



CSE140: Components and Design Techniques for Digital Systems

Specifying logic functions

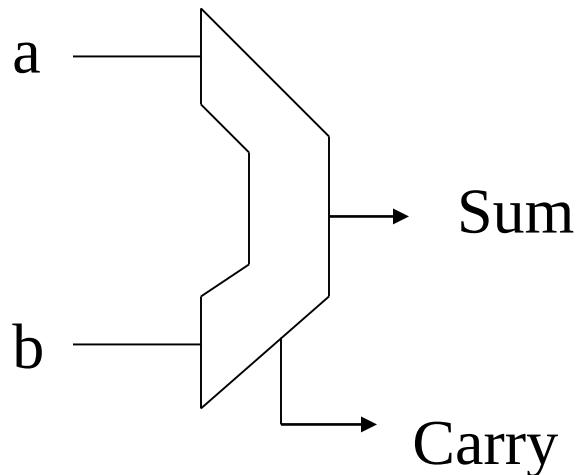
Last lecture

We have seen various concepts:

- **Logic function (as a mathematical formula):** some $F(X_0, X_1, X_2, \dots)$ with Boolean variables
- **Truth table:** a table that reports all possible combination of inputs and the corresponding output
- **Logic circuit (made of gates):** allows you to give a graphical representation of the circuit implementing a logic function
- They all are possible ways of representing the same thing
- Today: we will use these concepts and see how to “jump” between them
- We are going to need more definitions and more formalism !

Specifying Logic Problems with Truth Tables

- **Half Adder:** Truth table $\rightarrow$ Boolean Eq. $\rightarrow$ Logic Circuit



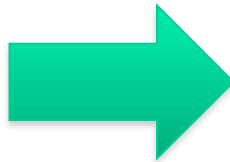
Truth Table

a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Going from a Truth Table to a Boolean Equation

Truth Table

a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Boolean Equation:

$$\begin{aligned}\text{Sum (a, b)} &= a' b + ab' = \\ &= a \text{ XOR } b\end{aligned}$$

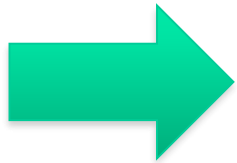
$$\text{Carry (a, b)} = ab$$

How do we get a Boolean Equation from the Truth Table?

Truth Table

a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

How?



Boolean Equation

$$\begin{aligned}\text{Sum (a, b)} &= a' b + ab' \\ \text{Carry (a, b)} &= ab\end{aligned}$$

Need Some Definitions

- **Complement:** variable with a bar over it or a '
 A', B', C'
- **Literal:** variable or its complement
 A, A', B, B', C, C'
- **Implicant:** product of literals
 ABC, AC, BC
- **Implicate:** sum of literals
 $(A+B+C), (A+C), (B+C)$
- **Minterm:** AND that includes all input variables
 $ABC, A'BC, AB'C$
- **Maxterm:** OR that includes all input variables
 $(A+B+C), (A'+B+C), (A'+B'+C)$

Minterms of Two Input Functions

A	B	Minterm	Maxterm
0	0	$A'B'$	$A+B$
0	1	$A'B$	$A+B'$
1	0	AB'	$A'+B$
1	1	AB	$A'+B'$



CSE140: Components and Design Techniques for Digital Systems

Canonical representation

Canonical Form -- Sum of Minterms

- Truth tables are too big for numerous inputs
- Use standard form of equation instead
 - Known as **canonical form**
 - Regular algebra: group terms of polynomial by power
 - $ax^2 + bx + c$ ($3x^2 + 4x + 2x^2 + 3 + 1 \rightarrow 5x^2 + 4x + 4$)
 - Boolean algebra: create a sum of **minterms** (or a product of **maxterms**)

Canonical form?

Is $F(a,b)=ab+a'$ in canonical form?

A) True

B) False

Sum-of-products Canonical Form

- Also known as disjunctive normal form
- Minterm expansion:

					$F = 001 \quad 011 \quad 101 \quad 110 \quad 111$				
					$F = A'B'C + A'BC + AB'C + ABC' + ABC$				
A	B	C	F	F'					
0	0	0	0	1					
0	0	1	1	0					
0	1	0	0	1					
0	1	1	1	0					
1	0	0	0	1					
1	0	1	1	0					
1	1	0	1	0					
1	1	1	1	0					
					$F' = A'B'C' + A'BC' + AB'C'$				

Sum-of-products canonical form (cont'd)

- Product minterm
 - ANDed product of literals – input combination for which output is 1
 - each variable appears exactly once, true or inverted (but not both)
 - You can write a canonical form in multiple ways

A	B	C	minterms	F
0	0	0	A'B'C' m0	0
0	0	1	A'B'C m1	1
0	1	0	A'BC' m2	0
0	1	1	A'BC m3	1
1	0	0	AB'C' m4	0
1	0	1	AB'C m5	1
1	1	0	ABC' m6	1
1	1	1	ABC m7	1

short-hand notation for
minterms of 3 variables

F in canonical form:

$$\begin{aligned}
 F(A, B, C) &= \Sigma m(1,3,5,6,7) \\
 &= m1 + m3 + m5 + m6 + m7 \\
 &= A'B'C + A'BC + AB'C + ABC' + ABC
 \end{aligned}$$

canonical form $\neq$ minimal form

$$\begin{aligned}
 F(A, B, C) &= A'B'C + A'BC + AB'C + ABC + ABC' \\
 &= (A'B' + A'B + AB' + AB)C + ABC' \\
 &= ((A' + A)(B' + B))C + ABC' \\
 &= C + ABC' \\
 &= ABC' + C \\
 &= AB + C
 \end{aligned}$$

Sum of Products Canonical Form

Minterm	A	B	Carry	Sum
$A'B'$	0	0	0	0
$A'B$	0	1	0	1
AB'	1	0	0	1
AB	1	1	1	0

I. $\text{sum}(A,B) =$

II. $\text{carry}(A,B) =$

Sum of Products Canonical Form

A	B	Y
0	0	0
1	0	0
0	1	1
1	1	1

Does the following SOP canonical expression correctly express the above truth table:

$$Y(A,B) = \sum m(2,3)$$

A. Yes

B. No

Product-of-sums canonical form

- Also known as conjunctive normal form
- Also known as **maxterm** expansion
- Implements “zeros” of a function

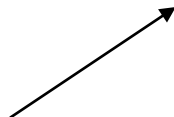
					$F =$	000	010	100
					$F =$	$(A + B + C)$	$(A + B' + C)$	$(A' + B + C)$
A	B	C	F	F'				
0	0	0	0	1				
0	0	1	1	0				
0	1	0	0	1				
0	1	1	1	0				
1	0	0	0	1				
1	0	1	1	0				
1	1	0	1	0				
1	1	1	1	0				

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

Product-of-sums canonical form (cont'd)

- Sum term (or maxterm)
 - ORed sum of literals – input combination for which output is false
 - each variable appears exactly once, true or inverted (but not both)

A	B	C	maxterms		F
0	0	0	A+B+C	M0	0
0	0	1	A+B+C'	M1	1
0	1	0	A+B'+C	M2	0
0	1	1	A+B'+C'	M3	1
1	0	0	A'+B+C	M4	0
1	0	1	A'+B+C'	M5	1
1	1	0	A'+B'+C	M6	1
1	1	1	A'+B'+C'	M7	1



short-hand notation for
maxterms of 3 variables

F in canonical form:

$$\begin{aligned}
 F(A, B, C) &= \prod M(0, 2, 4) \\
 &= M0 \cdot M2 \cdot M4 \\
 &= (A + B + C) (A + B' + C) (A' + B + C)
 \end{aligned}$$

canonical form $\neq$ minimal form

$$\begin{aligned}
 F(A, B, C) &= (A + B + C) (A + B' + C) (A' + B + C) \\
 &= (A + B + C) (A + B' + C) \\
 &\quad (A + B + C) (A' + B + C) \\
 &= (A + C) (B + C)
 \end{aligned}$$

Product of Sum Canonical Form

Maxterm	A	B	Carry	Sum
	0	0	0	0
	0	1	0	1
	1	0	0	1
	1	1	1	0

Carry(A,B)=

Product of Sum Canonical Form

Maxterm	A	B	Carry	Sum
	0	0	0	0
	0	1	0	1
	1	0	0	1
	1	1	1	0

The POS expression for $\text{sum}(A,B)$

- A. $(A' + B) * (A + B')$
- B. $A' B + AB'$
- C. $(A + B') * (A' + B)$
- D. Either A or C

When would you use the SOP instead of the POS to express the switching function?

- A. When the output of the function is TRUE for most input combinations
- B. When the output of the function is FALSE for most input combinations
- C. We always prefer the SOP form because it's more compact
- D. None of the above

Mapping between canonical forms

- Minterm to maxterm conversion
 - use maxterms whose indices do not appear in minterm expansion
 - e.g., $F(A,B,C) = \sum m(1,3,5,6,7) = \prod M(0,2,4)$
- Maxterm to minterm conversion
 - use minterms whose indices do not appear in maxterm expansion
 - e.g., $F(A,B,C) = \prod M(0,2,4) = \sum m(1,3,5,6,7)$
- Minterm expansion of F to minterm expansion of F'
 - use minterms whose indices do not appear
 - e.g., $F(A,B,C) = \sum m(1,3,5,6,7) \quad F'(A,B,C) = \sum m(0,2,4)$
- Maxterm expansion of F to maxterm expansion of F'
 - use maxterms whose indices do not appear
 - e.g., $F(A,B,C) = \prod M(0,2,4) \quad F'(A,B,C) = \prod M(1,3,5,6,7)$



The SOP and POS forms don't usually give the optimal circuit or the simplest Boolean expression of the switching function

To optimize the circuit, we need to simplify using:

1. Boolean algebra axioms and theorems
2. Karnaugh Maps (K-Maps)



BREAK !



CSE140: Components and Design Techniques for Digital Systems

Logic Simplification with K-maps

Tajana Simunic Rosing

Two-level minimization

- Let's look at a SOP canonical form:

SOP

$$F = A'B'C' + A'B'C + A'BC' + A'BC + AB'C' + ABC'$$

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

To easily get there, we need a “tool” called k-map, which allows to easily apply the minimization algorithm

Equivalent minimum 2-level form:

$$F = A' + C'$$

Key to simplification: the uniting theorem

- Uniting theorem: $X(Y' + Y) = X$
- Essence of simplification of two-level logic
 - find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

$$F = A'B' + AB' = (A' + A)B' = B'$$

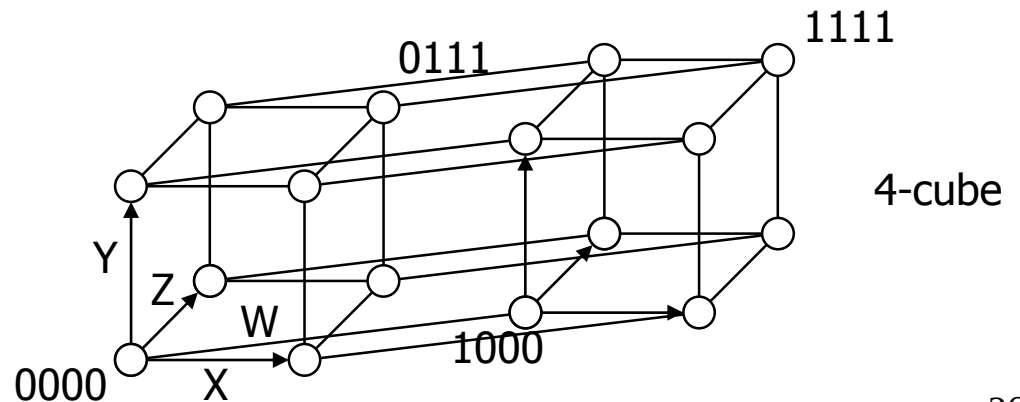
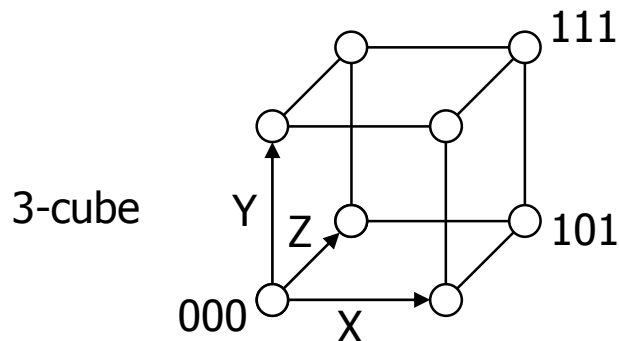
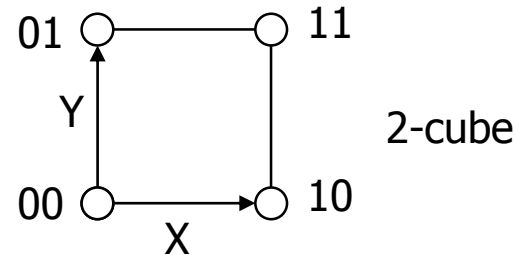
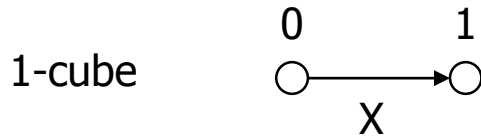
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

B has the same value in both on-set rows
– B remains

A has a different value in the two rows
– A is eliminated

Boolean cubes

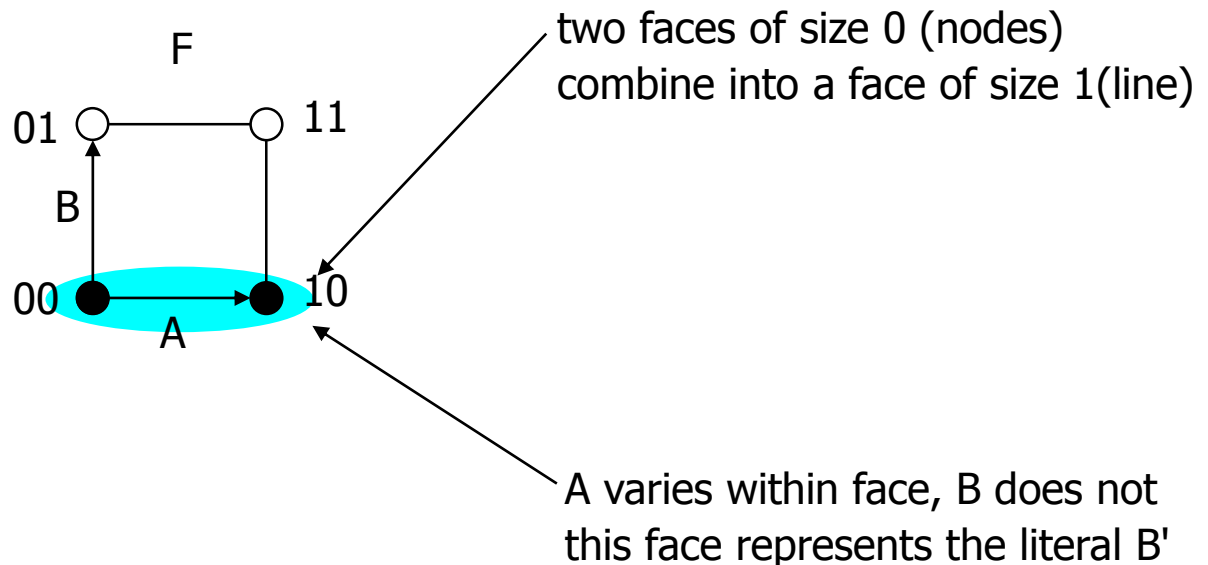
- Visual technique for applying the uniting theorem
- n input variables = n -dimensional "cube"
- RULE: adjacent points should differ by only 1 bit



Mapping truth tables onto Boolean cubes

- Uniting theorem combines two “faces” of a cube into a larger “face”
- Example:

A	B	F
0	0	1
0	1	0
1	0	1
1	1	0



ON-set = solid nodes
OFF-set = empty nodes
DC-set = x'd nodes

Karnaugh maps

- Flat map of Boolean cube
 - wrap-around at edges
 - hard to draw and visualize for more than 4 variables
- Alternative to truth-tables to help visualize adjacencies
 - guide to applying the uniting theorem
 - on-set elements with only one variable changing value are adjacent unlike the linear truth-table

ID	A	B	f(A,B)
0	0	0	f(0,0)
1	0	1	f(0,1)
2	1	0	f(1,0)
3	1	1	f(1,1)

	A=0	A=1
B=0	$A'B'$	AB'
B=1	$A'B$	AB

A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

	A	0	1
B	0	1 0	1 2
	1	0 1	0 3

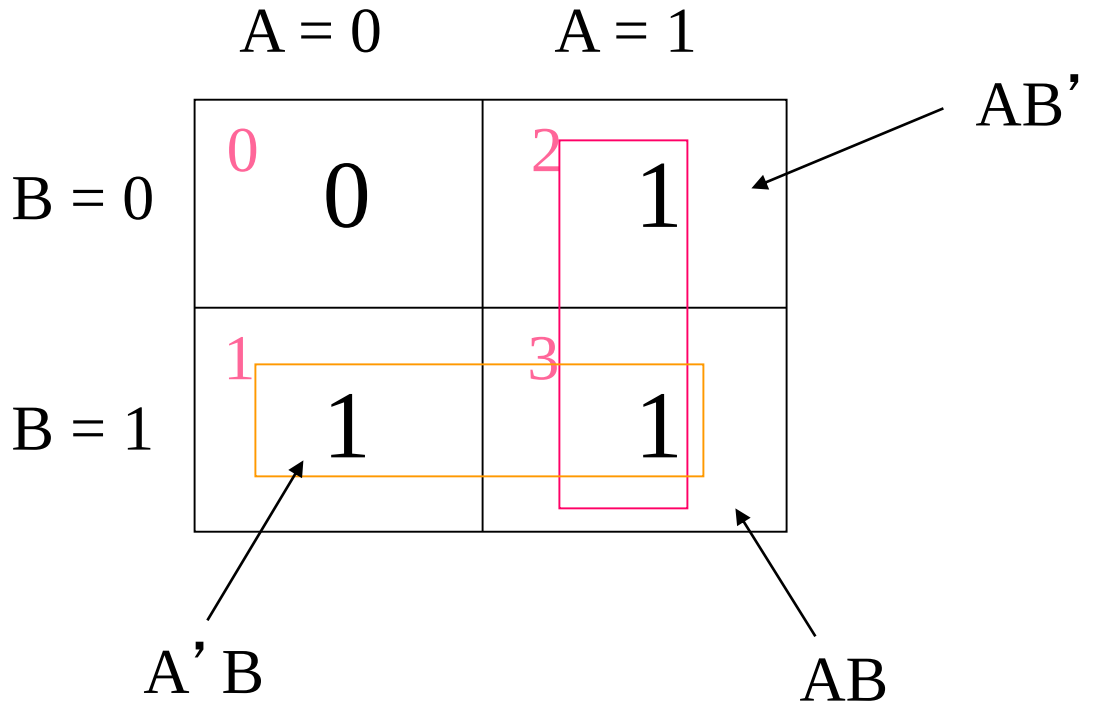
2-Variable K-Map Example

ID	A	B	f(A,B)
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

	A = 0	A = 1
B = 0	00	21
B = 1	11	31

2-Variable K-Map Example Cont.

ID	A	B	f(A,B)
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1



$$f(A,B) = A + B$$



CSE140: Components and Design Techniques for Digital Systems

“Don’t cares”

Tajana Simunic Rosing

Incompletely Specified Functions

- Situations where the output of a function can be either 0 or 1 for a particular combination of inputs
- This is specified by a **don't care** in the truth table

Id	a	b	f (a, b)
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	X: don't care

This happens when

- 1) The input does not occur.

E.g. Decimal numbers 0... 9 use 4 bits, so (1,1,1,1) does not occur.

- 2) The input may happen but we don't care about the output.

E.g. The output driving a seven segment display – we don't care about illegal inputs (greater than 9)

Don't care! Does it help to simplify?

Do we get a smaller circuit implementation if we assume $f(1,1)=1$ instead of $f(1,1)=0$?

A. Yes

B. No

ID	a	b	f (a, b)
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	X

Completely Specifying Boolean Functions

- Onset Σm (set of minterms where the function is 1):
 - All the input conditions for which the output is 1
- Offset ΠM (set of maxterms where the function is 0):
 - All the input conditions for which the output is 0
- Don't cares DC:
 - All the input conditions for which the output is a 'don't care'

ID	a	b	$g(a,b)$
0	0	0	0
1	0	1	X
2	1	0	1
3	1	1	0

**We need two of the three sets
to represent any function:
on-set, off-set, and dc-set**

$\Sigma m(2); \Pi M (0,3); DC(1)$

$g(a,b) = \Sigma m(2) + DC(1)$

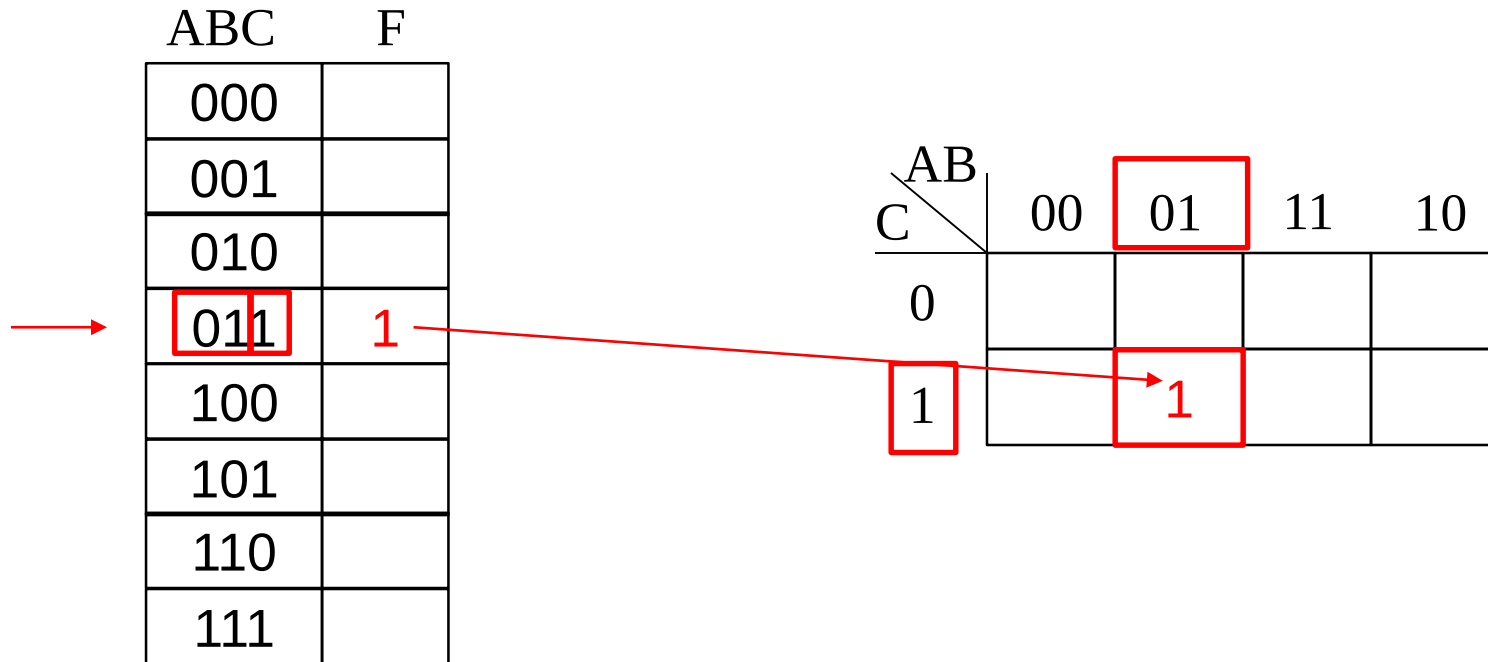
Karnaugh Maps with 3 Variables

- Numbering scheme is based on **Gray-code**
 - e.g., 00, 01, 11, 10
 - only a single bit changes in code for adjacent map cells

C \ AB				
	00	01	11	10
0				
1				

Karnaugh Maps with 3 Variables

- Numbering scheme is based on **Gray-code**
 - e.g., 00, 01, 11, 10
 - **only a single bit changes in code for adjacent map cells**



Three Variable K-map

Is this K-map correct?

A. Yes

B. No

(a,b)		(0,0)	(0,1)	(1,0)	(1,1)			
c = 0	0	1	2	1	4	1	6	1
c = 1	1	0	3	0	5	0	7	0

Truth Table

Id	a	b	c	f (a,b,c)
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Reduction in three variable K-maps

		(a,b)			
		(0,0)	(0,1)	(1,1)	(1,0)
c	= 0	0 0	2 1	6 1	4 0
	= 1	1 0	3 1	7 1	5 0

Truth Table

Id	a	b	c	f (a,b,c)
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

Reduction in three variable K-maps

(a,b)		(0,0)	(0,1)	(1,1)	(1,0)
c	0	0 1	2 0	6 0	4 1
	1	1 1	3 0	7 0	5 1

Truth Table

Id	a	b	c	f (a,b,c)
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	X
7	1	1	1	1

K-map with Don't Cares

	(0,0)	(0,1)	(1,1)	(1,0)
c = 0	0 0	2 1	6 X	4 1
c = 1	1 0	3 0	7 1	5 1

4-input K-Maps with Don't Cares

- $f(A,B,C,D) = \sum m(1,3,5,7,9) + d(6,12,13)$

without don't cares

$$f = B'C'D + A'D$$

with don't cares

$$f = C'D + A'D$$

		AB			
		00	01	11	10
CD	00	0	0	x	0
	01	1	1	x	1
	11	1	1	0	0
	10	0	x	0	0

		AB			
		00	01	11	10
CD	00	0	0	x	0
	01	1	1	x	1
	11	1	1	0	0
	10	0	x	0	0

don't cares can be treated as 1s or 0s
depending on which is more advantageous

4-input K-Maps with Don't Cares

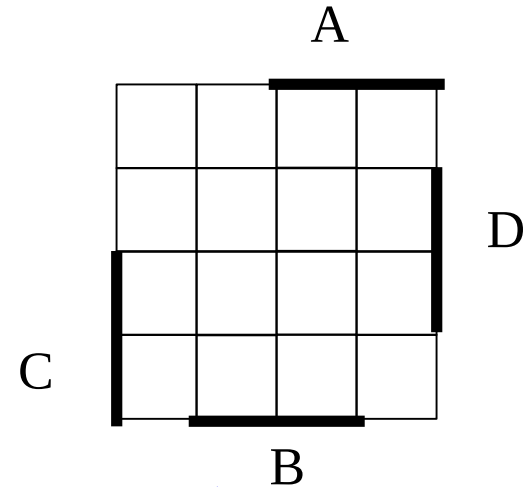
- $f(A,B,C,D) = \sum m(1,3,5,7,9) + d(6,12,13)$

$$f = (D') * (C' + A')$$

AB \ CD		AB			
		00	01	11	10
CD	00	0	0	x	0
	01	1	1	x	1
	11	1	1	0	0
	10	0	x	0	0

4-input K-Mmap: Alternative Representation

CD \ AB	AB			
	00	01	11	10
00				
01				
11				
10				



Place a bold bar where the value of a variable is 1 and label it.
Outside of the bar, the value of the variable is 0

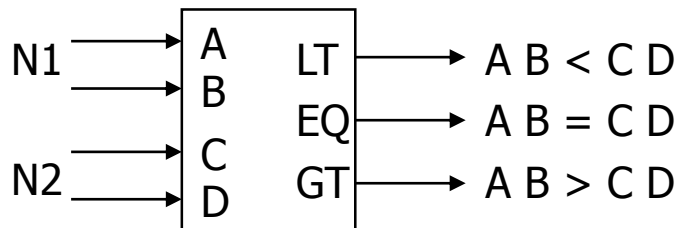
Another Example

- $F = \Sigma m(0, 2, 7, 8, 14, 15) + d(3, 6, 9, 12, 13)$



BREAK !

Design example: two-bit comparator



block diagram
and
truth table

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

we'll need a 4-variable Karnaugh map
for each of the 3 output functions

Design example: two-bit comparator (cont'd)

	A				
	0	0	0	0	
	1	0	0	0	
C	1	1	0	1	D
	1	1	0	0	
	B				

K-map for LT

	A				
	1	0	0	0	
	0	1	0	0	
C	0	0	1	0	D
	0	0	0	1	
	B				

K-map for EQ

	A				
	0	1	1	1	
	0	0	1	1	
C	0	0	0	0	D
	0	0	1	0	
	B				

K-map for GT

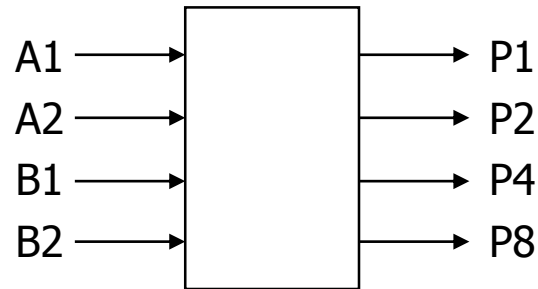
$$LT = A' B' D + A' C + B' C D$$

$$EQ = A' B' C' D' + A' B C' D + A B C D + A B' C D' = (A \text{ xnor } C) \cdot (B \text{ xnor } D)$$

$$GT = B C' D' + A C' + A B D'$$

LT and GT are similar (flip A/C and B/D)

Design example: 2x2-bit multiplier



block diagram
and
truth table

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

4-variable K-map
for each of the 4
output functions

Design example: 2x2-bit multiplier (cont'd)

K-map for P8

		A2		
	0	0	0	0
	0	0	0	0
B2	0	0	1	0
	0	0	0	0
	A1			
			B1	

K-map for P4

		A2		
	0	0	0	0
	0	0	0	0
B2	0	0	0	1
	0	0	1	1
	A1			
			B1	

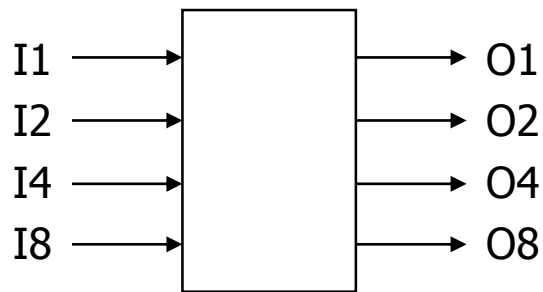
K-map for P2

		A2		
	0	0	0	0
	0	0	1	1
B2	0	1	0	1
	0	1	1	0
	A1			
			B1	

K-map for P1

		A2		
	0	0	0	0
	0	1	1	0
B2	0	1	1	0
	0	0	0	0
	A1			
			B1	

Design example: BCD + 1



block diagram
and
truth table

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

4-variable K-map for each of
the 4 output functions

Design example: BCD + 1 (cont'd)

				I8		
					<u>O8</u>	
I2	0	0	X	1	I1	
	0	0	X	0		
	0	1	X	X		
	0	0	X	X		
				I4		

				I8		
					<u>O4</u>	
I2	0	1	X	0	I1	
	0	1	X	0		
	1	0	X	X		
	0	1	X	X		
				I4		

				I8		
					<u>O2</u>	
I2	0	0	X	0	I1	
	1	1	X	0		
	0	0	X	X		
	1	1	X	X		
				I4		

				I8		
					<u>O1</u>	
I2	1	1	X	1	I1	
	0	0	X	0		
	0	0	X	X		
	1	1	X	X		
				I4		