alex-levesque.com

ELEC 271 - Lecture Notes

Digital Systems

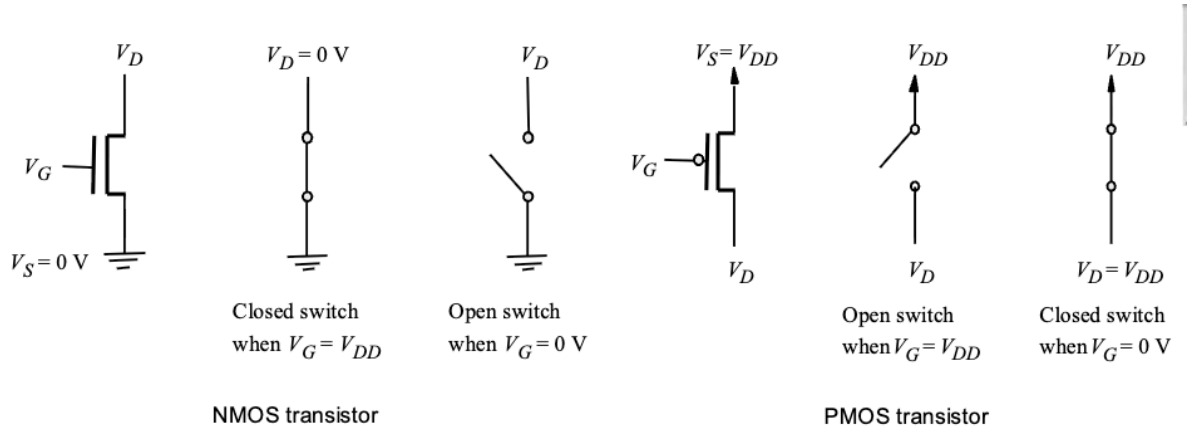
Prof. Kleber Cabral • Fall 2025 • Queen's University

Contents

1 Digital Hardware and Binary Numbers

Digital hardware, Moore's law, technology trends, chip types, layers of abstraction, design process, design flow for logic circuits, fixed point numbers and positional number representation

Analog signal is not discrete values, ex: radio signal **Digital** signal relies on discrete values, such as 1 or 0, on or off. A switch is basic element in implementing a digital system



1.1 Transistors

An NMOS (N-channel Metal-Oxide-Semiconductor) is a type of transistor used as a switching element in digital systems

When the gate voltage V_G is high, or equal to the supply voltage V_{DD}, the NMOS acts like a closed switch. When V_G is low (0V), the NMOS acts as an open switch

A PMOS is the opposite. When the gate voltage is high, the PMOS acts like an open switch. When the gate voltage is low, the PMOS acts as a closed switch.

Moore's Law and Chips

This law states that the number of transistors on a chip is doubling every 18 months. Some may predict the end of Moore's Law, and yet it keeps going.

1.2 Binary Numbers

The general form for determining the decimal value of a number in base k is given by: $Value = \Sigma_{i=-m}^{n-1} b_i k^i$

n is the highest power of the base m is the number of fractional digits after the decimal point

Conversion:

Each digit in a binary number represents a power of 2, depending on its position

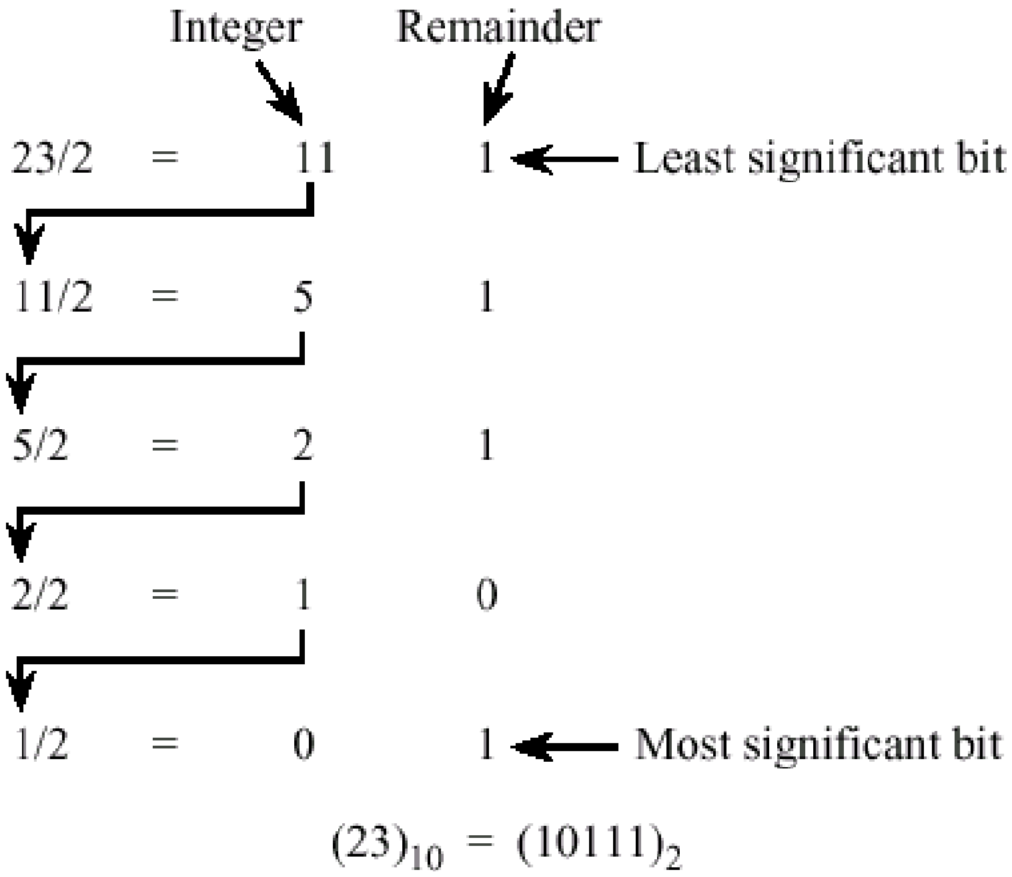
To convert a binary number to decimal, multiply each digit b_i by 2^i, where i is the position

Example: $(1010.01)_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0\ 8 + 0 + 2 + 0 + 0 + 0.25 = (10.25)_2$

Converting decimal to binary, we divide the number by 2 as many times as possible

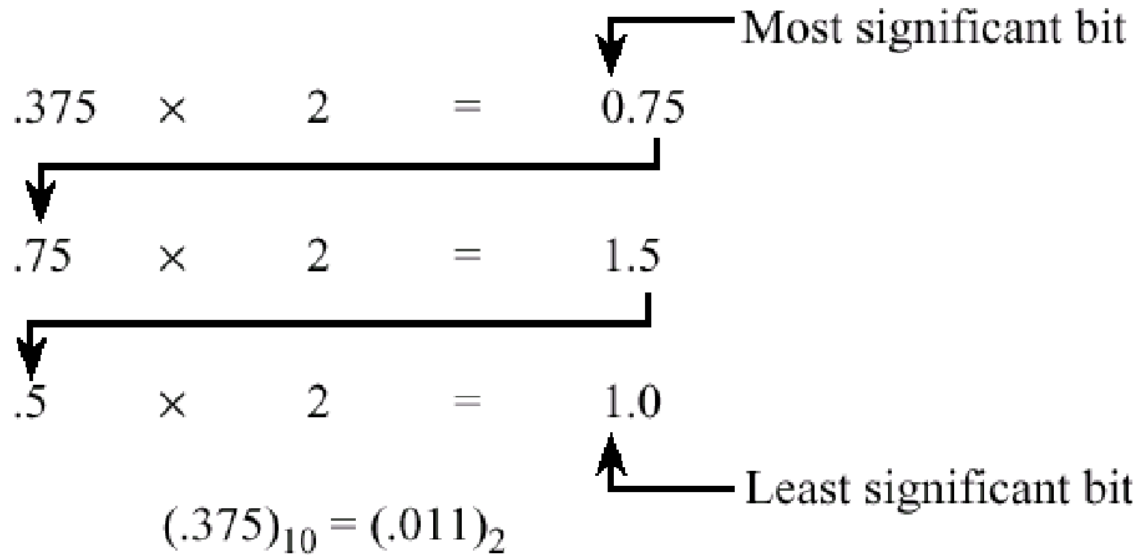
Example: Convert 23.375 to base 2:

Start with the integer part:



$$(23)_{10} = (10111)_2$$

Then do the decimal part:



$$(.375)_{10} = (.011)_2$$

Beyond base 2

The hexadecimal value 0xFF converts to: $0xFF = (15 * 16^1) + (15 * 16^0) = 240 + 15 = 255$

The $0x$ in front of the number indicates that the number is written in hexadecimal notation

2 Logic Circuits

A **logic function** $L(x)$ is a collection of signals $x_1, \ldots, x_n$

2.1 Variables and Functions

The logical AND ($\wedge$) function serves for a series connection, where the function $L(x_1, x_2) = x_1 * x_2$

The logical OR ($\vee$) function serves for a parallel connection, where the function $L(x_1, x_2) = x_1 + x_2$

The logical XOR ($\oplus$) outputs 1 if exactly one of the inputs is 1

XOR: $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$

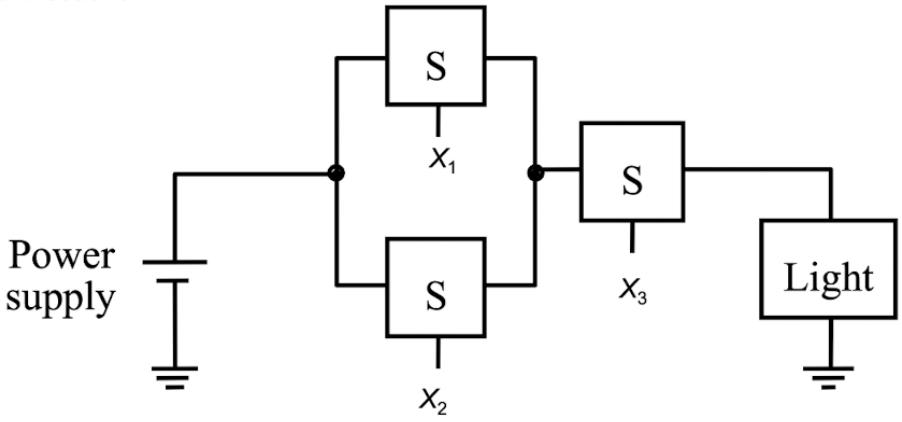
The logical XNOR ($\odot$) outputs 1 if both inputs are the same, this is the complement of XOR

XNOR: $A \odot B = (A \wedge B) \vee (\neg A \wedge \neg B)$

XOR/XNOR identity: $\overline{x_1}(x_2 \odot x_3) + x_1(x_2 \oplus x_3) = x_1 \oplus x_2 \oplus x_3$

Example:

- A series-parallel connection:

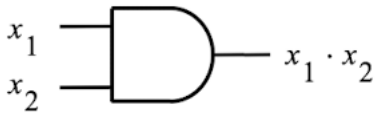


$L(x_1, x_2, x_3) = (x_1 + x_2) * x_3$

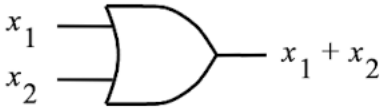
2.2 Logic Gates and Circuits

We can use several gates to represent connectors

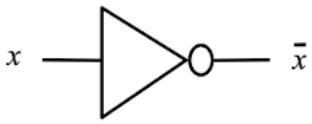
AND gates

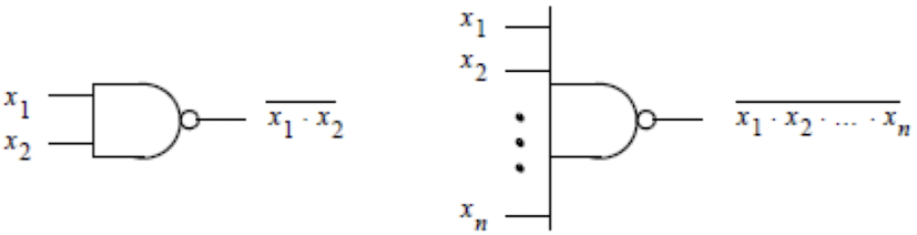


OR gates



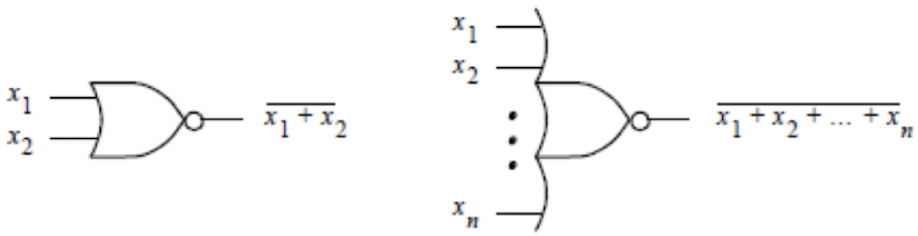
NOT gate





Truth table?

NAND gates



Truth table?

NOR gates

So, we can draw a logic network into an equivalent logic circuit using these symbols.

3 Boolean Algebra and Minterm/Maxterm

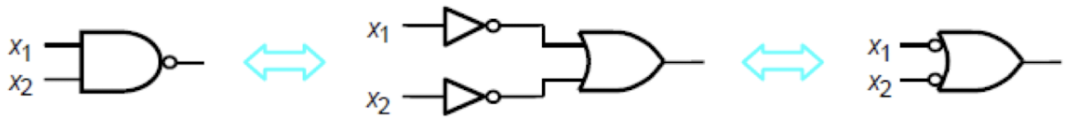
This is very similar to MTHE 217:

$$1 + 1 = 1$$
$$x + x = x$$
$$x + \bar{x} = 1$$
$$\bar{\bar{x}} = x$$

Duality: a dual of a Boolean expression is obtained by replacing all "+" operators with "." operators

Example: De Morgan's theorem states that $(x \bar{+} y) = \bar{x} * \bar{y}$

We can also showcase DeMorgan's theorem as follows:



Use DeMorgan's theorem

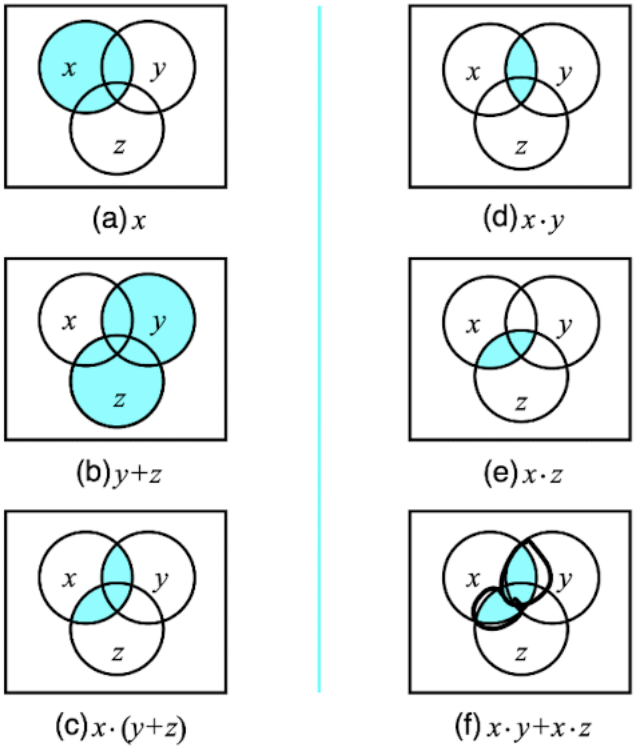
$$\overline{X_1 X_2} = \bar{X}_1 + \bar{X}_2$$

$$\neg(x_1 \wedge x_2) = \neg x_1 \vee \neg x_2$$

An important result when simplifying a logic equation is $x * \bar{x} = 0$

3.1 Venn Diagrams

We can intuitively how two expressions may be equivalent with Venn diagrams



3.2 Synthesis using operators

Minterm (m): a minterm is a product (AND) term in which each of the n variables appears once (ANDed product of literals)

Maxterm (M): a maxterm is the complement of minterm, a sum (OR) term in which each of the n variables appears once (ORed sum of literals)

$\bar{m_i} = M_i$

In a three literal circuit, examples: $m_0 = \bar{x_1} * \bar{x_2} * \bar{x_3}$ $M_4 = \bar{x_1} + x_2 + x_3$ $m_6 = x_1 * x_2 * \bar{x_3}$ $M_6 = \bar{x_1} + \bar{x_2} + x_3$

If each product term is a minterm, the expression is called a canonical sum-of-products (SOP)

If each sum term is a maxterm, the expression is called canonical product-of-sums (POS)

We can simplify a canonical SOP to a minimal-cost realization by reducing the number of gates and reducing number of input variables

POS example:

$$f(x_1, x_2) = m_0 + m_1 + m_3$$
$$f(\bar{x_1}, x_2) = m_2 = x_1\bar{x_2}$$
$$f(\bar{\bar{x_1}}, x_2) = f = (x_1 \bar{x_2}) = \bar{x_1} + x_2 = M_2$$

Simplification Tricks

$A + CB = (A + B)(A + C)$ $A + \bar{A}x = A + x$

If every case of two propositions is covered, let it = 1 Example: $(x_1x_2 + \bar{x_1}x_2 + x_1\bar{x_2} + \bar{x_1}\bar{x_2}) = 1$

4 Multiplexer and VHDL introduction

4.1 Multiplexer Circuit

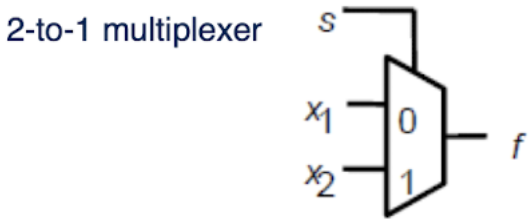
This is a circuit that chooses data from exactly one of the number of possible input sources

If we are given two sources of data x_1, x_2, an output f, and a select input control signal s, the result of f depends on s

Example:

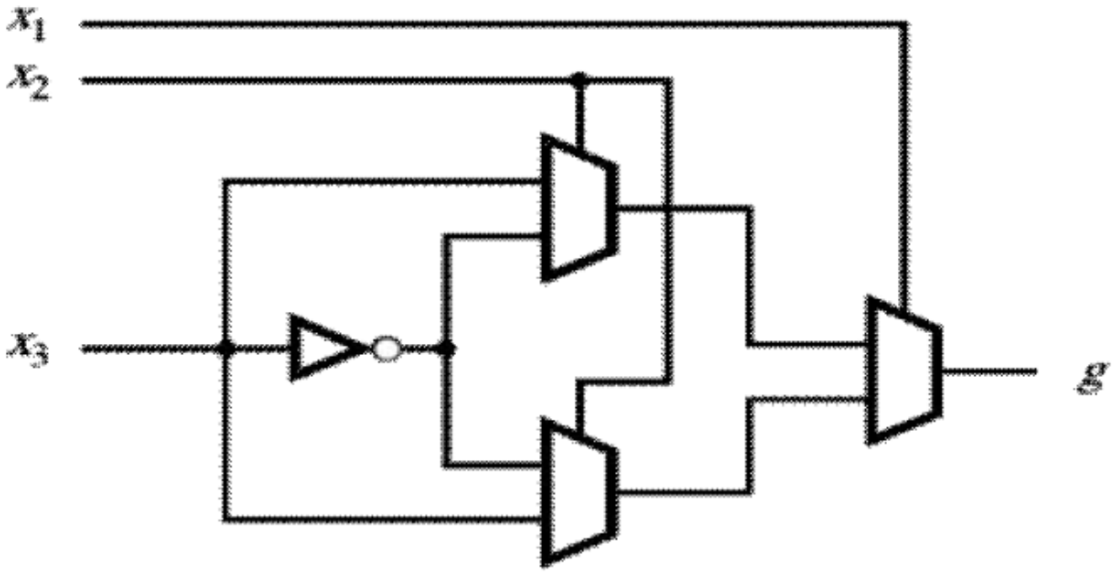
$f = (I_0 * \bar{S}) + (I_1 * S)$, or formally, $MUX(S; A, B) = \overline{S}A + SB$

If $S = 0$, $f = I_0$ If $S = 1$, $f = I$



The numbers written inside the block represent which input is connected to the output depending on the value of s

Example:



$MUX_1 = (x_2; x_3, \overline{x_3}) = \overline{x_2}x_3 + x_2\overline{x_3} = x_2 \oplus x_3$ $MUX_2 = (x_2; \overline{x_3}, x_3) = \overline{x_2 x_3} + x_2 x_3 = x_2 \odot x_3$
$MUX_3 = (x_1; x_2 \oplus x_3, x_2 \odot x_3) = x_1 \oplus x_2 \oplus x_3$

4.2 Introduction to VHDL

Entity: Defines the interface of a circuit block (its input and output ports)

Architecture: Describes the internal implementation or behaviour of the entity

Signal assignment: Defines how outputs relate to inputs, often using logic equations

```
entity example-2 IS
    port (x1, x2, x3, x4: in bit;
          f, g: out bit);
end entity example-2;

architecture LogicFunc OF example-2 IS
begin
    f <= (x1 AND x3) OR (x2 AND x4);
    g <= (x1 OR NOT x3) AND (NOT x2 OR x4);
end LogicFunc;
```

Example-2 is a circuit block with four input signals and two output signals

architecture: describes how the outputs f and g are logically derived from the inputs. The assignments to f and g are called concurrent signal assignments (they happen simultaneously in hardware)

5 Transistor Switches

5.1 Logic Values and Voltage Levels

Digital systems use binary values: 0 (low voltage, ~0V) and 1 (high voltage, e.g., 5 V)

Positive logic: 0 = low, 1 = high. This is the usual system.

The **noise margin** is the amount of "buffer" a logic signal has against electric noise before it risks being misinterpreted as the wrong logic level.

- High-state margin: $NM_H = V_{IL_{max}} - V_{OL_{max}}$

NM_H tells us how much noise a high-level signal can tolerate before being misread as low. It is the difference between the "maximum voltage guaranteed to be recognized as logic 0" and "maximum output voltage when driving logic 0"

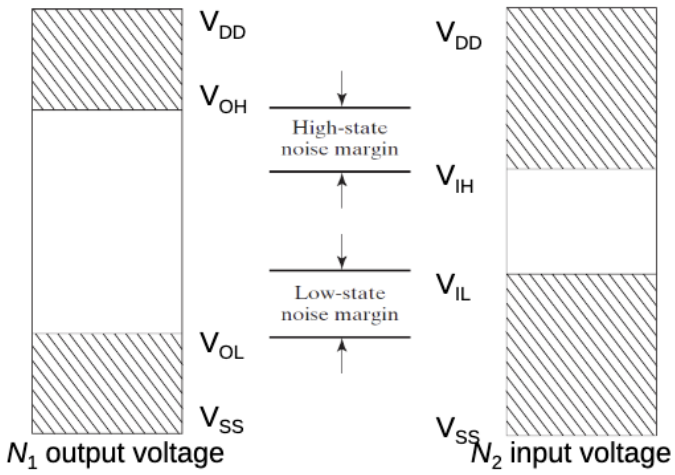
- Low-state margin: $NM_H = V_{OH_{min}} - V_{IH_{min}}$

NM_L tells us how much noise a low-level signal can tolerate before being misread as high. It is the difference between the "minimum output voltage when driving logic 1" and "minimum voltage guaranteed to be recognized as logic 1"

Below: V_{SS} is the lowest voltage V_{DD} is the highest voltage

High state noise margin:
$NM_H = V_{OH(min)} - V_{IH\ (min)}$

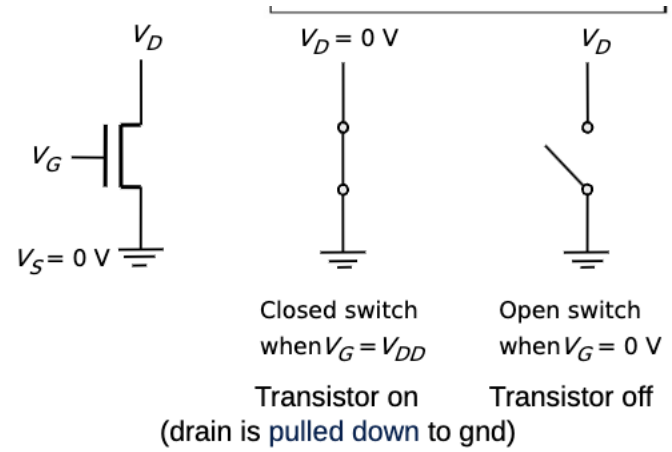
Low state noise margin
$NM_L = V_{IL(max)} - V_{OL(max)}$

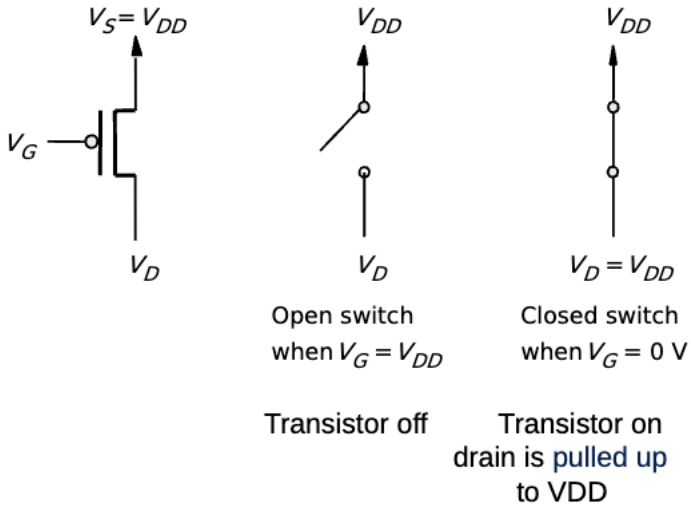


5.2 Transistor Switches

Logic gates are built from metal oxide semiconductor field-effect transistor (MOSFET), with a N-channel (NMOS) and P-channel (PMOS)

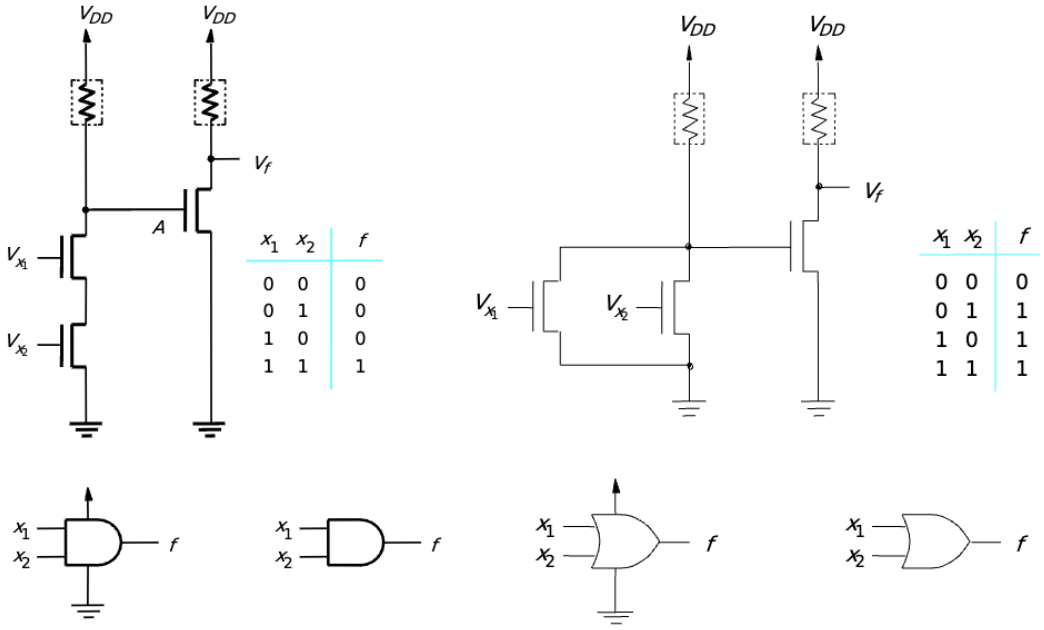
V_S is the source voltage, where charge carriers enter the channel V_G is the gate voltage, applies a voltage to control whether current flows V_D is the drain voltage, where charge carriers leave the channel V_{DD} is the positive supply voltage





AND and OR gate using NMOS

- An AND and an OR gate using NMOS technology:
-



6 CMOS Logic Gates

CMOS: Complementary MOS (both NMOS and PMOS)

CMOS is popular because no power is dissipated under steady state conditions (no current flows when the input is either low or high)

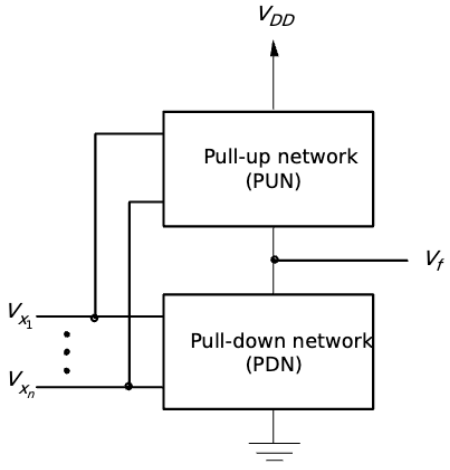
In the image below, the **pull-up network (PUN)** is made of PMOS transistors and connects the output node to V_{DD} when the logic function requires a `1`

The **pull-down network (PDN)** is made of NMOS transistors and connects the output node to Ground (0V) when the logic function requires a `0`

General formula:

PUN = PMOS transistors -> Vdd PDN = NMOS transistors -> GND

- Pull-up device is replaced with a pull-up network (PUN) using PMOS transistors.
- PDN and PUN networks have complementary functions, being dual of each other, one having transistors in series, the other in parallel, and vice versa.

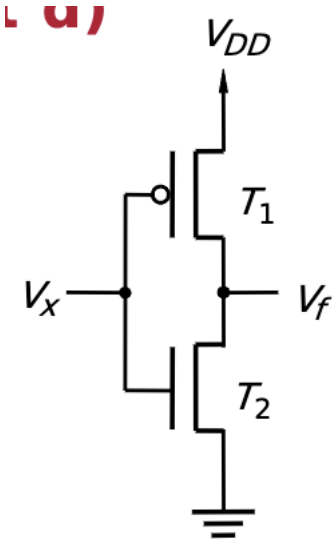


Structure of an CMOS circuit

CMOS Examples

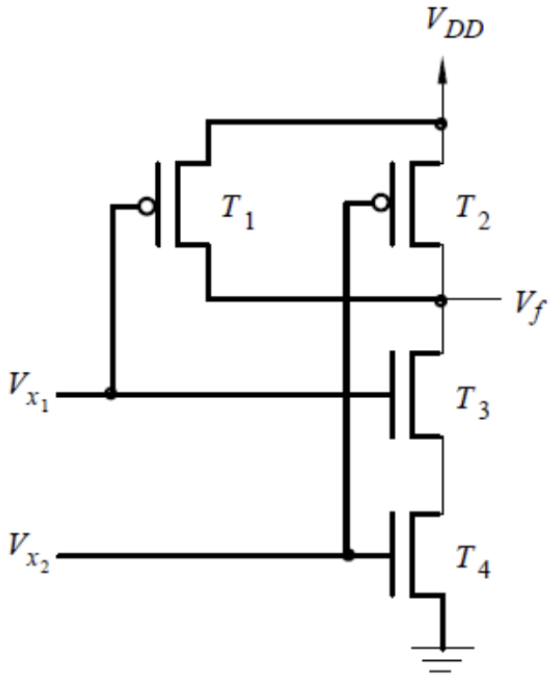
A **NOT gate** reverses the input logic state.

When the input $V_x = 0$, T_1 is ON and connects V_{DD} to V_f When the input $V_x = 1, T_2$ is OFF and disconnects



A **NAND** gate outputs the opposite of an AND gate, $f = \overline{x_1 x_2}$

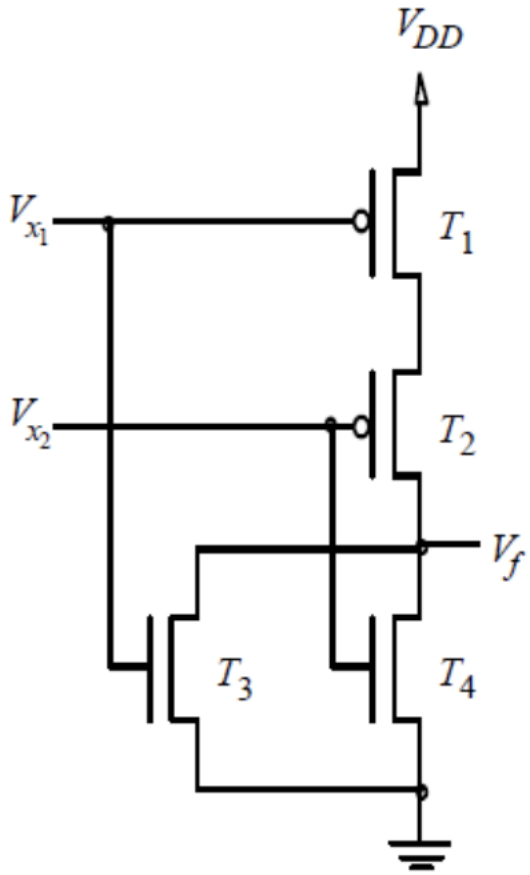
Here, when x_1 and x_2 are 0, the PUN is on and **the signal is being pulled up to** V_{DD}, not the ground



$$f = \overline{x_1 x_2}$$

x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

A **NOR** gate outputs the opposite of an OR gate, $f = \overline{x_1 + x_2}$, where the PMOS transistors are placed in series and the NMOS transistors are placed in parallel

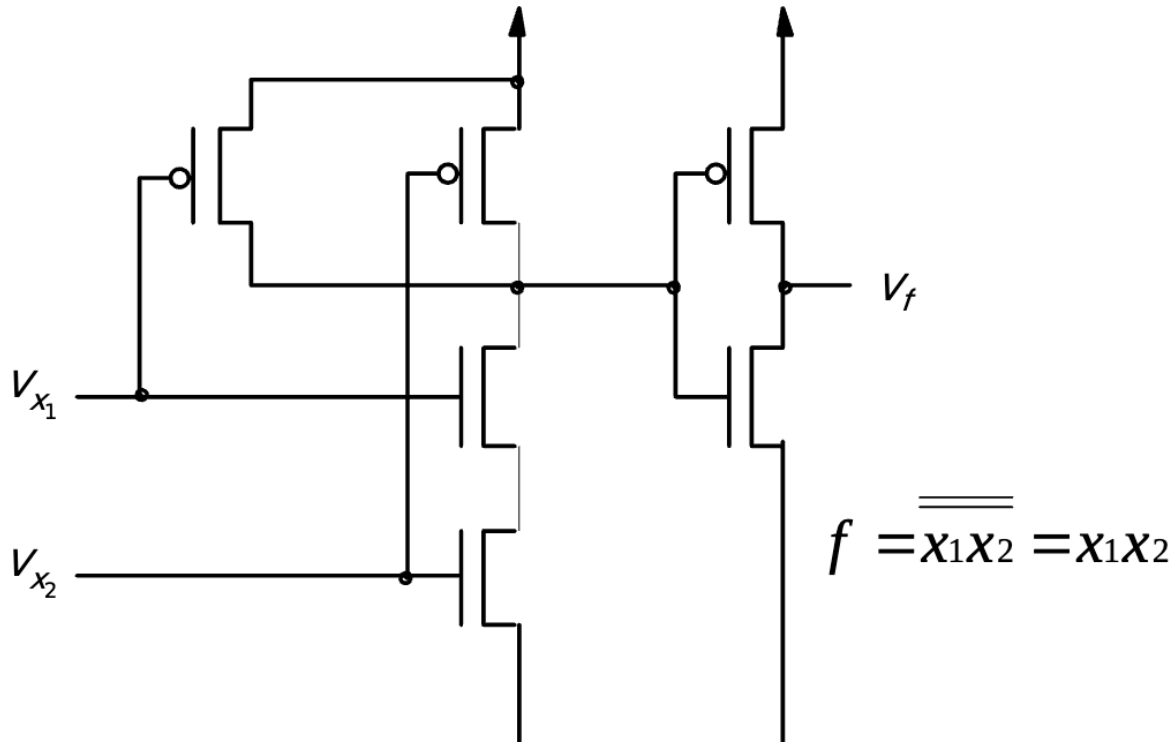


$$f = \overline{x_1 + x_2}$$

x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	on	on	off	off	1
0	1	on	off	off	on	0
1	0	off	on	on	off	0
1	1	off	off	on	on	0

Circuit Truth table and transistor states

An **AND** gate is mote complex than one may think. If you try to directly wire PMOS/N-MOS to match AND truth table, you get something messy. It is better to treat AND as *NAND + NOT*



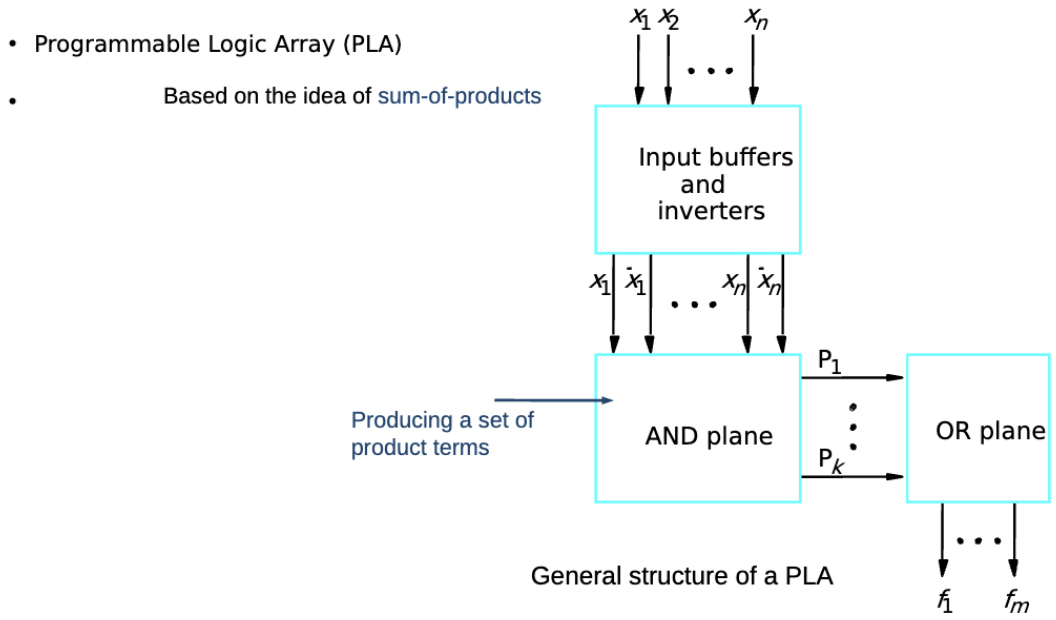
Similarly, an OR can be obtained from a NOR followed by a NOT

7 Programmable Logic Devices

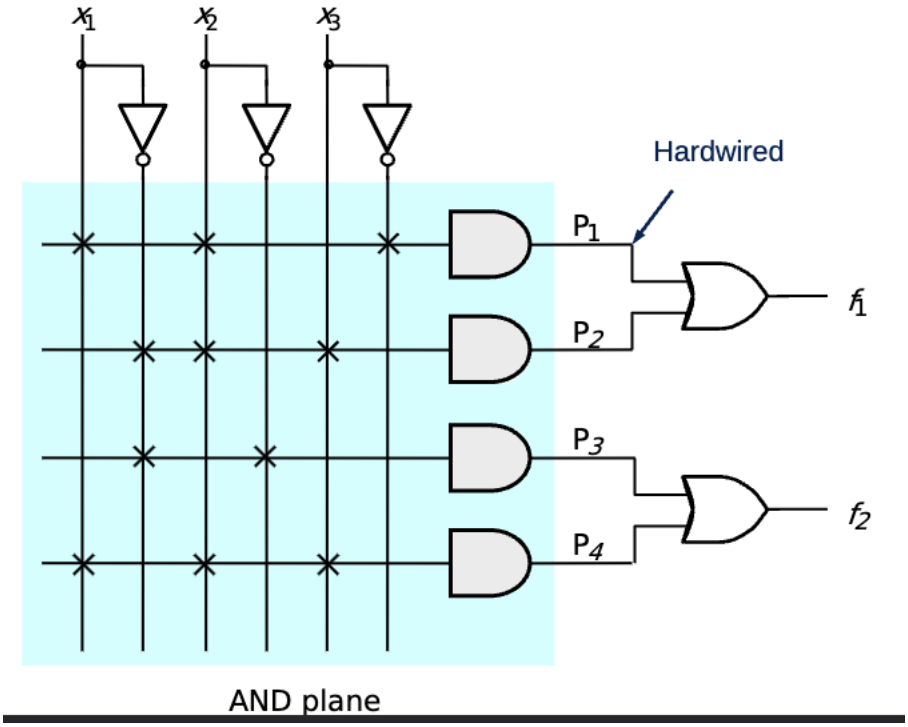
Programmable Logic Devices (PLDs) is a general-purpose chip for implementing logic circuits, where the input is logic variables and the output is logic functions

A **Programmable Logic Array** (PLA) is a type of logic device with an AND plane followed by an OR plane

Inputs can be fed into the AND plane to form any desired set of product terms, and then fed into the OR place, where they can be combined to form SOP expressions for the outputs



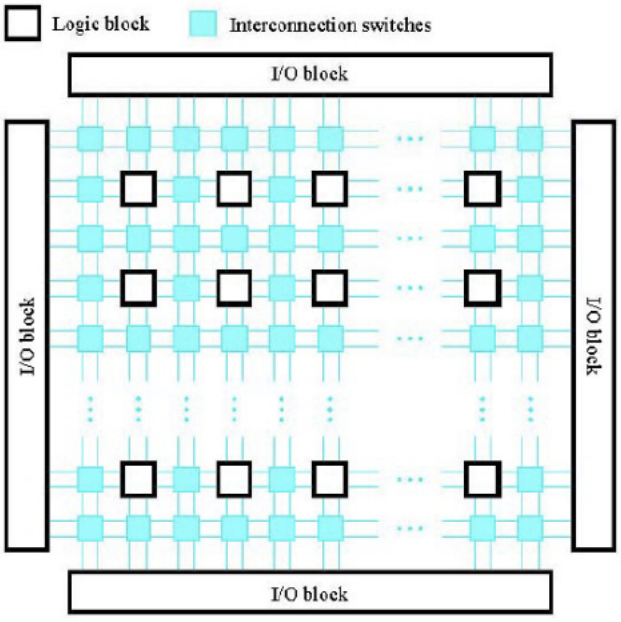
A **Programmable Array Logic** (PAL): only programmable AND array, for cost reason and better performance, but less flexible



A **Field-Programmable-Gate-Array** (FPGA) is for larger logic circuits, and do not contain AND or OR planes

FPGA chips contain three main type of resources: logic blocks, I/O blocks, and interconnection wires and switches

- General structure of an FPGA



Each logic block has a small number of inputs and outputs, with lookup tables (LUT) being the most common implementation. Commercial FPGAs typically use LUTs with up to five inputs.

A LUT is essentially a small block of memory:

- the input bits serve as the address
- the stored bit at that address is the output (0 or 1)

Because the LUT stores the complete truth table for its inputs, it can represent any Boolean function

Python analogy example:

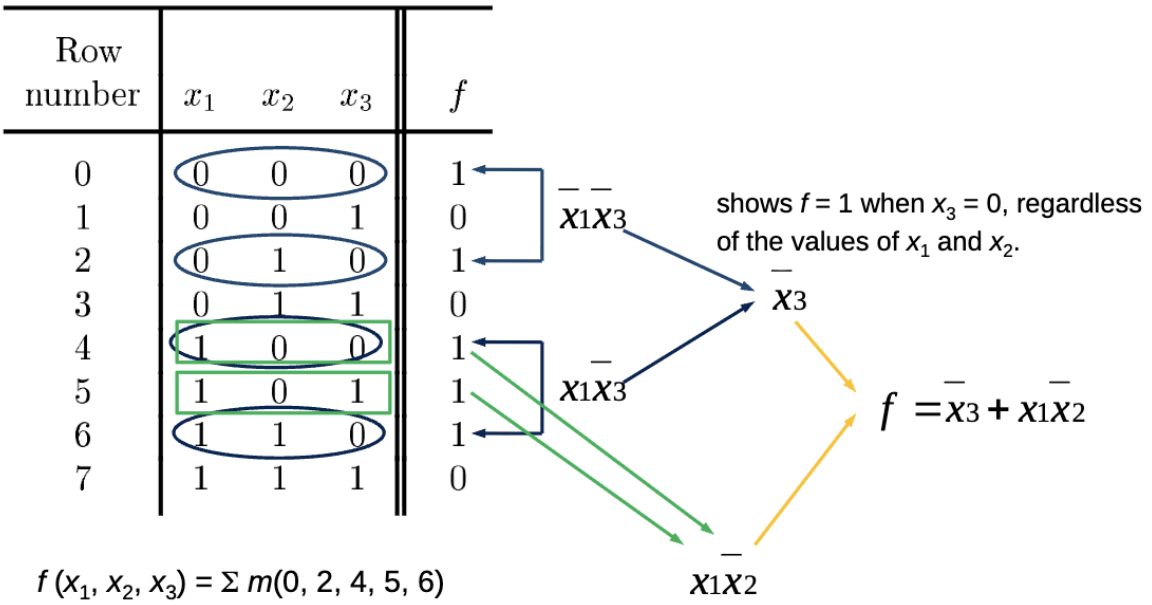
```
# XOR function implemented as a LUT
lut = {
    (0, 0): 0,
    (0, 1): 1,
    (1, 0): 1,
    (1, 1): 0
}

print(lut[(1, 0)])  # Output: 1
```

8 Karnaugh Maps (K-maps)

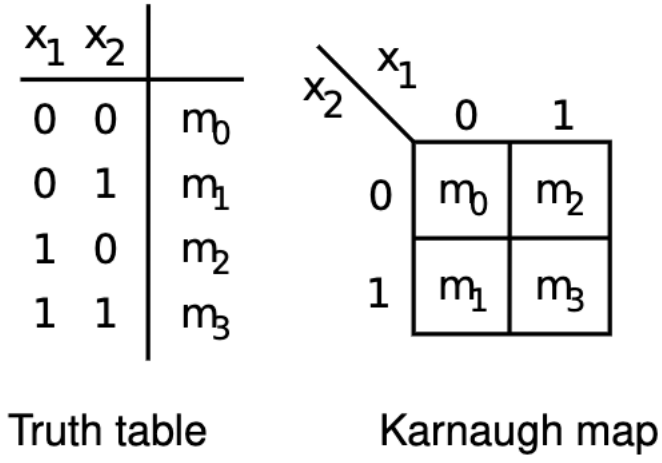
A **Karnaugh Map** provides a systematic way of optimizing logic functions. It is an alternative to the truth table and allows easy discover of groups of *minterms* for which $f = 1$ can be combined into single terms

Key point: Allows to replace **two** *minterms* that differ in the value of one variable with a single product term that does not include that variable



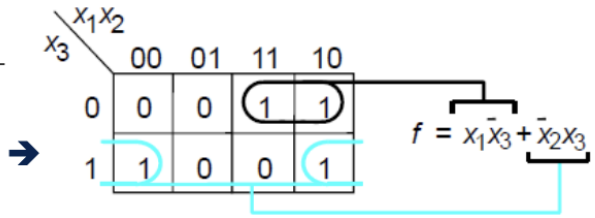
Building a Karnaugh Map:

Two variables:

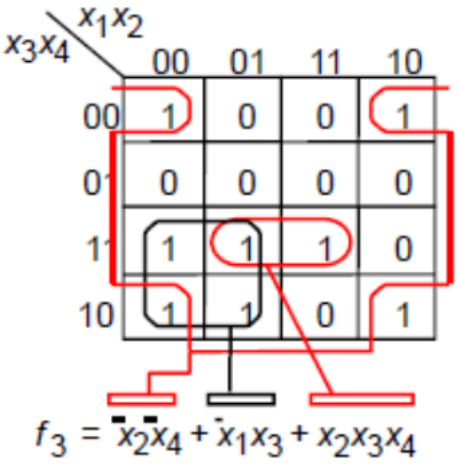


Three variables:

Row number	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0



Four variables:



8.1 Solving using a K-map

Strategy: To find as *few as possible* and as *large as possible* groups of 1s that cover all cases where the function has a value of 1

Cost: The number of gates + the number of inputs to the gates

$$f = x_1\bar{x}_2 + \bar{x}_3\bar{x}_4 \qquad \text{Cost} = 9$$

4 literals + 2 AND gates + 1 OR gate + 2 NOT gates

Literal: each appearance of variable (uncomplemented or complemented) in a product term

Implicant: A product term that indicates the input valuations for which a given input is equal to 1 (essentially a group of 1's)

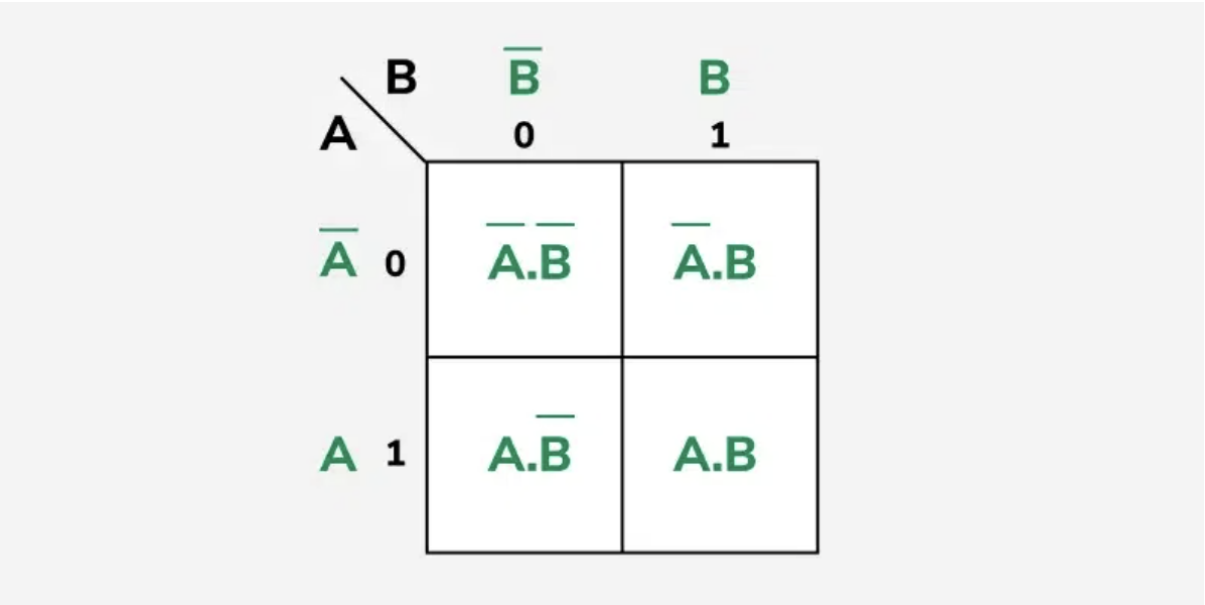
Prime Implicant: A group of 1's that cannot be combined into a larger group of 1's (maximal rectangle of 1's)

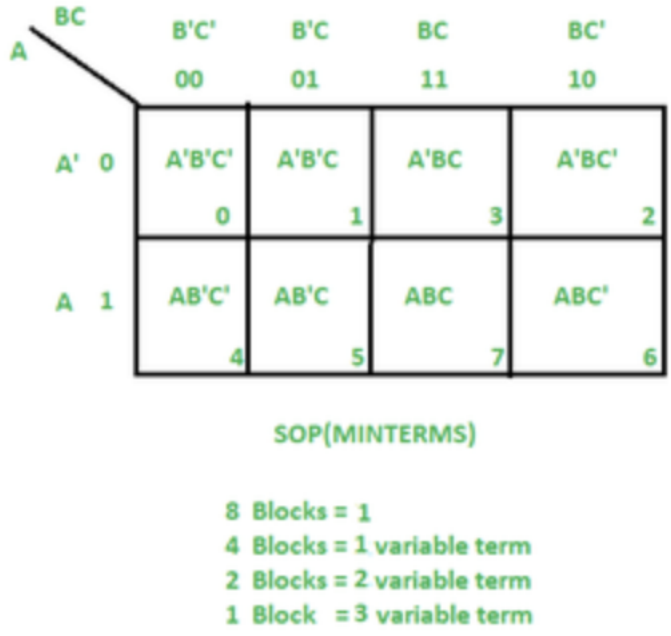
Essential Prime Implicant (EPI): A prime implicant that covers at least one 1 in the K-map that no other prime implicant covers. These are must-have groups in your final simplified function.

Cover: A collection of implicants that account for all valuations for which a given function is equal to.

8.1.1 SOP Solution

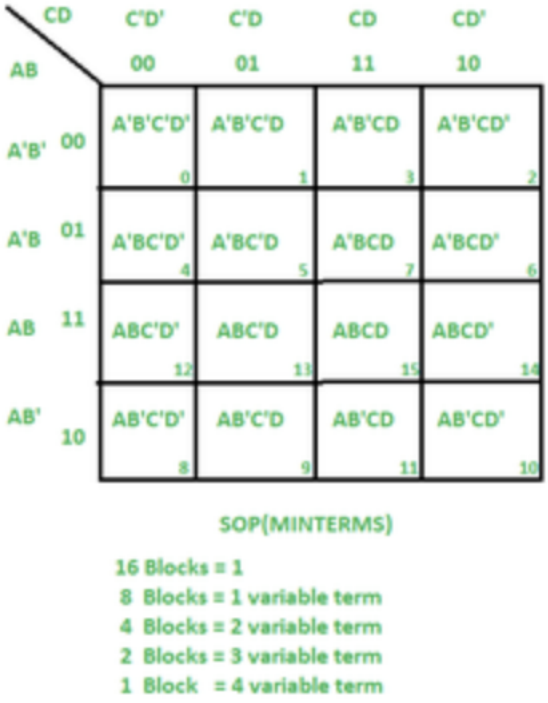
Select K-map according to number of variables





SOP(MINTERMS)

8 Blocks = 1
4 Blocks = 1 variable term
2 Blocks = 2 variable term
1 Block = 3 variable term



Identify minterms or maxterms as given in the problem and assign them to the corresponding box

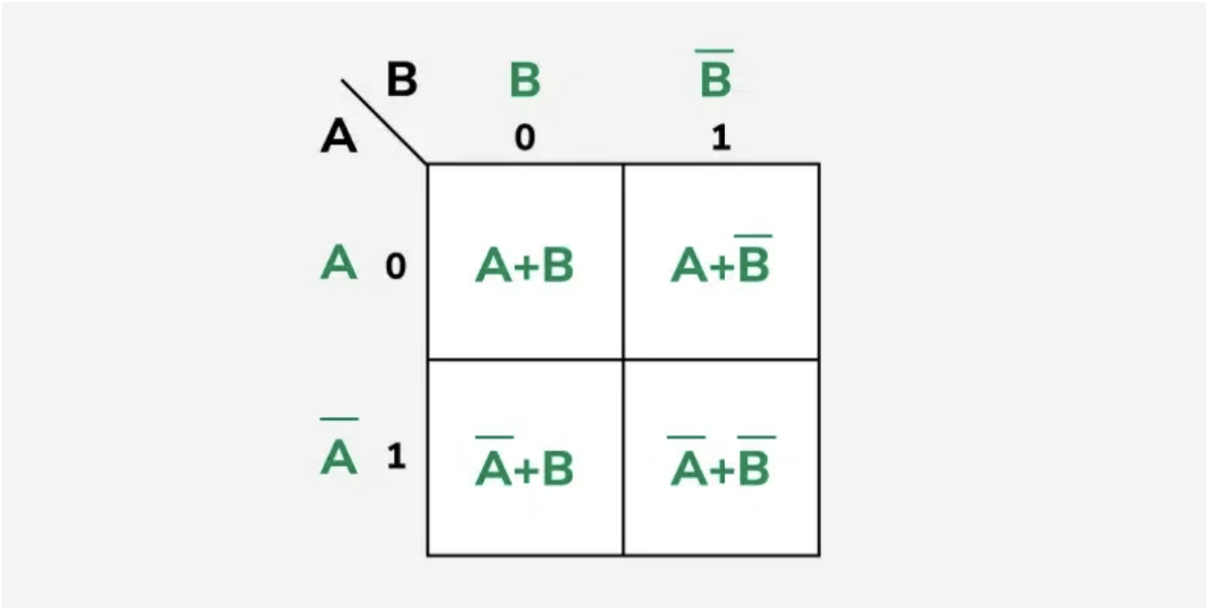
Put 1's in blocks of K-map respective to the minterms (0's elsewhere)

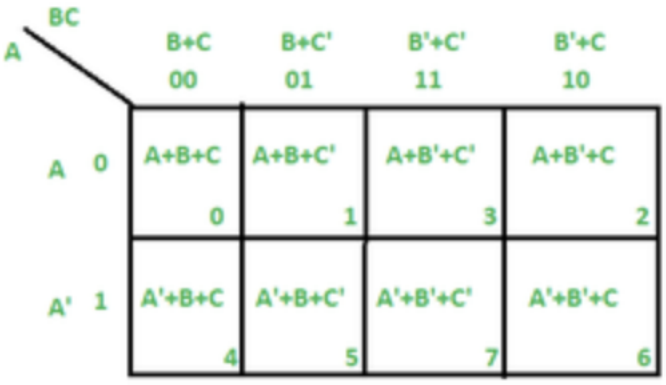
Make rectangular groups of 1 blocks containing total terms in power of two like 2,4,8 and try to cover as many elements as you can in one group

From the groups made in step 5 find the product terms and sum them up for SOP form

8.1.2 POS Solution

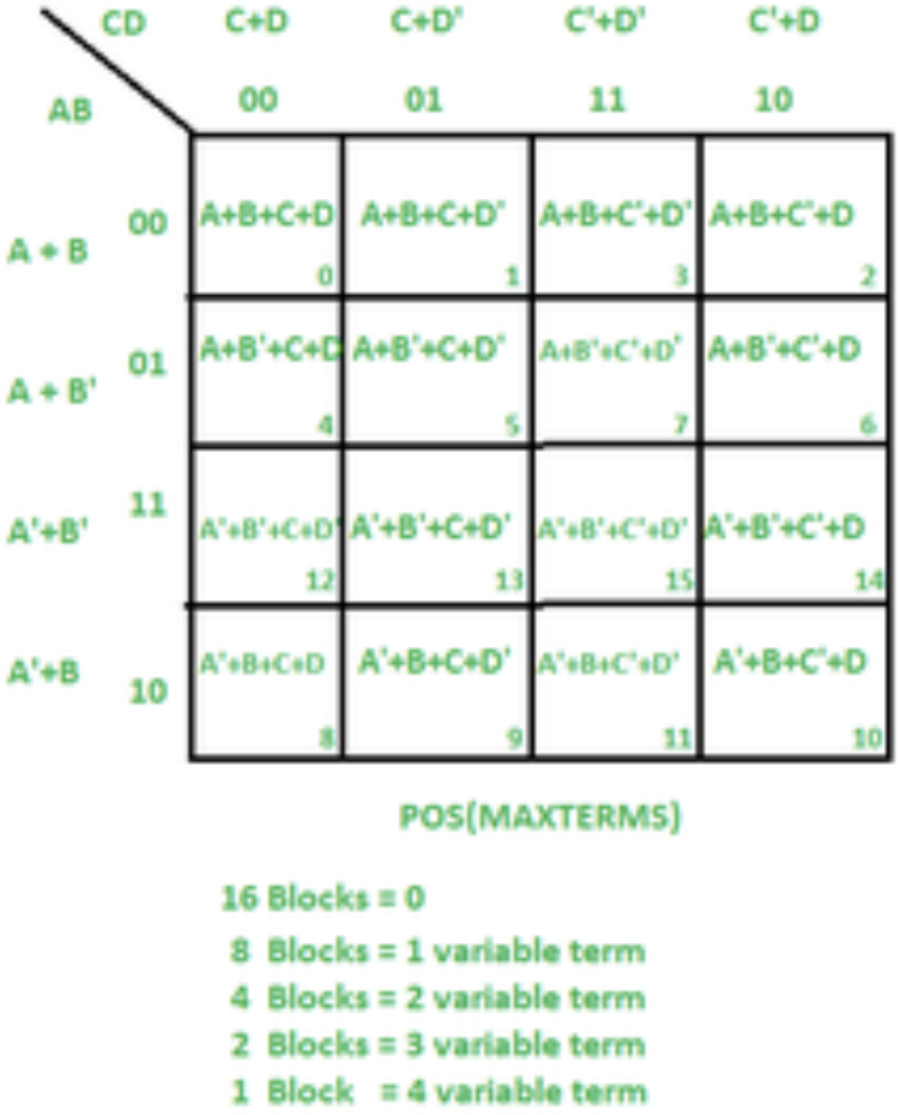
Select K-map according to number of variables





POS (MAXTERMS)

8 Blocks = 0
4 Blocks = 1 variable term
2 Blocks = 2 variable term
1 Block = 3 variable term



Identify minterms or maxterms as given in the problem and assign them to the corresponding box

For POS put 0's in blocks of K-map respective to the maxterms (1's elsewhere)

Make rectangular groups of 0 blocks containing total terms in power of two like 2,4,8 and try to cover as many elements as you can in one group

Take the complement of the groups and sum the literals. Ex: $\overline{C'DB} = (C + D' + B')$

From the groups made in step 5 find the product terms and sum them up for POS form

9 Karnaugh Maps (cont.)

9.1 Incompletely Specified Functions

When an input combination can't ever happen, it's called a don't-care condition A function with one or more don't-care conditions is called incompletely specified

Ex: Allowed inputs for x_1 and x_2 are 00, 01, 10. $(x_1, x_2) = 11$ is a don't care

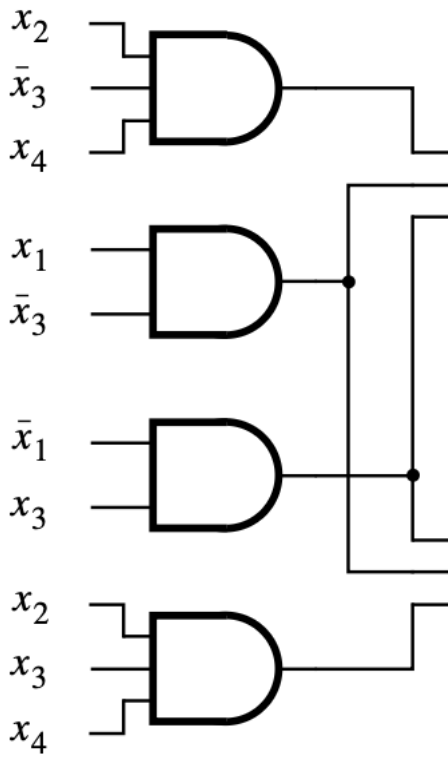
We can treat "d" in a k-map as either 0 or 1, which is useful when trying to find a minimum-cost function

9.2 Multiple-Output Circuits

A circuit can have multiple outputs $f_1, f_2, \ldots, f_n$

Example: $f_1 = x_1\overline{x_3} + \overline{x_1}x_3 + x_2\overline{x_3}x_4$ and $f_1 = x_1\overline{x_3} + \overline{x_1}x_3 + x_2x_3x_4$

The cost of each is 14, and would cost 28 in two circuits. A less-expensive realization is possible if the two circuits are combined into a single circuit with two outputs



We can utilize that the first two product terms are identical to build:

The combined circuit above shows a cost of 22, which is less than the cost of two separate circuits (28)

9.3 Multilevel Synthesis

Before, logic functions were implemented in SOP or POS form. These are two-level circuits: - SOP -> first-level AND gates feeding a second-level OR - POS -> first level OR gates feeding a second-level AND

As the number of inputs grows, two-level circuits can create gates with very large fan-in (too many inputs)

Multilevel synthesis fixes this issue with smaller fan-in gates per stage, and several more stages to decompose the function into

Example:

$f = x_1x_3x_6 + x_1x_4x_5x_6 + x_2x_3x_7 + x_2x_4x_5x_7$ needs 5 LUTs (4 for product terms and 1 for OR gate)

We can factorize into $f = (x_1x_6 + x_2x_7)(x_3 + x_4x_5)$, which is now 2 LUTs (multilevel) This is multilevel because we have a three levels (ANDs in each parentheses, one OR in each parentheses, and one AND to sum them all up).