Chapter 2: Switching Algebra and Logic Circuits

Formal Foundation of Digital Design

- In 1854 George Boole published
  An investigation into the Laws of Thoughts
  - Algebraic system with two values 0 and 1
  - Used to formally determine the truth or falsehood of propositions
- In 1938 Claude Shannon showed in his MS thesis how Boolean algebra can be used for analysis of circuits
  - Algebraic system with two values 0 and 1
  - At that time circuits were built from relays
  - Hence the term "switching algebra"
  - Two states hence a bit

Boolean Functions

- In arithmetic there are certain, familiar functions, such as:

$$2 \times 3 = 6$$

- In logic another set of functions is defined. Unlike arithmetic functions these have binary inputs and binary outputs.

Boolean Algebra Axioms

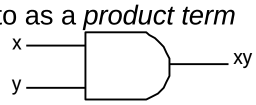
- Set of two values:

{0,1} or {false,true} or {low,high}

- There are 2 binary and one unary operations defined for elements in Boolean algebra

"AND"

- Operation: TRUE if both inputs are TRUE
- Symbol: x AND y = x•y = xy = x ^ y
- often referred to as a *product term*
- Logic gate:

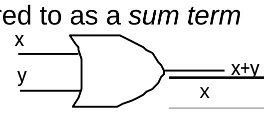


- Truth table:

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

"OR"

- Operation: TRUE if either or both inputs is TRUE
- Symbol: x OR y = x+y = x v y
- often referred to as a *sum term*
- Logic gate:

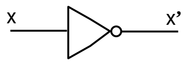


- Truth table:

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

“NOT”

- Operation: TRUE iff the input is FALSE
- Symbol: NOT x = ~x = x’ = $\overline{x}$
- often referred to as an *inverter* or a *complement*
- Logic gate:



- Truth table:

x	x’
0	1
1	0

Basic Properties of Switching Algebra

- Operations can be combined using parentheses
- With parentheses, order of operations is from the innermost to the outermost parentheses
- Order:
  1) negation,
  2) multiplication,
  3) addition
- 1-variable theorems
- 2- and 3-variable theorems

1-variable theorems

- T1: x+0 = x x•1 = x *identities*
- T2: x+1 = 1 x•0 = 0 *null elements*
- T3: x+x = x x•x = x *idempotency*
- T4: (x′)′ = x *involution*
- T5: x+x′ = 1 x•x′ = 0 *complements*

duality

Proofs are done by perfect induction

Consider all possible combinations on the *lhs* and *rhs*, and check whether they are equal

Perfect Induction

(T3) $\overbrace{x+x}^{\text{LHS}} = \overbrace{x}^{\text{RHS}}$

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

$0+0 \stackrel{?}{=} 0$

$1+1 \stackrel{?}{=} 1$

(T3) $\overbrace{x \bullet x}^{\text{LHS}} = \overbrace{x}^{\text{RHS}}$

x	y	•
0	0	0
0	1	0
1	0	0
1	1	1

$0 \bullet 0 \stackrel{?}{=} 0$

$1 \bullet 1 \stackrel{?}{=} 1$

2- and 3-variable theorems

commutativity
- T6: x+y = y+x x•y = y•x

associativity
- T7: (x+y)+z = x+(y+z) (x•y)•z = x•(y•z)

distributivity
- T8: x•y+x•z = x•(y+z) (x+y)•(x+z) = x+(y•z)

covering
- T9: x+x•y = x x•(x+y) = x

combining
- T10: x•y + x•y′= x (x+y)•(x+y′) = x

consensus
- T11: x+(x′•y) = x+y x•(x′+y) = x•y
- T12: x•y+x′•z+y•z = x•y+x′•z
  (x+y)•(x′+z)•(y+z) = (x+y)•(x′+z)

Swap 0 & 1, AND & OR, theorems stay true

Proofs

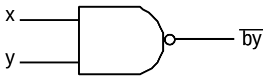
(T8) (x+y)•(x+z) = x+(y•z), *distributivity*

Proof: use perfect induction

x y z	LHS (x+y)•(x+z)	RHS x+y•z
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	1	1
1 0 0	1	1
1 0 1	1	1
1 1 0	1	1
1 1 1	1	1

“NAND”

- Operation: TRUE if either or both inputs is FALSE
- Symbol: x NAND y = (x•y)′ = $\overline{xy}$ = $\overline{x \wedge y}$
- Logic gate:

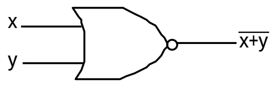


- Truth table:

x	y	(xy)′
0	0	1
0	1	1
1	0	1
1	1	0

“NOR”

- Operation: TRUE if both inputs are FALSE
- Symbol: x OR y = = (x+y)’ = $\overline{x+y}$ = $\overline{x \vee y}$
- Logic gate:



- Truth table:

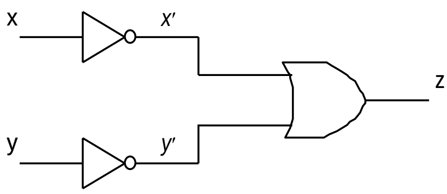
x	y	(x+y)′
0	0	1
0	1	0
1	0	0
1	1	0

Algebraic expressions, Equations and Circuits

z = x′+y′

Given inputs x and y,

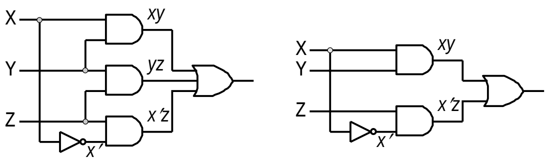
the output is z = x′+y′



Algebraic expressions, Equations and Circuits (cont.)

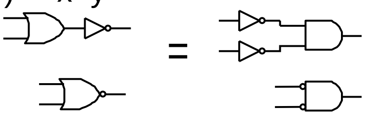
Consensus theorem T12

LHS: x•y+x′•z+y•z = x•y+x′•z :RHS

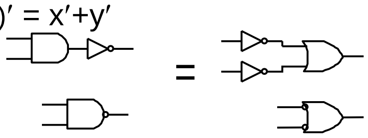


DeMorgan Laws

- (x+y)′ = x′•y′

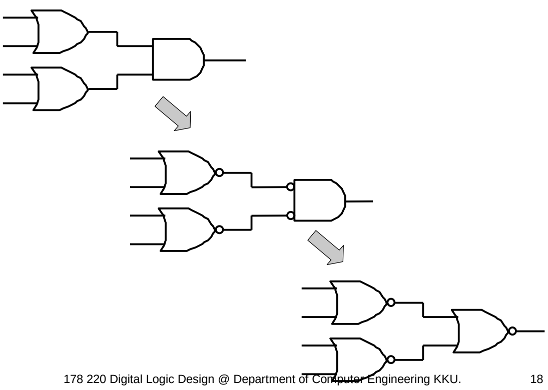


- (x•y)′ = x′+y′

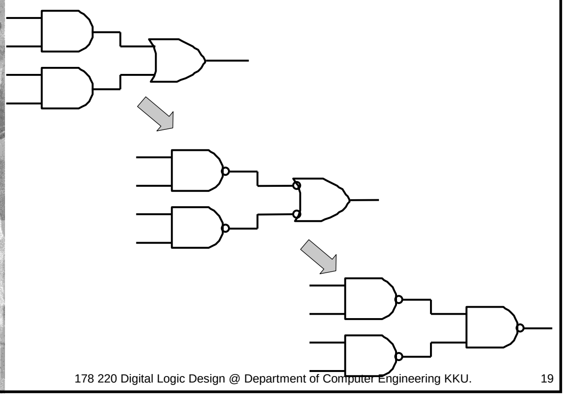


‘pushing the bubble’

From AND and ORs to NORs

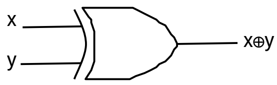


From ANDs and OR to NANDs



"XOR" (Exclusive-OR)

- Operation: TRUE iff either inputs is TRUE
- Symbol: x XOR y = $x \oplus y$
- Often referred to as an *unequivalent* gate
- Logic gate:



- Truth table:

x	y	(x ⊕ y)
0	0	0
0	1	1
1	0	1
1	1	0

Simplifying Logic Functions

Logic Minimisation: reduce complexity of the gate level implementation

- reduce number of literals (gate inputs)
- reduce number of gates
- reduce number of levels of gates

Simplifying Logic Functions (cont.)

- reduce number of gates
- fewer inputs implies faster gates in some technologies
- *fan-ins* (number of gate inputs) are limited in some technologies
- fewer levels of gates implies reduced signal propagation delays
- minimum delay configuration typically requires more gates
- number of gates (or gate packages) influences manufacturing costs

Alternative Logic Implementations

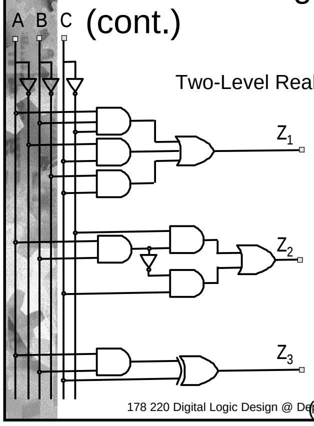
A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$Z_1 = ABC' + A'C + B'C$

$Z_2 = (AB \cdot C') + ((AB)' \cdot C)$

$Z_3 = AB \oplus C$

Alternative Logic Implementations (cont.)



TTL Package Counts:

Two-Level Realisation *(first set of INVs doesn't count)*
3 packages
(1x 6-INVs, 1x 3-input AND, 1x 3-input OR)

Multi-Level Realisation
Adv: Reduced Gate Fan-ins
3 packages
(1x 6-INVs, 1x 2-input ANDs, 1x 2-input OR)

Complex Gate: XOR
Adv: Fewest Gates
2 packages
(1x 2-input AND, 1x 2-input XOR)

Derivation of Expression

- *Given*:- desired truth table
- *Problem*:- to derive the boolean expression
- Simplest way is to form the *product terms*

Any logic expression can always be expressed in one of the two standard forms:

1. **Sum-of-Product (SOP) form**

Each term in the standard SOP form is known as *minterm*.

2. **Product-of-Sum (POS) form**

Each term in the standard POS form is known as *maxterm*.

Derivation of Expression (cont.)

Sum-of-Product form (SOP)

Procedure :-

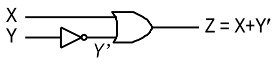
1. Form 'product terms' column
2. Complement the variables in each product if the corresponding input is '0'
3. Form SOP expression from rows where output is '1'

Derivation of Expression (cont.)

X	Y	Z	Product terms
0	0	1	X′Y′
0	1	0	X′Y
1	0	1	XY′
1	1	1	XY

$Z=X'Y' + XY' + XY$

However, consider this circuit!



Derivation of Expression (cont.)

Product-of-Sum form (POS)

Procedure :-

1. Form ‘sum terms’ column
2. Complement the variables in each sum if the corresponding input is ‘1’
3. Form POS expression from rows where output is ‘0’

Derivation of Expression (cont.)

X	Y	Z	Sum terms
0	0	1	X+Y
0	1	0	X+Y′
1	0	0	X′+Y
1	1	1	X′+Y′

$Z = (X+Y')(X'+Y)$

$= XX'+XY+X'Y'+YY'$

$= XY+X'Y'$