

ENUMERATING AND COMPOSING GATES



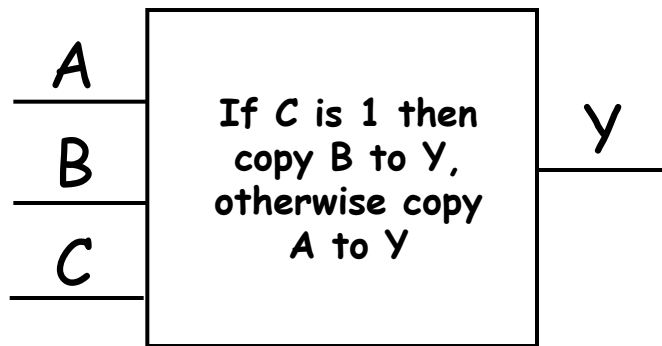
- Combinational logic as/is truth tables
- Composing gates
- What gates do we have?
- What gates do we need?
- Making gates from others
- A systematic approach for implementing combinational logic

Midterm results next Tuesday
PS#3 is posted

NOW CAN WE DESIGN LARGER SYSTEMS



We need to start somewhere -
usually with a functional specification



Argh... I'm tired of word games



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

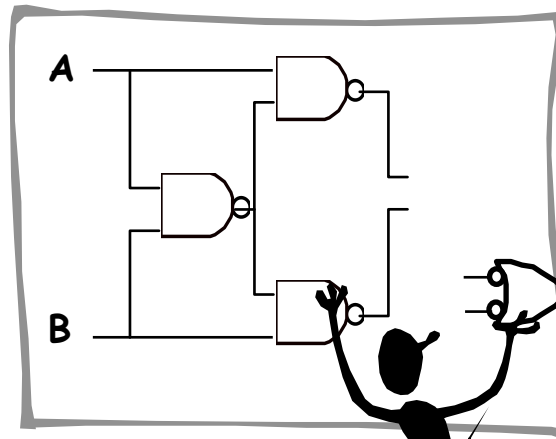
If you are like most pragmatists you'd rather be given a table or formula than solve a puzzle to understand a function. The fact is, **every combinational function can be expressed as a table.**

"Truth tables" are a concise description of the combinational system's function, where an output is specified for ***every*** input combination.



TRUTH TABLES TO GATES?

We want to build a computer!

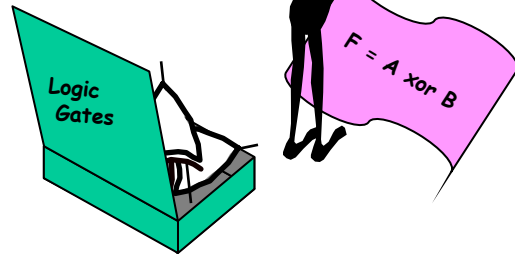


So far we know how to build a few CMOS gates using MOSFET transistors

(NAND, NOR, INVERTER)

But we are missing AND, OR, and XOR

What gates can we build using CMOS?





WHAT GATES CAN WE BUILD?

Recall, we need to design our gates using a pull-up network of P-FETs and a pull-down network of N-FETs.

What gates can we
- build?
- define?

AND		OR		NAND		NOR	
AB	Y	AB	Y	AB	Y	AB	Y
00	0	00	0	00	1	00	1
01	0	01	1	01	1	01	0
10	0	10	1	10	1	10	0
11	1	11	1	11	0	11	0

Let's start by
considering only
2-input gates.

How many possible 2-input gates are there?

KEY IDEA: As many as there are 2-input truth tables.

2-inputs $\rightarrow 2^2 = 4$ rows, each with an output

4-outputs $\rightarrow 2^4 = 16$ possible functions



ALL THE GATES

There are only 16 possible 2-input gates... Let's examine all of them. Some we already know, others are just silly.

I N P U T	Z	A	A	B	X	N	X	N	A	N	B	A	O		
AB	O	D	B	A	A	B	R	R	R	'B'	B	'A'	A	D	E
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1

How many of these gates can be implemented using a single CMOS gate?



N-FETs can only pull the output to "0", and only if one or more of their inputs is a "1".

P-FETs can only pull the output to "1", and only if one or more of their inputs is a "0".

Do we really need all of these gates?

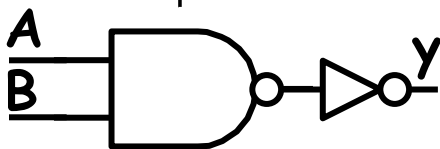
Nope! Once we realize that we can describe all of them using just AND, OR, and NOT

COMPOSING GATES TO BUILD OTHERS

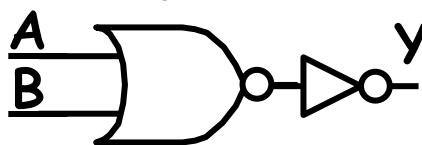


Let's start with a couple of basics, AND and OR. Each can be constructed using a pair of CMOS gates, AND is just NAND with an inverter, and OR is just NOR with an inverted output.

AND	
AB	Y
00	0
01	0
10	0
11	1



OR	
AB	Y
00	0
01	1
10	1
11	1



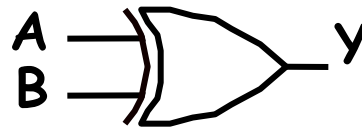
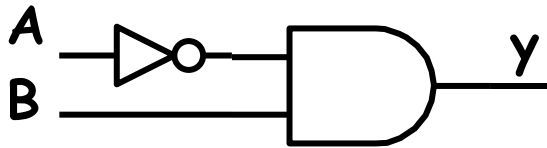
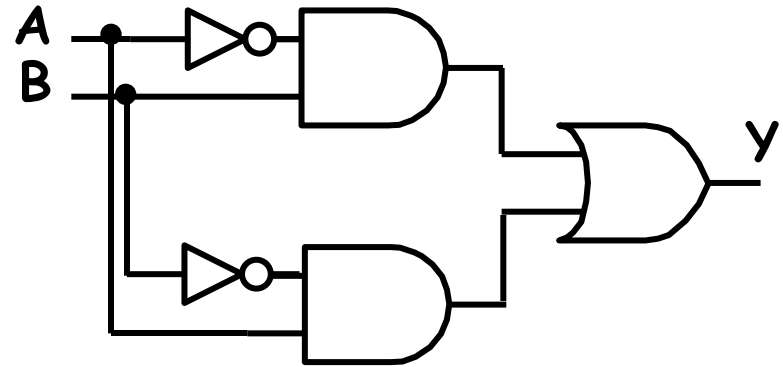
These two gates are particularly important. Using them will allow us to develop a systematic approach for constructing any combinational function.



COMPOSING ARBITRARY GATES

B > A		
AB	Y	
00	0	
01	1	
10	0	
11	0	

XOR		
AB	Y	
00	0	
01	1	
10	1	
11	0	



How many different gates do we really need?

We can always do it with 3 different types of gates (AND, OR, INVERT), and sometimes with 2, but, can we use fewer?

The TRICK is to OR the ANDs of all input combinations that generate an output of "1". You don't need the OR gate if only one input combination results in a "1".



You need Inverters to handle input combinations involving "0"s, ANDs, and ORs.

ONE WILL DO!

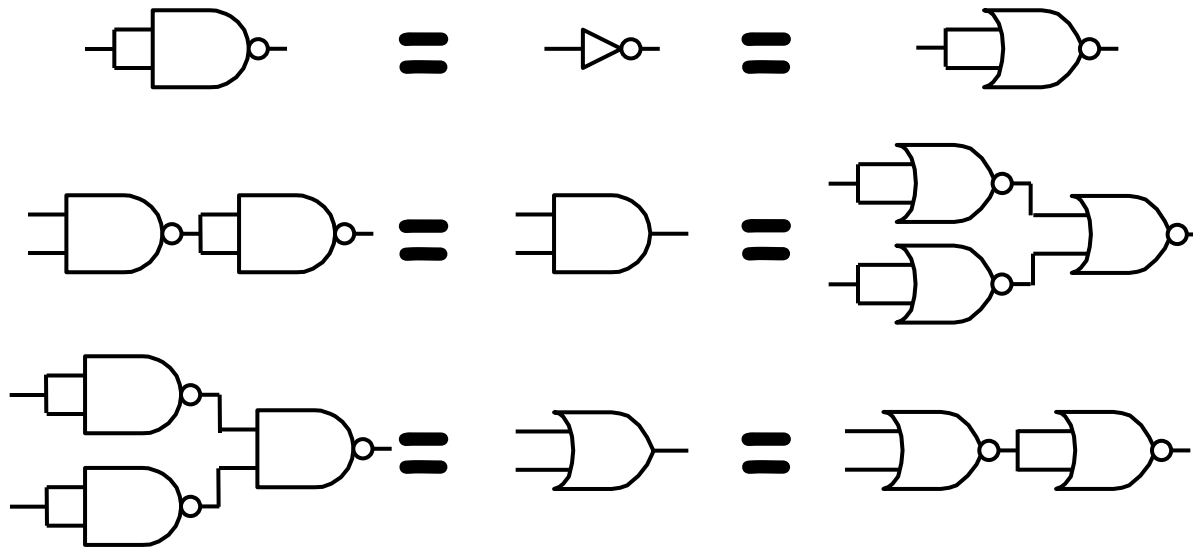
NANDs and NORs are UNIVERSAL!

A UNIVERSAL gate is one that can be used to implement *ANY* COMBINATIONAL FUNCTION. There are many UNIVERSAL gates, but not all gates are UNIVERSAL



Q: What is a COMBINATIONAL FUNCTION?

A: Any function that can be written as a truth table.

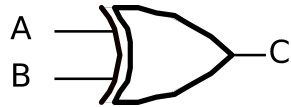


Ah!, but what if we want more than 2-inputs?



STUPID GATE TRICKS

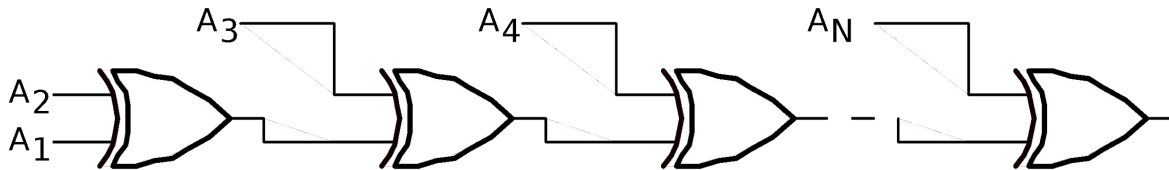
Suppose we have some 2-input XOR gates:



$$t_{pd} = 1 \text{ ns}$$

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

And we want an N-input XOR:



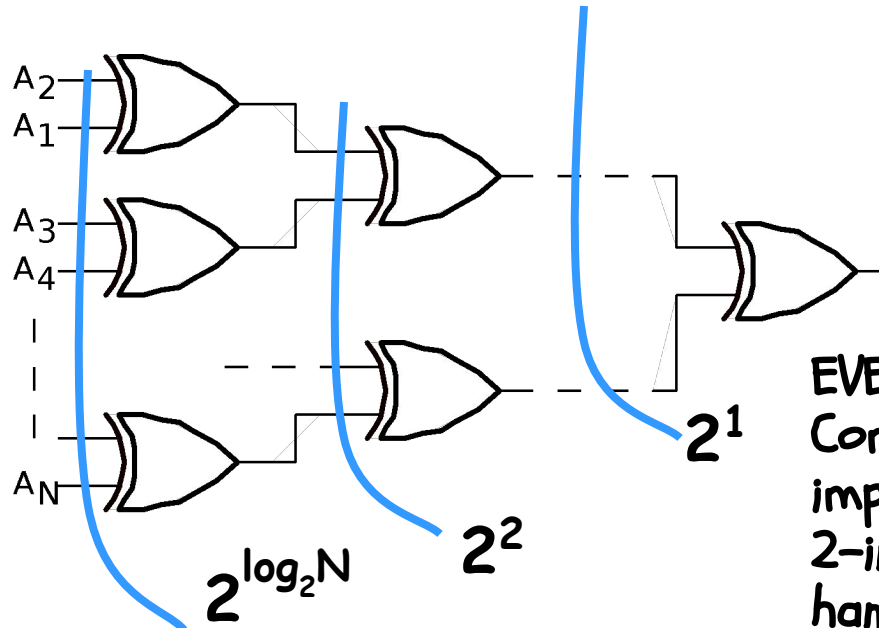
output = 1
iff number
of "1"s input is
ODD ("PARITY")

$$t_{pd} = N \text{ ns} \text{ -- WORST CASE.}$$

Can we compute an N-input XOR faster?



I THINK THAT I SHALL NEVER SEE A GATE LOVELY AS A ...



EVERY N-Input
Combinational function be
implemented using only
2-input gates? But, it's
handy to have gates with
more than 2-inputs when
needed.

N-input TREE has $O(\log N)$ levels...

Signal propagation takes $O(\log N)$ gate delays.

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 get to go home!



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

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

- 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



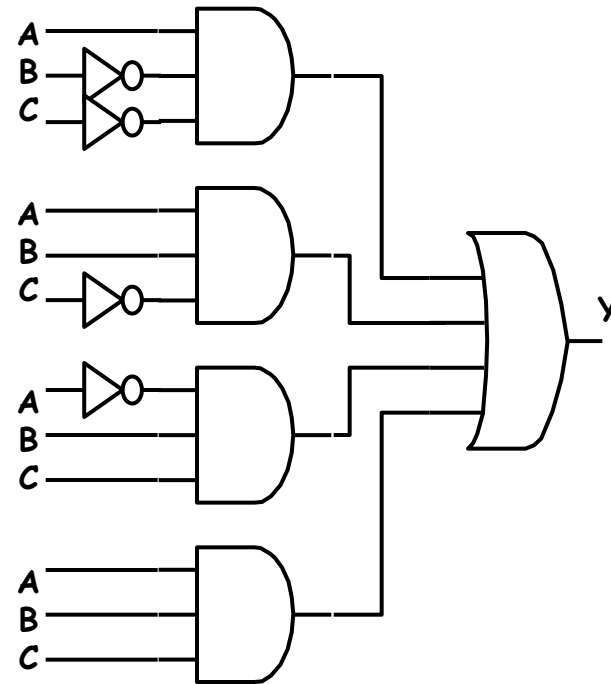
STRAIGHTFORWARD SYNTHESIS

We can implement

SUM-OF-PRODUCTS

with just 3 levels of logic.

INVERTERS/AND/OR

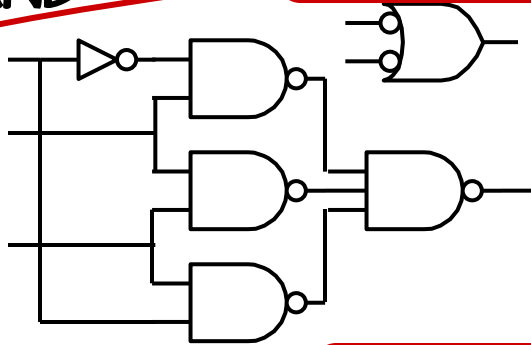




OTHER USEFUL GATE COMBINATIONS

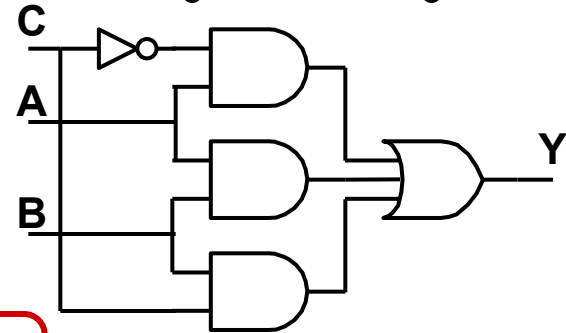
NAND-NAND

$$\!(A \& \& B) = \!A \parallel \!B$$



≡

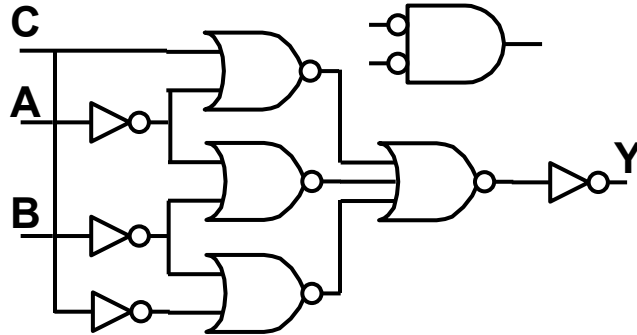
"Pushing and Cancelling Bubbles"



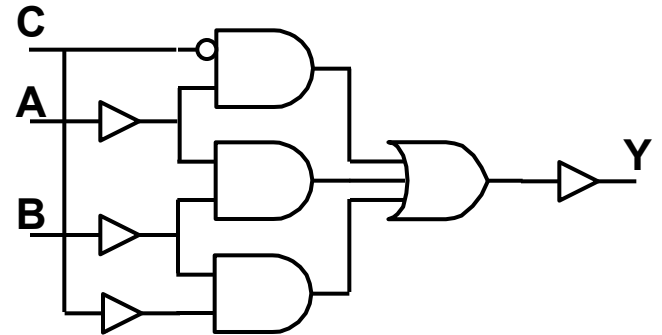
DeMorgan's Laws

NOR-NOR

$$\!(A \parallel B) = \!A \& \& \!B$$



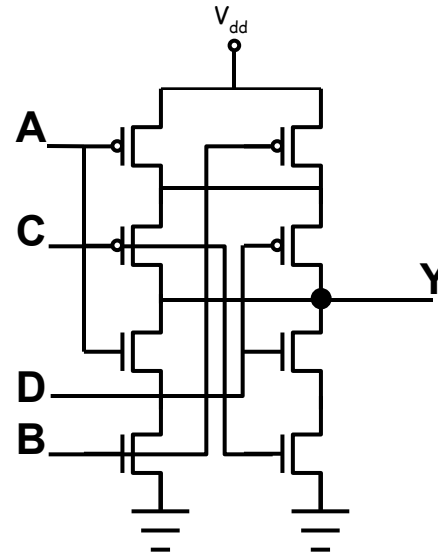
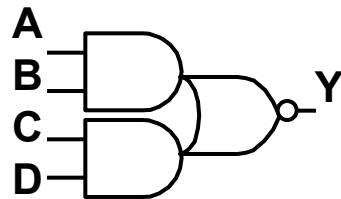
≡



OTHER USEFUL CMOS GATES

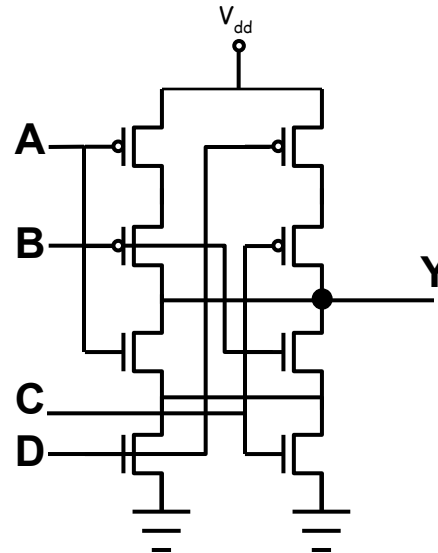
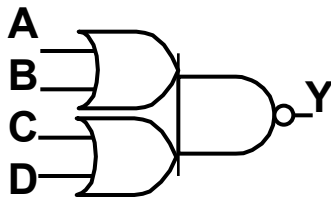


AOI (AND-OR-INVERT)



AOI and OAI structures can be realized as a single CMOS gate. However, their function is equivalent to 3 levels of logic.

OAI (OR-AND-INVERT)



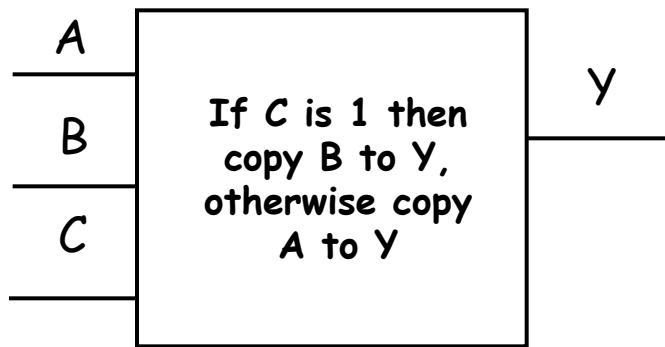
An OAI's DeMorgan equivalent is usually easier to think about.





AN INTERESTING 3-INPUT GATE

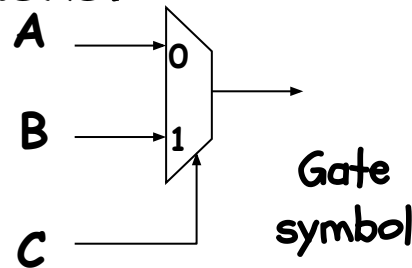
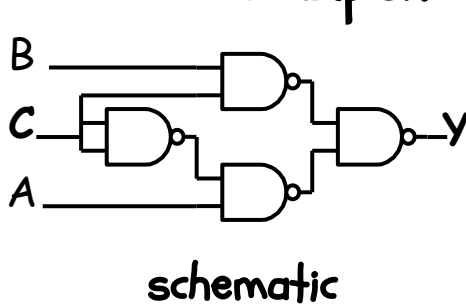
Based on C, select the A or B input to be copied to Y.



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

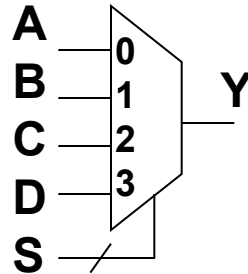
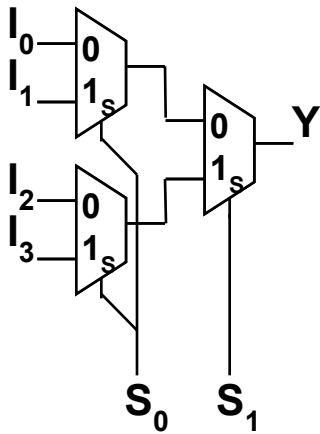
2-input Multiplexer



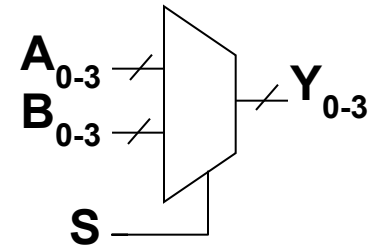
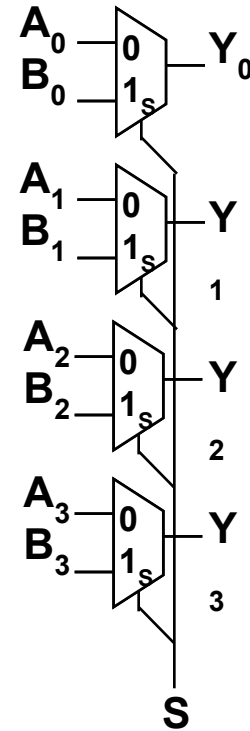
MUX COMPOSITIONS AND SHORTCUTS



A 4-input Mux
(implemented as a tree)



A 4-bit wide 2-input Mux

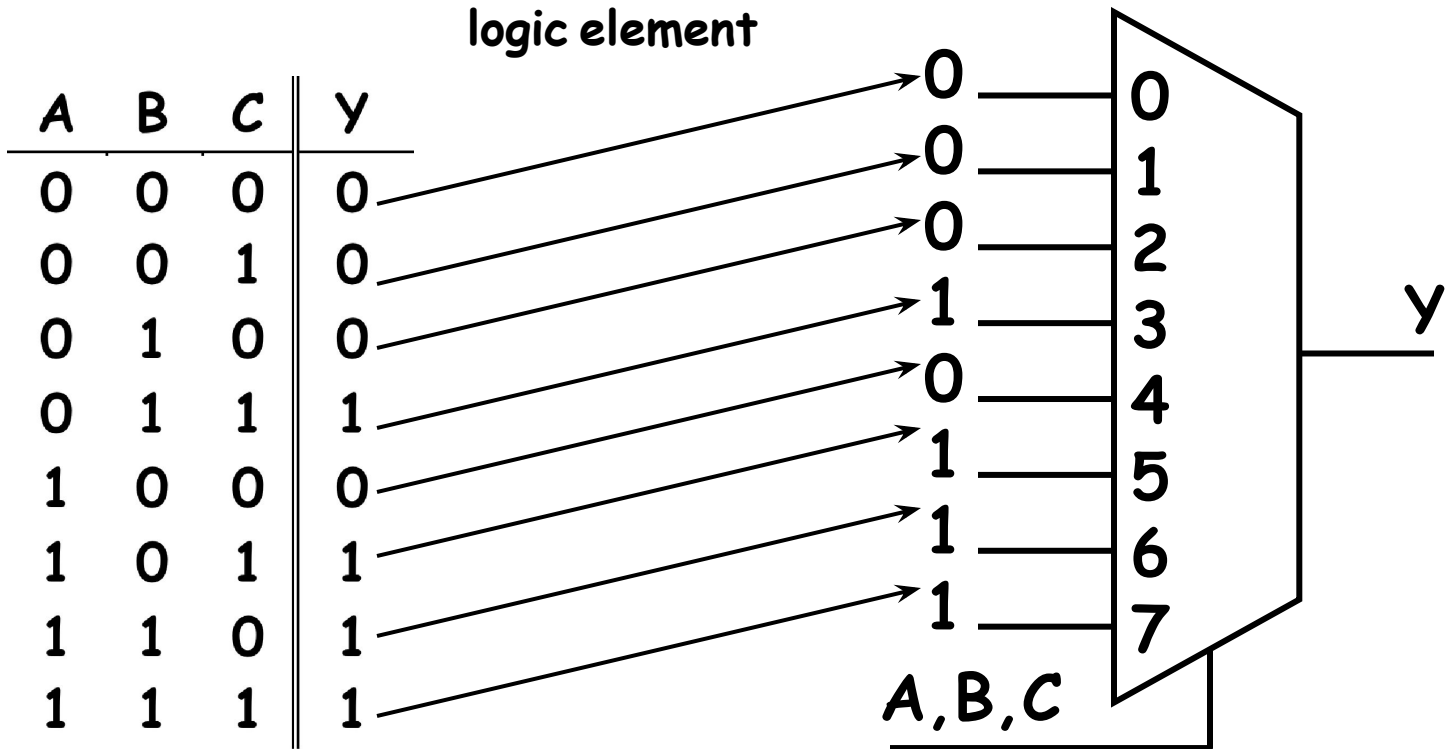




MUX FUNCTION SYNTHESIS

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

Mux Logic: An example "configurable" logic element

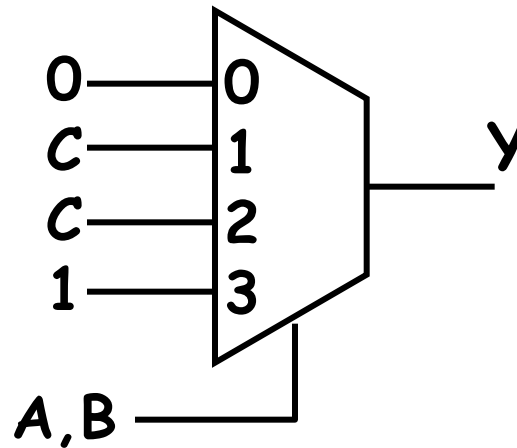
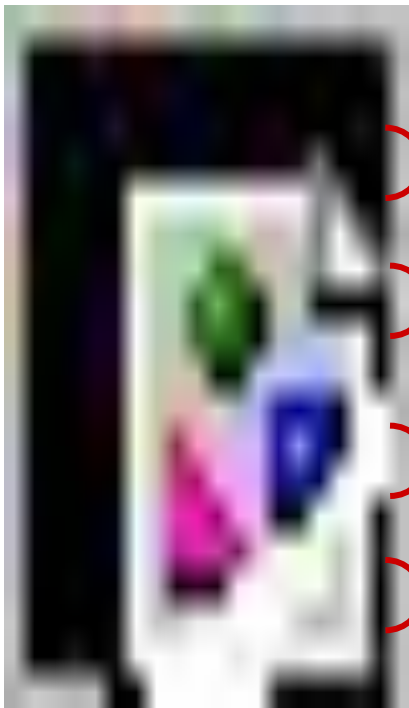




MUX LOGIC TRICKS

We can apply certain optimizations to MUX Function synthesis

Desired Logic Function



- Largely by inspection or exhaustive search



- N-input gate with an N-1 input MUX & one inverter



NEXT TIME

Binary Circuits that:

ADD

SUBTRACT

SHIFT

