ECE 550D
Fundamentals of Computer Systems and Engineering

Fall 2023

Combinational Logic

Xin Li & Dawei Liu
Duke Kunshan University

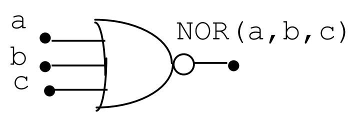
Slides are derived from work by
Andrew Hilton, Tyler Bletsch and Rabih Younes (Duke)

Last time….

- Who can remind us what we talked about last time?
  - Electric circuit basics
    - Vcc = 1
    - Ground = 0
  - Transistors
    - PMOS
    - NMOS
  - Gates
    - Complementary PMOS + NMOS
    - Output is logical function of inputs

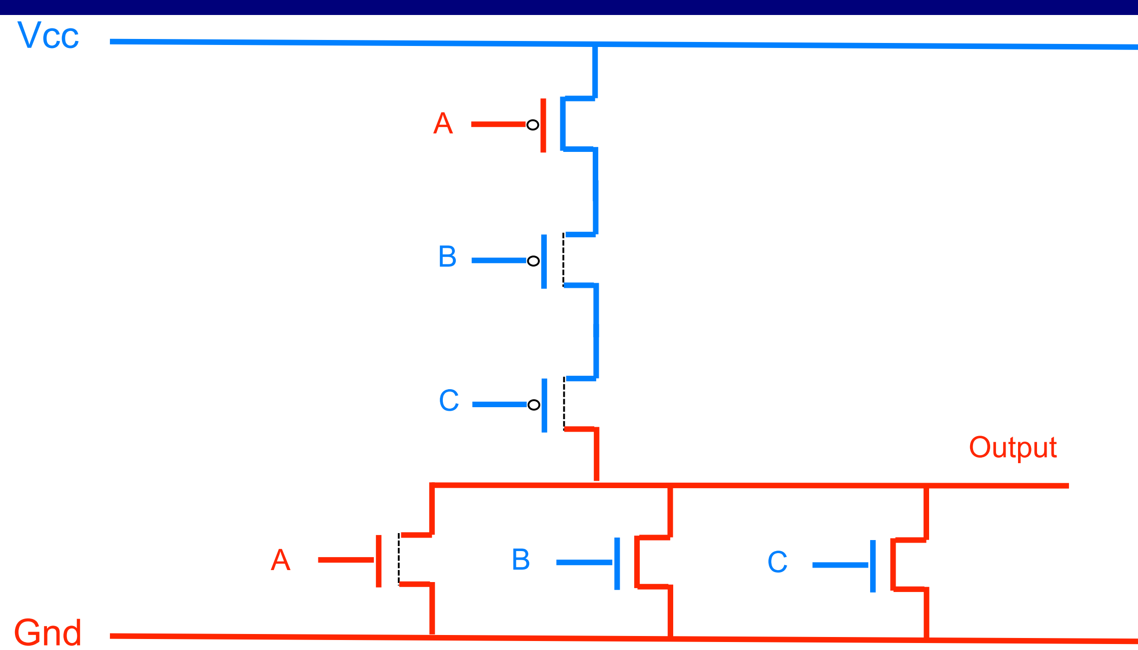
Multi-input gates

- So far gates have had 1 or 2 inputs
  - Can have more, though typically stop at 3 or 4
  - Symbols stay the same, just have more input lines



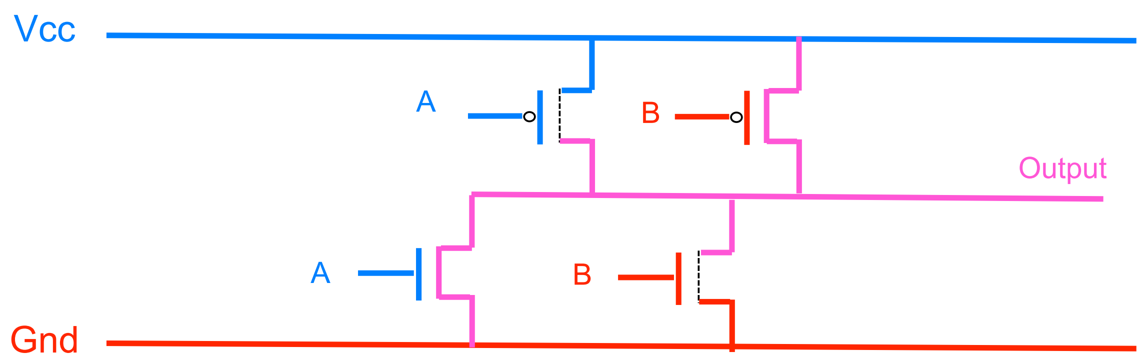
A	B	C	Ouput
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Three input NOR Gate



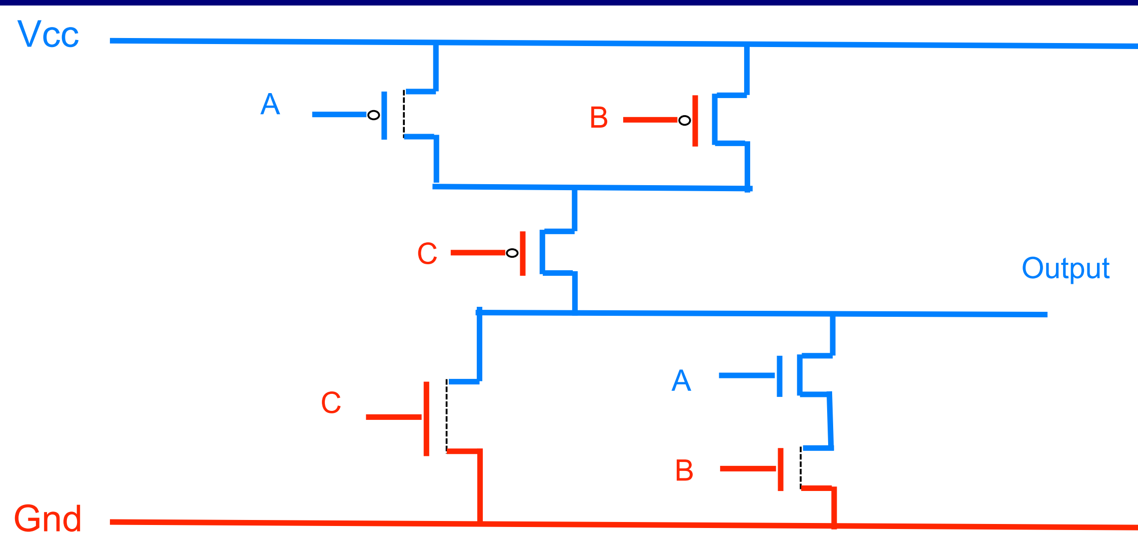
- Similar to two input, more transistors
  - Slightly slower

Complementary: very important



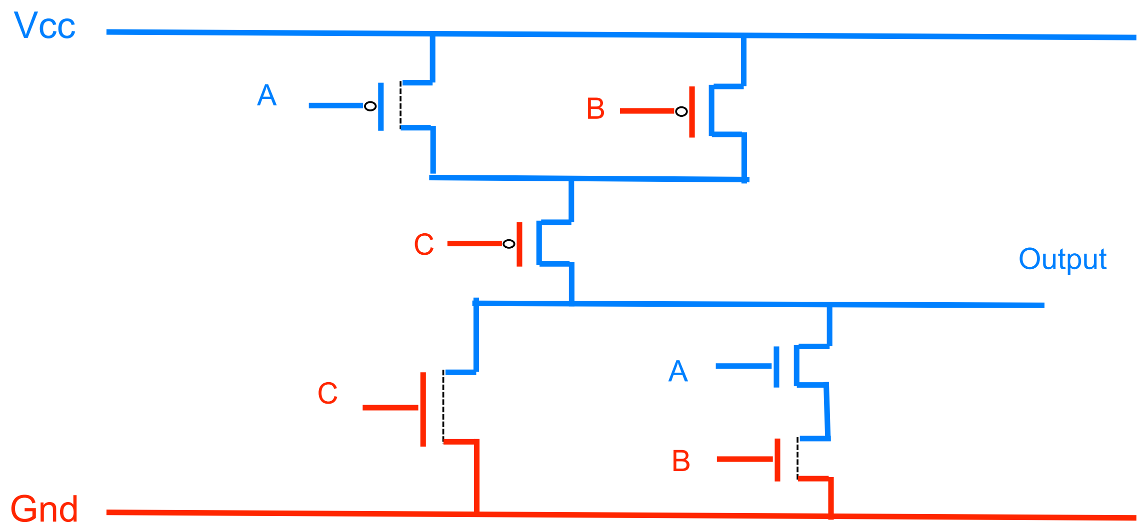
- Complementary nature: very important
  - Without it, we have a problem
  - Here: both PMOS and NMOS in parallel…
  - A=1, B= 0 (or A =0, B=1) forms short-circuit
    - Chip catches fire ☹

Complementary: very important



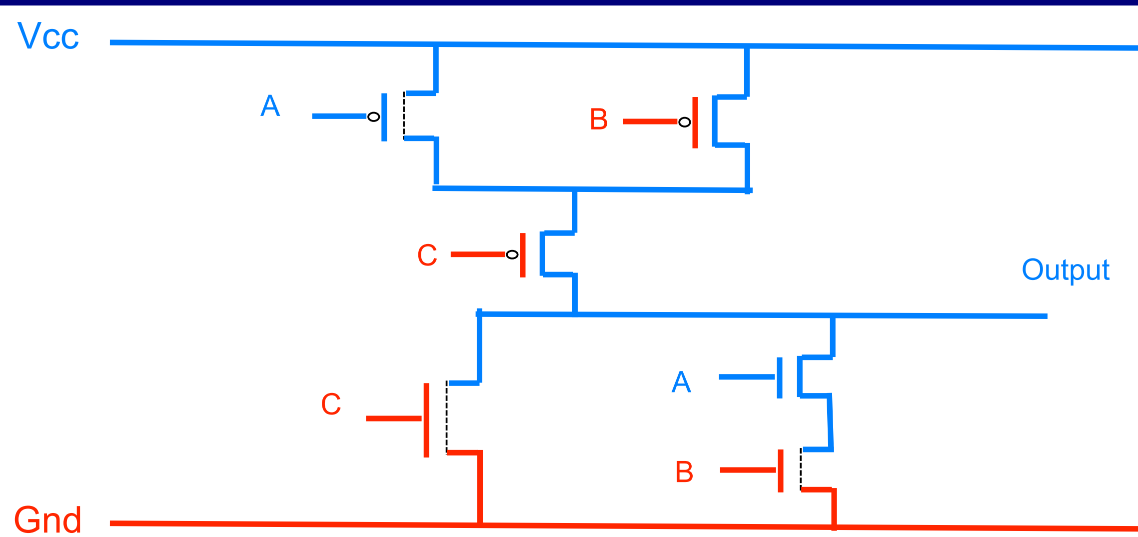
- With more than 2 inputs, can get very complicated
  - Are the PMOS and NMOS complementary here?
  - We can go the other way: transistors -> formulas
  - Check if formulas logically equivalent

Complementary: very important



- Everyone take a second to write down the formulas
  - PMOS: NOTs on inputs
  - NMOS: NOT around the outside

Complementary: very important



- Everyone take a second to write down the formulas
  - PMOS: ((Not A) or (Not B)) and (Not C)
  - NMOS: Not (C or (A and B))
    = (Not C) and (Not (A and B))
    = (Not C) and ((Not A) or (Not B))

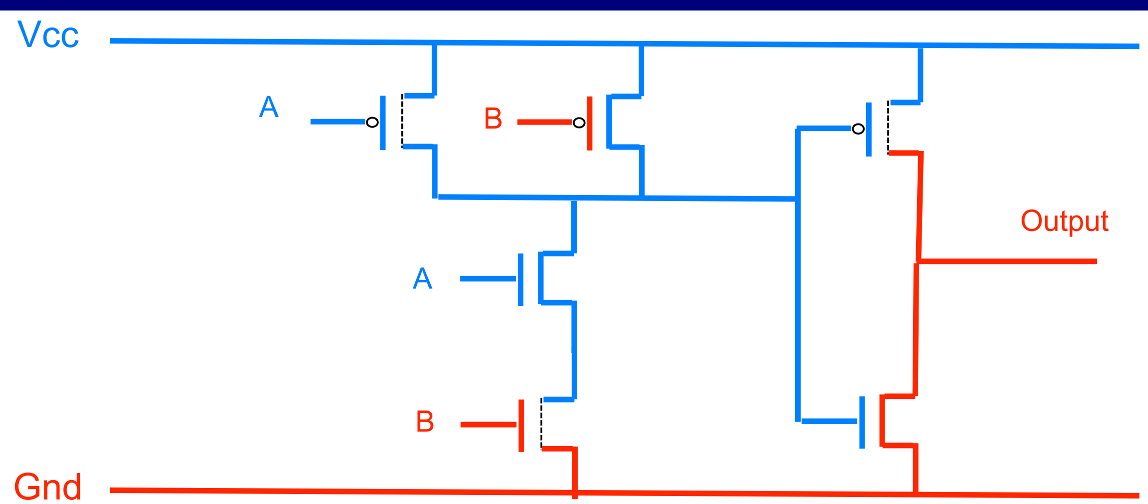
What about… AND?

- Saw and did NAND, but what about AND?
  - Truth table on the right…

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

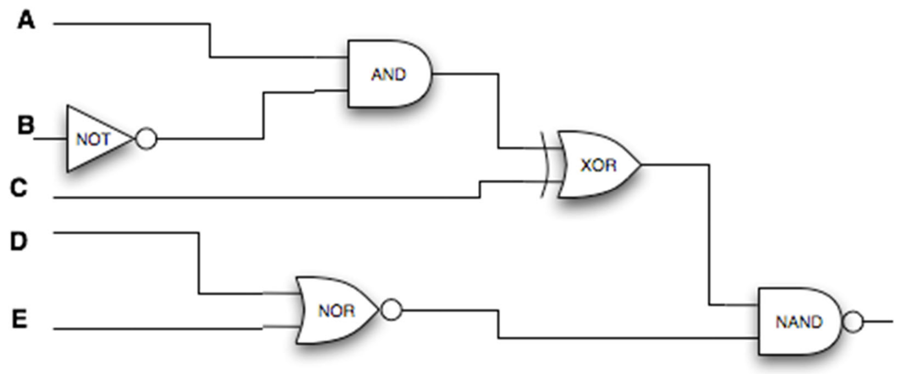
  - Trying to do this causes problems
    - PMOS formula: NOTs on inputs
    - NMOS formula: NOT around outside
    - … can’t seem to find a formula which works (need more NOTs):
  - AND gate is really a couple gates squished together
    - Not (Nand (A,B))
    - Nor(Not A, Not B)

The AND Gate



- The AND gate
  - A NAND gate followed by a NOT gate
  - Also a good example of how gates connect together
    - Output of one gate goes to input of another

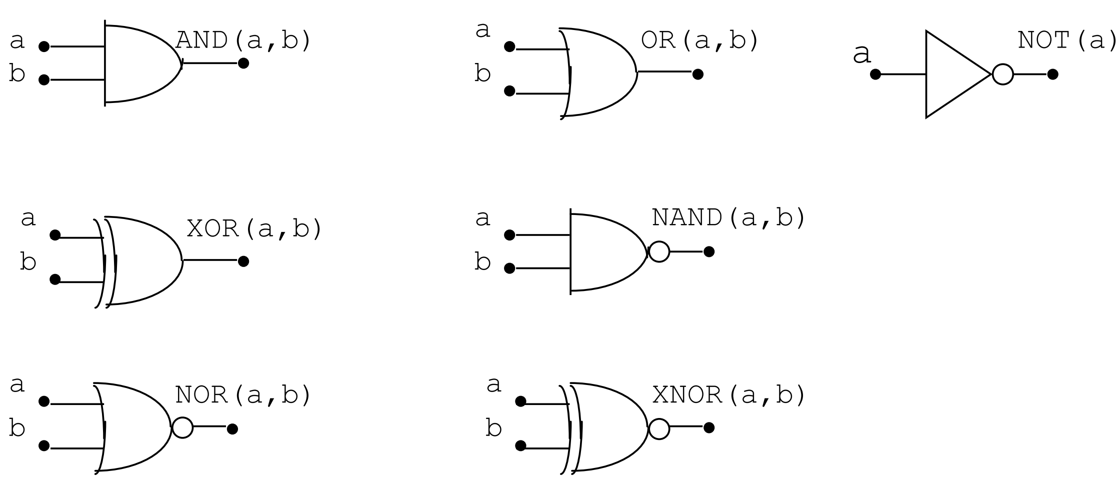
Going forwards: Logic Gates



- Going forwards, will mostly design from gates
  - Abstract away transistor level implementation

Boolean Gates

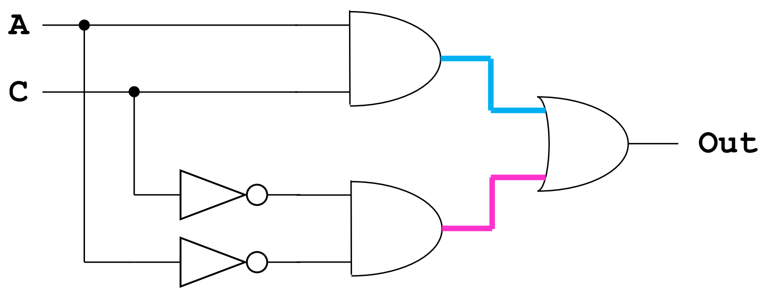
- Saw these gates
  - Mnemonic to remember them



Boolean Functions, Gates and Circuits

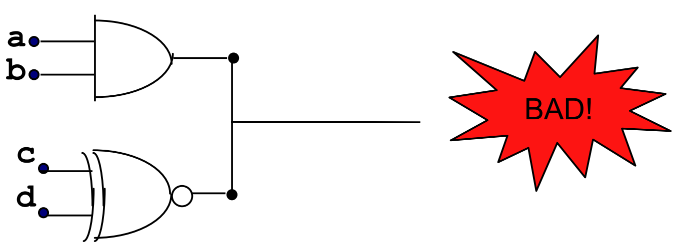
- Circuits are made from a network of gates.

`(!A & !C) | (A & C)`



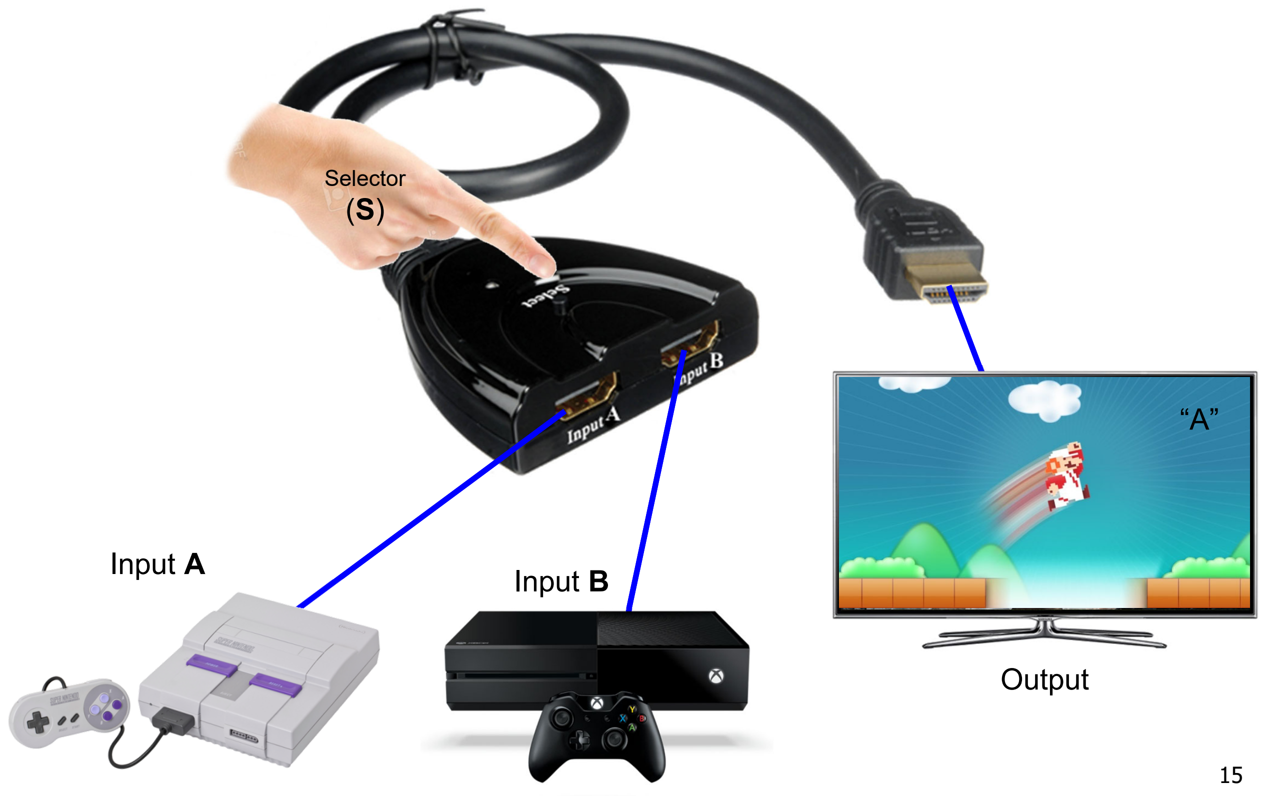
A few more words about gates

- Gates have inputs and outputs
  - If you try to hook up two outputs, get short circuit
    (Think of the transistors each gate represents)

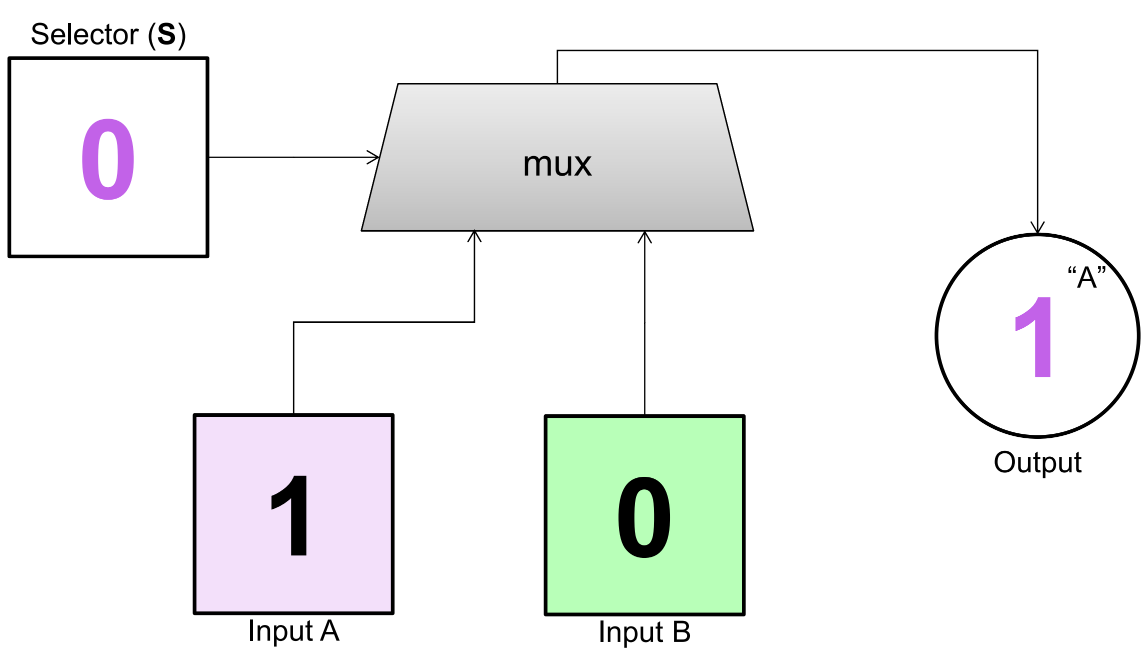


- If you don't hook up an input, it behaves kind of randomly
(also not good, but not set-your-chip-on-fire bad)

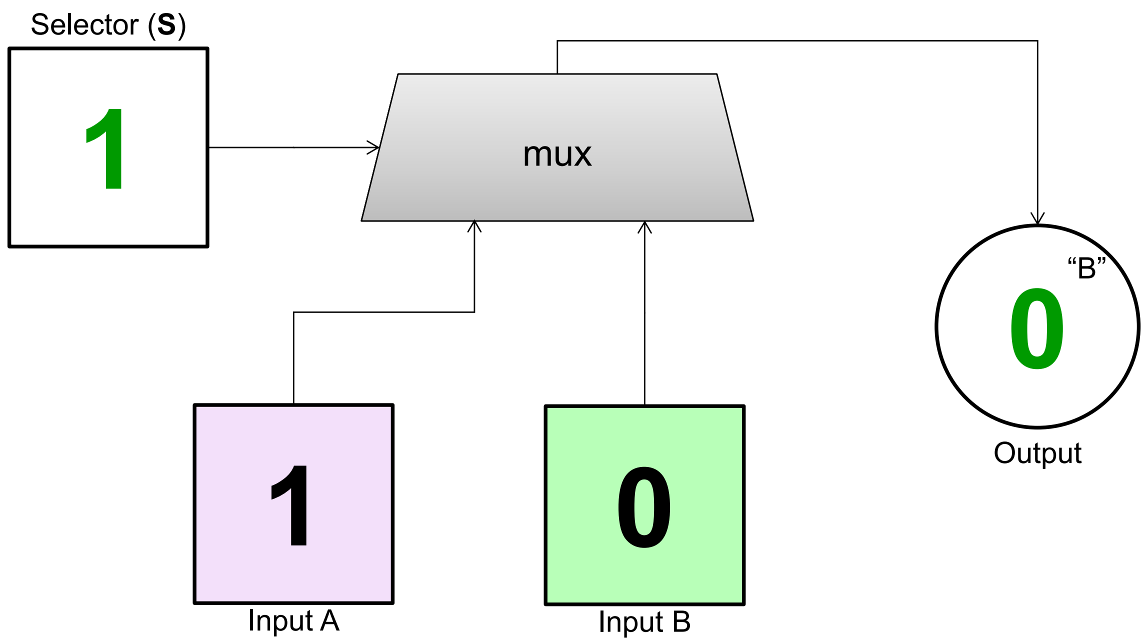
Introducing the Multiplexer ("mux")



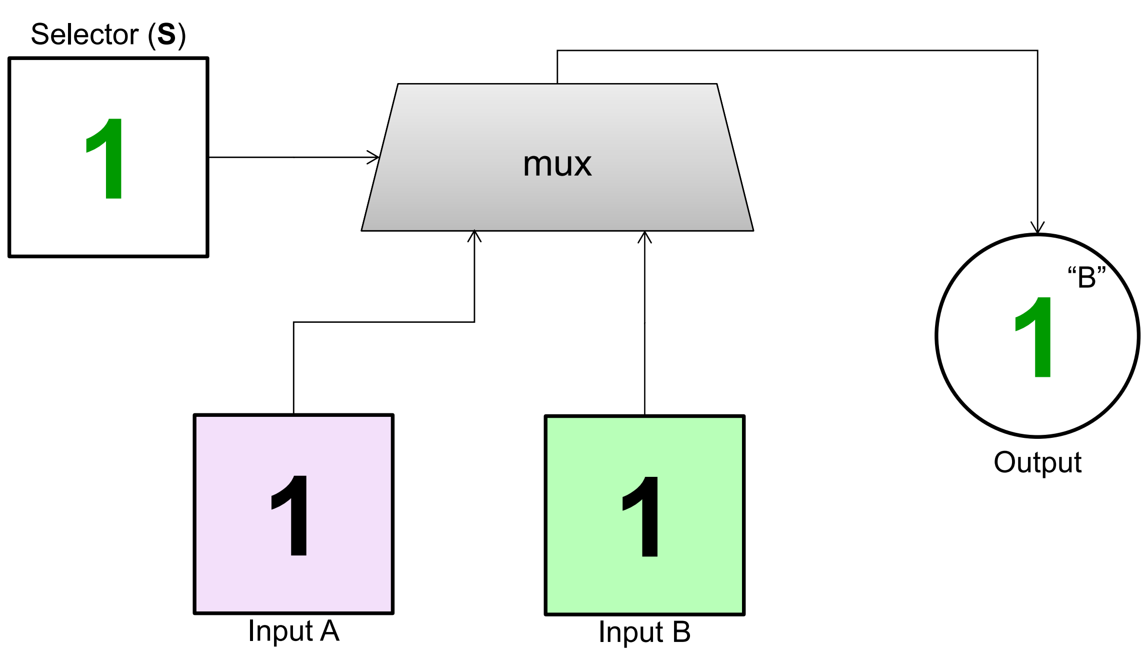
Introducing the Multiplexer ("mux")



Introducing the Multiplexer ("mux")



Introducing the Multiplexer ("mux")



Let's Make a Useful Circuit

- Pick between 2 inputs (called 2-to-1 MUX)
  - Short for multiplexor

- What might we do first?
  - Make a truth table?
    - S is selector:
      - S=0, pick A
      - S=1, pick B

- Next: how to get formula?

  Sum-of-products

A	B	S	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Sum of Products

- Find the rows where the output is 1
- Write a formula that exactly specifies each row
- OR these all together
  - Possible ways to get 1.

	A	B	S	Output
	0	0	0	0
	0	0	1	0
	0	1	0	0
!A & B & S →	**0**	**1**	**1**	**1**
A & !B & !S →	**1**	**0**	**0**	**1**
	1	0	1	0
A & B & !S →	**1**	**1**	**0**	**1**
A & B & S →	**1**	**1**	**1**	**1**

Let's Make a Useful Circuit

- **Sum-of-products:**

```
(!A & B & S) |
(A & !B & !S) |
(A & B & !S ) |
(A & B & S)
```

- This is long, though. Need to **simplify**.

A	B	S	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Simplifying The Formula

- Simplifying this formula:

```
(!A & B & S) |
(A & !B & !S) |
(A & B & !S ) |
(A & B & S)
```

B doesn't matter

Simplifying The Formula

- Simplifying this formula:

```
(!A & B & S) |
(A & !S) |
(A & B & S)
```

A doesn't matter

Simplifying The Formula

- Simplifying this formula:

```
(A & !S) |
(B & S)
```

Let's Make a Useful Circuit

- Simplified formula:

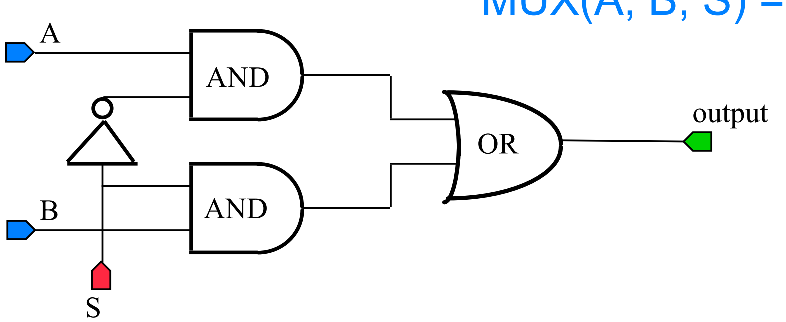
```
(A & !S) |
(B &  S)
```

A	B	S	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

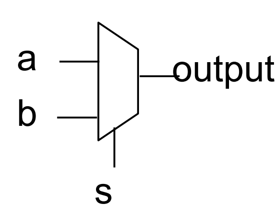
Circuit Example: 2x1 MUX

Draw it in gates:

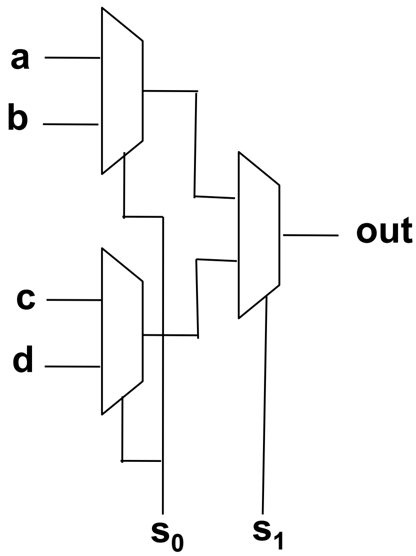
MUX(A, B, S) = (A & !S) | (B & S)

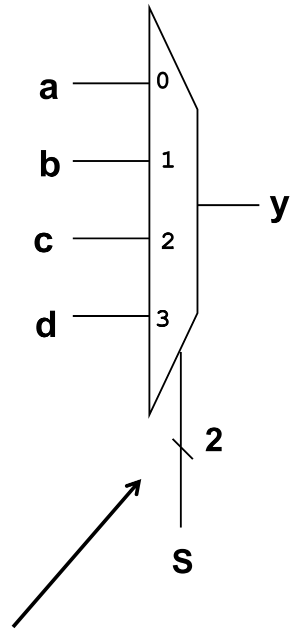


So common, we give it its own symbol:



Example 4x1 MUX





The / 2 on the wire means "2 bits"

Boolean Function Simplification

- Boolean expressions can be simplified by using the following rules (bitwise logical):
  - `A & A = A` `A | A = A`
  - `A &  0 = 0` `A | 0 = A`
  - `A & 1 = A` `A | 1 = 1`
  - `A & !A = 0` `A | !A = 1`

  - `!!A  = A`

  - & and | are both commutative and associative
  - & and | can be distributed: `A & (B | C) = (A & B) | (A & C)`
  - & and | can be subsumed: `A | (A & B) = A`

  - Can typically just let synthesis tools do this dirty work, but good to know

DeMorgan’s Laws

- De Morgan's laws

```
!(A & B) = (!A) | (!B)

!(A | B) = (!A) & (!B)
```


Wrap Up

- Combinatorial Logic
  - Putting gates together
  - Sum-of-products
  - Simplification
  - Muxes