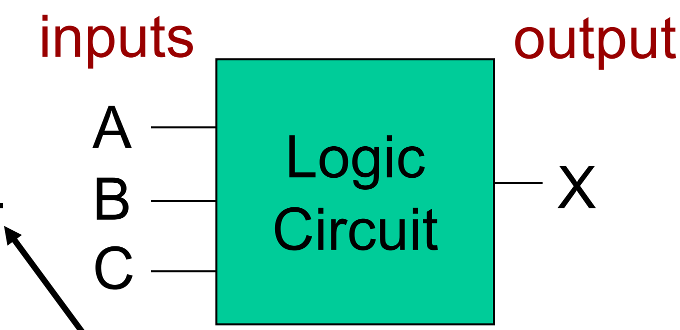
EBU4202: Digital Circuit Design
Karnaugh Maps

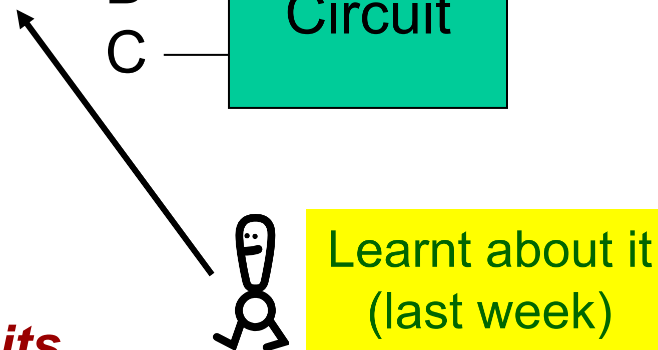
Dr. Md Hasanuzzaman Sagor (Hasan)
Dr. Chao Shu (Chao)
Dr. Farha Lakhani (Farha)

School of Electronic Engineering and Computer Science,
Queen Mary University of London,
London, United Kingdom.

Recap: Logic Circuits – Analysis & Synthesis

- **Digital Circuits**:
  - **Combinational Logic Circuit**: a circuit whose outputs depend on its current inputs.
  - **Sequential Logic Circuit**: a circuit whose outputs depend not only on current inputs, but also on past inputs.



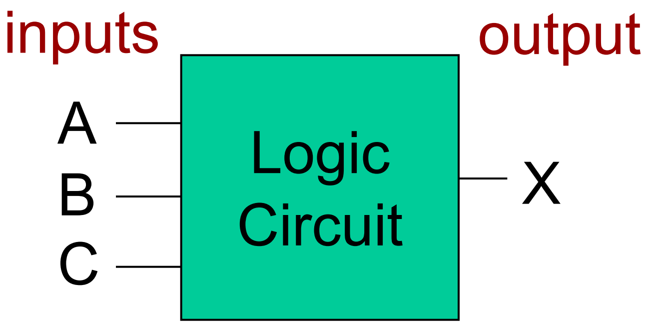


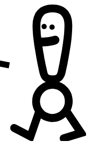
- These slides are concerned with the ***analysis*** and ***synthesis*** of ***combinational*** logical ***circuits***.
  - **Analysis** → start from a logic diagram of a circuit and derive a formal description of the function of the circuit.
  - **Synthesis** → start with a formal description of the function of a circuit and proceed to a logic diagram that performs the required function.

Queen Mary University of London

Recap: Logic Circuits – Analysis & Synthesis

- **Digital Circuits**:
  - **Combinational Logic Circuit**: a circuit whose outputs depend on its current inputs.
  - **Sequential Logic Circuit**: a circuit whose outputs depend not only on current inputs, but also on past inputs.





Studied later in the course.

- These slides are concerned with the ***analysis*** and ***synthesis*** of ***combinational*** logical ***circuits***.
  - **Analysis** → start from a logic diagram of a circuit and derive a formal description of the function of the circuit.
  - **Synthesis** → start with a formal description of the function of a circuit and proceed to a logic diagram that performs the required function.

Overview: Switching Algebra & Combinational Logic Design

- **Switching Algebra**
- **Combinational Circuit Analysis & Synthesis**

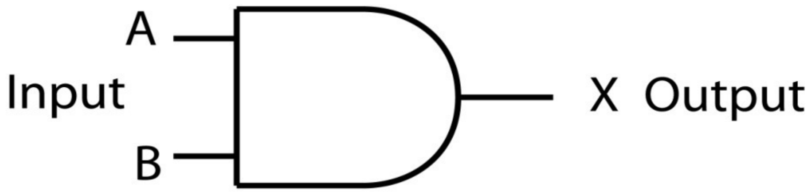
Chapters 4 & 6 – "Digital Design: Principles and Practices" book

Combinational Circuit Analysis

- ***Analysis of combinational circuits***:
  - It requires a formal description of their logic function.
  - *Logic function description* allows:
    - *Determination* of circuit behaviour for different input combinations.
    - *Manipulation* of an algebraic description to derive different circuit structures for the logic function.
    - *Transformation* of an algebraic description into a form corresponding to an available circuit structure.

Karnaugh Maps

- *Karnaugh Map*:
  - Another approach to represent (and simplify) Boolean equations.
  - **Example** (Karnaugh map for a 2-input AND gate):



Input		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

A \ B	0	1
0	0	0
1	0	1

Karnaugh Map

Karnaugh Map Simplification

- **Why use Karnaugh map simplification**:
  - Working with algebraic equations is often tedious and error-prone.
  - A mapping technique can be used to reduce an algebraic equation to its simplest form.
  - Simplification approaches used (consisting of grouping adjacent *1s* or *0s* in *powers of 2*) are *minterm* or *maxterm*.
  - Equations of up to 5 variables can be easily simplified by hand. However, it becomes complicated to simplify equations with more than 5 variables.

2-Variable Map (1/2)

- How to **build a 2-variable Karnaugh map**:
  - There will be 4 *minterms* for a 2-variable equation, so use a Karnaugh map with 4 squares (i.e., a 2x2 table).
  - *1's* and *0's* on the left side and top of the map designate the values of the variable.
  - Variable *X* is ***complemented*** in row *0* and ***uncomplemented*** in row *1*.
  - Variable *Y* appears ***complemented*** in column *0* and ***uncomplemented*** in column *1*.

X \ Y	0	1
0	m0	m1
1	m2	m3

2-Variable Map (2/2)

- How **a 2-variable Karnaugh map works**:
  - The concept is to put a *1* in each square that has a corresponding *minterm* in the Boolean equation.
  - Only four terms are possible in *Sum of Products* form.

X \ Y	0	1
0	m0	m1
1	m2	m3

Equates to

X \ Y	0	1
0	$\overline{X}\bullet\overline{Y}$	$\overline{X}\bullet Y$
1	$X\bullet\overline{Y}$	$X\bullet Y$

Example: 2-Variable Map

- Simplify the boolean equation:

  $$F = X' \bullet Y + X \bullet Y' + X \bullet Y$$

  - Function ***F*** is in *Sum of Products* form.
  - Put *1's* in boxes of corresponding terms and put *0's* in all other boxes.

X \ Y	0	1
0	0	1
1	1	1

- Simplified expression for ***F*** is obtained by **grouping adjacent 1's** (in powers of 2) and eliminating unnecessary variables.
- Here, **there are two ways of grouping *1's***: one way is to circle the row where *X=1*; the other way is to circle the column where *Y=1*.
- ***Y*** is eliminated from the two product terms where *X=1* and ***X*** is eliminated from the two product terms where *Y=1*.

Answer**: F = X + Y.**

Golder Rules for Grouping

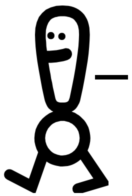
Remember to circle the **largest groupings possible!**

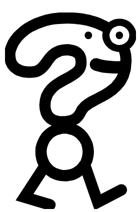
Golden Rules:

1. No zeroes allowed.
2. No diagonal groupings
3. Only power of 2 cells in each grouping
4. Every 1 must be at least in one grouping
5. Overlapping is allowed
6. Groups may “Wrap-around”
7. Fewest number of groups possible.

3-Variable Map (1/2)

- How to **build a 3-variable Karnaugh map**:
  - There will be 8 *minterms* for a 3-variable equation, so use a Karnaugh map with 8 squares (usually, a 2x4 table).
  - The *minterm* numbers do not follow the normal binary counting order sequence.
  - Only 1 bit changes from one adjacent column to the next.





What does this remind you of?

X \ YZ	00	01	11	10
0	m0	m1	m3	m2
1 (X)	m4	m5	m7	m6

Y: columns 11, 10. Z: columns 01, 11.

3-Variable Map (2/2)

- How **a 3-variable Karnaugh map works**:

X \ YZ	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

Equates to

X \ YZ	00	01	11	10
0	$\overline{X}\bullet\overline{Y}\bullet\overline{Z}$	$\overline{X}\bullet\overline{Y}\bullet Z$	$\overline{X}\bullet Y\bullet Z$	$\overline{X}\bullet Y\bullet\overline{Z}$
1	$X\bullet\overline{Y}\bullet\overline{Z}$	$X\bullet\overline{Y}\bullet Z$	$X\bullet Y\bullet Z$	$X\bullet Y\bullet\overline{Z}$

Example 1: 3-Variable Map

Simplify: F(X, Y, Z) = Σm(1, 2, 5, 6) **= 001, 010, 101, 110**

or

$= \overline{X}\,\overline{Y}Z \quad \overline{X}Y\overline{Z} \quad X\overline{Y}Z \quad XY\overline{Z}$

Note: Circle *1's* in horizontal or vertical groups of 1, 2, 4 or 8 only (*powers of 2*)!

m1 + m5 $= \overline{X}\bullet\overline{Y}\bullet Z + X\bullet\overline{Y}\bullet Z$

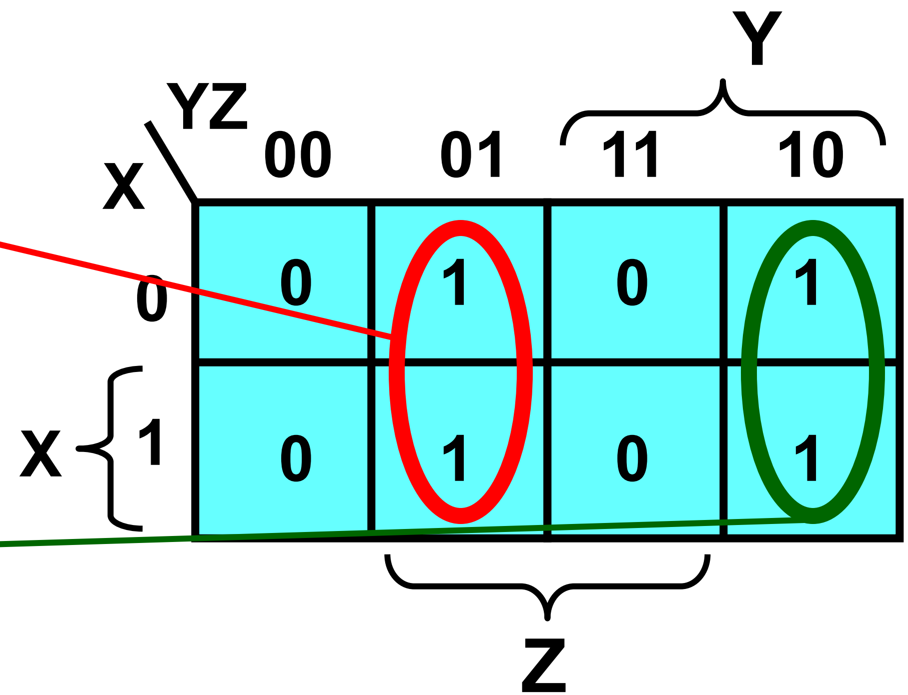
$= (\overline{X} + X)(\overline{Y}\bullet Z)$

$\therefore$ X is redundant

m2 + m6 $= \overline{X}\bullet Y\bullet \overline{Z} + X\bullet Y\bullet \overline{Z}$

$= (\overline{X} + X)(Y\bullet \overline{Z})$

$\therefore$ X is redundant



ANSWER: F = Y'Z + YZ'

Example 2: 3-Variable Map

Simplify, using a Karnaugh map:

$F(X, Y, Z) = \Sigma m(3, 4, 5, 6, 7)$

X \ YZ	00	01	11	10
0				
1				

Check using Boolean Algebra

$$
\begin{aligned}
F &= X'YZ + XY'Z' + XY'Z + XYZ' + XYZ \\
&= XZ(Y' + Y) + XZ'(Y' + Y) + X'YZ \\
&= XZ + XZ' + X'YZ \\
&= X(Z + Z') + X'YZ \\
&= X + X'YZ \\
&= X + YZ \quad [\text{since } X + X'Y = X + Y]
\end{aligned}
$$

To be completed in class …

Example

F(X, Y, Z) = Σm(3, 4, 5, 6, 7)

YZ

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

X

Check using Boolean Algebra

F = X'YZ + XY'Z' + XY'Z + XYZ' + XYZ
 = XZ(Y' + Y) + XZ'(Y' + Y) + X'YZ
 = XZ + XZ' + X'YZ
 = X(Z + Z') + X'YZ
 = X + X'YZ
 = X + YZ [since X + X'Y = X + Y]

4-Variable Map

WX \ YZ	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

Example 1: 4-Variable Map

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

Y (columns 11, 10); X (rows 01, 11); W (rows 11, 10); Z (columns 01, 11)

ANSWER: $F = X\overline{Y} + \overline{Y}Z + Y\overline{Z} + WX$

Truth Table

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

Example 2: 4-Variable Map

- This is an example of a Karnaugh map with all possible groupings (except circles with a single value):

$$F = W'Y' + WXZ + WX'Y.$$

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

Prime Implicants

- **Challenge when using K-maps**: To select the right groups.
  - **IF** the number of groups is not minimised **AND**
  - the size of each group is not maximised **THEN**,
    - Resulting expression will still be equivalent to the original one. **BUT**
    - Resulting expression will not be a ***minimal*** sum of products (or MSP).

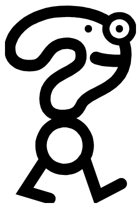
- Good **approach to finding an actual MSP**:
  - Find all of the largest possible groupings of *1's* (these are called the ***prime implicants***); but only in <u>powers of 2</u>!
  - The final MSP will contain a subset of these prime implicants.

Example: K-Map with Prime Implicants

- Here is an example of a Karnaugh map with *prime implicants* (all of the marked groups are prime implicants):

almost

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



Why is this not a prime implicant?

Additional: K-Map with Prime Implicants

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	1
10	1	0	1	1

K-Map with Prime Implicants

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	1
10	1	0	1	1

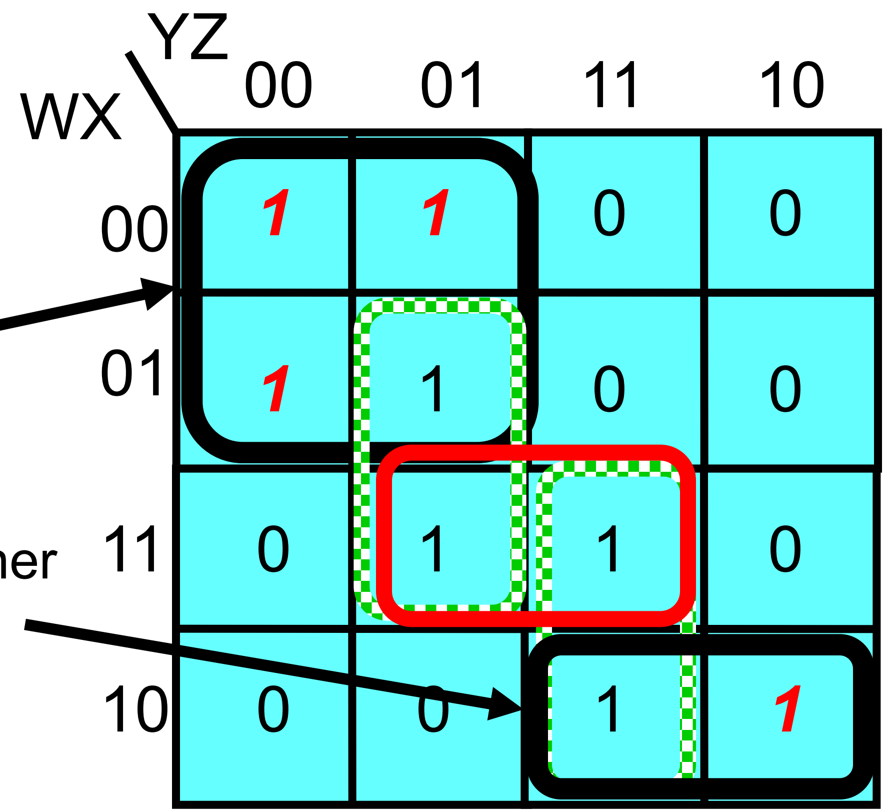
Good approach: Use the fewest possible number of maximal groupings needed to cover all of the squares marked with a 1.

Essential Prime Implicants (EPIs)

- If any group contains a ***minterm*** that isn't also covered by another overlapping group, then that is an ***EPI***.
- ***EPIs*** appear in the MSP, since they contain ***minterms*** that no other groups include.
- **Example**:

EPI: because no other group covers *m0*, *m1*, and *m4*.

EPI: because no other group covers *m10.*



Queen Mary
University of London

Another Example : Covering the other *Minterms*

- Example from previous slide:
  - Pick as few other prime implicants as necessary, to ensure that all the *minterms* are covered.
  - After choosing the **green** rectangles in the example, there are just two *minterms* left to be covered: *m13* and *m15*.
  - These are both included in the **red** prime implicant, WXZ.
  - The resulting equation is,

$$\boxed{F = W'Y' + WXZ + WX'Y}$$

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

Additional: Essential Prime Implicants

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	0	1	1
10	0	0	1	0

Additional: Essential Prime Implicants

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	0	1	1
10	0	0	1	0

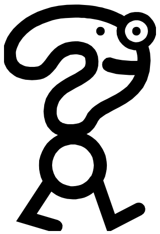
Relationships: F, F′, Σ, Π

Karnaugh Map for F		Karnaugh Map for F′	
Minimal Sum of Products for F	**Minimal Product of Sums for F**	**Minimal Sum of Products for F′**	**Minimal Product of Sums for F′**
What to do: Loop *1's* and read as *minterms*.	***What to do***: Loop *0's* and read as *maxterms* (or complement SOP for *F*).	***What to do***: Loop *1's*.	***What to do***: Loop *0's* and read as *maxterms* (or complement SOP for *F'*).

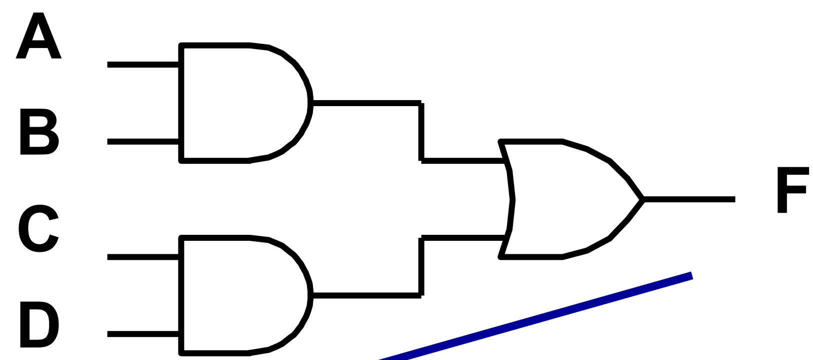
Queen Mary
University of London

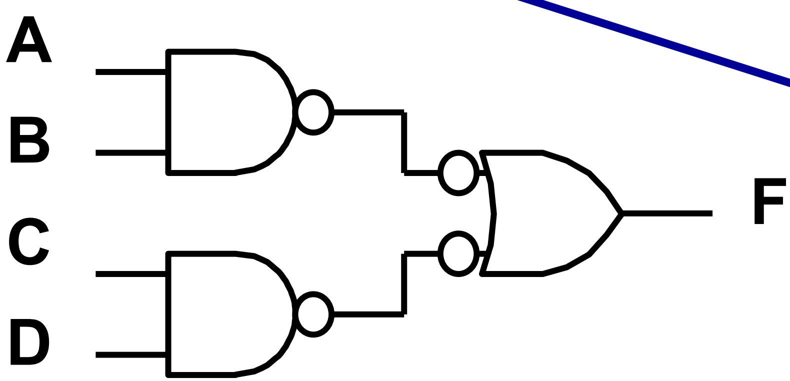
Equivalent Circuits

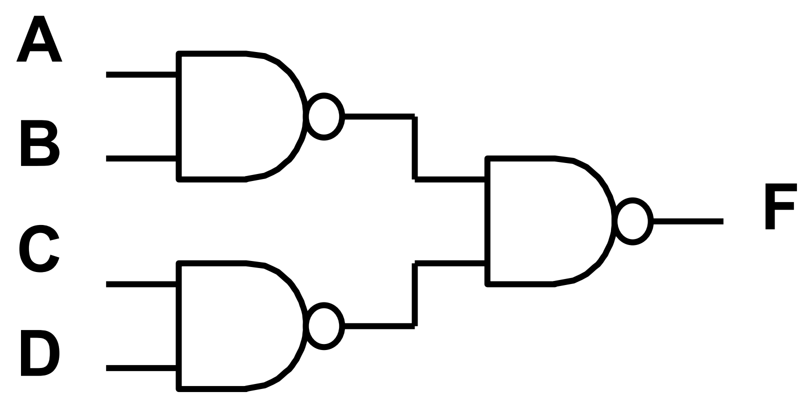
- Three different (but *equivalent*) circuits that implement the equation **F = AB + CD**:



What are the equations describing the two bottom diagrams?







Example: SOP of F (1/2)

- Find the *minimal sum of products* and draw the *NAND gate* implementation for:

  $F = X'Y'Z + X'YZ' + X'YZ + XY'Z' + XY'Z$

- ***What to do***:
  - ✓ First look for the largest possible group.
  - ✓ If there are still *1's* to cover, keep circling until all are covered.
  - ✓ Read off essential prime implicants necessary for minimal cover.

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

$F = XY' + X'Y + Y'Z$

$F = F'' = (XY' + X'Y + Y'Z)''$

$F = ((XY')'(X'Y)'(Y'Z)')'$

Example: SOP of F (2/2) *To be completed in class …*

- Draw the *NAND gate* implementation of: F = XY′ + X′Y + Y′Z

  ✓ 1. Draw the normal circuit:

  ✓ 2. Apply De Morgan’s theorems graphically:

  ✓ 3. Convert the 3-inverted input gate to *NAND*:

Queen Mary
University of London

Example: SOP of F′

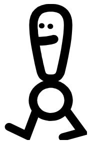
To be completed in class …

- Find the *minimal sum of products* for F′ where,

  F = X′Y′Z + X′YZ′ + X′YZ + XY′Z′ + XY′Z

  ∴ F′ = (X+Y+Z′)(X+Y′+Z)(X+Y′+Z′)(X′+Y+Z)(X′+Y+Z′)

- ***What to do***:
  ✓ Map F′. ✓ Circle the *1's*.
  ✓ Read off essential prime implicants as *minterms*.

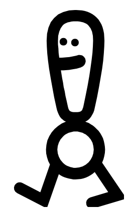


F is as in previous example.

X \ YZ	00	01	11	10
0				
1				

Summary: Karnaugh Maps (1/2)

- Karnaugh maps are an *alternative* to Switching Algebra for *simplifying expressions*:
  - The result is an MSP (or MPS) that leads to a minimal 2-level circuit.
  - It is easy to handle ***don't-care conditions***.
  - Karnaugh maps are really only good for manual simplification of fairly small expressions.



We'll look at these **in a few slides**.

Queen Mary
University of London

Summary: Karnaugh Maps (2/2)

- Things to keep in mind:
  - Remember the correct order of *minterms* on the K-map.
  - When grouping, it is possible to wrap around all sides of the K- map, and the groups can overlap.
  - Make as few groups (of adjacent *1s* or *0s*) as possible, but make each of them as large as possible (*only in powers of 2* i.e., *1*, *2*, *4*, *8*, …).
  - There may be more than one valid solution!!

Map Manipulation

- Often a truth table will be generated that has several combinations that the designer doesn't care about.
- **Expressions can be simplified by using DON'T CARE cases**:
  - Sometimes it is possible to eliminate essential prime implicants.
- **Example**:
  - Consider the following *incompletely specified function*:

$$\begin{cases} F(W,X,Y,Z) = \Sigma m(1, 3, 7, 11, 15) \\ d(W,X,Y,Z) = \Sigma m(0, 2, 5) \end{cases}$$

$\rightarrow$ These are DON'T CARE cases.

4-Variable Karnaugh Map

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

$F(W,X,Y,Z) = \Sigma m(1, 3, 7, 11, 15)$

$d(W,X,Y,Z) = \Sigma m(0, 2, 5)$

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

Answer:

F = YZ + W'X'

4-Variable Karnaugh Map

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

$F(W,X,Y,Z) = \Sigma m(1, 3, 7, 11, 15)$
$d(W,X,Y,Z) = \Sigma m(0, 2, 5)$

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

Alternative Answer:

F =

Recap: Binary Coded Decimal (BCD)

- BCD uses a *4-bit* pattern to express each digit of a *base 10* number.
- How each digit in BCD is encoded:
  - *in BCD*
    - 123 : $\underbrace{0001}_{1}\ \underbrace{0010}_{2}\ \underbrace{0011}_{3}$
    - +123 : 1010 0001 0010 0011
    - -123 : 1011 0001 0010 0011
  - *in simple Binary*
    - 123 : 111 1011

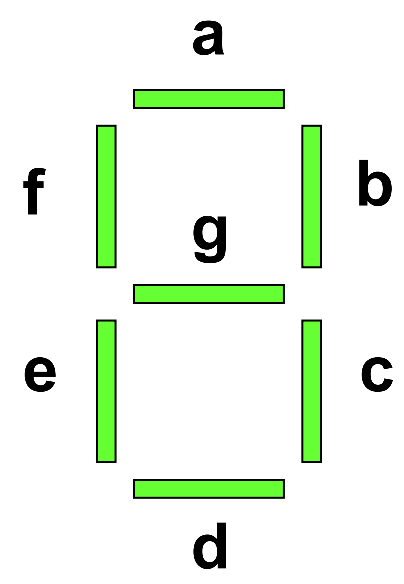
Some implementations only; usually, the last 6 values are not used.

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
+	1010
−	1011

Queen Mary
University of London

Example (1/9): Design Procedure

- **Problem Specification**:
  - *Design a code converter* to decode from BCD to a 7-segment display.
  - The combinational circuit must accept a BCD digit *and* generate the appropriate outputs to display the digit in a 7-segment format as shown here.

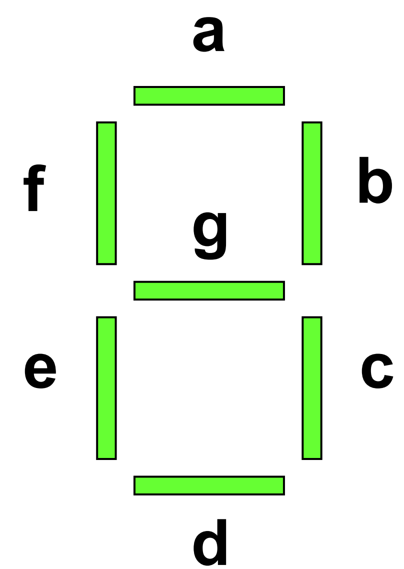


Example (2/9): BCD and I/O

- *BCD numbers are 4 digits long*, so
  - e.g., $1_{10} = 0001_2$ and $10_{10} = 00010000_2$.

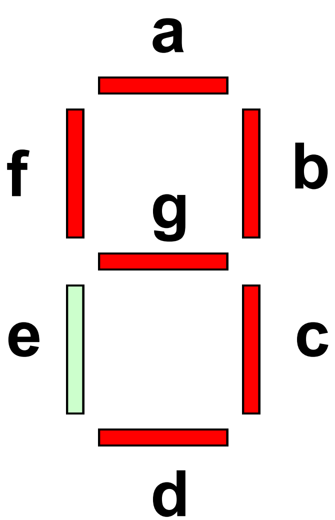
- **Inputs**:
  - We only have to output decimal digits *0*-*9* on the 7-segment display, so only 4 inputs are needed!

- **Outputs**:
  - Need one for each segment *a*-*g*.
  - Output for each function *a*-*g* should be:
    - *1* if that segment should be *on*.
    - *0* if it should be *off*.



Queen Mary
University of London

Example (3/9): BCD 7-Segment Truth Table



	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	0	0	0	0	0	0	0
	1	0	1	1	0	0	0	0	0	0	0
	1	1	0	0	0	0	0	0	0	0	0
	1	1	0	1	0	0	0	0	0	0	0
	1	1	1	0	0	0	0	0	0	0	0
	1	1	1	1	0	0	0	0	0	0	0

Example (4/9): Karnaugh Map for 'a'

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$a = B'C'D' + AB'C' + A'BD + A'C$

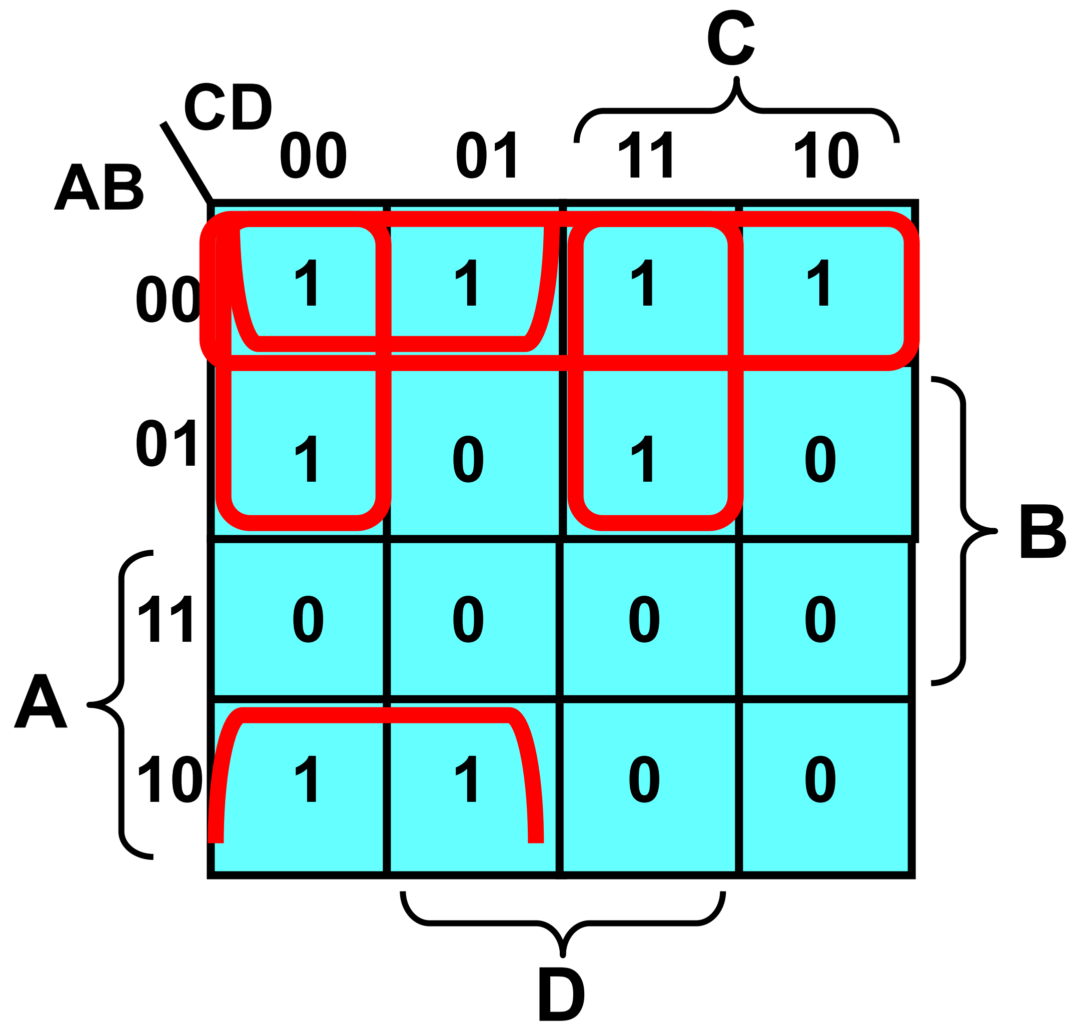
AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

C, B, A, D

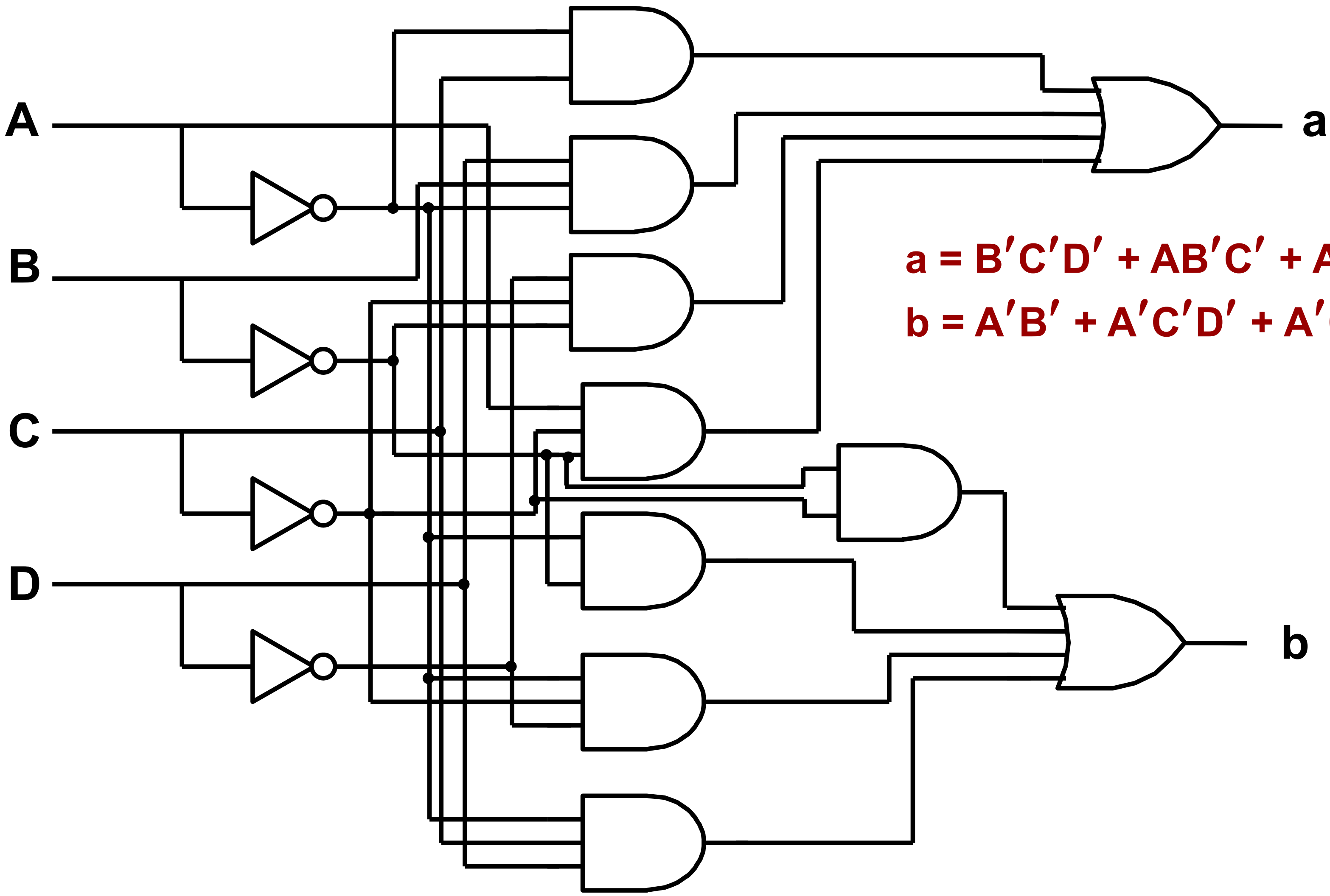
Example (5/9): Karnaugh Map for 'b'

$b = A'B' + A'C'D' + A'CD + B'C'$

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



Example (6/9): BCD 7-Segment Decoder (for *a* & *b*)



$a = B'C'D' + AB'C' + A'BD + A'C$

$b = A'B' + A'C'D' + A'CD + B'C'$

Example (7/9): Karnaugh Map for 'c'

c =

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

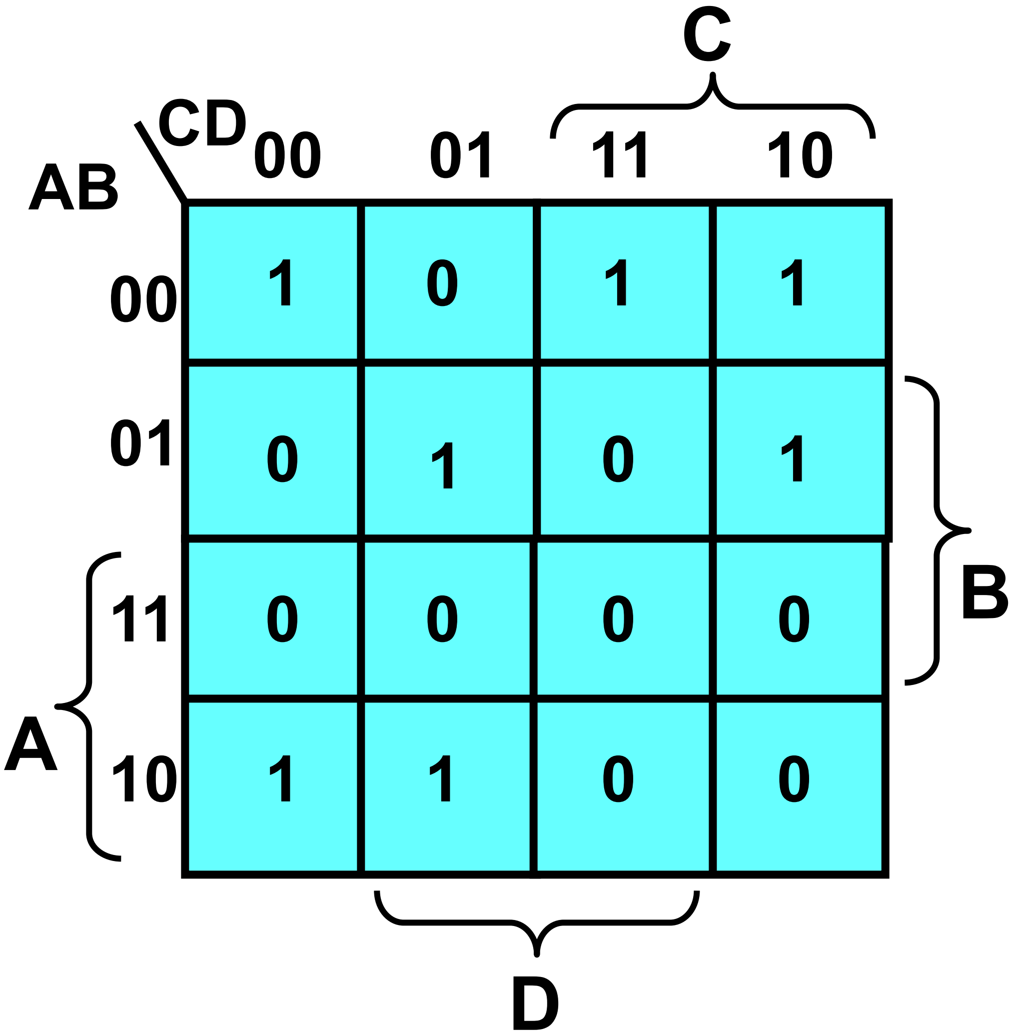
C

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

B

A

Example (8/9): Karnaugh Map for ‘d’



d =

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Example (9/9): BCD 7-Segment Decoder (for *c* & *d*)

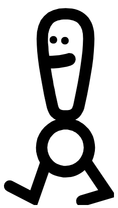
c =

d =

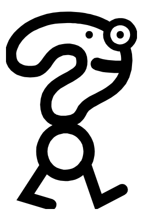
To be completed in class …

Example (1/2): Parity Generators

- **Problem Specification**: **Build a network** that will generate the appropriate even parity bit for the 3-bit input.
  - *Inputs*: X = Bit1; Y = Bit2; Z = Bit3
  - *Outputs*: F = parity bit



Remember: In a 4-bit number, there will be *even parity*, if there is an even number of *1's*.



What is the requirement for having *odd parity*?

Example (2/2):
3-Bit Even Parity Generator

To be completed in class …

Truth Table

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

X \ YZ	00	01	11	10
0				
1				

Example (2/2):
3-Bit Even Parity Generator

To be completed in class …

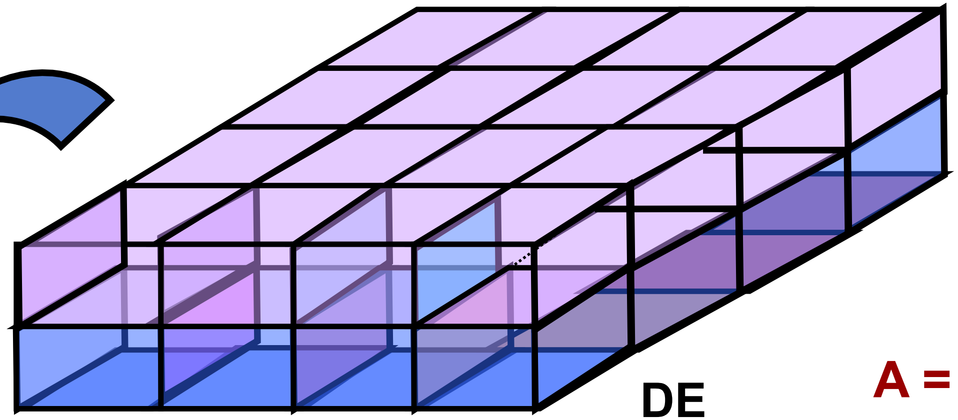
Truth Table

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

$\rightarrow$

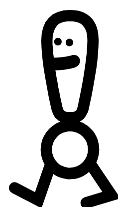
X \ YZ	00	01	11	10
0	0	1	0	1
1	1	0	1	0

5-Variable Karnaugh Map



A = 0

BC \ DE	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10



Each square in *A=0* is also adjacent to squares in *A=1*.

A = 1

BC \ DE	00	01	11	10
00	m16	m17	m19	m18
01	m20	m21	m23	m22
11	m28	m29	m31	m30
10	m24	m25	m27	m26

Example (1/2): 5-Variable Karnaugh Map

F(A,B,C,D,E) = Σm(0, 10, 11, 14, 15, 16, 20, 24, 26, 27, 28, 30, 31)

= 00000, 01010, 01011, 01110, 01111, 10000, 10100
11000, 11010, 11011, 11100, 11110, 11111

A = 0

BC \ DE	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	1	1
10	0	0	1	1

A = 1

BC \ DE	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	0	1	1
10	1	0	1	1

Example (2/2): 5-Variable Karnaugh Map

Next step: Grouping ... Start with largest possible – in this map it's *32*, down to *1*. Don't circle smaller groups if they don't include uncovered *1's*!

Cross-map (i.e., A=0 and A=1) so no A; but every other variable is the same, so B'C'D'E'.

Only in map *A=1*; *B* and *C* change, so AD'E'.

A = 0

BC \ DE	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	1	1
10	0	0	1	1

Remember the cross-map adjacency!

A = 1

BC \ DE	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	0	1	1
10	1	0	1	1

F = B′C′D′E′ + AD′E′ + BD

Cross-map (i.e., *A=0* and *A=1*) so no *A*; in both maps *C* and *E* change, so *BD*.

General Procedure: Designing Combinational Circuits

1. Specification: Write specification for the circuit if not already available.
   - Specify/label input(s) and output(s).
2. Formulation: Derive *Truth Table* or *initial Boolean equations* defining the relationships between inputs and outputs, if not in the specification.
3. Optimisation: Minimise the design using *Switching Algebra*, *Karnaugh Map*, *software*.
   - Draw logic diagram for the resulting circuit using AND/ OR/ NOT gates.
4. Technology Mapping: Map the logic diagram to the implementation technology selected (e.g., map into NANDs).
5. Verification: Verify the correctness of the final design *manually* or *using simulation*.

Practical Considerations:
- Cost of gates (Number)
- Maximum allowed delay
- Fan-in/Fan-out

Example: Circuit Design (1/2)

- *Question*: Design a circuit that has a 3-bit input and a single output (F) specified as follows:
  - F = 0, when the input is less than $(5)_{10}$
  - F = 1, otherwise

<u>Step 1: Specification</u>

- Label the inputs (3 bits) as X, Y, Z: *X* is the <u>most significant bit</u>, *Z* is the <u>least significant bit</u>.
- The output (1 bit) is F:
  - F = 1 → $(101)_2$, $(110)_2$, $(111)_2$
  - F = 0 → other inputs

Example: Circuit Design (2/2)

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

Step 2: Formulation

- Obtain the Truth Table.

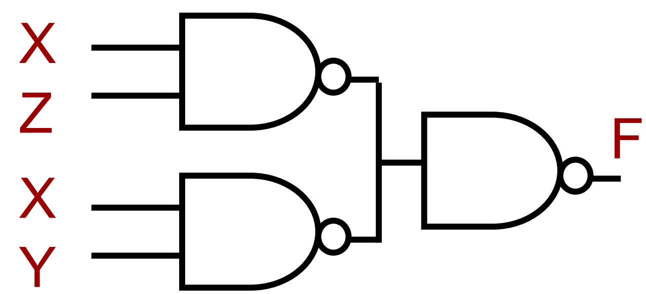
Step 3: Optimisation

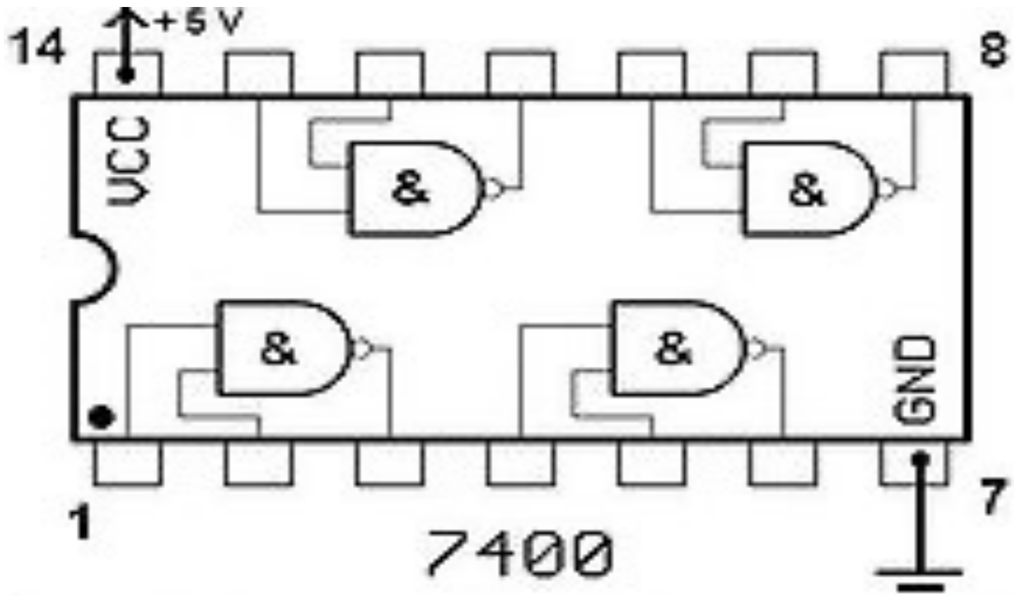
X \ YZ	00	01	11	10
0	*0*	*0*	*0*	*0*
1	*0*	*1*	*1*	*1*

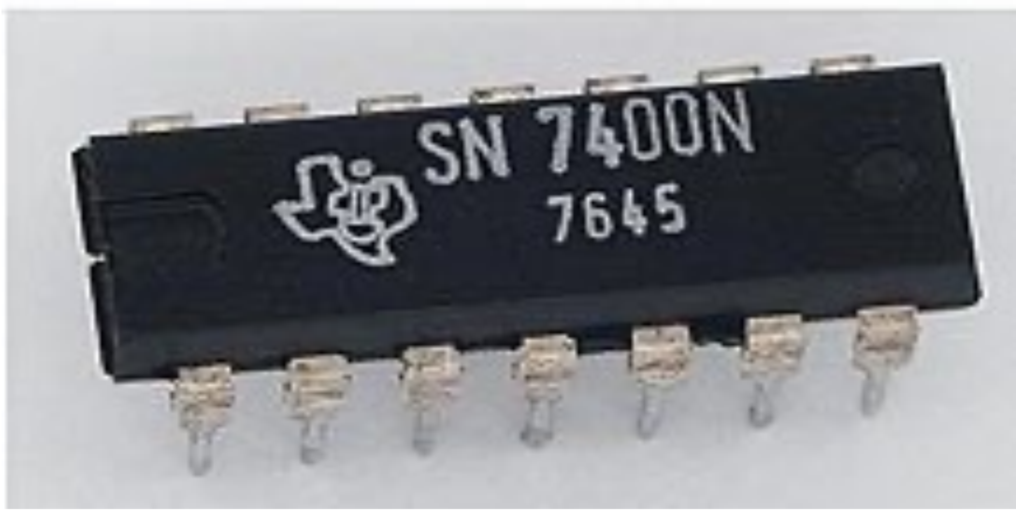
F = XZ + XY

Step 4: Technology mapping *(e.g., NAND implementation)*

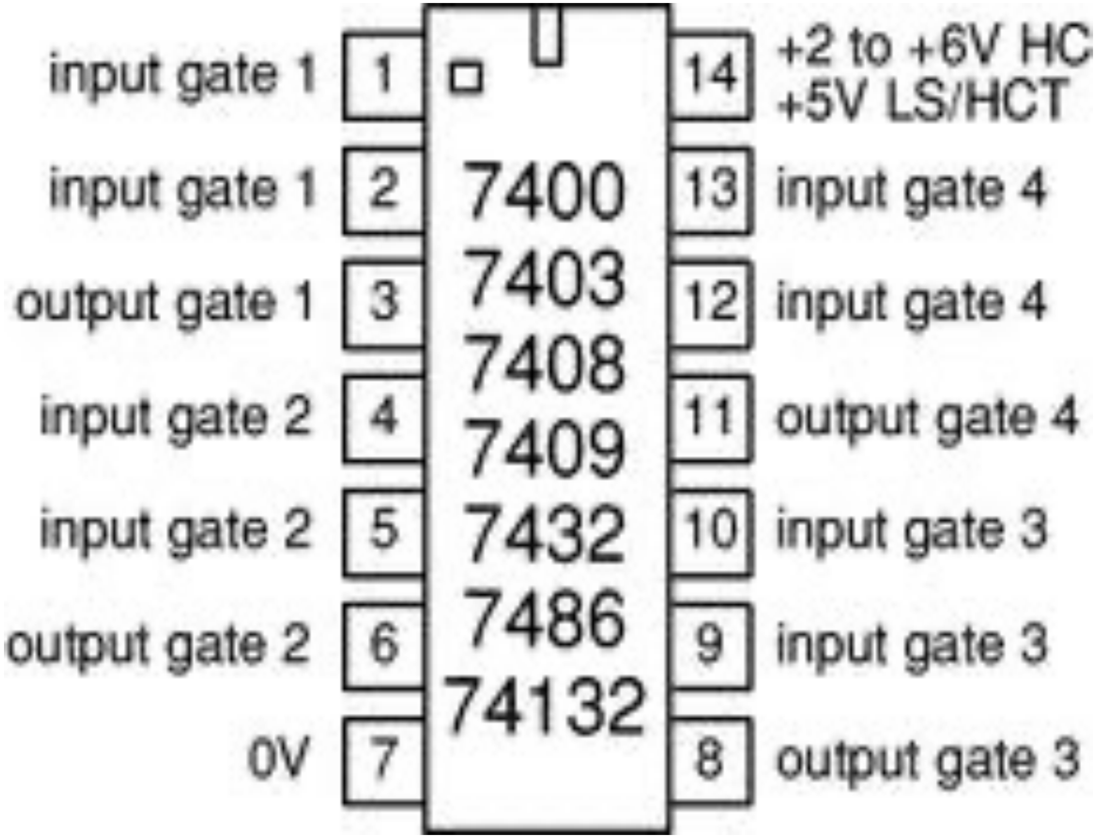
- F = F'' = (XZ + XY)'' = ((XZ)'.(XY)')',
  applying *De Morgan's theorem*





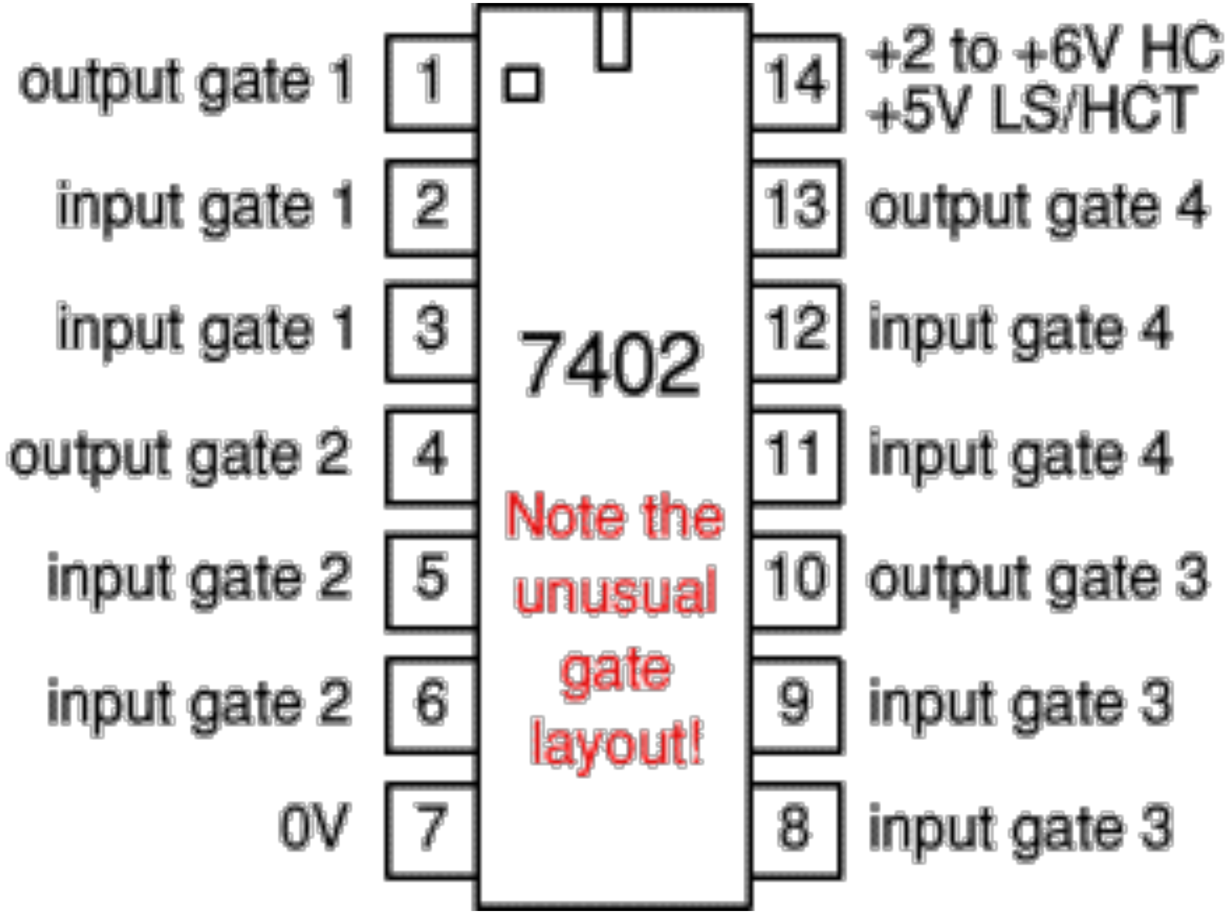


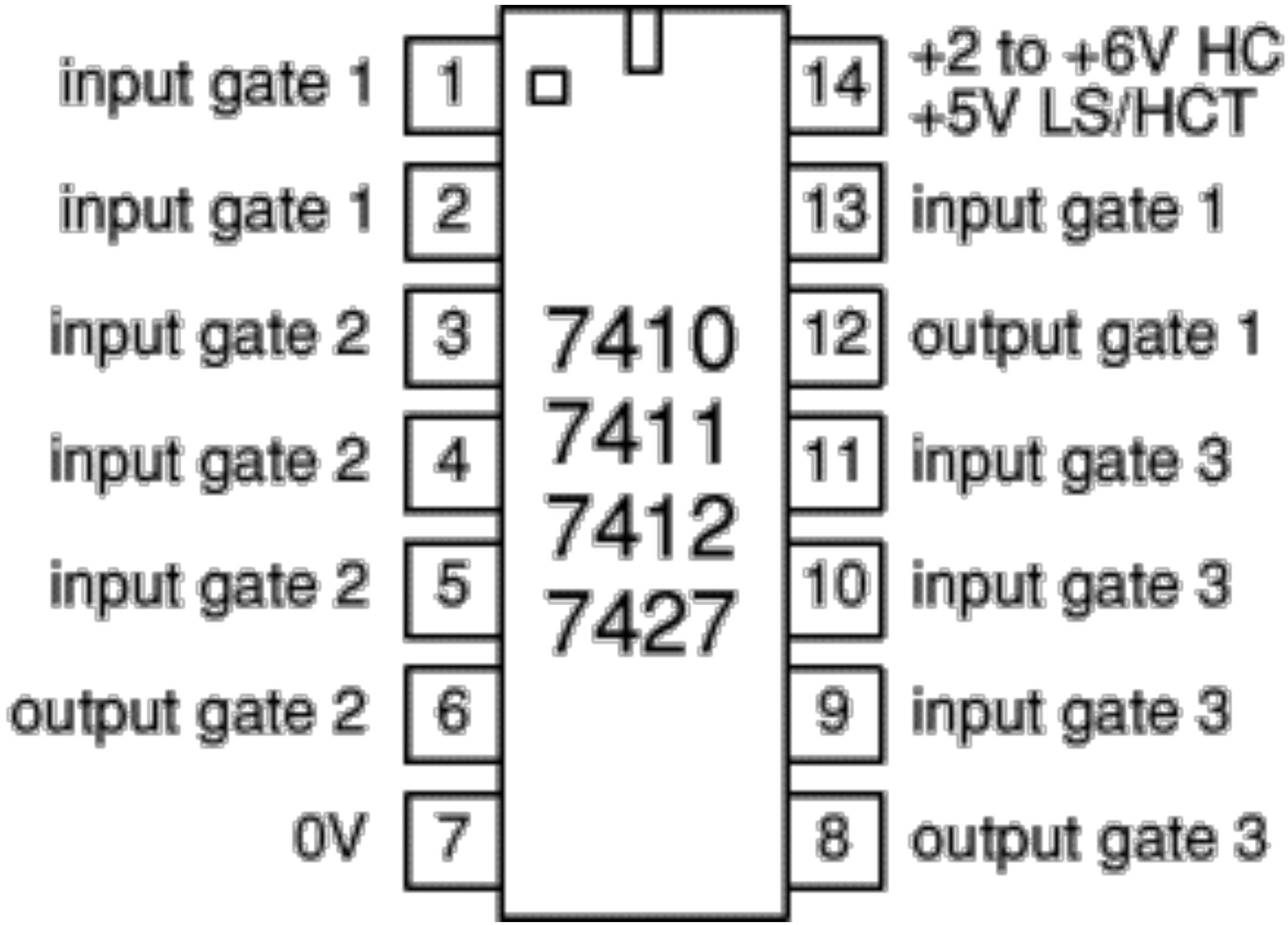
Queen Mary
University of London


input gate 1
input gate 1
output gate 1
input gate 2
input gate 2
output gate 2
0V
1
2
3
4
5
6
7
7400
7403
7408
7409
7432
7486
74132
14
13
12
11
10
9
8
+2 to +6V HC
+5V LS/HCT
input gate 4
input gate 4
output gate 4
input gate 3
input gate 3
output gate 3

Quad 2-input gates

7400 quad 2-input NAND
7402 quad 2-input NOR
7403 quad 2-input NAND with open collector outputs
7408 quad 2-input AND
7409 quad 2-input AND with open collector outputs
7432 quad 2-input OR
7486 quad 2-input EX-OR
74132 quad 2-input NAND with Schmitt trigger inputs


output gate 1
input gate 1
input gate 1
output gate 2
input gate 2
input gate 2
0V
1
2
3
4
5
6
7
7402
Note the unusual gate layout!
14
13
12
11
10
9
8
+2 to +6V HC
+5V LS/HCT
output gate 4
input gate 4
input gate 4
output gate 3
input gate 3
input gate 3


input gate 1
1
input gate 1
2
input gate 2
3
input gate 2
4
input gate 2
5
output gate 2
6
0V
7
7410
7411
7412
7427
14
+2 to +6V HC
+5V LS/HCT
13
input gate 1
12
output gate 1
11
input gate 3
10
input gate 3
9
input gate 3
8
output gate 3

Triple 3-input gates

7410 triple 3-input NAND

7411 triple 3-input AND

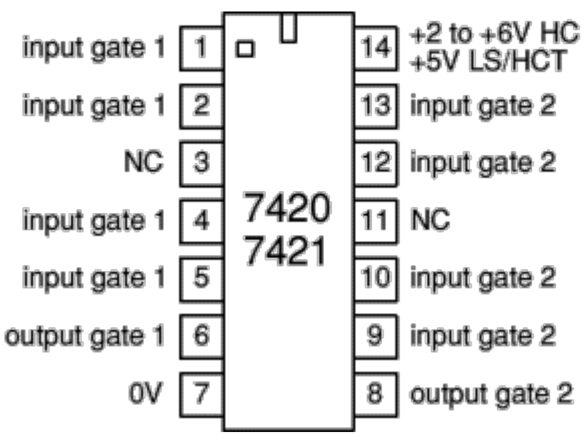
7412 triple 3-input NAND with open collector outputs

7427 triple 3-input NOR

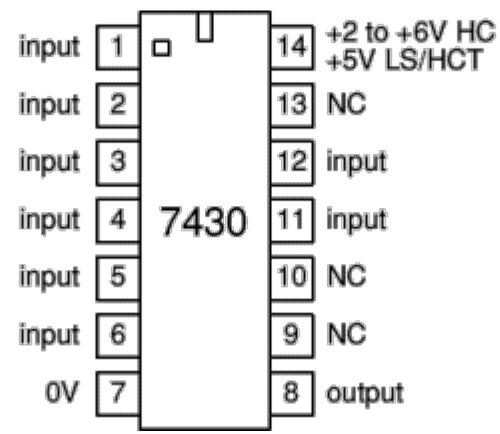
Dual 4-input gates

7420 dual 4-input NAND

7421 dual 4-input AND



7430 8-input NAND gate



Hex NOT gates

7404 hex NOT

7405 hex NOT with open collector outputs

7414 hex NOT with Schmitt trigger inputs