



Digital Logic and System Design

3: Boolean Algebra and Logic Gates

COL215, I Semester 2022-2023

Venue: LHC 111

'E' Slot: Tue, Wed, Fri 10:00-11:00

Instructor: Preeti Ranjan Panda

panda@cse.iitd.ac.in

www.cse.iitd.ac.in/~panda/

Dept. of Computer Science & Engg., IIT Delhi

Processing with Binary Logic

- **Storage**: 0/1
- Processing: **Operations**
 - Which operation have we already used?
- Need an Algebra: **Boolean Algebra**
 - Set of **values**: {0, 1} or {true, false}
 - Define **operations**

Boolean Algebra

- Set of elements $B = \{0, 1\}$ and Operations **AND** (\bullet), **OR** ($+$)
- Postulates:
 - 1. Closure**
 - if $x, y \in B$, then $z = x + y \in B$
 - if $x, y \in B$, then $z = x \bullet y \in B$
 - 2. Identity**
 - 0 is identity element for $+$ ($0 + x = x$)
 - 1 is identity element for \bullet ($1 \bullet x = x$)
- Postulates...contd.:
 - 3. Commutativity**
 - $+$ is commutative [$x + y = y + x$]
 - \bullet is commutative [$x \bullet y = y \bullet x$]
 - 4. Distributivity**
 - $+$ is distributive over \bullet [$x \bullet (y + z) = x \bullet y + x \bullet z$]
 - \bullet is distributive over $+$ [$x + (y \bullet z) = (x + y) \bullet (x + z)$]
 - 5. Inverse (Complement)**
 - For every $x \in B$, $\exists x' \in B$ such that
 - $x + x' = 1$
 - $x \bullet x' = 0$

Rules for Boolean Algebra Operations

AND (\bullet) Operation

x	y	$x \bullet y$
0	0	0
0	1	0
1	0	0
1	1	1

OR ($+$) Operation

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT ($'$) Operation

x	x'
0	1
1	0

(Inverse/Complement Postulate)

Truth Table: What is the output for every input combination?

Basic Theorems

1. $x + x = x$
2. $x + 1 = 1$
3. **Involution**: $(x')' = x$
4. **Associativity**: $x + (y + z) = (x + y) + z$
5. **De Morgan**: $(x + y)' = x'y'$
6. **Absorption**: $x + xy = x$

Dual (Exchange $+/\bullet$ and $0/1$)

1. $x \bullet x = x$
2. $x \bullet 0 = 0$
3. $x \bullet (y \bullet z) = (x \bullet y) \bullet z$
4. $(x \bullet y)' = x' + y'$
5. $x \bullet (x + y) = x$

Need to be proved from Postulates

Let us prove: **$x + x = x$**

$$x + x$$

$$= (x + x) \bullet 1$$

$$= (x + x) \bullet (x + x')$$

$$= x + (x \bullet x')$$

$$= x + 0$$

$$= x$$

Identity: $y \bullet 1 = y$

Complement: $y + y' = 1$

Distributive: $a + bc = (a + b)(a + c)$

Complement: $xx' = 0$

Identity: $y + 0 = y$

Exercises

- Prove the other theorems
 - using only Postulates
- Simplification: use Truth Tables

Simplifying Boolean Expressions

- Simplify $F = x'y'z + xyz + x'yz + xy'z$
- Repeated application of Postulates and Basic Theorems

Develop Truth Table

- $F = x'y'z$
- AND of multiple variables?

x	y	z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Boolean Functions

- Boolean Function: Algebraic expression of
 - Boolean variables
 - Constants 0/1
 - Logic operations
- Value of a function
 - 0 or 1 for a given set of values of input variables
 - What are the **domain** and **co-domain** of a Boolean Function?

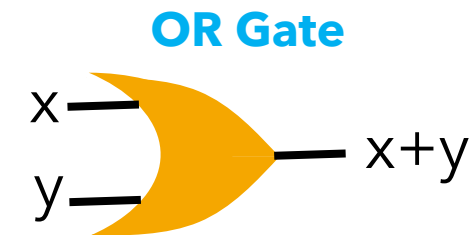
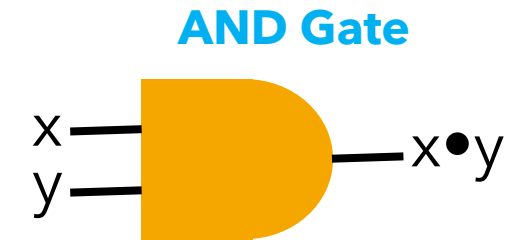
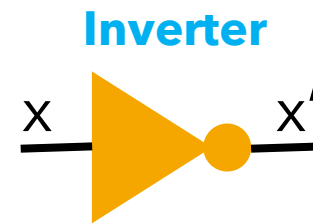
Building more complex functions

- Use AND/OR to build complex expressions
 - representing logical conditions
- **$F = x + y'z$**
- $F = 1$ if
 - $x = 1$, OR
 - $y'z = 1$
 - if $y' = 1$ AND $z = 1$
 - i.e., $y = 0$ AND $z = 1$
- Else, $F = 0$
- F can be represented with Truth Table

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

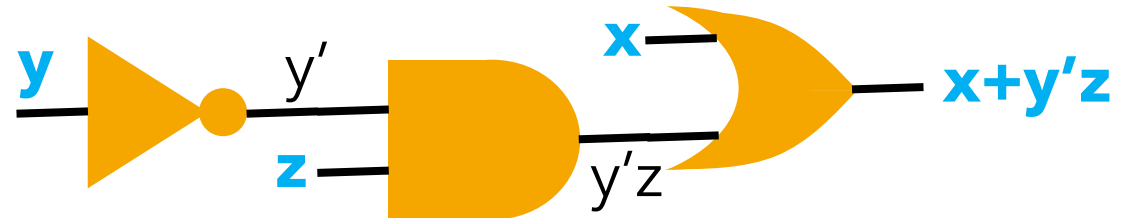
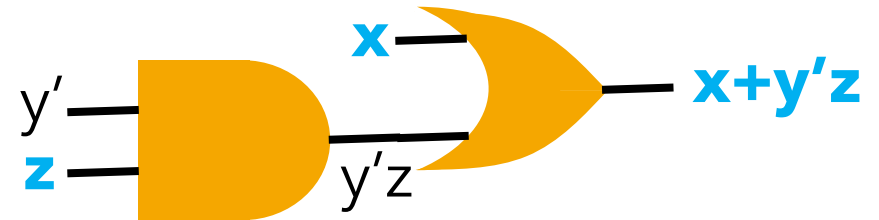
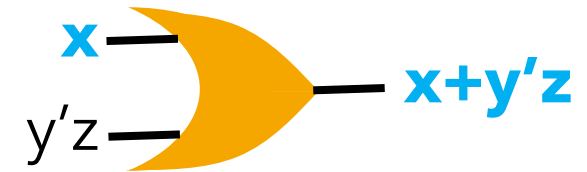
Representing with gates

- Logic gates represent Boolean functions
 - Variables are input
 - Function value is output



Representing $F = x + y'z$

- **Gates:** Same info. as Boolean expression
 - Precedence: **()**, **'**, **AND**, **OR**
- Break down complex expression ($y'z$) until we have elementary gates
- **Primary inputs** to Boolean circuit are variables



Which representation is best?

- Expression vs Gates vs Truth Table
- Gates: diagram view
 - easier to visualise function (if simple)
- Expression: enables automation
- Truth Table: unique

Boolean manipulation

- **Truth Table**: no further simplification
- **Expressions** could be simplified
 - fewer **literals** (x or x' counts as 1 **literal**)
 - $x + xy'$ (3 literals) = x (1 literal)
- How does simplification help?

Simplification Example

- $F = x'y'z + x'yz + xy'$
- Draw Gate circuit/schematic
- Simplify
- Draw Gate circuit of simplified function
- Simplification: **Cost/Area reduction**
- Approx. comparison: **literal count**
- Verify with Truth Table: How?

More Simplifications

- $x(x' + y)$
- $x + x'y$
- $(x + y)(x + y')$
- $xy + x'z + yz = xy + x'z$ [Consensus Theorem]
- $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$

Complement of a function

- De Morgan's Theorem: $(x + y)' = x'y'$ $(xy)' = x' + y'$
- What about $(x + y + z)'$?
- Generalisation (A, B,... are expressions):
 - $(A + B + C + \dots)' = A'B'C' \dots$
 - $(ABC \dots)' = A' + B' + C' + \dots$
 - Convert to **Dual**, complement literals
- Exercise: Find complement of $F = (x'y'z + x'y'z)$

Complementing in Truth Table

- New column for every new function (F , F')
- Complement F column to obtain F'

x y z	F	F'
0 0 0	0	1
0 0 1	1	0
0 1 0	0	1
0 1 1	0	1
1 0 0	1	0
1 0 1	1	0
1 1 0	1	0
1 1 1	1	0

Complement the column



Canonical Forms: Minterm

- Function of 3 variables: x, y, z
- AND of 3 variables, each in normal/complemented form:
 - $xyz, xyz', xy'z, \dots$ (8 terms)
 - each is a **minterm**
- Function F of n variables: **2^n minterms**
- F can be represented as **Sum of minterms**
 - Select those minterms for which **$F = 1$**

x y z	minterms	F
0 0 0	$x'y'z'$	0
0 0 1	$x'y'z$	1
0 1 0	$x'yz'$	0
0 1 1	$x'yz$	0
1 0 0	$xy'z'$	1
1 0 1	$xy'z$	1
1 1 0	xyz'	0
1 1 1	xyz	1

$$F = x'y'z + xy'z' + xy'z + xyz$$

Complement from Truth Table

x y z	minterms	F
0 0 0	$x'y'z'$	0
0 0 1	$x'y'z$	1
0 1 0	$x'yz'$	0
0 1 1	$x'yz$	0
1 0 0	$xy'z'$	1
1 0 1	$xy'z$	1
1 1 0	xyz'	0
1 1 1	xyz	1

$$F = x'y'z + xy'z' + xy'z + xyz$$

Complement of F: Pick minterms corresponding to 0s

$$F' = x'y'z' + x'yz' + x'yz + xyz'$$

Invert F' to obtain F :

$$\begin{aligned} F &= (F')' = (x'y'z' + x'yz' + x'yz + xyz')' \\ &= (x'y'z')'(x'yz')'(x'yz)'(xyz')' \\ &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y' + z) \end{aligned}$$

Each of $(x + y + z)$, $(x + y' + z)$, etc., is a **maxterm**

Function F can equivalently be represented as a **product of maxterms**

Also canonical representation of a function

Alternative Canonical Representations

x y z	minterms	maxterms	F
0 0 0	$x'y'z'$	$x + y + z$	0
0 0 1	$x'y'z$	$x + y + z'$	1
0 1 0	$x'yz'$	$x + y' + z$	0
0 1 1	$x'yz$	$x + y' + z'$	0
1 0 0	$xy'z'$	$x' + y + z$	1
1 0 1	$xy'z$	$x' + y + z'$	1
1 1 0	xyz'	$x' + y' + z$	0
1 1 1	xyz	$x' + y' + z'$	1

$$F = x'y'z + xy'z' + xy'z + xyz$$

$$F = (x + y + z)(x + y' + z)(x + y' + z')(x' + y' + z)$$

Abbreviated Representations

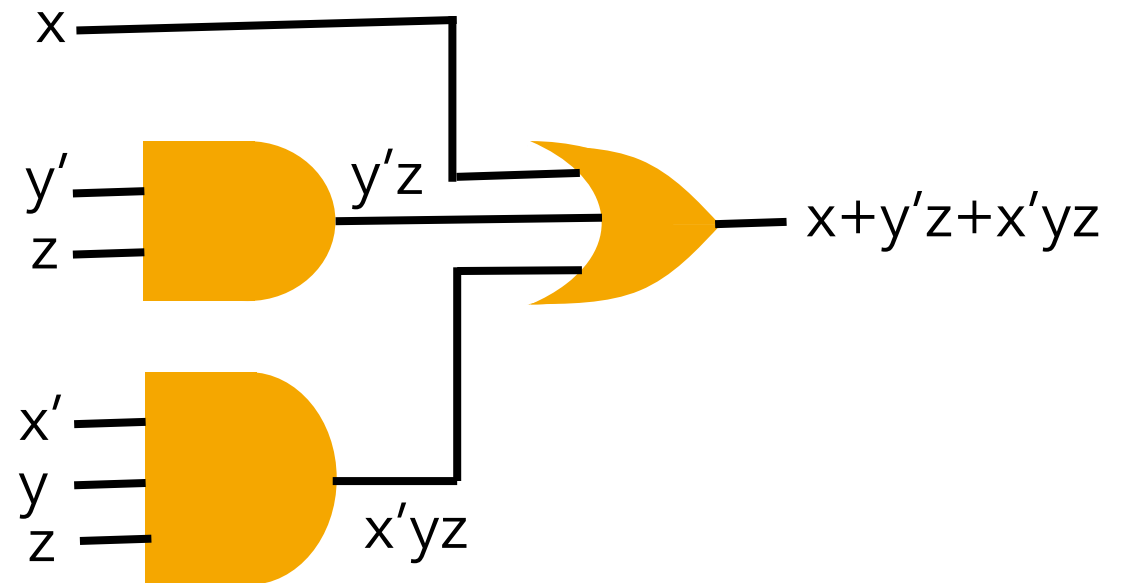
x y z	minterms		maxterms		F
0 0 0	$x'y'z'$	m_0	$x + y + z$	M_0	0
0 0 1	$x'y'z$	m_1	$x + y + z'$	M_1	1
0 1 0	$x'yz'$	m_2	$x + y' + z$	M_2	0
0 1 1	$x'yz$	m_3	$x + y' + z'$	M_3	0
1 0 0	$xy'z'$	m_4	$x' + y + z$	M_4	1
1 0 1	$xy'z$	m_5	$x' + y + z'$	M_5	1
1 1 0	xyz'	m_6	$x' + y' + z$	M_6	0
1 1 1	xyz	m_7	$x' + y' + z'$	M_7	1

$F = x'y'z + xy'z' + xy'z + xyz$
 $= m_1 + m_4 + m_5 + m_7$
 $= \sum (1, 4, 5, 7)$

$F = (x + y + z)(x + y' + z)(x + y' + z')(x' + y' + z)$
 $= M_0 M_2 M_3 M_6$
 $= \prod (0, 2, 3, 6)$

Standard Forms

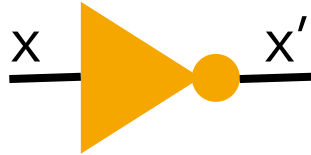
- Sum of Products
- Product of Sums
- 2-level implementation



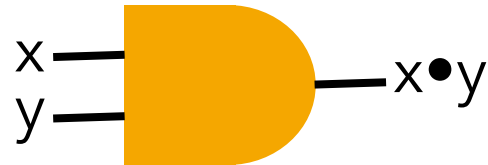
Sum of Products
2 Levels: AND followed by OR

Other Logic Gates

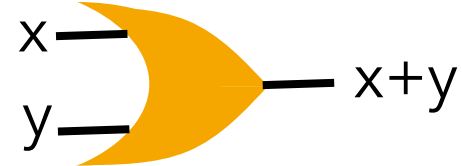
Inverter



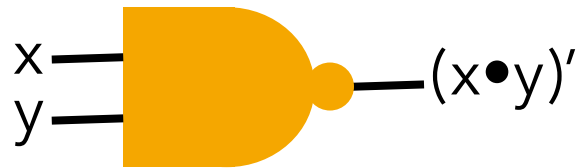
AND Gate



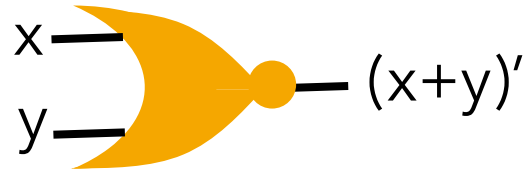
OR Gate



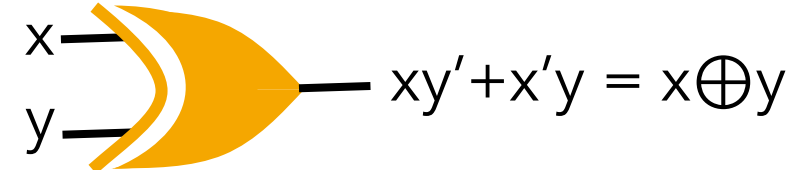
NAND Gate



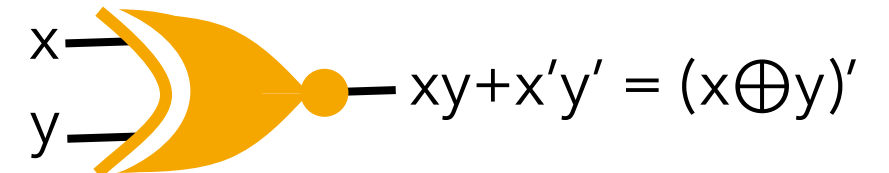
NOR Gate



XOR (Difference) Gate

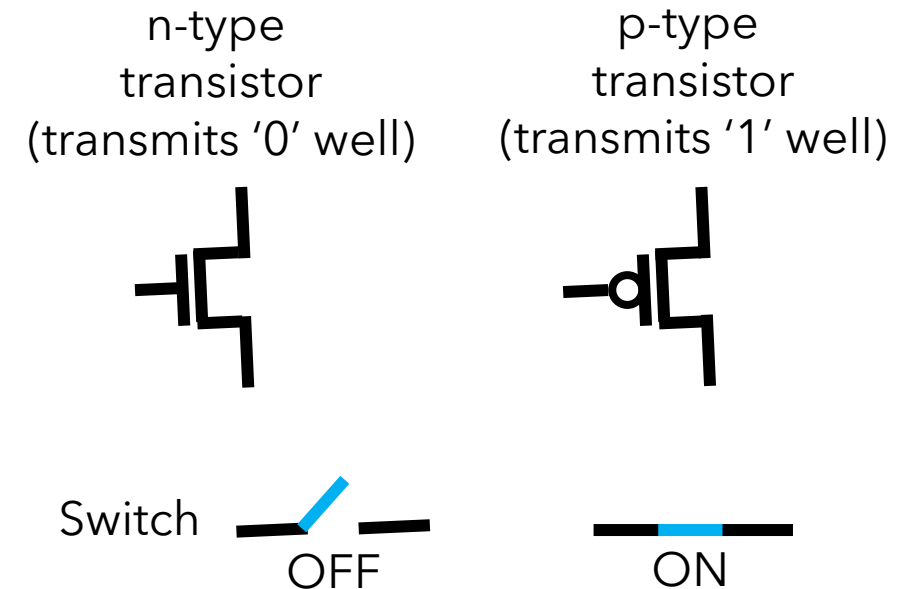


XNOR (Equality) Gate



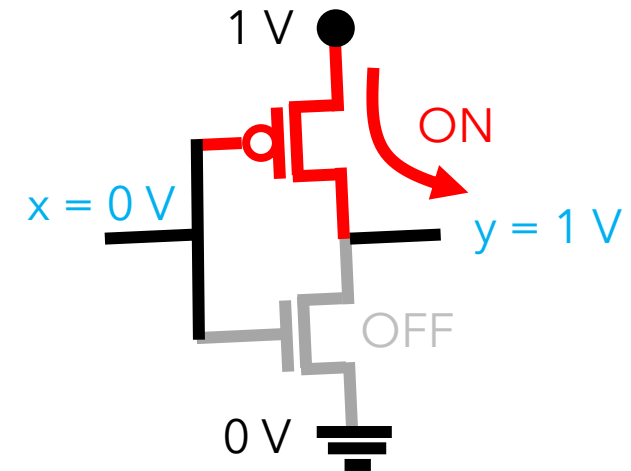
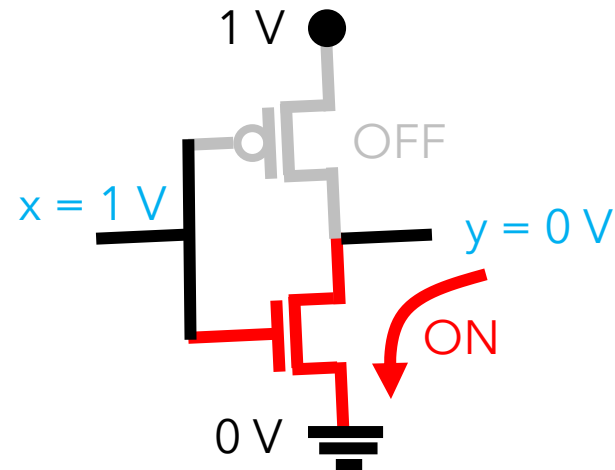
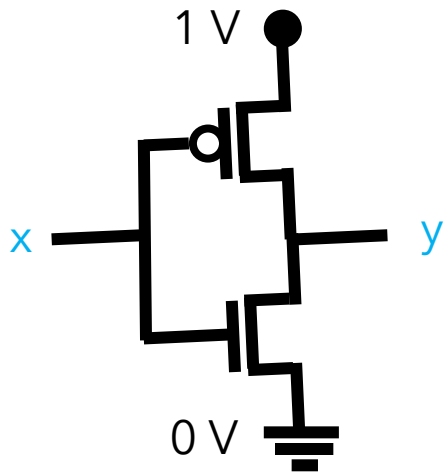
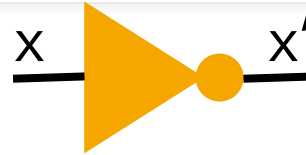
Implementing a Gate

- CMOS Technology popular today
 - Complementary Metal Oxide Semiconductor
- **Transistor** used as a **switch**
 - Digital abstraction begins here
 - transistor: **physical domain**
 - switch: **logical domain**



Realising an Inverter with Transistors

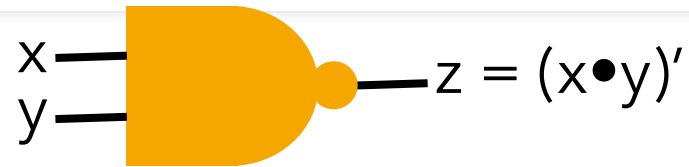
Inverter



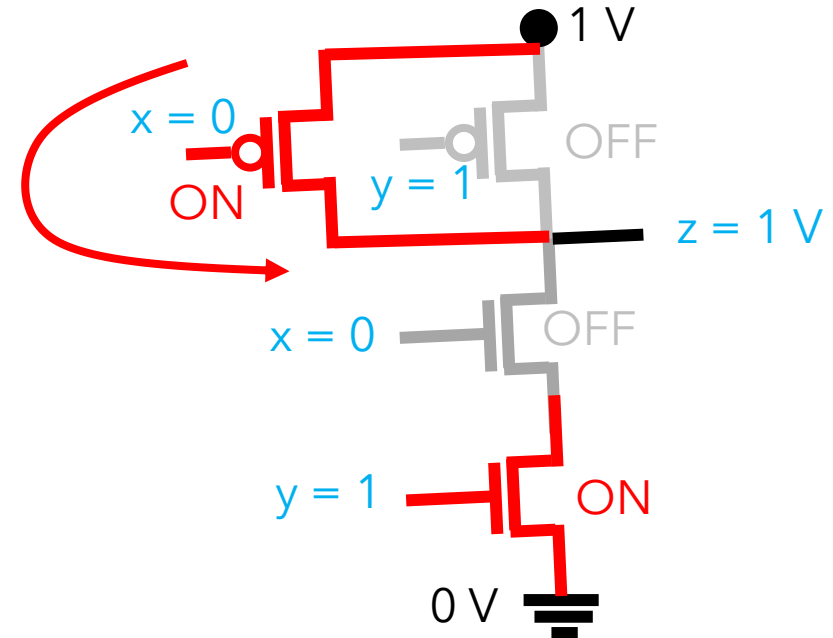
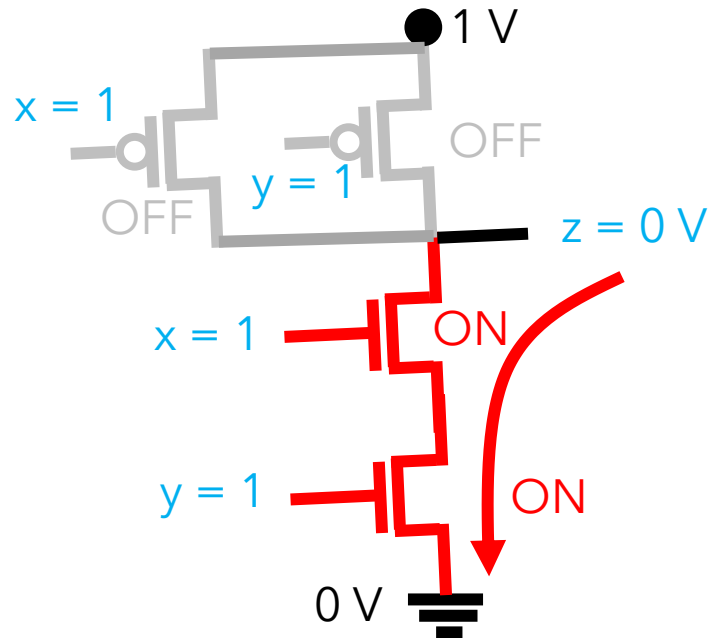
Inverter function: $y = x'$

Implementing a NAND Gate

NAND Gate



x	y	z
0	0	1
0	1	1
1	0	1
1	1	0



NAND function: $z = (xy)'$

Assignment 2

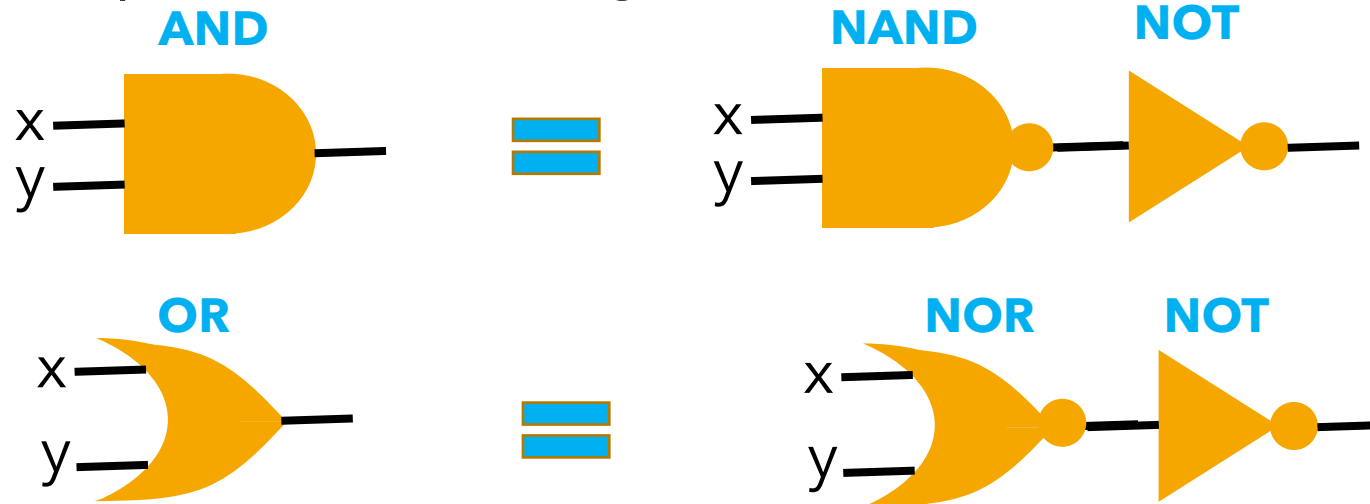
- Expanded Boolean Function
- Each term is **maximally expanded**
 - keep **expanding the region** as long as it is **legal**
- #terms in input function = #terms in expanded function

Implementing other gates

- NOR gate is similarly implemented
- General principle?
 - Transistors in **series** = what function?
 - Transistors in **parallel** = what function?

Positive Logic: AND/OR

- How do we implement AND gate?



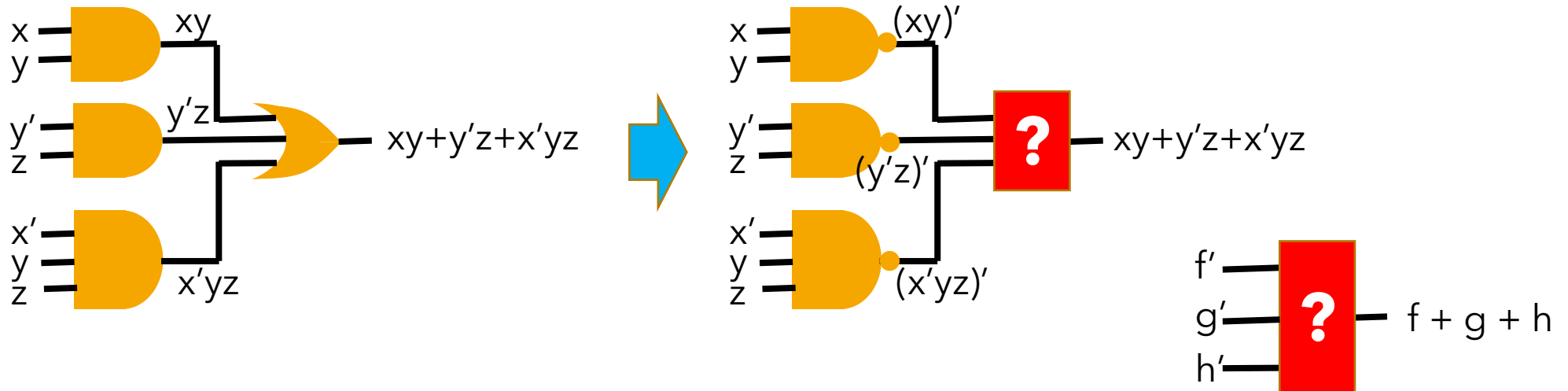
- NAND/NOR are simpler circuits** (4 transistors)
 - AND/OR are more expensive (6 transistors)

Area Estimate with Literals

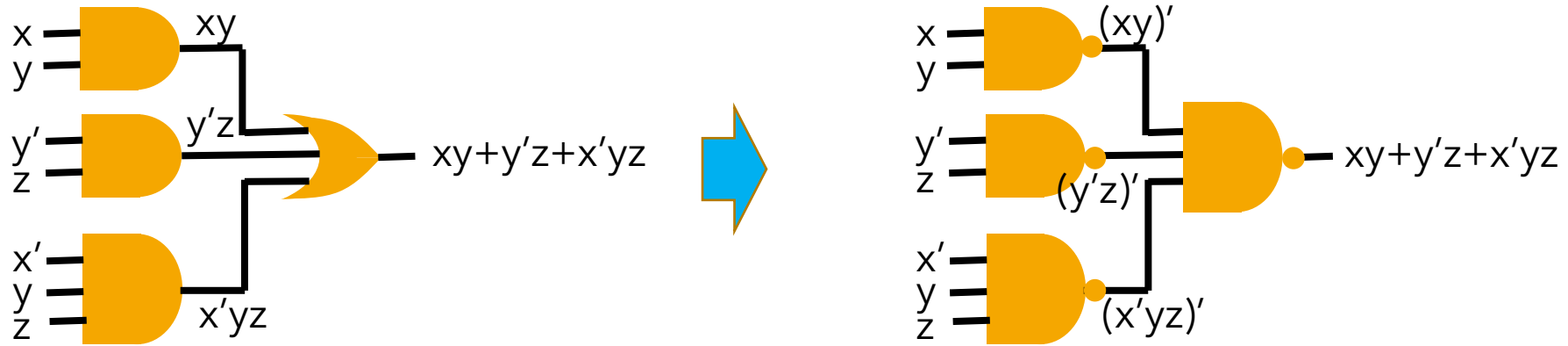
- Recall **literal** definition: we ignored primes
 - x and x' were both considered 1 literal
 - Why?
- Sometimes **x'** is easier to implement (e.g., **$x'y'$: NOR**)

Replacing with NAND

- 2-level implementation with AND-OR
- Replace AND with NAND gates instead?
- Replace OR with what?

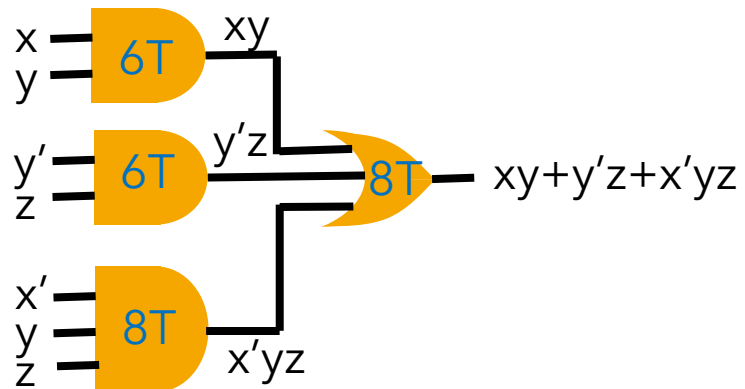


NAND-NAND functionally equivalent to AND-OR

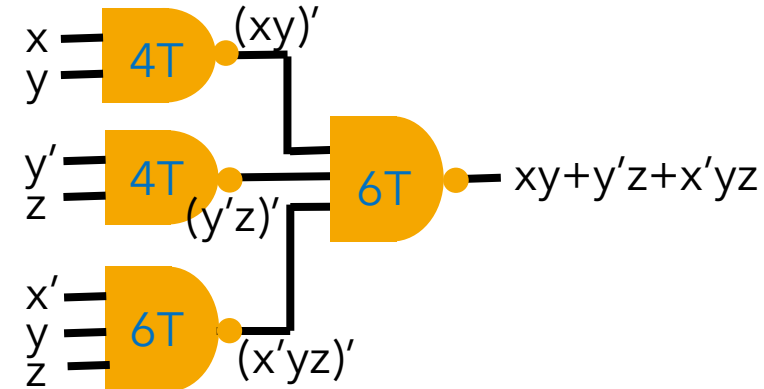


Cost of AND-OR vs. NAND-NAND

- 2-input NAND: 4 Transistors (4T)
- 3-input NAND: 6T
- 2-input AND: 6T (2-i/p NAND + NOT)
- 3-input AND: 8T (3-i/p NAND + NOT)
- 3-input OR: 8T (3-i/p NOR + NOT)



$$6T + 6T + 8T + 8T = 28 \text{ Transistors}$$



$$4T + 4T + 6T + 6T = 20 \text{ Transistors}$$

+2 Transistors for each inverter