18-100 Lecture 22: Implementing Combinational Logic

James C. Hoe
Dept of ECE, CMU
April 9, 2015

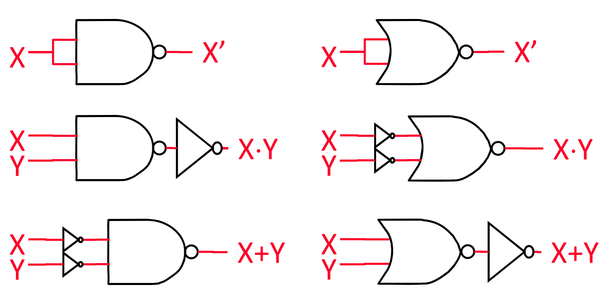
Today's Goal: Design some combinational logic circuits

Announcements: Read Rizzoni 12.4 and 12.5
HW 8 due today

Handouts: Lab 11 for week of 4/20 (on Blackboard)
HW 9 due Tuesday 4/21 (on Blackboard)
HW 8 Solutions (on Blackboard later today)

Universality of NAND and NOR Gates

- Either NAND or NOR is sufficient to implement AND/OR/NOT
  - convenient because inverting logic gates arise naturally in transistor-based implementations
  - NOT gate:
  - AND gate
  - OR gate



- But still, how do we know AND/OR/NOT is sufficient to implement any combinational function imaginable?

Let's Design Something

- Input: 1-bit A, 1-bit B, and 1-bit C
- Output: 2-bit S that is a 2-bit unsigned integer; indicates how many bits of the input A, B and C are asserted; possible values are 0, 1, 2, 3
- Start with the Truth Table
  - 2^N rows for a function of N inputs
  - better to enumerate input patterns systematically, e.g., as if counting
  - uniquely and unambiguously define a combinational function

Now what?

A	B	C	S[1]	S[0]
0	0	0	**0**	**0**
0	0	1	**0**	**1**
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**	**1**

Canonical Sum-of-Products (SOP)

- Each row of a truth table corresponds to a distinct input combination (aka, a "minterm")
  - e.g., row 0 corresponds to A'·B'·C', *aka m0*
    row 1 corresponds to A'·B'·C, *aka m1*
    row 2 corresponds to A'·B·C', *aka m2*
    etc.
- Construct a function by OR'ing the minterms in the "on-set" (i.e., the 1s)

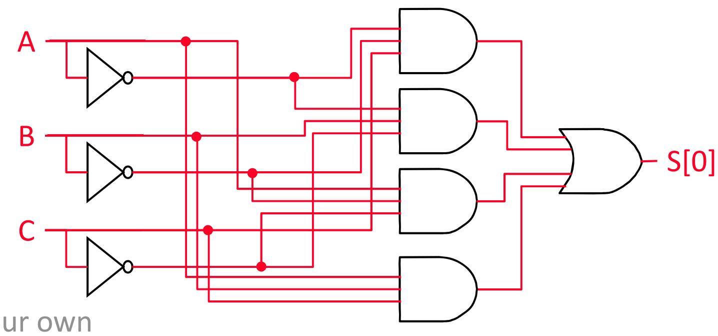
S[0]=A'B'C+A'BC'+AB'C'+ABC
m1 m2 m4 m7

S[1]=A'BC +AB'C +ABC'+ABC
m3 m5 m6 m7

A	B	C	S[1]	S[0]
0	0	0	**0**	**0**
0	0	1	**0**	**1**
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**	**1**

2-level SOP Implementation

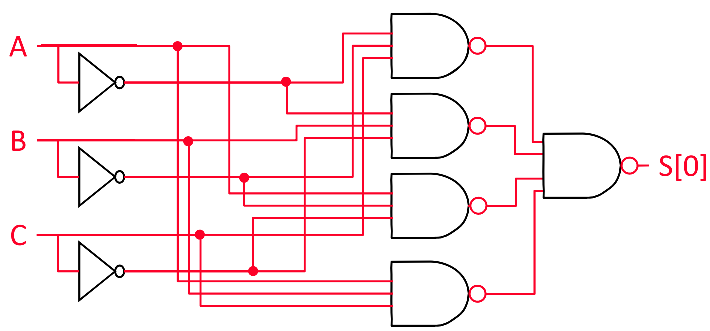
- S[0]=A'B'C+A'BC'+AB'C'+ABC
  - each minterm is a "product" (AND is boolean multiply)
  - S[0] is the sum of products (OR is boolean addition)
- Every function has a truth table; every truth table has a corresponding "canonical" SOP form
  $\Rightarrow$ every fxn can be implemented using AND/OR/NOT



Try S[1] on your own

NAND-NAND is SOP

- What if you have only NANDs



Canonical Product of Sums (POS)

- What if the truth table is mostly 1s and you are too lazy to write out all those minterms?

  ANS: write SOP for F'

- E.g., F'=A'B'C'+AB'C

- Apply De Morgan to get back F in POS form

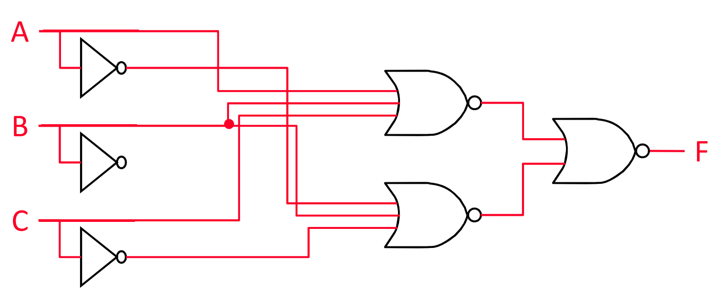
F=(A+B+C)(A'+B+C')

This is a maxterm, that is, (A+B+C) is true everywhere except A'B'C'

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

2-level POS Implementation

- F=(A+B+C)(A'+B+C')
  - each maxterm is a "sum" (OR is boolean addition)
  - F is the product of sums (AND is boolean multiply)
- Every function also has a "canonical" POS form
- NOR-NOR is POS



Minimum SOP Form

- Simplified SOP expression that minimizes the number of gates and the number of input to gates, e.g.,
- F =A'B'C+A'BC+AB'C+ABC'+ABC
  - =A'B'C+A'BC+AB'C+ABC'+ABC+ABC rule 3
  - =A'B'C+A'BC+AB'C+ABC+ABC'+ABC rule 10
  - =A'B'C+A'BC+AB'C+ABC+AB(C'+C) rule 14
  - =A'B'C+A'BC+AB'C+ABC+AB·1 rule 4
  - =A'B'C+A'BC+AB'C+ABC+AB rule 6
- Let's make the last 3 steps into a theorem
  - XY'+XY=X(Y'+Y)=X·1=X (Uniting Theorem)
- F = (A'B'C+A'BC)+(AB'C+ABC)+AB
  - = A'C+AC+AB
  - = C+AB

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

Can we do any better? If not, can we know for sure?

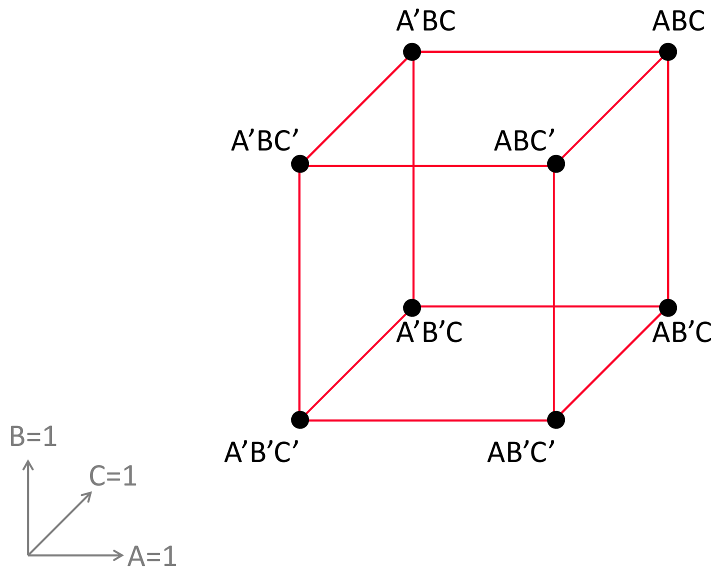
Applying the Uniting Theorem

- How to recognize opportunities to "unite"?
  - looking for pairs of minterms in the on-set that are distance-1 (reachable by toggling one input)
- E.g., **AB**C' and **AB**C
  - F asserted if AB, independent of C
  - replace ABC'+ABC by AB in SOP
- But this is tricky to do . . .
  - not all minterm neighbors are easy to see in a truth table
  - united terms themselves can be further united, e.g., A'C+AC=C

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

Electrical & Computer ENGINEERING

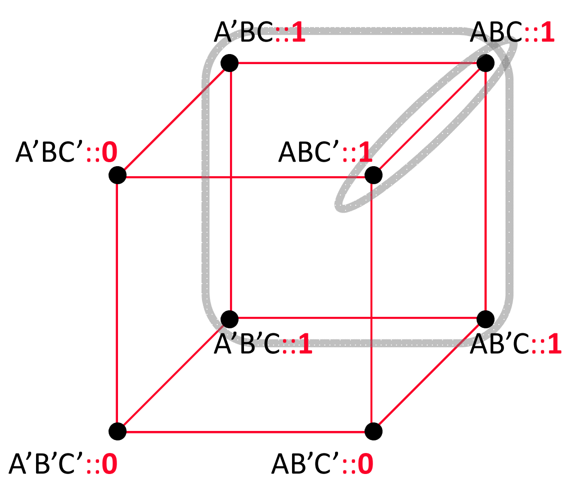
Truth Table on a Cube

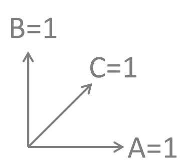


A	B	C	F
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Electrical & Computer ENGINEERING

Truth Table on a Cube





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

F= C + AB

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

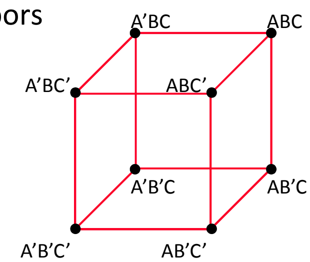
Karnaugh Map (K-map)

C=1 half

	A'B'C'	A'B'C	A'BC	A'BC'
A=1 half	AB'C'	AB'C	ABC	ABC'

B=1 half

- Same as the 3-d cube before, but easier to draw
  - each minterm has 3 adjacent neighbors
  - minterms that differ in only 1 input are adjacent
  - any two adjacent minterms form a 1-d sub-cube
  - any 4 minterms in a rectangle or square form a 2-d sub-cube



Karnaugh Map (K-map)

C

	0	1	1	0
A	0	1	1	1

B

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

- F = A'B'C+A'BC+AB'C+ABC+ABC'
- F = A'C+AC+AB
  = B'C+BC+AB
- F = C + AB

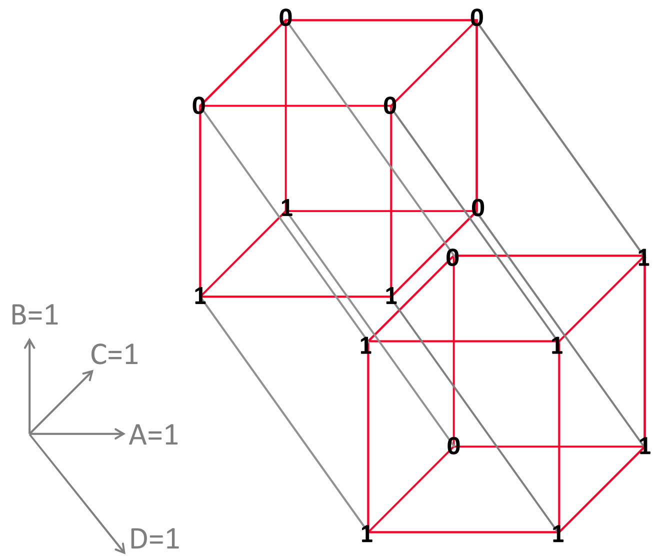
each product is an "implicant" of F; i.e., a sufficient condition for F

Finding minimum SOP corresponds to covering all members of the on-set using the largest cubes possible without redundant cubes

4-Variable Hypercube

(Example 12.11)

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



4-Variable K-map: Example 12.11

B (columns 2–3); A (columns 3–4); D (rows 2–3); C (rows 3–4)

1	0	0	1
1	1	1	1
0	0	1	1
1	0	0	0

F= AD+C'D
+ B'C'
+ A'B'D'

Attempt to cover using, in order, 4-d, 3-d, 2-d, 1-d, 0-d (minterm) cubes
Stop when all members of the on-set are covered
Check for redundancies (Be sure to study examples in Rizzoni 12.4)

Does XY+YZ+X’Z=XY+X’Z ?

		Z	Z	
	0	1	1	0
X	0	0	1	1
			Y	Y

- Proof strategy I
  - Expand XY, YZ, X’Z to the 4 minterms
  - Unite XYZ and XYZ’ to XY; unite X’YZ and X’Y’Z to X’Z
- Proof strategy II
  - expand YZ to minterms XYZ and X’YZ
  - XYZ absorbs into XY; X’YZ absorbs into X’Z

Does XY+YZ+X’Z=XY+X’Z ?

	XY+YZ+X’Z	
=	XY·1+YZ·1+X’Z·1	rule 6
=	XY(Z+Z’)+YZ(X+X’)+X’Z(Y+Y’)	rule 4
=	XYZ+XYZ’+YZX+YZX’+X’ZY+X’ZY’	rule 14
=	XYZ+XYZ’+XYZ+X’YZ+X’YZ+X’Y’Z	rule 11
=	XYZ+XYZ+XYZ’+X’YZ+X’YZ+X’Y’Z	rule 10
=	XYZ+XYZ’+X’YZ+X’Y’Z	rule 3
=	XY(Z+Z’)+X’Z(Y+Y’)	rule 14
=	XY·1+X’Z·1	rule 4
=	XY+X’Z	rule 6

Does XY+YZ+X'Z=XY+X'Z ?

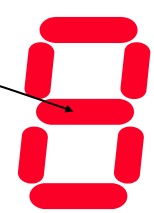
(done another way)

	$XY+YZ+X'Z$	
=	$XY+YZ\cdot 1+X'Z$	rule 6
=	$XY+YZ(X+X')+X'Z$	rule 4
=	$XY+YZX+YZX'+X'Z$	rule 14
=	$XY+XYZ+X'ZY+X'Z$	rule 11
=	$XY+XYZ+X'Z+X'ZY$	rule 10
=	$XY\cdot 1+XYZ+X'Z\cdot 1+X'ZY$	rule 6
=	$XY(1+Z)+X'Z(1+Y)$	rule 14
=	$XY\cdot 1+X'Z\cdot 1$	rule 2
=	$XY+X'Z$	rule 6

rule 15 absorption

Don't Care

- BCD code encodes a decimal digit using 4 bits
  - 0_{10} as 0000_2; 1_{10} as 0001_2; …….; 9_{10} as 1001_2
  - the patterns 1010_2 ~ 1111_2 are not used
- Suppose F controls the on/off of this segment of a 7-segment display



  F= { 1 for 0010,0011,0100,0101,0110,1000,1001
  0 for 0000,0001,0111
  "don't care" for 1010 ~ 1111

- In a K-map, "don't care" can have cake and eat it too
  - treat them as 1s to enable larger cube covers
  - but also okay to leave uncovered (like 0s)

4-Variable Example with Don't Care

B

0	1	1	x
0	1	1	x
1	0	x	x
1	1	x	x

D

C

A

F= BC'
+CD'
+B'C

F=
1 for ABCD=0010,0011,0100,0101,0110,1000,1001
0 for ABCD=0000,0001,0111
"don't care" for ABCD=1010 ~ 1111?

Tricky K-map Example

B

0	0	1	0
1	1	1	0
0	1	1	1
0	1	0	0

D

C

A

Following the simple K-map recipe will result in a redundant cover in this example.

Useful K-map Conventions

- Think of min-terms in a truth table like binary numbers
- With N inputs, count from 0 to 2^N-1
- For example, N=4

W	X	Y	Z	corresponding minterm
0	0	0	0	W'X'Y'Z'
0	0	0	1	W'X'Y'Z
0	0	1	0	W'X'YZ'
0	0	1	1	W'X'YZ
⋮				⋮
1	1	0	1	WXY'Z
1	1	1	0	WXYZ'
1	1	1	1	WXYZ

Useful K-map Conventions

- Consistency saves time and reduces errors, e.g., N=4
  - vary the 2 more significant bits (WX) left-and-right
  - vary the 2 less significant bits (YZ) up-and-down.
- Transcribing from T-table to K-map by Gray counting
  - the neighboring columns (or rows) differ by only 1 bit
  - 00->01->11->10 (and not 00->01->10->11)

W (over columns WX=11, WX=10); X (under columns WX=01, WX=11); Y (beside rows YZ=11, YZ=10); Z (beside rows YZ=01, YZ=11)

	WX=00	WX=01	WX=11	WX=10
YZ=00	0000	0100	1100	1000
YZ=01	0001	0101	1101	1001
YZ=11	0011	0111	1111	1011
YZ=10	0010	0110	1110	1010

	WX=00	WX=01	WX=11	WX=10
YZ=00	W'X'Y'Z'	W'XY'Z'	WXY'Z'	WX'Y'Z'
YZ=01	W'X'Y'Z	W'XY'Z	WXY'Z	WX'Y'Z
YZ=11	W'X'YZ	W'XYZ	WXYZ	WX'YZ
YZ=10	W'X'YZ'	W'XYZ'	WXYZ'	WX'YZ'