

CS/COE0447: Computer Organization and Assembly Language

Logic Design Introduction (Brief?)

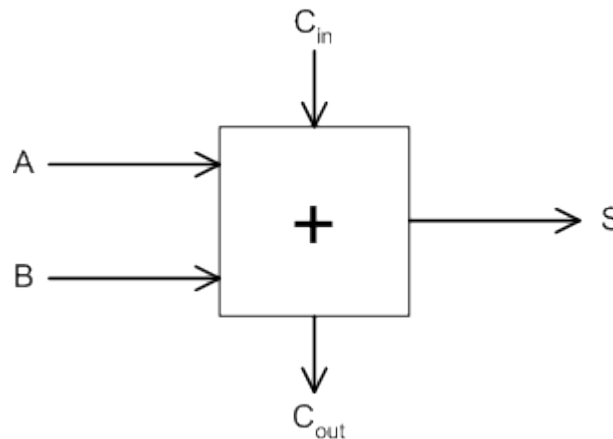
Appendix C: The Basics of Logic Design

**modified by Bruce Childers
Sangyeun Cho**

**Dept. of Computer Science
University of Pittsburgh**

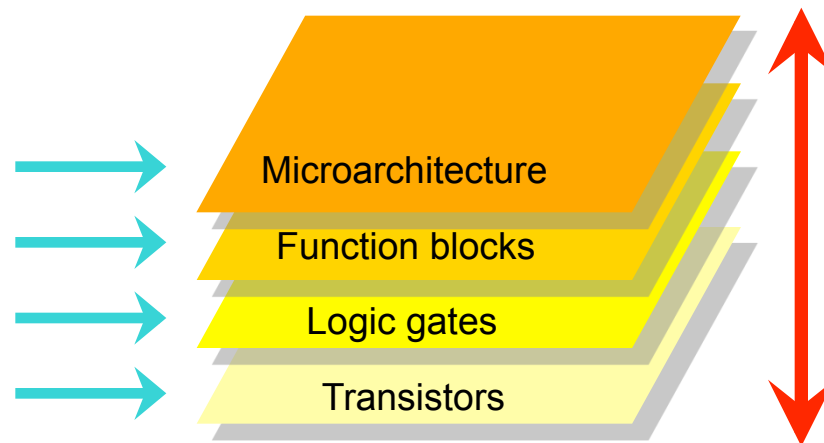
Logic design?

- Digital hardware is implemented by way of *logic design*
- Digital circuits process and produce two discrete values: 0 and 1
- Example: 1-bit full adder (FA)



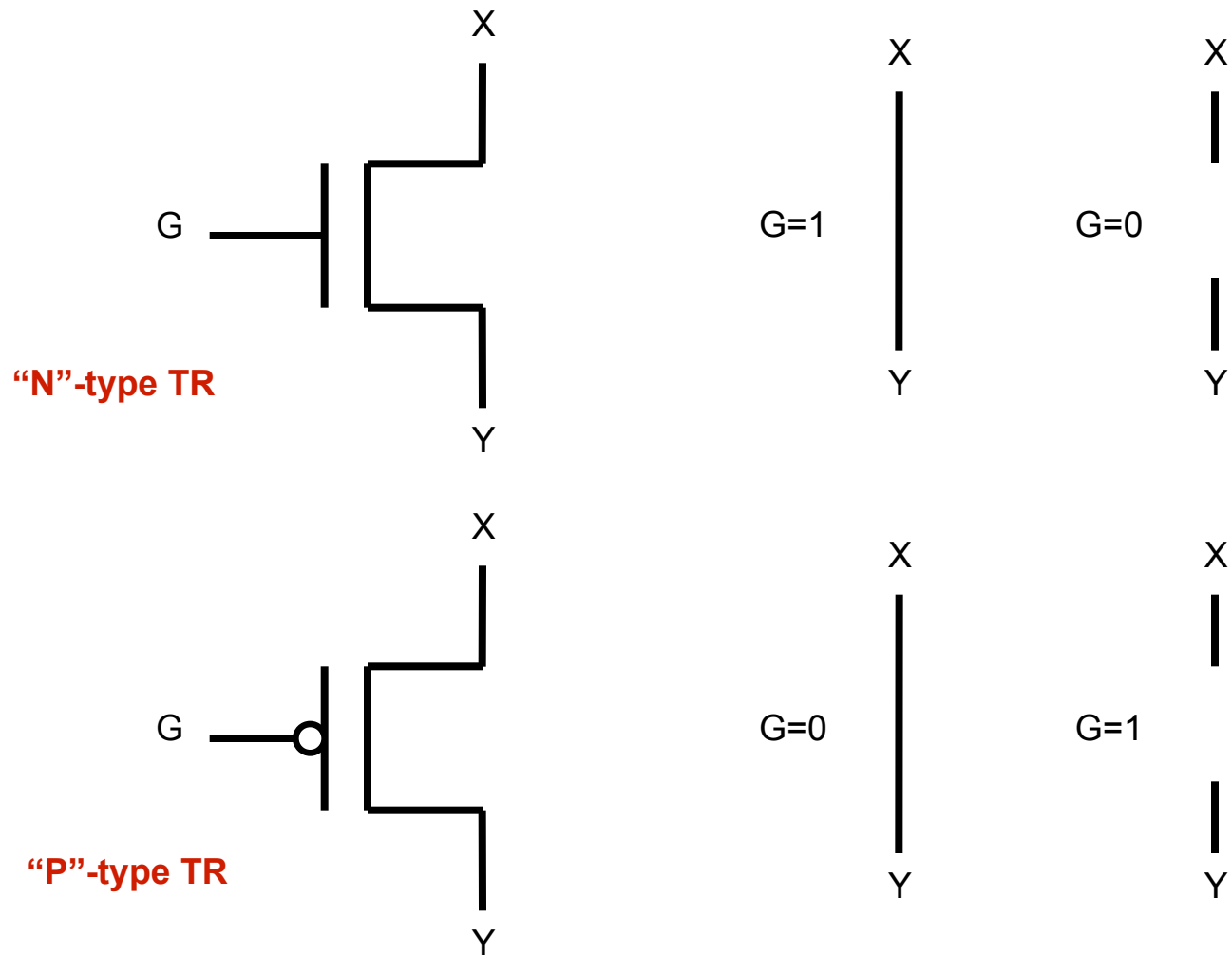
Layered design approach

- Logic design is done using **logic gates**
- Often we design a desired hardware function using high-level languages (HDLs) and somewhat higher level than logic gates
- Two approaches in design
 - Top down
 - Bottom up

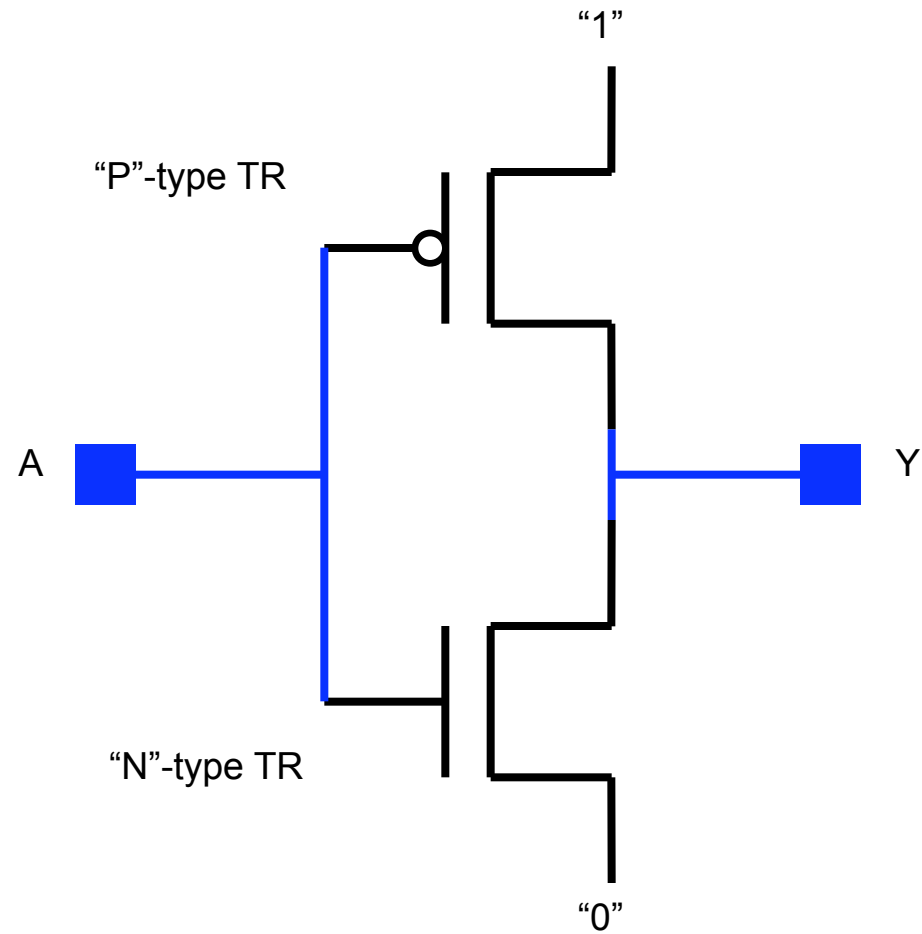


We'll do logic bottom up

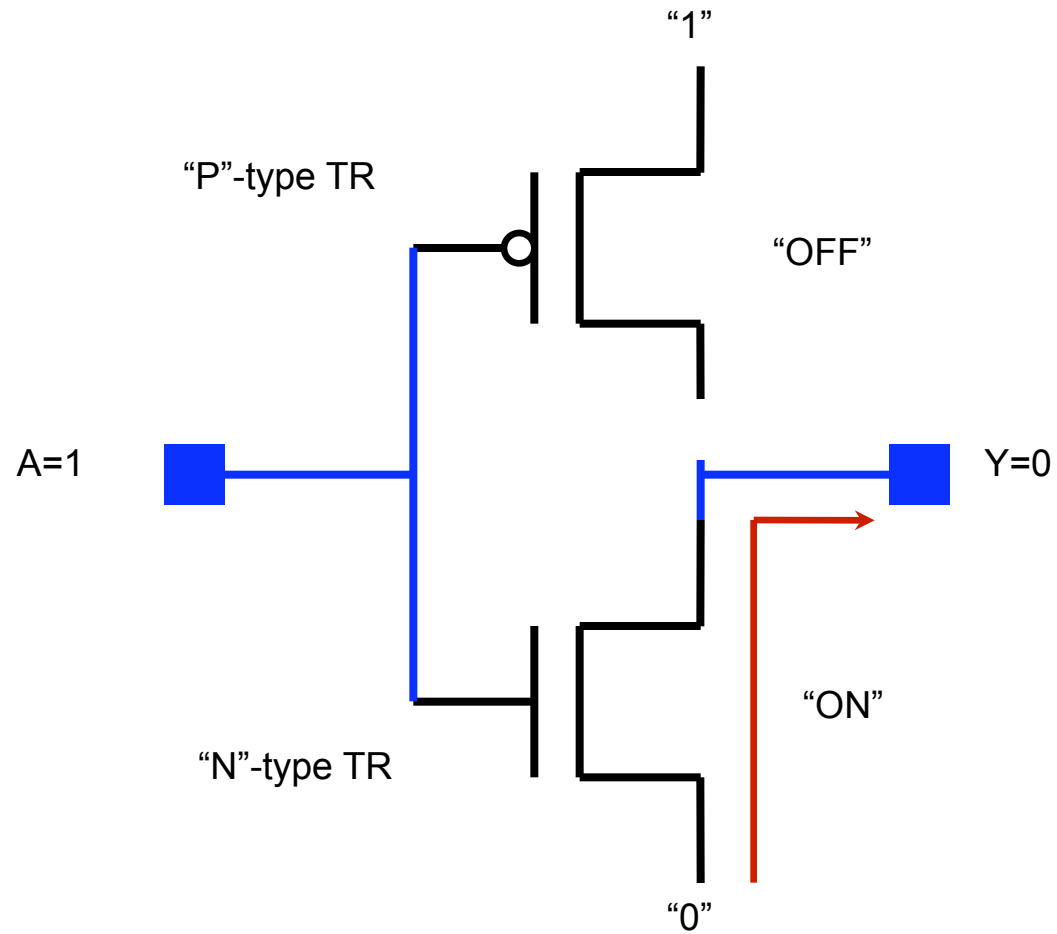
Transistor as a switch



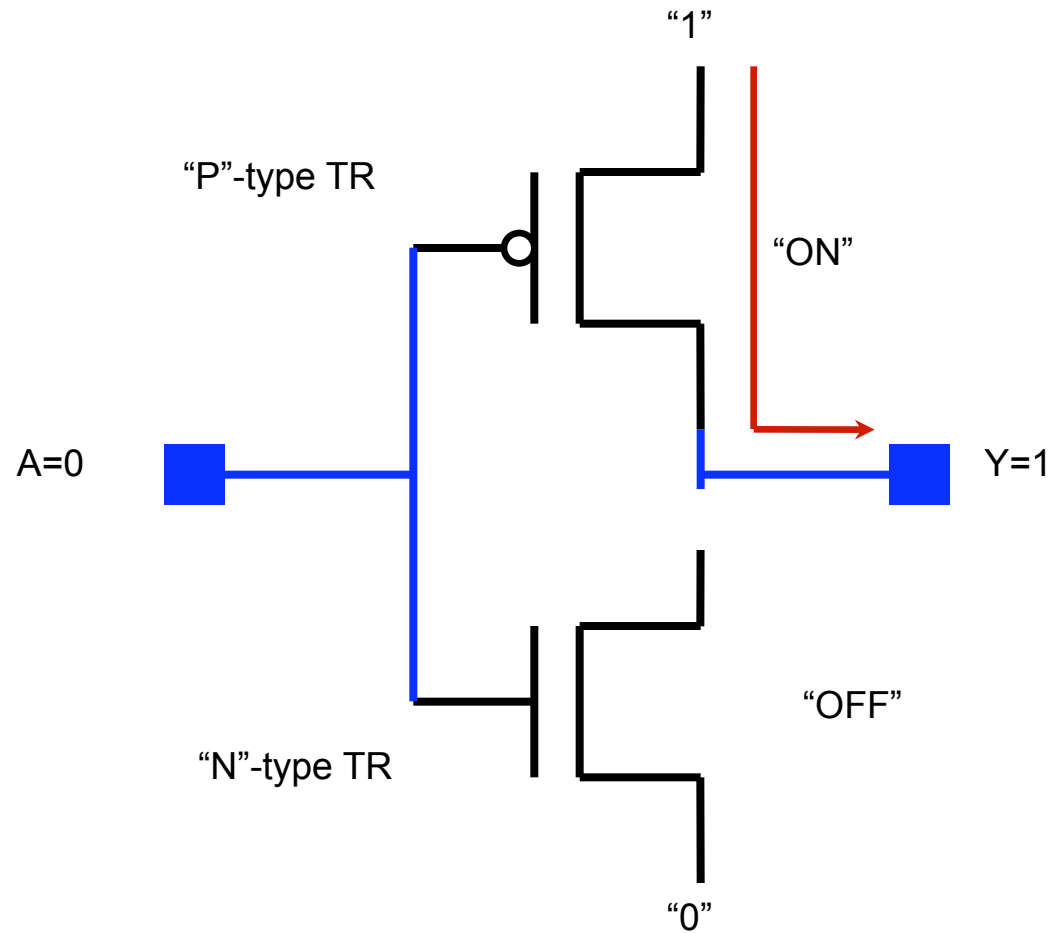
An inverter



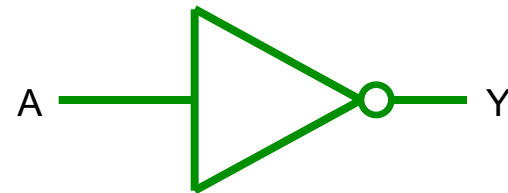
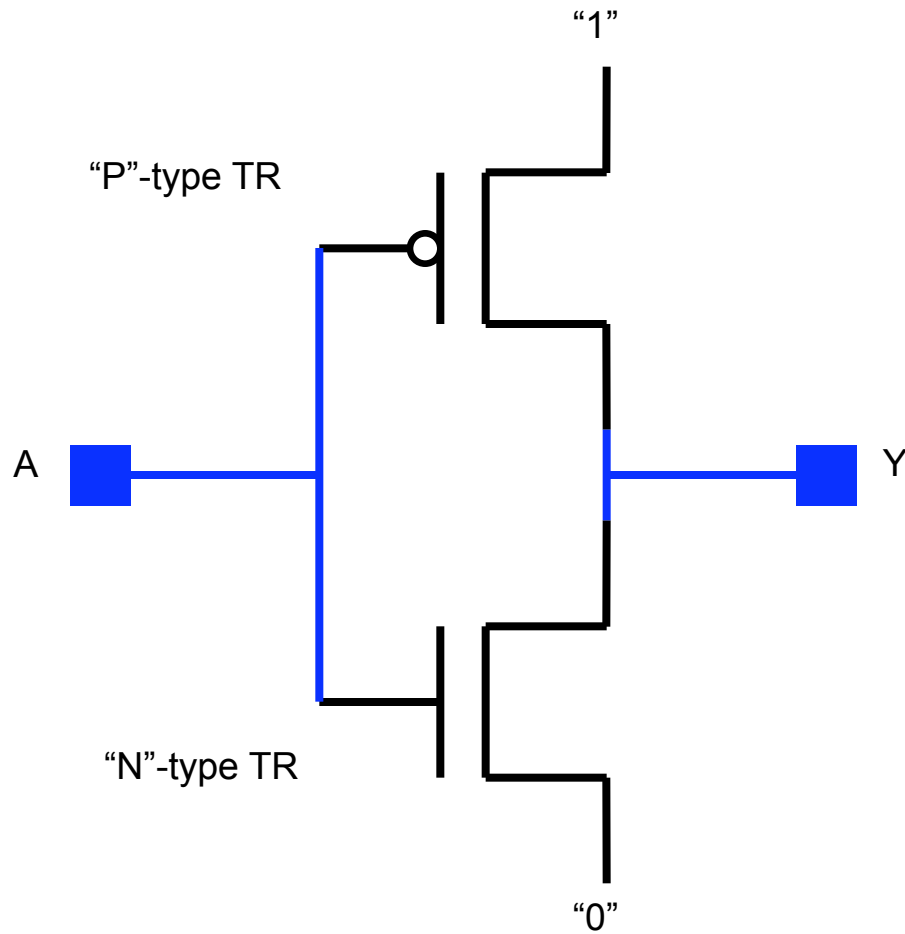
When A = 1



When $A = 0$

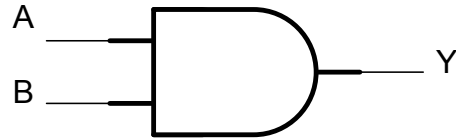


Abstraction



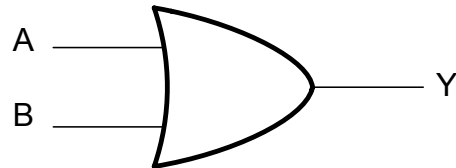
Logic gates

2-input AND



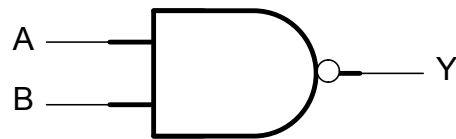
$$Y = A \& B$$

2-input OR



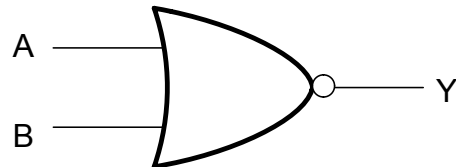
$$Y = A | B$$

2-input NAND



$$Y = \sim(A \& B)$$

2-input NOR



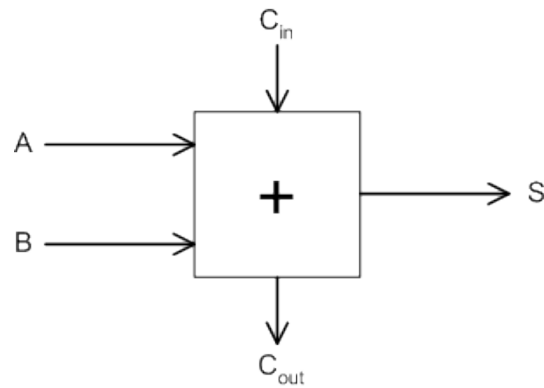
$$Y = \sim(A | B)$$

Describing a function

- $\text{Output}_A = F(\text{Input}_0, \text{Input}_1, \dots, \text{Input}_{N-1})$
- $\text{Output}_B = F'(\text{Input}_0, \text{Input}_1, \dots, \text{Input}_{N-1})$
- $\text{Output}_C = F''(\text{Input}_0, \text{Input}_1, \dots, \text{Input}_{N-1})$
- ...

- Methods
 - Truth table
 - Sum of products
 - Product of sums

Truth table



Input			Output	
A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum of products

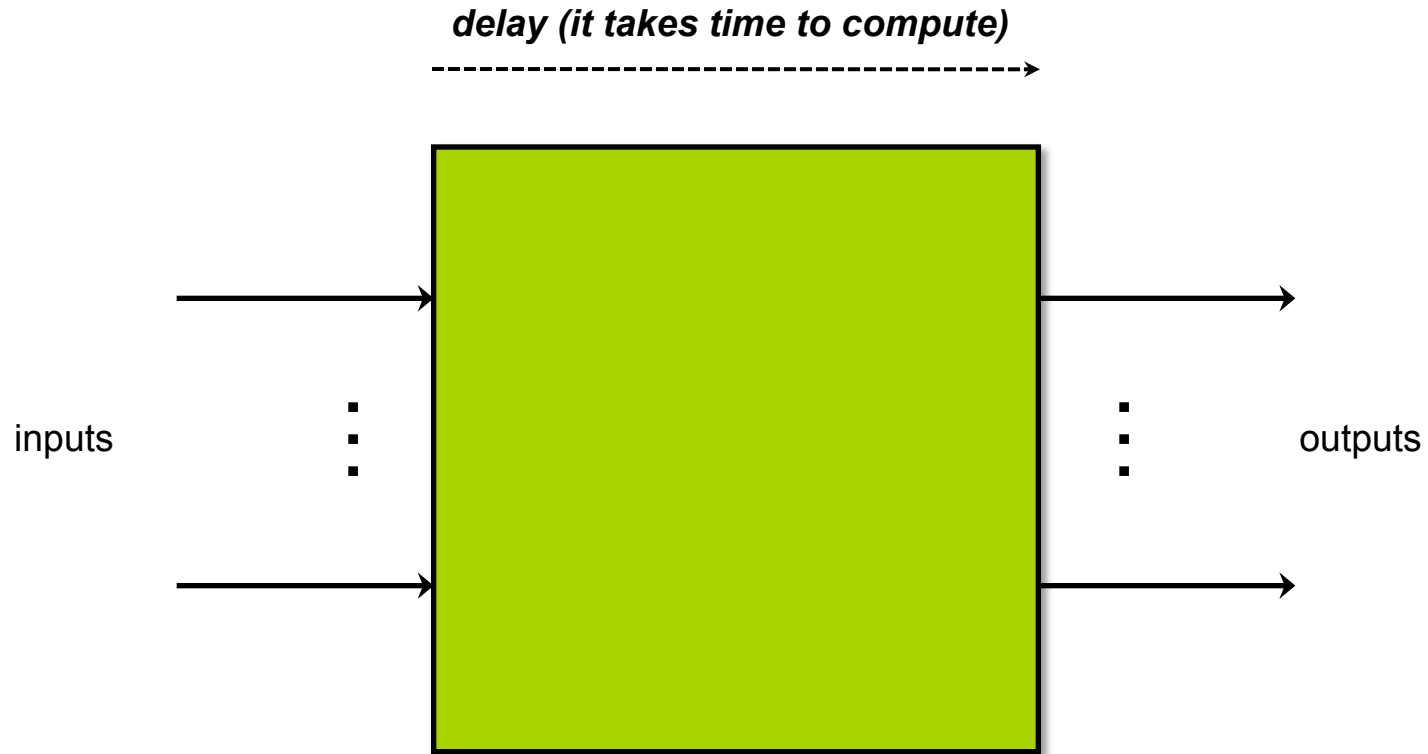
Input			Output	
A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- $S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$
- $C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$

Combinational vs. sequential logic

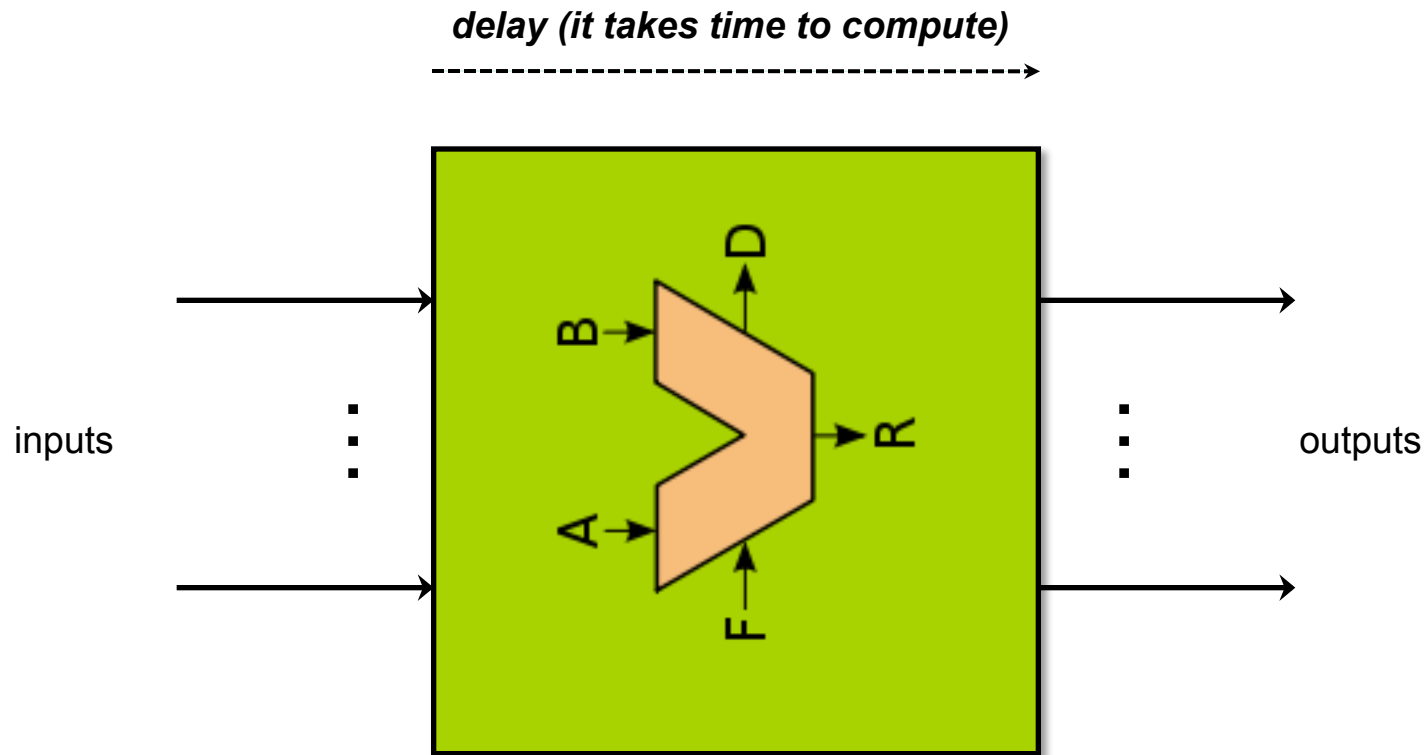
- Combinational logic = **function**
 - A function whose outputs are dependent only on the current inputs
 - As soon as inputs are known, outputs can be determined
- Sequential logic = **combinational logic + memory**
 - Some memory elements (i.e., “state”)
 - Outputs are dependent on the current state and the current inputs
 - Next state is dependent on the current state and the current inputs

Combinational logic



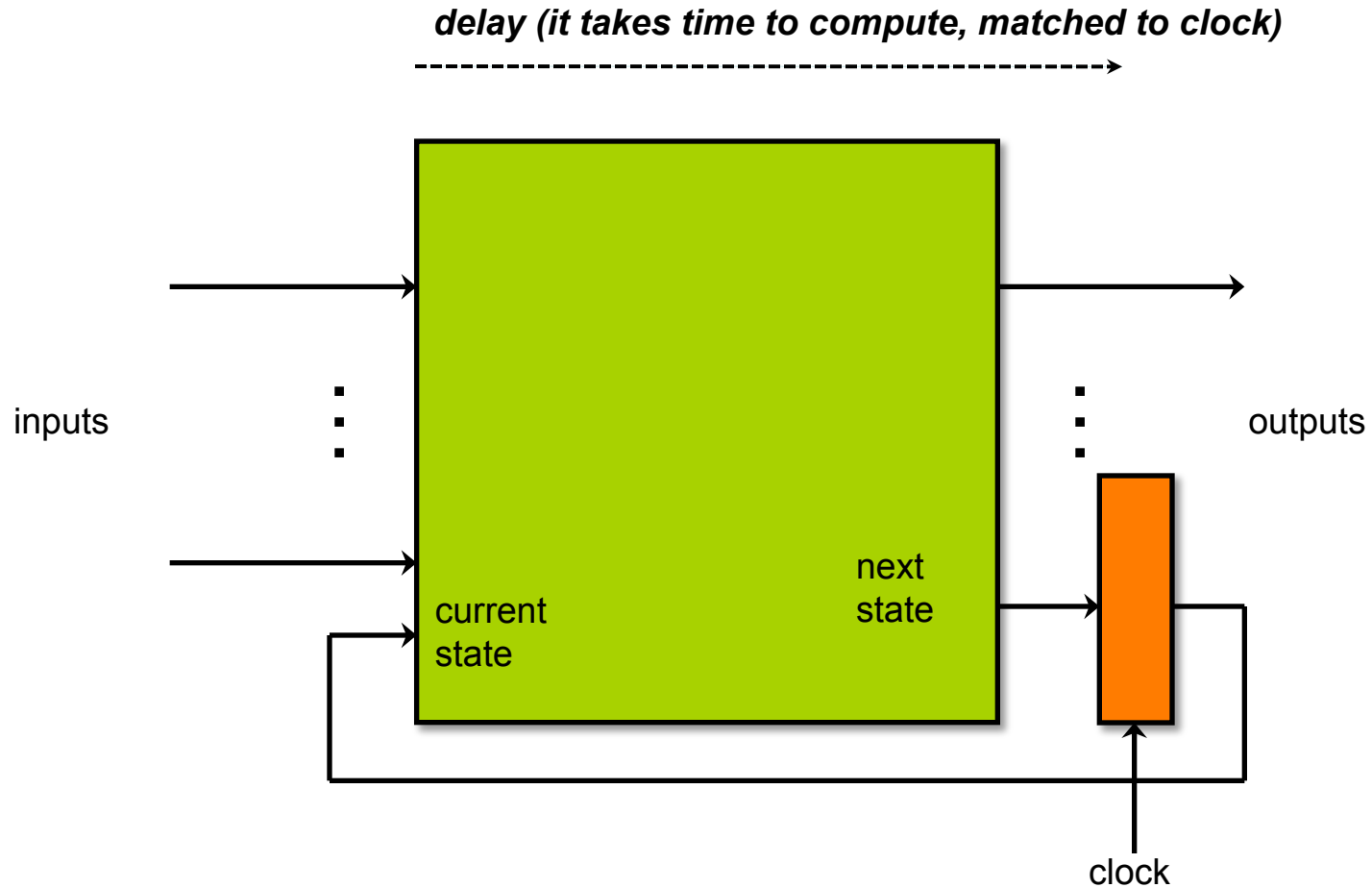
Outputs are uniquely determined by the inputs at any moment

Combinational logic



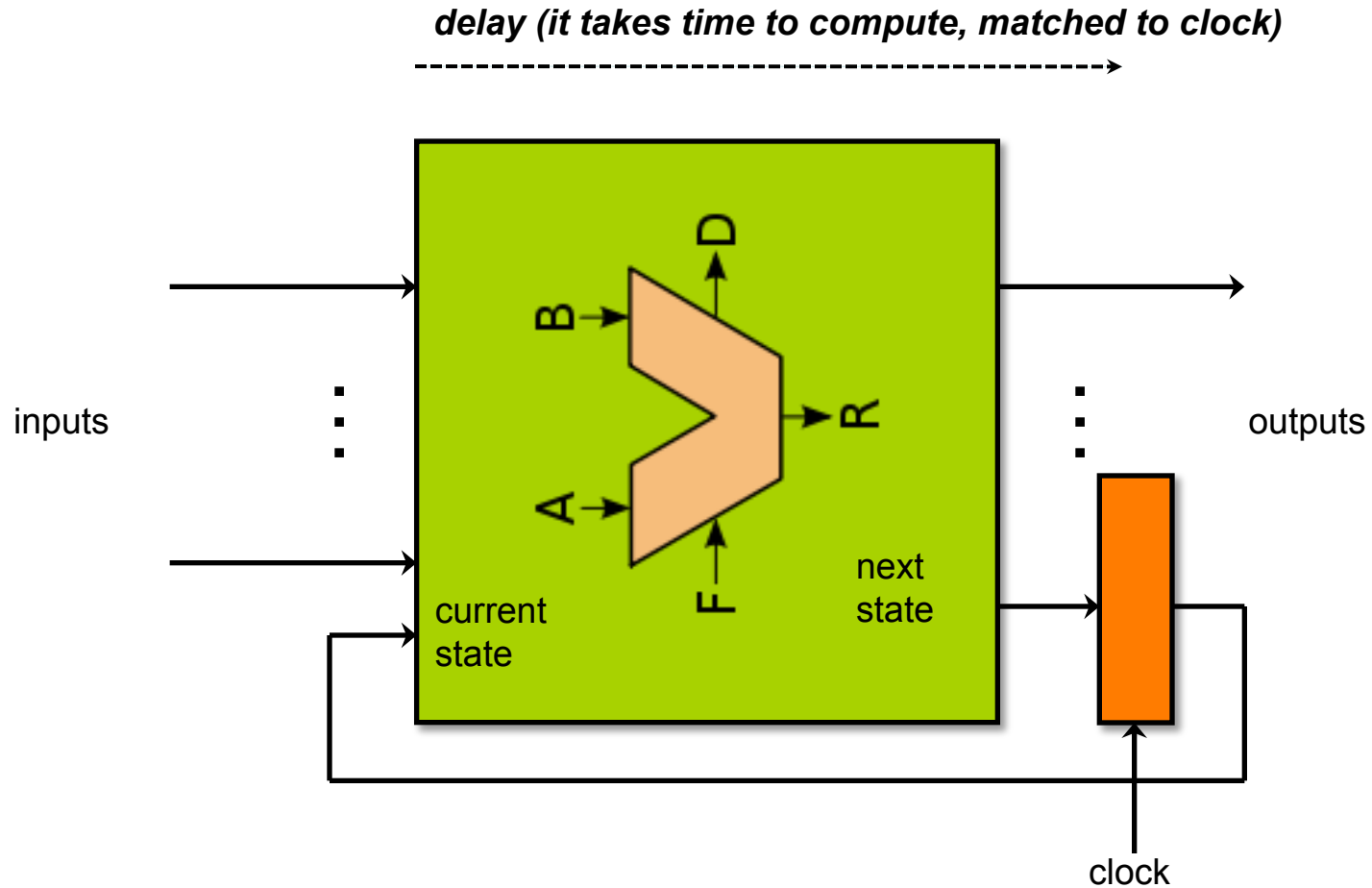
Outputs are uniquely determined by the inputs at any moment

Sequential logic



Outputs are determined by current & past inputs (past is “state”)

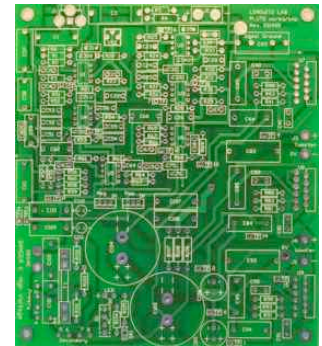
Sequential logic



Outputs are determined by current & past inputs (past is “state”)

Combinational logic

- Any combinational logic can be implemented using sum of products (OR-AND) or product of sums (AND-OR)
- Input-output relationship can be defined in a **truth table** format
- From truth table, derive each **output function**
- And then we can derive a circuit!! Let's try it!
- Boolean expressions can be further manipulated (e.g., to reduce cost) using various Boolean algebraic rules



Boolean algebra

- Boole, George (1815~1864): mathematician and philosopher; inventor of Boolean Algebra, the basis of all computer arithmetic
- Binary values: $\{0, 1\}$
- Two binary operations: AND (\times/\cdot), OR ($+$)
- One unary operation: NOT (\sim)

Boolean algebra

- Binary operations: AND (\cdot), OR ($+$)
 - Idempotent
 - ♦ $a \cdot a = a + a = a$
 - Commutative
 - ♦ $a \cdot b = b \cdot a$
 - ♦ $a + b = b + a$
 - Associative
 - ♦ $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
 - ♦ $a + (b + c) = (a + b) + c$
 - Distributive
 - ♦ $a \cdot (b + c) = a \cdot b + a \cdot c$
 - ♦ $a + (b \cdot c) = (a + b) \cdot (a + c)$

Boolean algebra

- De Morgan's laws

- $\sim(a \cdot b) = \sim a + \sim b$
- $\sim(a + b) = \sim a \cdot \sim b$

It is not true I ate the sandwich and the soup.

same as:

I didn't eat the sandwich or I didn't eat the soup.

- More...

- $a + (a \cdot b) = a$
- $a \cdot (a + b) = a$
- $\sim\sim a = a$
- $a + \sim a = 1$
- $a \cdot (\sim a) = 0$

It is not true that I went to the store or the library.

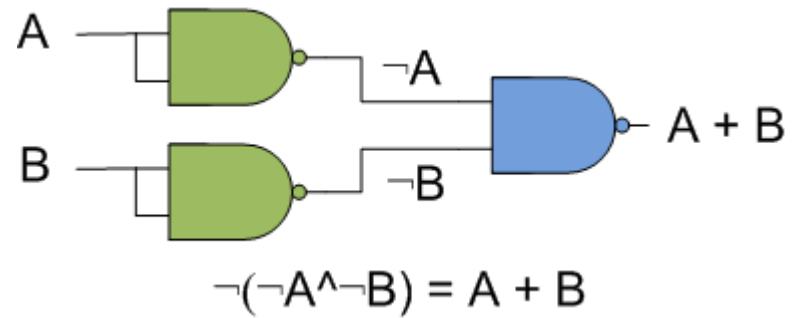
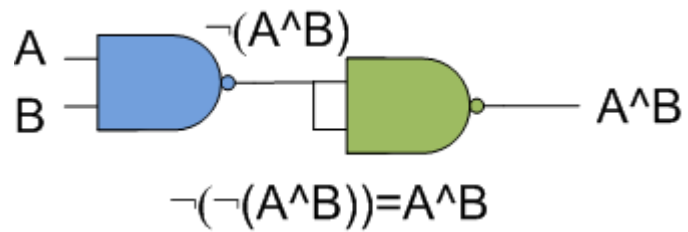
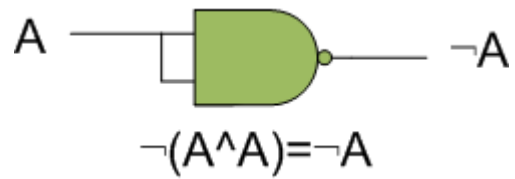
same as:

I didn't go to the store and I didn't go to the library.

Expressive power

- With AND/OR/NOT, we can express any function in Boolean algebra
 - Sum (+) of products (\cdot)
- What if we have NAND/NOR/NOT?
- What if we have NAND only?
- What if we have NOR only?

Using NAND only



Using NOR only (your turn)

- Can you do it?
- NOR is $\neg(A + B)$
 - I.e., We need to write NOT, AND, and OR in terms of NOR

NOT

$$= \neg(A + A)$$

$$= \neg A \wedge \neg A$$

$$= \neg A$$

AND

$$= \neg(\neg(A + A) + \neg(B + B))$$

$$= \neg(\neg A \wedge \neg A + \neg B \wedge \neg B)$$

$$= \neg(\neg A + \neg B)$$

$$= \neg(\neg A) \wedge \neg(\neg B)$$

$$= A \wedge B$$

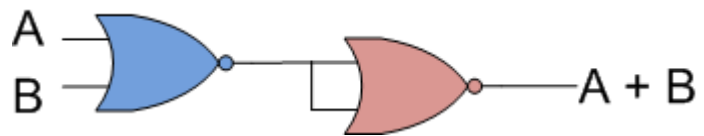
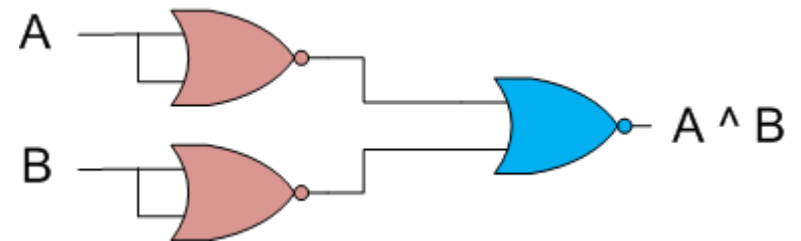
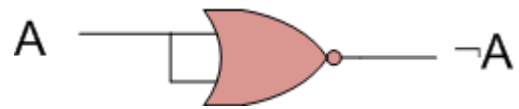
OR

$$= \neg(\neg(A + B) + \neg(A + B))$$

$$= (A + B) \wedge (A + B)$$

$$= A + B$$

Using NOR only (your turn)

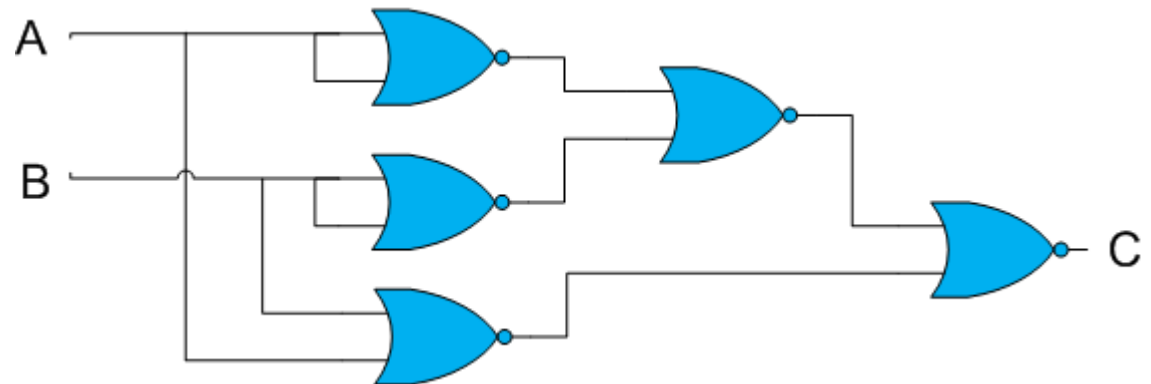
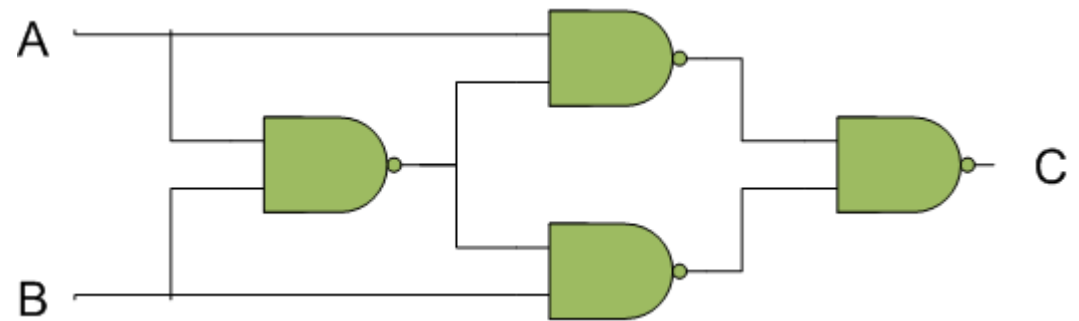
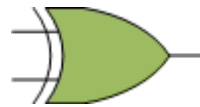


Now, it's really your turn....

- How about XOR?

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

$$C = A'B + AB'$$



Now, it's really your turn....

- How about XOR?

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

$$C = A'B + AB'$$

