

# Gate-Level Minimization

BME208 – Logic Circuits

Yalçın İŞLER

[islerya@yahoo.com](mailto:islerya@yahoo.com)

<http://me.islerya.com>

# 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
    - Quine-McCluskey
    - Karnaugh (K-) map technique
    - Espresso

# Quine-McCluskey Method

- $F(x_1, x_2, x_3, x_4) = \sum 2, 4, 6, 8, 9, 10, 12, 13, 15$

<b>mi</b>	<b>x1</b>	<b>x2</b>	<b>x3</b>	<b>x4</b>
<b>2</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>4</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>
<b>8</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>6</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>9</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>10</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>12</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>
<b>13</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>15</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>

# Quine-McCluskey Method

List 1						List 2					List 3					
mi	x1	x2	x3	x4		mi	x1	x2	x3	x4		mi	x1	x2	x3	x4
2	0	0	1	0	ok	2,6	0	-	1	0		8,9,12,13	1	-	0	-
4	0	1	0	0	ok	2,10	-	0	1	0		8,12,9,13	1	-	0	-
8	1	0	0	0	ok	4,6	0	1	-	0		Finished				
6	0	1	1	0	ok	4,12	-	1	0	0						
9	1	0	0	1	ok	8,9	1	0	0	-	ok					
10	1	0	1	0	ok	8,10	1	0	-	0						
12	1	1	0	0	ok	8,12	1	-	0	0	ok					
13	1	1	0	1	ok	9,13	1	-	0	1	ok					
15	1	1	1	1	ok	12,13	1	1	0	-	ok					
						13,15	1	1	-	1						

# Quine-McCluskey Method

List 1					List 2					List 3					
mi	x1	x2	x3	x4	mi	x1	x2	x3	x4	mi	x1	x2	x3	x4	
					2,6	0	-	1	0	8,9,12,13	1	-	0	-	t1
					2,10	-	0	1	0						
					4,6	0	1	-	0	Finished					
					4,12	-	1	0	0						
					8,10	1	0	-	0						
					13,15	1	1	-	1						

# Quine-McCluskey Method

	2	4	6	8	9	10	12	13	15
t1				X	X		X	X	
t2	X		X						
t3	X					X			
t4		X	X						
t5		X					X		
t6				X		X			
t7								X	X

	2	4	6	10
t2	X		X	
t3	X			X
t4		X	X	
t5	— X —			
t6	—			X

t5 is a subset of t4  
t6 is a subset of t3

$$F(x_1, x_2, x_3, x_4) = t_1 + t_7 + t_3 + t_4$$

$$= x_1 x_3' + x_1 x_2 x_4 + x_2' x_3 x_4' + x_1' x_2 x_4'$$

# Two-Variable K-Map

- Two variables:  $x$  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	0

- $F = m_0 + m_1 + m_2 = x'y' + x'y + xy'$
- $F = \dots$
- $F = \dots$
- $F = \dots$
- $F = x' + y'$
- 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

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

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

- $F_1(x, y, z) =$

- $F_2(x, y, z) = \sum (3, 4, 6, 7)$

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

- $F_1(x, y, z) =$

# Example: Three-Variable K-Map

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

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

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

- $F_2(x, y, z) = \sum (3, 4, 6, 7)$

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

- $F_2(x, y, z) = xz' + yz$

# 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

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

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

$$F_1(x, y, z) =$$

# Example

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

		y			
		00	01	11	10
x	0	1	0	0	1
	1	1	1	0	1

$$F_1(x, y, z) =$$

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

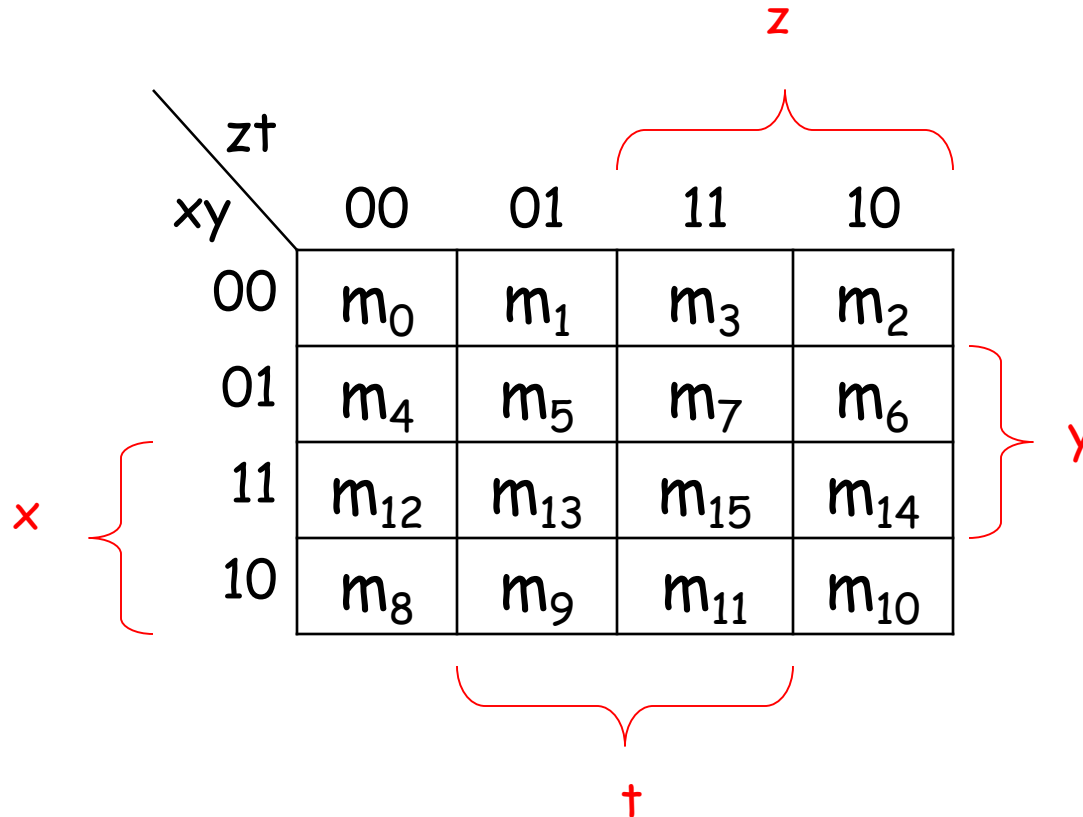
x \ yz	00	01	11	10
0				
1				

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

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

# Four-Variable K-Map

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



# Example: Four-Variable K-Map

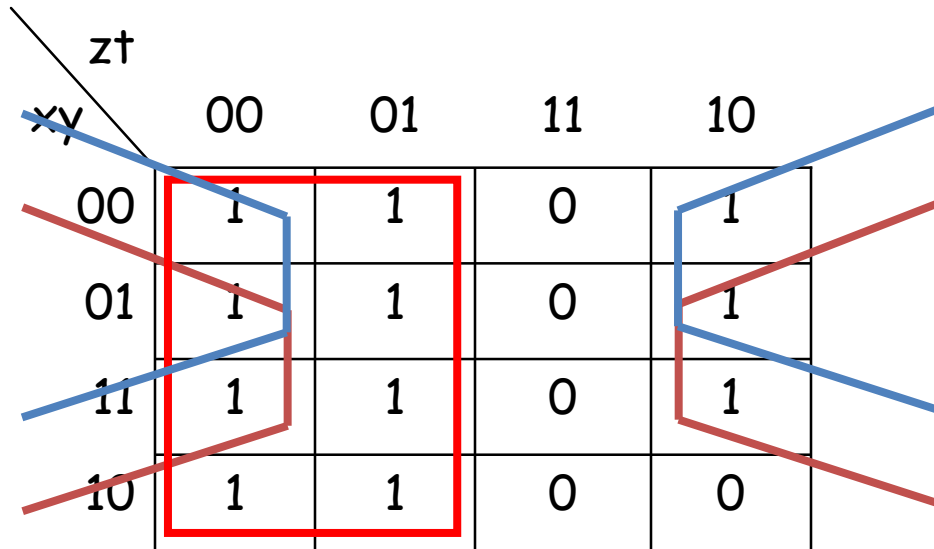
–  $F(x,y,z,t) = \Sigma (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

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

–  $F(x,y,z,t) =$

# Example: Four-Variable K-Map

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



–  $F(x,y,z,t) =$

# Example: Four-Variable K-Map

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

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

- $F(x,y,z,t) =$

# Example: Four-Variable K-Map

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

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

- $F(x,y,z,t) =$

# Example: Four-Variable K-Map

zt \ xy	00	01	11	10
00				
01				
11				
10				

•  $F(x,y,z,t) =$

# 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		xy			
		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

- Both  $y'z$  and  $zt$  covers  $m_3$  and  $m_{11}$ .
- $m_9$  can be covered in two different prime implicants:
  - $xt$  or  $xy'$
- $m_3, m_{11} \rightarrow zt$  or  $y'z$
- $m_9 \rightarrow xy'$  or  $xt$

# Example: Prime Implicants

- $F(x, y, z, t) = yt + y't' + zt + xt$  or
- $F(x, y, z, t) = yt + y't' + zt + xy'$  or
- $F(x, y, z, t) = yt + y't' + y'z + xt$  or
- $F(x, y, z, t) = yt + y't' + y'z + xy'$
- 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			
		00	01	11	10
yz	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	$m_8$	$m_9$	$m_{11}$	$m_{10}$

$x = 0$

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

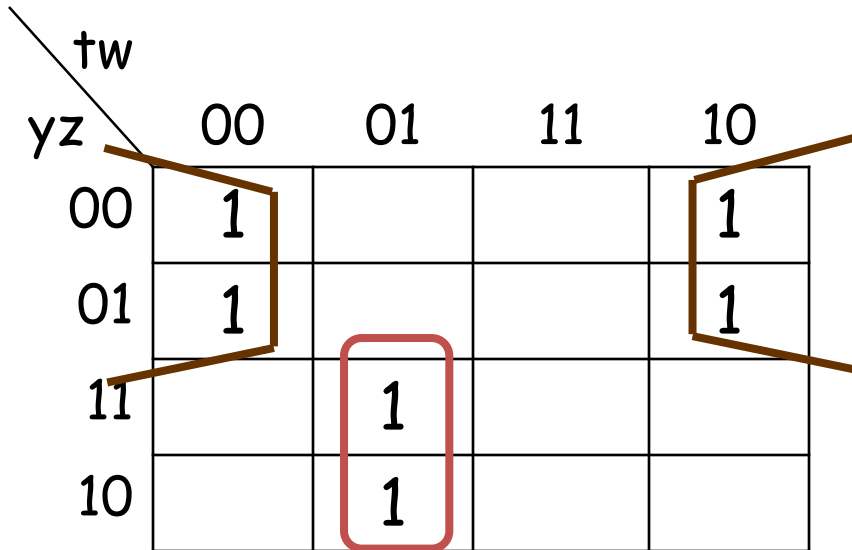
$x = 1$

# Many-Variable Maps

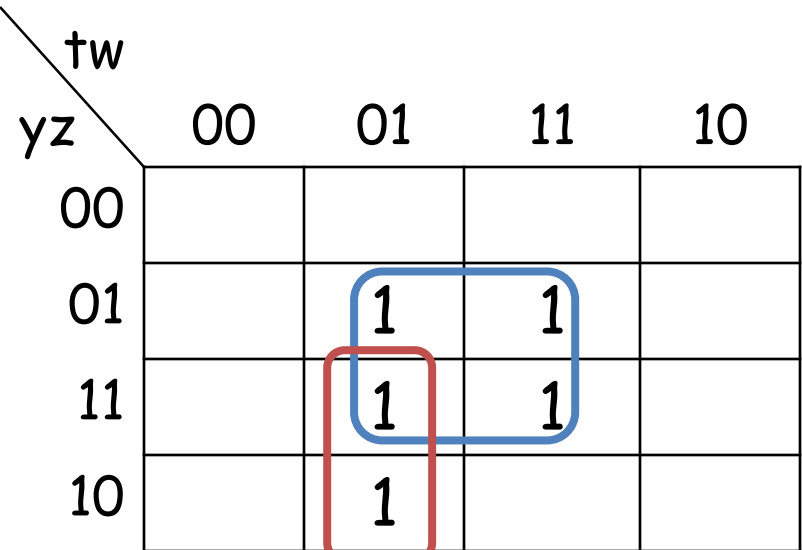
- Adjacency:
  - Each square in the  $x = 0$  map is adjacent to the corresponding square in the  $x = 1$  map.
  - For example,  $m_4 \rightarrow m_{20}$  and  $m_{15} \rightarrow m_{31}$
- Use four 4-variable maps to obtain 64 squares required for six variable optimization
- Alternative way: Use computer programs
  - Quine-McCluskey method
  - Espresso method

# Example: Five-Variable Map

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



$x = 0$



$x = 1$

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

# 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 as we treat 1's
  - The result is a simplified expression in product of sums form.



# Example: Product of Sums

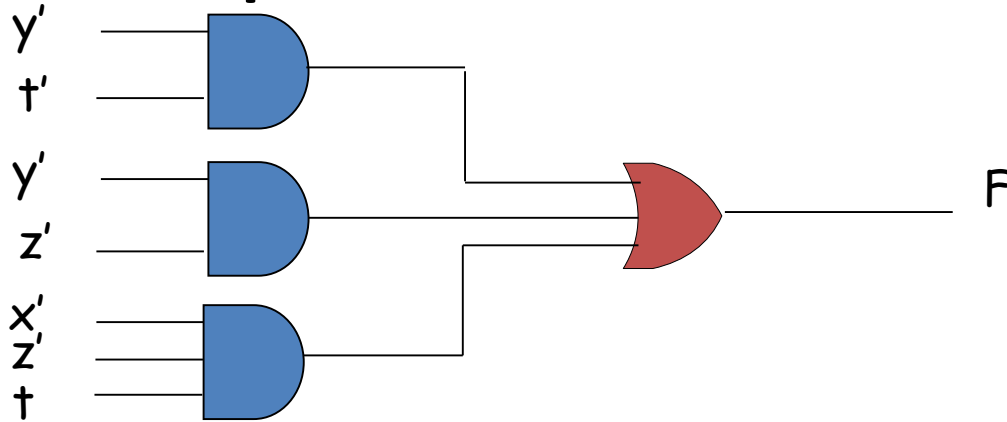
- $F'(x,y,z,t) =$
- Apply DeMorgan's theorem (use dual theorem)
- $F =$

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

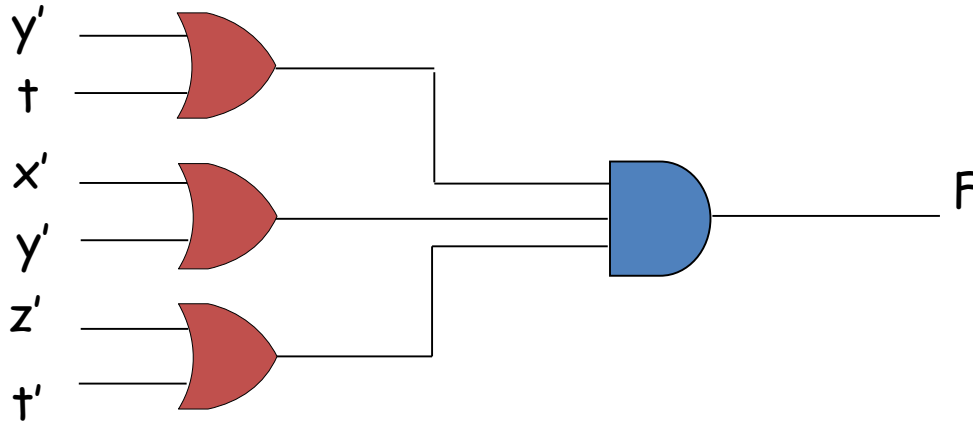
$$F(x,y,z,t) = y't' + y'z' + x'z't$$

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

# 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:

$$- F(x, y, z) = \Pi (0, 2, 5, 7)$$

x \ yz	00	01	11	10
0				
1				

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

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

# Product of Sums

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

- Example:

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

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

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

# Don't Care Conditions 1/2

- Some functions are not defined for certain input combinations
  - Such functions are referred to as incompletely specified functions
  - 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 reduce the circuit complexity

# Don't Care Conditions 2/2

- Example: A circuit that takes the 10's complement of decimal digits

# Unspecified Minterms

- For unspecified minterms, we do not care what the value the function produces.
- 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
- $d(x, y, z, t) = \Sigma(0, 2, 5)$  – don't care conditions

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

$F =$

$F_1 =$

or

$F_2 =$

# 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	X	1	1	X
	01	0	X	1	0
	11	0	0	1	0
	10	0	0	1	0

$F' =$

$F =$

# Another Way to Handle K-Maps - SOP

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1 <sub>43</sub>	1	1	1	0

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

		yz			
		00	01	11	10
x	0	1	t	t'	0
	1	t	1	t'	0

# Another Way to Handle K-Maps - SOP

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1 <sub>14</sub>	1	1	1	0

zt

xy

	00	01	11	10
00	1	1	1	0
01	0	0	0	1
11	0	0	0	1
10	0	1	1	1

yz

x

	00	01	11	10
0	t' + t	t	t'	0
1	t	t + t'	t'	0

# Another Way to Handle K-Maps - SOP

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1 <sub>45</sub>	1	1	1	0

zt

xy

	00	01	11	10
00	1	1	1	0
01	0	0	0	1
11	0	0	0	1
10	0	1	1	1

yz

x

	00	01	11	10
0	t' + t	t	t'	0
1	t	t + t'	t'	0

# Another Way to Handle K-Maps -SOP

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1 <sub>46</sub>	1	1	1	0

zt

xy

	00	01	11	10
00	1	1	1	0
01	0	0	0	1
11	0	0	0	1
10	0	1	1	1

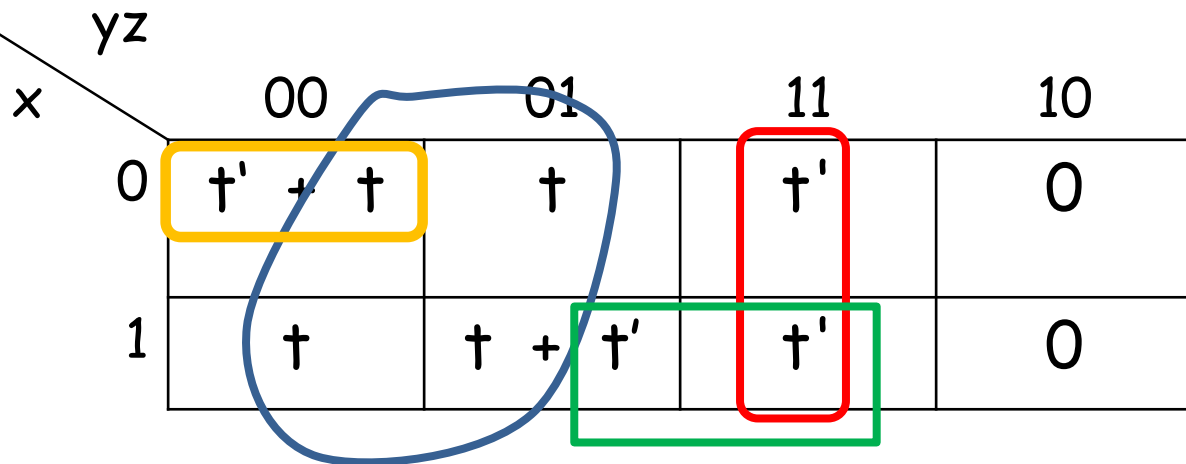
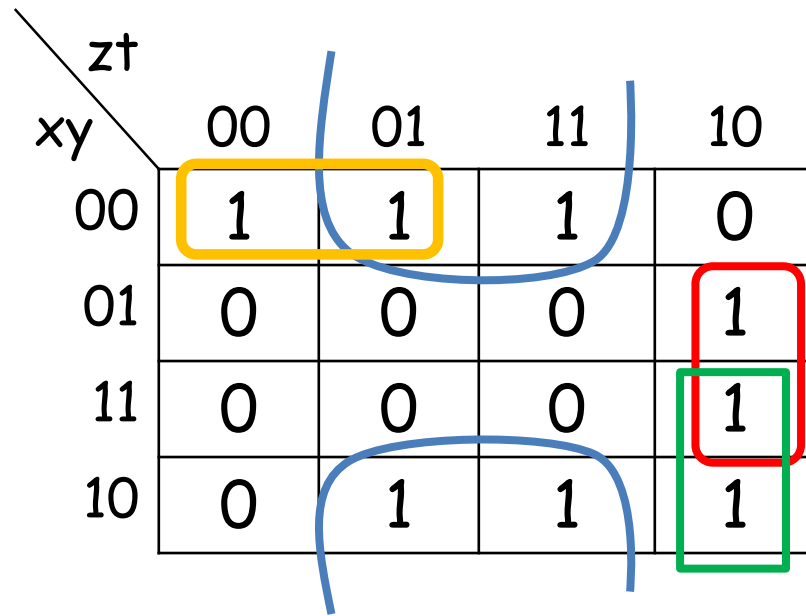
yz

x

	00	01	11	10
0	t' + t	t	t'	0
1	t	t + t'	t'	0

# Another Way to Handle K-Maps - SOP

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1 <sub>47</sub>	1	1	1	0



# Another Way to Handle K-Maps - SOP

- We have 1's in the boxes
- $1 = x+x' = 1+x = 1+x'$  Use this wherever useful
- If you partition  $1 = x+x'$  then include  $x$  in one term,  $x'$  in another
- If you use  $1 = 1+x$ , then include  $x$  in a neighboring bigger block, and process 1 as usual

# Another Way to Handle K-Maps - POS

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1 <sub>49</sub>	1	1	1	0

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

		yz			
		00	01	11	10
x	0	1	t	t'	t' . 0
	1	t	1	t'	t' . t

# Another Way to Handle K-Maps - POS

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

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

		yz			
		00	01	11	10
x	0	1	t	t'	t'. 0
	1	t	1	t'	t'. t

# Another Way to Handle K-Maps - POS

x	y	z	t	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

zt

xy

	00	01	11	10
00	1	1	1	0
01	0	0	0	1
11	0	0	0	1
10	0	1	1	1

yz

x

	00	01	11	10
0	1	t	t'	t'.0
1	t	1	t'	t'.t

# Another Way to Handle K-Maps - POS

- We have 0's in the boxes
- $0 = x.x' = 0.x = 0.x'$  Use this wherever useful
- If you partition  $0 = x.x'$  then include  $x$  in one term,  $x'$  in another
- If you use  $0 = 0.x$ , then include  $x$  in a neighboring bigger block, and process 0 as usual

# Simultaneous Minimization of Multiple Boolean Functions

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

# Simultaneous Minimization of Multiple Boolean Functions

$x'z$

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

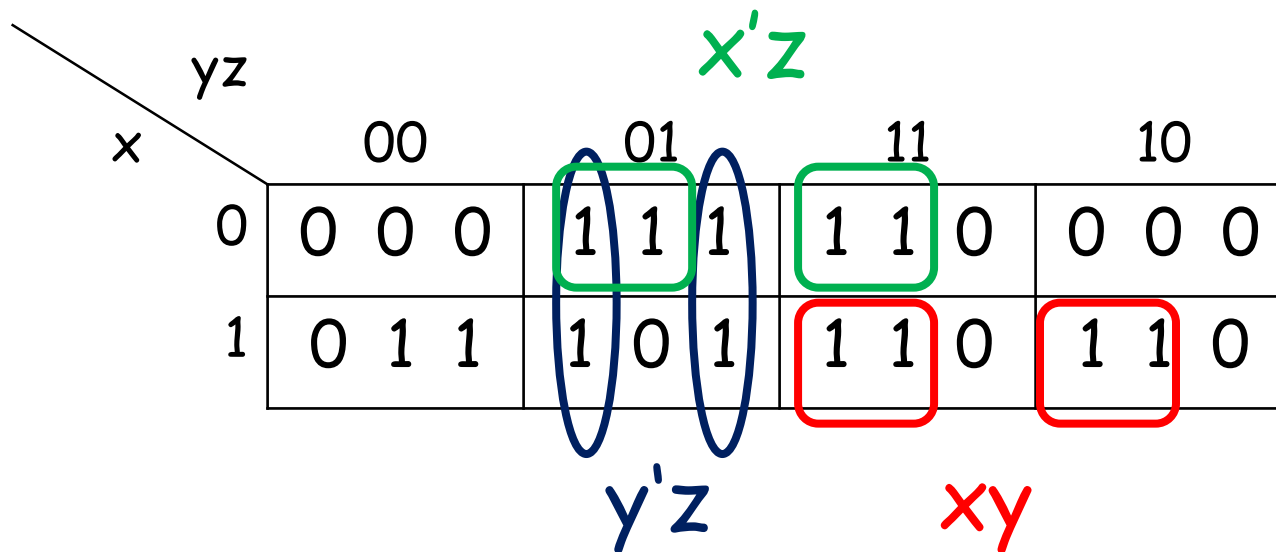
# Simultaneous Minimization of Multiple Boolean Functions

$x'z$

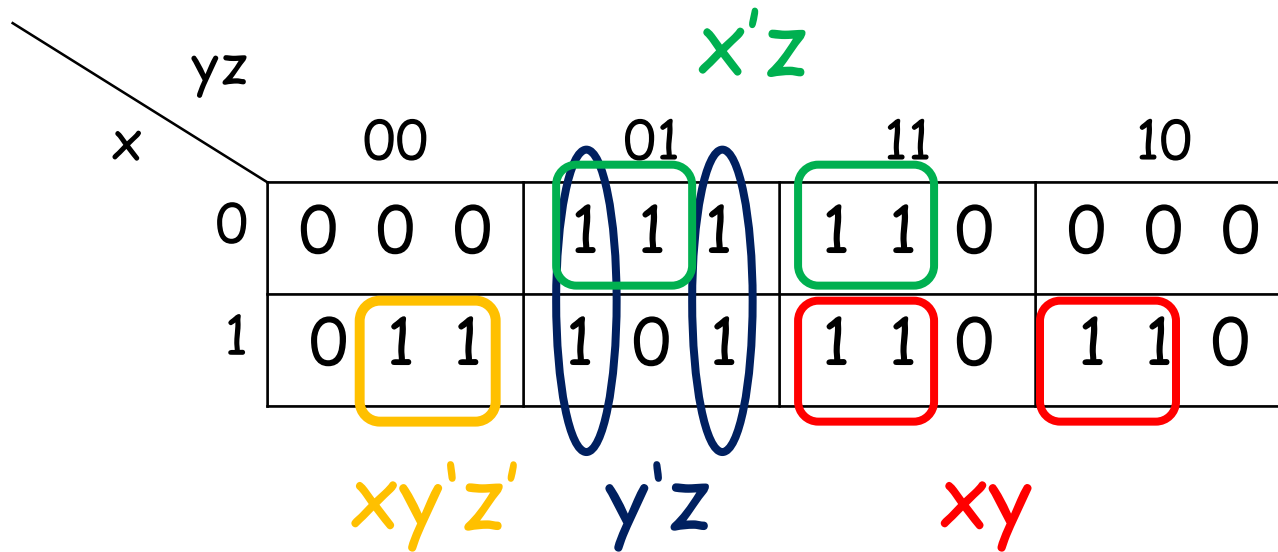
$x \backslash yz$	00	01	11	10
0	0 0 0	1 1 1	1 1 0	0 0 0
1	0 1 1	1 0 1	1 1 0	1 1 0

$xy$

# Simultaneous Minimization of Multiple Boolean Functions

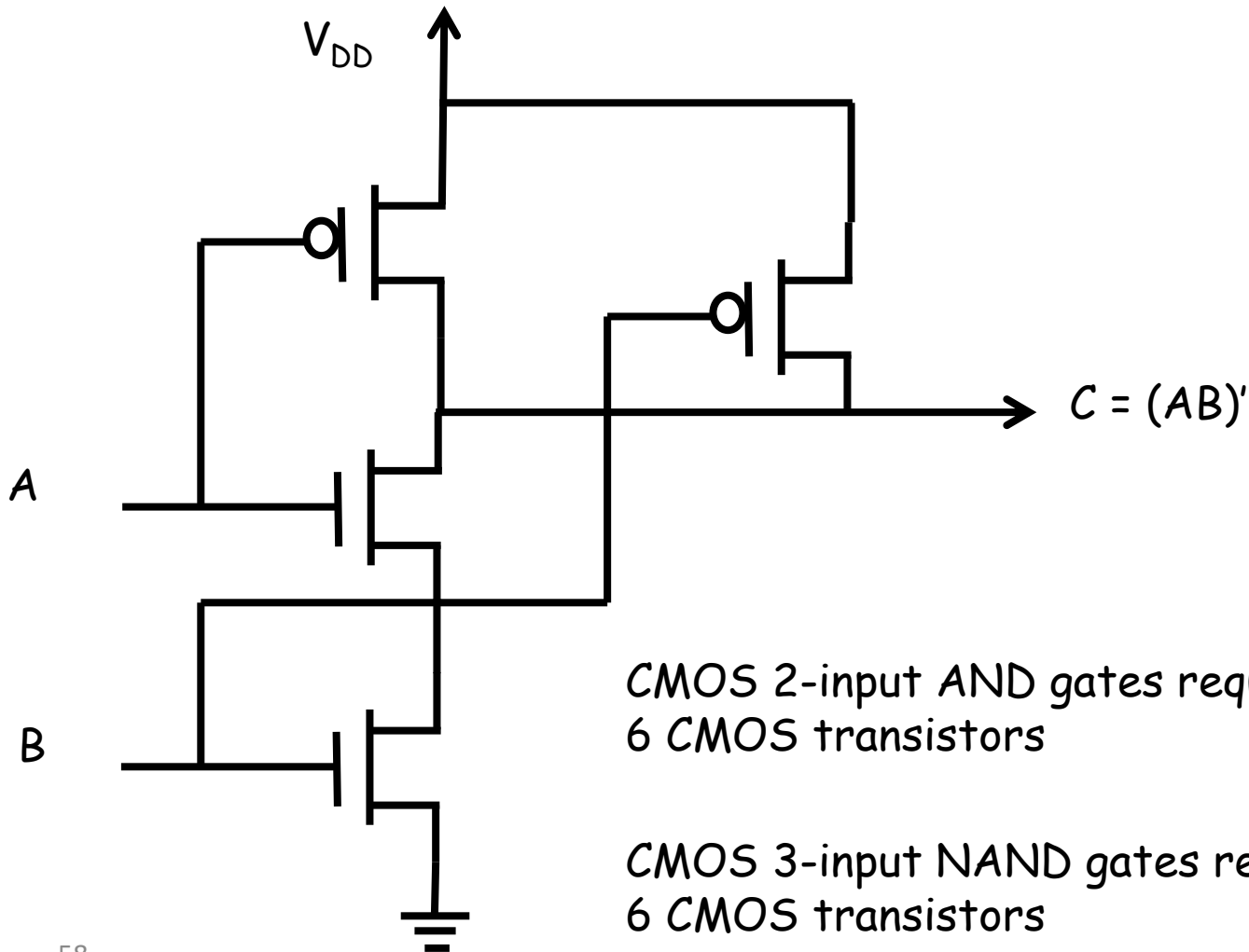


# Simultaneous Minimization of Multiple Boolean Functions



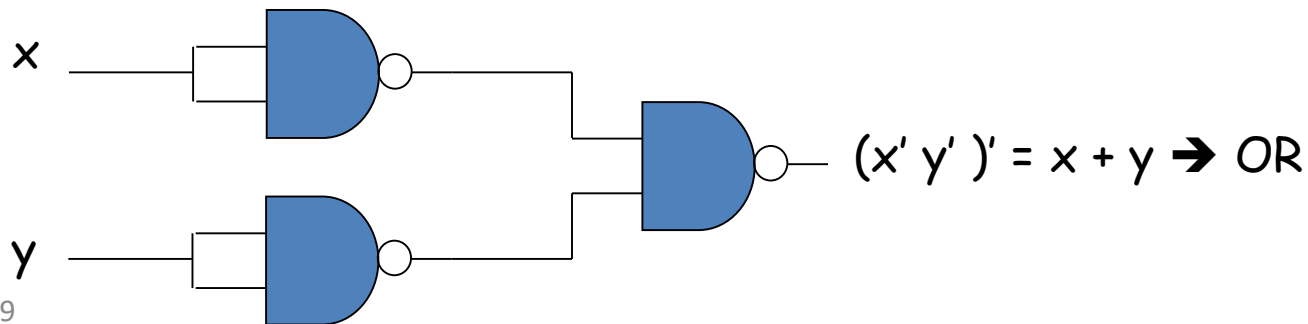
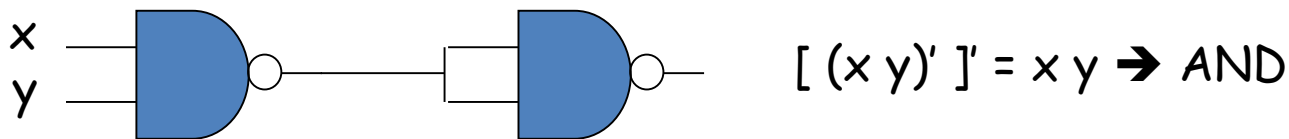
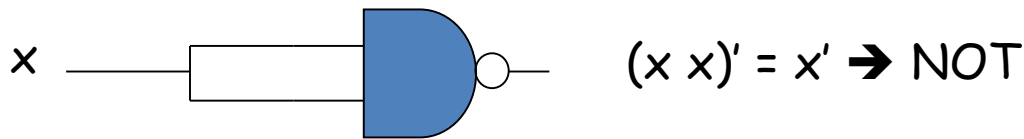
# NAND and NOR Gates

- NAND and NOR gates are easier to fabricate

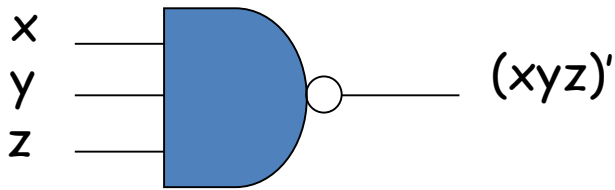


# Design with NAND or NOR Gates

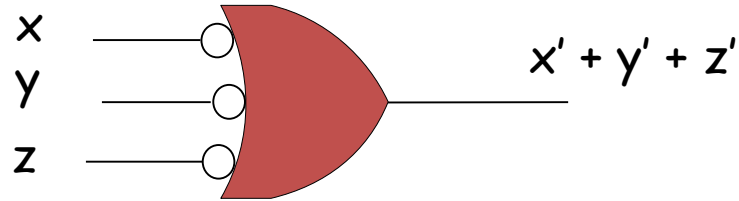
- It is beneficial to derive conversion rules from Boolean functions given in terms of AND, OR, and NOT gates into equivalent NAND or NOR implementations



# New Notation

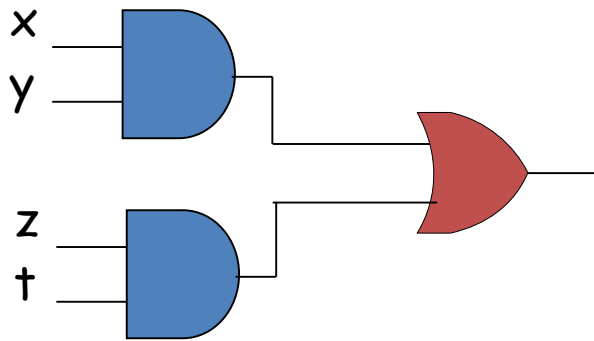


AND-invert

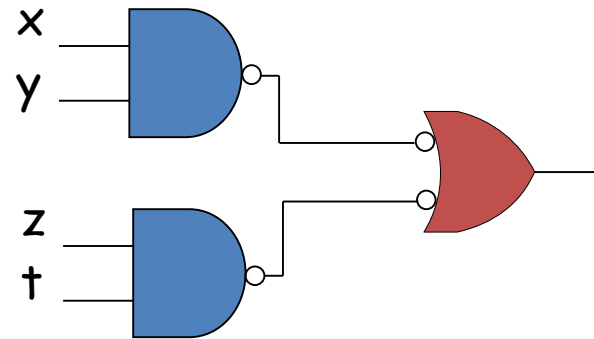


Invert-OR

- Implementing a Boolean function with NAND gates is easy if it is in sum of products form.
- Example:  $F(x, y, z, t) = xy + zt$

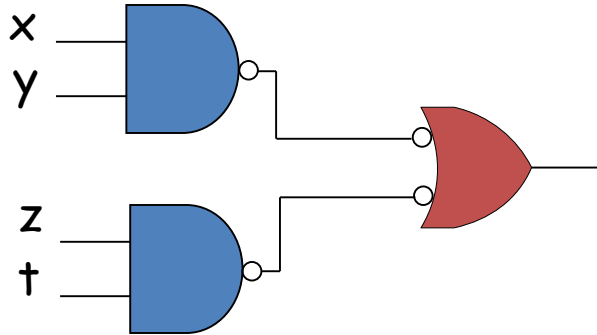


$$F(x, y, z, t) = xy + zt$$



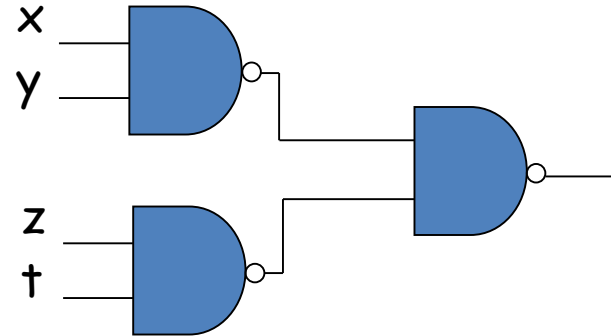
$$F(x, y, z, t) = ((xy)')' + ((zt)')'$$

# The Conversion Method



$$((xy)')' + ((zt)')'$$

$$= xy + zt =$$



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

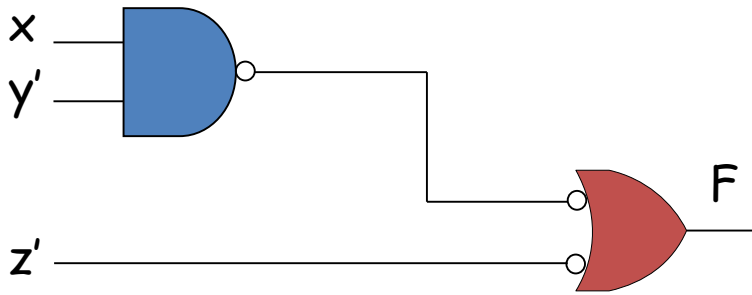
- Example:  $F(x, y, z) = \sum(1, 3, 4, 5, 7)$

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

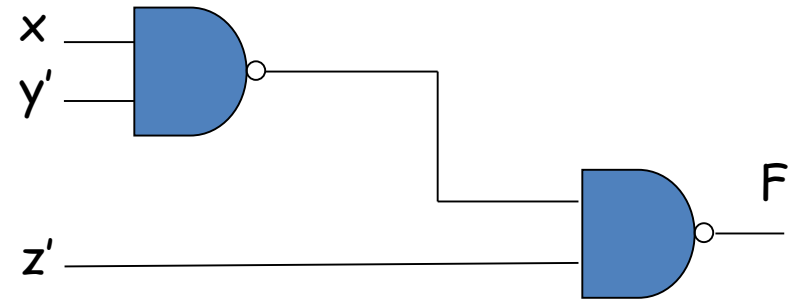
$$F = z + xy'$$

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

# Example: Design with NAND Gates



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



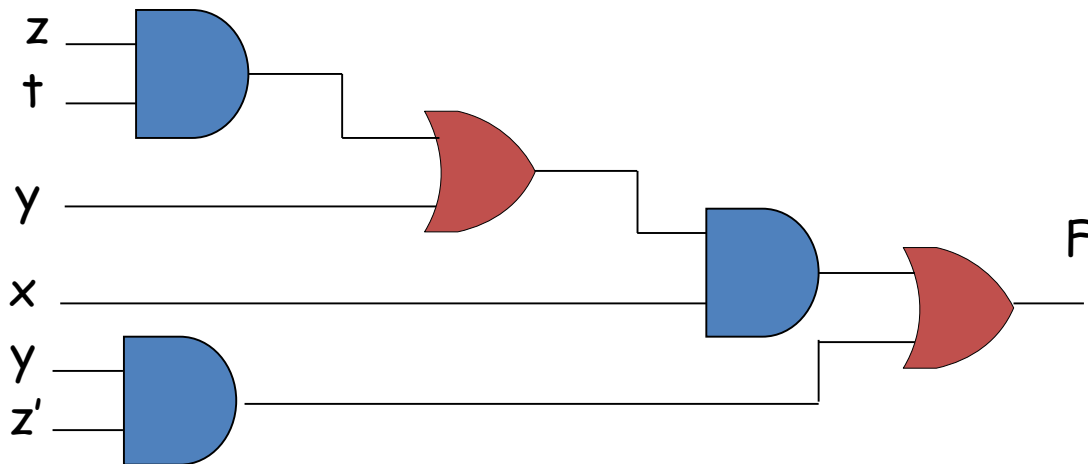
$$F = z + xy'$$

- **Summary**

1. Simplify the function
2. Draw a NAND gate for each product term
3. Draw a NAND gate for the OR gate in the 2<sup>nd</sup> level,
4. A product term with single literal needs an inverter in the first level. Assume single, complemented literals are available.

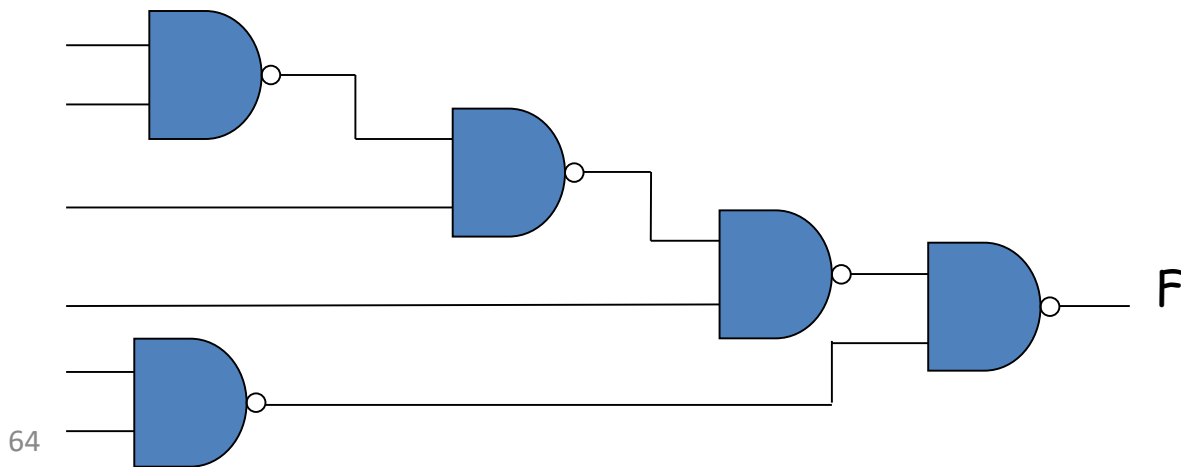
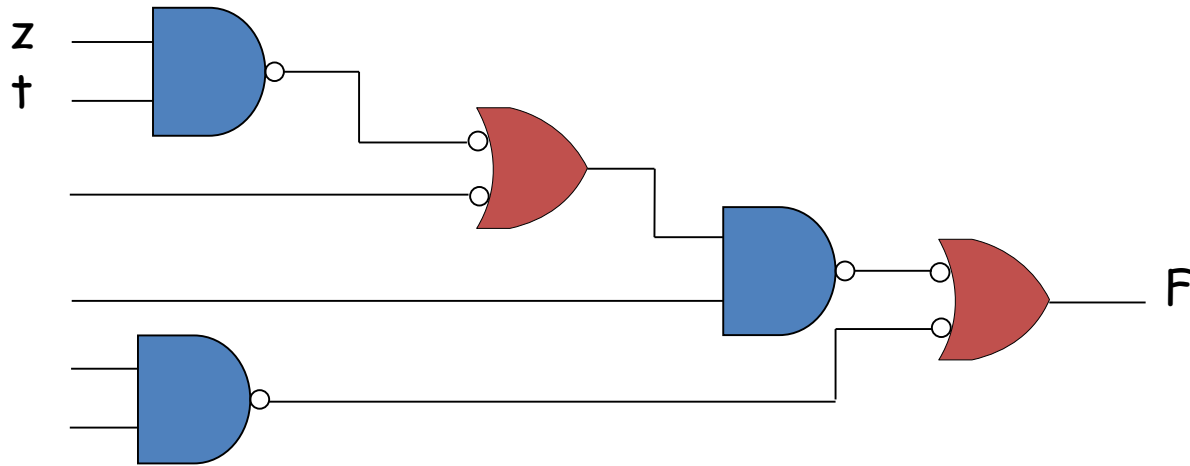
# Multi-Level NAND Gate Designs

- 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...

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

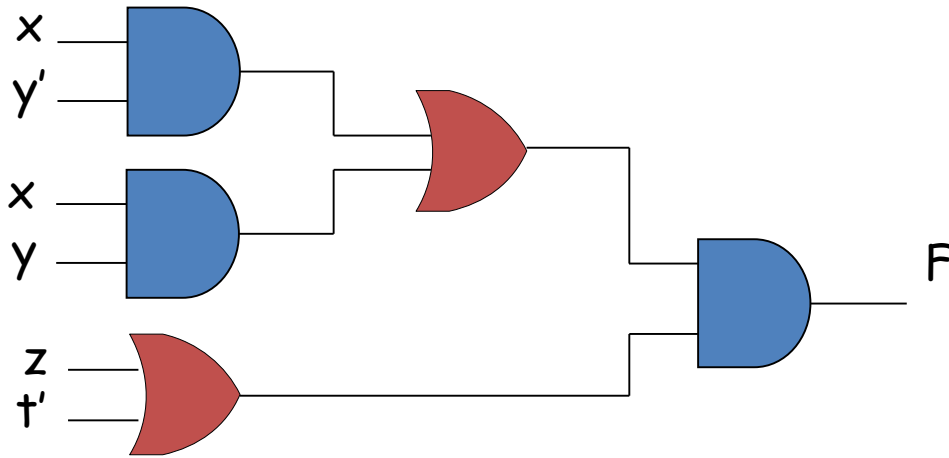


# Design with Multi-Level NAND Gates

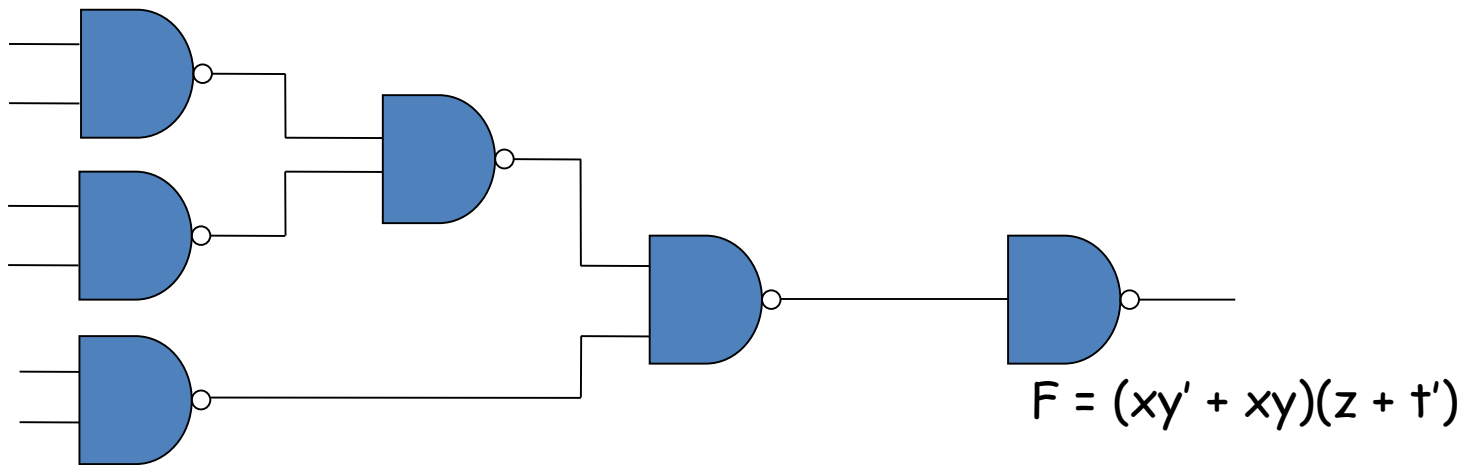
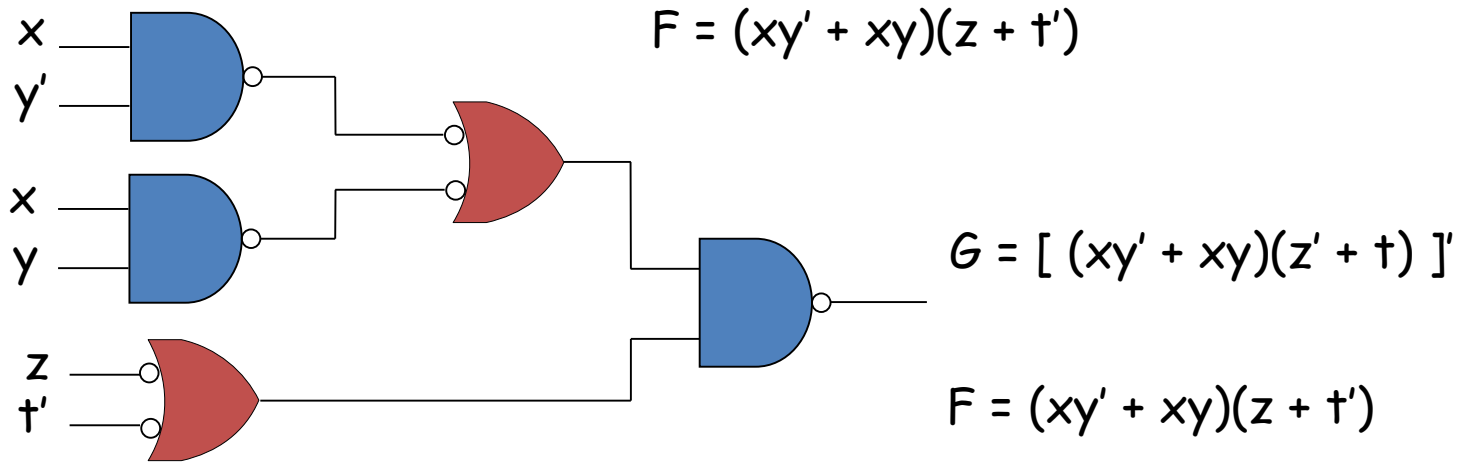
- Rules
  1. Convert all AND gates to NAND gates
  2. Convert all OR gates to NAND gates
  3. Insert an inverter (one-input NAND gate) at the output if the final operation is AND
  4. Check the bubbles in the diagram. For every bubble along a path from input to output there must be another bubble. If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

# Another (Harder) Example

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

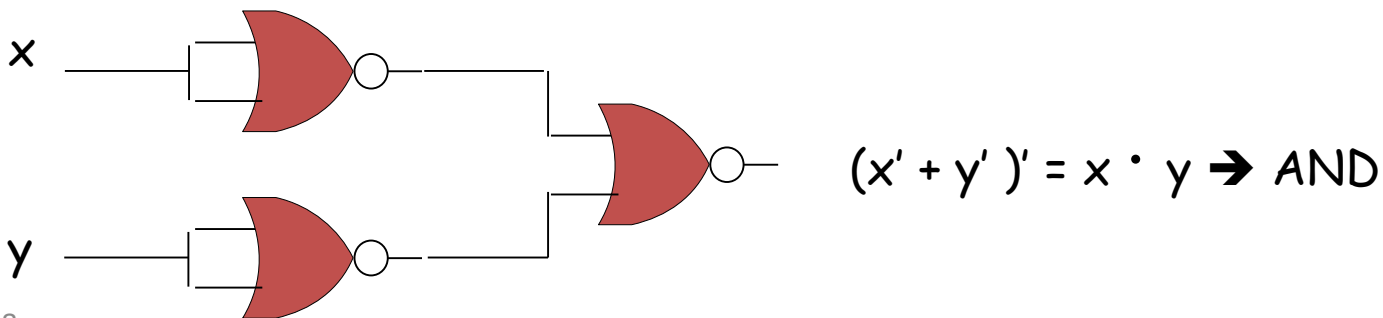
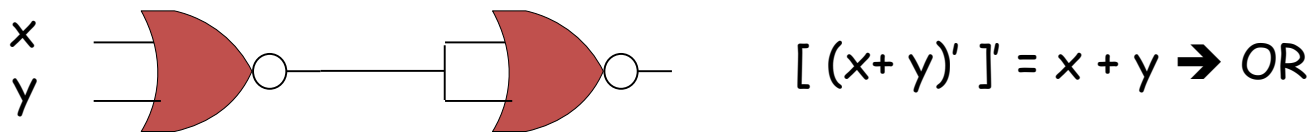
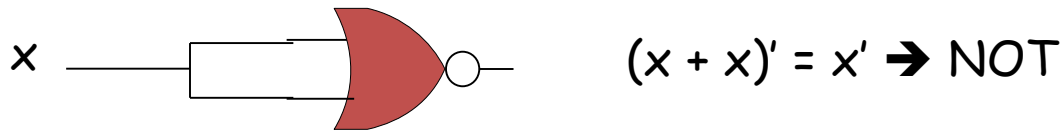


# Example: Multi-Level NAND Gates



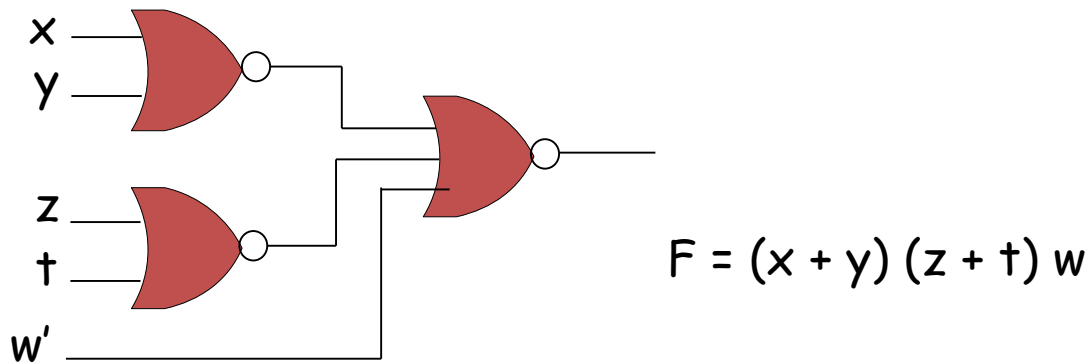
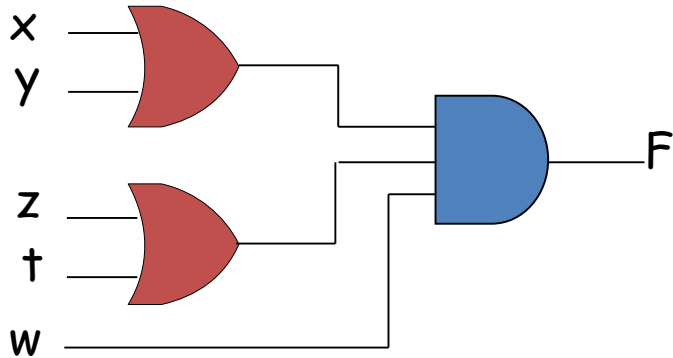
# Design with NOR Gates

- NOR is the dual operation of NAND.
  - All rules and procedure we used in the design with NAND gates apply here in a similar way.
  - Function is implemented easily if it is in product of sums form.



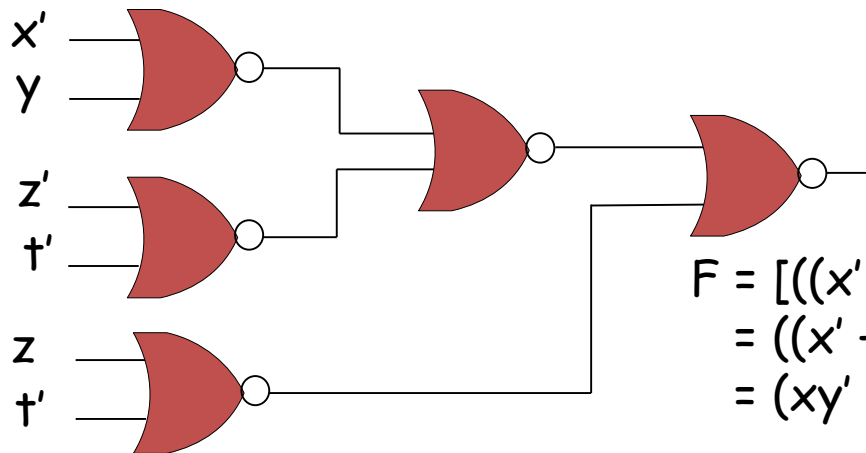
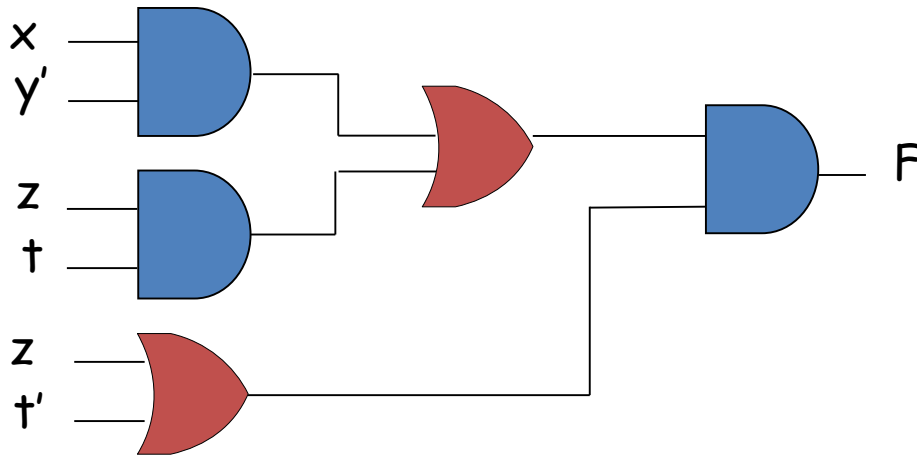
# Example: Design with NOR Gates

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



# Example: Design with NOR Gates

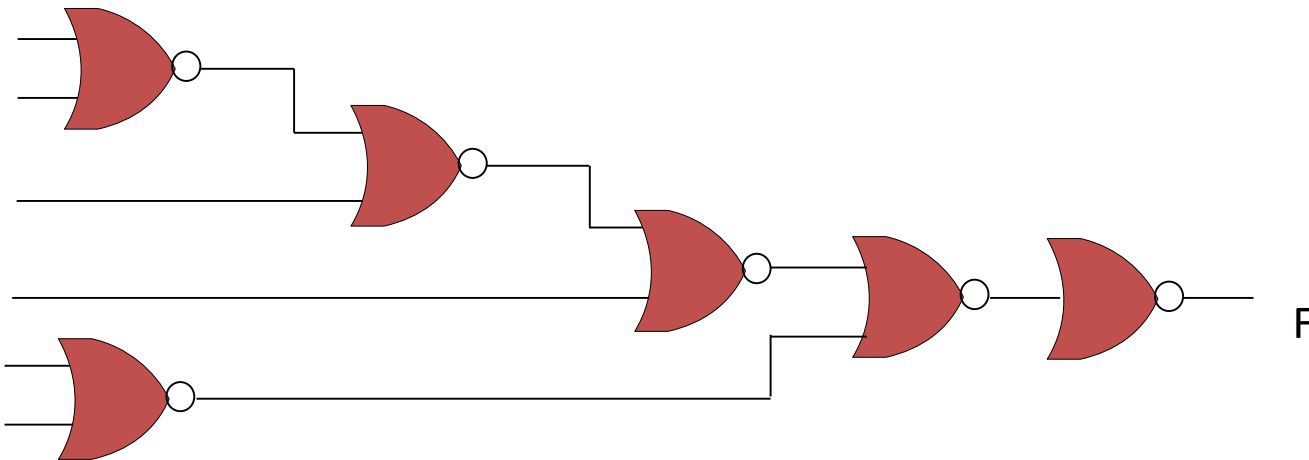
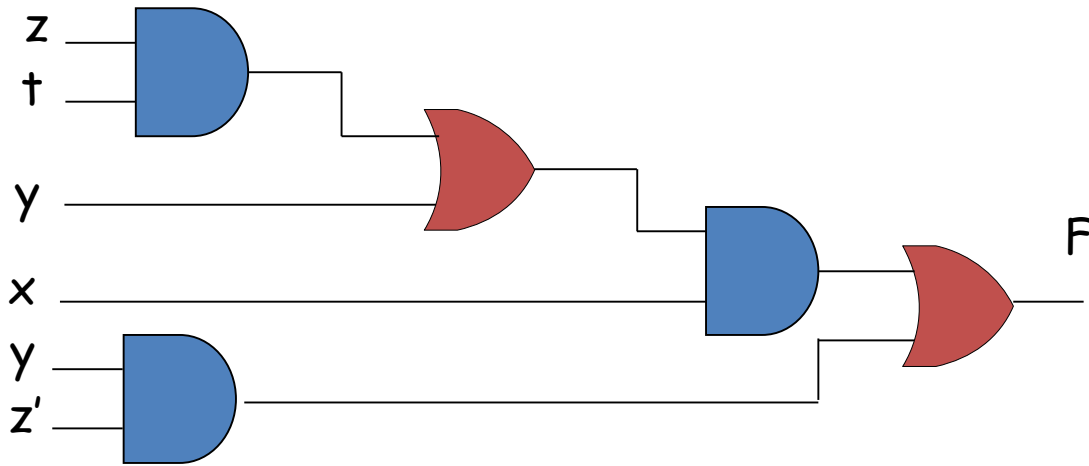
- $F = (xy' + zt)(z + t')$



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

# Harder Example

- Example:  $F = x(zt + y) + yz'$



# 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

$$\begin{aligned} - (x \oplus y) \oplus z &= (xy' + x'y) \oplus z \\ &= (xy' + x'y) z' + (xy' + x'y)' z \\ &= xy'z' + x'yz' + (xy + x'y') z \\ &= xy'z' + x'yz' + xyz + x'y'z \\ &= \Sigma (4, 2, 7, 1) \end{aligned}$$

# 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.

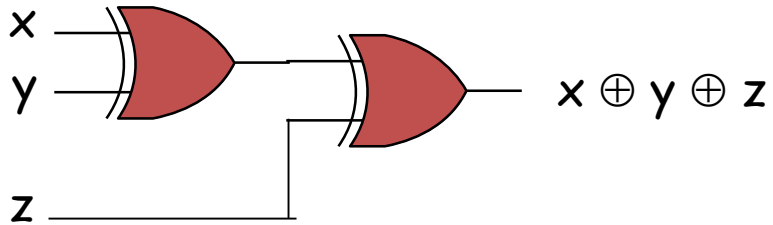
x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Odd function

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

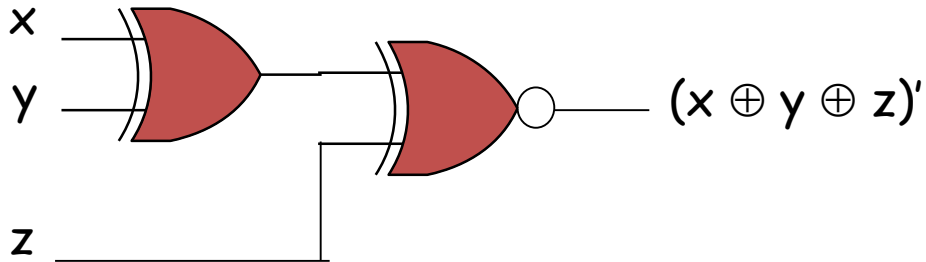
Even function

# Odd & Even Functions



Odd function

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



# Adder Circuit for Integers

- Addition of two-bit numbers
  - $Z = X + Y$
  - $X = (x_1 x_0)$  and  $Y = (y_1 y_0)$
  - $Z = (z_2 z_1 z_0)$
- Bitwise addition
  1.  $z_0 = x_0 \oplus y_0$  (sum)  
 $c_1 = x_0 y_0$  (carry)
  2.  $z_1 = x_1 \oplus y_1 \oplus c_1$   
 $c_2 = x_1 y_1 + x_1 c_1 + y_1 c_1$
  3.  $z_2 = c_2$

# Adder Circuit

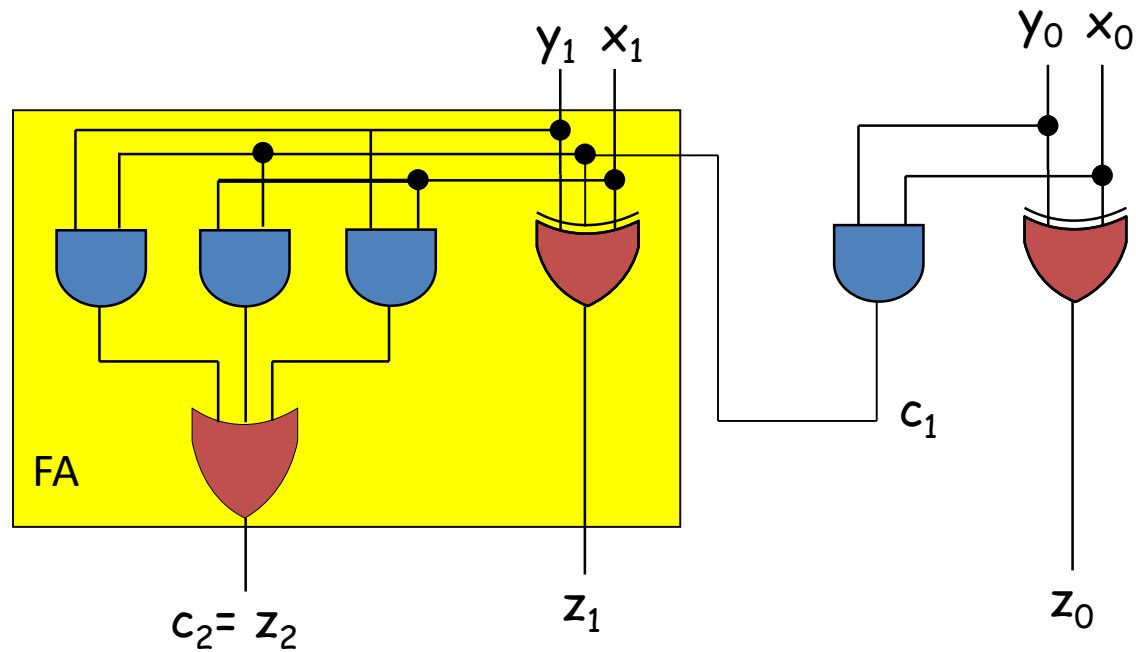
$$z_1 = x_1 \oplus y_1 \oplus c_1$$

$$c_2 = x_1 y_1 + x_1 c_1 + y_1 c_1$$

$$z_0 = x_0 \oplus y_0$$

$$c_1 = x_0 y_0$$

$$z_2 = c_2$$



# Comparator Circuit with NAND gates

- $F(X > Y)$

–  $X = (x_1 x_0)$  and  $Y = (y_1 y_0)$

$y_1 y_0$ $x_1 x_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

–  $F(x_1, x_0, y_1, y_0) = x_1 y_1' + x_1 x_0 y_0' + x_0 y_0' y_1'$

# Comparator Circuit - Schematic

