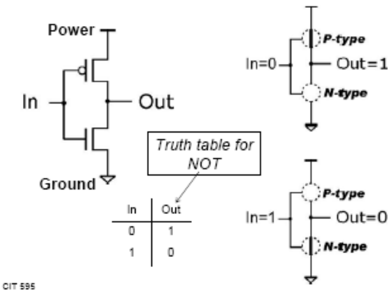
Dept. of Computer Science and Engineering
University of Rajshahi
www.ru.ac.bd

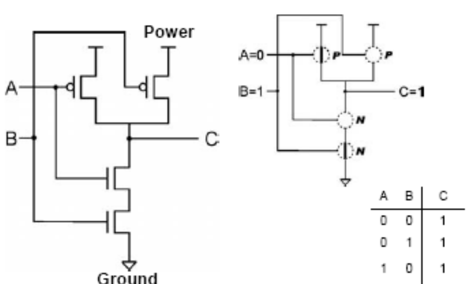
Dr. Shamim Ahmad

Inverter (NOT Gate) using CMOS



In	Out
0	1
1	0

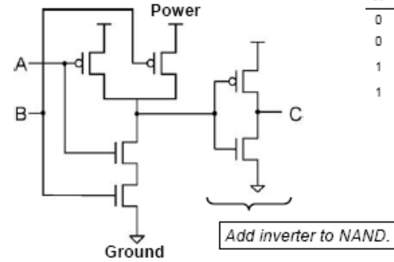
NAND Gate (NOT-AND) using CMOS



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

AND Gate using CMOS

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



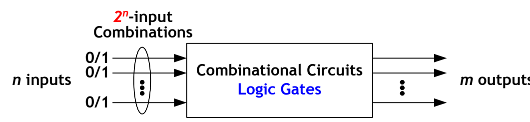
Combinational and Sequential Circuits

Logic Circuit

Combinational Circuit:
- Output only depends on the present combination of inputs
- Specified by a set of Boolean Functions

Sequential Circuit:
- Output depends on the input and the state of the storage (past inputs)

Block Diagram of Combinational Circuits



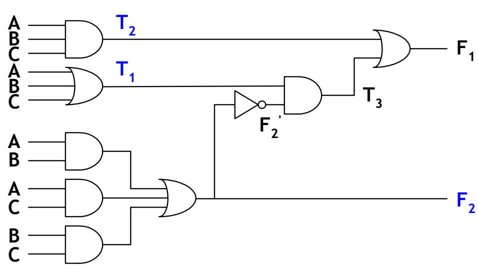
Analysis Procedure of a Combinational Circuit

1. Make sure the given circuit is a combinational circuit
   - Combinational Circuit without feedback paths or memory elements
   - Feedback paths in digital circuits define a sequential circuit
2. Obtain the output Boolean functions or the truth table

Procedure to Obtain the Output Boolean Functions from a Logic Diagram

1. Label all gate outputs that are a function of input variables with arbitrary symbol. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outline in step 2 until the outputs of the circuits are obtained
4. By repeated substitution of previously defined functions, obtain the output Boolean function in terms of input variables.

Procedure Example



Procedure Example

Step 1:	F_2 = AB + AC + BC T_1 = A + B + C T_2 = ABC
Step 2:	$T_3 = F_2'T_1$ $F_1 = T_3 + T_2$
Step 3-4:	$F_1 = T_3 + T_2 = F_2'T_1$ + ABC = (AB + AC + BC)' (A+B+C) + ABC = A'BC' + A'B'C + AB'C' + ABC

Procedure to Obtain the Output Boolean Functions from the Truth Table

1. Determine the number of input variables in the circuit. For *n* inputs, form the 2^n possible input combinations and list the binary numbers from 0 to 2^n-1 in a table
2. Label the output of the selected gates with arbitrary symbols
3. Obtain the truth table for the outputs of those gates that are a function of the input variables only
4. Proceed to obtain the truth table for the outputs of those gates that are a function of previously defined values until the columns for all outputs are determined

Procedure Example

Procedure →

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Truth Table for Example

Boolean Operator

- A Boolean operator can be completely described using a truth table
- The truth table for the Boolean operators AND and OR are shown at the right
- The AND operator is also known as a "Boolean product". It is also represented with dot symbol. E.g x.y
- The OR operator is the "Boolean sum". It is also represented with '+' symbol. E.g. x + y
- The NOT operation is most often designated by an overbar. It is sometimes indicated by a prime mark (') or an "elbow" (¬) or tilda (~)

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\bar{X}$
0	1
1	0

Boolean Operator Precedence

- As with common arithmetic, Boolean operations have rules of precedence
- The NOT operator has highest priority, followed by AND and then OR
- Hence to evaluate the expression, z is negated first, then x is ANDed with the previous result and finally ORed with y

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Boolean Identity Group I

- Most Boolean identities have an AND (product) form as well as an OR (sum) form. We give our identities using both forms. Our first group is rather intuitive:

Identity Name	AND Form	OR Form
Identity Law	1x = x	0 + x = x
Null Law	0x = 0	1 + x = 1
Idempotent Law	xx = x	x + x = x
Inverse Law	$x\bar{x} = 0$	$x + \bar{x} = 1$

Boolean Identity Group II

- Our second group of Boolean identities should be familiar to you from your study of algebra:

Identity Name	AND Form	OR Form
Commutative Law	xy = yx	x+y = y+x
Associative Law	(xy)z = x(yz)	(x+y)+z = x + (y+z)
Distributive Law	x+yz = (x+y)(x+z)	x(y+z) = xy+xz

Boolean Identity Group III

- Our last group of Boolean identities are perhaps the most useful.
- If you have studied set theory or formal logic, these laws are also familiar to you.

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \bar{x} + \bar{y}$	$\overline{(x+y)} = \bar{x}\bar{y}$
Double Complement Law	$\overline{(\bar{x})} = x$	

Example using Boolean Identity

- We can use Boolean identities to simplify the function:

$(x + y)(x' + y)$

$= xx' + xy + yx' + yy$	Distributive Law
$= 0 + xy + yx' + y$	Inverse & Idempotent Law
$= xy + yx' + y$	Identity Law
$= y(x + x') + y$	Distributive Law
$= y(1) + y$	Inverse Law
$= y + y$	Identity Law
$= y$	Idempotent Law

De Morgan's Law

- Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly
- DeMorgan's law provides an easy way of finding the complement of a Boolean function

$$\overline{(xy)} = \bar{x} + \bar{y} \quad \text{and} \quad \overline{(x+y)} = \bar{x}\bar{y}$$

Standard or Canonical Form

- There are two canonical forms for Boolean expressions: *sum-of-products* and *product-of-sums*
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  - For example: $F(x,y,z) = xy + xz + yz$
- In the product-of-sums form, ORed variables are ANDed together:
  - For example: $F(x,y,z) = (x+y)(x+z)(y+z)$

Conversion to Sum-of-Products Form

- It is easy to convert a function to sum-of-products form using its truth table
- We are interested in the values of the variables that make the function "true" (i.e. output 1)
- Using the truth table, we list the values of the variables that result in a true value
  - The variables corresponding to row with output 1 are "ANDed"
  - If the variable's input value is 1 then it is written as it is else the complement of that variable is written
- Each group of variables is then "Ored" together

$F(x,y,z) = x\bar{z}+y$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

CIT 595 3 - 24

Conversion to Sum-of-Products form (contd..)

- The sum-of-products form for function is:

$$F(x,y,z) = \bar{x}y\bar{z}+\bar{x}yz+x\bar{y}\bar{z}+xy\bar{z}+xyz$$

One ANDed Group is known as Minterm

Note: This function is not in simplest terms. It was just show how the function can be rewritten in canonical sum-of-products form.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

CIT 595 3 - 25

Conversion to Product-of-Sums

- We are interested in the values of the variables that make the function "false" (i.e. output 0)
- Using the truth table, we list the values of the variables that result in a false value
  - The variables corresponding to row with output 0 are "ORed"
  - If the variable's input value is 0 then it is written as it is else the complement of that variable is written
- Each group of variables is then "ANDed" together

$F(x,y,z) = x\bar{z}+y$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

CIT 595 3 - 26

Conversion to Product-of-Sums (contd..)

- The sum-of-products form for function is:

$$(x+y+z).(x+y+\bar{z}).(\bar{x}+y+\bar{z})$$

One ORed Group is known as Maxterm

$F(x,y,z) = x\bar{z}+y$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

CIT 595 3 - 27

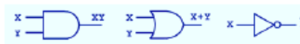
Logic Gate

- We have looked at Boolean functions in abstract terms
- In this section, we see that Boolean functions are implemented in digital computer circuits are called gates
- A gate is an electronic device that produces a result based on one or more input values
  - In reality, gates consist of one to six transistors, but digital designers think of them as a single unit
  - Integrated circuits contain collections of gates suited to a particular purpose

CIT 595 3 - 28

Basic Gates

- The three simplest gates are the AND, OR, and NOT gates



X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\bar{X}$
0	1
1	0

- They correspond directly to their respective Boolean operations, as you can see by their truth tables

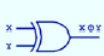
CIT 595 3 - 29

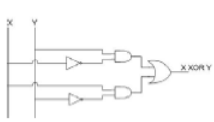
XOR Gate

- Another very useful gate is the exclusive OR (XOR) gate
- The output of the XOR operation is true only when the values of the inputs differ

X XOR Y

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0





Note the special symbol $\oplus$ for the XOR operation.

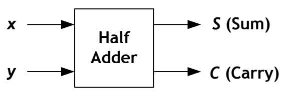
Sum of Product Form: $\bar{X}y + x\bar{y}$

CIT 595 3 - 30

- **4 Possible Operations for Addition of Two Binary Digits**

0 + 0 = 0	0 + 1 = 1
1 + 0 = 1	1 + 1 = 10 (1 = Carry)

Half Adder

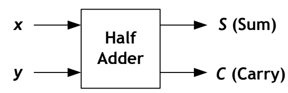


x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder Truth Table

$S = x'y + xy'$
$C = xy$

Half Adder



x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder Truth Table

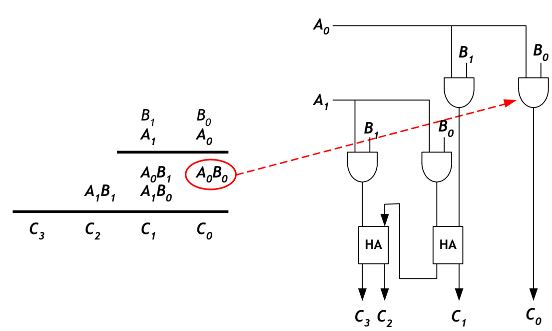
$S = x'y + xy'$
$C = xy$

Full Adder

1. Full adders perform the arithmetic sum of three bits
2. Full adders is implemented by a 3-input 2-output combinational circuit
3. Truth Table:

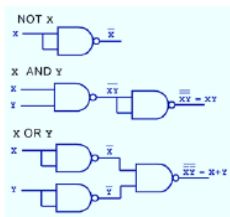
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Binary Multiplier



Universal Gates

- NAND and NOR are known as *universal gates* because they are inexpensive to manufacture and any Boolean function can be constructed using only NAND or only NOR gates



CIT 595 3 - 32

NAND and NOR gates

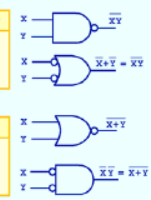
- NAND and NOR are two very important gates. Their symbols and truth tables are shown at the right

X NAND Y

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0



CIT 595 3 - 31

Kmap for Sum-Of-Product Form

- The output values placed in each cell of the matrix are derived from the "minterms" of a Boolean function
- A *minterm* is a product term that contains all of the function's variables exactly once, either complemented or not complemented

CIT 595 3 - 38

Minterm Example with Two variables

- For example, the minterms for a function having the inputs *x* and *y* are: $\overline{X}\,\overline{Y}$, $\overline{X}Y$, $X\overline{Y}$, and XY

Minterm	X	Y
$\overline{X}\,\overline{Y}$	0	0
$\overline{X}Y$	0	1
$X\overline{Y}$	1	0
XY	1	1

Note: If variable input is 1, then it is written as it is else the complement of that variable is written

CIT 595 3 - 39

Minterm Example with Three variables

- Similarly, a function having three inputs, has the minterms that are shown in this diagram

Minterm	X	Y	Z
$\bar{x}\bar{y}\bar{z}$	0	0	0
$\bar{x}\bar{y}z$	0	0	1
$\bar{x}y\bar{z}$	0	1	0
$\bar{x}yz$	0	1	1
$x\bar{y}\bar{z}$	1	0	0
$x\bar{y}z$	1	0	1
$xy\bar{z}$	1	1	0
xyz	1	1	1

CIT 595 3 - 40

Kmap Cell using SOP form: Example 1

- A Kmap has a cell for each minterm
- This means that it has a cell for each line for the truth table of a function
- The truth table for the function *F(x,y) = xy* is shown at the right along with its corresponding Kmap

F(X,Y) = XY

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X \ Y	0	1
0	0	0
1	0	1

CIT 595 3 - 41

Kmap Cell using SOP Form: Example 2

- As another example, we give the truth table and Kmap for the function, *F(x,y) = x + y*
- This function is equivalent to the OR of all of the minterms that have a value of 1 (Sum of Product Form). Thus:

$$F(x,y) = X+Y = \bar{X}Y + X\bar{Y} + XY$$

F(X,Y) = X+Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

X \ Y	0	1
0	0	1
1	1	1

CIT 595 3 - 42

Kmap Simplification for Two Variables using SOP Form

- Of course, the minterm function that we derived from our Kmap was not in simplest terms
  - That's what we started with in this example
- We can, however, reduce our complicated expression to its simplest terms by finding adjacent 1s in the Kmap that can be collected into groups that are "powers of two"
- In our example, we have two such groups
  - Can you find them?

X \ Y	0	1
0	0	1
1	1	1

CIT 595 Example 2 3 - 43

Reduced Expression for Example 2

- In the "green" group (vertical), it does not matter what value x has, hence the group is only dependent on variable y
- Similarly in the "pink" group (horizontal), it does not matter what value y has, the group is only dependent on variable x
- Hence the Boolean function reduces to x + y

x \ y	0	1
0	0	1
1	1	1

Kmap Simplification for Two Variables using SOP Form

- The best way of selecting two groups of 1s form our simple Kmap is shown below
- We see that both groups are powers of two and that the groups overlap.
- The next slide gives guidance for selecting Kmap groups

x \ y	0	1
0	0	1
1	1	1

Rules for Kmap Simplification using Sum of Products Form (SOP)

The rules of Kmap simplification are:

- Groupings can contain only 1s; no 0s
- Groups can be formed only at right angles; diagonal groups are not allowed
- The number of 1s in a group must be a power of 2 – even if it contains a single 1
- The groups must be made as large as possible
- Groups can overlap and wrap around the sides of the Kmap

Kmap with Three Variables (SOP form)

- A Kmap for three variables is constructed as shown in the diagram below
- We have placed each minterm in the cell that will hold its value
  - Notice that the values for the *yz* combination at the top of the matrix form a pattern that is not a normal binary sequence
  - A Kmap must be ordered so that each minterm differs only in one variable from each neighboring cell hence 11 appears before 10 – Rule!! (will help simplification)

x \ yz	00	01	11	10
0	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$\bar{x}y\bar{z}$
1	$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz	$xy\bar{z}$

Kmap with Three Variables (SOP Form)

Note:

- Thus, the first row of the Kmap contains all minterms where *x* has a value of zero
- The first column contains all minterms where *y* and *z* both have a value of zero

X \ YZ	00	01	11	10
0	$\bar{X}\bar{Y}\bar{Z}$	$\bar{X}\bar{Y}Z$	$\bar{X}YZ$	$\bar{X}Y\bar{Z}$
1	$X\bar{Y}\bar{Z}$	$X\bar{Y}Z$	XYZ	$XY\bar{Z}$

CIT 595 3 - 48

Kmap - Three Variable (SOP Form): Example 1

- Consider the function:

$$F(X,Y,Z) = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}Z + XYZ$$

- Its Kmap is given below:
  - What is the largest group of 1s that is a power of 2?

X \ YZ	00	01	11	10
0	0	1	1	0
1	0	1	1	0

CIT 595 3 - 49

Kmap - Three Variable (SOP form): Example1

- This grouping tells us that changes in the variables *x* and *y* have no influence upon the value of the function: They are irrelevant
- This means that the function,

$$F(X,Y,Z) = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}Z + XYZ$$

reduces to *F(X,Y,Z) = Z*

You could verify this reduction with identities or a truth table.

X \ YZ	00	01	11	10
0	0	1	1	0
1	0	1	1	0

CIT 595 3 - 50

Kmap - Three Variable (SOP Form): Example 2

- Now for a more complicated Kmap. Consider the function:

$$F(X,Y,Z) = \bar{X}\bar{Y}\bar{Z} + \bar{X}\bar{Y}Z + \bar{X}YZ + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XY\bar{Z}$$

- Its Kmap is shown below. There are (only) two groupings of 1s.
  - Can you find them?

X \ YZ	00	01	11	10
0	1	1	1	1
1	1	0	0	1

CIT 595 3 - 51

Kmap - Three Variable (SOP form): Example 2

- In this Kmap, we see an example of a "group that wraps around the sides" of a Kmap.
- This group tells us that the values of *x* and *y* are not relevant to the term of the function that is encompassed by the group
  - ➢ What does this tell us about this term of the function?
    - It is dependent on $\overline{Z}$

What about the green group in the top row?

x \ yz	00	01	11	10
0	1	1	1	1
1	1	0	0	1

Kmap - Three Variable (SOP): Example 2

- The "green group" in the top row tells us that only the value of *x* is significant in that group.
- We see input value of x is 0 i.e. minterm is complemented in that row, so the other term of the reduced function is $\overline{x}$
- Our reduced function is: $F(x,y,z) = \overline{x} + \overline{z}$

Recall that we had six minterms in our original function !! The function is considerably minimized

x \ yz	00	01	11	10
0	1	1	1	1
1	1	0	0	1

Kmap Simplification for Four Variables (SOP Form)

- The model can be extended to accommodate the 16 minterms that are produced by a four-input function
- This is the format for a 16-minterm Kmap

WX \ YZ	00	01	11	10
00	$\overline{W}\overline{X}\overline{Y}\overline{Z}$	$\overline{W}\overline{X}\overline{Y}Z$	$\overline{W}\overline{X}YZ$	$\overline{W}\overline{X}Y\overline{Z}$
01	$\overline{W}X\overline{Y}\overline{Z}$	$\overline{W}X\overline{Y}Z$	$\overline{W}XYZ$	$\overline{W}XY\overline{Z}$
11	$WX\overline{Y}\overline{Z}$	$WX\overline{Y}Z$	$WXYZ$	$WXY\overline{Z}$
10	$W\overline{X}\overline{Y}\overline{Z}$	$W\overline{X}\overline{Y}Z$	$W\overline{X}YZ$	$W\overline{X}Y\overline{Z}$

Kmap Four Variables (SOP Form) Example

- We have populated the Kmap shown below with the nonzero minterms from the function:

$$F(W,X,Y,Z) = \overline{W}\overline{X}\overline{Y}\overline{Z} + \overline{W}\overline{X}\overline{Y}Z + \overline{W}\overline{X}Y\overline{Z} + \overline{W}XY\overline{Z} + W\overline{X}\overline{Y}\overline{Z} + W\overline{X}\overline{Y}Z + W\overline{X}Y\overline{Z}$$

  - ➢ Can you identify (only) three groups in this Kmap?

Recall the Rules of Simplification

WX \ YZ	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

Kmap Four Variables (SOP Form) Example

- The three groups consist of:
  - A purple group entirely within the Kmap at the right
  - A pink group that wraps the top and bottom
  - A green group that spans the corners
- Thus we have three terms in our final function:

$F(W,X,Y,Z) = \bar{X}\bar{Y} + \bar{X}\bar{Z} + \bar{W}Y\bar{Z}$

WX \ YZ	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

Choosing Kmap Groups

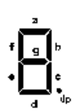
- It is possible to have a choice as to how to pick groups within a Kmap, while keeping the groups as large as possible
- The (different) functions that result from the groupings below are logically equivalent

WX \ YZ	00	01	11	10
00	1		1	
01	1		1	1
11	1			
10	1			

WX \ YZ	00	01	11	10
00	1		1	
01	1		1	1
11	1			
10	1			

Don't Care Conditions

- Real circuits don't always need to have an output defined for every possible input
  - For example, some calculator displays consist of 7-segment LEDs. These LEDs can display 2^7 -1 patterns, but only ten of them are useful
- If a circuit is designed so that a particular set of inputs can never happen, we call this set of inputs a *don't care* condition
- They are very helpful to us in Kmap circuit simplification



Don't Care Example (SOP Form)

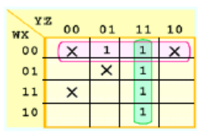
- In a Kmap, a don't care condition is identified by an *X* in the cell of the minterm(s) for the don't care inputs, as shown below
- In performing the simplification, we are free to include or ignore the *X*'s when creating our groups

WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11	X		1	
10			1	

Don't Care Example (SOP Form)

- In one grouping in the Kmap below, we have the function:

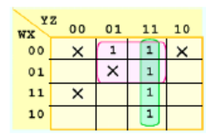
$$F(W,X,Y,Z) = \bar{W}\bar{X} + YZ$$



Don't Care Example w/ Different Grouping

- A different grouping gives us the function:

$$F(W,X,Y,Z) = \bar{W}Z + YZ$$

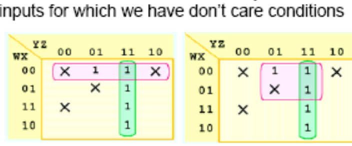


Don't Care Condition Example

- The truth table of: $F(W,X,Y,Z) = \bar{W}Z + YZ$
  is different from the truth table of:

$$F(W,X,Y,Z) = \bar{W}\bar{X} + YZ$$

- However, the values for which they differ, are the inputs for which we have don't care conditions



Kmap using Product-of-Sum (POS) Form

- The output values placed in each cell are derived from the "maxterm" of a Boolean function
- A *maxterm* is a sum term that contains all of the function's variables exactly once, either complemented or not complemented

Maxterm Example

X	Y	Maxterm
0	0	X + Y
0	1	X + Y'
1	0	X' + Y
1	1	X' + Y'

Note: If variable input is 0, then it is written as it is else the complement of that variable is written

Kmap Rules using Product-of-Sum Form

- Groupings can contain only 0s; no 1s
- Groups can be formed only at right angles; diagonal groups are not allowed
- The number of 0s in a group must be a power of 2 – even if it contains a single 0
- The groups must be made as large as possible
- Groups can overlap and wrap around the sides of the Kmap
- Use don't care conditions when you can

Binary Adder-Subtractor

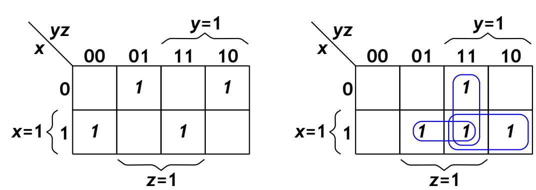
- Full Adder

1. Full adders perform the arithmetic sum of three bits
2. Full adders is implemented by a 3-input 2-output combinational circuit
3. Truth Table:

x	y	z	C	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

Binary Adder-Subtractor

- K-Maps for Full Adders

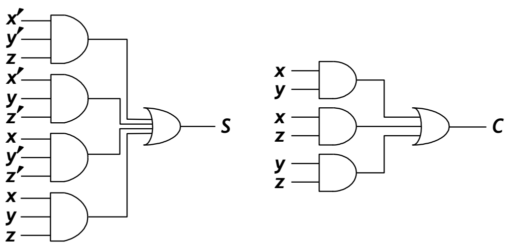


$S = xy'z' + x'yz' + xyz + x'y'z$
$= x \oplus y \oplus z$

$C = xy + xz + yz$
$= xy + xy'z + x'yz$

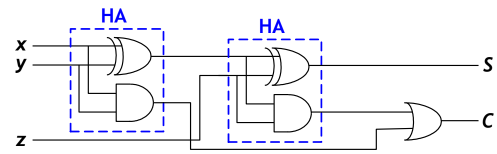
Binary Adder-Subtractor

- SOP Logic Implementations of Full Adders



Binary Adder-Subtractor

- Full Adder Implementation with Two Half Adders and an OR Gate



Binary Adder-Subtractor

- Binary Adders

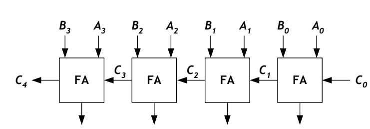
1. Binary adders perform the arithmetic sum of two numbers
2. Binary adders can be constructed with full adders connected in cascade

Binary Adder-Subtractor

- 4-bits Binary Adders

Subscript:	3	2	1	0	
Input Carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	+ 0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output Carry	0	0	1	1	C_{i+1}

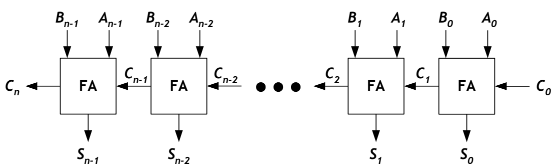
4-bit Addition Example



4-bit Adder

Binary Adder-Subtractor

■ *n*-bits Binary Adders

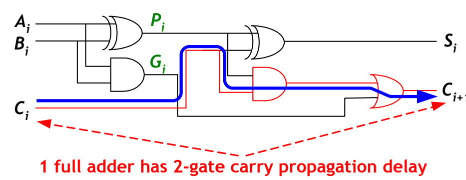


n-bit Adder

Binary Adder-Subtractor

■ Carry Propagation Delay:

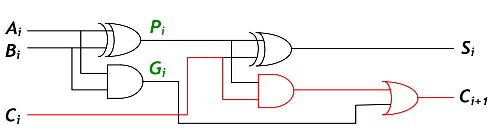
N-bit adder has *2n* gate carry propagation delay !!



Binary Adder-Subtractor

■ Carry Lookahead: Reduce Carry Propagation Delay

G_i : Carry Generate
P_i : Carry Propagate



$$\begin{cases} P_i = A_i \oplus B_i \\ G_i = A_iB_i \end{cases} \rightarrow \begin{cases} S_i = P_i \oplus C_i \\ C_{i+1} = G_i + P_iC_i \end{cases}$$

Binary Adder-Subtractor

■ Carry Lookahead: Carry Bits

C_0 = Input Carry

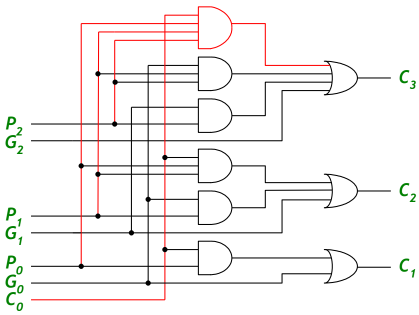
$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

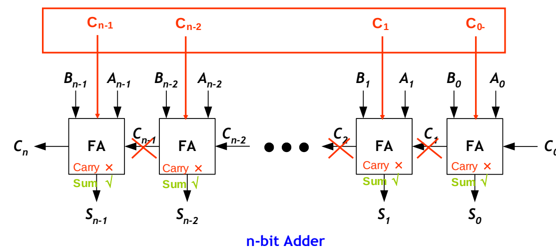
Binary Adder-Subtractor

- Carry Lookahead Generator



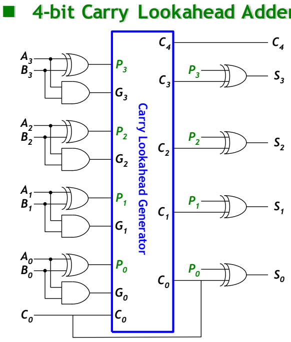
Binary Adder-Subtractor

n-bits Binary Adders (Carry look Ahead)



Binary Adder-Subtractor

- 4-bit Carry Lookahead Adder



Binary Adder-Subtractor

- Binary Subtractor

1. Implement subtraction with 2's complement number system
2. A-B = A + (-B) = A + 1'sc (B) + 1
3. Implement 1'sc with XOR gates:

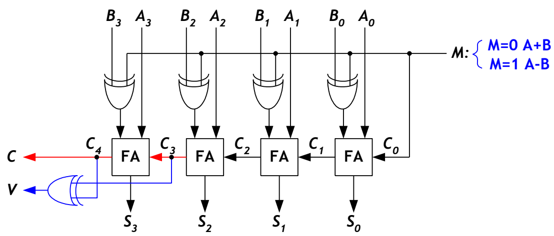
B	*M*	*Output*
0	0	0
0	1	1
1	0	1
1	1	0

1'sc (Output = B) when M=0

1'sc (Output = $\bar{B}$) when M=1

Binary Adder-Subtractor

- **4-Bit Binary Adder/Subtractor**



Binary Adder-Subtractor

- **Overflow:** When two numbers of *n* digits each are added and the sum occupies *n+1* digits, we say that an overflow occurred.

8-bit 2'sc number presents [-128, +127]