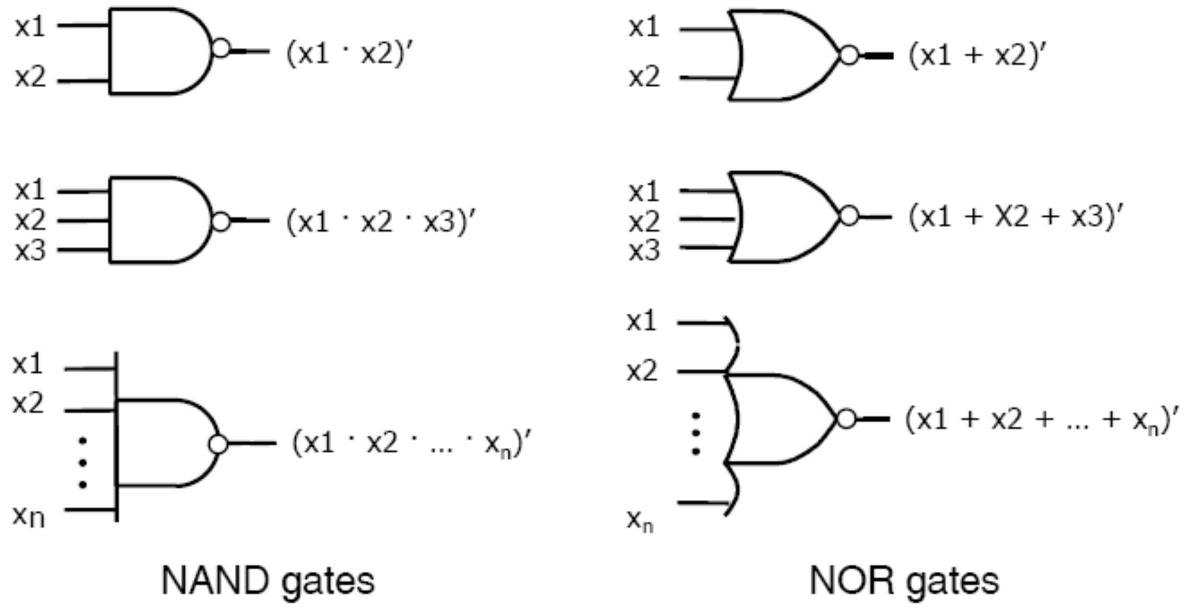
Digital Logic Design

Sung-Soo Lim

KESL
Kookmin Univ. Embedded System Lab

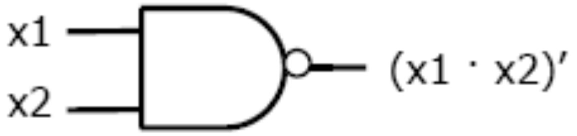
NAND and NOR Logic Networks

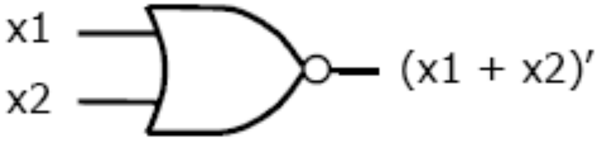
- Other basic logic gates in addition to AND, OR, NOT
  - NAND gate – obtained by complementing AND gate
  - NOR gate – obtained by complementing OR gate
    - Inversion bubble on output to represent the complement output signal



Why We Use NAND/NOR?

- NAND and NOR gates are popular because the underlying implementation is simpler





NAND

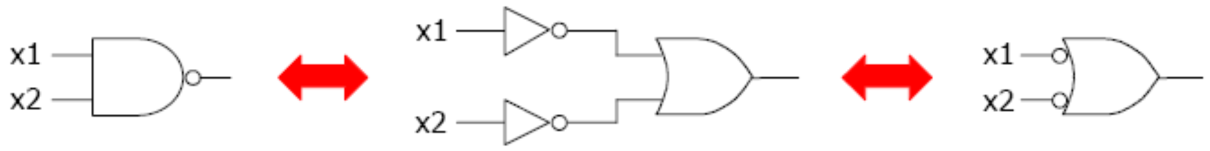
a	b	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR

a	b	F
0	0	1
0	1	0
1	0	0
1	1	0

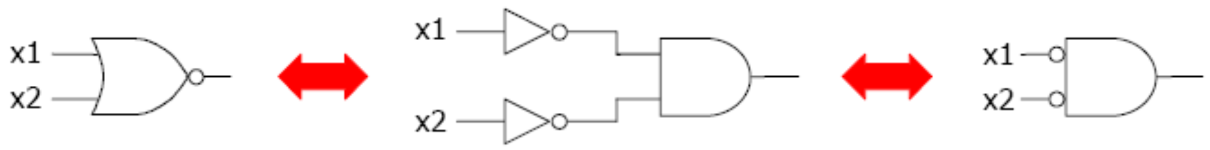
DeMorgan’s Theorem as Logic Circuits

- Can we use NAND and NOR gates to implement various logic circuits?
  - Let’s look at logic gate implementation of DeMorgan’s Theorem



15a. $(x1x2)' = x1' + x2'$

NAND of two variables the same as complementing variables first, then ORing them



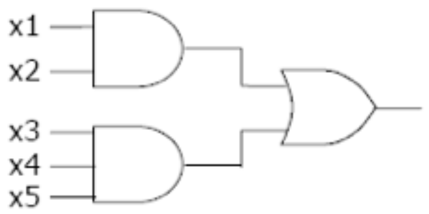
15b. $(x1 + x2)' = x1'x2'$

NOR of two variables the same as complementing variables first, then ANDing them

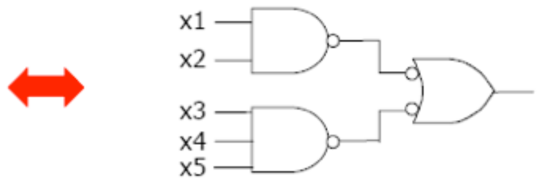
Far right circuits, NOT gates represented as inversion bubbles

DeMorgan’s Theorem as Logic Circuits

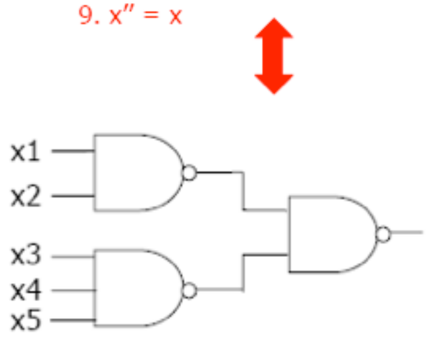
- Any function can be implemented in sum-of-products form
  - We can transform into a network using only NAND gates



general sum-of-products form

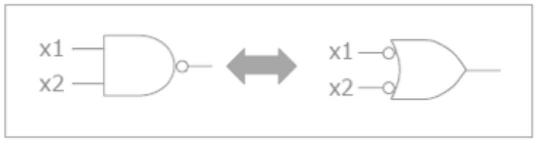


connections between AND and OR gates includes 2 INV gates –functionally equivalent 9. x″ = x



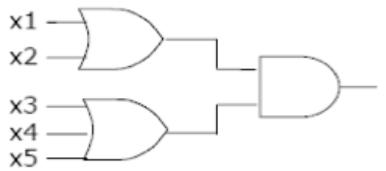
Network can be transformed into a network of NAND gates

Just showed this equivalency – DeMorgan’s Theorem

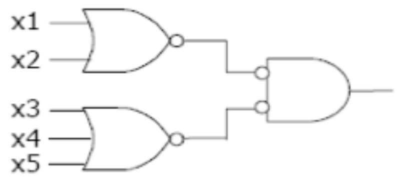


DeMorgan's Theorem as Logic Circuits

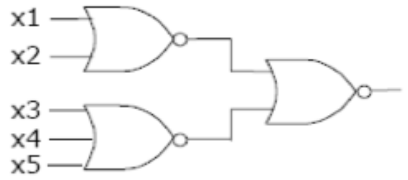
- Any function can be implemented in product-of-sums form
  - We can transform into a network using only NOR gates



general product-of-sums form

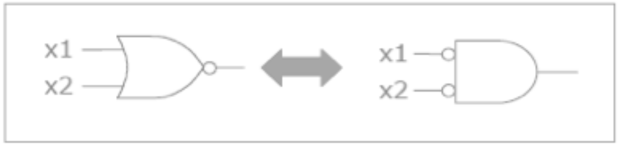


connections between OR and AND gates includes 2 INV gates –functionally equivalent 9. x'' = x



Network can be transformed into a network of NOR gates

Just showed this equivalency – DeMorgan's Theorem



Design Example

- Converting English Problem to Boolean Logic:
  - A fire sprinkler system should spray water if high heat is sensed and the system is set to enabled.
  - Identify and label variables:
    - h represents “high heat is sensed”
    - e represents “enabled”
    - F represents “spraying water”
- Write Boolean Equation expressing functionality described

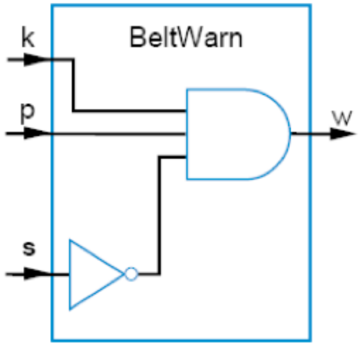
$$\boxed{F = h \text{ AND } e}$$

KESL
Kookmin Univ. Embedded Systems Lab

Seat Belt Warning Light System

- Circuit Description
  - Turn warning light on if driver in driver's seat, key inserted, seat belt not fastened
- Identify and label variables
  - s=1: seat belt fastened
  - k=1: key inserted
  - p=1: person in seat
  - w=1: warning light on
- Write Boolean Equation expressing functionality described
  - person in seat, and seat belt not fastened, and key inserted
- We can further implement as a circuit

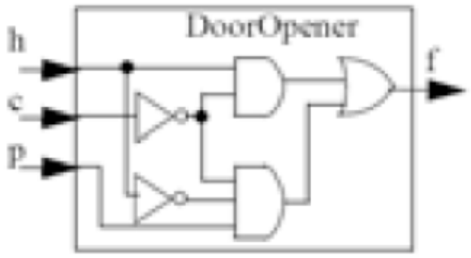
w = p AND NOT(s) AND k



Sliding Door

- Circuit Description
  - Design an automatic sliding door.
  - Open the door if the door is set to be manually held open
  - Open the door if the door is not set to be manually open, and a person is detected
  - However, in either case, we only open the door if the door is not set to stay closed
- Identify and label variables
  - p=1: person in front of door
  - h=1: held open manually
  - c=1: force door to stay closed
  - f=1: open sliding door
- Write Boolean Equation expressing functionality described
  - not forced close and manually held open, or not forced closed and not manually held open and person detected

$f = hc' + h'pc'$



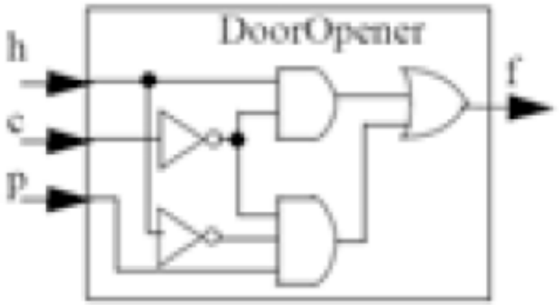
KESL
Kookmin Univ. Embedded Systems Lab

Simplification using Boolean Algebra

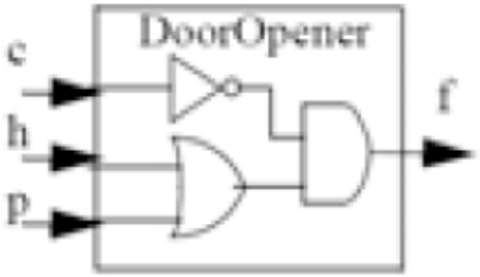
- We can further simplify circuit using Boolean Algebra

f = hc′ + h′pc′

f = c′h + c′h′p	(commutative property)
f = c′(h + h′p)	(distributive property)
f = c′((h + h′) * (h + p))	(distributive property)
	f = c′ (h + (h′ * p)) → ((h + h′) * (h+p))
f = c′((1)*(h + p))	(by the complement property)
f = c′(h + p)	(by the identity property)



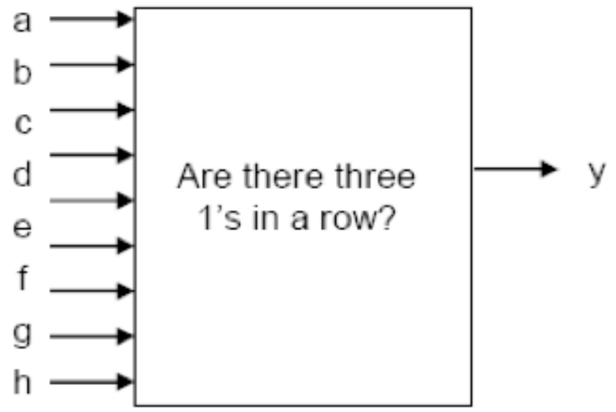
f = hc’ + h’pc’

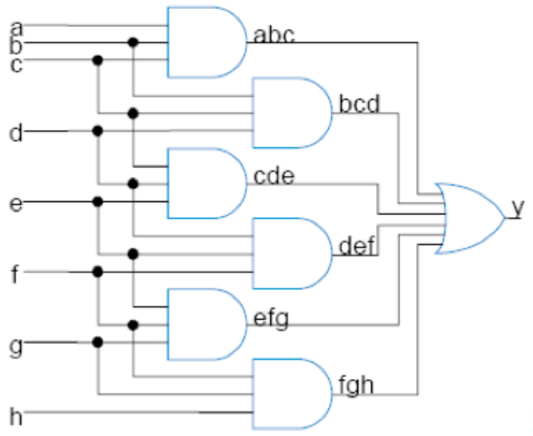


f = c’(h+p)

Three One's Detector

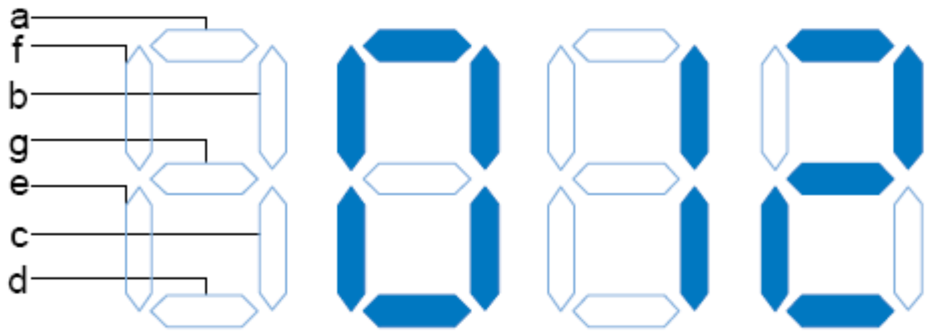
- Circuit Description
  - Detect three consecutive 1s in an 8-bit input: abcdefgh
  - Example: 00011101 yields 1, 10101011 yields 0, 11110000 yields 1
- Truth table or logic expression?
  - Truth table too big (28 = 256 entries)
- Logic expression
  - Create terms for each possible case of three consecutive 1s
    - abc, bcd, cde, def, efg, fgh
  - When any of these appear we want to output a 1
    - f = abc + bcd + cde + def + efg + fgh





Seven Segment Display

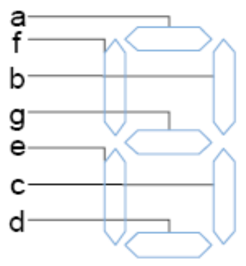
- Seven segment display
  - Each light segment controlled independently
  - Value of 1 indicates light segment illuminate
  - Configure to display various numbers and letters

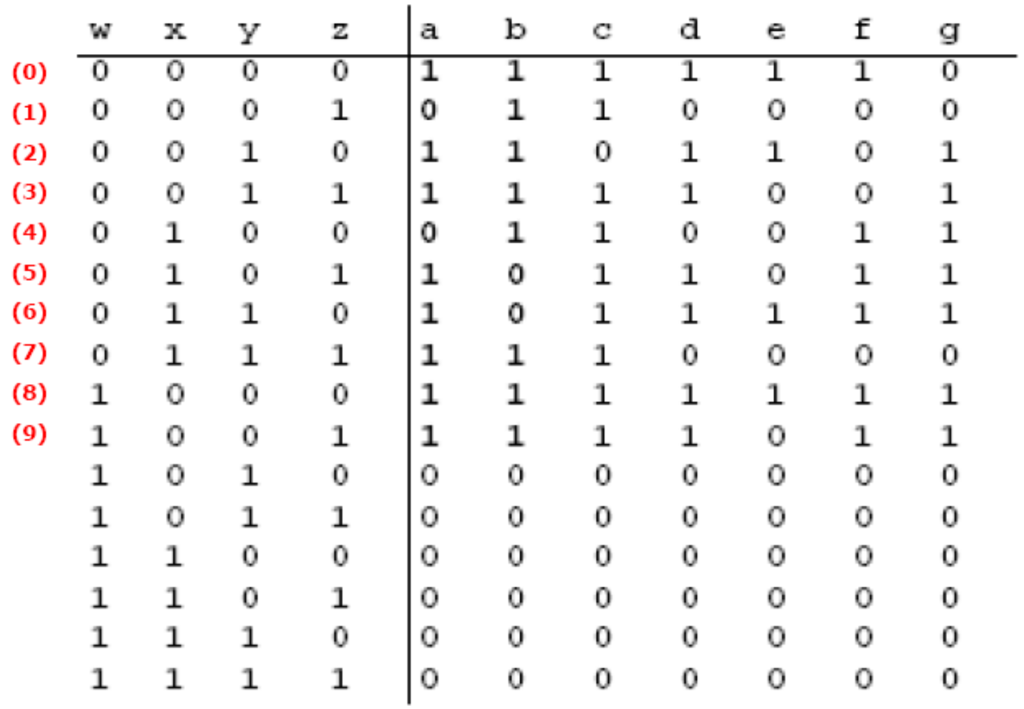


KESL
Kookmin Univ. Embedded Systems Lab

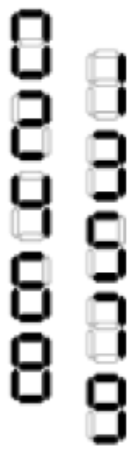
Seven Segment Display

- Create truth table to represent numbers 0 to 9, based on 4-bit encoding wxyz





	w	x	y	z	a	b	c	d	e	f	g
(0)	0	0	0	0	1	1	1	1	1	1	0
(1)	0	0	0	1	0	1	1	0	0	0	0
(2)	0	0	1	0	1	1	0	1	1	0	1
(3)	0	0	1	1	1	1	1	1	0	0	1
(4)	0	1	0	0	0	1	1	0	0	1	1
(5)	0	1	0	1	1	0	1	1	0	1	1
(6)	0	1	1	0	1	0	1	1	1	1	1
(7)	0	1	1	1	1	1	1	0	0	0	0
(8)	1	0	0	0	1	1	1	1	1	1	1
(9)	1	0	0	1	1	1	1	1	0	1	1
	1	0	1	0	0	0	0	0	0	0	0
	1	0	1	1	0	0	0	0	0	0	0
	1	1	0	0	0	0	0	0	0	0	0
	1	1	0	1	0	0	0	0	0	0	0
	1	1	1	0	0	0	0	0	0	0	0
	1	1	1	1	0	0	0	0	0	0	0

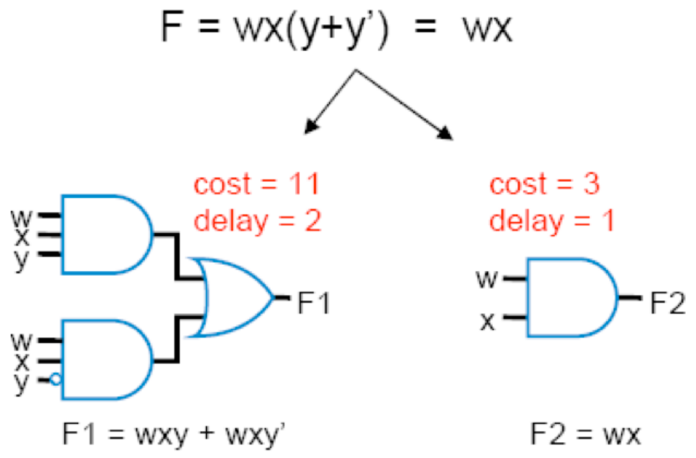


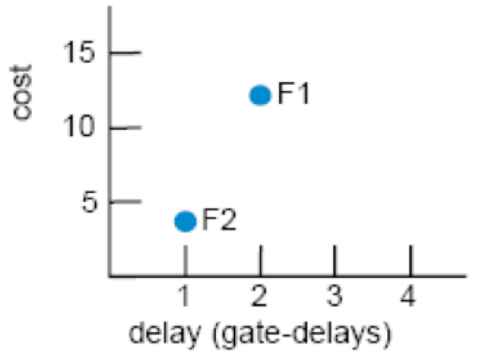
a = w'x'y'z' + w'x'yz' + w'x'yz + w'xy'z + w'xyz' + w'xyz + wx'y'z' + wx'y'z

b = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz + w'xy'z' + w'xyz + wx'y'z' + wx'y'z

Introduction to Optimization and Tradeoffs

- We now know how to build digital circuits
  - How can we build better circuits?
- Let’s consider two important design criteria
  - Delay – the time from inputs changing to new correct stable output
    - Every gate has delay of “1 gate-delay”
    - Ignore inverters
  - Size – the number of transistors
    - Every gate has cost = number of gates + number of gate inputs
    - Ignore inverters

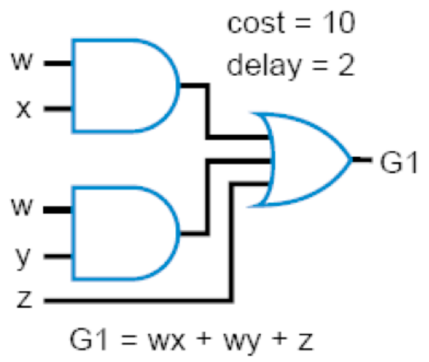


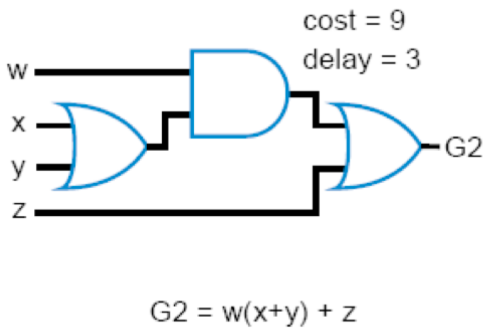


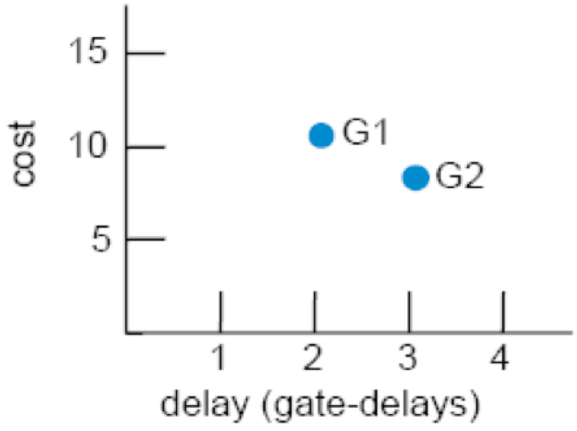
Transforming F1 to F2 represents an ***optimization***: Better in all criteria of interest

Introduction to Optimization and Tradeoffs

- Tradeoff
  - Improves some, but worsens other, criteria of interest





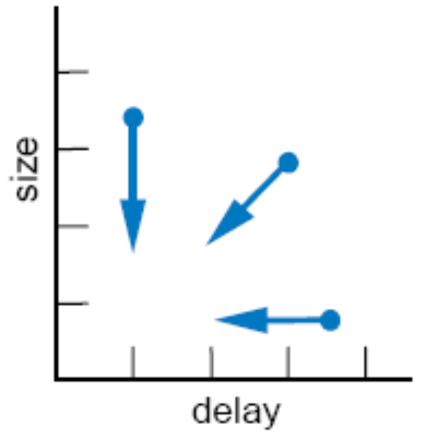


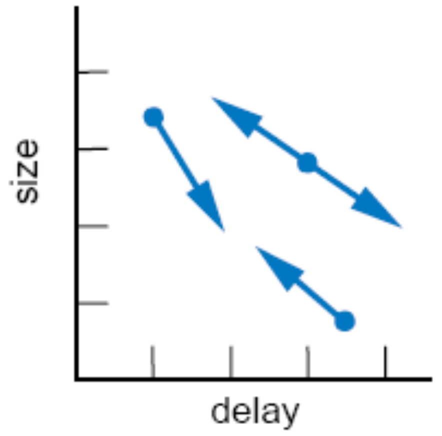
Transforming G1 to G2 represents a ***tradeoff***: Some criteria better, others worse.

KESL
Kookmin Univ. Embedded Systems Lab

Introduction to Optimization and Tradeoffs

- We obviously prefer optimizations, but often must accept tradeoffs
  - You can't build a car that is the most comfortable, and has the best fuel efficiency, and is the fastest – you have to give up something to gain other things.





Optimizations

All criteria of interest are improved (or at least kept the same)

Tradeoffs

Some criteria of interest are improved, while others are worsened

Optimization Through Algebraic Manipulation

- Algebraic manipulation
  - Multiply out to sum-of-products, then, apply following as much possible
    - ab + ab' = a(b + b') = a*1 = a
    - Combining terms to eliminate a variable
    - (Formally called the "Uniting theorem")
  - Duplicating a term sometimes helps
    - Note that doesn't change function
    - c + d = c + d + d = c + d + d + d + d ...
  - Sometimes after combining terms, can combine resulting terms

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

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

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

$F = xy + x'y'$

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

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

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

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

$G = xy'z' + xy'z + xyz + xyz'$

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

$G = xy' + xy$ *(now do again)*

$G = x(y'+y)$

$G = x$

K-Map (Karnaugh Map)

- Algebraic Manipulation
  - Which “rules” to use and when?
  - Easy to miss “seeing” possible opportunities to combine terms
- Karnaugh Maps (K-maps)
  - Graphical method to help us find opportunities to combine terms
  - Create map where adjacent minterms differ in one variable
  - Can clearly see opportunities to combine terms
    - – look for adjacent 1s

General K-map Method

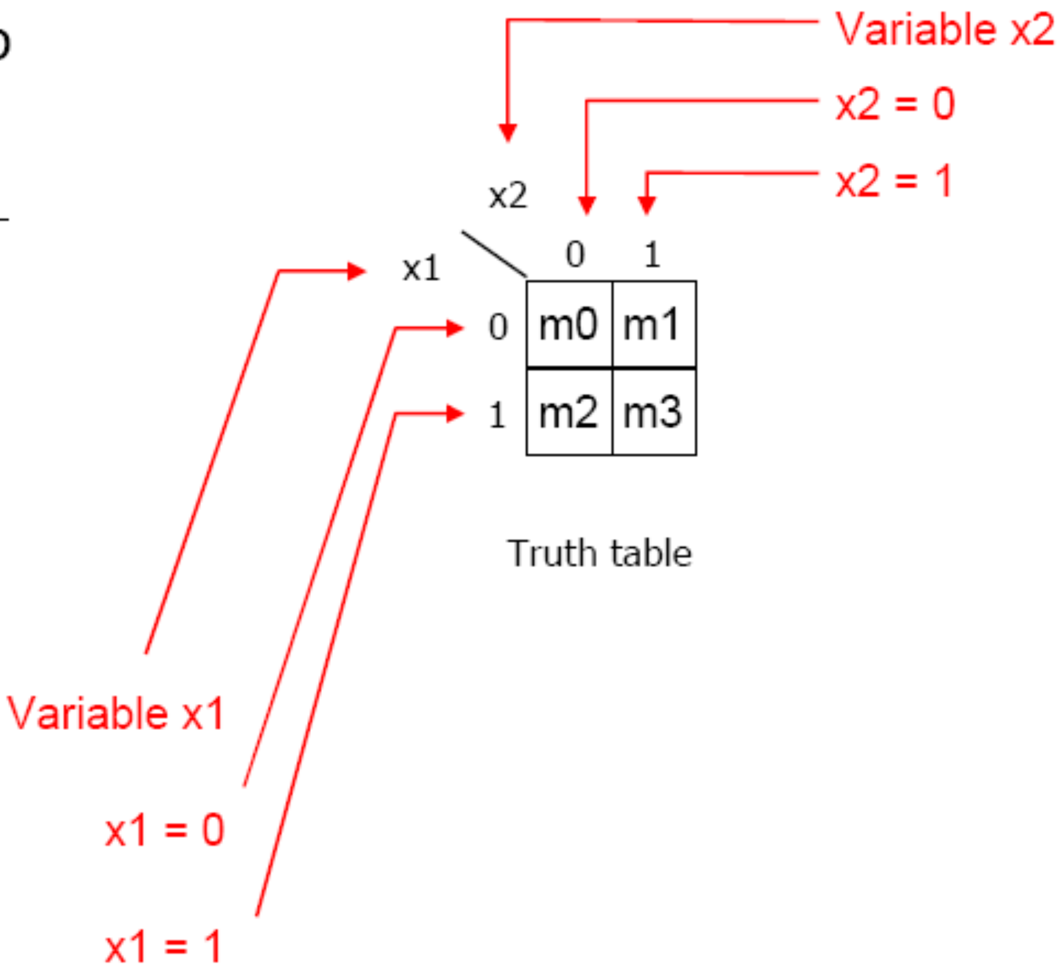
- General K-map method
  - Convert the function's equation into sum-of-products form (or truth table)
  - Place 1s in the appropriate K-map cells for each term
  - Cover all 1s by drawing the fewest largest circles, with every 1 included at least once; write the corresponding term for each circle
  - OR all the resulting terms to create the minimized function.

Generalized Two-Variable K-Map

Two-Variable Map

a	b	F
0	0	m0
0	1	m1
1	0	m2
1	1	m3

Truth table

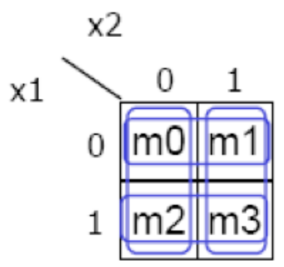


Generalized Two-Variable K-Map

Two-Variable Map

a	b	F
0	0	m0
0	1	m1
1	0	m2
1	1	m3

Truth table



Truth table

K-map graphically place minterms next to each other when they differ by one variable

m0 – x1'x2'

m1 – x1'x2

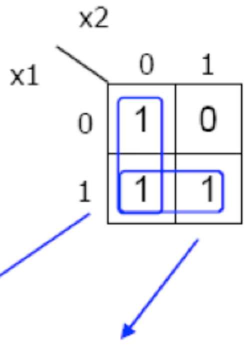
m2 – x1x2'

m3– x1x2

Two-Variable K-Map Example

- Fill in each cell with corresponding value of F
- Draw circles around adjacent 1's
  - Groups of 1, 2 or 4
- Circle indicates optimization opportunity
  - We can remove a variable
- To obtain function OR all product terms contained in circles
  - Make sure all 1's are in at least one circle

a	b	F
0	0	1
0	1	0
1	0	1
1	1	1

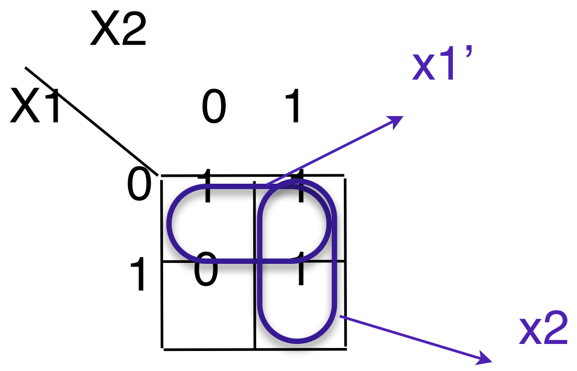


x1'x2' + x1x2'	x1x2' + x1x2
x2'(x1' + x1)	x1(x2' + x2)
x2'(1)	x1(1)
x2'	x1

F = x1 + x2'

Two-Variable K-Map Example

a	b	F
0	0	1
0	1	1
1	0	0
1	1	1



$F = x1' + x2$

Generalized Three-Variable K-Map

Three-Variable Map

a	b	c	F
0	0	0	m0
0	0	1	m1
0	1	0	m2
0	1	1	m3
1	0	0	m4
1	0	1	m5
1	1	0	m6
1	1	1	m7

Truth table

x1 \ x2x3	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m6	m7

Truth table

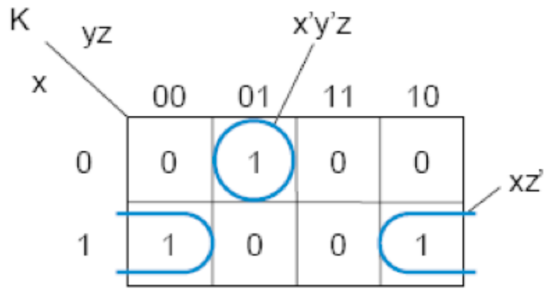
REMEMBER: K-map graphically place minterms next to each other when they differ by one variable

m1 cannot be placed next to m2 (a'b'c, a'bc')

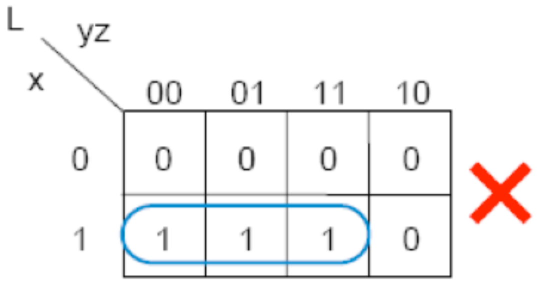
m1 can be placed next to m3 (a'b'c, a'bc)
m2 can be placed next to m3 (a'bc', a'bc)

Three-Variable K-Map Optimization

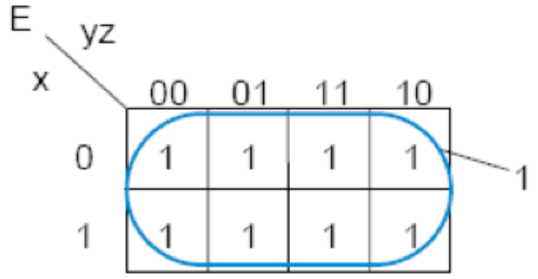
- Circles can cross left/right sides
  - Remember, edges are adjacent
    - Minterms differ in one variable only



- Circles must have 1, 2, 4, or 8 cells – 3, 5, or 7 not allowed
  - 3/5/7 doesn't correspond to algebraic transformations that combine terms to eliminate a variable



- Circling all the cells is OK
  - Function just equals 1



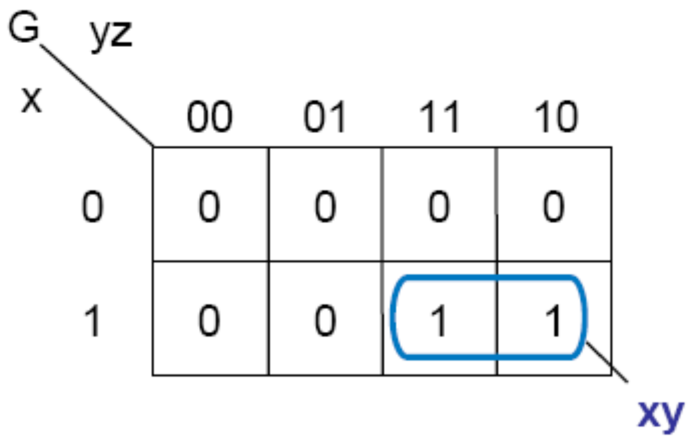
Three-Variable K-Map Optimization

- Two adjacent 1s means one variables can be eliminated
  - Same as in two-variable K-maps

$G = xyz + xyz'$

$G = xy(z + z')$

$G = xy$



Three-Variable K-Map Optimization

- Four adjacent 1s means two variables can be eliminated
  - Makes intuitive sense – those two variables appear in all combinations, so one must be true
  - Draw one big circle – shorthand for the algebraic transformations above

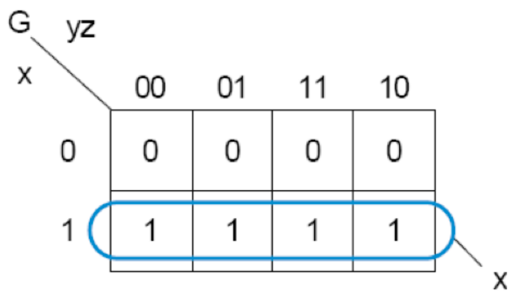
G = xy'z' + xy'z + xyz + xyz'

G = x(y'z'+ y'z + yz + yz') (must be true)

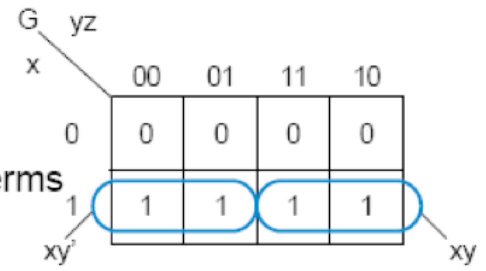
G = x(y'(z'+z) + y(z+z'))

G = x(y'+y)

G = x



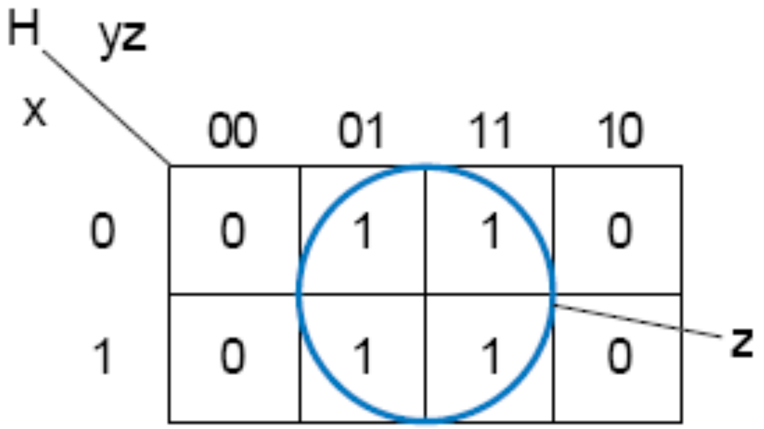
Draw the biggest circle possible, or you'll have more terms than really needed



Three-Variable K-Map Optimization

- Four adjacent cells can be in shape of a square

H = x'y'z + x'yz + xy'z + xyz
(xy appears in all combinations)

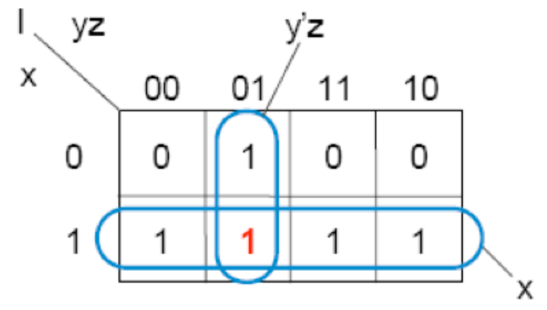


$$\begin{array}{c|cccc} x \backslash yz & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{array}$$

Three-Variable K-Map Optimization

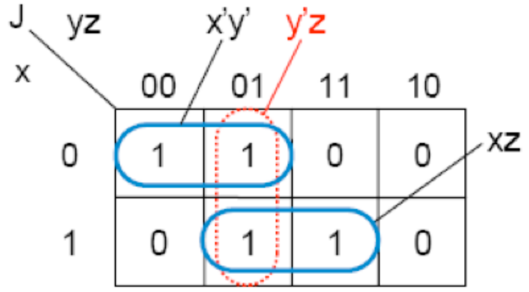
- Okay to cover a 1 twice
  - Just like duplicating a term
    - Remember, c + d = c + d + d

- No need to cover 1s more than once
  - Yields extra terms – not minimized



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

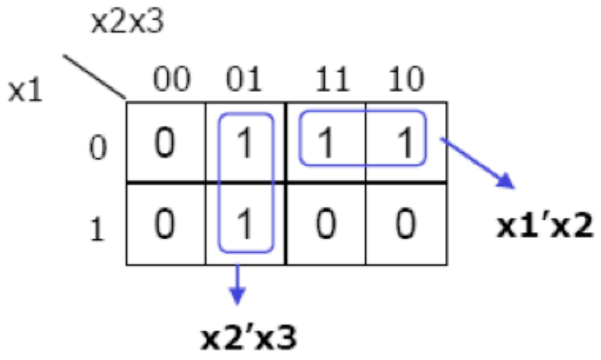
The two circles are shorthand for:
I = x'y'z + xy'z' + xy'z + xyz + xyz'
I = x'y'z + xy'z + xy'z' + xy'z + xyz + xyz'
I = (x'y'z + xy'z) + (xy'z' + xy'z + xyz + xyz')
I = (y'z) + (x)



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

Three-Variable K-Map Example

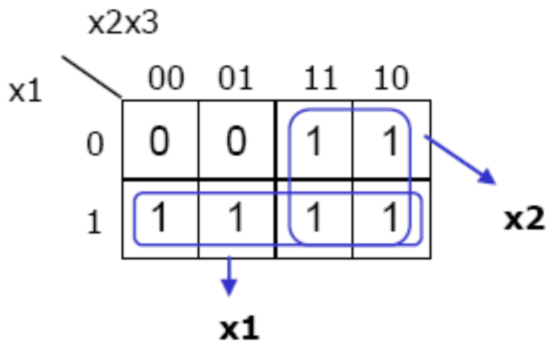
x1	x2	x3	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



F = x1'x2 + x2'x3

Three-Variable K-Map Example (2)

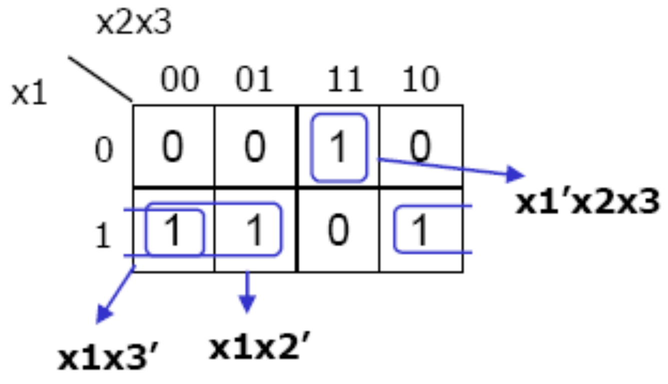
x1	x2	x3	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



F = x1 + x2

Three-Variable K-Map Example (3)

x1	x2	x3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



F = x1x3′ + x1x2′ + x1′x2x3

Generalized Four-Variable K-Map

Four-variable Map

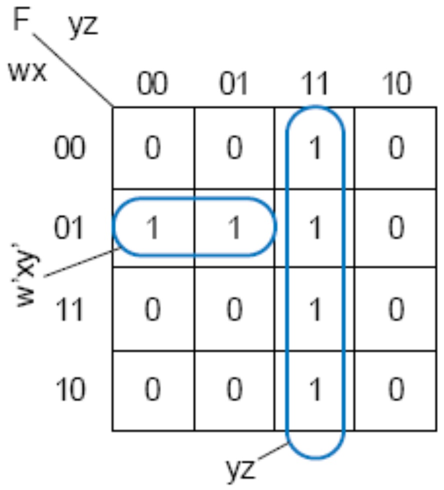
a	b	c	d	F
0	0	0	0	m0
0	0	0	1	m1
0	0	1	0	m2
0	0	1	1	m3
0	1	0	0	m4
0	1	0	1	m5
0	1	1	0	m6
0	1	1	1	m7
1	0	0	0	m8
1	0	0	1	m9
1	0	1	0	m10
1	0	1	1	m11
1	1	0	0	m12
1	1	0	1	m13
1	1	1	0	m14
1	1	1	1	m15

ab \ cd	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

Kookmin Univ. Embedded Systems Lab

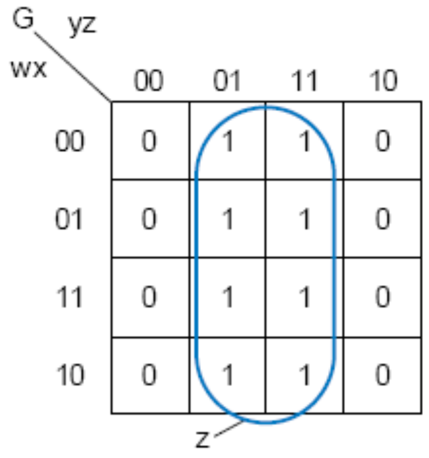
Four-Variable K-Map Optimization

- Four-variable K-map follows same principle
  - Left/right adjacent
  - Top/bottom also adjacent
- Adjacent cells differ in one variable
  - Two adjacent 1’s mean two variables can be eliminated
  - Four adjacent 1s means two variables can be eliminated
  - Eight adjacent 1s means three variables can be eliminated



F: wx \ yz	00	01	11	10
00	0	0	1	0
01	1	1	1	0
11	0	0	1	0
10	0	0	1	0

F=w’xy’+yz

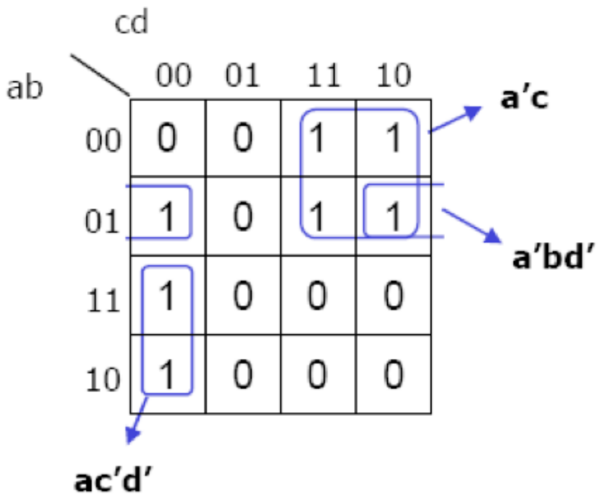


G: wx \ yz	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

G=z

Four-Variable K-Map Example

a	b	c	d	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

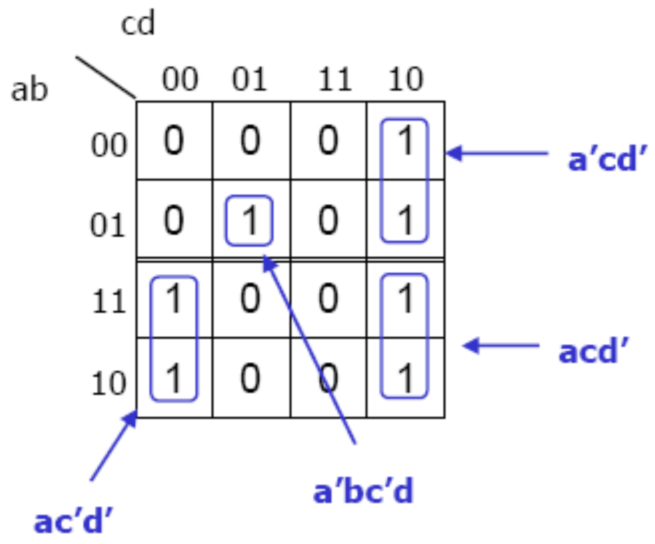


ab \ cd	00	01	11	10
00	0	0	1	1
01	1	0	1	1
11	1	0	0	0
10	1	0	0	0

F = a'c + a'bd' + ac'd'

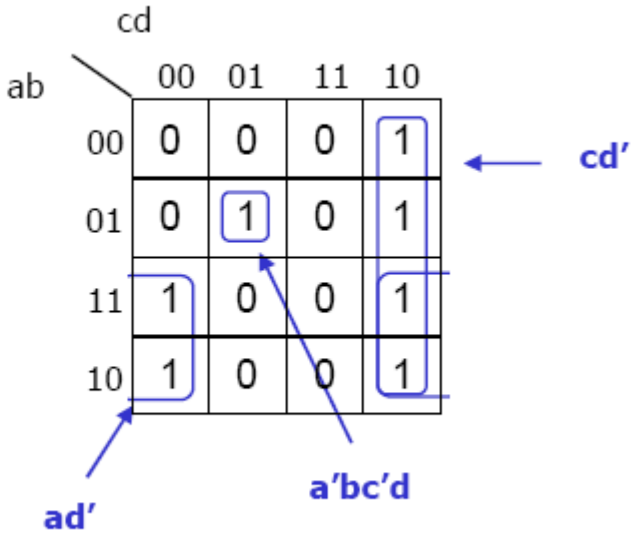
Four-Variable K-Map Example (2)

- Minimize F = a'cd' + a'bc'd + acd' + ac'd'
  - 1. First we fill in the K-map



Four-Variable K-Map Example (2)

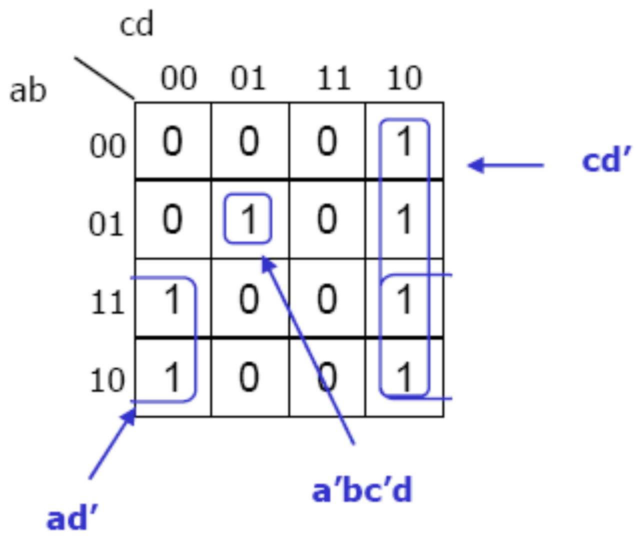
- Minimize F = a'cd' + a'bc'd + acd' + ac'd'
  - First we fill in the K-map
  - Second, draw largest circles to cover all 1s



ab \ cd	00	01	11	10
00	0	0	0	1
01	0	1	0	1
11	1	0	0	1
10	1	0	0	1

Four-Variable K-Map Example (2)

- Minimize F = a’cd’ + a’bc’d + acd’ + ac’d’
  - First we fill in the K-map
  - Second, draw largest circles to cover all 1s
  - Third, OR all product terms



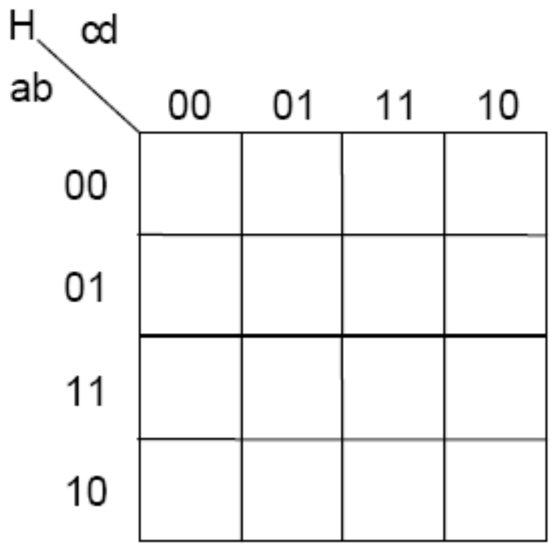
ab \ cd	00	01	11	10
00	0	0	0	1
01	0	1	0	1
11	1	0	0	1
10	1	0	0	1

F = cd’ + ad’ + a’bc’d

KESL
Kookmin Univ. Embedded Systems Lab

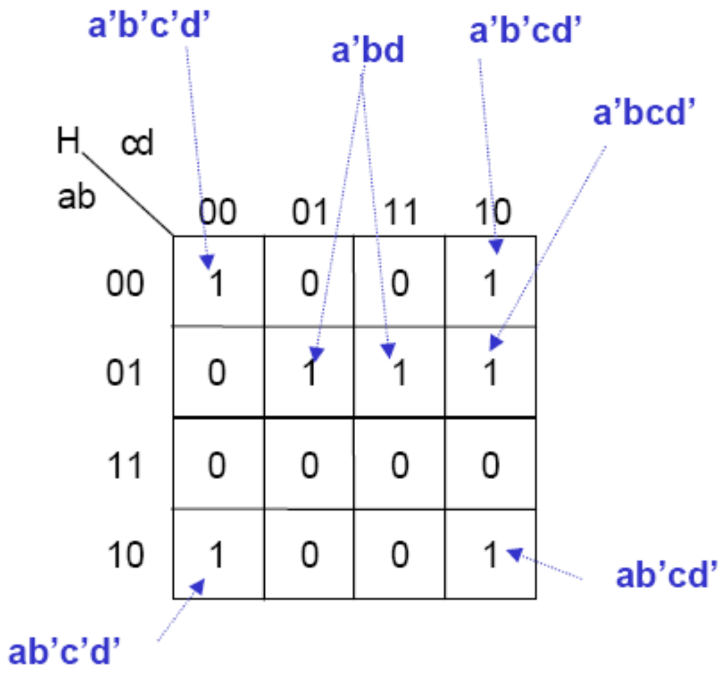
Four-Variable K-Map Example (3)

- Minimize: H = a’b’(cd’ + c’d’) + ab’c’d’ + ab’cd’ + a’bd + a’bcd’
  - Convert to sum-of-products
    - H = a’b’cd’ + a’b’c’d’ + ab’c’d’ + ab’cd’ + a’bd + a’bcd’



Four-Variable K-Map Example (3)

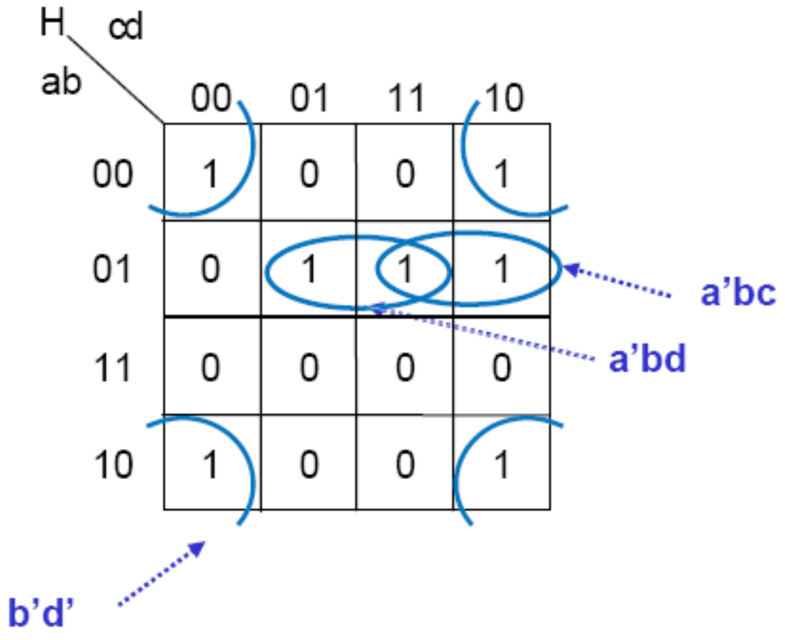
- Minimize: H = a'b'(cd' + c'd') + ab'c'd' + ab'cd' + a'bd + a'bcd'
  - Convert to sum-of-products
    - H = a'b'cd' + a'b'c'd' + ab'c'd' + ab'cd' + a'bd + a'bcd'
  - Place 1s in K-map cells



H ab \ cd	00	01	11	10
00	1	0	0	1
01	0	1	1	1
11	0	0	0	0
10	1	0	0	1

Four-Variable K-Map Example (3)

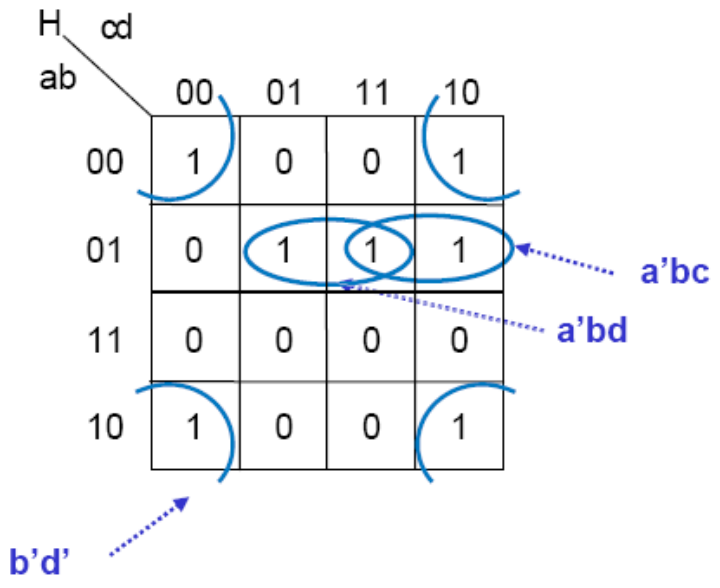
- Minimize: H = a'b'(cd' + c'd') + ab'c'd' + ab'cd' + a'bd + a'bcd'
  - Convert to sum-of-products
    - H = a'b'cd' + a'b'c'd' + ab'c'd' + ab'cd' + a'bd + a'bcd'
  - Place 1s in K-map cells
  - Cover 1s



ab \ cd	00	01	11	10
00	1	0	0	1
01	0	1	1	1
11	0	0	0	0
10	1	0	0	1

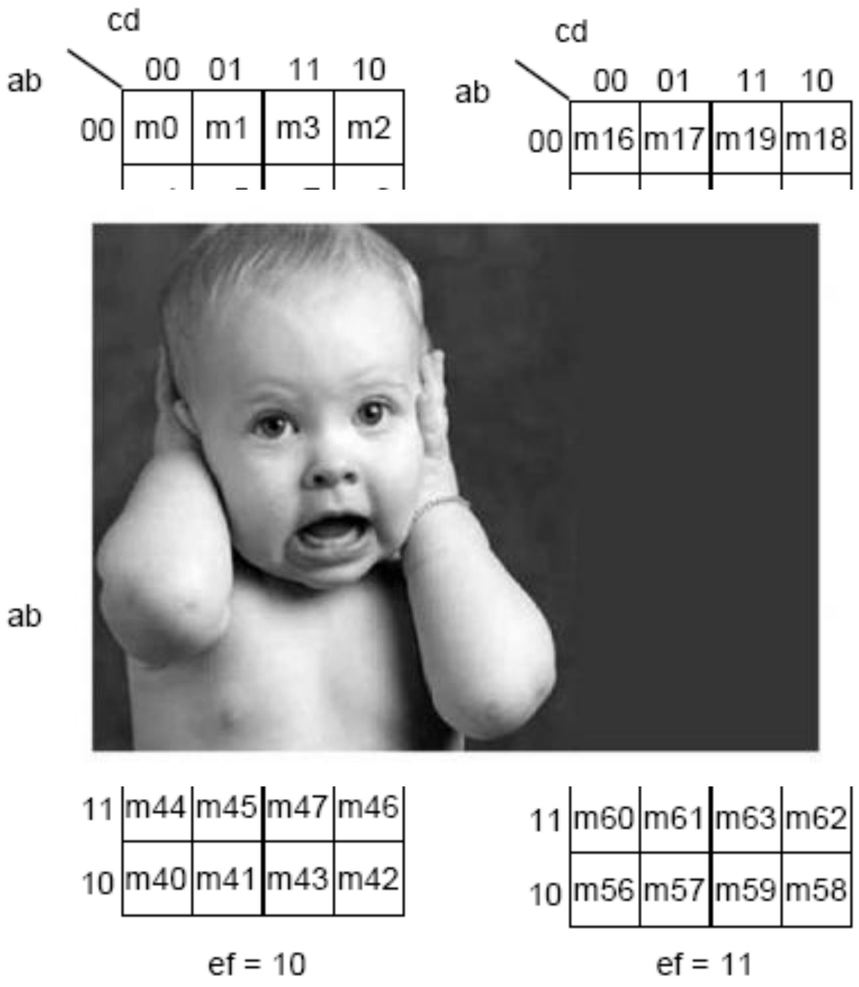
Four-Variable K-Map Example (3)

- Minimize: H = a’b’(cd’ + c’d’) + ab’c’d’ + ab’cd’ + a’bd + a’bcd’
  - Convert to sum-of-products
    - H = a’b’cd’ + a’b’c’d’ + ab’c’d’ + ab’cd’ + a’bd + a’bcd’
  - Place 1s in K-map cells
  - Cover 1s
  - OR resulting terms



H = b’d’ + a’bc + a’bd

Six-Variable K-Map



ab \ cd	00	01	11	10
00	m0	m1	m3	m2

ab \ cd	00	01	11	10
00	m16	m17	m19	m18

ab

11	m44	m45	m47	m46
10	m40	m41	m43	m42

ef = 10

11	m60	m61	m63	m62
10	m56	m57	m59	m58

ef = 11

Six-variable Map