Unit 9

Implementing Combinational Functions with Karnaugh Maps

Outcomes

- I can use Karnaugh maps to synthesize combinational functions with several outputs
- I can determine the appropriate size and contents of a memory to implement any logic function (i.e. truth table)

Covering Combinations

- A minterm corresponds to ("covers") 1 combination of a logic function
- As we remove variables from a product term, more combinations are covered
  - The product term will evaluate to true regardless of the removed variables value (i.e. the term is independent of that variable)

F = WX'YZ = m11

W	X	Y	Z	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	**0**
0	1	0	1	**0**
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	**0**

F = WX'Z = m9+m11

W	X	Y	Z	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	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**0**
1	0	0	0	**0**
1	0	0	1	**1**
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	**0**

Covering Combinations

- The more variables we can remove the more combinations a single product term covers
  - Said differently, a small term will cover (or expand to) more combinations
- The smaller the term, the smaller the circuit
  - We need fewer gates to check for multiple combinations
- For a given function, how can we find these smaller terms?

F = X'Z
= m1+m3+m9+m11

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**0**
1	0	0	0	**0**
1	0	0	1	**1**
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	**0**

F = X'
= m0+m1+m2+m3+m8+m9+m10+m11

W	X	Y	Z	F
0	0	0	0	**1**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**0**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**0**

A new way to synthesize your logic functions

KARNAUGH MAPS

Logic Function Synthesis

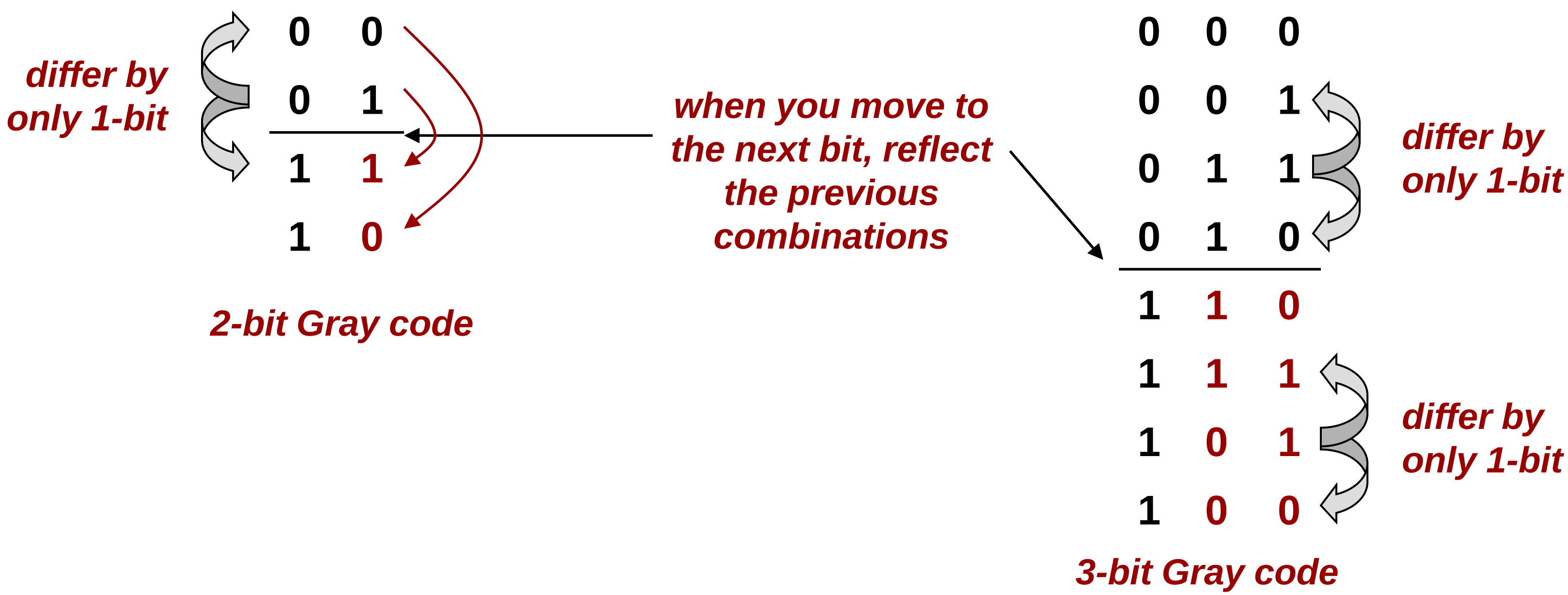
- Given a function description as a T.T. or sum of minterm (product of maxterm) form, how can we arrive at a circuit implementation or equation (i.e. perform logic synthesis)?
- Methods
  - Minterms / maxterms
    - Use Boolean Algebra to find minimal 2-level implementation
  - Karnaugh Maps [we will learn this one now]
    - Graphical method amenable to human visual inspection and can be used for functions of **up to 6 variables (but becomes large and unwieldy after just 4 variables)**
  - Quine-McCluskey Algorithm (amenable to computer implementations
  - Others: Espresso algorithm, Binary Decision Diagrams, etc.

Karnaugh Maps

- If used correctly, will always yield a minimal, **2-level implementation**
  - There may be a more minimal 3-level, 4-level, 5-level… implementation but K-maps produce the minimal two-level (SOP or POS) implementation
- Represent the truth table graphically as a series of adjacent squares that allows a human to see where variables can be removed

Gray Code

- Different than normal binary ordering
- Reflective code
  - When you add the (n+1)th bit, reflect all the previous n-bit combinations
- Consecutive code words differ by only 1-bit



Karnaugh Map Construction

- Every square represents 1 input combination
- Must label axes in Gray code order
- Fill in squares with given function values

X	Y	Z	F
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

F(x,y,z)=m1 + m4 + m5 + m6

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

3 Variable Karnaugh Map

G(w,x,y,z)=m1+m2+m3+m5+m6+m7+m9+ m10+m11+m14+m15

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

4 Variable Karnaugh Map

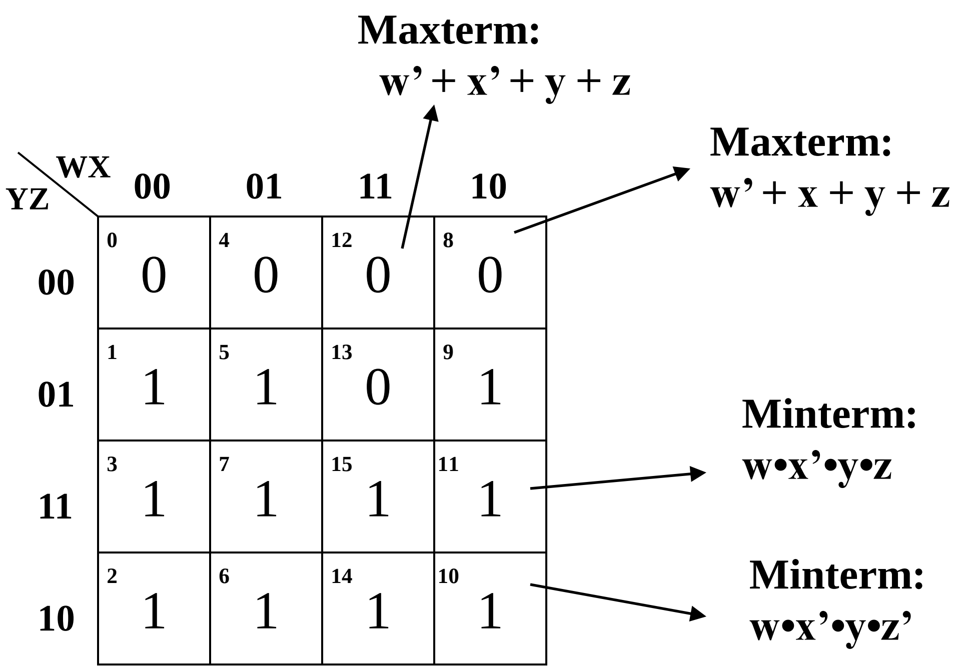
Karnaugh Maps

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

YZ \ WX	00	01	11	10
00	0: 0	4: 0	12: 0	8: 0
01	1: 1	5: 1	13: 0	9: 1
11	3: 1	7: 1	15: 1	11: 1
10	2: 1	6: 1	14: 1	10: 1

Karnaugh Maps

- Squares with a '1' represent minterms that must be included in the SOP solution
- Squares with a '0' represent maxterms that must be included in the POS solution



Karnaugh Maps

- Groups (of 2, 4, 8, etc.) of adjacent 1's will always simplify to smaller product term than just individual minterms

F=m0+m2+m4+m5+m6

Z \ XY	00	01	11	10
0	(0) 1	(2) 1	(6) 1	(4) 1
1	(1) 0	(3) 0	(7) 0	(5) 1

3 Variable Karnaugh Map

$= m0 + m2 + m6 + m4$
$= x'y'z' + x'yz' + xyz' + xy'z'$
$= z'(x'y' + x'y + xy + xy')$
$= z'(x'(y'+y) + x(y+y'))$
$= z'(x'+x)$
$= z'$

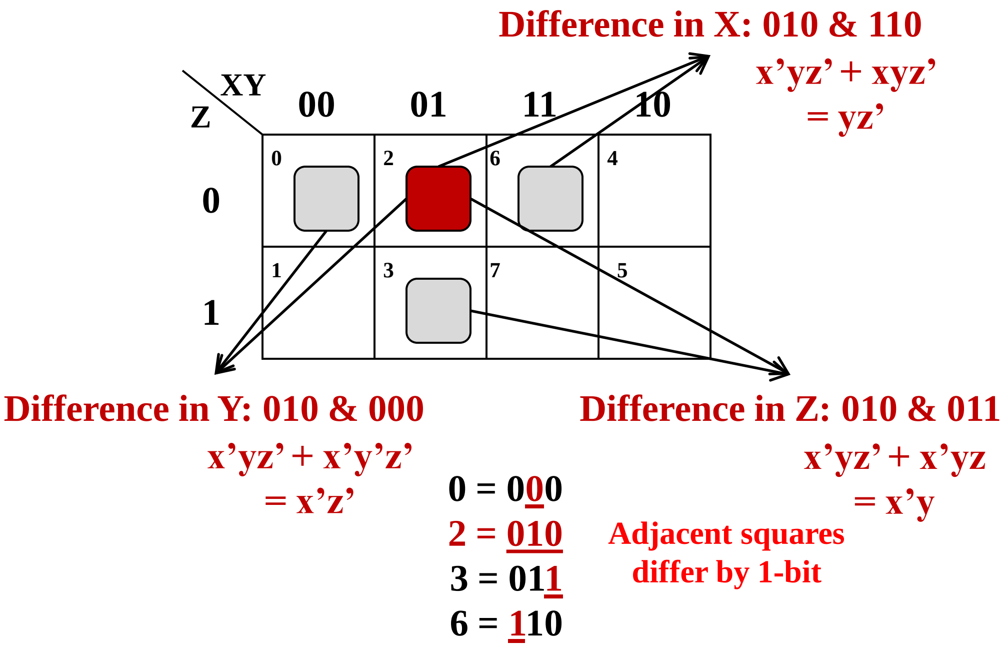
$= m4 + m5$
$= xy'z' + xy'z = xy'(z'+z)$
$= xy'$

USC Viterbi School of Engineering

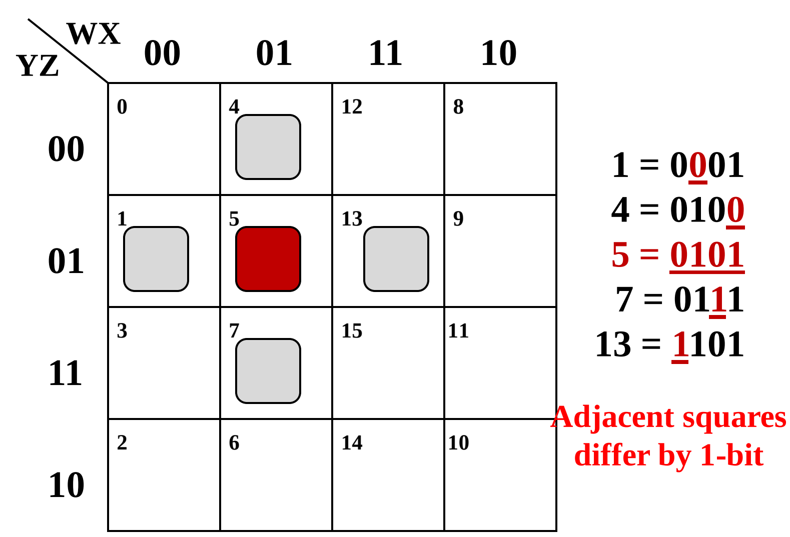
Karnaugh Maps

- Adjacent squares differ by 1-variable
  - This will allow us to use T10 = AB + AB’= A or T10’ = (A+B’)(A+B) = A

3 Variable Karnaugh Map



4 Variable Karnaugh Map



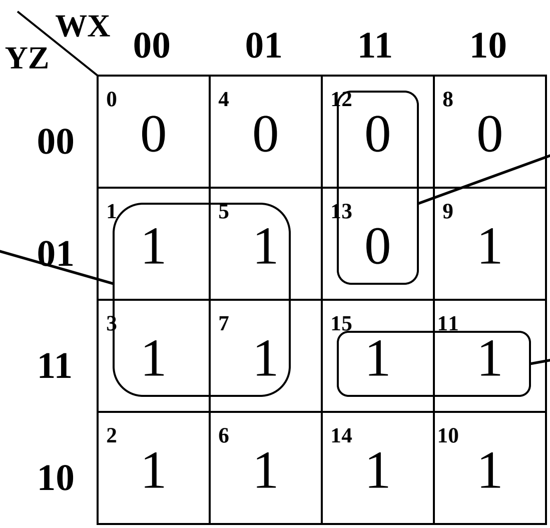
Karnaugh Maps

- 2 adjacent 1's (or 0's) differ by only one variable
- 4 adjacent 1's (or 0's) differ by two variables
- 8, 16, … adjacent 1's (or 0's) differ by 3, 4, … variables
- By grouping adjacent squares with 1's (or 0's) in them, we can come up with a simplified expression using T10 (or T10' for 0's)

w'•x'•y'•z + w'•x'•y•z + w'•x•y'•z + w'•x•y•z

= **w'•z**

w'z are constant while all combos of x and y are present (x'y', x'y, xy', xy)



(w'+x'+y+z)•(w'+x'+y+z') = (w'+x'+y)

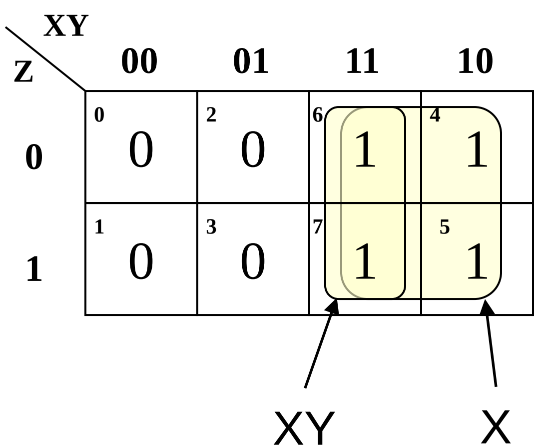
w•x•y•z + w•x'•y•z = w•y•z

K-Map View of the Theorems

- The 2 & 3 variable theorems used to simplify expressions can be illustrated using K-Maps.

T9: Covering

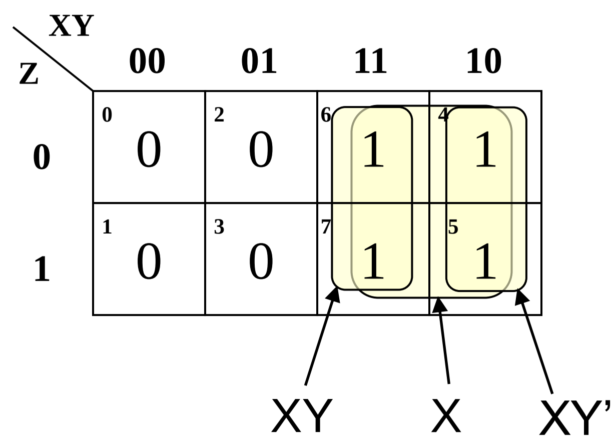
$$X + XY = X$$



X "covers" XY so XY not needed

T10: Combining

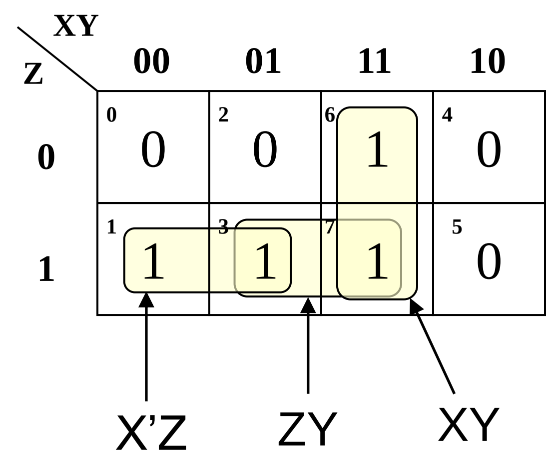
$$XY + XY' = X$$



XY and XY' can be combined to form X

T11: Consensus

$$XY + X'Z + ZY = XY + X'Z$$



Don't need ZY if you have X'Z and XY

K-Map Grouping Rules

- Cover the 1's [=on-set] or 0's [=off-set] with as few groups as possible, but make those groups as large as possible
  - Make them as large as possible even if it means "covering" a 1 (or 0) that's already a member of another group
- Make groups of 1, 2, 4, 8, ... and they must be rectangular or square in shape.
- Wrapping is legal

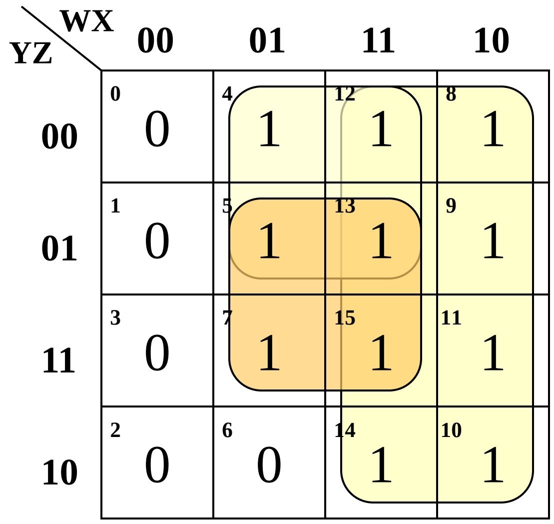
Group These K-Maps

Z \ XY	00	01	11	10
0	(0) 0	(2) 1	(6) 0	(4) 0
1	(1) 1	(3) 0	(7) 0	(5) 0

Z \ XY	00	01	11	10
0	(0) 1	(2) 1	(6) 0	(4) 0
1	(1) 1	(3) 0	(7) 0	(5) 0

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 1	(8) 1
01	(1) 1	(5) 1	(13) 1	(9) 0
11	(3) 1	(7) 1	(15) 1	(11) 0
10	(2) 0	(6) 0	(14) 0	(10) 1

Karnaugh Maps



- Cover the remaining ‘1’ with the largest group possible even if it “reuses” already covered 1’s

Karnaugh Maps

- Groups can wrap around from:
  - Right to left
  - Top to bottom
  - Corners

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 1	(8) 0
01	(1) 1	(5) 0	(13) 0	(9) 1
11	(3) 1	(7) 0	(15) 0	(11) 1
10	(2) 0	(6) 0	(14) 1	(10) 0

F = X’Z + WXZ’

YZ \ WX	00	01	11	10
00	(0) 1	(4) 0	(12) 0	(8) 1
01	(1) 0	(5) 0	(13) 0	(9) 0
11	(3) 0	(7) 0	(15) 0	(11) 0
10	(2) 1	(6) 0	(14) 0	(10) 1

F = X’Z’

Group This

YZ \ WX	00	01	11	10
00	[0] 0	[4] 0	[12] 0	[8] 0
01	[1] 1	[5] 1	[13] 0	[9] 1
11	[3] 1	[7] 1	[15] 1	[11] 1
10	[2] 1	[6] 1	[14] 1	[10] 1

K-Map Translation Rules

- When translating a group of 1's, find the variable values that are constant for each square in the group and translate only those variables values to a product term
- Grouping 1's yields SOP
- When translating a group of 0's, again find the variable values that are constant for each square in the group and translate only those variable values to a sum term
- Grouping 0's yields POS

Karnaugh Maps (SOP)

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 0	(8) 0
01	(1) 1	(5) 1	(13) 0	(9) 1
11	(3) 1	(7) 1	(15) 1	(11) 1
10	(2) 1	(6) 1	(14) 1	(10) 1

F =

Karnaugh Maps (SOP)

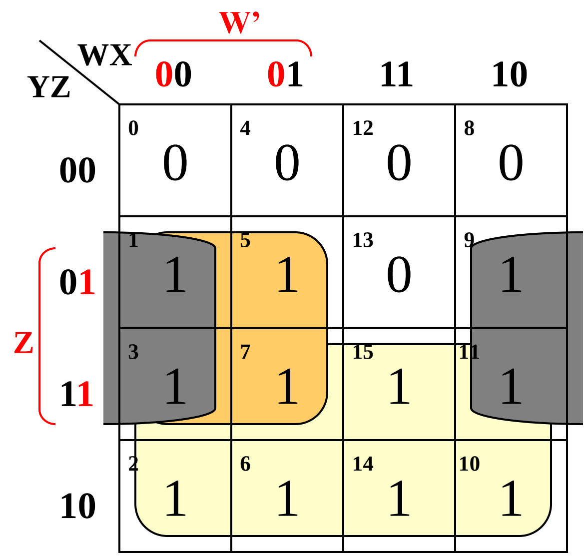
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 0	(8) 0
01	(1) 1	(5) 1	(13) 0	(9) 1
Y 11	(3) 1	(7) 1	(15) 1	(11) 1
Y 10	(2) 1	(6) 1	(14) 1	(10) 1

F = Y

Karnaugh Maps (SOP)

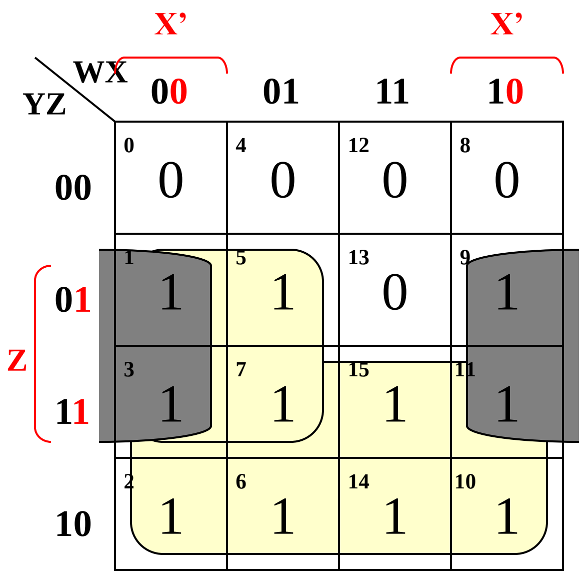
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



F = Y + W'Z + …

Karnaugh Maps (SOP)

W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



F = Y + W'Z + X'Z

Karnaugh Maps (POS)

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 0	(8) 0
01	(1) 1	(5) 1	(13) 0	(9) 1
11	(3) 1	(7) 1	(15) 1	(11) 1
10	(2) 1	(6) 1	(14) 1	(10) 1

F =

Karnaugh Maps (POS)

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

YZ \ WX	00	01	11	10
Y,Z 00	(0) 0	(4) 0	(12) 0	(8) 0
01	(1) 1	(5) 1	(13) 0	(9) 1
11	(3) 1	(7) 1	(15) 1	(11) 1
10	(2) 1	(6) 1	(14) 1	(10) 1

F = (Y+Z)

Karnaugh Maps (POS)

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

WX

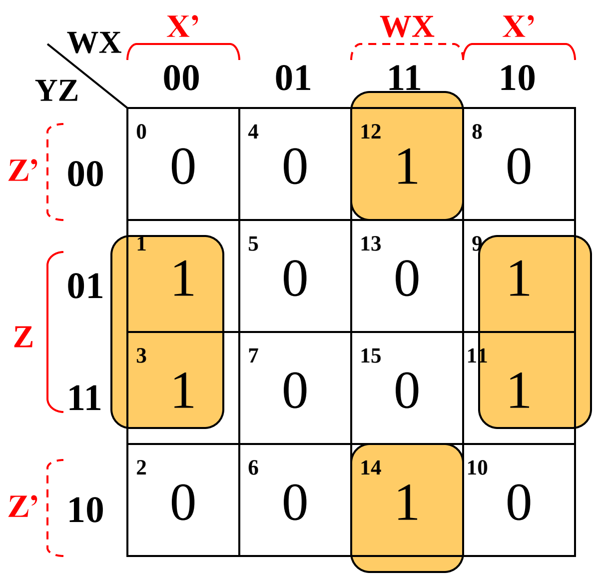
YZ \ WX	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	1	1	1	1
10	1	1	1	1

Y

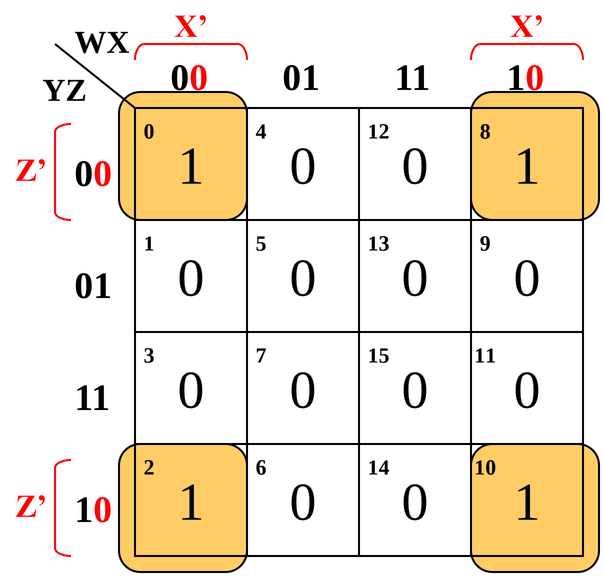
F = (Y+Z)(W'+X'+Y)

Karnaugh Maps

- Groups can wrap around from:
  - Right to left
  - Top to bottom
  - Corners



F = X'Z + WXZ'



F = X'Z'

Exercises

YZ \ WX	00	01	11	10
00	[0] 1	[4] 0	[12] 0	[8] 1
01	[1] 1	[5] 0	[13] 0	[9] 1
11	[3] 0	[7] 0	[15] 0	[11] 0
10	[2] 1	[6] 0	[14] 1	[10] 1

$F_{SOP}=$

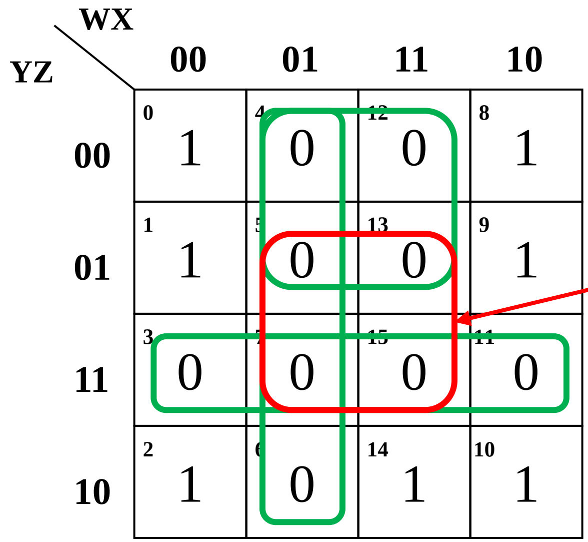
YZ \ WX	00	01	11	10
00	[0] 1	[4] 0	[12] 0	[8] 1
01	[1] 1	[5] 0	[13] 0	[9] 1
11	[3] 0	[7] 0	[15] 0	[11] 0
10	[2] 1	[6] 0	[14] 1	[10] 1

$F_{POS}=$

P(x,y,z)=m2+m3+m5+m7

P=

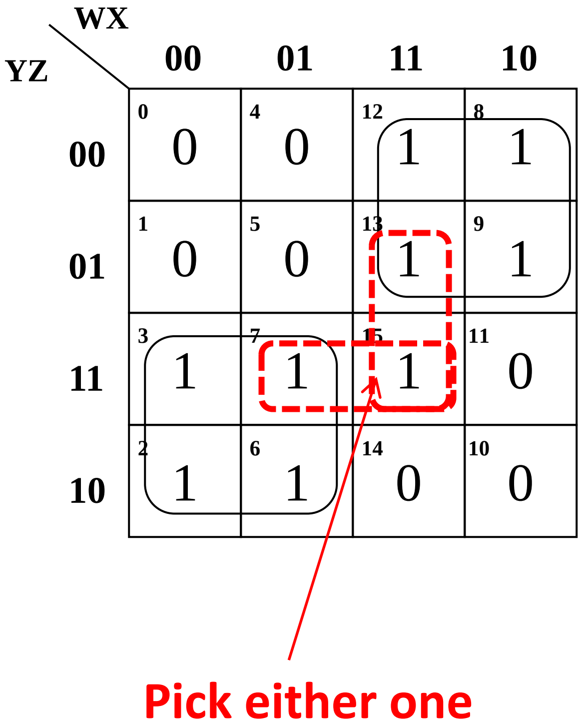
No Redundant Groups



This group does not cover new squares that are not already part of another essential grouping

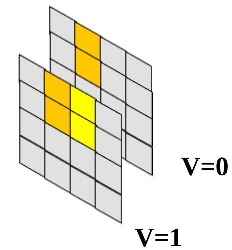
Multiple Minimal Expressions

- For some functions, multiple minimal expressions (multiple minimal groups) exist…Pick one

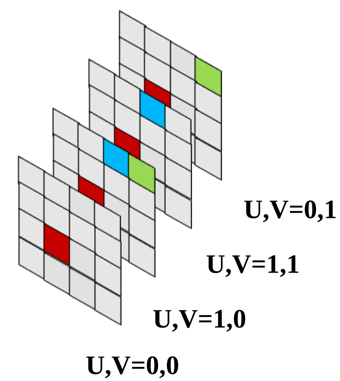


Karnaugh Maps Beyond 4 Variables

- Recall, K-Maps require an adjacency for each variable
  - To see the necessary adjacencies, 5 and 6 variable K-Maps can be thought of in three dimensions
- Can we have 7-variable K-Maps?
  - No! We would need to see 7 adjacencies per square and we humans cannot visualize 4 dimensions
- Other computer-friendly minimization algorithms
  - Quine-McCluskey
    - Still exponential runtime
    - Minimization is NP-hard problem
  - Espresso-heuristic Minimizer
    - Achieves "good" minimization in far less time (may not be absolute minimal)



5 Variable K-Maps



6 Variable K-Maps

DON'T CARE OUTPUTS

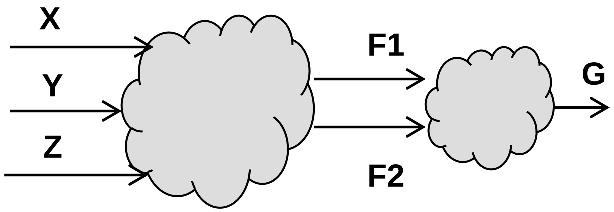
Don't-Cares

- Sometimes there are certain input combinations that are illegal (due to physical or other external constraints)
- The outputs for the illegal inputs are "don't-cares"
  - The output can either be 0 or 1 since the inputs can never occur
  - Don't-cares can be included in groups of 1 or groups of 0 when grouping in K-Maps
  - Use them to make as big of groups as possible

Use 'Don't care' outputs as wildcards (e.g. the blank tile in Scrabble™). They can be either 0 or 1 whatever helps make bigger groups to cover the ACTUAL 1's

Combining Functions

- Given intermediate functions F1 and F2, how could you use AND, OR, NOT to make G
- Notice certain F1,F2 combinations never occur in G(x,y,z)…what should we make their output in the T.T.



X	Y	Z	F1	F2	G
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	0

F1	F2	G
0	0	
0	1	
1	0	
1	1	

Invalid Input Combinations

- An example of where Don't-Cares may come into play is Binary Coded Decimal (BCD)
  - Rather than convert a decimal number to unsigned binary (i.e. summing increasing powers of 2) we can represent each decimal digit as a separate group of 4-bits (with weights 8,4,2,1 for each group of 4 bits)
  - **Combinations 1010-1111 cannot occur!**

$(439)_{10}$

BCD Representation: 0100 0011 1001

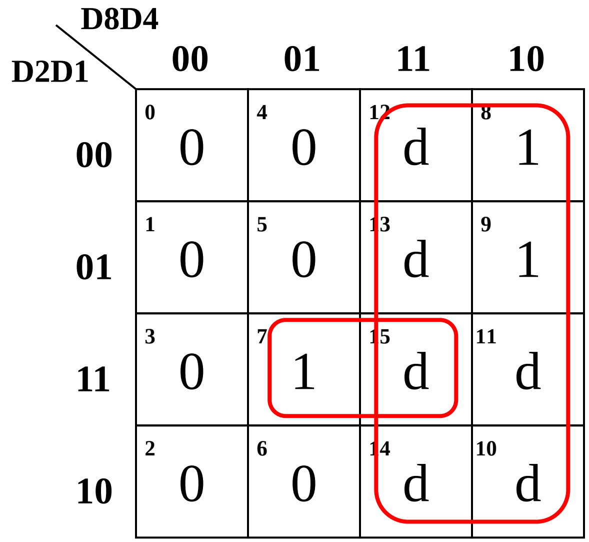
8 4 2 1   8 4 2 1   8 4 2 1

This is not the binary representation of 439, it is the Binary Coded Decimal (BCD) representation

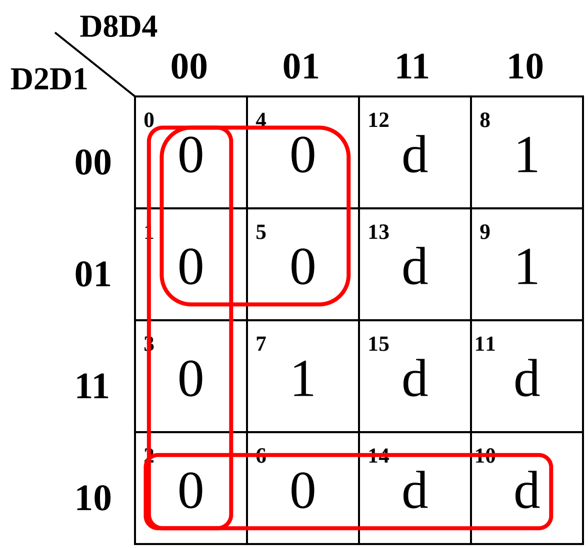
Important: BCD represent each decimal digit with a separate group of bits

Don't Care Example

D8	D4	D2	D1	GT6
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**d**
1	0	1	1	**d**
1	1	0	0	**d**
1	1	0	1	**d**
1	1	1	0	**d**
1	1	1	1	**d**



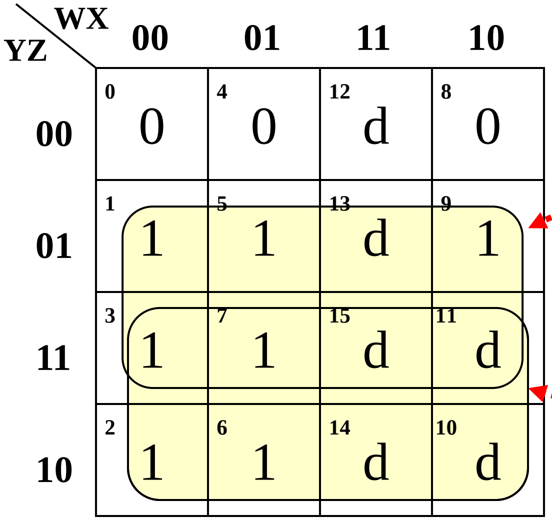
$GT6_{SOP}$=



$GT6_{POS}$=

Don't Cares

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**d**
1	0	1	1	**d**
1	1	0	0	**d**
1	1	0	1	**d**
1	1	1	0	**d**
1	1	1	1	**d**



F = Z + Y

Reuse "d's" to make as large a group as possible to cover 1,5, & 9

Use these 4 "d's" to make a group of 8

Don’t Cares

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**d**
1	0	1	1	**d**
1	1	0	0	**d**
1	1	0	1	**d**
1	1	1	0	**d**
1	1	1	1	**d**

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) d	(8) 0
01	(1) 1	(5) 1	(13) d	(9) 1
11	(3) 1	(7) 1	(15) d	(11) d
10	(2) 1	(6) 1	(14) d	(10) d

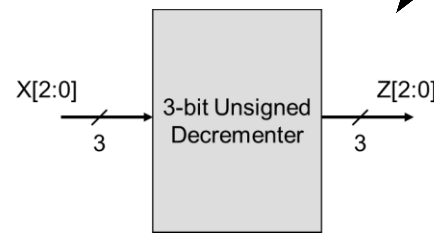
F = Y+Z

You can use “d’s” when grouping 0’s and converting to POS

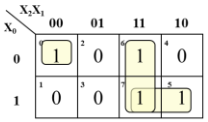
A GENERAL, COMBINATIONAL CIRCUIT DESIGN PROCESS

Combinational Design Process

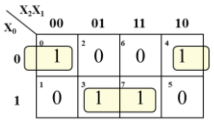
- Understand the problem
  - How many input bits and their representation system
  - How many output bits need be generated and what are their representation
  - Draw a block diagram
- Write a truth table
- Use a K-map to derive an equation for EACH output bit
- Use the equation to draw a circuit for EACH output bit, letting each circuit run in parallel to produce their respective output bit



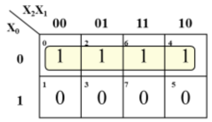
X_2	X_1	X_0	Z_2	Z_1	Z_0
0	0	0	**1**	**1**	**1**
0	0	1	**0**	**0**	**0**
0	1	0	**0**	**0**	**1**
0	1	1	**0**	**1**	**0**
1	0	0	**0**	**1**	**1**
1	0	1	**1**	**0**	**0**
1	1	0	**1**	**0**	**1**
1	1	1	**1**	**1**	**0**



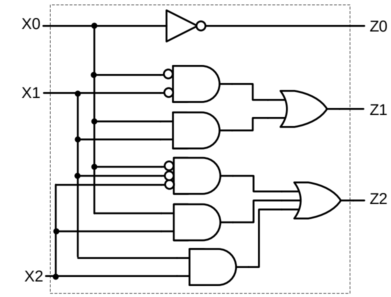
$Z_2 = X_2X_0 + X_2X_1 + X_2'X_1'X_0'$



$Z_1 = X_1'X_0' + X_1X_0$

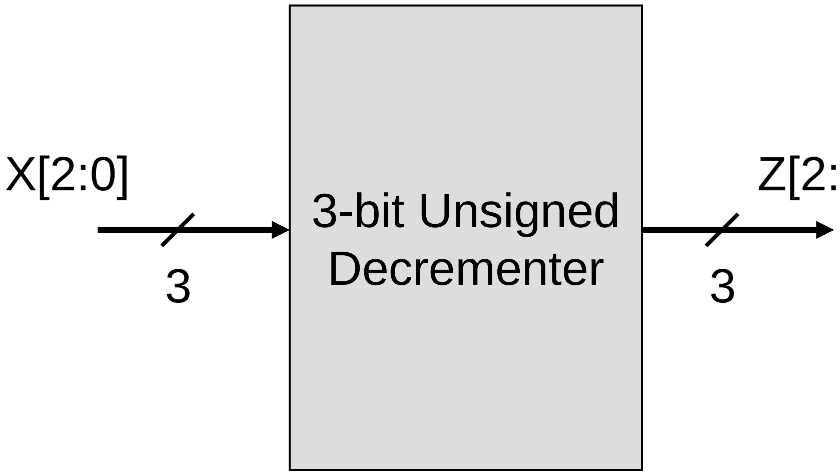


$Z_0 = X_0'$



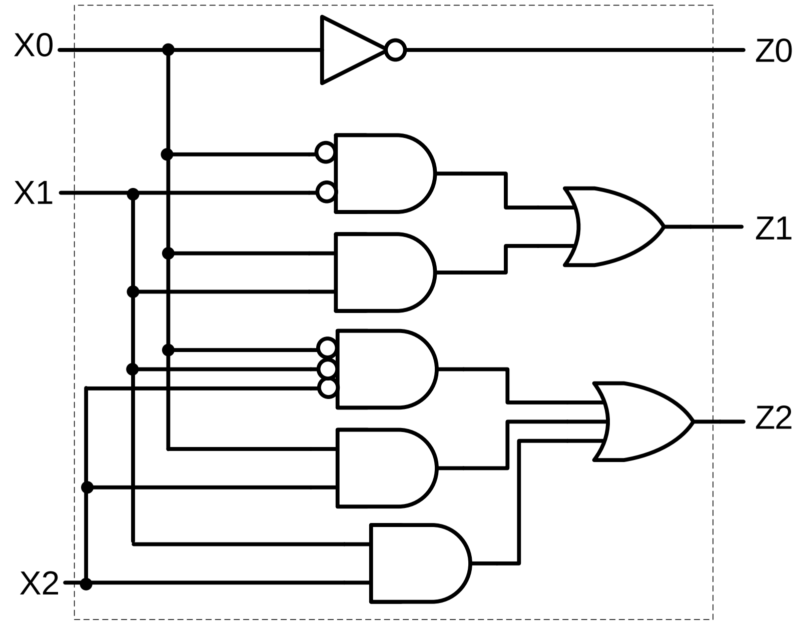
Designing Circuits w/ K-Maps

- Given a description…
  - Block Diagram
  - Truth Table
  - K-Map for each output bit (each output bit is a separate function of the inputs)
- 3-bit unsigned decrementer (Z = X-1)
  - If X[2:0] = 000 then Z[2:0] = 111, etc.



3-bit Number Decrementer

X_2	X_1	X_0	Z_2	Z_1	Z_0
0	0	0	**1**	**1**	**1**
0	0	1	**0**	**0**	**0**
0	1	0	**0**	**0**	**1**
0	1	1	**0**	**1**	**0**
1	0	0	**0**	**1**	**1**
1	0	1	**1**	**0**	**0**
1	1	0	**1**	**0**	**1**
1	1	1	**1**	**1**	**0**



X_0 \ X_2X_1	00	01	11	10
0	1	0	1	0
1	0	0	1	1

$Z_2 = X_2X_0 + X_2X_1 + X_2'X_1'X_0'$

X_0 \ X_2X_1	00	01	11	10
0	1	0	0	1
1	0	1	1	0

$Z_1 = X_1'X_0' + X_1X_0$

X_0 \ X_2X_1	00	01	11	10
0	1	1	1	1
1	0	0	0	0

$Z_0 = X_0'$

Squaring Circuit

- Design a combinational circuit that accepts a 3-bit number and generates an output binary number equal to the square of the input number. ($B = A^2$)
- Using 3 bits we can represent the numbers from _______ to ______ .
- The possible squared values range from _______ to _______ .
- Thus to represent the possible outputs we need how many bits? ________

3-bit Squaring Circuit

A	Inputs A_2	A_1	A_0	Outputs B_5	B_4	B_3	B_2	B_1	B_0	$B=A^2$
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	1
2	0	1	0	0	0	0	1	0	0	4
3	0	1	1	0	0	1	0	0	1	9
4	1	0	0	0	1	0	0	0	0	16
5	1	0	1	0	1	1	0	0	1	25
6	1	1	0	1	0	0	1	0	0	36
7	1	1	1	1	1	0	0	0	1	49

A0 \ A2A1	00	01	11	10
0	0	0	1	0
1	0	0	1	0

B5 = A2A1

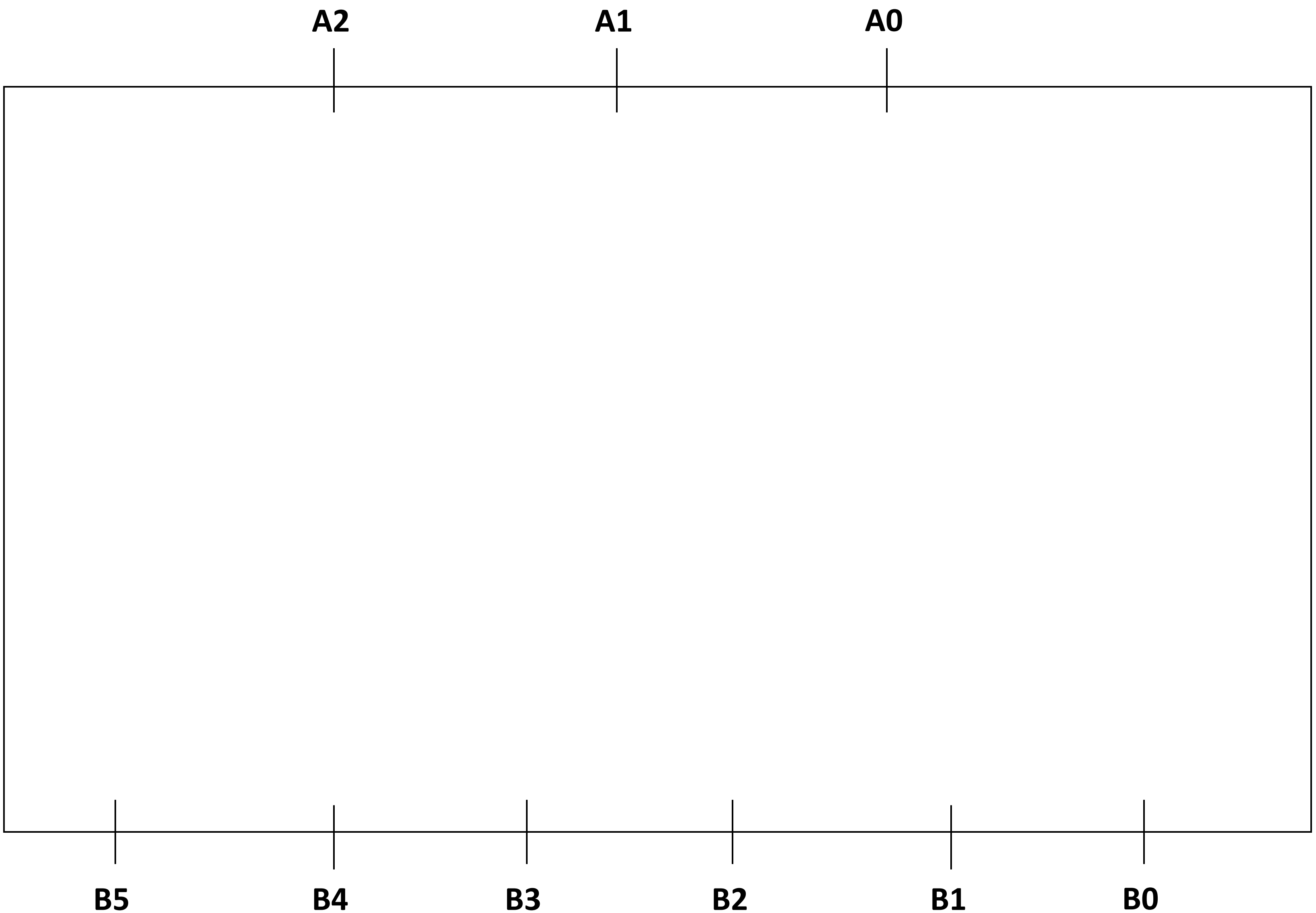
A0 \ A2A1	00	01	11	10
0	0	0	0	1
1	0	0	1	1

B4 = A2A0 + A2A1’

A0 \ A2A1	00	01	11	10
0	0	0	0	0
1	1	1	1	1

B0 = A0

3-bit Squaring Circuit



If time permits…

FORMAL TERMINOLOGY FOR KMAPS

Terminology

- Implicant: A product term (grouping of 1's) that covers a subset of cases where F=1
  - the product term is said to "imply" F because if the product term evaluates to '1' then F='1'
- Prime Implicant: The largest grouping of 1's (smallest product term) that can be made
- Essential Prime Implicant: A prime implicant (product term) that is needed to cover all the 1's of F

Implicant Examples

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

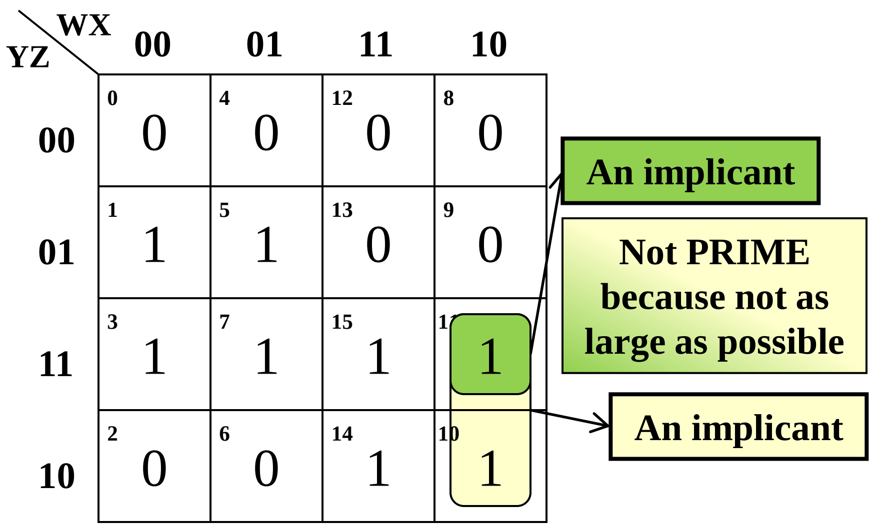
YZ \ WX	**00**	**01**	**11**	**10**
00	(0) 0	(4) 0	(12) 0	(8) 0
01	(1) 1	(5) 1	(13) 0	(9) 0
11	(3) 1	(7) 1	(15) 1	(11) 1
10	(2) 0	(6) 0	(14) 1	(10) 1

An implicant

Not PRIME because not as large as possible

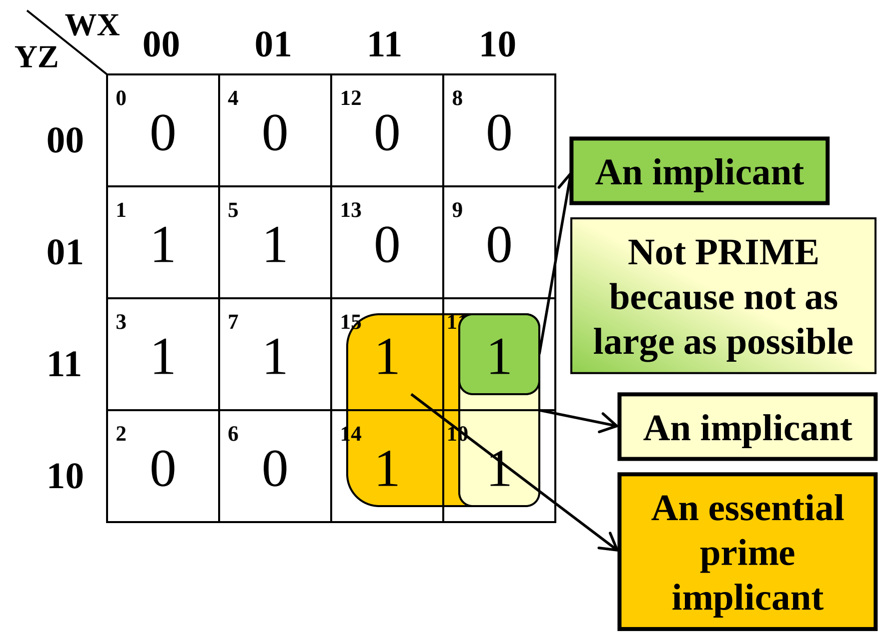
Implicant Examples

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



Implicant Examples

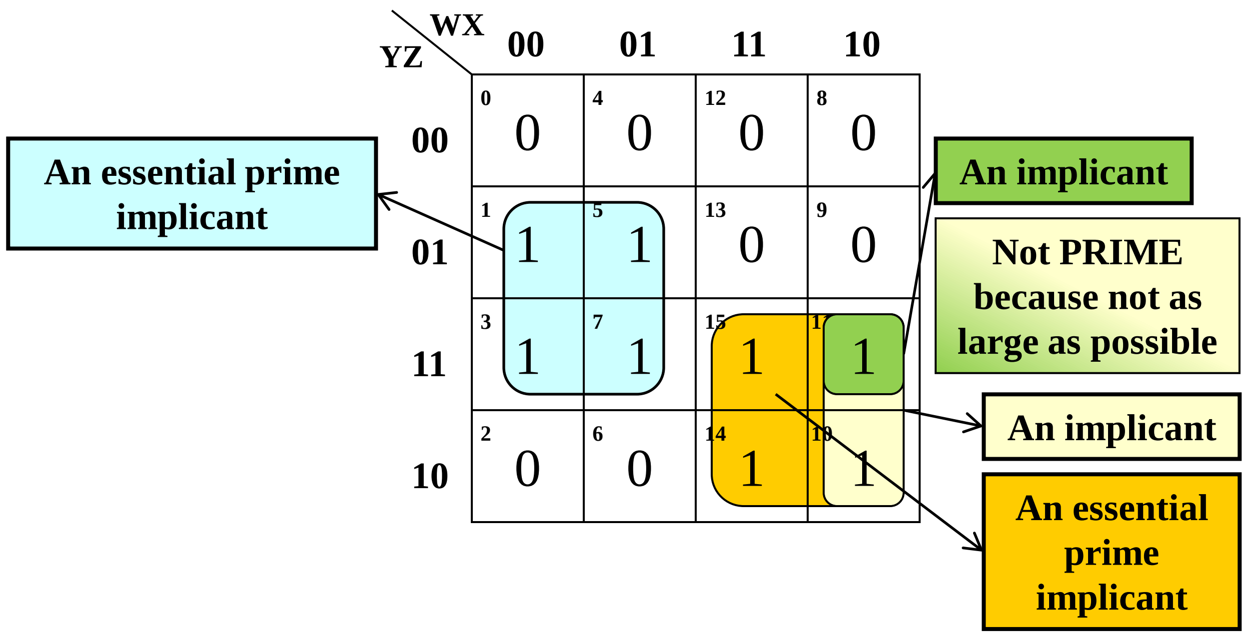
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



An essential prime implicant (largest grouping possible, that must be included to cover all 1's)

Implicant Examples

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

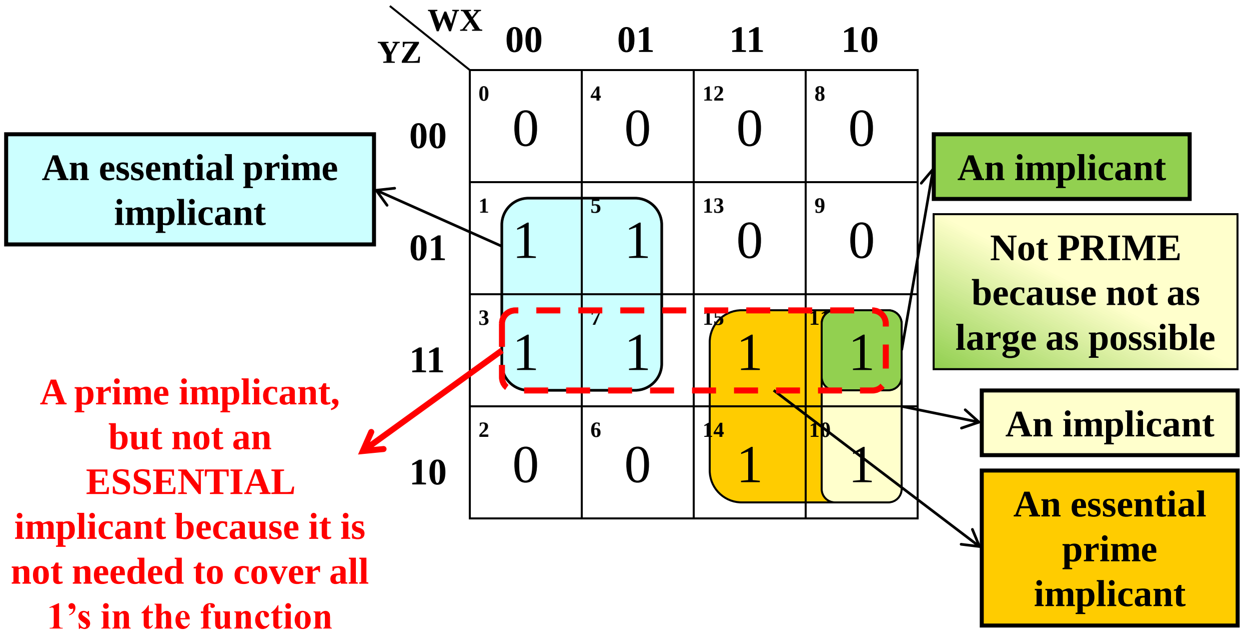


An essential prime implicant (largest grouping possible, that must be included to cover all 1's)

USC Viterbi School of Engineering

Implicant Examples

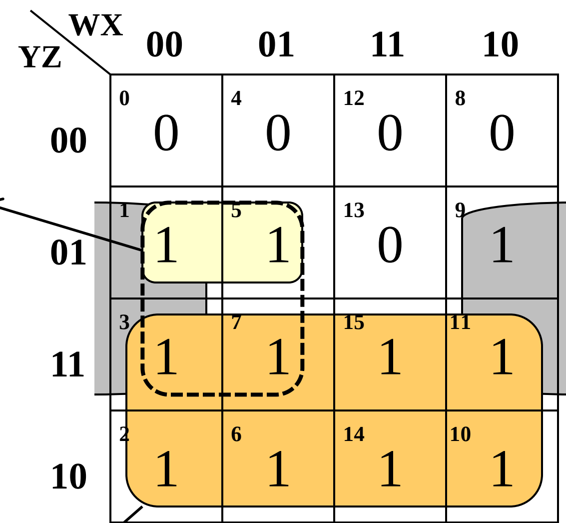
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



An essential prime implicant (largest grouping possible, that must be included to cover all 1's)

Implicant Examples

W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



An implicant, but not a PRIME implicant because it is not as large as possible (should expand to combo's 3 and 7)

An essential prime implicant (largest grouping possible, that must be included to cover all 1's)

An essential prime implicant

K-Map Grouping Rules

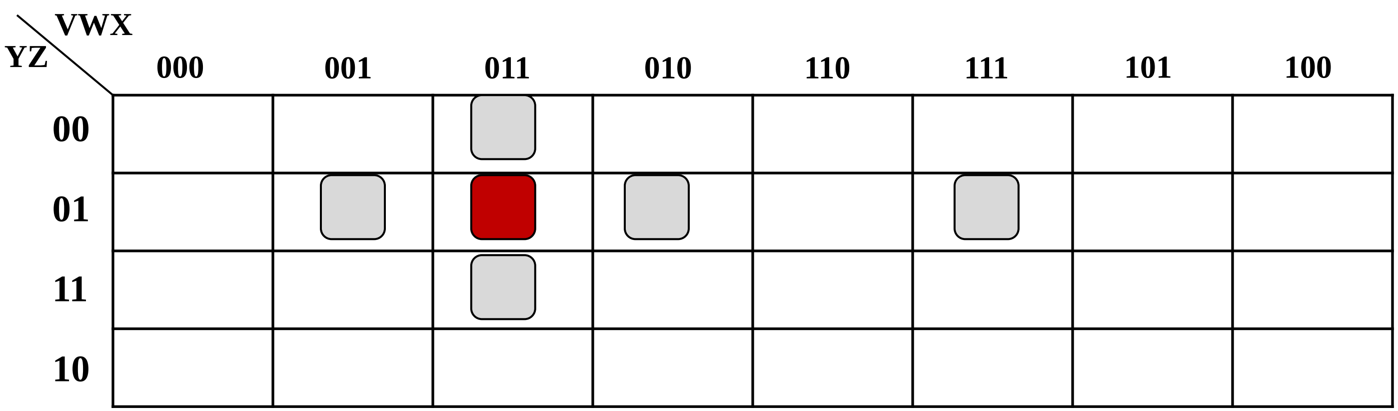
- Make groups (implicants) of 1, 2, 4, 8, ... and they must be rectangular or square in shape.
- Include the minimum number of essential prime implicants
  - Use only *essential* prime implicants (i.e. as few groups as possible to cover all 1's)
  - Ensure that you are using ***prime*** implicants (i.e. Always make groups as large as possible reusing squares if necessary)

Informational: You won't be asked to perform 5- or 6-variable K-Maps

5- & 6-VARIABLE KMAPS

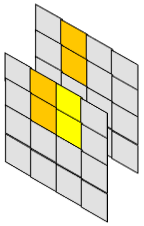
5-Variable K-Map

- If we have a 5-variable function we need a 32-square KMap.
- Will an 8x4 matrix work?
  - Recall K-maps work because adjacent squares differ by 1-bit
- How many adjacencies should we have for a given square?
- 5!! But drawn in 2 dimensions we can't have 5 adjacencies.



5-Variable Karnaugh Maps

- To represent the 5 adjacencies of a 5-variable function [e.g. f(v,w,x,y,z)], imagine two 4x4 K-Maps stacked on top of each other
  - Adjacency across the two maps



YZ \ WX	00	01	11	10
00	0 0	4 1	12 1	8 0
01	1 0	5 1	13 1	9 0
11	3 0	7 0	15 0	11 0
10	2 0	6 0	14 0	10 0

V=0

These are adjacent

YZ \ WX	00	01	11	10
00	0 0	4 1	12 0	8 0
01	1 0	5 1	13 0	9 0
11	3 0	7 0	15 0	11 0
10	2 0	6 0	14 0	10 0

V=1

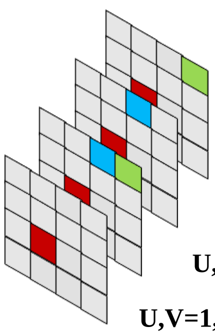
Traditional adjacencies still apply (Note: v is constant for that group and should be included) => v'xy'

Adjacencies across the two maps apply (Now v is not constant) => w'xy'

F = v'xy' + w'xy'

6-Variable Karnaugh Maps

- 6 adjacencies for 6-variables (Stack of four 4x4 maps)



U,V=0,0

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 1	(8) 0
01	(1) 0	(5) 0	(13) 0	(9) 0
11	(3) 0	(7) 1	(15) 0	(11) 0
10	(2) 0	(6) 0	(14) 0	(10) 0

U,V=0,1

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 0	(8) 1
01	(1) 0	(5) 0	(13) 0	(9) 0
11	(3) 0	(7) 1	(15) 0	(11) 0
10	(2) 0	(6) 0	(14) 0	(10) 0

U,V=1,0

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 1	(8) 1
01	(1) 0	(5) 0	(13) 0	(9) 0
11	(3) 0	(7) 1	(15) 0	(11) 0
10	(2) 0	(6) 0	(14) 0	(10) 0

U,V=1,1

YZ \ WX	00	01	11	10
00	(0) 0	(4) 0	(12) 0	(8) 0
01	(1) 0	(5) 0	(13) 0	(9) 0
11	(3) 0	(7) 1	(15) 0	(11) 0
10	(2) 0	(6) 0	(14) 0	(10) 0

Group of 2

Not adjacent

Group of 4

7-Variable K-maps and Other Techniques

- Can we have 7-variable K-Maps?
- No! We would need to see 7 adjacencies per square and we humans cannot visualize 4 dimensions
- Other computer-friendly minimization algorithms
  - Quine-McCluskey
    - Still exponential runtime
    - Minimization is NP-hard problem
  - Espresso-heuristic Minimizer
    - Achieves "good" minimization in far less time (may not be absolute minimal)