

Gate-Level Minimization

Logic and Digital System Design - CS 303

Erkay Savaş

Sabanci University

Complexity of Digital Circuits

- Directly related to the complexity of the algebraic expression we use to build the circuit.
- Truth table
 - may lead to different implementations
 - Question: which one to use?
- Optimization techniques of algebraic expressions
 - So far, ad hoc.
 - Need more systematic (algorithmic) way
 - Karnaugh (K-) map technique
 - Quine-McCluskey

Two-Variable K-Map

- To variable: x and y
 - 4 minterms:
 - $m_0 = x'y'$ $\rightarrow 00$
 - $m_1 = x'y$ $\rightarrow 01$
 - $m_2 = xy'$ $\rightarrow 10$
 - $m_3 = xy$ $\rightarrow 11$

		y	
		0	1
x	0	m_0	m_1
	1	m_2	m_3

		y	
		0	1
x	0	$x'y'$	$x'y$
	1	xy'	xy

Example: Two-Variable K-Map

		y	
		0	1
x	0	1	1
	1	1	

- $F = m_0 + m_1 + m_2 = x'y' + x'y + xy'$
- $F = x'(y + y') + xy'$
- $F = x' + xy'$
- $F = (x' + x)(x' + y')$
- $F = x' + y'$
- We can do the same optimization by combining adjacent cells.

Three-Variable K-Map

		yz			
		00	01	11	10
x	0	m_0	m_1	m_3	m_2
	1	m_4	m_5	m_7	m_6

- Adjacent squares: they differ by only one variable, which is primed in one square and not primed in the other
 - $m_2 \leftrightarrow m_6$, $m_3 \leftrightarrow m_7$
 - $m_2 \leftrightarrow m_0$, $m_6 \leftrightarrow m_4$

Example: Three-Variable K-Map

- $F_1(x, y, z) = \Sigma (2, 3, 4, 5)$

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

- $F_1(x, y, z) = xy' + x'y$
- $F_2(x, y, z) = \Sigma (3, 4, 6, 7)$

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

- $F_1(x, y, z) = xz' + yz$

Three Variable Karnaugh Maps

- One square represents one minterm with three literals
- Two adjacent squares represent a term with two literals
- Four adjacent squares represent a term with one literal
- Eight adjacent squares produce a function that is always equal to 1.

Example

- $F_1(x, y, z) = \Sigma (0, 2, 4, 5, 6)$

		y			
		yz		11	10
x	0	1	0	0	1
	1	1	1	0	1

Diagram annotations: A bracket labeled 'y' groups the columns 11 and 10. A bracket labeled 'z' groups the columns 01 and 11. A bracket labeled 'x' groups the rows 0 and 1.

$$F_1(x, y, z) = z' + xy'$$

Finding Sum of Minterms

- If a function is not expressed in sum of minterms form, it is possible to get it using K-maps

- Example: $F(x, y, z) = x'z + x'y + xy'z + yz$

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

$$F(x, y, z) = x'yz + x'y'z + x'yz + x'y'z' + xy'z + xyz + x'yz$$

$$F(x, y, z) = z + x'y$$

Four-Variable K-Map

- Four variables: x, y, z, t
 - 4 literals
 - 16 minterms

The diagram shows a 4x4 grid of minterms for four variables: x, y, z, t . The grid is labeled with xy on the vertical axis and zt on the horizontal axis. Red curly braces and labels indicate the mapping of the grid to the variables: z for the horizontal axis (top), y for the vertical axis (right), x for the vertical axis (left), and t for the horizontal axis (bottom).

$xy \backslash zt$	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Example: Four-Variable K-Map

- $F(x,y,z,t) = \Sigma (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

xy \ zt				
	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	0

- $F(x,y,z,t) = z' + x't' + yt'$

Example: Four-Variable K-Map

- $F(x,y,z,t) = x'y'z' + y'zt' + x'yz't' + xy'z'$

xy \ zt				
	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- $F(x,y,z,t) = y'z' + y't' + x'zt'$

Prime Implicants

- A product term obtained by combining maximum possible number of adjacent squares in the map
- If a minterm is covered by only one prime implicant, that prime implicant is said to be essential.
 - A single 1 on the map represents a prime implicant if it is not adjacent to any other 1's.
 - Two adjacent 1's form a prime implicant, provided that they are not within a group of four adjacent 1's.
 - So on

Example: Prime Implicants

- $F(x,y,z,t) = \Sigma (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

		zt			
		00	01	11	10
xy	00	1	0	1	1
	01	0	1	1	0
	11	0	1	1	0
	10	1	1	1	1

- Prime implicants
 - $y't'$ - essential since m_0 is covered only in it
 - yt - essential since m_5 is covered only in it
 - They together cover $m_0, m_2, m_8, m_{10}, m_5, m_7, m_{13}, m_{15}$

Example: Prime Implicants

zt \ xy	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

- m_3, m_9, m_{11} are not yet covered.
- How do we cover them?
- There are actually more than one way.

Example: Prime Implicants

zt \ xy	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

- $y'z$ and zt covers both m_3 and m_{11} .
- m_9 can be covered in two different prime implicant:
 - xt or xy'

Example: Prime Implicants

- $F(x, y, z, t) = y\bar{t} + y'\bar{t}' + z\bar{t} + x\bar{t}$ or
- $F(x, y, z, t) = y\bar{t} + y'\bar{t}' + z\bar{t} + xy'$ or
- $F(x, y, z, t) = y\bar{t} + y'\bar{t}' + y'z + x\bar{t}$ or
- $F(x, y, z, t) = y\bar{t} + y'\bar{t}' + y'z + xy'$
- Therefore, what to do
 - Find out all the essential prime implicants
 - Other prime implicants that covers the minterms not covered by the essential prime implicants
 - Simplified expression is the logical sum of the essential implicants plus the other implicants

Five-Variable Map

- Downside:
 - Karnaugh maps with more than four variables are not simple to use anymore.
 - 5 variables \rightarrow 32 squares, 6 variables \rightarrow 64 squares
 - Somewhat more practical way for $F(x, y, z, t, w)$

tw \ yz		tw			
		00	01	11	10
00	yz	m_0	m_1	m_3	m_2
01	yz	m_4	m_5	m_7	m_6
11	yz	m_{12}	m_{13}	m_{15}	m_{14}
10	yz	m_8	m_9	m_{11}	m_{10}

$x = 0$

tw \ yz		tw			
		00	01	11	10
00	yz	m_{16}	m_{17}	m_{19}	m_{18}
01	yz	m_{20}	m_{21}	m_{23}	m_{22}
11	yz	m_{28}	m_{29}	m_{31}	m_{30}
10	yz	m_{24}	m_{25}	m_{27}	m_{26}

$x = 1$

Many-Variable Maps

- Adjacency:
 - Each square in the $x = 0$ map is adjacent to the corresponding square in the $x = 1$ map.
 - For example, $m_4 \rightarrow m_{20}$ and $m_{15} \rightarrow m_{31}$
- In line with the same reasoning we can use four 4-variable maps to obtain 64 squares required for six variable optimization
- Alternative way: Use computer programs
 - Quine-McCluskey method
- In n -variable map, 2^k adjacent squares ($k = 0, 1, \dots, n$) represent an area that gives a term of $n-k$ literals. When $n = k$, entire map gives the identity function

Number of Literals

# of adjacent squares	n = 2	n = 3	n = 4	n = 5
$2^0 = 1$	2	3	4	5
$2^1 = 2$	1	2	3	4
$2^2 = 4$	0	1	2	3
$2^3 = 8$	-	0	1	2
$2^4 = 16$	-	-	0	1
$2^5 = 32$	-	-	-	0

Example: Five-Variable Map

- $F(x,y,z,t,w) = \Sigma (0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$

		tw			
		00	01	11	10
yz	00	1			1
	01	1			1
	11		1		
	10		1		

$x = 0$

		tw			
		00	01	11	10
yz	00				
	01		1	1	
	11		1	1	
	10		1		

$x = 1$

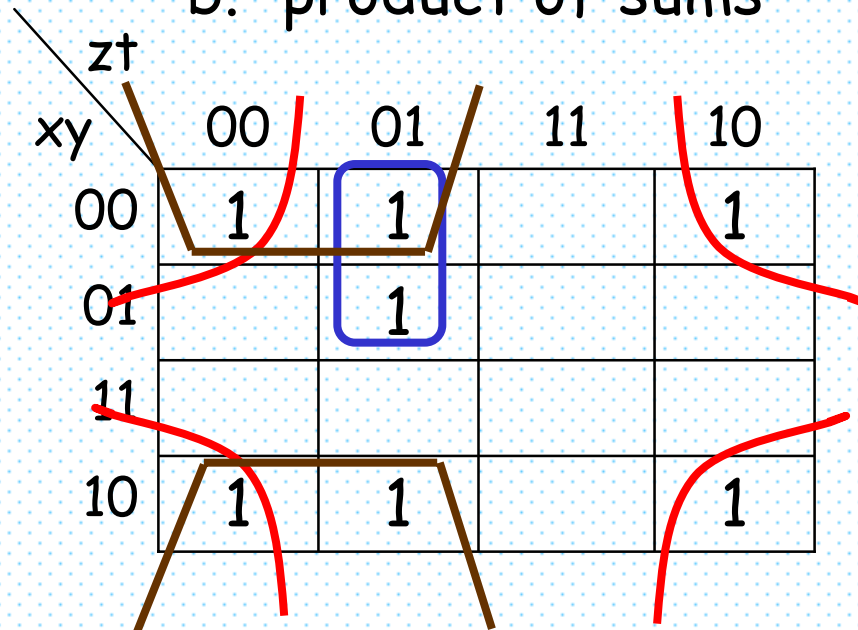
- $F(x,y,z,t,w) = x'y'w' + yt'w + xzw$

Product of Sums Simplification

- So far
 - simplified expressions from Karnaugh maps are in sum of products form.
- Simplified product of sums can also be derived from Karnaugh maps.
- Method:
 - A square with 1 actually represents a minterm
 - Similarly an empty square (a square with 0) represents a maxterm.
 - Treat the 0's in the same manner we treat 1's
 - The result is a simplified expression in product of sums form.

Example: Product of Sums

- $F(x,y,z,t) = \Sigma (0, 1, 2, 5, 8, 9, 10)$
 - Simplify this function in
 - a. sum of products
 - b. product of sums



$$F(x,y,z,t) = y't' + y'z' + x'z't$$

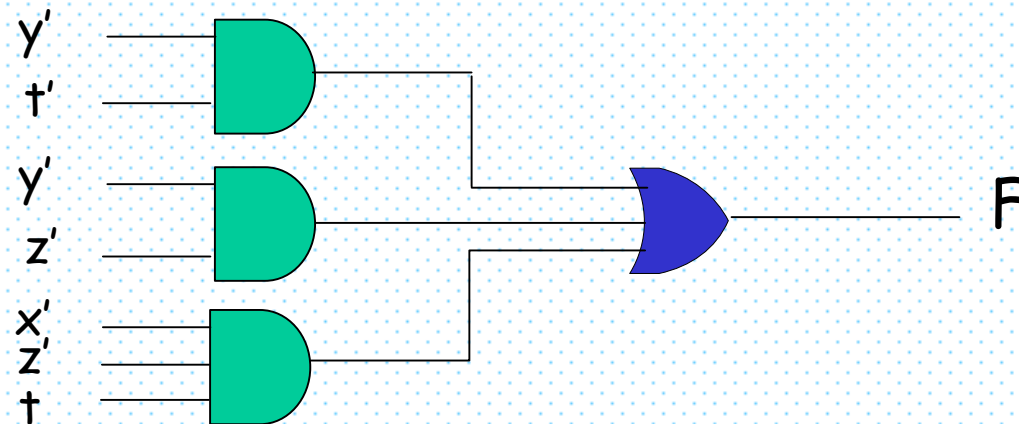
Example: Product of Sums

zt \ xy	00	01	11	10
00			0	
01	0		0	0
11	0	0	0	0
10			0	

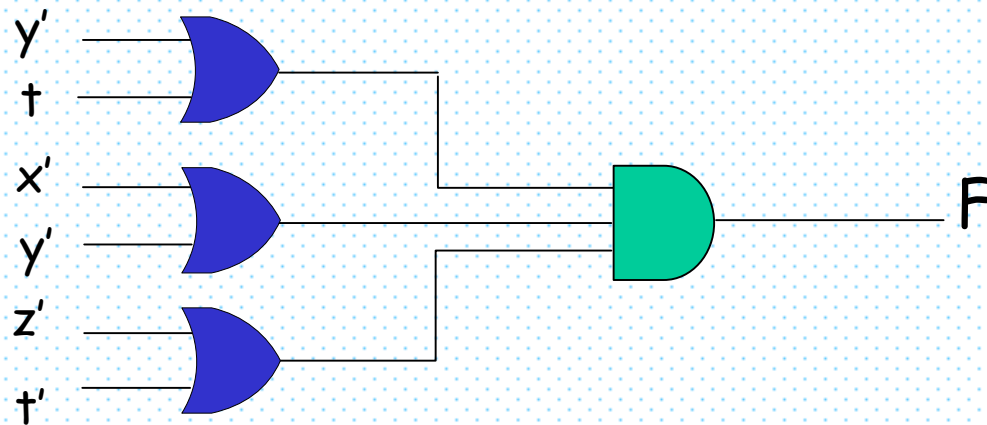
$$yt' + xy + zt$$

- $F'(x,y,z,t) = yt' + xy + zt$
- Apply DeMorgan's theorem
- $F = (y' + t)(x' + y')(z' + t')$

Example: Product of Sums



$F(x,y,z,t) = y't' + y'z' + x'z't$: sum of products implementation



$F = (y' + t)(x' + y')(z' + t')$: product of sums implementation

Product of Maxterms

- If the function is originally expressed in the product of maxterms canonical form, the procedure is also valid
- Example:
 - $F(x, y, z) = \Pi (0, 2, 5, 7)$

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

$$F(x, y, z) = (x' + z')(x+z)$$

$$F(x, y, z) = x'z + xz'$$

Product of Sums

- To enter a function F , expressed in product of sums, in the map
 - take its complement, F'
 - Find the squares corresponding to the terms in F' ,
 - Fill these square with 0's and others with 1's.

- Example:

- $F(x, y, z, t) = (x' + y' + z')(y + t)$

- $F'(x, y, z, t) = xyz + y't'$

xy \ zt				
	00	01	11	10
00	0			0
01				
11			0	0
10	0			0

Don't Care Conditions

- Some functions are not defined for certain combinations of the values for the variables
 - For instance, a circuit defined by the function has never certain input values;
 - therefore, the corresponding output values do not have to be defined
 - This may significantly reduces the circuit complexity
 - Such function are referred as incompletely specified functions
- Example: Four-bit binary code for the decimal digits

Unspecified Minterms

- For unspecified minterms, we do not care what the value function assumes.
- Unspecified minterms of a function are called don't care conditions.
- We use "X" symbol to represent them in Karnaugh map.
- Useful for further simplification
- The symbol X's in the map can be taken 0 or 1 to make the Boolean expression even more simplified

Example: Don't Care Conditions

- $F(x, y, z, t) = \Sigma(1, 3, 7, 11, 15)$ - function
- $d(x, y, z, t) = \Sigma(0, 2, 5, \dots)$ - don't care conditions

$xy \backslash zt$					
		00	01	11	10
00	X	1	1	X	
01	0	X	1	0	
11	0	0	1	0	
10	0	0	1	0	




$$F = zt + x'y't$$

$$F_1 = zt + x'y' \text{ or}$$

$$F_2 = zt + x't$$

Example: Don't Care Conditions

- $F_1 = zt + x'y' = \Sigma(0, 1, 2, 3, 7, 11, 15)$
- $F_2 = zt + x't = \Sigma(1, 3, 5, 7, 11, 15)$
- The two functions are algebraically unequal
 - As far as the function F is concerned both functions are acceptable
- Look at the simplified product of sums expression for the same function F

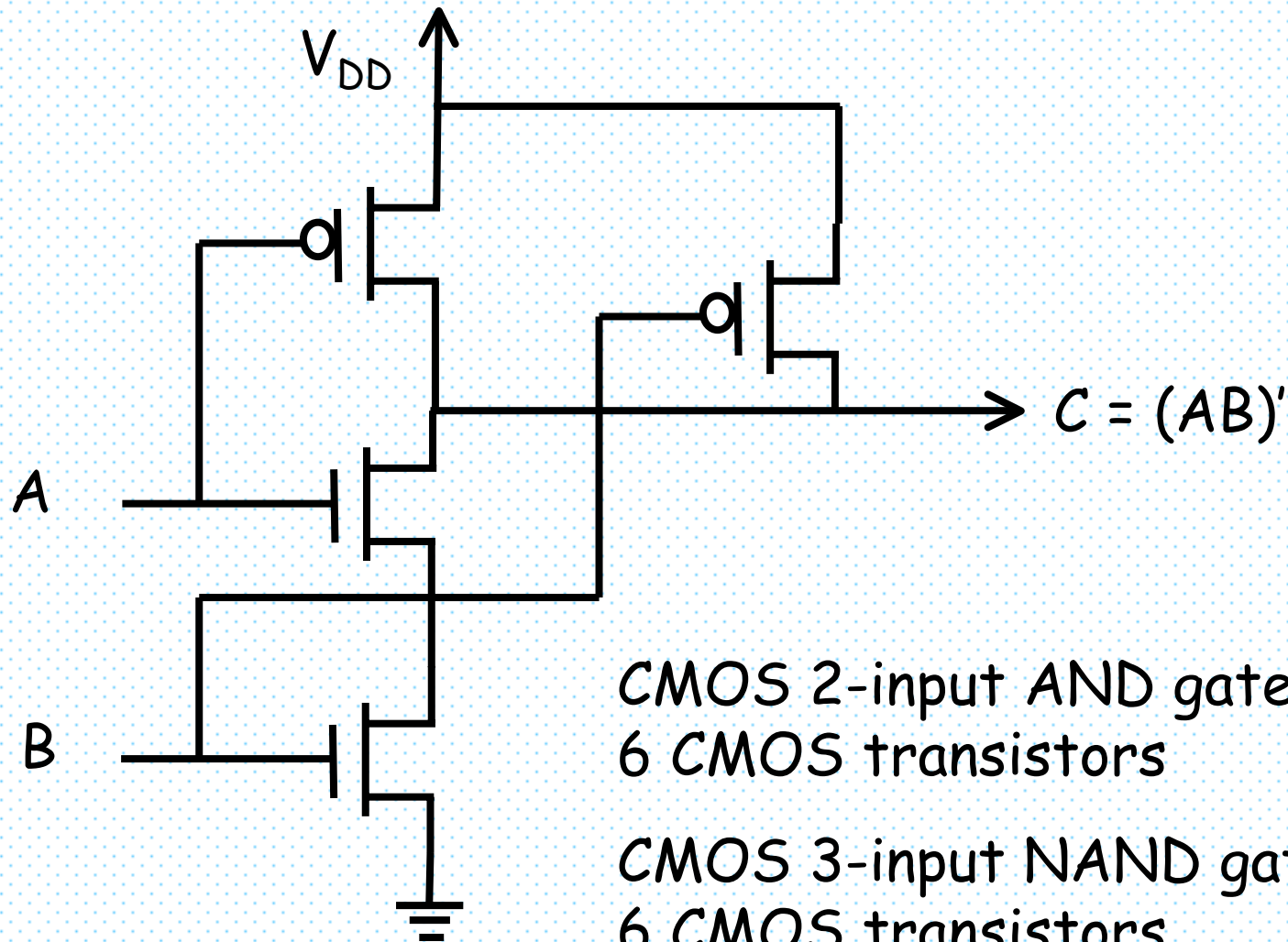
		zt			
		00	01	11	10
xy	00		1	1	
	01	0		1	0
	11	0	0	1	0
	10	0	0	1	0

$$F' = t' + xz'$$

$$F = t(x' + z)$$

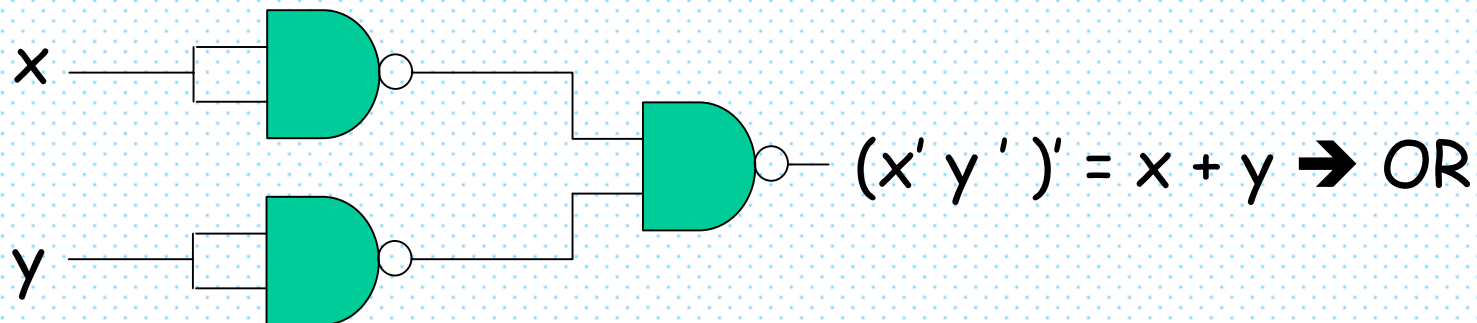
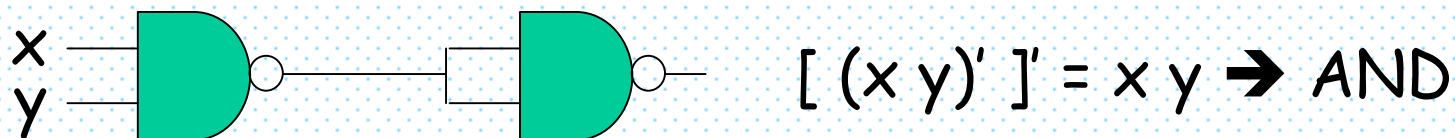
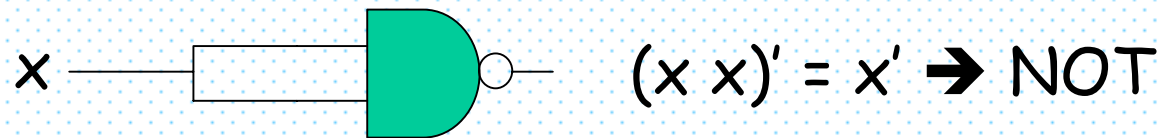
NAND or NOR Gates

- NAND and NOR gates are easier to fabricate

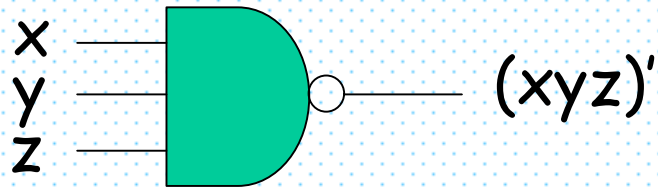


Design with NAND or NOR Gates

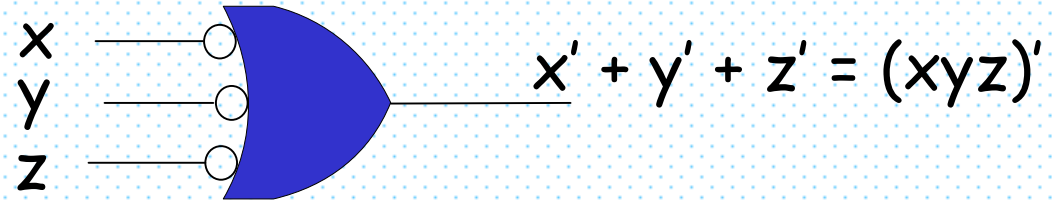
- It is beneficial to derive conversion rules from Boolean functions given in terms of AND, OR, an NOT gates into equivalent NAND or NOR implementations



New Notation

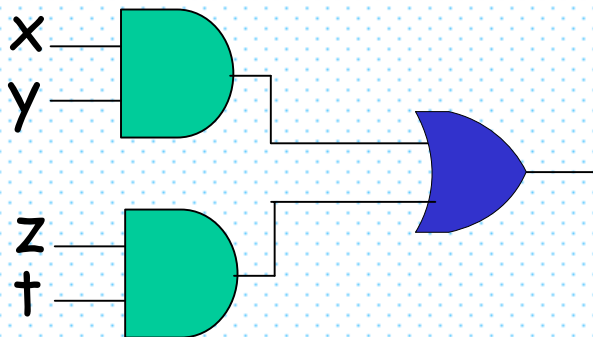


AND-invert

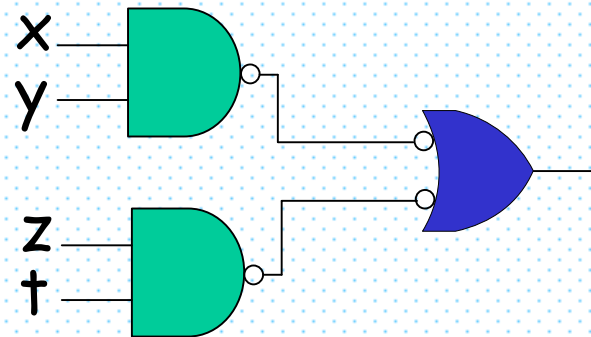


Invert-OR

- To implement a Boolean function with NAND gates easily it must be in sum of products form.
- Example: $F(x, y, z, t) = xy + zt$

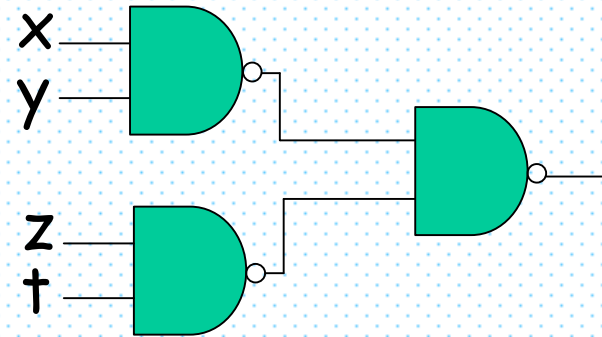
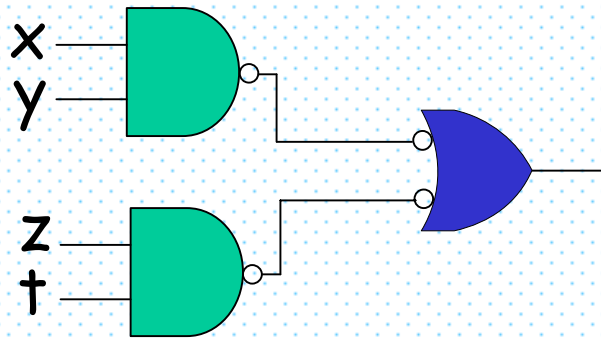


$$F(x, y, z, t) = xy + zt$$



$$F(x, y, z, t) = ((xy)')' + ((zt)')'$$

The Conversion Method



$$((xy)')' + ((zt)')' = xy + zt = [(xy)' (zt)']'$$

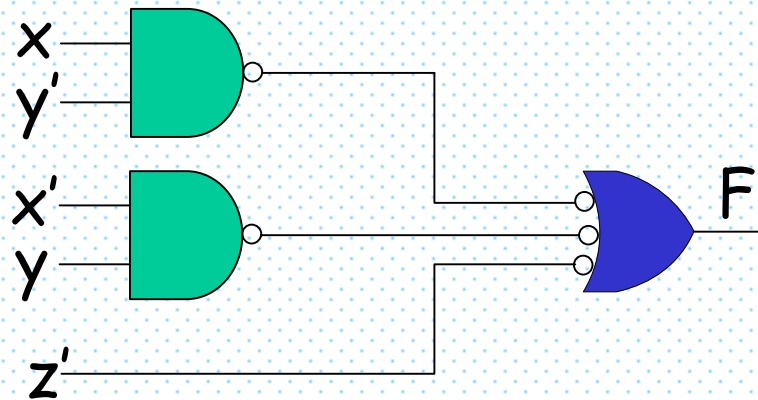
- Example: $F(x, y, z) = \Sigma(1, 2, 3, 4, 5, 7)$

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

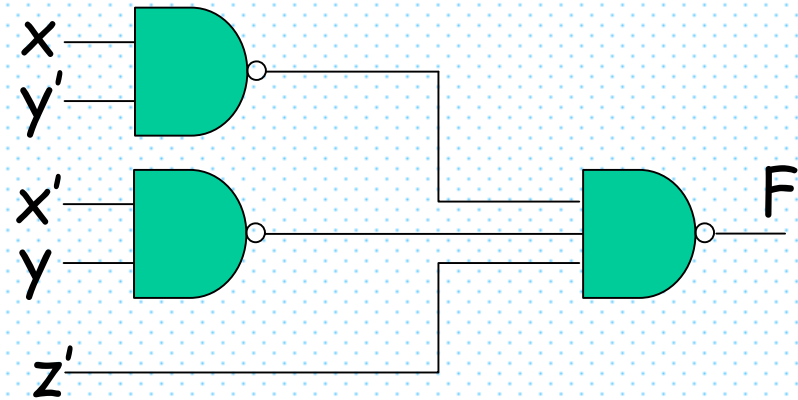
$$F = z + xy' + x'y$$

$$F = (z')' + ((xy')')' + ((x'y)')'$$

Example: Design with NAND Gates



$$F = (z')' + ((xy')')' + ((x'y')')'$$



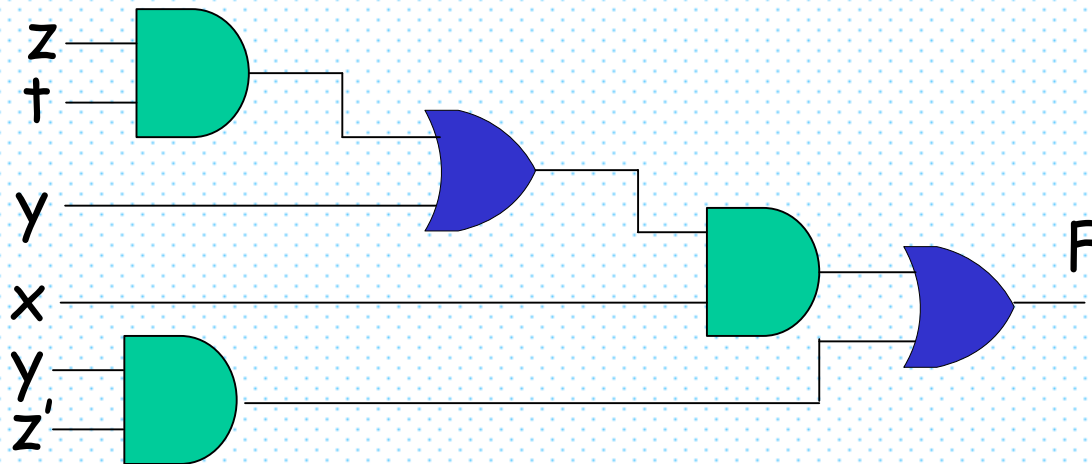
$$F = z + xy' + x'y$$

- Summary

1. Simplify the function (in sum of products form)
2. Draw a NAND gate for each product term
3. Draw a NAND gate for each OR gate in the 2nd level, with inputs coming from the outputs of the first level gates
4. A term with single literal needs an inverter in the first level. Assume single, complemented literals are available.

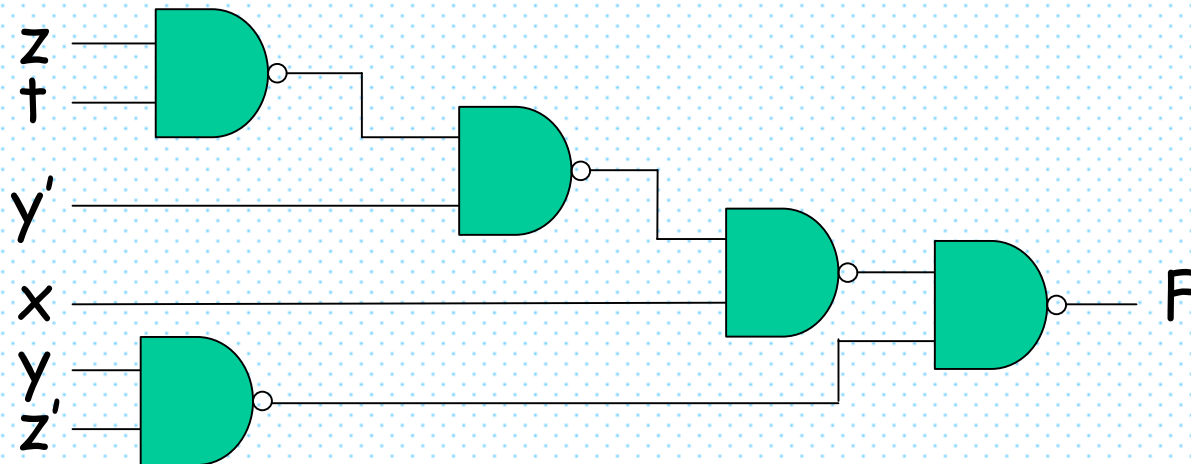
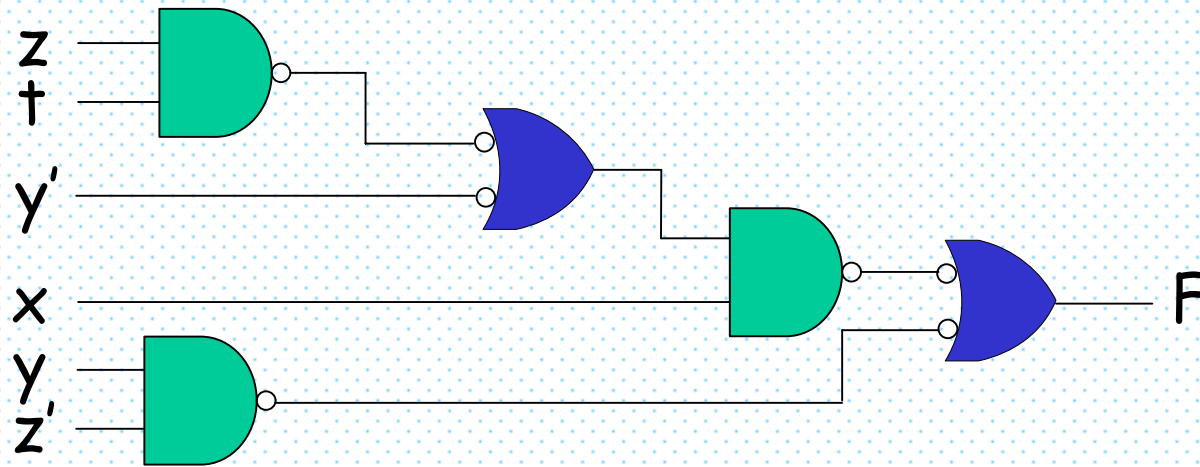
Multi-Level NAND Gate Designs

- The standard form results in two-level implementations
- Non-standard forms may raise a difficulty
- Example: $F = x(zt + y) + yz'$
 - 4-level implementation



Example: Multilevel NAND...

$$F = x(zt + y) + yz'$$



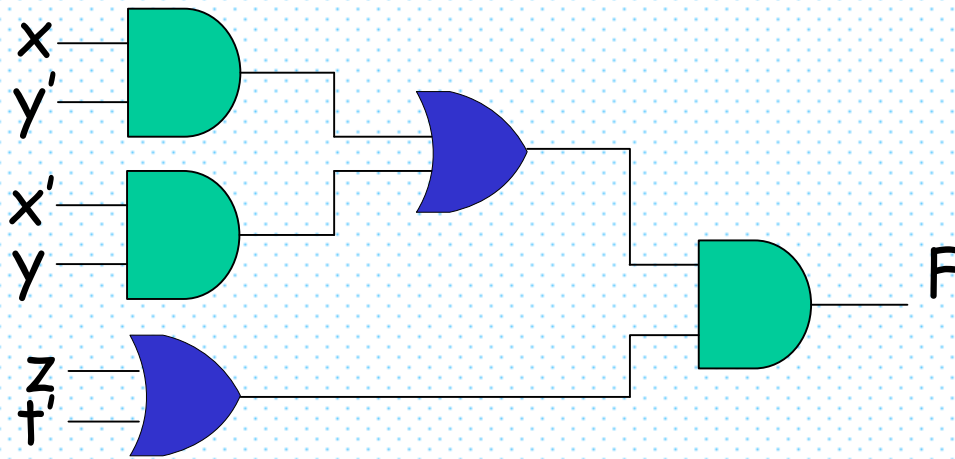
Design with Multi-Level NAND Gates

- Rules

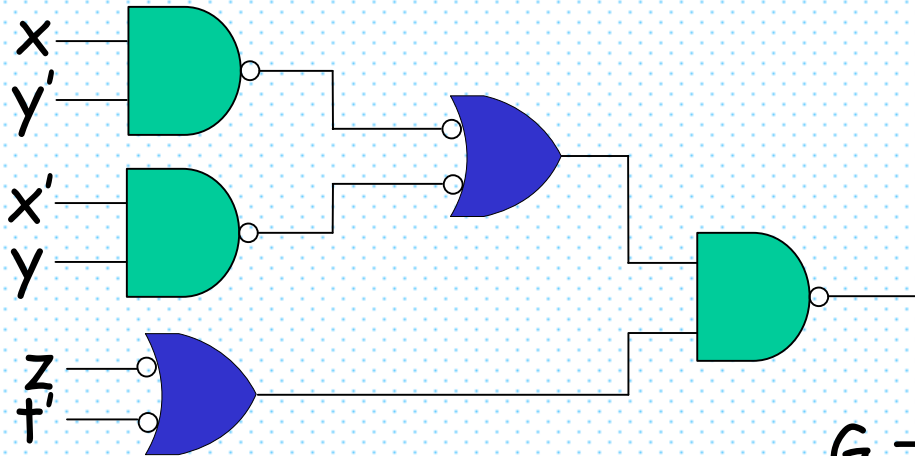
1. Convert all AND gates to NAND gates
2. Convert all OR gates to NAND gates
3. Check the bubbles in the diagram. For every bubble along a path from input to output there must another bubble. If not so,
 - a. Insert an inverter (one-input NAND gate) or
 - b. complement the input literal

Another (Harder) Example

- Example: $F = (xy' + x'y)(z + t')$
- (three-level implementation)

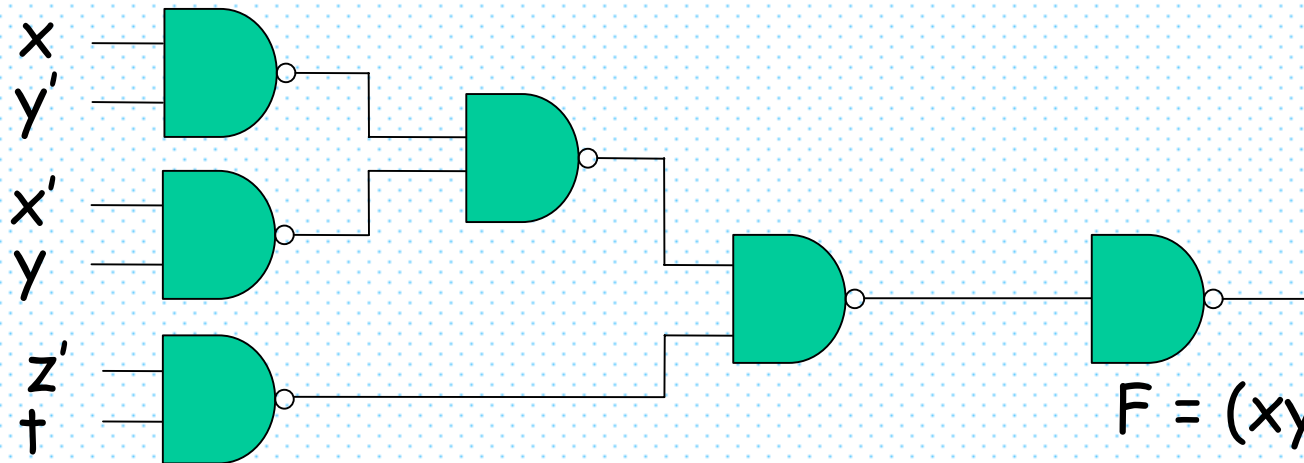


Example: Multi-Level NAND Gates



$$G = [(xy' + x'y)(z' + t)]'$$

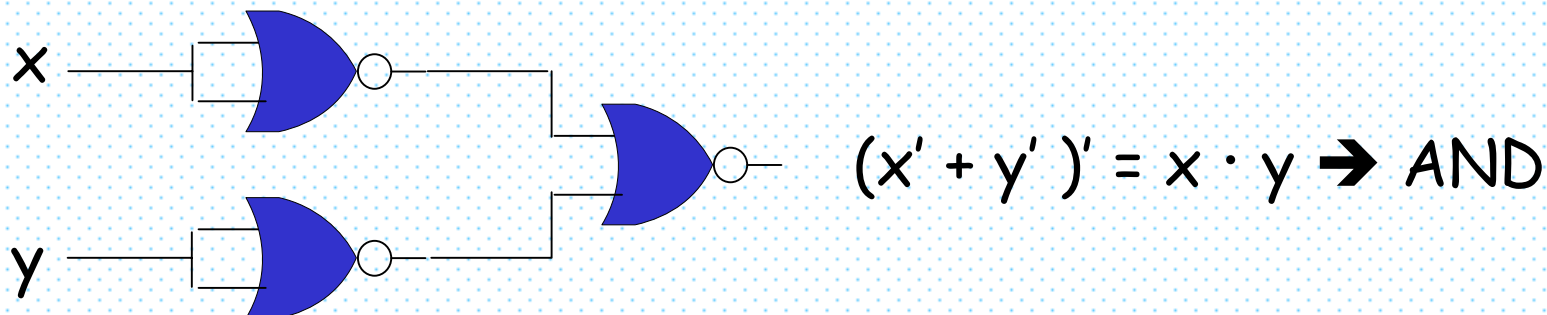
$$F = (xy' + x'y)(z + t')$$



$$F = (xy' + x'y)(z + t')$$

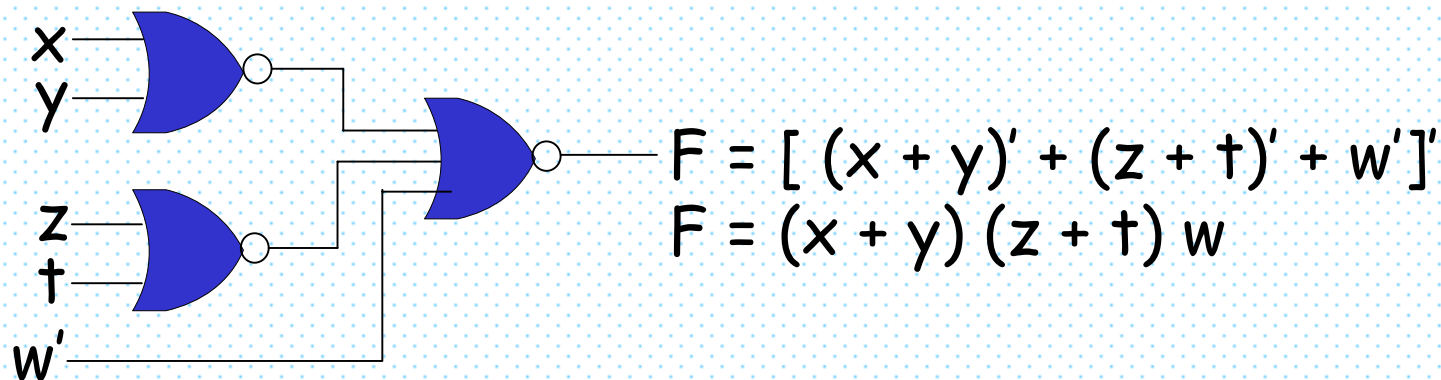
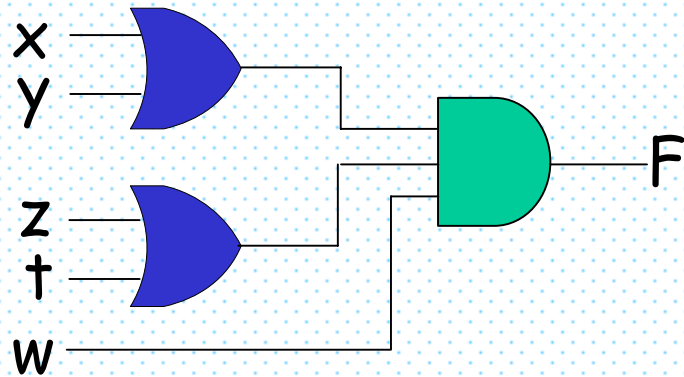
Design with NOR Gates

- NOR is the dual operation of NAND.
 - All rules and procedure we used in the design with NAND gates applies here in a similar way.
 - Function to be implemented must be in product of sums form.



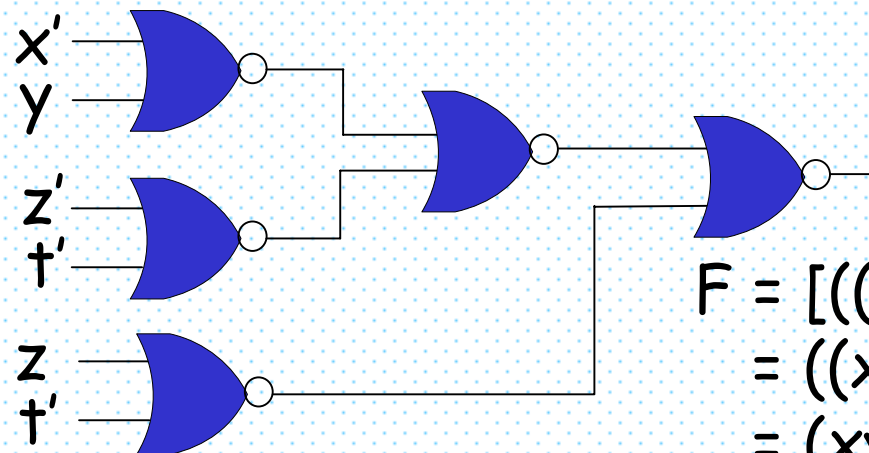
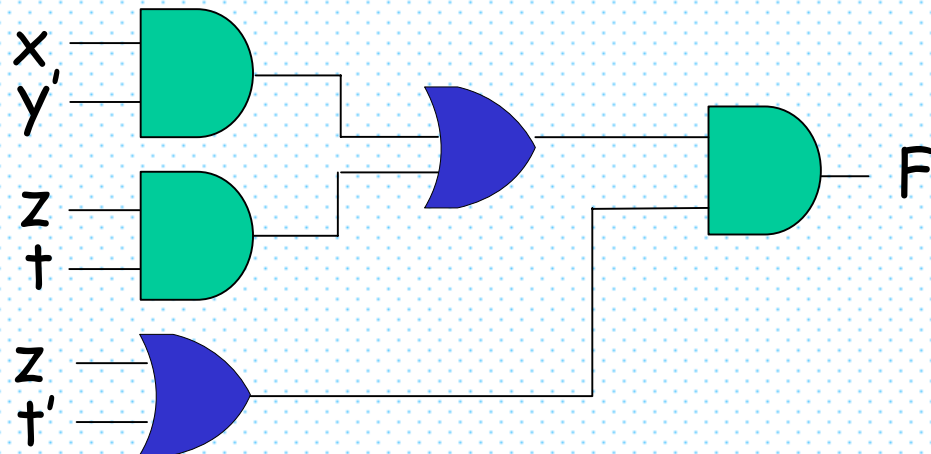
Example: Design with NOR Gates

- $F = (x+y)(z+t)w$



Example: Design with NOR Gates

- $F = (xy' + zt)(z + t')$



$$\begin{aligned} F &= [((x' + y)' + (z' + t'))' + (z + t')]' \\ &= ((x' + y)' + (z' + t'))(z + t') \\ &= (xy' + zt)(z + t') \end{aligned}$$

Exclusive-OR Function

- The symbol: \oplus
 - $x \oplus y = xy' + x'y$
 - $(x \oplus y)' = xy + x'y'$
- Properties
 1. $x \oplus 0 = x$
 2. $x \oplus 1 = x'$
 3. $x \oplus x = 0$
 4. $x \oplus x' = 1$
 5. $x \oplus y' = x \oplus y' = (x \oplus y)'$ - XNOR
- Commutative & Associative
 - $x \oplus y = y \oplus x$
 - $(x \oplus y) \oplus z = x \oplus (y \oplus z)$

Exclusive-OR Function

- XOR gate is not universal
 - Only a limited number of Boolean functions can be expressed in terms of XOR gates
- XOR operation has very important application in arithmetic and error-detection circuits.
- Odd Function
 - $(x \oplus y) \oplus z = (xy' + x'y) \oplus z$
 $= (xy' + x'y) z' + (xy' + x'y)' z$
 $= xy'z' + x'yz' + (xy + x'y') z$
 $= xy'z' + x'yz' + xyz + x'y'z$
 $= \Sigma (1, 4, 5, 7)$

Odd Function

- If an odd number of variables are equal to 1, then the function is equal to 1.
- Therefore, multivariable XOR operation is referred as "odd" function.

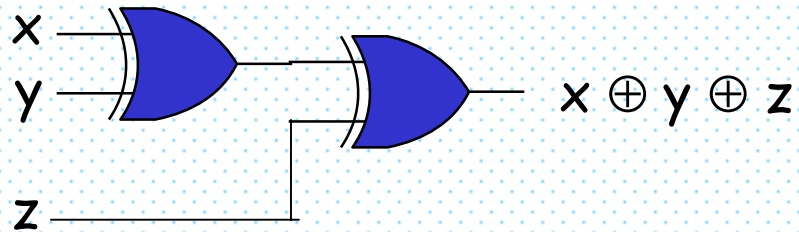
x \ yz				
	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Odd function

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

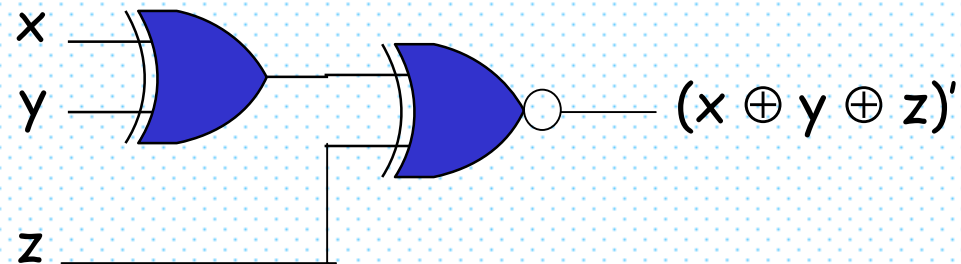
Even function

Odd & Even Functions



Odd function

- $(x \oplus y \oplus z)' = ((x \oplus y) \oplus z)'$



Adder Circuit

- Addition of two-bit numbers

- $c = a + b$
- $a = (a_1 a_0)$ and $b = (b_1 b_0)$
- $c = (c_2 c_1 c_0)$

- Bitwise addition

1. $c_0 = a_0 \oplus b_0$
 $C = a_0 b_0$
2. $c_1 = a_0 \oplus b_0 \oplus C$
 $C = a_0 b_0 + a_0 C + b_0 C$
3. $c_2 = C$