CpE358/CS381

Switching Theory and Logical Design

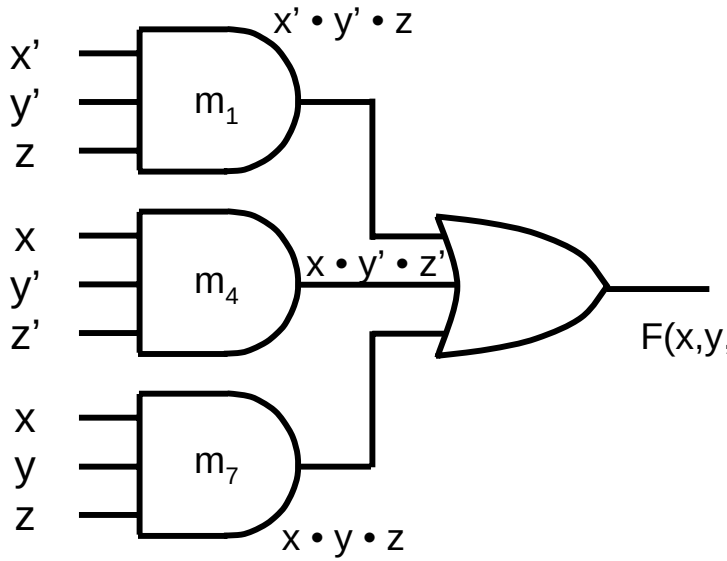
Class 3

Today

- Fundamental concepts of digital systems (Mano Chapter 1)
- Binary codes, number systems, and arithmetic (Ch 1)
- Boolean algebra (Ch 2)
- Simplification of switching equations (Ch 3)
- Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
- Combinatoric logical design including LSI implementation (Chapter 4)
- Hazards, Races, and time related issues in digital design (Ch 9)
- Flip-flops and state memory elements (Ch 5)
- Sequential logic analysis and design (Ch 5)
- Synchronous vs. asynchronous design (Ch 9)
- Counters, shift register circuits (Ch 6)
- Memory and Programmable logic (Ch 7)
- Minimization of sequential systems
- Introduction to Finite Automata

Boolean Functions in Terms of Minterms

- A logical function is TRUE if any of it's minterms are true:



$F(x,y,z) = \Sigma(1,4,7)$
$= x' \cdot y' \cdot z + x \cdot y' \cdot z' + x \cdot y \cdot z$
$= m_1 + m_4 + m_7$

- Algebraic manipulation of the literal expression of the function is one way to minimize it, manipulation of minterms is another

Two Variable Minterm Map

- Represent Boolean functions in terms of minterms in a Karnaugh map:

m_0	m_1
m_2	m_3

x \ y	0	$\overline{y}$ (1)
0	x'y'	x'y
1	xy'	xy

Two Variable Minterm Map

- Represent Boolean functions in terms of a Karnaugh map:

m_0	m_1
m_2	m_3

x \ y	0	$\overline{y}$
0	x'y'	x'y
1	xy'	xy

- Consider the XOR function

$$F(x, y) = x \oplus y = x'y + xy' = m_1 + m_2$$

x \ y	0	$\overline{y}$
0		1
1	1	

Two Variable Minterm Map

- Represent Boolean functions in terms of a Karnaugh map:

m_0	m_1
m_2	m_3

x \ y	0	1
0	x'y'	x'y
1	xy'	xy

- Consider the XOR function

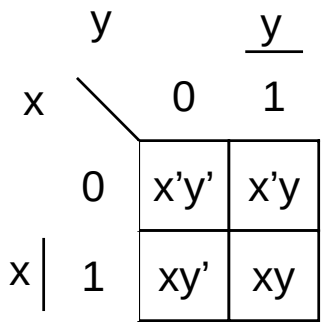
$$F(x, y) = x \oplus y = x'y + xy' = m_1 + m_2$$

x \ y	0	1
0		1
1	1	

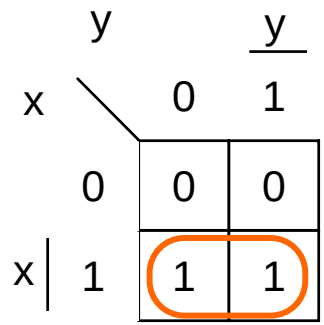
x \ y	0	1
0	0	1
1	1	0

Set the non-asserted minterms to zero

Minimizing Function of Two Variables



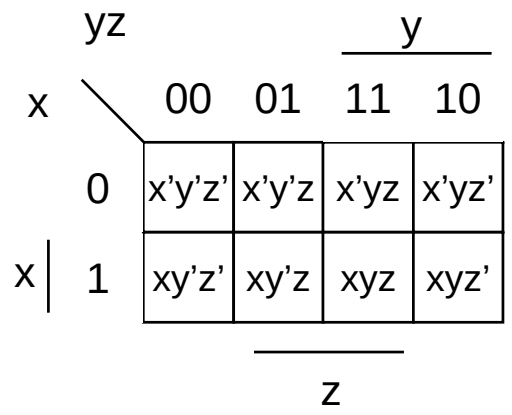
$$F(x,y) = xy' + xy = m_2 + m_3$$
$$= x(y' + y)$$
$$= x$$



- Covering "adjacent" minterms with a single region defines the variables needed to represent the function

Minimizing Function of Three Variables

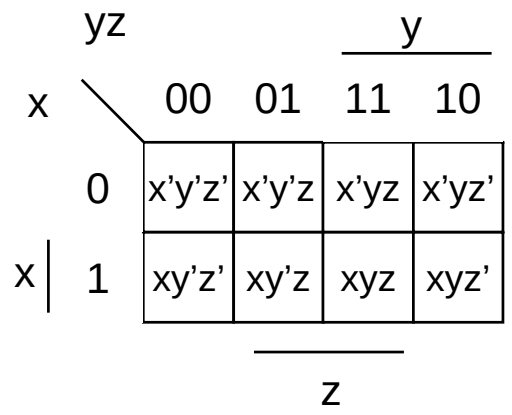
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



- Minterms are numbered in Gray code order – adjacent minterms differ in only one variable
- If the function is asserted (i.e., TRUE) for ***both*** of these adjacent minterms, then the terms defined by those minterms do not depend on the variable that is changing between them

Three Variable Map

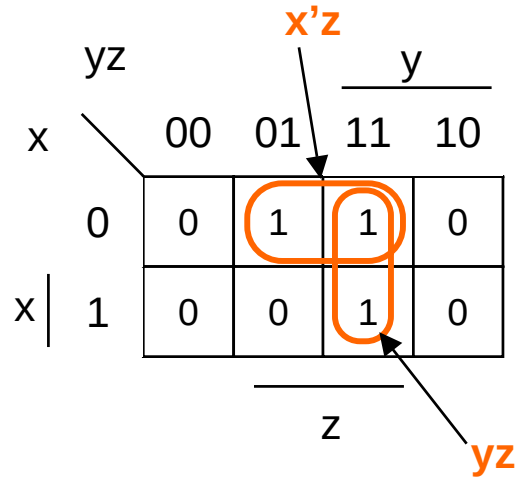
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



- Consider F(x,y,z)=Σ(1,3,7)

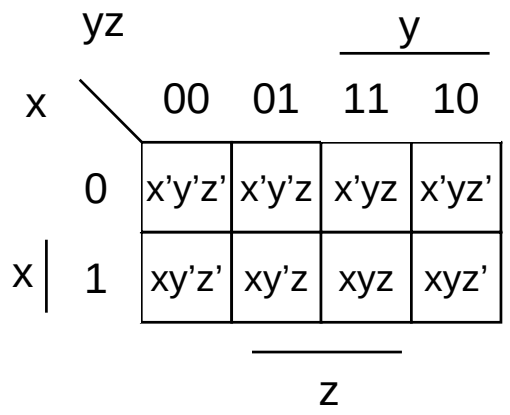
$$F(x,y,z) = m_1 + m_3 + m_7$$

$$\begin{aligned} F(x,y,z) &= x'y'z + x'yz + xyz \\ &= x'y'z + x'yz + x'yz + xyz \\ &= x'(y'+y)z + (x'+x)yz \\ &= x'z + yz \end{aligned}$$



Three Variable Map

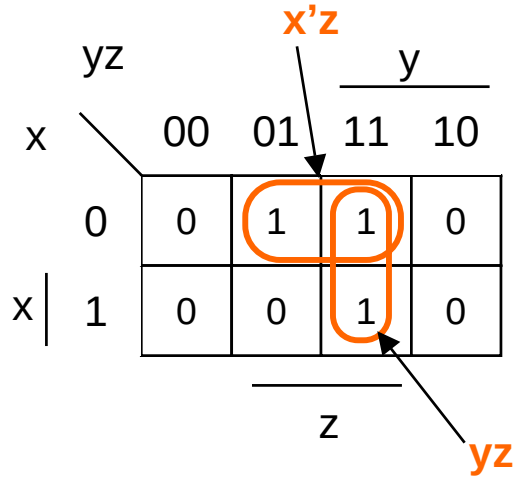
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



- Observations:
  - All minterms must be covered
  - Number of variables defining a sum term inversely proportional to number of minterms covered
  - Number of sum terms required to define function equal to number of separate regions

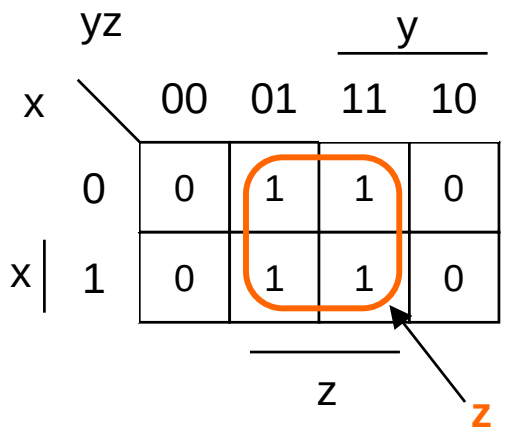
    → Maximize region size

    → Minimize number of regions

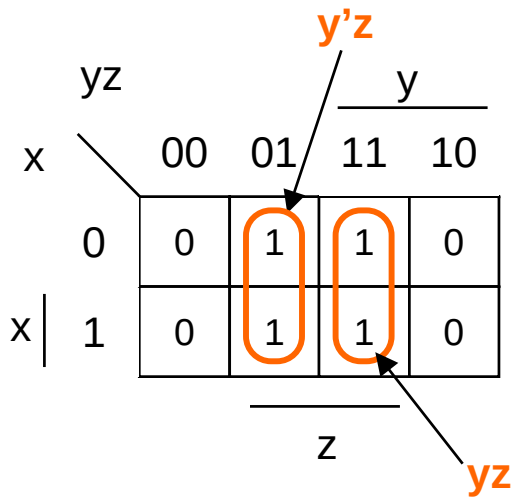


Three Variable Map

- For a 3-variable map:
  - Covering 4 minterms with one 4-minterm region defines the function in terms of a single variable

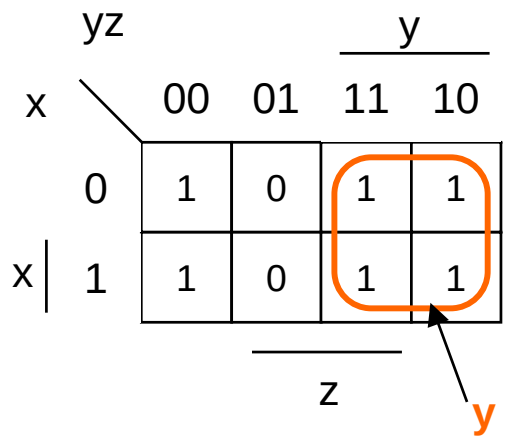


  - Covering the same 4 minterms with 2 2-minterm regions defines the function in terms of two terms, each requiring two variables.



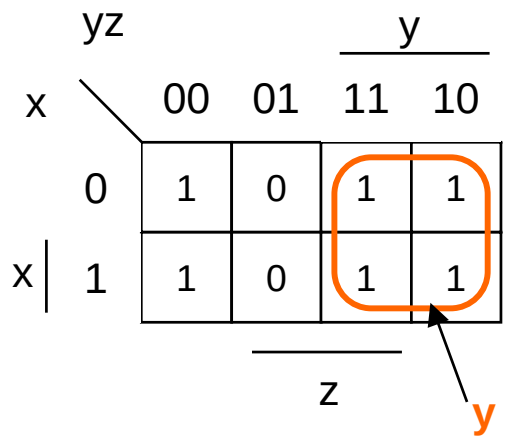
Three Variable Map

- "Adjacency" sometimes exists in subtle ways:
  - These four minterms are obviously adjacent to each other.

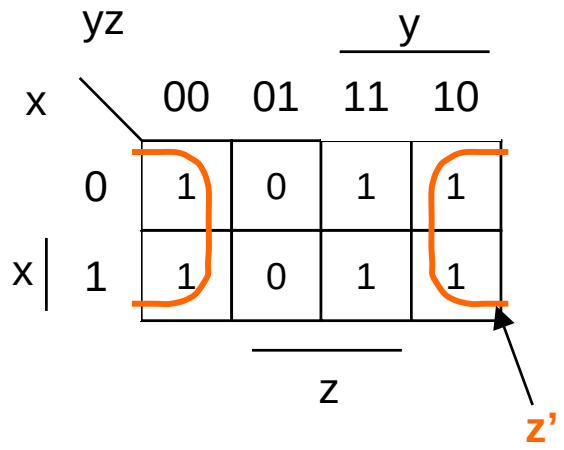


Three Variable Map

- "Adjacency" sometimes exists in subtle ways:
  - These four minterms are obviously adjacent to each other.

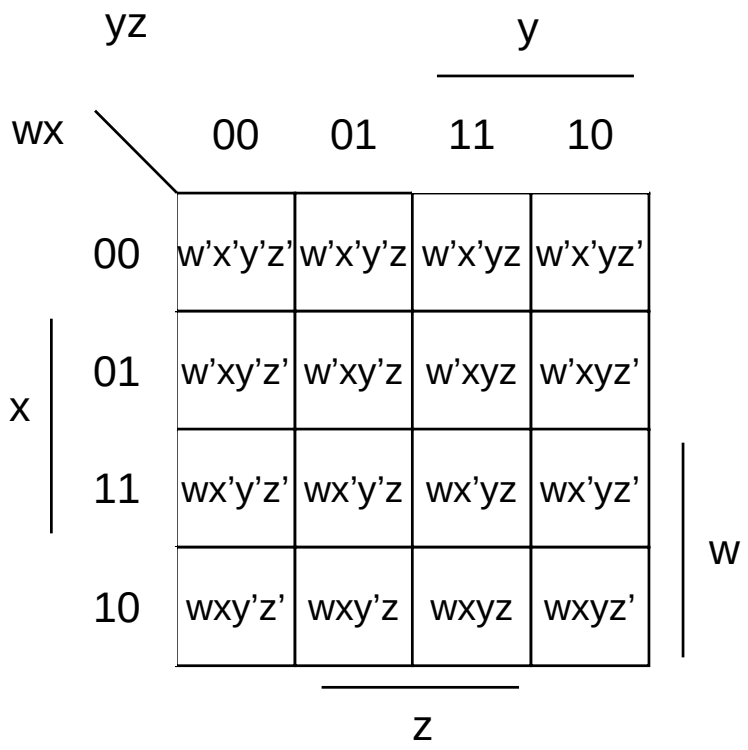


  - But so are these, if we consider the map to wrap around on itself



Four Variable Map

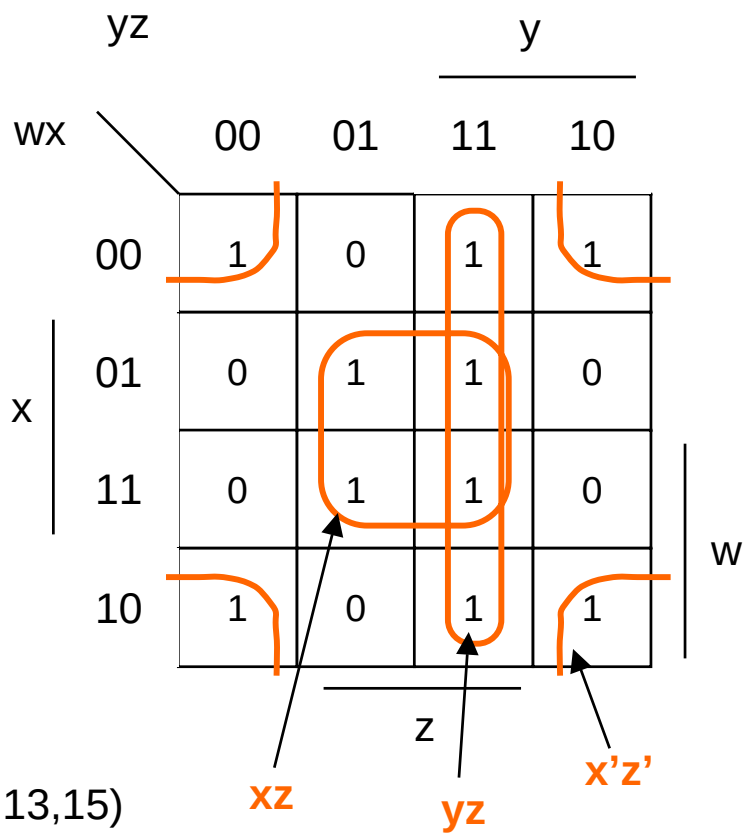
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



- The 4-variable map extends the concept of the 2- and 3-variable map

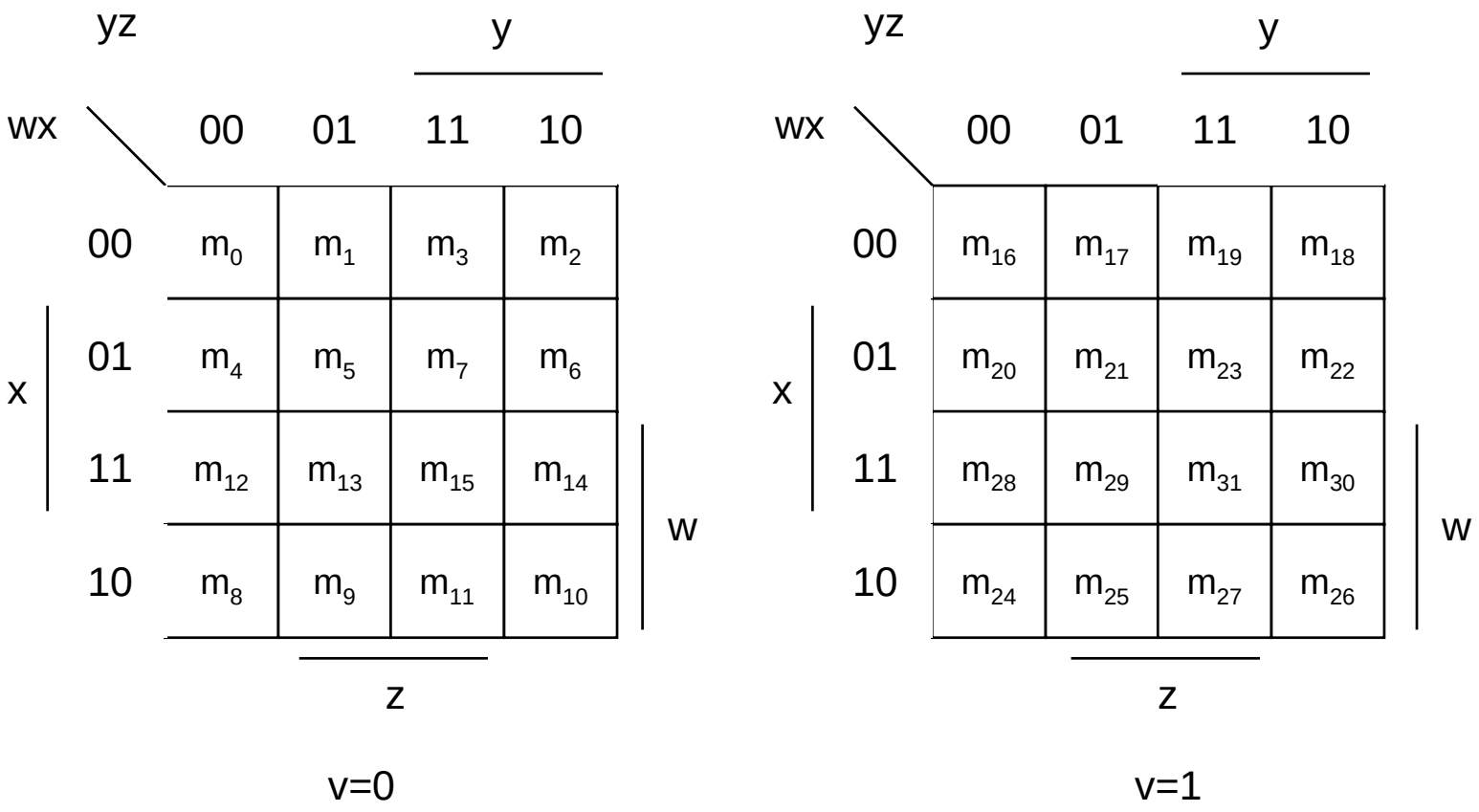
Minimizing Four Variable Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



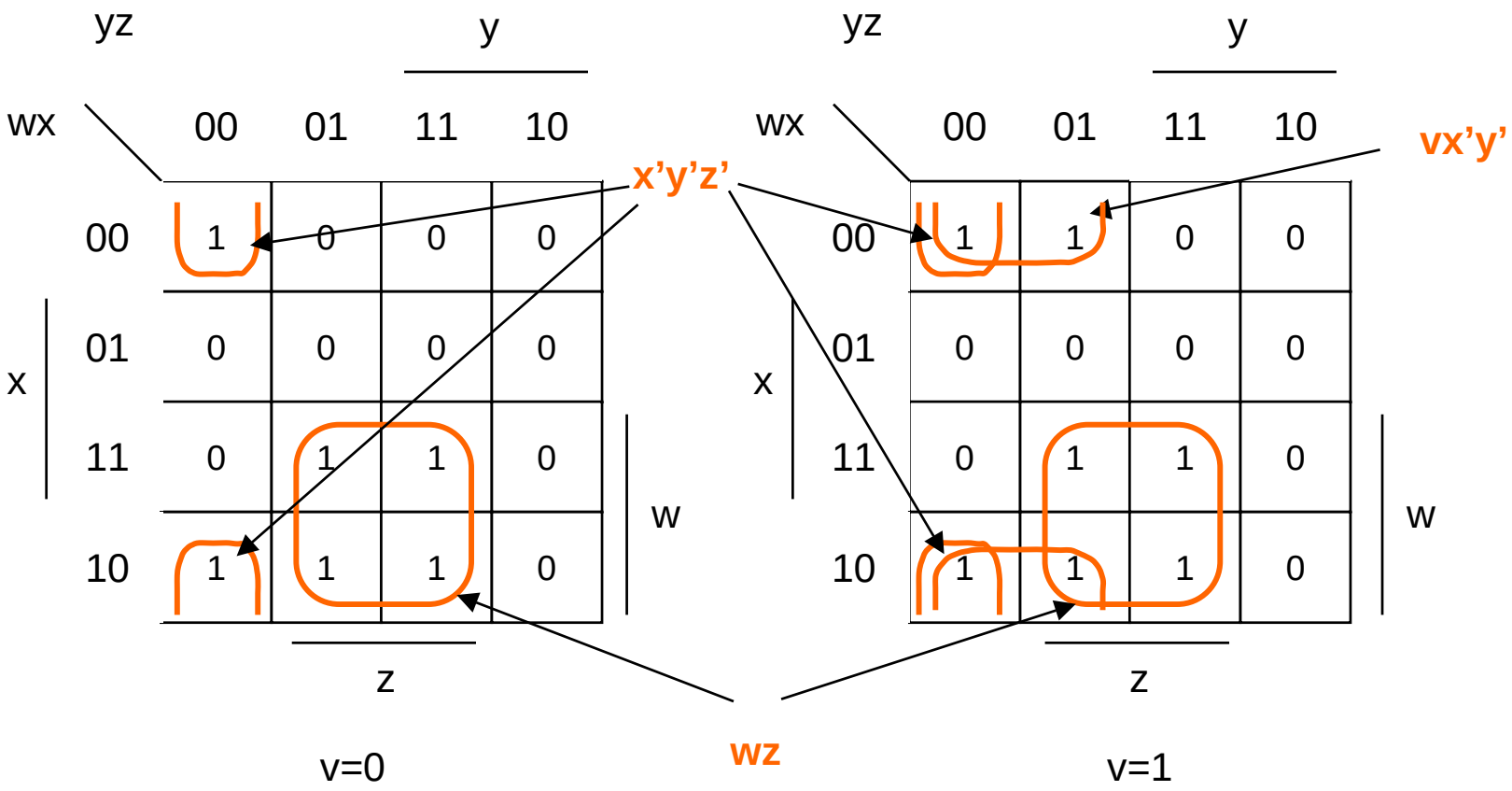
- Minimize F(w,x,y,z)=Σ(0,2,3,5,7,8,10,11,13,15)
- F(w,x,y,z)=xz+x'z'+yz

Five Variable Map



- 5-variable map is extension of 4-variable map, adjacency must be considered between pairs of 4-variable maps

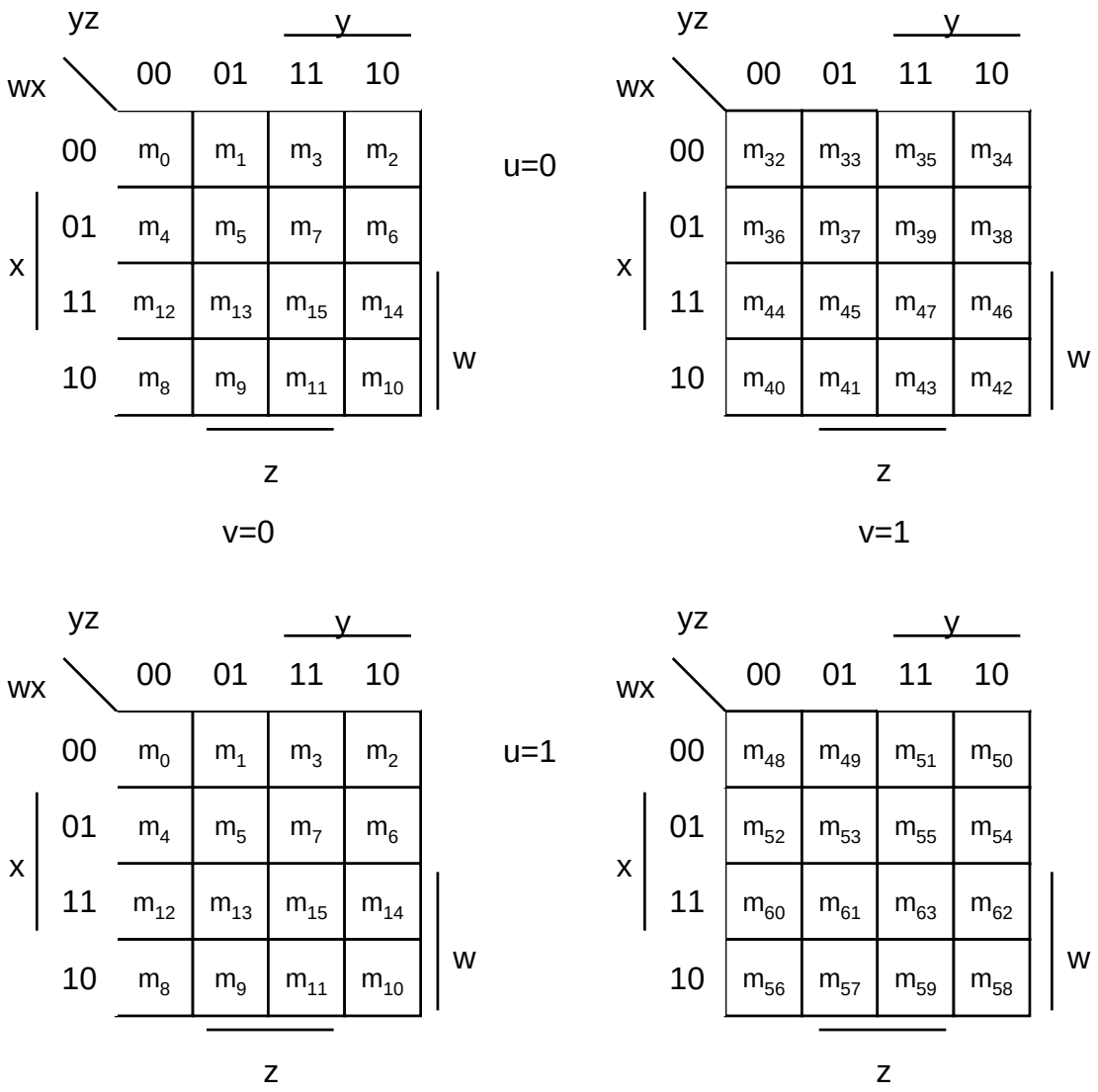
Minimizing Five Variable Map



- Minimize F(v,w,x,y,z)=Σ(0,8,9,11,16,17,24,25,27,29,31)
- F(v,w,x,y,z)=wz+x'y'z'+vx'y'

Six Variable Map

- Keeping track of what minterms are adjacent becomes tedious
- Ensuring the maximum coverage for each term is challenging
- 6-variable maps usable, but perhaps the design needs to be modularized instead



Product of Sums – Covering 0's Instead of 1's

- A function of N variables, $F(v_1,v_2,\ldots,v_N)$, can be represented by a Karnaugh map with 2^N cells. $(v_1,v_2,\ldots,v_N)$ = (0,0,…0), (0,1,…0), …, (1,1,…,1)
- F( ), and it's Karnaugh map have K minterms (1's) and 2^N-K maxterms (0's)
- If $K > 2^N$-K, it might be easier to cover the maxterms rather than the minterms. E.g.:

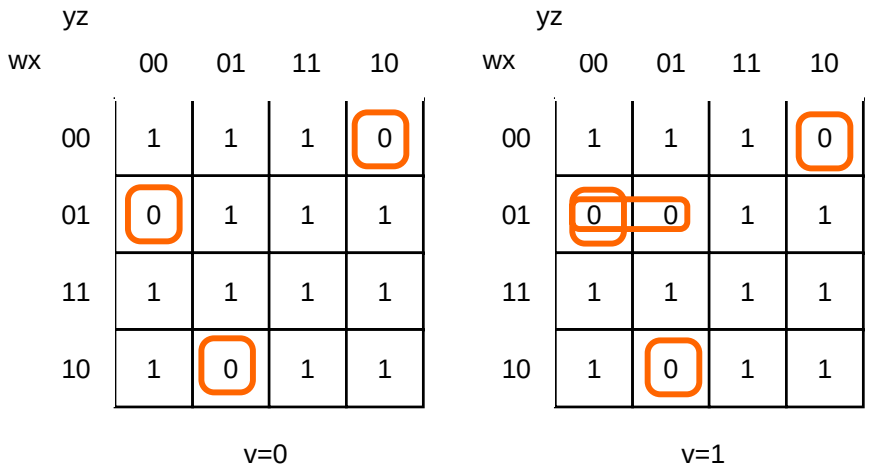
wx \ yz	00	01	11	10
00	1	1	1	0
01	0	1	1	1
11	1	1	1	1
10	1	0	1	1

v=0

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

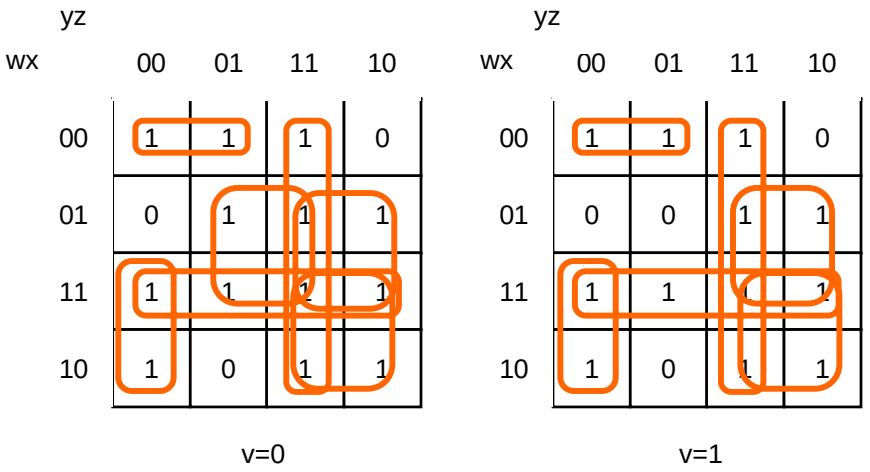
v=1

Product of Sums – Covering 0's Instead of 1's



F(v,w,x,y,z)' has 4 terms

F(v,w,x,y,z)' = w'x'yz' + w'xy'z' + vw'xy' +wx'y'z

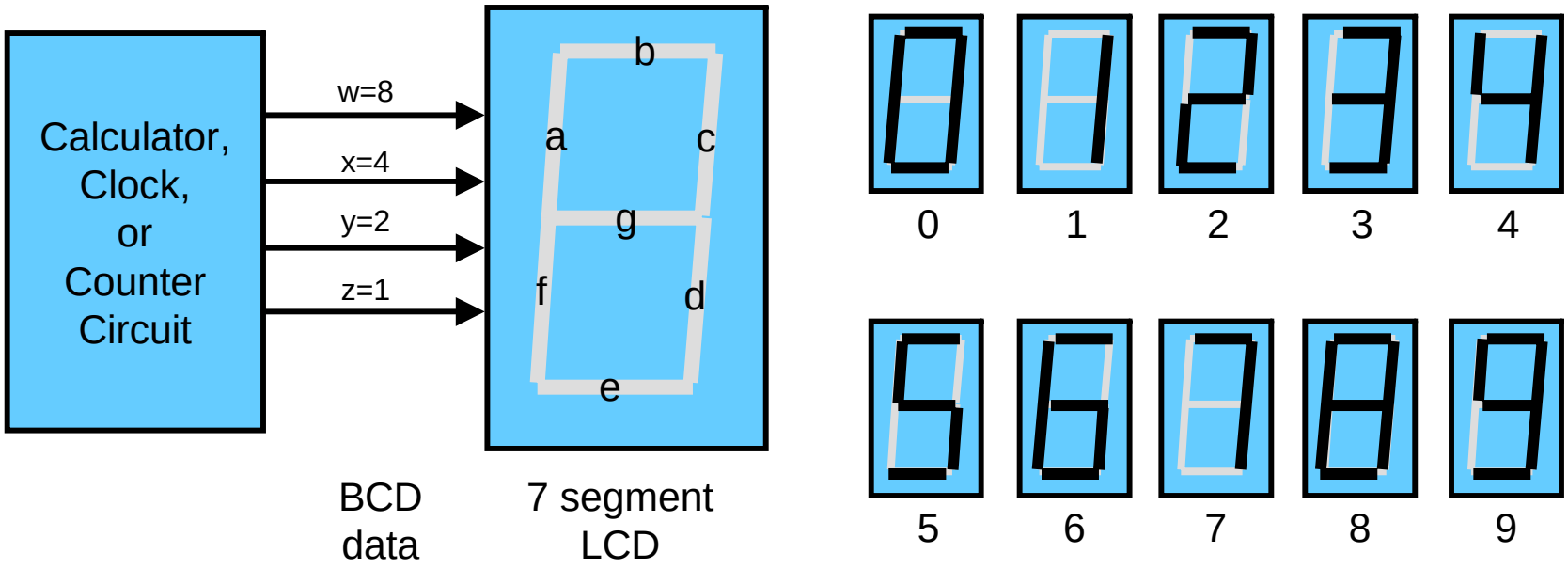


F(v,w,x,y,z) has 7 terms

F(v,w,x,y,z) = w'x'y' + yz + v'xz + xy + wx + wy'z' + wy

Don’t Care Conditions

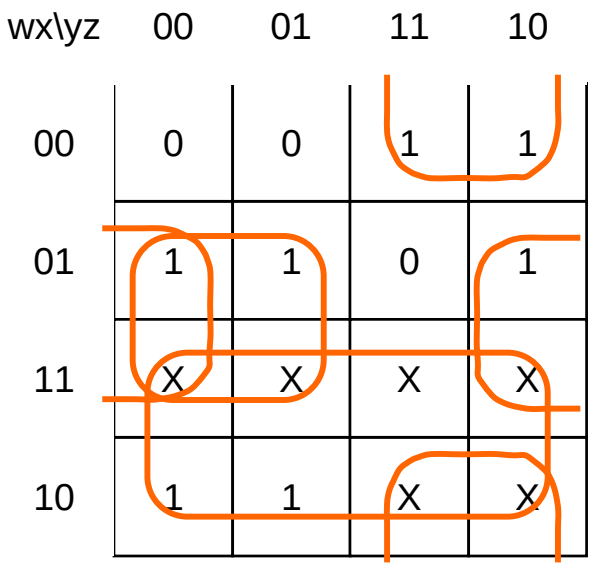
- Sometimes, not all possible output values are specified in system design, e.g.:



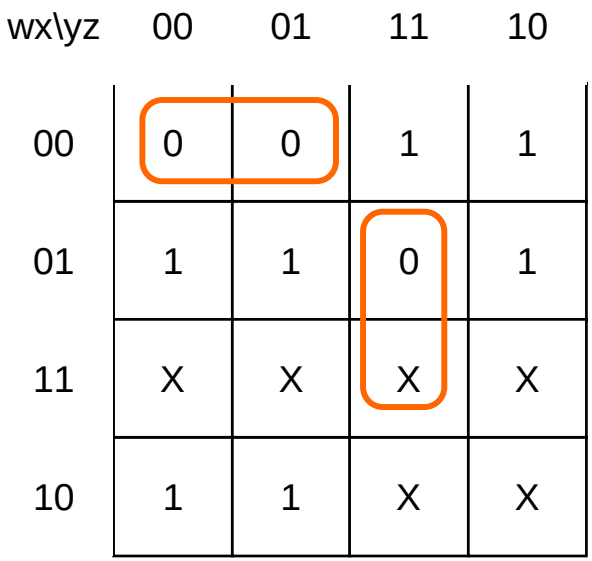
- Consider the horizontal line in the middle of the display (segment g):

  $F_g(w,x,y,z)=\Sigma(2,3,4,5,6,8,9)$, but we don’t care what happens to minterms 10, 11, 12, 13, 14, or 15, since the display will not be sent those states

Don’t Care Conditions



F(w,x,y,z) = xy + xz’ + xy’ + w



F’(w,x,y,z) = w’x’y’ + xyz

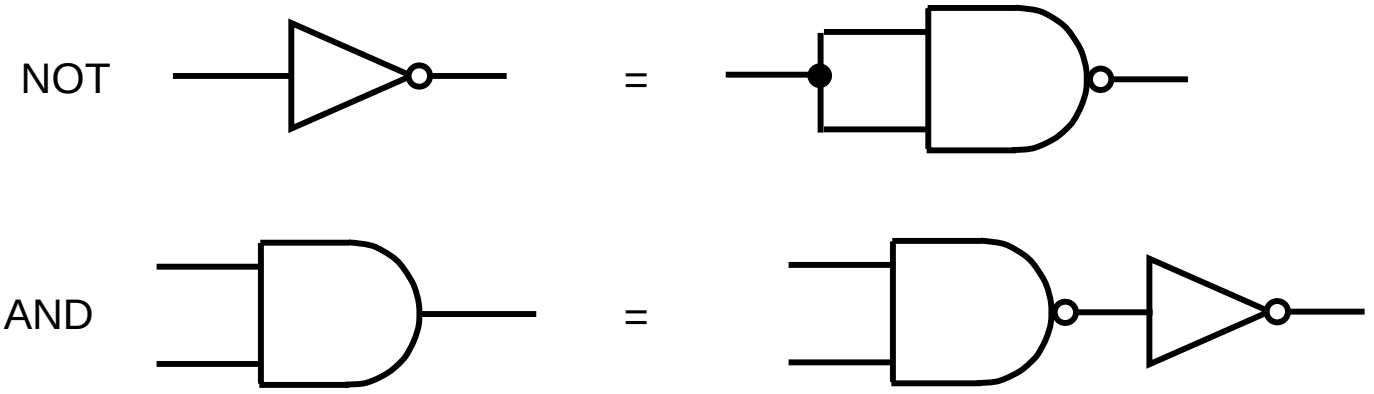
- We are free to assign whatever values we want to for minterms 10, 11, 12, 13, 14, and 15. Assign them a value X to indicate they may be covered, or not, whichever results in the simplest expression

Logical Completeness

1. AND, OR, NOT can implement any Boolean function – They form a "Logically Complete" set of operators

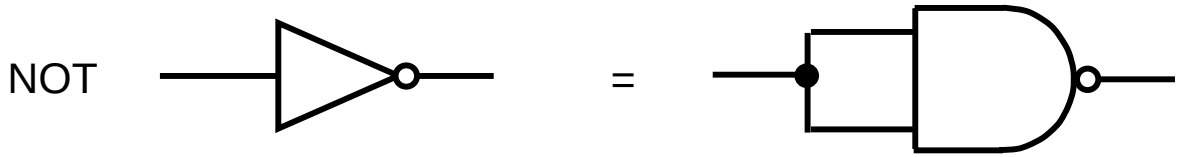
Logical Completeness

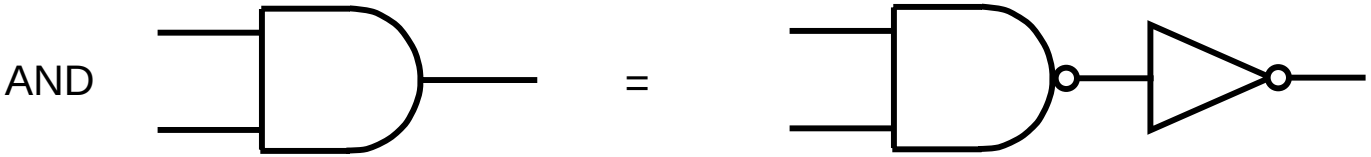
1. AND, OR, NOT can implement any Boolean function – They form a "Logically Complete" set of operators
2. NAND can implement AND and NOT directly:



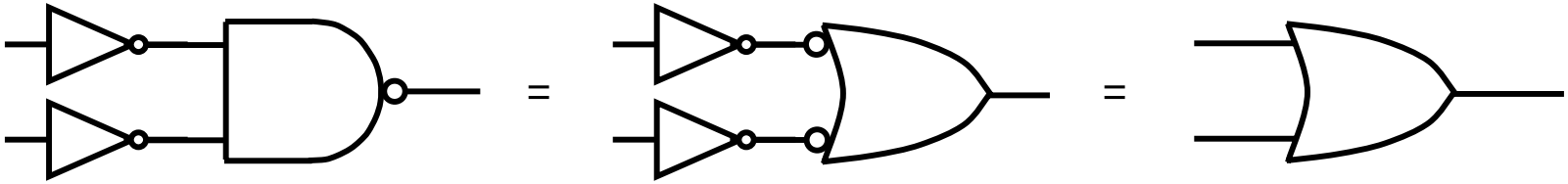
Logical Completeness

1. AND, OR, NOT can implement any Boolean function – They form a "Logically Complete" set of operators
2. NAND can implement AND and NOT directly:





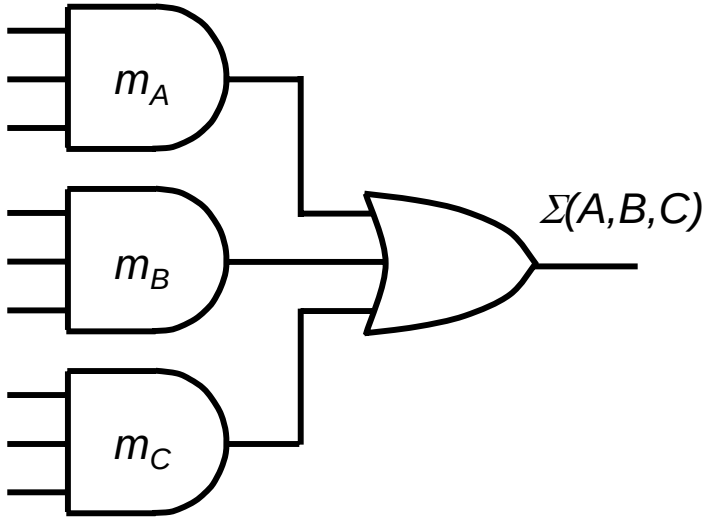
3. NAND can implement OR by DeMorgan's Law:



$\rightarrow$ NAND is logically complete (so is NOR)

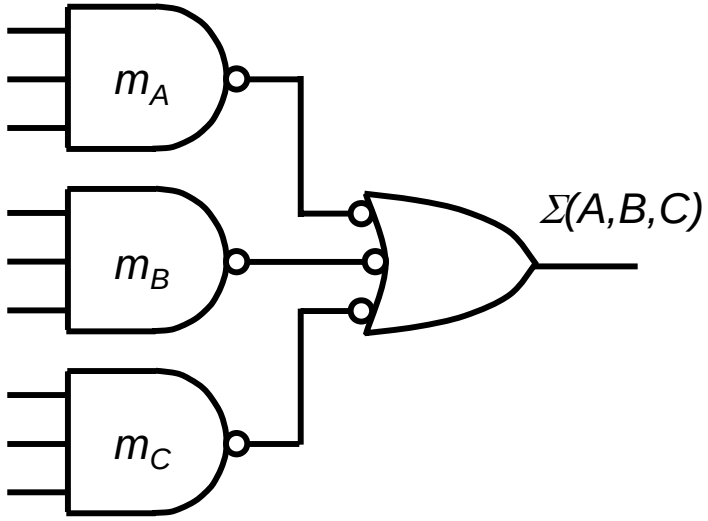
NAND Implementation of Sum of Products

- Consider an arbitrary Sum of Products:



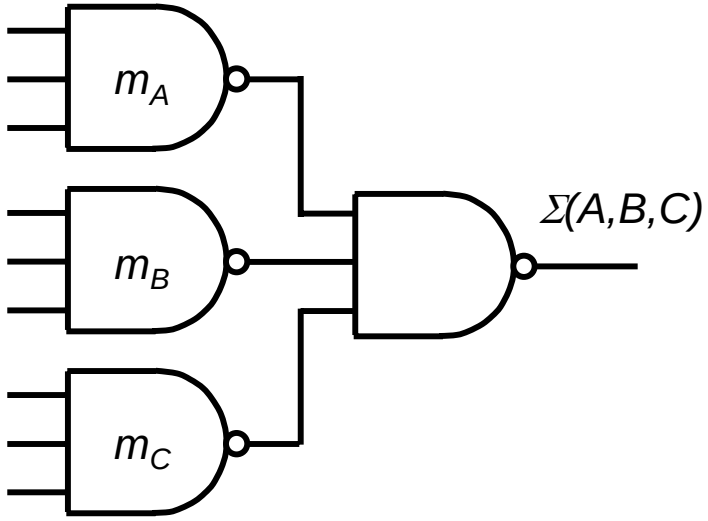
NAND Implementation of Sum of Products

- Consider an arbitrary Sum of Products:
- Add inversions at each term. This is allowed, since (x')'=x



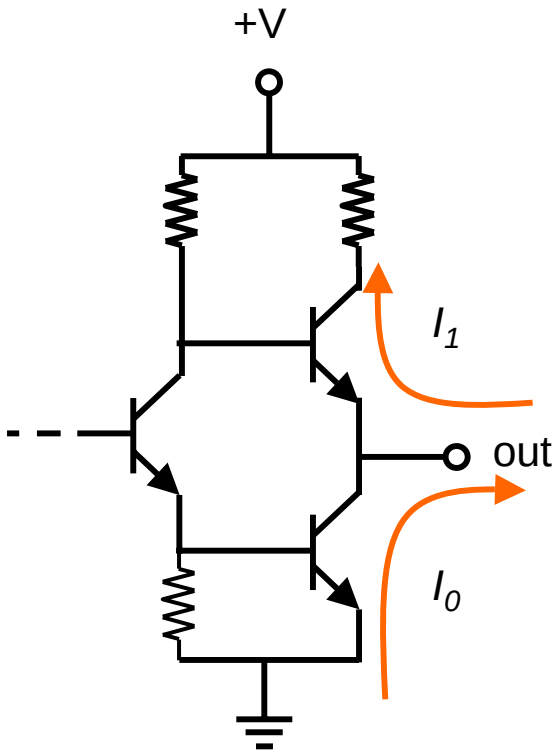
NAND Implementation of Sum of Products

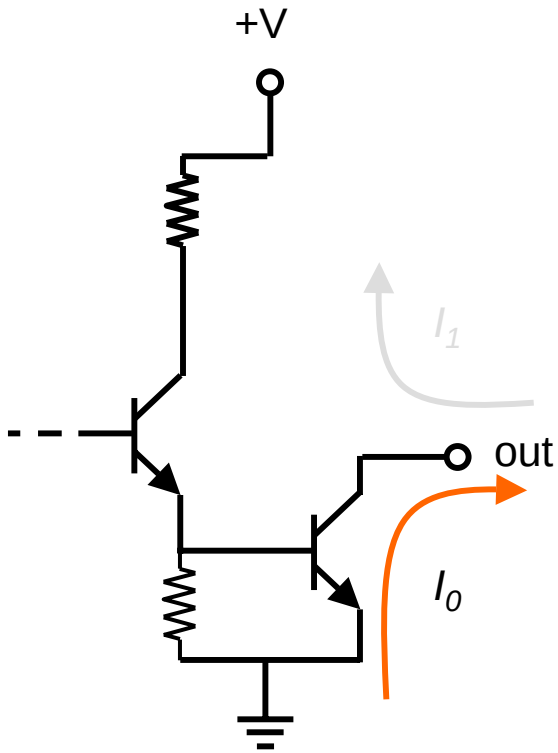
- Consider an arbitrary Sum of Products:
- Add inversions at each term. This is allowed, since (x')'=x
- Convert output gate by DeMorgan's Law:



Wired-AND and Open Collector

- Typical TTL “totem-pole” output circuit:

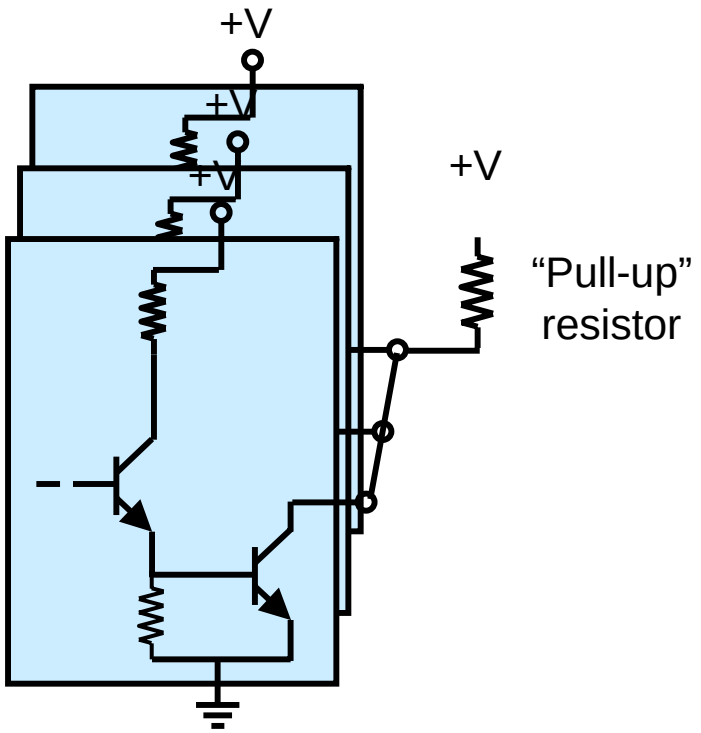


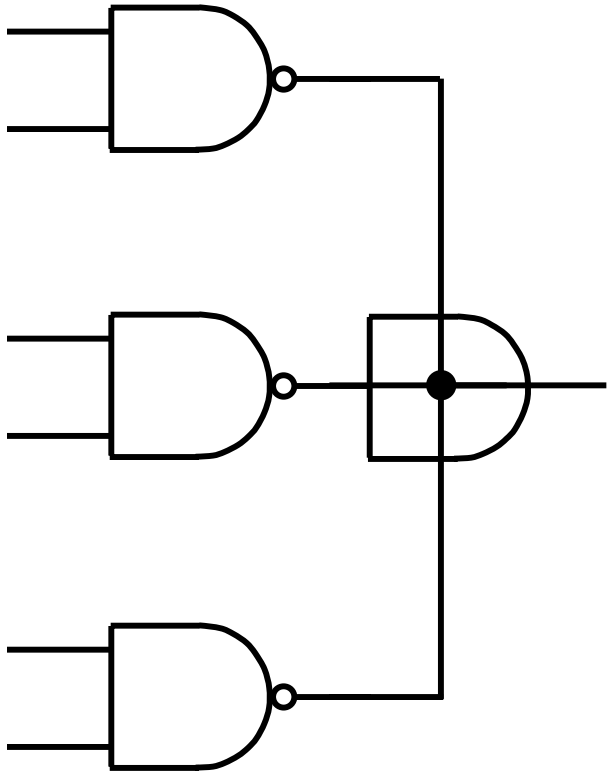


- TTL with Open Collector output circuit

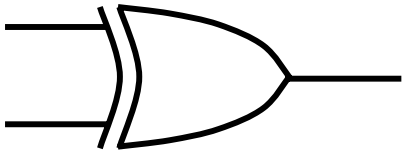
Wired-AND and Open Collector

- Each gate asserts 0-output – with no pull-up transistor, no gate can cause output to become “1.” External “pull-up” resistor needed
- Used for wiring multiple devices together on bus, but speed is limited





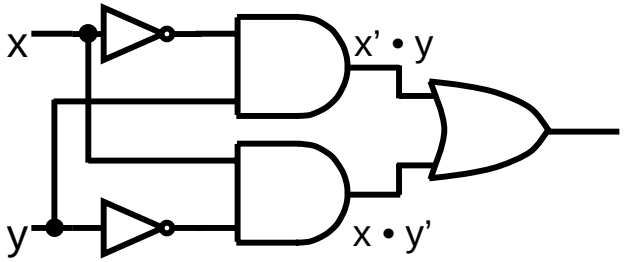
XOR Function

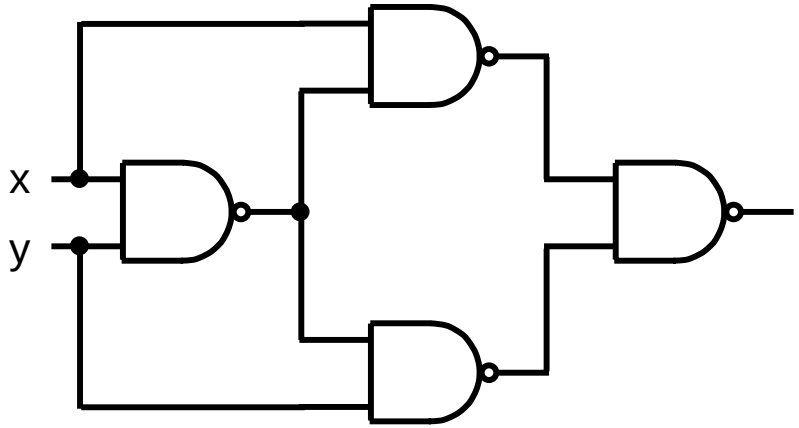


Exclusive OR
(XOR)

⊕	x = 0	x = 1
y = 0	0	1
y = 1	1	0

$$x \oplus y = x' \bullet y + x \bullet y'$$

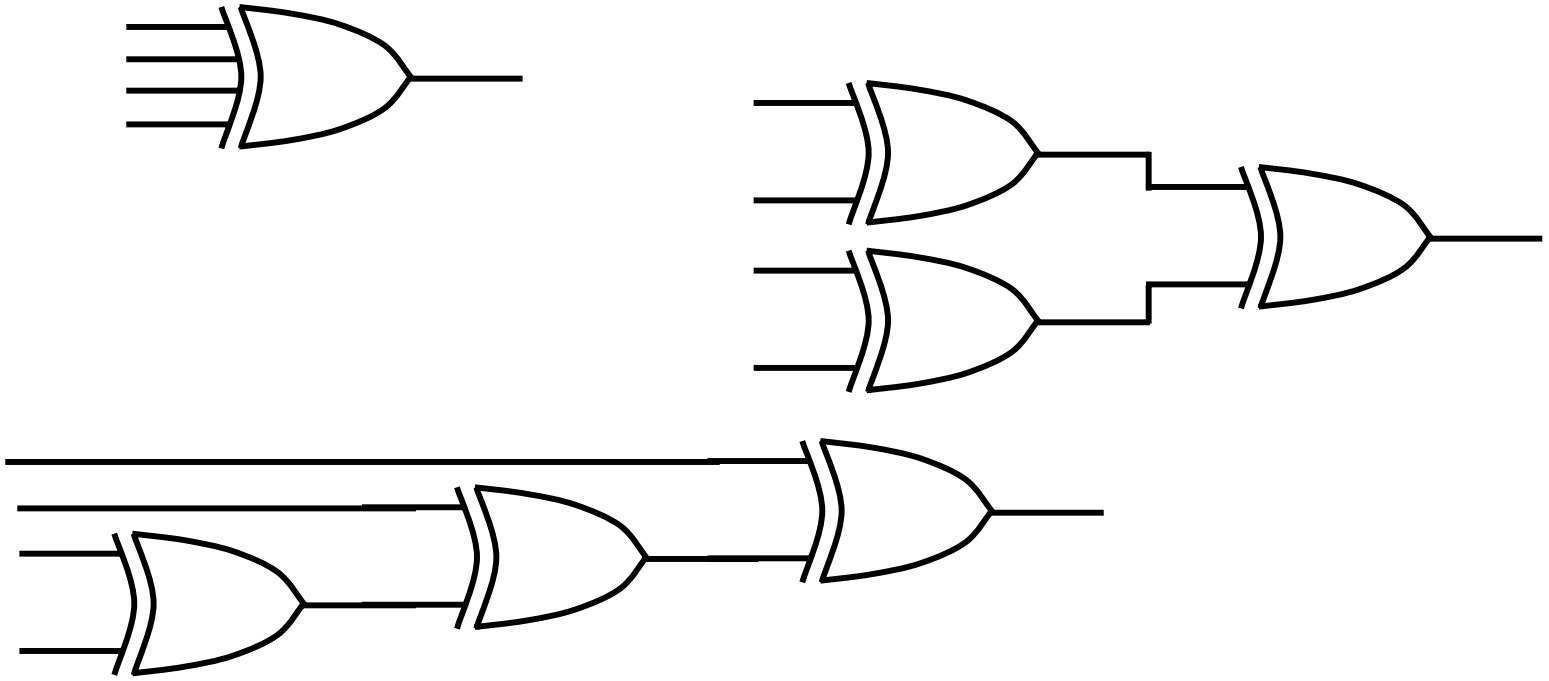




$$w \oplus x \oplus y \oplus z = ((w \oplus x) \oplus y) \oplus z = (w \oplus x) \oplus (y \oplus z)$$

- XOR applications:
  - Addition, parity, data scramblers, encryption, shift register sequences

N-input XOR

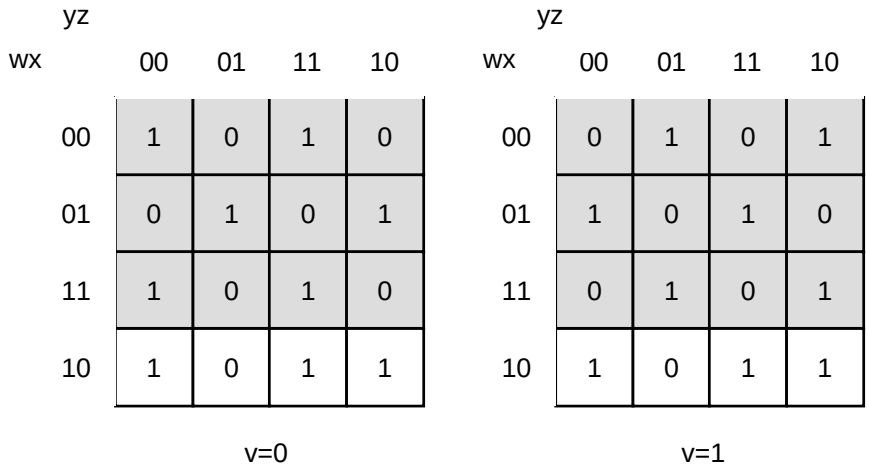


$$w \oplus x \oplus y \oplus z = ((w \oplus x) \oplus y) \oplus z = (w \oplus x) \oplus (y \oplus z)$$

- These three designs are all logically equivalent (for ***static*** signals)

Hard To Minimize Functions

- Consider this map:



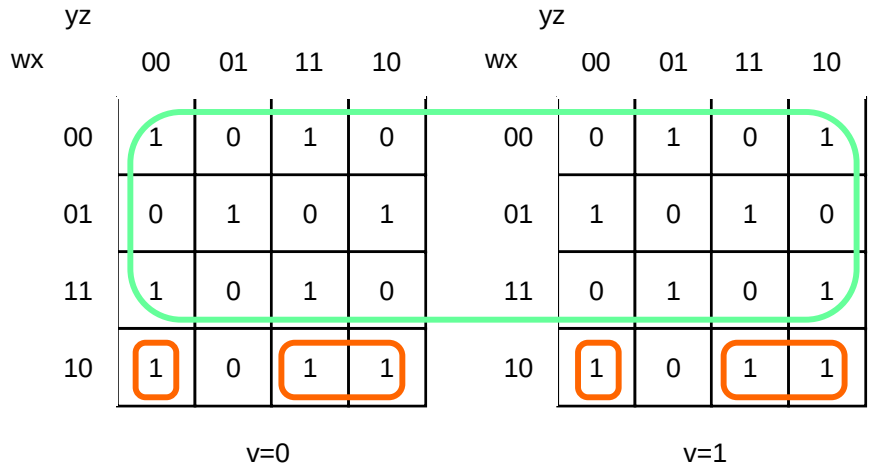
Isolated minterms cannot be grouped

Hard To Minimize Functions

- Consider this map:

Cover part of the map with XOR

Treat the rest normally



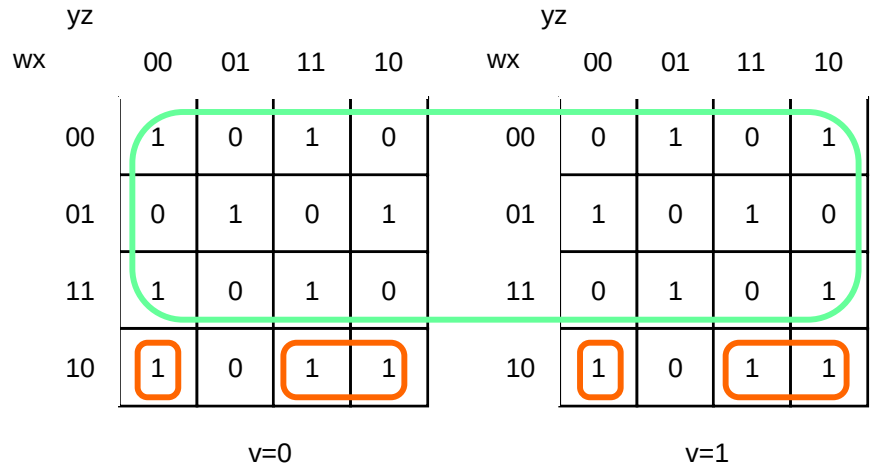
$$F(v,w,x,y,z) = (wx')'\bullet(v \oplus w \oplus x \oplus y \oplus z) + wx'y'z' + wx'y$$

Hard To Minimize Functions

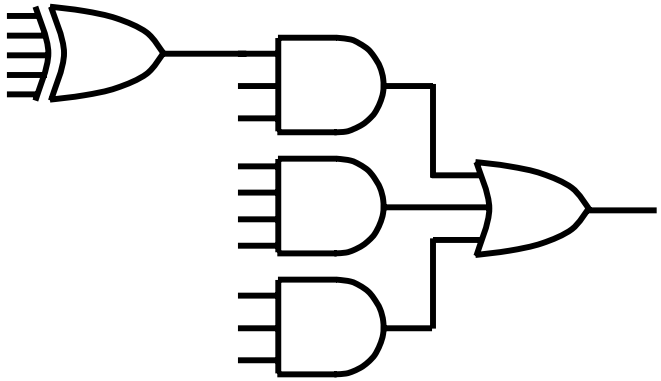
- Consider this map:

Cover part of the map with XOR

Treat the rest normally



$$F(v,w,x,y,z) = (wx')'\bullet(v \oplus w \oplus x \oplus y \oplus z) + wx'y'z' + wx'y$$



Summary

- Fundamental concepts of digital systems (Mano Chapter 1)
- Binary codes, number systems, and arithmetic (Ch 1)
- Boolean algebra (Ch 2)
- Simplification of switching equations (Ch 3)
- Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
- Combinatoric logical design including LSI implementation (Chapter 4)
- Hazards, Races, and time related issues in digital design (Ch 9)
- Flip-flops and state memory elements (Ch 5)
- Sequential logic analysis and design (Ch 5)
- Synchronous vs. asynchronous design (Ch 9)
- Counters, shift register circuits (Ch 6)
- Memory and Programmable logic (Ch 7)
- Minimization of sequential systems
- Introduction to Finite Automata

Homework 3 – due in Class 5

As always, show all work:

- Problems 3-5, 3-7, 3-18.
- Design a BCD to seven segment decoder for any 2 of the 6 segments (a-f) we did not discuss in class.