

9/30/26



COMP 311: *Computer Organization!*

Lecture 11: Basic Circuit Design, Digital Tutorial



The University
of North Carolina
at Chapel Hill

<https://tinyurl.com/comp311-fa26>

Logistics Update



10/7

- Lab 1 is due tomorrow
- I'm pushing back quiz 3. The others are staying!
- WA2 released last Wednesday. Due 10/5

Plan for Class Today



- 3:35-4:00: Finish going between truth tables, logic equations
- 4:00-4:15: Strategies for circuit design
- 4:15-4:20: Break
- 4:20-4:50: Digital Tutorial!

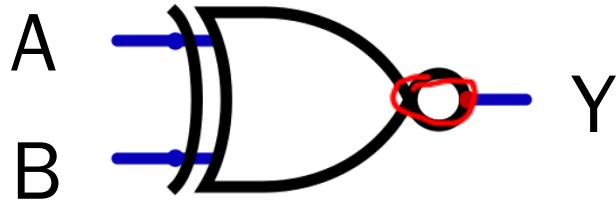
Guiding Question ☺

How can we start building circuits with a digital simulator?

Goal: Going between these 3 formats!



Symbol



Truth Table

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

Equation

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





Forming *Truth Tables* from *Equations*

Ex1: Forming Truth Tables from Equations



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

A	B	Y
0	0	
0	1	
1	0	
1	1	

Ex1: Forming Truth Tables from Equations



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

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

Ex2: Forming Truth Tables from Equations



$$Y = AB + \bar{C}$$

A	B	C	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	

Ex2: Forming Truth Tables from Equations



$$Y = AB + \bar{C}$$

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

Forming Truth Tables from Equations



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

A	B	C	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	



Forming *Equations* from *Truth Tables*

Sum of Products Form



When two or more products (AND) are summed (OR) together

Sum of Products

$$Y = AB + AC$$

Not Sum of Products

$$Y = (A + B)(C + A)$$

Approach: write down a boolean expression for every '1' in the output



Ex: Forming Equations from Truth Tables

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



Ex: Forming Equations from Truth Tables

	A	B	C	Y	
	0	0	0	1	$\bar{A}\bar{B}\bar{C}$
	0	0	1	0	
	0	1	0	1	$\bar{A}B\bar{C}$
	0	1	1	0	
	1	0	0	0	
	1	0	1	1	$A\bar{B}C$
	1	1	0	0	
	1	1	1	0	

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

Forming Equations from Truth Tables



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

A Systematic Design Approach



Truth Table

C	B	A	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

-it's systematic!
-it works!
-it's easy!
-we are done! (?)



- 1) Write the functional spec as a truth table
- 2) Write down a Boolean expression for every '1' in the output

$$Y = (!C \&\& !B \&\& A) \mid\mid (!C \&\& B \&\& A) \mid\mid (C \&\& B \&\& !A) \mid\mid (C \&\& B \&\& A)$$

- 3) Wire up the ideal gates, replace them with equivalent realizable gates, call it a day, and go home!

This approach will **always** give us logic expressions in a particular form:

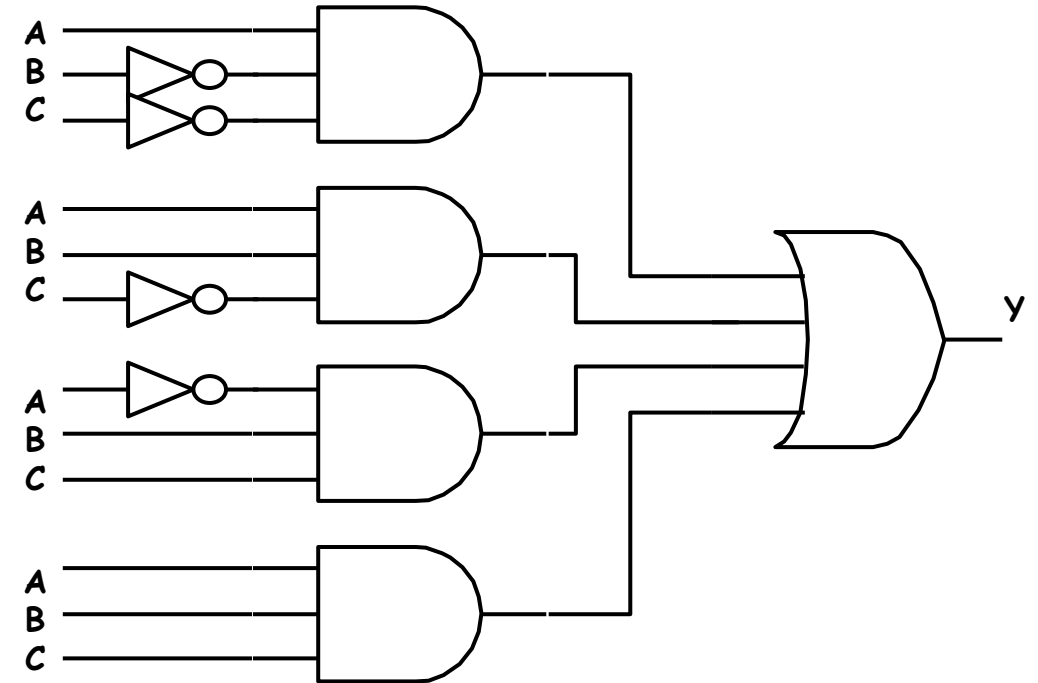
SUM-OF-PRODUCTS

Straightforward Synthesis



We can implement sum-of-products with just 3 levels of logic, using...

INVERTERS/AND/OR!





Digital Tutorial!

Setup Checkpoint



- Open Digital successfully
- If tutorial pops up, follow it!
- If not, View → Start tutorial
- Turn on Component Tree View
- Mac users: two-finger tap = right-click
- Starter code on Canvas

Before moving on:

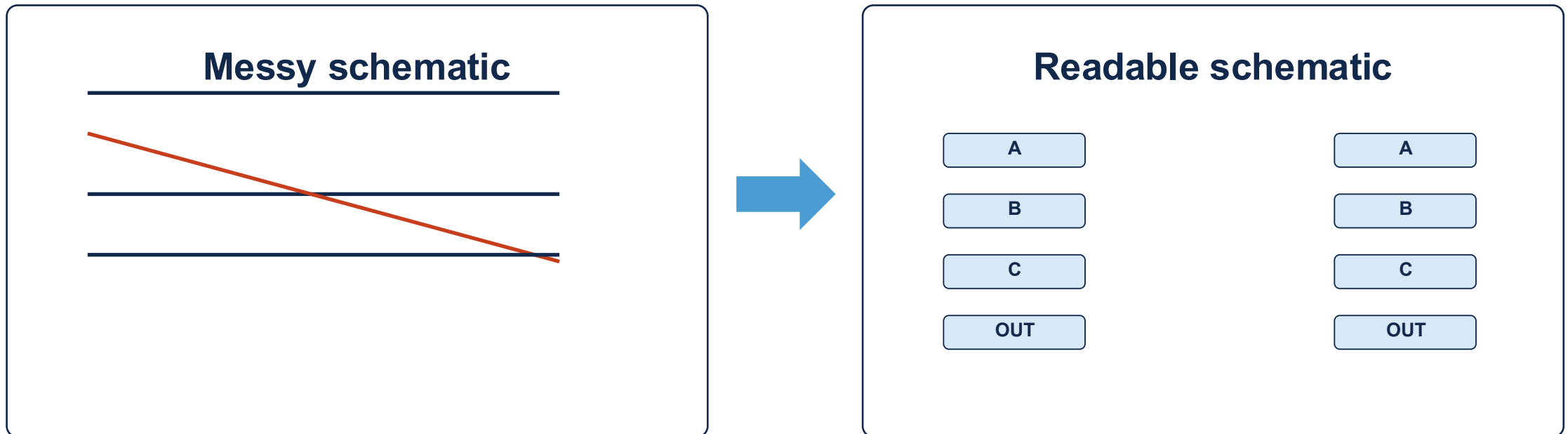
- Can you place an input?
- Can you place an output?
- Can you connect a wire?
- Can you start simulation?

Raise your hand if you need help!!

Exercise 1: Make Wires Readable with Tunnels



- Tunnels: Invisible Wires w/ Matching Labels
- Same label = same signal, labels are case sensitive
- Red wires mean an invalid connection

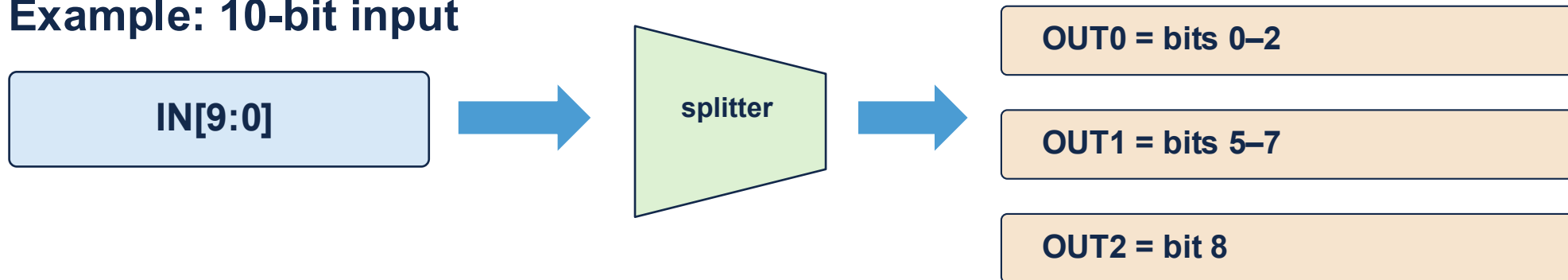




Exercise 2: Split Multi-Bit Values

- Splitters/Mergers let us name and use a piece of a value
- Habit: set component bit widths first... if widths do not match, simulation will complain or show errors!

Example: 10-bit input



Try doing exercise 2-0 to start!
2-1 is a challenge problem...



Testing is part of our workflow

- Don't wait until the end to run tests!
- When a test fails, ask:
 - Are my pin names correct?
 - Are my bit widths correct?
 - Do the outputs match my truth table?

1

Build a small piece

Place components and wire them intentionally. Use tunnels to avoid wiring spaghetti.

2

Simulate it

Use the play button. Try a few input values manually before running the full test fixture.

3

Run tests

Simulation → Run Tests. Passing tests should show expected green outputs.

Exercise 3

A_1	A_0	B_1	B_0	Y
0	0	0	0	
0	0	1	0	
0	0	0	1	
0	0	1	1	
1	0	0	0	
1	0	1	0	
1	0	0	1	
1	0	1	1	
0	1	0	0	
0	1	1	0	
0	1	0	1	
0	1	1	1	
1	1	0	0	
1	1	1	0	
1	1	0	1	
1	1	1	1	

I want to build a circuit that will output 1 if $A > B$, and 0 if $A \leq B$. A and B are both two-bit unsigned numbers.

Additionally, define A_1 and B_1 as the left bits of A and B, and A_0 and B_0 as the right bits.

Exercise 3

A_1	A_0	B_1	B_0	Y
0	0	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	1	1	0
1	0	0	0	1
1	0	1	0	0
1	0	0	1	1
1	0	1	1	0
0	1	0	0	1
0	1	1	0	0
0	1	0	1	0
0	1	1	1	0
1	1	0	0	1
1	1	1	0	1
1	1	0	1	1
1	1	1	1	0

I want to build a circuit that will output 1 if $A > B$, and 0 if $A \leq B$. A and B are both two-bit unsigned numbers.

Additionally, define A_1 and B_1 as the left bits of A and B, and A_0 and B_0 as the right bits.

Let's Try it Out

A_1	A_0	B_1	B_0	Y
0	0	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	1	1	0
1	0	0	0	1
1	0	1	0	0
1	0	0	1	1
1	0	1	1	0
0	1	0	0	1
0	1	1	0	0
0	1	0	1	0
0	1	1	1	0
1	1	0	0	1
1	1	1	0	1
1	1	0	1	1
1	1	1	1	0

I want to build a circuit that will output 1 if $A > B$, and 0 if $A \leq B$. A and B are both two-bit unsigned numbers.

Additionally, define A_1 and B_1 as the left bits of A and B, and A_0 and B_0 as the right bits.

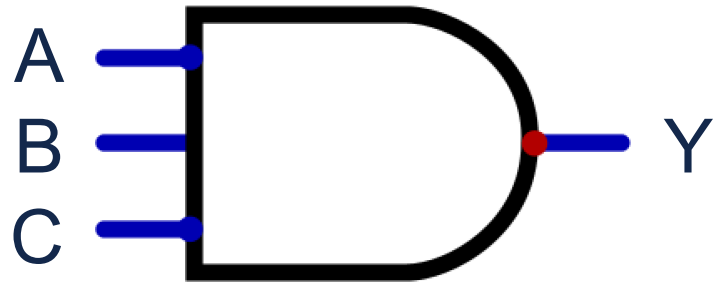
$$\bar{A}_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 B_0 + A_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 \bar{B}_0$$

Is the simplest possible form?? 🤔

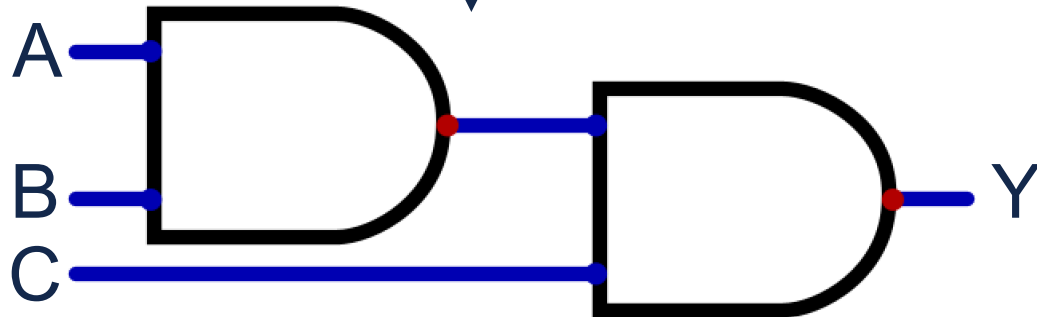


Logic with 2+ Inputs

3-input AND

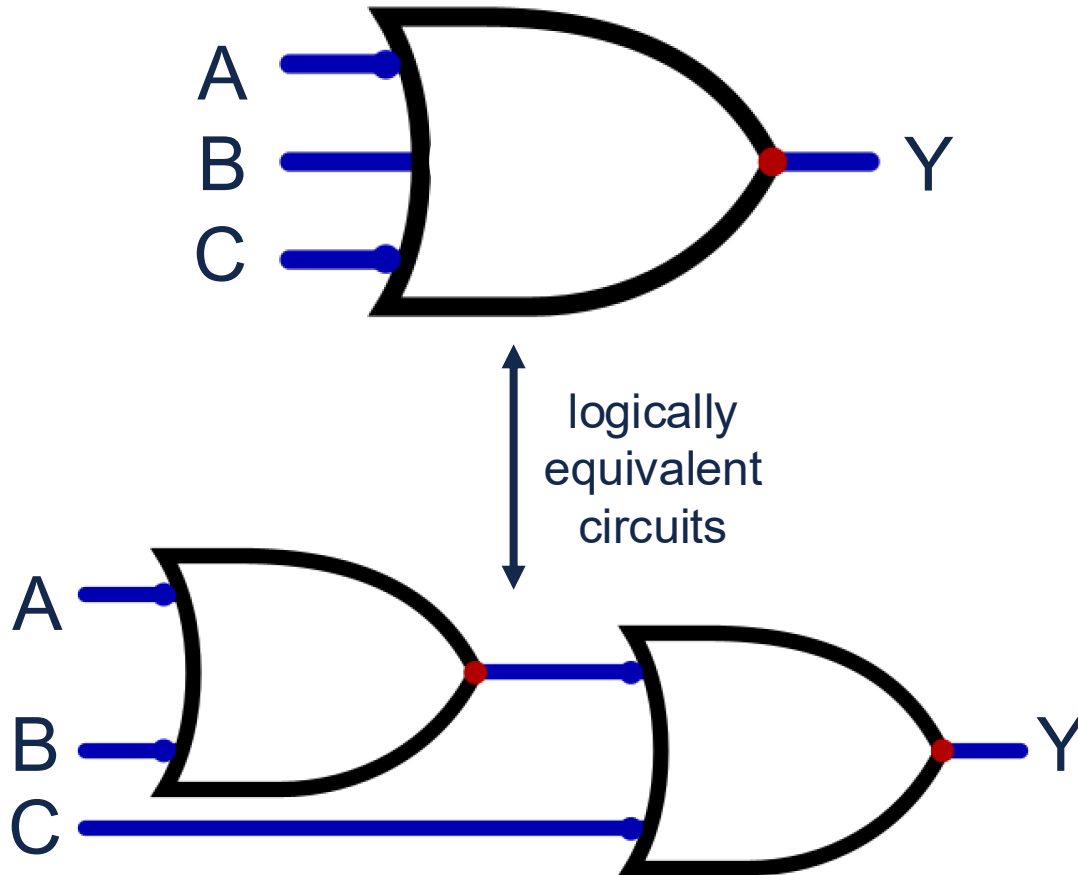


logically
equivalent
circuits



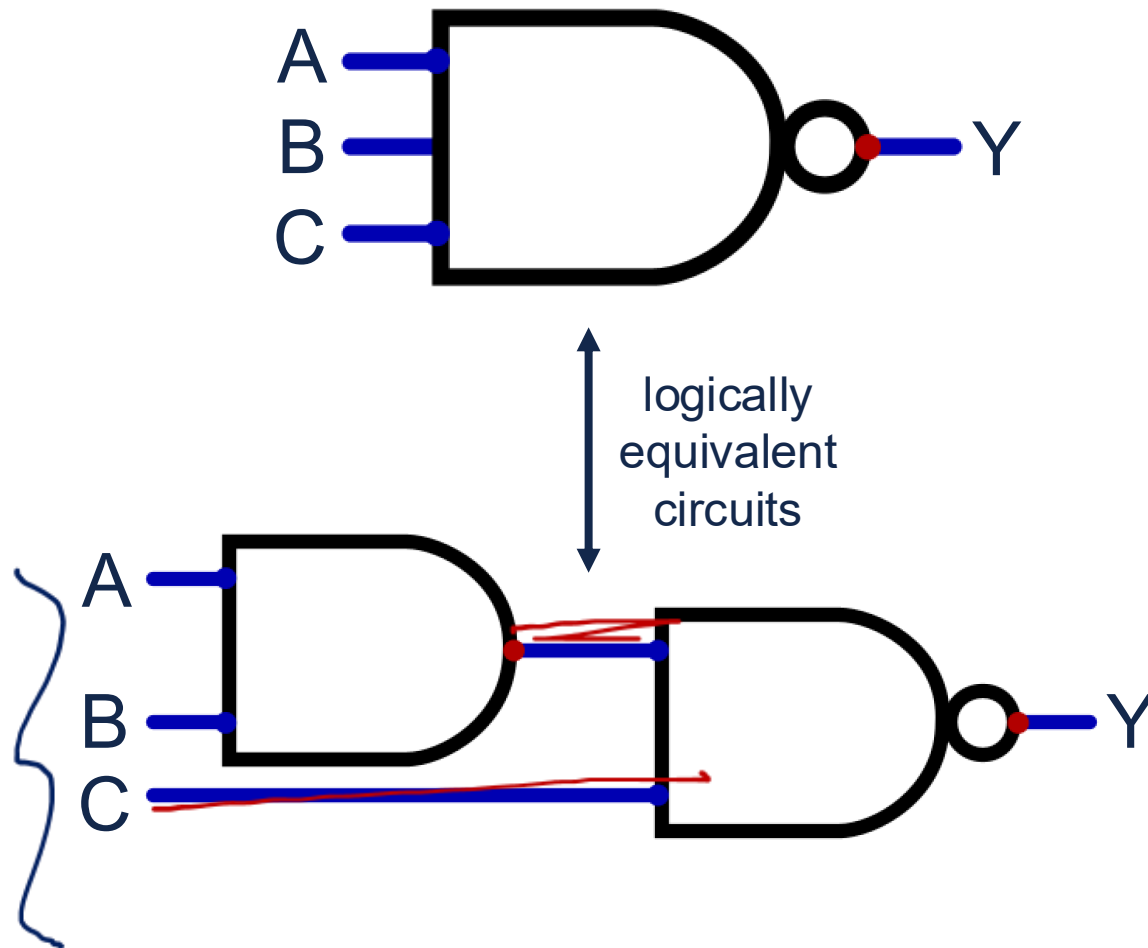
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

3-input OR



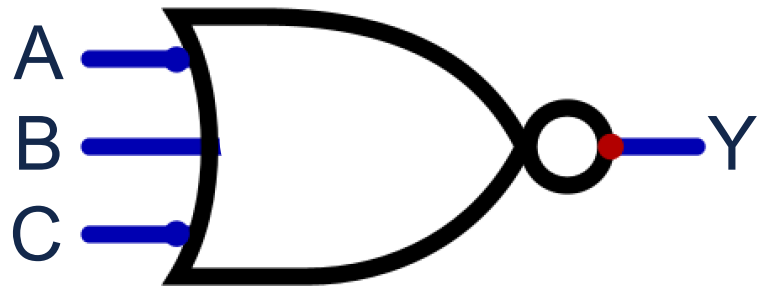
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

3-input NAND

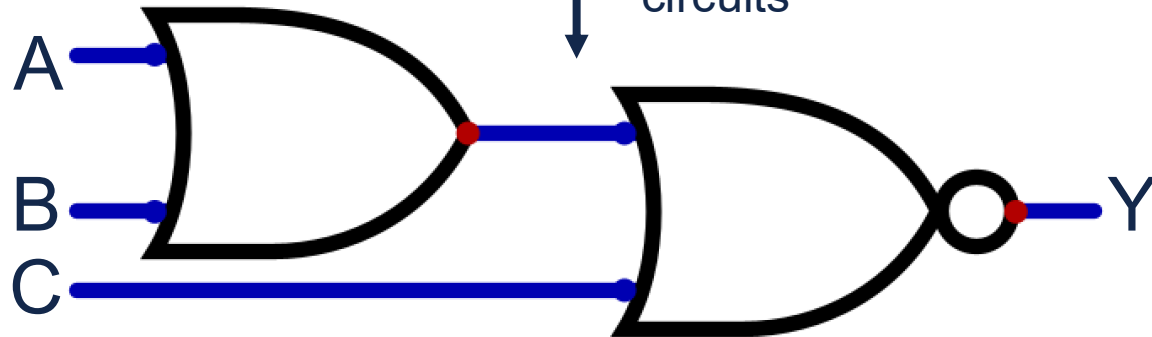


A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

3-input NOR

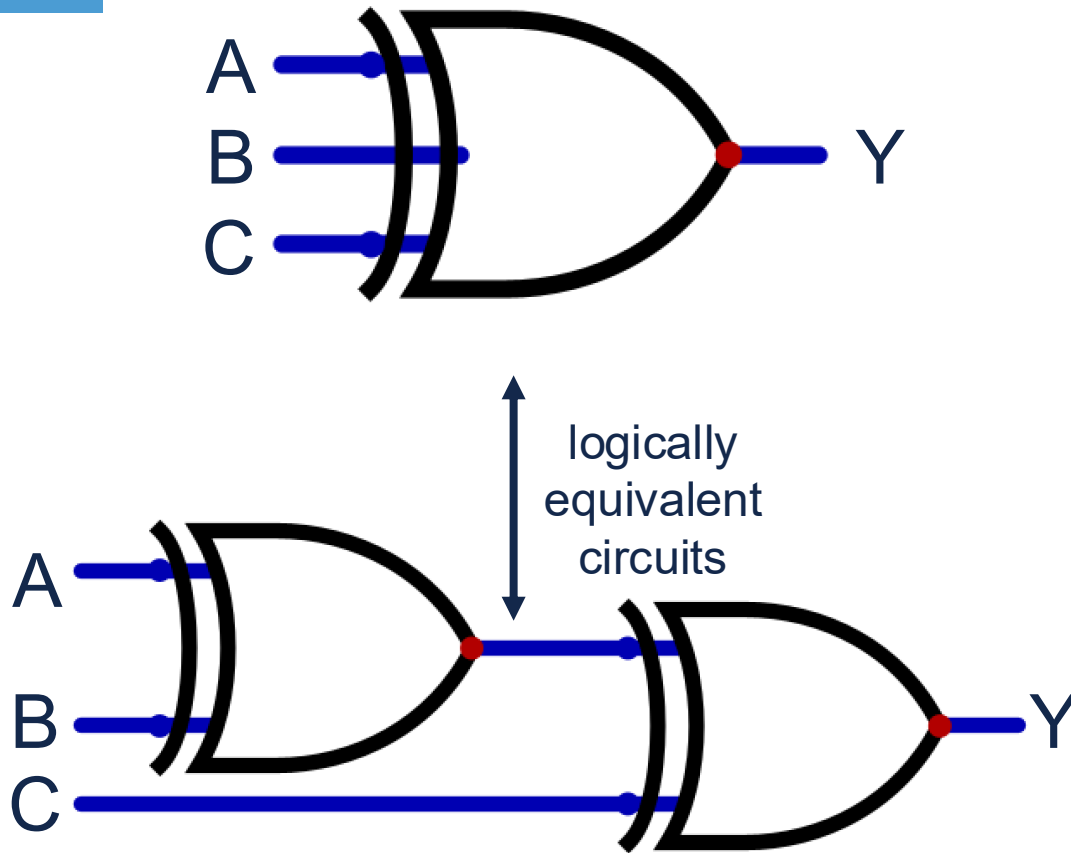


logically
equivalent
circuits



A	B	C	Y
0	0	0	1
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	0

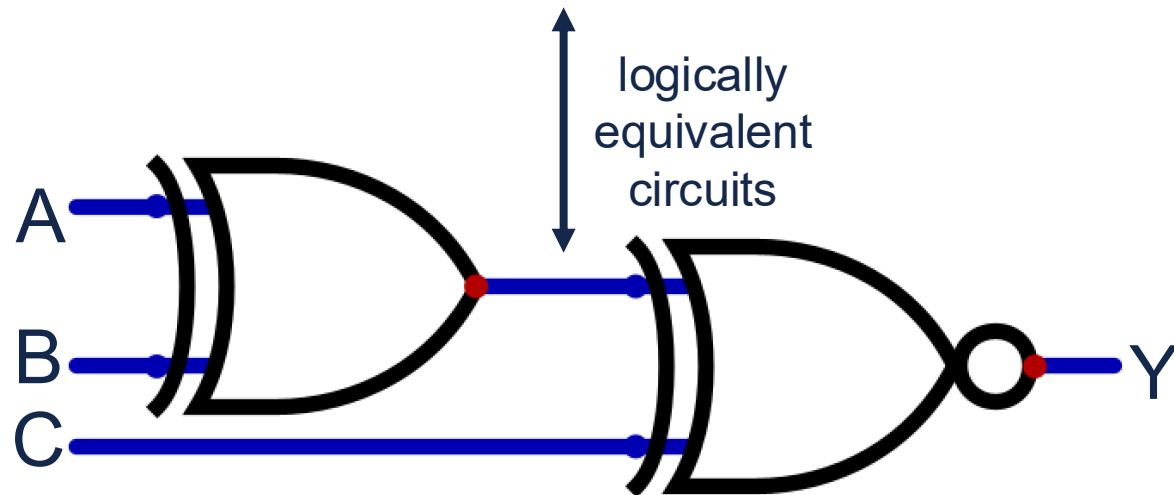
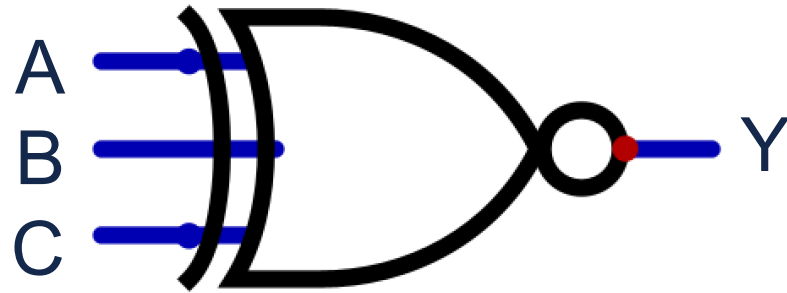
3-input XOR



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

For any number of inputs, the output is 1 when an odd number of inputs is 1

3-input XNOR

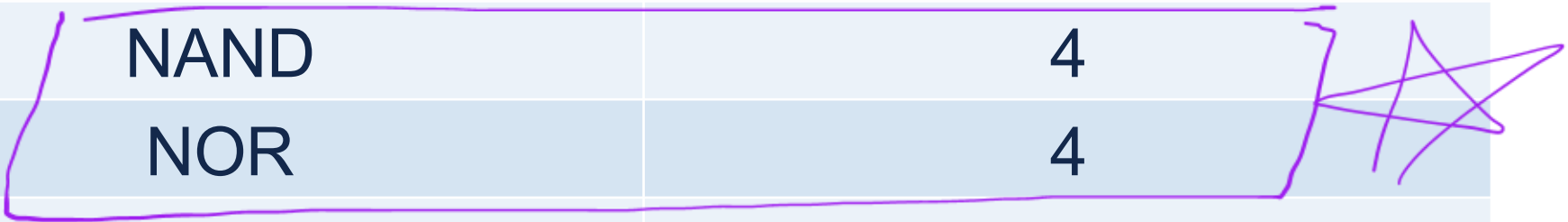


A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Recall: Transistor Count



Gate	Number of Transistors
NOT	2
AND	6
OR	6
NAND	4
NOR	4
XOR	12
XNOR	12



Transistor Count



Gate	# of Transistors
NOT	2
n-input AND	$2n + 2$
n-input OR	$2n + 2$
n-input NAND	$2n$
n-input NOR	$2n$
2-input XOR	12
3-input XOR	28
2-input XNOR	12
3-input XNOR	28

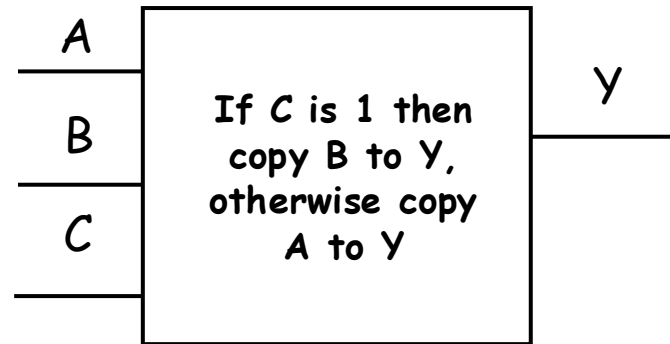


Multiplexers

An interesting 3-input gate....!



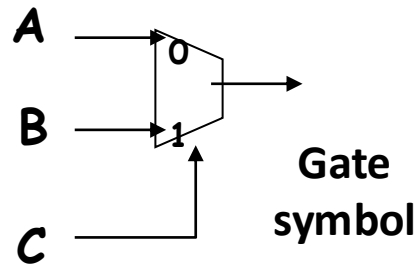
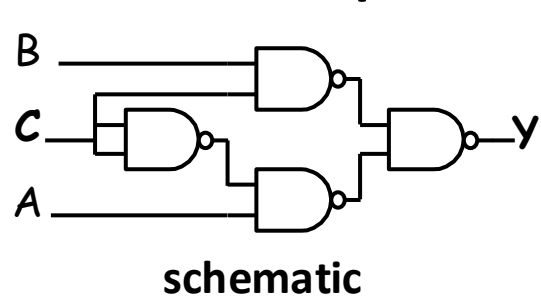
Based on C, select the A or B input to be copied to Y.



Truth Table

C	B	A	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

2-input Multiplexer



A Circuit for If-Statements! 😊



^C
if (S == 0)
 Y = A
elseif (S == 1)
 Y = B

In Assembly...



```
beq  x10, x0, S_is_zero          # if S == 0, jump
addi x14, x0, 1
beq  x10, x14, S_is_one          # if S == 1, jump

S_is_zero:
    add  x13, x11, x0             # Y = A

S_is_one:
    add  x13, x12, x0             # Y = B
```

Multiplexers



Truth Table

S	A	B	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	

Equation

Diagram

Multiplexers



Truth Table

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

Equation

Diagram

Multiplexers



Truth Table

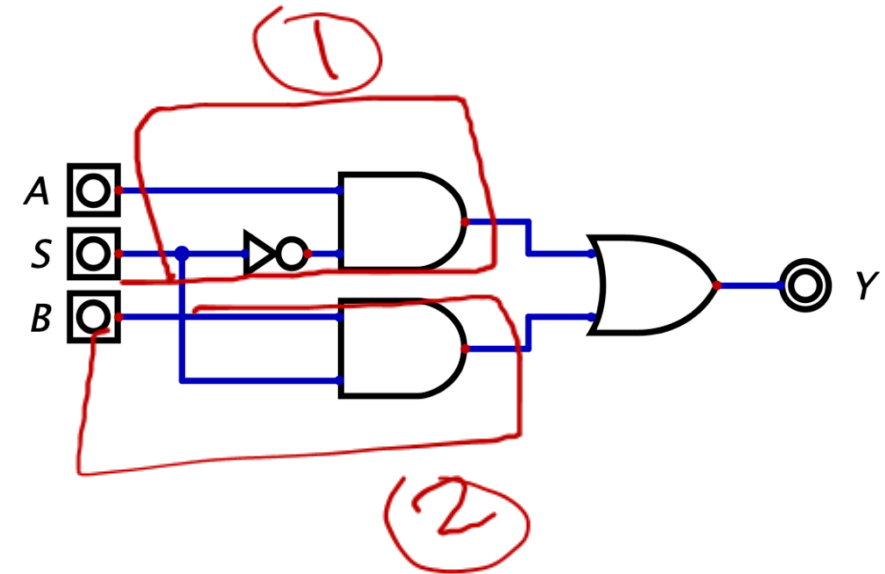
S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

S	A	B	Y
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

Equation

$$Y = \overset{(1)}{A}\bar{\overset{(2)}{S}} + \bar{A}\overset{(2)}{S}$$

Diagram



Multiplexers



Truth Table

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

S	A	B	Y
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

Equation

Diagram

Don't-cares can be used to simplify logic!

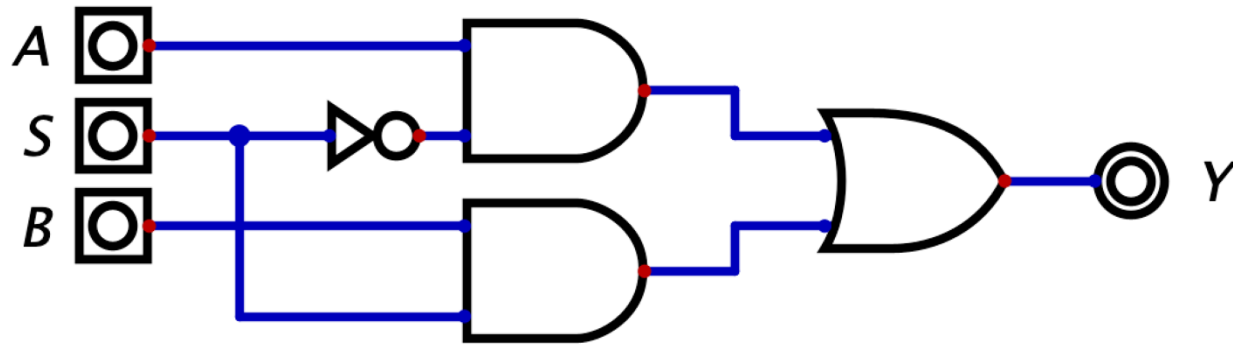
For input combinations that never occur, the designer may assume either 1 or 0, whichever is more useful to achieving a minimum-cost implementation.

Y

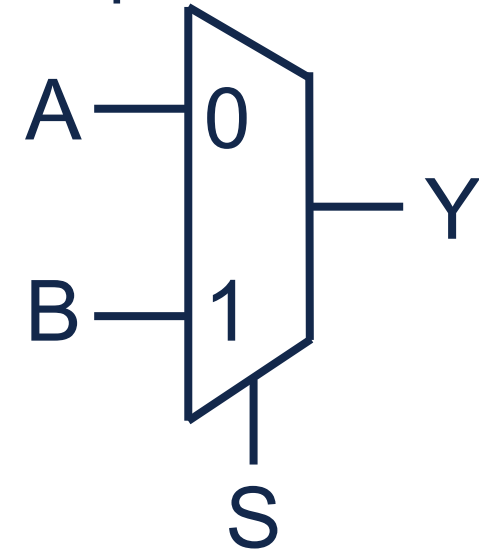
2:1 Multiplexer (2:1 mux)



- A circuit that chooses an output from among two inputs based on the value of a select signal

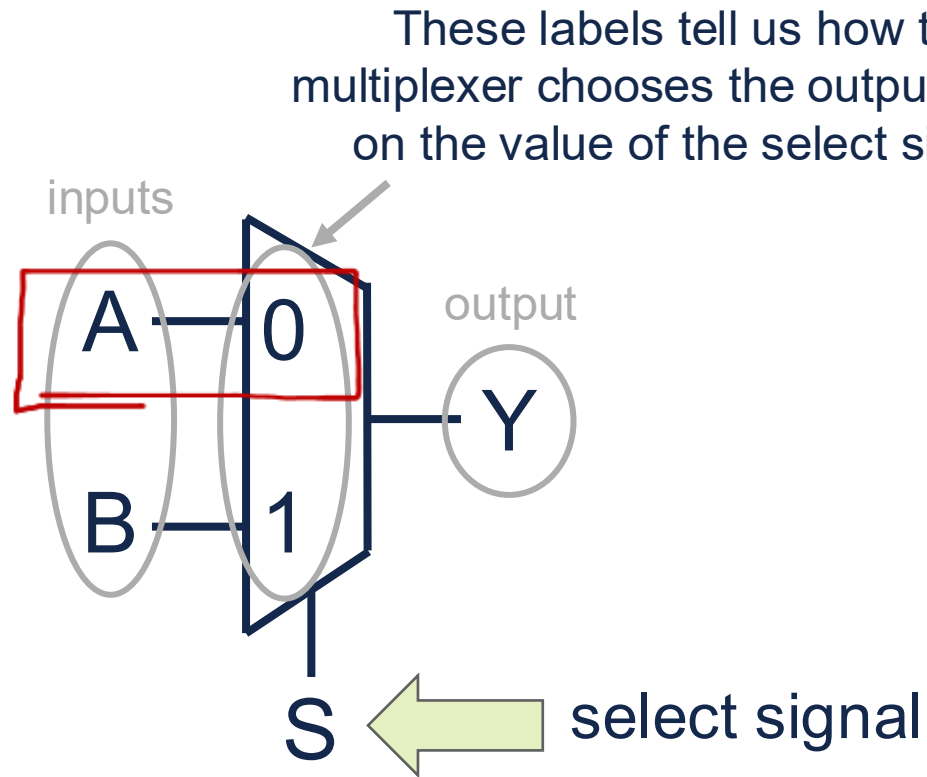


2:1 Multiplexer Schematic
(Gate Diagram)



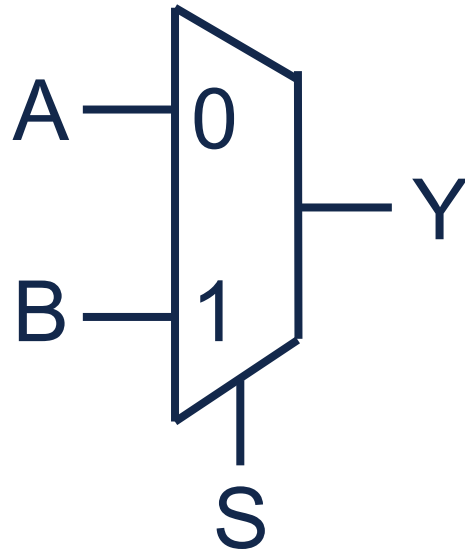
2:1 Multiplexer Symbol

2:1 Multiplexer (2:1 mux)

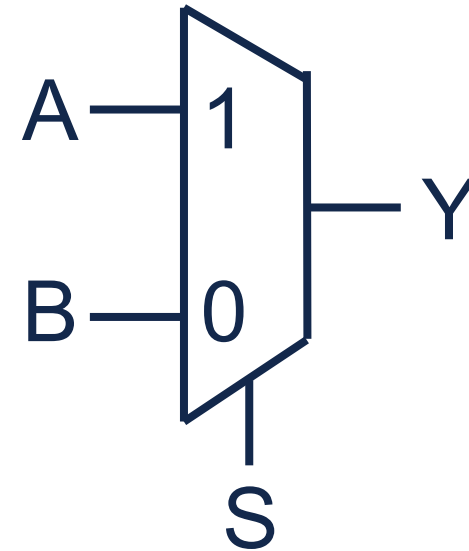


→ When $S = 0$, $Y = A$
→ When $S = 1$, $Y = B$

2:1 Multiplexer (2:1 mux)



When $S = 0$, $Y = A$
When $S = 1$, $Y = B$

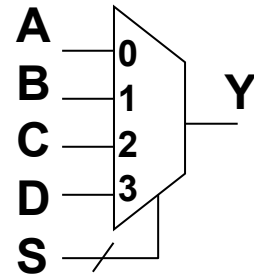
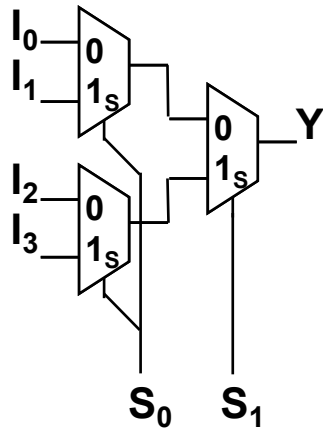


When $S = 0$, $Y = B$
When $S = 1$, $Y = A$

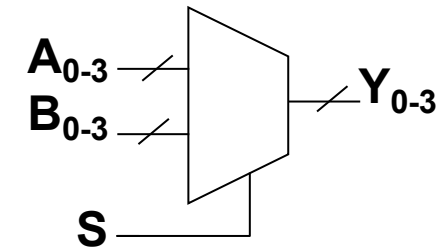
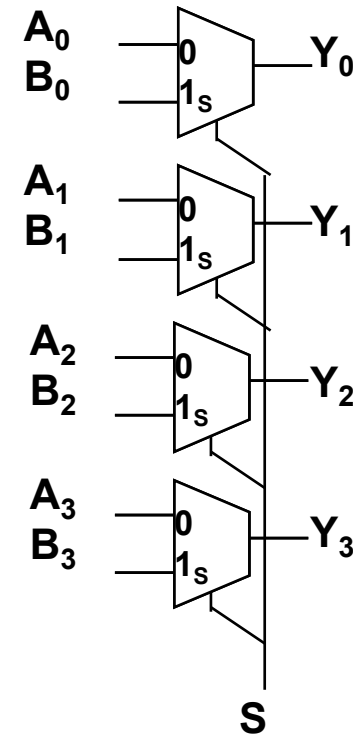
MUX Compositions and Shortcuts



A 4-input Mux
(implemented as a tree)



A 4-bit wide 2-input Mux

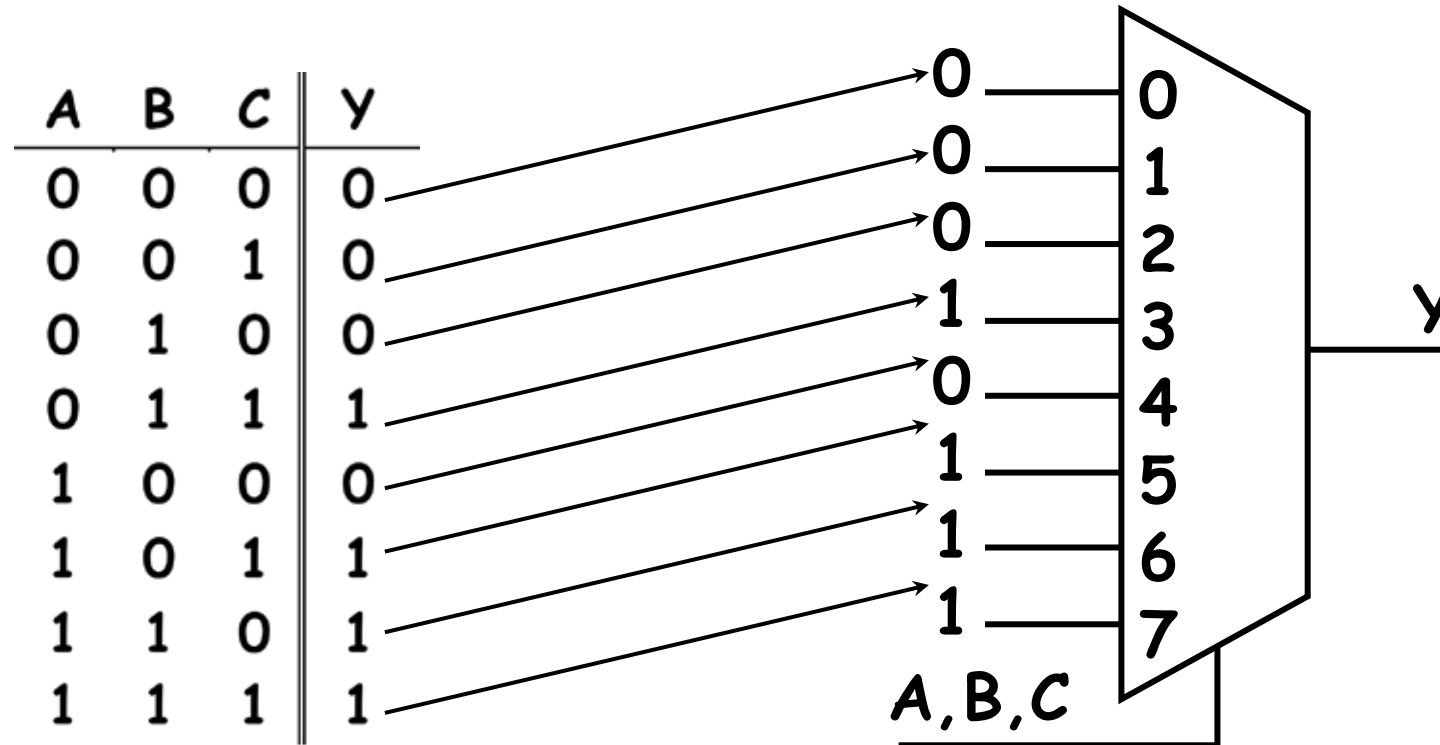


MUX Function Synthesis

A mux plus constants is enough to implement *any* Boolean function

Consider implementation of some arbitrary Combinational function, $F(A,B,C)$... using a MULTIPLEXER as the only circuit element:

Mux: An example “configurable” logic element



8:1
MUX

MUX Logic Tricks

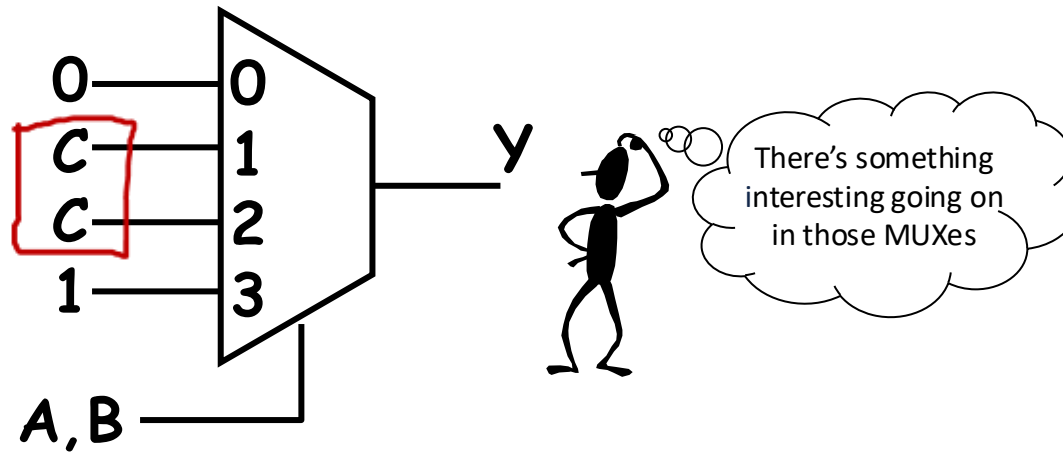


We can apply certain optimizations to MUX Function synthesis

Desired Logic
Function

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

- Largely by inspection or exhaustive search



- N-input function == N-1 input MUX & 1 inverter

If adjacent truth-table rows differ by just one bit, that bit can become a mux input