Overview

- Last lecture
  - Logic simplification
    - Boolean cubes
    - Karnaugh maps
- Today
  - Incompletely specified functions
  - Design examples
  - k-maps for POS minimization

Incompletely specified functions

- Functions of n inputs have 2^n possible configurations
  - Some combinations may be unused
  - Call unused combinations "don't cares"
  - Exploit don't cares during logic minimization
  - Don't care ≠ no output
- Example: A BCD increment-by-1
  - Function F computes the next number in a BCD sequence
    - If the input is 0010_2, the output is 0011_2
  - BCD encodes decimal digits 0–9 as 0000_2–1001_2
    - Don't care about binary numbers 1010_2–1111_2

Truth table for a BCD increment-by-1

INPUTS				OUTPUTS			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

off-set for W: m0–m6, m9

on-set for W: m7 and m8

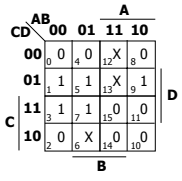
Don't care set for W: We **don't care** about the output values

Notation

- Don't cares in canonical forms
  - Three distinct logical sets
    - {on}, {off}, {don't care}
- Canonical representations of a BCD increment-by-1
  - Minterm expansion
    - W = m7+m8+d10+d11+d12+d13+d14+d15
  - Maxterm expansion
    - W = M0•M1•M2•M3•M4•M5•M6•M9•D10•D11•D12•D13•D14•D15

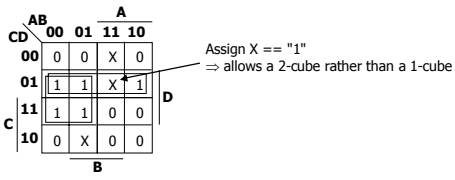
Karnaugh maps and don't cares

- Can treat don't cares as 0s or 1s
  - Depending on which is more advantageous
- F(A,B,C,D) = Σm(1,3,5,7,9) + d(6,12,13)
  - Example: Minimize F with and without don't cares

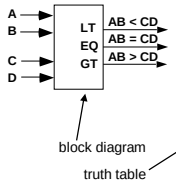


Karnaugh maps and don't cares (con't)

- F(A,B,C,D) = Σm(1,3,5,7,9) + d(6,12,13)
  - F = A'D + B'C'D *without using* don't cares
  - F = A'D + C'D *using* don't cares



Design example: A two-bit comparator

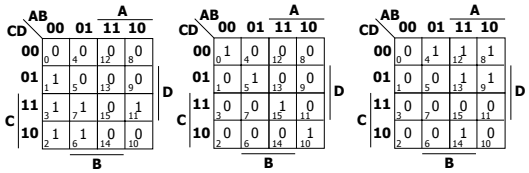


A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0

Need a 4-variable Karnaugh map for each of the 3 output functions

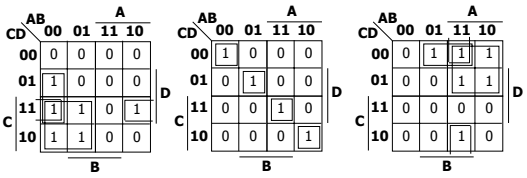
Two-bit comparator (con't)

K-map for LT **K-map for EQ** **K-map for GT**



LT = ? EQ = ? GT = ?

Two-bit comparator (con't)



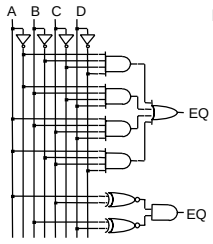
LT = A'B'D+A'C+B'CD

GT = BC'D'+AC'+ABD'

EQ = A'B'C'D'+A'BC'D+ABCD+AB'CD'
= (A xnor C)•(B xnor D)

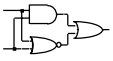
Two-bit comparator (con't)

- Two ways to implement EQ:

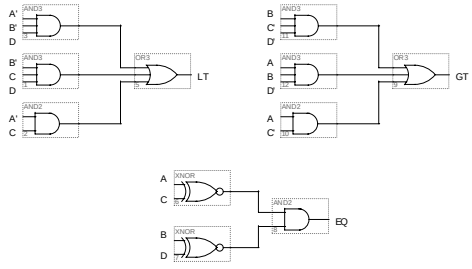


EQ = A'B'C'D'+A'BC'D+ABCD+AB'CD'
= (A xnor C) •(B xnor D)

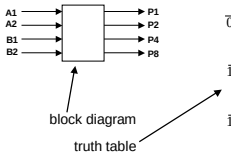
- XNOR approach looks simpler But XNORs are complex
- Example: XNOR constructed from 3 simple gates



Two-bit comparator design



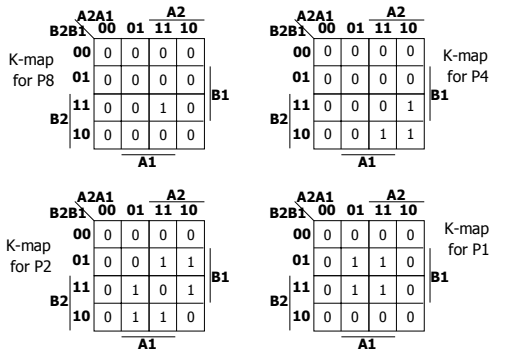
Design example: Two-bit multiplier



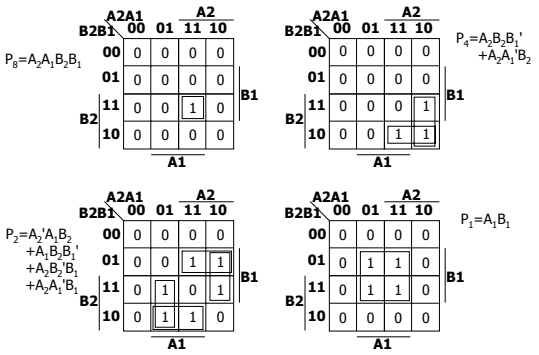
A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

Need a 4-variable Karnaugh map for each of the 4 output functions

Two-bit multiplier (cont'd)



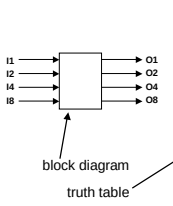
Two-bit multiplier (cont'd)



Two-bit multiplier design

- Draw the circuit schematic
  - $P_8=A_2A_1B_2B_1$
  - $P_4=A_2B_2B_1'+A_2A_1'B_2$
  - $P_2=A_2'A_1B_2+A_1B_2B_1'+A_2B_2'B_1+A_2A_1'B_1$
  - $P_1=A_1B_1$

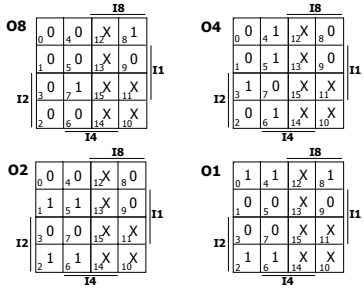
Design example: BCD increment by 1



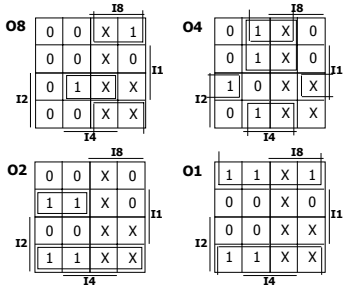
I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Need a 4-variable Karnaugh map for each of the 4 output functions

BCD increment by 1 (con't)



BCD increment by 1 (con't)



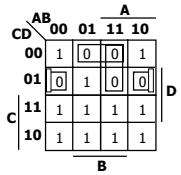
We greatly simplify the logic by using the don't cares

BCD increment by 1 design

- Draw the circuit schematic
  - $O_8 = I_4I_2I_1 + I_8I_1'$
  - $O_4 = I_4I_2' + I_4I_1' + I_4'I_2I_1$
  - $O_2 = I_8'I_2'I_1 + I_2I_1'$
  - $O_1 = I_1'$

Loose end: POS minimization using k-maps

- Using k-maps for POS minimization
  - Encircle the zeros in the map
  - Interpret indices complementary to SOP form



F = (B'+C+D)(B+C+D')(A'+B'+C)

Check using de Morgan's on SOP

F' = BC'D'+B'C'D+ABC'

(F')' = (BC'D'+B'C'D+ABC')'

F = (BC'D')'(B'C'D)'(ABC')'

F = (B'+C+D)(B+C+D')(A'+B'+C)