Topic 4

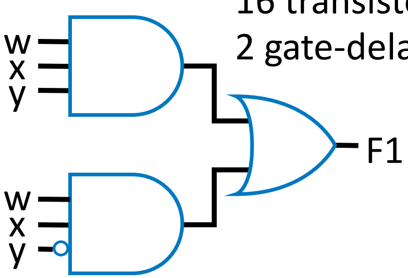
Logic Optimization

Simplification and Optimization

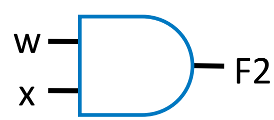
- **How can we build better circuits?**
- **Let's consider two important design criteria**
  - ***Delay*** **–** the time from input change to correct stable output response
  - ***Size*** **–** the number of transistors
  - For quick estimation, assume
    - Every gate has delay of "1 gate-delay"
    - Every gate *input* requires 2 transistors
    - Ignore inverters for simplicity

Transforming F1 to F2 represents an ***optimization***: Better in all criteria of interest

16 transistors
2 gate-delays



4 transistors
1 gate-delay

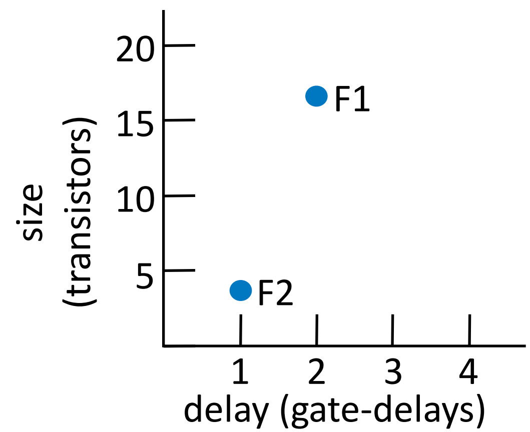


F1 = wxy + wxy'

F2 = wx

= wx(y+y') = wx

(a) (b)



(c)

Logic Optimization

- **Two-level size optimization using algebraic methods**
  - Goal: circuit with only two levels (AND-OR network), with minimum transistors
  - Sum-of-products yields two levels
    - F = abc + abc' is sum-of-products
    - G = w(xy + z) is not

Example

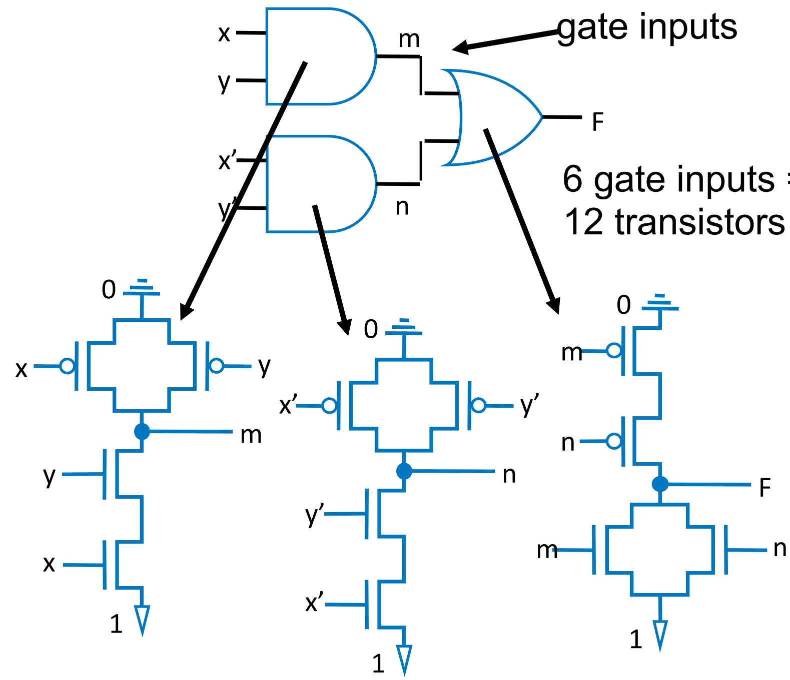
F = xyz + xyz' + x'y'z' + x'y'z

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

F = xy*1 + x'y'*1

F = xy + x'y'

4 literals + 2 terms = 6 gate inputs

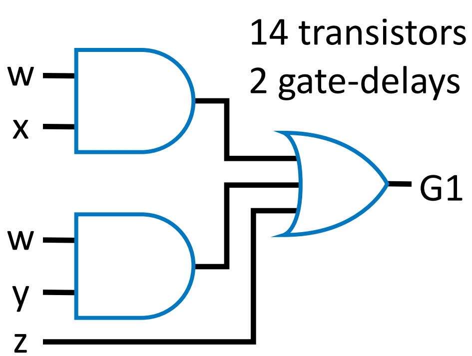


6 gate inputs = 12 transistors

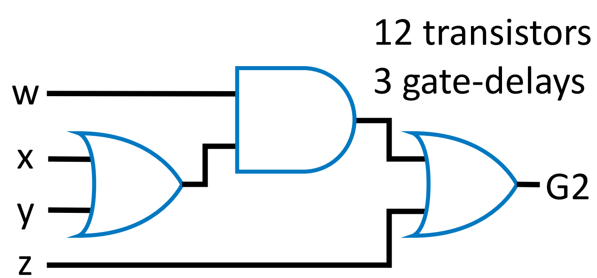
Logic Optimization

- **Multi-level optimization**
  - Improves some, but worsens other

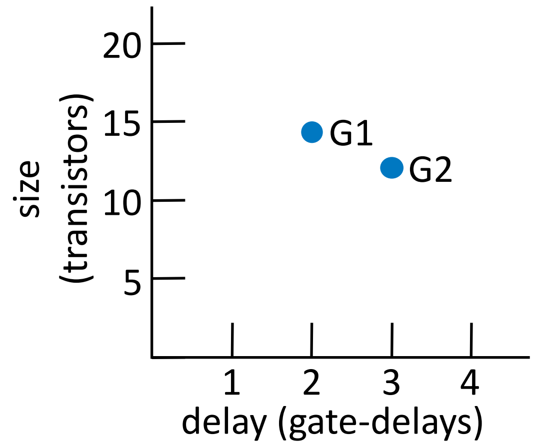
Transforming G1 to G2 represents a ***tradeoff***: Some criteria better, others worse



G1 = wx + wy + z

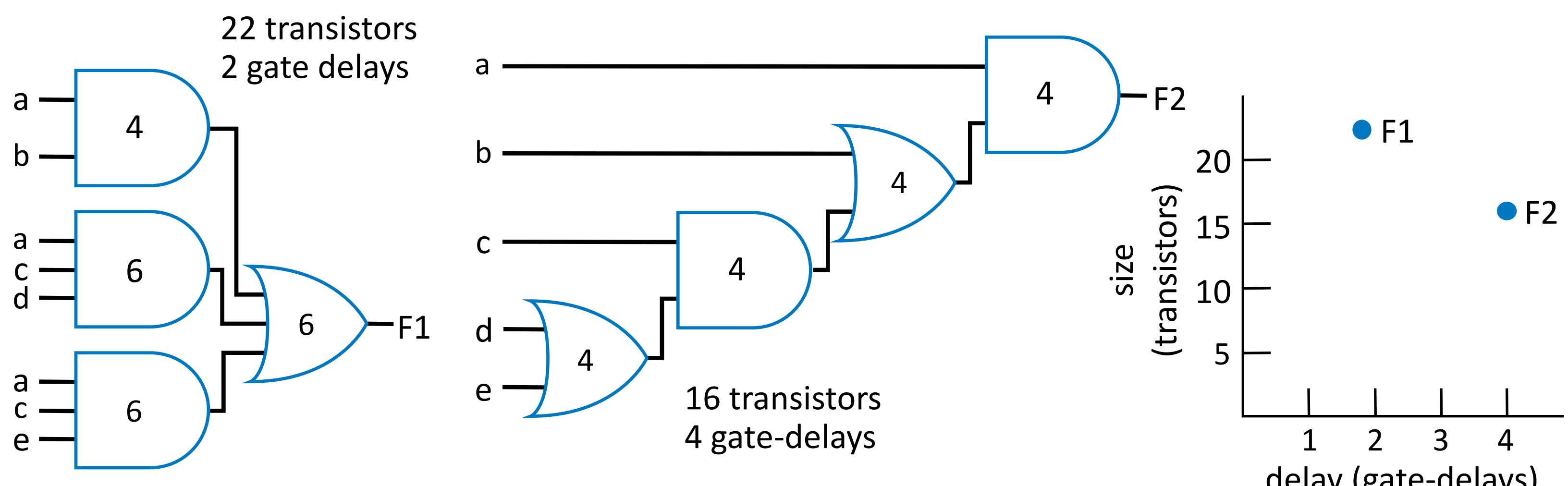


G2 = w(x+y) + z



Performance/Size Tradeoffs

- **Delay & Size tradeoff**
  - We don't always need the speed of two level logic
  - Multiple levels may yield fewer gates
  - Example
    - F1 = ab + acd + ace → F2 = ab + ac(d + e) = a(b + c(d + e))
    - General technique: Factor out literals



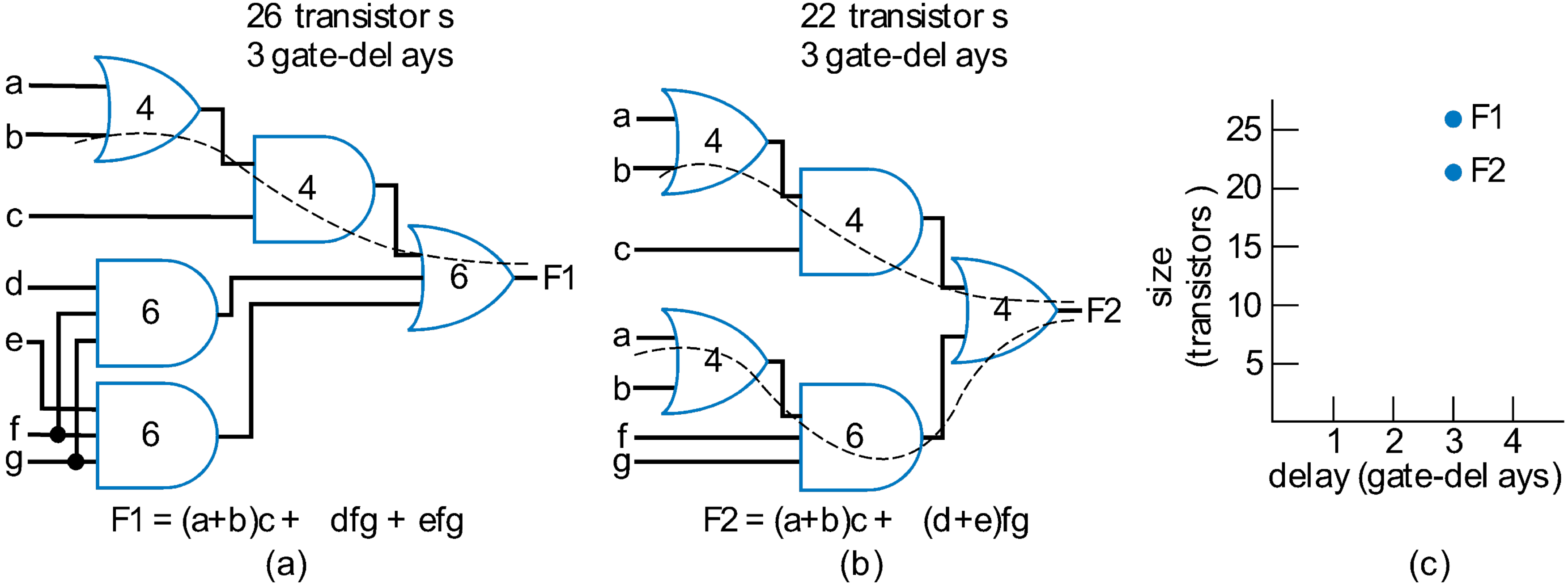
F1 = ab + acd + ace
(a)

F2 = a(b+c(d+e))
(b)

(c)

Critical Path

- **Critical path: longest delay path from an input to output**
- **Optimization**
  - Reduce delay by shortening length of critical path
  - Reduce size by using multiple levels on non-critical paths
    - But may make non-critical path become critical path



Logic Optimization

- **Optimization using Boolean Algebra**
  – To obtain Boolean equations with fewer literals (optimization in size)
  – To obtain circuit with shorter delay
- **Optimization using other techniques**
  – Karnaugh-map (to achieve simplified two-level circuit)
  – Quine-McCluskey (not in this class)

Karnaugh Map (K-map) Technique

- **A graphical technique used to *simplify* a logic equation**
- **A way to show the *relationship* between the logic inputs and corresponding output**
  - Like truth table
- **Much cleaner and more *procedural* than algebraic simplification by theorems of Boolean algebra**
- **Theoretically, it can be used for any number of input variables,**
  - BUT is only practical for less than six, we will limit our discussion to logic equations with *five or less* variables

Building a K-map

- **K-map can be filled up directly from a truth table**
  - Each minterm corresponds to a cell in the K-map
- **K-map cells are labeled so that both horizontal and vertical movement differ only in one variable**

F (X \ Y)	**Y'**	**Y**
X'	m0	m1
X	m2	m3

F (X \ YZ)	**Y'Z'**	**Y'Z**	**YZ**	**YZ'**
X'	m0	m1	m3	m2
X	m4	m5	m7	m6

F (WX \ YZ)	**Y'Z'**	**Y'Z**	**YZ**	**YZ'**
W'X'	m0	m1	m3	m2
W'X	m4	m5	m7	m6
WX	m12	m13	m15	m14
WX'	m8	m9	m11	m10

- **Since the adjacent cells differ in only one variable, they can be grouped to create simpler terms in the sum-of-product expression.**

Two-Variable K-map

- **There are four minterms – 2 by 2 square map**

Show the variables along which the labels varying vertically

Show the variables along which the labels varying horizontally

X	Y	F		
0	0	1	m0	X’Y’
0	1	0	m1	X’Y
1	0	1	m2	XY’
1	1	1	m3	XY

F, X \ Y

	Y’	Y
X’	1 (m0)	0 (m1)
X	1 (m2)	1 (m3)

Three-Variable K-map

- **There are $2^3 = 8$ minterms – 2 by 4 rectangular map**

X	Y	Z	F		
0	0	0	1	m0	X'Y'Z'
0	0	1	0	m1	X'Y'Z
0	1	0	1	m2	X'YZ'
0	1	1	1	m3	X'YZ
1	0	0	0	m4	XY'Z'
1	0	1	0	m5	XY'Z
1	1	0	0	m6	XYZ'
1	1	1	1	m7	XYZ

Show the variables along which the labels varying vertically

Show the variables along which the labels varying horizontally

X \ YZ	Y'Z'	Y'Z	YZ	YZ'
X'	1 (m0)	0 (m1)	1 (m3)	1 (m2)
X	0 (m4)	0 (m5)	1 (m7)	0 (m6)

Four-Variable K-map

- **There are 2^4=16 minterms – 4 by 4 square map**

W	X	Y	Z	F		
0	0	0	0	1	m0	W'X'Y'Z'
0	0	0	1	0	m1	W'X'Y'Z
0	0	1	0	1	m2	W'X'YZ'
0	0	1	1	1	m3	W'X'YZ
0	1	0	0	0	m4	W'XY'Z'
0	1	0	1	0	m5	W'XY'Z
0	1	1	0	0	m6	W'XYZ'
0	1	1	1	1	m7	W'XYZ
1	0	0	0	1	M8	WX'Y'Z'
1	0	0	1	1	m9	WX'Y'Z
1	0	1	0	0	M10	WX'YZ'
1	0	1	1	0	m11	WX'YZ
1	1	0	0	1	m12	WXY'Z'
1	1	0	1	0	m13	WXY'Z
1	1	1	0	0	m14	WXYZ'
1	1	1	1	1	m15	WXYZ

Show the variables along which the labels varying vertically or horizontally

WX \ YZ	Y'Z'	Y'Z	YZ	YZ'
W'X'	1 m0	0 m1	1 m3	1 m2
W'X	0 m4	0 m5	1 m7	0 m6
WX	1 m12	0 m13	1 m15	0 m14
WX'	1 m8	1 m9	0 m11	0 m10

Label the Rows and Columns by 0 and 1

F — X \ Y	Y' 0	Y 1
X' 0	1 m0	0 m1
X 1	1 m2	1 m3

F — X \ YZ	Y'Z' 00	Y'Z 01	YZ 11	YZ' 10
X' 0	1 m0	0 m1	1 m3	1 m2
X 1	0 m4	0 m5	1 m7	0 m6

F — WX \ YZ	Y'Z' 00	Y'Z 01	YZ 11	YZ' 10
W'X' 00	1 m0	0 m1	1 m3	1 m2
W'X 01	0 m4	0 m5	1 m7	0 m6
WX 11	1 m12	0 m13	1 m15	0 m14
WX' 10	1 m8	1 m9	0 m11	0 m10

0 represents the primed form
1 represents the unprimed form

Simplify – Grouping and Canceling

- **Group is in shape of rectangle or square**
- **Group the adjacent 1's until all the 1's are grouped**

F: X \ Y	0	1
0	1	0
1	0	0

X'Y'

No adjacent 1's, the minterm cannot be further simplified:
F = X'Y'

both 0 and 1 for Y in the same group

F: X \ Y	0	1
0	1	1
1	0	0

X'

Two adjacent 1's:
F = X'Y' + X'Y
= X'(Y' + Y)
= X' • 1
= X'

If both primed and unprimed forms of a letter appear in the same group, the letter can be canceled

A group corresponds to a Sum-of-Minterm expression

Grouping and Canceling

- No zeros in the group

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

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

- The number of 1's in the group should be 2^N, $N = 0, 1, 2, \ldots$

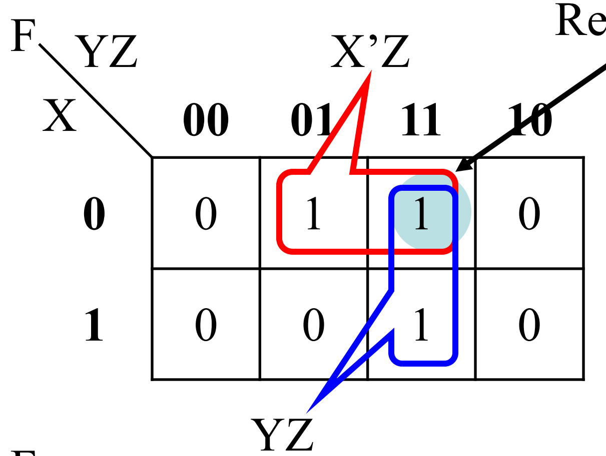
F: X \ YZ	00	01	11	10
0	1	1	1	0
1	0	0	0	0

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

Grouping and Canceling

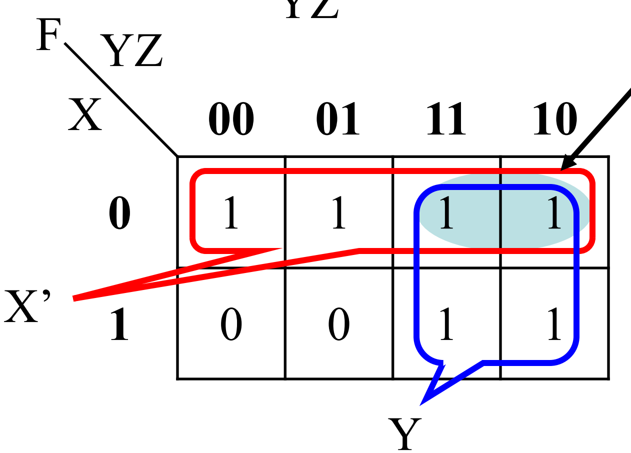
- **Group as many adjacent 1's as possible**

Redundant term



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

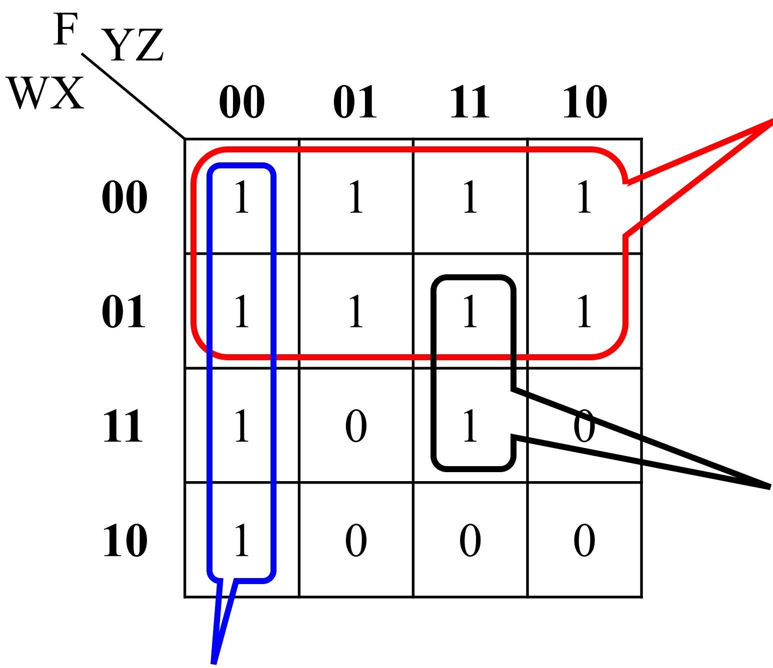
Redundant terms



$$\mathbf{F = X'Y'Z' + X'Y'Z + X'YZ + X'YZ' + XYZ + XYZ'}$$
$$\mathbf{= (X'Y'Z' + X'Y'Z + X'YZ + X'YZ') +}$$
$$\mathbf{(X'YZ + X'YZ' + XYZ + XYZ')}$$
$$\mathbf{= (X'Y' + X'Y) + (X'Y + XY)}$$
$$\mathbf{= X' + Y}$$

Grouping and Canceling

- **Group as many adjacent 1's as possible**



W', because
horizontally, both Y and Y' appear, Y cancels
both Z and Z' are included, Z cancels;
vertically, both X and X' appear, X cancels.
That leaves the 0 for W – primed W.

XYZ, because
only W and W' appear, W cancels.
The remaining term 111 implies XYZ

Y'Z', because
both W and W' appear, and both X and X' appear,
So W and X cancel.
That leaves the 00 for YZ – primed Y and primed Z.

Grouping and Canceling

- **Edges wrap around**

F (X \ YZ)	00	01	11	10
0	1	0	0	1
1	1	0	0	1

F = Z’

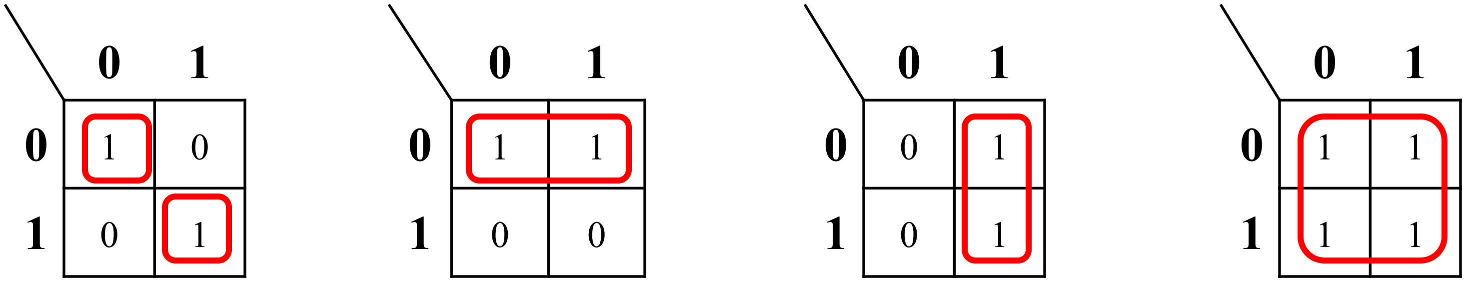
F (WX \ YZ)	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	0	0	0
10	1	0	0	1

F = W’Z’ + X’Z’

Grouping and Canceling

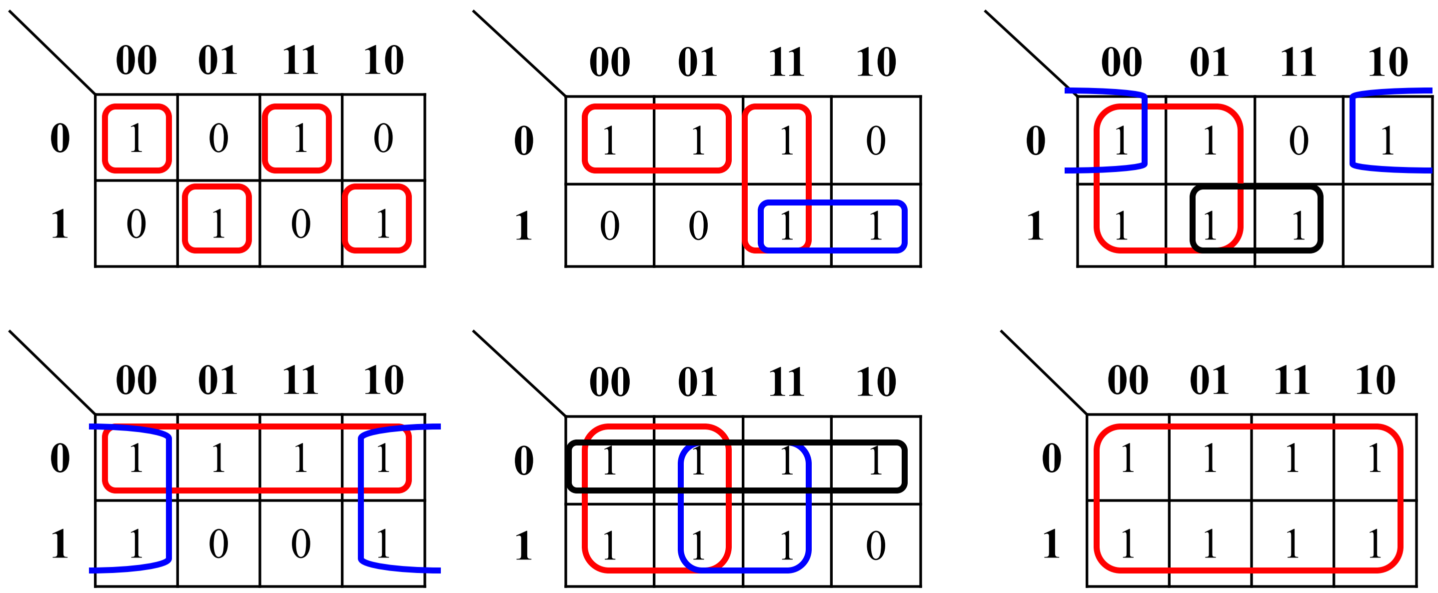
- **Summary**
  - Group is in shape of rectangle or square
  - Group the adjacent 1's until all the 1's are grouped
  - The number of 1's in the group should be 2^N, N = 0, 1, 2, …
  - Collect as many 1's as possible in the same group
  - No zeros in the group
  - Edges wrap around
  - If both primed and unprimed forms of a letter appear in a same group, the letter cancels
  - The simplified result will be a sum-of-product form; the number of the product terms is decided by the number of the groups

Group Patterns of 2-Variable Map



- **Summary**
  - A group of one cell represents a minterm, giving a term of two literals
  - A group of two cells represents a term of one literal
  - A group of all the four cells gives a logic 1

Group Patterns of 3-Variable Map



Group Patterns of 3-Variable Map

- **Summary**
  - A group of one cell represents a minterm, giving a term of three literals
  - A group of two cells represents a term of two literals
  - A group of four cells represents a term of one literal
  - A group of all the eight cells gives a logic 1

Group Patterns of 4-Variable Map

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

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

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

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

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

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

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

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

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

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

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

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

Group Patterns of 4-Variable Map

- **Summary**
  - A group of one cell represents a minterm, giving a term of four literals
  - A group of two cells represents a term of three literals
  - A group of four cells represents a term of two literals
  - A group of eight cells represents a term of one literal
  - A group of all the sixteen cells gives a logic 1
- **The more the number of cells in one group, the less the number of literals that group represents, hence cheaper to implement using logic gates**

Don’t Care Conditions

- The possible input combinations might not be all valid or not for consideration for a device
  - Hence we don’t care what the corresponding outputs are under those conditions
  - Called **don’t care** conditions
  - Mark the corresponding outputs by **X**

A	B	C	D	F
0	*0*	*0*	*0*	*X*
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	*0*	*0*	*1*	*X*
1	0	1	0	1
1	0	1	1	1
1	*1*	*0*	*0*	*X*
1	*1*	*0*	*1*	*X*
1	*1*	*1*	*0*	*X*
1	1	1	1	1

Don’t Care Conditions

- By employing **don’t care** conditions, logic equations can be further simplified
- Example:

  F (A, B, C, D) = Σm(2, 3, 5, 6, 7, 10, 11, 15)
  + Σd(0, 9, 12, 13, 14)
  - Fill out the K-map with 1’s and X’s
  - Each “X” can be either 0 or 1 depending upon the needs of simplification
  - Not all X’s have to be considered
- Apply the same grouping and canceling rules before using ‘X’: F = A’C + A’BD + B’C + CD

  after: F = C + BD (Essential PI?)

F: AB \ CD	00	01	11	10
00	X $_{0}$	0 $_{1}$	1 $_{3}$	1 $_{2}$
01	0 $_{4}$	1 $_{5}$	1 $_{7}$	1 $_{6}$
11	X $_{12}$	**1** $_{13}$	1 $_{15}$	**1** $_{14}$
10	0 $_{8}$	X $_{9}$	1 $_{11}$	1 $_{10}$

Prime Implicants

- Implicant: is a product term
- A **prime implicant (PI)** is a group that cannot be entirely contained by another implicant

F

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

Prime implicant

Not prime implicants

F

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

Essential Prime Implicants

- A **prime implicant (PI)** is **essential** if a a cell is covered **ONLY** by that PI
- The **essential PIs** can be found by
  – looking at each cell marked as 1 and not covered by any other essential PI
  – and checking the number of PIs that cover it

F

WX \ YZ	**00**	**01**	**11**	**10**
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

Essential Prime Implicants

- **Check each cell marked as 1, only if it has not been covered by an essential PI**

F WX \ YZ	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

Essential PI: X’Z’

F WX \ YZ	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

No essential PIs found

Essential Prime Implicants

F: WX \ YZ	00	01	11	10
00	1	0	1	1
01	0	***1***	1	0
11	0	1	1	0
10	1	1	1	1

Essential PI: XZ

F: WX \ YZ	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	***1***	1	1

No essential PIs found

Essential Prime Implicants

F: WX \ YZ	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

No essential PIs found

Essential Prime Implicants

F (WX \ YZ)	00	01	11	10
00	1	0	**1**	1
01	0	1	1	0
11	0	1	1	0
10	1	**1**	**1**	1

Essential PIs

F (WX \ YZ)	00	01	11	10
00	1	0	**1**	1
01	0	1	1	0
11	0	1	1	0
10	1	**1**	**1**	1

Non essential PIs

- **Essential PIs have to be used in the simplified equation**
- **Cells not covered by essential PIs can be represented by any PIs covering them**

F = X'Z' + XZ + WX'(or WZ) + X'Y(or YZ)

Product-of-Sum Simplification – An Alternate Method

- **Redraw the K-map for F' by switching 1's and 0's**

W	X	Y	Z	F	F'
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	0	1

F: WX \ YZ	**00**	**01**	**11**	**10**
00	1	1	1	1
01	0	0	1	0
11	1	0	0	0
10	1	1	0	0

F': WX \ YZ	**00**	**01**	**11**	**10**
00	0	0	0	0
01	1	1	0	1
11	0	1	1	1
10	0	0	1	1

Product-of-Sum Simplification – An Alternate Method

- **Two forms of the same truth table**

F

WX \ YZ	**00**	**01**	**11**	**10**
00	1	1	1	1
01	0	0	1	0
11	1	0	0	0
10	1	1	0	0

Sum-of-Product form:

$F = \underline{W'X'} + \underline{X'Y'} + \underline{WY'Z'} + \underline{W'YZ}$

F'

WX \ YZ	**00**	**01**	**11**	**10**
00	0	0	0	0
01	1	1	0	1
11	0	1	1	1
10	0	0	1	1

$F' = \underline{WY} + XY'Z + W'XZ'$

Product-of-Sum form:

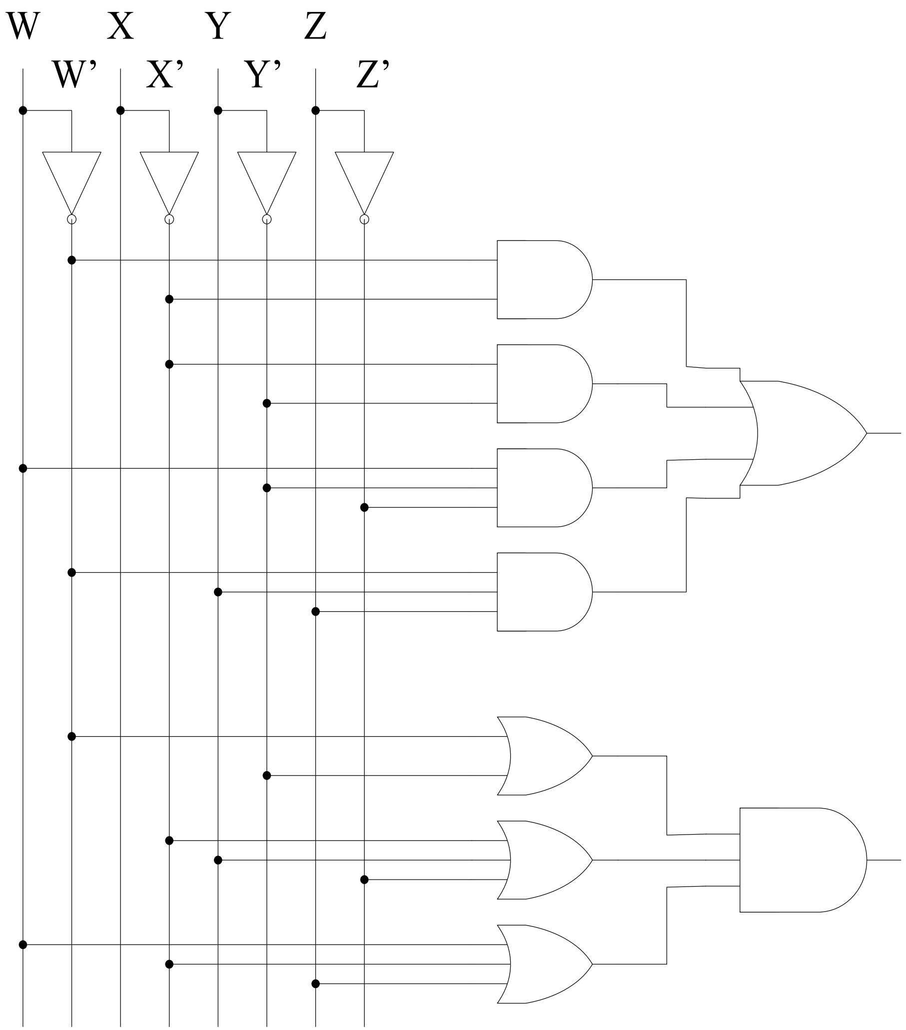
$F = (F')'$ **(DeMorgan's Law)**

$= (WY + XY'Z + W'XZ')'$

$= (WY)'\,(XY'Z)'\,(W'XZ')'$

$= (W'+Y')(X'+Y+Z')(W+X'+Z)$

Product-of-Sum Simplification – An Alternate Method



F = W'X' + X'Y' + WY'Z' + W'YZ

F = (W'+Y')(X'+Y+Z')(W+X'+Z)

Simpler

Simplify Any Standard Sum-of-Product Form

Method 1: fill out the table directly

- **F = A'C + A'BD + AB'C + BCD**

A	B	C	D	F	
0	0	0	0	0	**m0**
0	0	0	1	0	**m1**
0	0	1	0	1	**m2**
0	0	1	1	1	**m3**
0	1	0	0	0	**m4**
0	1	0	1	1	**m5**
0	1	1	0	1	**m6**
0	1	1	1	1	**m7**
1	0	0	0	0	**m8**
1	0	0	1	0	**m9**
1	0	1	0	1	**m10**
1	0	1	1	1	**m11**
1	1	0	0	0	**m12**
1	1	0	1	0	**m13**
1	1	1	0	0	**m14**
1	1	1	1	1	**m15**

F: AB \ CD	**00**	**01**	**11**	**10**
00	0	0	1	1
01	0	1	1	1
11	0	0	1	0
10	0	0	1	1

Simplify Any Standard Sum-of-Product Form

- **F = A'C + A'BD + AB'C + BCD**
  - **Method 2: convert any form of equation to sum-of-minterm**
    - AND with sum of the primed and unprimed forms of the missing literal, one at a time until all the missing literals are considered
    - Remove the duplicated minterms

F = A'C + A'BD + AB'C + BCD

= A'C (B+B') + A'BD (C+C') + AB'C (D+D') + BCD (A+A')

= A'BC + A'B'C + A'BCD + A'BC'D + AB'CD + AB'CD' + ABCD + A'BCD

= A'BC (D+D') + A'B'C (D+D') + A'BCD + A'BC'D + AB'CD + AB'CD'+ABCD+A'BCD

= A'BCD + A'BCD' + A'B'CD + A'B'CD' + A'BCD + A'BC'D + AB'CD + AB'CD'+ABCD+A'BCD

= Σ m(7, 6, 3, 2, 7, 5, 11, 10, 15, 7)

= Σ m(2, 3, 5, 6, 7, 10, 11, 15)

Simplify Any Standard Sum-of-Product Form

- **F = A'C + A'BD + AB'C + BCD**

A	B	C	D	F	
0	0	0	0	0	**m0**
0	0	0	1	0	**m1**
0	0	1	0	1	***m2***
0	0	1	1	1	***m3***
0	1	0	0	0	**m4**
0	1	0	1	1	***m5***
0	1	1	0	1	***m6***
0	1	1	1	1	***m7***
1	0	0	0	0	**m8**
1	0	0	1	0	**m9**
1	0	1	0	1	***m10***
1	0	1	1	1	***m11***
1	1	0	0	0	**m12**
1	1	0	1	0	**m13**
1	1	1	0	0	**m14**
1	1	1	1	1	***m15***

F: AB \ CD	**00**	**01**	**11**	**10**
00	0	0	1	1
01	0	1	1	1
11	0	0	1	0
10	0	0	1	1

After simplification:

F = A'C + A'BD + B'C + CD

(Essential PI?)

Dealing with Five Variables

E	A	B	C	D	F
0	0	0	0	0	X
0	0	0	0	1	0
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	X
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	X
0	1	1	0	1	X
0	1	1	1	0	X
0	1	1	1	1	1

E	A	B	C	D	F
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	X
1	0	1	0	0	X
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	X
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	0	1	1
1	1	1	1	0	X
1	1	1	1	1	0

E'

F: AB \ CD	00	01	11	10
00	X_{0}	0_{1}	1_{3}	1_{2}
01	0_{4}	1_{5}	1_{7}	1_{6}
11	X_{12}	X_{13}	1_{15}	X_{14}
10	0_{8}	X_{9}	1_{11}	1_{10}

E

F: AB \ CD	00	01	11	10
00	1_{0}	1_{1}	X_{3}	0_{2}
01	X_{4}	1_{5}	0_{7}	0_{6}
11	1_{12}	1_{13}	0_{15}	X_{14}
10	1_{8}	X_{9}	1_{11}	0_{10}

F = E'(C+BD) + E(C'+B'D)

Power Optimization

- **Power is another important design criteria**
  - Measured in Watts (energy/second)
    - Rate at which energy is consumed
- **Increasingly important as more transistors on a chip**
  - Power not scaling down at same rate as size
    - cooling is difficult
  - CMOS technology: Switching a wire from 0 to 1 consumes power (known as ***dynamic power***)
    - $P = k * CV^2 f$
      - k: constant; C: capacitance of wires; V: voltage; f: switching frequency
    - Power reduction methods
      - Reduce voltage: But slower, and there's a limit
      - What else?

Using Low-Power Gates on Non-Critical Paths

- **Another method: Use low-power gates**
  - Multiple versions of gates may exist
    - Fast/high-power, and slow/low-power, versions
  - Use slow/low-power gates on non-critical paths
    - Reduces power, without increasing delay