

Unit 9

Implementing Combinational Functions with Karnaugh Maps or Memories

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)

A new way to synthesize your logic functions

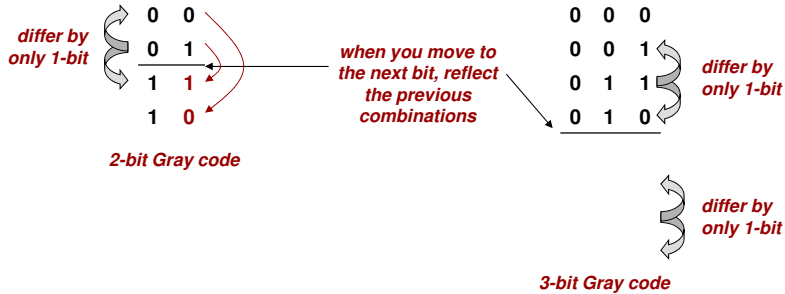
KARNAUGH MAPS

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



2-bit Gray code

3-bit Gray code

Karnaugh Maps

- If used correctly, will always yield a minimal, _____ 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 _____ 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

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

Z \ XY	00	01	11	10
0	0	0	1	1
1	1	0	0	1

3 Variable Karnaugh Map

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

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

4 Variable Karnaugh Map

Karnaugh Maps

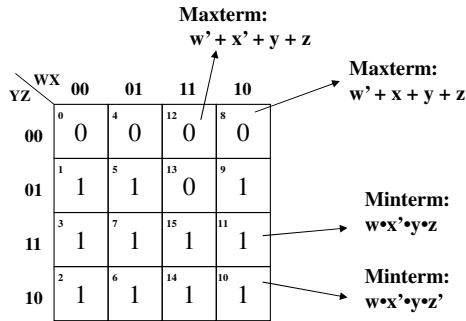
W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



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

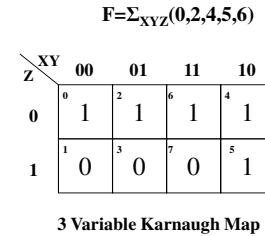
Karnaugh Maps

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



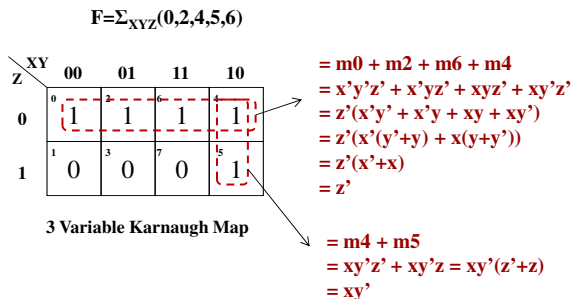
Karnaugh Maps

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



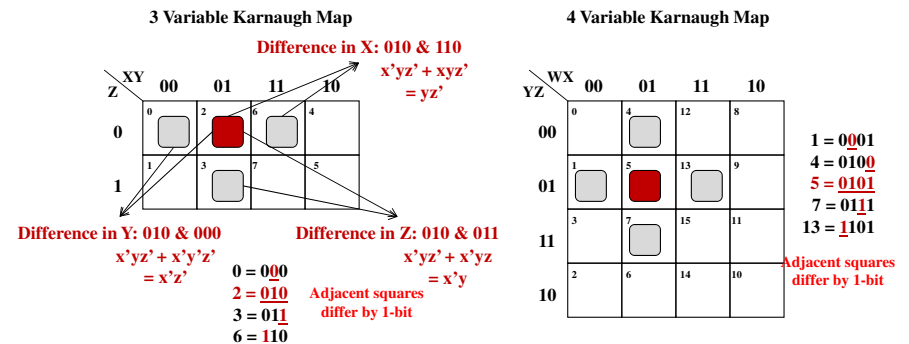
Karnaugh Maps

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



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)

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

$w'x'y'z + w'x'yz + w'xy'z + w'xyz = w'z$
w'z are constant while all combos of x and y are present (x'y', x'y, xy', xy)

$(w'+x'+y+z)(w'+x'+y+z) = (w'+x'+y)$

$w'xy'z + w'x'yz = w'y'z$

K-Map Grouping Rules

- Cover the 1's [=on-set] or 0's [=off-set] with _____ groups as possible, but make those groups _____ 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 _____, ... and they must be rectangular or square in shape.
- Wrapping is legal

Group These K-Maps

		XY			
		00	01	11	10
Z	0	0	1	0	0
	1	1	0	0	0

		XY			
		00	01	11	10
Z	0	1	1	0	0
	1	1	0	0	0

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

Karnaugh Maps

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

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

$$F = X'Z + WXZ'$$

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

$$F = X'Z'$$

Group This

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

	WX			
YZ	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 =

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

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

$$F = Y + W'Z + \dots$$

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

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

$F = (Y+Z)$

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	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

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

$F = X'Z + WXZ'$

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

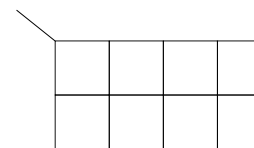
YZ \ WX	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	0	0	0
10	1	0	1	1

$F_{SOP} =$

$P = \sum_{xyz}(2,3,5,7)$

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

$F_{POS} =$



$P =$

No Redundant Groups

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

Multiple Minimal Expressions

- For some functions, _____ groupings exist which will lead to alternate minimal _____ ...Pick one

	D8D4			
	00	01	11	10
D2D1				
00	0 ⁰ 0	4 ⁰ 0	12 ¹ 1	8 ¹ 1
01	1 ¹ 0	5 ⁰ 0	13 ¹ 1	9 ¹ 1
11	3 ¹ 1	7 ¹ 1	15 ¹ 1	11 ⁰ 0
10	2 ¹ 1	6 ¹ 1	14 ⁰ 0	10 ⁰ 0

Best way to cover this '1'??

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

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

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

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

An implicant

Not PRIME because not as large as possible

An implicant

An essential prime implicant

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

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

An essential prime implicant

An implicant

Not PRIME because not as large as possible

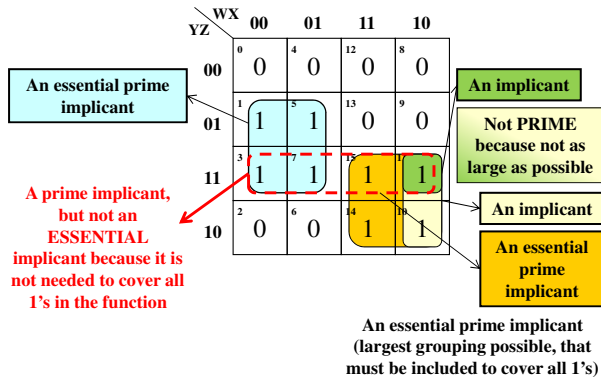
An implicant

An essential prime implicant

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

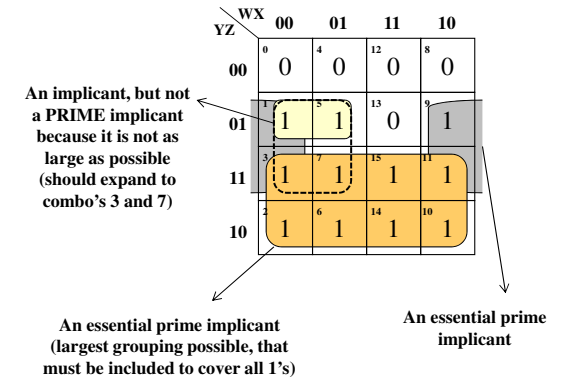
Implicant Examples

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



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 *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)

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?
- ___!! But drawn in 2 dimensions we can't have ___ adjacencies.

YZ \ VWX	000	001	011	010	110	111	101	100
00								
01								
11								
10								

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

V=0

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

V=1

These are adjacent

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

Adjacencies across the two maps apply
(Now v is not constant)
 $\Rightarrow w'xy'$

$F = v'xy' + w'xy'$

6-Variable Karnaugh Maps

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

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

U,V=0,0

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

U,V=0,1

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

U,V=1,0

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

U,V=1,1

Not adjacent

Group of 4

DON'T CARE OUTPUTS

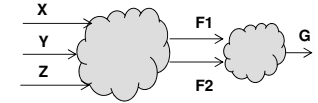
Don't-Cares

- Sometimes there are certain input combinations that are illegal (i.e. in BCD, 1010 – 1111 can never occur)
- 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	

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

		D8D4				GT6 _{SOP}
D2D1		00	01	11	10	
00	0	0	0	d	1	
	1	0	0	d	1	
01	3	7	1	d	d	
	2	6	0	d	d	

		D8D4				GT6 _{POS}
D2D1		00	01	11	10	
00	0	0	0	d	1	
	1	0	0	d	1	
01	3	7	1	d	d	
	2	6	0	d	d	

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



		WX			
YZ		00	01	11	10
00	0	0	0	d	0
	1	1	1	d	1
01	3	7	1	d	d
	2	6	1	d	d

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

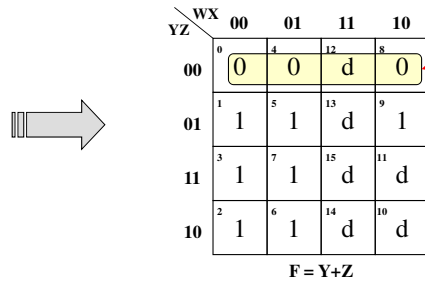
Use these 4 "d's" to make a group of 8

$$F = Z + Y$$

Don't Cares

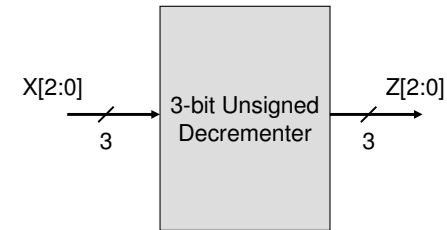
W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	d
1	0	1	1	d
1	1	0	0	d
1	1	0	1	d
1	1	1	0	d
1	1	1	1	d

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



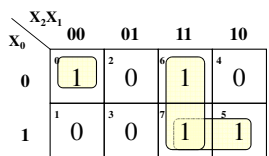
Designing Circuits w/ K-Maps

- 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 decremter ($Z = X - 1$)
 - If $X[2:0] = 000$ then $Z[2:0] = 111$, etc.

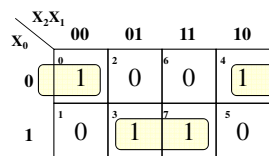


3-bit Number Decrementer

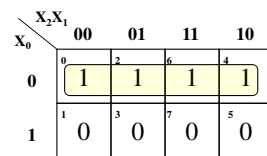
X_2	X_1	X_0	Z_2	Z_1	Z_0
0	0	0	1	1	1
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0



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



$$Z_1 = X_1'X_0' + X_1X_0$$



$$Z_0 = X_0'$$

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

A		Inputs			Outputs						
		A ₂	A ₁	A ₀	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	B=A ²

A ² A ¹		A ⁰			
		00	01	11	10
A ⁰	0	0	2	6	4
	1	1	3	7	5

B₅ =

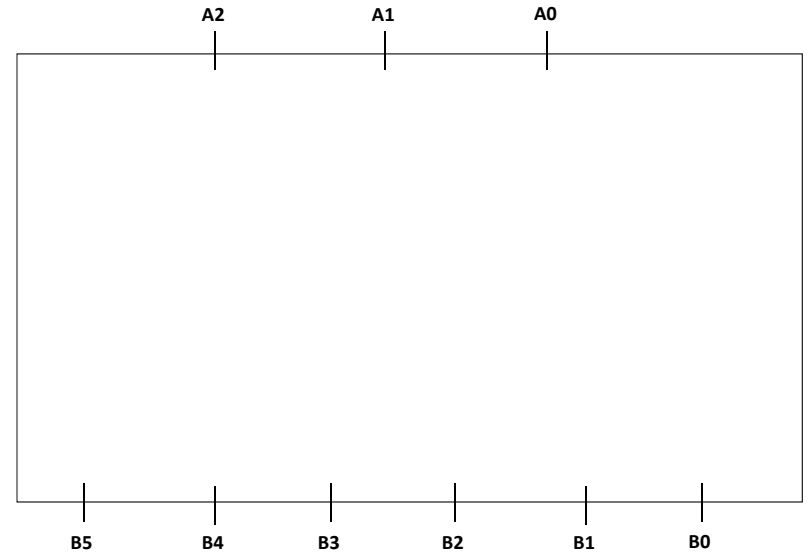
A ² A ¹		A ⁰			
		00	01	11	10
A ⁰	0	0	2	6	4
	1	1	3	7	5

B₄ =

A ² A ¹		A ⁰			
		00	01	11	10
A ⁰	0	0	2	6	4
	1	1	3	7	5

B₀ =

3-bit Squaring Circuit



USING MEMORIES TO BUILD COMBINATIONAL CIRCUITS

Dimensions and Operations

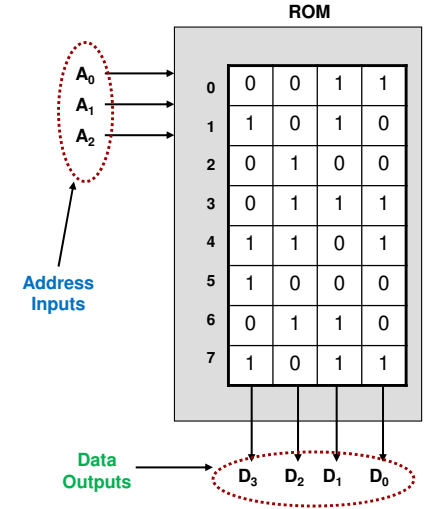
MEMORY BASICS

Memories

- Memories store (write) and retrieve (read) data
 - Read-Only Memories (ROM's): Can only retrieve data (contents are initialized and then cannot be changed)
 - Read-Write Memories (RWM's): Can retrieve data and change the contents to store new data

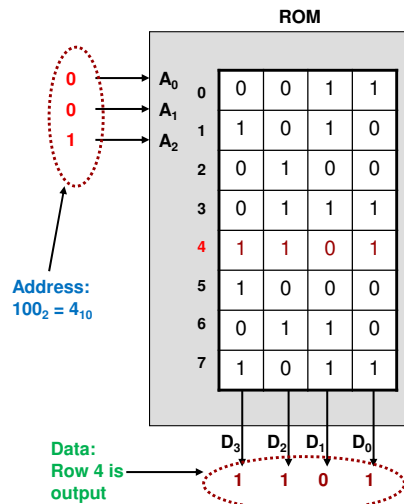
ROM's

- Memories are just _____ of data with rows and columns
- When data is read, one entire _____ of data is read out
- The row to be read is selected by putting a binary number on the _____ inputs



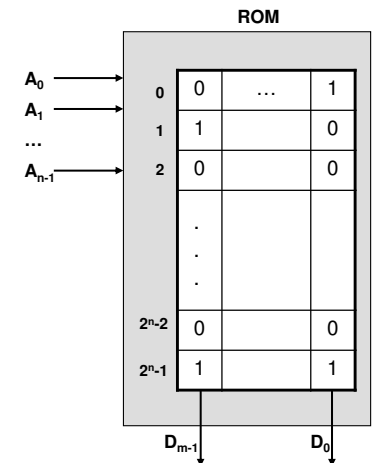
ROM's

- Example
 - Address = 4 dec. = 100 bin. is provided as input
 - ROM outputs data in that row (1101 bin.)



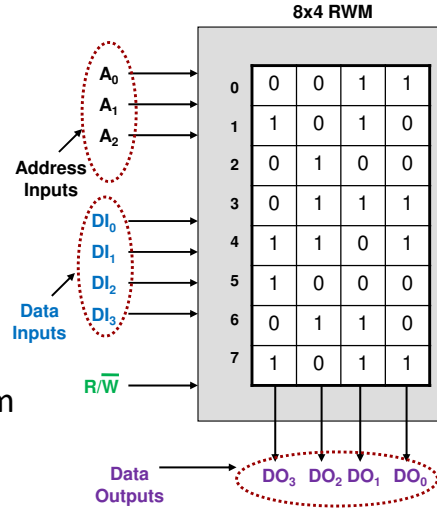
Memory Dimensions

- Memories are named by their dimensions:
 - Rows x Columns
- n rows and m columns \Rightarrow $n \times m$ ROM
- 2^n rows \Rightarrow n address bits
- ...or... k rows \Rightarrow $\log_2 k$ address bits
- m cols \Rightarrow m data outputs



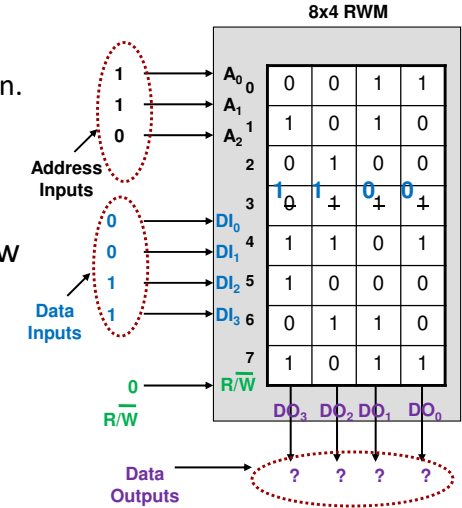
RWM's

- Writable memories provide a set of **data inputs** for **write data** (as opposed to the **data outputs** for **read data**)
- A control signal R/\bar{W} (1=READ / 0 = WRITE) is provided to tell the memory what operation the user wants to perform



RWM's

- Write example
 - Address = 3 dec. = 011 bin.
 - DI = 12 dec. = 1100 bin.
 - $R/\bar{W} = 0 \Rightarrow$ Write op.
- Data in row 3 is overwritten with the new value of 1100 bin.



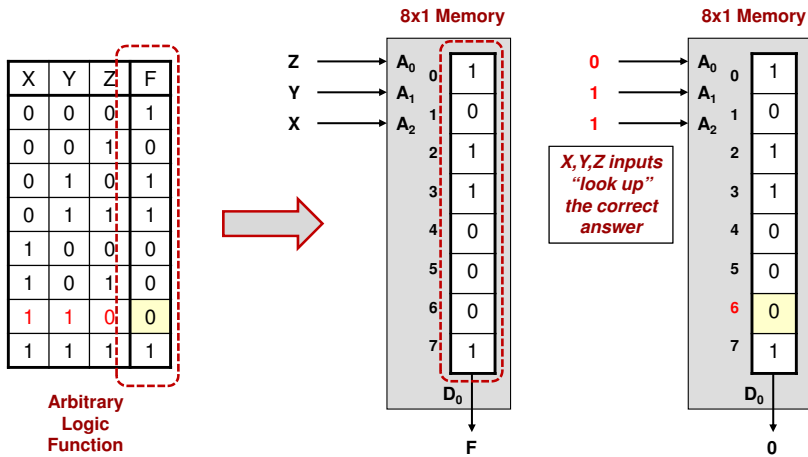
Look-up tables...

USING MEMORIES TO BUILD COMBINATIONAL FUNCTIONS

Memories as Look-Up Tables

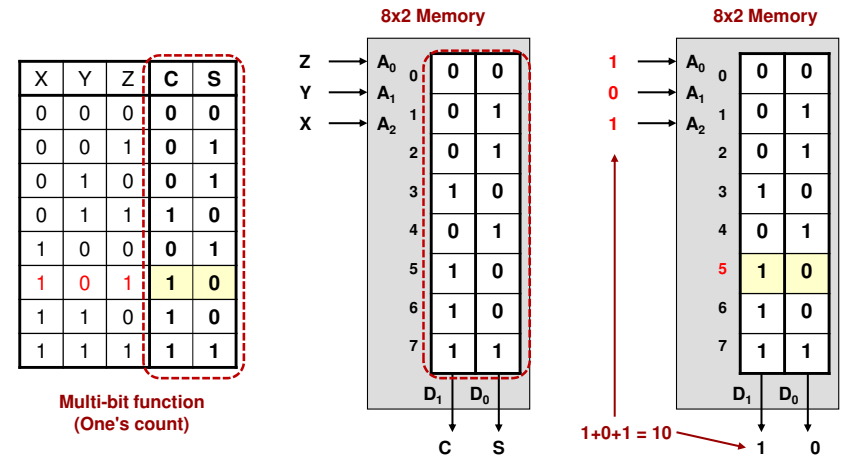
- One major application of memories in digital design is to use them as LUT's (Look-Up Tables) to implement logic functions
 - This is the core technology used by FPGAs (Field-Programmable Gate Arrays) which are used pervasively in robotics, space applications, networking, and even in search engines
- **Idea:** Use a memory to hold the _____ of a function and feed the inputs of the function to the _____ inputs to " _____ " the answer

Implementing Functions w/ Memories



Use a memory with the same dimensions as 'output' side of the truth table. It's almost TOO easy.

Implementing Functions w/ Memories



Use a memory with the same dimensions as 'output' side of the truth table. It's almost TOO easy.

3-bit Squaring Circuit

- **Q:** What size memory would you use to build our 3-bit squaring circuit?
- **A:** ___ memory
- **Q:** What would you connect to the address inputs of the memory?
- **A:** _____
- **Q:** What bits would you program into row 5 of the memory?
- **A:** _____

A	Inputs				Outputs						B=A ²
	A ₂	A ₁	A ₀	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀		
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	1
2	0	1	0	0	0	0	1	0	0	0	4
3	0	1	1	0	0	1	0	0	1	0	9
4	1	0	0	0	1	0	0	0	0	0	16
5	1	0	1	0	1	1	0	0	1	0	25
6	1	1	0	1	0	0	1	0	0	0	36
7	1	1	1	1	1	0	0	0	1	0	49

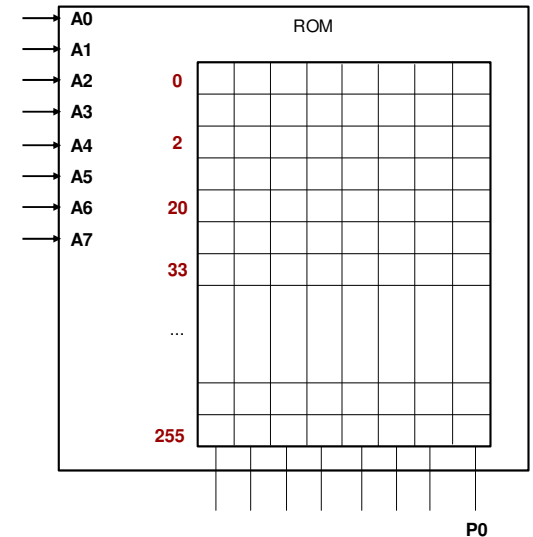
4x4 Multiplier Example

Determine the dimensions of the memory that would be necessary to implement a 4x4-bit unsigned multiplier with inputs X[3:0] and Y[3:0] and outputs P[?:0]

Question: How many bits are needed for P?

Question: What are the contents of the numbered rows?

Example:
 $X_3X_2X_1X_0=0010$
 $Y_3Y_2Y_1Y_0=0001$
 $P = X * Y = 2 * 1 = 2$
 $= 00010$



Implementing Functions w/ Memories

- To implement a function w/ n -variables and m outputs
- Just place the output truth table values in the memory
- Memory will have dimensions: 2^n rows and m columns
 - Still does not scale terribly well (i.e. n -inputs requires memory w/ 2^n outputs)
 - But it is easy and since we can change the contents of memories it allows us to create "reconfigurable" logic
 - This idea is at the heart of FPGAs