Combinational Circuits: Design

Digital Design and Computer Architecture
Mohammad Sadrosadati
Frank K. Gürkaynak

http://safari.ethz.ch/ddca

What will we learn

- Converting desired functionality into Boolean equations
- Karnaugh Maps (simplification of equations)
- Circuits with more than one output
- Timing of combinational circuits

Boolean Equations Example

- **You are going to the cafeteria for lunch**
  - You won't eat lunch ($\overline{E}$)
  - If it is not open ($\overline{O}$) or
  - If they only serve cabbage (C)
- **Write a truth table for determining if you will eat lunch (E):**

open O	*only cabb.* C	*eat* E
0	0	
0	1	
1	0	
1	1	

Boolean Equations Example

- **You are going to the cafeteria for lunch**
  - You won't eat lunch ($\overline{E}$)
  - If it is not open ($\overline{O}$) or
  - If they only serve cabbage (C)
- **Write a truth table for determining if you will eat lunch (E)**

open O	*only cabb.* C	*eat* E
0	0	0
0	1	0
1	0	1
1	1	0

Some Definitions

- ***Complement:*** **variable with a bar over it**
  $\overline{A}$, $\overline{B}$, $\overline{C}$

- ***Literal:*** **variable or its complement**
  A , $\overline{A}$, B , $\overline{B}$, C , $\overline{C}$

- ***Implicant:*** **product (AND) of literals**
  $(A \cdot B \cdot \overline{C})$, $(\overline{A} \cdot C)$, $(B \cdot \overline{C})$

- ***Minterm:*** **product (AND) that includes all input variables**
  $(A \cdot B \cdot \overline{C})$, $(\overline{A} \cdot \overline{B} \cdot C)$, $(\overline{A} \cdot B \cdot \overline{C})$

- ***Maxterm:*** **sum (OR) that includes all input variables**
  $(A + \overline{B} + \overline{C})$, $(\overline{A} + B + \overline{C})$, $(A + B + \overline{C})$

Sum-of-Products (SOP) Form

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

$$Y = F(A, B) = ?$$

- **All Boolean equations can be written in SOP form**
  - Each row in a truth table has a minterm
  - A minterm is a product (AND) of literals
  - Each minterm is TRUE for that row (and only that row)
  - ***blackboard***
- **Formed by ORing the minterms for which the output is TRUE**

Sum-of-Products (SOP) Form

A	B	Y	minterm
0	0	0	$\overline{A}\ \overline{B}$
0	1	1	$\overline{A}\ B$
1	0	0	$A\ \overline{B}$
1	1	1	$A\ B$

$$Y = F(A, B)$$
$$= (\overline{A} \cdot B) + (A \cdot B)$$

- **All Boolean equations can be written in SOP form**
  - Each row in a truth table has a minterm
  - A minterm is a product (AND) of literals
  - Each minterm is TRUE for that row (and only that row)
  - ***blackboard***
- **Formed by ORing the minterms for which the output is TRUE**

The Dual: Product-of-Sums (POS) Form

- **All Boolean equations can be written in POS form**
  - Each row in a truth table has a maxterm
  - A maxterm is a sum (OR) of literals
  - Each maxterm is FALSE for that row (and only that row)
- **Formed by ANDing the maxterms for which the output is FALSE**

A	*B*	*Y*	maxterm
0	0	0	$A + B$
0	1	1	$A + \overline{B}$
1	0	0	$\overline{A} + B$
1	1	1	$\overline{A} + \overline{B}$

$$Y = F(A, B) = (A + B) \cdot (\overline{A} + B)$$

SOP & POS Form

- **SOP: sum-of-products**

O	C	E	minterm
0	0	0	$\overline{O}\,\overline{C}$
0	1	0	$\overline{O}\,C$
1	0	1	$O\,\overline{C}$
1	1	0	$O\,C$

- **POS: product-of-sums**

O	C	E	maxterm
0	0	0	$O + C$
0	1	0	$O + \overline{C}$
1	0	1	$\overline{O} + C$
1	1	0	$\overline{O} + \overline{C}$

SOP or POS?

SOP & POS Form

- **SOP: sum-of-products**

O	C	E	minterm
0	0	0	$\overline{O}\ \overline{C}$
0	1	0	$\overline{O}\ C$
1	0	1	$O\ \overline{C}$
1	1	0	$O\ C$

$$E = O \cdot \overline{C}$$

SOP shorter if the output is TRUE in only a few cases

- **POS: product-of-sums**

O	C	E	maxterm
0	0	0	$O + C$
0	1	0	$O + \overline{C}$
1	0	1	$\overline{O} + C$
1	1	0	$\overline{O} + \overline{C}$

$$E = (O + C) \cdot (O + \overline{C}) \cdot (\overline{O} + \overline{C})$$

POS is shorter if the output is FALSE in only a few cases

Karnaugh Maps (K-Maps)

- **Boolean expressions can be minimized by combining terms**
- **K-maps minimize equations graphically**

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

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

Y: C \ AB	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

- ***Blackboard***
- ***Works well for up to four variables***

Karnaugh Map Rules

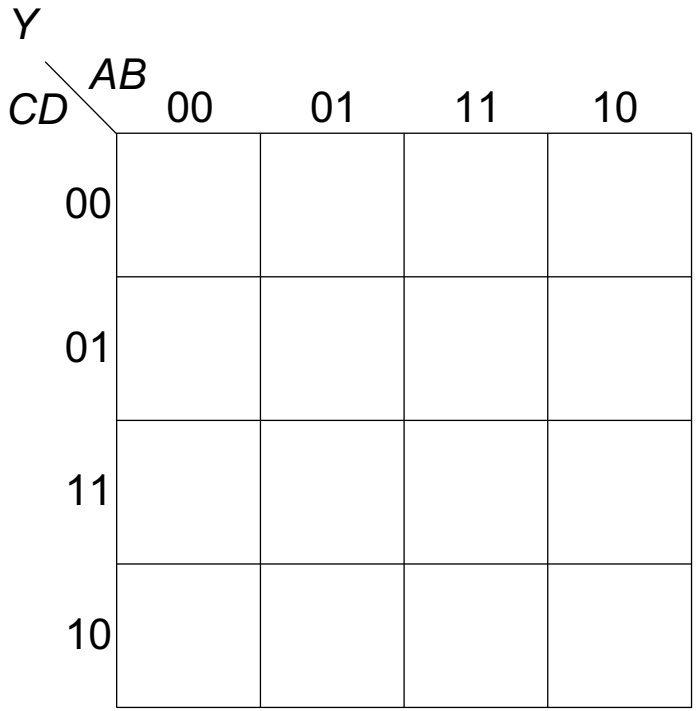
- **Special order for bit combinations: 00, 01, 11, 10**
  (only one bit changes from one to next)
- **Every 1 in a K-map must be circled at least once**
- **Each circle must span a power of 2 (i.e. 1, 2, 4) squares in each direction**
- **Each circle must be as large as possible**
- **A circle may wrap around the edges of the K-map**
- **A "don't care" (X) is circled only if it helps minimize the equation**

Karnaugh Map Example

- **Blackboard**

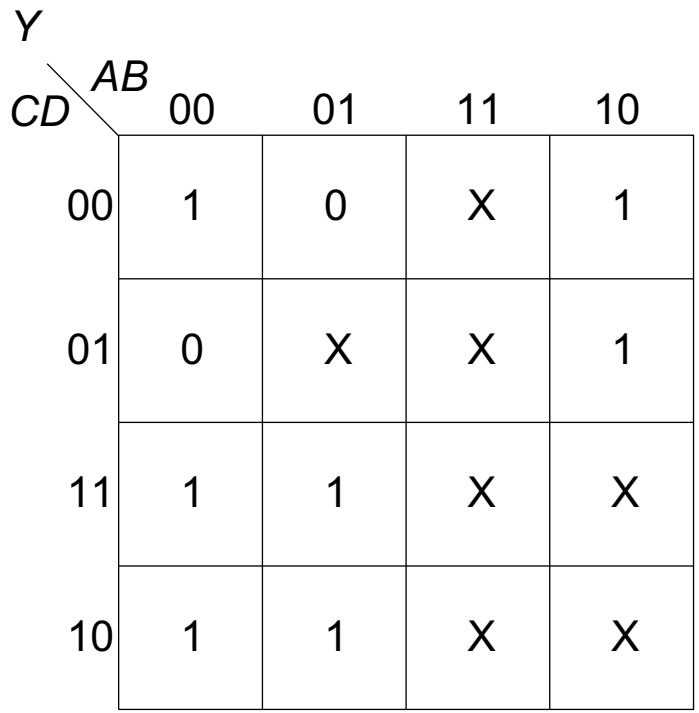
Karnaugh maps with Don’t Cares

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



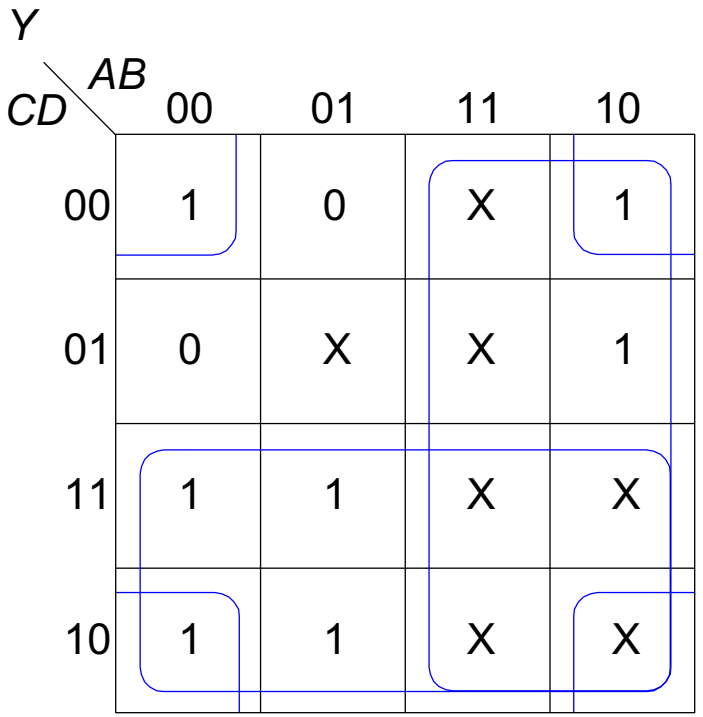
Karnaugh maps with Don't Cares

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



Karnaugh maps with Don't Cares

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



$Y = A + \overline{B}\overline{D} + C$

Is this useful?

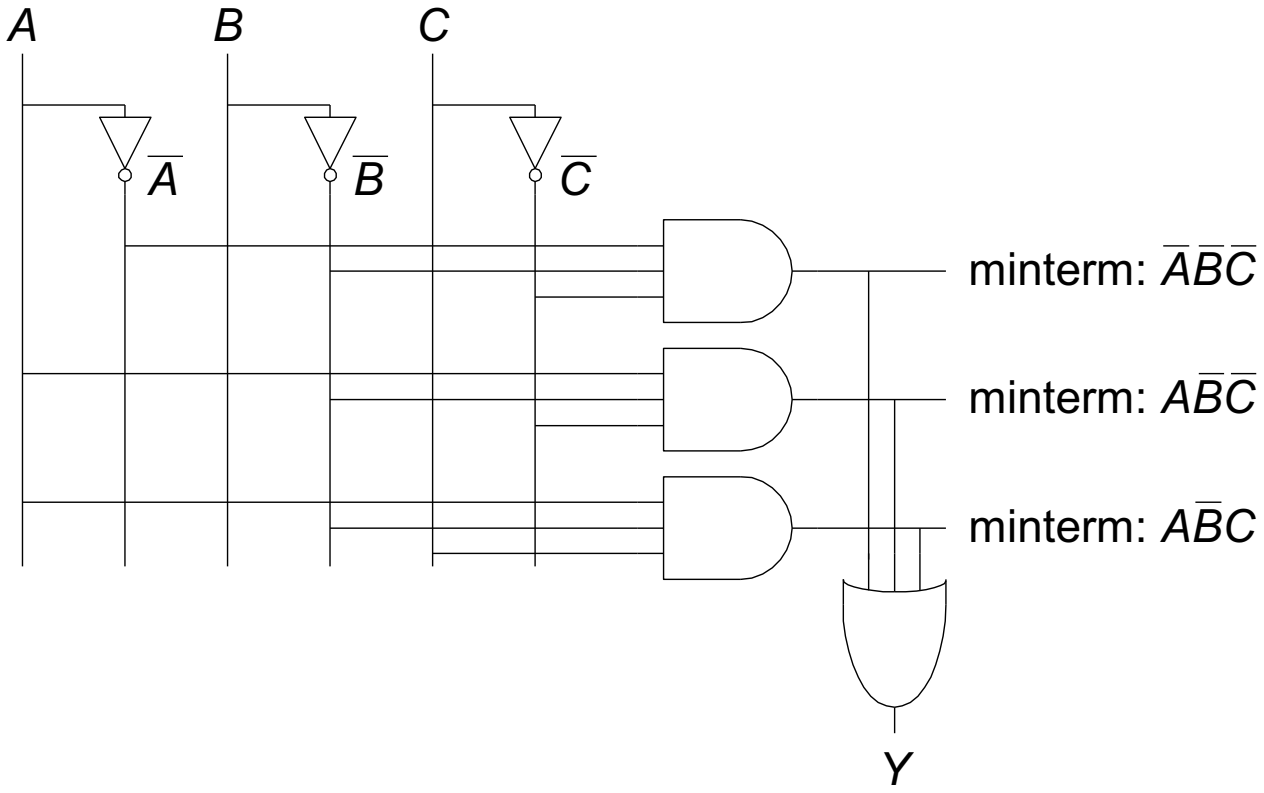
- In modern engineering practice, computer programs called *logic synthesizers* produce simplified circuits from a description of the logic function (later in chap 4)

- For large problems, logic synthesizers are much more efficient than humans. For small problems, a human with a bit of experience can find a good solution by inspection.

- *Yes:* you understand the basic concepts.

From Logic to Gates

- **SOP (sum-of-products) leads to two-level logic**
- **Example:** $Y = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot \overline{B} \cdot C)$



Circuit Schematics

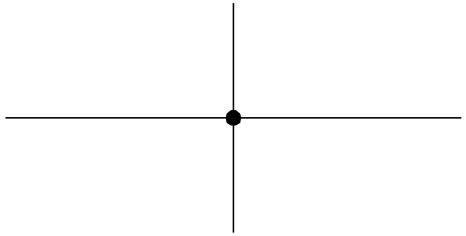
- Inputs: left (or top) side of a schematic
- Outputs: right (or bottom) side of a schematic
- Circuits should flow from left to right
- Straight wires are better than wires with multiple corners

Circuit Schematic (cont.)

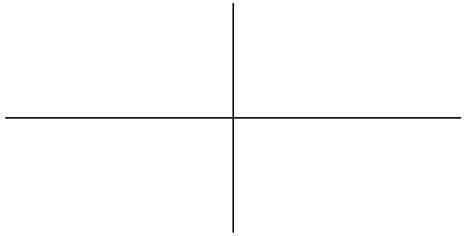
- **Wires always connect at a T junction**
- **A dot where wires cross indicates a connection between the wires**
- **Wires crossing *without* a dot make no connection**

wires connect at a T junction

wires connect at a dot

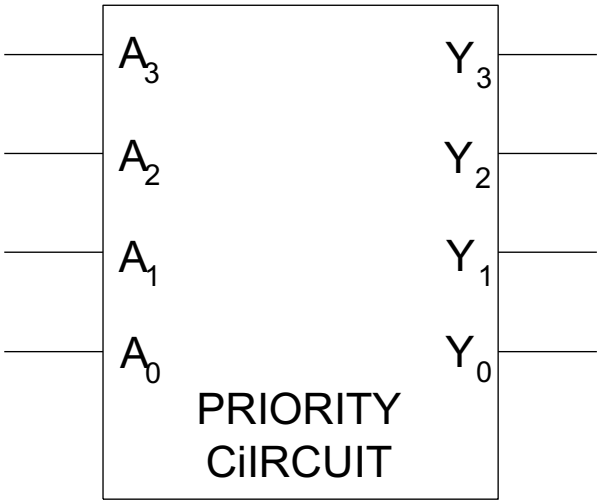


wires crossing without a dot do not connect



Multiple Output Circuits: Priority Circuit

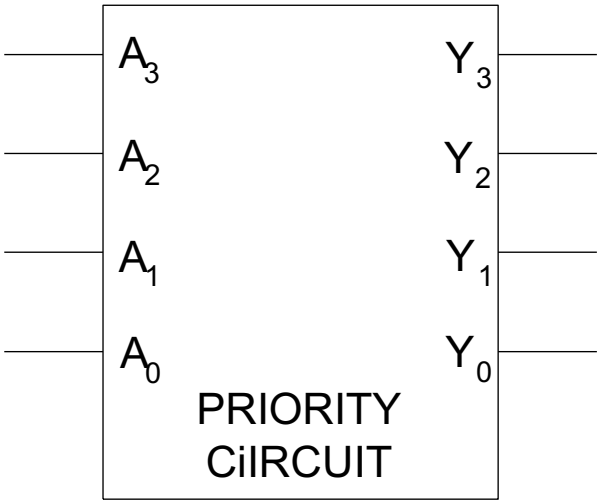
- **Output is the most significant TRUE input**



A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
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				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Multiple Output Circuits : Priority Circuit

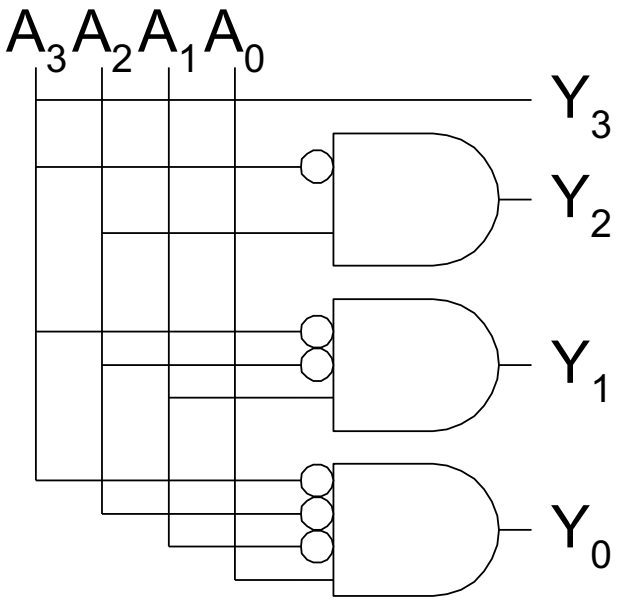
- **Output is the most significant TRUE input**



A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

Priority Encoder Hardware *"by inspection"*

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0



- **We could write sum-of-products form and reduce the equations. From the functional description :**
  - Y_3 is TRUE whenever A_3 is asserted, so $Y_3 = A_3$.
  - Y_2 is TRUE if A_2 is asserted and A_3 is not asserted, so $Y_3 = \overline{A}_3 A_2$

Compressing the Truth Table: Don't Cares

Truth Table

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

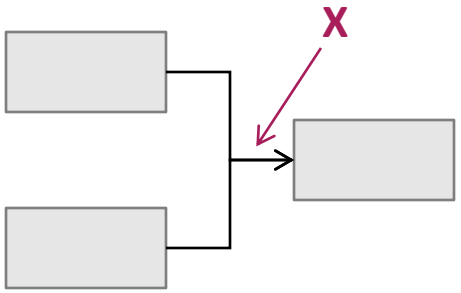
Compressed with don't cares

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

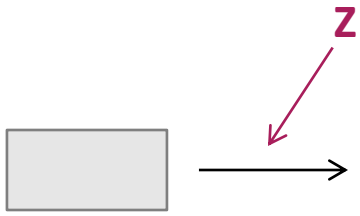
Additional logic levels: X and Z

- **So far we used only 1s and 0s for our circuits.**
  - For some cases we need several more, slightly strange, signals
  - These do not encode information, but represent different levels or cases where the output is neither 1 or 0

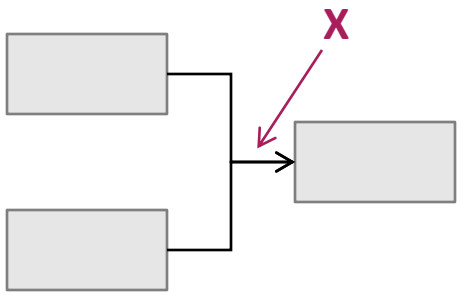
- **Contention: X**
  - When a signal is being driven to 1 and 0 simultaneously
  - Not a real level, could be any value (1, 0 or something in between)



- **High-impedance or tri-state: Z**
  - When an output is not driving to any specific value
  - Means the output is disconnected
  - Not a real level, some other output is able to determine the level

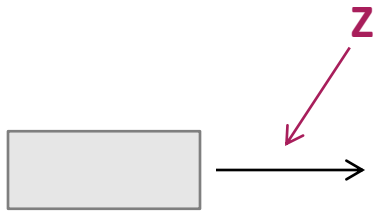


Contention: X

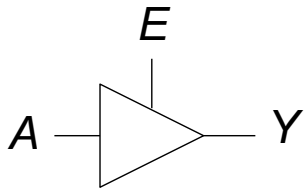


- **Usually a problem**
  - Two outputs drive one node to opposite values
  - Normally there should only be one driver for every connection.
- **Warning: “don’t care” and “contention” are both called X**
  - These are not the same
  - Verilog (we will see later) uses X for both,
    VHDL uses ‘-’ for don’t care, and ‘X’ for contention
  - Don’t care: degree of freedom that is fixed at implementation time
  - Contention: a bug really, undetermined behaviour

Floating: Z



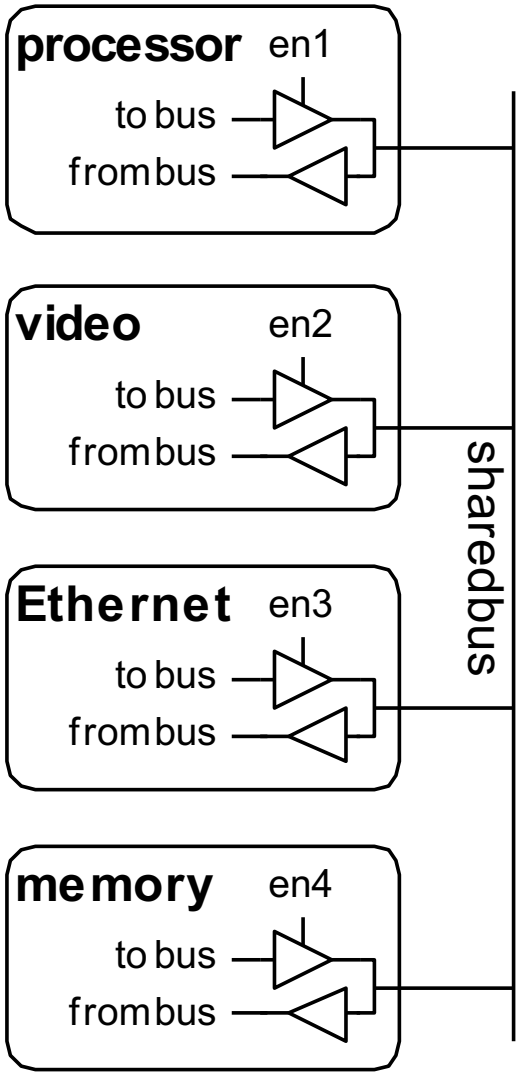
- Output is called: Floating, high impedance, tri-stated, high-Z
- Floating output might be 0, 1, or somewhere in between
- Example: tri-state buffer



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

Tristate Busses

- **Floating nodes are used in tri-state busses**
  - Many different drivers share one common connection
  - ***Exactly one driver is active*** at any time
  - All the other drivers are ***"disconnected"***
  - The disconnected drivers are said to be ***floating***, allowing exactly one node to drive.
  - More than one input can listen to the shared bus without problems

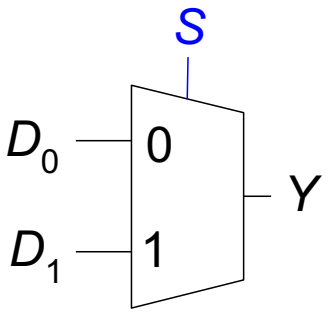


Combinational Building Blocks

- **Combinational logic is often grouped into larger building blocks to build more complex systems**
- **Hide the unnecessary gate-level details to emphasize the function of the building block**
- **We have already studied some building blocks**
  - full adders
  - priority circuits
- **We now look at:**
  - multiplexers
  - decoders

Multiplexer (Mux)

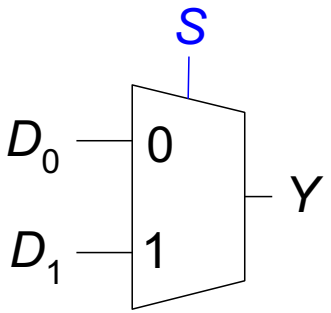
- **Selects between one of *N* inputs to connect to the output.**
- **Needs $\log_2 N$-bit control input**
- **2:1 Mux Example:**



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

Multiplexer (Mux)

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



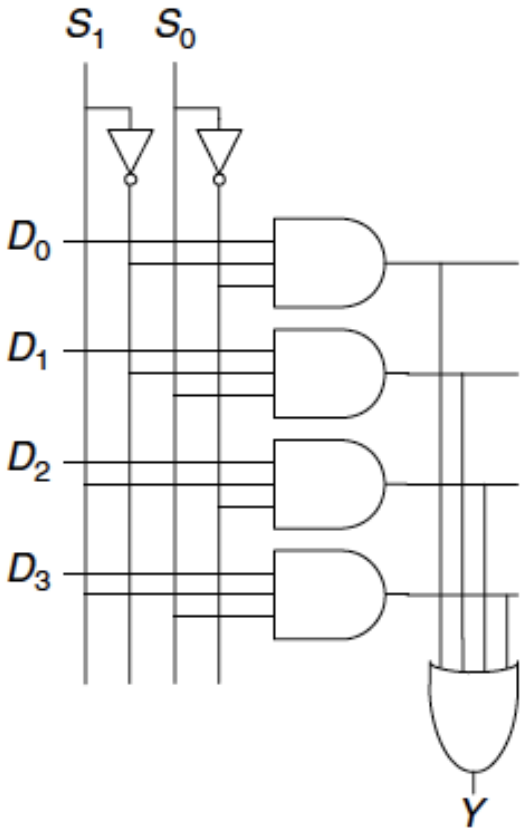
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

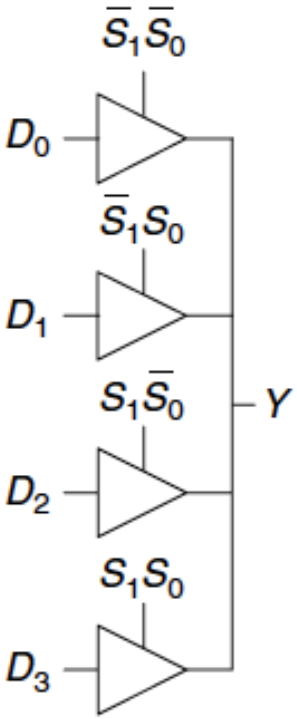
Compressed version

4:1 Multiplexer Implementations

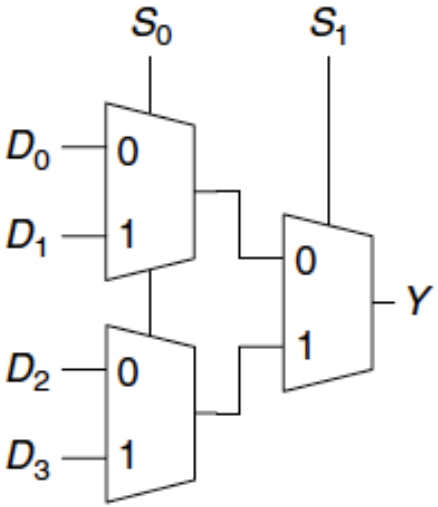
using two-level logic



using tristate buffers



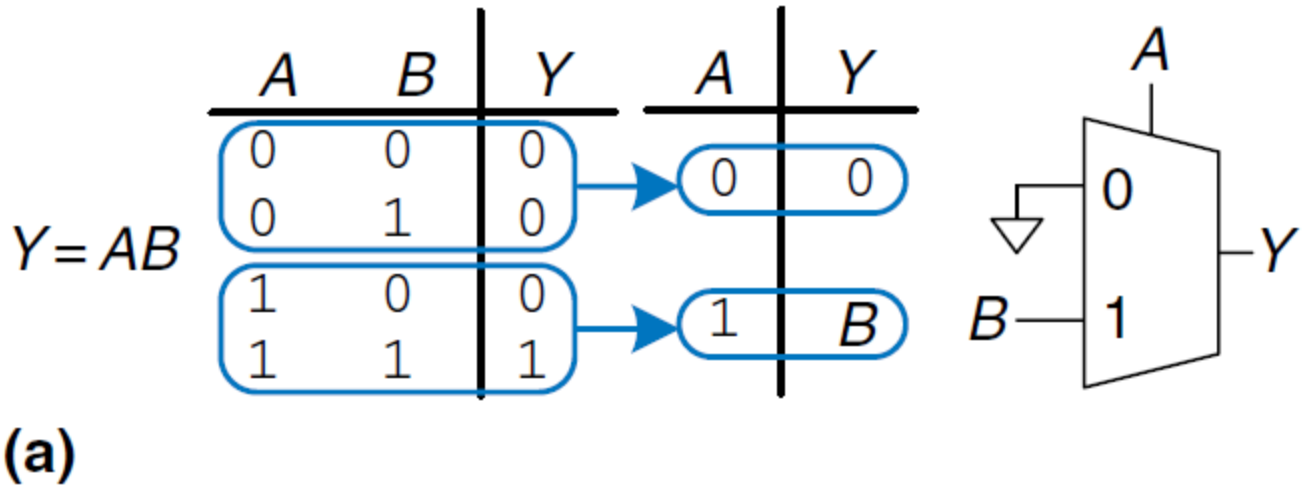
using tree of 2:1 muxes



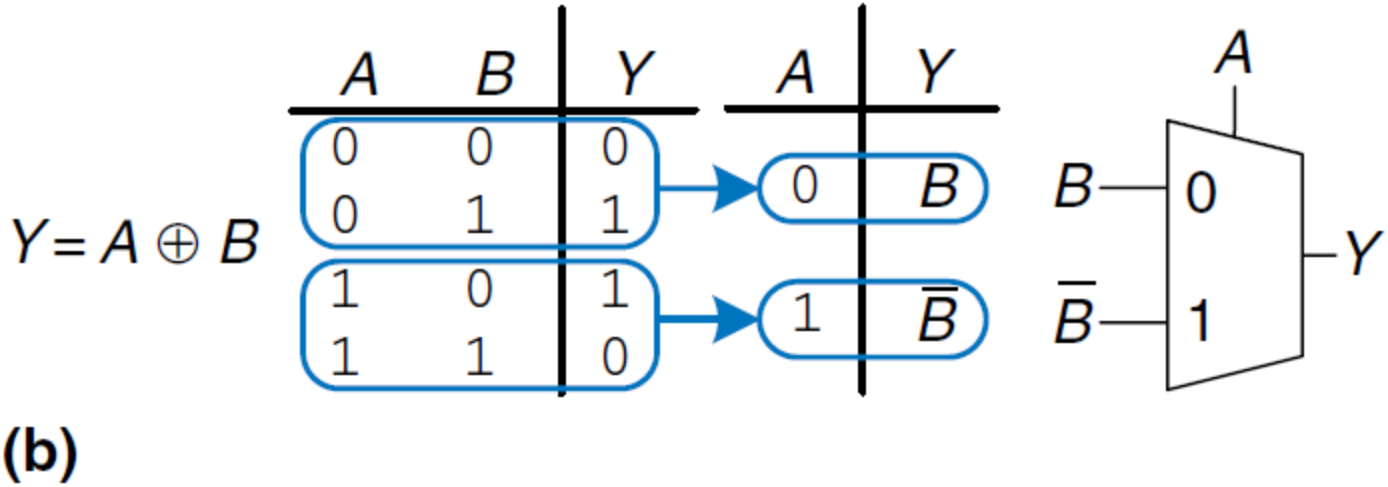
Logic Using Multiplexers

- **Implement Y = AB using a multiplexer; use A as a control**

Logic Using Multiplexers



(a)



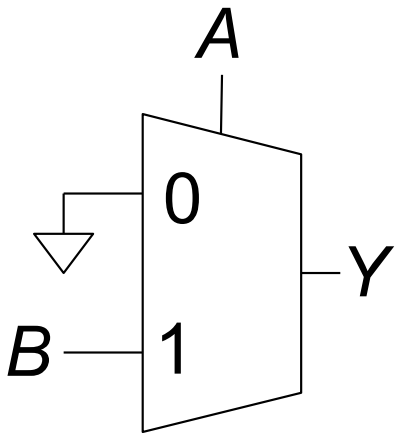
(b)

Logic Using Multiplexers

$Y = AB$

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

A	Y
0	0
1	B



- **We start with an ordinary truth table, and then combine pairs of rows to eliminate the rightmost input variable by expressing the output in terms of this variable.**

Logic using Multiplexers

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

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

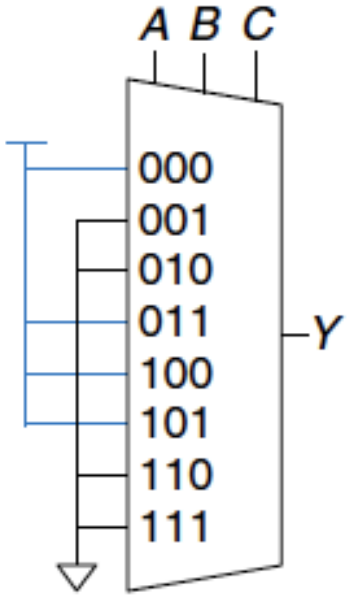
(a)

Logic using Multiplexers

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

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

(a)

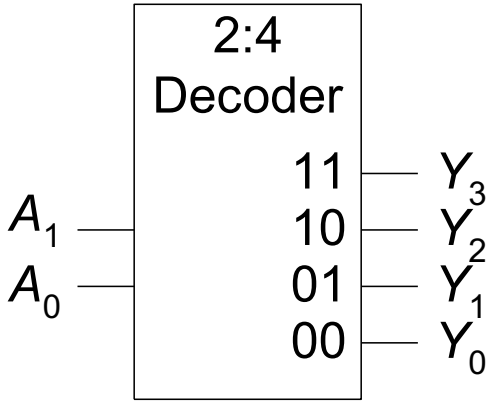


(b)

- In general, a 2^N-input multiplexer can be programmed to perform any N-input logic function by applying 0's and 1's to the appropriate data inputs: *it's a lookup table!*

Decoders

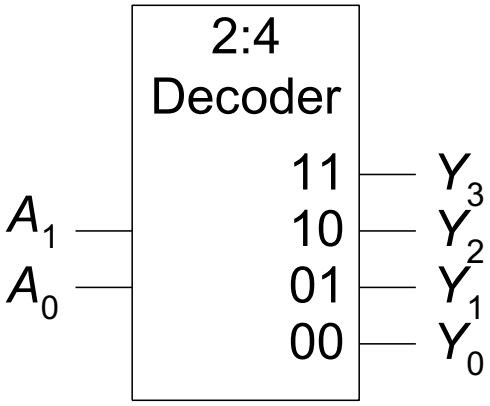
- **N inputs, 2^N outputs**
- **One-hot outputs: only one output HIGH at once**



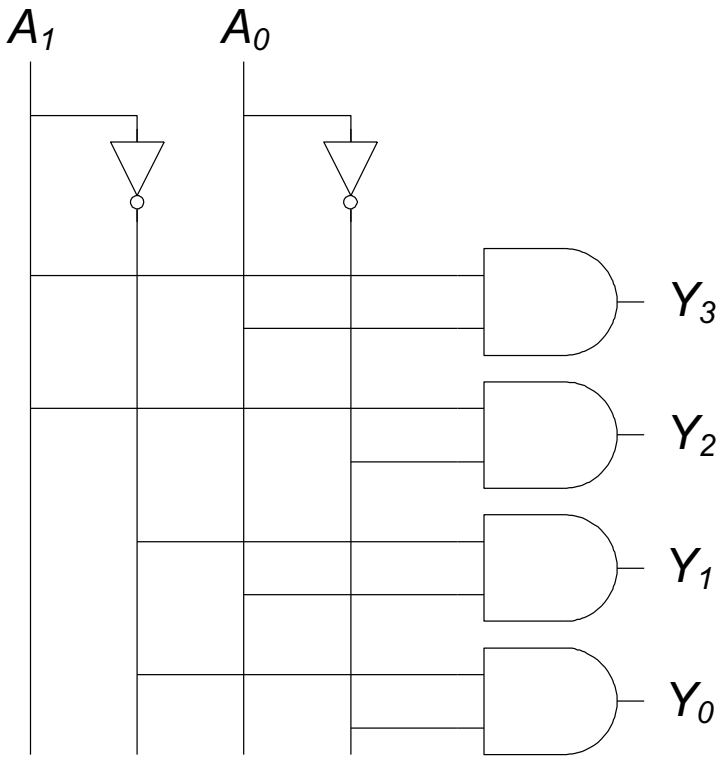
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

← ***These are exactly the minterms***

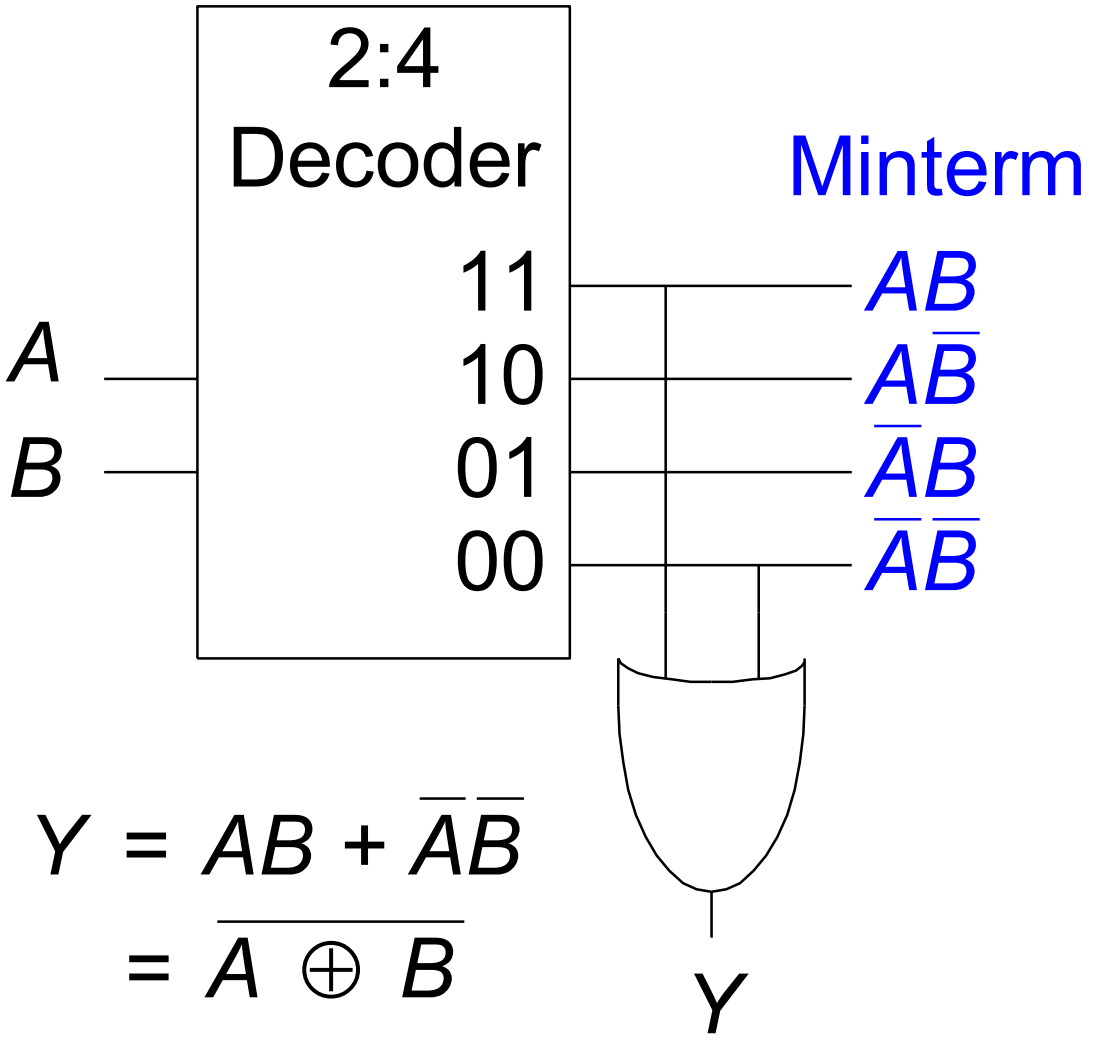
Decoder Implementation



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



Logic using Decoders

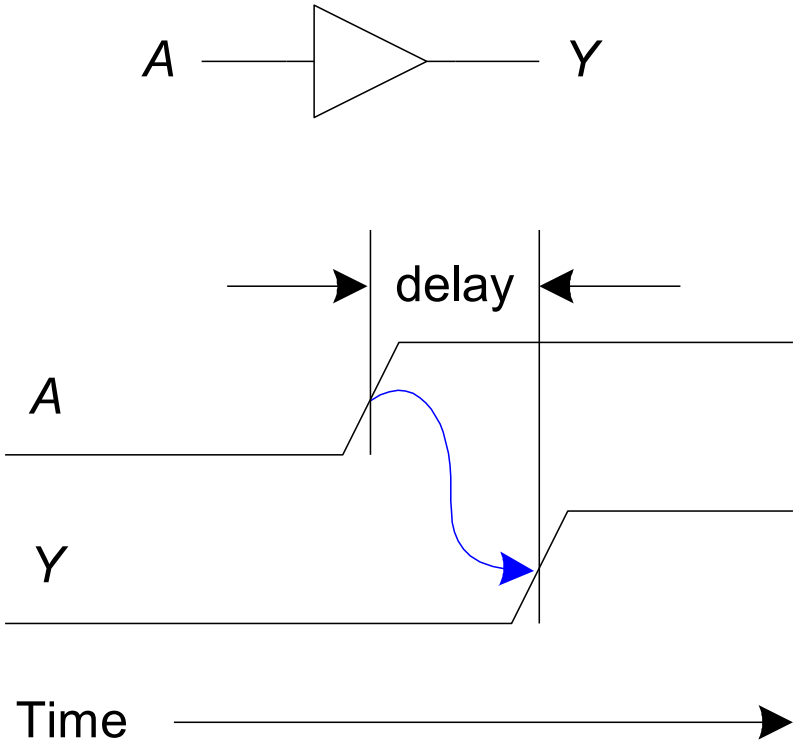


$$Y = AB + \bar{A}\bar{B}$$

$$= \overline{A \oplus B}$$

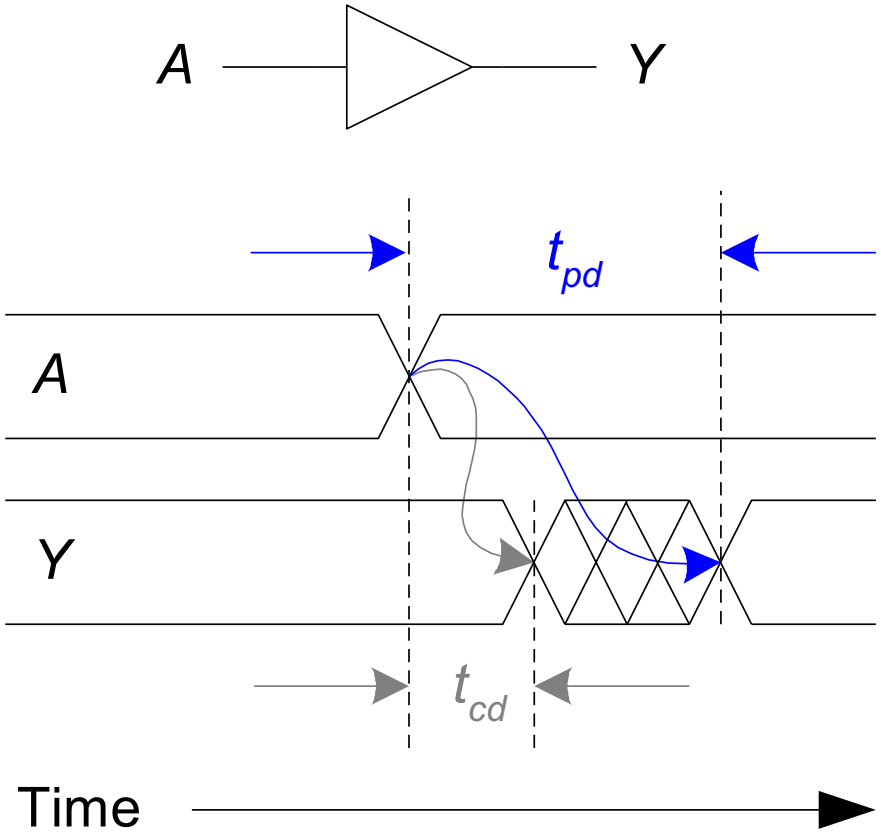
Timing

- **Until now, we investigated mainly functionality**
- **What determines how fast a circuit is and how can we make faster circuits?**



Propagation and Contamination Delay

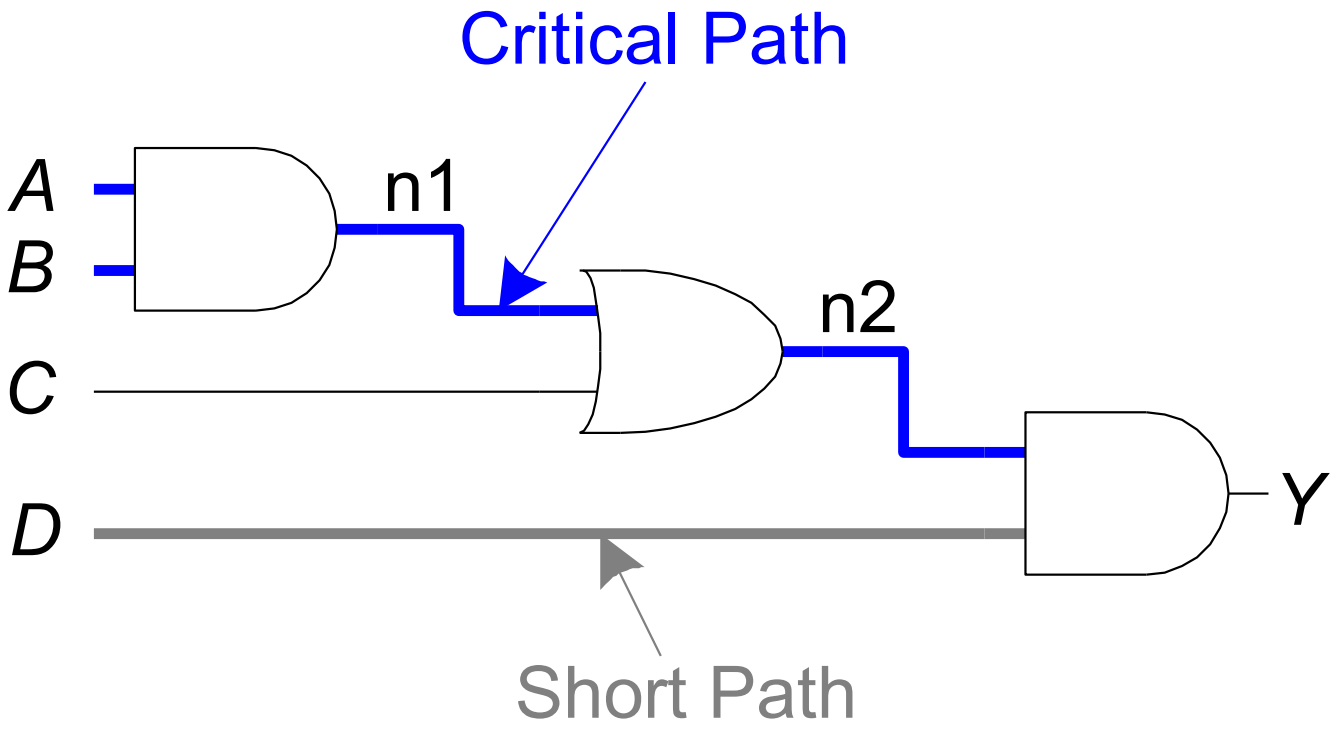
- **Propagation delay**: t_{pd} = max delay from input to output
- **Contamination delay**: t_{cd} = min delay from input to output



Propagation & Contamination Delay

- **Delay is caused by**
  - Capacitance and resistance in a circuit
  - Speed of light limitation (not as fast as you think!)
- **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) and Short Paths



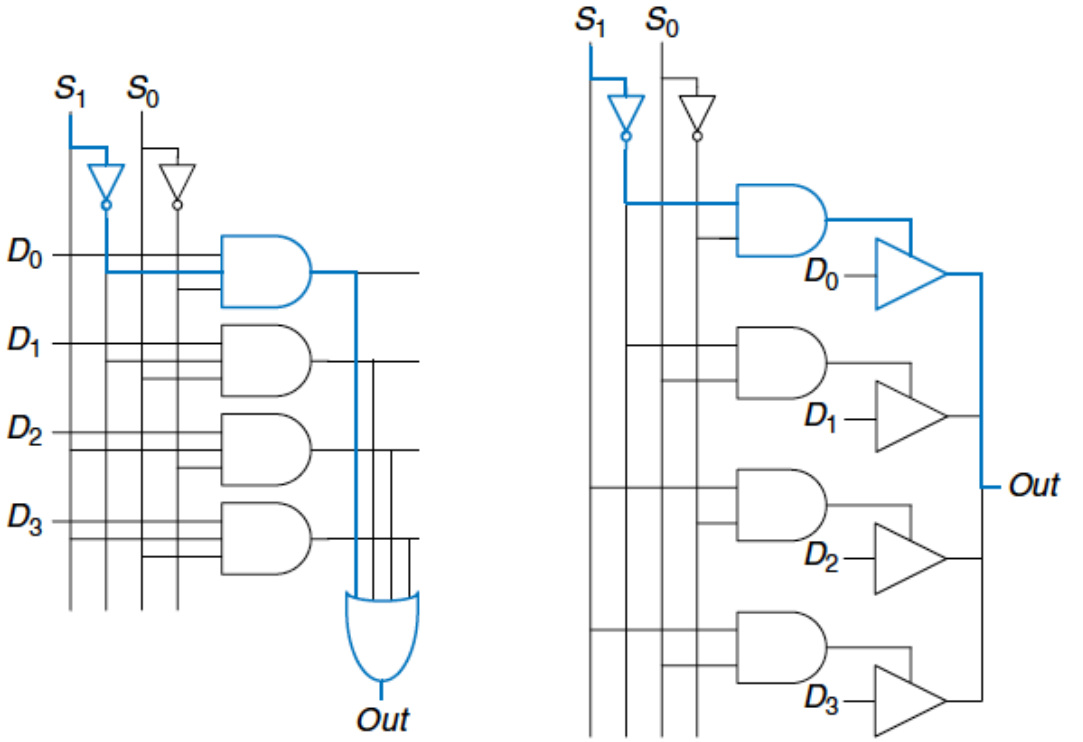
- **Critical (Long) Path:** $t_{pd} = 2\, t_{pd\_AND} + t_{pd\_OR}$
- **Short Path:** $t_{cd} = t_{cd\_AND}$

Propagation times

Table 2.7 Timing specifications for multiplexer circuit elements

Gate	t_{pd} (ps)
NOT	30
2-input AND	60
3-input AND	80
4-input OR	90
tristate (*A* to *Y*)	50
tristate (enable to *Y*)	35

Propagation times



(a)

$$t_{pd\_sy} = t_{pd\_INV} + t_{pd\_AND3} + t_{pd\_OR4}$$
$$= 30\text{ ps} + 80\text{ ps} + 90\text{ ps}$$
$$= \mathbf{200\text{ ps}}$$
$$t_{pd\_dy} = t_{pd\_AND3} + t_{pd\_OR4}$$
$$= \mathbf{170\text{ ps}}$$

(b)

$$t_{pd\_sy} = t_{pd\_INV} + t_{pd\_AND2} + t_{pd\_TRI\_SY}$$
$$= 30\text{ ps} + 60\text{ ps} + 35\text{ ps}$$
$$= \mathbf{125\text{ ps}}$$
$$t_{pd\_dy} = t_{pd\_TRI\_AY}$$
$$= \mathbf{50\text{ ps}}$$

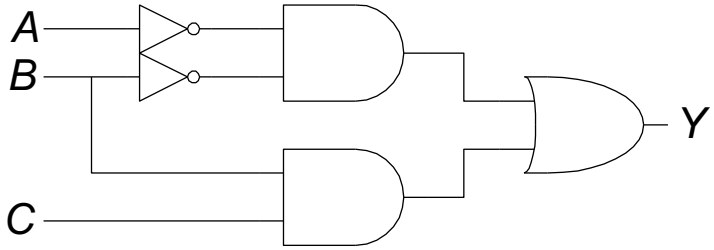
Figure 2.73 4:1 multiplexer propagation delays: (a) two-level logic, (b) tristate

Glitches

- **Glitch: when a single input change causes multiple output changes**
- **Glitches don't cause problems because of synchronous design conventions (which we'll talk about in a bit)**
- **But it's important to recognize a glitch when you see one in timing diagrams**

Glitch Example

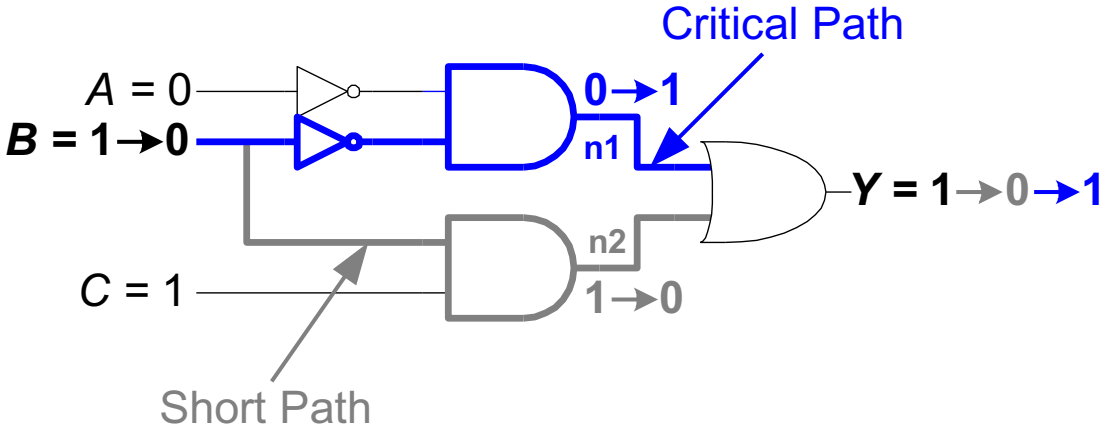
- **What happens when A = 0, C = 1, B falls (1->0) ?**

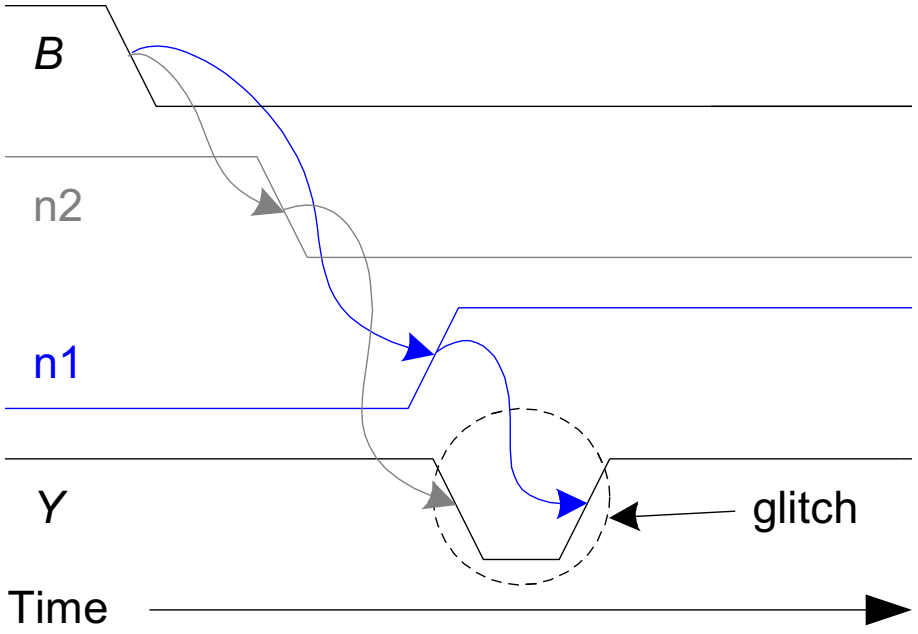


Y	AB = 00	01	11	10
C = 0	1	0	0	0
C = 1	1	1	1	0

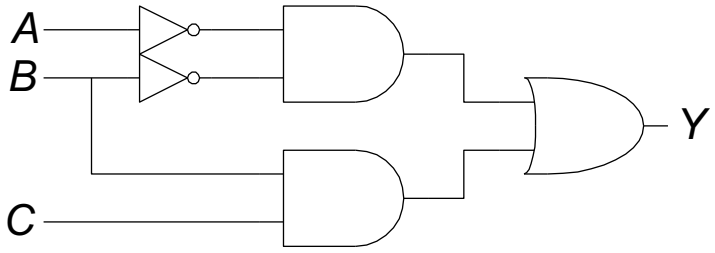
$$Y = \bar{A}\bar{B} + BC$$

Glitch Example (cont.)





Noticing a Glitch

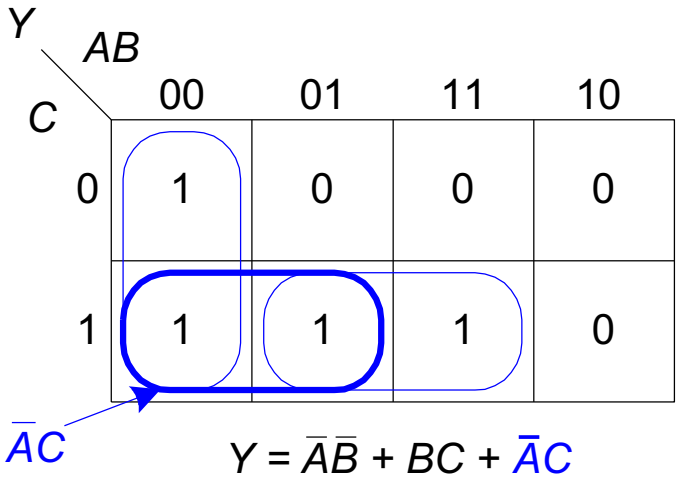


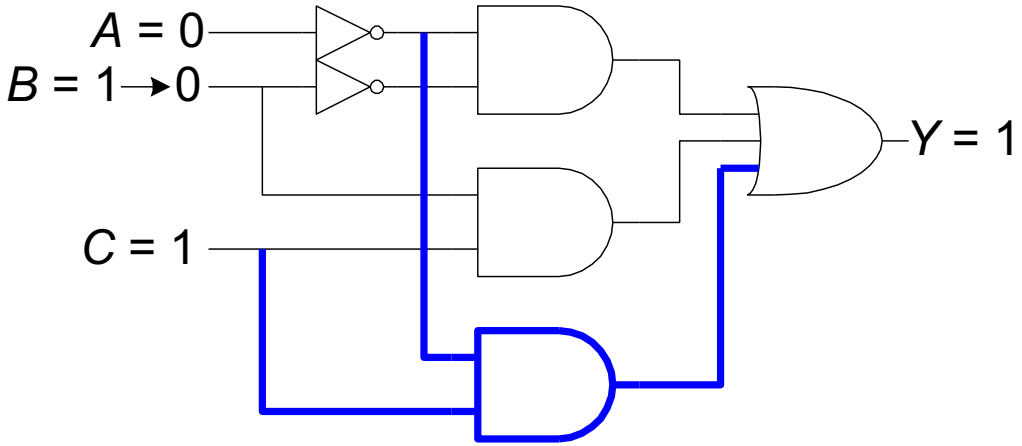
Y	AB 00	01	11	10
C = 0	1	0	0	0
C = 1	1 →	1	1	0

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

- **In general, a glitch can occur when a change in a single variable crosses the boundary between two prime implicants in a K-map.**
  - Transition on B (ABC=001 to ABC 011) moves from one prime implicant to the other.
  - We can eliminate the glitch by adding redundant implicants to the K-map to cover these boundaries.

Fixing the Glitch





Why Understand Glitches?

- Glitches don't cause problems because of synchronous design conventions (which we'll talk about later)
- But it's important to recognize a glitch when you see one in simulations or on an oscilloscope
- Can't get rid of all glitches – simultaneous transitions on multiple inputs can also cause glitches

What have we learned?

- **How to construct truth tables**
- **Converting truth tables into SOP and POS form**
- **Using Karnaugh maps to simplify Boolean Equations**
- **Timing of combinational circuits**
  - Propagation delay: longest time through the circuit
  - Contamination delay: shortest time in which the output reacts