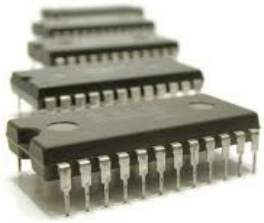
9. Programmable Logic Devices (PLDs)

9.1 Introduction



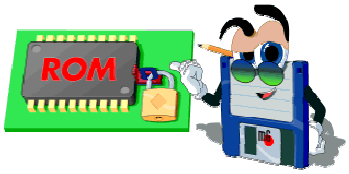
- So far in this module (and particularly in our laboratories), we have implemented digital circuits using **integrated circuits** (ICs).
- As we know, the IC (often referred to as a chip) contains the electronic components for the digital gates and storage elements. The various components are interconnected in the chip.
- The number of pins on a chip can vary from 14 on a small IC package (as we have used in our labs) to several hundred on a large package.
- Each IC has an alphanumeric identifier printed on its surface and typically has an associated datasheet containing all its relevant information (such as pin layout etc.). These datasheets are typically found on the manufacturer's website.
- Implementation technology using and connecting together ICs is regarded as **fixed** in the sense that the hardware functionality has been specified at the manufacturing stage. A CMOS 4081 (AND) IC, for example, will only implement the logic operation of an AND gate.

- An alternative implementation technology, known as programmable logic devices (PLDs) have undefined functions at the time of manufacture.
- They are fabricated with structures that can implement various logic functions and structures that are used to control connections or to store information specifying the actual logic functions implemented.
- Such devices require **programming (or reconfiguring)** post manufacturing. Programming relates to making the necessary hardware connections to implement relevant functions.
- These connections can consist of **fuses**, which offer a once-off solution, or, more commonly, they can consist of metal oxide semiconductor field effect transistors (MOSFETs).
- **MOSFETs offer an erasable solution**, i.e. can be programmed, erased and reprogrammed as needed. These will be studied in detail in later relevant modules.
- In this section of the notes, we are going to examine three types of basic programmable logic devices, namely the **Read Only Memory (ROM)**, the **Programmable Array Logic (PAL)** and the **Programmable Logic Array (PLA)**. More advanced programmable devices, such as field-programmable gate arrays (FPGAs), will be studied in later modules.
- ROM, PAL and PLA all have similar structures but differ in their programmability.
- As a result, they have slightly different requirements on the minimisation process needed to implement a set of logic functions. These will be outlined in the next sections.

9.2 Read Only Memory (ROM)

- Consider the implementation of the following multi-output functions:

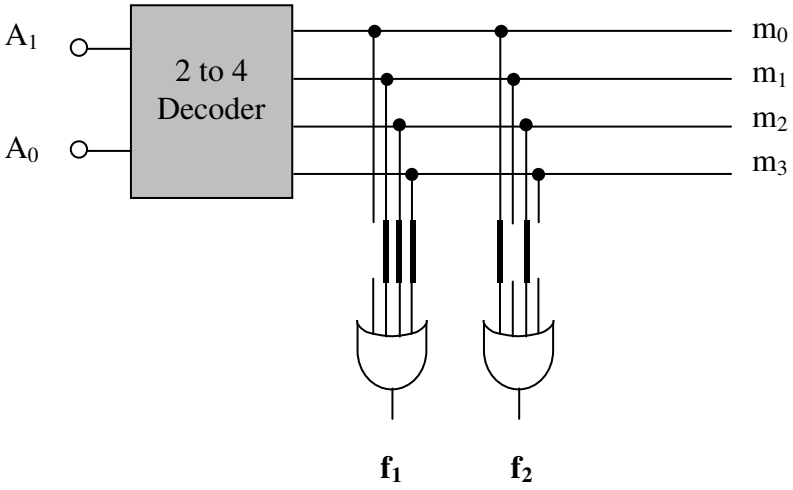
$f_{1(A1, A0)} = \sum(1, 2, 3)$
$f_{2(A1, A0)} = \sum(0, 2)$



- In tabular form we can express the functions as follows:

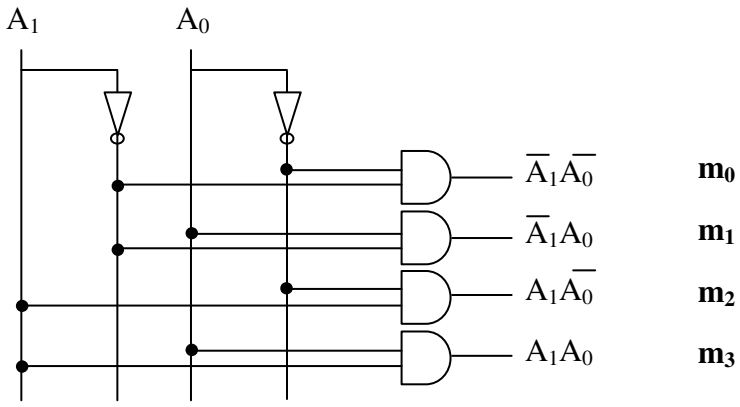
	A_1	A_0	f_1	f_2
m_0				
m_1				
m_2				
m_3				

- A simple conceptual circuit to implement these functions would be:



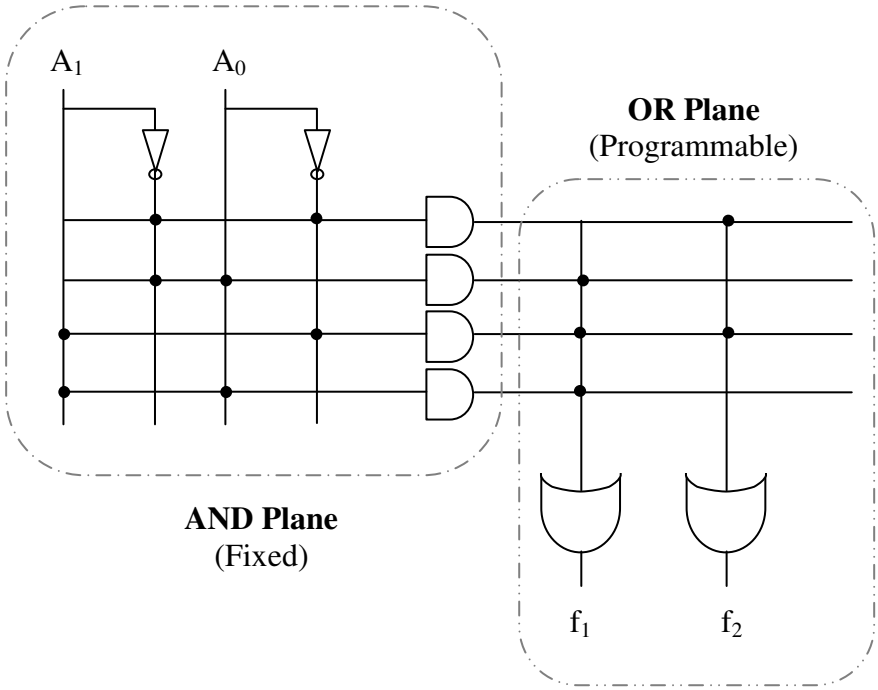
- The decoder is a standard circuit that produces all combinations of a given set of inputs. Here, we have 2 inputs and hence $2^2 = 4$ possible outputs, i.e. the minterms.

- The typical 2 to 4 decoder circuit is:



- For convenience, we use the following alternative representation:

- Using this terminology, the previous circuit can be described as follows:



- This circuit is, in effect, a programmable **ROM** (or PROM).

- This structure has two distinct sections – the AND plane and the OR plane. This is the same basic structure for the PAL and PLA devices also.

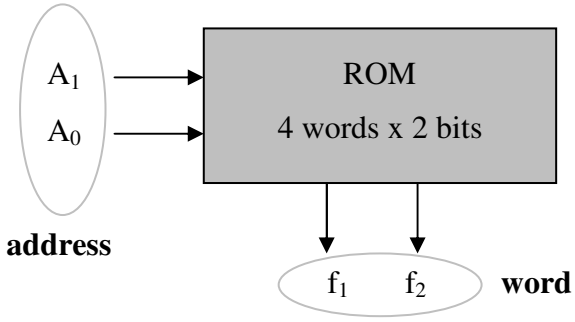
- Here, the AND plane is fixed, i.e. it is not programmable. All combinations of the inputs are produced as outputs of the AND plane, irrespective of whether or not they are needed.

- The OR plane, on the other hand, is programmable and we can program a particular function based on the appropriate selection of minterms.

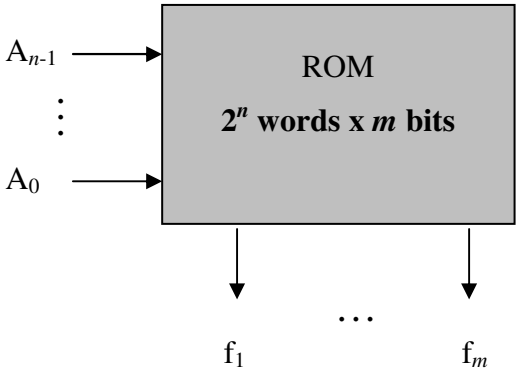
- The functionality of the ROM can be viewed in two different ways. Firstly, it can be viewed as implementing multiple functions, as we have just seen. The inputs are A_1 and A_0 and f_1 is the first ROM output and f_2 is the second ROM output.

- It can also be viewed as a means of storing code. In this scenario, the inputs A_0 and A_1 are regarded as **address lines** and the **outputs are words** consisting of bits from f_1 and f_2.

- For example, when address $A_1A_0 = 00$, the output word is $f_1f_2 = 01$ (refer to earlier truth table).
- Here, as we have 2 inputs, there are 4 possible words produced.
- The size of each word depends on the number of ROM outputs.



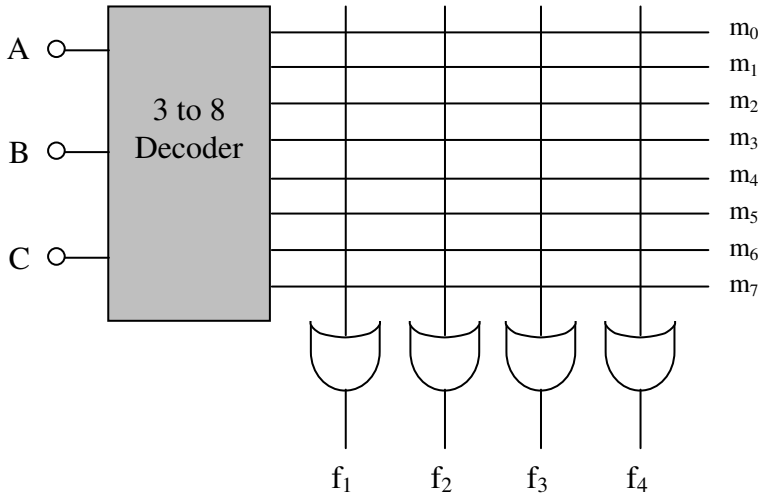
- In general, a ROM with *n* input lines and *m* output lines contains an array of **2^n words each of *m* bits**:



- ***Ex. 9.1 Implement the multi-output circuit represented by the following truth table on a ROM structure:***

A	B	C	f_1	f_2	f_3	f_4
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	1	0	1
1	1	0	0	0	1	0
1	1	1	0	0	0	0

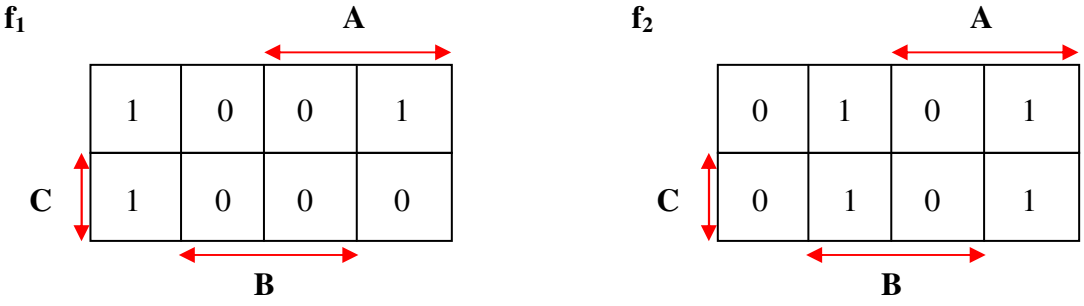
- Implementing this on a **ROM** requires a 3 to 8 decoder, i.e. 8 minterms are produced.
- The OR plane is then programmed by choosing the appropriate minterms for each of the four functions as per the table. Note, **there is no minimisation necessary**.
- The size of the ROM is 8 words x 4 bits, with the following implementation:

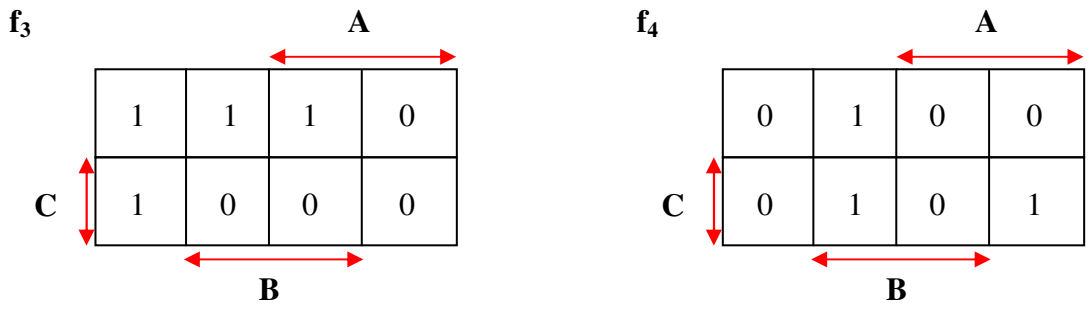


9.3 Programmable Logic Array (PLA)

- A Programmable Logic Array (PLA) comprises a programmable **AND plane connected to a programmable OR plane**.
- It does not decode the AND plane and, therefore, does not provide all possible minterms.
- Instead, it allows the AND plane to be programmed so as to generate product terms of the inputs variables.
- In the case of the PLA structure, we need to first minimise each of the output functions in order to obtain a minimal set of produce terms and, hence, a reduced AND plane.
- Furthermore, since the OR plane is also programmable and the functions can share the product terms from the AND plane, we need to **carry out multi-output minimisation**.

- ***Ex. 9.2 Implement the multi-output circuit given in Ex. 9.1 on a PLA structure.***
- Before we can implement the circuit, we need to first use Karnaugh Maps and multi-output minimisation in order to obtain the least number of product terms in the AND plane.
- The Karnaugh Maps for each function are as follows:





- This gives:

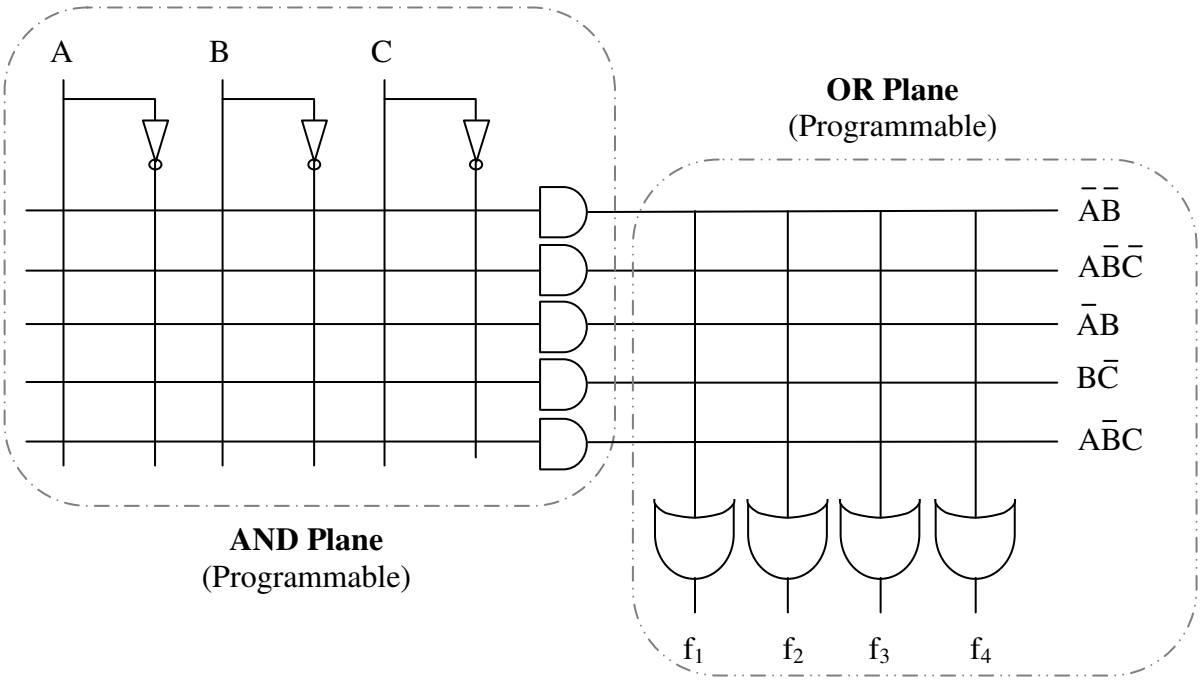
$$f_1 = A\bar{B}\bar{C} + \bar{A}\bar{B}$$

$$f_2 = \bar{A}B + A\bar{B}\bar{C} + A\bar{B}C$$

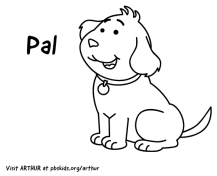
$$f_3 = \bar{A}\bar{B} + B\bar{C}$$

$$f_4 = \bar{A}B + A\bar{B}C$$

- This requires 5 different product terms and hence 5 AND gates.
- The PLA implementation is thus:

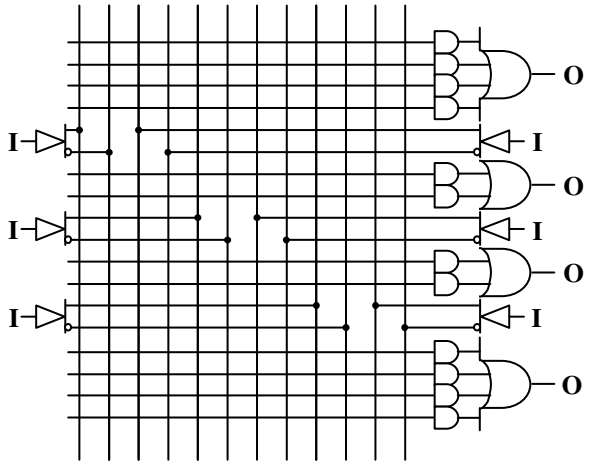


9.4 Programmable Array Logic (PAL)

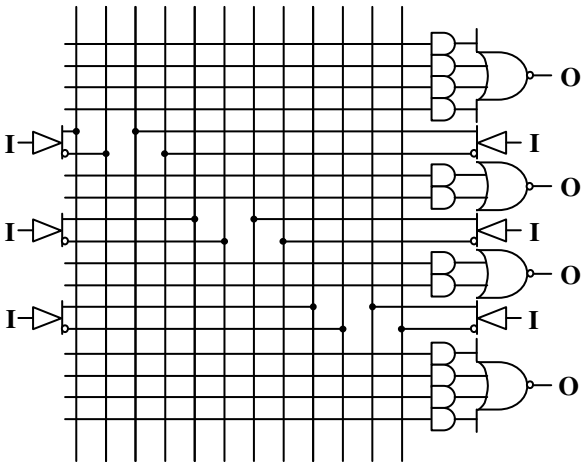


- A Programmable Array Logic (PAL) represents the third variation of the programmable logic devices where the OR matrix is fixed.

- Two sample PAL structures are shown below:



PAL - 6H4



PAL - 6L4

- Here, in the name of the structure, 6 refers to the number of available hardware inputs (denoted by I in the diagrams) and 4 refers to the number of available hardware outputs (denoted by O).

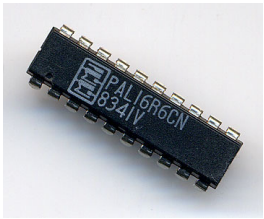
- The 'H' and 'L' refers to High and Low in terms of how the final output is achieved. ***Note the additional inverter at the end of each OR gate in the PAL 6L4 structure.***

- In other words, the first structure (PAL 6**H**4) supports an AND-OR solution format where the function solution is obtained by grouping 1's (or HIGHs) in a Karnuagh Map (i.e. f_1).

- The second structure (PAL 6**L**4) supports an AND-OR-NOT solution format where the function solution is obtained by grouping 0's (or LOWs) in a Karnuagh Map (i.e. $\overline{f_1}$).

- **In both cases, the final output is still implemented HIGH** (i.e. f_1, as opposed to $\overline{f_1}$).

- More importantly, note how the **OR plane is fixed**. **Each OR gate has an associated fixed set of AND gates**.



- In the PAL structure, only the AND plane is programmable.

- Moreover, we can see from the PAL structure that each OR gate has its own unique set of AND gates. Therefore, the PAL outputs are prevented from sharing common AND gates and, hence, AND product terms.

- Hence, **for PAL**, we only need to **minimise each function individually**.

- ***Ex. 9.3 Implement the multi-output circuit given in Ex. 9.1 on a PAL 6H4 structure.***

- Before we can implement the circuit, we need to first use Karnaugh Maps and individually minimise each of the four functions.

- Since we are using the PAL 6H4 structure, we need to group the 1's in the Karnaugh Maps.

f_1

C \ A				
	1	0	0	1
	1	0	0	0

B

f_2

C \ A				
	0	1	0	1
	0	1	0	1

B

f_3

C \ A				
	1	1	1	0
	1	0	0	0

B

f_4

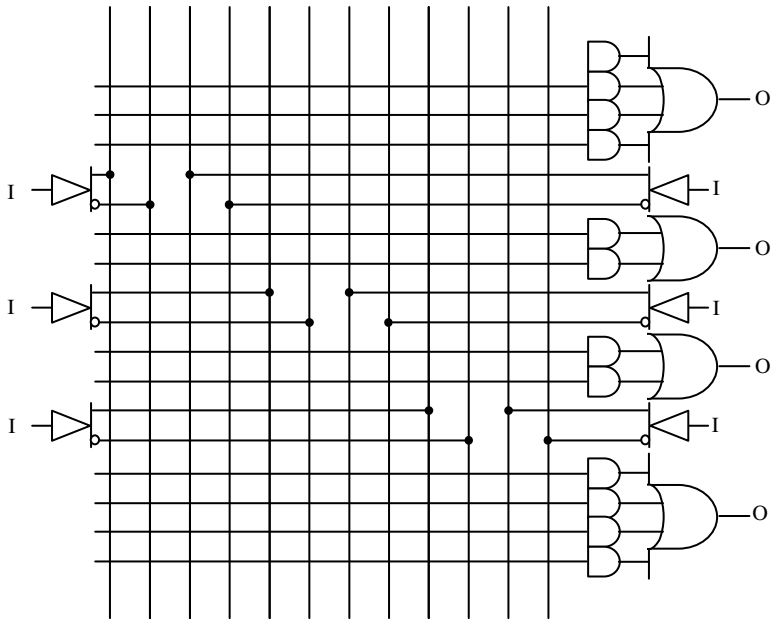
C \ A				
	0	1	0	0
	0	1	0	1

B

- This gives:

$f_1 = \overline{B}\overline{C} + \overline{A}\overline{B}$ $\quad$ $f_2 = \overline{A}B + A\overline{B}$ $\quad$ $f_3 = \overline{A}\overline{B} + B\overline{C}$ $\quad$ $f_4 = \overline{A}B + A\overline{B}C$

- Hence the PAL implementation is as follows:



9.5 Summary of ROM, PLA and PAL

- We can summarise the programmability of each device as follows:

	AND plane	OR plane
ROM		
PAL		
PLA		

- We can summarise the minimisation process required for each device as follows:

ROM	
PAL	
PLA	

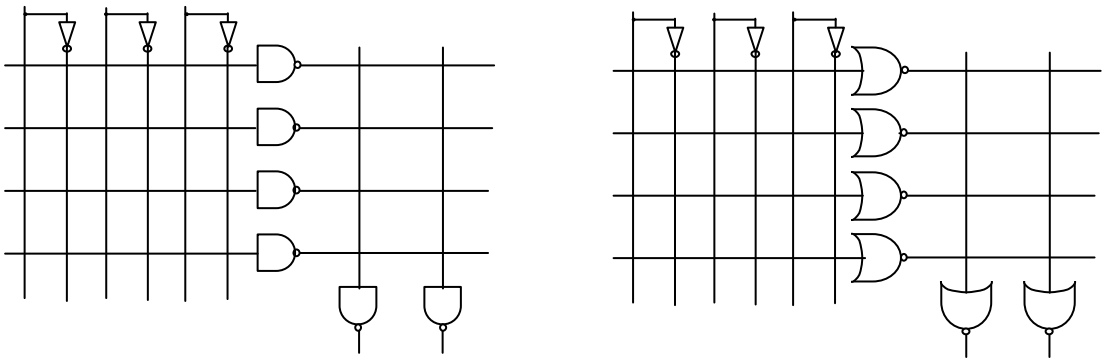
- In terms of the PAL device, we need to group the 1's or the 0's depending on which structure we are given (i.e. 1's for the PAL 6H4 and 0's for the PAL 6L4).

- The PLA also offers a more flexible solution in terms of grouping either 1's or 0's. We will highlight this in more detail in the next section.

9.6 Further PLA implementations

- In the previous sections we presented the PLA structure as a standard AND-OR configuration.

- We can easily present this structure in NAND-only and NOR-only configurations as follows:



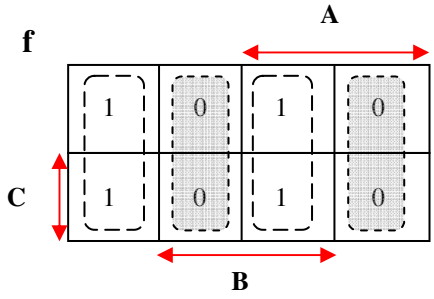
- We know how to convert a sum-of-products (SOP) solution to NAND-only and NOR-only format using De Morgan's theorems.

- Hence, using the NAND-only and NOR-only PLA devices is very straightforward.

- However, in theory we could derive two different NAND-only solutions for the same function – one by grouping the 1's and the other by grouping the 0's.

- The former leads to a solution with two levels of NAND gates while the latter leads to a solution with 3 levels of NAND gates.

- By way of illustration:



1's : $f = AB + \overline{A}\,\overline{B} = \overline{\overline{AB}\;\overline{\overline{A}\,\overline{B}}}$

2 levels of NAND gates for a PLA

0's : $\overline{f} = \overline{A}B + A\overline{B} = \overline{\overline{AB}\;\overline{A\overline{B}}}$

$\Rightarrow f = \overline{\overline{AB}\;\overline{A\overline{B}}}$

3 levels of NAND gates for a PLA

- The PLA structure given only has two levels of NAND gates and therefore the latter solution is not implementable. A similar argument exists for the NOR-only PLA.

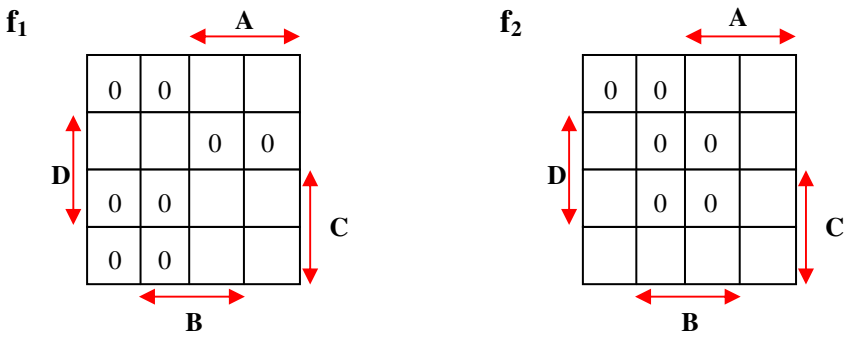
- In essence, for a **NAND-NAND PLA** we need to **group the 1's** in a Karnaugh Map and for a **NOR-NOR PLA** we need to **group the 0's**.

- Multi-output minimisation is expected either way.

- ***Ex. 9.4 Implement the following functions using a PLA with NOR-NOR architecture.***

$$f_1 = \sum(1, 5, 8, 10, 11, 12, 14, 15)$$
$$f_2 = \sum(1, 2, 3, 6, 8, 9, 10, 11, 12, 14)$$

- Since we are using a PLA with NOR-NOR architecture, we group the 0's in the Karnaugh of both functions and use multi-output minimisation.



- This gives:

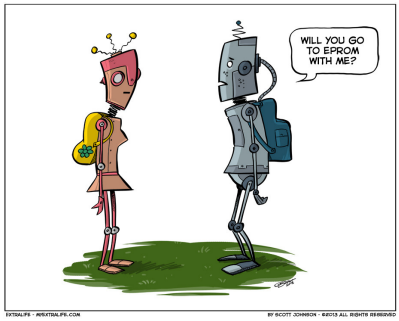
$$\overline{f_1} = \bar{A}\bar{C}\bar{D} + A\bar{C}D + \bar{A}C = \overline{(A + C + D)} + \overline{(\bar{A} + C + \bar{D})} + \overline{(A + \bar{C})}$$

$$\Rightarrow f_1 = \overline{\overline{(A + C + D)} + \overline{(\bar{A} + C + \bar{D})} + \overline{(A + \bar{C})}}$$

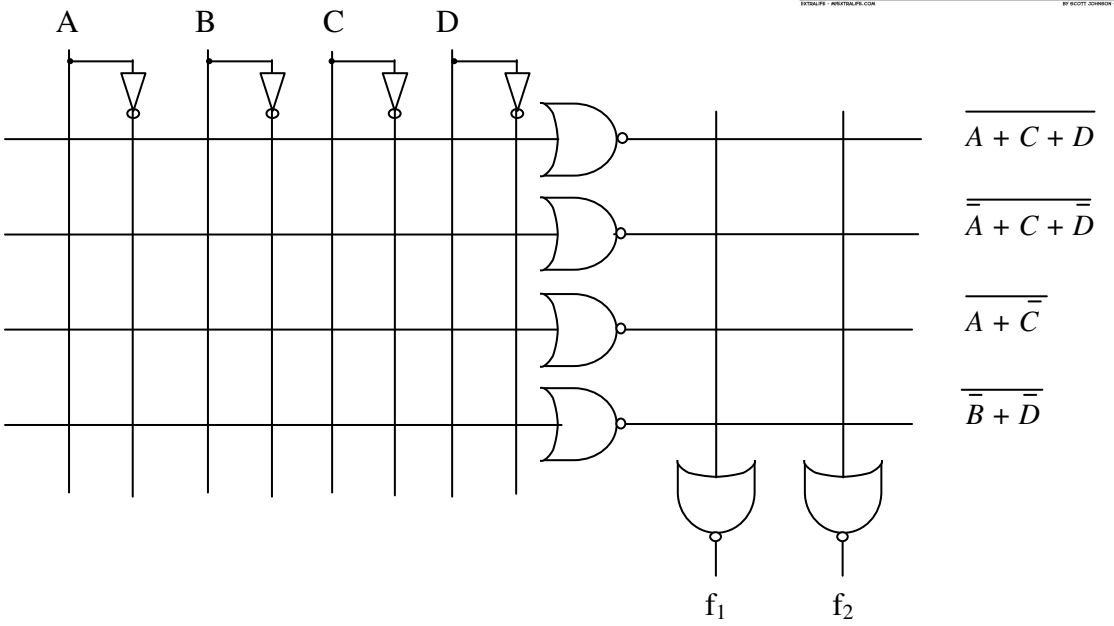
and

$$\overline{f_2} = \bar{A}\bar{C}\bar{D} + BD = \overline{(A + C + D)} + \overline{(\bar{B} + \bar{D})}$$

$$\Rightarrow f_2 = \overline{\overline{(A + C + D)} + \overline{(\bar{B} + \bar{D})}}$$

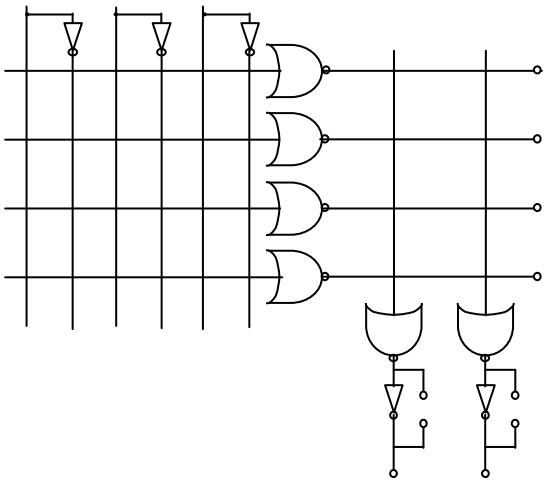


- Hence, the following PLA implementation:



NOR-NOR-NOT …

- Finally, a more flexible PLA implementation exists in the form of a NOR-NOR-NOT structure as shown below (*similarly, we can also have a NAND-NAND-NOT structure*):



- With this structure we have the option of having an additional inverter on the output for each function.

- Hence, we now have the option of implementing functions as either NOR-NOR or NOR-NOR-NOT. In other words we can consider grouping both 0's and 1's in a Karnaugh Map if need be.

- Furthermore, as each function has both options available, we can now also consider grouping 0's for one function while grouping 1's for another function at the same time, to see if this produces an even better (and hence more minimal) solution.

- This structure gives us the most flexibility but also demands the most work in terms of the minimisation process.

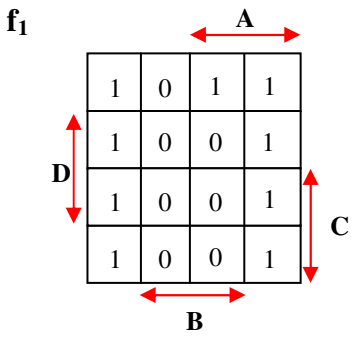
- ***Ex. 9.5 Implement the following functions using a PLA with a NOR-NOR-NOT architecture.***

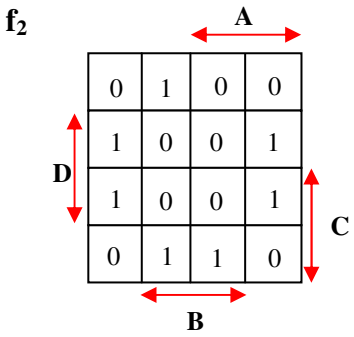
$$f_1 = \sum(0, 1, 2, 3, 8, 9, 10, 11, 12)$$
$$f_2 = \sum(1, 3, 4, 6, 9, 11, 14)$$

- For this structure, we have to consider all combinations of the outputs, i.e.:

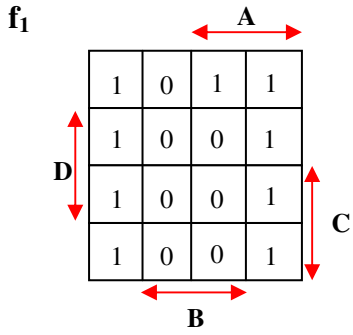
f_1f_2, $f_1\overline{f_2}$, $\overline{f_1}f_2$ and $\overline{f_1}\,\overline{f_2}$

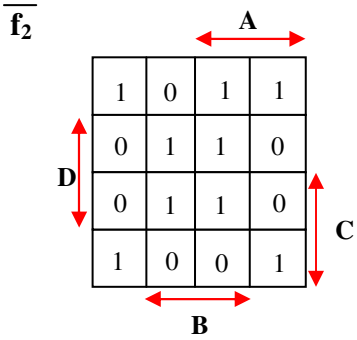
- Multi-output minimisation is required in each case.





f_1f_2 – needs 5 product terms (i.e. 5 groups)





$f_1\overline{f_2}$ – needs 4 product terms

$\overline{f_1}$

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

C

f_2

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

C

$\overline{f_1}f_2$ – needs 4 product terms

$\overline{f_1}$

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

C

$\overline{f_2}$

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

C

$\overline{f_1}\overline{f_2}$ – needs 5 product terms

- Both $f_1\overline{f_2}$ and $\overline{f_1}f_2$ offer the best minimisation solutions.
- Taking $f_1\overline{f_2}$, we can write down the following solution:

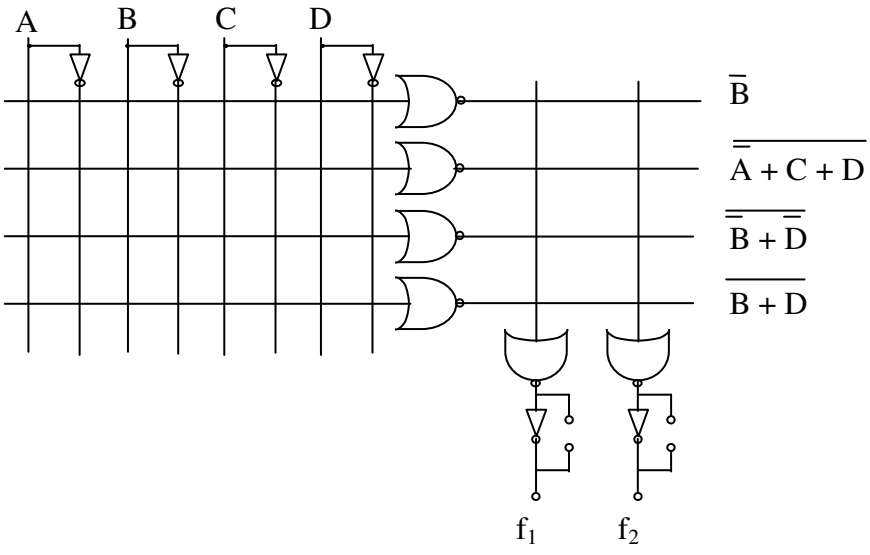
$$f_1 = \overline{B} + A\overline{C}\,\overline{D} = \overline{\overline{B} + (\overline{A} + C + D)}$$ ***NOR-NOR-NOT format***

and

$$\overline{f_2} = BD + \overline{B}\,\overline{D} + A\overline{C}\,\overline{D} = \overline{(\overline{B} + \overline{D})} + \overline{(B + D)} + \overline{(\overline{A} + C + D)}$$

$$\Rightarrow f_2 = \overline{(\overline{B} + \overline{D}) + (B + D) + (\overline{A} + C + D)}$$ ***NOR-NOR format***

- Hence the PLA implementation is given as:



- Although B is a single literal, it is still regarded as a product term as it requires a gate for implementation purposes.

- Note the line across the connections on the output f_2. This represents a short circuit connection and effectively by-passes the inverter.

- The corresponding ROM implementation for this example would have required $2^4 = 16$ product terms (13 used).

… finally!