# EE103 Digital Systems 1

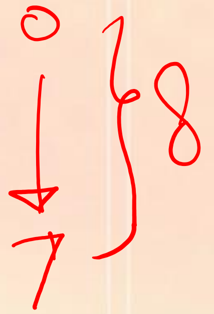
**Wong Chin Hong**

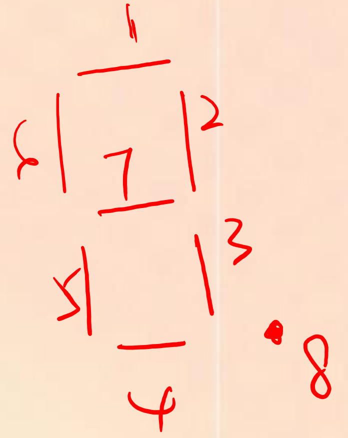
106

Maynooth International Engineering College
Fuzhou University

# So far … !

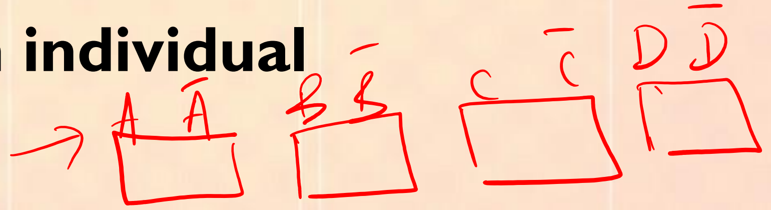
- We've used Karnaugh Maps to minimise logic …
- We've implemented circuits using NAND only or NOR only gates …
- We've analysed and designed a counter …





***TODAY, we are going to look at Multi-output Minimisation and ALSO the 7-segment display***

# Multi-output Minimisation



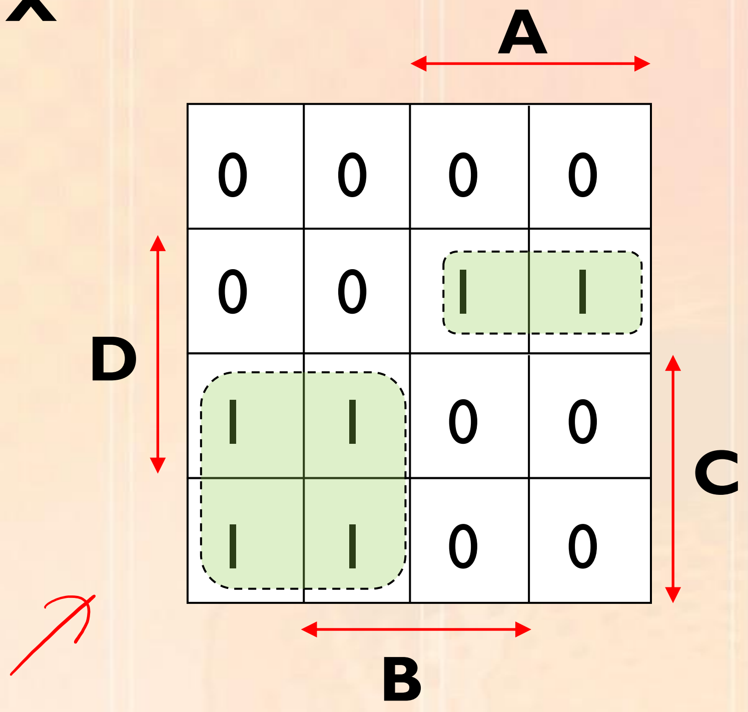
- Until now, we have considered minimisation **on an individual level**.
- In other words, we have minimised logic for a single output for a given set of inputs.
- However, many digital circuits have multiple outputs associated with the same set of inputs.
- For example, consider the JK counter in the previous set of notes. This had 4 flipflops, with two inputs each. Thus, we had 8 outputs in total (and 8 Karnaugh Maps) associated with the same set of inputs.

## Multi-output Minimisation

- Such circuits **require the implementation of several functions for the same set of input variables**.
- As such, it is useful to solve these functions from a collective viewpoint, i.e. we look to minimise the logic across all outputs simultaneously (and not treat each output individually). X
- By way of a simple illustration, consider the following two Karnaugh Maps for arbitrary functions X and Y:

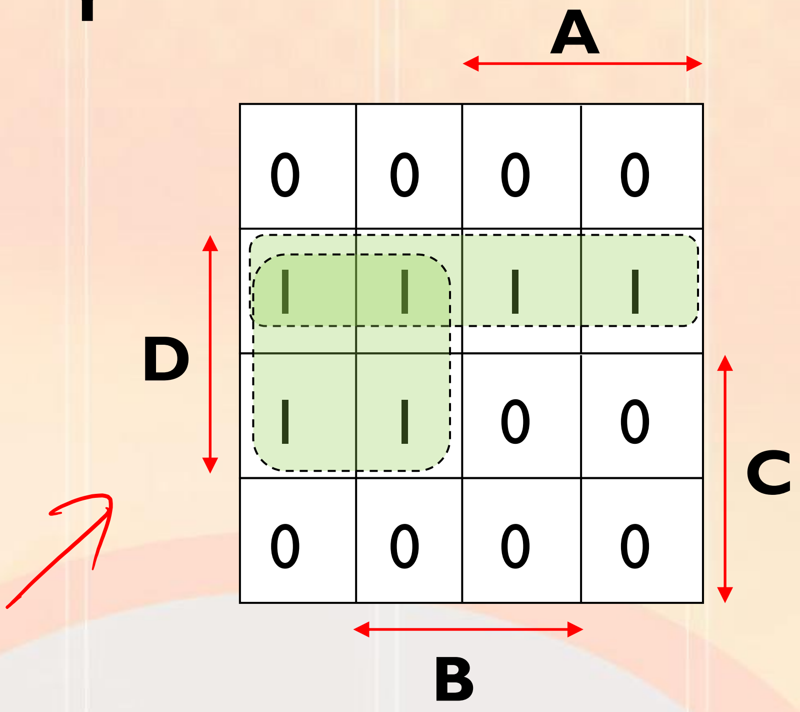
# Multi-output Minimisation

X



| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |

Y



| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |

**Minimising Individually …**

# Multi-output Minimisation

- Minimising each function individually gives:

$$X = \overset{\checkmark}{\bar{A}C} \bigcirc\!\!\!\!+\; \overset{\checkmark}{A\bar{C}D}$$

4 AND

and

$$\xrightarrow{\quad} 2 \text{ OR}$$

$$Y = \overset{\checkmark}{\bar{C}D} \bigcirc\!\!\!\!+\; \overset{\checkmark}{\bar{A}D}$$

- This requires a **total of** $\boxed{\textbf{6 gates}}$ (4 x AND and 2 x OR).

- However, if we consider both maps collectively, we obtain the following minimal solution:

# Multi-output Minimisation

① ②

$X = \underline{AD\bar{C}} \; \textcircled{+} \; \bar{A}C$

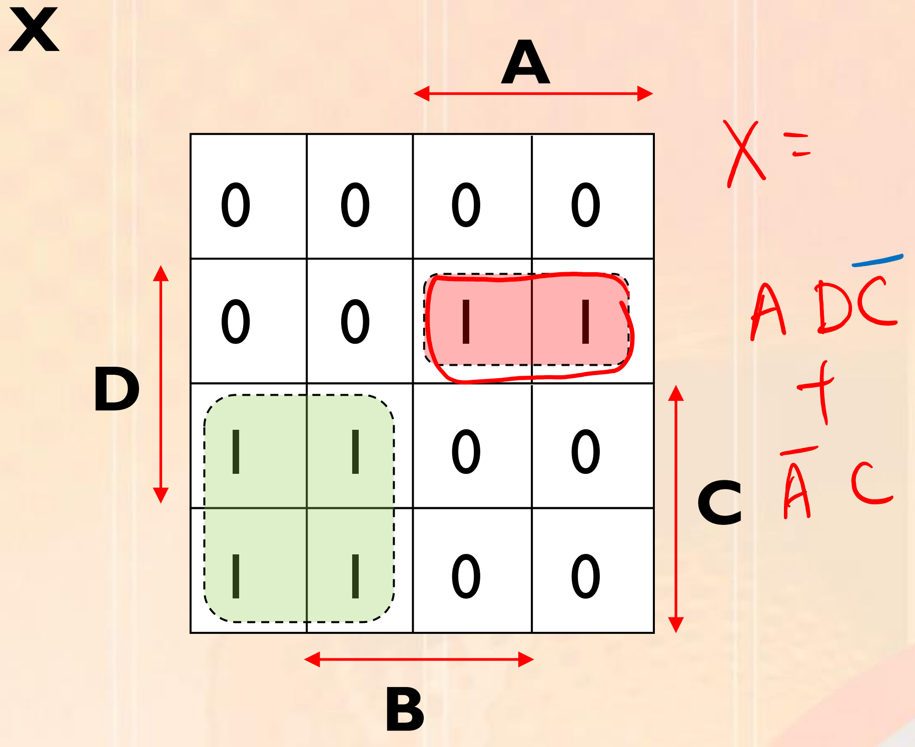
$Y = \underline{AD\bar{C}} \; \textcircled{+} \; \bar{A}D$

$\cancel{4}$ ③

5 logic gates

3 AND

2 OR



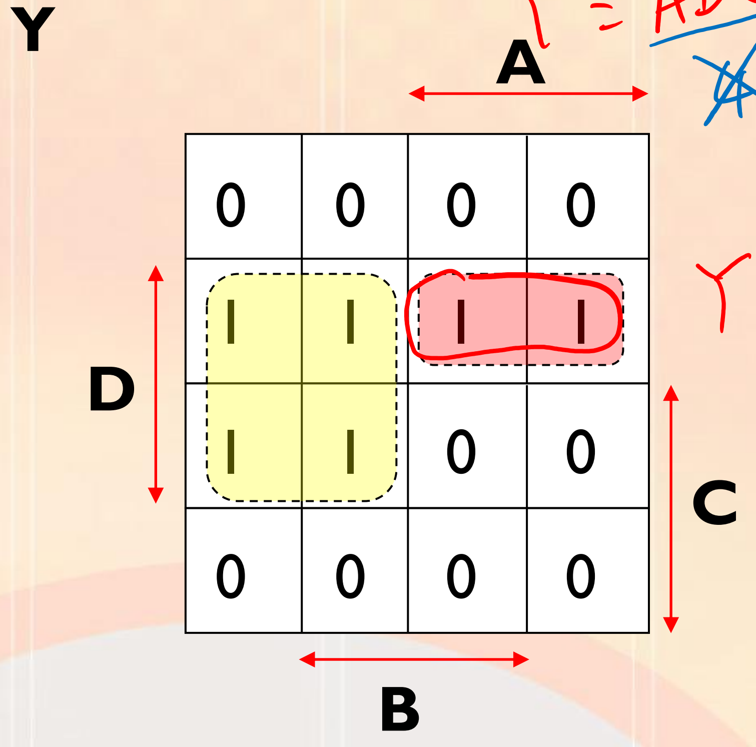
| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |

X =

$AD\bar{C}$

$+$

$\bar{A}C$



| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Y =

$AD\bar{C}$

$+$

$\bar{A}D$.

**Minimising Collectively …**

## Multi-output Minimisation

- Minimising each function individually gives:

$$X = \overline{A}C + A\overline{C}D$$

and

$$Y = A\overline{C}D + \overline{A}D$$

- This requires a **total of 5 gates** (3 x AND and 2 x OR).

- Hence, although we do not have a minimal individual solution for function Y, we have nevertheless an overall multi-output solution that is minimal in terms of gates required.

## Multi-output Minimisation

- This is because ***we are reusing available gates from other functions***. For example, we use the AND gate $A\overline{C}D$ for function Y as this is already required by function X.

- This technique is referred to as **multi-output (or multiple-output) minimisation.**

# Multi-output Minimisation

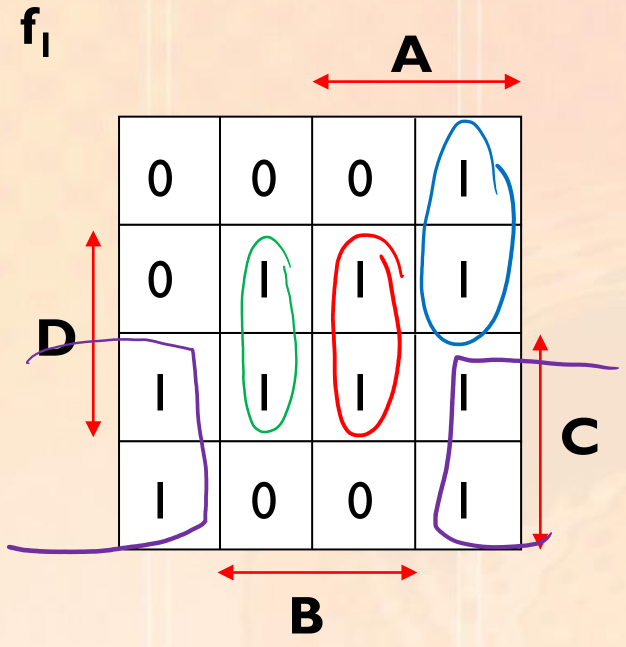
- *Ex. 8.1 Using multi-output minimisation, design a circuit with 4 inputs and 3 outputs which implements the functions:*

$$f_{1(A,B,C,D)} = \sum(2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$$

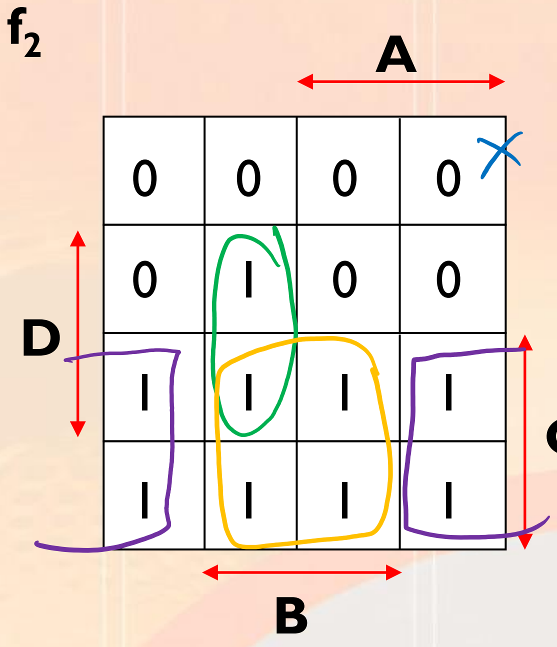
$$f_{2(A,B,C,D)} = \sum(2, 3, 5, 6, 7, 10, 11, 14, 15)$$

$$f_{3(A,B,C,D)} = \sum(6, 7, 8, 9, 13, 14, 15)$$

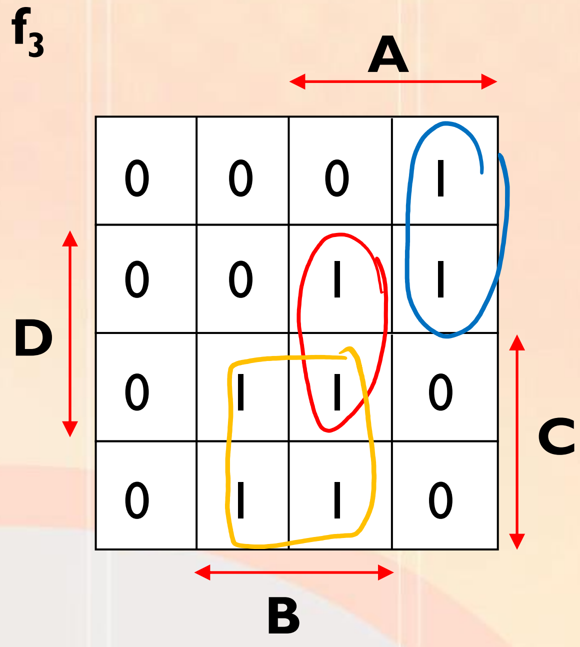
# Multi-output Minimisation



| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |



| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |



| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 |

# Multi-output Minimisation

- Assuming that complemented inputs are available, this **gives an 8 gate solution** (3 x OR and $\boxed{5 \text{ x AND}}$) as follows:

✓ $f_1 = \overset{1}{\bar{A}BD} + \overset{2}{\boxed{ABD}} + \overset{3}{A\bar{B}\bar{C}} + \overset{4}{\bar{B}C}$

✓ $f_2 = C + \overset{1}{\bar{A}BD}$

✓ $f_3 = \underset{5}{BC} + \underset{3}{A\bar{B}\bar{C}} + \overset{2}{\boxed{ABD}}$

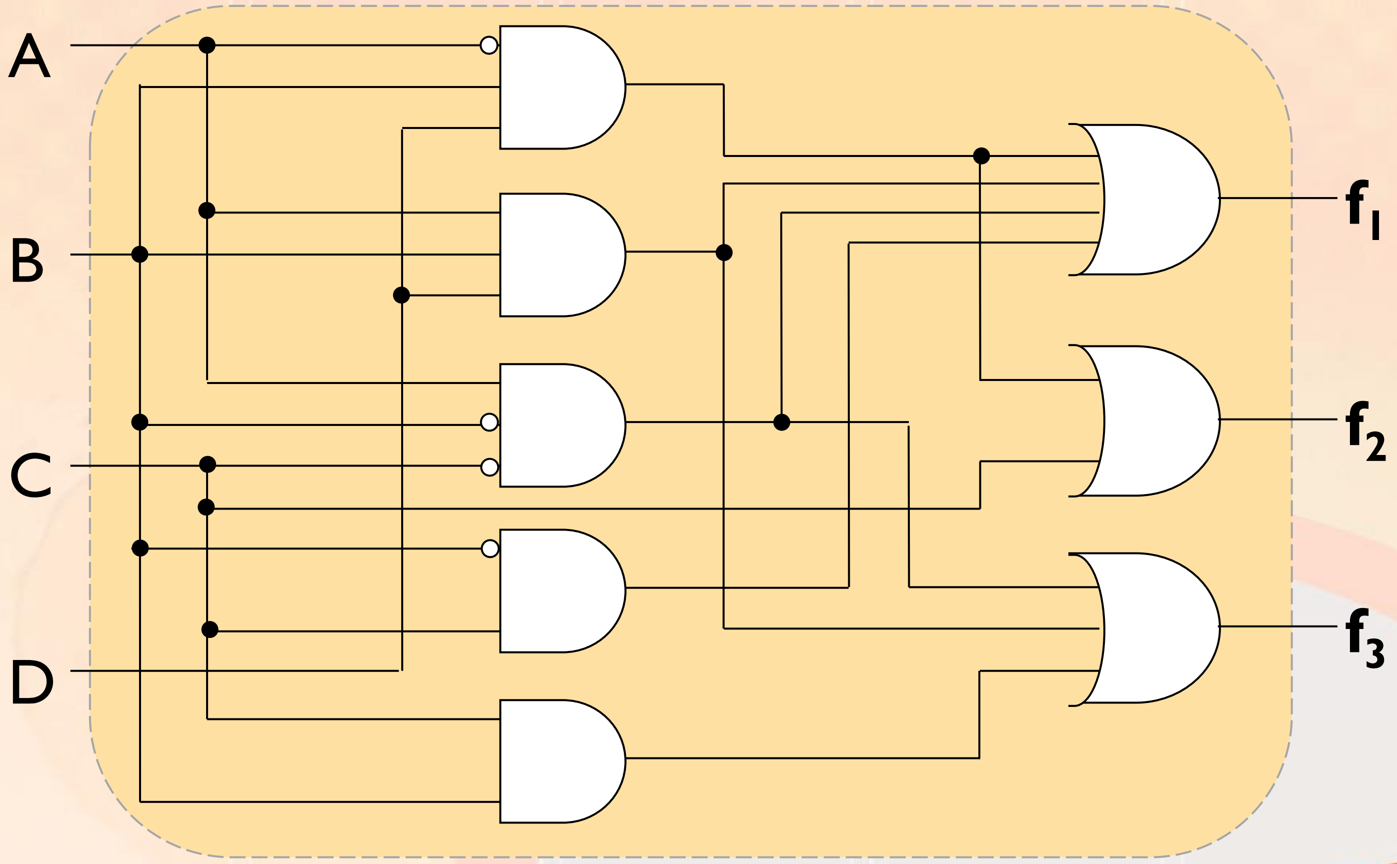
5 AND gates.
3 OR gates

- Note – individually, these are not minimal solutions. However, **had we minimised each function individually**, we would require an overall **total of 10 gates**. *Verify this yourself.*

minimised if individually.

# Multi-output Minimisation

- Finally, the overall circuit implementation is as follows:



# Multi-output Minimisation

- *Ex. 8.2 Using multi-output minimisation, obtain a minimal implementation of the following four functions:*

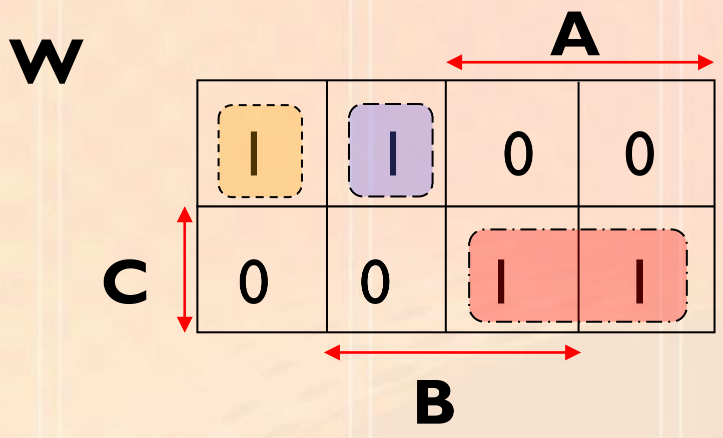
$$W_{(A,B,C)} = \Sigma(0, 2, 5, 7)$$

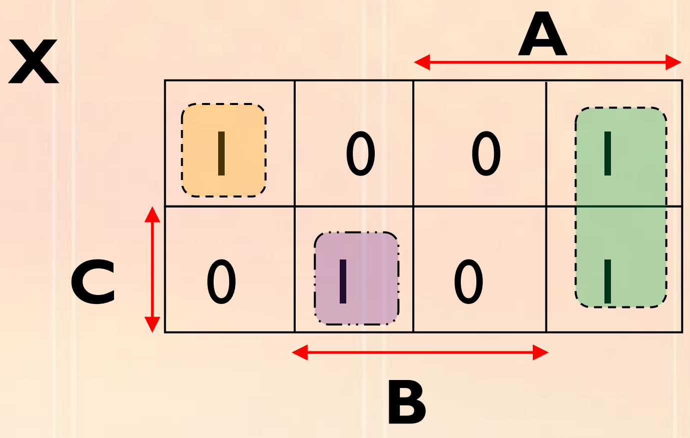
$$X_{(A,B,C)} = \Sigma(0, 3, 4, 5)$$

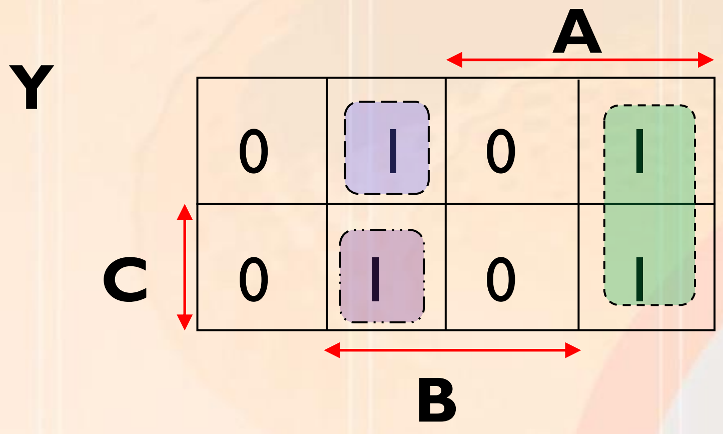
$$Y_{(A,B,C)} = \Sigma(2, 3, 4, 5)$$

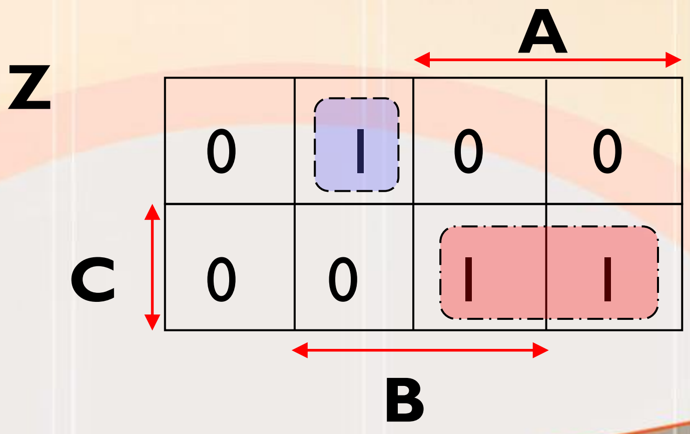
$$Z_{(A,B,C)} = \Sigma(2, 5, 7)$$

# Multi-output Minimisation









# Multi-output Minimisation

- Assuming that complemented inputs are available, this **gives a <u>9</u> gate solution** (4 x OR and 5 x AND) as follows:

✓ $W = \overset{1}{\boxed{\overline{A}\,\overline{B}\,\overline{C}}} + \overset{3}{\underset{\sim\sim\sim}{\overline{A}B\overline{C}}} + \overset{4}{(AC)}$

✓ $X = \boxed{\overline{A}\,\overline{B}\,\overline{C}} + \overset{2}{\bigcirc\!\!\!\!\!\!\!\!\!\!\!\!\;\overline{A}BC} + [A\overline{B}]^{5}$

✓ $Y = \underset{\sim\sim\sim}{\overline{A}B\overline{C}} + \overline{A}BC + [A\overline{B}]$

✓ $Z = \underset{\sim\sim\sim}{\overline{A}B\overline{C}} + (AC)$

5 AND

4 OR

# The 7-segment display

- A widely used and commonly seen multi-output unit is the 7-segment display.
- These displays are used in conjunction with logic circuits that can decode a BCD (binary coded decimal) number to activate the appropriate digits on the display.
- The 7-segment displays can be found in a wide range of digital devices including watches, alarm clocks, microwave ovens, petrol pumps, etc.

# The 7-segment display

- In this section of the notes, we will look at designing such a decoder circuit, but first we will examine the actual 7-segment display in a little more detail.

- The diagram to the right shows the seven segments (hence the name!) that make up the display.

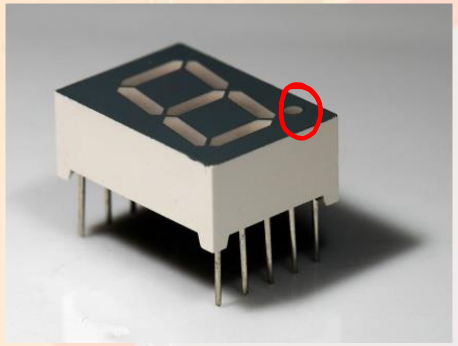
- The segments are labelled a, b, c, d, e, f and g as shown.

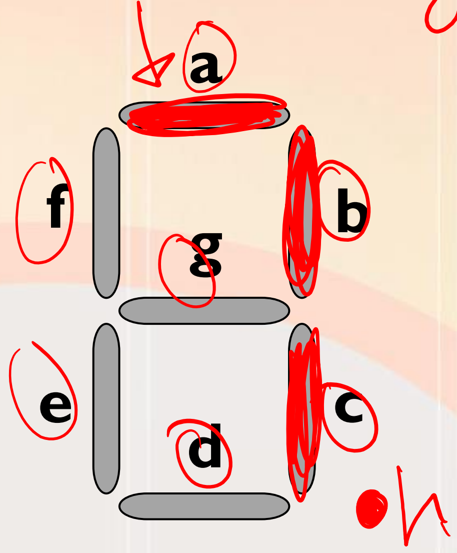
7

0 7 8
4 9
7

a = 1
b = 1
c = 1

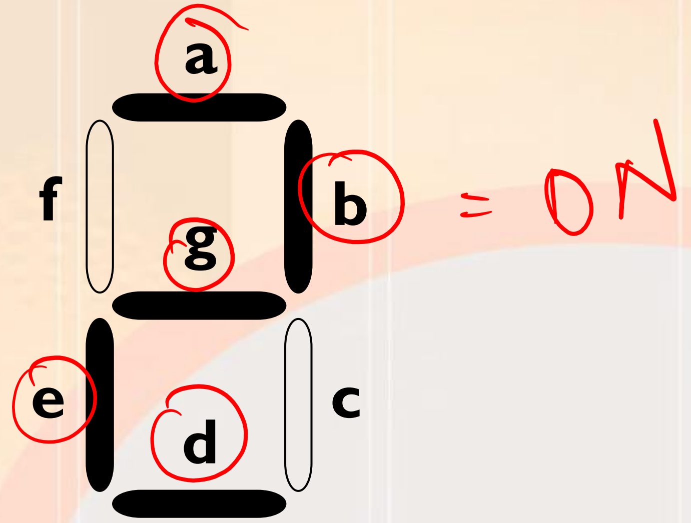
a = 1
c = 1
g = 0
d
e
f
g
: 0





## The 7-segment display

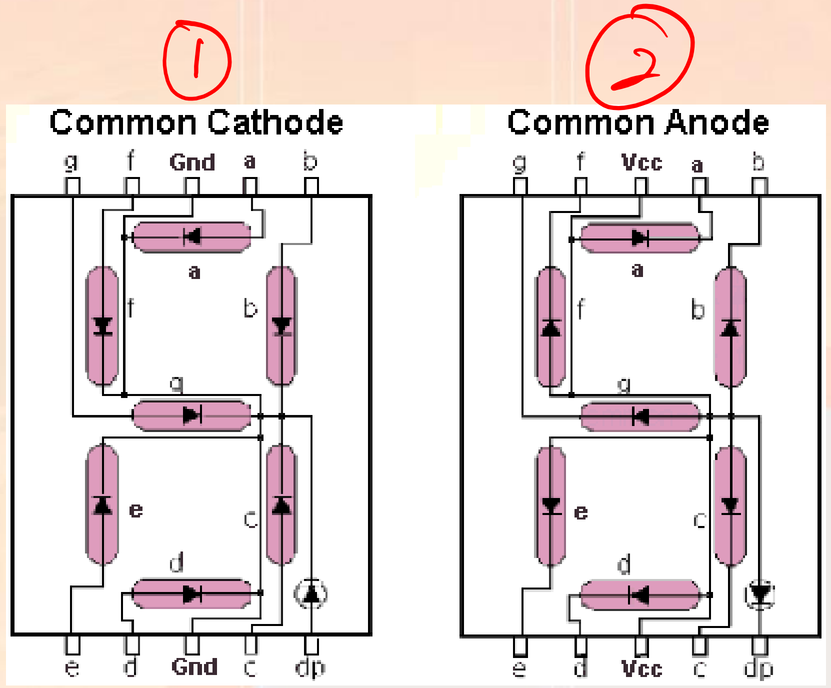
- Each of these segments constitutes a single LED which can be turned on or left off as required.
- Thus, for example, if we wish to display the digit '2' on the display, we would need to turn ON segments a, b, d, e and g, i.e.:



# The 7-segment display

- There are two different ways in which the 7-segment display can be configured and hence, there are two different ways in which the segments can be turned on.

a→g



# The 7-segment display

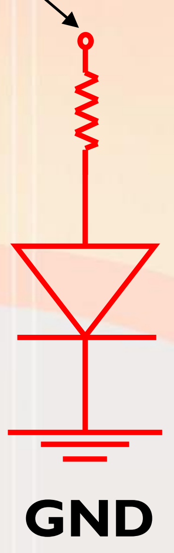
***Common Cathode Mode …***

- In common **cathode (negative)** mode, each segment is **connected to ground** (i.e. all segments have a common ground connection).

- Thus, in order to turn the segment on, we need to **add a logic '1' (i.e. 5V) to its input**.

**Common Cathode Mode**
**- logic '1' to turn on** $\Rightarrow$ 5V or VCC

*5V placed here will cause LED to emit light*



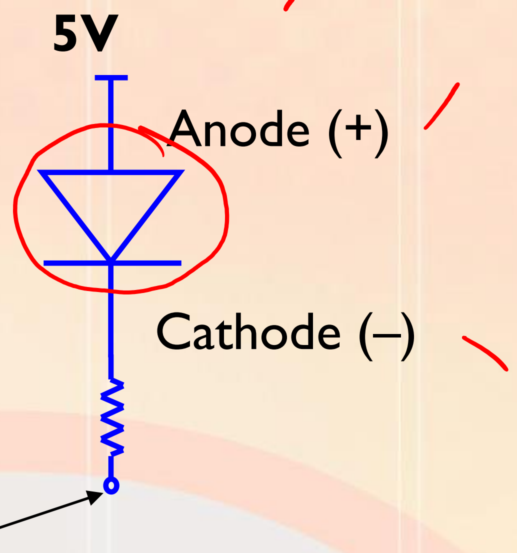
# The 7-segment display

***Common Anode Mode …***

- In common **anode (positive)** mode, each segment is **connected to power** (i.e. all segments have a common 5V connection), as shown below.

- Thus, in order to turn the segment on, we need to **add a logic ‘0’ (i.e. ground) to its input**.

**Common Anode Mode**
**- logic '0' to turn on**



*GND (Ground) placed here will cause LED to emit light*

# BCD to 7-segment Decoder

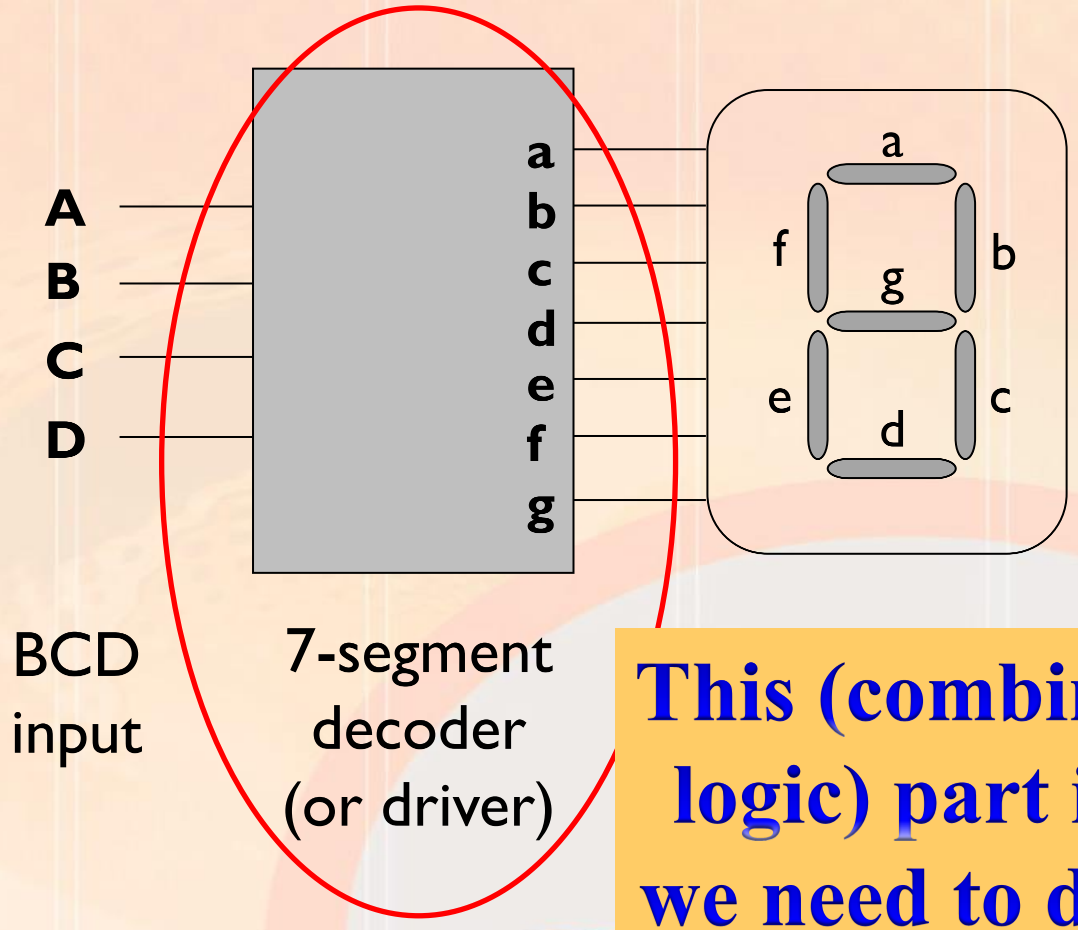
- A BCD to 7-segment decoder is a combinational circuit that accepts a decimal digit in BCD and generates the appropriate outputs for selection of segments in the 7-segment display indicator.

- Recall that the $\overset{ABCD}{8421}$ BCD represents each decimal digit from 0 to 9 using a 4-bit binary code.

- The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the 7-segment display, as shown previously.

# BCD to 7-segment Decoder

- The overall setup for the circuit is as follows:



**This (combinational logic) part is what we need to design…**

# BCD to 7-segment Decoder

- ***Ex. 8.3 Design a BCD to 7-segment decoder for use with a 8421 BCD counter. Invalid states of the counter are to result in a blank display. Implement using NAND gates. Note that the display is connected in common cathode mode.*** → 0

- A single 7-segment display can only display digits 0 to 9. → 4 input

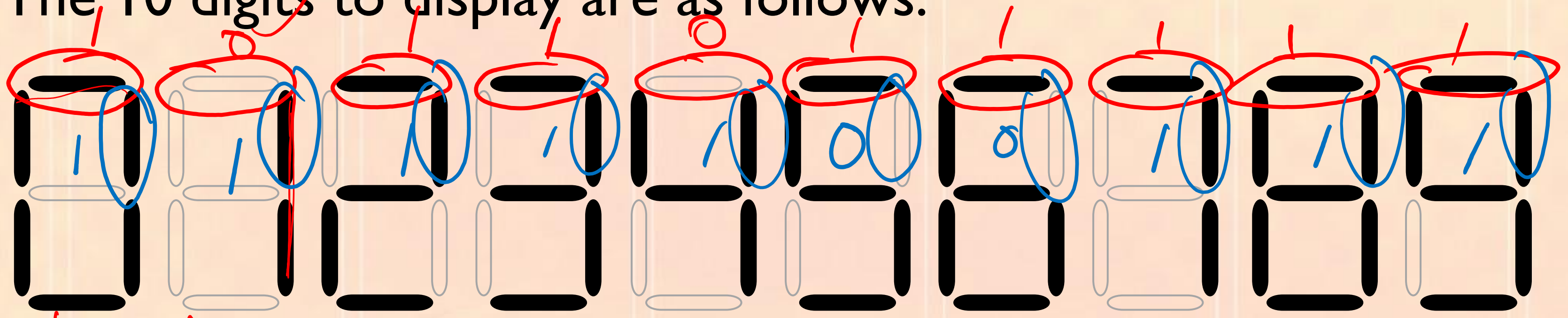
- An 8421 BCD counter can count from 0 to $\boxed{1}\boxed{5}$. 2 7-segment displays

- Hence, the counter outputs 10 to 15 are invalid states.

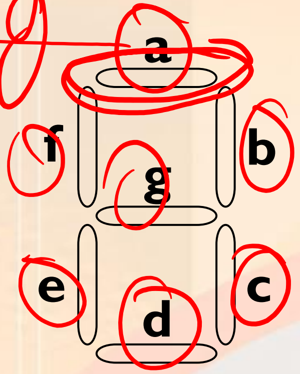
- The 7-segment display is connected in cathode mode and therefore we need to apply a logic '1' in order to switch a segment ON.

# BCD to 7-segment Decoder

- The 10 digits to display are as follows:

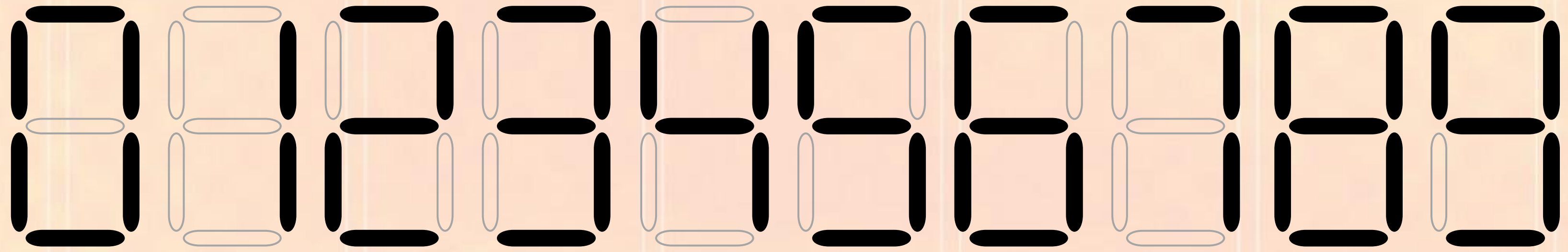


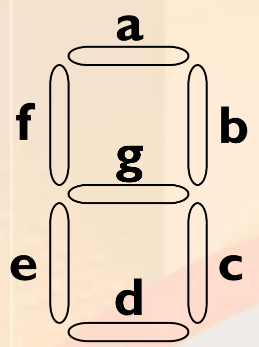
| count | A | B | C | D | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | | | | | |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | | | | | |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | | | | | |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | | | | | |
| 4 | | | | | 0 | 1 | | | | | |
| | | | | | 1 | 0 | | | | | |
| | | | | | 1 | 0 | | | | | |
| | | | | | 1 | 1 | | | | | |
| | | | | | 1 | 1 | | | | | |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | | | | | |



# BCD to 7-segment Decoder

- The 10 digits to display are as follows:





- From this, we can determine a table that shows which of the seven segments need to be switched on and when they need to be switched on.

# BCD to 7-segment Decoder

- Hence the following table:

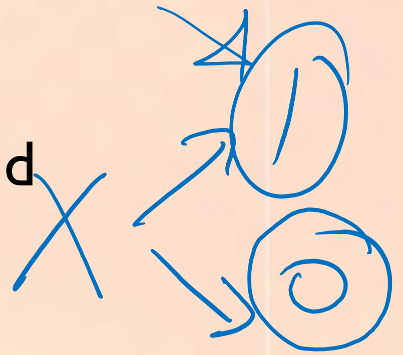
| Count | A | B | C | D | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

1 0 1 0

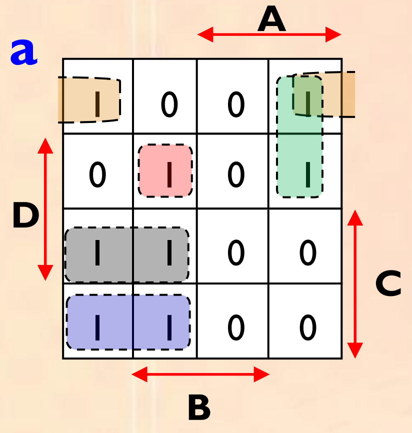
# BCD to 7-segment Decoder

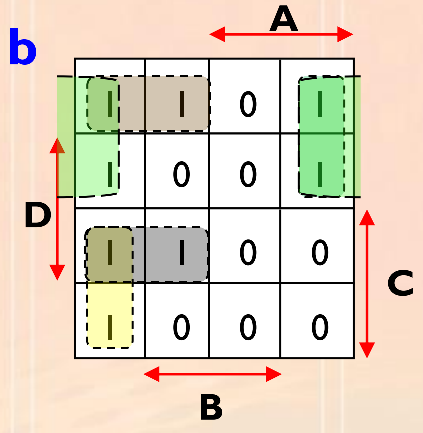
- The next step is to derive a Karnaugh Map for each segment and to carry out **multi-output minimisation** in order to achieve an overall minimal solution.

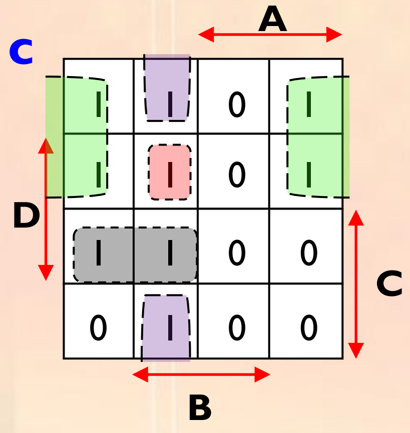
- Unused or invalid states are required to be blank in this case and so need to be inserted as a logic '0' in the Karnaugh Maps.

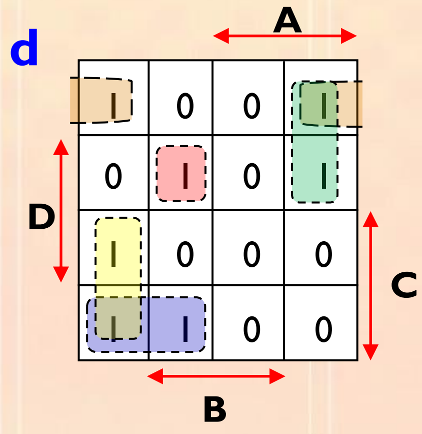


# BCD to 7-segment Decoder

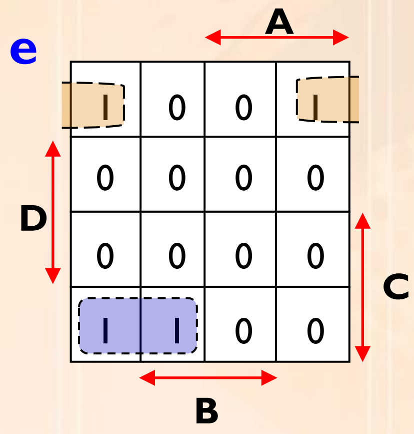


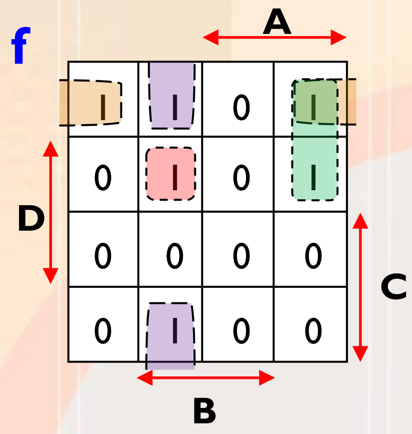


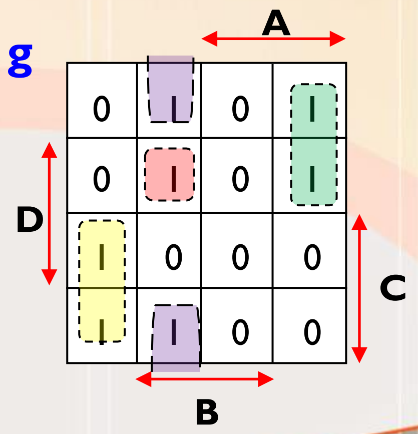




**And hence the final solution !**







# BCD to 7-segment Decoder

- This gives, in NAND-only implementation: → De Morgan.

①

SOP

$$a = A\bar{B}\bar{C} + \bar{A}B\bar{C}D + \bar{A}CD + \bar{A}C\bar{D} + \bar{B}\bar{C}\bar{D} = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{C}D)}.\overline{(\bar{A}CD)}.\overline{(\bar{A}C\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}}$$

$$b = \bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}\bar{C}\bar{D} + \bar{A}CD = \overline{\overline{(\bar{B}\bar{C})}.\overline{(\bar{A}\bar{B}C)}.\overline{(\bar{A}\bar{C}\bar{D})}.\overline{(\bar{A}CD)}}$$

$$c = \bar{B}\bar{C} + \bar{A}B\bar{D} + \bar{A}B\bar{C}D + \bar{A}CD = \overline{\overline{(\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{D})}.\overline{(\bar{A}B\bar{C}D)}.\overline{(\bar{A}CD)}}$$

$$d = A\bar{B}\bar{C} + \bar{A}C\bar{D} + \bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C + \bar{A}B\bar{C}D = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}C\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}.\overline{(\bar{A}\bar{B}C)}.\overline{(\bar{A}B\bar{C}D)}}$$

$$e = \bar{A}C\bar{D} + \bar{B}\bar{C}\bar{D} = \overline{\overline{(\bar{A}C\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}}$$

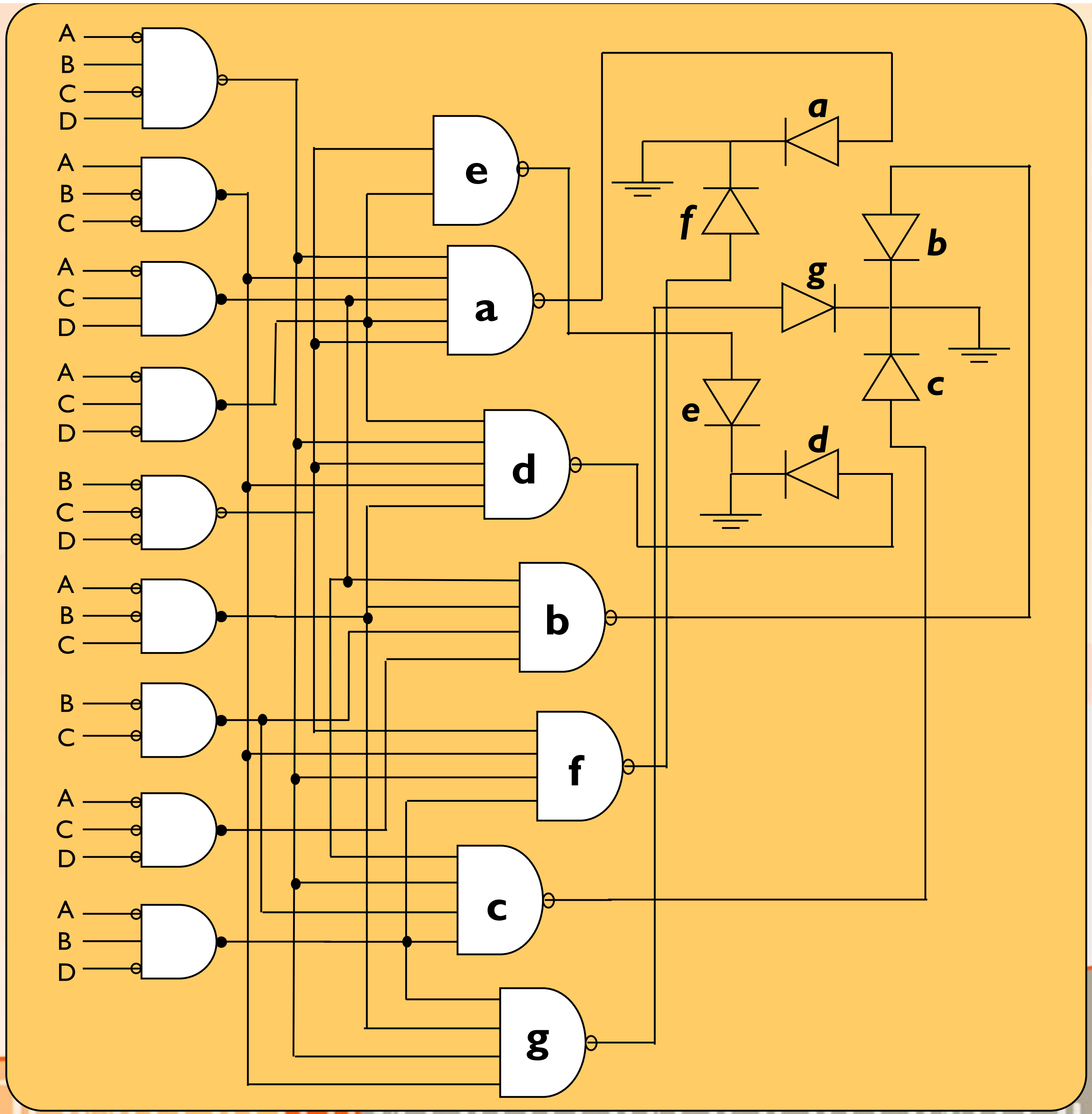
$$f = A\bar{B}\bar{C} + \bar{A}B\bar{D} + \bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}.\overline{(\bar{A}B\bar{C}D)}}$$

$$g = A\bar{B}\bar{C} + \bar{A}B\bar{D} + \bar{A}\bar{B}C + \bar{A}B\bar{C}D = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{D})}.\overline{(\bar{A}\bar{B}C)}.\overline{(\bar{A}B\bar{C}D)}}$$

# BCD to 7-segment Decoder

- This yields a 16 NAND gate solution.
- Had we minimised individually, we would have obtained a $\boxed{\text{22 NAND}}$ gate solution which would require an additional two NAND ICs.
- Thus, the final decoder circuit looks like:

# BCD to 7-segment Decoder



# BCD to 7-segment Decoder

- Note – we could have made the minimisation a little easier, if given the choice, by using don't care terms for the unused states instead.

- Finally, the standard 8421 BCD to 7-segment decoder (or driver) comes in chip format – namely the 5V CMOS 4511 IC.

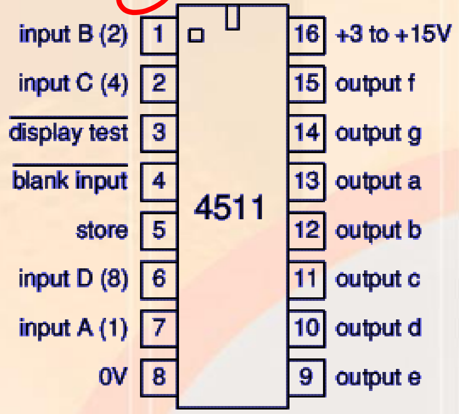
Class Test 2 →

$\boxed{\text{23-Dec 2021}}$

$\boxed{\text{7pm - 8:40pm}}$

CSSD MMWD

EE R125



input B (2)
input C (4)
display test
blank input
store
input D (8)
input A (1)
0V
1
2
3
4
5
6
7
8
4511
16
15
14
13
12
11
10
9
+3 to +15V
output f
output g
output a
output b
output c
output d
output e