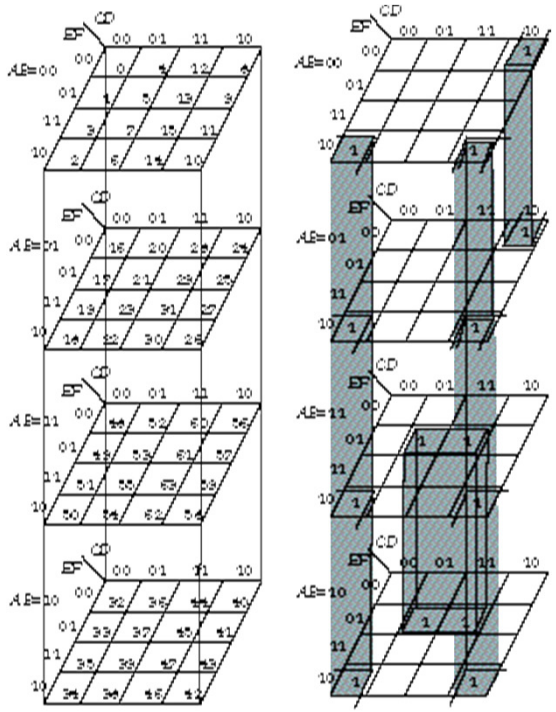
Trust me…its easier!!!
CSCI 255



(a) Six-variable K-map (b) Example

Figure 2.52 Six-variable K-map and example.

http://www2.elo.utfsm.cl/~lsb/elo211/aplicaciones/katz/chapter2/chapter02.doc3.html

K-maps/SoP/PoS

- K-Maps, short for: Karnaugh Maps
- Maurice Karnaugh from Bell Labs
- We all have learned to love this guys at some point
- Now that we have done plenty of Boolean, K-maps becomes a better way to reduce expressions
- Optimization of expression: better circuit designs

3V K-Map

- K-maps: best way to explain them is to use an example.
- Ex: Random Truth Table

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

1 – Learn to setup the K-map

- We have three-inputs (A,B,C); therefore, we need a 3-variable K-Map
- We are trying to derive an optimized expression for output (OUT)
- All outcomes of the truth table must be mapped to the K-map

3V K-Map

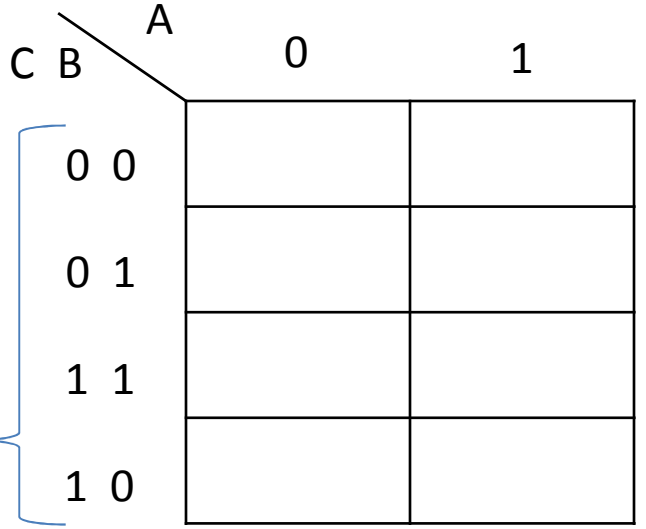
msb lsb

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

1 – Learn to setup the K-map (con't)

- The map looks like a matrix table
- Labeling ROWS & COLUMNS

Usually you want to have the MSBits as rows & LSBits as columns… Go to class and find out why



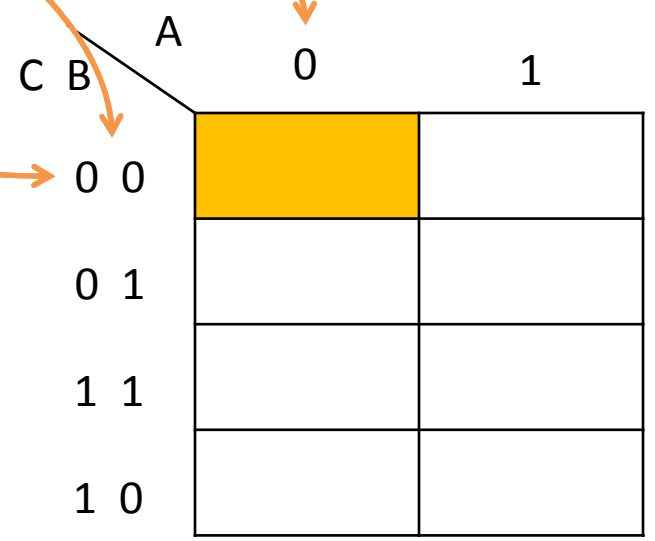
For bits C & B, the order of the bits are not in sequence. The difference between the each row is the 1-bit difference of change between C & B.

3V K-Map

2 – How to read coordinates on K-map

- Each element on the map represents a row on the truth table

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

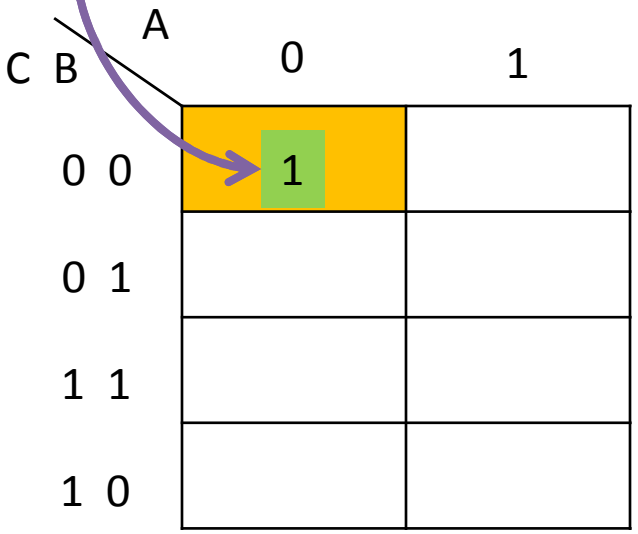


3V K-Map

2 – How to read coordinates on K-map (con't)

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Once coordinates have been identified, place output value to the element on the k-map

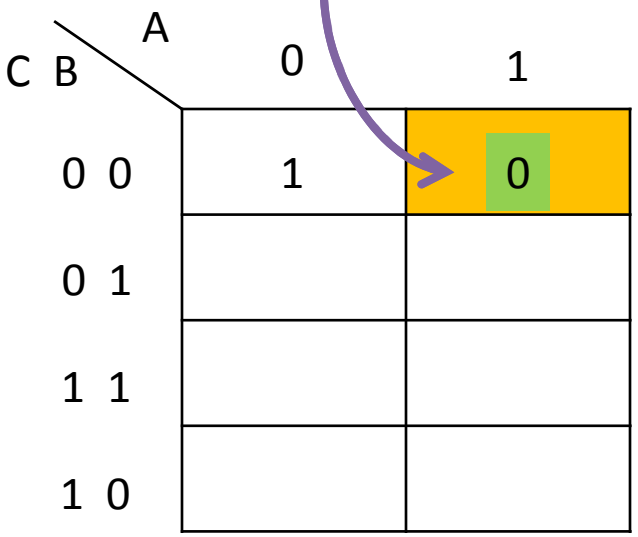


3V K-Map

2 – How to read coordinates on K-map (con't)

- Fill in the rest…

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

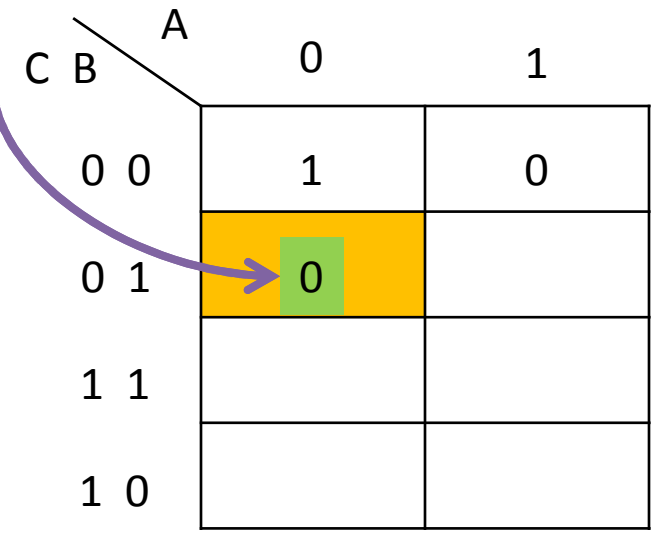


3V K-Map

2 – How to read coordinates on K-map (con't)

- Fill in the rest…

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

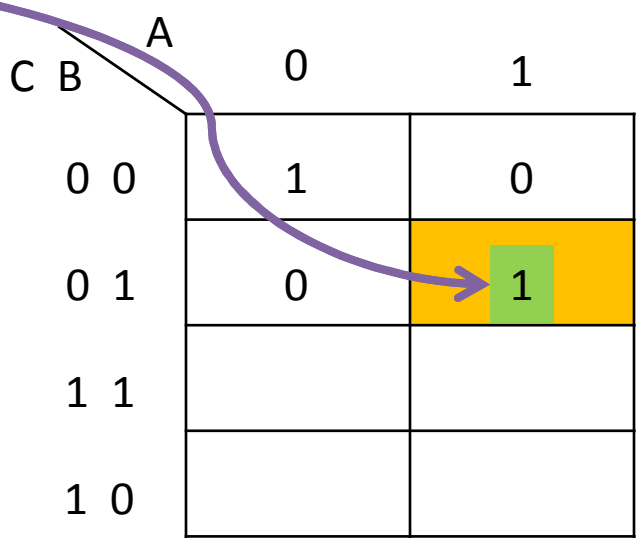


3V K-Map

2 – How to read coordinates on K-map (con't)

- Fill in the rest…

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

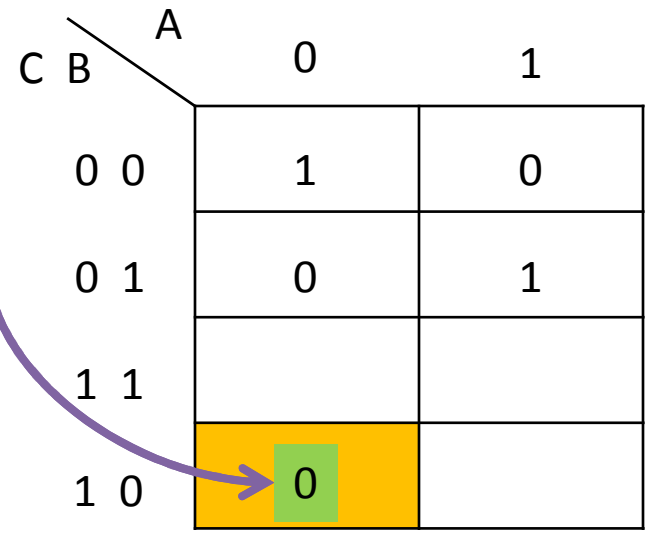


3V K-Map

2 – How to read coordinates on K-map (con't)

- Fill in the rest…

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

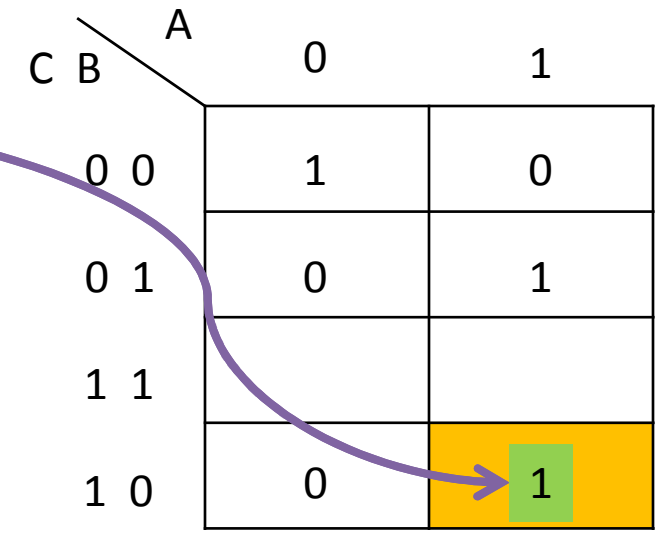


3V K-Map

2 – How to read coordinates on K-map (con't)

- Fill in the rest…

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

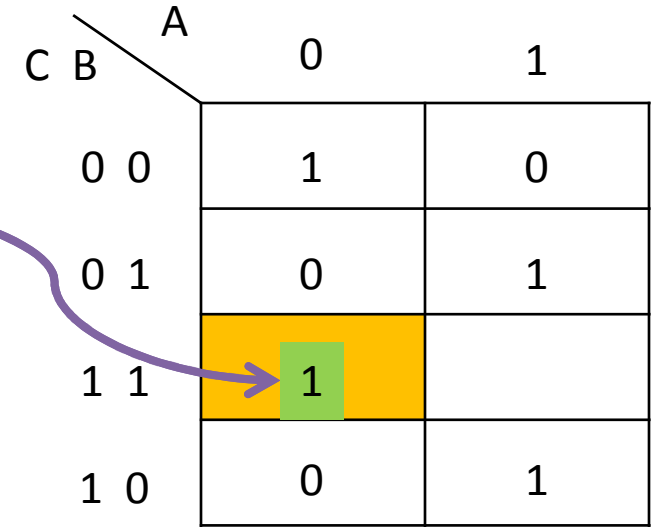


3V K-Map

2 – How to read coordinates on K-map (con't)

- Fill in the rest…

C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

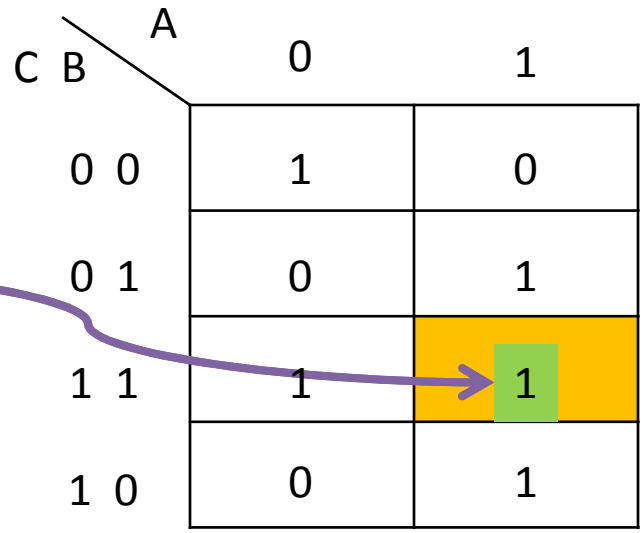


3V K-Map

2 – How to read coordinates on K-map (con't)

- Fill in the rest…

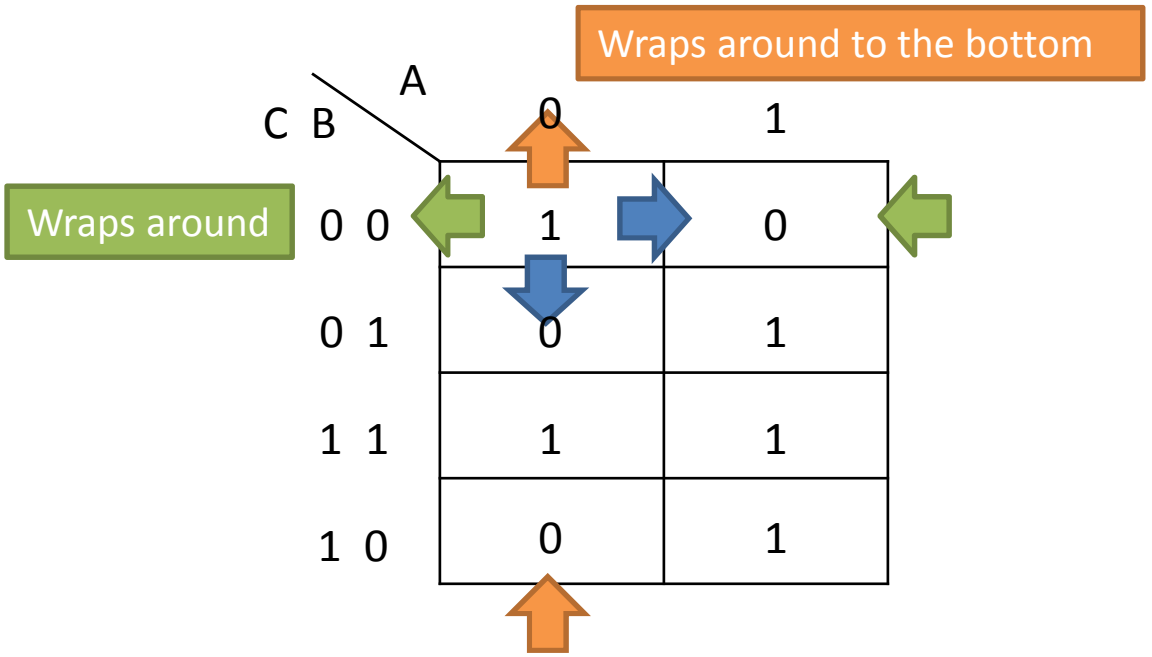
C	B	A	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



3V K-Map

3 – Grouping the ones:

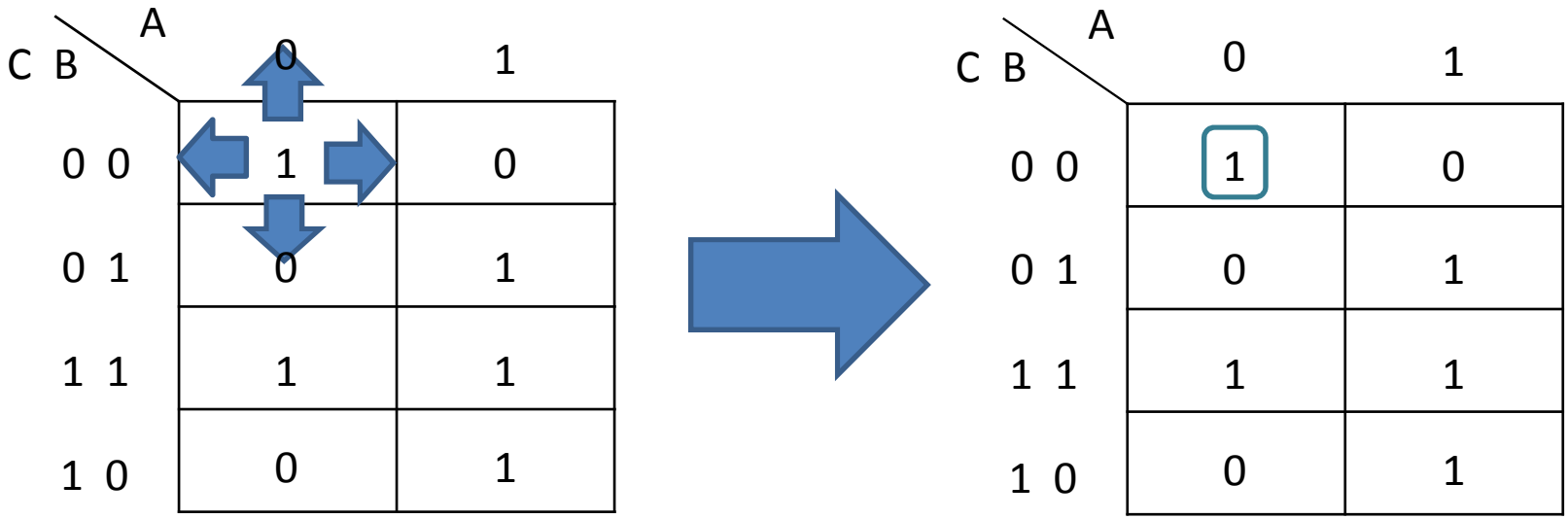
- Find the maximum group of 1s on the map
- The size of the groups must be of log2
  - i.e: 1, 2, 4, 8, 16,…
- A 1 can only be grouped with neighboring 1s, why? ONE BIT OFFSET
- The neighboring of a 1 can be found around all sides: up,down,left,right



3V K-Map

3 – Grouping the ones (con't)

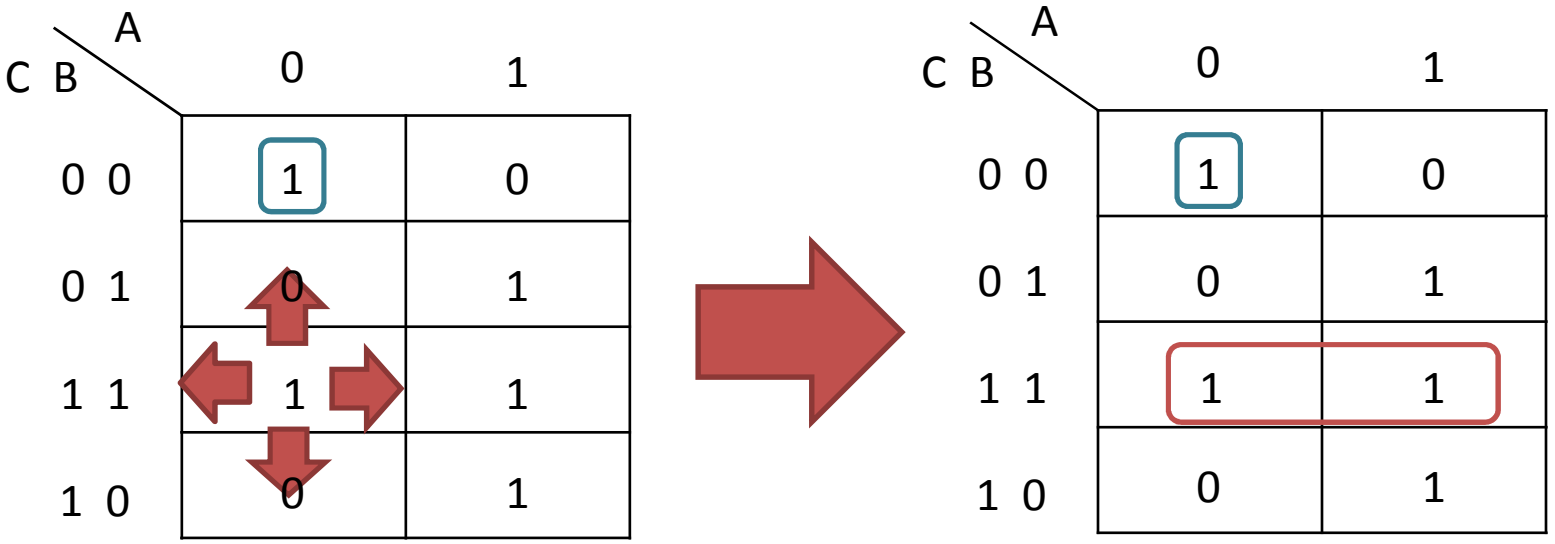
- For this first 1, all neighbors are 0's
- Therefore, the largest group of this 1 is one



3V K-Map

3 – Grouping the ones (con't)

- Move to another free 1, check its neighbors, find max grouping

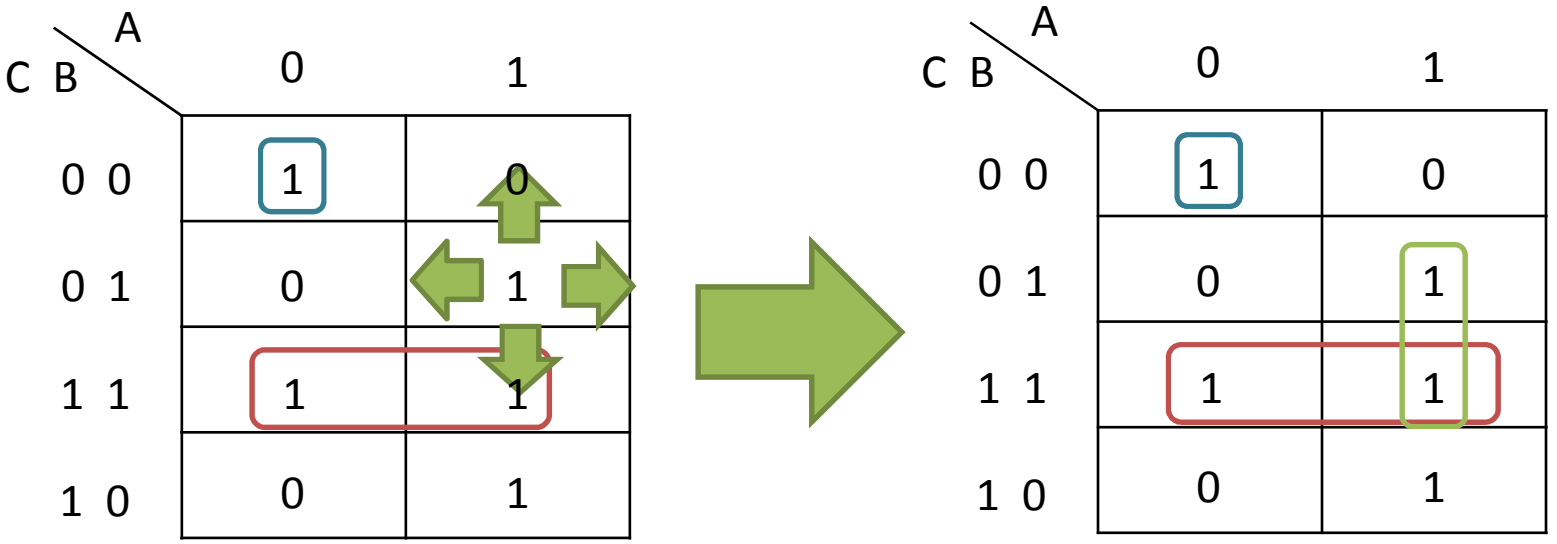


- In this grouping, a neighboring 1 exists to the right (or left…same 1).
- Maximum group is 2

3V K-Map

3 – Grouping the ones (con't)

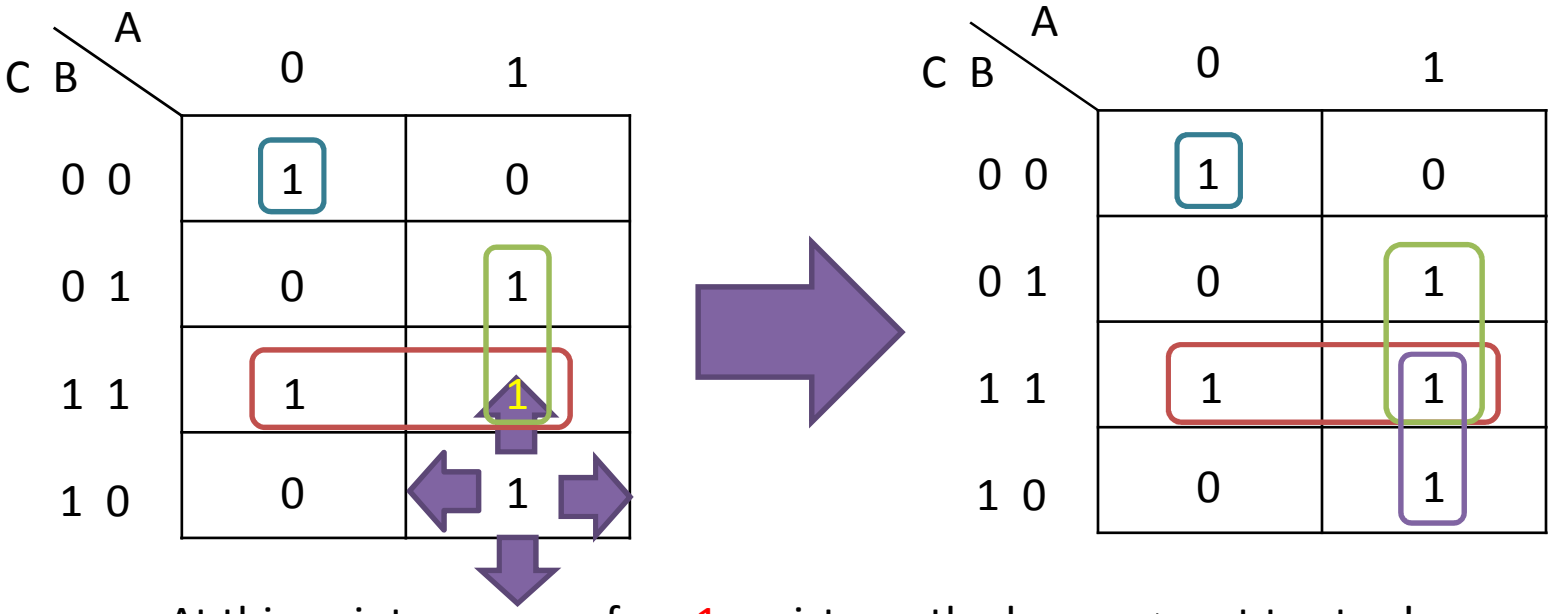
- Move to another free 1, check its neighbors, find max grouping



- A free 1 can use another 1 that is already part of another grouping
- This is part of maximizing the grouping → optimizing expression

3 – Grouping the ones (con't)

- Move to another free 1, check its neighbors, find max grouping

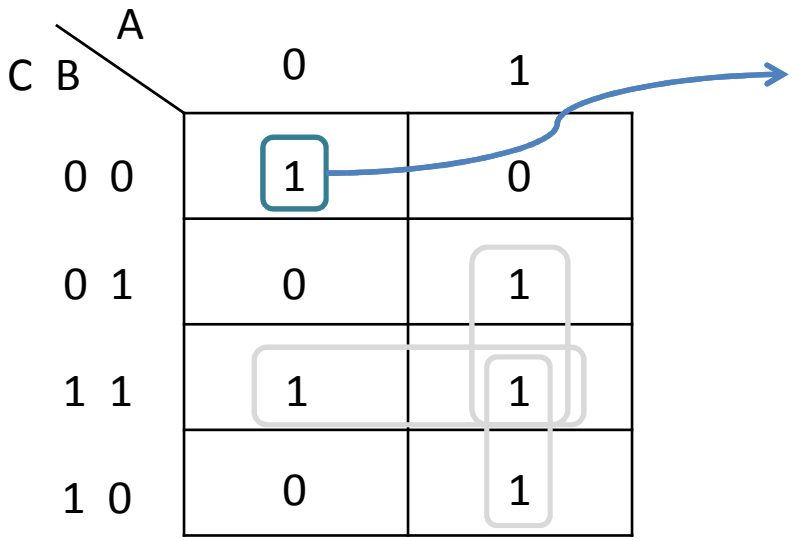


- At this point, no more free 1s exists on the k-map -> got to stop!
- Four groups were form with the k-map

3V K-Map

4 – Derive Expression from K-map

- Once all grouping are formed, need to derived OUT's expression
- Each group represents a PRODUCT (AND) term of the input variables
- We derived the term of the 1st group:

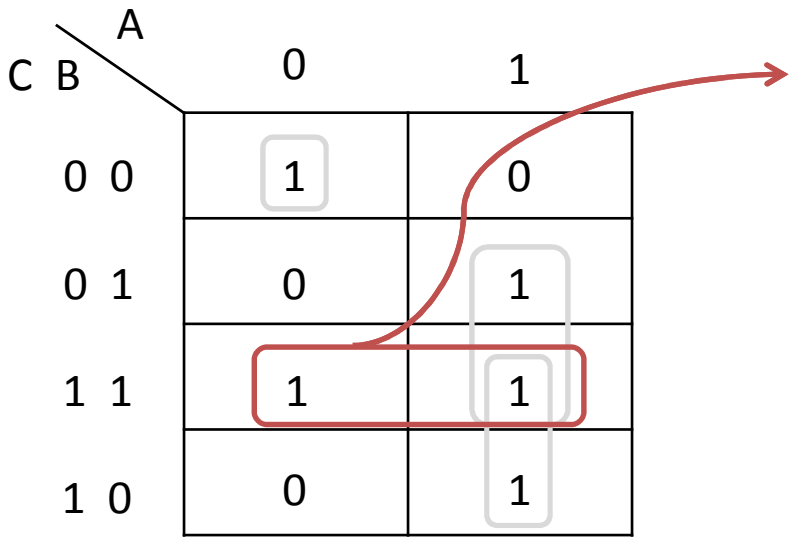


For this grouping, we observe:

- It consists of a single element
- Within this group:
  - A=0, B=0 & C=0
- Values of A,B,C remain constant
- Therefore, no need to cancel any of the input variables
- At the end, the term of this group is: A'B'C'
- They are complemented because of the values of A, B & C = 0

3V K-Map

4 – Derive Expression from K-map (con't)

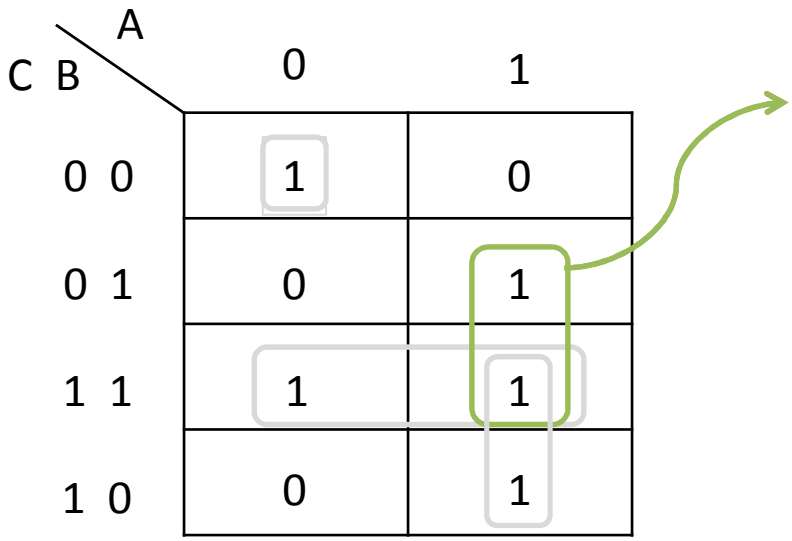


For this group, we observe:

- Size of group: 2
- Within this group:
  - B=1, C=1, but A: changes
- Because A changes values, it is not included in the expression
- At the end, the term of this group is: **BC**
- They are not complemented because of the values of B & C = 1

3V K-Map

4 – Derive Expression from K-map (con't)

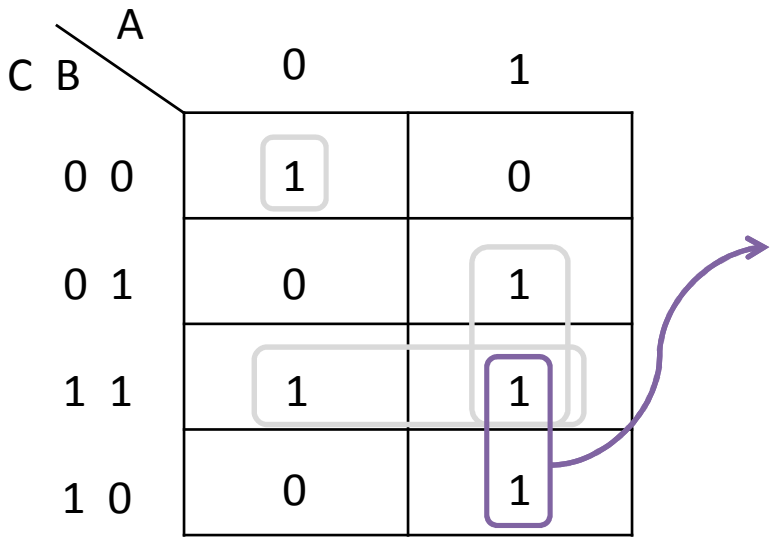


For this group, we observe:

- Size of group: 2
- Within this group:
  - B=1, A=1, but C: changes
- Because C changes values, it is not included in the expression
- At the end, the term of this group is: AB
- They are not complemented because of the values of A & B = 1

3V K-Map

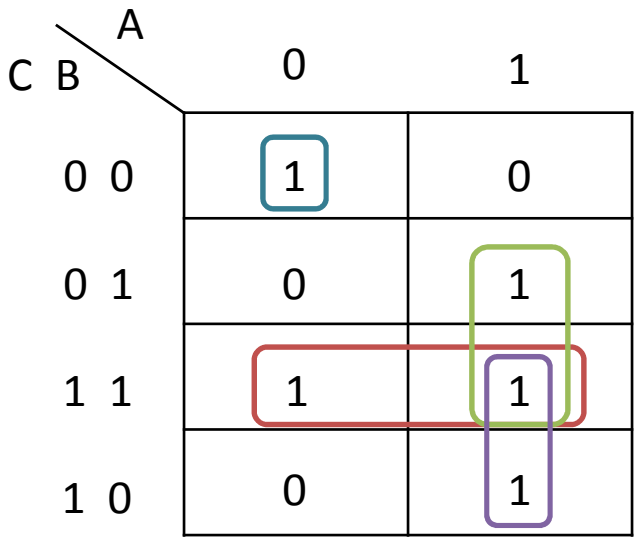
4 – Derive Expression from K-map (con't)



For this group, we observe:

- Size of group: 2
- Within this group:
  - C=1, A=1, but B: changes
- Because B changes values, it is not included in the expression
- At the end, the term of this group is: AC
- They are not complemented because of the values of A & C = 1

4 – Derive Expression from K-map (con't)



When putting it all together, we end up with an expression:

OUT = A'B'C + BC + AB + AC

….if we compare this with the traditional way… (next slide)

- If we look back at the Random Truth Table
- We extract the terms when the output 'OUT' is true or Sum-of-Products (SoP… on this a bit later)

C	B	A	OUT	
0	0	0	1	A'B'C'
0	0	1	0	
0	1	0	0	
0	1	1	1	ABC'
1	0	0	0	
1	0	1	1	AB'C
1	1	0	1	A'BC
1	1	1	1	ABC

OUT= A'B'C' + ABC' + AB'C + A'BC + ABC

Factor out BC, Theorem 5

OUT = A'B'C' + ABC' + AB'C + BC

Theorem 11D

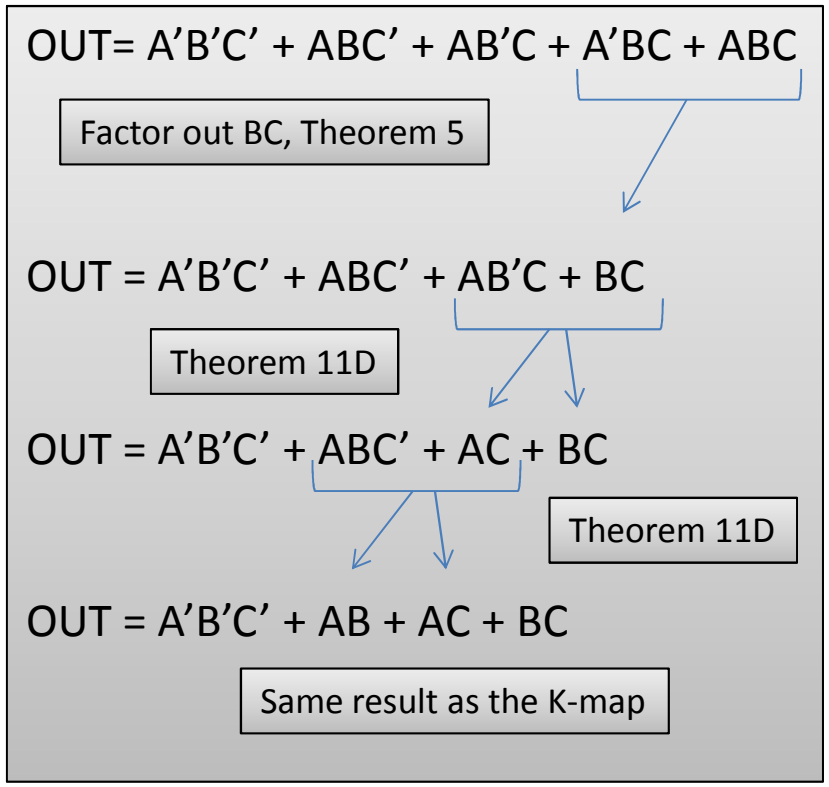
OUT = A'B'C' + ABC' + AC + BC

Theorem 11D

OUT = A'B'C' + AB + AC + BC

Same result as the K-map

- You may say it wasn't easier or faster to do the K-map vs. Boolean
- However, if the truth table expands to 4/5/6/… input variables -> terms are a bit longer
- More input variables -> cover more outcomes
- If more outcomes are true, more product (AND) terms exists -> more Booleans steps -> more mistakes -> may not reach optimum expression



4V K-Map

- Up a level …:::> 4-variable K-map, let's use another random truth table
- Since there are 4 variables: 2 variables for ROWS (MSBits), 2 variables for COLUMNS (LSBits)

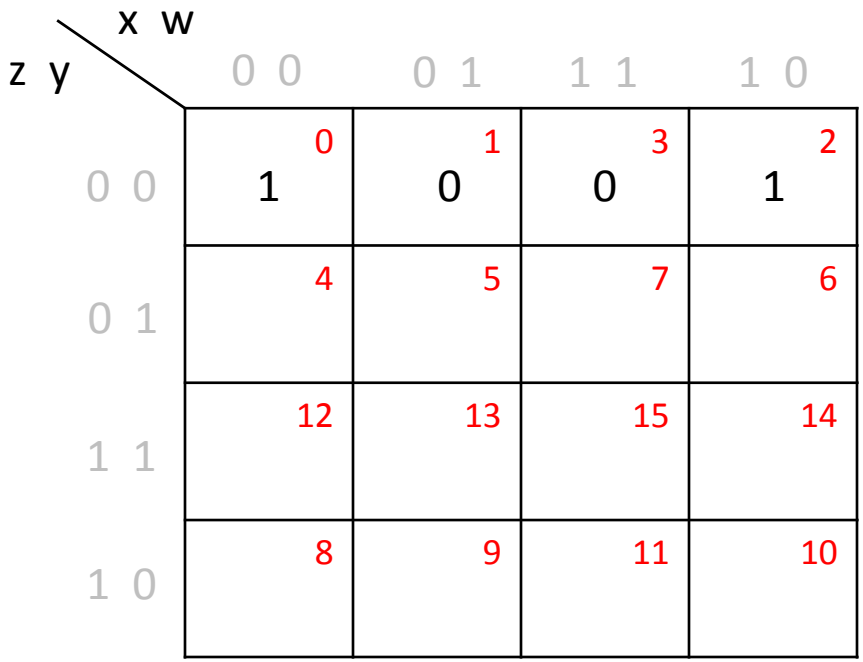
	z	y	x	w	f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

z y \ x w	0 0	0 1	1 1	1 0
0 0	0	1	3	2
0 1	4	5	7	6
1 1	12	13	15	14
1 0	8	9	11	10

4V K-Map

- Place ‘f’-output values
- First 4 outcomes…..

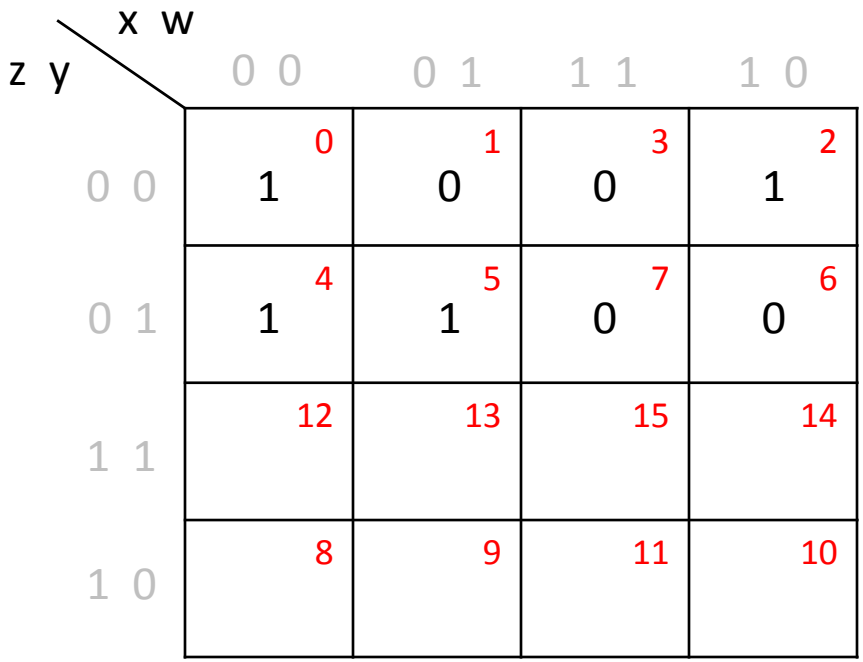
	z	y	x	w	f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1



4V K-Map

- Place ‘f’-output values
- Next 4 outcomes…..

	z	y	x	w	f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1



4V K-Map

- Place ‘f’-output values
- Next 4 outcomes…..

	z	y	x	w	f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

z y \ x w	0 0	0 1	1 1	1 0
0 0	0: 1	1: 0	3: 0	2: 1
0 1	4: 1	5: 1	7: 0	6: 0
1 1	12:	13:	15:	14:
1 0	8: 1	9: 0	11: 0	10: 1

4V K-Map

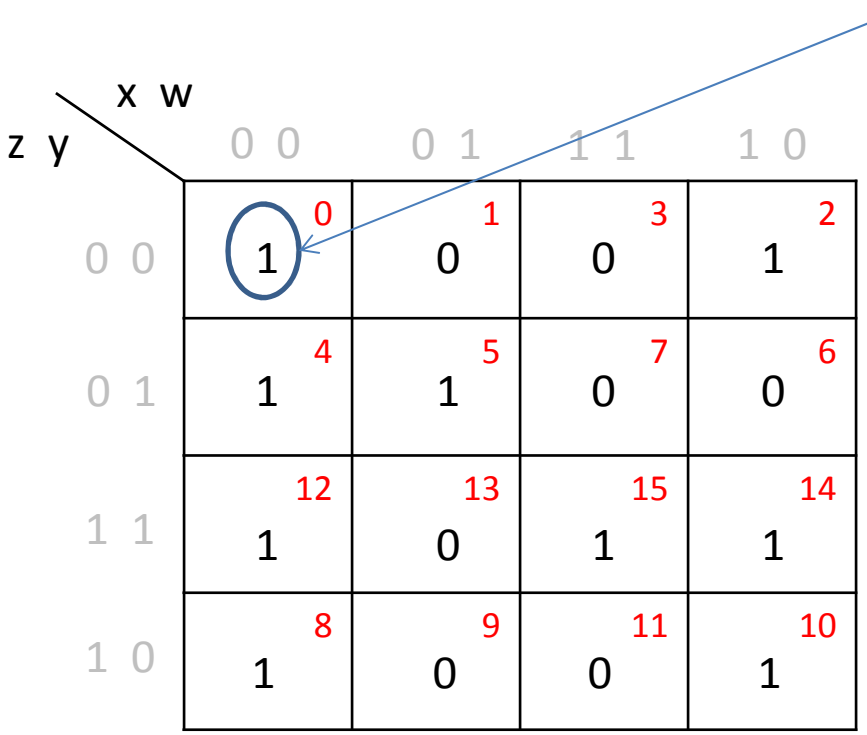
- Place ‘f’-output values
- Last 4 outcomes…..

	z	y	x	w	f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

z y \ x w	0 0	0 1	1 1	1 0
0 0	0: 1	1: 0	3: 0	2: 1
0 1	4: 1	5: 1	7: 0	6: 0
1 1	12: 1	13: 0	15: 1	14: 1
1 0	8: 1	9: 0	11: 0	10: 1

4V K-Map

- Let the grouping of 1's begin!!!
- Very interesting case when observing this first 1



The first intent is to group this 1 with the 1s going down the column -> max group of 4!

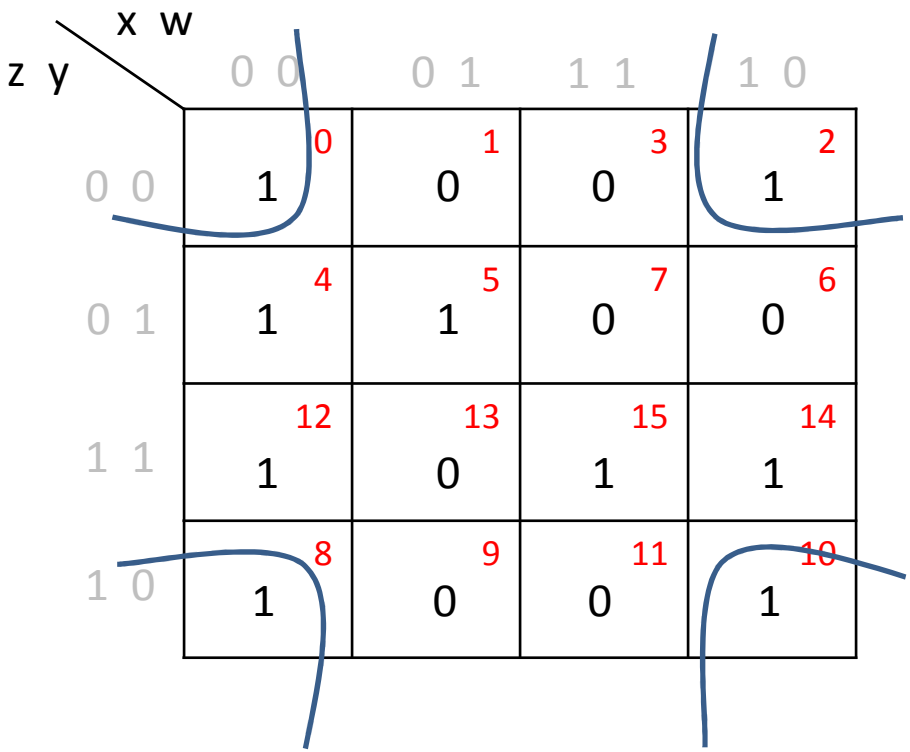
However, since groups exists because of the ONE BIT OFFSET, it also applies to the OFFSET that exist on the corners!

- 0->2, 0->8: 1 bit-offset
- 2->0, 2->10: 1 bit-offset
- 8->0, 8->10: 1 bit offset
- 10->8; 10->2: 1 bit offset

Ergo, they can be grouped!

4V K-Map

- The way we group the corners:



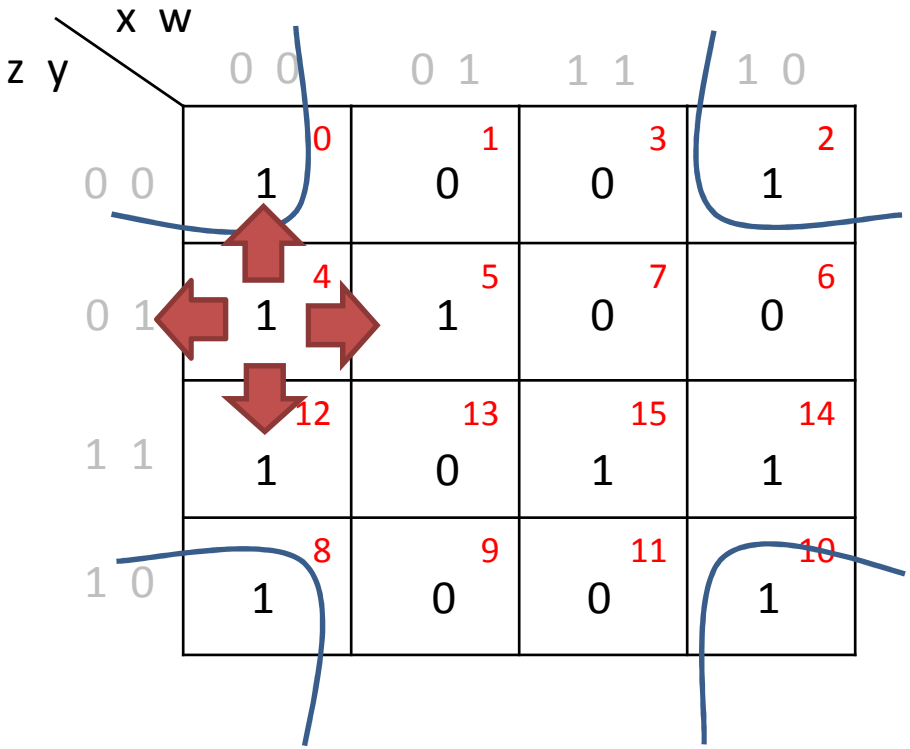
This yields to a derived term:

w'y'

….the bigger the group, less number of variables in the term

4V K-Map

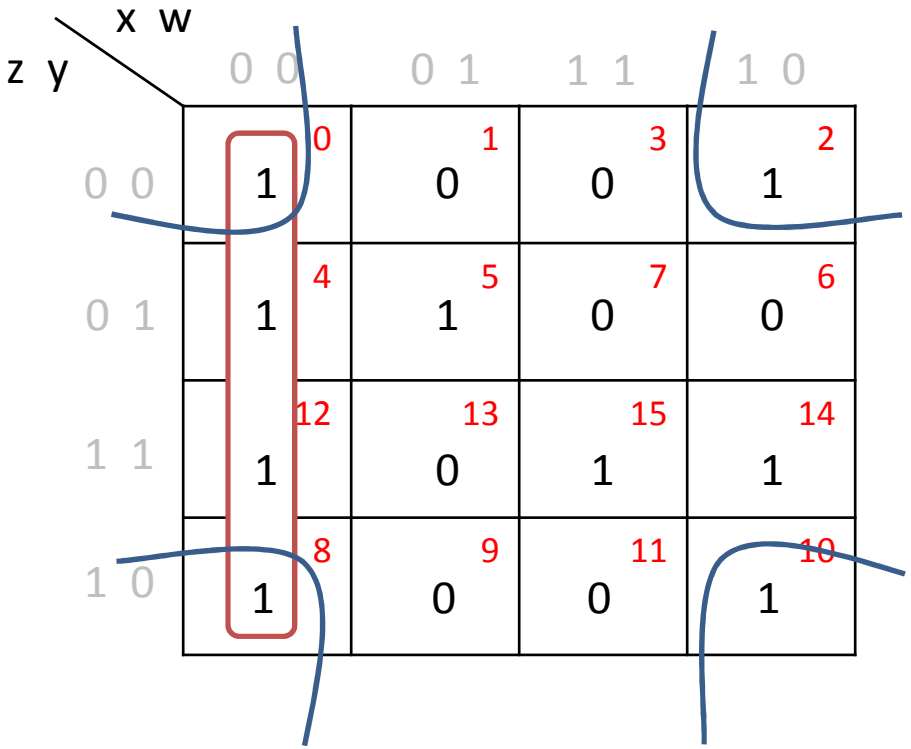
- Move to another free 1, check its neighbors, find max grouping



Since we can use 1's already part of other group(s), we can then expand the group by selecting the whole column…..

We CAN group square 4 & 5; however, it is not the MAX size group square 4 can be part of.

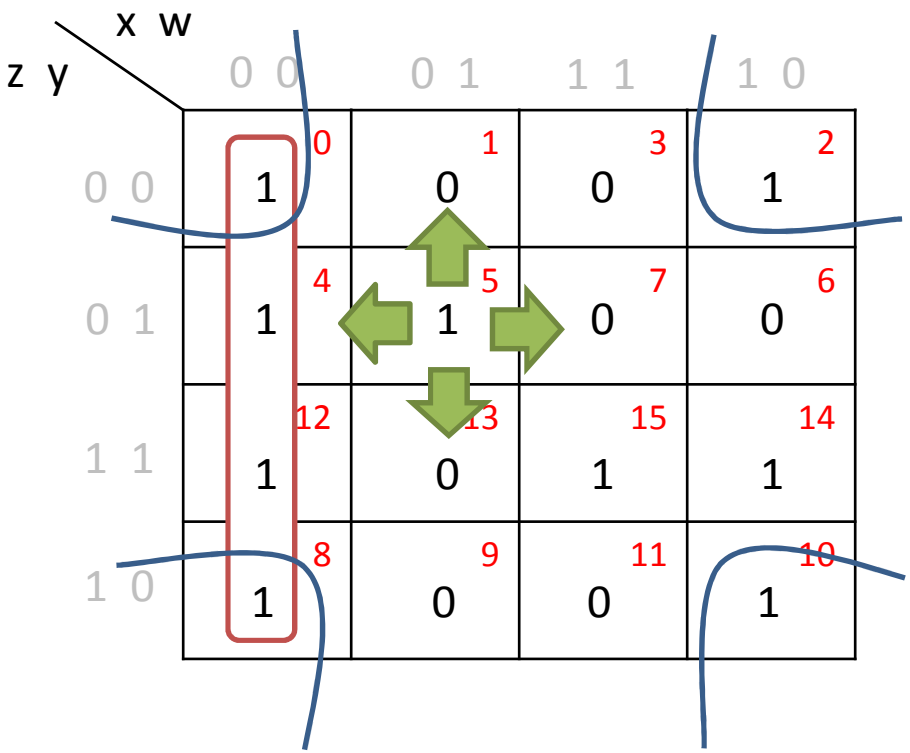
4V K-Map



The term is: w'x'

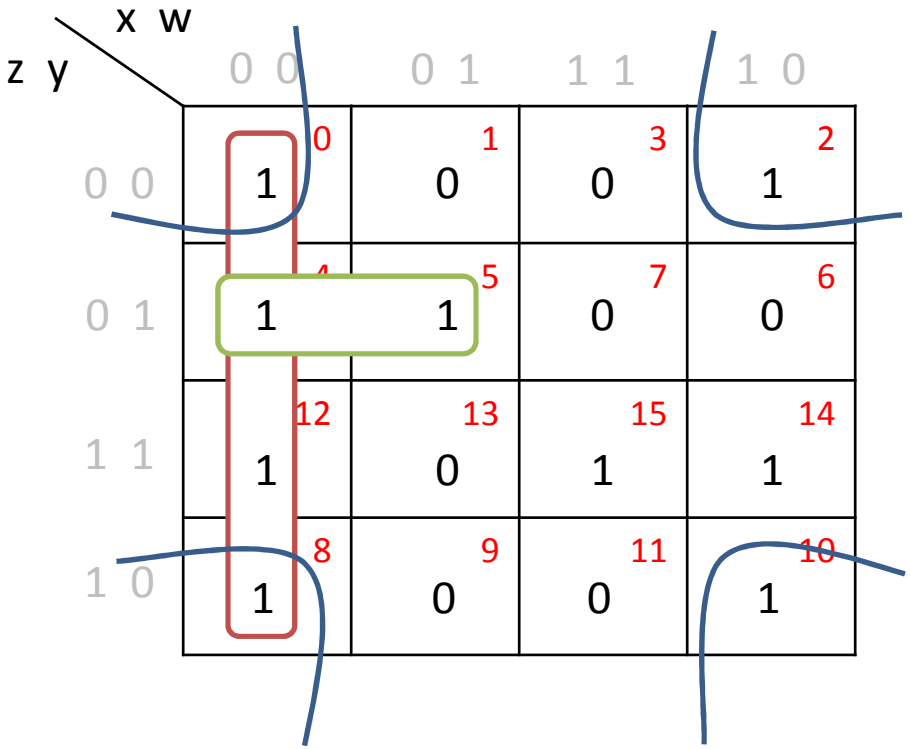
4V K-Map

- Move to another free 1, check its neighbors, find max grouping



In this case, the max sizable group square 5 can be part of is with square 4…

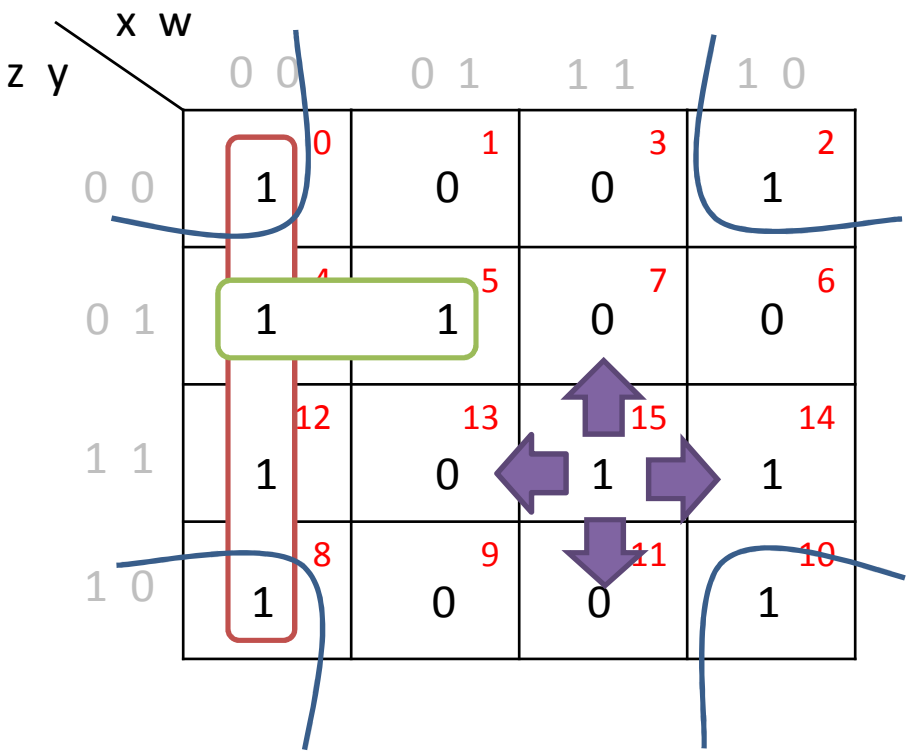
4V K-Map



Term: x'yz'

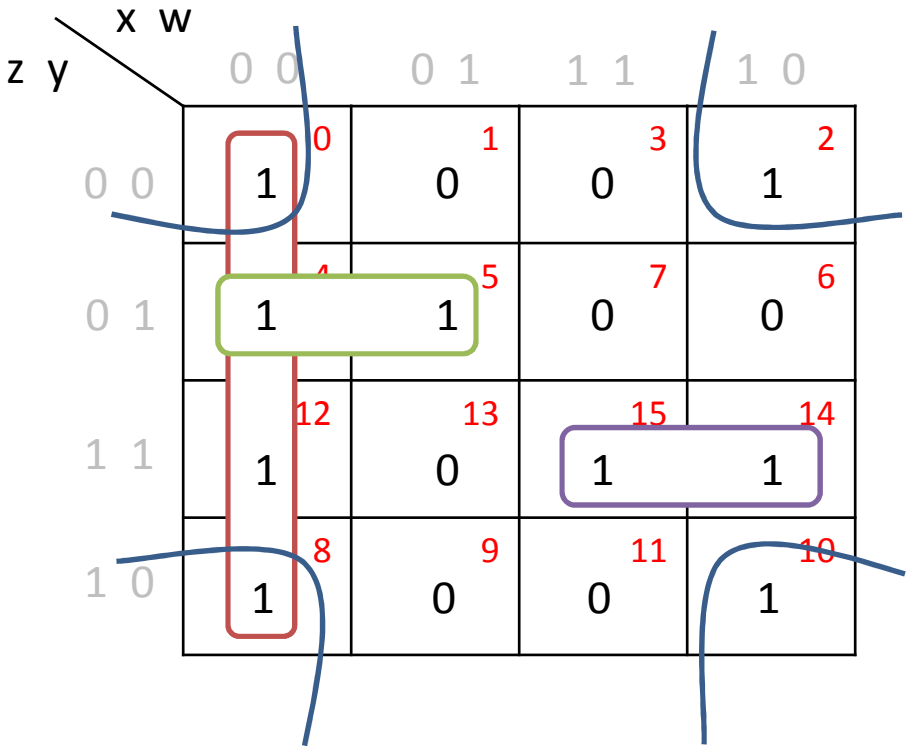
4V K-Map

- Move to another free 1, check its neighbors, find max grouping



In this case, the max sizable group square 15 can be part of is with square 14…

4V K-Map



Term: xyz

Final expression for 'f':

f = w'y' + w'x' + x'yz' + xyz

- When using K-maps, the final expression contains terms that are products (ANDed-grouping of 1's) of the input variables & all terms are summed (OR-ed)

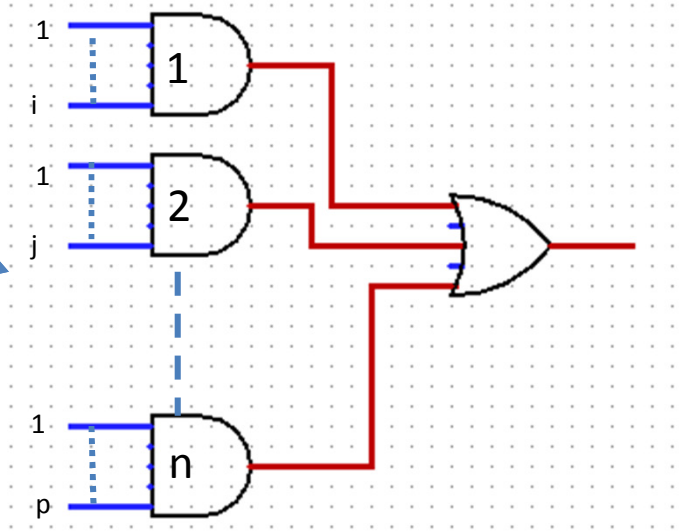
- The expression is known as Sum-of Products (SoP)

- SoP expressions generally have circuits that look like:

  - 1 to n - # of AND gates, with
  - 1 to i,j…p - # of inputs at each AND gates

Final expression for 'f':

f = w'y' + w'x' + x'yz' + xyz



- There are times on a truth table in when….we don't care about the outcome
- So, the expression "don't care" came to existence to the digital design
- We mark the don't-cares with an "X" and became very useful on k-maps
- Don't-cares have a value of either 0 or 1….so, pick the one that is more beneficial to the design….or on the K-map!!

http://www.viralblog.com/wp-content/uploads/2013/03/Knipsel1-500x287.jpg

- Assume we have a system with 3 inputs -> total of 8 possible outcomes
- But, we only need the first 6 outcomes of the table for the design

Z	Y	X	OUT
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	X

- Place don't-cares (X) on the outcomes that wont matter to the truth table
- The significance of X => the outcome can be 0 or 1 (think of Schrödinger's cat without the quantum)
- Placing don't-cares on the K-map is the same as placing logical values

http://www.splitreason.com//Product_Images/10822ad1917e-xl.jpg

- K-mapping the Table…..

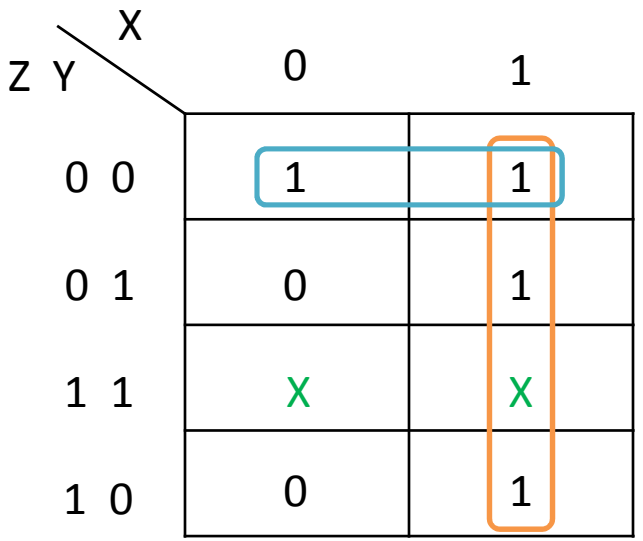
Z	Y	X	OUT
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	X

Z Y \ X	0	1
0 0	1	1
0 1	0	1
1 1	X	X
1 0	0	1

- Since don't-care can be either 0 or 1, lets make them logical 1s….because that's what we are grouping & want max group sizes

- K-mapping the Table…..

Z	Y	X	OUT
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	X



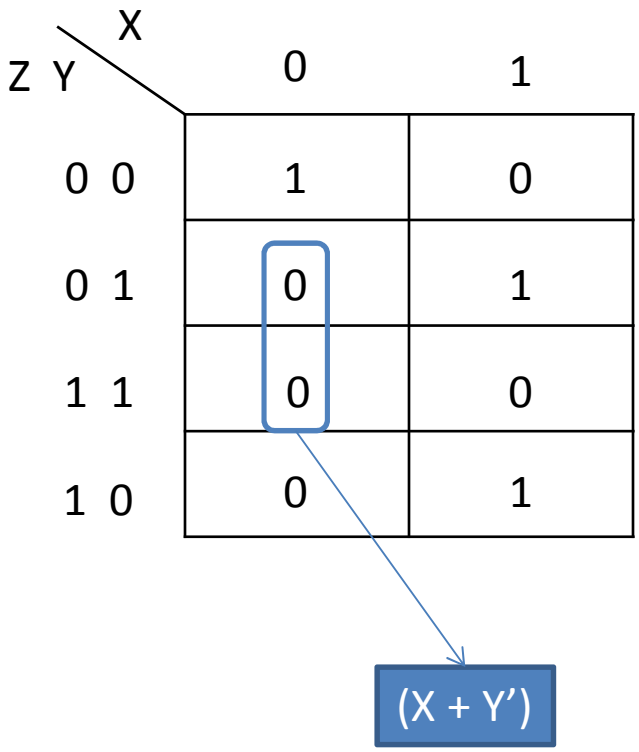
OUT = X + Y'Z'

This is the variable….not the don't-care

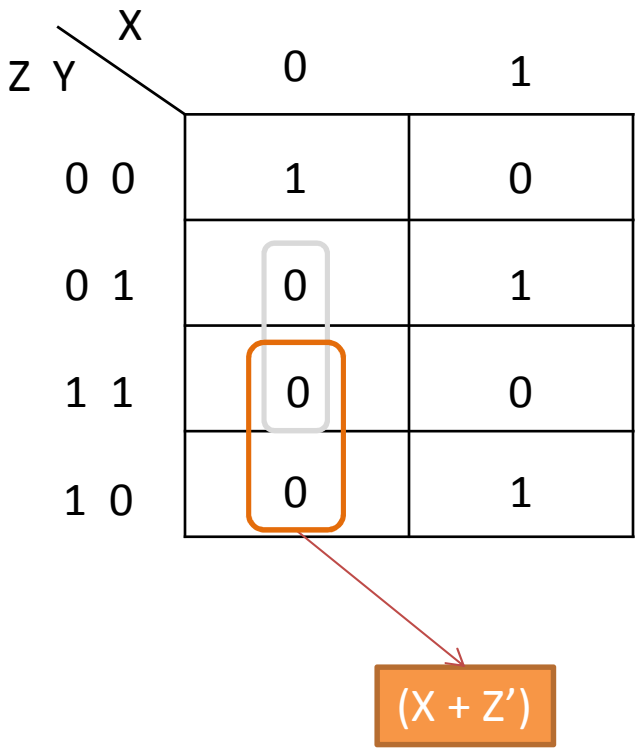
- What happens if you group the 0's instead of the 1's?
- Same grouping rules apply
- Don't-cares can be used as 0's
- In this case:
  - Complement variables that are constant 1
  - Each term is a SUM (OR) of the variables
- You get the opposite expression:
  - Product-of-Sums (PoS)

- Example with pictures is the best way for me to stop typing…just scroll

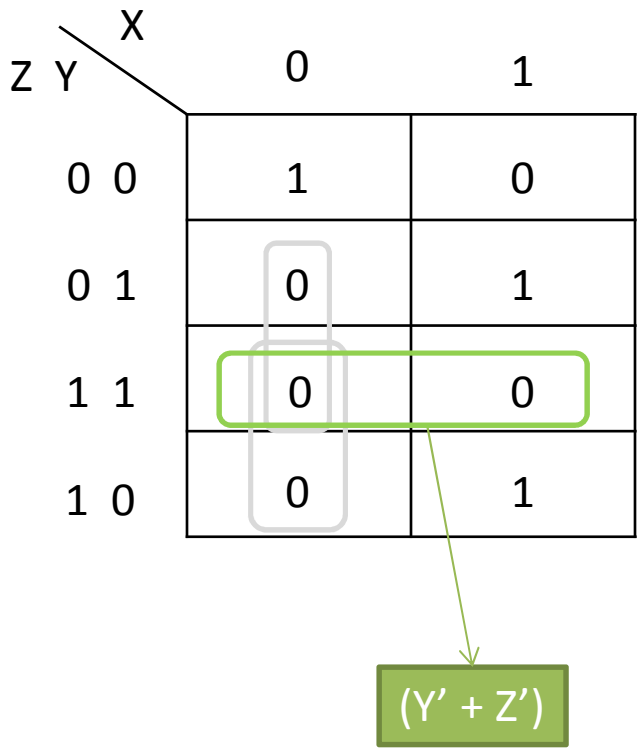
Z	Y	X	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



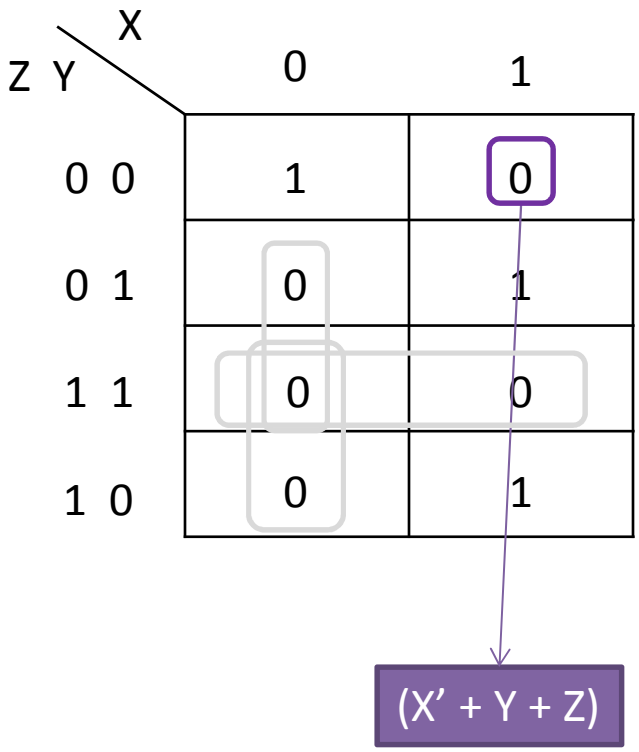
Z	Y	X	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



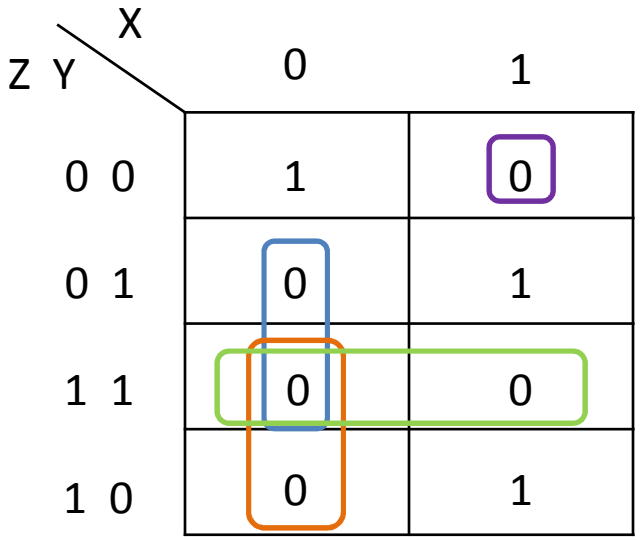
Z	Y	X	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Z	Y	X	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Z	Y	X	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



OUT = (X + Y') • (X + Z') • (Y' + Z') • (X' + Y + Z)

- Done with K-MAPS!!!..... Now go play some….