Logic gates

Boolean algebra
De Morgan's Laws
Simplify logic circuits
Truth tables for common logic circuits
Flip-flop & its role as data storage element
Karnaugh Maps and the benefits of using them
Solve logic problems using Karnaugh Maps

1

Boolean algebra basics

Logic Gates

- AND  A.B (reads as **A AND B**)
- OR  A+B (reads as **A OR B**)
- NOT  $\overline{A}$ (reads as **NOT A**)
- NAND  $\overline{A.B}$ (reads as **Not A AND B**)
- NOR  $\overline{A+B}$ (reads as **Not A OR B**)
- XOR  $A \oplus B$ (reads as **Exclusive A OR B**)

Inputs		NOT	AND	OR	NAND	NOR	XOR
A	B	$\overline{A}$	A . B	A + B	$\overline{A . B}$	$\overline{A + B}$	$A \oplus B$
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

2

Boolean algebra basics

Rules for Boolean Algebra

Identity/Law	**AND form**	**OR form**
Identity	1.A=A	0+A=A
Null	0.A=0	1+A=1
Idempotent	A.A=A	A+A=A
Inverse	$A.\bar{A}=0$	$A+\bar{A}=1$
Commutative	A.B=B.A	A+B=B+A
Associative	(A. B).C=A.(B.C)	(A+B)+C=A+(B+C)
Distributive	A+B.C=(A+B).(A+C)	A.(B+C)=A.B+A.C
Absorption	A.(A+B)=A	A+A.B=A
De Morgan's	$\overline{A.B}=\bar{A}+\bar{B}$	$\overline{(A+B)}=\bar{A}.\bar{B}$
Double Complement	$\bar{\bar{A}} = A$	

3

Boolean algebra basics

DeMorgan's Law

- $\overline{A.B}=\bar{A}+\bar{B}$
- The inverse of a Boolean product becomes the sum of the inverses of the individual values in the product
- $\overline{(A+B)}=\bar{A}.\bar{B}$
- The inverse of a Boolean sum is the product of the individual inverses
- **NOTE: AND is operated before OR**

4

Boolean algebra basics

- E.g. A.B + B.C.(B+C)
- **Distributive** law A.(B+C)=A.B+A.C used with **Commutative** Law A.B=B.A
- A.B + B.B.C + B.C.C
- AND form of the **Idempotent** identity A.A=A (B.B.C & B.C.C)
- A.B + B.C + B.C
- OR form of the **Idempotent** identity A+A=A
- A.B + B.C
- OR form on **Distributive** Law
- B.(A+C)

5

Boolean algebra basics

- E.g. Justifying your reasoning, show that:
- A + B(A + C) + AC ↔ A + BC

A + B(A + C) + AC

↓ Distributing terms

A + AB + BC + AC

↓ Applying rule **A + AB = A** to 1st and 2nd terms

A + BC + AC

↓ Applying rule **A + AB = A** to 1st and 3rd terms

A + BC

6

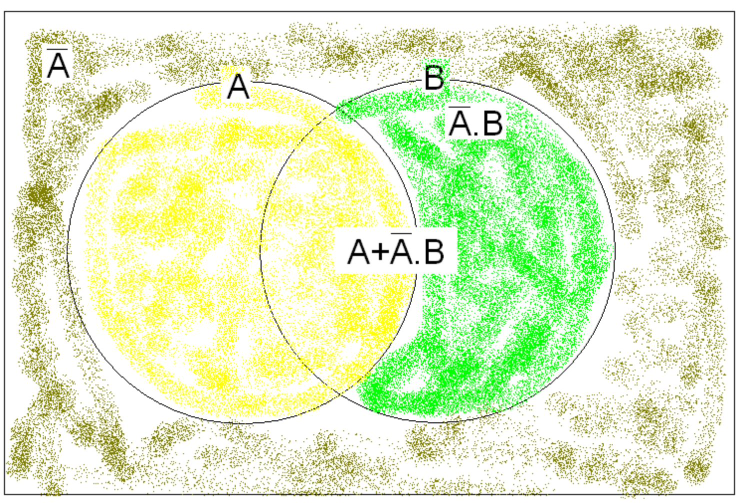
Boolean algebra basics

- E.g. A+Ā.B=A+B
- How? **Remember AND before OR**
- Note that with the **absorption identity** … A+A.B = A
- Thus the first A can be replaced by A+A.B
- → A+Ā.B can be written as A+A.B+Ā.B
- In the case of A.B+Ā.B using the **AND of the commutative law** then A.B+Ā.B = B.A+B.Ā
- Applying now the **OR form of the distributive law**, then B.A+B.Ā = B.(A+Ā)
- However thinking of the **inverse identity**, (A+Ā)=1
- Thus, A+A.B+Ā.B=A+B.1=A+B
- → A+Ā.B=A+B

7

Boolean algebra basics

- A+Ā.B=A+B



8

Simplify Boolean expressions

- Use the laws and identities of Boolean algebra to reduce Boolean expressions
- E.g. X = AB'C+ABC + (C+D)(D'+E)
- Using reverse OR form of DISTRIBUTIVE law
- X = AC(B'+B) + (C+D)(D'+E)
- Using OR form of INVERSE identity
- X = AC + (C+D)(D'+E)

9

Simplify Boolean expressions

- X = AC + (C+D)(D'+E)
- Using reverse AND form of DISTRIBUTIVE law
- X = AC + CD'+CE+DE+DD'
- Using AND form followed by OR form of INVERSE IDENTITY
- X = AC + CD'+CE+DE
- Thus,
- AB'C+ABC + (C+D)(D'+E) = AC+ CD'+CE+DE

10

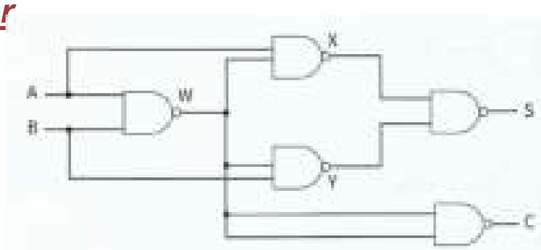
Simplify Boolean expressions

- E.g. X = (AB’(C+BD)+A’B’)C
- Using OR form of DISTRIBUTIVE law (open brackets)
- X =(AB’C +AB’BD+A’B’)C
- Using AND form of INVERSE identity
- X =(AB’C + +A’B’)C
- Using OR form of DISTRIBUTIVE law
- X =AB’CC + +A’B’C
- Finally, using AND form of IDEMPOTENT law
- X =AB’C + +A’B’C

11

Logic circuits

- ***Half Adder***

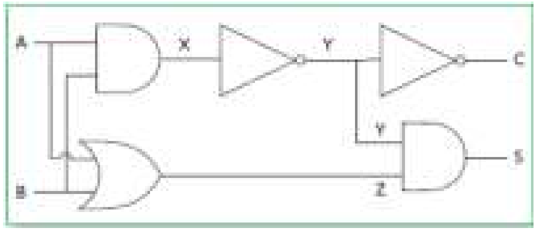


- *Create the truth table*

12

Logic circuits

Half Adder



Truth table

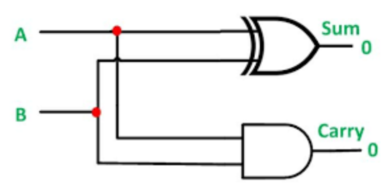
A	B	X	Y	Z	C	S
0	0	0	1	0	0	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	0	1	1	0

This arrangement of logic gates is used as an **accumulator** to add two numbers

13

Boolean algebra basics

Half Adder

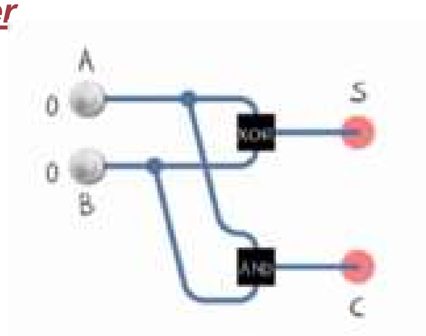


Inputs		Carry (AND)	Sum (XOR)
A	**B**	A . B	A ⊕ B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

14

Logic circuits

- ***Half Adder***

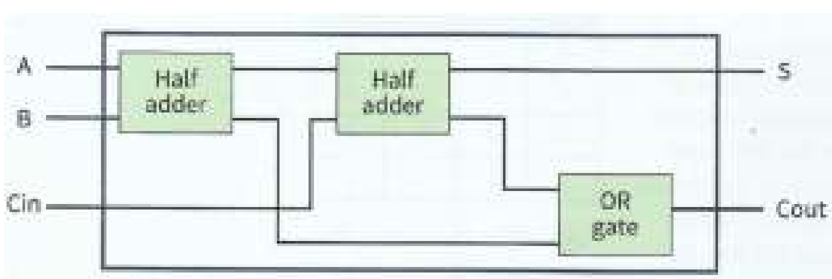


15

Boolean algebra basics

Full Adder

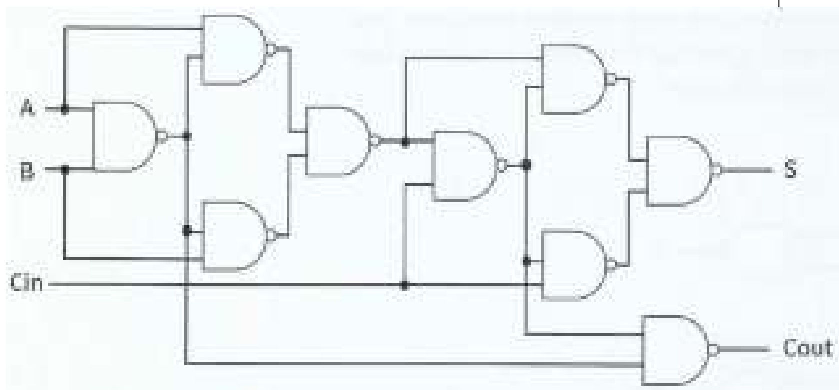
- Half adders can add only 2 bits
- If a sequence of binary additions needs to be performed, a third bit (carry bit) will be required



16

Boolean algebra basics

Full Adder



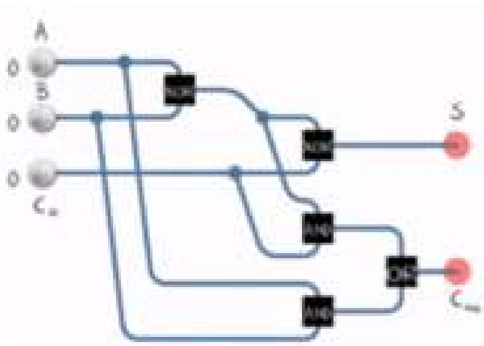
- *Create the truth table*

17

Boolean algebra basics

Full Adder

- Truth table



Inputs			Output	
A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

18

Boolean algebra basics

Flip-Flop Circuit

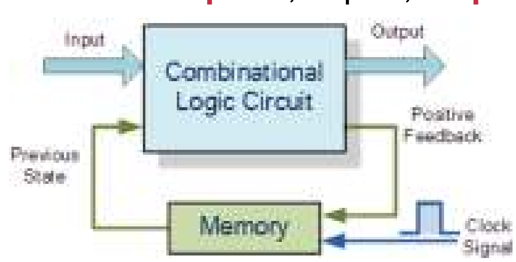
- **Combinational circuit:** a circuit in which the output is dependent only on the input values
- **Sequential circuit:** a circuit in which the output depends on the input values and the previous output
- Thus, the output of one stage affects the output of the next
- Think about **feedback**

19

Boolean algebra basics

Flip-Flop Circuit

- Flip-flop circuits use **sequential logic**
- Adders and other circuits were all examples of **combinational circuits**
- Sequential circuits employ **one or more inputs and one or more outputs**, whose **states** are related by defined rules that **depend**, in part, on **previous states**



20

Boolean algebra basics

Flip-Flop Circuit

- Are digital logic circuits that can be in one of two states (also called bistable gates)
- Maintain their state indefinitely until an input pulse called a trigger is received
- When a trigger is received, the flip-flop outputs change state according to defined rules and remain in those states until another trigger is received

21

Boolean algebra basics

Flip-Flop Circuit

- Flip-flop circuits are interconnected to form the logic gates for ICs used in memory chips and microprocessors
- Can be used to store one bit of data that represents the state of a sequencer, the value of a counter, an ASCII character in a computer's memory or any other piece of information

22

Boolean algebra basics

Flip-Flop Circuit

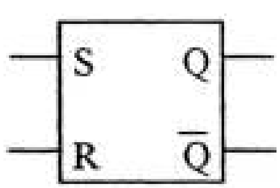
- There are several different kinds of flip-flop circuits:
  - S-R (set/reset)
  - J-K (possibly named for Jack Kilby - IC inventor)
  - T (toggle)
  - D (delay)
- A flip-flop typically includes zero, one, or two input signals as well as a clock signal and an output signal
- Some flip-flops also include a clear input signal to reset the current output

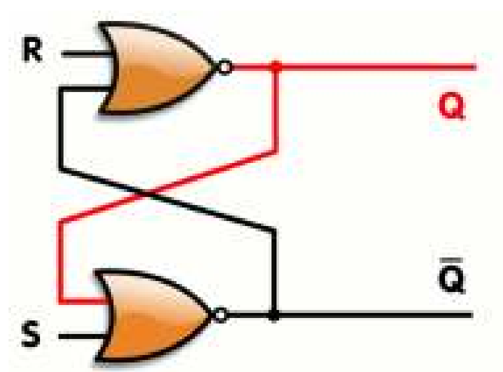
23

Boolean algebra basics

S-R Flip-Flop Circuit

- Also referred as 'latch'
- It can be constructed with 2 NAND or two NOR gates
- The two outputs are of different state





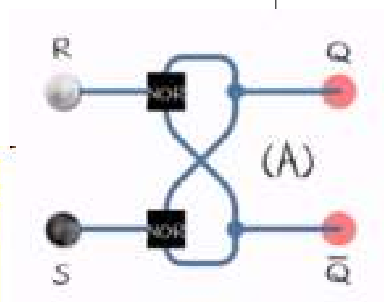
24

Boolean algebra basics

S-R Flip-Flop Circuit

- Truth Table

Inputs			Outputs		
S	R	C	Q	Q'	Comments
0	0	↑	Q	Q'	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	?	?	Invalid

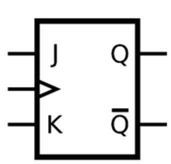


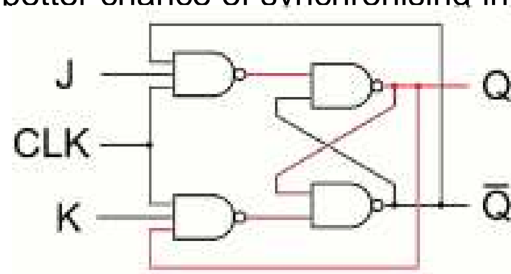
25

Boolean algebra basics

J-K Flip-Flop Circuit

- Similar to S-R Flip-Flop but with no invalid state
- A circuit may enter in an uncertain state when inputs do not arrive quite at the same time
- To prevent this, a circuit may include a clock pulse input to give a better chance of synchronising inputs





26

Boolean algebra basics

J-K Flip-Flop Circuit

- Truth Table

Inputs			Outputs		
J	K	C	Q	Q'	Comments
0	0	↑	Q	Q'	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	Q'	Q	Toggle

- Note that all actions take place only after a pulse is send/detected

27

Karnaugh maps (K-maps)

- Is a method of creating a Boolean algebra expression for a particular problem from a truth table
- Using the **sum-of-products approach** (only the 1's at the output are considered), a boolean expression is constructed
- The K-map can be easily used for circuits with 2, 3, or 4 inputs
- It consists of an array of cells, each representing a possible combination of inputs
- Each **cell** in a Karnaugh map **shows the value** of the **output X** for a **combination of input values** for **A** and **B**

28

Karnaugh maps (K-maps)

- The cells are arranged so that each cell's input combination differs from adjacent cells by only a single bit
- This is called **Gray code ordering** - it ensures that physical neighbours in the array are logical neighbours as well (neighbouring bit patterns are nearly the same, differing by only 1 bit)
- The interpretation of a Karnaugh map follows specific rules

29

Karnaugh maps (K-maps)

Karnaugh map rules:

- Only cells containing a 1 are considered
- Groups of cells containing 1s are identified where possible, with a group being a row, a column or a rectangle
- Groups must contain 2, 4, 8 and so on cells
- Each group should be as large as possible
- Groups can overlap
- If an individual cell cannot be contained in any group it is treated as being a group
- Within each group, the only input values retained are those which retain a constant value throughout the group

30

Karnaugh maps (K-maps)

- 2 bit inputs

A'B' 00	A'B 01
AB' 10	AB 11

A \ B	0	1
0		
1		

31

Karnaugh maps (K-maps)

- 3 bit inputs

A'B'C' 000	A'B'C 001	A'BC 011	A'BC' 010
AB'C' 100	AB'C 101	ABC 111	ABC' 110

Note that the numbers **are not in binary order**, but are arranged so that **only a single bit changes between neighbours**

A \ BC	00	01	11	10
0				
1				

32

Karnaugh maps (K-maps)

- 4 bit inputs

A'B'C'D' 0000	A'B'C'D 0001	A'B'CD 0011	A'B'CD' 0010
A'BC'D' 0100	A'BC'D 0101	A'BCD 0111	A'BCD' 0110
ABC'D' 1100	ABC'D 1101	ABCD 1111	ABCD' 1110
AB'C'D' 1000	AB'C'D 1001	AB'CD 1011	AB'CD' 1010

Note that the numbers **are not in binary order**, but are arranged so that **only a single bit changes between neighbours**

AB \ CD	00	01	11	10
00				
01				
11				
10				

33

Karnaugh maps (K-maps)

- After creating the truth table, check for any trends
- E.g. Any specific output due to specific inputs

- Whenever B=1, then X=1
- Thus, the expected final expression will be of the form B & (something else)

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

34

Karnaugh maps (K-maps)

- Write expressions that yield X=1 (**Sum of products**)
- $\overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C + A.B.\overline{C} + A.B.C$
- Combine input values in the columns
- Rows represents values of A
- Columns represent combinations of values for B and C

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

35

Karnaugh maps (K-maps)

- Write expressions that yield X=1
- $\overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C + A.B.\overline{C} + A.B.C$

Gray coding sequence

A \ BC	00	01	11	10
0	1	0	1	1
1	0	0	1	1

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

36

Karnaugh maps (K-maps)

- Write expressions that yield X=1
- $\bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.B.C + A.B.\bar{C} + A.B.C$

Gray coding sequence

A \ BC	00	01	11	10
0	1	0	1	1
1	0	0	1	1

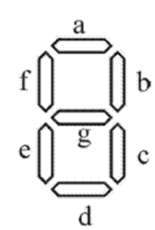
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- Reduces to:
- $\bar{A}.\bar{C}+B$

37

Karnaugh maps (K-maps)

- A practical example: the 7-segment display must represent numbers 0 - 9
- i.e. Requires 4 bits (3 bits ⇒ 0 - 7)
  - 0 is represented by 0000
  - 1 is represented by 0001
  - And so on, until
  - 9 is represented by 1001
- However 6 more combinations are “wasted” (1010-1111)
- Use “X” for these combinations and treat them similar to 1’s

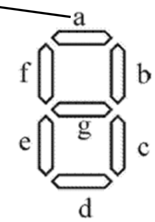


38

Karnaugh maps (K-maps)

- Truth table

A	B	C	D	Out a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



39

Karnaugh maps (K-maps)

A	B	C	D	Out a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

K-map

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	1
11	X	X	X	X
10	1	1	X	X

$a = AB' + A'C + A'BD + B'C'D'$

OR

$a = AB' + A'C + A'BD + A'B'D'$

Using the X's

$a = A + C + BD + B'C'D'$

40

Simplify Boolean expressions

- Use the laws and identities of Boolean algebra to reduce Boolean expressions
- E.g. X = AB'C+ABC + (C+D)(D'+E)
- Using reverse OR form of DISTRIBUTIVE law
- X = AC(B'+B) + (C+D)(D'+E)
- Using OR form of INVERSE identity
- X = AC + (C+D)(D'+E)

41

Simplify Boolean expressions

- X = AC + (C+D)(D'+E)
- Using reverse AND form of DISTRIBUTIVE law
- X = AC + CD'+CE+DE+DD'
- Using AND form followed by OR form of INVERSE IDENTITY
- X = AC + CD'+CE+DE
- Thus,
- AB'C+ABC + (C+D)(D'+E) = AC+ CD'+CE+DE

42

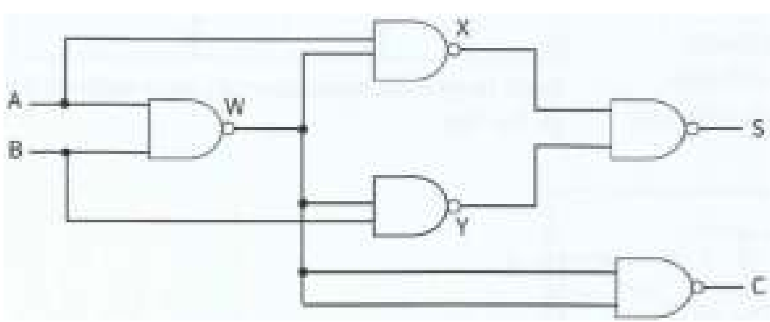
Simplify Boolean expressions

- E.g. X = (AB'(C+BD)+A'B')C
- Using OR form of DISTRIBUTIVE law (open brackets)
- X =(AB'C +AB'BD+A'B')C
- Using AND form of INVERSE identity
- X =(AB'C + +A'B')C
- Using OR form of DISTRIBUTIVE law
- X =AB'CC + +A'B'C
- Finally, using AND form of IDEMPOTENT law
- X =AB'C + +A'B'C

43

Simplify Boolean expressions

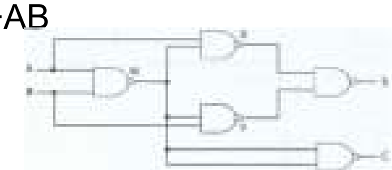
- The Half-Adder using NAND gates



44

Simplify Boolean expressions

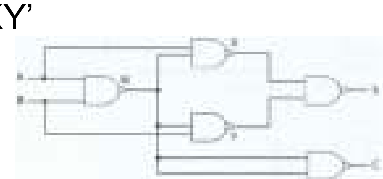
- W=A'B'+A'B+AB'
- X=A'W'+A'W+AW' (Note: W'=AB)
- X=A'AB+A'(A'B'+ A'B + AB')+AAB *(Distributive)*
- *Note that A'A=0 (Inverse) and AA=A* *(Idempotent)*
- X=A'AB+A'A'B'+A'A'B+A'AB'+AAB'
- X= 0 + A'B' + A'B + 0 + AB *(Idempotent)*
- X=A'B'+A'B+AB *(Distributive)*
- X=A'(B'+B)+AB *(Inverse)*
- X=A'+AB



45

Simplify Boolean expressions

- Likewise, to find Y:
- Y=B'W'+B'W+BW'
- Y=B'+AB
- Knowing that X=A'+AB and Y=B'+AB, try to find S and prove that **S=AB'+A'B**
- S=X'Y'+X'Y+XY'



46

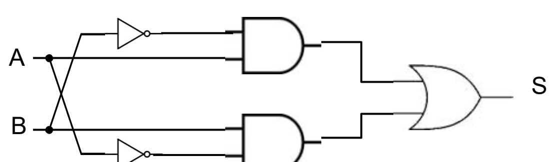
Simplify Boolean expressions

- S=(A’+AB)’(B’+AB)’+(A’+AB)’(B’+AB)+(A’+AB)(B’+AB)’
- *Using De Morgan’s Law*
- S=A’’(AB)’B’’(AB)’+A’’(AB)’(B’+AB)+(A’+AB)B’’(AB)’
- *Using Double Compliment, Inverse, Commutative & De Morgan’s*
- S=AB(AB)’+A(AB)’(B’+AB) +(A’+AB)B(AB)’
- S= 0 +A(A’+B’)(B’+AB)+(A’+AB)B(A’+B’)
- S=0+(AA’+AB’)(B’+AB)+(A’+AB)(A’B+BB’)
- S=0+(0 +AB’)(B’+AB) +(A’+AB)(A’B+0’)
- S=AB’B’+AAB’B +A’A’B+AA’BB
- S= AB’ + 0 + A’B + 0
- **S=AB’+A’B**

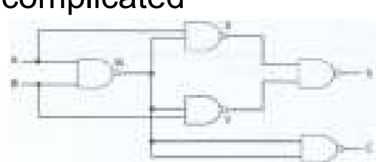
47

Simplify Boolean expressions

- Try now to create a different logic circuit using the equation: **S=AB’+A’B**



- Still complicated

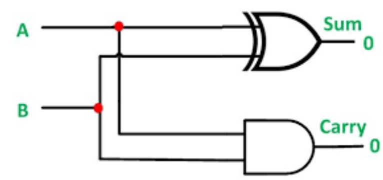


48

Simplify Boolean expressions

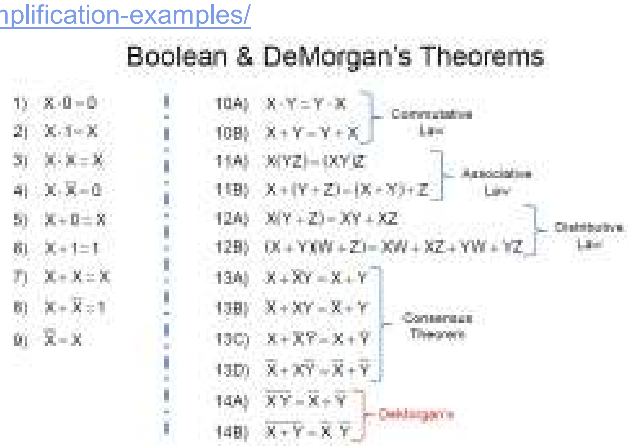
- Now look at the truth table of the half adder

Inputs		Carry (AND)	Sum (XOR)
A	**B**	A . B	A B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



49

- http://www.allaboutcircuits.com/textbook/digital/chpt-7/circuit-simplification-examples/


Boolean & DeMorgan's Theorems

1) $X \cdot 0 = 0$
2) $X \cdot 1 = X$
3) $X \cdot X = X$
4) $X \cdot \overline{X} = 0$
5) $X + 0 = X$
6) $X + 1 = 1$
7) $X + X = X$
8) $X + \overline{X} = 1$
9) $\overline{\overline{X}} = X$

10A) $X \cdot Y = Y \cdot X$
10B) $X + Y = Y + X$
Commutative Law

11A) $X(YZ) = (XY)Z$
11B) $X + (Y + Z) = (X + Y) + Z$
Associative Law

12A) $X(Y + Z) = XY + XZ$
12B) $(X + Y)(W + Z) = XW + XZ + YW + YZ$
Distributive Law

13A) $X + \overline{X}Y = X + Y$
13B) $\overline{X} + XY = \overline{X} + Y$
13C) $X + \overline{X}\overline{Y} = X + \overline{Y}$
13D) $\overline{X} + X\overline{Y} = \overline{X} + \overline{Y}$
Consensus Theorem

14A) $\overline{XY} = \overline{X} + \overline{Y}$
14B) $\overline{X + Y} = \overline{X}\,\overline{Y}$
DeMorgan's

50