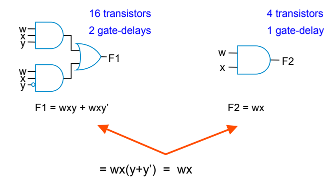
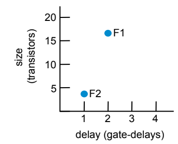
ECE 474A/57A
Computer-Aided Logic Design

Logic Optimization 1

Logic Optimization

- 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
  - Size – the number of transistors
- Assumption
  - Every gate has delay of "1 gate-delay"
  - Every gate *input* requires 2 transistors
  - Ignore inverters





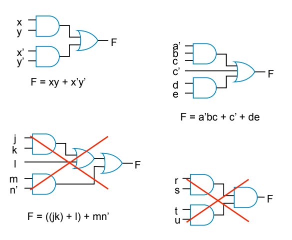
Transforming F1 to F2 represents an ***optimization***: Better in all criteria of interest

---

Two-level Logic Optimization

- Two-level logic
  - Circuit with only two levels (ORed AND gates)
- Basically sum-of-products form
  - An equation written as an ORing of product terms

Are these two-level logic?

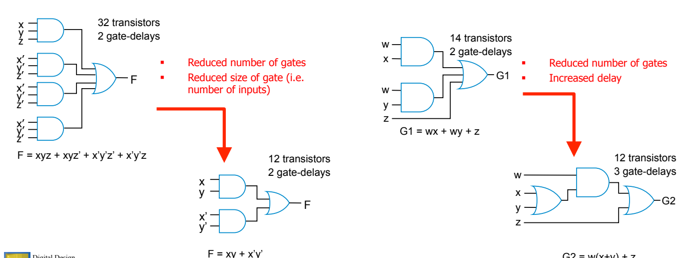


technically yes, but not what we mean in terms of logic minimization

---

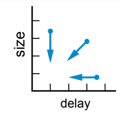
Optimization vs. Tradeoff

- Optimization - Defined as better in all criteria of interest
  - Delay and size - we consider size minimization only (2-level logic only)
  - In reality requires a balance of many criteria metrics
    - Cost, reliability, time-to-market, etc...
- Tradeoff - Improves some, but worsens other, criteria of interest

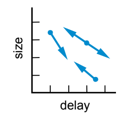


Pareto Points

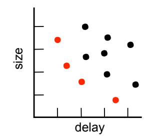
- We obviously prefer optimizations, but often must accept tradeoffs
  - You can't build a car that is the most comfortable, and has the best fuel efficiency, and is the fastest – you have to give up something to gain other things
- Many options in solution space
  - Pareto point
    - Point in solution space in which no other point better in all metrics
    - Shown in red
    - Pareto points yield the trade-off curve



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



Tradeoffs
Some criteria of interest are improved, while others are worsened

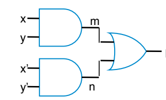


Combinational Logic Optimization and Tradeoffs

- Two-level size optimization using algebraic methods
  - Goal: circuit with only two levels (ORed AND gates), with minimum transistors
    - Though transistors getting cheaper (Moore's Law), they still cost something
- Define problem algebraically
  - Sum-of-products yields two levels
    - F = abc + abc' is sum-of-products; G = w(xy + z) is not.
  - Transform sum-of-products equation to have fewest literals and terms
    - Each literal and term translates to a gate input, each of which translates to about 2 transistors
    - Ignore inverters for simplicity

Example

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'



= 6 gate inputs * 2 transistor/input
= 12 transistors

Boolean Algebra

- How do we use Boolean algebra to obtain fewest literals and terms?

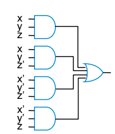
1a. 0 · 0 = 0
1b. 1 + 1 = 1
2a. 1 · 1 = 1
2b. 0 + 0 = 0
3a. 0 · 1 = 1 · 0 = 0
3b. 0 + 1 = 1 + 0 = 1
4a. If x = 0, then x' = 1
4b. If x = 1, then x' = 0
5a. x · 0 = 0
5b. x + 1 = 1
6a. x · 1 = x
6b. x + 0 = x
7a. x · x = x
7b. x + x = x
8a. x · x' = 0
8b. x + x' = 1
9. x'' = x

10a.	**x · y = y · x**	***(Commutative)***
10b.	**x + y = y + x**	
11a.	**x · (y · z) = (x · y) · z**	***(Associative)***
11b.	**x + (y + z) = (x + y) + z**	
12a.	**x · (y + z) = x · y + x · z**	***(Distributive)***
12b.	**x + (y · z) = (x + y) · (x + z)**	
13a.	**x + x · y = x**	***(Absorption)***
13b.	**x · (x + y) = x**	
14a.	**x · y + x · y' = x**	***(Combining)***
14b.	**(x + y) · (x + y') = x**	
15a.	**(x · y)' = x' + y'**	***(DeMorgan's Theorem)***
15b.	**(x + y)' = x' · y'**	
16a.	**x + x' · y = x + y**	
16b.	**x · (x' + y) = x · y**	

Algebraic Two-Level Size Minimization
Uniting Theorem

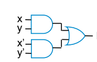
- Multiply out to sum-of-products, then apply Uniting Theorem
  - ab + ab' = a(b + b') = a*1 = a
  - "Combining terms to eliminate a variable"
  - (Formally called the "Uniting theorem")
- Sometimes after combining terms, can combine resulting terms

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'



delay = 2 gate delay
size = 16 * 2 = 32 transistors

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



delay = 2 gate delay
size = 6 * 2 = 12 transistors

Algebraic Two-Level Size Minimization
Duplication

- Duplicating a term sometimes helps
  - Note that doesn't change function
  - c + d = c + d + d = c + d + d + d + d ...

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

Algebraic Two-Level Size Minimization
Complex and Error Prone

- Algebraic Manipulation
  - Which "rules" to use and when?
  - Easy to miss "seeing" possible opportunities to combine terms

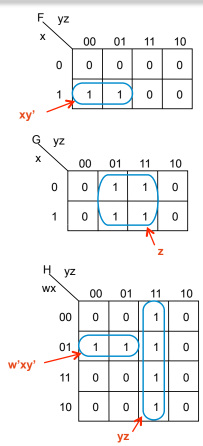
F(a, b, c) = b'c' + bc + a'b' + a'b
F(a, b, c) = b'c' + bc + a'b'

F(a, b, c, d) = a'b'cd + c'd + ab'd + acd + a'bcd + a'c'd
F(a, b, c, d) = d

F(a, b, c, d, e, f, g) = a'b'c + d'e'f + fa + eg + a'bcd'e'f'g + a'bc'efg + c
F(a, b, c, d, e, f, g) = ?

K-maps (Karnaugh Maps)

- Graphical method to help us find opportunities to combine terms
  - Graphical method to help us find opportunities to combine terms
  - Create map where *adjacent* minterms <u>differ in one variable</u>
  - Can clearly see opportunities to combine terms – look for adjacent 1s



Two-Level Size Minimization Using K-maps

General K-map method

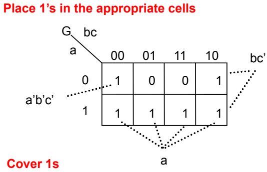
1. Convert the function's equation into sum-of-products form
2. Place 1s in the appropriate K-map cells for each term
3. Cover all 1s by drawing the fewest largest circles, with every 1 included at least once; write the corresponding term for each circle
4. OR all the resulting terms to create the minimized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

Example: Minimize G = a + a'b'c' + b*(c' + bc')

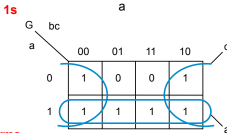
Step 1 - Convert to sum-of-products

G = a + a'b'c' + bc' + bc'

Step 2 - Place 1's in the appropriate cells



Step 3 - Cover 1s



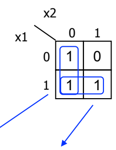
Step 4 - OR terms

G = a + c'

Two-Variable K-Maple 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

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



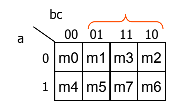
x1'x2' + x1x2'	x1x2' + x1x2	F = x1 + x2'
x2'(x1' + x1)	x1(x2' + x2)	
x2'(1)	x1(1)	
x2'	x1	

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



Truth table

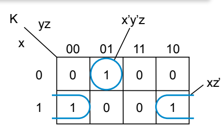
REMEMBER: K-map graphically place minterms next to each other when they differ by one variable

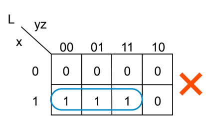
m1 cannot be placed next to m2 (a'b'c, a'bc')

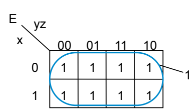
m1 can be placed next to m3 (a'b'c, a'bc)
m2 can be placed next to m3 (a'bc', a'bc)

Three-Variable K-Map Optimization Guidelines

- 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

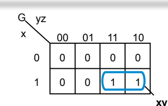




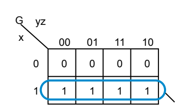


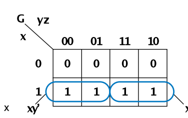
Three-Variable K-Map Optimization Guidelines

- Two adjacent 1s means one variables can be eliminated
  - Same as in two-variable K-maps
- 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
- Four adjacent cells can be in shape of a square



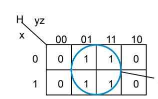
G = xyz + xyz'
G = xy(z+ z')
G = xy





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



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

4

Three-Variable K-Map Optimization Guidelines

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

I, y’z, x

I (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)

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

J, x’y’, y’z, xz

J (x \ yz)	00	01	11	10
0	1	1	0	0
1	0	1	1	0

Four-Variable K-Map Optimization Guidelines

- Four-variable K-map follows same principle
  - Left/right adjacent
  - Top/bottom also adjacent

w’xy’, yz

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

- 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

z

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-Maple Example

- Minimize: H = a’b’(cd’ + c’d’) + ab’c’d’ + ab’cd’ + a’bd + a’bcd’

1. Convert to sum-of-products
   H = a’b’cd’ + a’b’c’d’ + ab’c’d’ +
   ab’cd’ + a’bd + a’bcd’

H (ab \ cd)	00	01	11	10
00				
01				
11				
10				

Four-Variable K-Maple Example - Continued

- Minimize: H = a’b’(cd’ + c’d’) + ab’c’d’ + ab’cd’ + a’bd + a’bcd’

1. Convert to sum-of-products
   H = a’b’cd’ + a’b’c’d’ + ab’c’d’ +
   ab’cd’ + a’bd + a’bcd’
2. Place 1s in K-map cells

a’b’c’d’, a’bd, a’b’cd’, a’bcd’

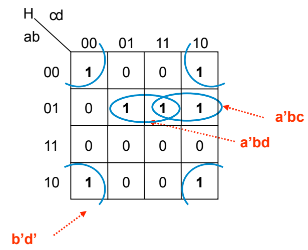
H (ab \ cd)	00	01	11	10
00	**1**	0	0	**1**
01	0	**1**	**1**	**1**
11	0	0	0	0
10	**1**	0	0	**1**

ab’c’d’, ab’cd’

Four-Variable K-Maple Example - Continued

- Minimize: H = a'b'(cd' + c'd') + ab'c'd' + ab'cd' + a'bd + a'bcd'

1. Convert to sum-of-products
   H = a'b'cd' + a'b'c'd' + ab'c'd' +
   ab'cd' + a'bd + a'bcd'
2. Place 1s in K-map cells
3. Cover 1s

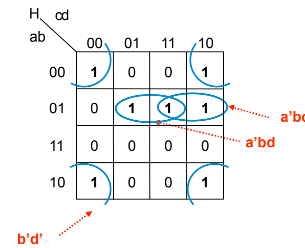


Funny-looking circle, but remember that left/ right adjacent, and top/bottom adjacent

Four-Variable K-Maple Example - Continued

- Minimize: H = a'b'(cd' + c'd') + ab'c'd' + ab'cd' + a'bd + a'bcd'

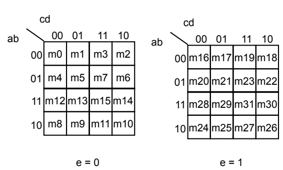
1. Convert to sum-of-products
   H = a'b'cd' + a'b'c'd' + ab'c'd' +
   ab'cd' + a'bd + a'bcd'
2. Place 1s in K-map cells
3. Cover 1s
4. OR resulting terms



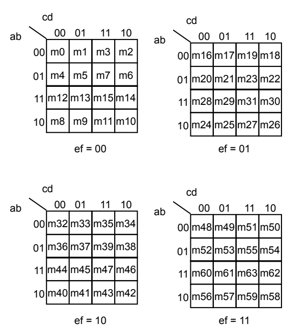
H = b'd' + a'bc + a'bd

Larger N-Variable K-Maps

- Graphical minimizing by hand
  - 5 and 6 variable maps exist, but hard to use
- May not yield minimum cover depending on order we choose
  - Is error prone
- Minimization thus typically done by automated tools



Five-variable Map



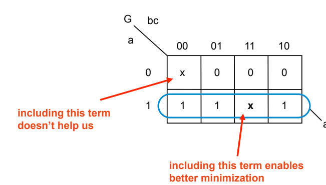
Six-variable Map

Don't Care Input Combinations

- Don't Care Input
  - Input combination that the designer doesn't care what the output is
    - i.e. input condition can never occur
  - Thus, make output be 1 or 0 for those cases *in a way that best minimizes the equation*
  - Represented as **X**s in K-map

abc = 000 and abc = 111 are unused inputs

a	b	c	Z
0	0	0	x
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	x



Simplified Notation for Sum-of-Products Form

- Instead of listing each product, simply list the minterm number
- F(a, b) = ∑m(0, 2) = m0 + m2
  - m – minterms, M – maxterms
  - 0_{10} – 00_2 - a'b'
  - 2_{10} – 10_2 - ab'

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

a \ b	0	1
0	^{0} 1	^{1} 0
1	^{2} 1	^{3} 1

Generalized Three-Variable K-Map

- F(a, b) = ∑m(4, 5, 6, 7)
  - Don't forget column 01 is followed by 11

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

a \ bc	00	01	11	10
0	^{0} 0	^{1} 0	^{3} 0	^{2} 0
1	^{4} 1	^{5} 1	^{7} 1	^{6} 1

Generalized Three-Variable K-Map

- F(a, b) = ∑m(0, 1) + ∑d(4, 5)
  - d – don't cares

a	b	c	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	x
1	0	1	x
1	1	0	0
1	1	1	0

a \ bc	00	01	11	10
0	^{0} 1	^{1} 1	^{3} 0	^{2} 0
1	^{4} x	^{5} x	^{7} 0	^{6} 0

Four-Variable K-Maple Example

- F(a, b, c, d) = ∑m(4, 5, 11, 15)
  - Don't forget in 4-variable K-map, columns and rows are out of sequence too (00, 01, 11, 10)

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

H

ab \ cd	00	01	11	10
00	^{0} 0	^{1} 0	^{3} 0	^{2} 0
01	^{4} 1	^{5} 1	^{7} 0	^{6} 0
11	^{12} 0	^{13} 0	^{15} 1	^{14} 0
10	^{8} 0	^{9} 0	^{11} 1	^{10} 0

Exact Algorithms vs. Heuristic

- Algorithm
  - Finite set of instructions/steps to solve a problem
  - Terminates in finite time at a known end state
- Many algorithms can exist that solve the same problem
- What makes one algorithm better than another?
  - Optimality – "best" quality solution found
  - Efficiency – "good" quality solution found fast
- Exact Algorithm
  - Finds optimal solution
  - May not be efficient
- Heuristic
  - Efficient
  - Finds good solution, but not necessarily optimal

Quine-McCluskey Overview

- Exact Algorithm
- Developed in the mid-50's
- Finds the minimized representation of a Boolean function
- Provides systematic way of generating all prime implicants then extracting a minimum set of primes covering the on-set
- Accomplishes this by repeatedly applying the Uniting theorem
  - Uniting theorem: ab + ab' = a(b+b') = a*1 = a

Review Definitions

- ***Minterm***
  - product term whose literals include every variable of the function exactly once in true or complemented form
- ***On-set***
  - All minterms that define when F=1
- ***Off-set***
  - All minterms that define when F=0

F(a, b, c) = a'b'c + ab

variables: a, b, c

minterms: a'b'c

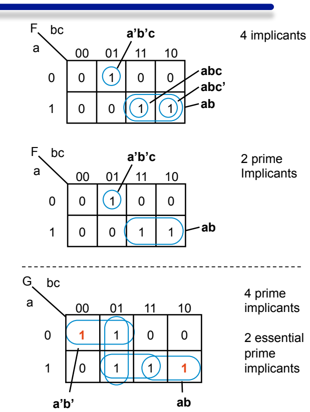
~~ab~~ abc
abc'

on-set: a'b'c, abc', abc

off-set: a'b'c', a'bc', a'bc, ab'c', ab'c

Review Definitions

- ***Implicant***
  - Any product term (minterm or other) that when 1 causes F=1
  - On K-map, any legal (but not necessarily largest) circle
- ***Prime implicant***
  - Maximally expanded implicant – any further expansion would cover 1s not in on-set
- ***Essential prime implicant***
  - The only prime implicant that covers a particular minterm in a function's on-set
  - Importance: We ***must*** include ***all*** essential PIs in a function's cover
  - In contrast, some, but not all, non-essential PIs will be included



Note: We use K-maps are for illustration purposes only

Quine-McCluskey Algorithm

1. Find all the prime implicants
2. Find all the essential prime implicants
3. Select a minimal set of remaining prime implicants that covers the on-set of the function

Quine-McCluskey – Example 1

Minimize F = a'b'c' + a'b'c + ab'c + abc' + abc

Step 1: Find all the prime implicants

- List all elements of on-set and don't care set, represented as a binary number
- Group minterms according to the number of 1's in the minterm

a'b'c' → (0) 000	G0	(0) 000	}	group G0 contains all minterms containing zero 1's
a'b'c → (1) 001	G1	(1) 001	}	group G1 contains all minterms containing one 1
ab'c → (5) 101	G2	(5) 101	}	group G2 contains all minterms containing two 1's
abc' → (6) 110		(6) 110		
abc → (7) 111	G3	(7) 111	}	group G3 contains all minterms containing three 1's

this grouping strategy will help us compare the minterms systematically

Quine-McCluskey – Example 1

Step 1: Find all the prime implicants(cont')

- Compare each entry in Gi to each entry in Gi+1
  - If they differ by 1 bit, we can apply the uniting theorem and eliminate a literal
  - Add check to minterm/implicant to remind us that it is not a prime implicant (combined with another element to form a larger implicant)

G0 ✓	(0) 000		G0	(0,1) 00-
G1 ✓	(1) 001		G1	(1,5) -01
G2 ✓	(5) 101		G2	(5,7) 1-1
✓	(6) 110			(6,7) 11-
G3 ✓	(7) 111			

no new implicants are generated – end of step 1

we have found all prime implicants (ones without check marks)

Quine-McCluskey – Example 1

Step 2: Find all essential prime implicants

- Create prime implicant chart
  - Columns are minterm indicies, rows are the prime implicants we determined

F = a'b'c' + a'b'c + ab'c + abc' + abc

	0 (000)	1 (001)	5 (101)	6 (110)	7 (111)
(0,1) 00-					
(1,5) -01					
(5,7) 1-1					
(6,7) 11-					

derived in Step1

Quine-McCluskey – Example 1

Step 2: Find all essential prime implicants (cont')

- Place "X" in a row if the prime implicant covers the minterm
- Essential prime implicants are found by looking for rows with a single "X"
  - If minterm is covered by one and only one prime implicant – it's an essential prime implicant
- Add essential prime implicants to the cover

essential prime implicants

	0	1	5	6	7
(0,1) 00-	X	X			
(1,5) -01		X	X		
(5,7) 1-1			X		X
(6,7) 11-				X	X

Cover:
F = a'b' + ab

Quine-McCluskey – Example 1

Step 3: Select a minimal set of remaining prime implicants that covers the on set of the function

- Step 2 determined essential prime implicants, and added to cover
  - Essential prime implicants may cover other minterms, cross out all minterms covered by the prime implicants
  - Minterm only needs to be covered once

	0	1	5	6	7
(0,1) 00-	X	X			
(1,5) -01		X	X		
(5,7) 1-1			X		X
(6,7) 11-				X	X

Cover:
F = a'b' + ab

Quine-McCluskey – Example 1

Step 3: Select a minimal set of remaining prime implicants that covers the on set of the function (cont')

- Based on which minterms are left, add minimal set of prime implicants to cover

Only minterm 5 remains – either prime implicant (1,5) or (5,7) will work

	0	1	5	6	7
(0,1) 00-	X	X			
(1,5) -01		X	X		
(5,7) 1-1			X		X
(6,7) 11-				X	X

Cover:
F = a'b' + ab + b'c

Quine-McCluskey – Example 1

- Summary
  - Is this an optimal solution?
    - **YES.**
    - We generate all the minterms and make sure they are all covered by the prime implicants
  - Is the solution unique?
    - **NOT NECESSARILY.**
    - There could be different sets of minimum covers.

Quine-McCluskey – Example 2

Minimize F = w'x'y'z' + w'x'yz + w'x'yz' + w'xy'z' + w'xyz + w'xyz' + wxy'z + wxyz + wx'y'z + wx'yz

Step 1: Find all the prime implicants

- List all elements of on-set and don't care set, represented as a binary number
- Group minterms according to the number of 1's in the minterm

w'x'y'z'	→ (0) 0000
w'x'yz	→ (3) 0011
w'x'yz'	→ (2) 0010
w'xy'z'	→ (4) 0100
w'xyz	→ (7) 0111
w'xyz'	→ (6) 0110
wxy'z	→ (13) 1101
wxyz	→ (15) 1111
wx'y'z	→ (9) 1001
wx'yz	→ (11) 1011

G0	(0) 0000
G1	(2) 0010
	(4) 0100
G2	(3) 0011
	(6) 0110
	(9) 1001
G3	(7) 0111
	(11) 1011
	(13) 1101
G4	(15) 1111

Quine-McCluskey – Example 2

Step 1: Find all the prime implicants (cont')

- Compare each entry in Gi to each entry in Gi+1
  - If they differ by 1 bit, we can apply the uniting theorem and eliminate a literal
  - Add check to minterm/implicant to remind us that it is not a prime implicant

G0	✓	(0) 0000
G1	✓	(2) 0010
	✓	(4) 0100
G2	✓	(3) 0011
	✓	(6) 0110
	✓	(9) 1001
G3	✓	(7) 0111
	✓	(11) 1011
	✓	(13) 1101
G4	✓	(15) 1111

(0,2) ?
(0,4) ?
(2,3) ?
(2,6) ?
(2,9) ? N
(4,3) ? N
(4,6) ?
(4,9) ? N
(3,7) ?
(3,11) ?
(3,13) ? N
(6,7) ?
(6,11) ? N
(6,13) ? N
(9,7) ? N
(9,11) ?
(9,13) ?
(7,15) ?
(11,15) ?
(13,15) ?

G0	✓	(0,2) 00-0
	✓	(0,4) 0-00
G1	✓	(2,3) 001-
	✓	(2,6) 0-10
	✓	(4,6) 01-0
G2	✓	(3,7) 0-11
	✓	(3,11) -011
	✓	(6,7) 011-
	✓	(9,11) 10-1
	✓	(9,13) 1-01
G3	✓	(7,15) -111
	✓	(11,15) 1-11
	✓	(13,15) 11-1

G0	(0,2,4,6) 0--0
G1	(2,3,6,7) 0-1-
G2	(3,7,11,15) --11
	(9,11,13,15) 1--1

no new implicants are generated – end of step 1

Quine-McCluskey – Example 2

Step 2: Find all essential prime implicants

- Create prime implicant chart
  - Columns are minterm indicies, rows are the prime implicants we determined
- Place "X" in a row if the prime implicant covers the minterm
- Essential prime implicants are found by looking for rows with a single "X"
  - Add essential prime implicant to the cover

essential prime implicants

	0	2	3	4	6	7	9	11	13	15
(0,2,4, 6) 0--0	✖	✖		✖	✖					
(2,3,6,7) 0-1-		✖	✖		✖	✖				
(3,7,11,15) --11			✖			✖		✖		✖
(9,11,13,15) 1--1							✖	✖	✖	✖

Cover:
F = w'z' + wz

Quine-McCluskey – Example 2

Step 3: Select a minimal set of remaining prime implicants that covers the on set of the function

- Cross out all minterms covered by the prime implicants
- Based on which minterms are left, add minimal set of prime implicants to cover

Minterm 3 and 7 remain – either prime implicant (2,3,6,7) or (3,7,11,15) will work

	0	2	3	4	6	7	9	11	13	15
(0,2,4, 6) 0--0	✖	✖		✖	✖					
(2,3,6,7) 0-1-		✖	✖		✖	✖				
(3,7,11,15) --11			✖			✖		✖		✖
(9,11,13,15) 1--1							✖	✖	✖	✖

Cover:
F = w'z' + wz + yz

Quine-McCluskey – Example 3
Petrick's Method

- What if determining minimum prime implicant cover is not so easy?
- Assume we have the implicant table below
  - Determine prime implicants, add to cover

essential prime implicants

		2	6	7	8	9	13	15
(2,6)	0-10	X	X					
(8,9)	-001				X	X		
(6,7)	011-		X	X				
(9, 13)	10-1					X	X	
(7, 15)	-111			X				X
(13, 15)	1-11						X	X

Cover:
F = w'yz' + x'y'z

Quine-McCluskey – Example 3
Petrick's Method

- Example 3 (cont')
  - Remove minterms covered by prime implicants
  - Leaves 3 minters – m7, m13, and m15
    - Which remaining prime implicants should we use to obtain the minimum cover?

		~~2~~	~~6~~	7	~~8~~	~~9~~	13	15
(2,6)	~~0-10~~	X	X					
(8,9)	-001				X	X		
(6,7)	011-		X	X				
(9, 13)	10-1					X	X	
(7, 15)	-111			X				X
(13, 15)	1-11						X	X

Cover:
F = w'yz' + x'y'z

Quine-McCluskey – Example 3
Petrick's Method

Petrick's Method – used to determine minimum cover

1. Reduce prime implicant chart by eliminating prime implicant rows and corresponding columns
2. Label rows of reduced prime implicant chart P1, P2 …
3. Form logical equation which is true when all columns are covered
4. Reduce to minimum sum of products by multiplying out and applying X + XY = X
5. Each term in solution represents a covering solution
   - Count number of terms in each, choose one corresponding to the minimum number

			~~2~~	~~6~~	7	~~8~~	~~9~~	13	15
	~~(2,6)~~	~~0-10~~	X	X					
	~~(8,9)~~	~~-001~~				X	X		
P1	(6,7)	011-		X	X				
P2	(9, 13)	10-1					X	X	
P3	(7, 15)	-111			X				X
P4	(13, 15)	1-11						X	X

P = (P1 + P3)(P2 + P4)(P3 + P4)

P = (P1 + P3)(P2P3 +P2P4 +P4P3 + P4P4)

P4P4 = P4

P = (P1 + P3)(P2P3 +P2P4 +P4P3 + P4)

P4 + P4P3 = P4

P = (P1 + P3)(P2P3 +P2P4 +P4)

P4 + P2P4 = P4

P = (P1 + P3)(P2P3 + P4)

P = P1P2P3 + P1P4 + P3P2P3 + P3P4

P3P2P3 = P2P3

$\text{P = P1P2P3 + } \underbrace{\text{P1P4 + P2P3 + P3P4}}_{\text{Any of these provide minimum cover}}$

more terms than other solutions

Actually - P1P2P3 + P2P3 = P2P3, so we can eliminate term altogether

Quine-McCluskey – Example 3
Petrick's Method

- Final cover = essential prime implicants + minimum prime implicant cover

Essential Prime Implicants
w'yz', x'y'z

Minimum prime implicant cover list:
(option 1 - P1P4) w'xy,wyz
(option 2 - P2P3) wx'z, xyz
(option 3 - P3P4) xyz, wyz

			~~2~~	~~6~~	7	~~8~~	~~9~~	13	15
	~~(2,6)~~	~~0-10~~	X	X					
	~~(8,9)~~	~~-001~~				X	X		
P1	(6,7)	011-		X	X				
P2	(9, 13)	10-1					X	X	
P3	(7, 15)	-111			X				X
P4	(13, 15)	1-11						X	X

$\underbrace{\text{P = P1P4 + P2P3 + P3P4}}_{\text{Any of these provide minimum cover (equal number of ``circles'')}}$

Minimized Equation F = w'yz' + x'y'z + xyz + wyz

Quine-McCluskey

- What about don't cares?
- Alternative methods to determine Minimum Cover
  - Row vs. Column Dominance