Week2 Lecture Boolean Logic

Abstractions

Abstraction is a technique for arranging complexity of computer systems

Top-down vs Bottom-up

Waterfall development / Agile development

procedural / oop

Discrete event / Agents

Building from Elementary Components

build from millions of nano size transistors (used to build logic gate)

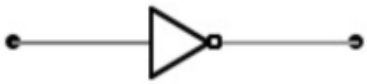
Boolean Logic

every chip can be built from AND OR NOT gate, and they can be built from NAND.

so: **every possible chip can be built from just the NAND gates**

Elementary Logic Gates

$A = \bar{A}$

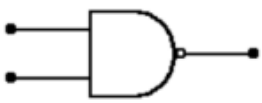


$A\ AND\ B = A \cdot B$

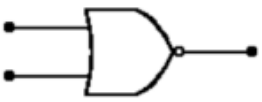
$A\ OR\ B = A + B$

$A\ XOR\ B = A \oplus B$

$A\ NAND\ B = \overline{A \cdot B}$



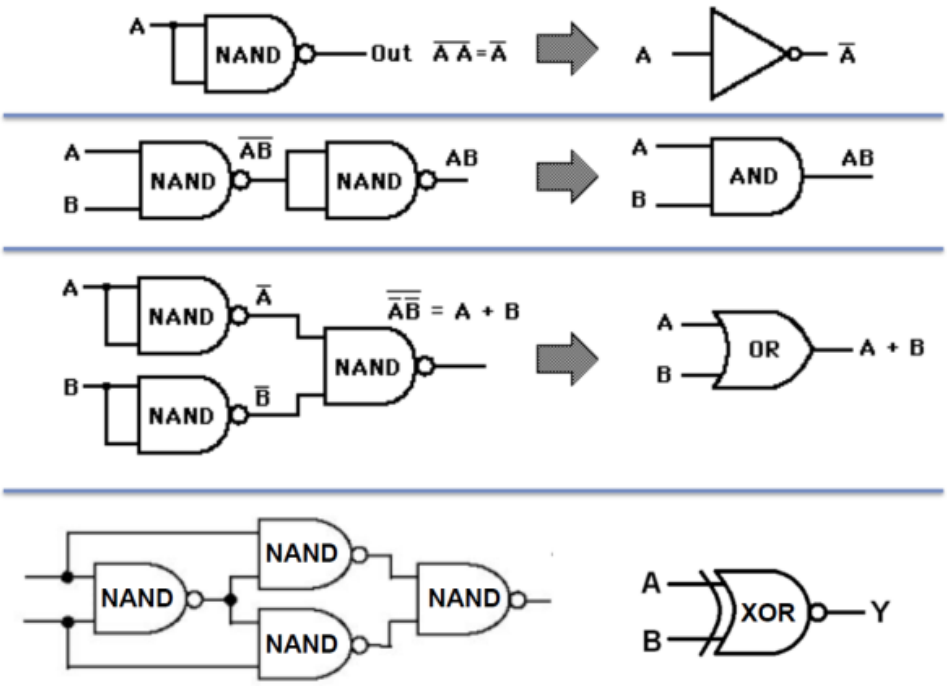
$A\ NOR\ B = \overline{A + B}$



Gate Logic

Gates with 3 or more inputs require composing a structure with multiple gates (hence called composite gates)

The Super NAND



Boolean Expressions

precedence

Parentheses > Not > And > Or

Laws of Boolean Algebra

1. Law of Identity	
$A = A$	$\overline{A} = \overline{A}$
2. Commutative Law	
$A \cdot B = B \cdot A$	$A + B = B + A$
3. Associative Law	
$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	$(A + B) + C = A + B + C$
4. Idempotent Law	
$A \cdot A = A$	$A + A = A$
5. Double Negative Law	
$\overline{\overline{A}} = A$	
6. Complementary Law	
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
7. Law of Intersection	
$A \cdot 1 = A$	$A \cdot 0 = 0$
8. Law of Union	
$A + 1 = 1$	$A + 0 = A$
9. Distributive Law	
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$
10. Law of Absorption	
$A \cdot (A + B) = A$	$A + A \cdot B = A$
11. Law of Common Identities	
$A \cdot (\overline{A} + B) = AB$	$A + (\overline{A} \cdot B) = A + B$
12. De Morgan's Law	
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$

Idempotent law

x Or x=x x And x=x

x Or x Or x Or x ..... = x

Associative law

(x Or y)Or z = x Or(y Or z)

(x And y)z = x(y And z)

Commutative law

x And y = y And x

x Or y=y Or x

Distributive law

x and (y or z) = (x and y) or (x and z)

x or (y and z) = (x or y) and (x or z)

De Morgans Law

Not(x And y) = Not(x) Or Not(y)

Not(x Or y) = Not(x) And Not(y)

Complement law

x And(Not(x))=0

x Or (Not(x))=1

Double negative Law

(NOT(NOT(x)))=x

example1

Not(Not(x) And Not(x Or y))

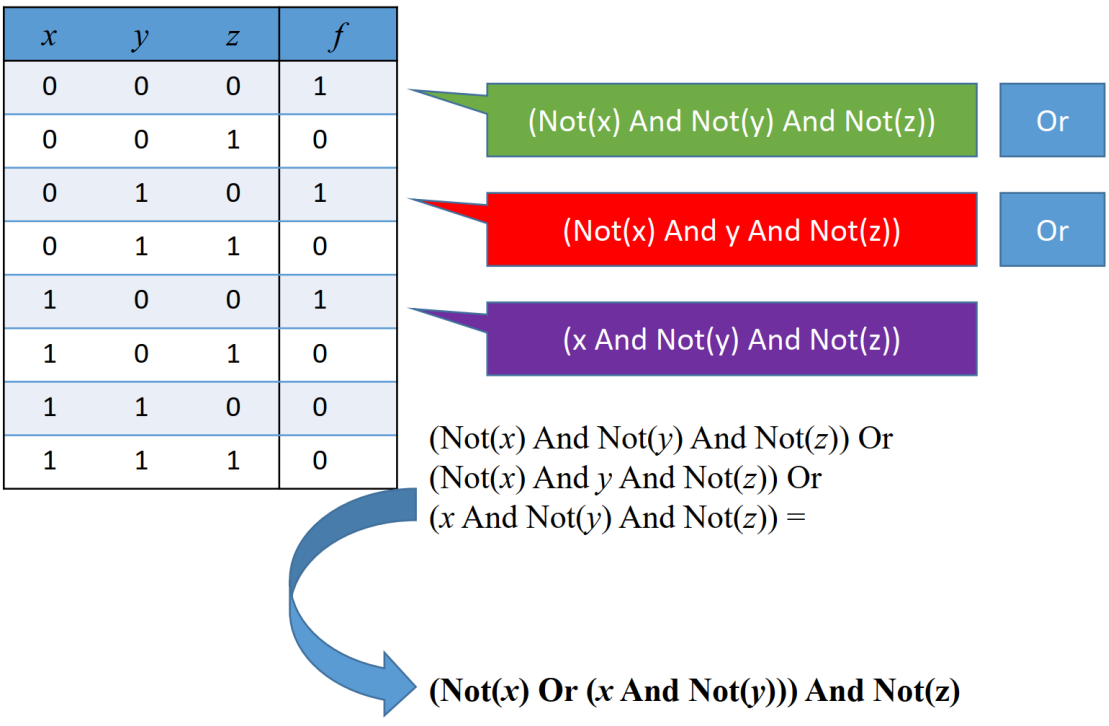
= x OR (x OR y)

= (x OR x) OR y

= x OR y

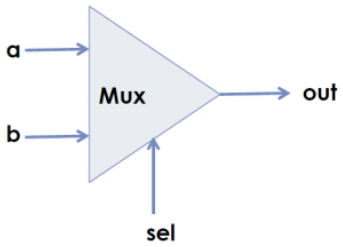
Boolean function synthesis

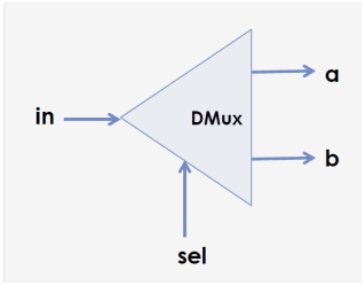
from truth table to boolean function



Multiplexers and Demultiplexers

- A multiplexer is a combinational circuit that provides **single output** but accepts **multiple data inputs**.
- A demultiplexer is a combinational circuit that takes **single input** but that input can be directed through **multiple outputs**.



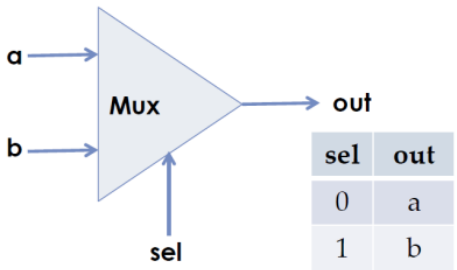


Multiplexer

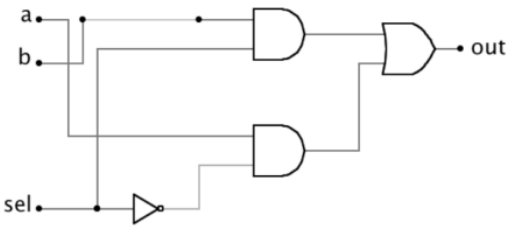
Multiplexer

a	b	SEL	OUT
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

If sel==1 then
  out = b
else
  out = a



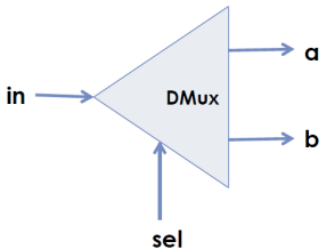
sel	out
0	a
1	b



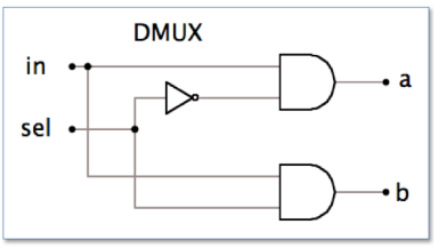
Demultiplexer

Demultiplexer

sel	a	b
0	in	0
1	0	in



If sel = 0 then
  {a = in; b = 0}
else
  {a = 0; b = in}



sel[1]	sel[0]	out
0	0	a
0	1	b
1	0	c
1	1	d