CS2100 Comp Org Notes

AY23/24 Sem 1, github.com/gerteck

12. Boolean Algebra

Digital Circuits

- Two voltage levels, 1 for high, 0 for low.
- Digital circuits over analog circuits are more reliable, specified accurarcy (determinable).
- Digital circuits abstracted using simple mathematical model: **(Boolean Algebra)**
- Design, Analysis and simplification of digital circuit: **Digital Logic Design.**
- **Combinational**: no memory, output depends solely on the input. (gates, adders, multiplexers)
- **Sequential:** with memory, output depends on both input and current state. (counters, registers, memories)

Boolean Algebra

connectives in order of precedence:

- **negation** A' equivalent to **NOT**
- **conjunction** $A \cdot B$ equivalent to **AND**
- **disjunction** $A + B$ equivalent to **OR**
- Note: always write the AND operator ·, do not omit, as it may be confused with a 2 bit value, AB.
- **Truth Table**: Provides listing of every possible combination of inputs and corresponding outputs. We may prove using truth table by comparing columns.

Duality

- **Duality**: if the AND/OR operators and identity elements 0/1 interchanged in a boolean equation, it remains valid.
- e.g. the dual equation of $a + (b \cdot c) = (a + b) \cdot (a + c)$ is $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$., where if one is valid, then its dual is also valid.

Laws & Theorems of Boolean Algebra

Identity laws	
A + 0 = 0 + A = A	A · 1 = 1 · A = A
Inverse/complement laws	
A + A' = A' + A = 1	A · A' = A' · A = 0
Commutative laws	
A + B = B + A	A · B = B · A
Associative laws *	
A + (B + C) = (A + B) + C	A · (B · C) = (A · B) · C
Distributive laws	
A · (B + C) = (A · B) + (A · C)	A + (B · C) = (A + B) · (A + C)

Idempotency	
X + X = X	X · X = X
One element / Zero element	
X + 1 = 1 + X = 1	X · 0 = 0 · X = 0
Involution	
(X')' = X	
Absorption 1	
X + X·Y = X	X·(X + Y) = X
Absorption 2	
X + X'·Y = X + Y	X·(X' + Y) = X·Y
DeMorgans' (can be generalised to more than 2 variables)	
(X + Y)' = X' · Y'	(X · Y)' = X' + Y'
Consensus	
X·Y + X'·Z + Y·Z = X·Y + X'·Z	(X+Y)·(X'+Z)·(Y+Z) = (X+Y)·(X'+Z)

left/right equations are duals of each other

Proving Theorems

- Theorems can be proved using truth table, or by algebraic manipulation using other theorems/laws.

  - Example: Prove absorption theorem X + X·Y = X

    X + X·Y = X·1 + X·Y (by identity law)
    = X·(1+Y) (by distributivity)
    = X·1 (by one element law)
    = X (by identity law)

  - By the principle of duality, we may also cite (without proof) that X·(X+Y) = X.

Boolean Functions, Complements

- Represented by F, e.g. $F1(x, y, z) = x \cdot y \cdot z'$.
- To prove $F1 = F2$, we may use boolean algebra, or use truth tables.
- Complement Function is denoted as F', obtained by interchanging 1 with 0 in function's output values.

Standard Forms

- **Literals**: A Boolean variable on its own or in its complemented form. (e.g. x, x')
- **Product Term**: A single literal or a logical product (AND, ·) of several literals. (e.g. x, $x \cdot y \cdot z'$)
- **Sum Term**: A single literal or a logical sum (OR +) of several literals. (e.g. $A + B'$)
- **sum-of-products (SOP) expression**: A product term or a logical sum (OR +) of several product terms.
- **product-of-sums (POS) expression**: A sum term or a logical product (AND) of several sum terms.
- Every boolean expr can be expressed in SOP/POS form.

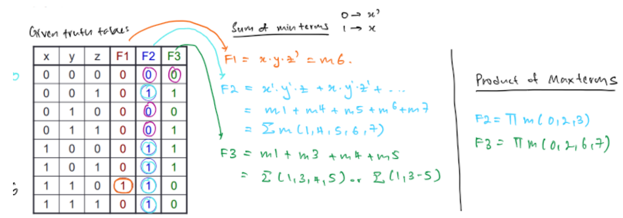
Minterms and Maxterms

- **minterm** (of n variables): a product term that contains n literals from all the variables; denoted $m0$ to $m[2^n - 1]$
- **maxterm** (of n variables): a sum term that contains n literals from all the variables; denoted $M0$ to $M[2^n - 1]$
- Each minterm is the complement ($m2' = M2$) of its corresponding maxterm, vice versa.

x	y	Minterms		Maxterms	
		Term	Notation	Term	Notation
0	0	x'·y'	m0	x+y	M0
0	1	x'·y	m1	x+y'	M1
1	0	x·y'	m2	x'+y	M2
1	1	x·y	m3	x'+y'	M3

Canonical Forms

- Canonical/normal form: a unique form of representation.
- **Sum-of-minterms** = Canonical sum-of-products
- **Product-of-maxterms** = Canonical product-of-sums



x	y	z	F1	F2	F3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	0	1	0

- We can convert between sum-of-minterms and product-of-maxterms easily, by DeMorgan's.

13. Logic Gates & Simplification

Logic Gates

- Fan-in: The number of inputs of a gate $\geq 1, 2$.
- Implement bool exp / function as logic circuit.

Universal Gates

- **universal gate**: can implement a complete set of logic.
- $\{AND, OR, NOT\}$ are a complete set of logic, sufficient for building any boolean function.
- $\{NAND\}$ and $\{NOR\}$ themselves a complete set of logic. Implement NOT/AND/OR using only NAND or NOR gates.

SOP and POS

- an SOP expression can be easily implemented using
  - 2-level AND-OR circuit or 2-level NAND circuit
- a POS expression can be easily implemented using
  - 2-level OR-AND circuit or 2-level NOR circuit

Algebraic Simplification

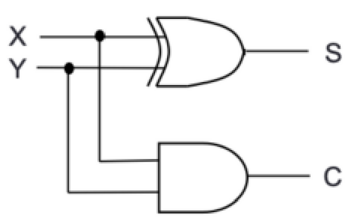
- **Function Simplification**: Make use of alegbraic (using theorems) or Karnaugh Maps (easier to use, limited to no more than 6 variables) or Quine-McCluskey.
- **Algebraic Simplification**: aims to minimise
  1. number of literals (prioritised over number of terms)
  2. number of terms.

Half Adder

- Half adder is a circuit that adds 2 single bits (X, Y) to produce a result of 2 bits (C, S).
  - $C = X \cdot Y$; $S = S \oplus Y$

Inputs		Outputs	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

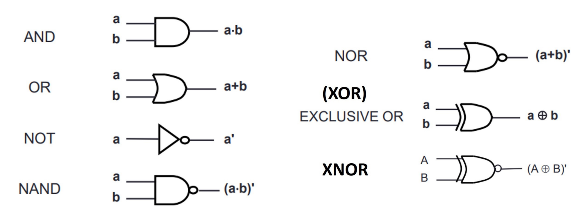
outputs



implementation of a half adder

Universal Gates

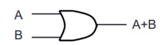
Gate Symbols



AND Gate

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

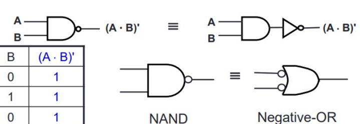


A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

(NOT gate)

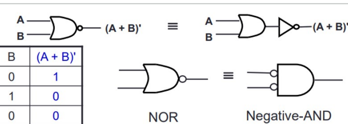
A	A'
0	1
1	0

- NAND gate



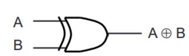
A	B	(A · B)'
0	0	1
0	1	1
1	0	1
1	1	0

- NOR gate



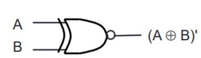
A	B	(A + B)'
0	0	1
0	1	0
1	0	0
1	1	0

- XOR gate



A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

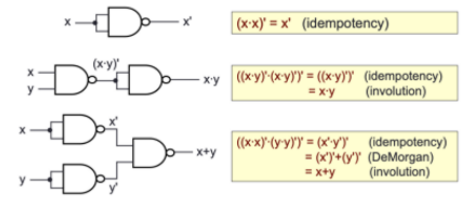
- XNOR gate



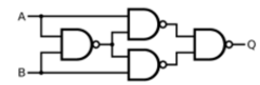
XNOR can be represented by ⊙
(Example: A ⊙ B)

A	B	(A ⊕ B)'
0	0	1
0	1	0
1	0	0
1	1	1

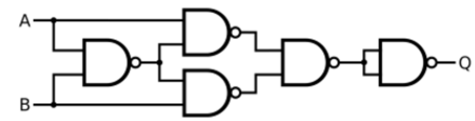
NAND as Universal Gate (Complete Logic Set)



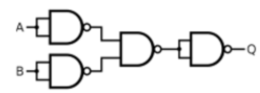
XOR



XNOR



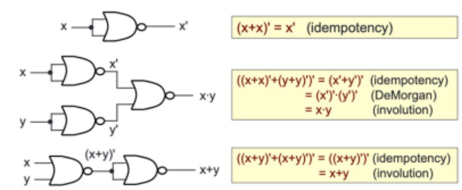
NOR



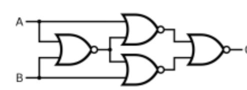
NOR as Universal Gate (Complete Logic Set)

NOR

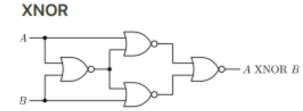
- Proof: Implement NOT/AND/OR using only NOR gates.



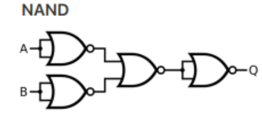
XOR



XNOR



NAND



Gray Code

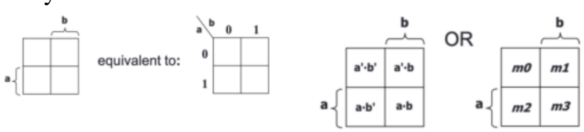
- Only a **single bit change** from one code value to the next. 4 bit standard gray code:

Decimal	Binary	Gray Code	Decimal	Binary	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

- not restricted to decimal digits: n bits can have up to 2^n values.
- aka reflected binary code. To generate gray code, reflect.
- not unique - multiple possible Gray code sequences

K Maps

- Simplify (SOP) expressions, with fewest possible product terms and literals.
- Based on **Unifying Theorem** ($A + A' = 1$)**, complement law.**
- Abstract form of Venn diagram, matrix of squares, each square represents a **minterm**.
- Two adjacent squares represent minterms that differ by exactly one literal.

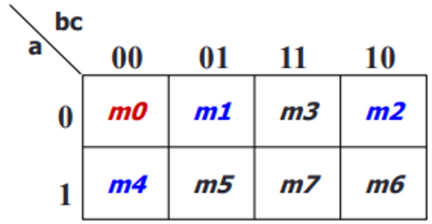


K Map for a function:

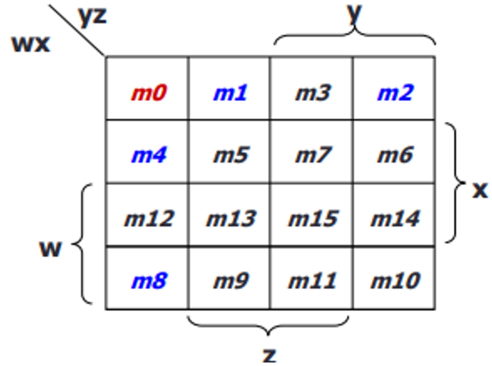
- The K-map for a function is filled by putting:
  - A **'1'** in the square the corresponds to a **minterm**
  - A '0' otherwise
- Each **valid grouping** of adjacent cells containing '1' corresponds to a simpler product term.
- Group must have width/length (size) in **powers of 2**.
- **larger group** = fewer literals in result product term
- **fewer groups** = fewer product terms in final SOP exp.
- Group maximum cells, and select fewest groups.

K-Maps

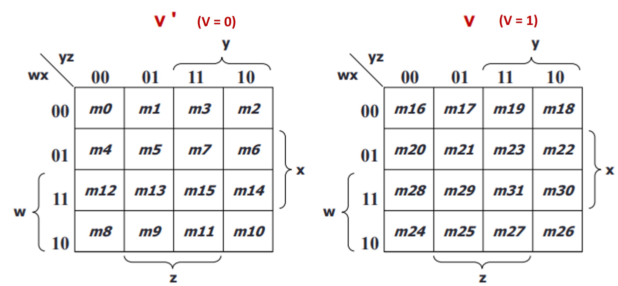
3-Variable



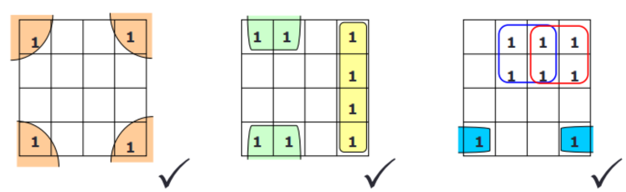
4-Variable



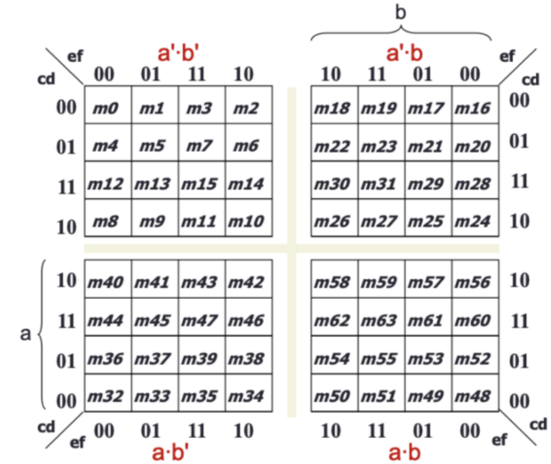
5-Variable



Valid Groupings



6-Variable



Using a K-map

- K-map of function easily filled in when function in sum-of-minterms form.
- If not in sum-of-minterms, convert into sum-of-products (SOP) form, expand SOP expr into sum-of-minterms, or fill directly based on SOP.

(E)PIs

- **implicant**: product term that could be used to cover minterms of the function.
- **prime implicant**: a product term obtained by combining the maximum possible number of minterms from adjacent squares in the map.
- **essential prime implicant**: a prime implicant that includes at least one minterm that is not covered by any other prime implicant

K-maps to find POS

- shortcut: group maxterms (0s) of given function
- long way: 1. convert K-map of F to K-map of F' (by flipping 0/1s), 2. get SOP of F' POS=(SOP)'.

Don't-Care Conditions

- denoted d, e.g.:

$F(A, B, C) = \sum m(3, 5, 6) + \sum d(0, 7)$