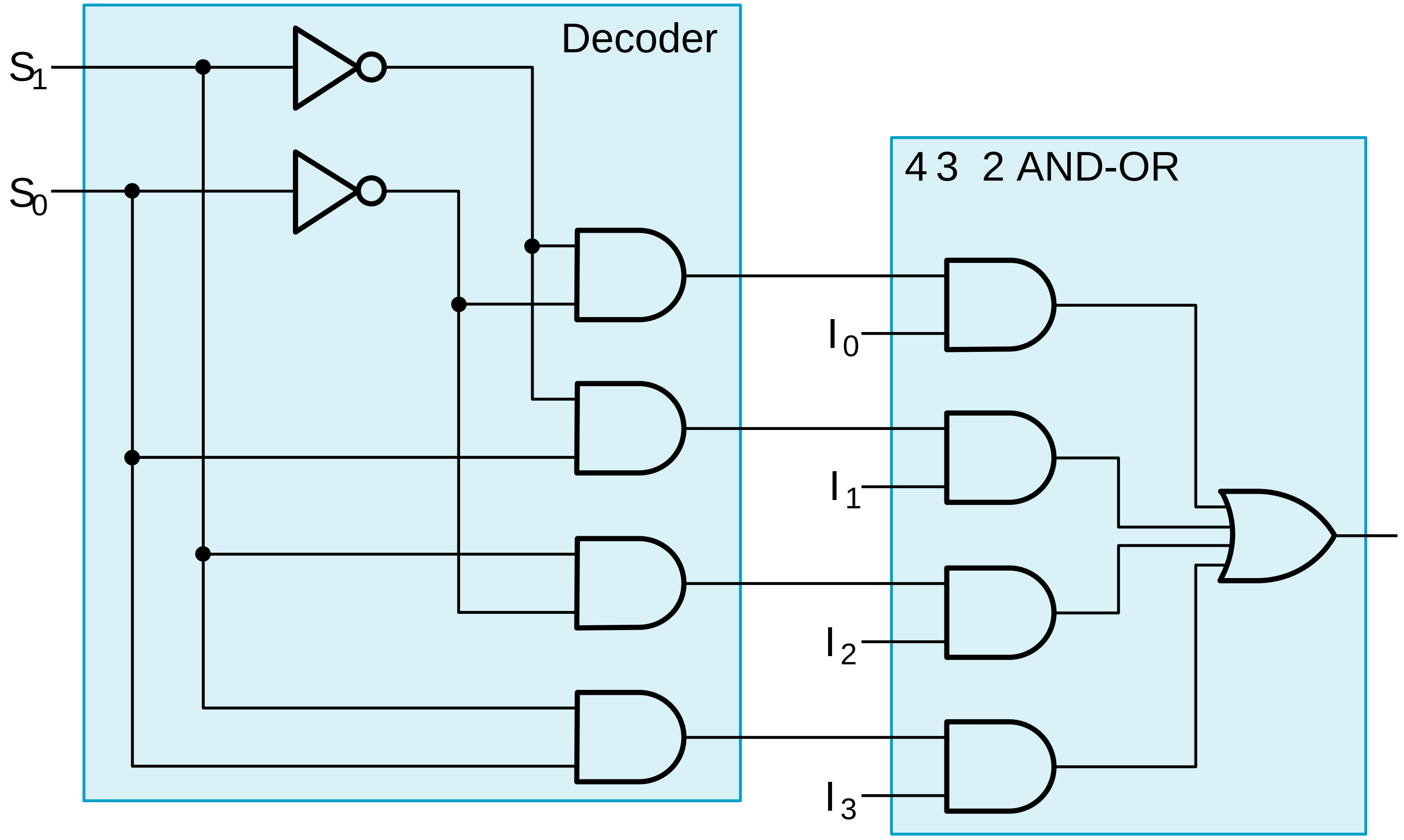


2-to-1-Line Multiplexer (continued)

- **Note the regions of the multiplexer circuit shown:**
  - **1-to-2-line Decoder**
  - **2 Enabling circuits**
  - **2-input OR gate**
- **To obtain a basis for multiplexer expansion, we combine the Enabling circuits and OR gate into a 2 × 2 AND-OR circuit:**
  - **1-to-2-line decoder**
  - **2 × 2 AND-OR**
- **In general, for an 2^n-to-1-line multiplexer:**
  - ***n*-to-2^n-line decoder**
  - **$2^n \times 2$ AND-OR**

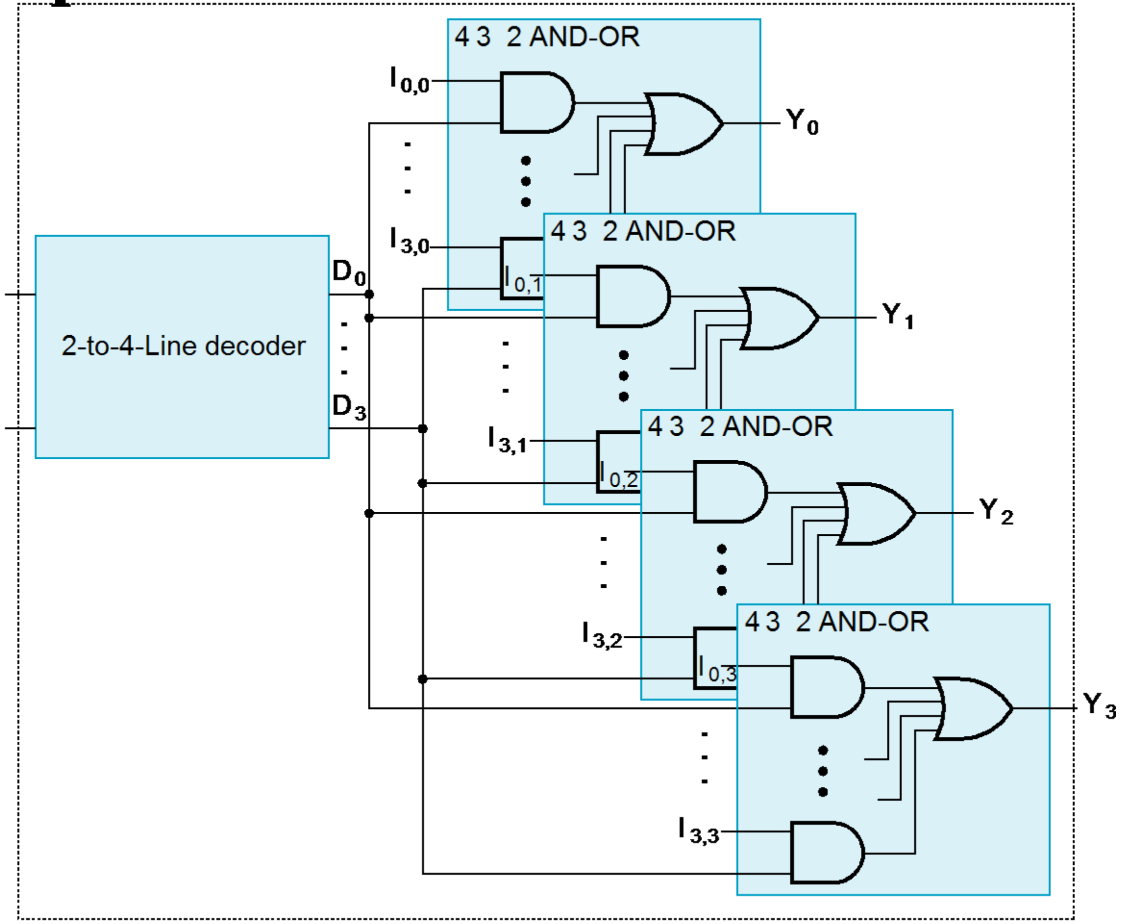
Example: 4-to-1-line Multiplexer

- 2-to-2^2-line decoder
- $2^2 \times$ 2 AND-OR



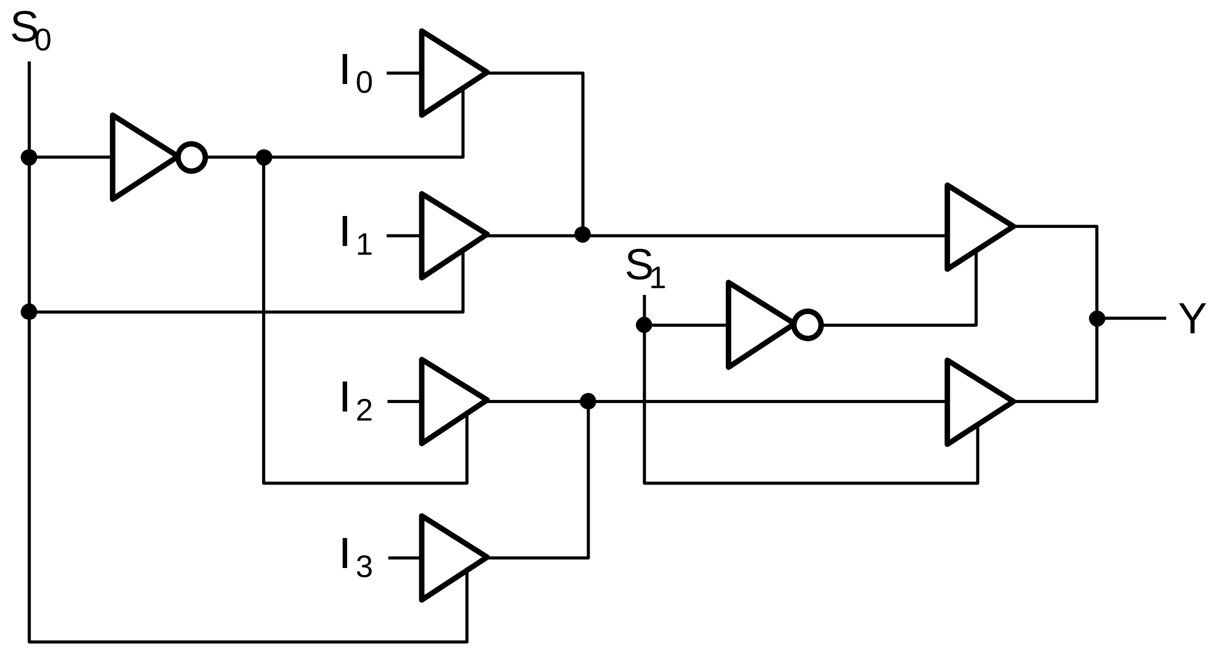
Multiplexer Width Expansion

- Select "vectors of bits" instead of "bits"
- Use multiple copies of $2^n \times 2$ AND-OR in parallel
- Example: 4-to-1-line quad multi-plexer



Other Selection Implementations

- **Three-state logic in place of AND-OR**



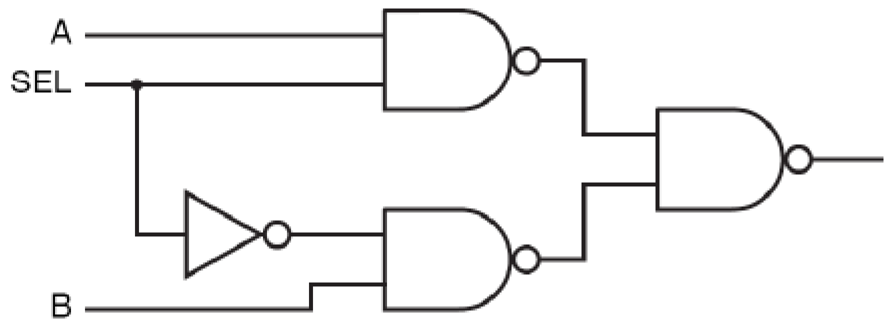
(b)

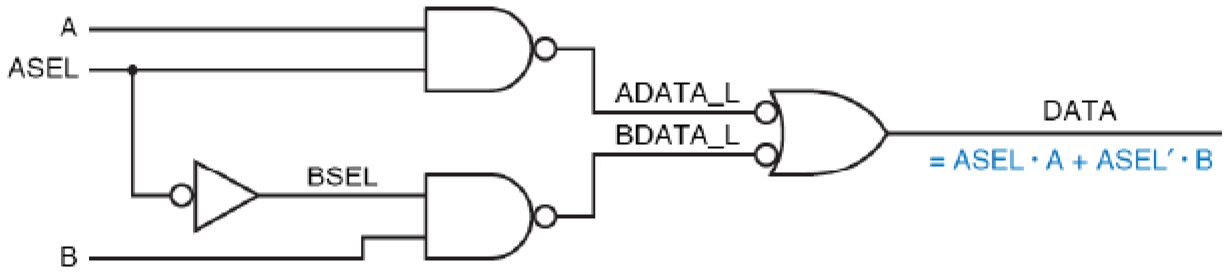
- **Gate input cost = 14 compared to 22 (or 18) for gate implementation**

Combinational Logic Implementation - Multiplexer Approach 1

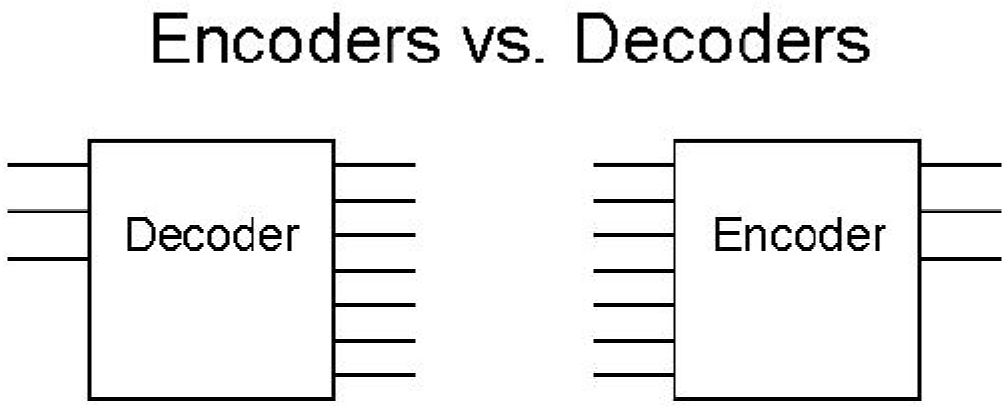
- **Implement *m* functions of *n* variables with:**
  - Sum-of-minterms expressions
  - An *m*-wide 2^n-to-1-line multiplexer
- **Design:**
  - Find the truth table for the functions.
  - In the order they appear in the truth table:
    - Apply the function input variables to the multiplexer inputs $S_{n-1}, \dots, S_0$
    - Label the outputs of the multiplexer with the output variables
  - Value-fix the information inputs to the multiplexer using the values from the truth table (for don't cares, apply either 0 or 1)

Examples of Multiplexers – 2-1



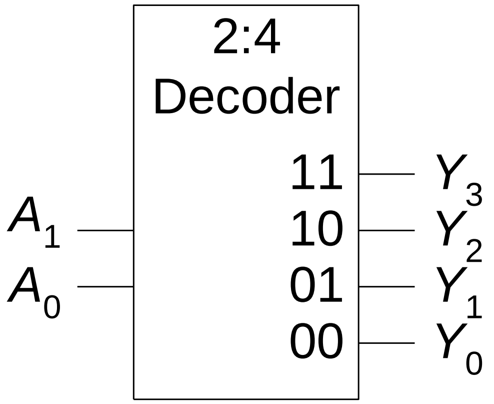


Encoder vs Decoder



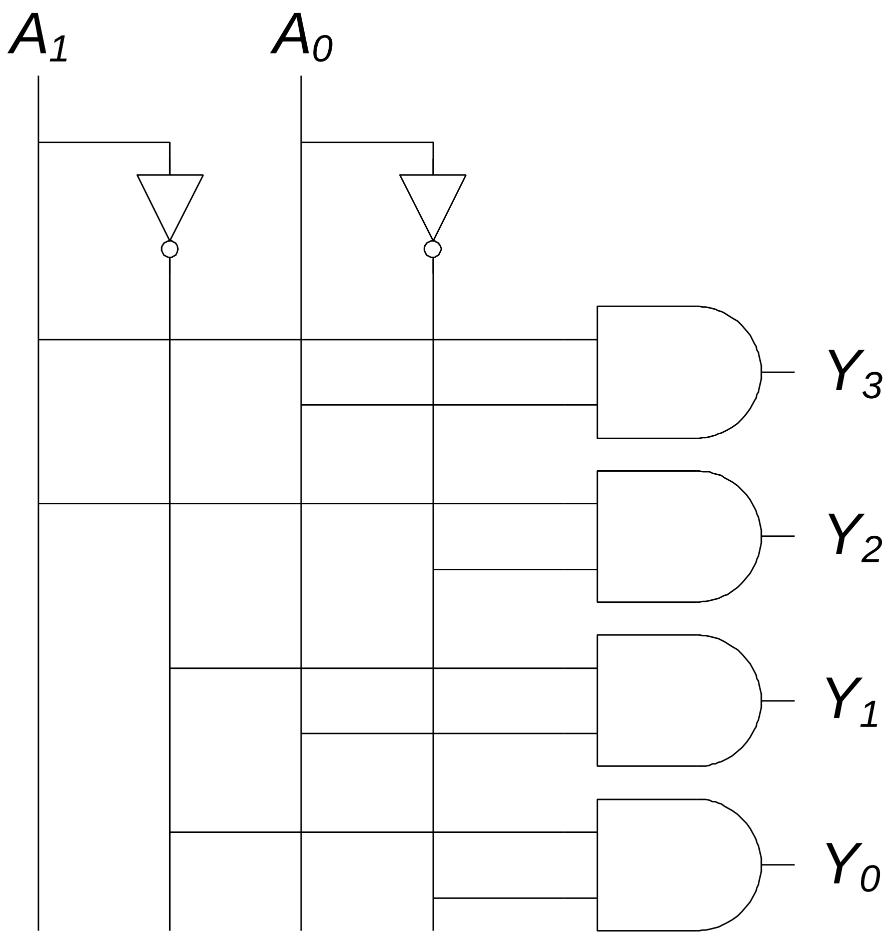
Decoders

- N inputs, 2^N outputs
- One-hot outputs: only one output HIGH at once



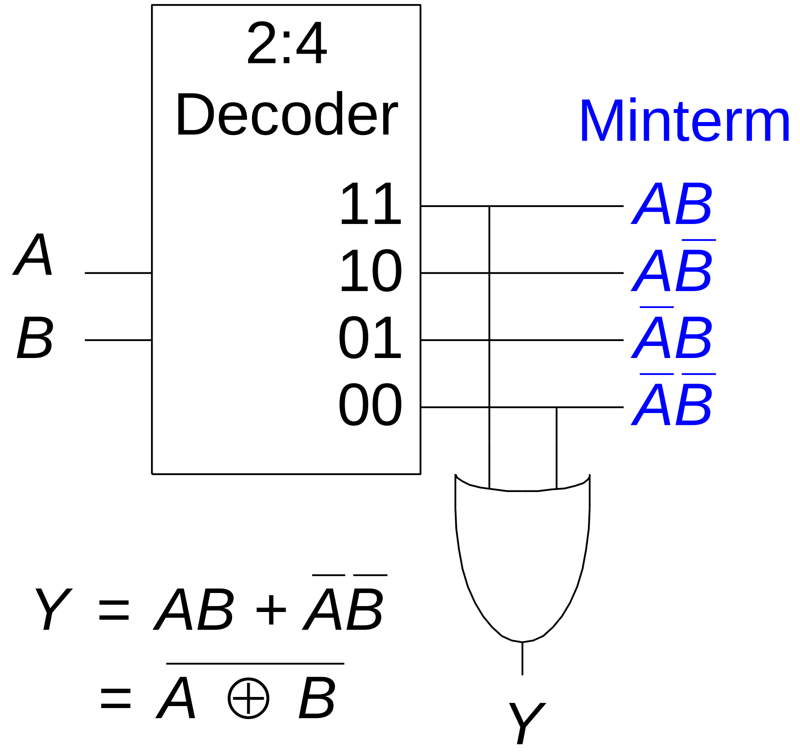
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoder Implementation



Logic Using Decoders

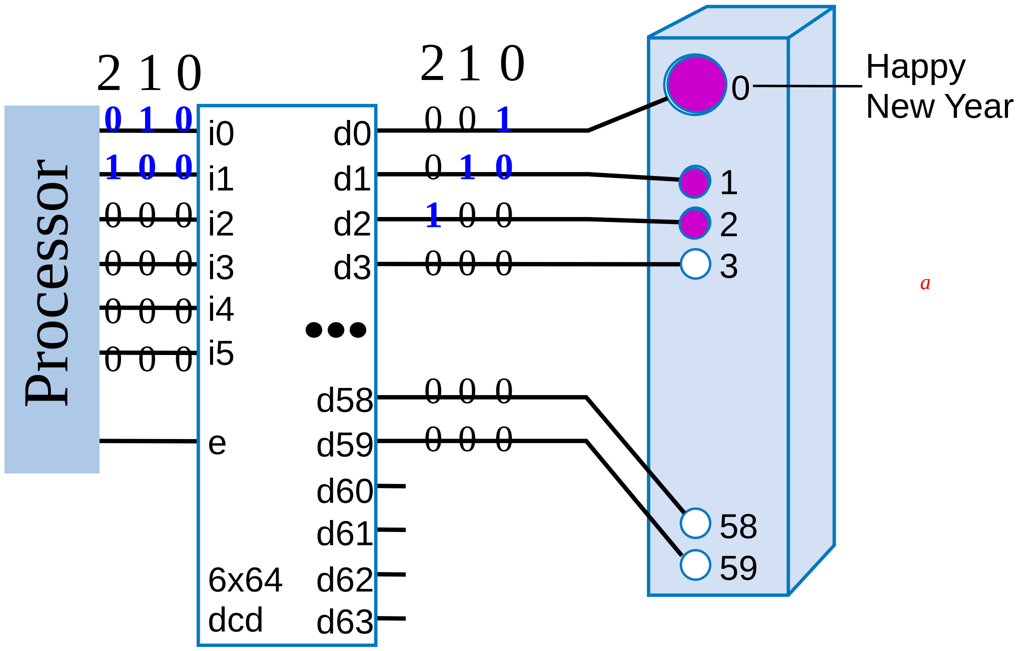
- OR minterms



$$Y = AB + \overline{A}\overline{B}$$
$$= \overline{A \oplus B}$$

Decoder Example

- New Year's Eve Countdown Display
  - Microprocessor counts from 59 down to 0 in binary on 6-bit output
  - Want illuminate one of 60 lights for each binary number
  - Use 6x64 decoder
    - 4 outputs unused

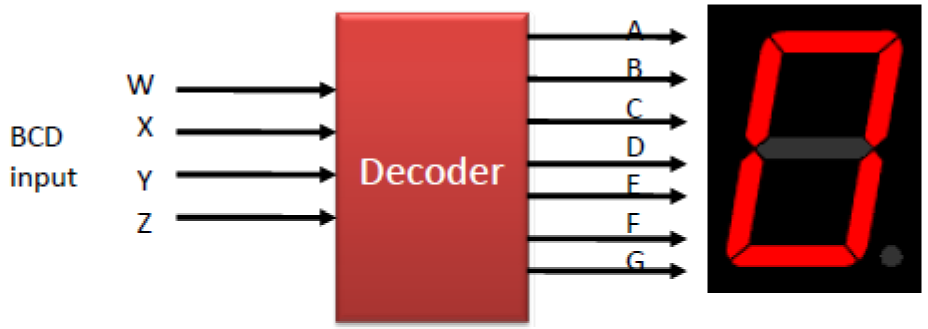


Decoder Applications

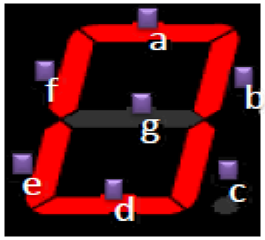
- Selecting different banks of memory
- Selecting different devices
- Instruction decoding

BCD To 7 segment

- BCD are 4 bit
- Design a decoder to drive 7 segment LED



Seven Segment Display

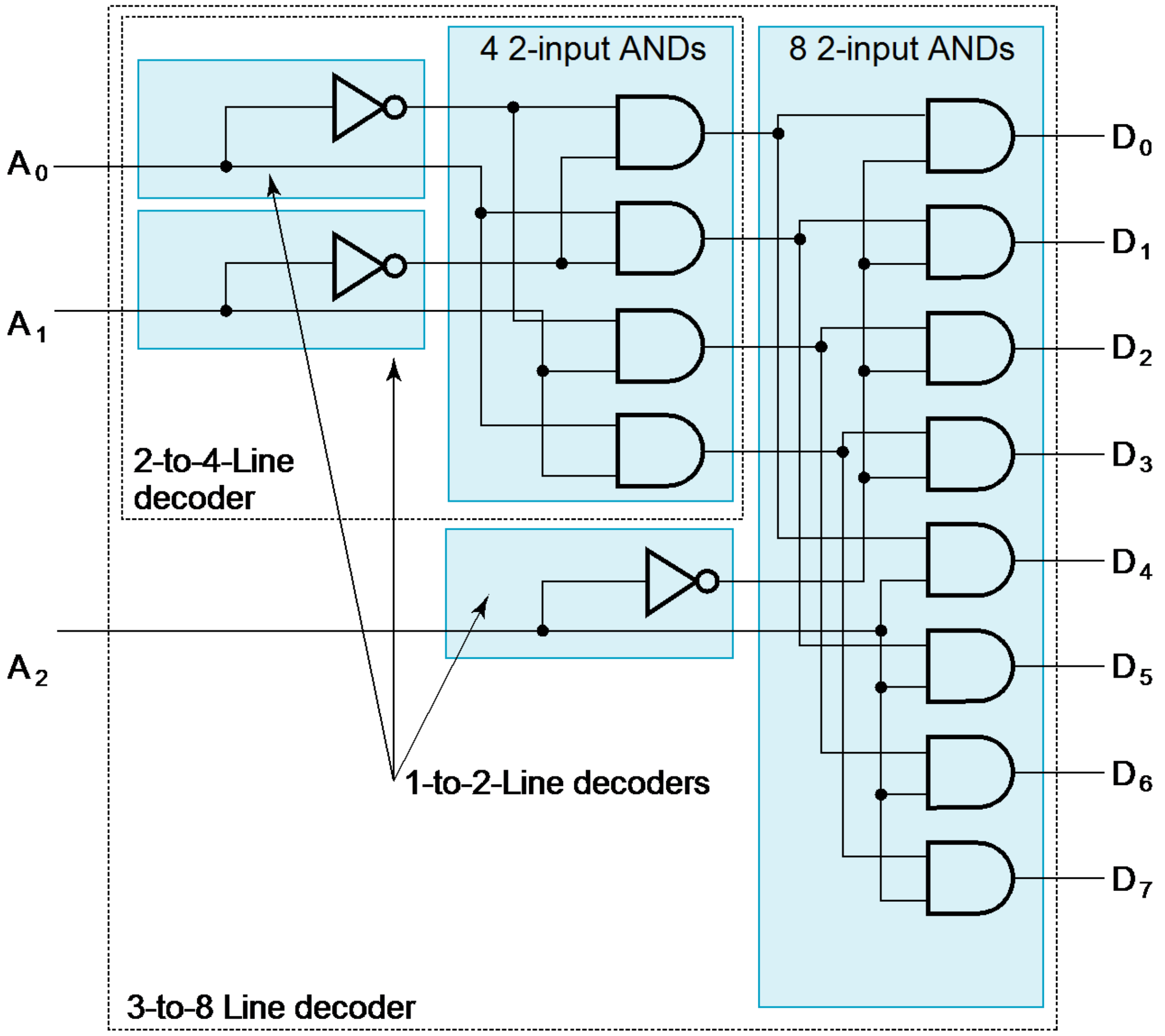


Activation of LEDs

- 0 : a,b,c,d,e,f
- 1: b,c
- 2:a,b,g,e,d
- 3:a,b,g,c,d
- 4:f,g,b,c
- 5:a,f,g,c,d
- 6:a,f,g,c,d,e
- 7:a,b,c
- 8:a,b,c,d,e,f,g
- 9:a,b,c,d,f,g

Decoder Expansion - Example 1

- **Result**



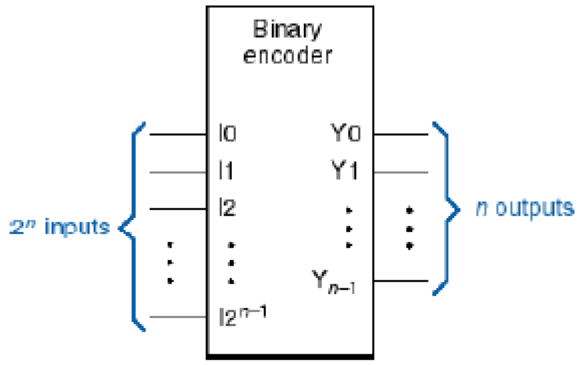
Decoder Expansion - Example 2

- 7-to-128-line decoder
  - Number of output ANDs = 128
  - Number of inputs to decoders driving output ANDs = 7
  - Closest possible split to equal
    - 4-to-16-line decoder
    - 3-to-8-line decoder
  - 4-to-16-line decoder
    - Number of output ANDs = 16
    - Number of inputs to decoders driving output ANDs = 2
    - Closest possible split to equal
      - 2 2-to-4-line decoders
  - Complete using known 3-8 and 2-to-4 line decoders

Encoding

- Encoding - the opposite of decoding - the conversion of an *m*-bit input code to a *n*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code
- Circuits that perform encoding are called *encoders*
- An encoder has 2^n (or fewer) input lines and *n* output lines which generate the binary code corresponding to the input values
- Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corres-ponding to the position in which the 1 appears.

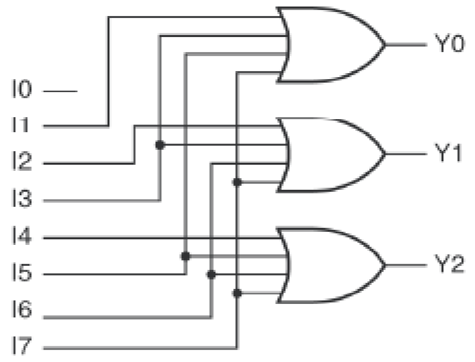
Binary Encoder

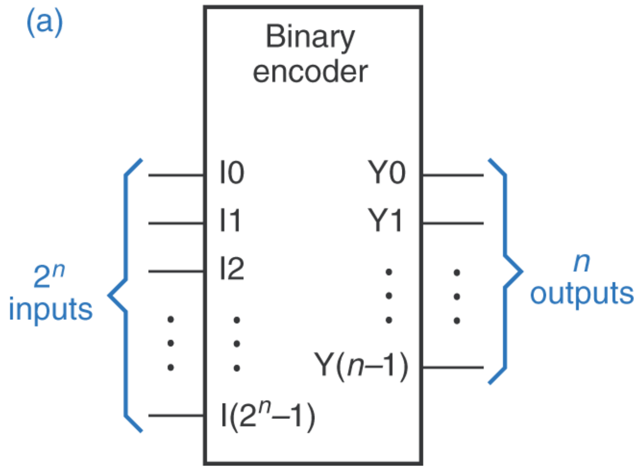


Binary Encoder

8-to-3 binary encoders

I0	I1	I2	I3	I4	I5	I6	I7	Y2	Y1	Y0
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





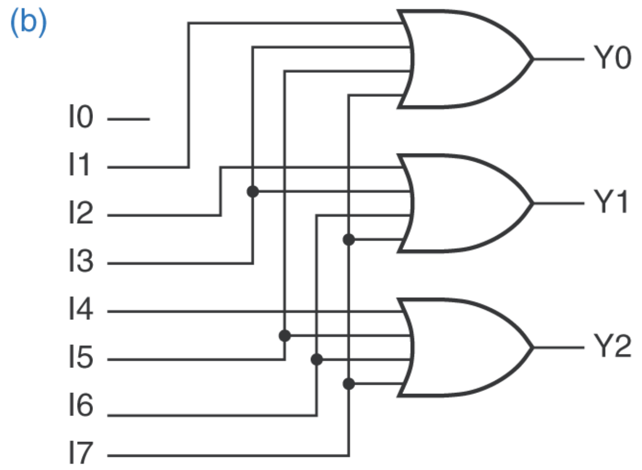


Figure 6-45

Binary encoder: (a) general structure; (b) 8-to-3 encoder.

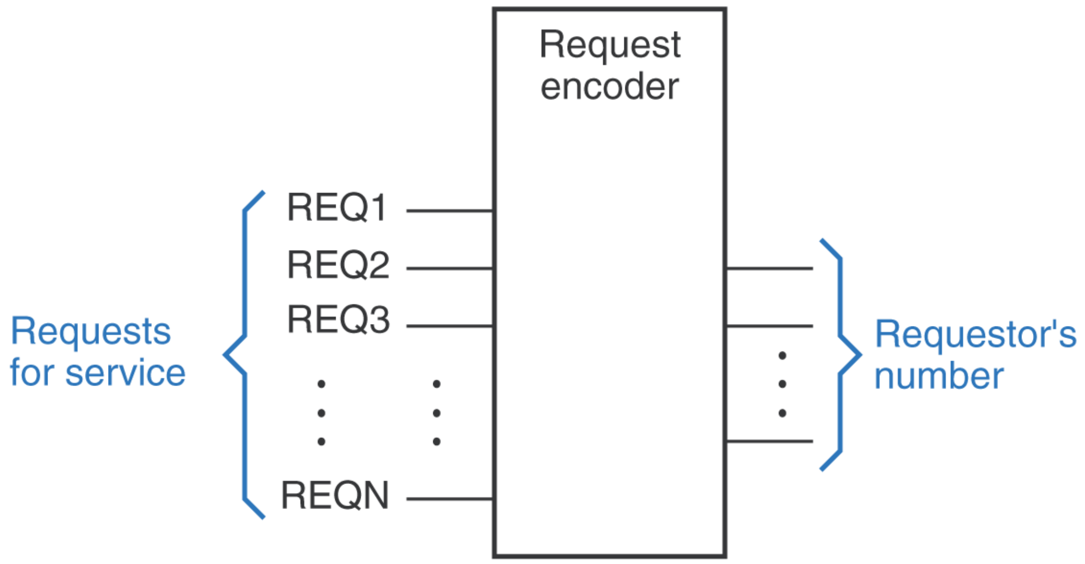


Figure 6-46

A system with 2^n requestors, and a “request encoder” that indicates which request signal is asserted at any time.

Encoder Example

- A decimal-to-BCD encoder
  - Inputs: 10 bits corresponding to decimal digits 0 through 9, (D_0, …, D_9)
  - Outputs: 4 bits with BCD codes
  - Function: If input bit D_i is a 1, then the output (A_3, A_2, A_1, A_0) is the BCD code for i,
- The truth table could be formed, but alternatively, the equations for each of the four outputs can be obtained directly.

Encoder Example (continued)

- Input D_i is a term in equation A_j if bit A_j is 1 in the binary value for i.
- Equations:

  $A_3 = D_8 + D_9$

  $A_2 = D_4 + D_5 + D_6 + D_7$

  $A_1 = D_2 + D_3 + D_6 + D_7$

  $A_0 = D_1 + D_3 + D_5 + D_7 + D_9$

- $F_1 = D_6 + D_7$ can be extracted from A_2 and A_1

Priority Encoder

- **If more than one input value is 1, then the encoder just designed does not work.**
- **One encoder that can accept all possible combinations of input values and produce a meaningful result is a *priority encoder*.**
- **Among the 1s that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the corresponding binary code for that position.**

Priority Encoder (Mano)

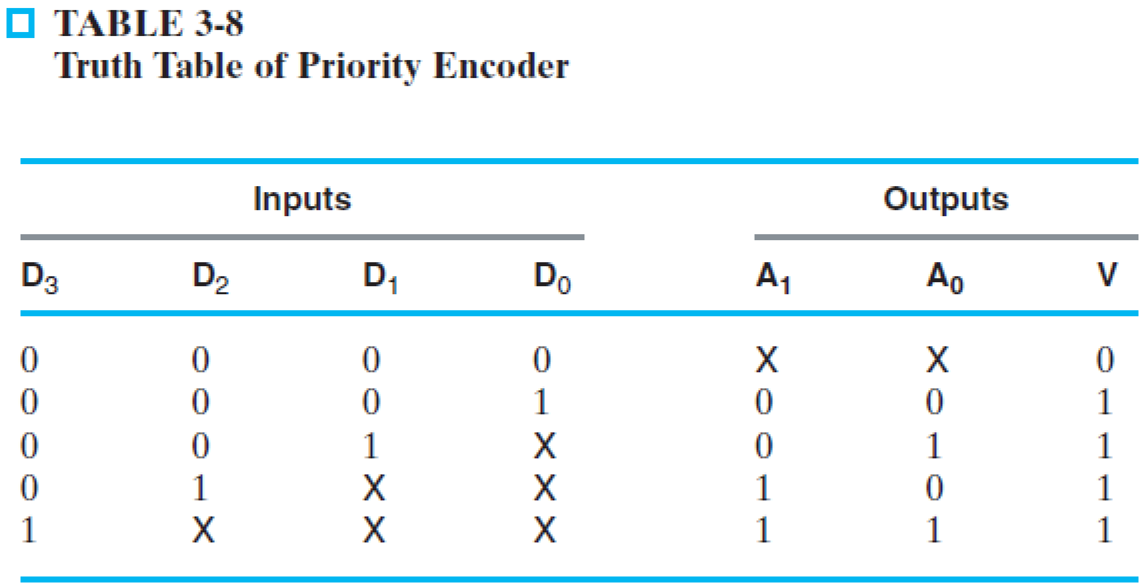


TABLE 3-8
Truth Table of Priority Encoder

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Priority Encoder Example

- Priority encoder with 5 inputs (D_4, D_3, D_2, D_1, D_0) - highest priority to most significant 1 present - Code outputs A2, A1, A0 and V where V indicates at least one 1 present.

No. of Min-terms/Row	Inputs					Outputs			
	D4	D3	D2	D1	D0	A2	A1	A0	V
1	0	0	0	0	0	X	X	X	0
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	1	X	X	X	X	1	0	0	1

- Xs in input part of table represent 0 or 1; thus table entries correspond to product terms instead of minterms.

Priority Encoder Example (continued)

- **Could use a K-map to get equations, but can be read directly from table and manually optimized if careful:**

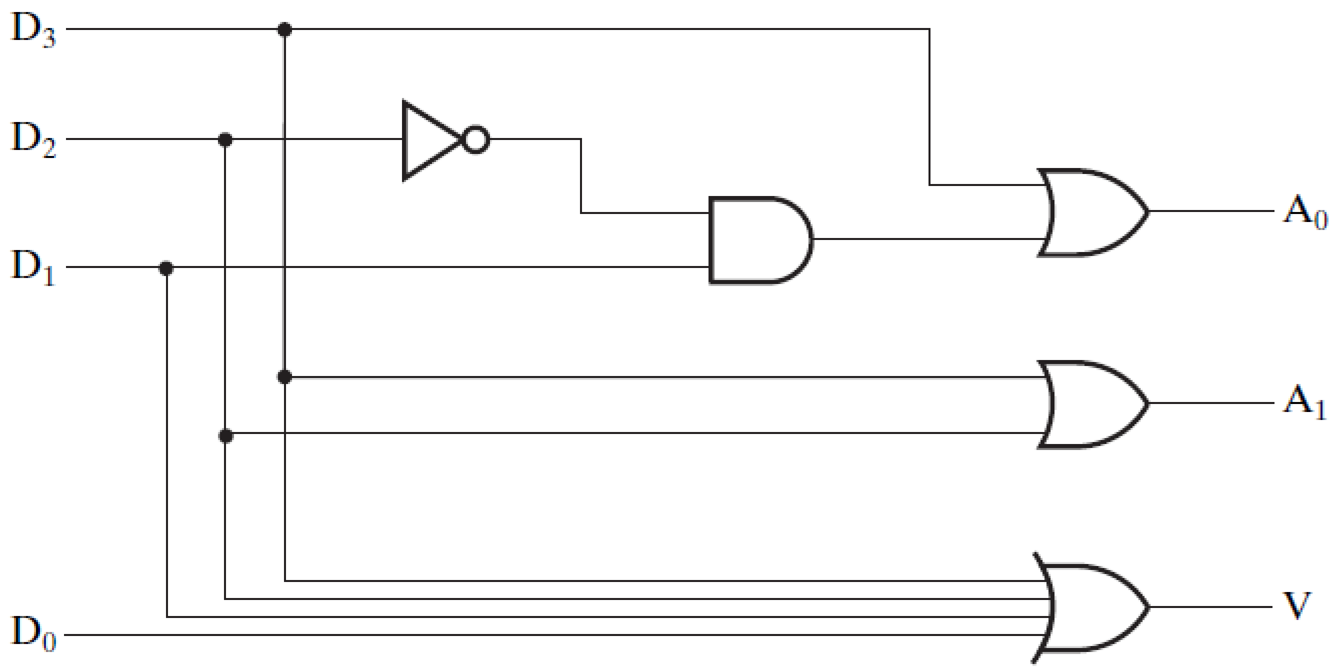
$$A_2 = D_4$$

$$A_1 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 D_2 = \overline{D}_4 F_1, \ F_1 = (D_3 + D_2)$$

$$A_0 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 \overline{D}_2 D_1 = \overline{D}_4 (D_3 + \overline{D}_2 D1)$$

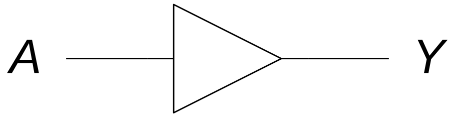
$$V = D_4 + F_1 + D_1 + D_0$$

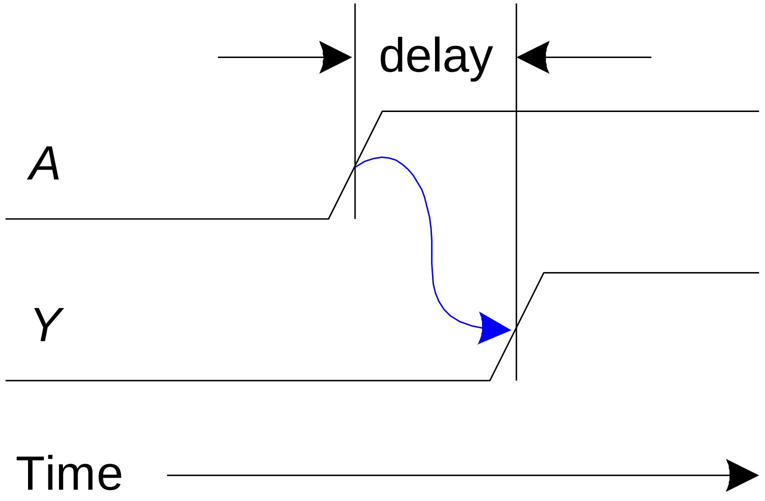
Priority Encoder (Mano)



Timing

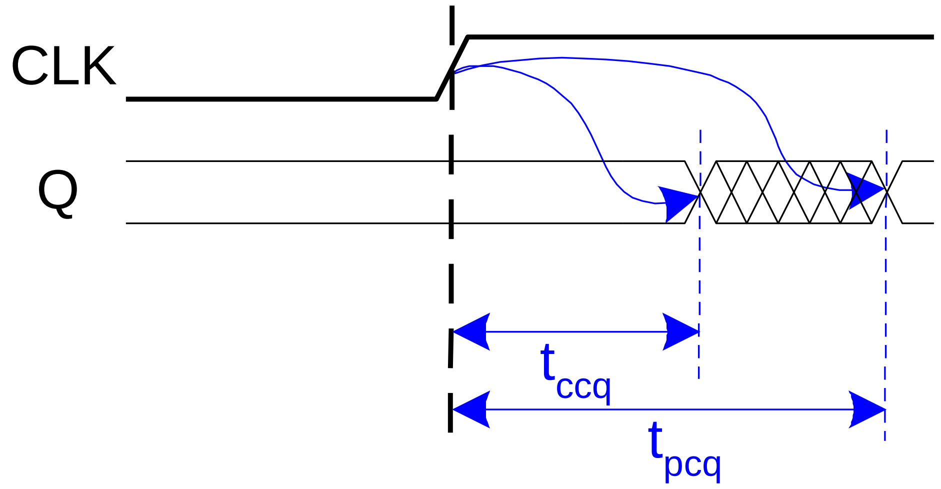
- Delay between input change and output changing
- How to build fast circuits?





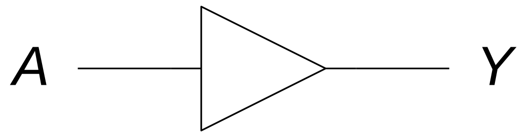
Output Timing Constraints

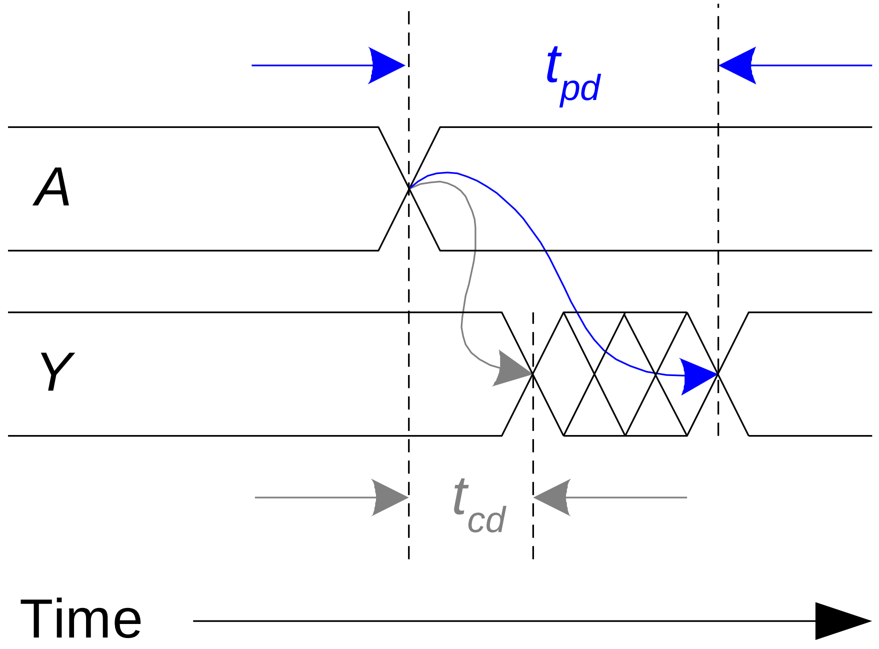
- **Propagation delay:** t_{pcq} = time after clock edge that the output Q is guaranteed to be stable (i.e., to stop changing)
- **Contamination delay:** t_{ccq} = time after clock edge that Q might be unstable (i.e., start changing)



Propagation & Contamination Delay

- **Propagation delay:** t_{pd} = max delay from input to output
- **Contamination delay:** t_{cd} = min delay from input to output

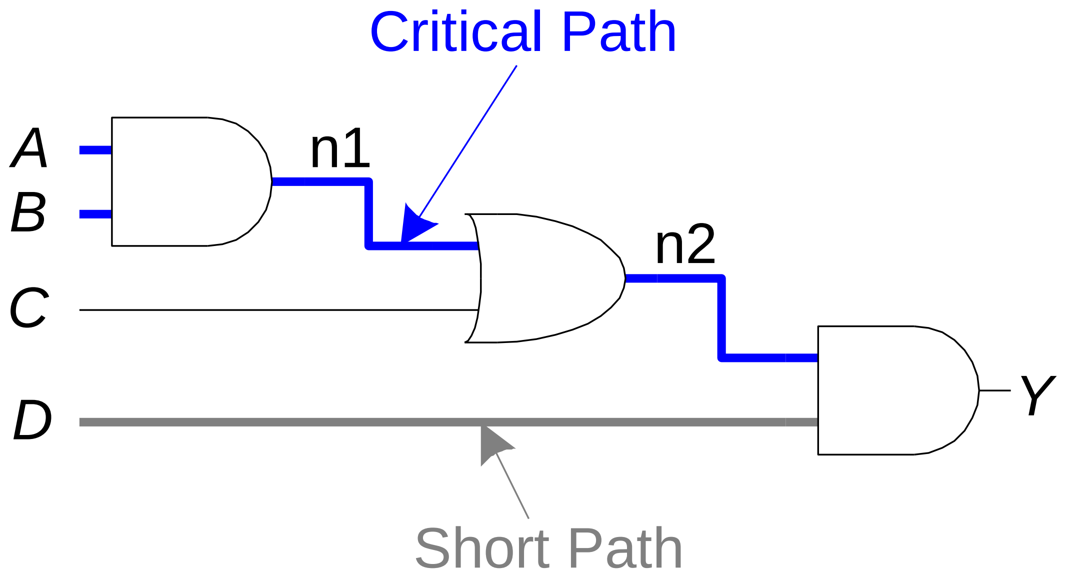




Propagation & Contamination Delay

- Delay is caused by
  - Capacitance and resistance in a circuit
  - Speed of light limitation
- Reasons why t_{pd} and t_{cd} may be different:
  - Different rising and falling delays
  - Multiple inputs and outputs, some of which are faster than others
  - Circuits slow down when hot and speed up when cold

Critical (Long) & Short Paths



Critical (Long) Path: $t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$

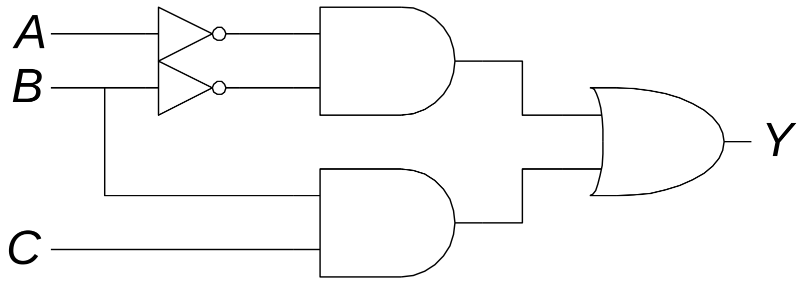
Short Path: $t_{cd} = t_{cd\_AND}$

Glitches

- When a single input change causes an output to change multiple times

Glitch Example

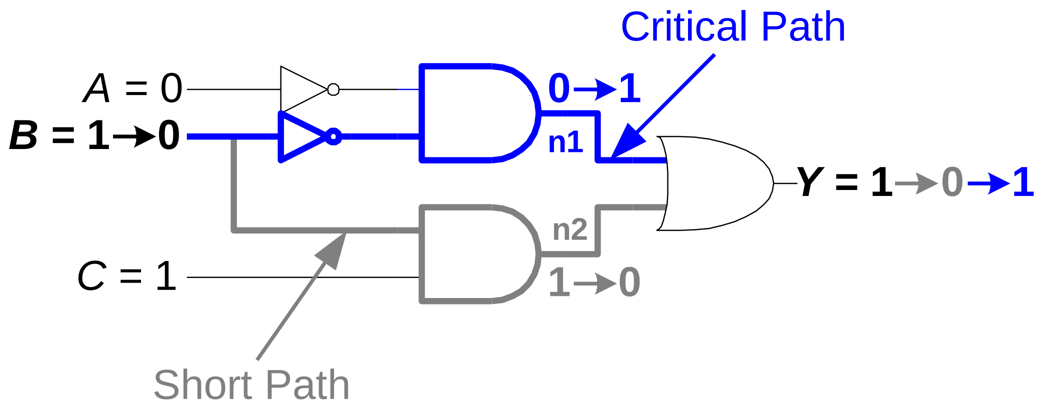
- What happens when A = 0, C = 1, B falls?

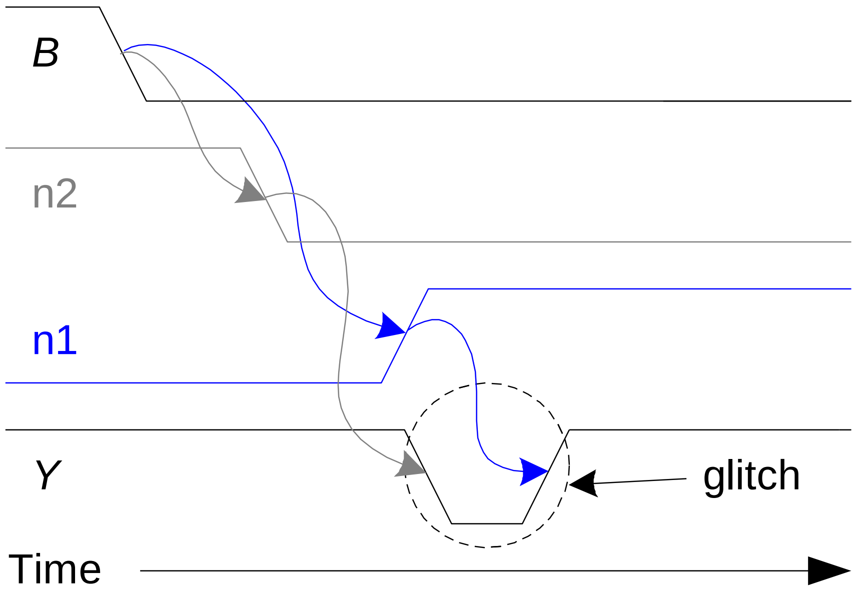


Y: C \ AB	00	01	11	10
0	1	0	0	0
1	1	1	1	0

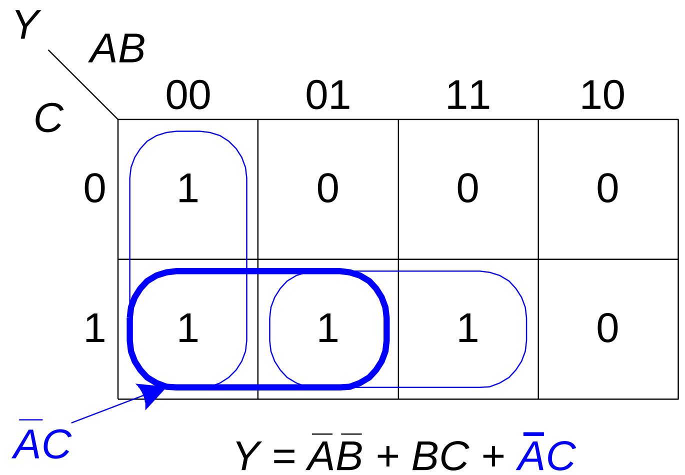
$$Y = \bar{A}\bar{B} + BC$$

Glitch Example (cont.)



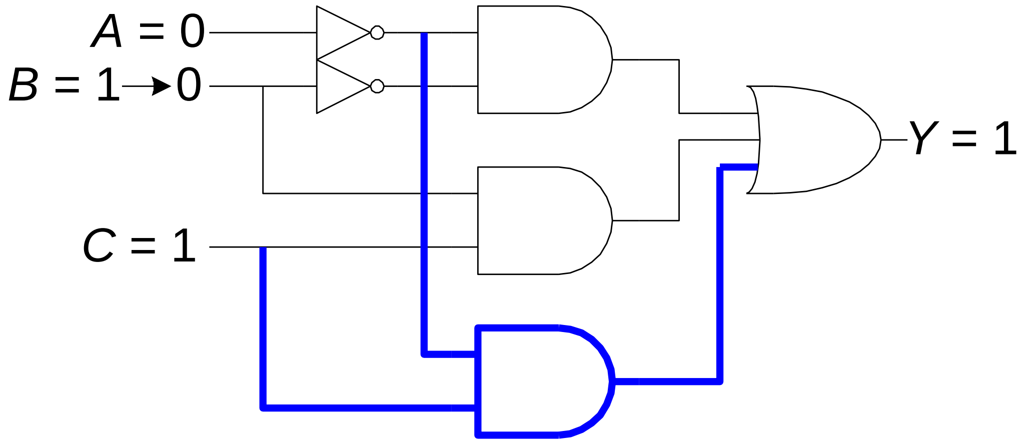


Fixing the Glitch



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

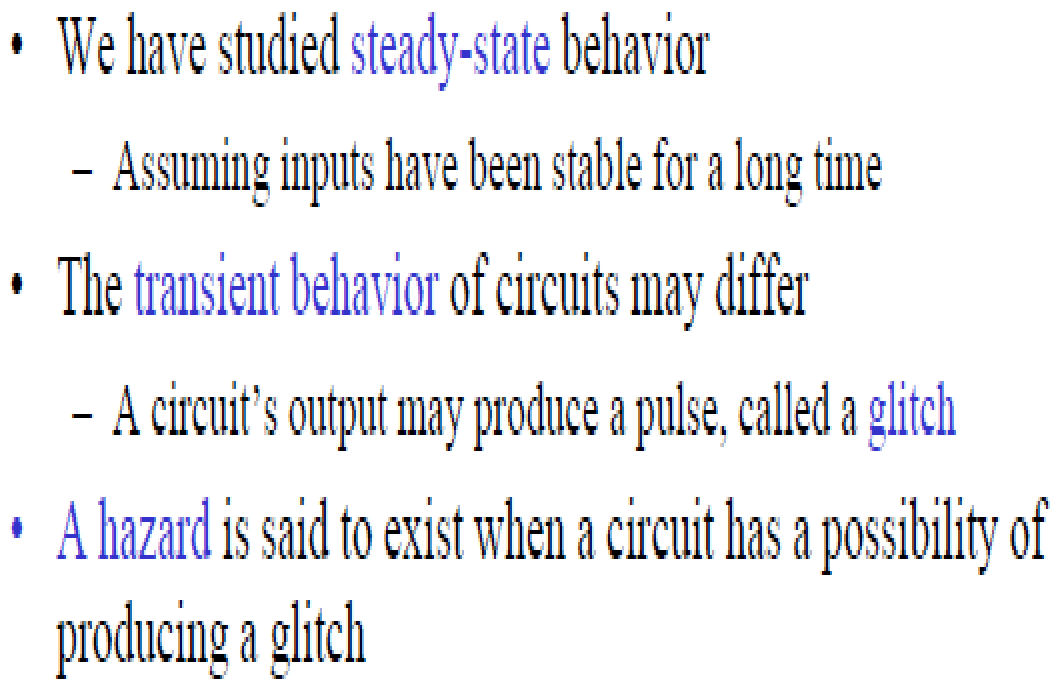
$Y = \bar{A}\bar{B} + BC + \bar{A}C$



Why Understand Glitches?

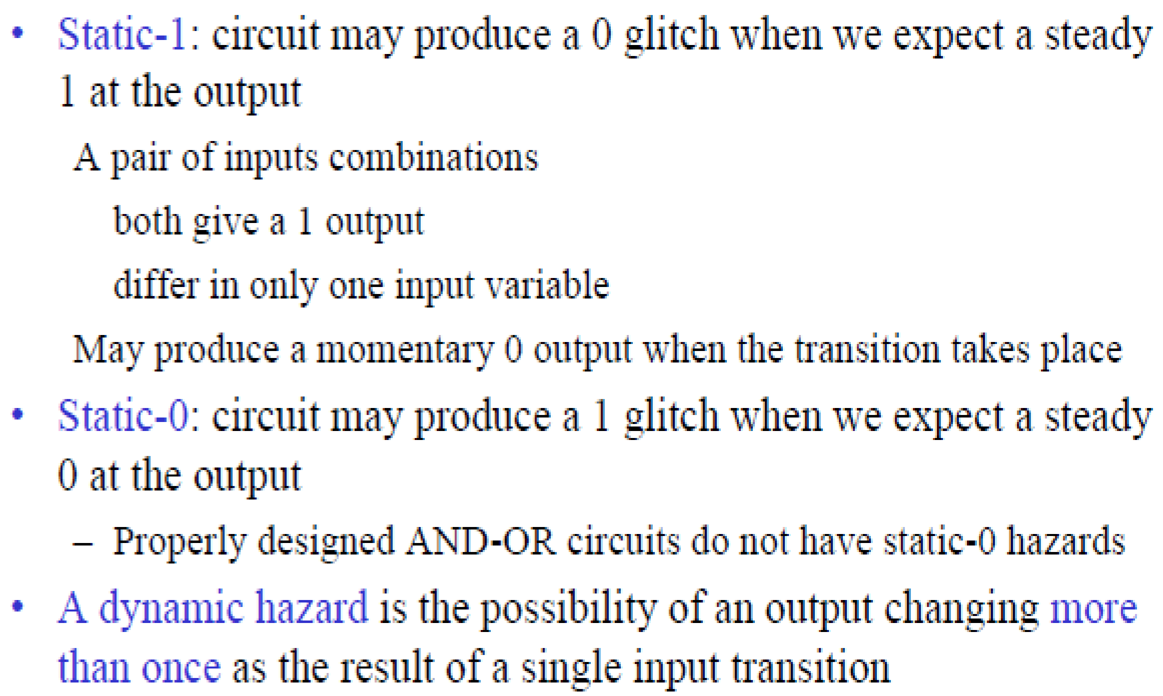
- Glitches don't cause problems because of **synchronous design** conventions (see Chapter 3)
- It's important to **recognize** a glitch: in simulations or on oscilloscope
- Can't get rid of all glitches – simultaneous transitions on multiple inputs can also cause glitches

Timing Hazard



- We have studied steady-state behavior
  - Assuming inputs have been stable for a long time
- The transient behavior of circuits may differ
  - A circuit's output may produce a pulse, called a glitch
- A hazard is said to exist when a circuit has a possibility of producing a glitch

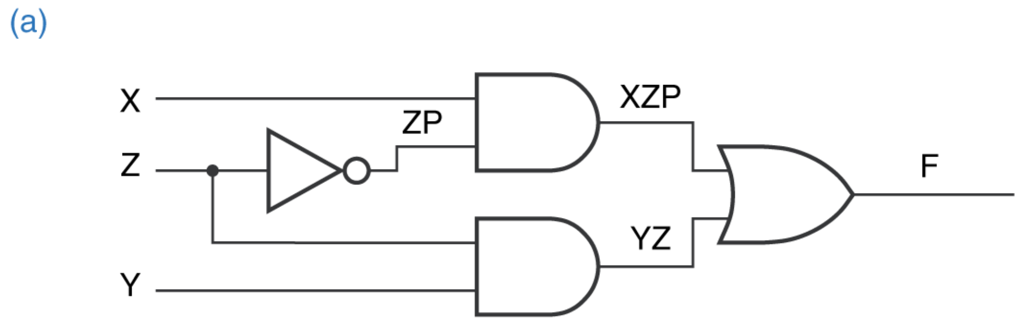
Static and Dynamic Hazard



- Static-1: circuit may produce a 0 glitch when we expect a steady 1 at the output
  - A pair of inputs combinations
    - both give a 1 output
    - differ in only one input variable
  - May produce a momentary 0 output when the transition takes place
- Static-0: circuit may produce a 1 glitch when we expect a steady 0 at the output
  - Properly designed AND-OR circuits do not have static-0 hazards
- A dynamic hazard is the possibility of an output changing more than once as the result of a single input transition

Static-1 Hazard

- Input combination differ in only one input variable
- Both give 1 output and there’s a possibility of momentary 0
- AND-OR
- X,Y,Z = 111 and X,Y,Z=110



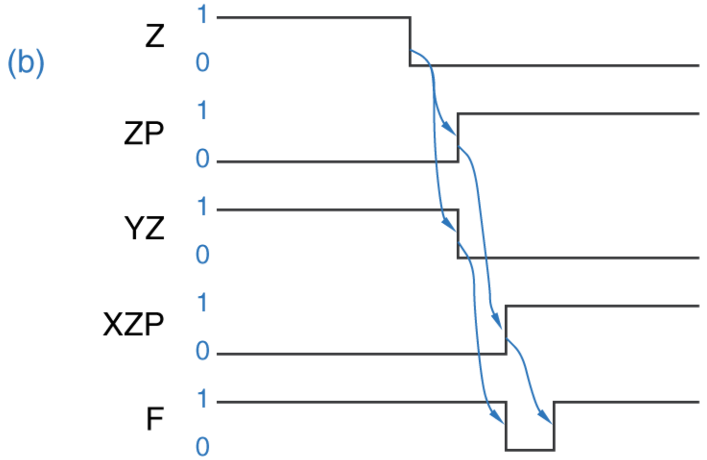


Figure 4-38

Circuit with a static-1 hazard: (a) logic diagram; (b) timing diagram.

Static-0 Hazard

- Input combination differ in only one input variable
- Both give 0 output and there’s a possibility of momentary 1
- OR-AND combination
- W,X,Y=000 and Z is changed

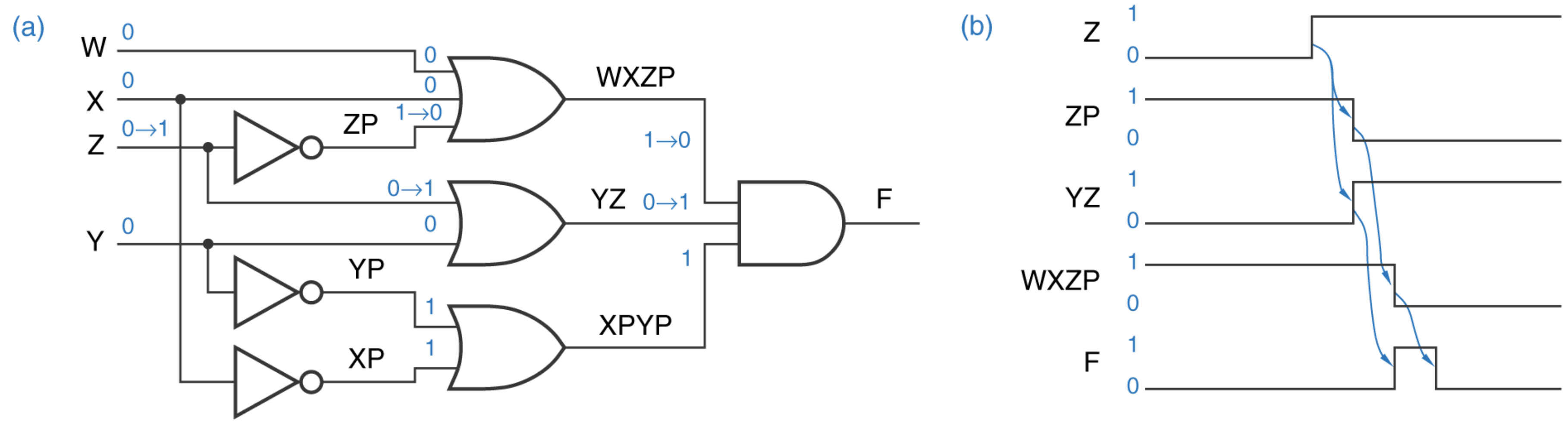
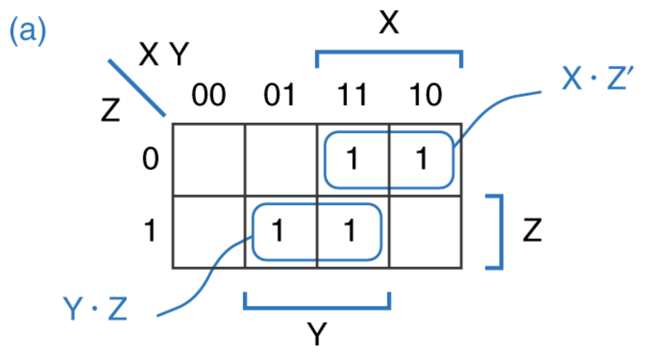
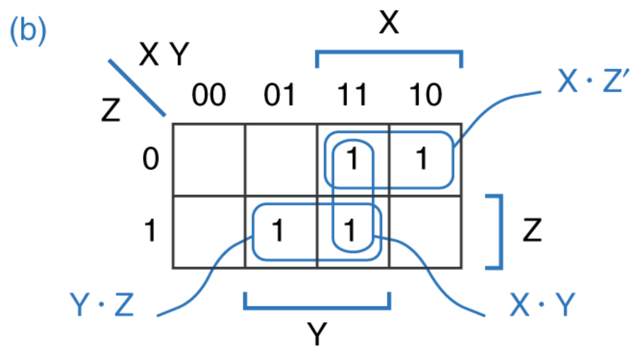


Figure 4-39

Circuit with static-0 hazards: (a) logic diagram; (b) timing diagram.



$F = X \cdot Z' + Y \cdot Z$



$F = X \cdot Z' + Y \cdot Z + X \cdot Y$

Figure 4-40

Karnaugh map for the circuit of Figure 4-38: (a) as originally designed; (b) with static-1 hazard eliminated.

Static-1 Hazard

- Properly designed AND-OR has no Static-0 Hazard but may have Static-1 Hazard
- X,Y,Z=111, X,Y,Z=110
- Output could glitch to 0 before the other input combination goes to 1
- Add an extra term – consensus term

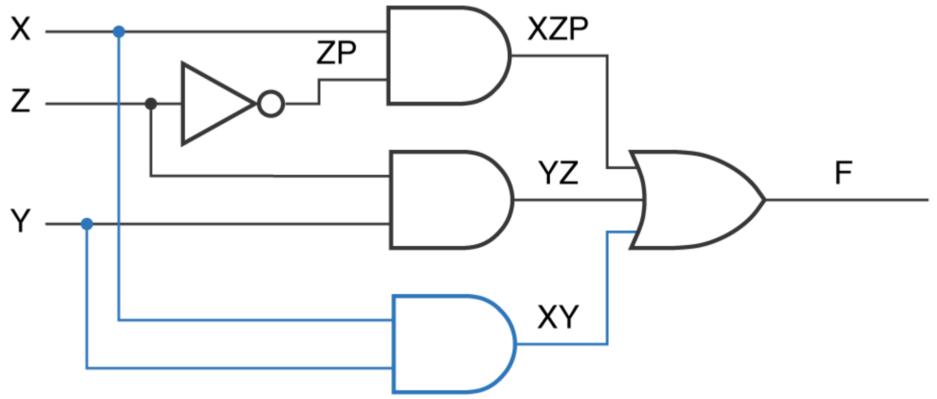
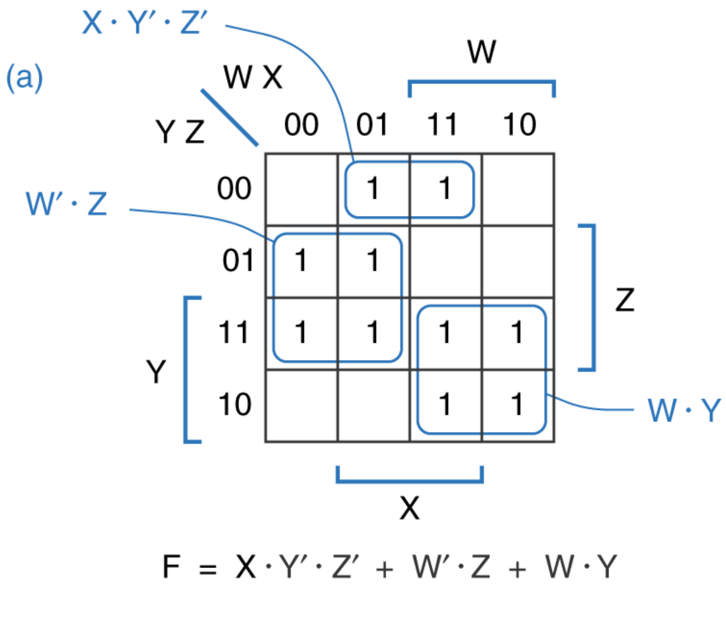


Figure 4-41

Circuit with static-1 hazard eliminated.



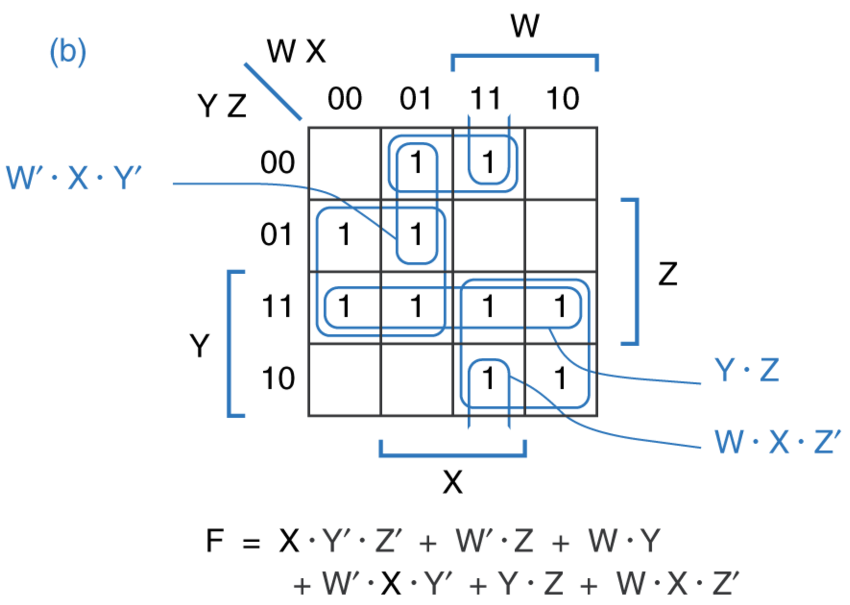

(b)
W′ · X · Y′
Y · Z
W · X · Z′
W X
Y Z
W
X
Y
Z
00
01
11
10
F = X · Y′ · Z′ + W′ · Z + W · Y + W′ · X · Y′ + Y · Z + W · X · Z′

Figure 4-42

Karnaugh map for another sum-of-products circuit:(a) as originally designed; (b) with extra product terms to cover static-1 hazards.

Dynamic Hazard

- Dynamic Hazards do not occur in a properly designed two level AND-OR or OR-AND circuits
- Finding Hazards are hard
- Hazard analysis is needed only in asynchronous sequential circuits and not needed in synchronous circuits as the outputs are not looked at until the clock edge

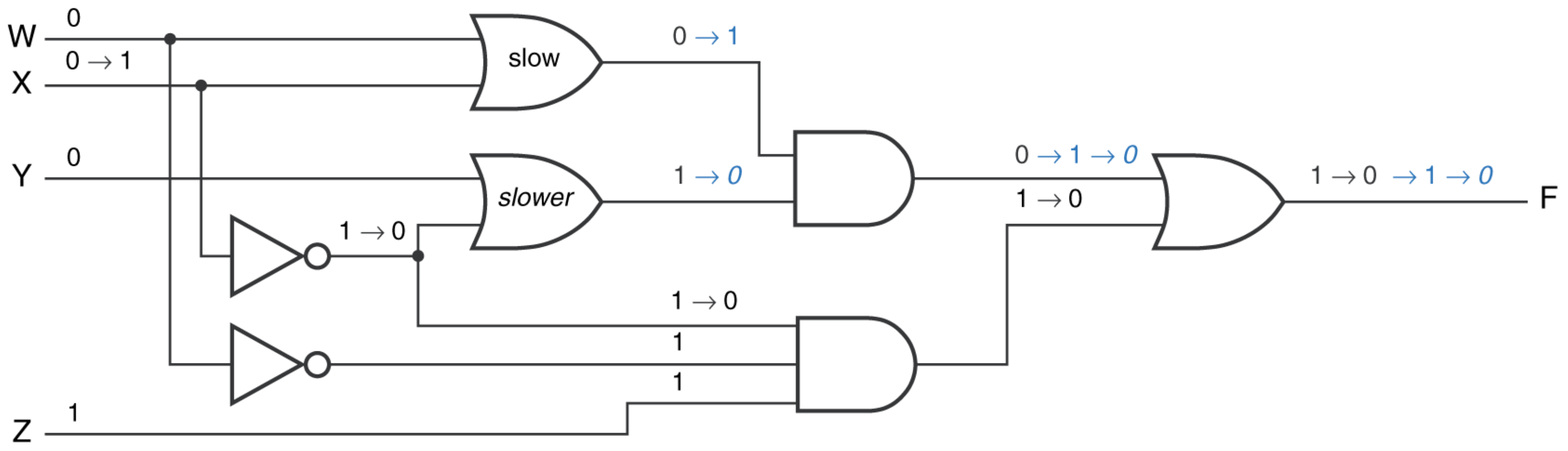


Figure 4-43

Circuit with a dynamic hazard.