CS 303
Logic & Digital System Design

Ömer Ceylan

Sabancı Üniversitesi

Gate-Level Minimization

Sabancı Üniversitesi

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 = …
- $F = x' + y'$

- 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

- 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

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

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

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

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

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

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

y: columns 11, 10; x: row 1; z: columns 01, 11

$F_1(x, y, z) =$

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	1	1
1		1	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

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
x	11	m_{12}	m_{13}	m_{15}	m_{14}	
	10	m_8	m_9	m_{11}	m_{10}	

t

Example: Four-Variable K-Map

- F(x,y,z,t) = Σ (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’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) y’t’ + y’z’+ 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 <u>essential</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	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

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

- 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

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
  - 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 → 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 $m_{15} \rightarrow m_{31}$
- 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

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

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

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

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

- $F(x, y, z, t) = \Sigma\ (0, 1, 2, 5, 8, 9, 10)$
  - 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

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

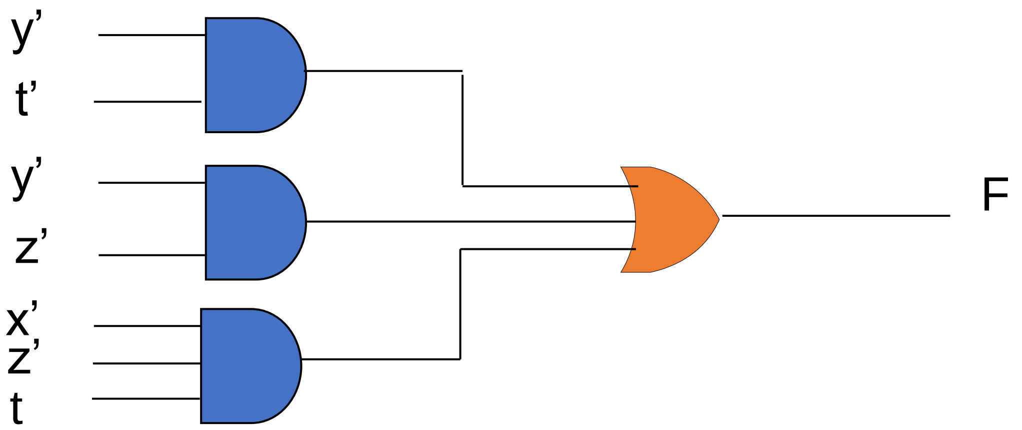
Example: Product of Sums

- F’(x,y,z,t) = zt + yt’ + xy
- Apply DeMorgan’s theorem (use dual theorem)
- F =

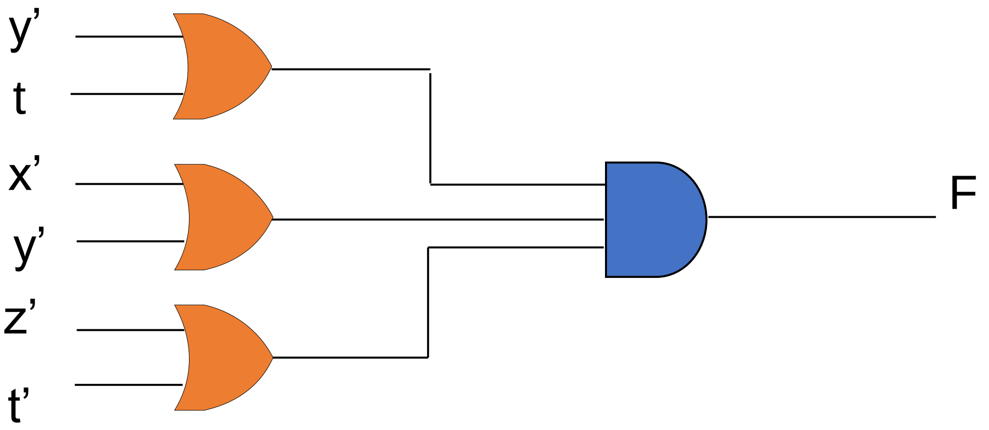
xy \ zt	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

F(x,y,z,t) = y’t’ + y’z’ + x’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:
  - $F(x, y, z) = \prod (0, 2, 5, 7)$

x \ yz	00	01	11	10
0	0	1	1	0
1	1	0	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:
  - $F(x, y, z, t) = (x' + y' + z')(y + t)$
  - $F'(x, y, z, t) = xyz + y't'$

xy \ zt	00	01	11	10
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 function are referred as <u>incompletely specified functions</u>
  - 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

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 <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 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

$F = zt + x'y't$

$F_1 = zt + x'y'$ 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.

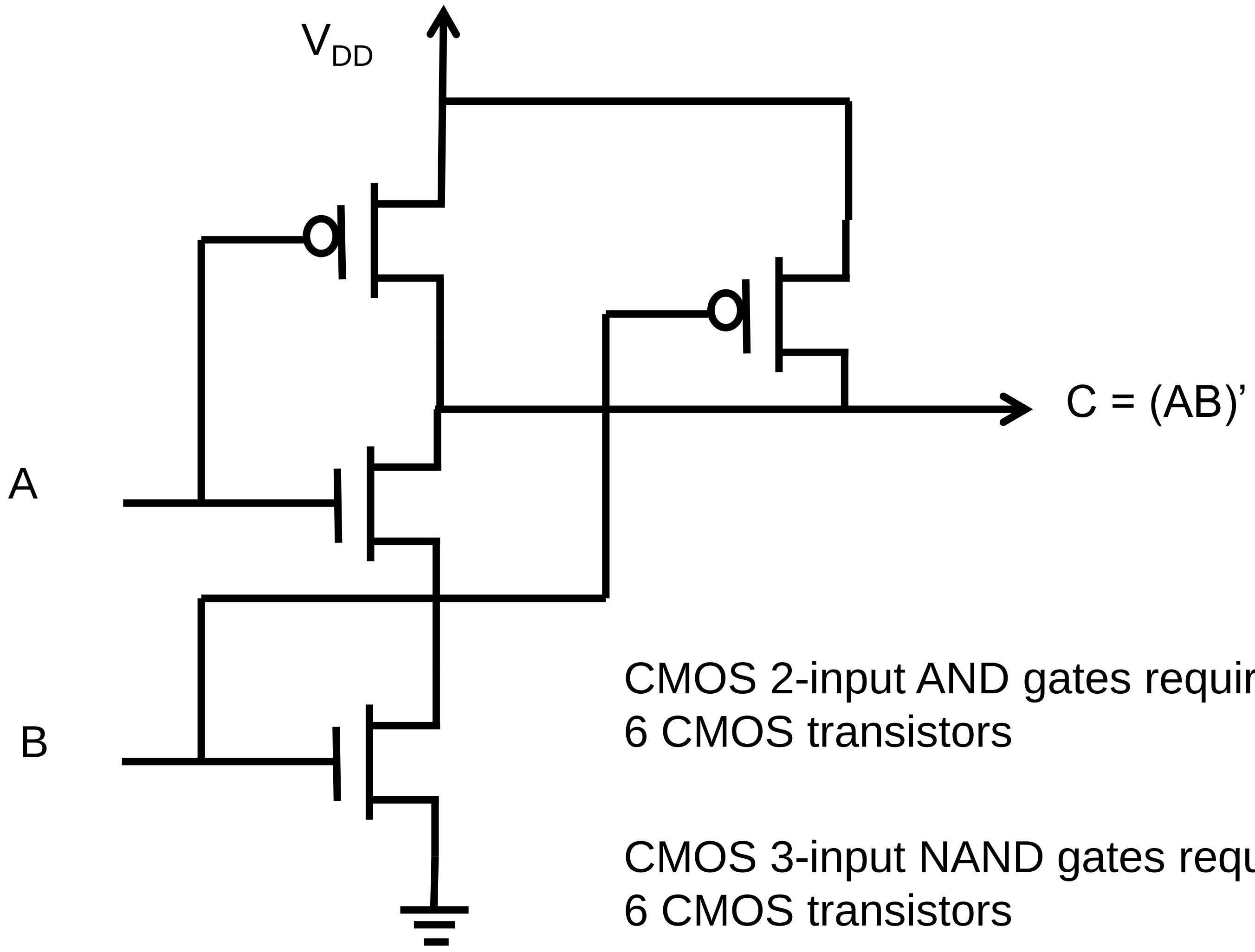
xy \ zt	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$F' = t' + xz'$

$F = t(x'+z)$

NAND and NOR Gates

- NAND and NOR gates are easier to fabricate

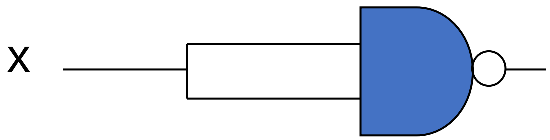


CMOS 2-input AND gates requires
6 CMOS transistors

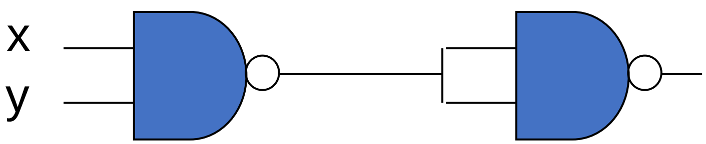
CMOS 3-input NAND gates requires
6 CMOS transistors

Design with NAND or NOR Gates

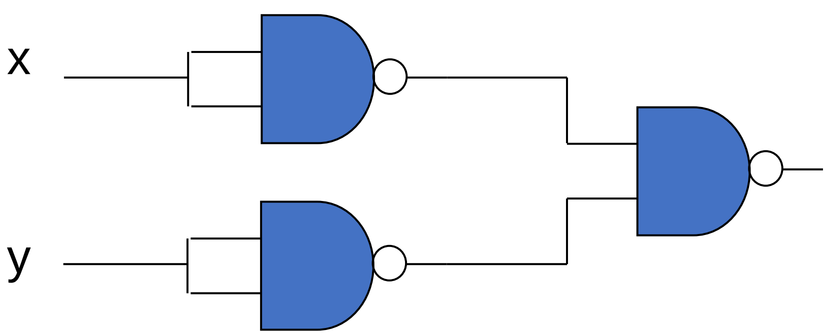
- It is beneficial to derive conversion rules <u>from</u> Boolean functions given in terms of AND, OR, an NOT gates <u>into</u> equivalent NAND or NOR implementations



(x x)’ = x’ ➔ NOT

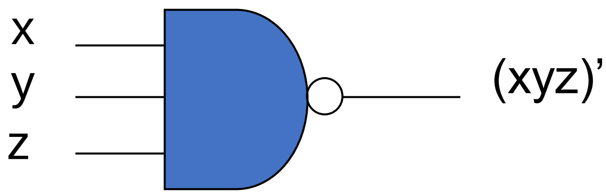


[(x y)’]’ = x y ➔ AND

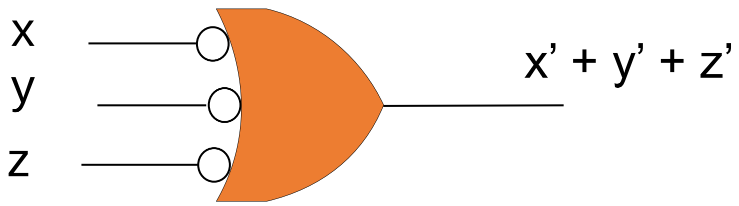


(x’ y’)’ = x + y ➔ OR

New Notation

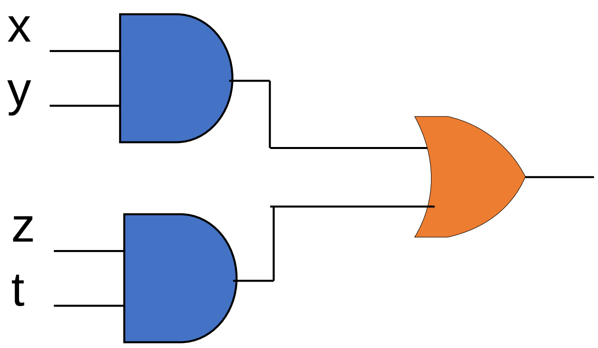


AND-invert

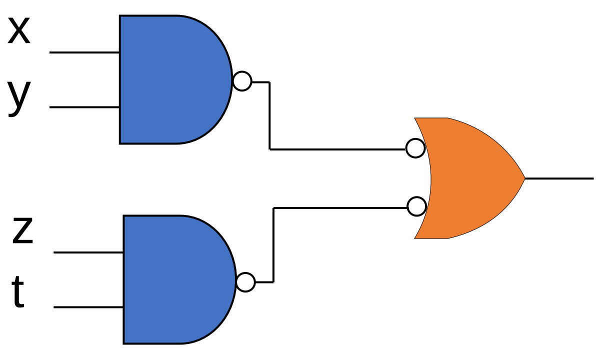


Invert-OR

- 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

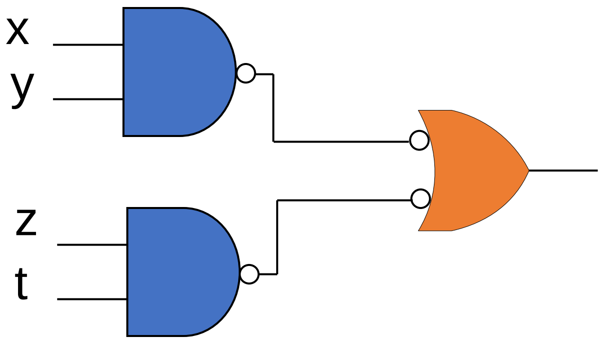


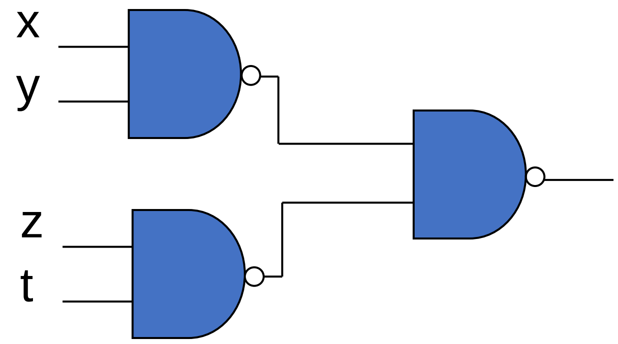
F(x, y, z, t) = xy + zt



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

The Conversion Method





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

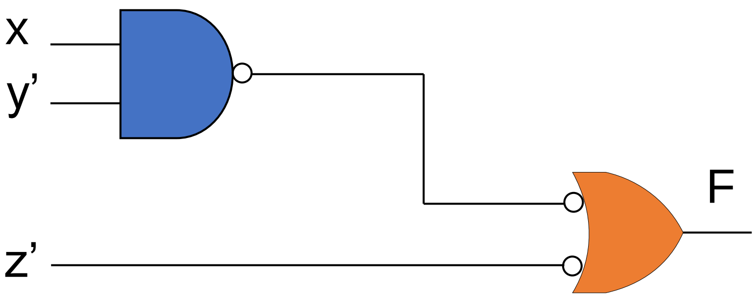
- Example: $F(x, y, z) = \Sigma(1, 3, 4, 5, 7)$

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

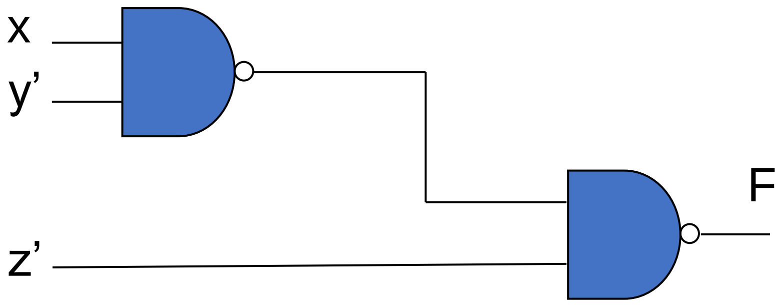
$F = z + xy'$

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

Example: Design with NAND Gates



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

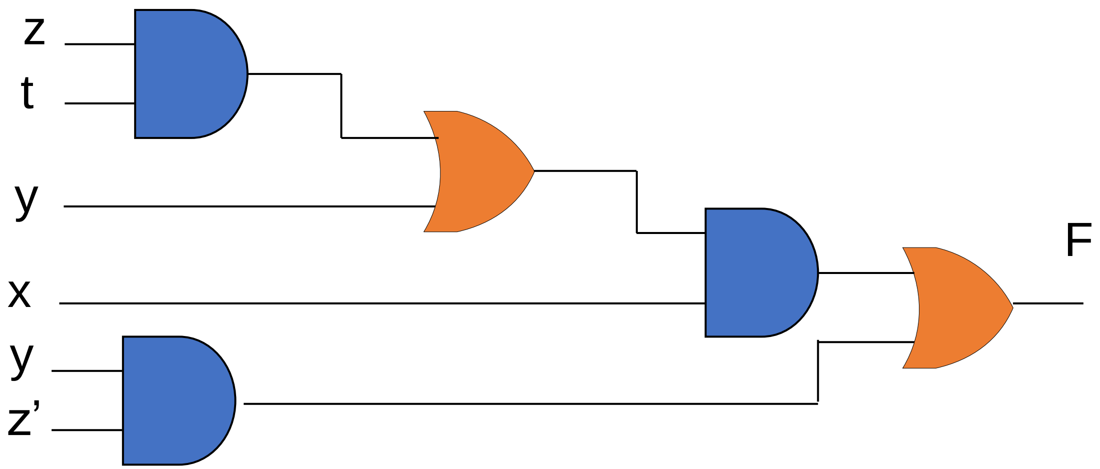


$F = z + xy'$

- 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 2nd level,
  4. A product term with single literal needs an inverter in the first level. Assume single, complemented literals are available.

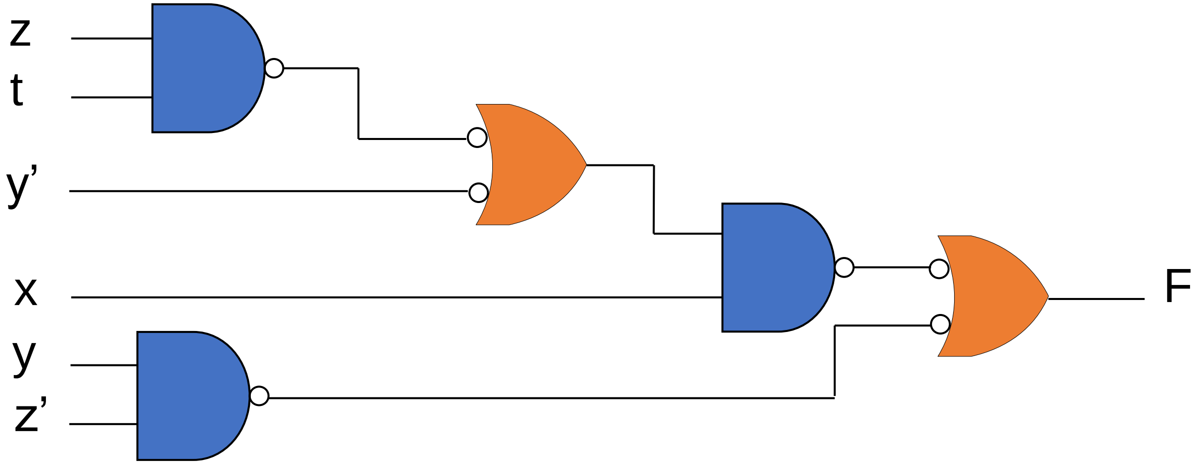
Multi-Level NAND Gate Designs

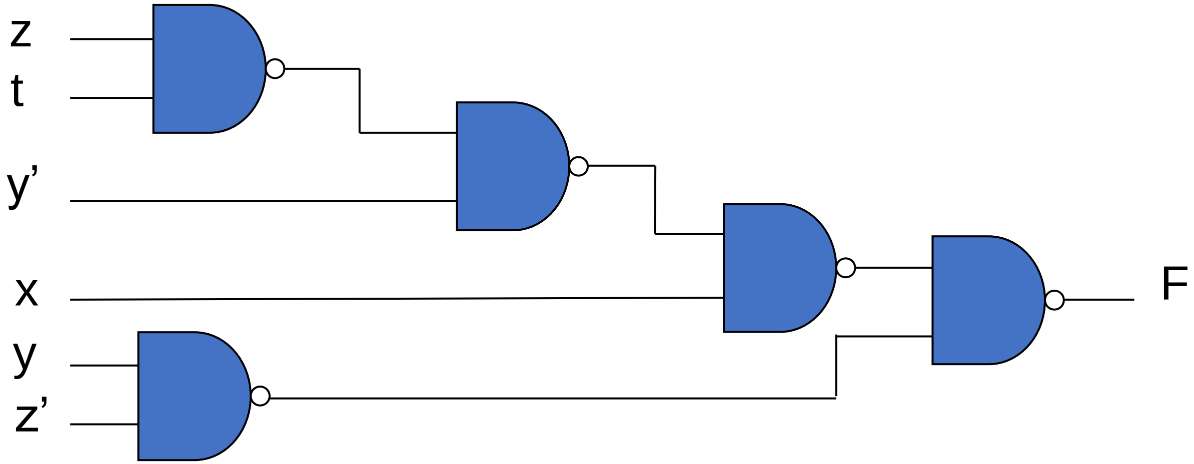
- The standard form results in two-level implementations
- Non-standard forms may raise a difficulty
- Example: F = x(zt + y) + yz’
  - 4-level implementation



Example: Multilevel NAND…

F = x(zt + y) + yz'





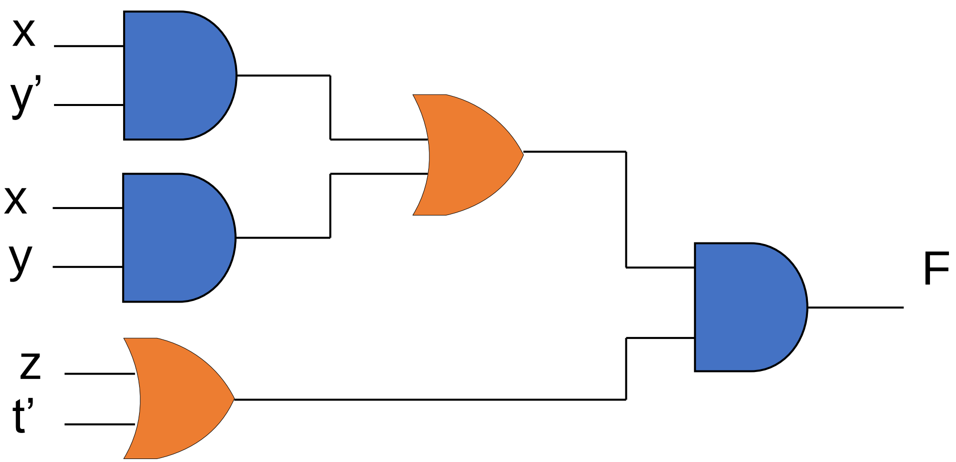
Design with Multi-Level NAND Gates

- <u>Rules</u>

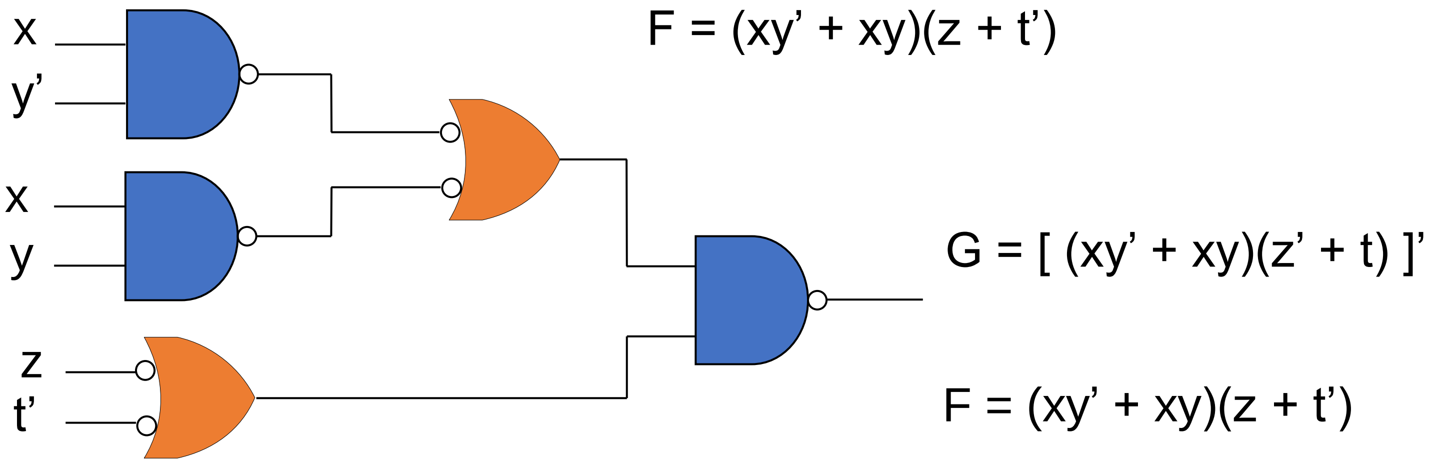
1. Convert all AND gates to NAND gates
2. Convert all OR gates to NAND gates
3. Insert an inverter (one-input NAND gate) at the output if the final operation is AND
4. Check the bubbles in the diagram. For every bubble along a path from input to output there must be another bubble. If not so,
   a. complement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

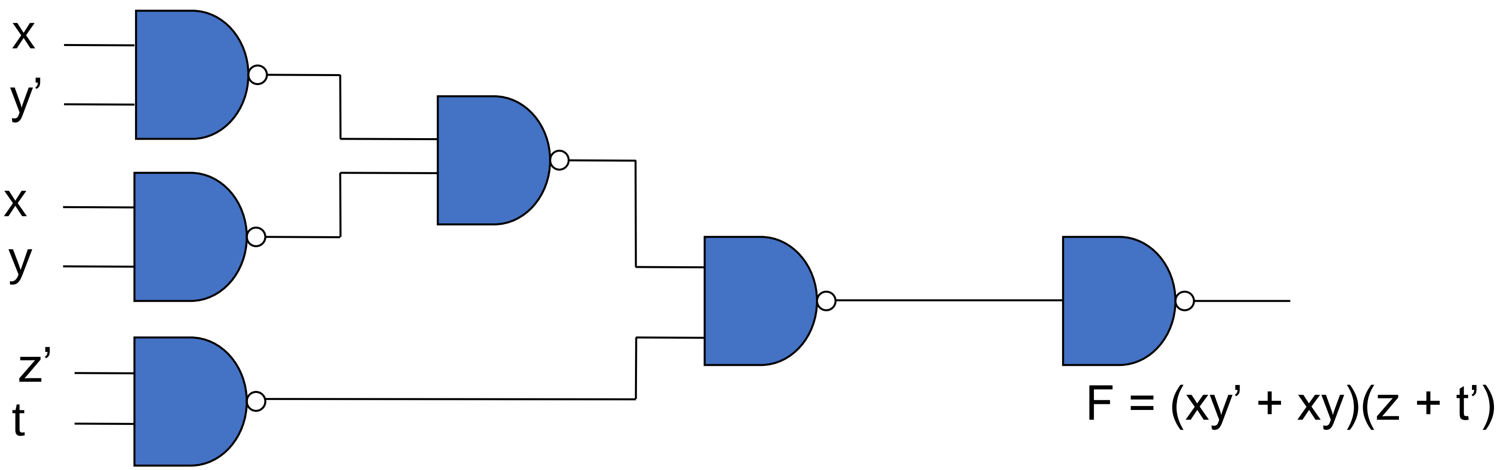
Another (Harder) Example

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



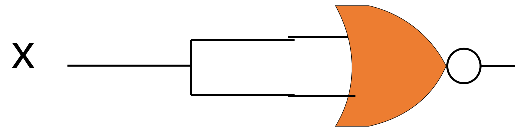
Example: Multi-Level NAND Gates



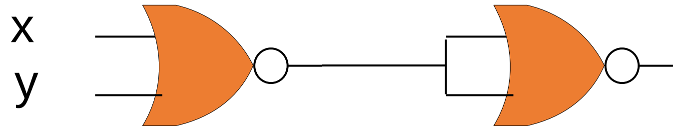


Design with NOR Gates

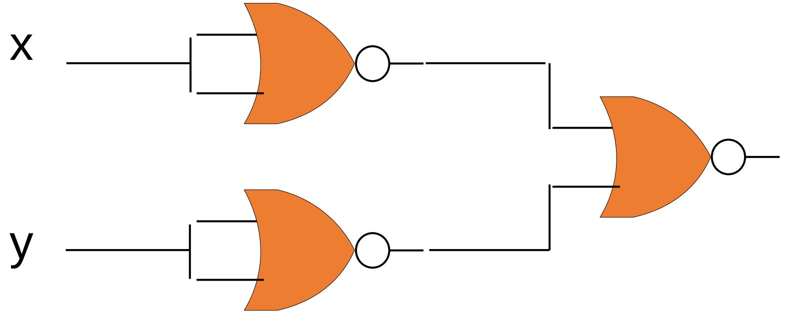
- NOR is the dual operation of NAND.
  - All rules and procedure we used in the design with NAND gates apply here in a similar way.
  - Function is implemented easily if it is in product of sums form.



$(x + x)' = x' \rightarrow$ NOT



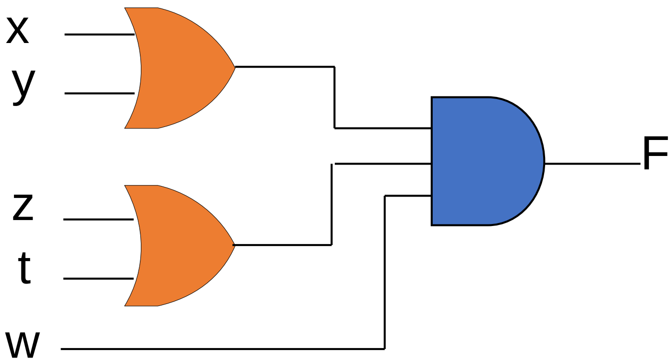
$[\,(x+ y)'\,]' = x + y \rightarrow$ OR

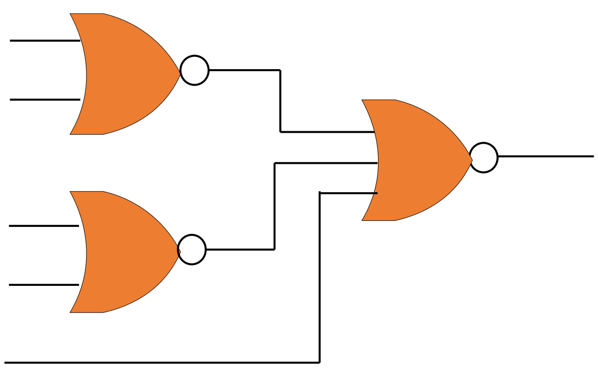


$(x' + y'\,)' = x \cdot y \rightarrow$ AND

Example: Design with NOR Gates

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

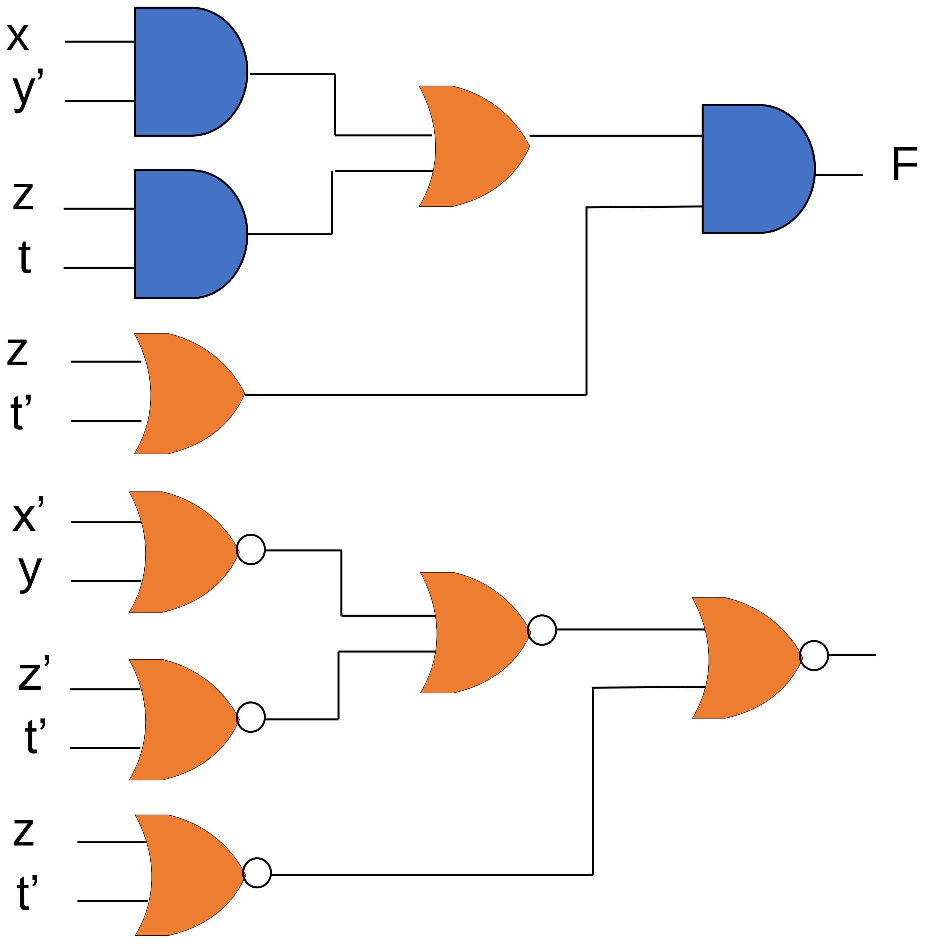




F = (x + y) (z + t) w

Example: Design with NOR Gates

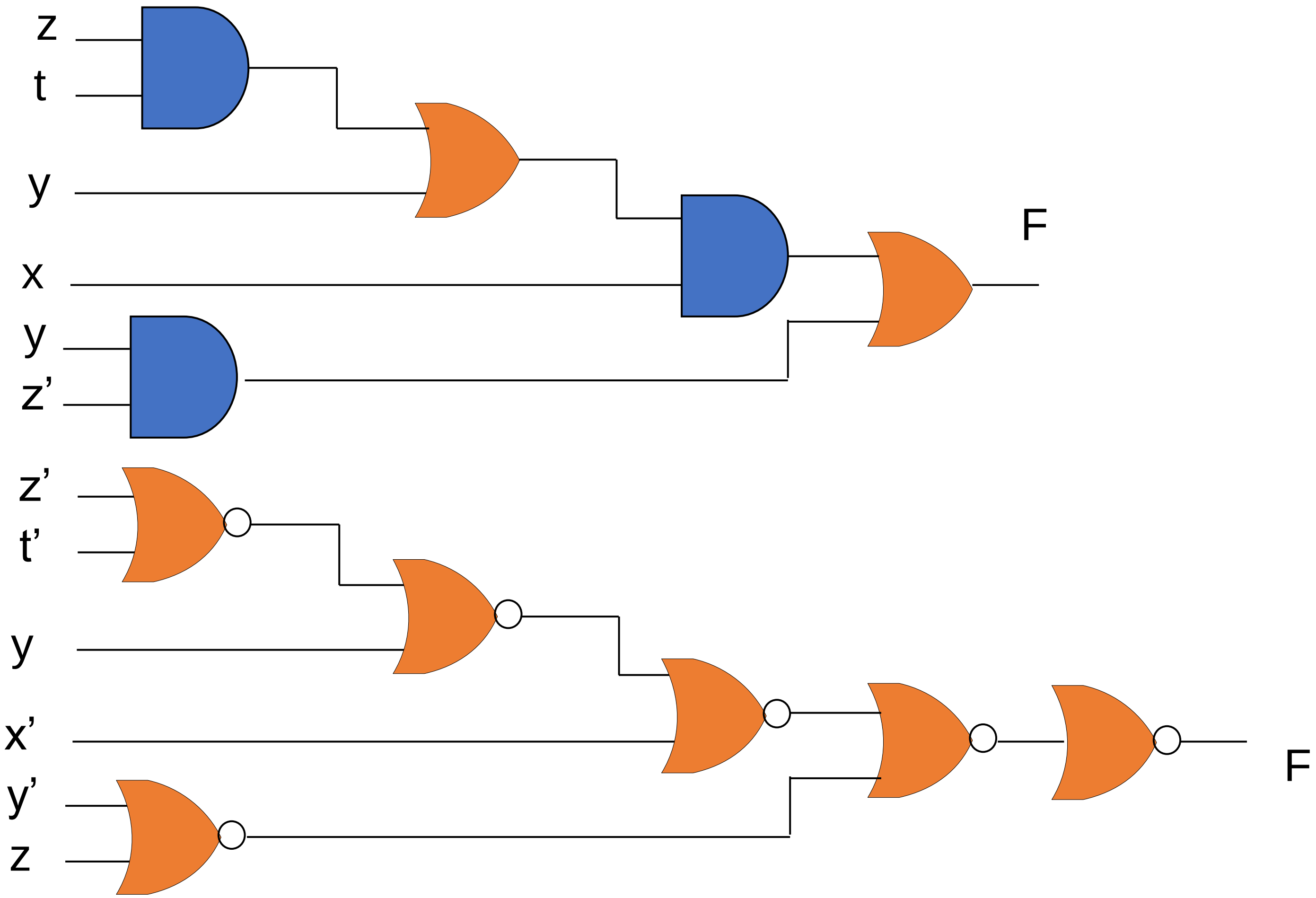
- F = (xy' + zt) (z + t')



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’



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

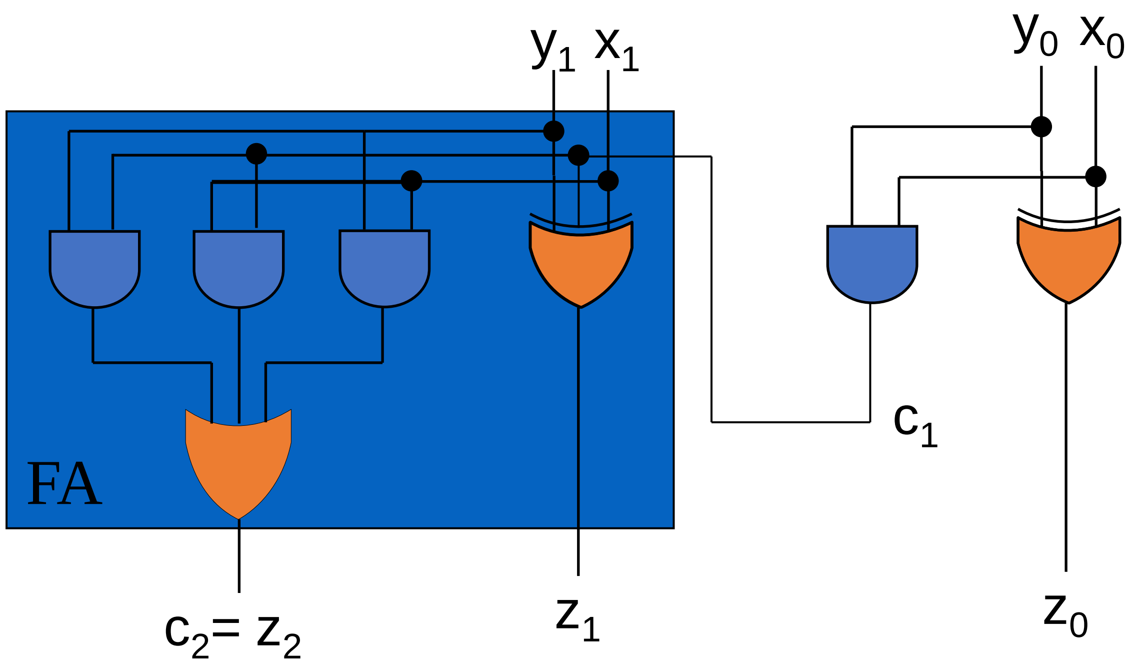
$z_1 = x_1 \oplus y_1 \oplus c_1$

$c_2 = x_1\, y_1 + x_1\, c_1 + y_1\, c_1$

$z_0 = x_0 \oplus y_0$

$c_1 = x_0\, y_0$

$z_2 = c_2$



Comparator Circuit with NAND gates

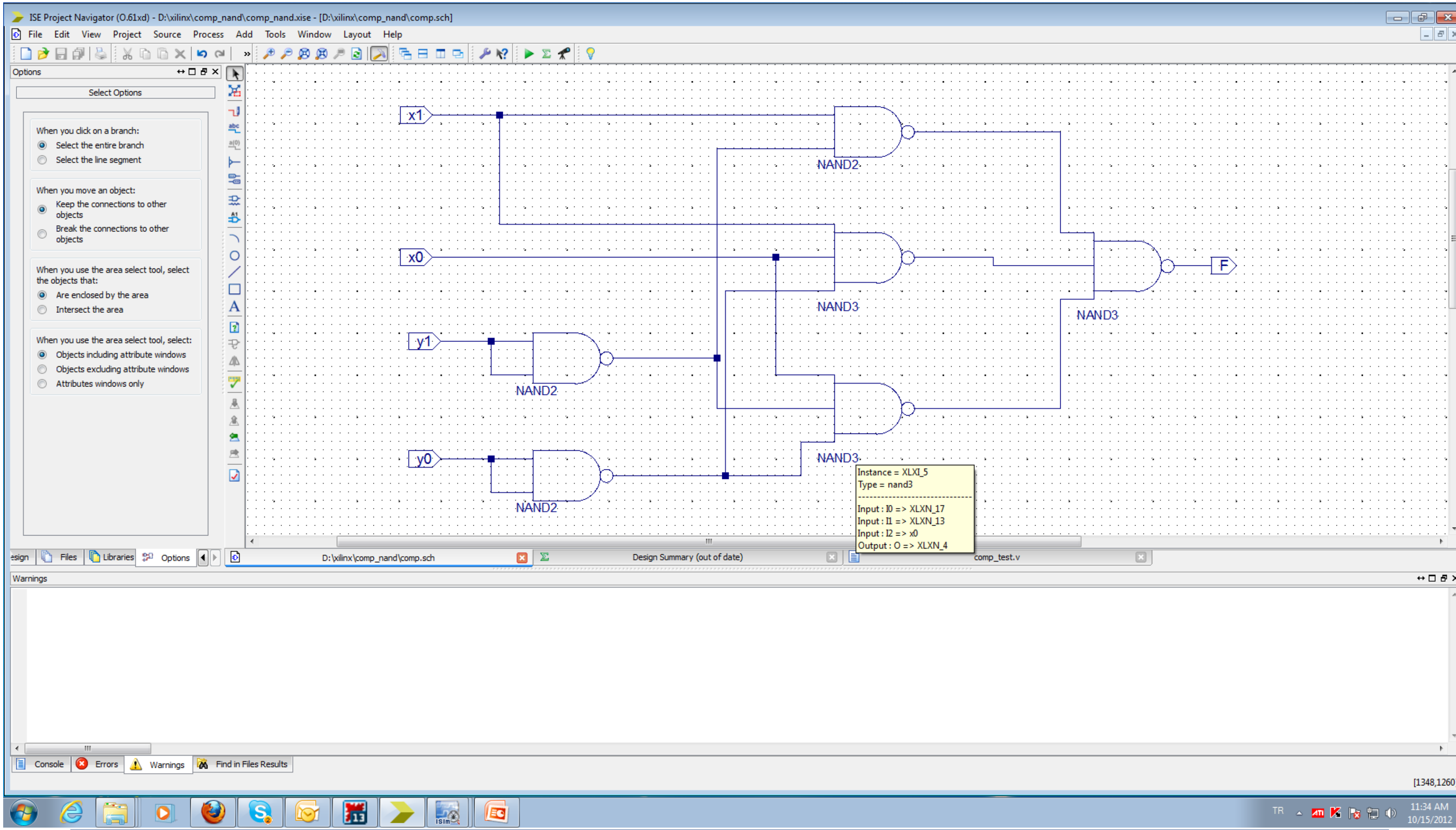
- F(X>Y)
  - $X = (x_1\ x_0)$ and $Y = (y_1\ y_0)$

x_1x_0 \ y_1y_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

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



Comparator Circuit - Simulation