Spiral 1 / Unit 5

Karnaugh Maps

Outcomes

- I know the difference between combinational and sequential logic and can name examples of each.
- I understand latency, throughput, and at least 1 technique to improve throughput
- I can identify when I need state vs. a purely combinational function
  - I can convert a simple word problem to a logic function (TT or canonical form) or state diagram
- I can use Karnaugh maps to synthesize combinational functions with several outputs
- I understand how a register with an enable functions & is built
- I can design a working state machine given a state diagram
- I can implement small logic functions with complex CMOS gates

A new way to synthesize your logic functions

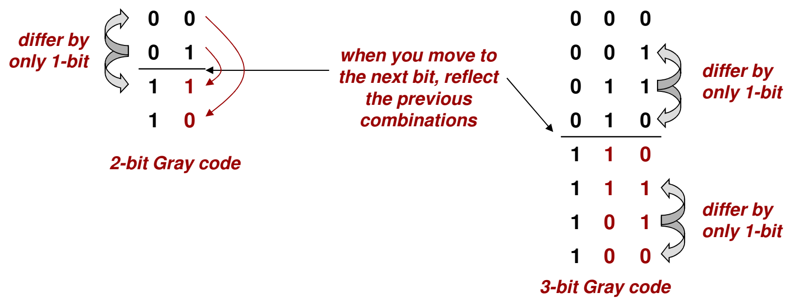
KARNAUGH MAPS

Logic Function Synthesis

- Given a function description as a T.T. or canonical form, how can we arrive at a circuit implementation or equation (i.e. perform logic synthesis)?
- First method
  - Minterms / maxterms
    - Can simplify to find minimal 2-level implementation
    - Use "off-the-shelf" decoder + 1 gate per output
- New, second method
  - Karnaugh Maps
    - Minimal 2-level implementation (though not necessarily minimal 3-, 4-, … level implementation)

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 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 will cancel

Karnaugh Map Construction

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

$F=\Sigma_{XYZ}(1,4,5,6)$

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

$G=\Sigma_{WXYZ}(1,2,3,5,6,7,9,10,11,14,15)$

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

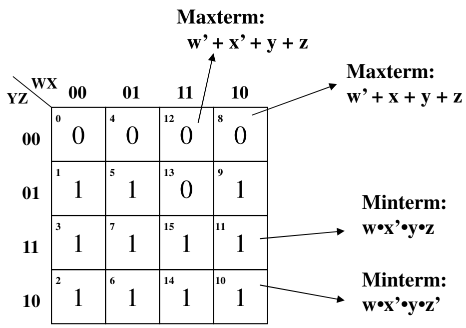
Karnaugh Maps

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

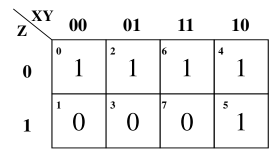
- Squares with a '1' represent minterms
- Squares with a '0' represent maxterms



Karnaugh Maps

- Groups of adjacent 1's will always simplify to smaller product term than just individual minterms

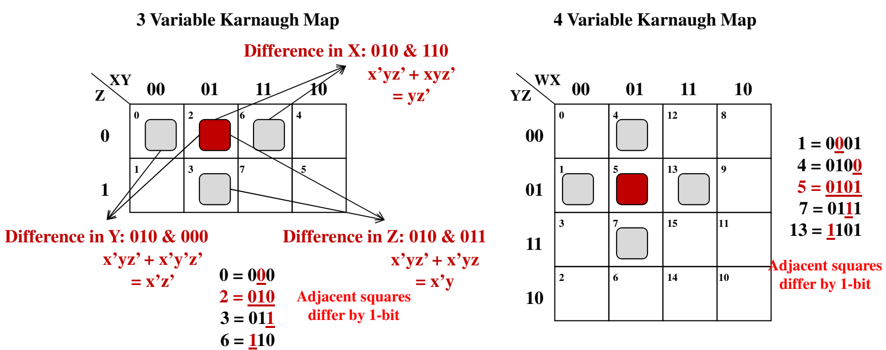
$F=\Sigma_{XYZ}(0,2,4,5,6)$



3 Variable Karnaugh Map

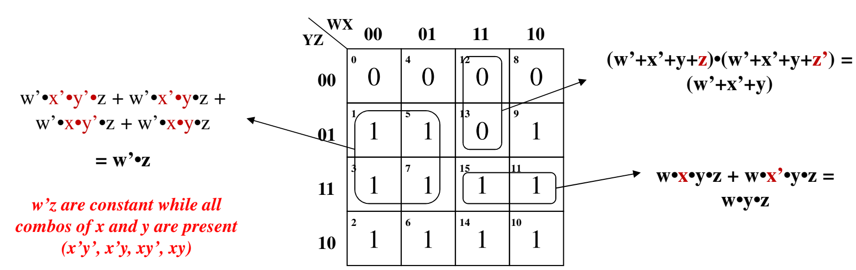
Karnaugh Maps

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



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)



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.
- Wraparounds are legal

K-Map Grouping Rules

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

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

- 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

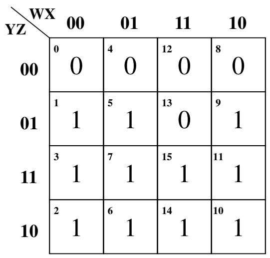
YZ \ WX	00	01	11	10
00	[0] 0	[4] 0	[12] 1	[8] 0
01	[1] 1	[5] 0	[13] 0	[9] 1
11	[3] 1	[7] 0	[15] 0	[11] 1
10	[2] 0	[6] 0	[14] 1	[10] 0

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

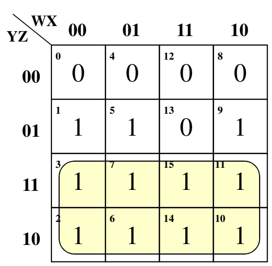


K-Map Translation Rules

- 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)

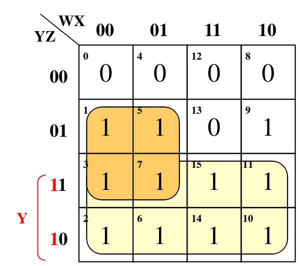
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



F =

Karnaugh Maps (SOP)

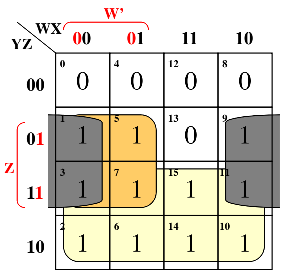
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



F = Y

Karnaugh Maps (SOP)

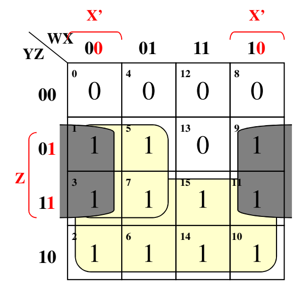
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



F = Y + W'Z + …

Karnaugh Maps (SOP)

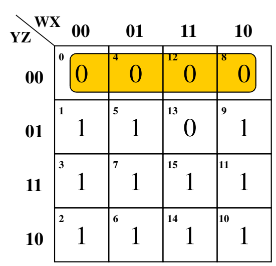
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



F = Y + W'Z + X'Z

Karnaugh Maps (POS)

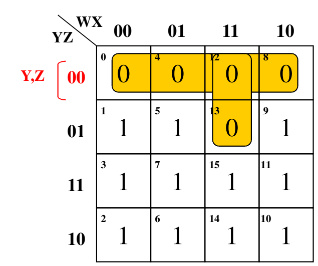
W	X	Y	Z	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**



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**



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, Y

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

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

Karnaugh Maps

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

X', WX, X', Z', Z, Z'

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

F = X'Z + WXZ'

X', X', Z', Z'

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

F = X'Z'

Exercises

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

F_{SOP}=

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

F_{POS}=

$P=\Sigma_{XYZ}(2,3,5,7)$

P=

No Redundant Groups

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

Multiple Minimal Expressions

- For some functions, ____________ minimal groupings exist which will lead to alternate _________ expressions…__________

D2D1 \ D8D4	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	1	0
10	1	1	0	0

Best way to cover this '1'??

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	0	0
01	1	1	0	0
11	1	1	1	1
10	0	0	1	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**

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

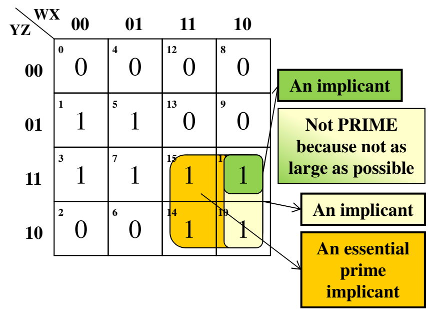
An implicant

Not PRIME because not as large as possible

An implicant

Implicant Examples

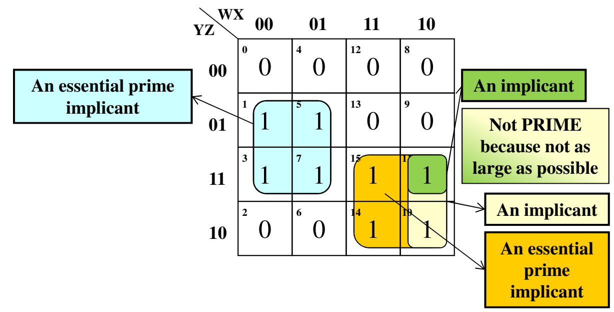
W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



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

Implicant Examples

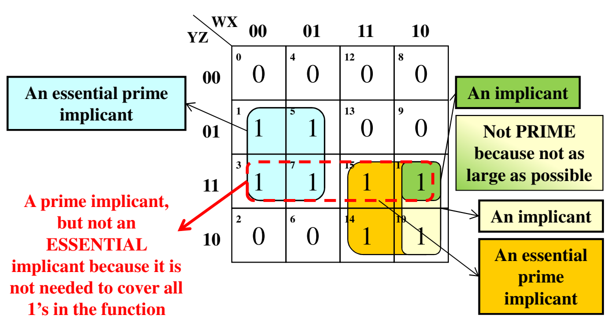
W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



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

Implicant Examples

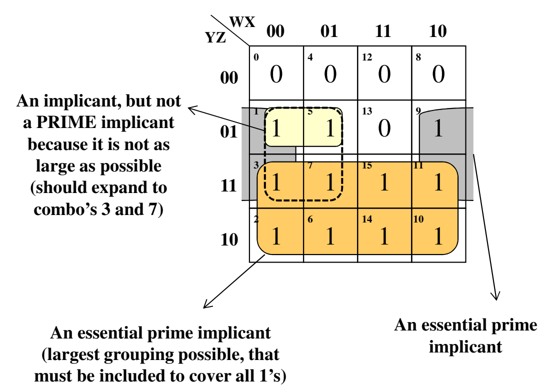
W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



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

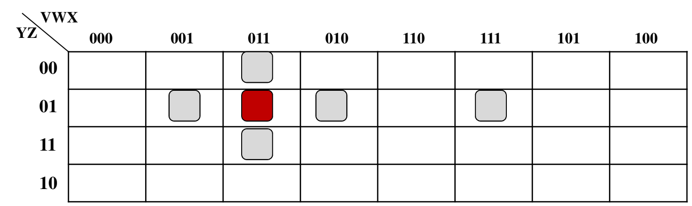


K-Map Grouping Rules

- 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 ____________________ 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)
- Wraparounds are legal

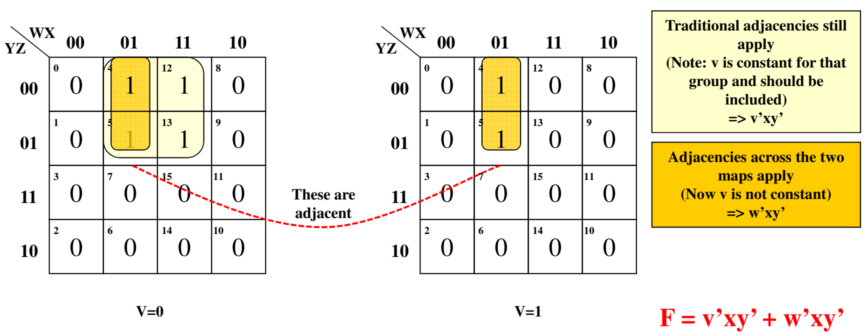
5-Variable K-Map

- 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.



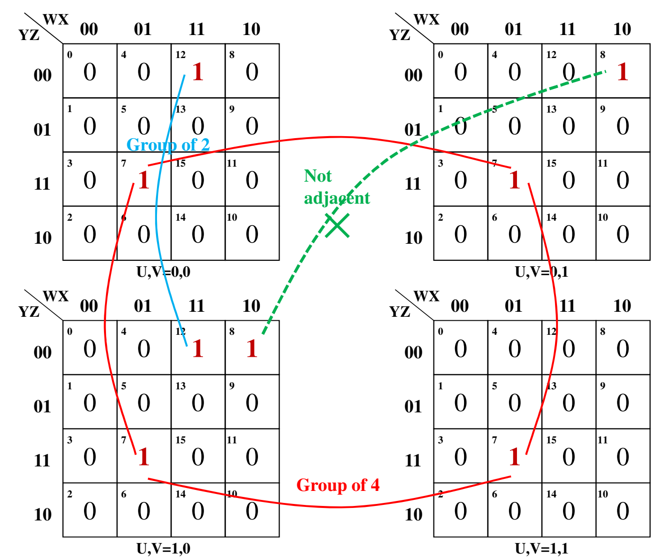
5-Variable Karnaugh Maps

- 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



6-Variable Karnaugh Maps

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

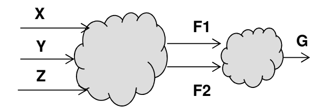


Don't-Cares

- Sometimes there are certain input combinations that are illegal (i.e. in BCD, 1010 – 1111 can ________________)
- 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 ___ or groups of ___ when grouping in K-Maps
  - Use them to make as big of groups as possible

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	

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**

D2D1 \ D8D4	00	01	11	10
00	0	0	d	1
01	0	0	d	1
11	0	1	d	d
10	0	0	d	d

$GT6_{SOP}$=

D2D1 \ D8D4	00	01	11	10
00	0	0	d	1
01	0	0	d	1
11	0	1	d	d
10	0	0	d	d

$GT6_{POS}$=

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**

D2D1 \ D8D4	00	01	11	10
00	0	0	d	1
01	0	0	d	1
11	0	1	d	d
10	0	0	d	d

$GT6_{SOP}$=

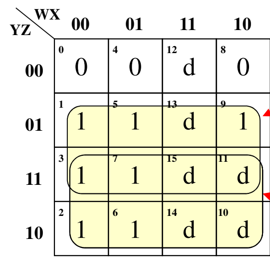
D2D1 \ D8D4	00	01	11	10
00	0	0	d	1
01	0	0	d	1
11	0	1	d	d
10	0	0	d	d

$GT6_{POS}$=

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**

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



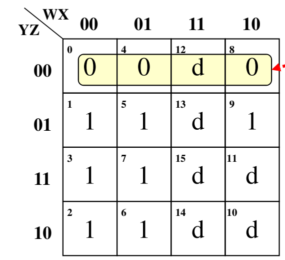
F = Y'Z + Y

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**

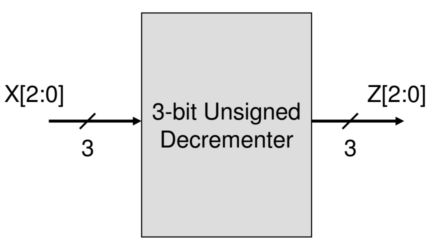
You can use "d's" when grouping 0's and converting to POS



F = Y+Z

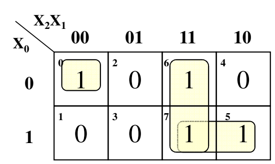
Designing Circuits w/ K-Maps

- 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.

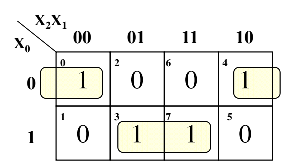


3-bit Number Decrementer

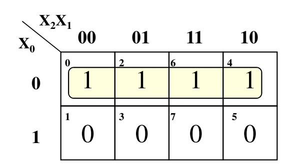
X_2	X_1	X_0	Z_2	Z_1	Z_0
0	0	0	**1**	**1**	**1**
0	0	1	**0**	**0**	**0**
0	1	0	**0**	**0**	**1**
0	1	1	**0**	**1**	**0**
1	0	0	**0**	**1**	**1**
1	0	1	**1**	**0**	**0**
1	1	0	**1**	**0**	**1**
1	1	1	**1**	**1**	**0**



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



$Z_1 = X_1'X_0' + X_1X_0$



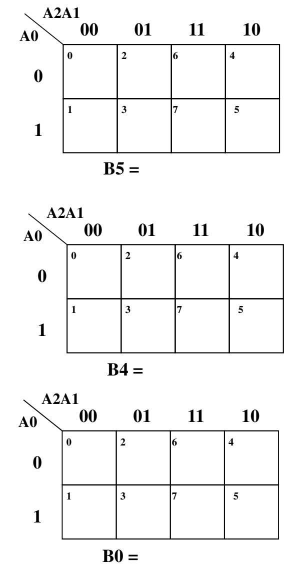
$Z_0 = X_0'$

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

	Inputs			Outputs						
A	A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	$B=A^2$



3-bit Squaring Circuit