Gate-Level Minimization

section instructor:

Ufuk Çelikcan

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
    - Espresso

Two-Variable K-Map

- Two variables: x and y

  4 minterms:

  - $m_0 = x'y'$ → 00
  - $m_1 = x'y$ → 01
  - $m_2 = xy'$ → 10
  - $m_3 = xy$ → 11

x \ y	0	1
0	m_0	m_1
1	m_2	m_3

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

Example: Two-Variable K-Map

x \ y	0	1
0	1	1
1	1	0

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

Remember the Shortcuts

- $x + x = x \quad \Leftrightarrow \quad x \cdot x = x$
- $x + 1 = 1 \Leftrightarrow x \cdot 0 = 0$
- $x + xy = x \Leftrightarrow x \cdot xy = x$ [Absorption]
- $(x + y)' = x' \cdot y' \Leftrightarrow (x.y)'=x'+y'$ [DeMorgan]

Example: Two-Variable K-Map

x \ y	0	1
0	1	1
1	1	0

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

- We can do the same optimization by combining <u>adjacent cells.</u>

Three-Variable K-Map

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

- <u>Adjacent squares</u>: 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

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

x \ yz	00	01	11	10
0				
1				

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

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

x \ yz	00	01	11	10
0				
1				

- $F_2(x, y, z) =$

Example: Three-Variable K-Map

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

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

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

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

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

- $F_2(x, y, z) =$

Example: Three-Variable K-Map

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

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

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

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

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

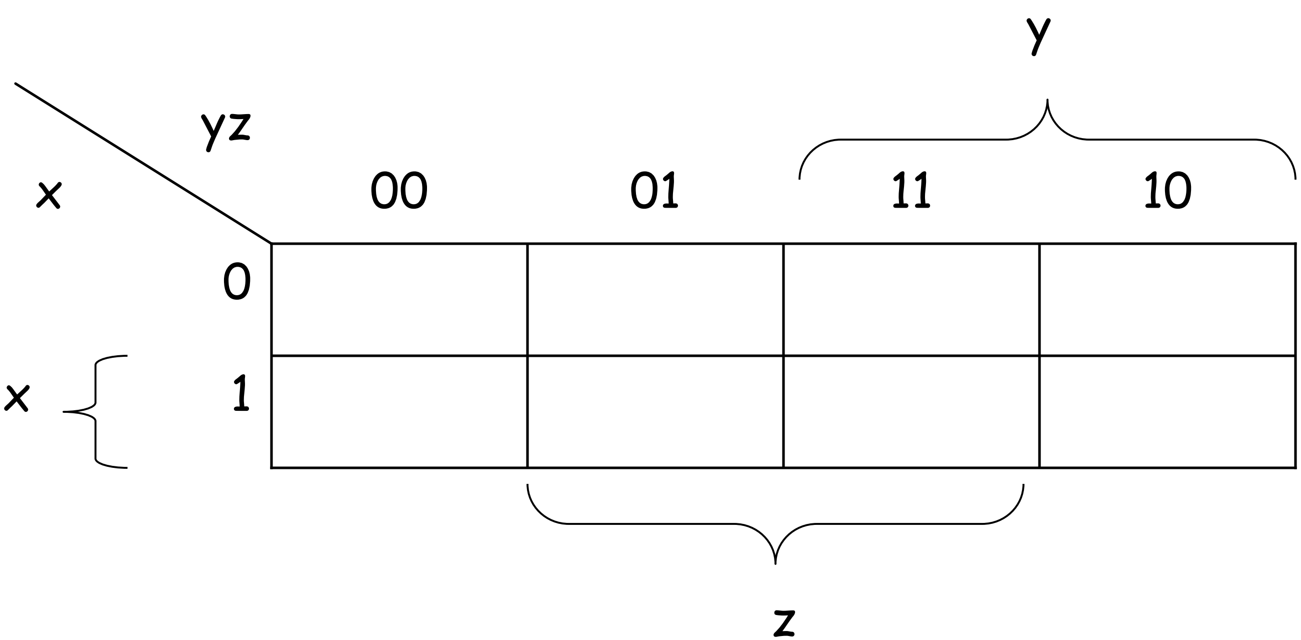
- $F_2(x, y, z) = xz' + yz$

In 3-Variable Karnaugh Maps

- 1 alone square represents one minterm with 3 literals
- 2 adjacent squares represent a term with 2 literals
- 4 adjacent squares represent a term with 1 literal
- 8 adjacent squares produce a function that is always equal to 1.

Example

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



$F_1(x, y, z) =$

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

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

y

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

z

$F_1(x, y, z) =$

Finding Sum of Minterms

- If a function is not expressed in sum of minterms form, it is possible to get it using K-maps
  - <u>Example</u>: F(x, y, z) = x'z + x'y + xy'z + yz

x \ yz	00	01	11	10
0				
1				

Finding Sum of Minterms

- If a function is not expressed in sum of minterms form, it is possible to get it using K-maps
  - <u>Example</u>: 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

Four-Variable K-Map

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

z

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

t

x

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				
01				
11				
10				

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

Example: Four-Variable K-Map

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

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

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

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

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

Example: Four-Variable K-Map

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

xy \ zt	00	01	11	10
00				
01				
11				
10				

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

Example: Four-Variable K-Map

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

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

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

Example: Four-Variable K-Map

- F(x,y,z,t) = x’y’z’ + y’zt’ + x’yzt’ + 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) =

Prime Implicants

- Prime Implicant: is a product term obtained by combining maximum possible number of adjacent squares in the map
- If a minterm can be covered by only one prime implicant, that prime implicant is said to be an **<u>essential prime implicant</u>.**
  - 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) = Σ (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)

xy \ zt	00	01	11	10
00				
01				
11				
10				

Example: Prime Implicants

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

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

Example: Prime Implicants

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

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

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

- Which ones are the **essential prime implicants?**

Example: Prime Implicants

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

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

- Why are these the **essential prime implicants?**

Example: Prime Implicants

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

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

xy \ 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}

- $y't'$ – 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

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

xy \ 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}

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

Example: Prime Implicants

xy \ zt	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} → zt or y’z
- m_9 → 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
  1. Find out all the essential prime implicants
  2. Then find the other prime implicants that cover the minterms that are not covered by the essential prime implicants. There are more than one way to choose those.
  3. 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 → 32 squares, 6 variables → 64 squares
  - Somewhat more practical way for F(x, y, z, t, w) :

yz \ tw	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}

x = 0

yz \ tw	00	01	11	10
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 $m_{15} \rightarrow m_{31}$
- 6-variables: Use **four** 4-variable maps to obtain 64 squares required for six variable optimization
- **Alternative way:** Can use computer programs
  - Quine-McCluskey method
  - Espresso method

Example: Five-Variable Map

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

yz \ tw	00	01	11	10
00				
01				
11				
10				

$x = 0$

yz \ tw	00	01	11	10
00				
01				
11				
10				

$x = 1$

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

Example: Five-Variable Map

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

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

x = 0

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

x = 1

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

Example: Five-Variable Map

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

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

x = 0

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

x = 1

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

Product of Sums Simplification

- So far
  - simplified expressions from Karnaugh maps are in <u>sum of products</u> form.
- Simplified <u>product of sums</u> can also be derived from Karnaugh maps.
- Method:
  - A square with 1 actually represents a "<u>minterm</u>"
  - Similarly an empty square (a square with 0) represents a "<u>maxterm</u>".
  - 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

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

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

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

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

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

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

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

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

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

Example: Product of Sums

1. Find F'(x,y,z,t) in Sum of Products form by grouping 0's
2. Apply DeMorgan's theorem to F' (use dual theorem) to find F in Product of Sums form

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

Example: Product of Sums

1. Find F’(x,y,z,t) in Sum of Products form by grouping 0’s
2. Apply DeMorgan’s theorem to F’ (use dual theorem) to find F in Product of Sums form

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

Example: Product of Sums

1. Find F’(x,y,z,t) in Sum of Products form by grouping 0’s
2. Apply DeMorgan’s theorem to F’ (use dual theorem) to find F in Product of Sums form

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

$F' = yt' + zt + xy$

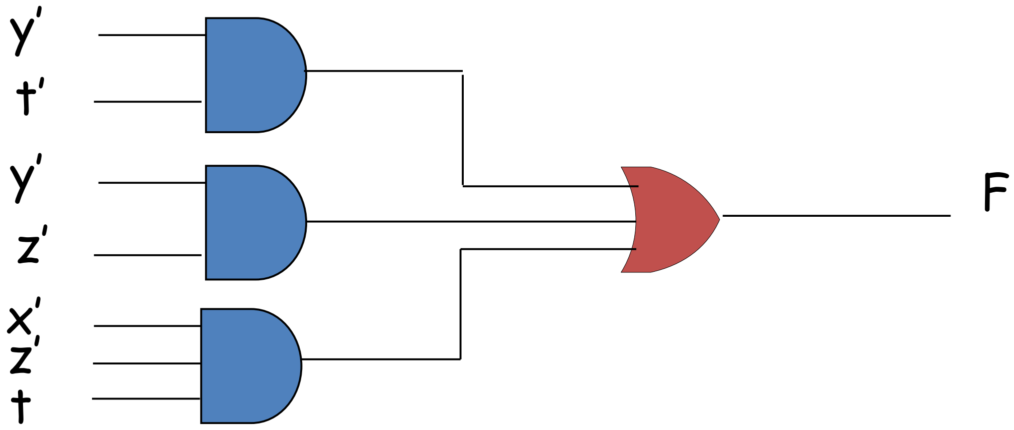
Example: Product of Sums

1. Find F'(x,y,z,t) in Sum of Products form by grouping 0's
2. Apply DeMorgan's theorem to F' (use dual theorem) to find F in Product of Sums form

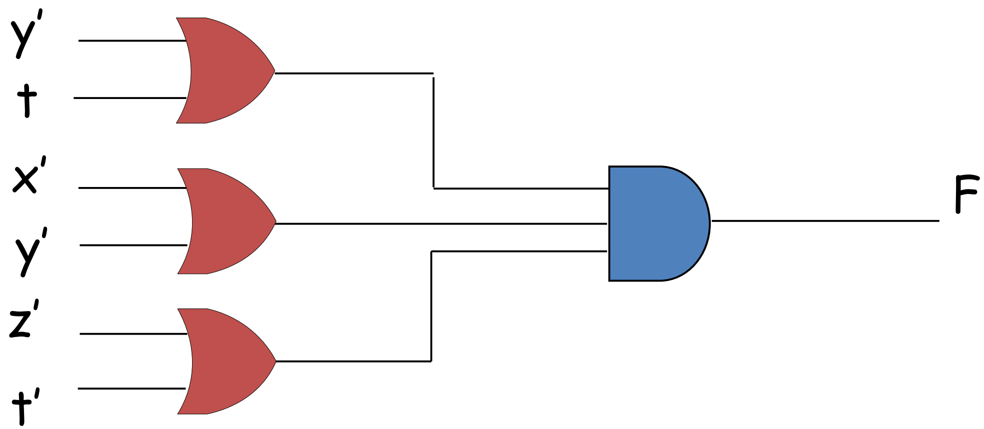
xy \ zt	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

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

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) =

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	0			0
1		0	0	

F(x, y, z) =

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	0			0
1		0	0	

$F(x, y, z) =$

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

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

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

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 input combinations
  - Such function are referred as <u>incompletely specified functions</u>
  - **therefore, the corresponding output values do not have to be defined**
  - This may significantly reduces the circuit complexity
  - 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 <u>don’t care</u> 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 as 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

xy \ 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

$F =$

$F_1 =$ or

$F_2 =$

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

xy \ 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

$F =$

$F_1 =$ 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
  - But 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.

xy \ 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’ =

F =

Quine-McCluskey Method

- Better than kmap for computers because a computer cant break down a graphical thing like K-map, but it can easily solve by QM
- It is functionally identical to Karnaugh mapping,
- but the tabular form makes it more efficient for use in computer algorithms,
- and it also gives a deterministic way to check that the minimal form of a Boolean function has been reached.
- It is sometimes referred to as the tabulation method.

Quine-McCluskey Method

- $F(x1,x2,x3,x4)=\Sigma\ 2,4,6,8,9,10,12,13,15$

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

Quine-McCluskey Method

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

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

List 3				
mi	x1	x2	x3	x4
8,9,12,13	1	-	0	-
8,12,9,13	1	-	0	-

Finished

Quine-McCluskey Method

List 1					
mi	x1	x2	x3	x4	

List 2					
mi	x1	x2	x3	x4	
2,6	0	-	1	0	t2
2,10	-	0	1	0	t3
4,6	0	1	-	0	t4
4,12	-	1	0	0	t5
8,10	1	0	-	0	t6
13,15	1	1	-	1	t7

List 3					
mi	x1	x2	x3	x4	
8,9,12,13	1	-	0	-	t1

Finished

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

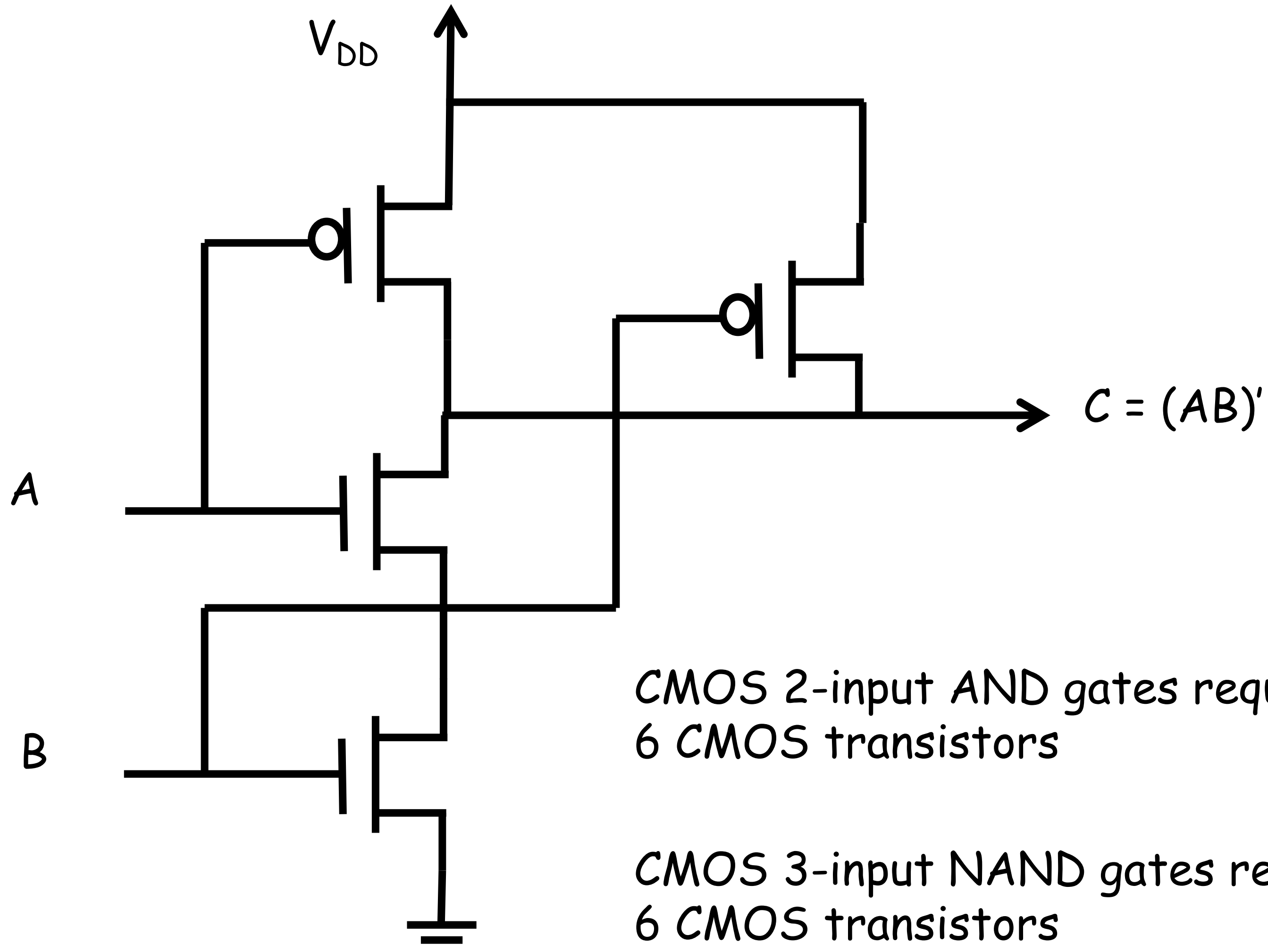
t6 is a subset of t3

$$F(x1,x2,x3,x4)=t1+t7+t3+t4$$

$$=x1x3' + x1x2x4 + x2'x3x4' + x1'x2x4'$$

NAND and NOR Gates

- NAND and NOR gates are easier to fabricate

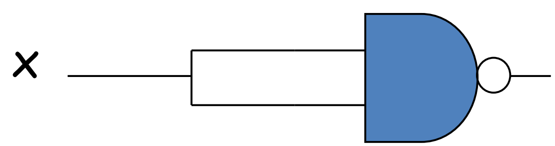


CMOS 2-input AND gates requires
6 CMOS transistors

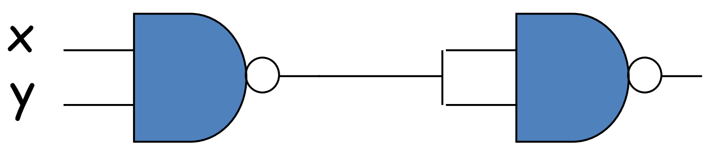
CMOS 3-input NAND gates requires
6 CMOS transistors

Design with NAND or NOR Gates

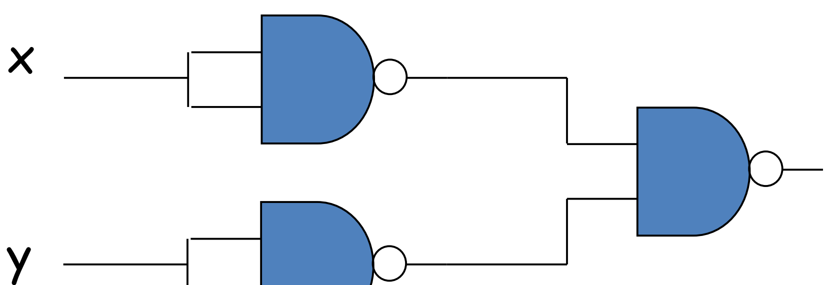
- It is beneficial to derive conversion rules from Boolean functions given in terms of AND, OR, an NOT gates into equivalent NAND or NOR implementations



$(x\,x)' = x'$ ➔ NOT

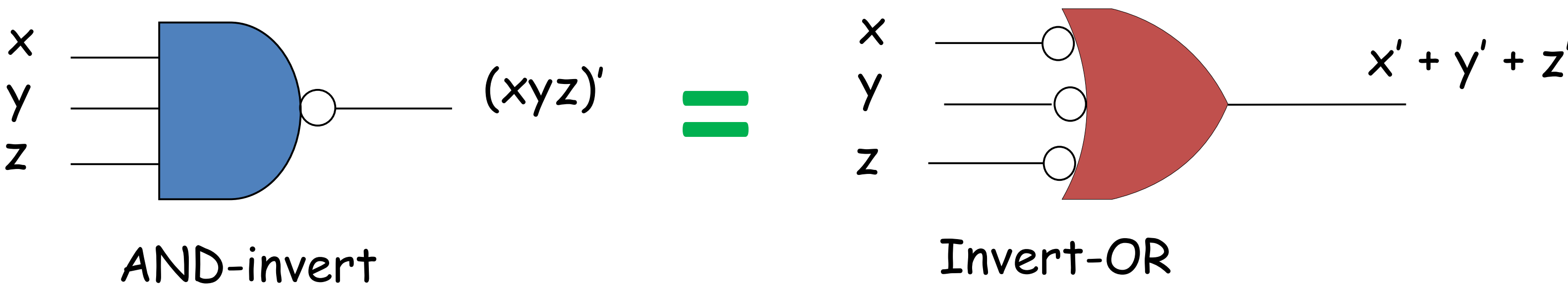


$[\,(x\,y)'\,]' = x\,y$ ➔ AND



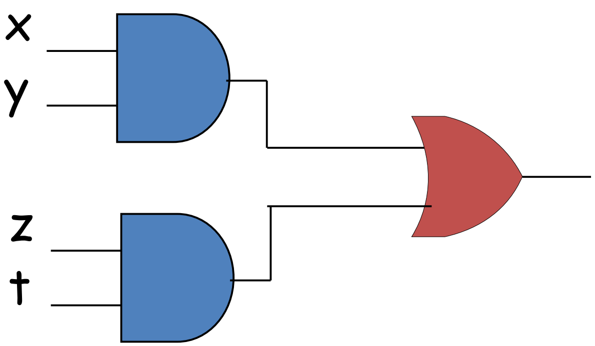
$(x'\,y'\,)' = x + y$ ➔ OR

New Notation

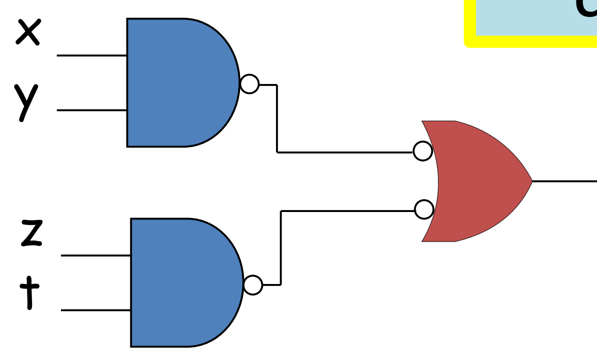


- Implementing a Boolean function with NAND gates is easy if it is in <u>sum of products form</u>.
- Example: F(x, y, z, t) = xy + zt

ALWAYS USE THIS INTERMEDIATE FORM WITH BUBBLES TO AVOID CONFUSION

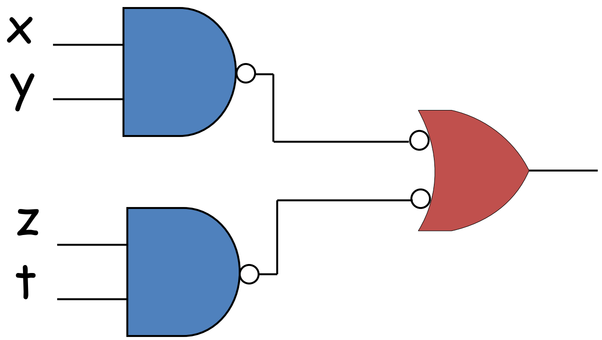


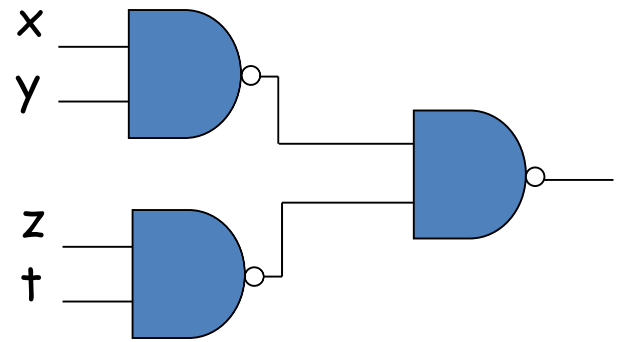
F(x, y, z, t) = xy + zt



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

The Conversion Method





$((xy)')' + ((zt)')' \quad = xy + zt = \quad [\,(xy)'\,(zt)'\,]\,'$

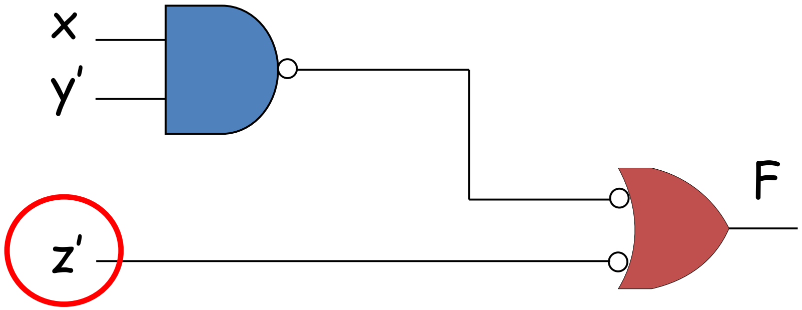
- Example: $F(x, y, z) = \sum(1, 3, 4, 5, 7)$

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

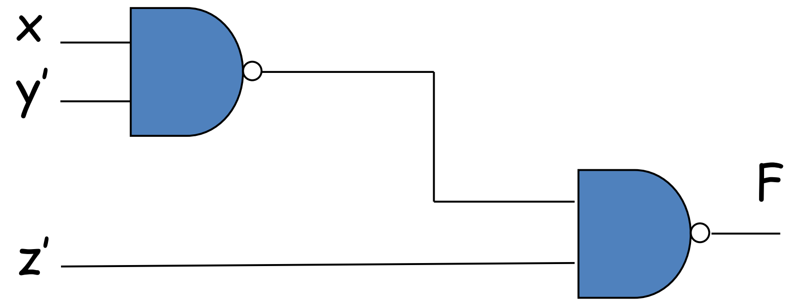
$F = z + xy'$

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

Example: Design with NAND Gates



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



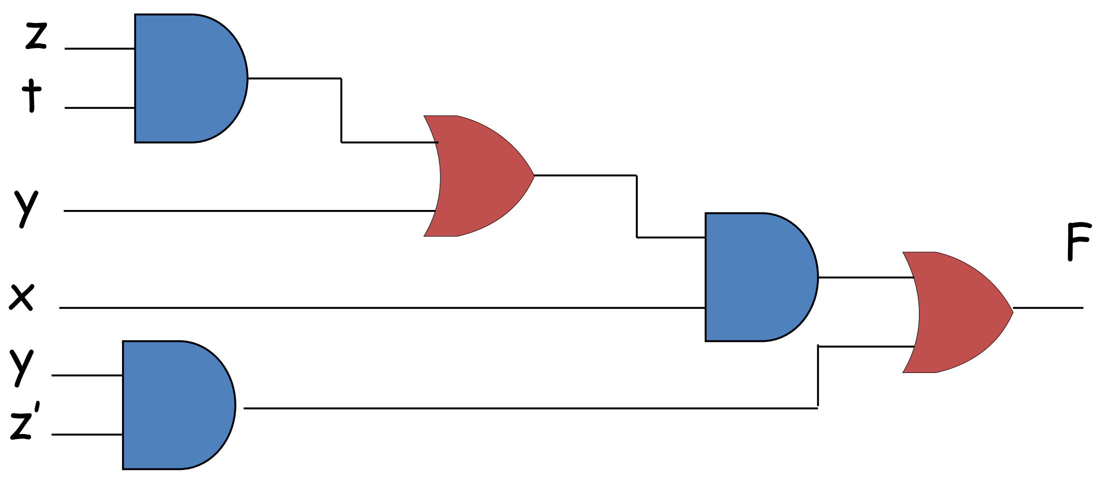
$F = z + xy'$

- 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^{nd} level,
  4. **A product term with single literal needs an inverter in the first level.** <u>Assume single, complemented literals are available.</u>

Multi-Level NAND Gate Designs

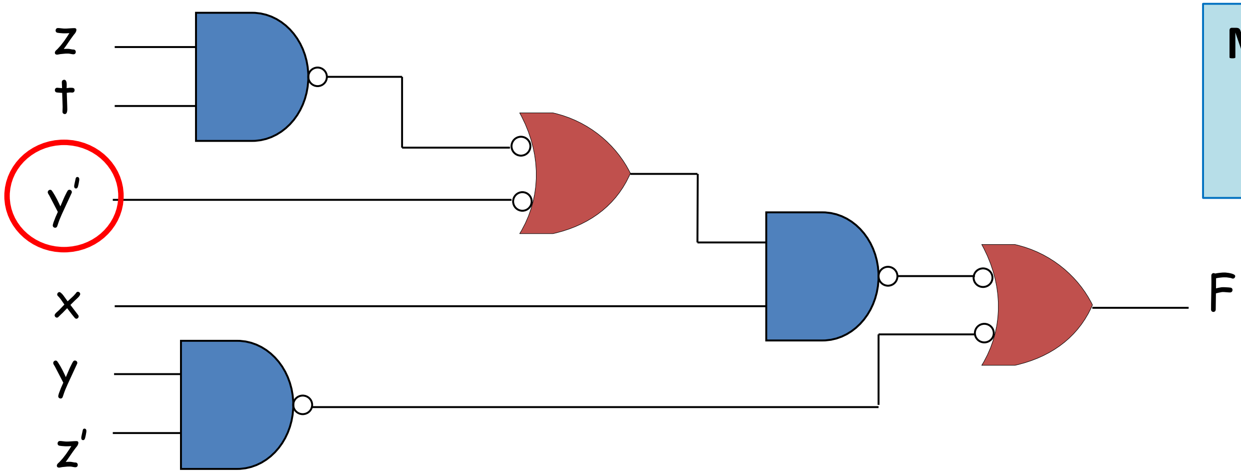
- The standard form results in two-level implementations
- Non-standard forms may raise a difficulty
- Example: F = x(zt + y) + yz'

4-level implementation

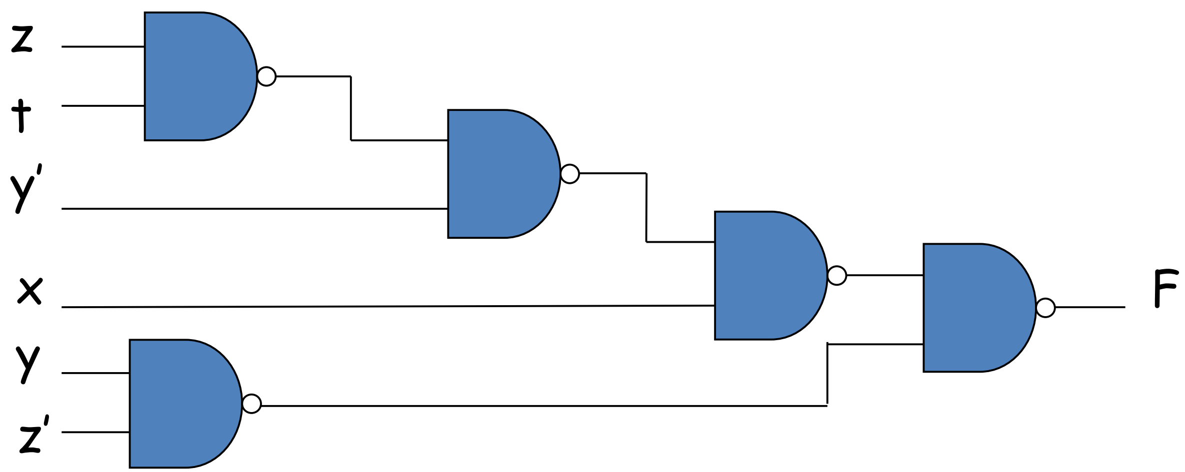


Example: Multilevel NAND…

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



MUST COMPENSATE FOR EVERY NON-COUPLED BUBBLE



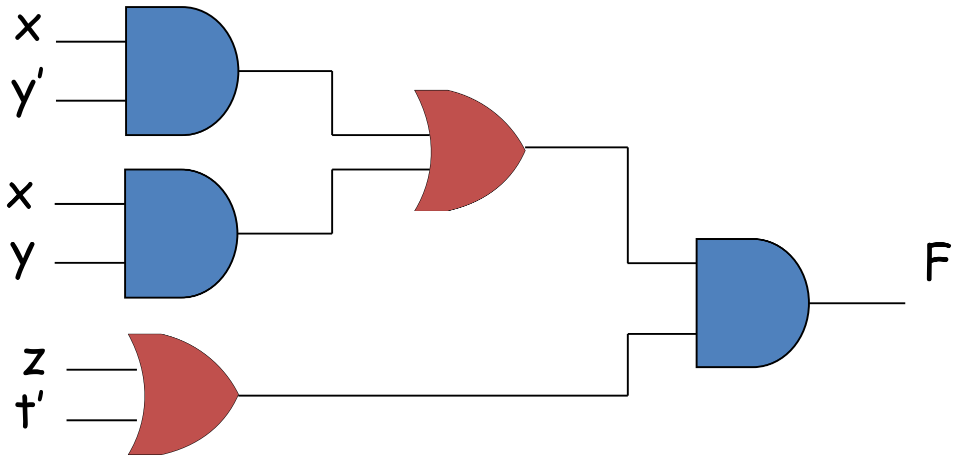
Design with Multi-Level NAND Gates

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, complement the input literal**

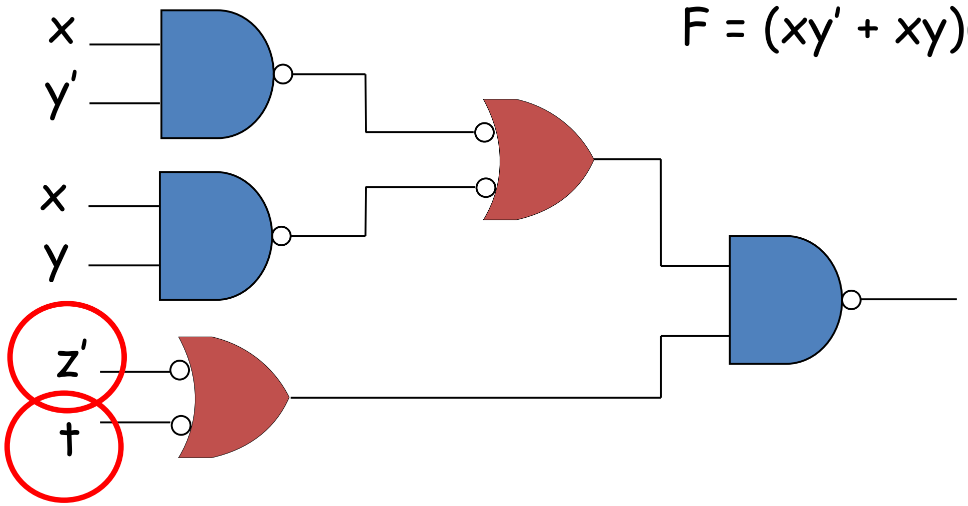
Another (Harder) Example

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



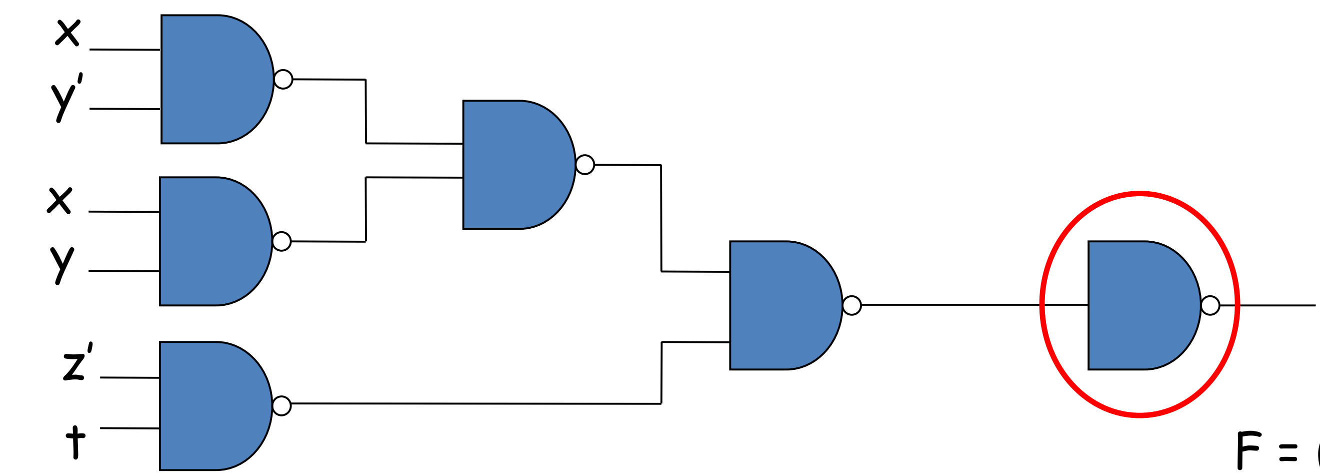
Example: Multi-Level NAND Gates

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



$G = [\,(xy' + xy)(z' + t)\,]'$

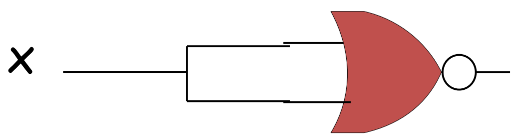
$F = (xy' + xy)(z + t')$



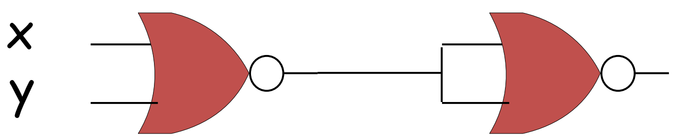
$F = (xy' + xy)(z + t')$

Design with NOR Gates

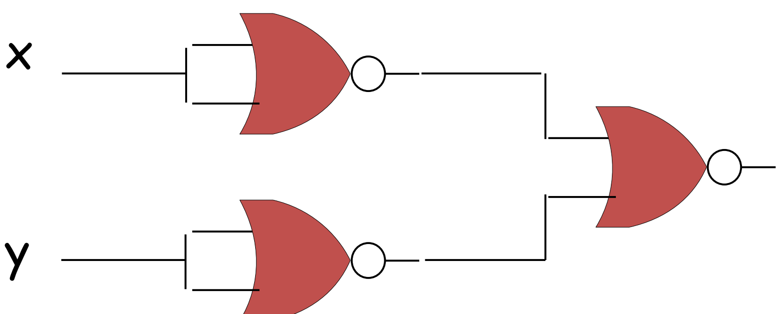
- NOR is the dual operation of NAND.
  – All rules and procedure we used in the design with NAND gates apply here in a similar way.
  – Function is implemented easily if it is in <u>product of sums form</u>.



$(x + x)' = x'$ ➔ NOT

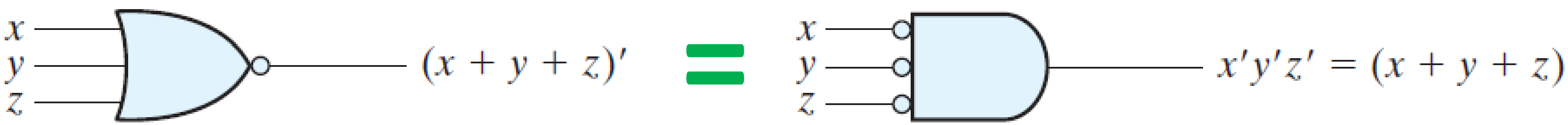


$[\,(x+ y)'\,]' = x + y$ ➔ OR



$(x' + y'\,)' = x \cdot y$ ➔ AND

Logic operations with NOR gates

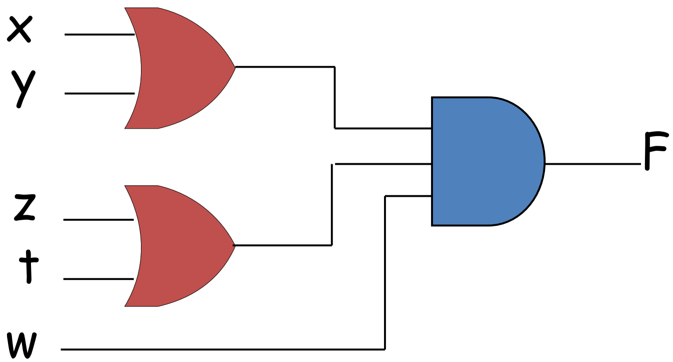


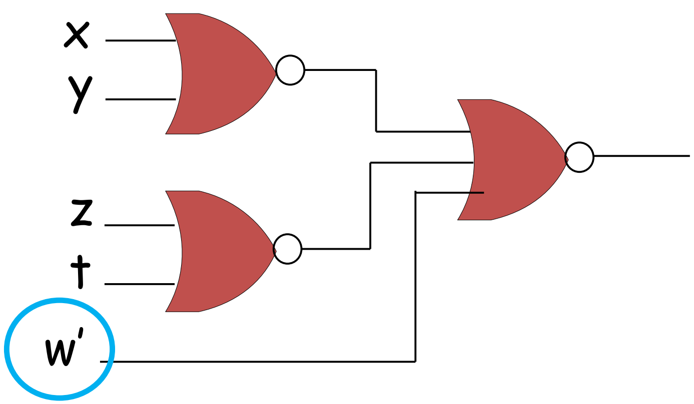
(a) OR-invert

(b) Invert-AND

Example: Design with NOR Gates

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

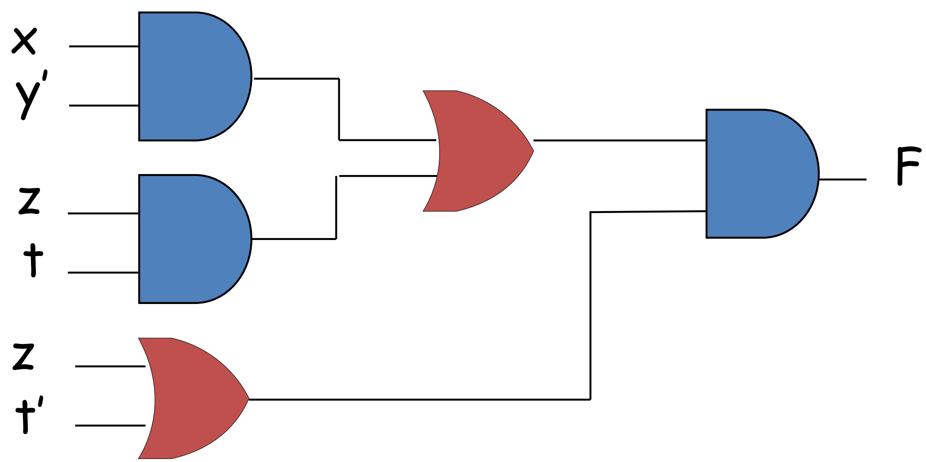


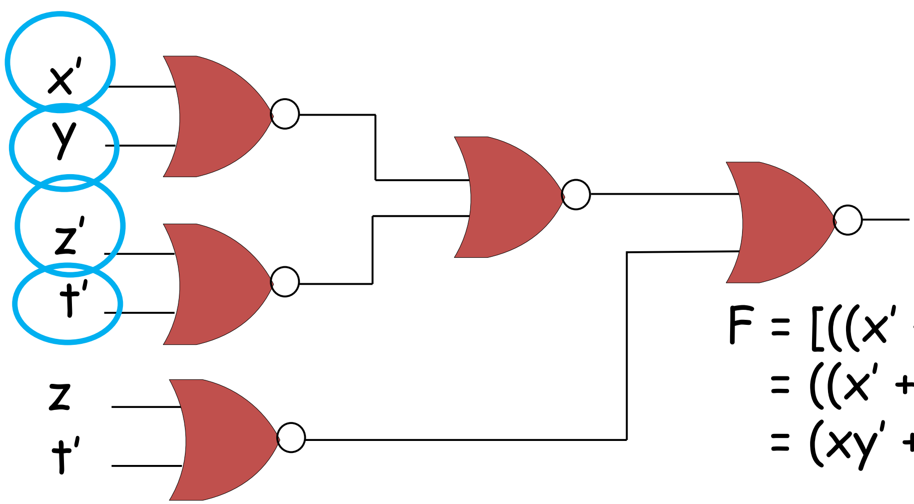


F = (x + y) (z + t) w

Example: Design with NOR Gates

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

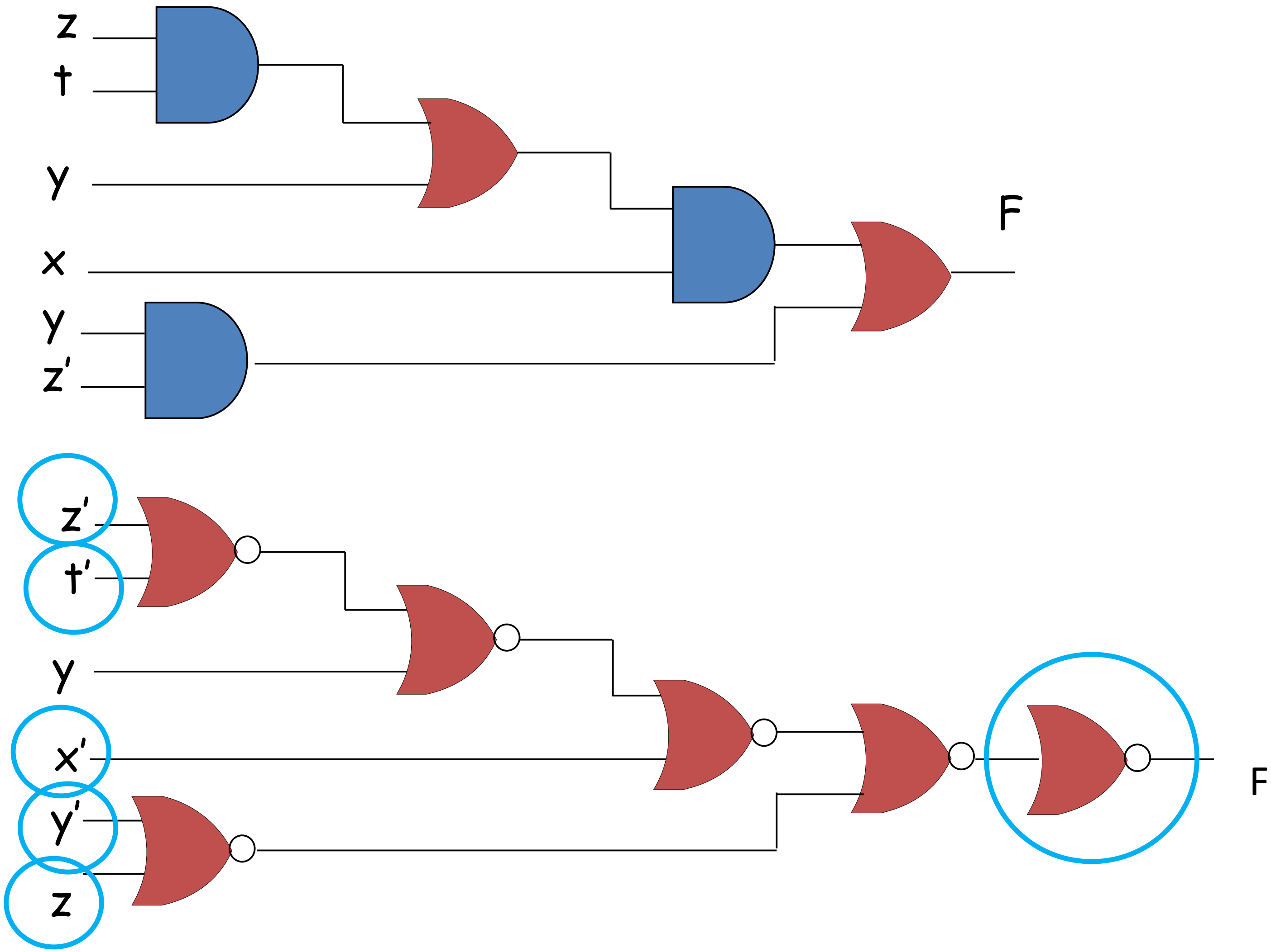




$$F = [((x' + y)' + (z' + t')')' + (z + t')']'$$
$$= ((x' + y)' + (z' + t')')(z + t')$$
$$= (xy' + zt)(z + t')$$

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 applications in arithmetic and error-detection circuits.
- Odd Function

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

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

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.

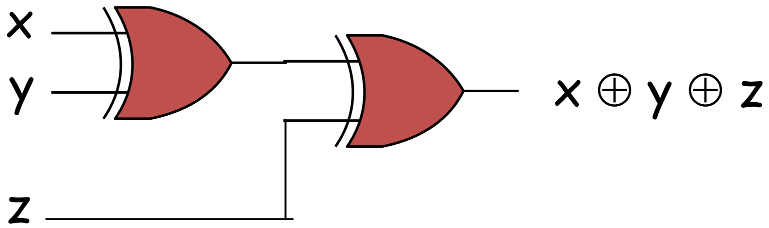
x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Odd function

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

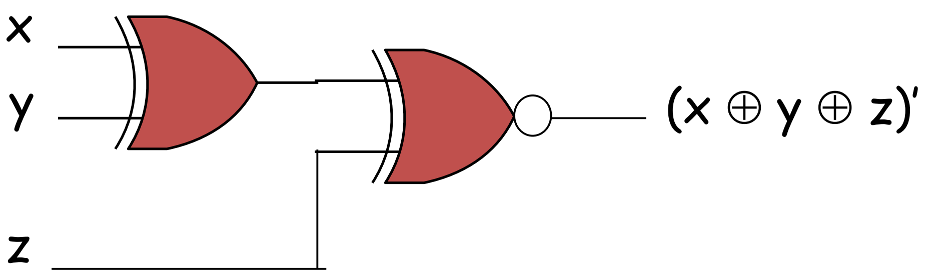
Even function

Odd & Even Functions



Odd function

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



Adder Circuit for Integers

- Addition of two 2-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

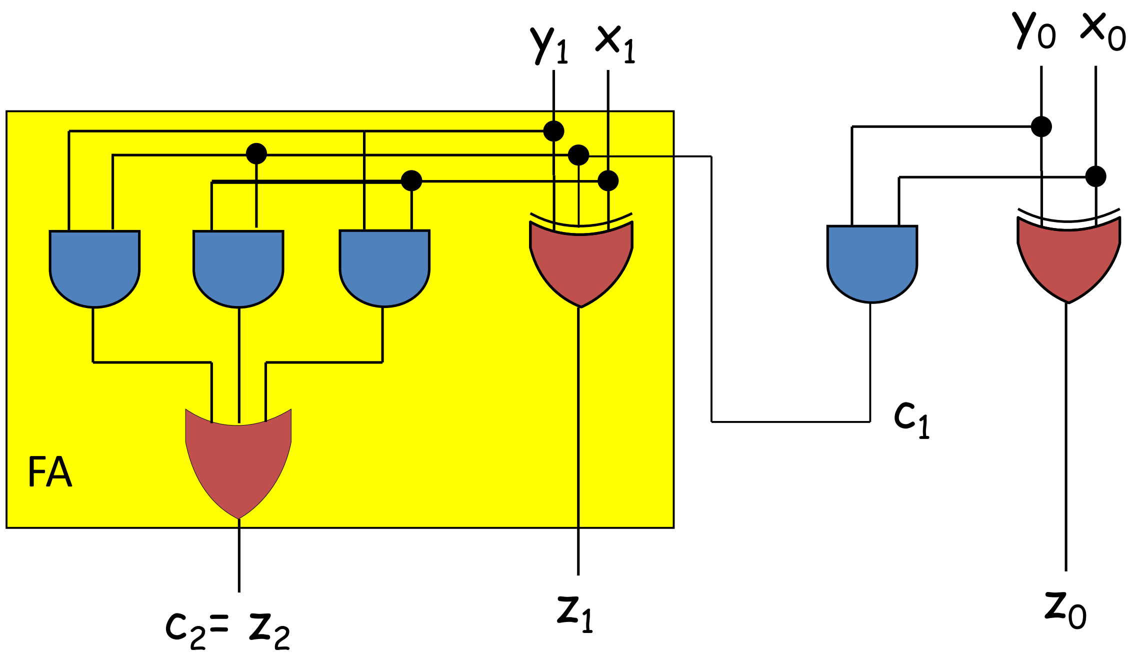
$z_0 = x_0 \oplus y_0$

$z_1 = x_1 \oplus y_1 \oplus c_1$

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

$z_2 = c_2$

$c_1 = x_0 y_0$



Comparator Circuit with NAND gates

- F(X>Y)

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

$x_1 x_0$ \ $y_1 y_0$	00	01	11	10
00				
01				
11				
10				

Comparator Circuit with NAND gates

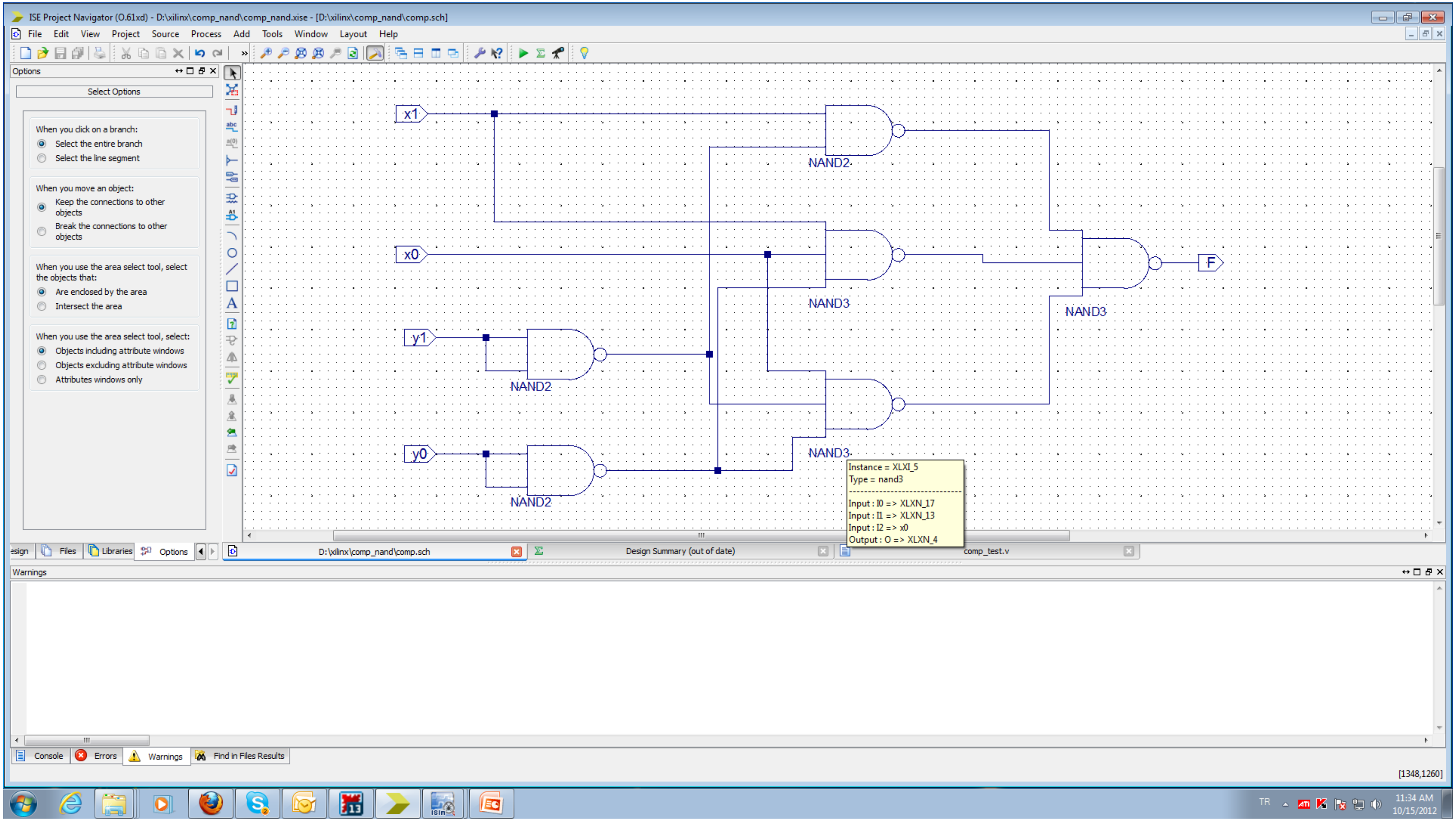
- F(X>Y)

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

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

$F(x_1, x_0, y_1, y_0) = x_1y_1' + x_1x_0y_0' + x_0y_0'y_1'$

Comparator Circuit - Schematic



Comparator Circuit - Simulation