Multiple Outputs From Same Circuit

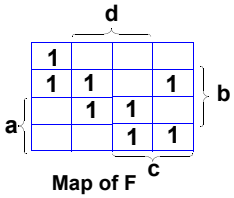
Two outputs, F and G
Same inputs

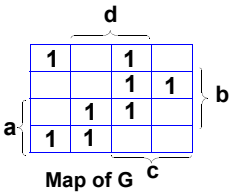
Find the circuits for F and G

- Need two maps
- Need two circuits

Often one can share some gates

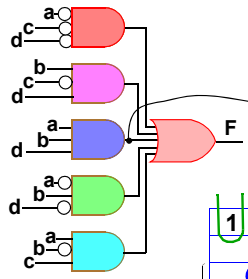
- We optimized maps individually
- got one common gate abd.



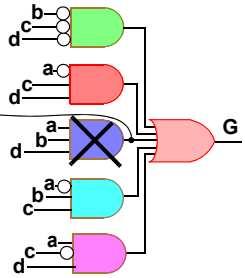


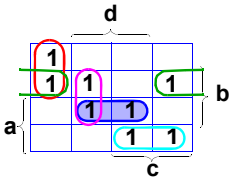
circuit size estimates
- 29 letters (literals) 13+13+3
- 37 gate inputs 15+5+12+5
- 11 gates

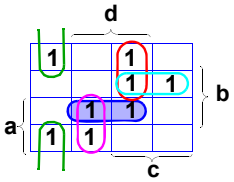
In CMOS (number transistors)/2



u = abd
3 letters







F = a·c·d+bcd
+u+abd+abc
13 letters
15 AND inputs + 5 OR inputs

G=b·c·d+acd
+u+abc+acd
13 letters
12 AND inputs + 5 OR inputs

Circuits With Two Outputs

Multiple Outputs

The difference between multiple outputs and single outputs

With multiple outputs, one can often find common gates that can be used for both outputs.
Often these common gates are not optimum for either individual circuit, but are optimum for the whole circuit.

The example

- In this slide the circuits were optimized individually with a half-hearted effort to find common terms.
- In the next slide, common terms were aggressively sought out.

Circuit complexity

There are several ways to estimate the size of the circuit. The same measures also estimate power dissipation which is now likely to be more important than size.

- Inverters are not usually counted in the gate count. This is because most will be absorbed when one does an AND/OR to NAND/NOR conversion.

Three methods of estimating circuit size:

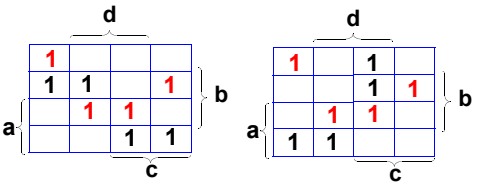
1. The number of gates.
2. The number of gate inputs. This admits that multi-input gates are larger.
   One gate input usually corresponds to two transistors in CMOS logic.
3. The number of letters on the right hand side of the expressions. This is easy to do, and some considered it the best estimate.

Note these are relative estimates; used for estimating if one circuit is significantly bigger than another. An absolute or an accurate estimate requires one to know the details of the implementation.

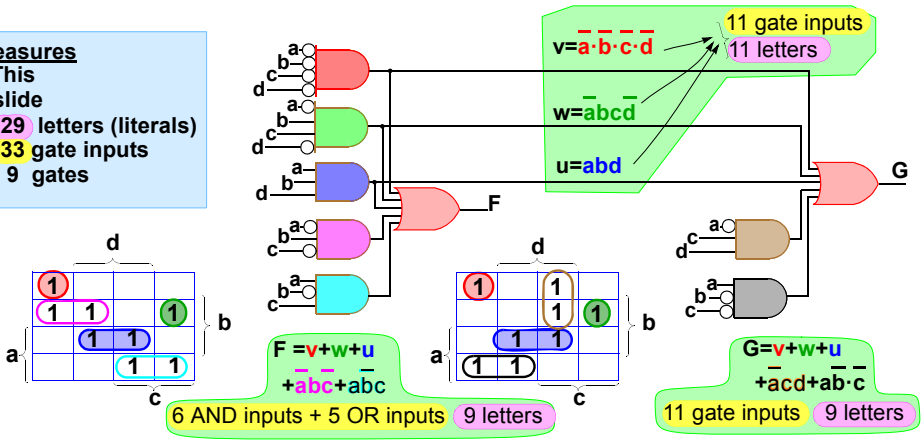
Multiple Outputs Minimization

Try to harder to share terms (gates)

- …Identify common squares on both maps
- …Loop common terms
  even if the individual maps
  allow larger loops
- …Check, sometimes common terms do not help.
- …Here it changed a 3-input AND to 4-input AND
  and removed another 3-input.



size measures		
Prev slide	This slide	
29	29	letters (literals)
37	33	gate inputs
11	9	gates



Multiple Outputs

Collecting the u+v+w terms would reduce the number of letters and gate inputs, but will increase the number of gates. However the total logic is clearly reduced.

x = u+v+w = **a·b·c·d + abcd + abd** (11 letters, 14 inputs, 4 gates))

F = **abc + abc + x** (7 letters, 9 inputs, 3 gates)

G = **acd + ab·c + x** (7 letters, 9 inputs, 3 gates)

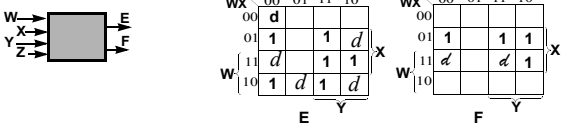
Total: 25 letters, 10 gates, 32 gate inputs

5-8.•PROBLEM

Find the S of P expressions with minimal logic for the two-output circuit E, F.

Soln. has 5 gates.

If it is not pure S of P, it can be done in 5 two-input gates, or, with factoring, 4 gates.

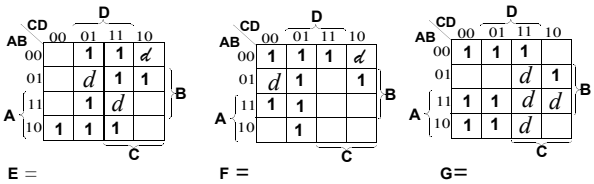


5-9.•PROBLEM

a) Find the S of P equations using the minimum total number of gates if no gates are shared between outputs.

b) Find the way to reduce this to 9 gates if gates are shared between maps.

One solution has: two 5-input gates,
one 4-input gate,
five 3-input gates,
and one 2-input gate.



MT04B

Multiple Outputs; BCD -> Braille

BCD to Braille Conversion

BCD (Binary Coded Decimal)

Four bits can define numbers up to 15.

BCD digits use 4 bits but not all combinations.

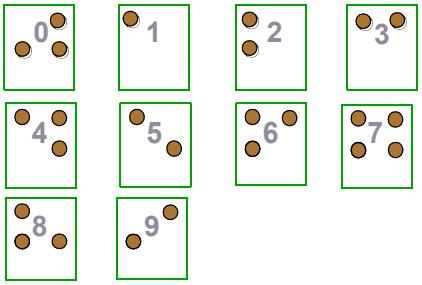
Binary 10 through 15 are not used

Braille

Braille symbols for digits are 4 raised dots.

The dot's position is denoted by W, X, Z, Y.
W is upper left, Y is lower right

Braille Digits



Binary	Dec
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

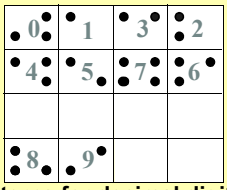
Binary Coded Decimals

AB\CD	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Binary positions on a K-map

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

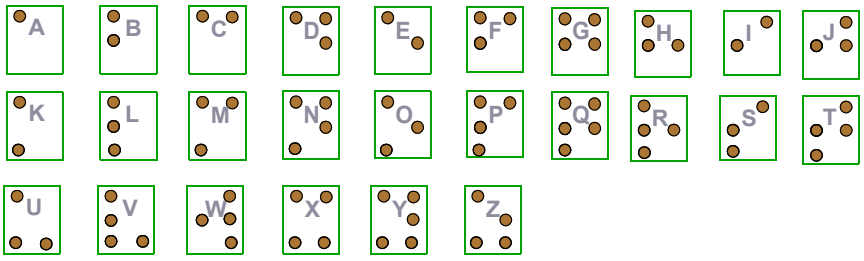
Corresponding decimal numbers



Braille patterns for decimal digits

Braille

The Braille code consists of 6 squares, only the upper 4 are used for digits.



The first ten letters of the alphabet are the same as the digits.

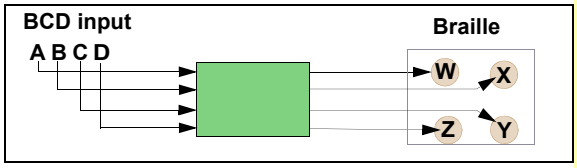
Multiple Outputs; Braille Maps

.BCD to Braille

Braille

Dot identification

Design a BCD to Braille Converter



Design a circuit to:

- accept binary-coded decimal input ABCD
- generate Braille output WXZY.

a) Draw maps for outputs W, X, Z and Y. Include don't cares.

b) Loop the maps for minimum V of S logic if each output is treated separately.

c) Loop the maps to minimize logic including gate sharing.

d) State (i) the minimum number of gates used
(ii) the number of gate inputs
(iii) the number of letters.

AB\CD	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

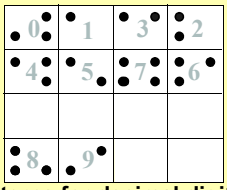
Not used Not used Not used

BCD positions on a K-map

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

Not used Not used Not used

Corresponding decimal numbers



Braille patterns for decimal digits

Four Output Minimization

The figure above on the right shows:
- the binary positions on a Karnaugh map,
- the corresponding decimal numbers,
- and the Braille pattern for the decimal digits

Braille symbols are a pattern of raised dots. The positions of the dots is given by the symbols W,X,Z and Y.

Design Example

Design a circuit which will accept a binary-coded decimal input ABCD and generate a Braille output WXZY. Binary-coded decimal digits only go from 0 to 9. The other numbers, 10 through 15 will never be received as inputs.

a) Draw the map for the W output. Be sure to include don't cares.

b) Draw the maps for each of the W, X, Y and Z outputs.

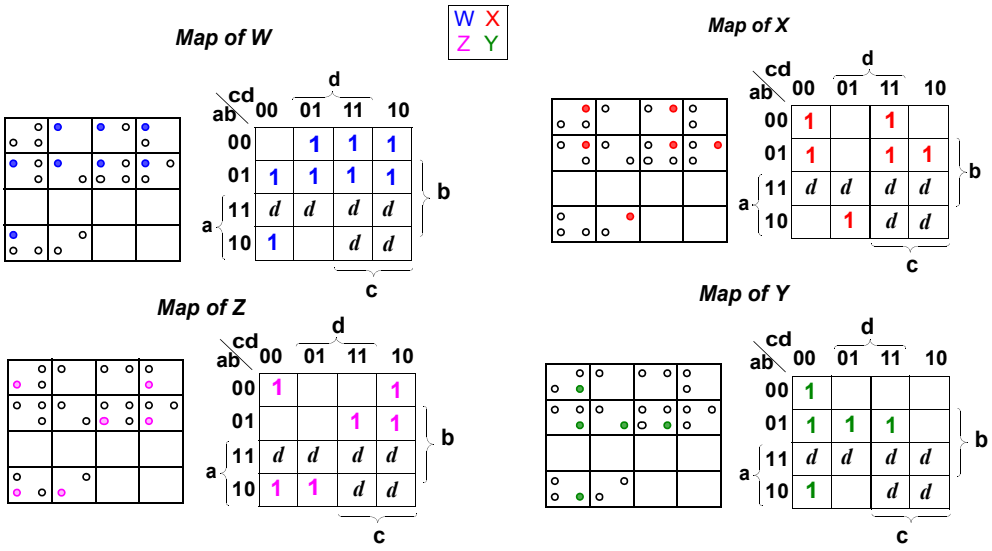
c) Loop the maps to give the minimum S of P logic if each output is treated separately.

d) Draw a new set of maps.
Then loop them to give minimum logic if gates can be shared between maps.

e) State (i) the minimum number of gates used and (ii) the number of gate inputs, (iii) the number of letters.

BCD--> Braille; Initial Maps

a) Draw maps for outputs W, X, Y and Z. Include don't cares.



Drawing the Four Maps

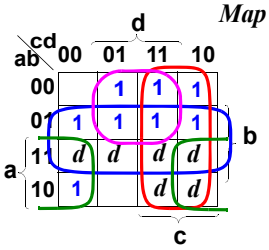
W is the dot in the upper left corner  . Thus the W map consists of all the numbers which have that dot, namely 1, 3, 2, 4, 5, 7,6 and 8. All the other squares on the W map are either 0 or "*d*".

X is the dot in the upper right corner  . The X map consists of all the numbers which have that dot, namely 0, 3, 4, 6, 7 and 9.

BCD--> Braille; No Gate Sharing (How not to do it)

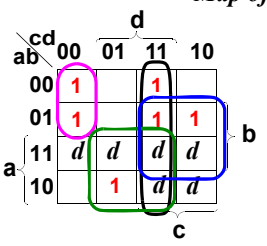
b) **Maps looped as though each output was independent.**
Later this will be used to show savings from sharing

Map of W



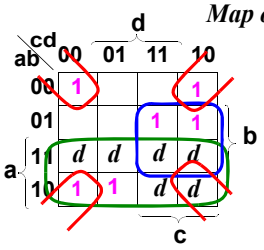
$W = b + c + \bar{a}d + a\bar{d}$

Map of X



$X = bc + cd + \bar{a}\cdot\bar{c}\cdot\bar{d} + ad$

Map of Z



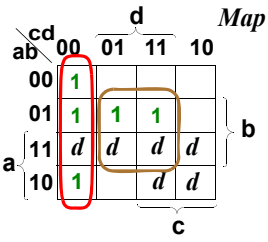
$Z = \bar{b}\cdot\bar{d} + a + bc$

6+9+5+4 = 24 letters

3+5+3+3 = 14 gates (10 AND)

8+13+7+6 = 34 inputs

Map of Y



$Y = \bar{c}\cdot\bar{d} + bd$

Circling the Maps as Thought They Were Completely Independent

These maps are looped, with no attempt to share AND terms; one gets:

24 letters (also called literals). Remember one counts only the letters on the right hand side.

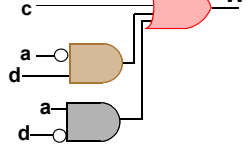
14 gates.

10 AND gates which will be the important number when we look at Programmable Logic Arrays.

34 gate inputs. One can easily count them by drawing a picture. This also turns out to be the number of letters plus 1 for each AND term.(2 or more inputs AND terms) in the W, X,Y, and Z equations.

6 letters
8 gate inputs

Each AND terms has an output into the OR gate that is not counted when you count the letters.



$W = b + c + \bar{a}d + a\bar{d}$

A set of circling guidelines to minimize gate the gate count

On the next few slides a set of heuristics are given to share gates. A heuristic is an algorithm which usually gives a good answer, but is not guaranteed to work, or as in this case give the best answer.

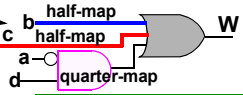
These heuristics are designed to emphasise a small gate count, and may actually increase the number of letters. These heuristics (except for rule 1) are especially good for Programmable Logic Arrays (done later) where the number of gates is all important, and the number of inputs is constant.

BCD--> Braille; Sharing Gates; Half-Maps

c) Loop the maps trying to share gates

Half-Map Rule
(Rule 1) Loop the half-maps first. They do not add any gates.

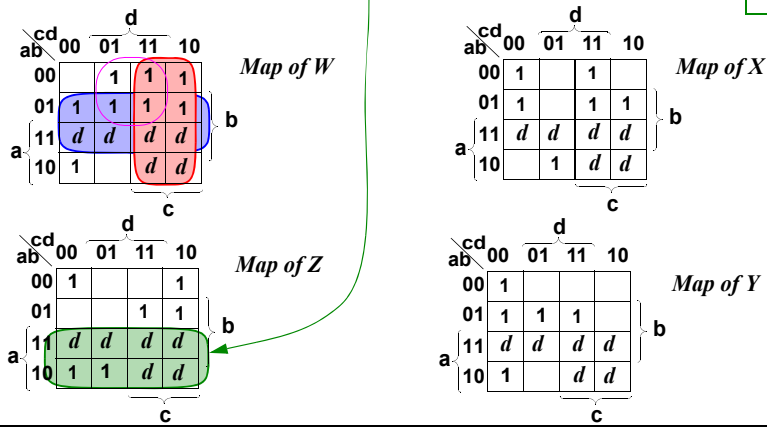
W needs only one AND gate for three loops



A "half-map" is a loop which covers half the squares

Rule (1) may not work on Programmable Logic Arrays (to be done later)

a (on Z); b, c, (on W);



Heuristic Rules for Looping Order in Multiple Output Maps

These rules will be followed in the next few pages

1. **Half Map Rule**: Loop the half-maps first. (like b and c on the W map above) They do not add any AND gates.
2. **No Friends Rule:** Loop squares that appear on only one map. Such squares can never be shared.

2a). **My Friends Are Gone Rule**: Their friends are looped or *d* on all the other maps. No useful sharing.

3. **Procrastenate Decisions:** With several looping choices for a square, do another square first if available.
4. **No Brothers Rule**: Loop "1"s with no neighbours on the same map (the family map), except looped neighbours or *d*. Loop the same square on other maps that have an unlooped "1" there.

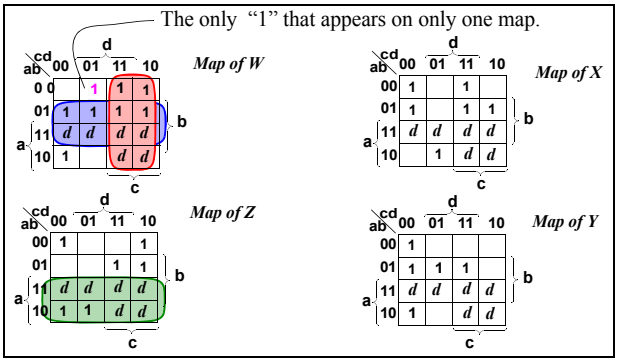
4a). Expand the family map loops provided you can also expand the loops on any friend's maps.
Keep the loops if they are max size, on the family map.
If they reach max size on another map first, these rules have no further advise.

5. You are on your own. Fortunately it is usually easy.

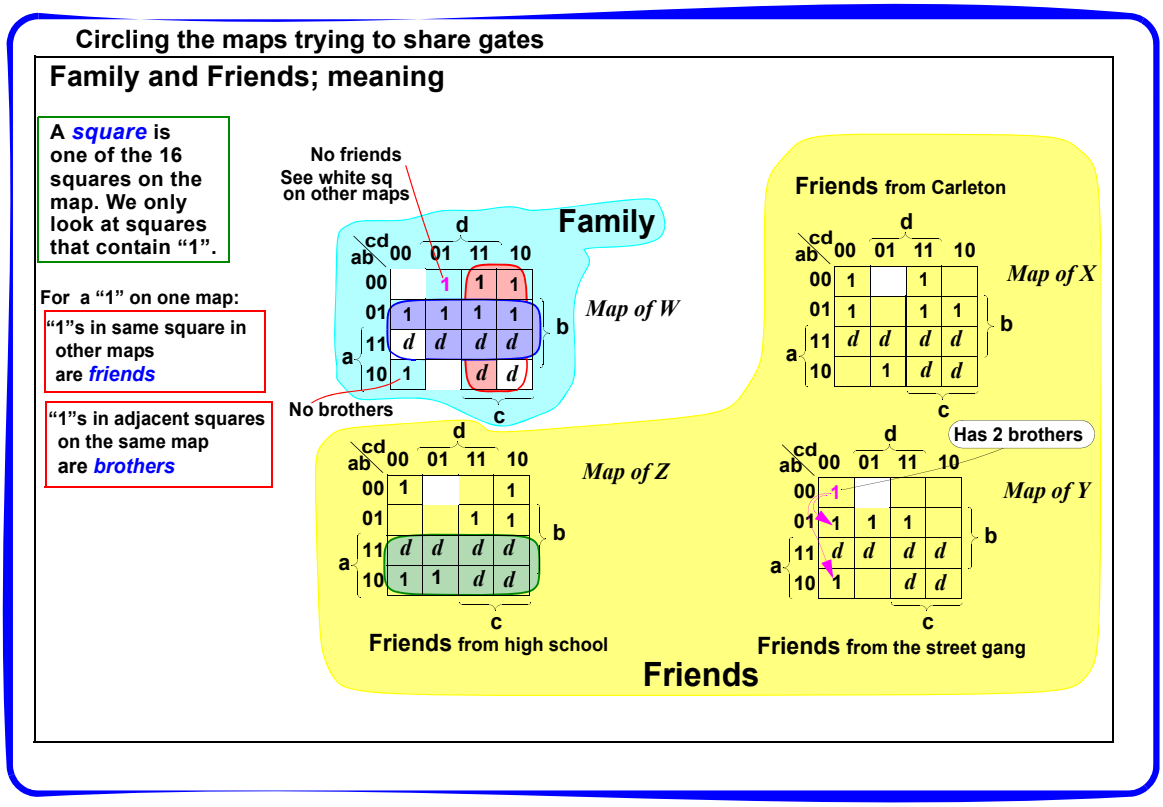
Looping maps trying to share gates

(1) A loop that encloses half the map, does not use an AND gate; it is only a piece of wire. Always loop them, there is no gain from sharing a piece of wire. However there is an exception. In a PLA (to be covered later) a piece of wire does take an AND gate. In that case do not automatically loop half maps.

(2) The "1"s that appear on only one map cannot share an AND gate with another map. Hence loop them with the largest possible loop immediately.



BCD--> Braille; Family and Friends



Family and Friends

Pretend you are a "1" in a square on one map

Family are other "1"s on the same map.
The important family members are brothers.
Brothers are in adjacent squares to you.

Friends are "1"s in the same position on other maps.

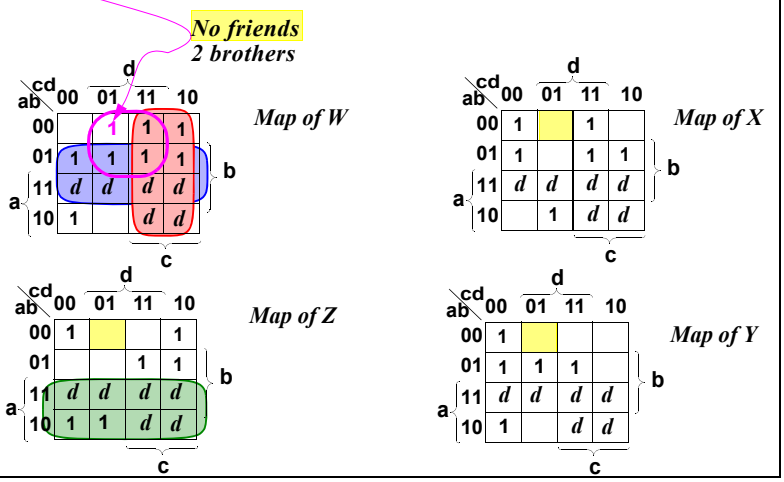
Once a friend is looped, one says he/she has gone to the dark side. There is no point in trying to share gates with a 1 that is already looped.

BCD--> Braille; Sharing Gates, No Friends Rule

(Rule2) No Friends Rule (for What to Loop Next)

For "1"s with no friends (a 1 appears on only one map).
There is no way to share them.
Loop them with as many brothers as possible.
(*You can include other "1"s and "d"s in the loop to make it larger*)

$\bar{a}\cdot\bar{b}\cdot\bar{c}\cdot d$ (on W) => best looped by $\bar{a}d$ (on W)



Circling the maps trying to share gates

Rehash of Rules (1 and 2)

We have now looped half-maps and "1"s that only appear on one map. These later loops will never be shared, hence we ignore them when we look for further sharing.

These loops will be gray shaded to make them easy to ignore from now on.

Rule (2a)

Now, we look for unlooped "1"s that are looped (gray) on all the other maps. These "1"s are now friendless; they will never be usefully shared between maps because the potential sharing partners are already looped.

Rule(1) and Programmable Logic Arrays (PLAs and PALs)

In these arrays, which will be studied later, the AND gates are prebuilt. It takes the same resources to make a 1-input AND gate as it does a 5-input one. In such cases sharing is all important. Half-maps should be treated like any other loops, and are likely not best because they are hard to share.

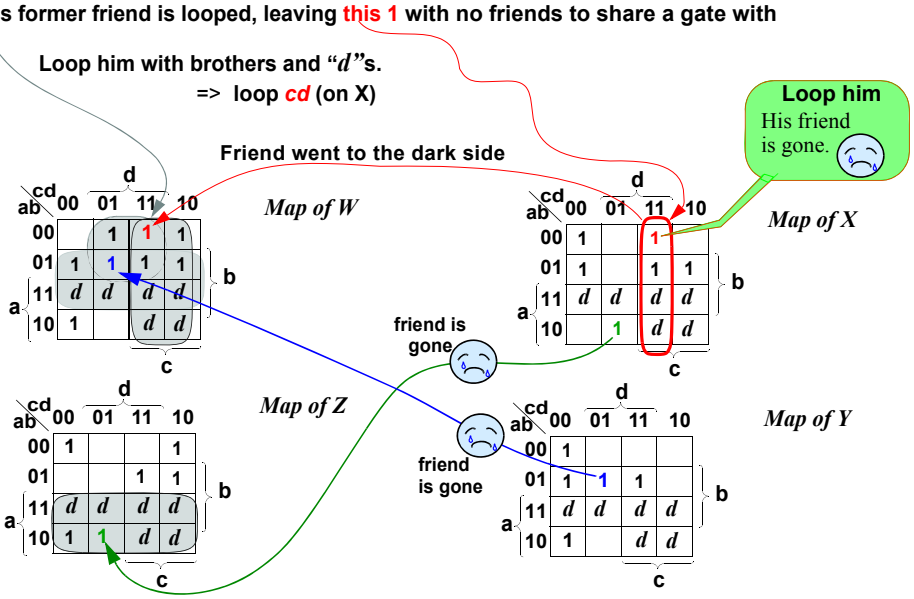
BCD--> Braille; Sharing; My Friends Are Gone

c) **Looping maps trying to share gates,**

(Rule 2a) **My Friends Are Gone Rule.** (Looped into the dark side)

With no friends left, we cannot usefully share loops.
Loop these new friendless "1"s.

This former friend is looped, leaving this 1 with no friends to share a gate with



Squares whose friends were looped earlier, and are now on the dark side.

We are going to have to loop these squares some time, and there is no point in trying to share gates with "1"s that are already included in another loop.
Squares where the friends are don't cares are not worth shareing.

BCD--> Braille; Sharing; My Friends Are Gone

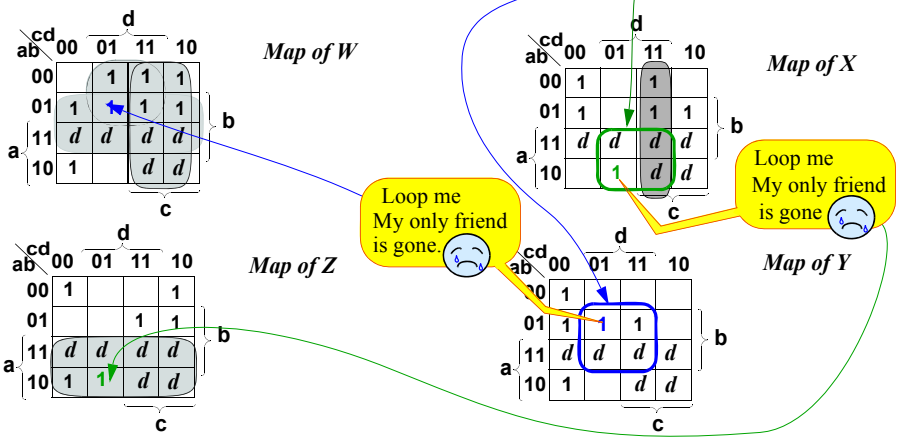
c) Looping maps trying to share gates

(Rule 2a) My Friends Are Gone Rule. ***(They went over to the dark side)***

Find "1"s that cannot be usefully shared; friends are already looped.
Loop them with brothers.

$a\cdot\bar{b}\cdot\bar{c}\cdot d$ **in Y cannot be usefully shared. => loop** ad **(on X)**

$\bar{a}\cdot b\cdot\bar{c}\cdot d$ **in Y cannot be usefully shared. => loop** bd **(on Y)**

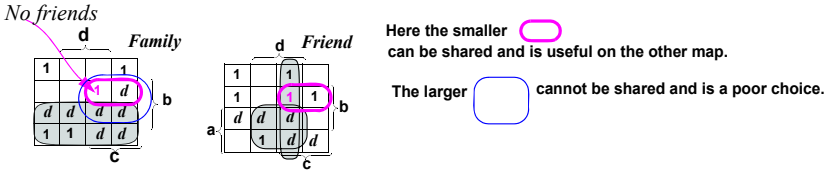


Looping the maps trying to share gates

There is no point in trying to share loops if all your friends have gone over to the dark side (already looped).

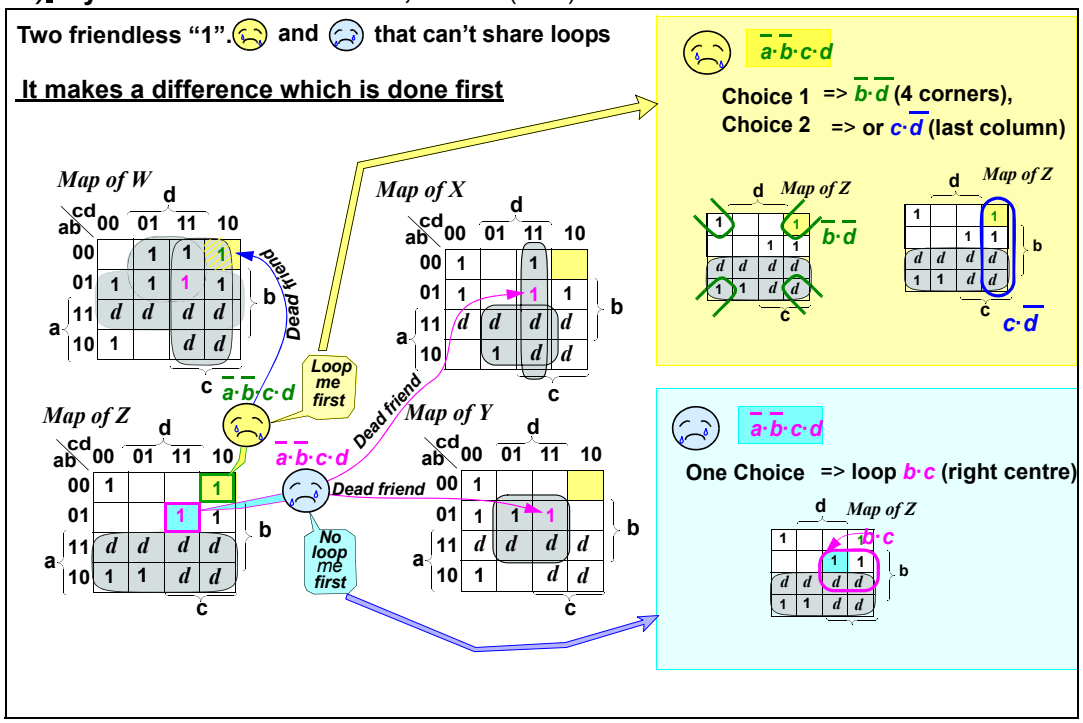
Loop these squares now.

Here, how to loop them is clear. In some cases it in not, for example:
If the the friends are looped or *d*, and the loop may include other squares.



BCD--> Braille; Sharing; Friends Are All Gone

c) **My Friends Are Gone Rule,** rule 2a (cont)



Circling the maps trying to share gates

(Rule 2a) (continued)

Continue to look for "1"s that are unlooped on only one map. Loop them.

Procrastenate if one square has several loop choices (Rule 3).

Sometimes they can be looped in several ways. Look at what else is included in the loop.

Look at $\bar{a}\cdot\bar{b}\cdot c\cdot d$, in the upper right corner .

Choice 1: Here a four corner loop will also cover one extra square.

Choice 2: A vertical column loop will also cover one extra square.

Look at $\bar{a}\cdot\bar{b}\cdot c\cdot d$, located about here .

Single choice: It can be looped to cover one more uncovered **family** squares.

That loop can also be shared so his brother covers another **friend** square.

Its loop also removes the vertical column choice for $\bar{a}\cdot\bar{b}\cdot c\cdot d$, the upper right corner.

Conclusion:

Loop first.

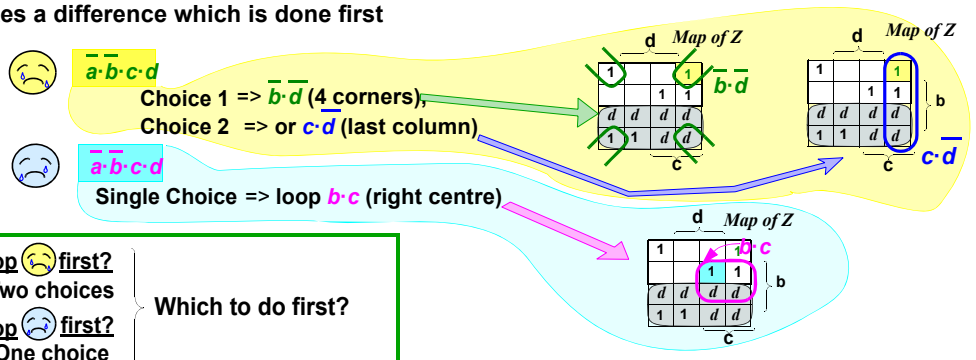
Then loop using the four corners.

BCD--> Braille; Procrastenate

c) **My Friends Are Gone Rule,** rule 2a (cont)

Two friendless "1s"☹ and ☹ that can't share loops

It makes a difference which is done first



Loop ☹ first?
Two choices
Loop ☹ first?
One choice
Which to do first?

Use Rule 3 Procrastenate

You must do both squares, but -

Do the single choice one first.
It may stop you from making a poor choice on the other one.

Do ☹ first.

Why Procrastenate?

There was a question of which square to loop next.

One square could be looped in only one way.
The other had two possible loops to choose from.

Usually the single choice square is the one to do next. It puts off making a choice which might not be the right one.

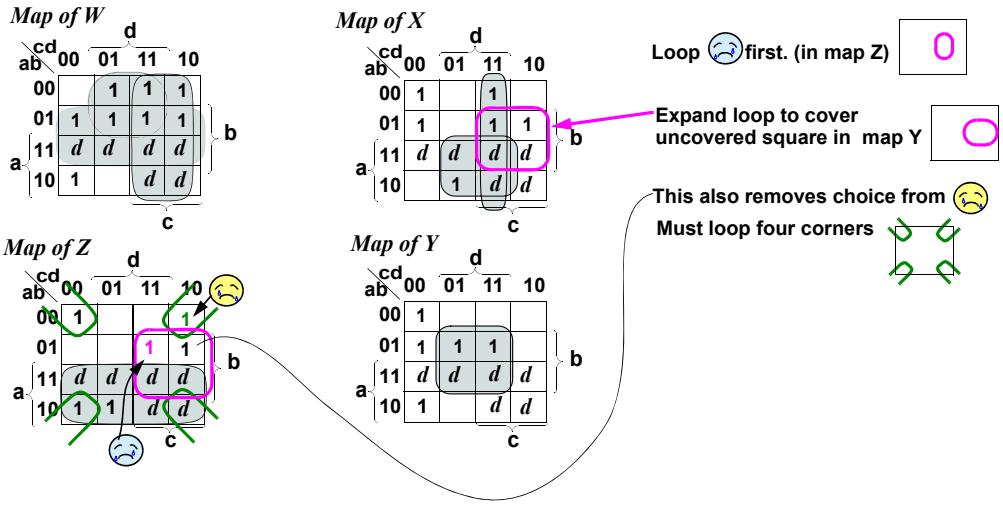
In this case it removed one of the choices from the other square, making the next loop a no-brainer.

BCD--> Braille; Procrastenation may help

*c)_* Looping the maps trying to share gates

Result of Looping using Procrastenation

It removes one choice from ☹



- **Final results of Rule 2a. My friends are gone.**

  Note that by looping just "1"s that one is sure cannot be shared, the size of the problem has been greatly reduced.

"What to Loop Next" rules in summary

1. **Half Map Rule**: Loop the half-maps first. They do not add any AND gates.
2. **No Friends Rule:** Loop squares that appear on only one map. Such squares can never be shared.

2a). **My Friends Are Gone Rule**: Loop squares whos friends are looped or *d* on all other maps. No useful sharing.

3. **Procrastenate Decisions:** With several looping choices for a square, do another square first if available.
4. **No Brothers Rule**: Loop "1"s with no neighbours on the same map (the family map), except looped neighbours or *d*. Loop the same square on other maps that have an unlooped "1".

4a). Expand the family map loops provided you can also expand the loops on any friend's maps.
Keep the loops if they are max size, on the family map.
If they reach max size on another map first, these rules have no further advise.

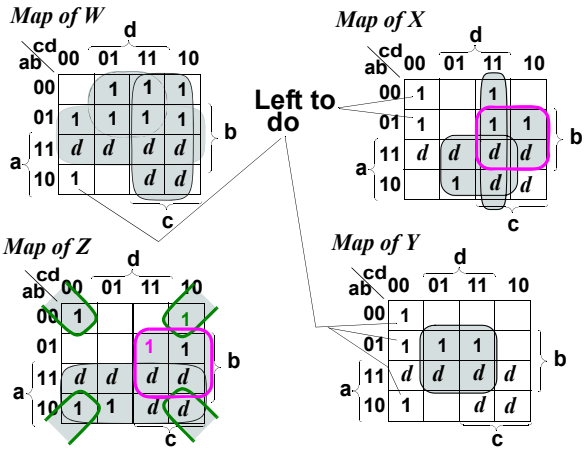
5. Use original thought. Fortunately it is usually easy.

BCD--> Braille; Sharing

*c)_* **Looping the maps trying to share gates**

Summary of what is looped up to now.

Loops from the previous slide are now shaded.



Circling the maps starting with the family

Rule (4): Squares with

The obviously unshareable loops have been found. We will now look for loops that can be shared.

To do this is to start small.
Look for single squares on one map that are also "1" and unlooped on one or more other maps. Call the original map the family map. We know we will have to loop that "1", and we will try to share its loop.

Then expand the family square's loop to include more squares. Do the same expansion on the other maps. Eventually you will have to stop. If you have to stop because limitations on the family map this square must be the best. You know that "1" must be looped, and this is the largest square that can do it.

If you have to stop because of limitations on the other maps, you will have to evaluate the benefits of sharing versus the benefits of having the larger square on the family map. This requires a different kind of thinking than just following the rules.

Remember

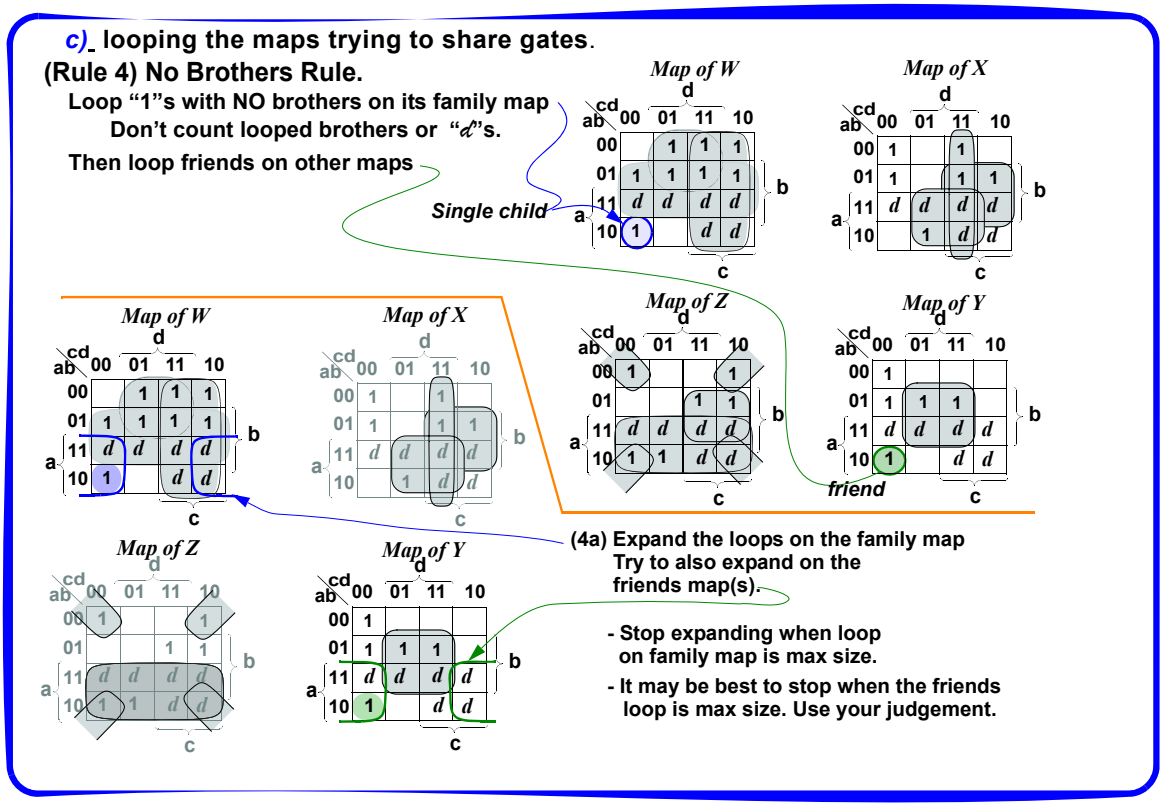
Larger loops make smaller gates.

Sharing loops eliminates a gate(s).

Could the friends square be included in another loop?

Good luck.

BCD--> Braille; Loop Single Children



The rules (Guidelines) have finished

The rest is:

- Common sense.
- Low cunning.
- Luck
- Here the way to loop the other squares is fairly clear.

5-10.•PROBLEM

Assume the term a,b,c,d = 1,1,0,1 is used for some other purpose, like a decimal point, and is no longer a don't care input. Find the new set of maps, and the new equations. You will increase the number of gate inputs by two (I think), but you should not have to add any new gates.

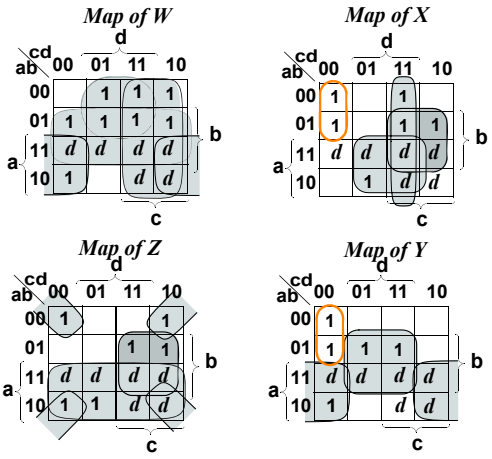
BCD--> Braille; Final Clean Up

c) Looping the maps trying to share gates.

Use Original Thought

Take what is left and try to loop and share

Here the solution is very clear.



Final Equations and Gate Count

The final equations show:

Letters 7.7% loss

Gates 14.3% saving

AND gates 20% saving

Gate inputs 8.9% saving.

Not too impressive?
For a more imressive result see Comment on Slide 46.

What commonly happens is that sharing does not save too much on the letter count, but has a better saving on the gate count. Unfortunately, computer-aided design software often counts letters.

AND gates may be the most important

For many CPLD (complex programmable logic devices) the internal logic is with PALs (Programmable Array Logic) in which the important parameter is the number of AND gates. Sharing gates is then much more important than the letter count.

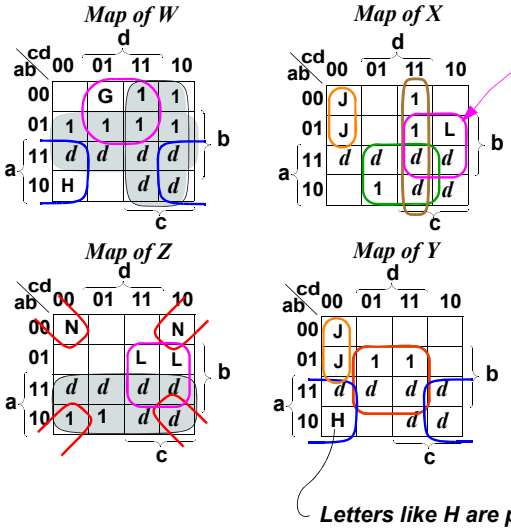
Who cares how many gates there are?

With ten million gates on piece of silicon that costs $3.00 why should one worry.
If you have a small circuit which may be replicated 1,000 times on a piece of silicon. It matters.

Also power usage increases with gate count. This means your cell phone runs down more quickly, or your lap top needs a bigger fan.
Delay in a circuit often increases with more gates.

BCD--> Braille; Final Equations and Gate Count

d) **Final equations and gate count,**



L = bc H = $\overline{ad}$ J = $\overline{a}\cdot\overline{c}\cdot\overline{d}$

G *on map*

W = $\overline{ad}$ + H + b + c

Z = $\overline{b}\cdot\overline{d}$ + L + a

N *on map*

X = J + cd + L + ad

Y = J + bd + H

This solution with gate sharing	With no sharing
26 letters	24
12 gates (8 ANDs)	14 (10 AND)
31 gate inputs	34

Letters like H are put on the map to help match loops with equations.

BCD--> Braille; Final Equations and Gate

■

Minimization of problems of this complexity would be done using a computer-aided design program.

However the initial setting up of the problem, probably with the aid of Karnaugh maps, would have to be done manually.

Also you need to know what the program is trying to do. For example, does it give half-maps complete priority? this is undesirable for *array logic* implementations.

Appendix: Another Multiple Output Example

The Common Errors Section Is At the End

The 7-Segment Display

Another Example with many Multiple Outputs

An example much like the BCD to Braille example

My opinion is that setting up the equations is the most useful thing to learn here.

It has 7 outputs instead of 4.

Before rule 2 was important and rule 3 was simple.
Here rule 2 does nothing, and rule 3 is complex.

This gives more practice rather than giving new concepts.

Seven-Segment Display Driver

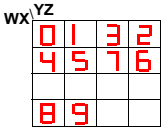
Design Example

The slide above, shows the bars in a seven segment display such as is used in many automotive dashboard displays, or other bright displays[1]. All the digits from 0 through nine can be shown by lighting the proper bars.

Design a circuit which takes a binary-coded-decimal (BCD) digit in on leads W,X,Y and Z and sends out the signals to light the 7-segment display on leads a, b, c, d, e, f, and g. Binary-coded decimal (BCD) digits only go from 0 to 9. The other numbers, 10 through 15 will never be received as inputs. Utilize this fact in your solution.

5-11.•PROBLEM

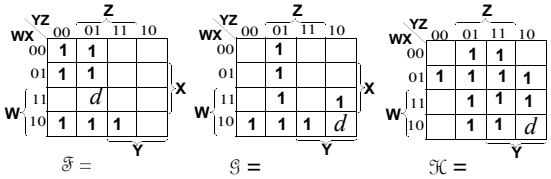
The digits on the right have a revised form for 1, 7, 6 and 9. Derive the maps for the display drivers.



Revised display

5-12.•PROBLEM

Find the minimum circuit with the three outputs defined by the maps below. This is a hard problem. You should read over the example for the 7-segment display drivers before attempting it.



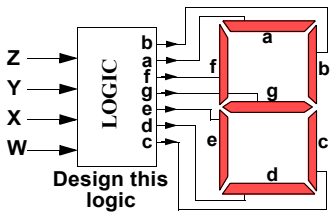
[1] The bright displays, such as on clocks and *Walk/Don't-Walk* signals use light-emitting diodes. The dimmer watch and control panel displays are usually liquid crystal and have more complex driver logic.

ExampleOutput Minimization; 7-Segment Display

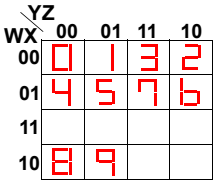
7-Segment Display Driver,

WX \ YZ	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

BCD Digits in binary



Design this logic



Decimal digits displayed

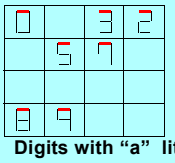
Design Driver Logic

4 inputs, 7 outputs
7 maps, each with 6 don't cares

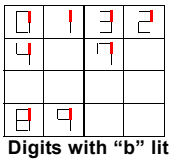
Generate Maps

Choose segment "a" find all the squares where "a" is lit.

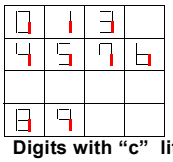
Repeat for "b", "c", . . .



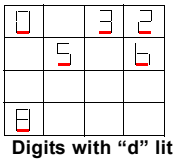
Digits with "a" lit



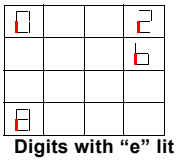
Digits with "b" lit



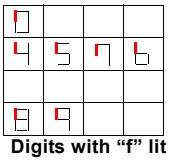
Digits with "c" lit



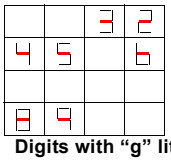
Digits with "d" lit



Digits with "e" lit



Digits with "f" lit



Digits with "g" lit

Rules for Multi-map Minimization

These are the same suggested rules as used in the BCD--> Braille example. If you have not thoroughly studied that, the rules will be hard to understand. However, the rules are explained in more detail in the next few pages.

The rules here are stated a little more formally than in the BCD--> Braille example.

The rules are fairly automatic up to 4a. At rule 4a one has to use more judgment, intuition and low cunning.

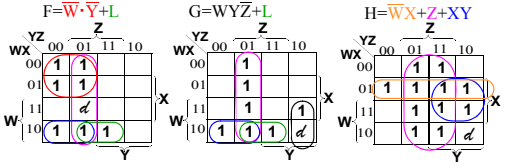
After doing these rules, circling any remaining squares is usually fairly easy.

If the minimization is for *array logic*, which will be studied later. The first rule (loop the half maps) is not done. In *array logic* AND and OR gates are prebuilt, so a 1-input AND and one with more inputs use the same resources.

Solution to Prob 5-12.•

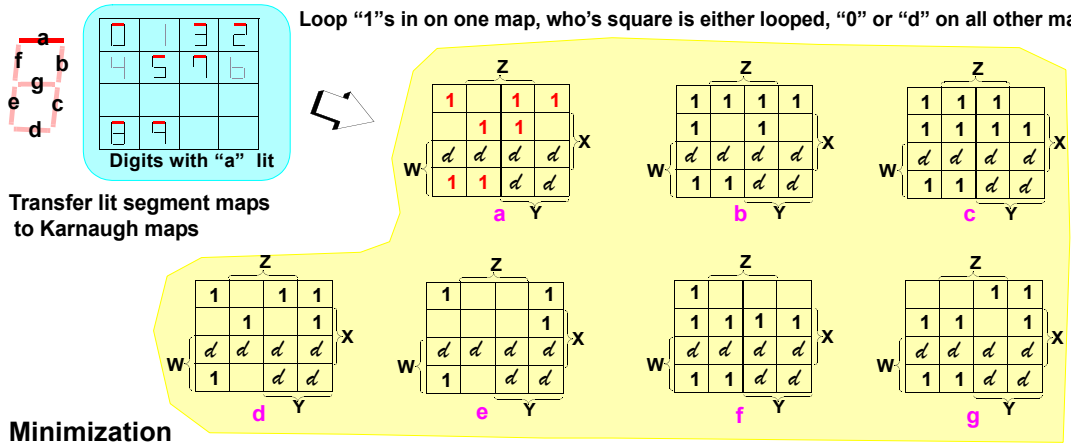
$L=\overline{Y}Z+W\overline{X}\cdot\overline{Y}+W\overline{X}Z$

$F=\overline{W}\cdot\overline{Y}+L$ $G=WY\overline{Z}+L$ $H=\overline{W}X+Z+XY$



20 letters, 27 gate inputs, 11 gates

Maps for 7-Segment Display Drivers



Minimization

Do find loops common to two or more maps
using these suggested rules introduced in the Braille example:

(1) Look for half-map loops
(2) Loop "1"s on one map who's squares on other maps are "0". (No friends rule)
(2a) Loop "1"s on one map, who's friends on other maps, are looped, "0" or "*d*".
(3) With several "loop next" candidates, put off the ones with choices of loops, until the end.
(4) Loop "1"s with brothers on the same map, except looped neighbours or "*d*".
(4a) Expand the candidate loops, created in (3), provided you can also expand on the other maps.
- Keep loops that are max size on the family map.
(4b) Otherwise think about it! Some expansions may interact with (4a)

Maps for 7-Segment Display Driver

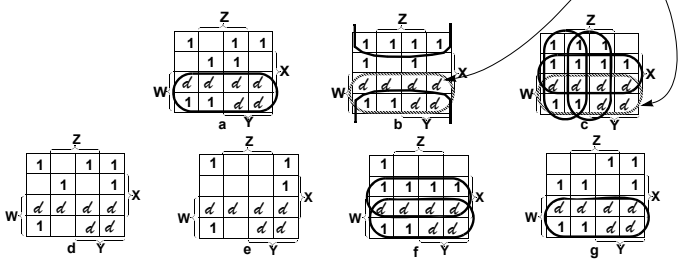
Minimization (continued)

Loops that cover half the map

These are represented by a single letter and are particularly good.
Since they only contain a single letter, they do not need an AND gate. The input can feed directly into the OR gate.
There is no advantage to sharing these terms between maps because there is no hardware to share.

5. Locate all loops which, with "*d*"s if needed, cover half of a map.
There are some ten of them in this example.

6. It is easy to overdo this rule Two of these loops cover no "1"s that are not covered by other loops. The only new squares they cover contain "d's and hence are useless. Remove such loops. The "b" and "c" maps have such useless loops.



7-Segment Display; Loop Half-Maps

Minimization Rule ,

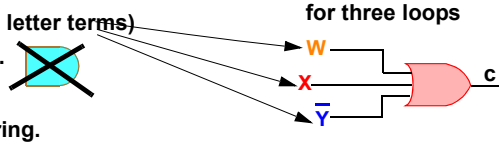
Half-Map Rule

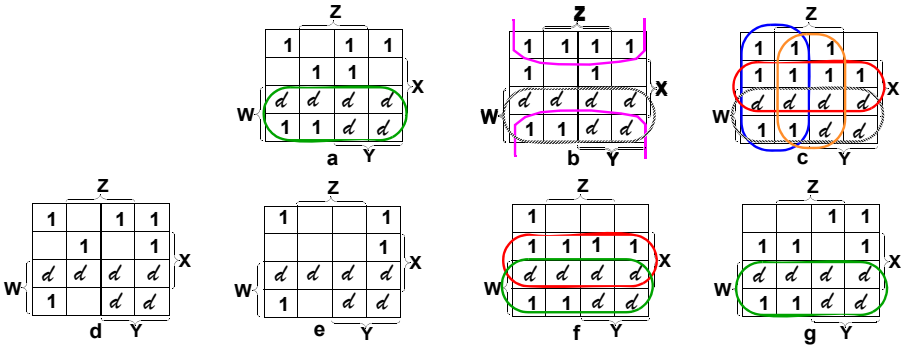
(Rule 1) Look for half-map loops (one letter terms)

These do not require an AND gate.

Hence they can always be looped without loss of potential gate sharing.

Example: c needs only one AND gate for three loops





These loops are all half-map loops.

A common error:

The dashed loops, W on the "b" and "c" maps, are redundant.

Minimization (continued)

Rule (2) The No Friends Rule

A "1" which appears on only one map, can never be shared. These should be looped on the one map with the largest possible loop.

Unfortunately there are no friendless "1"s in the seven-segment decoder.

Rule (2a) All My Friends Have Gone to The Dark Side Rule

These "1"s might as well be looped now, because there is no saving from sharing the loop with another map where the square is already looped or a "*d*".

Unfortunately, all the "1"s have friends on one one or more maps.

Which maps use any particular square is summarized in the map in the upper right corner above.

7-Segment Display; No Friends

Minimization Rule (2) and (2a)

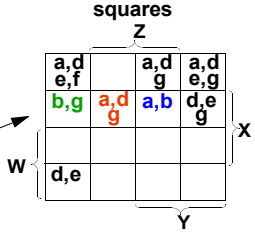
No Friends Rule

(2) Loop "1"s on one map who's square on other maps are "0"
(Have no friend on friends maps) ***There are none here.***

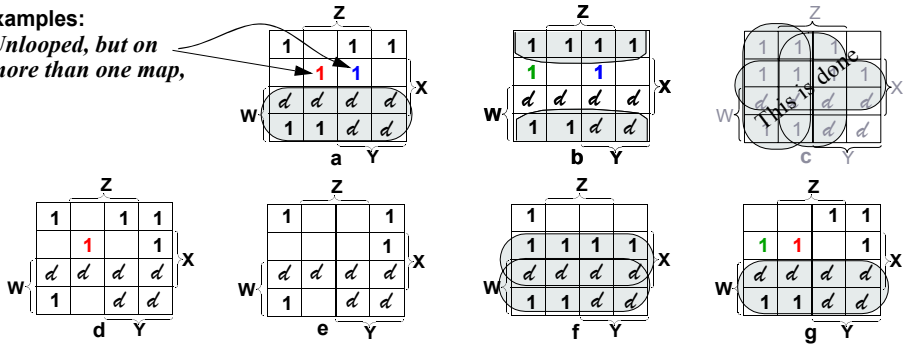
My Friends Have Gone to the Dark Side

(2a) Loop "1"s on one map, who's square is either "0", looped or "*d*" on all other maps.
These cannot be usefully shared with other maps.
There are none here.

Maps with unlooped squares



Examples:
Unlooped, but on more than one map,



Loop "1"s in on one map, who's square is either looped, "0" or "d" on all other maps.

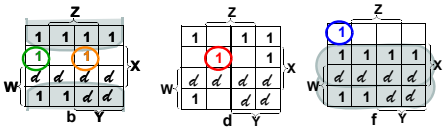
Minimization (continued)

Rule (4) Loop single "1"s which have no brothers on the same map except looped "1"s or "d"s.

Loop the single "1"s.
There have to be "1"s in the corresponding squares in other map(s), or they would have been looped in rule 2. These "1"s have great potential for sharing. Further, by starting with the smallest loop, we do not lose sharing opportunities.

In this example maps b, d and f, have single squares. The fact that the loops could expand into "*d*"s or into already looped areas does not matter. This is handled in the next rule.

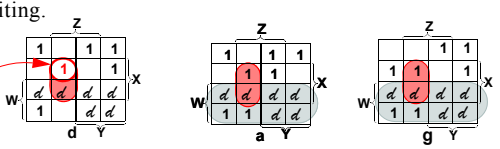
These single square loops are candidate loops.



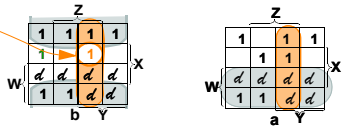
Rule (4a)Expand the loops on the family map and on the friends maps.

Stop when any of the loops cannot be further expanded.
Stop for good if the family map loop is doing the limiting.
Otherwise see Rule (4b)

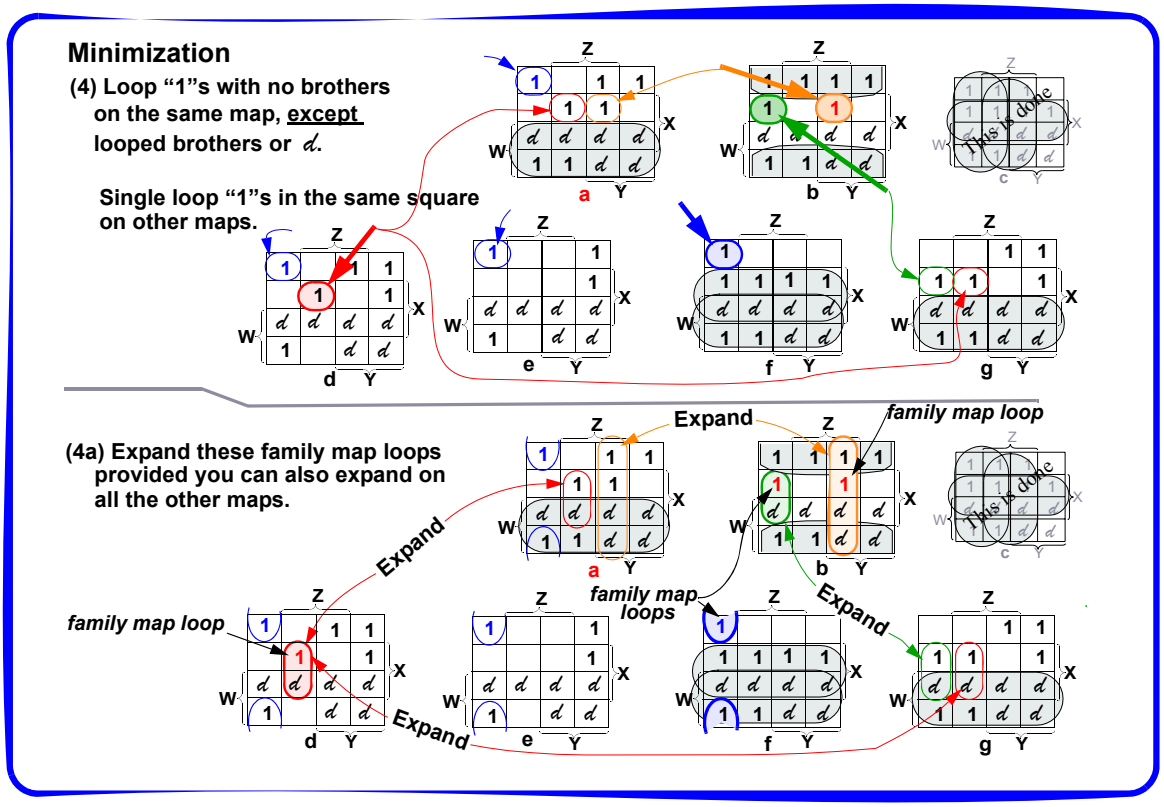
Example; the original family map loop in d can be expanded to two squares in d, a and g. This is the maximum expansion in d. and also on the friend's maps a and g,



Similarly, the family map expansion shown in b can be expanded in a. It is limited on the family map, and not further limited by the other maps, so we expand to the full 4 squares.



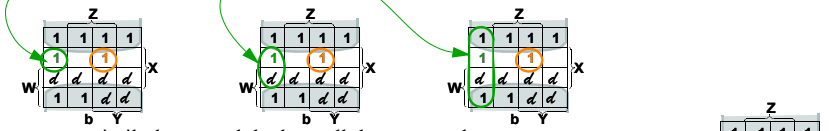
7-Segment Display; Single Child Rule



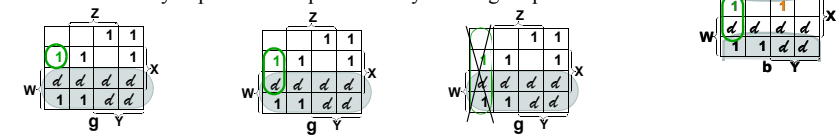
Minimization (continued)

Rule (4and 4b) Expand the family map loops but check before over expanding.

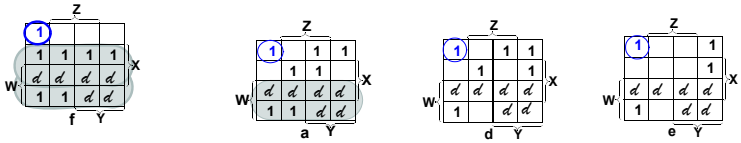
Previously the expansion of the loop on the family map was limited on that map
Here we are expanding loops that are limited by the friend maps.
Thus square $\overline{W}X\overline{Y}\cdot\overline{Z}$ can expand to $X\overline{Y}\cdot\overline{Z}$ or to $\overline{Y}\cdot\overline{Z}$ on map b.



However one cannot similarly expand the loop all the way on the g map.

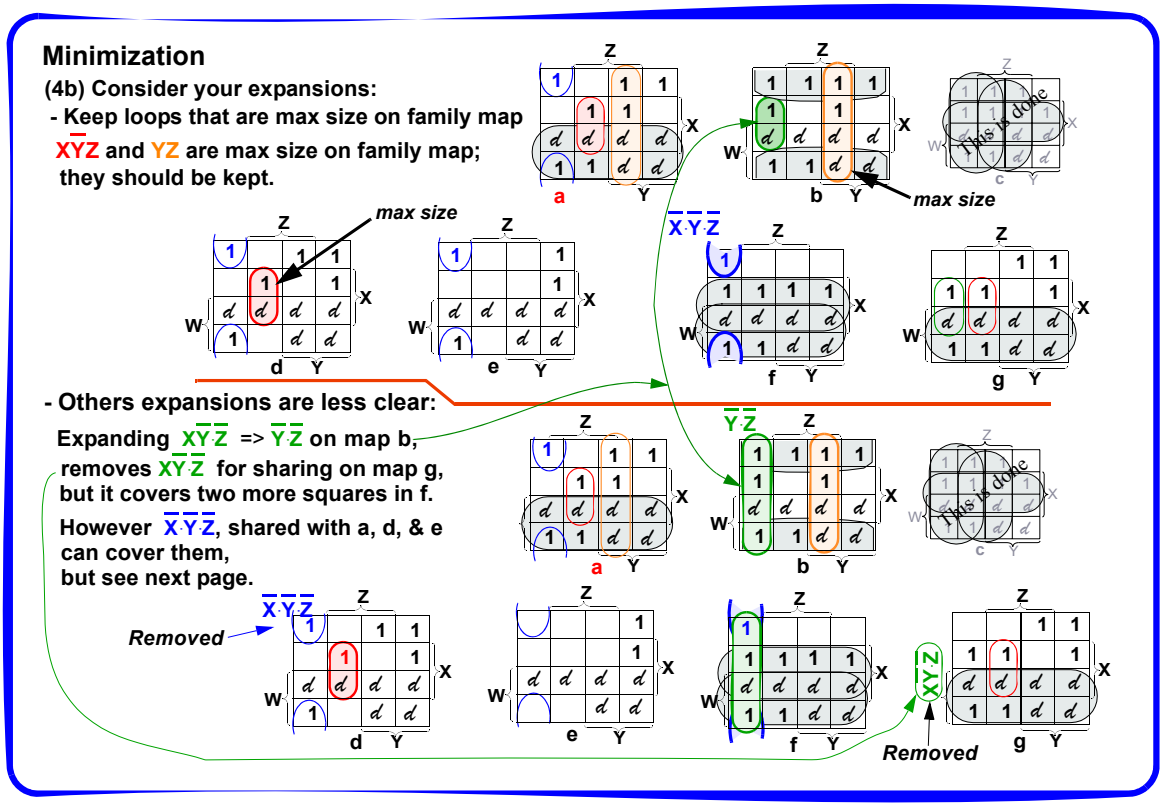


One could ignore sharing a gate with g and expand the starting loop to $\overline{Y}\cdot\overline{Z}$, the full column.
However look at f. There $\overline{Y}\cdot\overline{Z}$ can be used to cover the starting square in the corner. This saves a gate, provided there is a good way (4 corners?) to loop that corner square on the other maps, namely a, d and e.



Four corners is a good way, as shown on the next slide, but it turns out not to be quite as good.

7-Seg Display; Don't Overexpand On Family Map



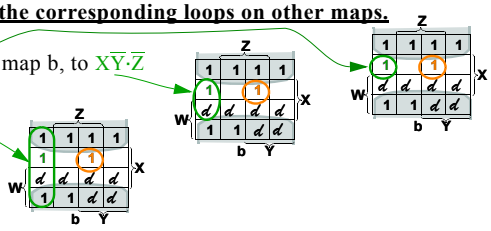
Minimization (continued)

Rule (4b) Expand the family map loops and all the corresponding loops on other maps.

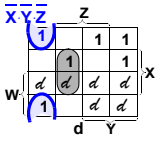
Two partial solutions emerge:

Solution (a) expands candidate square W̄XȲ·Z̄ on map b, to XȲ·Z̄

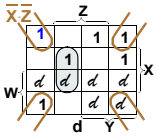
Solution (b) overexpands it to Ȳ·Z̄.



Solution (a) loops 2 corners.



Solution (b) loops 4 corners.



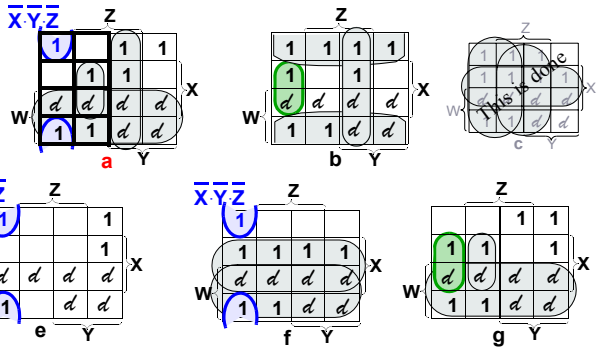
It is not yet clear which will be better.

7-Seg Display; Checking Expansion

Minimization

Solution (a)

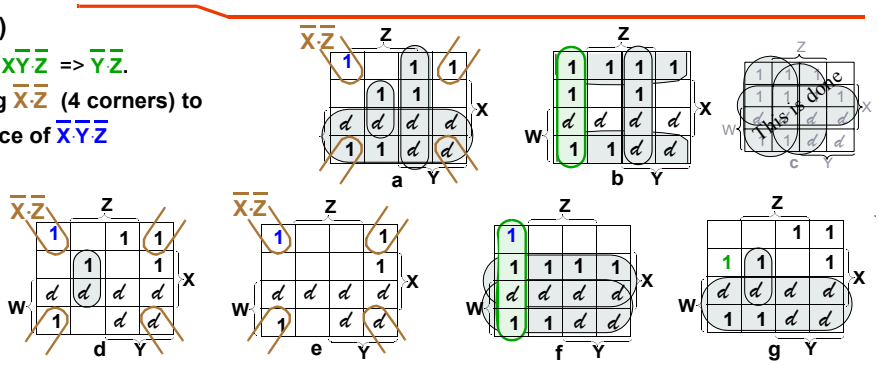
sharing $\overline{X}\cdot\overline{Y}\cdot\overline{Z}$ on a, d, e & f.



Solution (b)

expanding $XY\cdot\overline{Z}$ => $Y\cdot\overline{Z}$.

Then adding $\overline{X}\cdot\overline{Z}$ (4 corners) to take the place of $\overline{X}\cdot\overline{Y}\cdot\overline{Z}$



7-Seg Display; Checking Expansion ■

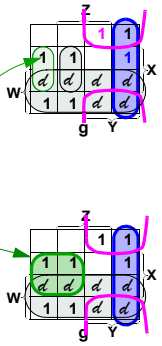
Minimization (the end)

Try the best trial solutions.

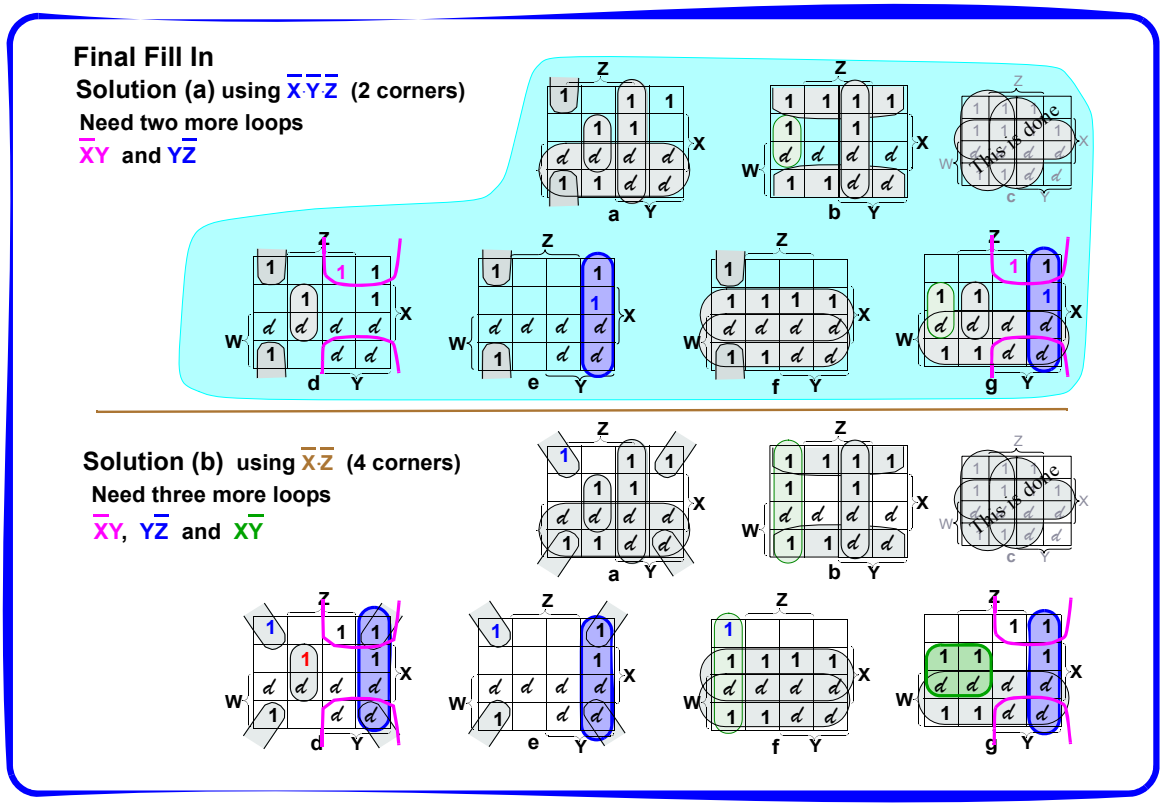
It is only at the end that we can tell which is the best solution. Here the g map decides the answer:

The (a) solution g map looks like it has more gates. However every AND gate is shared with another map.

The (b) solution has a 4-square loop, XY, which is only used on the g map, and thus costs an extra gate.



7-Seg Display; Final Fill In



Display Drivers, Forming Equations

One way of forming equations is to use a letter like J, H, L .. for each term and put the letter in the squares covered by the loop for the term.Thus $Q = \overline{X}\cdot\overline{Y}\cdot\overline{Z}$ is written in the two left-hand corners on four maps. To avoid confusion, leave a 1 in squares which are covered by several loops.

One writes the equations for the segments as the OR of these letters. Thus e = Q + P

Terms which have only one input like X, do not require a special letter, and we give them the name of the input variable.

Final Results

With no sharing of gates	Maximum Sharing Solution (a)	Less Sharing Solution (b)	Using outputs in Solution (a) as inputs for other outputs.
46 letters (literals) 23 gates (16 AND) 58 gate inputs	40 letters 13 gates (7 AND) 40 gate inputs	37 letters 14 gates (6 AND) 38 gate inputs	38 letters 13 gates (7 AND) 38 gate inputs

Here sharing saves much more than it did for the Braille problem.

As is generally observed, sharing mainly reduces the number of gates. It has less effect on letters or gate inputs, in fact it may even increase these. For example Solution (a) has more sharing and fewer gates, but it has more letters and more gate inputs.

5-13.•PROBLEM (Do 5-11.• before doing this.)

Minimize the equations for multiple outputs, using the revised display for numbers 1, 7, 6 and 9. Follow the methods used in the last few pages. Keep the same capital letters for all expressions that do not change. If you need new expressions, the letters K, Q, R, S, T, U and V have not been used

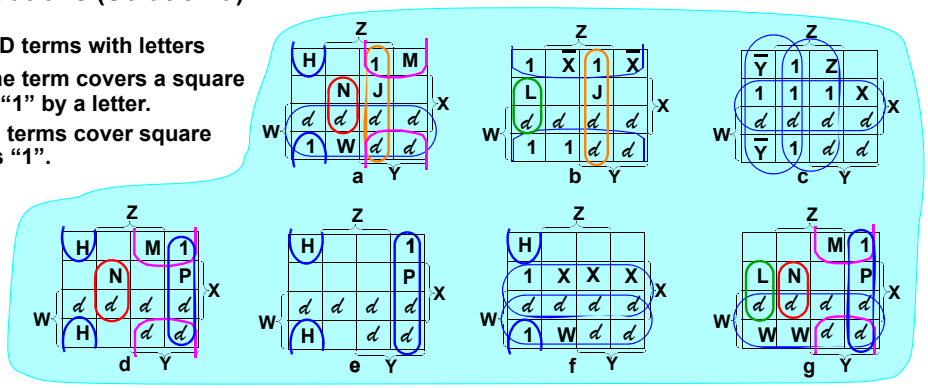
7-Seg Display; Final Equations

Form Equations (Solution a)

Label AND terms with letters

If only one term covers a square replace "1" by a letter.

If several terms cover square leave as "1".



$J = YZ$ $\quad M = \overline{X}Y$ $\quad a = H + J + M + N + W$ $\quad e = H + P$

$H = \overline{X}\cdot\overline{Y}\cdot\overline{Z}$ $\quad N = X\overline{Y}Z$ $\quad b = J + L + \overline{X}$ $\quad f = H + W + X$

$L = X\cdot\overline{Y}\cdot\overline{Z}$ $\quad P = Y\overline{Z}$ $\quad c = Z + \overline{Y} + X$ $\quad g = L + M + N + P + W$

$d = H + N + M + P$

All terms reused.

Size measures

40 letters (literals)

13 gates (6 ANDs)

40 gate inputs

Using: $c = f + Z$ $\quad d = e + N + M$

38 letters 13 gates 38 gate inputs

Common Errors With Karnaugh Maps

Check for wraparounds

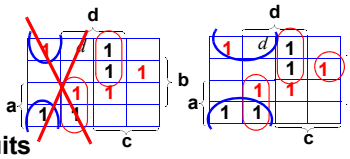
- Check for 4-corners
- Check for wrapround on opposite side
- Check for wraparound top-to-bottom
- Check for expansion of a loop to 4 squares or 8 squares
-
- Learn to spot squares with no friends.

Common Map Errors

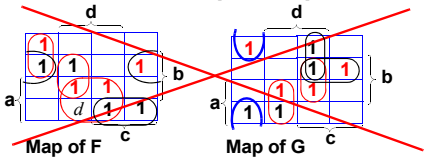
Common Errors

…Check Your Map Entries
If you put one variable in the wrong squares, you're toast!

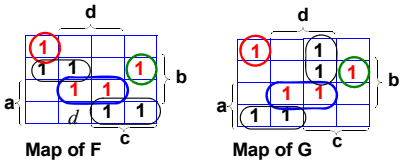
…Check for Wraparound, and Wide Wrap Around



…Don't Treat Multiple Output Problems Like Unrelated Circuits



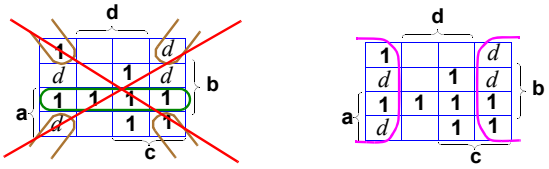
12 gates



9 gates, 3 shared

…Don't Treat Two-Output Problems Like 5-Variable Problems

…Do Half-Maps First (Except for PLAs in a later section)



…*No friends*, and other rules are heuristics, they often, but <u>not always,</u> work.