COMP311: *COMPUTER ORGANIZATION!*

Lecture 9: Don't Cares, Multiplexers, Adders

tinyurl.com/comp311-fa25

KARNAUGH MAPS

Karnaugh Maps (K-Maps)

A Karnaugh map is an *alternate truth-table layout* that places output terms such that shared variable states are adjacent. AS an example consider the 2-variable Karnaugh map for the NAND function.

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

NAND truth table

A \ B	0	1
0	1	1
1	1	0

NAND Karnaugh map

The first row of 1s is where Y = ~A and the left column is where Y = ~B, which together gives

Y = ~A + ~B

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

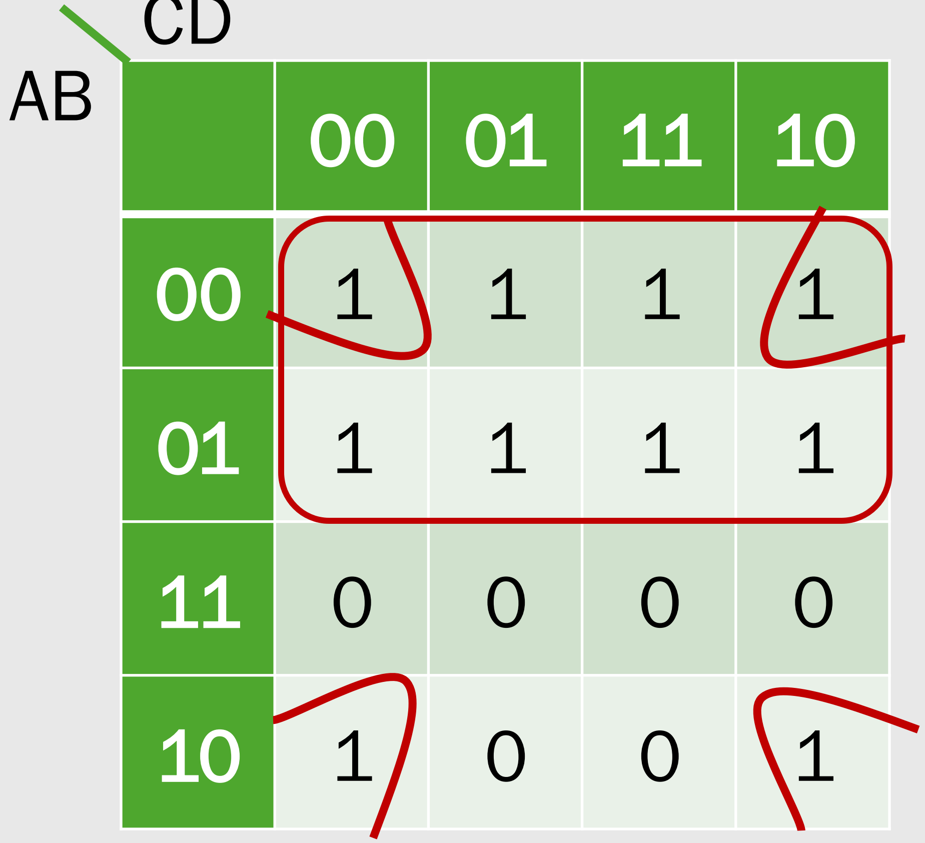
the De Morgan version of a NAND gate

Review: Reading a K-Map

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

$$y = \overline{A} + \overline{B}\,\overline{D}$$

Review: Reading a K-Map



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

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

Simplifying an Equation Using a K-Map

$$Y = \underline{\bar{C}\bar{D}} + ABC\bar{D} + ACD + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

AB \ CD	00	01	11	10
00				
01				
11				
10				

Simplifying an Equation Using a K-Map

$$Y = \bar{C}\bar{D} + \underline{ABC\bar{D}} + ACD + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

AB \ CD	00	01	11	10
00	1			
01	1			
11	1			
10	1			

Simplifying an Equation Using a K-Map

$$Y = \underline{\bar{C}}\bar{D} + ABC\bar{D} + \underline{ACD} + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

AB \ CD	00	01	11	10
00	1			
01	1			
11	1			1
10	1			

Simplifying an Equation Using a K-Map

$$Y = \bar{C}\bar{D} + ABC\bar{D} + ACD + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

CD

AB

	00	01	11	10
00	1			
01	1			
11	1		1	1
10	1		1	

Simplifying an Equation Using a K-Map

$$Y = \bar{C}\bar{D} + ABC\bar{D} + ACD + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	1		1	1
10	1		1	

Simplifying an Equation Using a K-Map

$$Y = \bar{C}\bar{D} + ABC\bar{D} + ACD + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	1		1	1
10	1		1	1

Simplifying an Equation Using a K-Map

$$Y = \bar{C}\bar{D} + ABC\bar{D} + ACD + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

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

$y = AC + \bar{D}$

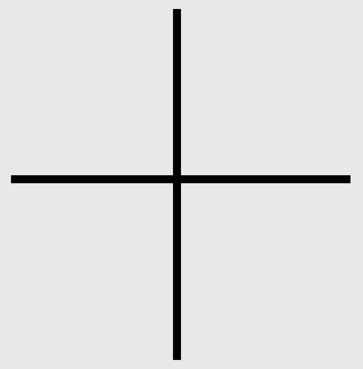
Simplifying an Equation Using a K-Map

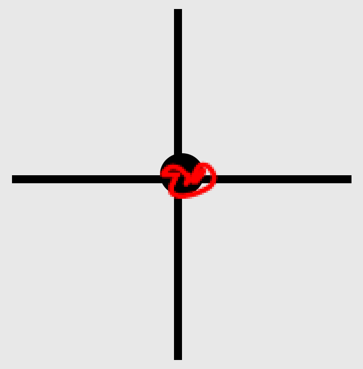
$$Y = \bar{C}\bar{D} + ABC\bar{D} + ACD + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$$

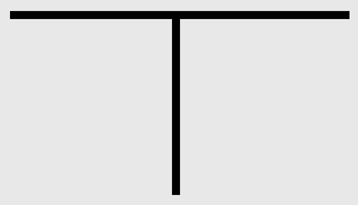
AB \ CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	1	1
10	1	0	1	1

$$Y = AC + \bar{D}$$

Wire Crossing Notation


Not connected


Connected


Connected

Multiple Output Circuits

A	B	C	Y_1	Y_0
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

Multiple Output Circuits

A	B	C	Y_1	Y_0
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

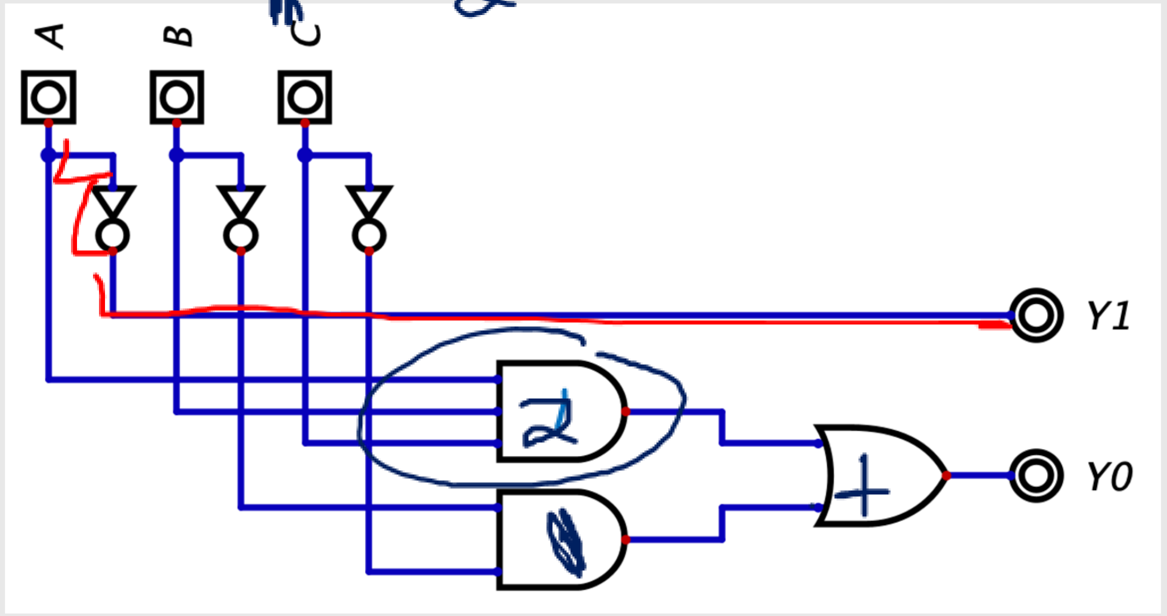
$$\rightarrow \quad Y_1 = \underline{\bar{A}\bar{B}\bar{C}} + \underline{\bar{A}\bar{B}C} + \underline{\bar{A}B\bar{C} + \bar{A}BC}$$

$$Y_1 = \bar{A}$$

$$\rightarrow \quad Y_0 = \underline{\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + ABC}$$

$$Y_0 = \bar{B}\bar{C} + ABC$$

2

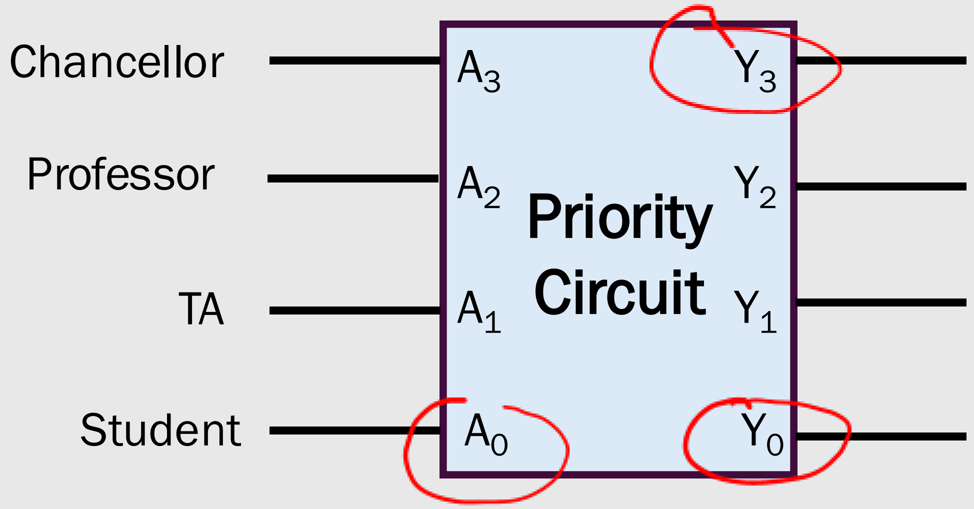


Priority Circuit

I want to design a circuit that will determine who can reserve a classroom space.

Anyone who wants to reserve the room will assert their input (set their input to 1).

The order of priority for room reservations is the chancellor, professors, TAs, and students. Assume that only one person from each category will submit a request at a given time. The requestor with the highest priority will get the room reservation.



The output that is 1 will tell us who gets the room reservation (only one output will be 1 at any time)

Circuit Design Activity

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			
1	0	1	0	1			
1	0	1	1	1			
1	1	0	0	1			
1	1	0	1	1			
1	1	1	0	1			
1	1	1	1	1			

1. Fill in the truth table
2. Write down the simplified SOP equation
3. Draw the circuit diagram

Create the truth table for this circuit

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

$$y_0 = \overline{A_3}\,\overline{A_2}\,\overline{A_1}\,A_0$$

$$y_1 = \overline{A_2}\,\overline{A_2}\,A_1$$

$$y_2 = \overline{A_3}\,A_2$$

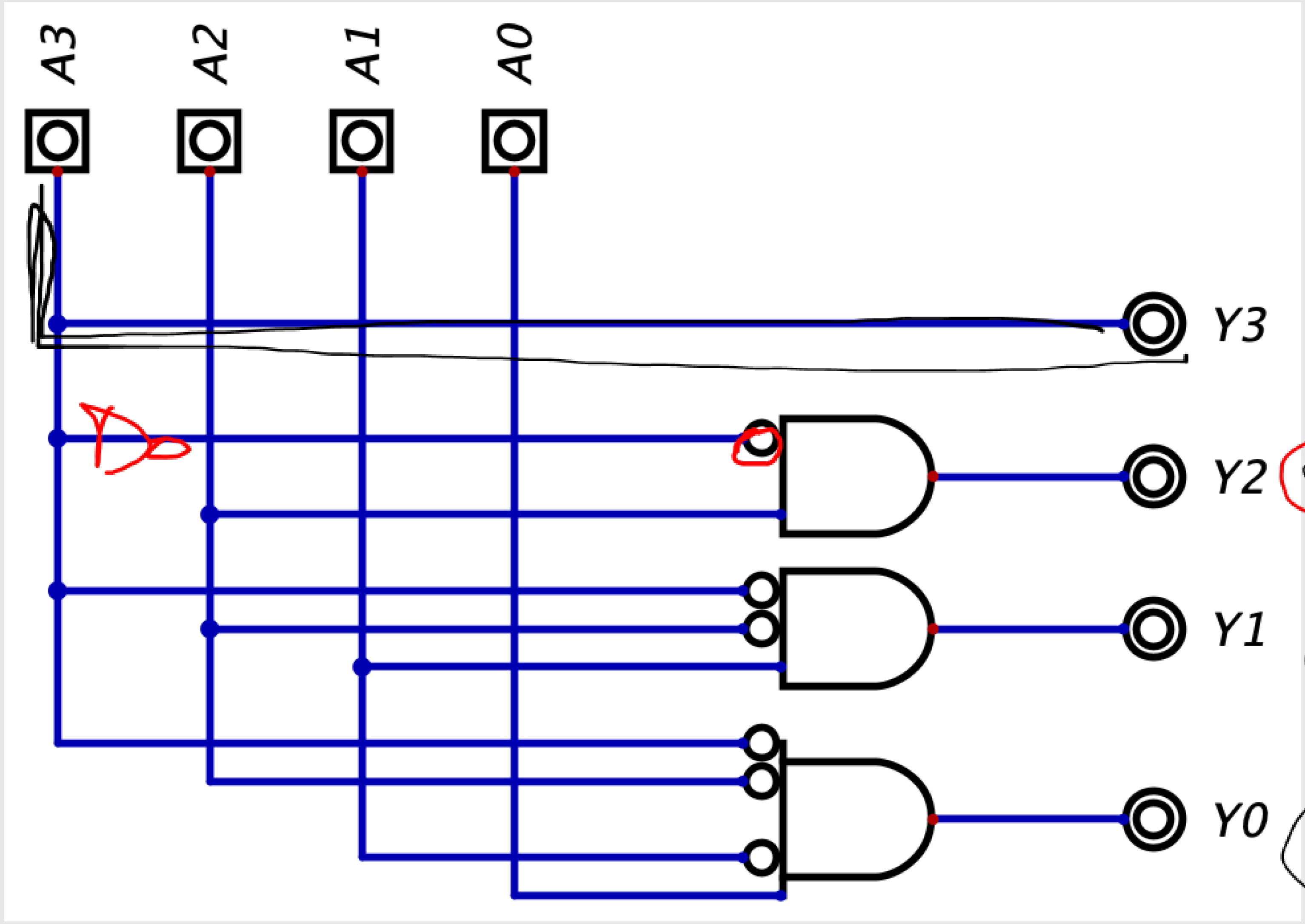
$$y_3 = A_3$$

Find the simplified SOP equation for each output

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

$$Y_3 = A_3$$
$$Y_2 = \overline{A_3}A_2$$
$$Y_1 = \overline{A_3}\,\overline{A_2}A_1$$
$$Y_0 = \overline{A_3}\,\overline{A_2}\,\overline{A_1}A_0$$

The schematic



$Y_3 = A_3$

$Y_2 = \overline{A_3}A_2$ + A_0

$Y_1 = \overline{A_3}\,\overline{A_2}A_1$

$Y_0 = \overline{A_3}\,\overline{A_2}\,\overline{A_1}A_0$

Don’t Cares in the Priority Circuit

Whenever A_3 = 1…

- Y_3 = 1
- Y_2 = 0
- Y_1 = 0
- Y_0 = 0

It does not matter wh

and A_0 are!

We can say that v

care” about the v

A_2, A_1, and A

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
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

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.

Don’t Cares in the Priority Circuit

Whenever $A_3 = 1$…

- $Y_3 = 1$
- $Y_2 = 0$
- $Y_1 = 0$
- $Y_0 = 0$

It does not matter what A_2, A_1, and A_0 are!

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	X	X	X	1	0	0	0

We can say that we **“don’t care”** about the values of A_2, A_1, and A_0

Don’t Cares in the Priority Circuit

Whenever $A_3 = 0$ and $A_2 = 1$

- $Y_3 = 0$
- $Y_2 = 1$
- $Y_1 = 0$
- $Y_0 = 0$

It does not matter what A_1, and A_0 are

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	X	X	X	1	0	0	0

We can say that we **“don’t care”** about the values of A_1 and A_0

Don’t Cares in the Priority Circuit

Whenever $A_3 = 0$ and $A_2 = 1$

- $Y_3 = 0$
- $Y_2 = 1$
- $Y_1 = 0$
- $Y_0 = 0$

It does not matter what A_1, and A_0 are

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	X	X	0	1	0	0
1	X	X	X	1	0	0	0

We can say that we **“don’t care”** about the values of A_1 and A_0

Don’t Cares in the Priority Circuit

Whenever $A_3 = 0$, $A_2 = 0$, and $A_1 = 1$

- $Y_3 = 0$
- $Y_2 = 0$
- $Y_1 = 1$
- $Y_0 = 0$

It does not matter what A_0 is

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	X	X	0	1	0	0
1	X	X	X	1	0	0	0

We can say that we “don’t care” about the value A_0

Don’t Cares in the Priority Circuit

Whenever $A_3 = 0$, $A_2 = 0$, and $A_1 = 1$

- $Y_3 = 0$
- $Y_2 = 0$
- $Y_1 = 1$
- $Y_0 = 0$

It does not matter what A_0 is

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

We can say that we “don’t care” about the value A_0

Boolean Equations from 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

$$Y_3 = A_3$$
$$Y_2 = \overline{A_3}A_2$$
$$Y_1 = \overline{A_3}\,\overline{A_2}A_1$$
$$Y_0 = \overline{A_3}\,\overline{A_2}\,\overline{A_1}A_0$$

Transforming the inputs to don’t cares makes finding the simplified SOP equations much easier!

Output Values Can Also Be Don’t Care!

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

When A is 0 and and B is 1, the output can be 0 or 1!

Output Values Can Also Be Don’t Care!

There are two possible implementations of this truth table

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

Output Values Can Also Be Don’t Care!

There are two possible implementations of this truth table

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

Since the output of row1 can be either a 0 or 1, let’s see what happens if we treat the output as 0.

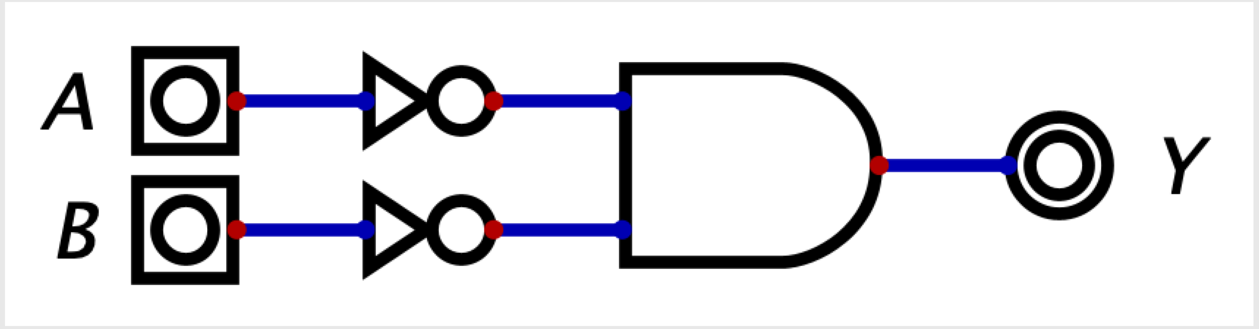
Output Values Can Also Be Don't Care!

There are two possible implementations of this truth table

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

If we treat the X as a 0, then we get the following circuit:

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



Output Values Can Also Be Don't Care!

There are two possible implementations of this truth table

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

Since the output of row1 can be either a 0 or 1, let's see what happens if we treat the output as 1.

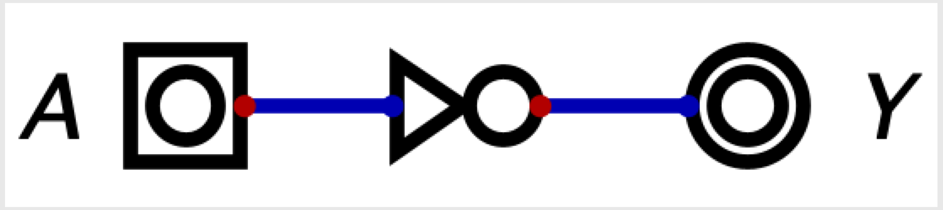
Output Values Can Also Be Don't Care!

There are two possible implementations of this truth table

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

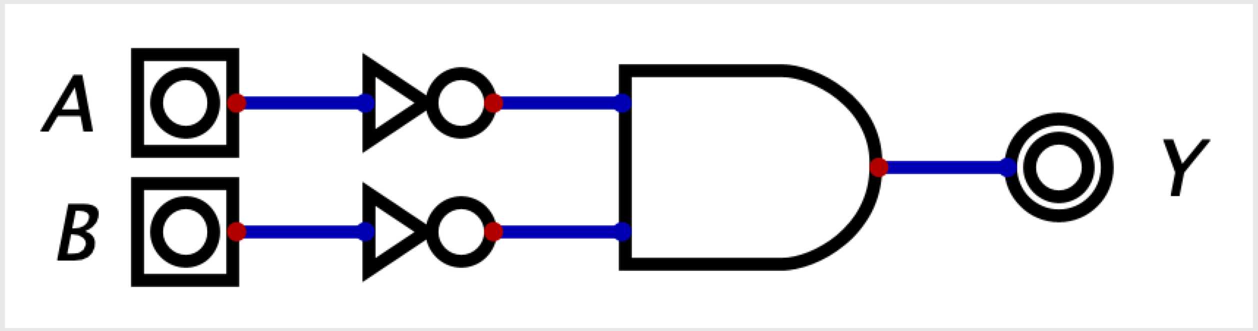
If we treat the X as a 1, then we get the following circuit:

$$Y = \bar{A}$$

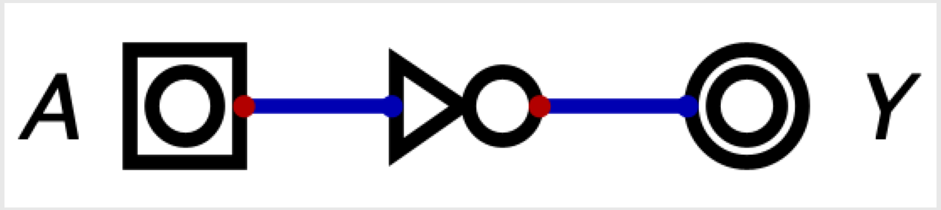


Which circuit should I choose for my design?

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

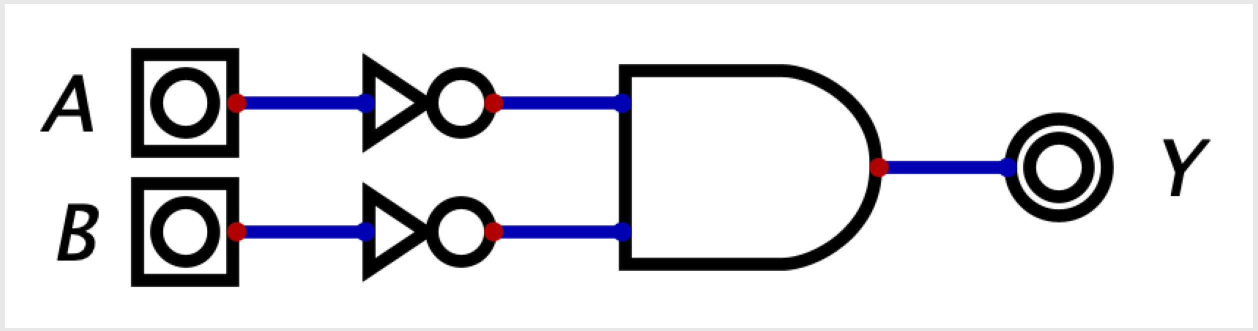


$Y = \bar{A}$

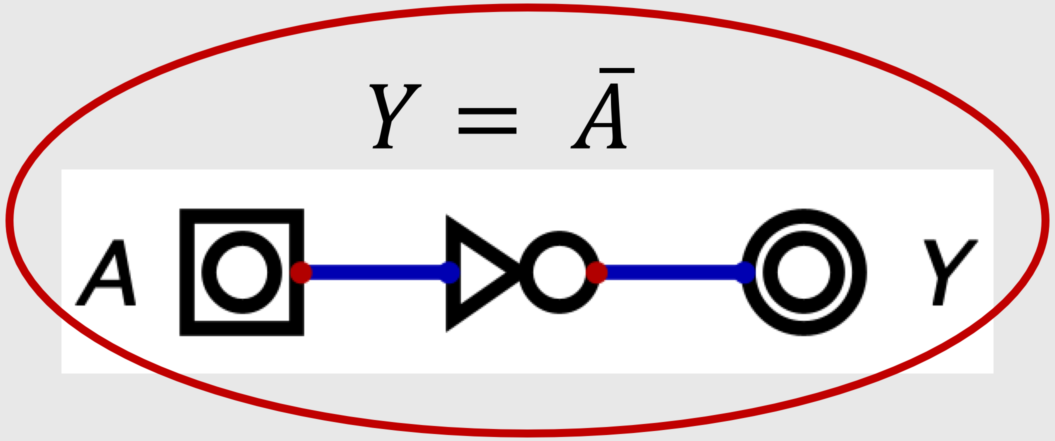


Which circuit should I choose for my design?

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



$$Y = \bar{A}$$



This circuit is better because it contains fewer transistors

Don’t Care Outputs

- When designing our circuits, we will treat X as a 0 or a 1 depending on which decision yields the fewest number of transistors.
- How do we know which decision is better?
  - *In a K-map, if treating the X as a 1 yields fewer or larger circles, then treat it as a 1.*
  - *If not, treat the X as a 0.*
  - *If there are multiple X’s in a truth table, handle them individually*
    - i.e. you can treat some X’s as 1s and some X’s as 0s

Don’t Cares in K-Maps

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	X	0
11	0	1	X	0
10	0	0	0	0

Don’t Cares in K-Maps

- There are four ways we can interpret these don’t cares. Which way yields the design with the fewest transistors?

CD
AB

	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	1	0
10	0	0	0	0

CD
AB

	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

CD
AB

	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	0	0	0

CD
AB

	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	0	0
10	0	0	0	0

Don’t Cares in K-Maps

- There are four ways we can interpret these don’t cares. Which way yields the design with the fewest transistors?

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

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

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

$Y = B\bar{C}D$

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

Don’t Cares in K-Maps

- There are four ways we can interpret these don’t cares. Which way yields the design with the fewest transistors?

CD

AB

	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	1	0
10	0	0	0	0

$Y = \underline{B}\bar{C}D \boxed{+} \underline{AB}D$

CD

AB

	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

CD

AB

	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	0	0	0

$Y = B\bar{C}D$

CD

AB

	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	0	0
10	0	0	0	0

Don’t Cares in K-Maps

- There are four ways we can interpret these don’t cares. Which way yields the design with the fewest transistors?

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

$Y = B\bar{C}D + ABD$

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

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

$Y = B\bar{C}D$

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

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

Don’t Cares in K-Maps

- There are four ways we can interpret these don’t cares. Which way yields the design with the fewest transistors?

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

$Y = B\bar{C}D + ABD$

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

$Y = BD$

This implementation contains the fewest number of transistors

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

$Y = B\bar{C}D$

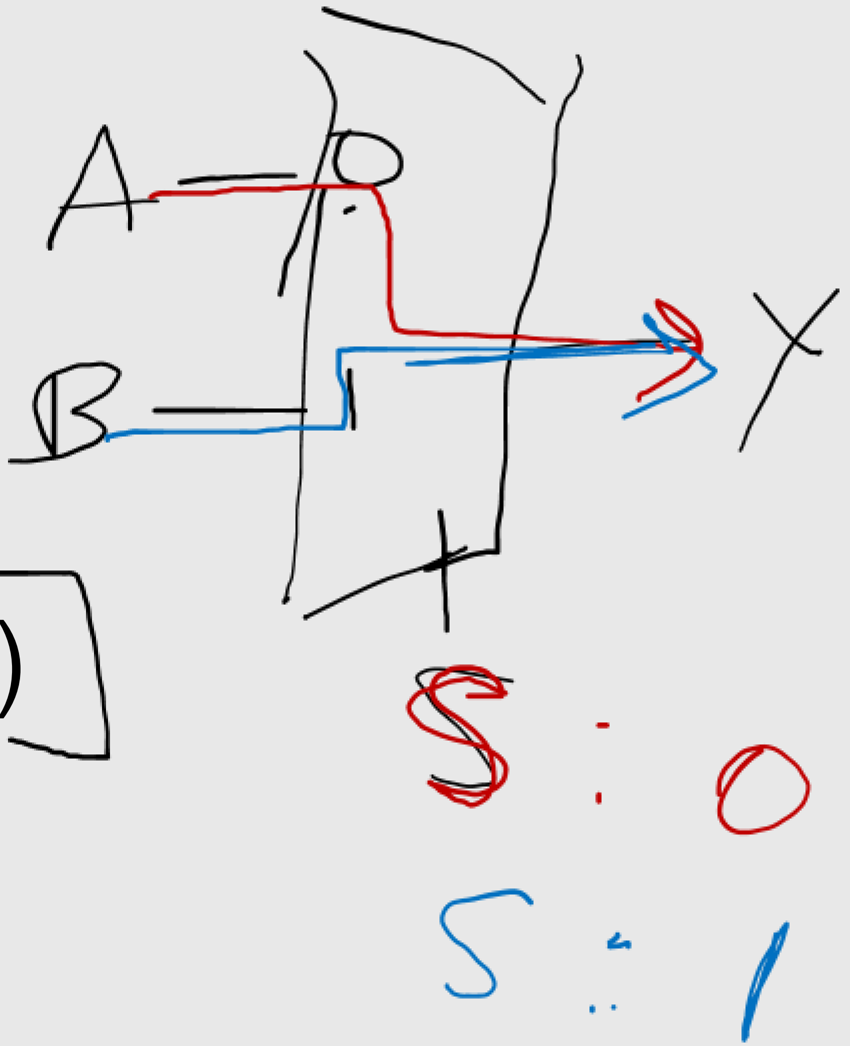
AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	0	0
10	0	0	0	0

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

MULTIPLEXERS

A Circuit for If-Statements

```
if (S == 0)
    Y = A
elseif (S == 1)
    Y = B
```



Implementing an if-statement

Truth Table

Equation

Diagram

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

Implementing an if-statement

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

Implementing an if-statement

Truth Table

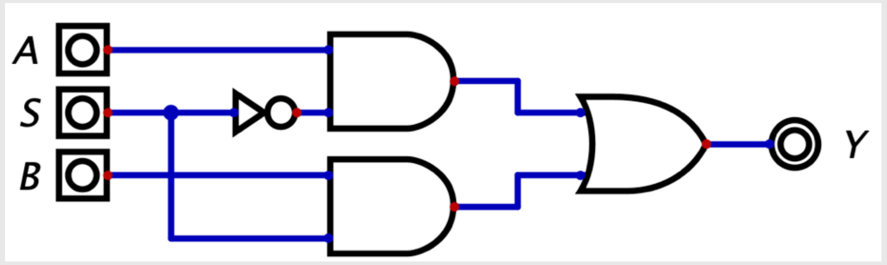
S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

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

Equation

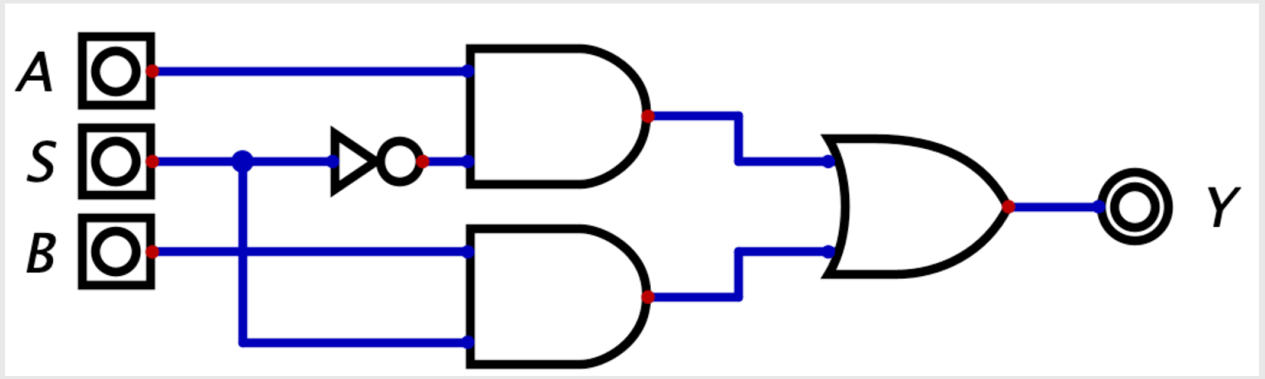
$$Y = A\bar{S} + BS$$

Diagram

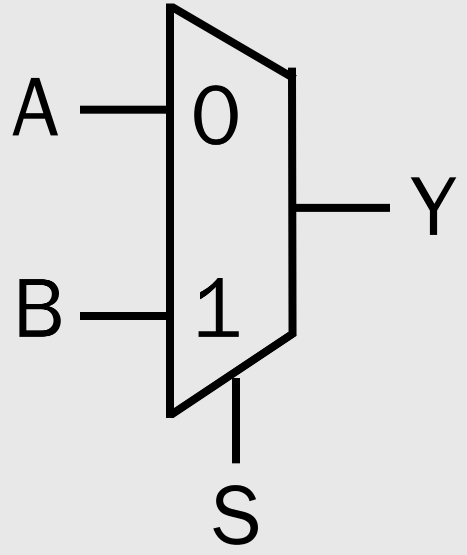


2:1 Multiplexer (2:1 mux)

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

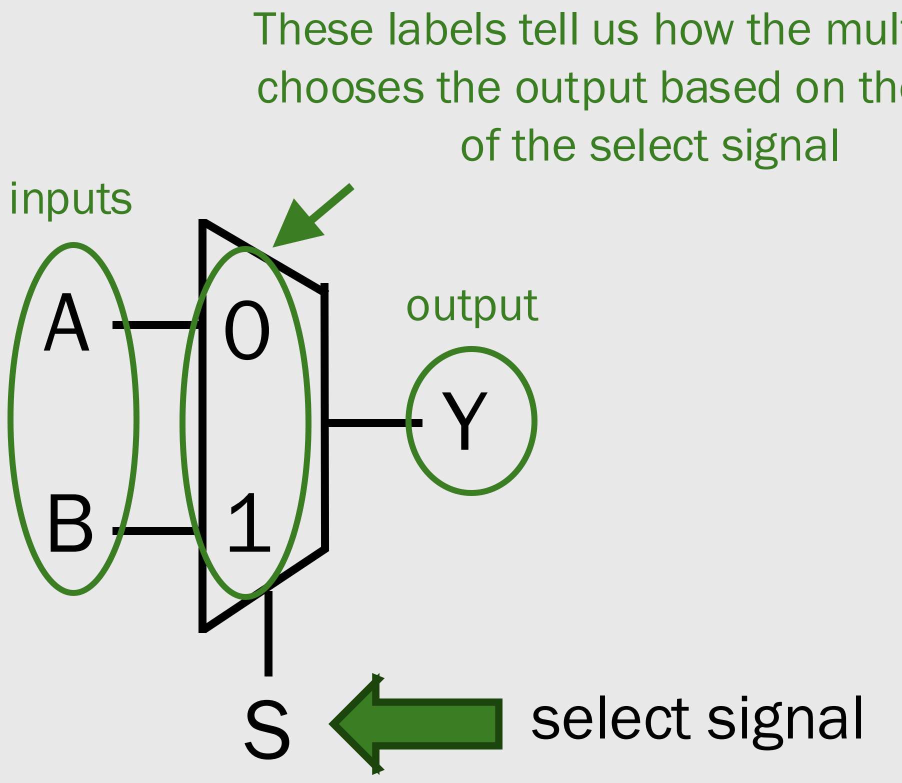


2:1 Multiplexer Schematic
(Gate Diagram)



2:1 Multiplexer Symbol

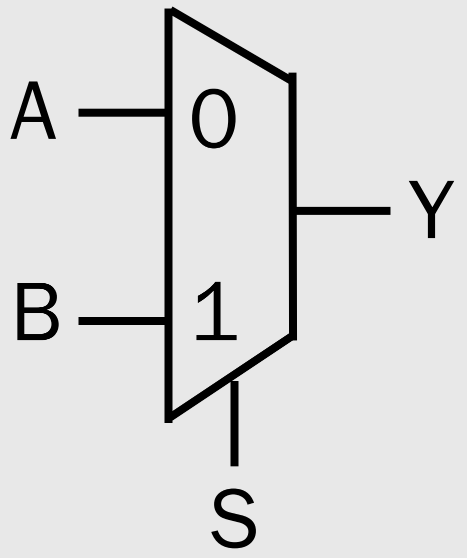
2:1 Multiplexer (2:1 mux)


These labels tell us how the multiplexer chooses the output based on the value of the select signal
inputs
A
B
0
1
output
Y
S
select signal

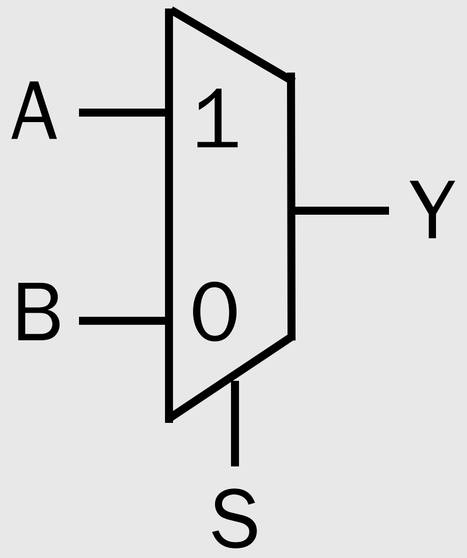
$\longrightarrow$ When S = 0, Y = A

$\longrightarrow$ When S = 1, Y = B

2:1 Multiplexer (2:1 mux)



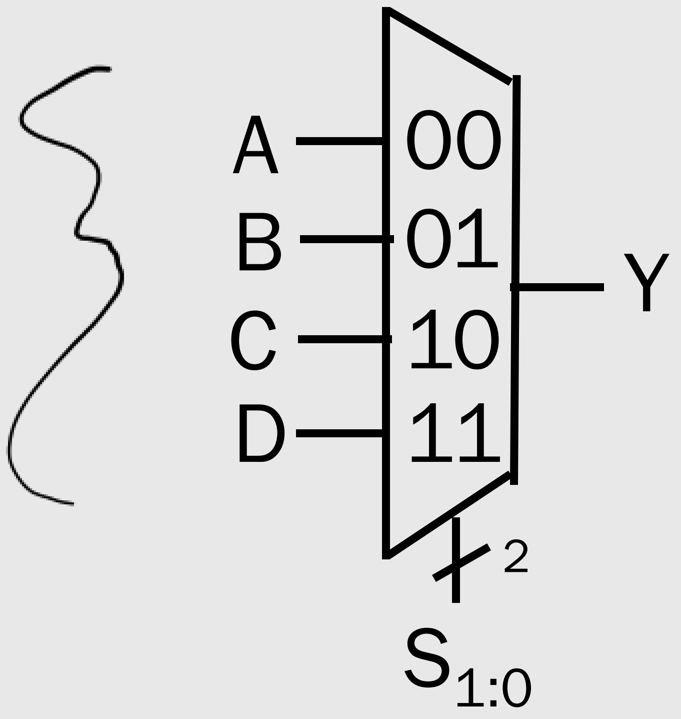
When S = 0, Y = A
When S = 1, Y = B



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

4:1 Multiplexer (4:1 mux)

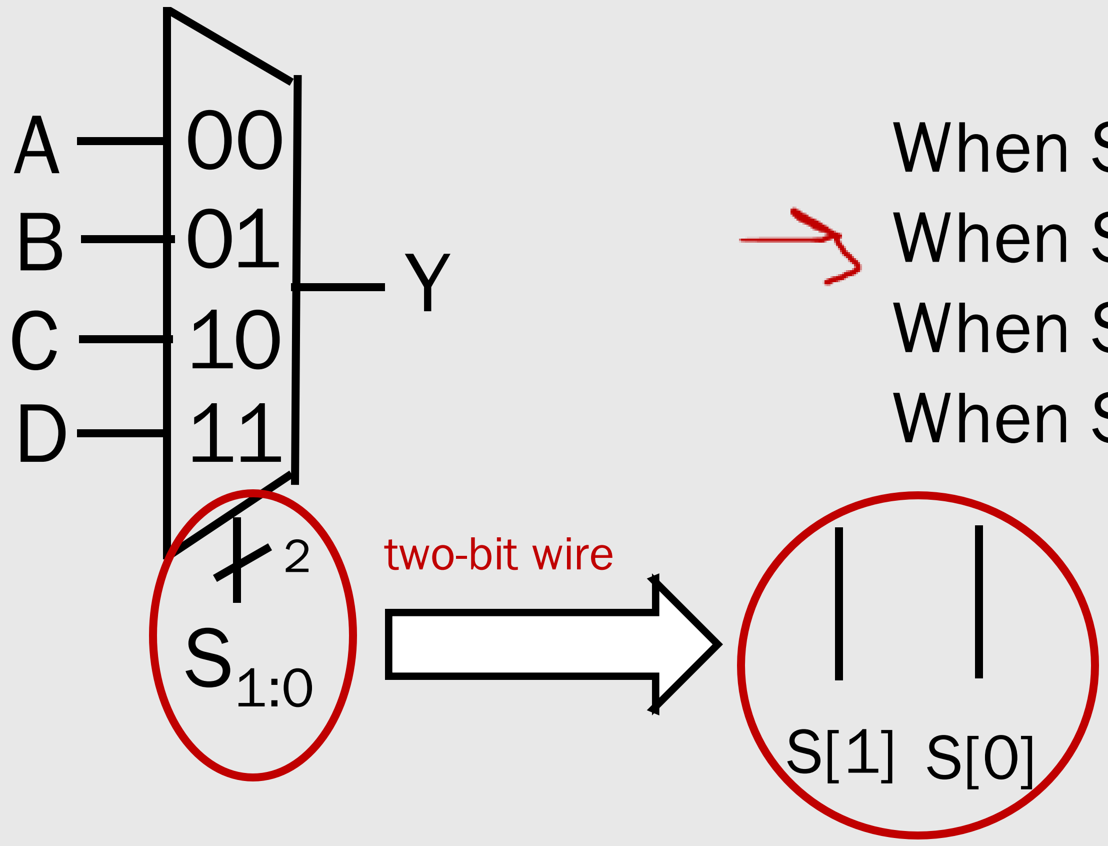
- A circuit that chooses an output from among four inputs based on the value of a select signal



When S = 0b00, Y = A
When S = 0b01, Y = B
When S = 0b10, Y = C
When S = 0b11, Y = D

4:1 Multiplexer (4:1 mux)

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



When S = 0b00, Y = A

When S = 0b01, Y = B

When S = 0b10, Y = C

When S = 0b11, Y = D

Create the gate-level implementation of a 4:1 Multiplexer (4:1 mux)

1. Draw the truth table
2. Find the equation
3. Draw the schematic

Create the gate-level implementation of a 4:1 Multiplexer (4:1 mux)

S1	S0	A	B	C	D	Y
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0					
1	0					
1	1					
1	1					

Create the gate-level implementation of a 4:1 Multiplexer (4:1 mux)

S1	S0	A	B	C	D	Y
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

$$Y = A\overline{S}_1\overline{S}_0 + B\overline{S}_1S_0 + CS_1\overline{S}_0 + D\,S_1S_0$$

4:1 Multiplexer (4:1 mux)

Equation

$$Y = A\overline{S}_1\overline{S}_0 + B\overline{S}_1S_0 + CS_1\overline{S}_0 + D\ S_1S_0$$

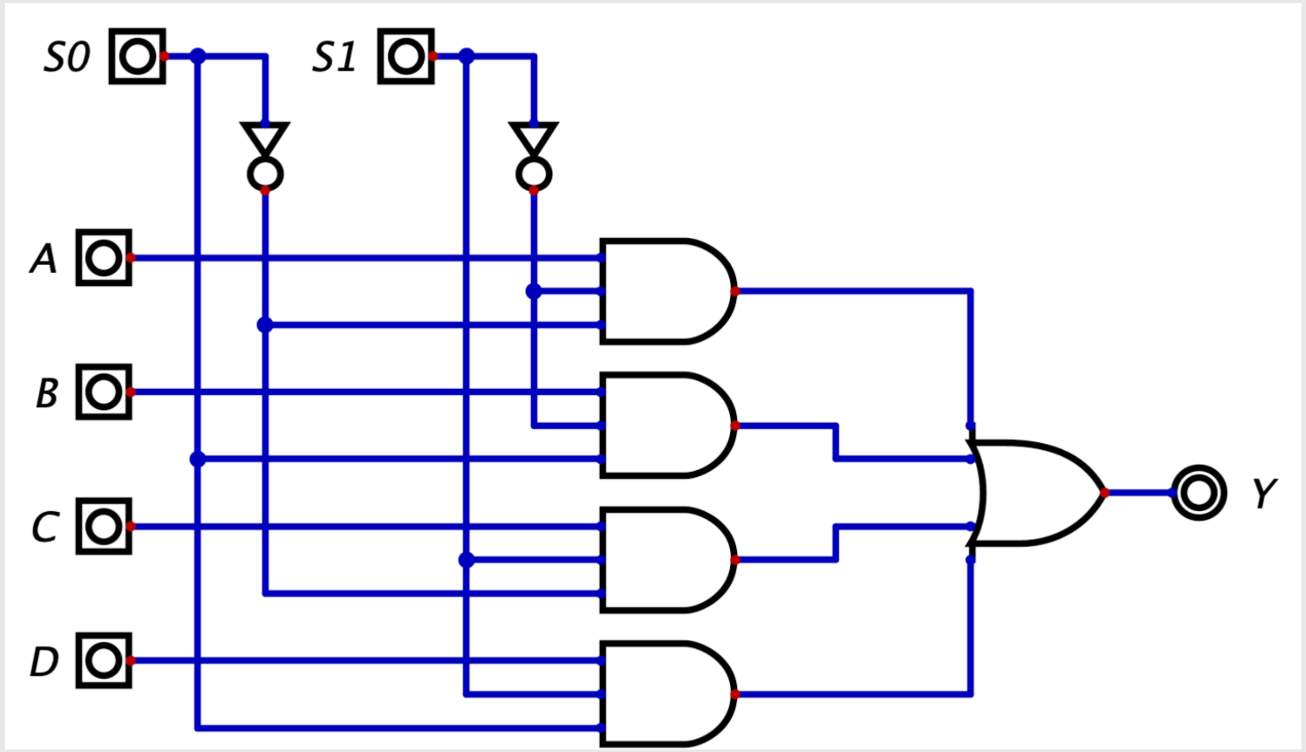
Diagram

4:1 Multiplexer (4:1 mux)

Equation

$$Y = A\overline{S_1}\,\overline{S_0} + B\overline{S_1}S_0 + CS_1\overline{S_0} + D\,S_1S_0$$

Diagram



Multiplexers

- Create a 4:1 multiplexer out of 2:1 multiplexers.

S1	S0	Y
0	0	A
0	1	B
1	0	C
1	1	D

A

B

C

D

S_1

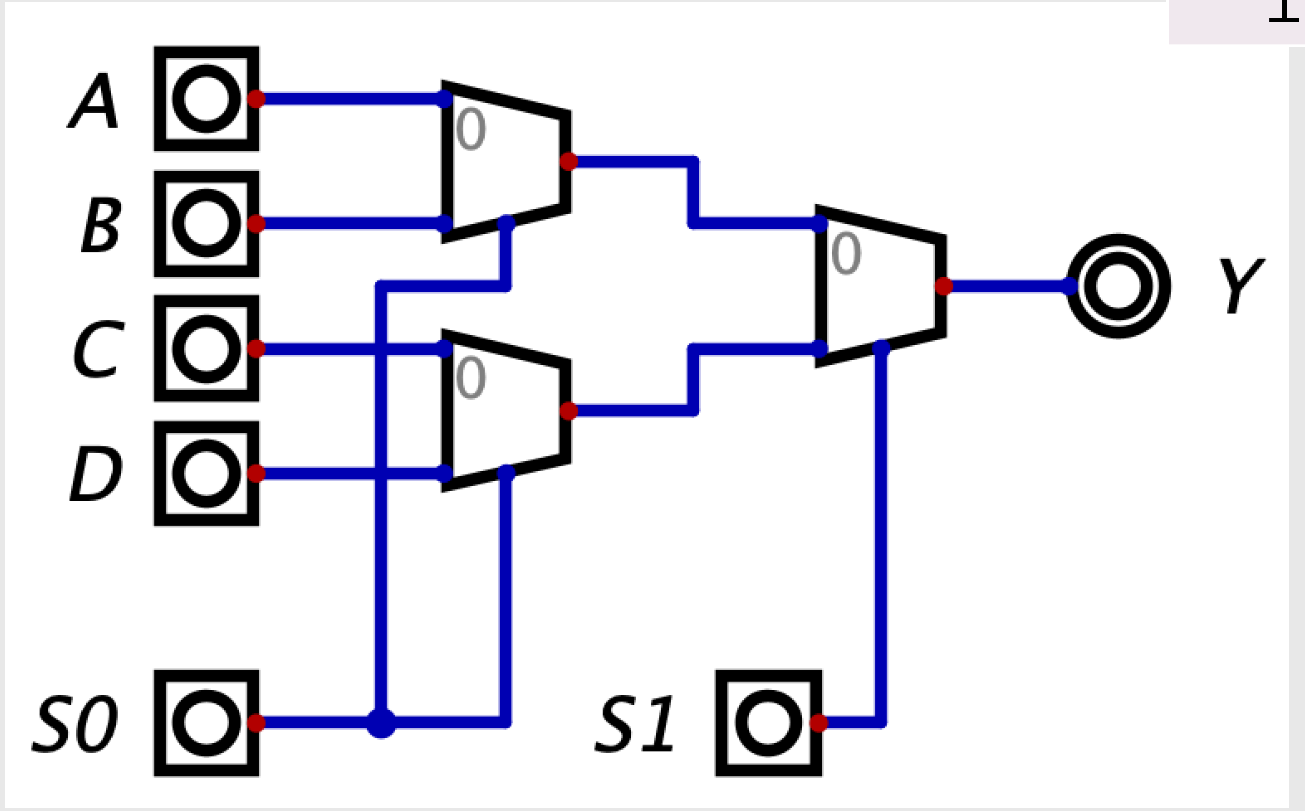
S_0

Y

Multiplexers

- Create a 4:1 multiplexer out of 2:1 multiplexers.

S1	S0	Y
0	0	A
0	1	B
1	0	C
1	1	D


A
B
C
D
0
0
0
Y
S0
S1