MAINTENANCE INSTRUMENTATION

PLC PROCESS AUTOMATION

TRAINING MANUAL
Course EXP-MN-SI090
Revision 0

MAINTENANCE INSTRUMENTATION

PLC PROCESS AUTOMATION

CONTENTS

1. OBJECTIVES

The purpose of this course is to ensure the future instrumentation specialist has a basic knowledge of process automation (via Programmable Logic Control - PLC) on a predominantly oil-oriented industrial site.

2. INTRODUCTION TO AUTOMATION

2.1. WHAT IS AN AUTOMATED SYSTEM?

An industrial process consists of a set of equipment that makes it possible, using power and raw materials or unfinished products, to manufacture finished products or directly usable objects. The automation of a process means placing it under the control and command of a system that leads towards a given goal despite the disturbances it is submitted to.

An automated system consists of three parts:

- The industrial process (called the operative part),
- The control or command system (called the command part),
- The operator

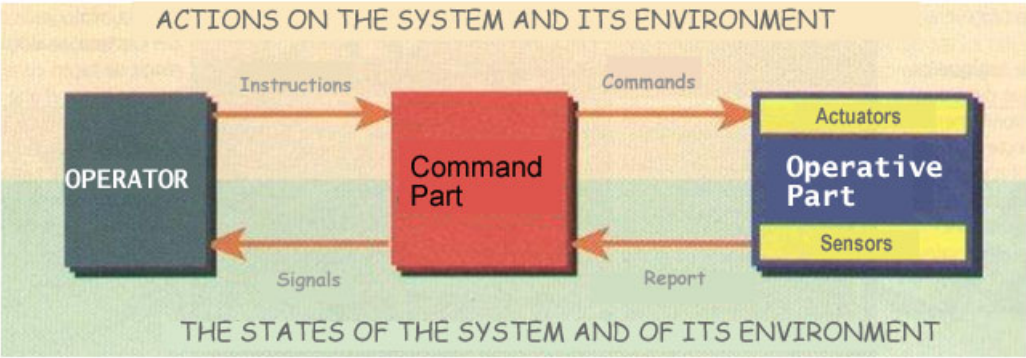


Figure 1: General description of an automated system

The operative part performs the operations by means of actuators (motor, cylinder, etc.)

It receives orders from the command part.

It sends reports to the command part.

The command part receives its **instructions** from the operator and reports from the operative part.

It sends **orders** to the operative part and signals to the operator.

It is the command part's **program** that manages all these exchanges of information.

The operator is a person who gives instructions to the system and who is capable of understanding the signals that the command part returns to him/her.

The industrial process consists of:

- Production and distribution of electrical power (hydro-electrical, thermal or nuclear power station, remote management of network control),
- Treatment of fluids (extraction, transport and refining of oil) or of minerals (mines, steel industry, cement works, etc.),
- Transformation of metals or chemicals,
- Manufacture of miscellaneous products (paper, plastic),
- Production of manufactured parts and their assembly to make machines,
- Manufacture of objects and products useful for man's work and leisure,
- Manufacture, processing and packaging of food produce,
- Transport infrastructures (sea or river ports, national or urban rail networks, motorways, airports).

The control-command system consists of:

- **Information sensors** that measure all the physical values required to monitor and command the correct operation of the automated system; they transform movements, flow rates, pressures, temperatures, levels into electrical signals. They involve several technologies, sometimes certain sensors are automated and we call them analysers (ph-meter, chromatograph, spectrograph, densimeter, refractometer, etc.…).
- **Information processing circuits** which create the command orders, the information signals that they process may be in their original form, in which case we say that processing is analog (case of regulators, synchro systems, etc.); but these signals may also be converted into binary number coded form (analog-to-digital conversion) and then processed by a computer or by a Programmable Logic Controller (PLC) and then processing is said to be digital.
- **Actuators** that perform the actions and make the physical interventions that the command system imposes on the industrial process. These actuators belong to the following three technologies: electrical (motors, generators, pump or valve commands, etc.), pneumatic (valves, cylinders, etc.), hydraulic (cylinders, hydraulic motors, brakes, etc.).

2.2. EXPRESSION OF REQUIREMENTS FOR AN AUTOMATED SYSTEM

In order to enable the dialogue between the automated system's designer-manufacturer and the user-customer of the automated system, to arrive at a clear and unambiguous description of the hardware, to take into account the technical limitations and operating restrictions, you will have to write a document specifying all the functions, all the values, the physical magnitudes and all the utilisation modes for the hardware that the two parties want to obtain.

When you look at a problem for the first time, some things are not clear and a host of details may make you forget important events. It is recommended to divide up and spread the description of the hardware over several – at least three – levels.

2.2.1. First level

This corresponds to the functional specifications that describe the automated system through its functioning coupled with the process, independently from the technique used.

These specifications must enable the designer of the automated system to understand its role (the functions), define the actions to be performed and their sequencing. All the triangular exchanges of information and energy between the automated system, the process and the outside world, all the functional safety systems must be described at this level.

2.2.2. Second level

This corresponds to the technical specifications and operational specifications which indicate the way the exchanges of information and energy are made between the automated system and the process.

The technical specifications list and explain all the information on the nature and physical characteristics of the sensors, actuators and electronic circuits (constraints, performance, limitations), to which we add the environmental conditions (temperature, humidity, corrosive or explosive atmosphere, electrostatic or electromagnetic disturbances, etc.) and the reliability conditions required to ensure operating safety.

The operational specifications indicate in a very practical way how the complete automated system is implemented (sensors, actuators, control-command) in its operating context: availability between two successive maintenance operations, ease of modification, troubleshooting, On-Off modes, absence of dangerous failures.

2.2.3. Third level

This concerns the documentation describing the functioning, utilisation, maintenance and troubleshooting of the hardware. This documentation must be constantly updated gradually as modifications are made. This will make it possible to simplify the maintenance and troubleshooting operations

3. THE HARDWARE PART: THE PLC

PLCs (Programmable Logic Controller) appeared in the United States in about 1969 where they were developed to satisfy the desire of industry to set up automated manufacturing and production lines that could follow the changes in the techniques used and in the models manufactured.

A PLC differs from a calculator in that it is a programmable electronic system specially designed for non-IT specialists. In general it is designed to be placed in the hands of personnel whose training has above all been oriented towards instrumentation and automation.

PLCs have taken the place of relay boxes because of their flexibility (putting into service, modification, etc.), and also because in complex command automated systems, the costs of wiring and debugging were becoming prohibitive.

3.1. EXTERNAL APPEARANCE OF THE PLC

PLCs are available in modular form (different types of microprocessors and inputs-outputs) and various presentations: case, rack, bay or boards.

For difficult atmospheres (temperature, dust, risk of projections, etc.) the PLCs used are contained in a sealed box, capable of withstanding a broad temperature range.

The constraints of the industrial environment take three forms:

- physical and mechanical environment;
- chemical pollution;
- electrical disturbances.

3.2. DEFINITIONS

In France these pieces of equipment are called **API** (Automate **P**rogrammable **I**ndustriel), and in the United States, **PLC** (**P**rogrammable **L**ogic **C**ontrollers).

3.3. STRUCTURE OF A PLC

Figure 2: SIEMENS S7-400 PLC

3.3.1. The base

The base is quite simply the rack in which you plug in all your input-output boards, the power supply board, the CPU board (microprocessor associated with the memory) and, lastly, your communication board.

Figure 3: A base rack

The base is very practical because the PLC's power supply is distributed to all the modules connected to this backpanel base, so you only need one power supply (via the power supply board).

It is attached to a cabinet chassis by means of its DIN rail.

There are several base variants which are defined according to the number of modules to be interconnected on it (e.g. 9 or 18 modules maximum per base).

The bases are attached in cabinets that we call 19-inch bays. Opposite you can see an example of a bay:

Figure 4: Example of a PLC bay

In the DCS Centralised Automation course, we will see that we can also install servers in this type of bay.

3.3.2. Power supply

The power supply board is used to power all of the PLC's boards installed in the base. This type of board is most often powered with 24 VDC via a stabilised filtered 230VAC/24VDC power supply.

You must always use a 230VAC power outlet taken from the uninterruptible power supply to power a PLC.

Figure 5: Various power supply boards

You will see that there is a battery on the power supply board, and above all don't take it to power your TV remote control, the PLC needs these batteries. We will see what they are really used for in the PLC Description and Operation chapter.

3.3.3. The Input / Output boards

3.3.3.1. The digital input board

The digital input board will allow us to monitor all the logic inputs of the following types:

- Valve end-of-travel,
- Pushbutton,
- Motor operating feedback,
- Pressure switch,
- Thermostat,
- Level detector,
- Etc.

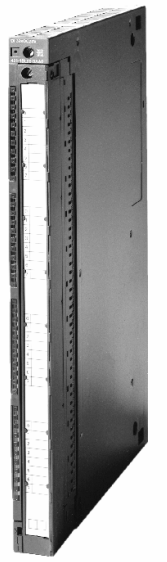


Figure 6: Digital input board

These Digital Input boards may have 8, 16 or 32 channels.

They are equipped with LEDs which indicate to us the logic status of the inputs assigned on the board.

3.3.3.2. The digital output board

The digital output board allows us to command all actuators of the following types:

- Solenoid valve,
- Motor command,
- Indicator light,
- Etc……

Figure 7: Digital output board

Digital Output boards may have 8, 16 or 32 channels. They also have LEDs indicating the logic state of the outputs assigned on the board.

3.3.3.3. The analog input board

The analog input board will allow us to have all the measurements of the following types in the PLC:

- Pressure,
- Flow rate,
- Temperature,
- Level,
- Etc.

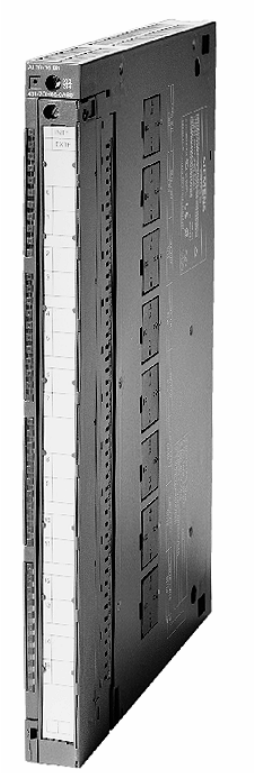


Figure 8: Analog input board

Analog Input boards may have 4, 8 or 16 channels.

3.3.3.4. The analog output board

The analog output board will allow us to regulate all the following actuators:

- Regulation valve,
- Speed variator,
- Etc.

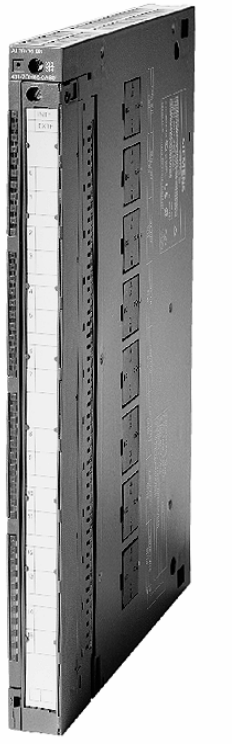


Figure 9: Analog output board

Analog Output boards may have 4, 8 or 16 channels.

You must take care, because analog input and out boards look very much alike, you are therefore advised to examine carefully the references of the boards which are often indicated on each board.

Depending on the references, you can consult the manufacturer's documentation to check what type of board you are installing. Anyone can make a mistake!!!!.

3.3.4. The Microprocessor

The microprocessor is the PLC's brain, this is what is going to manage all the PLC's I/Os according to the program that has been installed in it. We will see exactly how it functions in the following chapters.

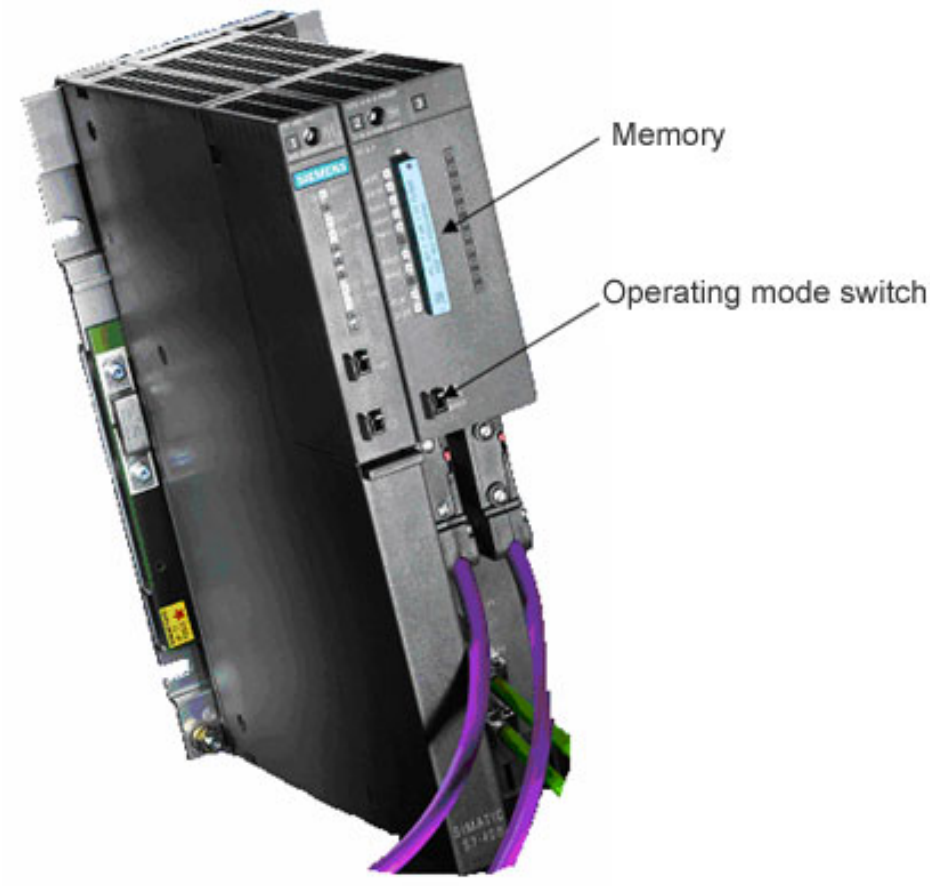


Figure 10: Microprocessor associated with a memory

Each CPU has a mode switch making it possible to change operating mode.

This is essentially a removable key switch that is used to switch between the RUN and STOP operating modes.

The following operating modes are possible:

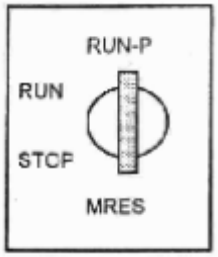
	**RUN-P**	To run the program All the PG functions are enabled
	RUN	To run the program Only the PG read functions are enabled
	STOP	The program is not run. All the PG functions are enabled
	MRES	Position in which you can perform a general memory reset.

Figure 11: A CPU's various operating modes

PG means Programming console.

General Reset:

This function clears all the user data from the CPU. This must be performed once before starting to program the PLC (at the beginning of a project for example).

This is done in several steps:

Step	Action	Result
1	Place the operating mode switch in the **STOP** position	The STOP indicator light comes on
2	Place the switch in the **MRES** position and hold it in that position (for about 3 seconds) until the STOP indicator light comes back on	The STOP indicator light goes out and after about 3 seconds it comes back on again. For new CPU models, wait until the STOP indicator light comes on for the second time. **Important**: you must not wait more 3 seconds between steps 2 and 3.
3	Return the switch to the **STOP** position and then back to the **MRES** position after 2 seconds	The STOP indicator light flashes for about 3 seconds and then comes on steady: everything is OK, **the general CPU reset** has been successful

Table 1 : Steps for performing a general CPU reset

3.3.5. The communication board

The communication board makes it possible to communicate in a network with several PLCs; we can connect up to a desktop PC to program the PLC via a coupler.

Figure 12: Communication board

It is also used by the maintenance technician who can connect his laptop PC to it to check what is not functioning correctly.

In the photograph you can see that there are two Ethernet ports, these will allow you to connect your PCs with an RJ45 network cable.

4. HOW DOES A PLC FUNCTION?

In this chapter, we are going to look at how the microprocessor (and therefore the CPU board) functions.

4.1. MICROPROCESSOR OPERATION

4.1.1. Why use a microprocessor?

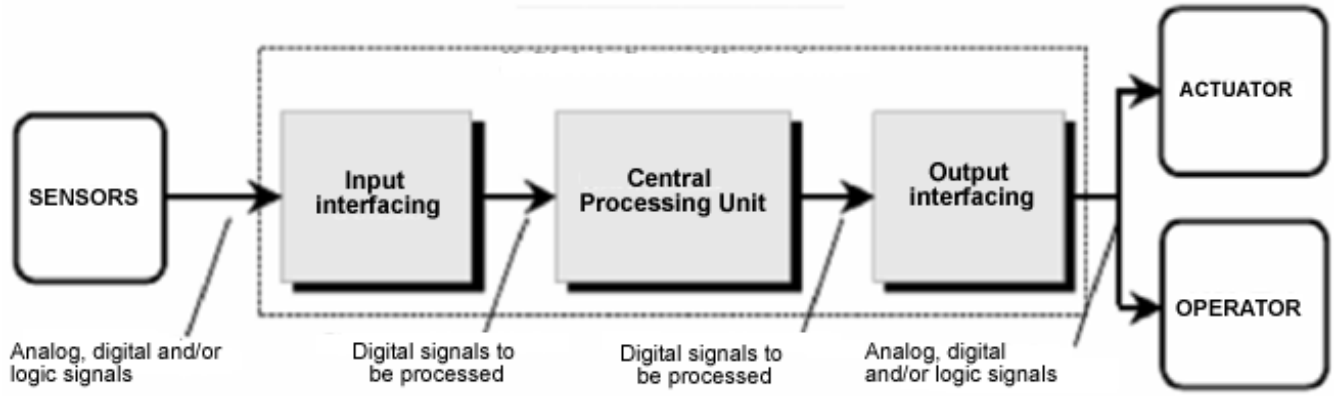


Figure 13: Microprocessor organisation

A technical object, integrating electronics, often has functions whose purpose is to process information: arithmetic (addition, multiplication, etc.) or logic (AND, OR, etc.) operations between several input signals making it possible to generate output signals.

These various functions can be performed by analog or digital integrated circuits.

But when the technical object becomes complex, and it is then necessary to perform a large amount of information processing operations, it is simpler to use a structure based on a ***microprocessor***.

4.1.2. Description and internal structure of the microprocessor

A microprocessor takes the form of an integrated circuit combining all the following components:

- The processor often called CPU (Central Processing Unit),
- The data memory (RAM and EEPROM),
- The program memory (ROM, OTPROM, UVPROM or EEPROM),

- Parallel interfaces for connecting the Inputs / Outputs,
- Series interfaces (synchronous or asynchronous) for the dialogue with other units,
- Timers to generate or measure signals with a great time precision.
- Analog-to-Digital Converters for the processing of analog signals.

4.1.2.1. Advantages

- Small dimensions,
- Not very complex printed circuit,
- Low consumption,
- Low cost.

4.1.2.2. Drawbacks

- System that is costly to develop,
- Programming requiring appropriate hardware.

4.1.2.3. Functional diagram

The functional diagram below represents a "Von Neumann" architecture (common to all microprocessors) where the program memory shares the same bus as the data memory.

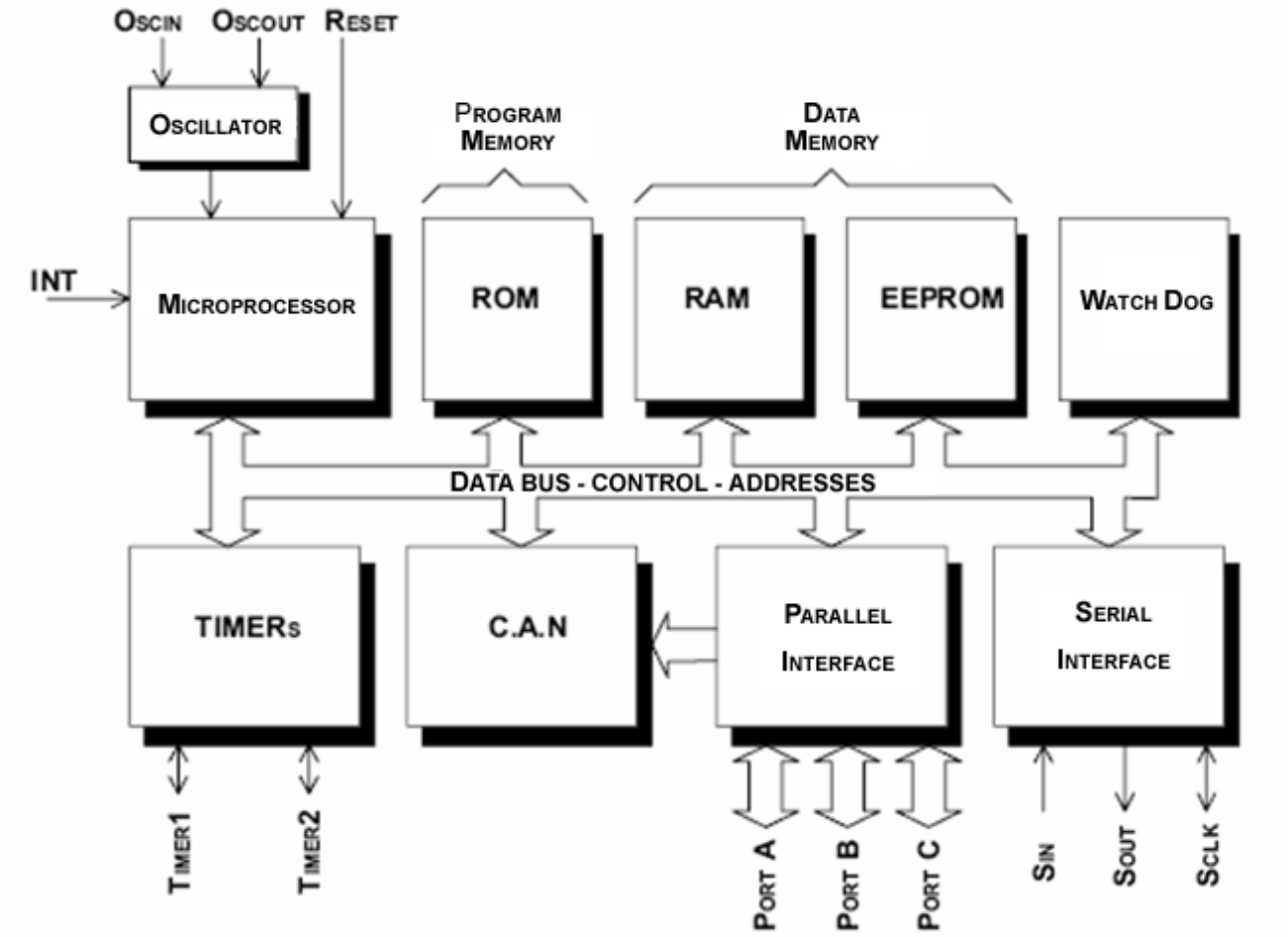


Figure 14: Functional diagram of a microprocessor

4.1.2.4. The CPU

A microprocessor sequentially executes the instructions stored in the program memory (ROM). It is therefore capable of operating on binary words whose size in bits is that of the data bus (sometimes the double for certain microprocessors).

It is generally made up of the following elements:

- One or more **accumulator registers** temporarily containing the operands as well as the results of the operations,
- **Auxiliary registers** making it possible to relay the accumulators,
- **Index registers** for the indirect addressing mode,
- A **program counter** pointing to the address of the next instruction to be executed, its size is the same as that of the address bus,

- An **Arithmetic Logic Unit** (ALU) making it possible to perform operations between the accumulator and the operands,
- A **condition code register** indicating certain particularities concerning the result of the last operation (carry, zero, interrupt).

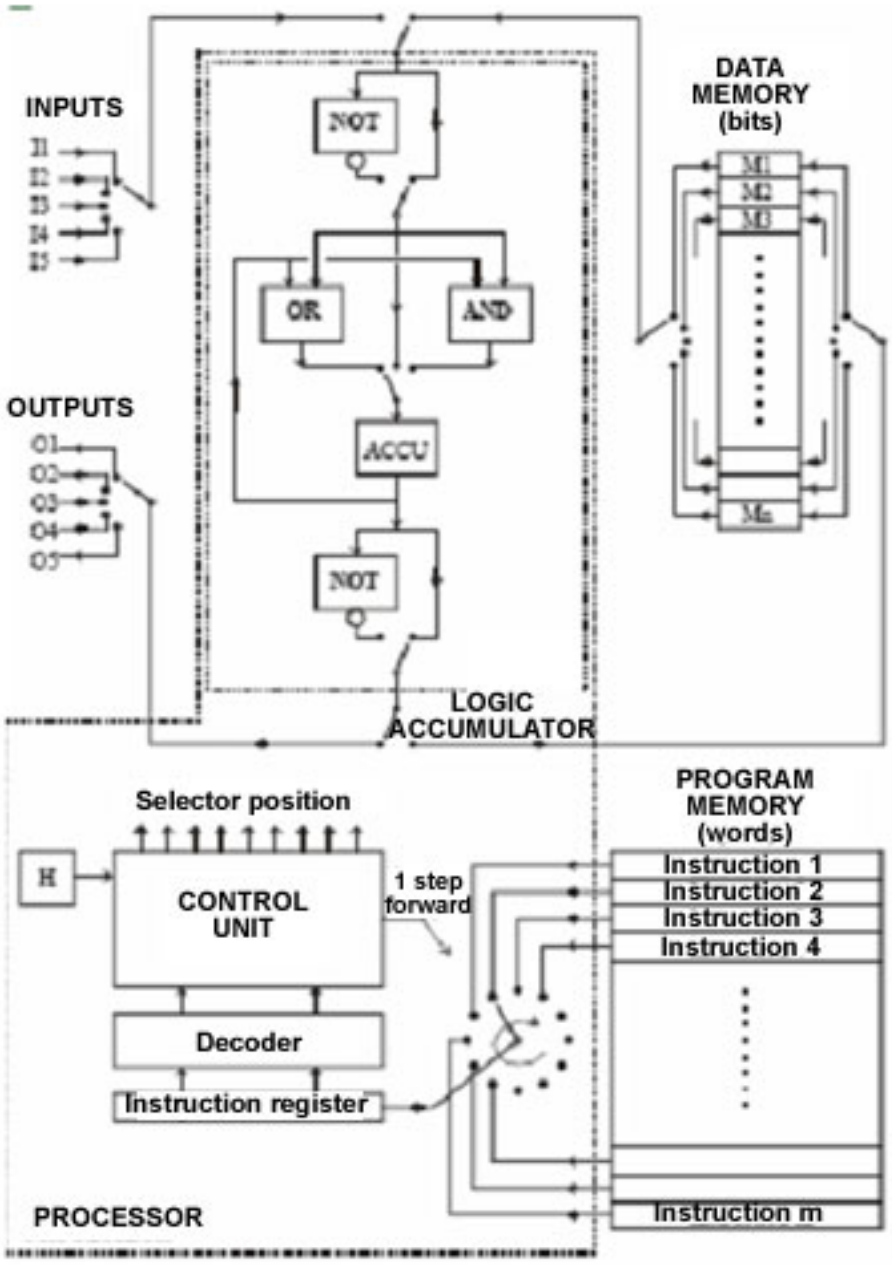


Figure 15: Detailed diagram of a microprocessor

As you can see in the figure above, the **logic unit** is associated with a data memory (which is bistable and made up of n bits) which is used to memorise the sequential information and store the intermediate results.

On the processor's **control unit** you also have a series of selectors that allow the logic operators, inputs, outputs and memories to establish communications.

The logic accumulator (ACCU) that is contained in the **logic unit** is a special type of memory cell which intervenes in a privileged way in the logic operations. It is also used to store the result obtained by the logic operation.

The processor's **control unit** manages overall operation under the control of the **instructions** stored in the program.

The **instructions** are sent in succession to the instruction register, and are then decoded and, according to the operations that they indicate, the control unit generates the orders required for their execution. This represents a **sequential and cyclic** execution of the program.

Instruction format

An instruction must provide all the indications required to execute the operation and it uses several bits forming a word.

Reminder: a word is made up of 16 bits. A byte is made up of 8 bits, so a word consists of two bytes.

In our instruction we can see that in our word we have 5 bits for the operator code and 11 bits for the address.

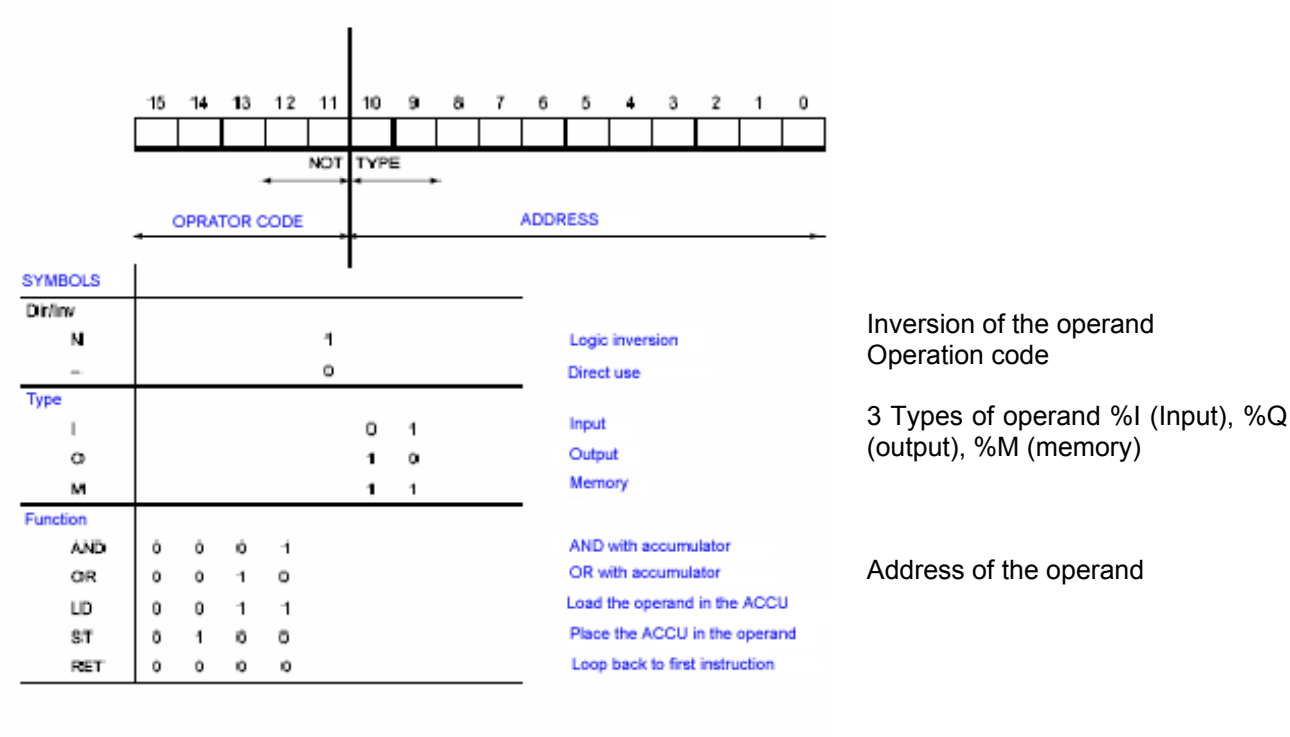
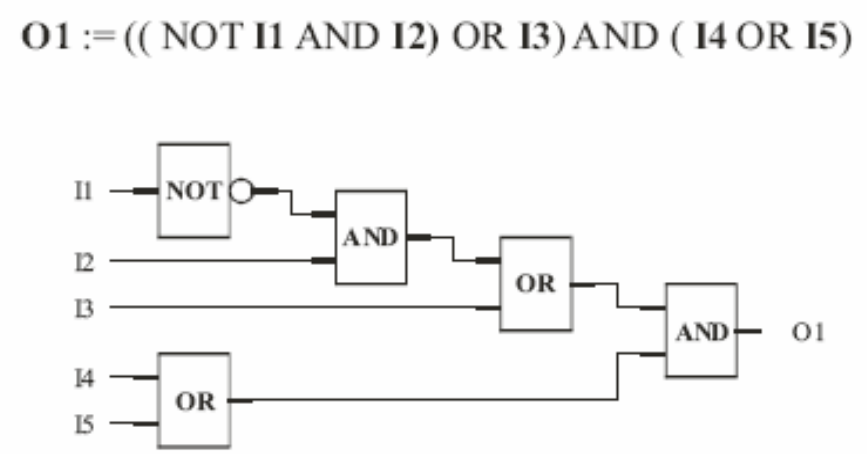


Figure 16: Instruction format

Example:

The output is O1, the inputs are I1, I2, I3, I4, I5



The list of instructions is as follows:

1	**LND I1**	Put inverted I1 in the ACCU
2	**AND I2**	And between ACCU and I2 ➲ result in the ACCU
3	**OR I3**	Or between ACCU and I3 ➲ result in the ACCU
4	**ST M1**	Save ACCU in memory M1
5	**LD I4**	Put I4 in the ACCU
6	**OR I5**	Or between ACCU and I5 ➲ result in the ACCU
7	**AND M1**	And between ACCU and M1 ➲ result in M1
8	**ST O1**	Put the value of ACCU in output O1
9	**RET**	Loop back to instruction No. 1

In this example, we have an instruction in a textual language.

4.1.2.5. The program memory

This device contains the instructions for the program that the microprocessor must execute.

This type of memory, called Read Only Memory, can only be read-accessed.

A special and appropriate procedure is required to program this type of memory.

There are different types depending on their programming mode:

- ROM whose content is programmed when it is manufactured,
- PROM which is electrically programmable once only by the developer (also called OTPROM),
- EPROM which is electrically programmable and erasable by means of UV (also called UVPROM),
- EEPROM which is electrically programmable and erasable.

4.1.2.6. The data memory

This device is used to temporarily store the data generated by the microprocessor during the various digital processing phases (result of operations, sensor states, etc.). These memories are read- and write-accessible.

There are two types:

- Random Access Memory (RAM) (data lost if the power is cut out) with a relatively short read and write time (some nanoseconds),
- non-volatile read only memory (EEPROM) (data kept if the power is cut out) with a relatively long write time (some milliseconds) with respect to the read time which is quite short (some nanoseconds).

4.1.2.7. The parallel interface

This type of interface, distributed over several ports (8 bits maximum), makes it possible to take into account the logic states applied at input (sensor state) or to generate binary output signals (actuator commands).

The pins of these ports can be configured as input or output, with different options (pull-down resistors, open collector outputs, interrupt, etc.). These pins' configuration and logic state is obtained by means of write or read operations in different registers associated with each port.

You will also find:

- A direction register for an input or output configuration,
- A data register that copies the logic states of each of the port's pins,
- An option register enabling several input or output configurations.

4.1.2.8. The serial interface

This type of interface allows the microcontroller to communicate with other microprocessor-based systems. The data sent or received takes the form of a succession over time (in a single bit) of binary values that are the image of a word.

There are two types of serial link:

- Synchronous link,
- Asynchronous link.

Synchronous serial link

In this device transmission is synchronised by a clock signal delivered by the master unit.

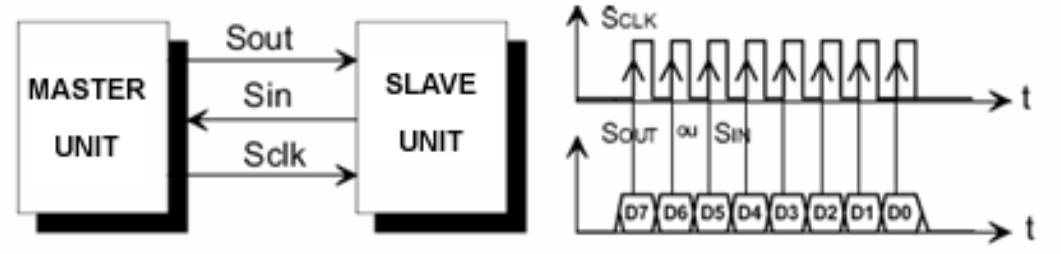


Figure 17: Synchronous serial link

Asynchronous serial link

This device does not have a synchronisation clock signal. The units that are linked together each have an internal clock set to the same frequency. When one unit wants to transmit a binary word, it generates a falling edge on the transmitter line.

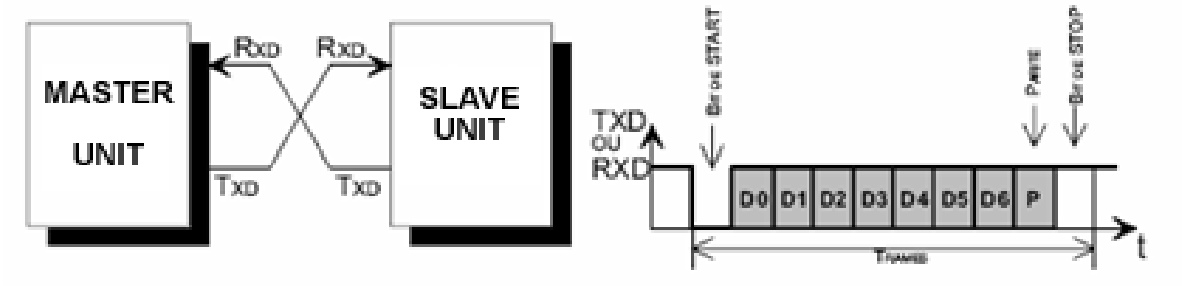


Figure 18: Asynchronous serial link

At the end of transmission of that word, the line switches back to high. The data item to be transmitted may contain an additional bit called the '***parity***' bit which is used to correct errors.

This is the most commonly used type of link in automated systems.
Parameters used for the RS232 standard:

- *Word length: 7 bits (e.g. ASCII character) or 8 bits.*
- *Transmission speed: this is defined in bits per second or bauds. It can take values going from 110 to 115200 bauds.*
- *Parity: the transmitted word may or may not be followed by a parity bit which is used to detect any transmission errors that may occur.*
- *Start bit: at rest the line is at logic state 1 to indicate that a word is going to be transmitted, the line passes to low before starting the transfer. This bit makes it possible to synchronise the receiver clock.*
- *Stop bit: after transmission, the line is positioned at rest for 1.2 or 1.5 clock periods depending on the number of stop bits.*
- *Voltage level: a logic "0" is materialised by a voltage comprised between 3 and 25 V, a logic "1" by a voltage comprised between -25V and -3V.*

4.1.2.9. The ADC (Analog-to-Digital Converter)

The ADC is generally of the "successive approximations" type. It has several multiplexed inputs, accessible via the pins of the parallel interface ports. The ADC normally has two registers:

- a data register containing the result of the conversion,
- and a control register that is used to run and monitor the conversion.

4.1.2.10. The Timer

The timer is used to perform the following functions:

- Generation of a periodic signal which may or may not be modulated in pulse width,
- Generation of a calibrated pulse,
- Timeout,
- Event counting.

There are several registers associated with the Timer making it possible to configure the various modes described above.

4.1.2.11. The Watch Dog

Don't worry, this isn't a Pit Bull that guards the PLC!!!!

This device is a system that prevents the PLC from crashing. It makes sure that there is no prolonged execution of the same sequence of instructions.

It also checks the correct operation of the hardware and software.

In fact it is a pre-loadable counter that counts down regularly in step with the clock's frequency.

If no pre-loading is performed before it reaches "0" a Reset is generated to restart the PLC.

You must therefore ensure that this watch dog is regularly pre-loaded by a program when it is activated.

4.1.2.12. The clock signals

The clock signal is used to set the PLC's operating frequency. The PLC includes a Schmitt trigger gate that constitutes an oscillator. To obtain this we place a quartz between the two "OscIn" and "OscOut" pins as shown in the diagram

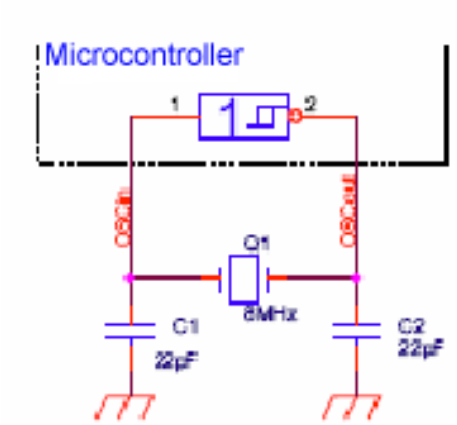


Figure 19: Clock signals

4.1.2.13. The exchange bus

We can see that:

- The exchanges of information between the different parts of the PLC (inputs, outputs, memories, etc.) always pass through the processor,
- Because operation is sequential, there is never more than one part in communication with the processor at a time,

It is therefore possible to use a common path and a common procedure for the exchanges. This path is called a **BUS**.

The bus consists of address lines, data lines and control lines.

The PLC thus adopts the typical structure of a computer.

4.2. HOW IS THE PROGRAM PROCESSED IN THE PLC?

As you have understood, the way the PLC functions depends on a program that we have assigned to it in the CPU board. In the following chapters, we will look at the ***different programming languages***.

In this chapter we are going to focus on the interpretation and management of the program by the PLC.

Processing of the program in the PLC is cyclic and takes place as follows:

1. After powering up the PLC, the processor which represents the PLC's brain, checks whether each input is powered or not. The state of these inputs is recorded in the input image memory (**IIM**). If the input is powered, information 1 or "High" will be recorded. If the input is not powered, information 0 or "Low" will be recorded.

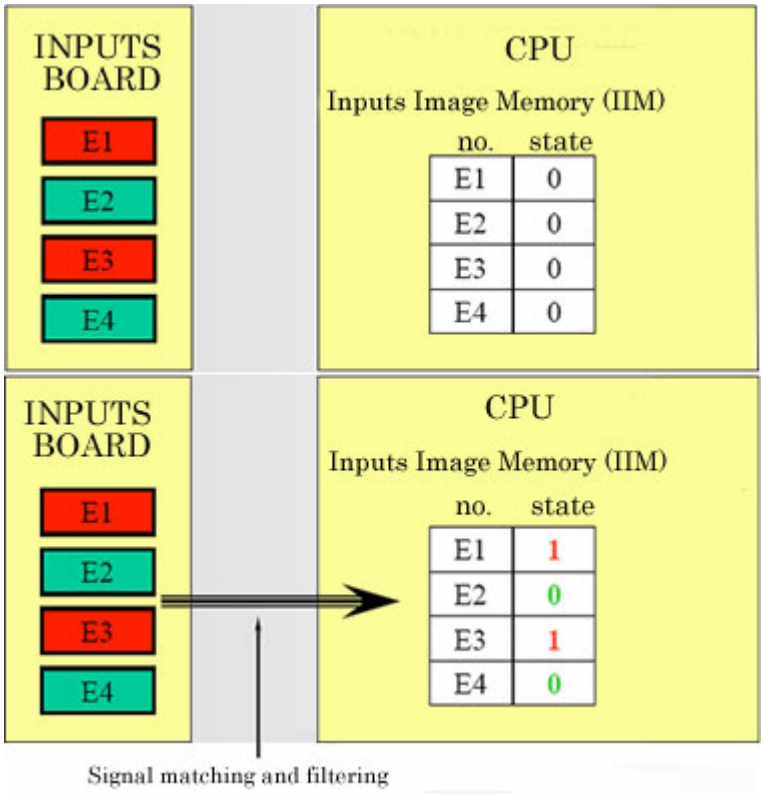
   Example:

   The E1 and E2 input LEDs are on, so their logic state is "**1**":

   Figure 20: Example of program processing (1)

   In the top image the PLC is in the STOP position.

   In the bottom image, the PLC is in the RUN position.



2. This processor runs the program stored in program memory. This consists of a list of instructions and logic operations executed sequentially (various standardised languages can be used). The input information required for this purpose is taken from the input image memory read beforehand and the logic results are written in an output image memory (**OIM**). During program execution, the processor also accesses the counters, timeouts and mnemonics memory zones (data memory).

Example:

In this example, the PLC program is based on the Sequential Function Chart language (we will see how this functions in detail later on in this course):

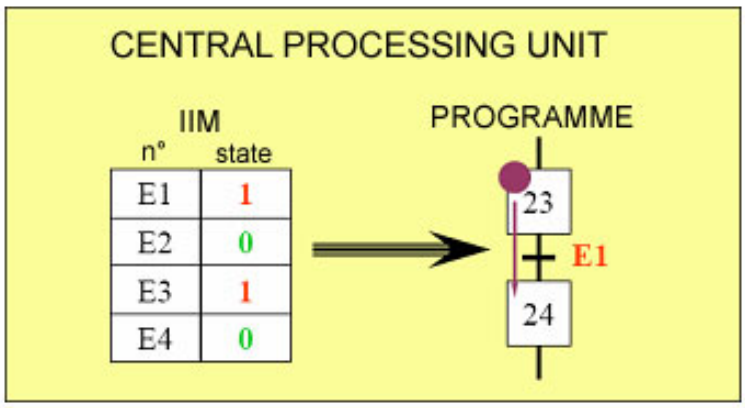


Figure 21: Example of program processing (2)

In this example we can see that once inputs E1 and E2 are active, this makes us go from one step to the next (progression from step 23 to step 24) in the program according to input E1 (see Sequential Function Chart chapter).

3. In the third step, the state is transmitted after execution of the program that uses the OIM on the outputs, activating or deactivating them. Program execution therefore returns to point 1.

Example:

Here is the state of the output internal memory when the program has not yet been polled by the PLC:

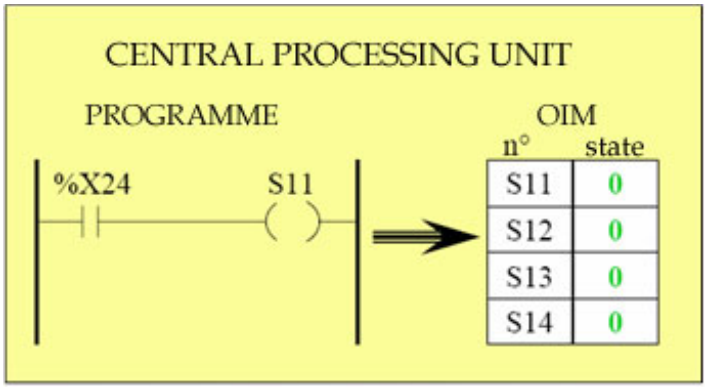
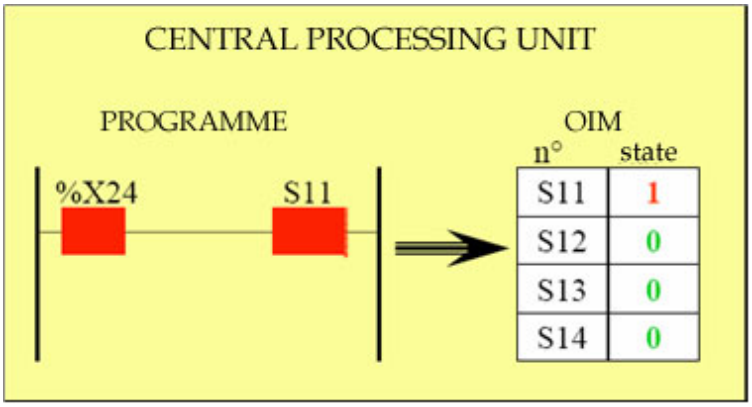


Figure 22: Example of program processing (3)



Once the PLC has effectively polled the program, we can see that it effectively activates step %X24 which passes to 1 and therefore output S11 effectively changes state in the OIM: the execution order is effectively given.

Figure 23: Example of program processing (4)

From this we can deduce with the help of the diagram below, the general functioning of a program in a PLC:

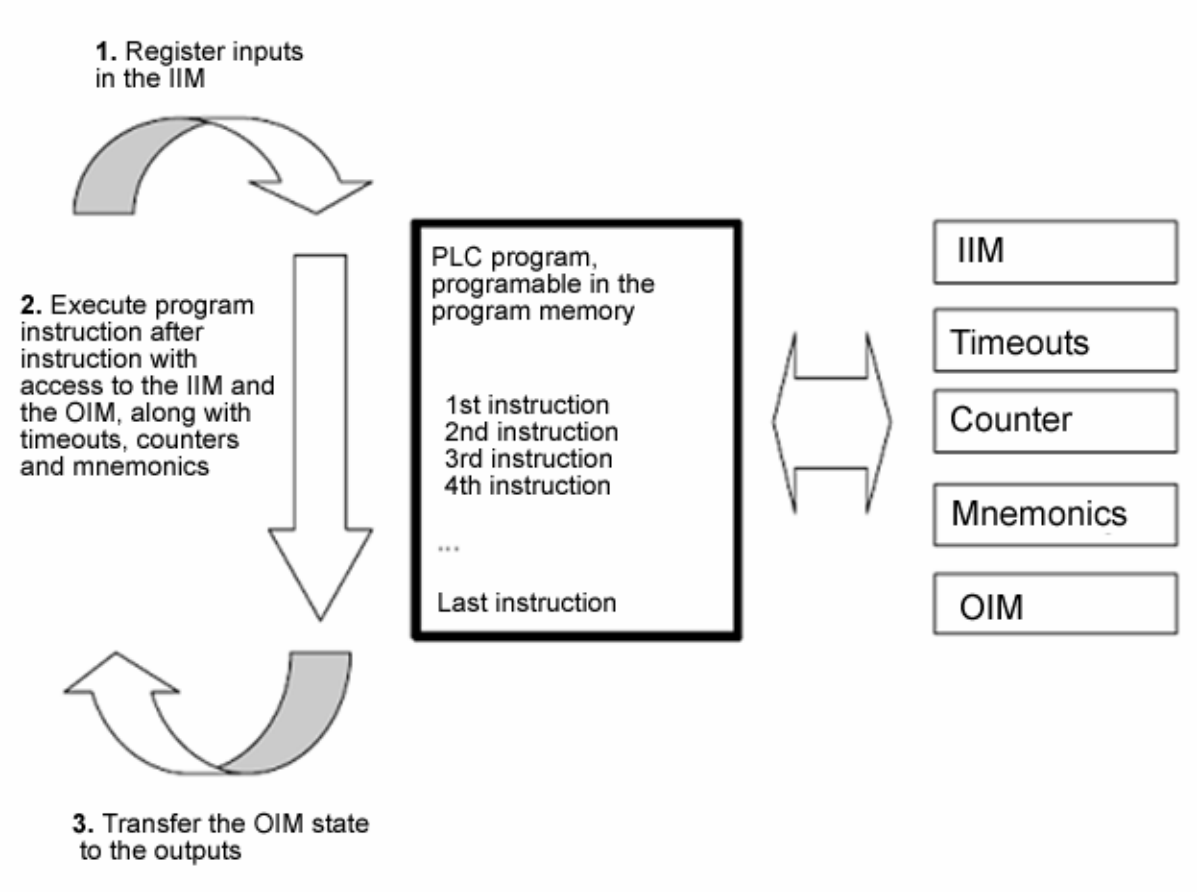


Figure 24: Schematic diagram of program processing by the PLC

Remark:

The time required by the processor to execute the program is called the **cycle time**. The latter depends, among other things, on the number and type of instructions.

To summarise:

The PLC reads all the states of the input boards which consist of 0s or 1s: this is the **machine language.**

Once this reading has been completed, the processor processes the program stored in the program memory and then decodes all the information in assembler (language understood by the microprocessor) and can then give its orders on the output boards.

4.3. HOW THE PLC'S INPUT/OUTPUT BOARDS FUNCTION

There are two sorts of I/O signal:

Digital I/O: these represent logic signals. Besides acquiring information, binary input devices perform the following operations:

- signal shaping (calibration)
- filtering (elimination of disturbances)
- Isolation (galvanic or by light)

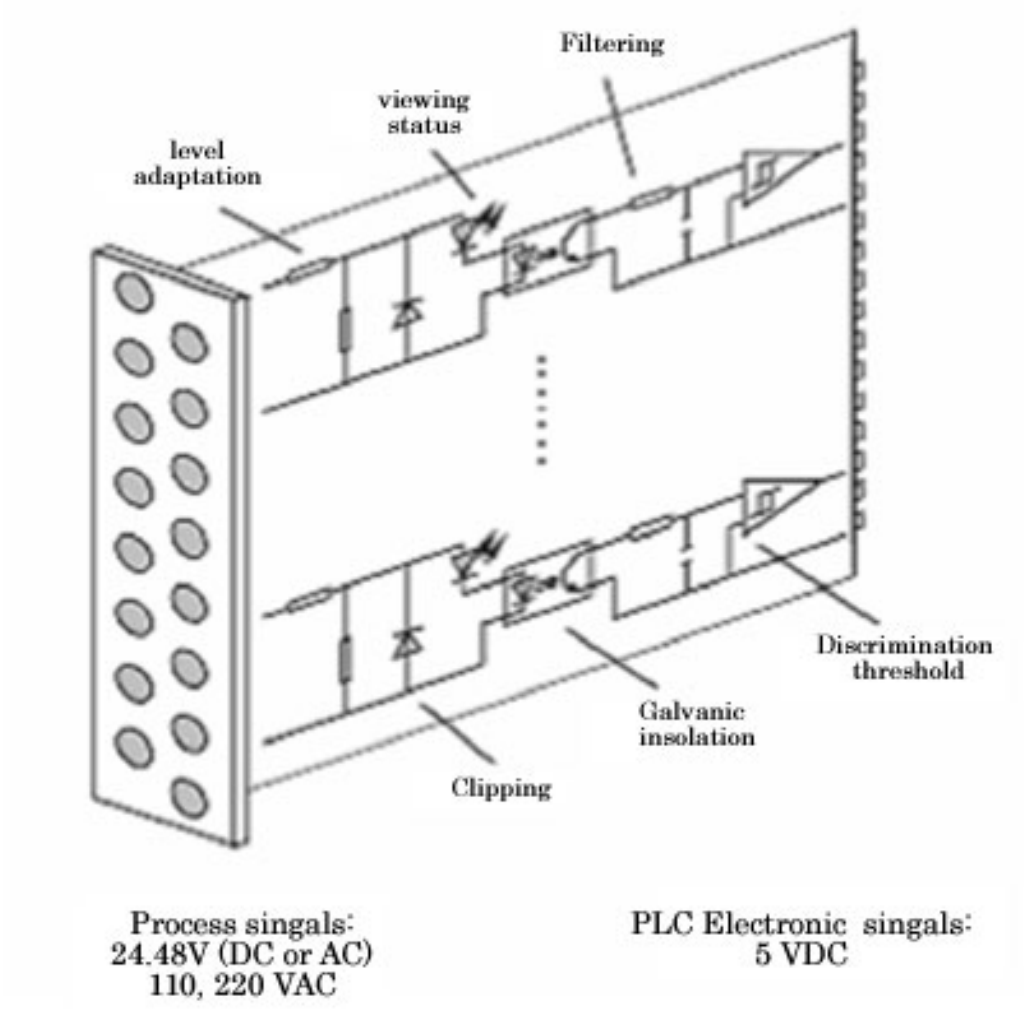


Figure 25: PLC input/output board

The input/output boards therefore create the interface between the signals from the process (to the actuators or from the sensors) and the signals from the PLC's internal bus.

Analog I/O: analog inputs transform a continuously varying analog value into a digital code, generally on 11 bits plus one sign bit.

These inputs have a single Analog-to-Digital Converter (ADC), they are polled one after the other by a multiplexer (MUX). Analog outputs, however, only have one converter per channel.

The analog I/Os are characterised by the amplitude of the signal (VH and VL), by the conversion speed and by the electrical value (current or voltage).

At the level of the inputs, there are three types of analog inputs:

- High: 0-10V, 0-20mA, 4-20mA
- For a thermocouple : 0-20mV, 0-50mV, 0-100mV
- For Pt100 probes: 0-100mV, 0-250mV, 0-400mV

Likewise, the analog inputs can be differentiated according to these intervals:

0-10V, 0-5V, ±10V, ±5V, 0-20mA, 4-20mA.

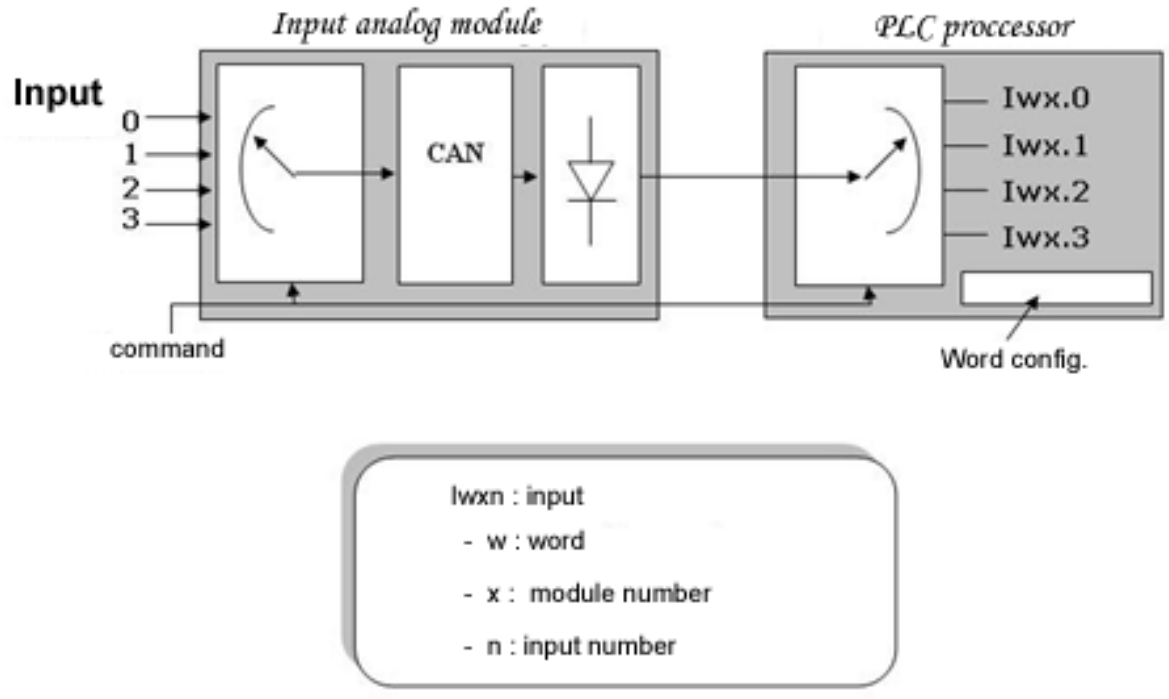


Figure 26: Conversion of analog signals

This module has four channels and a single Analog-to-Digital Converter.

Each channel is processed by means of the associated multiplexer.

The conversion time per channel depends on the converter and therefore on the PLC.

In the case of the TSX 17-20 PLC, this time is 15ms; so the total conversion time is:
4 * 15 = 60ms.

To correctly assess the performance of a PLC on the basis of its critical speed, you will have to calculate its effective conversion time as follows:

$$T_{conv.effect} = T_{cycle} \times (1 + \text{integer part } (T_{conv.total} / T_{cycle}))$$

4.3.1. Logic inputs operation

The PLC receives the information on the process via the signal sensors connected to the PLC's inputs.

These signal sensors may, for example, be sensors that will recognise whether a part being machined is in a given position, or simple switches that can be closed or open.

A distinction must also be made between normally closed contacts that are closed at rest and normally open contacts that are open at rest.

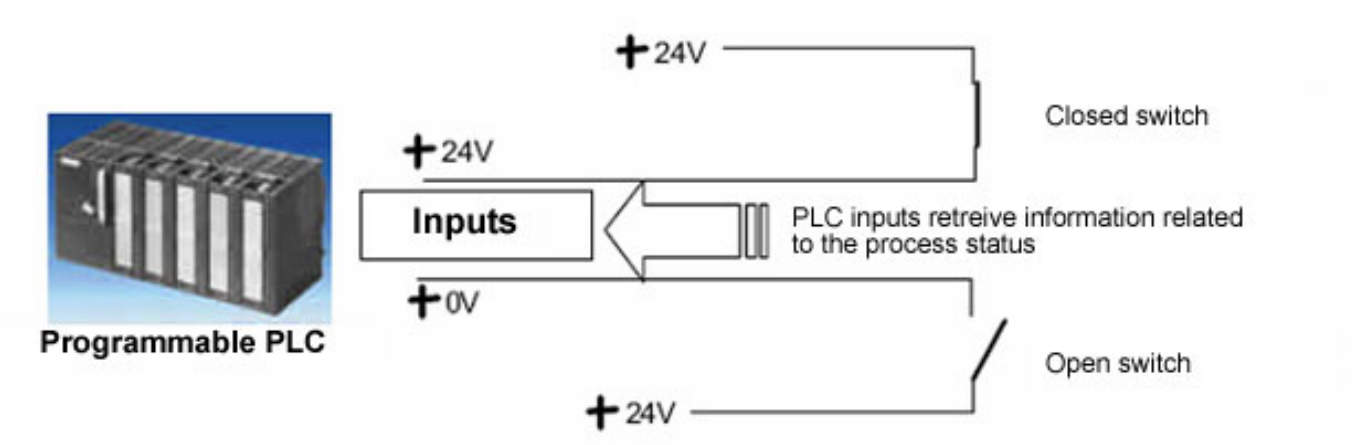


Figure 27: Logic input operation

4.3.2. Logic outputs operation

The PLC commands the process by applying a voltage of 24V, for example, to the actuators via the PLC's connection points called ***outputs***. This makes it possible to activate or deactivate motors, to make solenoid valves rise or fall or to switch lights on or off.

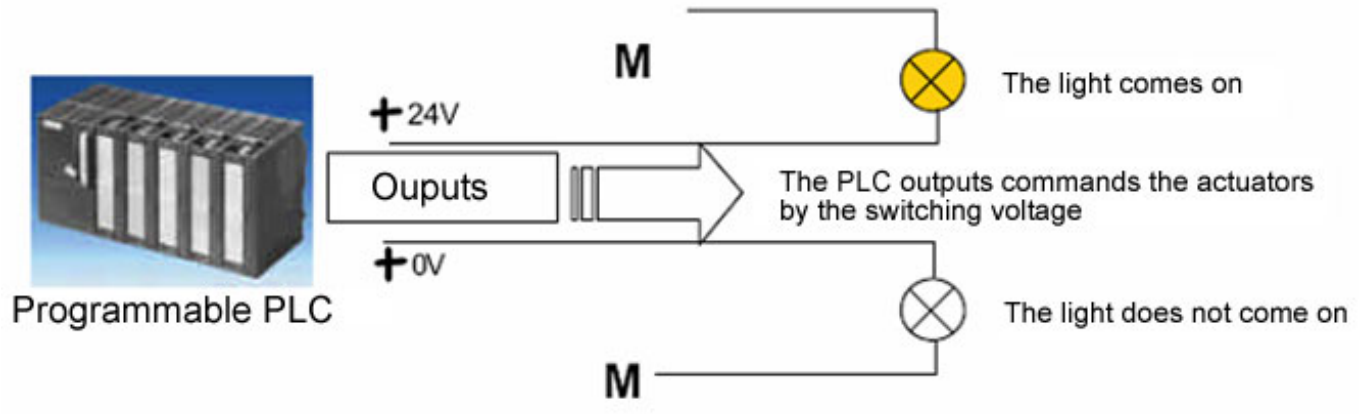


Figure 28: Logic output operation

4.3.3. Passive or active input / output board?

A distinction must be made between an active or passive board.

What does this actually mean?

In the case where your PLC has active input/output boards, it is the PLC itself that delivers the power supply (24VDC, 48VDC, 110VAC, etc.).

In the case where your PLC has passive input/output boards, you will imperatively have to power each of your inputs/outputs by means of an external power supply.

4.3.4. How does the PLC address the input/output signals?

The declaration of an input or output within a program is called **addressing**.

The inputs to and outputs from PLCs are usually grouped together in sets of eight digital inputs or outputs.

This set of eight inputs or outputs is called a **byte**. Each group receives a number called the **byte address**.

In order to enable the addressing of an input or output inside a byte, each byte is divided into eight bits.

The latter are numbered 0 to 7. We thus obtain the **bit's address**.

The PLC represented below has input bytes 0 and 1, and output bytes 4 and 5.

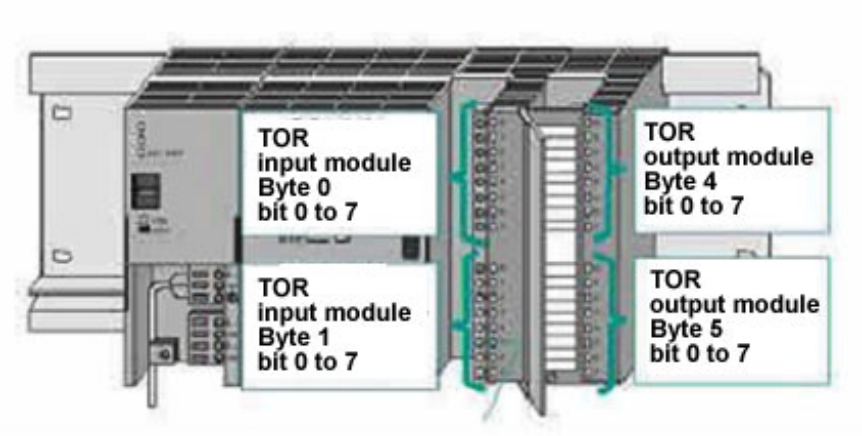


Figure 29: Inputs/outputs address on a PLC

The inputs/outputs addressing is written by default in the PLC and we will see in the software part of this course how each input/output should be filled in correctly with our own comments, this operation is carried out in the CPU board's **mnemonics table**.

Example 1:

To address the fifth input from the top, you must enter the following address:

I 0 . 4

I indicates the type of address (in this case the 'I' means Input: it is an input)
0 indicates the byte's address
4 is the bit's address (this is simply your board's channel number)

The byte's address and the bit's address are always separated by a dot.

You are probably saying to yourself: "I don't understand, we wanted the fifth input and we have written 4 for the bit's address": this is normal because numbering starts at 0.

Example 2 :

To address the last output, you must enter the following address:

O 5 . 7

O indicates the type of address (in this case the 'O' means Output: it is an output)
5 indicates the byte's address
7 is the bit's address

5. THE SOFTWARE PART

5.1. INTRODUCTION

5.1.1. History of programming languages

Like for any computer, the program is constructed using a language. This language is a set of instructions that apply to variables. Due to the specific function of industrial machine and process control, a history, not to say a culture has developed around specific languages.

On the one hand, computer languages of course. These languages have been developed according to two tendencies: the PLC's application and processor. Must they meet the needs of mathematicians, software specialists or manufacturers of electronic microcircuits?

It must not be forgotten that, in the end, it is a machine code that will be executed.

On the other hand, the automation practitioners: electrical, mechanical, instrumentation technicians? Here, we can distinguish between two main cultural families due to their history.

In North America, through the specialisation of tasks, it would seem that it is the electricians who have the monopoly of the language. Based on drawings and control diagrams, it is the ladder diagrams language that has been generalised.

Figure 30: Examples of programming consoles

It is true that there have been some attempts by "IT specialists" to impose a so-called mnemonics language, but it wasn't popular with the electricians and instrumentation specialists. It is however the language closest to the assembler that the PLC's processor uses.

We used to use small, simple programming consoles that make it possible to see and/or enter the instructions one by one. These consoles can still be used from time to time for simple modifications.

In short, it is the **ladder** that had won, so now it was left to the IT specialists to develop editors (that draw symbols: contacts and windings on a screen), and compilers (that transform those symbols into instructions… mnemonics that the PLC can execute) so that the electricians and instrumentation specialists can find their way.

These PLC programming software applications were either dedicated microcomputers (figure below) or software applications installed on a PC (such as PL7 pro, STEP 7, etc.).

Figure 31: Examples of microcomputers dedicated to programming

In the rest of the world, the specialisation of the automation experts is not so clearly cut. Or at least it was becoming a specialisation in its own right. Furthermore, a technology came between the relay and the PLC: static logic.

Based on the concept of wired gate logic, it uses the symbols of gates but with a technology suited to the reality of the industrial world: 24VDC voltage, shielding and decoupling…. All the suppliers have invested in this, with greater or lesser success.

Square D developed its Norpack (nothing but NOR gates), Allen Bradley, Westinghouse, Siemens (Siematic), Télémécanique (Téléstatic) and Merlin Gérin with its Silimog.

Just as the relay logic gave ladder language, gate logic has given the function block language. We found the same programming software applications: a symbols (the blocks) editor and a compiler to transform the symbols into instructions that can be run by the PLC.

All these languages were, in the end, poorly suited to describing complex operations that often include sequences. They were made for combinatorial logic; even if sequential instructions had been developed such as the cyclic programmer (Drum), but this was very limited.

A solution was found by the mechanics because they found that it was difficult to describe the operation of a machine or of a process using words (and syntax).

This gave something like this:

Put the part in the vice, tighten, when it has been tightened start the drill's motor, lower the drill quickly to the edge of the part, then lower slowly, raise the drill to clear if the hole is deep, and drilled to the correct depth. Raise the drill, and once it has been fully raised, loosen the part and remove it…

All you needed to know then was how you put the part in the vice, how you tightened it, when did you know it had been tightened, how do you start the motor, how do you lower the drill quickly, not so quickly…. How do you know you are close to the part? That the hole is deep? That you have drilled to the right depth…. And what happens if the motor doesn't work, if the drill bit breaks?

Towards the end of the seventies, the AFCET committee examined the question of how to describe an automatic system and created a language for describing an automatic system: the **Sequential Function Chart** was born.

It was standardised under the reference CEI60848 in 1988 and revised in 2002.

It was only much later on that Sequential Function Chart became a programming language for PLCs. However, as the description language existed, it just had to be translated into a control language.

This period of Sequential Function Chart-ladder coexistence seems to be lasting, to such an extent that we now find automatic translation software applications that use the ladder instruction sets of PLCs.

Here we must make an important remark: Sequential Function Chart as a programming language (therefore more than a description language) has had significant implications on the operating principle of PLCs. The cycle that we use for conventional logic is modified quite significantly and is simplified when you use the **Sequential Function Chart language** in a PLC.

When the PLC arrives at the Sequential Function Chart section, it only looks at the validated transitions: the transitions for which all the immediately preceding steps are active. If a step must be passed because the transition is true, it activates the immediately following steps and deactivates all the previous steps.

To conclude this introduction, in the face of the challenges of globalisation, it is necessary that a machine or process designed in Italy, for example, should function in Japan: this required standardisation.

5.1.2. The IEC 61131-3 standard

This is what the IEC did starting in 1993, in cooperation with the PLCopen organisation which together took an active part in writing the IEC61131 standard.

This standard defines the program design procedures for ensuring transportability: achieve independence from the specific features of a given type of hardware (PLC brand and model).

This ever-evolving standard defines how to implement programs on the basis of this standard. It makes the distinction between the common elements and **two families of languages*: textual languages*** and ***graphic languages***.

The textual languages are:

- The *Instruction List (IL*), this is a mnemonic language used in part by the IT specialists,
- The *Structured Text Language (ST),* which is used very little these days.

The graphic languages are:

- The *Ladder Diagram (LD*),
- The *Sequential Function Chart (SFC*),
- The *Function Block Diagram (FDB*)

5.2. LADDER LANGUAGE (LD)

5.2.1. Introduction

The ladder diagram (LD) is a graphic programming language. This is a visual language that is very easy to use and whose graphic representation is close to that of electrical diagrams. Associated with the Function Block Diagram (FBD) the ladder becomes a complete programming language.

A Ladder Diagram is made up of several networks. As you can see in the figure below, each network has a left-hand feed line, a right-hand feed line and branches connecting the inputs on the left to the outputs on the right.

Each network is evaluated from left to right.

All of the networks are evaluated from the top downwards.

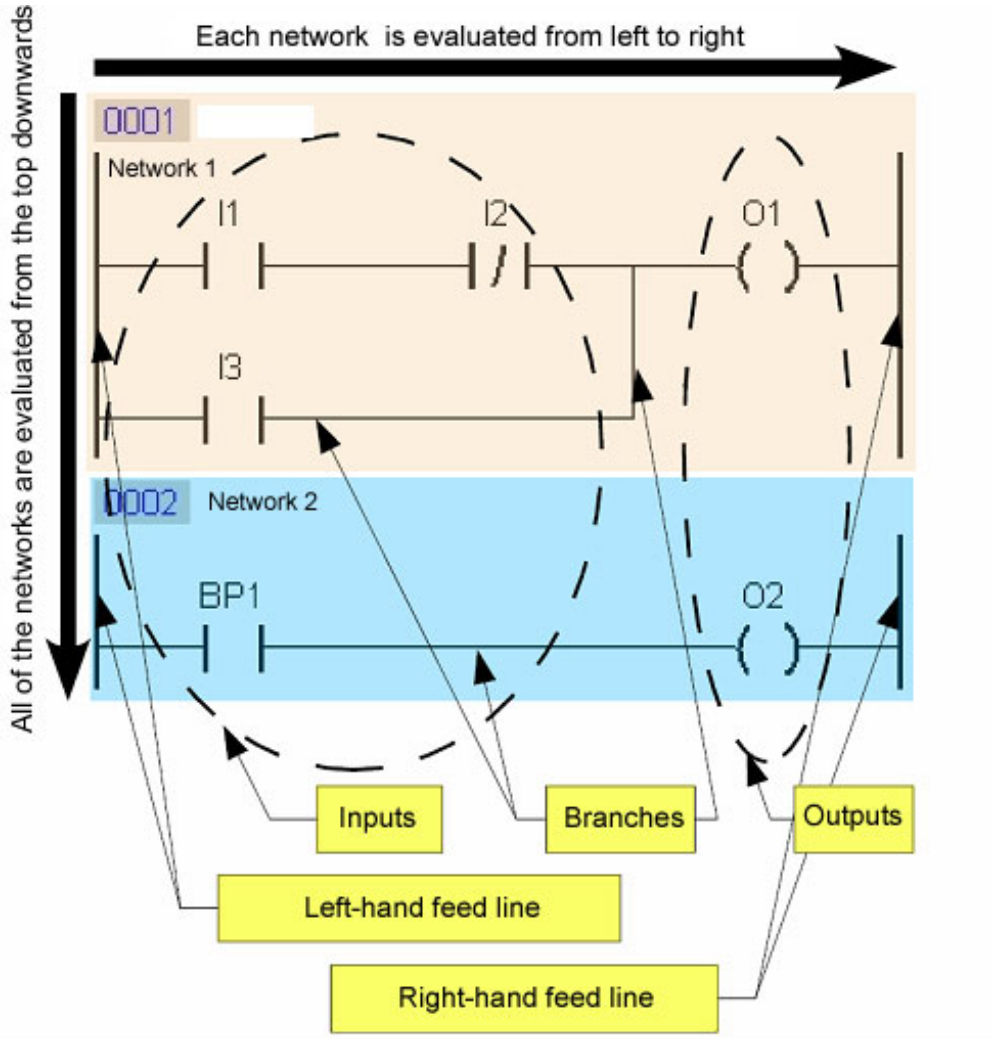


Figure 32: Ladder Diagram operating principle

We make a distinction between the instructions on bits and the instructions on words. We also make a distinction between the test instructions and the write or assign instructions.

Example:

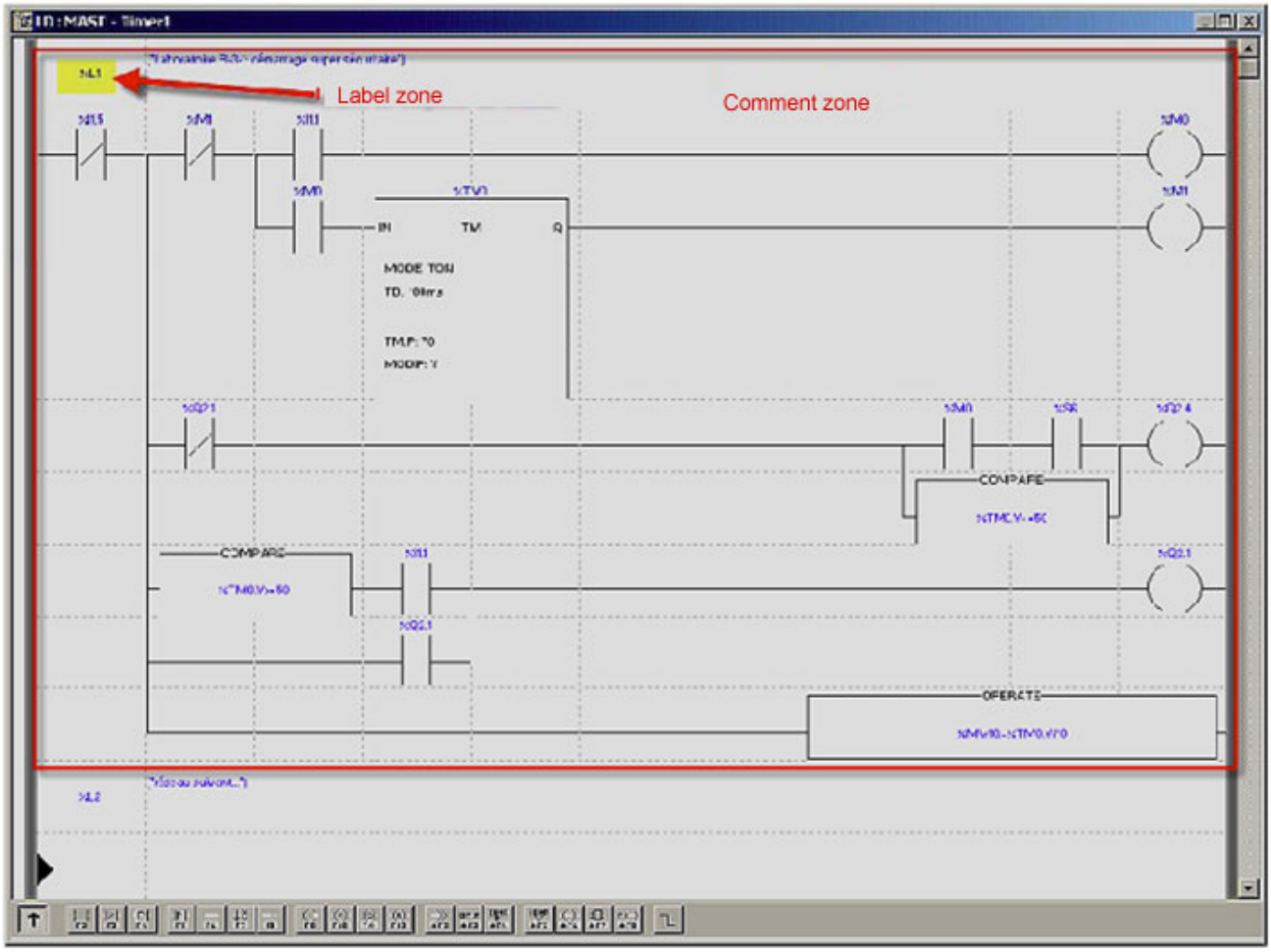


Figure 33: Ladder contact diagram in Télémécanique's PL7 software:

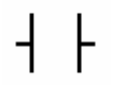
A network is a zone comprised between two tables (labels). In the figure above you can see network %L1 and part of network %L2. Each network may have comments. A label must be unique in a program.

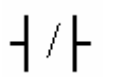
The graphic zone is situated below the label and comment zones. It is divided into cells.

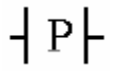
There are 11 columns of cells and you can have up to a maximum of 16 rows in a network. The elements can occupy one or more cells.

5.2.2. Instructions on bits

5.2.2.1. The tests (contacts)


The contact closes when the bit is at 1


The contact closes when the bit is at 0



Rising edge: the contact closes during 1 scan when the bit passes from 0 to 1

Falling edge: the contact closes during 1 scan when the bit passes from 1 to 0

The AND (in series) and OR (in parallel) logic operations are realised by graphic connection symbols.

5.2.2.2. Series contacts

By associating contacts in series you can make logic ANDs.

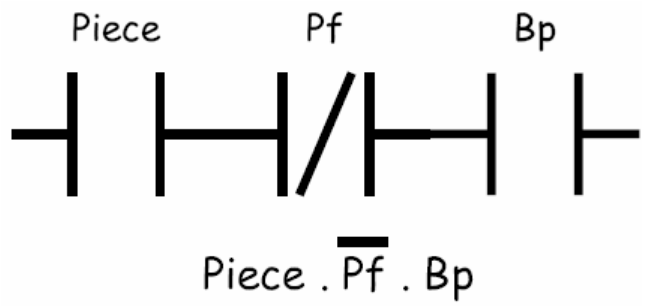
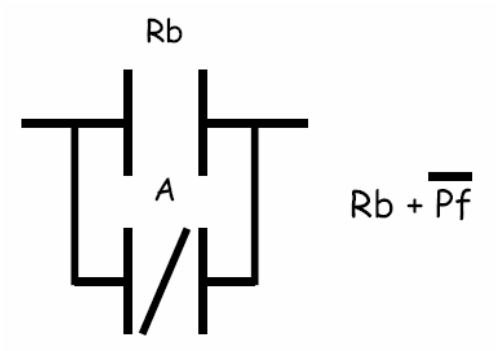


Figure 34 : Example of series contacts

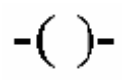
5.2.2.3. Parallel contacts



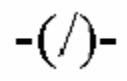
By associating contacts in parallel you can make logic ORs.

Figure 35 : Example of parallel contacts

5.2.2.4. Writing (coils)



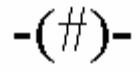
The coil is activated, the bit passes to 1 as long as the preceding logic is at 1 and the coil is deactivated, the bit passes to 0 if the logic is at 0.



The coil is activated, the bit passes to 1 as long as the preceding logic is at 0 and the coil is deactivated, the bit passes to 0 if the logic is at 1.

The coil is activated, the bit passes to 1 as long as the preceding logic is at 1 and remains like that even if the logic returns to 0.

The coil is deactivated, the bit passes to 0 if the preceding logic is at 1 and remains like that even if the logic returns to 0.



Instruction reserved for a reception from the Sequential Function Chart. If the associated transition is validated, the transition is made.

5.2.2.5. Network jumps and comments

Jump

The "Jump Label" conditional instruction makes it possible to skip, that is to say exclude, the evaluation of all the networks placed between the "Jump Label" and the network "Label"»

RETURN

The "Return" conditional instruction makes it possible to skip, that is to say exclude, the evaluation of all the networks placed after the "Return"

Comments

It is possible to insert comments at the beginning of each network.

Example:

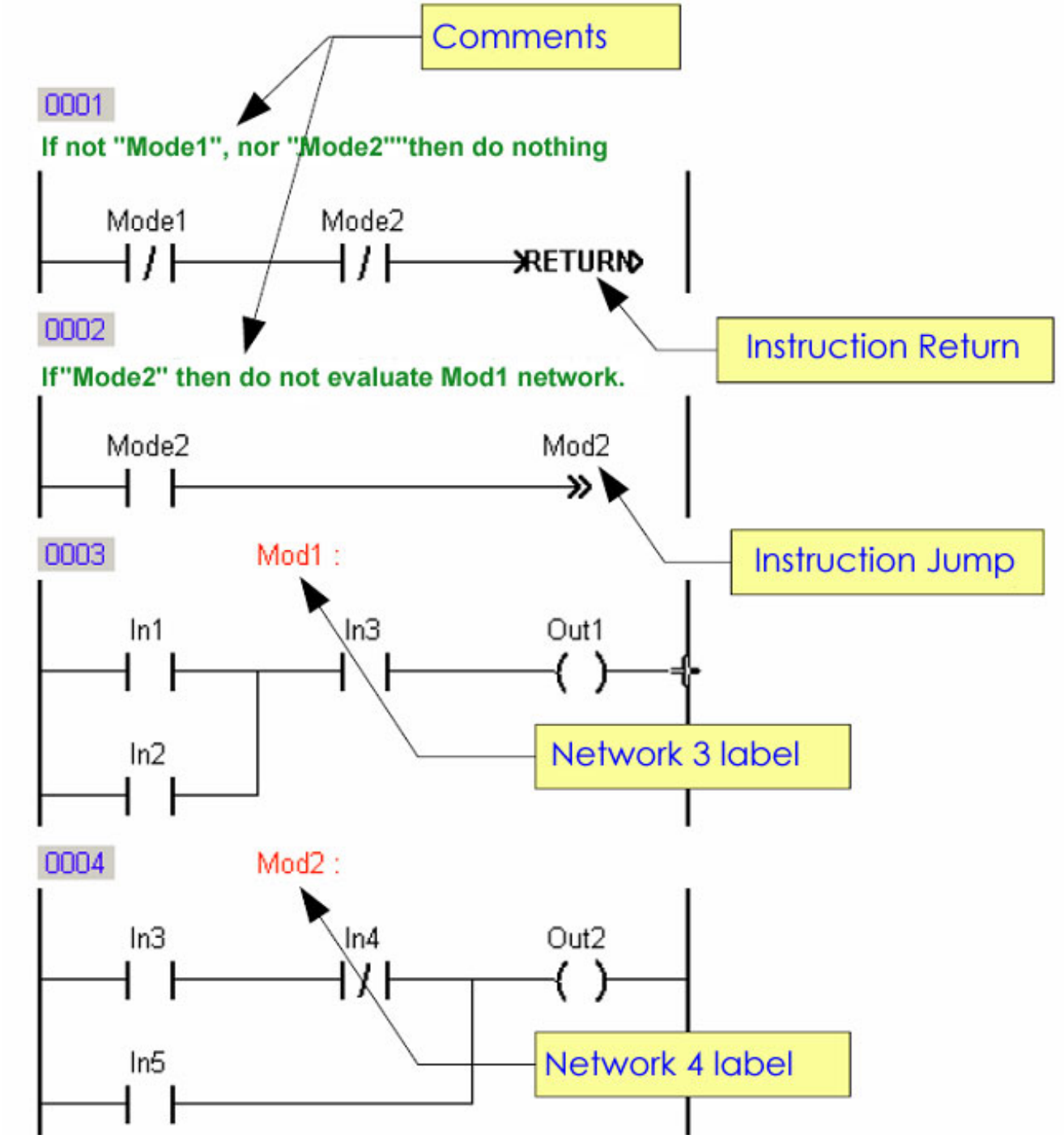


Figure 36 : Example of jumps and comments

5.2.3. Instruction on words

5.2.3.1. The tests (comparisons)

There are two types of comparison:

- The **horizontal comparison** (the most frequently used)

  The two variables are separated by numerical test signs: =, <, >, =<, =>, <> (different).

  The second variable may be a numerical value or an operation.

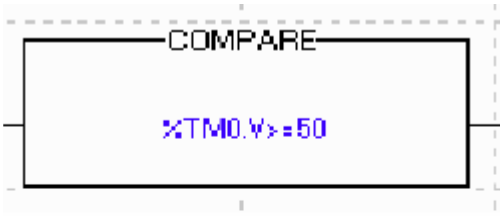


Figure 37: Horizontal comparison

- The **vertical comparison**:

  The four possibilities can be used and combined.

  For example, %M10 or %M11 will be true if %M10 is greater than or equal to %KW10.

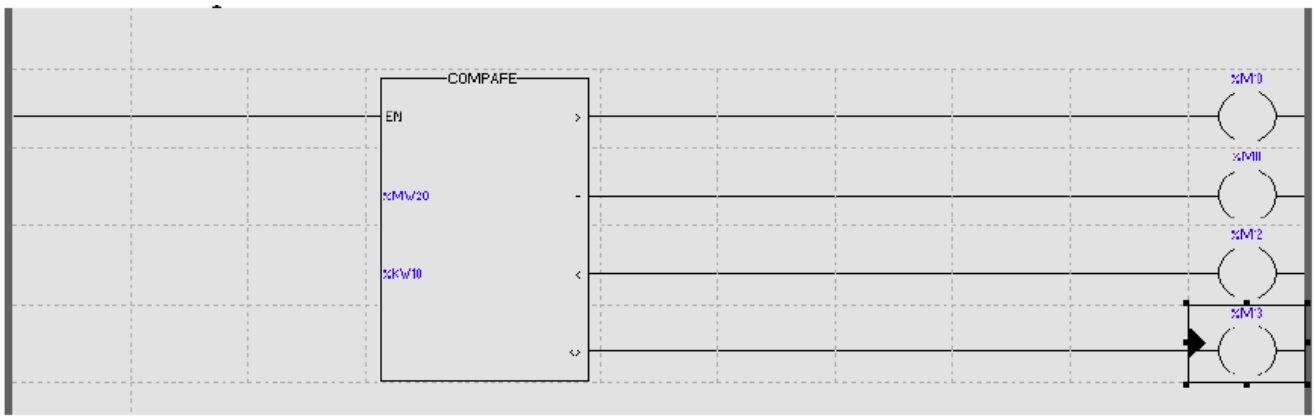


Figure 38: Vertical comparison

5.2.3.2. The assignments (OPERATE)

Simply put, this makes it possible to transfer a value in a variable.

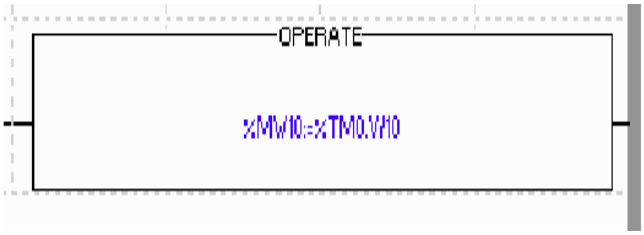


Figure 39: Assignments (OPERATE)

In the example above, %MW10 will be equal to %TM0.V divided by 10.

This also makes it possible to perform more complex calculations and functions: conversions, indexation, communication.

5.2.3.3. The combinatorial functions

We are going to program the following logic equations in ladder language:

Out1 = In1 • (In2 + In3)

Out2 = In1 • $\overline{In4}$ + In5

Out3 = $\overline{In5}$ • In4 • (In2 + In3)

Here is the program in Ladder Language for our logic equations:

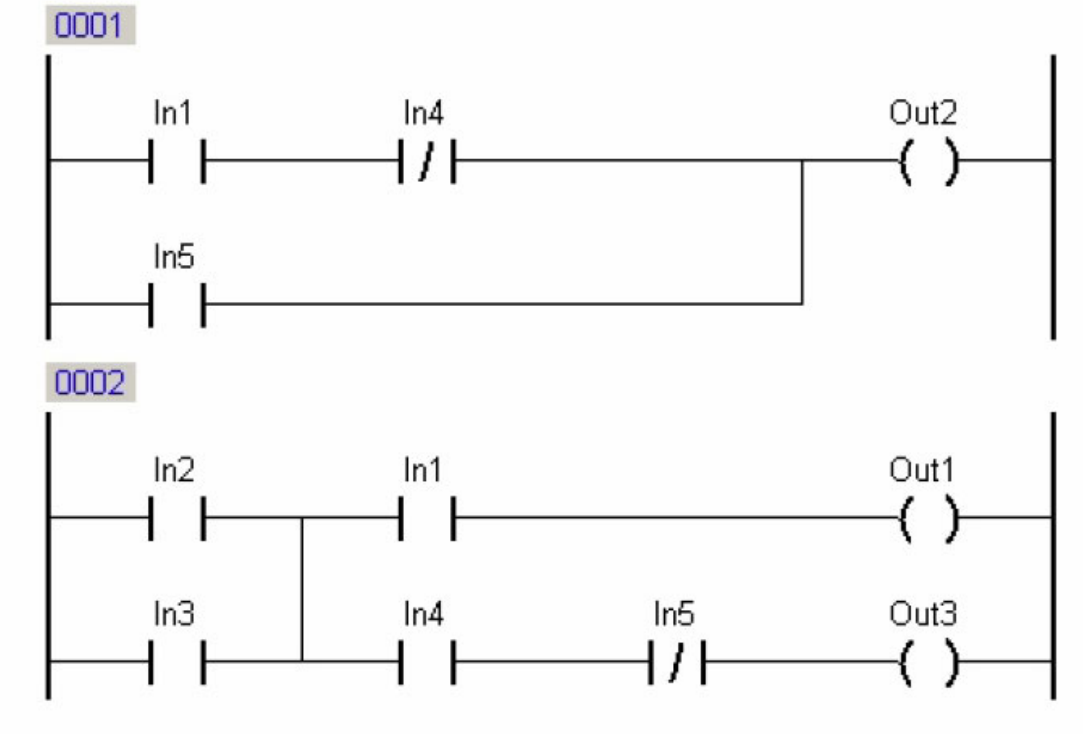


Figure 40: Logic equations in ladder language

5.2.3.4. The bistable and pulse functions

The Manual – Off – Automatic 3-position selector switch delivers 2 items of Boolean information according to the truth table below:

	Manual	Off	Automatic
Manual	1	0	0
Automatic	0	0	1

Let's take a small example:

If the selector is in the Manual position then the conveyor belt, Conv output, functions if the BPConv input is at 1.

If the selector is in the Automatic position, then a rising edge on the PEnt input starts up the conveyor belt and a falling edge on the PSor outputs stops the conveyor belt.

We are now going to write a brief expression of requirements:

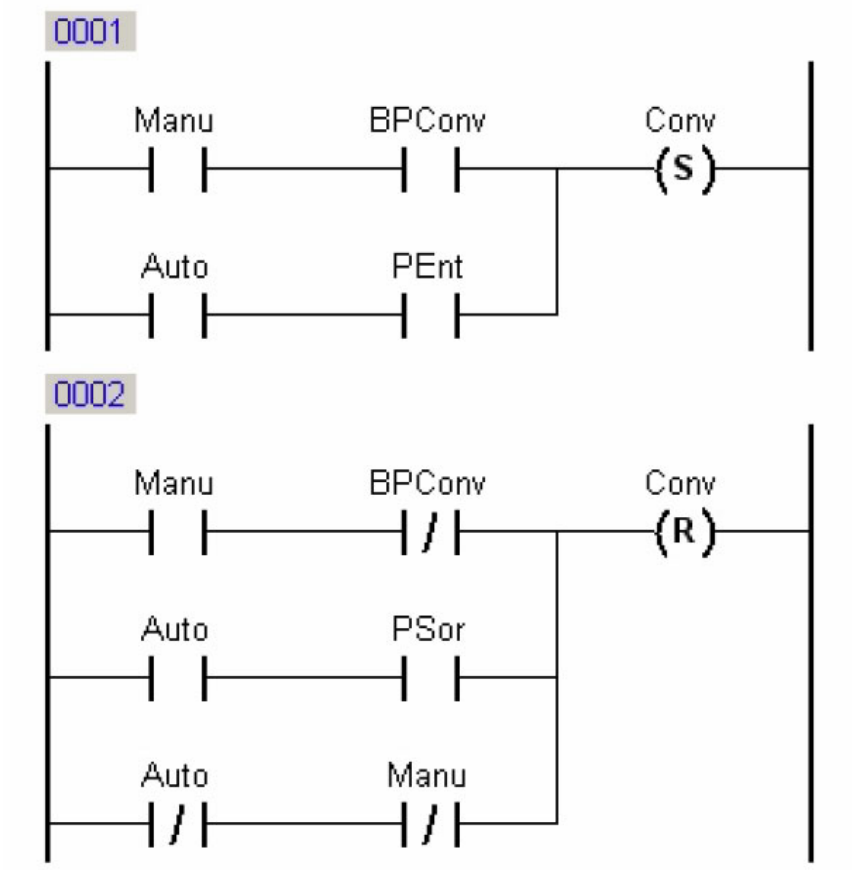


Figure 41: Example of selector switch programming

Using function blocks in FBD language

We are going to take the same example as above and combine the ladder with the FBD, and this gives:

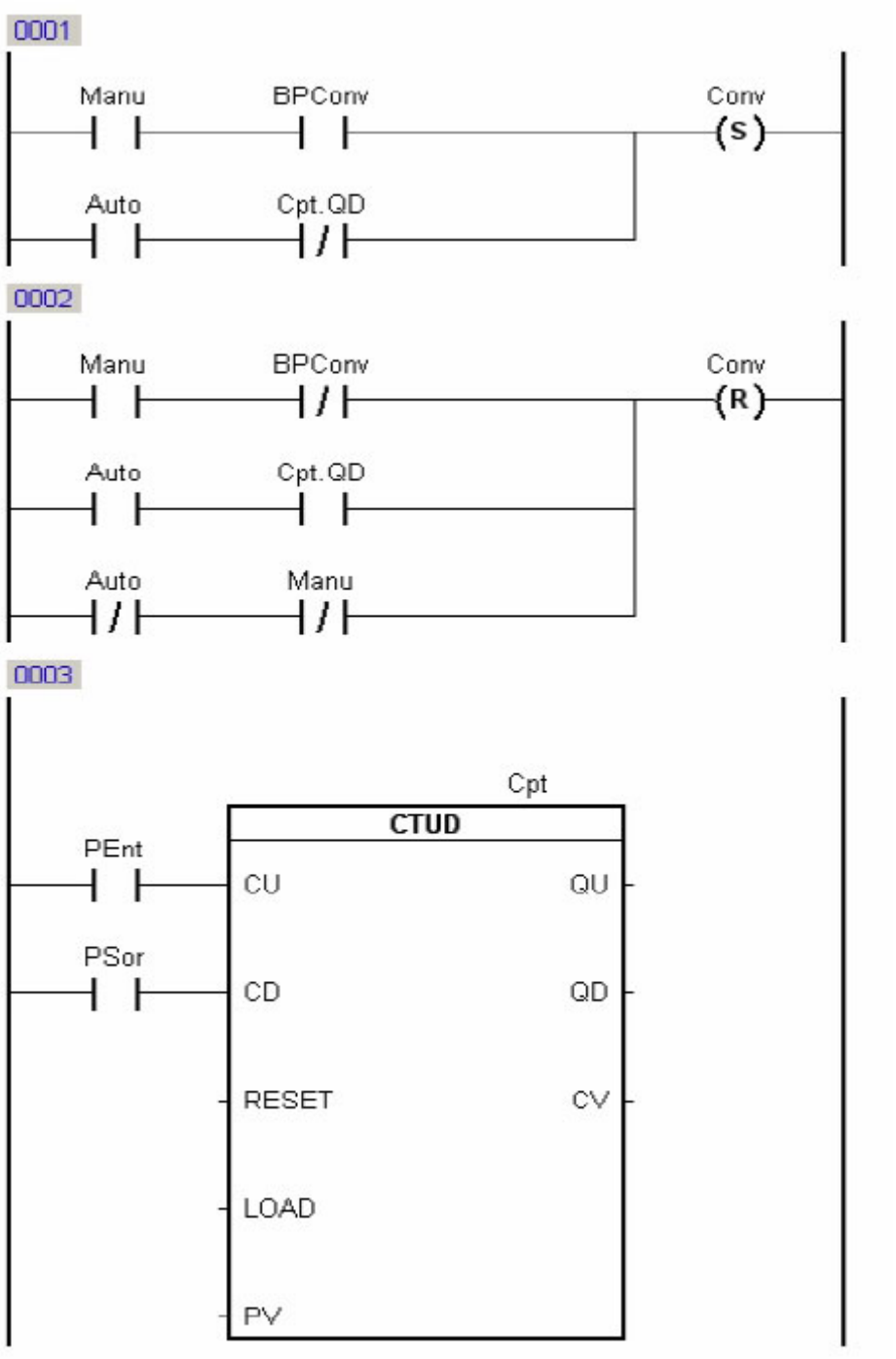
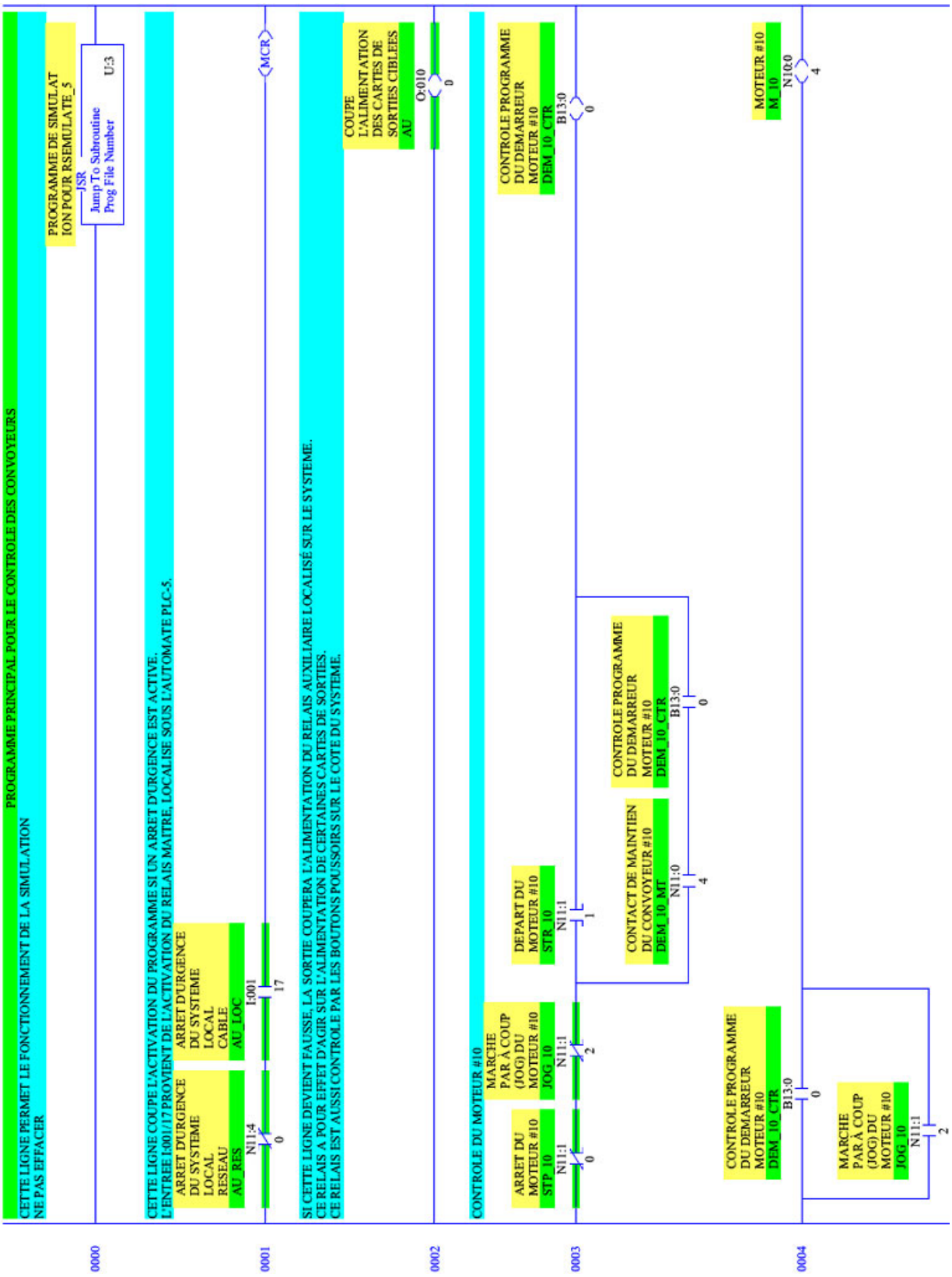
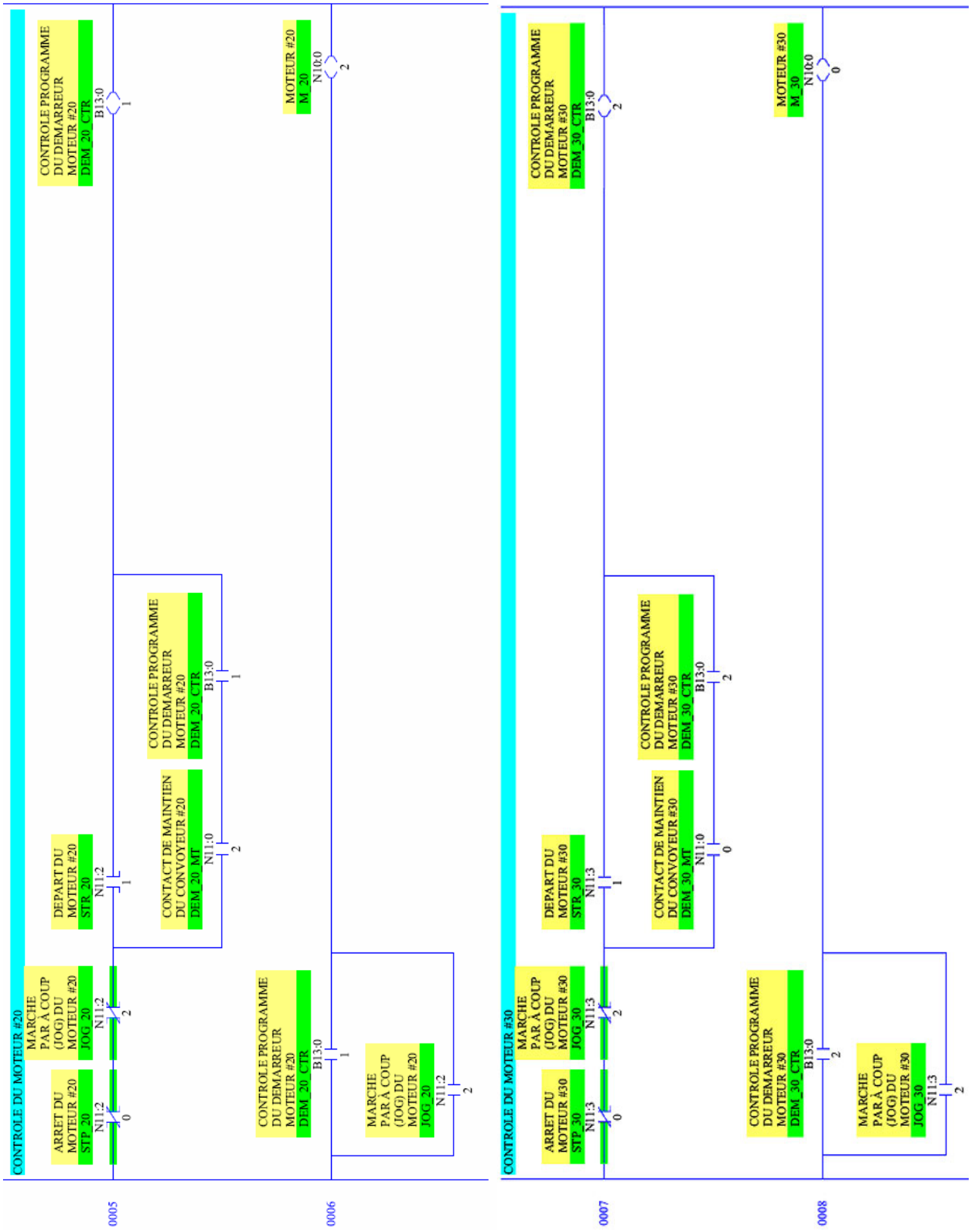
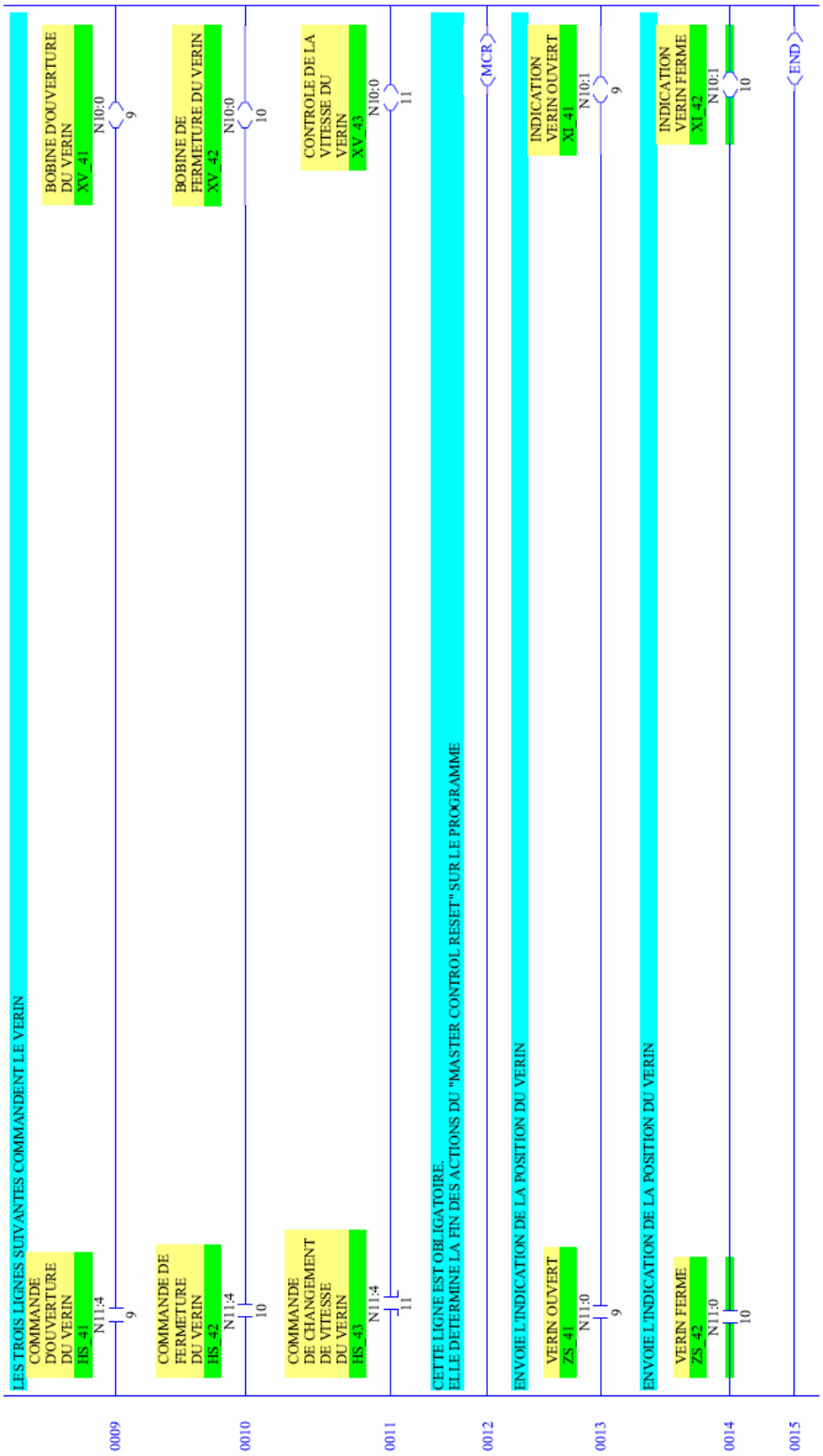


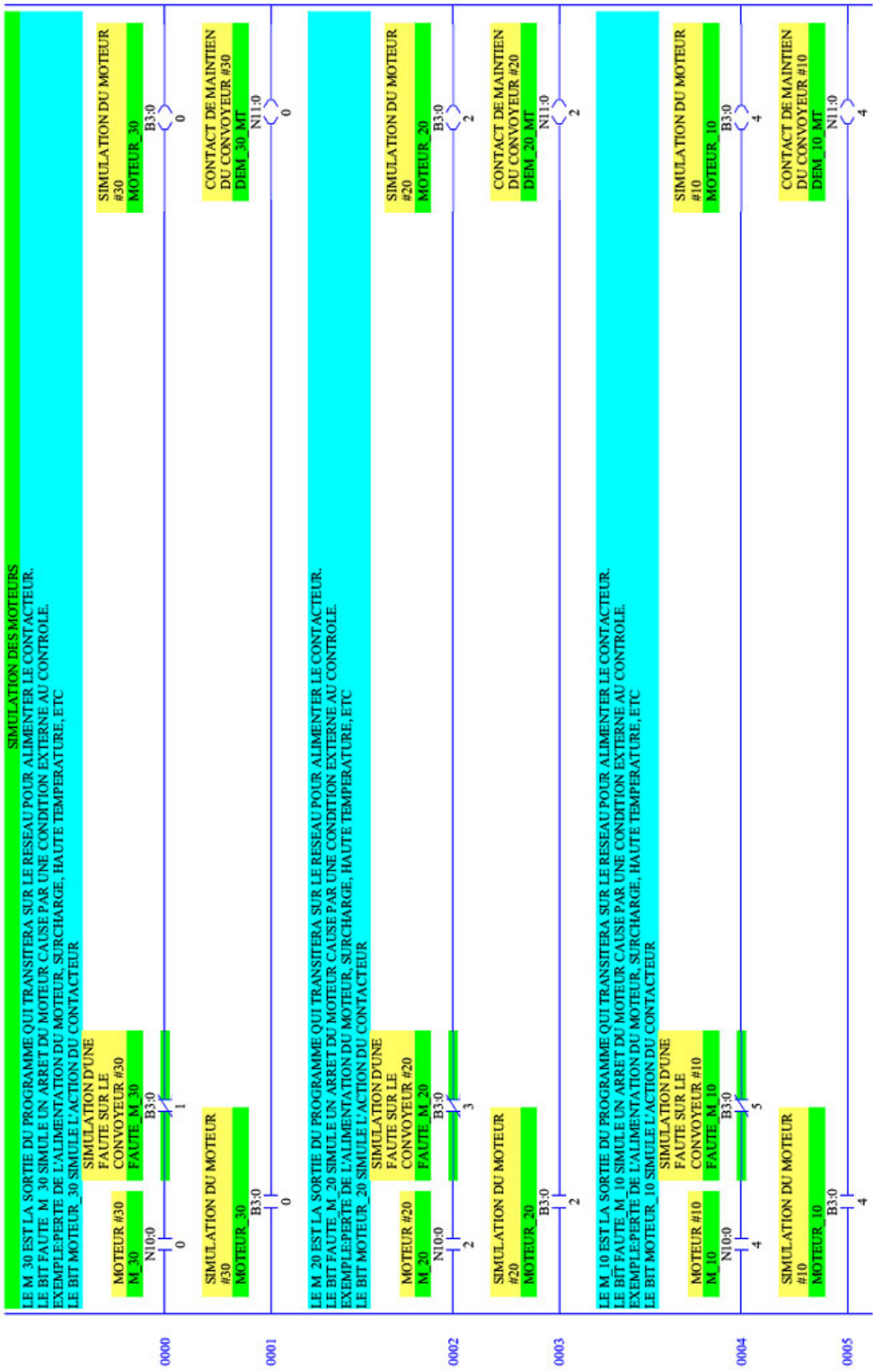
Figure 42: Using Function Blocks in FBD language

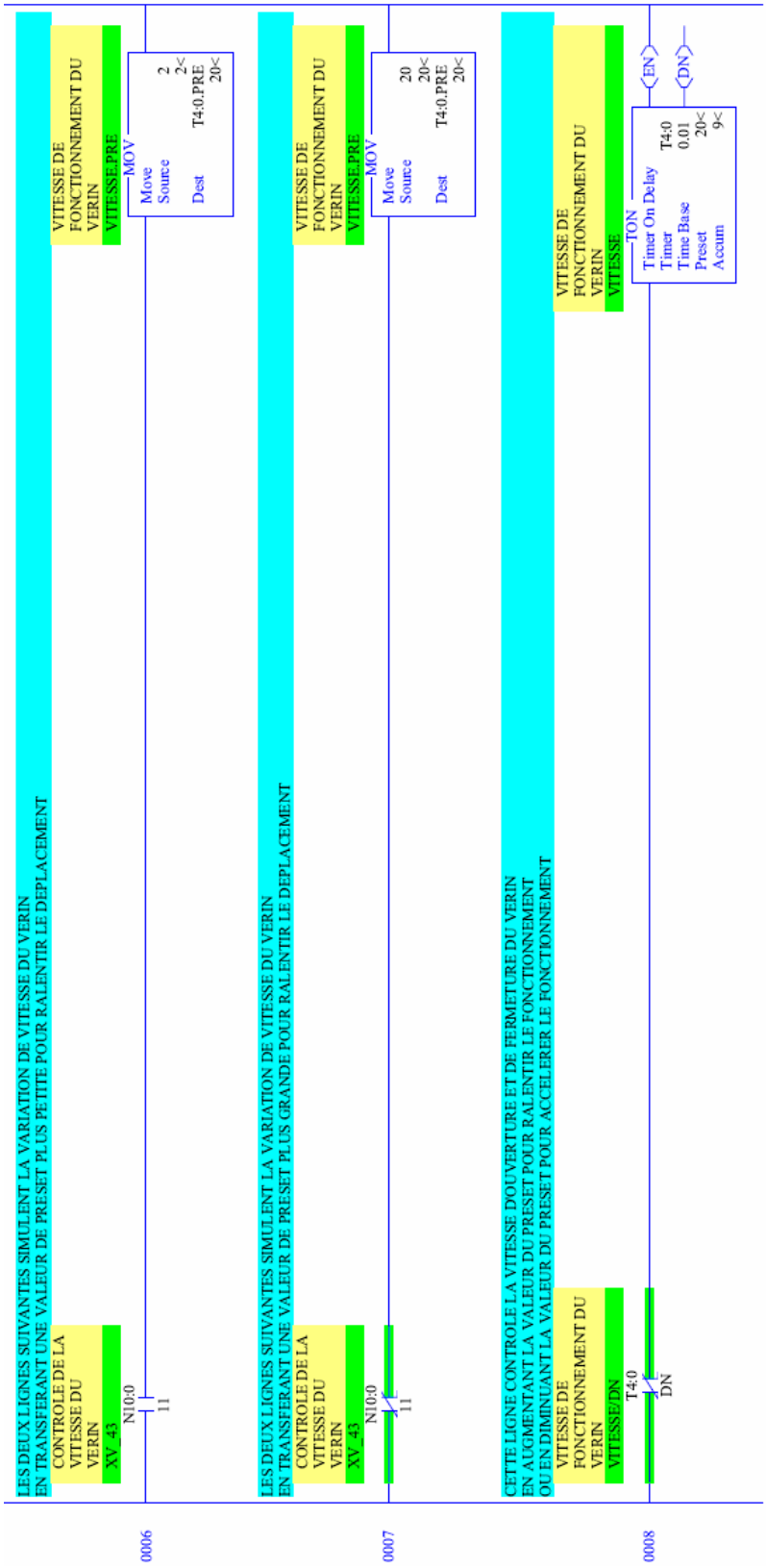
5.2.4. Example of a complete program

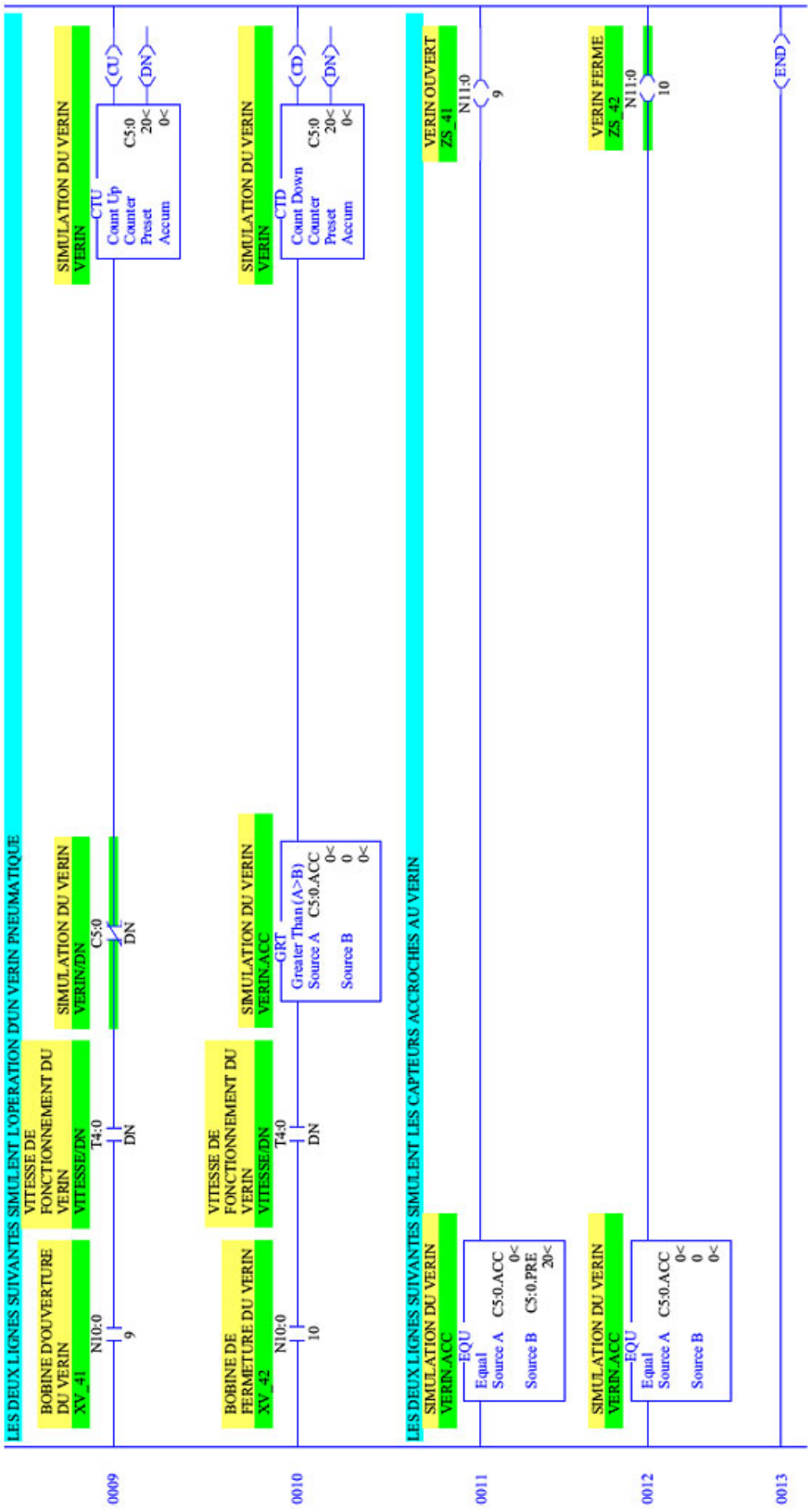



CONTROLE DU MOTEUR #20
ARRET DU MOTEUR #20
STP_20
N11:2
0
MARCHE PAR A COUP (JOG) DU MOTEUR #20
JOG_20
N11:2
2
DEPART DU MOTEUR #20
STR_20
N11:2
1
CONTROLE PROGRAMME DU DEMARREUR MOTEUR #20
DEM_20_CTR
B13:0
1
CONTACT DE MAINTIEN DU CONVOYEUR #20
DEM_20_MT
N11:0
2
CONTROLE PROGRAMME DU DEMARREUR MOTEUR #20
DEM_20_CTR
B13:0
1
MARCHE PAR A COUP (JOG) DU MOTEUR #20
JOG_20
N11:2
2
MOTEUR #20
M_20
N10:0
2
0005
0006
CONTROLE DU MOTEUR #30
ARRET DU MOTEUR #30
STP_30
N11:3
0
MARCHE PAR A COUP (JOG) DU MOTEUR #30
JOG_30
N11:3
2
DEPART DU MOTEUR #30
STR_30
N11:3
1
CONTROLE PROGRAMME DU DEMARREUR MOTEUR #30
DEM_30_CTR
B13:0
2
CONTACT DE MAINTIEN DU CONVOYEUR #30
DEM_30_MT
N11:0
0
CONTROLE PROGRAMME DU DEMARREUR MOTEUR #30
DEM_30_CTR
B13:0
2
MARCHE PAR A COUP (JOG) DU MOTEUR #30
JOG_30
N11:3
2
MOTEUR #30
M_30
N10:0
0
0007
0008


LES TROIS LIGNES SUIVANTES COMMANDENT LE VERIN
COMMANDE D'OUVERTURE DU VERIN
HS_41
N11:4
9
BOBINE D'OUVERTURE DU VERIN
XV_41
N10:0
9
COMMANDE DE FERMETURE DU VERIN
HS_42
N11:4
10
BOBINE DE FERMETURE DU VERIN
XV_42
N10:0
10
COMMANDE DE CHANGEMENT DE VITESSE DU VERIN
HS_43
N11:4
11
CONTROLE DE LA VITESSE DU VERIN
XV_43
N10:0
11
CETTE LIGNE EST OBLIGATOIRE.
ELLE DETERMINE LA FIN DES ACTIONS DU "MASTER CONTROL RESET" SUR LE PROGRAMME
MCR
ENVOIE L'INDICATION DE LA POSITION DU VERIN
VERIN OUVERT
ZS_41
N11:0
9
INDICATION VERIN OUVERT
XI_41
N10:1
9
ENVOIE L'INDICATION DE LA POSITION DU VERIN
VERIN FERME
ZS_42
N11:0
10
INDICATION VERIN FERME
XI_42
N10:1
10
END
0009
0010
0011
0012
0013
0014
0015


SIMULATION DES MOTEURS
LE M_30 EST LA SORTIE DU PROGRAMME QUI TRANSITERA SUR LE RESEAU POUR ALIMENTER LE CONTACTEUR.
LE BIT FAUTE_M_30 SIMULE UN ARRET DU MOTEUR CAUSE PAR UNE CONDITION EXTERNE AU CONTROLE.
EXEMPLE:PERTE DE L'ALIMENTATION DU MOTEUR, SURCHARGE, HAUTE TEMPERATURE, ETC
LE BIT MOTEUR_30 SIMULE L'ACTION DU CONTACTEUR
SIMULATION D'UNE FAUTE SUR LE CONVOYEUR #30
MOTEUR #30
M_30
N10:0
0
FAUTE_M_30
B3:0
1
SIMULATION DU MOTEUR #30
MOTEUR_30
B3:0
0
SIMULATION DU MOTEUR #30
MOTEUR_30
B3:0
0
CONTACT DE MAINTIEN DU CONVOYEUR #30
DEM_30_MT
N11:0
0
0000
LE M_20 EST LA SORTIE DU PROGRAMME QUI TRANSITERA SUR LE RESEAU POUR ALIMENTER LE CONTACTEUR.
LE BIT FAUTE_M_20 SIMULE UN ARRET DU MOTEUR CAUSE PAR UNE CONDITION EXTERNE AU CONTROLE.
EXEMPLE:PERTE DE L'ALIMENTATION DU MOTEUR, SURCHARGE, HAUTE TEMPERATURE, ETC
LE BIT MOTEUR_20 SIMULE L'ACTION DU CONTACTEUR
SIMULATION D'UNE FAUTE SUR LE CONVOYEUR #20
MOTEUR #20
M_20
N10:0
2
FAUTE_M_20
B3:0
3
SIMULATION DU MOTEUR #20
MOTEUR_20
B3:0
2
SIMULATION DU MOTEUR #20
MOTEUR_20
B3:0
2
CONTACT DE MAINTIEN DU CONVOYEUR #20
DEM_20_MT
N11:0
2
0002
LE M_10 EST LA SORTIE DU PROGRAMME QUI TRANSITERA SUR LE RESEAU POUR ALIMENTER LE CONTACTEUR.
LE BIT FAUTE_M_10 SIMULE UN ARRET DU MOTEUR CAUSE PAR UNE CONDITION EXTERNE AU CONTROLE.
EXEMPLE:PERTE DE L'ALIMENTATION DU MOTEUR, SURCHARGE, HAUTE TEMPERATURE, ETC
LE BIT MOTEUR_10 SIMULE L'ACTION DU CONTACTEUR
SIMULATION D'UNE FAUTE SUR LE CONVOYEUR #10
MOTEUR #10
M_10
N10:0
4
FAUTE_M_10
B3:0
5
SIMULATION DU MOTEUR #10
MOTEUR_10
B3:0
4
SIMULATION DU MOTEUR #10
MOTEUR_10
B3:0
4
CONTACT DE MAINTIEN DU CONVOYEUR #10
DEM_10_MT
N11:0
4
0004


LES DEUX LIGNES SUIVANTES SIMULENT LA VARIATION DE VITESSE DU VERIN
EN TRANSFERANT UNE VALEUR DE PRESET PLUS PETITE POUR RALENTIR LE DEPLACEMENT
CONTROLE DE LA VITESSE DU VERIN
XV_43
N10:0
11
VITESSE DE FONCTIONNEMENT DU VERIN
VITESSE PRE
MOV
Move
Source
2
Dest
T4:0.PRE
0006
LES DEUX LIGNES SUIVANTES SIMULENT LA VARIATION DE VITESSE DU VERIN
EN TRANSFERANT UNE VALEUR DE PRESET PLUS GRANDE POUR RALENTIR LE DEPLACEMENT
CONTROLE DE LA VITESSE DU VERIN
XV_43
N10:0
11
VITESSE DE FONCTIONNEMENT DU VERIN
VITESSE PRE
MOV
Move
Source
20
Dest
T4:0.PRE
0007
CETTE LIGNE CONTROLE LA VITESSE D'OUVERTURE ET DE FERMETURE DU VERIN
EN AUGMENTANT LA VALEUR DU PRESET POUR RALENTIR LE FONCTIONNEMENT
OU EN DIMINUANT LA VALEUR DU PRESET POUR ACCELERER LE FONCTIONNEMENT
VITESSE DE FONCTIONNEMENT DU VERIN
VITESSE/DN
T4:0
DN
VITESSE DE FONCTIONNEMENT DU VERIN
VITESSE
TON
Timer On Delay
Timer
T4:0
Time Base
0.01
Preset
20
Accum
9
EN
DN
0008


LES DEUX LIGNES SUIVANTES SIMULENT L'OPERATION D'UN VERIN PNEUMATIQUE
BOBINE D'OUVERTURE DU VERIN
XV_41
N10:0
9
VITESSE DE FONCTIONNEMENT DU VERIN
VITESSE/DN
T4:0
DN
SIMULATION DU VERIN
VERIN/DN
C5:0
DN
SIMULATION DU VERIN
VERIN
CTU
Count Up
Counter C5:0
Preset 20
Accum 0
CU
DN
BOBINE DE FERMETURE DU VERIN
XV_42
N10:0
10
VITESSE DE FONCTIONNEMENT DU VERIN
VITESSE/DN
T4:0
DN
SIMULATION DU VERIN
VERIN.ACC
GRT
Greater Than (A>B)
Source A C5:0.ACC 0
Source B 0
SIMULATION DU VERIN
VERIN
CTD
Count Down
Counter C5:0
Preset 20
Accum 0
CD
DN
LES DEUX LIGNES SUIVANTES SIMULENT LES CAPTEURS ACCROCHES AU VERIN
SIMULATION DU VERIN
VERIN.ACC
EQU
Equal
Source A C5:0.ACC 0
Source B C5:0.PRE 20
VERIN OUVERT
ZS_41
N11:0
9
SIMULATION DU VERIN
VERIN.ACC
EQU
Equal
Source A C5:0.ACC 0
Source B 0
VERIN FERME
ZS_42
N11:0
10
END
0009
0010
0011
0012
0013

5.3. FUNCTION BLOCK DIAGRAM (FBD) LANGUAGE

All the functions and all the function blocks in the various libraries can be inserted in the ladder networks.

The functions and function blocks are program organisation units common to all the programming languages.

5.3.1. The Functions

A function is a program organisation unit which, when it is executed, provides exactly a data element (value, structure, table, etc.). It is an operand in an expression.

A function does not contain any information concerning its internal state, that is to say the triggering of a function with the same input parameters always gives the same output value.

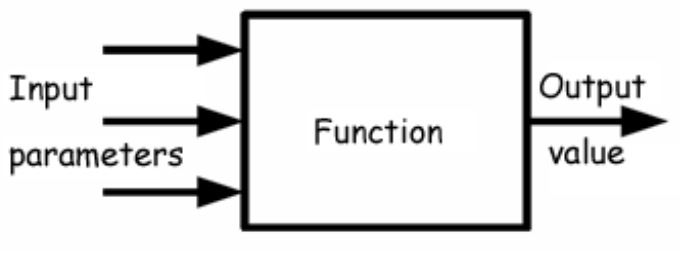


Figure 43: Functions

5.3.2. The standard functions

5.3.2.1. The conversion functions

The conversion functions make it possible to change a value's type. For example the BOOL_TO_INT function transforms a Boolean value into a 16-bit integer value.

5.3.2.2. The numerical functions

Function	Symbol	I/O type	Description
ABS		*ANY_NUM*	*Absolute value*
SQRT		*ANY_REAL*	*Square root*
LN		*ANY_REAL*	*Neperian logarithm*
LOG		*ANY_REAL*	*Decimal logarithm*
EXP		*ANY_REAL*	*Exponential*
SIN		*ANY_REAL*	*Sine (angle expressed in radians)*

Function	Symbol	I/O type	Description
COS		*ANY_REAL*	*Cosine (angle expressed in radians)*
TAN		*ANY_REAL*	*Tangent (angle expressed in radians)*
ASIN		*ANY_REAL*	*Principal arc sine (result in radians)*
ACOS		*ANY_REAL*	*Principal arc cosine (result in radians)*
ATAN		*ANY_REAL*	*Principal arc tangent (result in radians)*
ADD	+	*ANY*	*Addition*
SUB	-	*ANY*	*Subtraction*
MUL	*	*ANY_NUM*	*Multiplication*
DIV	/	*ANY_NUM*	*Division*
MOD		*ANY_INT*	*Remainder of the integer division*
EXPT	**	$ANY\_REAL^{ANY\_NUM}$	*Power* ($IN1^{IN2}$)
MOVE	:=	*ANY*	*Assignment*

Figure 44 : The numerical functions in FBD language

5.3.2.3. The selection and limitation functions

Function	I/O type	Description
SHL	ANY_INT	N-bit shift to the left
SHR	ANY_INT	N-bit shift to the right
ROL	ANY_INT	N-bit rotation to the left
ROR	ANY_INT	N-bit rotation to the right
AND	BOOL	AND logic operator
OR	BOOL	OR logic operator
NOT	BOOL	NOT logic operator
XOR	BOOL	Exclusive OR logic operator

Figure 45 : The selection and limitation functions in FBD language

5.3.2.4. The comparison functions

Function	Symbol	I/O type	Description
GT	>	ANY - BOOL	Strictly greater than
GE	>=	ANY – BOOL	Greater than or equal to
EQ	=	ANY – BOOL	Equal to
LT	<	ANY – BOOL	Strictly lower than
LE	<=	ANY – BOOL	Lower than or equal to
NE	<>	ANY - BOOL	Not equal to or Different

Figure 46 : The comparison functions in FBD language

5.3.2.5. The functions on character strings

Function	I/O type	Description
LEN	ANY_STRING – ANY_INT	Returns the length of the character string
LEFT	ANY_STRING – ANY_INT	Returns the character string made up of the first N characters
RIGHT	ANY_STRING – ANY_INT	Returns the character string made up of the last N characters
MID	ANY_STRING – ANY_INT	Returns the character string from the N^{th} and until the P^{th} character
CONCAT	ANY_STRING	Returns the concatenation of the character strings
INSERT	ANY_STRING – ANY_INT	Inserts a character string from the N^{th} character
DELETE	ANY_STRING – ANY_INT	Deletes N characters from the P^{th} character
REPLACE	ANY_STRING – ANY_INT	Replaces N characters from the P^{th} character
FIND	ANY_STRING – ANY_INT	Returns the first position of a character string in another character string

Figure 47 : The functions on character strings in FBD language

5.3.2.6. The date and time functions

Function	Symb.	IN1	IN2	OUT
ADD	+	TIME	TIME	TIME
ADD	+	TIME_OF_DAY	TIME	TIME_OF_DAY
ADD	+	DATE_AND_TIME	TIME	DATE_AND_TIME
ADD_TIME	+	TIME	TIME	TIME
ADD_TOD_TIME	+	TIME_OF_DAY	TIME	TIME_OF_DAY
ADD_DT_TIME	+	DATE_AND_TIME	TIME	DATE_AND_TIME
SUB	-	TIME	TIME	TIME
SUB	-	TIME_OF_DAY	TIME	TIME_OF_DAY
SUB	-	TIME_OF_DAY	TIME_OF_DAY	TIME
SUB	-	DATE_AND_TIME	TIME	DATE_AND_TIME
SUB	-	DATE_AND_TIME	DATE_AND_TIME	TIME
SUB	-	DATE	DATE	TIME
SUB_TIME	-	TIME	TIME	TIME
SUB_TOD_TIME	-	TIME_OF_DAY	TIME	TIME_OF_DAY
SUB_TOD_TOD	-	TIME_OF_DAY	TIME_OF_DAY	TIME
SUB_DT_TIME	-	DATE_AND_TIME	TIME	DATE_AND_TIME
SUB_DT_DT	-	DATE_AND_TIME	DATE_AND_TIME	TIME
SUB_DATE_DATE	-	DATE	DATE	TIME
MUL	*	TIME	ANY_NUM	TIME
MULTIME	*	TIME	ANY_NUM	TIME
DIV	/	TIME	ANY_NUM	TIME
DIVTIME	/	TIME	ANY_NUM	TIME
CONCAT_DATE_TOD		DATE	TIME_OF_DAY	DATE_AND_TIME

Figure 48 : The date and time functions in FBD language

5.3.3. The function blocks

5.3.3.1. Flip-flops

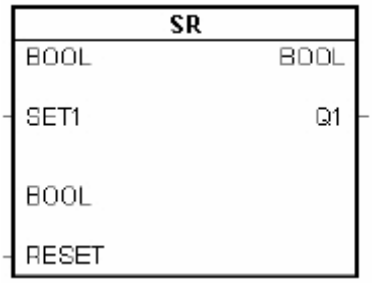
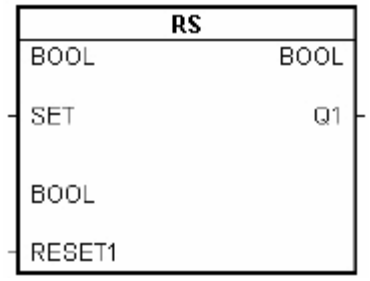
Set – Reset flip-flop with dominant Set	
	If Boolean input SET1 passes to 1 then Boolean output Q passes to 1 If Boolean input RESET passes to 1 and Boolean input SET1 is at 0 then Boolean output Q passes to 0
Set – Reset flip-flop with dominant Reset	
	If Boolean input SET passes to 1 and Boolean input RESET1 is at 0 then Boolean output Q passes to 1 If Boolean input RESET1 passes to 1 then Boolean output Q passes to 0

Figure 49 : The flip-flop function blocks in FBD language

5.3.3.2. Edge detection

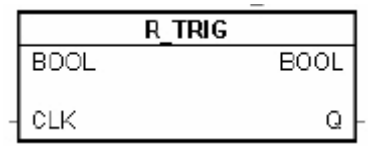
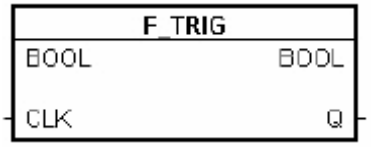
Rising edge detection	
	A rising edge on Boolean input CLK generates a one-cycle pulse on output Q
Falling edge detection	
	A falling edge on Boolean input CLK generates a one-cycle pulse on Boolean output Q

Figure 50 : Edge detection function blocks in FBD language

5.3.3.3. Counters

Inputs	Outputs	Description
Up counter		
VAR_INPUT CU : BOOL ; R : BOOL ; PV : ANY_INT END_VAR	VAR_OUTPUT Q : BOOL ; CV : ANY_INT ; END_VAR	If R Then CV = 0 Else If CU and CV < Pvmax then CV = CV + 1 End If If CV >= PV Then Q = True Else Q = False End If
Down counter		
VAR_INPUT CD : BOOL ; R : BOOL ; PV : ANY_INT END_VAR	VAR_OUTPUT Q : BOOL ; CV : ANY_INT ; END_VAR	If LD Then CV = PV Else If CD et CV > Pvmin Then CV = CV – 1 End If If CV <= 0 Then Q = TRUE Else Q = FALSE End If
Up and Down counter		
VAR_INPUT CU : BOOL ; CD : BOOL ; R : BOOL ; LD : BOOL ; PV : ANY_INT END_VAR	VAR_OUTPUT QU : BOOL ; QD : BOOL CV : ANY_INT ; END_VAR	If R Then CV = 0 Else If LD Then CV = PV Else If Not (CU and CD) Then If CU and CV < Pvmax then CV = CV + 1 Else If CD and CV > Pvmin Then CV = CV -1 End If End If If CV >= PV Then QU = TRUE Else QU = FALSE End If If CV <= 0 Then QD = TRUE Else QD = FALSE End If

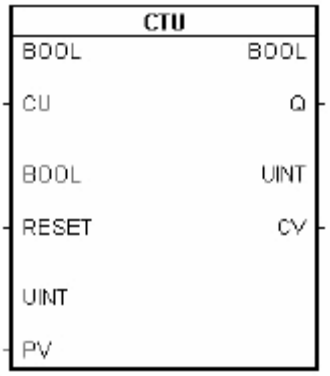
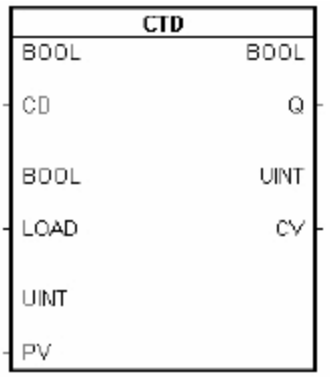
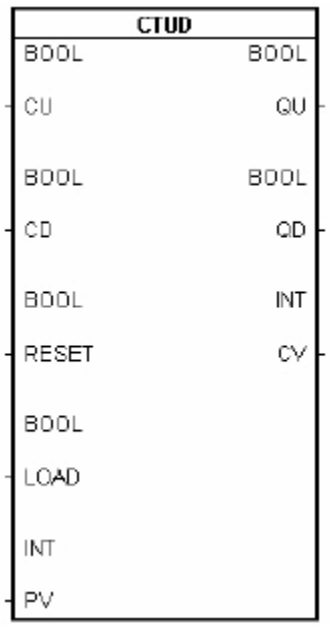
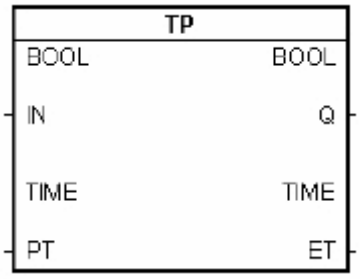
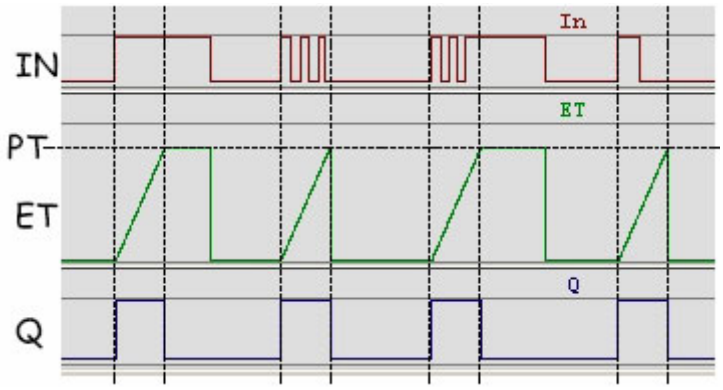
Up counter	
	A rising edge on Boolean input CU increments output CV by 1 (max value 32766) If Boolean input RESET is at 1 then whatever the state of Boolean input CU, output CV is forced to 0 The Boolean output Q is at 1 if the value of output CV is greater than or equal to the value of input PV
Down counter	
	A rising edge on Boolean input CD decrements output CV by 1 (min value 0) If Boolean input LOAD is at 1 then whatever the state of Boolean input CD, output CV is forced to the value of input PV The Boolean output Q is at 1 if the value of output CV is equal to 0
Up and Down counter	
	A rising edge on Boolean input CU increments output CV by 1 (max value 32766) A rising edge on Boolean input CD decrements output CV by 1 (min value 0) If Boolean input RESET is at 1 then whatever the state of Boolean input CU, output CV is forced to 0 If Boolean input LOAD is at 1 then whatever the state of Boolean input CD, output CV is forced to the value of input PV The Boolean output Q is at 1 if the value of output CV is greater than or equal to the value of input PV The Boolean output Q is at 1 if the value of output CV is equal to 0

Figure 51 : The counter function blocks in FBD language

5.3.3.4. The timers

Pulse timing





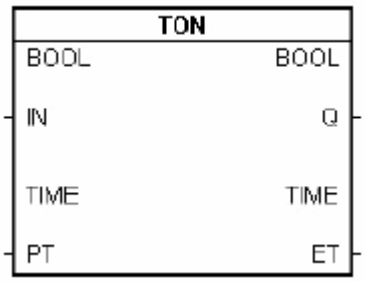
A rising edge on Boolean input IN makes output Q pass to 1. Function block TP then starts to count the time as long as ET (elapsed time) is lower than PT (pulse time)

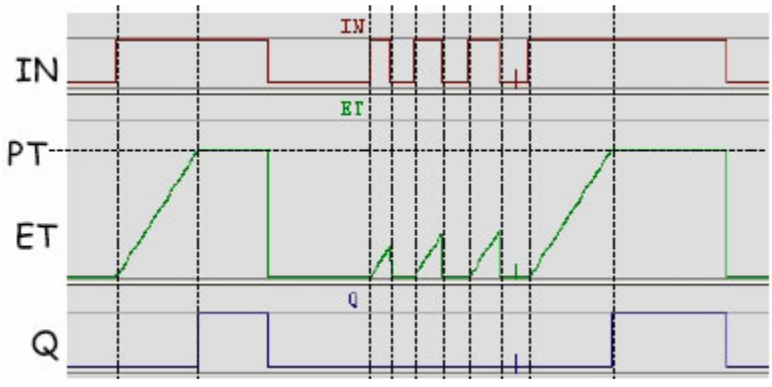
As soon as ET (elapsed time) is equal to PT (pulse time) output Q passes to 0 and function block TP stops counting the time

Output ET is forced to 0 if ET is equal to PT and if input IN is at 0

Function block TP makes it possible to generate, on a rising edge on input IN, a pulse (output Q) with a length equal to the value of PT.

On-delay timing



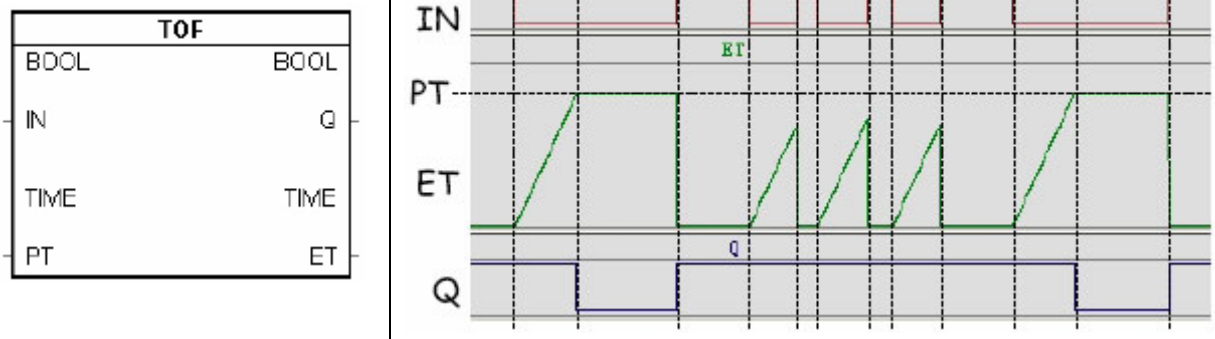


If Boolean input IN is at 0, then Boolean output Q is at 0 and the value of output ET (elapsed time) is also at 0. In this configuration function block TON does not count the time.

When Boolean input IN passes to 1, function block TON starts to count the time and the value of output ET increases until it reaches the preselection value PT (delay time). Function block TON then stops counting the time.

Boolean output Q is at 1 if the value of output ET is equal to the preselection value PT

Off-delay timing



If Boolean input IN is at 1 then Boolean output Q is at 1 and the value of output ET (elapsed time) is at 0. In this configuration function block TOF does not count the time.

When Boolean input IN passes to 0 function block TOF starts counting the time and the value of output ET increases until it reaches the preselection value PT (delay time). Function block TOF then stops counting the time.

Boolean output Q is at 0 if the value of output ET is equal to the preselection value PT

Figure 52 : The timer function blocks in FBD language

5.3.3.5. Library and function blocks

Each manufacturer proposes a set of libraries and function blocks to make it easier to program their hardware.

Each user can create his own libraries, each made up of a set of functions and function blocks. Whatever the programming language used, the functions and function blocks can be accessed in all the languages.

5.3.4. Conclusion

The language in function block form has been developed and is increasingly used these days in DCS programming. We will see this type of programming in detail in the '*Centralised Automated Systems: DCS*' course.

5.4. THE SEQUENTIAL FUNCTION CHART LANGUAGE (SFC)

In order to pass from the expression of requirements to the development of the automated system, it will be necessary to analyse the problem, and to do that we have various tools at our disposal, amongst which there is the SEQUENTIAL FUNCTION CHART. These tools are very useful because they represent, in graphic form, the sequential progress of the steps in an automated system's operation.

The Sequential Function Chart consists of a set of:

- STEPS with which ACTIONS are associated,
- TRANSITIONS with which RECEPTIVITIES are associated,
- ORIENTED LINKS linking the steps to the transitions and the TRANSITIONS to the STEPS.

5.4.1. Definitions

5.4.1.1. The steps

A STEP characterises an invariant behaviour of part or all of the command part. At any given moment and depending on the system's evolution:

- A STEP is either active or INACTIVE,
- The set of ACTIVE STEPS defines the SITUATION of the command part.

A step is represented by a square identified in its upper part. The input to a step is shown in the upper part, and the output in the lower part of each symbol.

Although it is necessary to indicate the situation of the Sequential Function Chart at any given moment, it is convenient to identify all the active steps at that moment by placing a dot in the lower part of the symbols of the steps concerned.

The initial step is represented by doubling the sides of the square of the corresponding step's symbol.

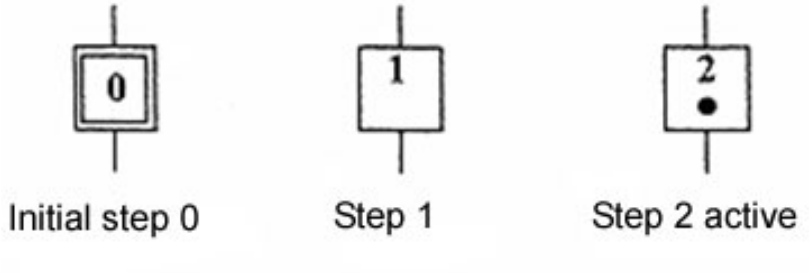


Figure 53: The steps of a Sequential Function Chart

5.4.1.2. Actions associated with the step

One or more basic or complex **ACTIONS** may be associated with a **STEP**. They indicate what must be done whenever the state with which they are associated is **ACTIVE**.

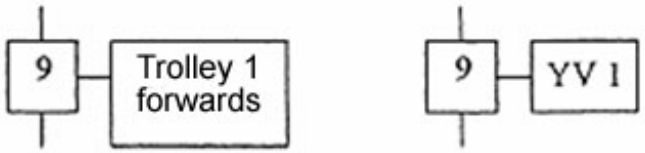
These **ACTIONS** may be external (outputs), or internal (triggering of delays, counting, *etc.)*.

The **OUTPUTS** from the command part correspond to the orders sent to the operative part or to the external elements.

A step that does not have any indication of an action cannot correspond to the act of waiting for an external (change of an input's state for example) or internal (activation of another step, end of a delay, etc.) event.

ACTIONS are described in a literal or symbolic way inside one or more rectangles connected to the symbol of the step with which they are associated.

Figure 54 : Example of an action associated with a step (step No. 9 here)



When the actions are described in symbolic form, a summary table must indicate the correspondence between each symbol used and the action to be executed.

Symbole	Désignation
YV 1	avance chariot 1

Figure 55: Example of an actions summary table

Several actions associated with a given step may be shown in different ways and the examples below express the equivalent forms of representation of two actions A and B associated with the same step.

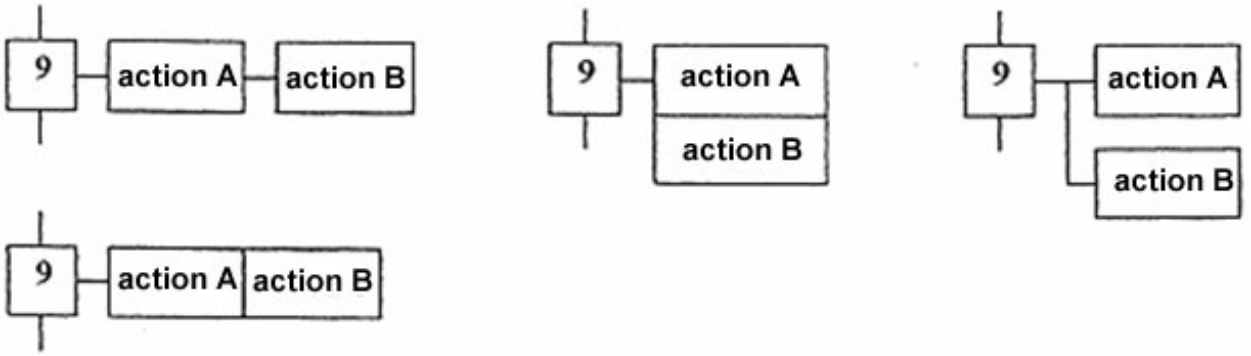
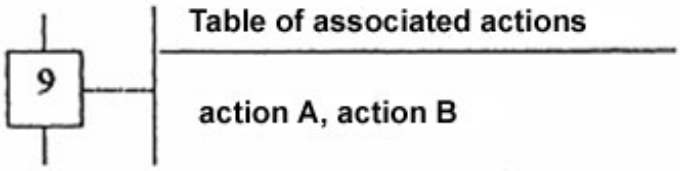


Figure 56 : Example of several actions associated with a step

The actions can also be indicated in a table placed on the same sheet as the **Sequential Function Chart**.

In order to provide a direct visual correlation, the action(s) must be written at the same height as the symbol of the step with a line linking the step to the associated actions.

Figure 57 : Link between a step and two actions



5.4.1.3. Transition

A **TRANSITION** indicates the possibility of changing between steps. This change is accomplished by **PASSING** the **TRANSITION**.

The passing of a transition causes the command part to go from one **SITUATION** to another **SITUATION**.

A transition is either **VALIDATED** or **NOT VALIDATED**. It is said to be VALIDATED when all the immediately preceding steps connected to that transition are active.

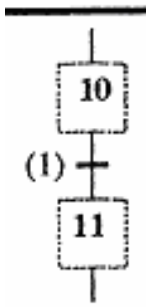


Figure 58 : Transition (1) upstream step 10 to downstream step 11

A transition between two steps is represented by a bar perpendicular to the oriented links.

To make it easier to describe the **Functional Sequence Chart,** each transition should preferably be identified to the left of the bar.

When several transitions are connected to the same step, the corresponding oriented links are grouped together upstream or downstream of the step.

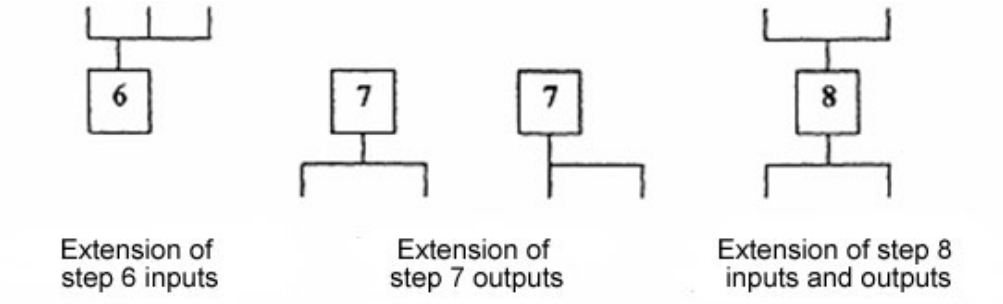


Figure 59: Extensions to inputs and outputs

When several steps are connected to the same transition, the oriented links corresponding to these steps are grouped together upstream or downstream on two parallel horizontal lines.

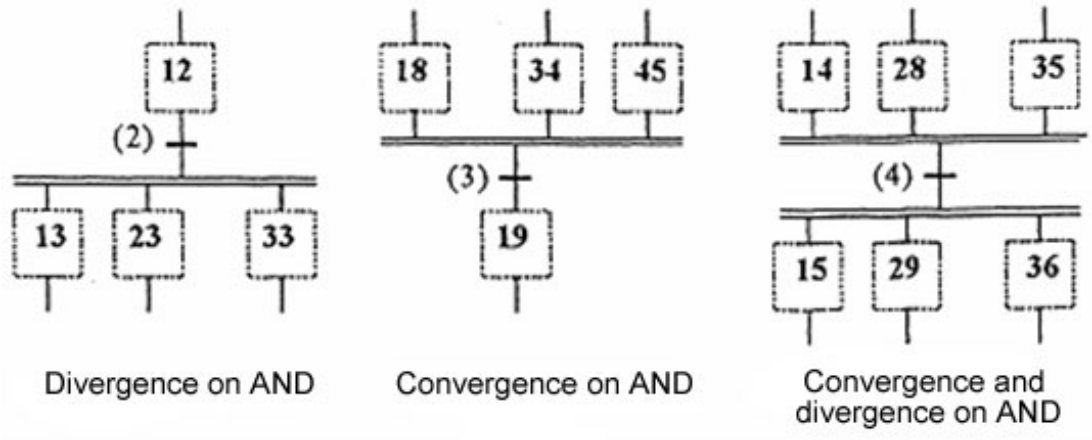


Figure 60 : Divergences and convergence on ET

5.4.1.4. Receptivity associated with the transition

A logical proposition, called receptivity, is associated with each **TRANSITION,** and it can be either **TRUE** or **FALSE**.

Amongst all the items of information available at a given instant, the receptivity groups together only those that are necessary to pass the transition.

This logical proposition depends on the external (inputs) or internal (counter states, delays, active or inactive states other steps) information.

The inputs to the command part correspond to the external information from the operative part or from the instructions given by the operator or from the information given by other systems.

The **RECEPTIVITY** is written, in literal or symbolic form, preferably to the right of the transition symbol.

When the receptivity is written in symbolic form, there must be a summary table indicating the correspondence between each symbol used and the corresponding information.

Figure 61: Literal writing of the receptivity associated with transition (5) and symbolic writing of the receptivity associated with transition (6)

(5) (high position + low position).$\overline{\text{left position}}$

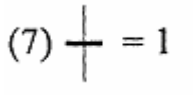
(6) $\overline{SQ10}.(5SQ2 + SQ6)$

So for our example, we have the following associated table.

Figure 62: Summary table

Symbol	Designation
SQ2	Grip high position
SQ6	Gantry 1 on machine
SQ10	Anchor closed

When there is no condition associated with a transition, the receptivity is said to be "always true" and is noted "=1"



In this little example, the receptivity associated with transition 7 is always true.

Remark : if the editing or viewing tools do not allow the use of over-lining, the notation â (complement of a) can be replaced by /a.

5.4.1.5. Oriented links

The ORIENTED LINKS connect the STEPS to the TRANSITIONS and the TRANSITIONS to the STEPS.

They indicate the evolution channels.

Oriented links are represented by horizontal or vertical lines. Oblique lines may be used in exceptional cases, where they make the diagram clearer.

By convention, the direction of the evolutions is always top-down. Arrows are used to mark the orientation of the links:

- when the convention is not respected,
- possibly, when their presence could make it easier to understand the Sequential Function Chart's evolutions.

	Top-down direction		Bottom-up direction

The intersection of a vertical oriented link with a horizontal oriented link is permitted but that does not correspond to a relationship between those links.

Consequently, to avoid any ambiguity such intersections should be avoided.

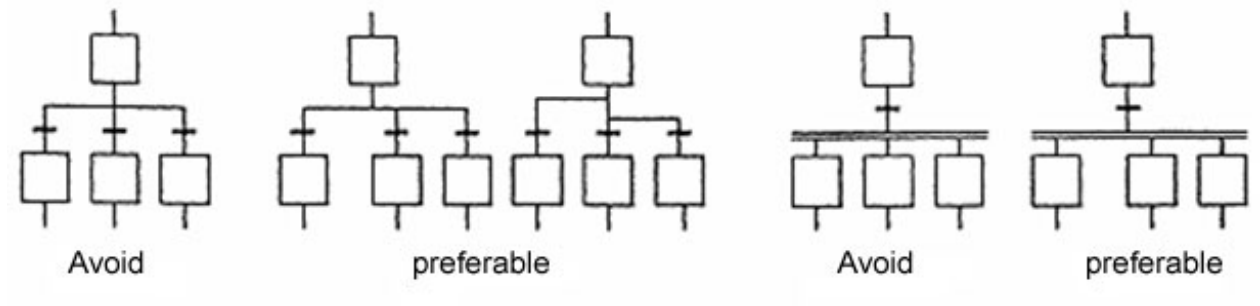
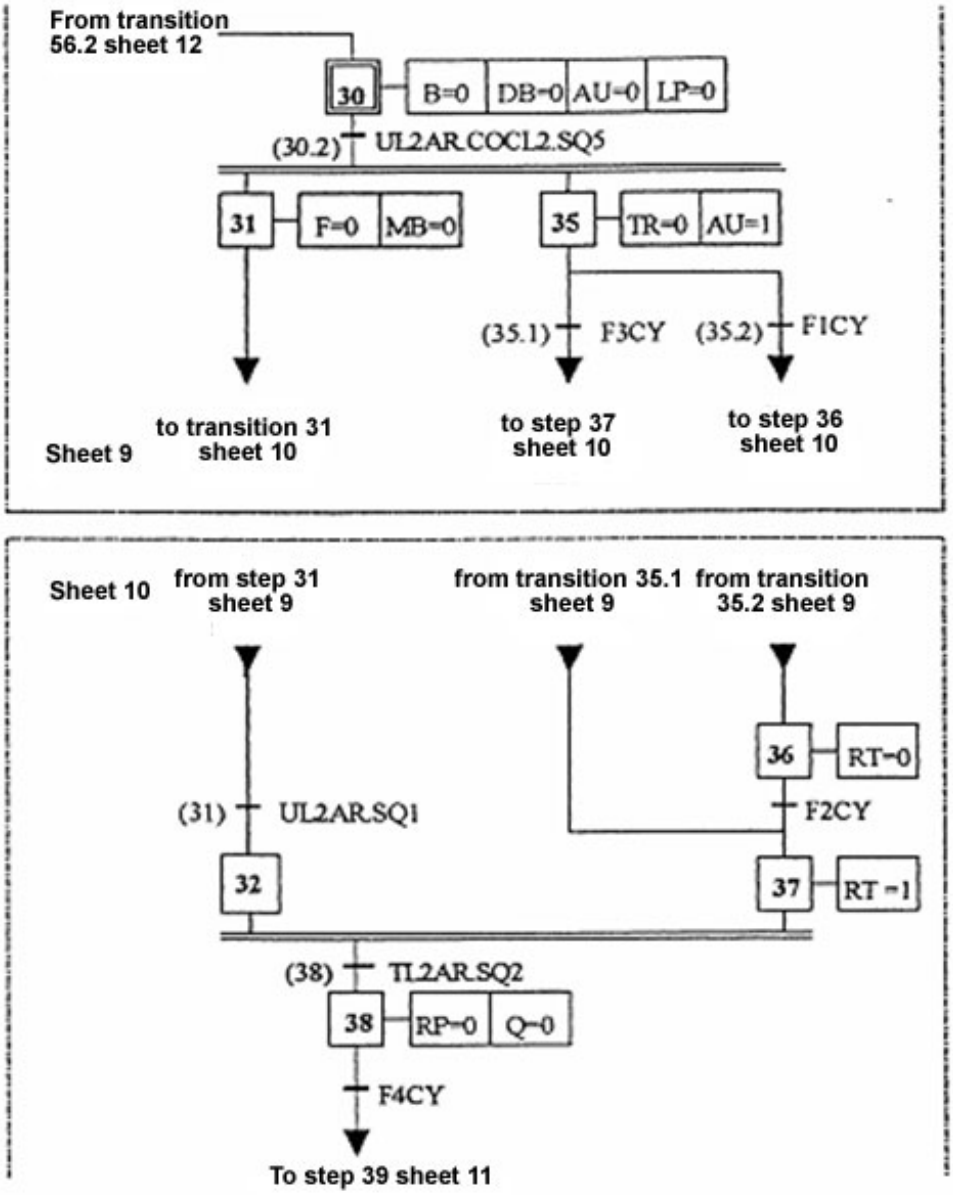


Figure 63: Intersection of links

5.4.1.6. Using references in the links



When an oriented link line cannot be drawn without a break (for example by drawings that are too complex or representations on several pages) references may be used to ensure reading continuity.

Figure 64 : Link references

5.4.2. Syntax rule

The **STEP TRANSITION** and **TRANSITION STEP** alternation must always be respected whatever the sequence passed through.

TWO STEPS MUST NEVER BE LINKED DIRECTLY, they must be separated by a transition.

TWO TRANSITIONS MUST NEVER BE LINKED DIRECTLY, they must be separated by a step.

5.4.3. Evolution rules

5.4.3.1. RULE 1: Initial Situation

The **INITIAL SITUATION** of the **Sequential Function Chart** characterises the initial behaviour of the command part with respect to the operative part and corresponds to the active steps at the beginning of the operation.

If this situation is always identical, as in the case of cyclic automated systems, it will be characterised by the initial steps. It then generally corresponds to behaviour at rest.

In the case where, in non-cyclic automated systems, the initial situation may depend on the state of the process when the command part is started up, the procedure for establishing this situation must be defined in the Sequential Function Chart or by associated documents.

5.4.3.2. RULE 2: passing a transition

The evolution of the **Sequential Function Chart**'s situation corresponding to the **PASSING** of a transition can ONLY take place:

- when that transition is validated
- and the RECEPTIVITY associated with that transition is **TRUE**.

When these conditions are met, the **TRANSITION** becomes **PASSABLE** and is then **OBLIGATORILY** passed.

5.4.3.3. RULE 3: Evolution of the active steps

The passing of a transition simultaneously causes the activation of all the immediately following steps and the deactivation of all the immediately preceding steps.

5.4.3.4. RULE 4: Simultaneous evolutions

Several simultaneously passable Transitions are passed simultaneously.

This simultaneous passing rule makes it possible to break down a Sequential Function Chart into several diagrams while ensuring their interconnections. In this case it is essential to involve the state of the steps in the receptivities. The active state of step "i" will be noted "Xi" and the inactive state "Xi

Examples:

X2 = 1 if step 2 is active

$\overline{X}$ 31 = 1 if step 31 is inactive.

Rule 4 enables, in particular, the simultaneous passing of transitions validated by steps situated in a separate diagram while excluding any ambiguity concerning the possibility of passing transition (6) before transition (5) or vice versa.

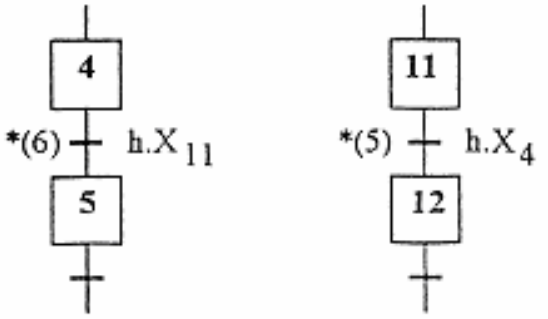


Figure 65: Example of simultaneous evolutions

5.4.3.5. RULE 5: Simultaneous activation and deactivation

If, during operation of the automated system, a step must be simultaneously deactivated and activated, it remains active.

5.4.3.6. RULE 6: Passing times for a transition or for a step's activity

Theoretically a transition's passing time is considered to be as short as you want, but non null, even if in practice this time may be imposed by the technology used for the automated system. Likewise, a step's activation time cannot be considered to be null.

5.4.4. Detailed description

5.4.4.1. Detailed description of the actions

The general term of actions associated with a step makes it possible to define "what must be executed" (effect to be achieved) and "how to obtain it" (orders to be issued)

In a large number of applications, these orders concern the commanding of mechanical or electrical actuators (cylinders, motors, etc.) or the commanding of automated system auxiliary functions (counting, delays, etc.).

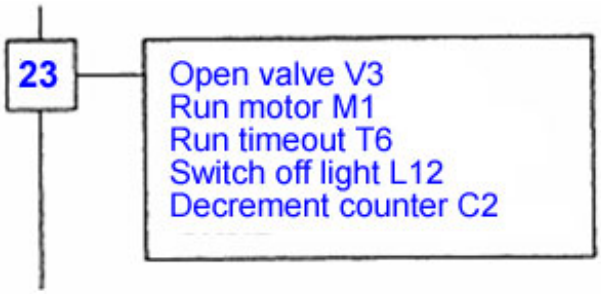
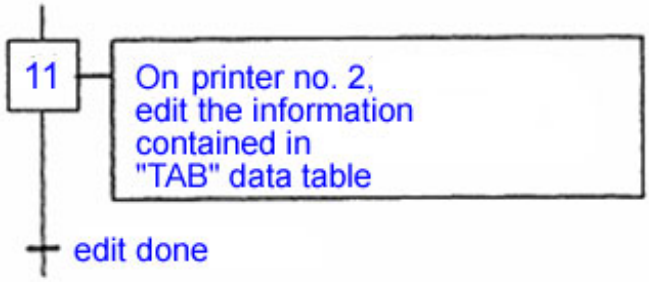


Figure 66: Example of actions associated with a given step.

In the example, the actions indicate what must be done by the operative part, by the external elements or by the command part when step 23 is active.



But the actions can also describe tasks entrusted to other logic systems, even, analog systems, such as regulation loops for example.

Figure 67: Example of actions associated with step 11 correspond to a management task

The specific description tools for a given technique, such as flowcharts for IT tasks, transfer functions or state equations for analog tasks may be used in the framework of each step to describe each of the tasks separately.

5.4.5. Classification of actions

For the classification of the most frequently used actions, listed below, the criterion adopted will be the length of the action compared with the length of the step's activity.

In accordance with the convention already used in the diagrams shown below, "Xi" corresponds to the active state of step "i".

5.4.5.1. Continuous action

The execution of the continuous action continues as long as the step with which it is associated remains active.

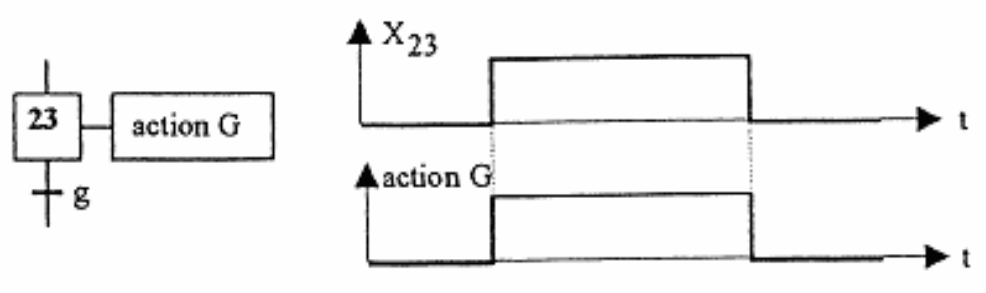


Figure 68 : Continuous action

5.4.5.2. Conditional action

A conditional action is a continuous action whose execution is subject to a logical condition being met.

This logical condition is noted inside the action rectangle. When the condition is complex and leads to a significant increase in the size of the rectangle, it can be referenced outside the rectangle by means of a line located on the upper part.

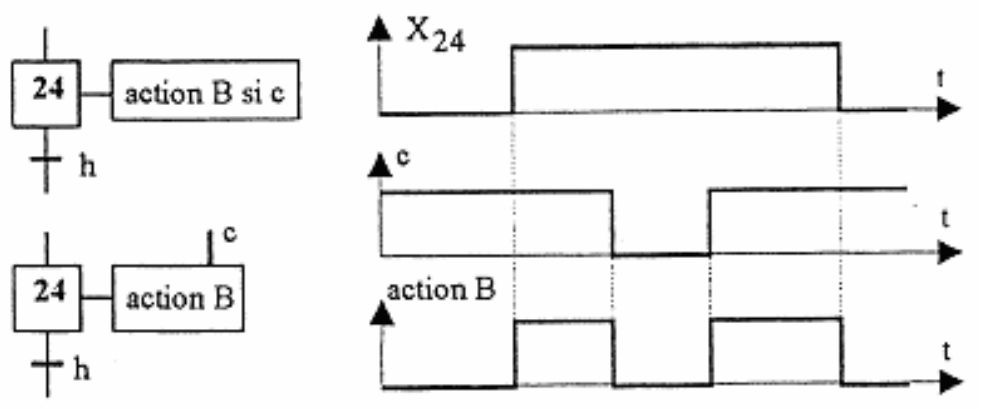


Figure 69 : Conditional action

5.4.5.3. Delayed action

This is a special case of conditional actions that is very frequently found in applications where time is involved as a logical condition. The time indication is given by the general notation "t/i/q" which fixes after the letter "t" the step identifier "i" which determines the origin of the time and its length.

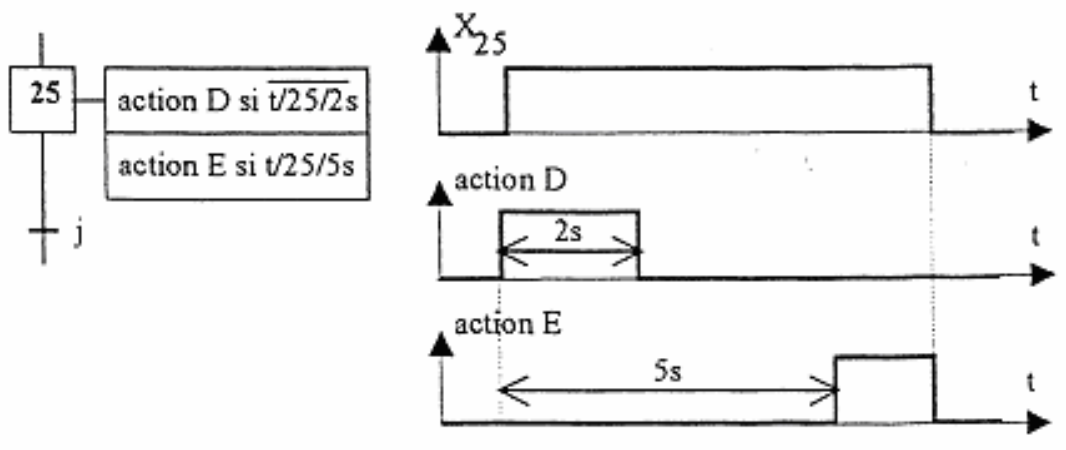


Figure 70 : Delayed action

This term "t/i/q" takes the logic value "1" as soon as "q" seconds have elapsed since the last activation of step "i".

5.4.5.4. Maintained effect

A difficulty that we frequently come across when creating the Sequential Function Chart concerns the representation of actions, or more precisely of orders, whose effect must continue to be felt throughout the duration of a certain number of consecutive steps. Two types of description can be considered.

Effect maintained by non-memorised continuous actions

It is the repetition of the action or of the order in all the steps concerned that ensures the continuity of the effect, or of a different way of using simultaneous sequence structures (figure below).

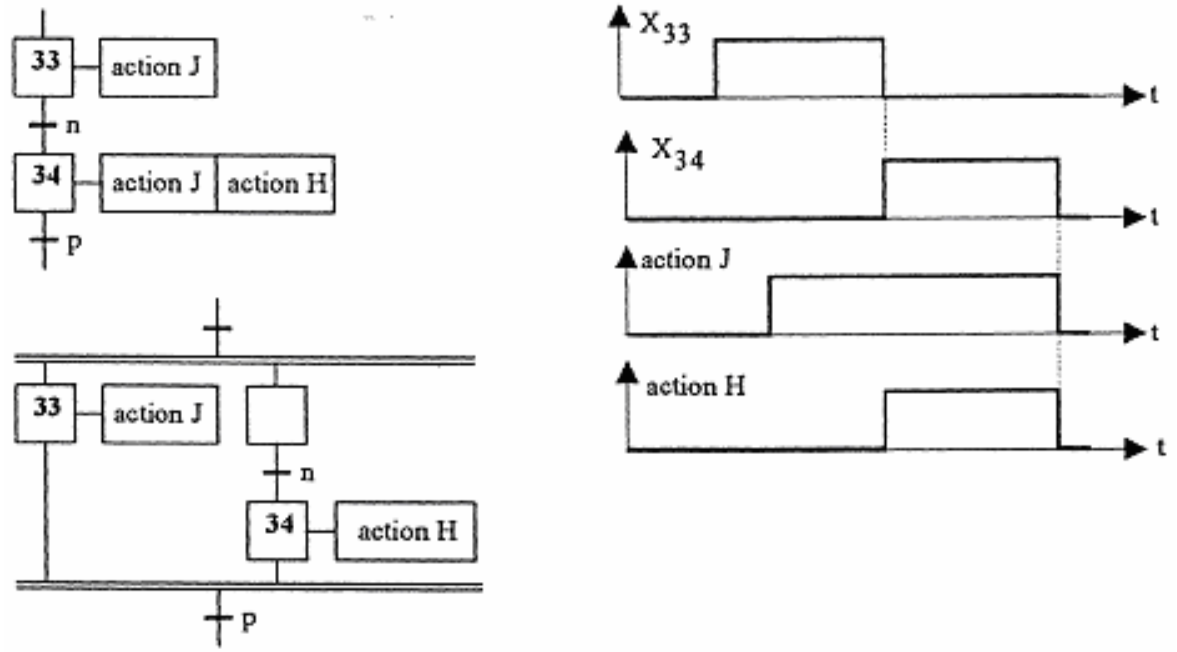


Figure 71 : Effect maintained by non-memorised continuous actions

Effect maintained by memorised continuous actions

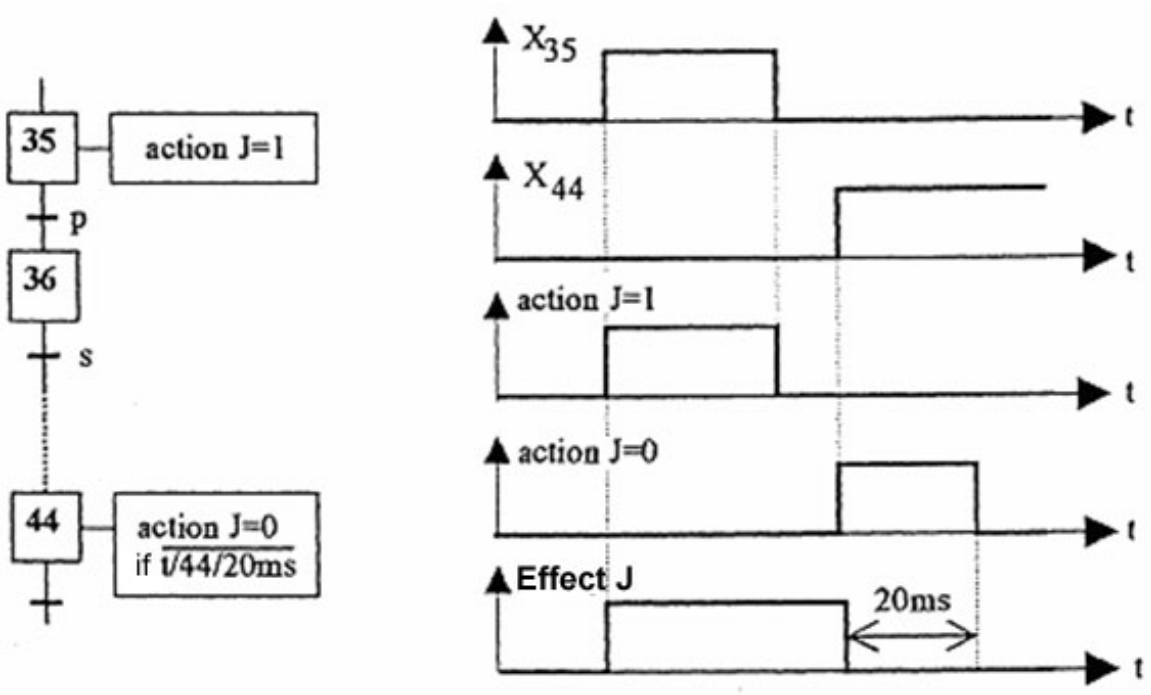


Figure 72 : Effect maintained by memorised continuous actions

The actions or orders are detailed in the steps in which the effect must begin (setting to logic state "1") or end (setting to logic state "0"). These setting or resetting actions or command orders may be continuous or of limited length.

This description of a maintained effect corresponds to a memorisation of command orders performed:

- either by the actual operative part (pre-actuators with a memory),
- or by an auxiliary memory in the command part (memorisation of the order).

Remark:

The description by means of memorised actions makes it possible to simplify the representation of the Sequential Function Chart but carries over part of the functional description to auxiliary functions, which means that the Sequential Function Chart no longer suffices to obtain directly an overall view of the effect of the orders given by the command part at each step.

The description by means of non-memorised continuous actions in paragraph a offers the advantage, however, of better representing the effective limits of the actions or of their effects, despite the fact that this description is more cumbersome.

5.4.6. Detailed description of the receptivities

The receptivities can be expressed in various forms, such as:

- moving part in right-hand position
- door opening end-of-travel actuated
- temperature at 300°C
- value of counter C10 = set value
- step "n" active
- 10 seconds elapsed since last activation of step 6, etc.

5.4.6.1. Taking time into account

In accordance with the notation already used, "t/8/10s" means 10 seconds elapsed since the last activation of step 8. Step 8 is at the origin of two times 5 and 10 seconds, consequently:

- The receptivity of transition 8 becomes true once 5 seconds have elapsed since activation of step 8.

- The receptivity of transition 9 becomes true once 10 seconds have elapsed since activation of step 8.

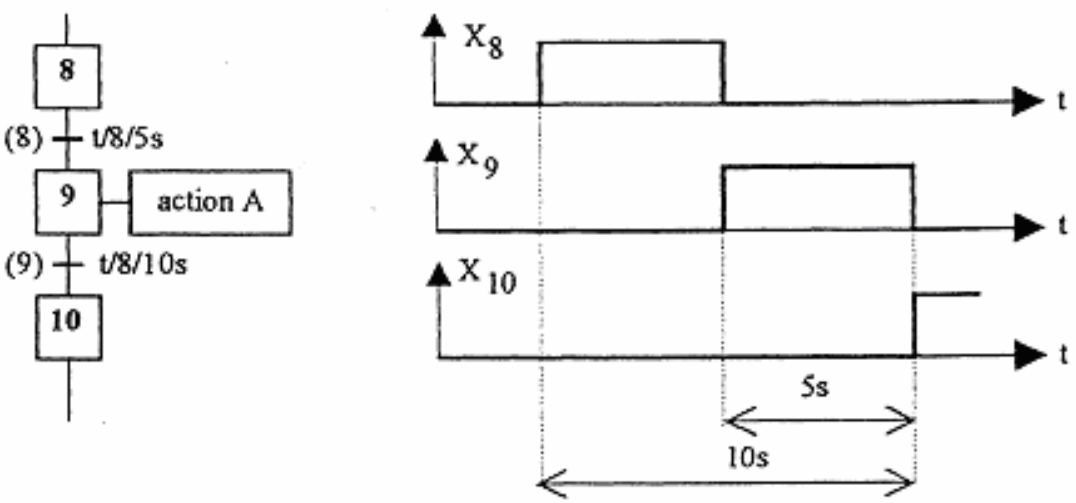


Figure 73 : Taking time into account

5.4.6.2. Taking changes of information state into account

The logical propositions used in the receptivities can use not only the logic states "1 or 0" of an item of information or of the combination of a group of items of information, for example, but also their change of state.

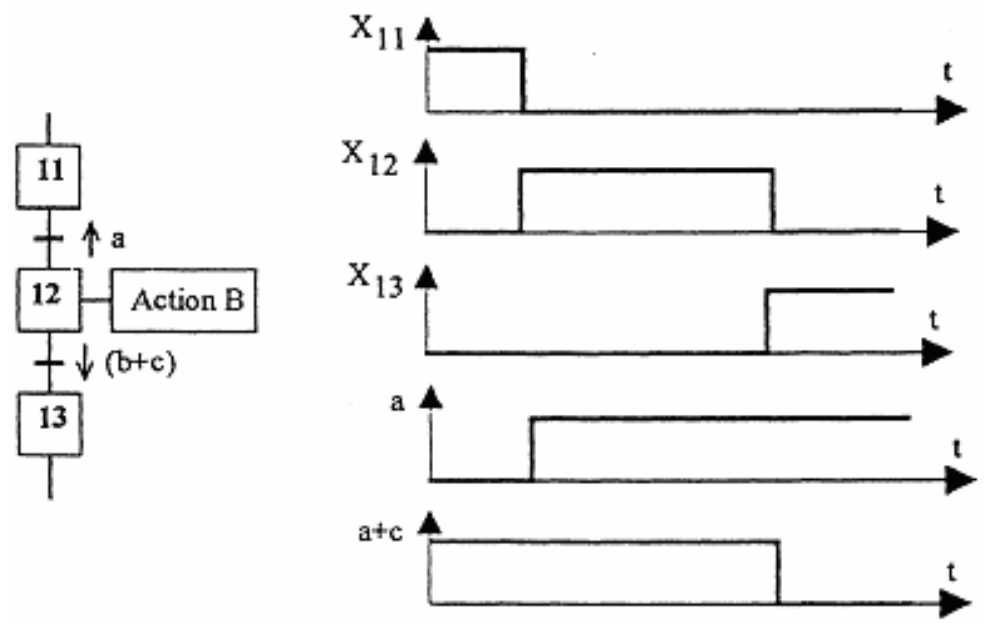


Figure 74 : Taking changes of state into account

The notation “↑a” represents the detection of the passing from logic state “a=0” to logic state “a=1” (rising edge of the logic variable “a”) and the notation “↓(a+b)" represents the passing from logic state "(b+c)=1" to state "(b+c)=0".

This receptivity only becomes true at the time of the information's specified change of state.

5.4.7. Unique sequence

A unique sequence consists of a sequence of steps that can be activated one after the other (figure opposite).

Each step is only followed by a single transition and each transition is only validated by a single step.

The sequence is said to be "active" if at least one step is active.

It is said to be "inactive" if all the steps are inactive.

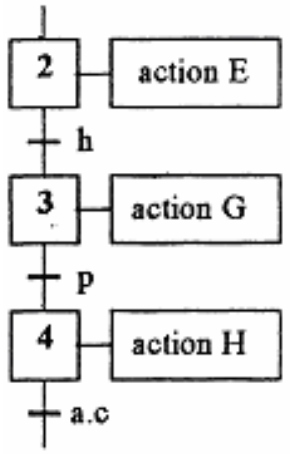


Figure 75 : Unique sequence

5.4.8. Sequence selection

A sequence selection or choice of change between several steps or sequences takes the form, starting from one or more steps, of as many validated transitions as there are possible changes.

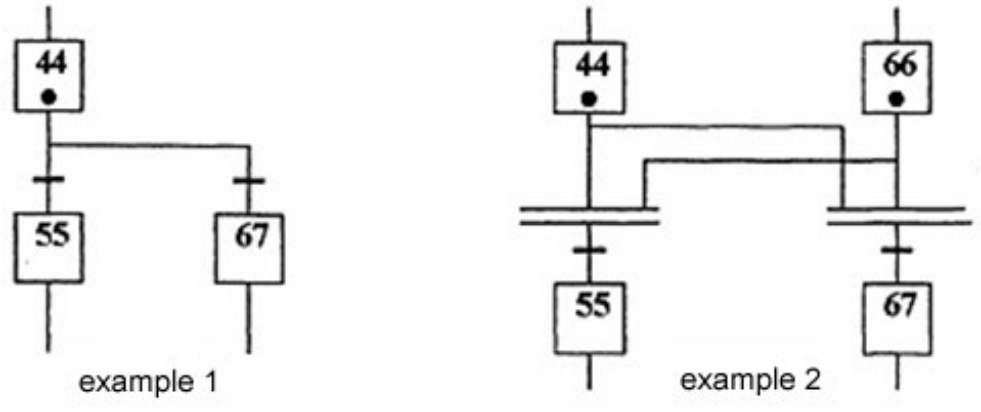


Figure 76 : Sequence selection

Example 1: selection between two possible changes starting from step 44.

Example 2: selection between two possible changes starting from two steps 44 and 66,

5.4.9. Exclusive sequences

To obtain an exclusive selection between several possible changes starting from one step, it is necessary to ensure that all the receptivities associated with the transitions are exclusive, that is to say that they cannot be true simultaneously. This exclusion may:

- either be of a physical order (mechanical or time incompatibility),
- or of a logical order in the writing of the receptivities.

Example 1:

Receptivities â. b and a. b are logically exclusive.

Consequently if "a and b" are true at the same time it will not be possible to pass any transition starting from step 44.

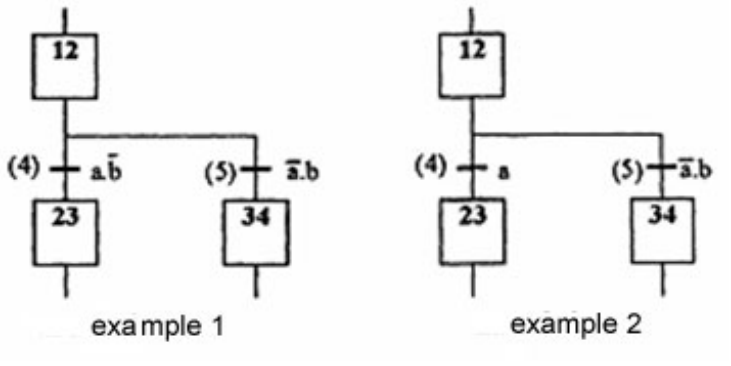


Figure 77 : Examples of exclusion of a logical order

Example 2:

The priority is given to transition (4) which allows it to be passed when "a and b" are true at the same time.

5.4.10. Step jumps and sequence resumptions

A step jump makes it possible to skip one or more steps when, for example, the actions to be performed in those steps no longer serve any purpose or are no longer applicable.

A sequence resumption, on the contrary, makes it possible to start a given sequence over again several times as long the condition that has been set is not met.

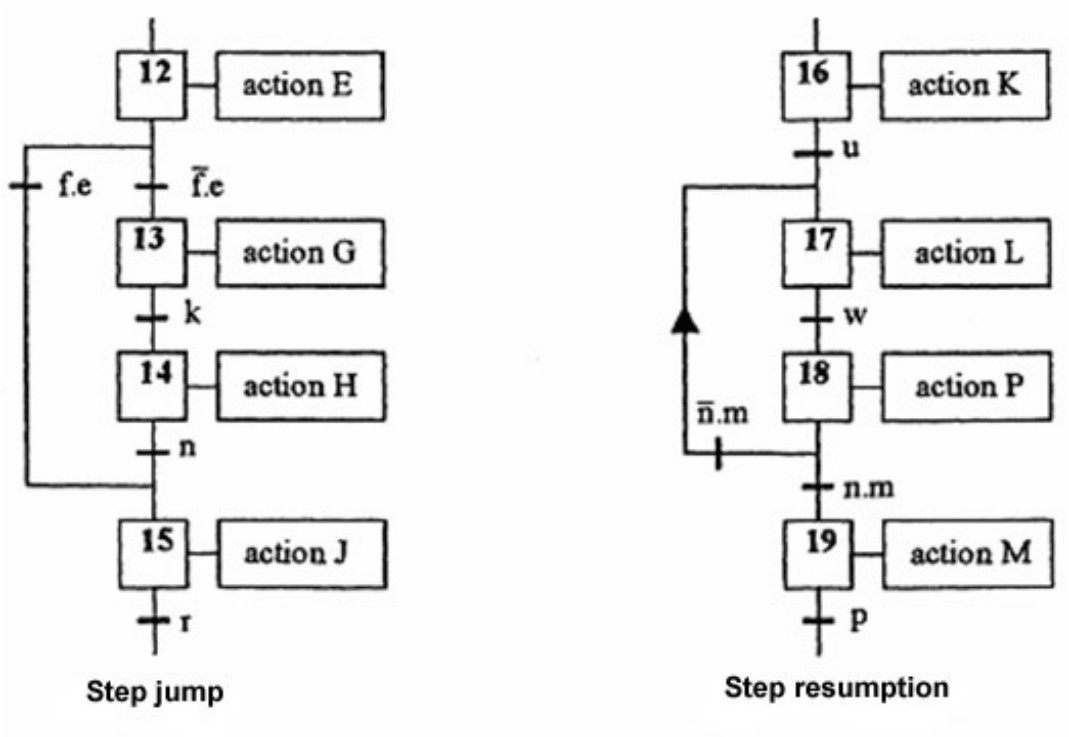


Figure 78 : Examples of step jumps and resumptions

5.4.11. Simultaneous sequences: structural parallelism

When the passing of a transition leads to several sequences being activated at the same time, these sequences are said to be "***simultaneous sequences***".

After the simultaneous activation of those sequences, the changes of active steps in each of the sequences then become independent.

The simultaneous activations or deactivations of those sequences may be performed in one or more times as shown in the example below.

To ensure the synchronisation of the deactivation of several sequences at the same time, reciprocal wait steps are generally provided for.

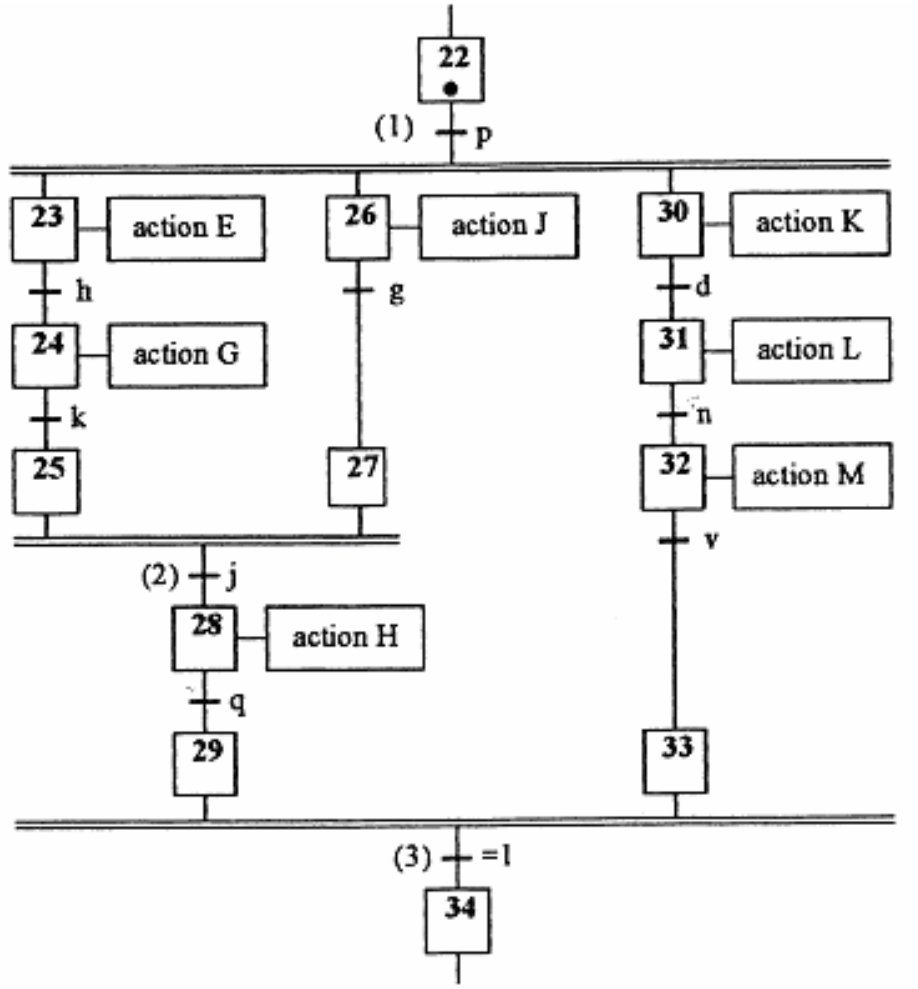


Figure 79 : Examples of simultaneous sequences

Starting from step 22 active, the passing of transition (1) by the receptivity "p = 1" causes the activation of three sequences S1 {23, 24, 251; S2 {26, 27} and S3 {30, 31, 32, 33} and the deactivation of step 22.

These three sequences then change in a completely independent way and when reciprocal wait steps 25 and 27 are active and the receptivity 1=1", the passing of transition (2) leads to the deactivation of two sequences S1 and S2 and to the activation of a new sequence S4 {28, 29).

Likewise, when in turn steps 29 and 33 are active at the same time, transition (3) is passed immediately (receptivity (3) always true) causing the activation of step 34 and the deactivation of sequences S4 and S3.

5.4.12. Interpreted parallelism

When the receptivities associated with the transitions validated by one or more steps are not exclusive, simultaneous changes may occur, leading to several steps being activated at the same time. This second type is called interpreted parallelism because, instead of being evidenced by the structure of the diagram as in the case of structural parallelism (see previous paragraph), these changes, whether simultaneous or not, are determined by the receptivities associated with the transitions.

This type of **OPERATION MUST BE USED WITH CARE** because the most difficult part is to correctly specify the way it ends.

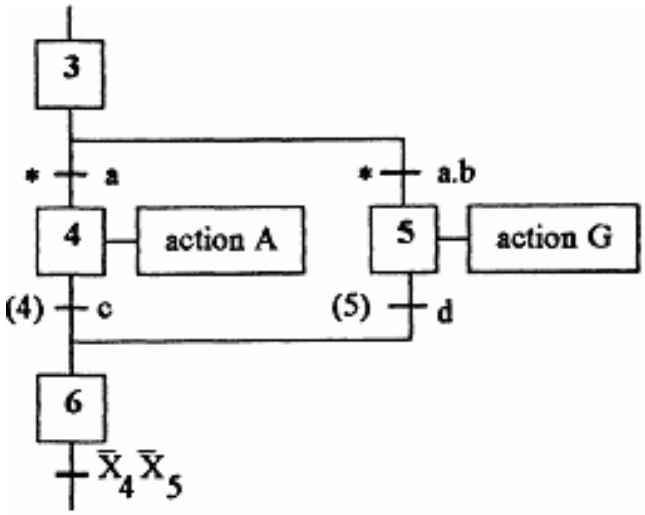


Figure 80 : Examples of simultaneous sequences with interpreted parallelism

Step 3 becomes active:

- if condition "a" becomes true without condition "b" being true, step 4 is alone activated.
- if condition "b" is true before condition "a" becomes true, the two steps 4 and 5 are activated simultaneously.

The receptivity associated with transition (6) must take into account the inactive states of steps 4 and 5 in order to perform a resynchronisation before pursuing the joint sequence.

Remark:

In the second case, the verification of receptivity "c" before receptivity "d" for example activates step 6 and deactivates step 4. The later passing of transition (5) deactivates step 5, and step 6 which is already deactivated, remains active. Receptivity "X4. X5" is then true and transition (6) is passed.

5.4.13. Reusing a given sequence

When a given sequence is used several times, that sequence can be organised in the same way as a subroutine.

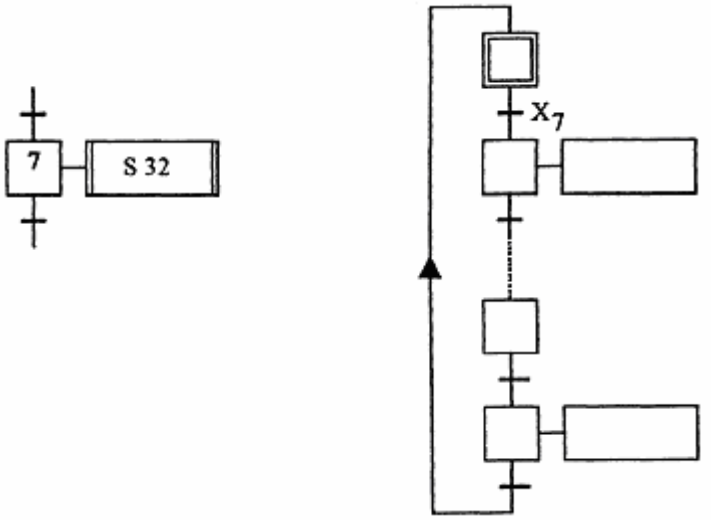


Figure 81 : Using subroutines to avoid repeating the sequences

5.4.14. Macrosteps

The purpose of macrosteps is to facilitate the description of complex systems. Macrosteps make it possible to lighten the graphics of a Sequential Function Chart by detailing certain parts separately.

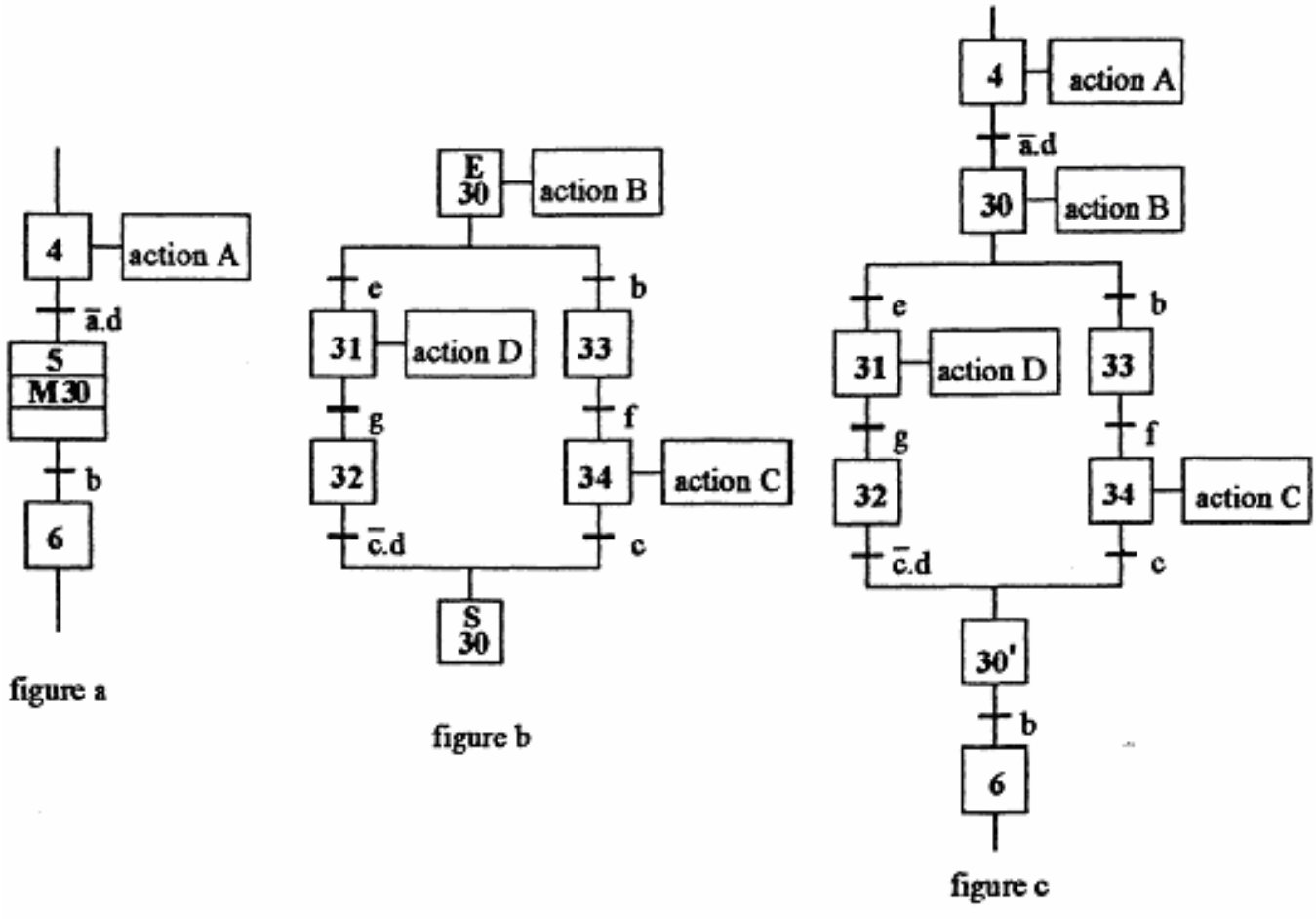


Figure 82 : Examples of macrosteps

A macrostep is represented by a square divided into three parts by two horizontal lines. A macrostep noted 5/M30 is shown in figure a.

It represents a part of the Sequential Function Chart that is detailed elsewhere and which is called the macrostep expansion. Figure b corresponds to the expansion of macrostep M30.

If you replace macrostep 5/M30 with this macrostep expansion you will obtain the Sequential Function Chart shown in figure c.

Rules concerning macrosteps

A macrostep and its expansion obey the following rules:

- A macrostep expansion only has one input step (noted E) and one output step (noted S).
- Any passing of a transition upstream of the macrostep, activates its expansion's input step
- The output step from the macrostep expansion contributes to the validation of the downstream transitions, in accordance with the structure of the Sequential Function Chart that contains that macrostep.
- There are no oriented links that arrive on or leave from the macrostep expansion.

5.4.15. Sharing of resources or of sequences

A "common physical or logical resource" can be shared between several user sequences, in the form of a step validating several transitions: the passing of one of them will lead to only one of those sequences being activated.

When that step is active, the resource will be assigned to the first transition that becomes passable, and to achieve that a logical priority will therefore be essential in the writing of the receptivities so as to avoid any conflict.

In the following example, the common resource represented by step 20 enables the validation of transitions 7, 8 or 9 and one of the three sequences will be passed through depending on which transition is the first one to become passable.

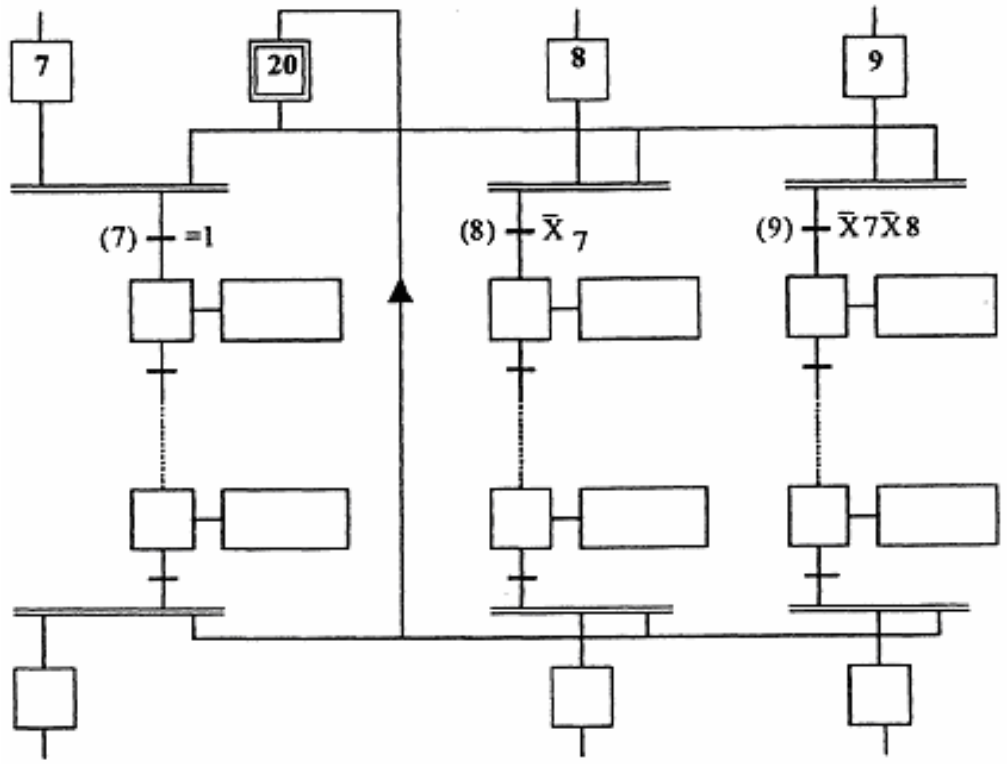


Figure 83 : Example of a common resource

5.4.16. Coupling between sequences

One or more steps may enable successive logical synchronisations of several sequences by "memorising" the necessary authorisations at the right moment.

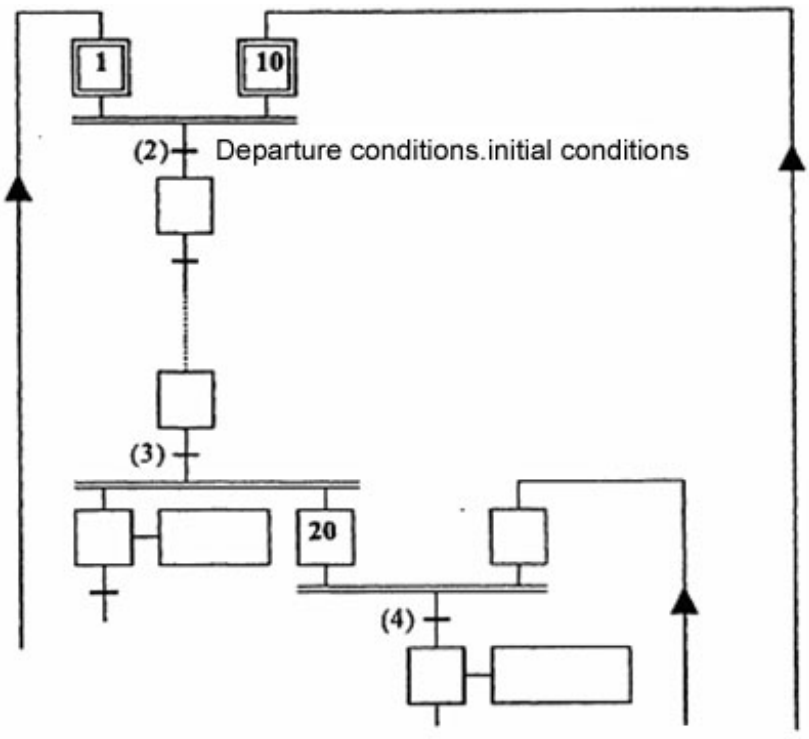


Figure 84 : Example of coupling between sequences

Steps 1 and 10 validate transition (2). Once the associated receptivity is true, sequence S1 will become active and when it has ended, step 20 will memorise the authorisation given to sequence S2 which will then be able to run when the conditions specific to that sequence are met.

5.4.17. Rules for transforming a Sequential Function Chart into equations

In order to write the equations relative to a Sequential Function Chart, you will have to give two equations for each step: an activation equation and a deactivation equation. The form of these equations depends on the structure of the Sequential Function Chart.

5.4.17.1. Unique branch

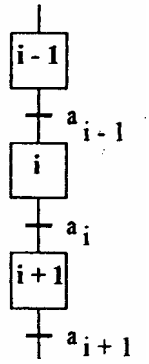
SFC	Equations	
	Activation (S -> set)	Deactivation (R -> reset)
	Step i : $S_i = X_{i-1}.a_{i-1}$	Step i : $R_i = X_{i+1}$

Figure 85 : Unique branch

5.4.17.2. Divergence in OR

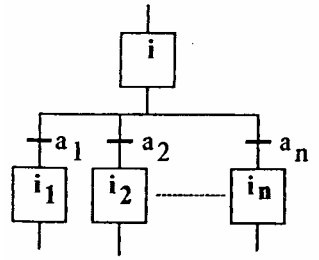
SFC	Equations	
	Activation (S -> set)	Deactivation (R -> reset)
	Step i_1 : $S_{i1} = X_i.a_1$ Step i_n : $S_{in} = x_i \ . \ a_n$	Step i : $R_i = X_1 + X_2 + \ldots + X_n$

Figure 86 : Divergence in OR

5.4.17.3. Convergence in OR

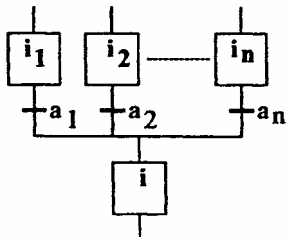
SFC	Equations	
	Activation (S -> set)	Deactivation (R -> reset)
	Step i : $S_i = X_1 . a_1 + \ldots + X_n . a_n$	Steps i_1 ; i_2 ; … ; i_n : $R_{i1} = R_{i2} = \ldots + R_{in} = X_i$

Figure 87 : Convergence in OR

5.4.17.4. Divergence in AND

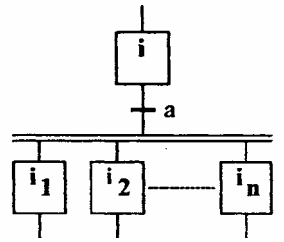
SFC	Equations	
	Activation (S -> set)	Deactivation (R -> reset)
	Step i_1 … i_n : $S_{i1} = S_{i2} = \ldots S_{in} = X_i . a$	Steps i : $R_i = X_{i1} . X_{i2} \ldots X_{in}$

Figure 88 : Divergence in AND

5.4.17.5. Convergence in AND

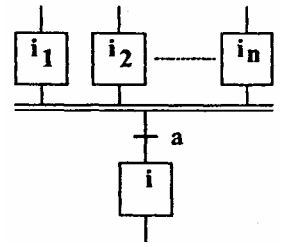
SFC	Equations	
	Activation (S -> set)	Deactivation (R -> reset)
	Step i : $S_i = (X_{i1} . X_{i2} \ldots X_{in})\ a$	Steps i_1 ; i_2 ; … ; i_n : $R_{i1} = R_{i2} = \ldots R_{in} = X_i$

Figure 89 : Convergence in AND

5.4.18. Organisation of Sequential Function Charts in RS memory: asynchronous sequencers

In the design of the Sequential Function Chart, we have seen that we can go into one step (activate it), stay there, then leave it (deactivate it) and go on to the next step. The logic circuit that simply performs these operations is the RS memory.

5.4.18.1. Reminder concerning RS memories

R	S	Q_n	$\overline{Q}_n$
0	0	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	1	0
1	0	0	1
1	1	-	-

Figure 90 : Reminder concerning RS memories

Remark:

The symbol of the RS memory, and the implication table, does not show how the memory is made, and an RS memory can be made in two ways:

- With NOR gates --> RS with priority Off (in this case for the combination R = 1 and S = 1 -› Q = 0 and Q = 0 – caution the outputs are no longer complementary)
- With NAND gates --> RS with priority On (in this case for the combination R =1 and S =1-~ Q =1 and Q =1 – caution the outputs are no longer complementary)

Furthermore, an RS memory does not have any forcing input, which means it is impossible to put it in a given state without acting on inputs R and S.

5.4.18.2. Creating a phase module

The phase module is a circuit materialising a Sequential Function Chart step, the sequencer will therefore be an association of the phase module, and the structure of this association depends on the structure of the Sequential Function Chart.

In order to see what a phase module consists of you must simply remember how it changes in a Sequential Function Chart. You go from one step to the next by passing a transition, which is expressed in the following way:

- we deactivate the previous step

- we pass the transition (receptivity equal to 1)
- we activate the next step

This means that we will only activate the next step's memory if the previous step is already in state 1 and if the variable (receptivity associated with the transition) is equal to 1.

Then the next step's memory having passed to state 1, we will have to deactivate the previous step by returning its memory to state 0.

Overall operating safety requires that, at startup, only the initial step's module should be activated and all the others deactivated. This is achieved by sending the initialisation signal (INIT) on input S of the first memory and on the inputs R of all the others.

So we will see the shape of the phase module appears and this will be used to construct the whole sequencer: an RS memory completed by an AND gate and an OR gate (you can increase the number of inputs on these gates

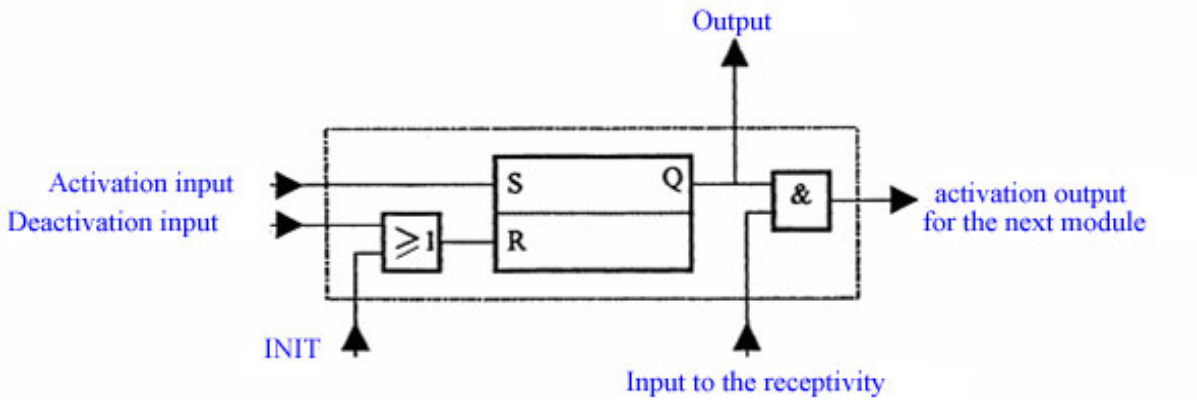


Figure 91: Phase module

5.4.19. Problems encountered in asynchronous sequencers

The layout of a Sequential Function Chart using an RS memory poses a certain number of problems, amongst which we have:

The problems specific to RS memories

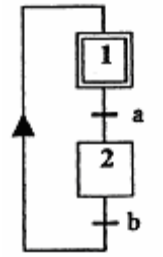
- When you set the two inputs R and S of an RS memory to 1, outputs Q and Q are no longer complementary, and this is true whatever the type of memory (priority Off or priority On); furthermore the outputs return randomly to (Q = 0, Q =1) or (Q =1 and Q = 0) when inputs R and S return from the combination 11 to combination 00 (rest).

- RS memories are asynchronous circuits, which means that inputs R and S are always sensitive because they are ready to react to an incoming signal at any moment. However, in an industrial environment disturbances are frequent and can act on the inputs like logic levels (0 or 1), and in our case here, the possible consequences would be: untimely activation or deactivation of one or more phase modules.

- RS memories do not have a forcing input, which poses a problem for setting the sequencer to its initial situation.

Impossibility of making certain Sequential Function Charts

Let us take the Sequential Function Chart in the figure below, transformation of this SFC into equations gives:

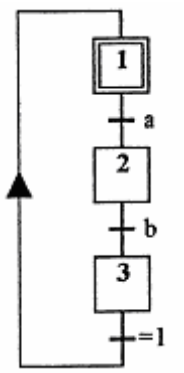


	Step 1	Step 2
Activation	$S_1 = X_2 . b$	$S_2 = X_1 . a$
Deactivation	$R_1 = X_2$	$R_2 = X_1$

But then, with step 2 active and b = 1 at the same time, we obtain $S_1 = R_i = 1$ and we know that RS memories respond in a random way after excitation 11→ 00.

The same applies for S2 = R2 =1 when we simultaneously have step 1 active and a =1 **which explains why it is prohibited to create loops between 2 steps when you use RS memories.**

One solution to the problem consists of making the expressions S (Set) and R (Reset) exclusive.



	Step 1	Step 2
Activation	$S_1 = X_2 . b$	$S_2 = X_1 . a$
Deactivation	$R_1 = X_2 . \bar{b}$	$R_2 = X_1 . \bar{a}$

We often prefer to use a different method which avoids the random behaviour that is still possible between a and $\bar{a}$, between b and $\bar{b}$, we create a third step (see figure above).

This last method is the one used in industrial sequencers and even in certain PLCs.

Momentary non-compliance with rule 3 of Sequential Function Chart change

Sequential Function Charts require that the passing of a transition causes the activation of the step(s) immediately downstream (output steps from the transition) and, simultaneously, the deactivation of all the steps immediately upstream (input steps to the transition).

However, with the structure that we have just described (phase module), the previous step is only deactivated after the next step has been activated; there is therefore an overlap period during which the memories representing the transition's input and output steps are simultaneously at 1: this can lead to random operation in the automated system, because opposing and incompatible commands present in two different and consecutive steps could be activated at the same time.

Example :

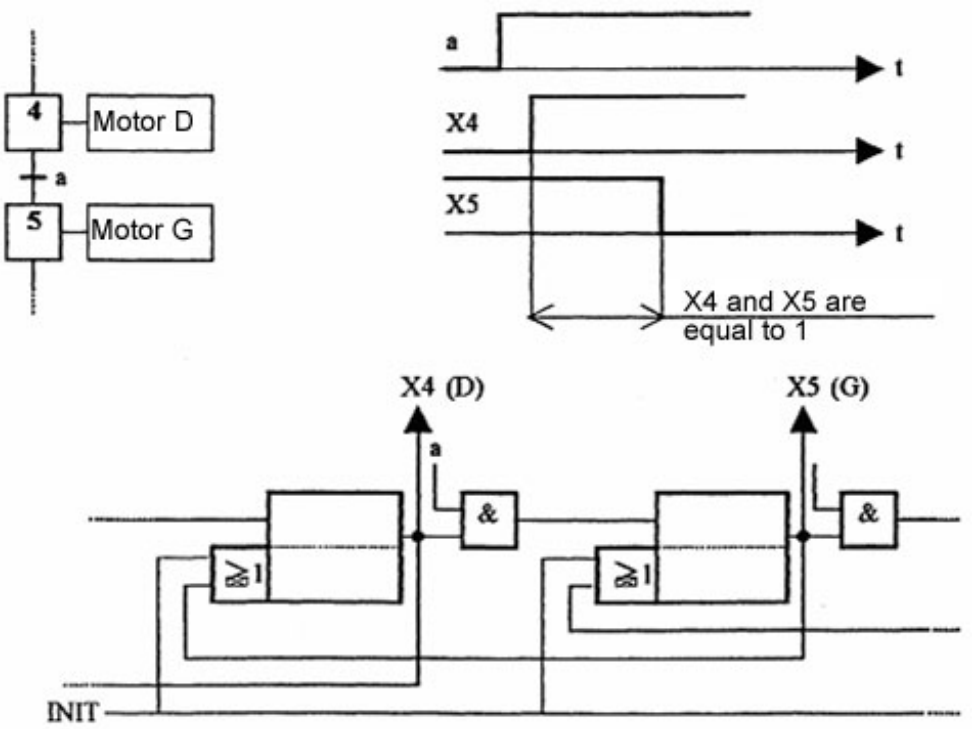


Figure 92: Example of momentary non-compliance with the Sequential Function Chart change rule

We can see that in step 4 the right-hand motor (D) is powered; when a=1 we pass to step 5, which powers the left-hand motor (G), but motor (D) remains powered up until step 5 deactivates step 4, which will cut out the power to motor (D).

Here, the unforeseen operation will not "break" anything because of the motors inertia which will create electromechanical response times that are much longer than the time the two memories overlap in state 1.

The same thing would not apply if the actions of steps 4 and 5 were "electronic".

Conclusion

You can try to improve the structure of the phase module, but you will never be able to obtain a totally fault-free circuit for it. However, despite the defects of asynchronous circuits that we have just mentioned, they must not be condemned out of hand, because they can always be used in simple systems provided you make sure there is no unforeseen switching.

5.4.20. Organisation as a JK flip-flop: synchronous sequencers

The principle of these sequencers is based on the fact that the associated activation and deactivation operations must be simultaneous, you must therefore take the same logic function for the activation of the modules to which you are going and the deactivation of the modules that you are leaving.

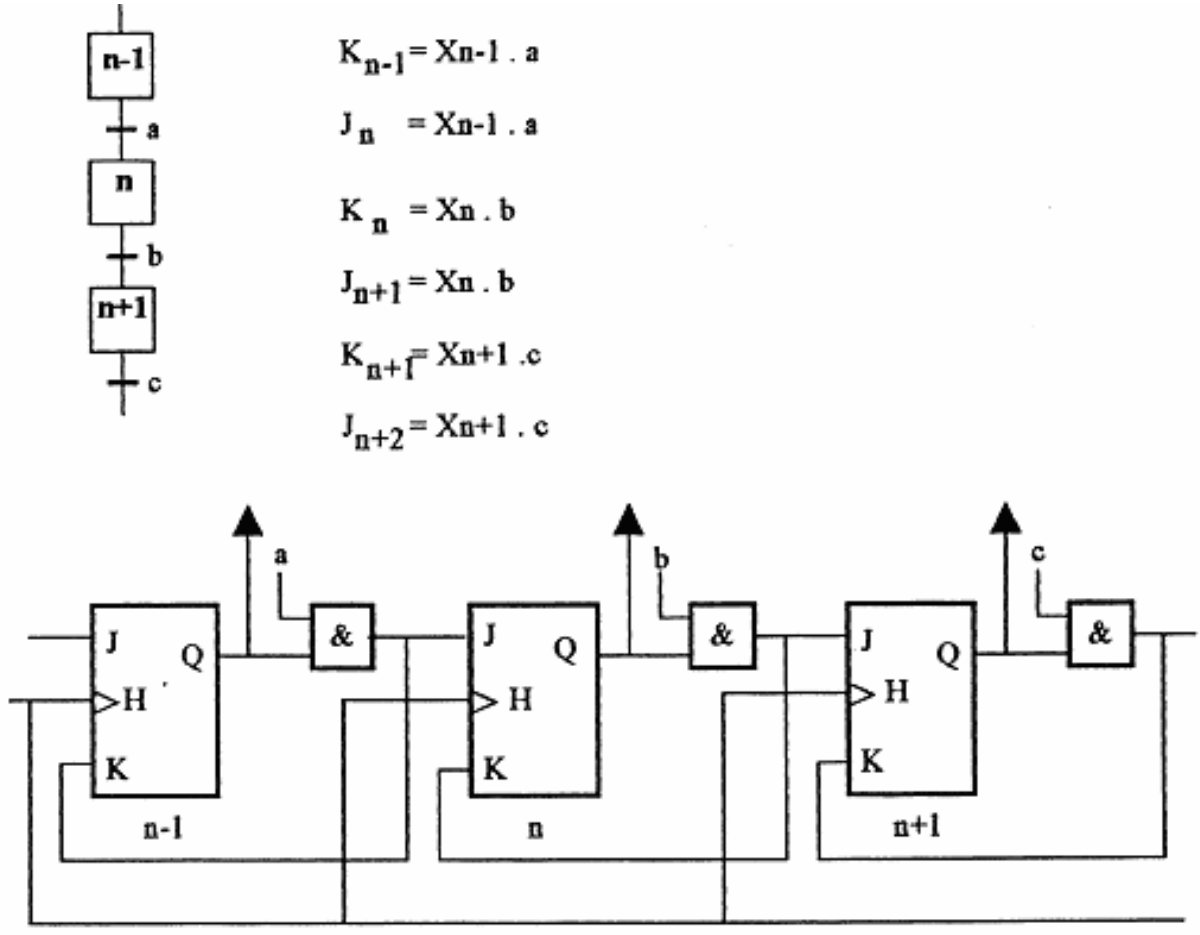


Figure 93 : Example of a synchronous sequencer with JK flip-flop

5.4.21. The forcing orders

5.4.21.1. Introduction

When designing an automated system, besides the normal operating cycle, you must take other considerations into account such as, for example, emergency shutdowns, initialisation of the command part after shutdown, etc. These constraints can be described at the level of the Sequential Function Chart by means of transitions as shown in the figure below.

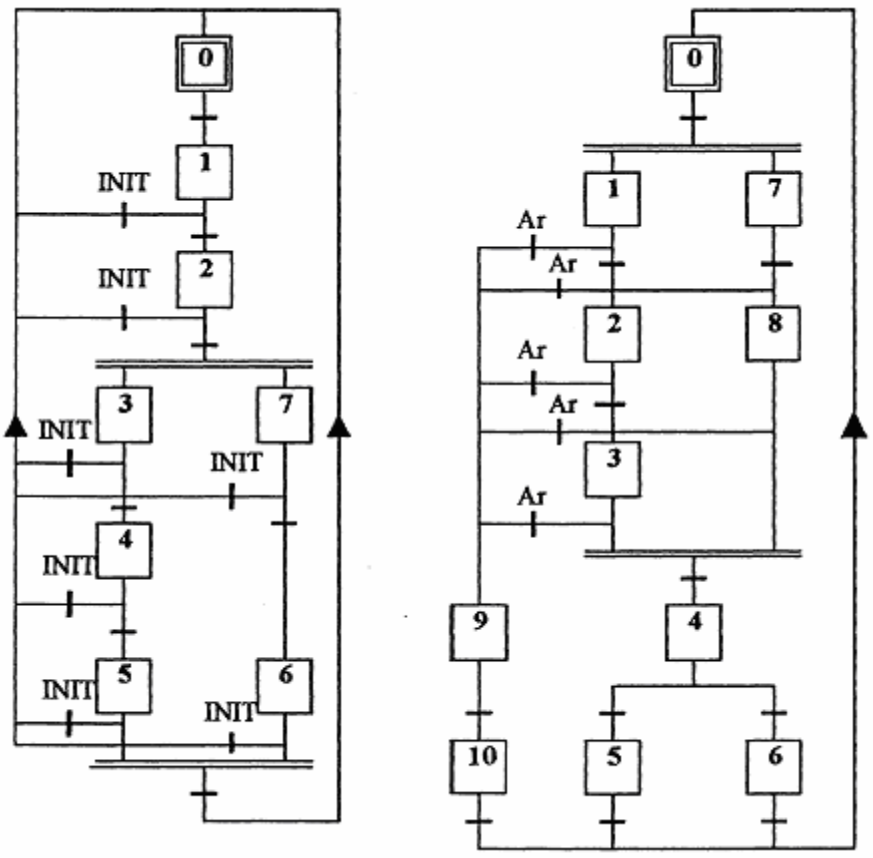


Figure 94: Forcing orders

We can straight away see that this type of representation quickly becomes very cumbersome, or even inextricable when several changes are possible due to combinations of events.

So it is preferable to define these types of behaviour by means of a hierarchical representation of the command part.

5.4.21.2. Hierarchical representation of the command part

Establishing a hierarchy for the command part is equivalent to structuring it in several levels, classified by order of importance, that is to say classified in a hierarchy. The simplest representation uses two levels as shown in the figure below.

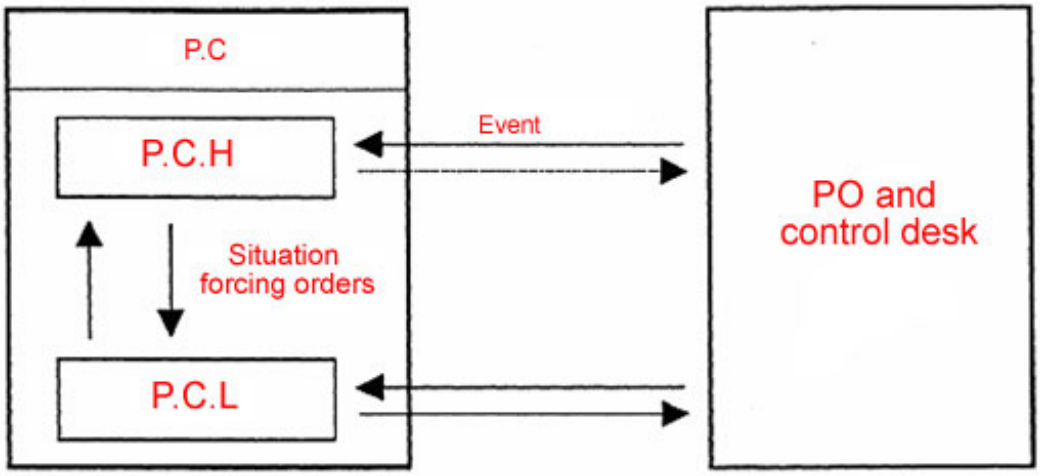


Figure 95 : Establishing the command part hierarchy

- A local level defining the command part of the process (Local PC noted PCL) processing the information delivered by the operative part (PO) and the control desk to elaborate the orders sent to them.
- A second level, defined as being hierarchically higher than the local level, forming the hierarchical command part (noted PCH). This PCH command part can receive information from the operative part, from the control desk and the local command part, and issue orders, in particular to the local command part.

In this type of structuring we must therefore define a specific order for forcing the situation of the local command part (PCL) by the hierarchical command part (PCH).

This order can be written with the following functional formulation:

Place the lower level command part (PCL) in a predetermined situation

5.4.21.3. Hierarchical forcing order

This type of forcing order will be symbolically designated by the following notation:

F \ identifier of the forced PC: forced situation

Examples:

F \ PCL : (1,10)

F \ MANAGEMENT (10,100)

The forcing order F/PCL: (1,10) means that the command part identified PCL is forced to situation (1,10) whatever its current situation.

In other words, the order F/PCL: (0,10) simultaneously causes:

- activation of steps 1 and 10 corresponding to the forced situation (1,10)
- deactivation of all the other steps of the Sequential Function Chart describing the PCL

This order has all the properties of orders and can therefore be continuous or pulsed, conditional or not. By default it is continuous: you must then note that the holding of forcing for the length of activation of the order issuing step blocks any change of the forced command part.

Remarks: the command part's identifier can be omitted if there is no ambiguity concerning the step numbers.

Example : F/(1,10, 7,100)

It is also possible to force simultaneously the situation of several independent command parts.

Example : F/PCL1: (1)/MANAGEMENT: (100)

Which corresponds to the simultaneous forcing of situation (1) of the command part PCL1 and of situation (100) of MANAGEMENT.

There is a special case corresponding to the forcing of the empty situation ( ), that is to say the deactivation of all the steps of the command part. This "deactivation" order of a command part by another hierarchically higher command part is then for example:
F/PCL2: ( )

5.4.21.4. Hierarchical description by Sequential Function Chart

An SFC description can be associated with the hierarchical representation of the command part that includes two separate levels noted **GCH** (associated with the **PCH**) and **GFN** (associated with the PCL):

- The behaviour of the **PLC** with respect to the operative part (including the control desk) is described by the normal operation graph (**GFN**).

- The hierarchical control graph **GCH** describes the behaviour of the **PCH** isolated with respect to its operative part **POH**, made up of the **PO** part (including the control desks) plus the **PCL**.

We can give the following descriptions for the two examples above:

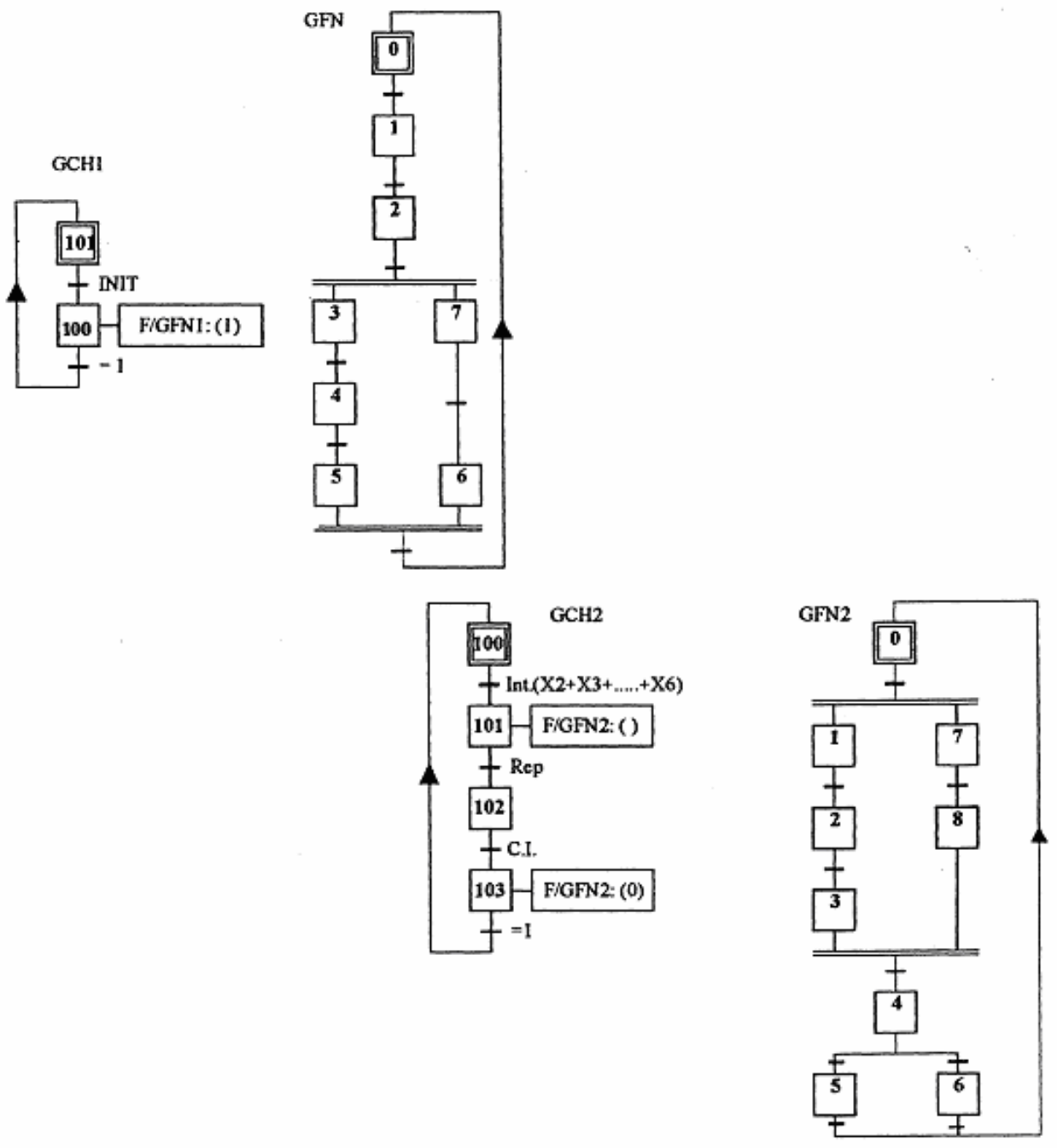


Figure 96: Hierarchical description

5.4.22. Special case of forcing: freezing

fi is often necessary for specifying a command part in a current situation. This type of specification is often associated with the On and Off modes.

This freezing in the current situation corresponds to a special type of forcing for which:

- The active steps are held at "1";
- The inactive steps are held at "0".

The desired behaviour can be obtained by introducing the freeze instruction as indication on every receptivity of the Sequential Function Chart transitions (figure a in the example below)

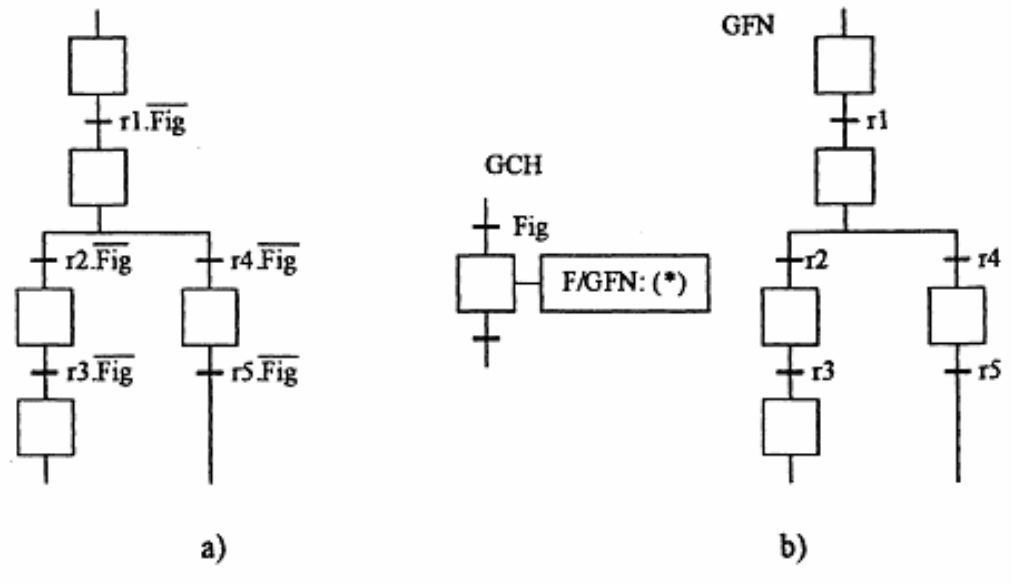


Figure 97: Freezing

However it would seem to be more judicious to obtain freezing by issuing an order from a hierarchical command part. By convention, we designate this type of order in the form:

F/GFN: (*)

The asterisk shows that we are forcing (therefore holding) the current situation of the command part described by GFN (figure b).

5.4.22.1. Conclusion on forcing and freezing orders

A forcing or freezing order can only be given by a command part hierarchically higher than the command part being commanded (order given by the "master" PC to the "slave" PC at any given moment).

Any situation can be forced, including:

- empty situation ( ) : deactivation;
- current situation (*) : freezing.

The forcing order can have all the characteristics of orders. In particular, the continuous forcing order blocks any change in the "slave" command part as long as it is given by the "master" command part.

The forcing order acts on all the steps in the "slave" command part's graph. All the steps are forced (either at activation, or at deactivation).

The forced steps may be identified by a specific initialisable step symbol:

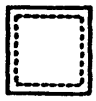


Figure 98: Forced step symbol

5.4.23. Application to the special case of On and Off modes

5.4.23.1. Analysis of an automated piece of equipment's run modes

Starting from the various specifications generally expressed in the expression of requirements, the designer can analyse the equipment's desired run modes. There are several methods for carrying out and formalising this task, the best know is GEMMA (On and Off modes analysis guidelines).

Generally speaking, we should expect the following results from this task:

- An overall definition of the desired run modes, associated with the formal identification of the various states in which the equipment can be placed;
- The formalisation of the conditions for passing from one state to the next. These conditions of course being partially the image applied to the run modes of the specification of the:
  - human/machine dialogue
  - operating manual.

- The expression of the overall constraints to be respected by the equipment for each of the envisaged states relative equally to the production imperatives and to the maintenance or safety imperatives.
- The expression of these constraints concerning both the command part and the operative part represents, generally, a summary of the analysis carried out previously: safety, production management, etc.

Example: semi-automated drilling

The simplified example (figure below) consists, for the operative part, of a drill spindle whose actuators ensure rotation, lowering and raising. The installation or removal of the part to be drilled is carried out by hand as is the installation of the protection cover.

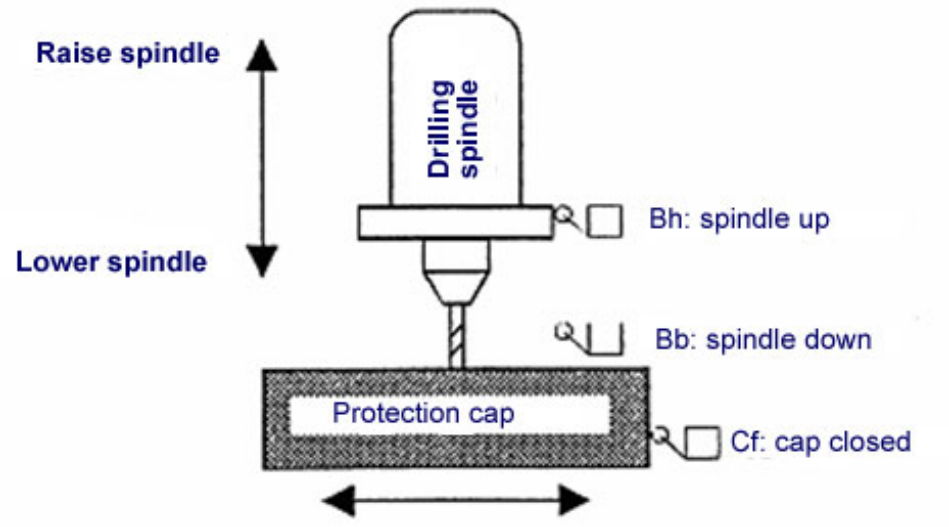


Figure 99 : Example of run mode for an automatic drill

The description of the run modes, which takes into account the production and safety requirements, provides for two main modes:

- Automated mode: → state 1 and state 2,
- Failure mode: → state 3 and state 4.

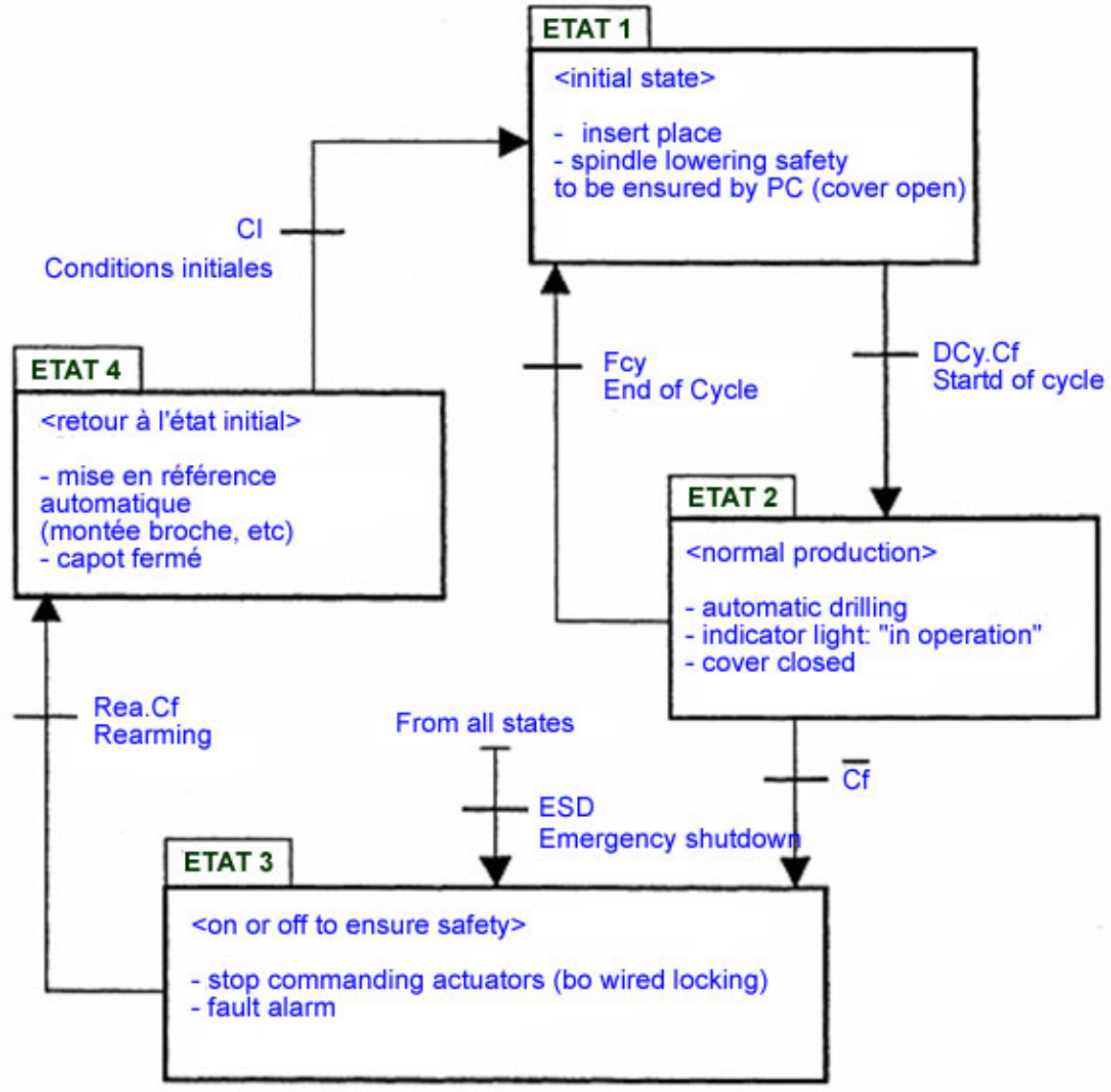


Figure 100: Description of the run modes

State 1: in this state it is possible to put the part in place or remove it, the command part must ensure the safety of spindle lowering as long as the protection cover is open.

State 2 : the "start cycle" button makes it possible to pass to state 2 in which automatic drilling is carried out, the end of the cycle causes the return to state 1. Opening of the cover in state 2 takes us to state 3.

State 3 : the "emergency shutdown" information takes us to state 3 from any other state. In this state the actuators are ordered to stop (plus locking wired directly on the service device).

State 4 : if the cover is closed, the "reset" information makes it possible, in state 4, to return the automatic equipment to its reference position. Once the initial conditions have been verified, the equipment will reach the state

5.4.23.2. Sequential Function Chart and run modes

The monitoring of the run modes is an easy command to describe in a Sequential Function Chart.

Indeed, it is always possible to make a **Sequential Function Chart** reflecting the equipment's possible run modes in which each step represents a state defined in the automated system, and for which each receptivity associated with a transition materialises the conditions for changing from one state to the other.

The Sequential Function Chart (see figure) shows an example of run mode monitoring:

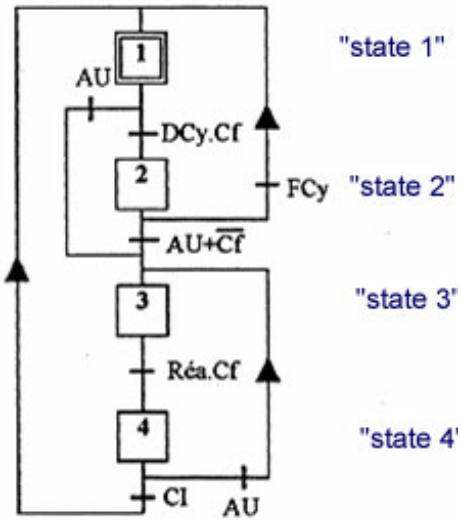


Figure 101: Example of run mode monitoring

This first approach must be refined and completed to effectively ensure management of the run modes by the command part.

Management of the run modes by the command part.

Management of the run modes consists of taking into account all the specifications that must be implemented by the command part (GEMMA is one possible form) of those specifications).

This can advantageously be accomplished by including the run modes amongst the structuring criteria chosen for the command part.

Structuring according to the run modes.

If in this case the modelling is done using the Sequential Function Chart tool, it is often desirable to use the forcing orders and the resulting hierarchical levels of the Sequential Function Chart in order to structure the command part.

Example (cont'd):

The application of these principles to the semi-automatic drilling example is represented by the Sequential Function Chart below.

This overall Sequential Function Chart is structured around run modes and includes the following partial Sequential Function Charts:

- GMM, of a higher hierarchical level, ensures the management of the run modes by forcing the two lower level partial Sequential Function Charts, GPN and GIP. (The choice of the initial step 3 is justified by the safety constraints when putting into service.)

- GPN, which does not necessarily include an initial step because of the forcing, performs the semi-automatic drilling cycle as soon as step 2 (of GMM) is active.

- GIP, which does not necessarily include an initial step because of the forcing, returns the operative part to its reference position on activation of step 3.

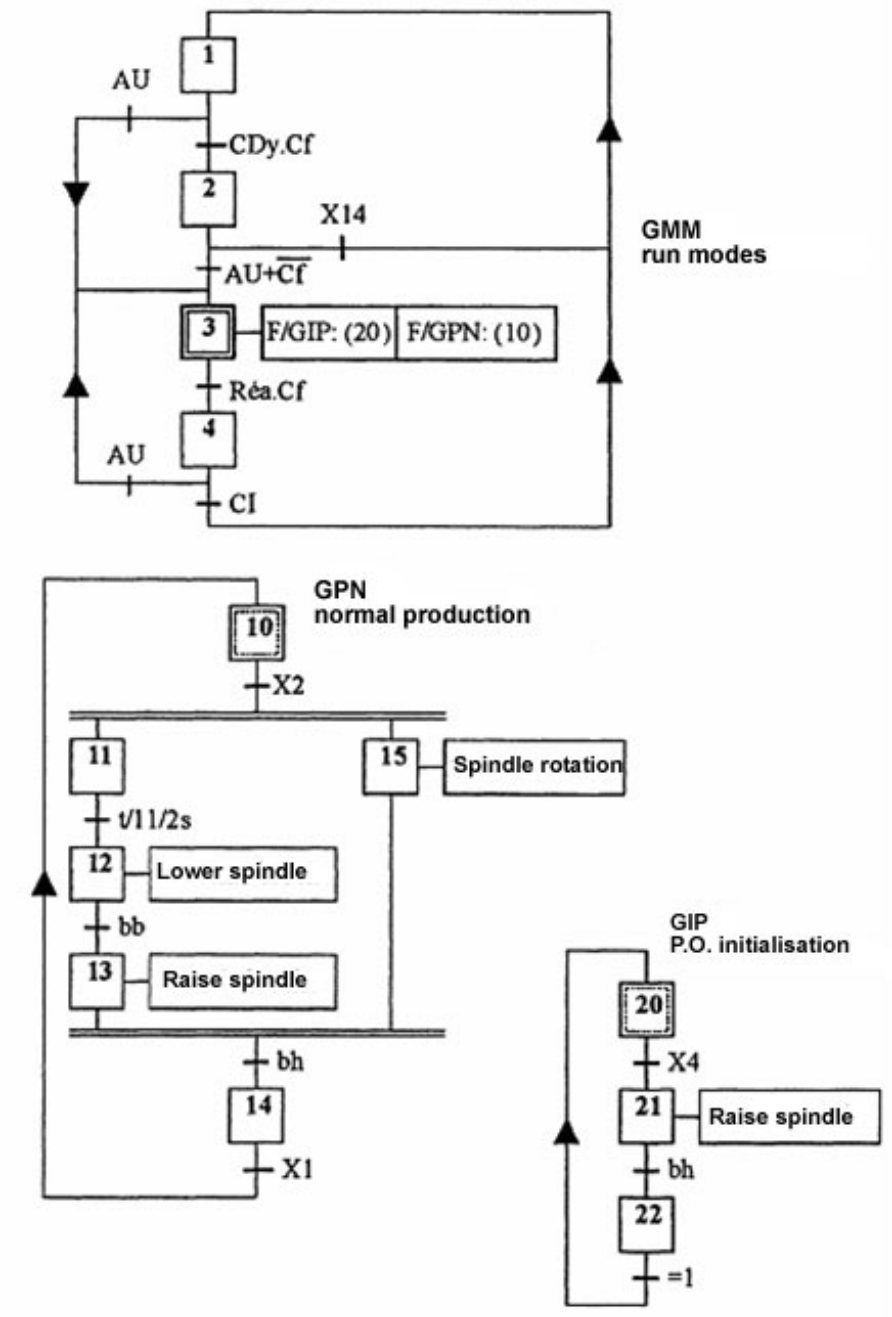


Figure 102 : Sequential Function Chart for the automatic drill

5.5. INSTRUMENT LIST (IL) LANGUAGE

The 'Instruction List' (IL) programming language is a language close to assembler.

The structure of IL language is as follows:

Label	Operator	Operand	Comments
<Label> :	<Operator>(<Modifier>)	<Operand>	(* Accumulator = Accumulator <Operator>(<Modifier>)<Operand>*)
FOOBAR:	ADD	25	(* Accumulator = Accumulator + 25 *)

The **label** always ends with the character " **:** ". It is optional and it precedes an instruction line. The label is used at the time of conditional jumps, or not.

The operator corresponds to the operation to be performed between the **accumulator** and the **operand**. The result is stored in the accumulator. A **modifier** may be associated with the operator. **The operand** is a constant or a variable.

It is possible to associate a **comment** with each instruction line. The comment starts with the two characters " **(** * ". The comment ends with the two characters " * **)** ".

5.5.1. Instructions

Operator	Description	Operator + Modifier	Description
Assignment operators			
LD	Accu = Operand (Op)	**LDN**	Accu = not Op
ST	Op = Accu	**STN**	Op = not Accu
S	If Accu = 1 then Set Op		
R	If Accu = 1 then Reset Op		
Logic operators			
AND	Accu = Accu AND Op	**ANDN**	Accu = Accu AND not Op
OR	Accu = Accu OR Op	**ORN**	Accu = Accu OR not Op
XOR	Accu = Accu XOR Op	**XORN**	Accu = Accu XOR not Op
NOT	Accu = not Accu		

Arithmetic operators			
ADD	Accu = Accu + Op		
SUB	Accu = Accu - Op		
MUL	Accu = Accu * Op		
DIV	Accu = Accu / Op		
MOD	Accu = Remainder of Accu / Op		
Comparison operators			
GT	Accu = true if Accu > Op		
LT	Accu = true if Accu < Op		
EQ	Accu = true if Accu = Op		
GE	Accu = true if Accu >= Op		
LE	Accu = true if Accu <= Op		
NE	Accu = true if Accu <> Op		
Branches			
JMPC	Jump to Label if Accu <> 0		
JMP	Jump to Label		
RET	"Return" Instruction		
RETC	"Return" if Accu <> 0		
JMPCN	Jump to Label if Accu = 0		
CAL	Call a function block		
RETCN	"Return" if Accu = 0		

Table 2: "Instrument List" Instructions

5.5.1.1. Modifier

The "(" modifier can be associated with the logic, arithmetic and comparison operators (for example "ADD("), it causes opening of a parenthesis through the creation of a second accumulator.

The ") " operator closes the parenthesis.

Example : Res = (A + B) * (C + D)

```
LD      A       (* Accu = A *)
ADD     B       (* Accu = Accu + B = A + B *)
MUL(    C       (* Opening of parenthesis, Accu2 = C *)
ADD     D       (* Accu2 = Accu2 + D = C + D *)
)               (* Closing of parenthesis Accu = Accu * Accu2 = (A + B) * (C + D) *)
ST      Res     (* Res = Accu *)
```

5.5.2. Using functions

Functions are used in the following way:

```
LD                  <First argument>
<Function Name>     <Second argument>, <Third argument>, …
ST                  Res
```

It is essential to comply with the order of the arguments.

Examples:

Res = SIN (Ang)

```
        LD      Ang         (* Accu = Ang *)
        SIN                 (* Accu = SIN (Accu) = SIN (Ang) *)
        ST      Res         (* Res = Accu *)
```

Res = LIMIT (Min, Val, Max)

```
        LD      Min         (* Accu = Min *)
        LIMIT   Val, Max    (* Accu = LIMIT (Accu, Val, Max) = SIN (Ang) *)
        ST      Res         (* Res = Accu *)
```

5.5.3. Using functional blocks

Functional blocks are used in the following way:

```
CAL    <InstanceName>(<Input1> : = <Value>, <Input2> : = <Value> , …)
LD     <InstanceName>. <OutputName>
ST     Res
```

An instance of each functional block must be created.

Examples:

Let Tempo be an instance of the "TON" functional block:

```
CAL    Tempo(IN := En, PT := T # 5s    (* Execution of the Tempo instance *)
LD     Tempo.Q                         (* Accu = Tempo . Q *)
ST     Out1                            (* Out1 = Accu *)
```

In the example below we have used the "GE" standard comparison function (greater than or equal to operator) and the "TON" standard functional block (Time on delay), Tempo instance, in order to generate an Alarm if the temperature of the oven remains higher than or equal to the maximum authorised temperature for more than 5 seconds.

```
LD     OvenTemp                        (* Accu = OvenTemp *)
GE     MaxOvenTemp                     (* Accu = (OvenTemp >= MaxOvenTemp) *)
ST     S1                              (* S1 = Accu *)
CALL   Tempo(IN := S1, PT := T # 5s    (* Execution of Tempo instance *)
LD     Tempo.Q                         (* Accu = Tempo.Q *)
ST     Alarm                           (* Alarm = Accu *)
```

5.5.4. Examples of Ladder correspondence in List

Phrase List

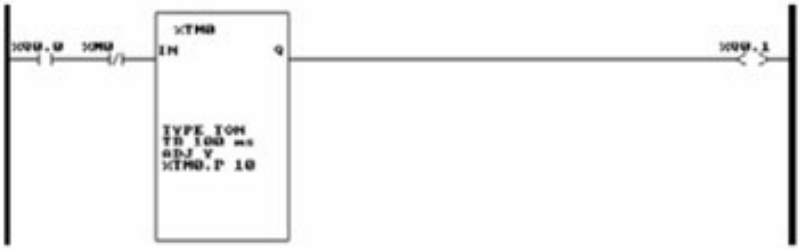
```
LD %I0.0
OR %Q0.0
ANDN %I0.1
ST %Q0.0
```

Network correspondance 0

Phrase List

```
BLK %TM0
LD %Q0.0
ANDN %M0
IN
OUT_BLK
LD Q
ST %Q0.1
END_BLK
```

Network correspondance 1



Phrase List

```
BLK %TM1
LD %Q0.1
IN
OUT_BLK
LD Q
ST %M0
END_BLK
```

Network correspondance 2

Figure 103: Ladder correspondences in List

5.6. STRUCTURED TEXT LANGUAGE (ST)

Like C language, "Structured Text" (ST) is a structured programming language. It is a high-performance language suited to automation systems.

The general structure of an instruction is as follows:

<Variable> :=<operand><operator><operand>… ;

Each instruction ends with the " ; " character

5.6.1. The operators

Operator	Description	Operator	Description
Assignment operator			
:=			
Logic operators			
NOT	Logic negation	**AND**	Logic AND
OR	Inclusive logic OR	**XOR**	Exclusive logic OR
Arithmetic operators			
+	Addition	**-**	Subtraction
*****	Multiplication	**/**	Division
mod	Remainder of integer division		
Comparison operators			
=	Equal to	**<>**	Different from
>	Greater than	**>=**	Greater than or equal to
<	Lower than	**<=**	Lower than or equal to

Table 3: The "Structured Text" operators

5.6.2. The control structures

5.6.2.1. Conditional instructions

Pseudo language	Structured Text	C language
If *(logic evaluation)* **Then** *Action block 1* **Else If** *(logic evaluation)* **Then** *Action block 2* …….. **Else** *Action block 3* **End If**	**IF** *logic evaluation* **THEN** *Action block 1 ;* **ELSIF** *logic evaluation* **THEN** *Action block 2 ;* …….. **ELSE** *Action block 3 ;* **END_IF ;**	**if** (*logic evaluation*) **{** *Action block 1 ;* **}** **else if** (*logic evaluation*) **{** *Action block 2 ;* **}** …….. **else** **{** *Action block 3 ;* **}**

Table 4: Conditional instructions

5.6.2.2. Choice instructions

Pseudo language	Structured Text	C language
Decide on *(expression)* **Between** *Val_1* **:** *Action block* …….. *Val_2, Val_3, …* **:** *Action block* …….. *Val_i, Val_j, …* **:** *Action block* …….. **Default :** *Action block* **End of Decide**	**CASE** *expression* **OF** *Val_1* **:** *Action block* …….. *Val_2, Val_3, …* **:** *Action block* …….. *Val_i, Val_j, …* **:** *Action block* …….. **ELSE** *Action block;* **END_CASE ;**	**switch** (*expression*) **{** **case** *Val_1* **:** *Action block;* ***break*** *;* …….. **case** *Val_2* **:** **case** *Val_3* **:** **case ….** *Action block;* ***return(n)*** *;* …….. ***No correspondence in C*** …….. **Default :** *Action block 3 ;* **}**

Table 5: The choice instructions

Structured Text	Description
CASE expression **OF**	Start of choice instruction
Val_1 **:**	Do if expression = Val_1
Val_2, Val_3, … **:**	Do if expression = Val_2 or Val_3 ou …
Val_4 .. Val_5 **:**	Do if i Val_4 <= expression <= Val_5
ELSE	Do else
END_CASE	End of choice instruction

Table 6: Choice instructions and their description

5.6.2.3. Repetitive instructions

Pseudo language	Structured Text	C language
As long as *(logic evaluation)* *Action block* **Repeat**	**WHILE** *logic evaluation* **DO** *Action block;* **END_WHILE**	**while (***logic evaluation***)** **{** *Action block;* **}**
Repeat *Action block* **As long as** *(logic evaluation)*	**REPEAT** *Action block;* **UNTIL** *logic evaluation* **END_REPEAT ;**	**do** **{** *Action block;* **}** **while (***logic evaluation***) ;**
For *variable* **=** *initial value* **Until** *final value* **By** *increment* **Do** *Action block* **End For**	**FOR** *variable* **:=** *initial value* **TO** *final value;* **BY** *increment* **DO** *Action block;* **END_FOR ;**	**for (***variable* **=** *initial value***;** *variable <> final value***;** *variable += increment***)** **}** *Action block;* **}**

Table 7: Repetitive instructions

CAUTION: the utilisation of repetitive structures is very dangerous in the case of cyclic or periodic tasks.

5.6.2.4. EXIT instruction

The "EXIT" instruction makes it possible to interrupt a repetitive loop.

Structured Text	C language
Start loop Action block; **EXIT ;** Action block; End loop;	Start loop Action block; **break ;** Action block; End loop;

Table 8: EXIT instruction

5.6.2.5. RETURN instruction

The "RETURN" instruction makes it possible to interrupt a program organisation unit such as a function, a functional block or a program. Unlike C language, the "RETURN" instruction does not make it possible to return a value.

5.6.3. Using functions

Functions are used in the following way:

<FunctionName>(<First argument>, <Second argument>, …)

It is essential to comply with the order of the arguments.

Examples :

```
Res := SIN(Ang) ;          (* Execution of the Sine function *)
Res := LIMIT(Min, Val, Max)  (* Execution of the Limit function *)
```

5.6.4. Using functional blocks

Functional blocks are used in the following way:

<InstanceName>(<Input1> := <Value>, Input2> := <Value>, …) ;

An instance of each functional block must be created.

Examples:

Let Tempo be an instance of the "TON" functional block:

```
Tempo(IN := En, PT := T # 5s) ;      (* Execution of the Tempo instance *)
Out1 := Tempo.Q ;                    (* Utilisation of the outputs from the functional block *)
```

In the example below we have used the "GE" standard comparison function (greater than or equal to operator) and the "TON" standard function block (Time on delay), Tempo instance, in order to generate an Alarm if the oven temperature remains higher than or equal to the maximum authorised temperature for more than 5 seconds.

```
IF OvenTemp > MaxOvenTemp
THEN
        Tempo(IN := TRUE, PT := T # 5s) ;
ELSE
        Tempo(IN := FALSE, PT := T # 5s) ;
END_IF
Alarm := Tempo.Q ;
```

6. INDUSTRIAL LOCAL AREA NETWORKS

It can be said that local area networks appeared about forty years ago, and they are used to transfer data between machines. They used to meet the need to transfer data (often basic) between a calculator and PLCs or between a calculator and measuring instruments.

We can mention two very well known networks that we will look at briefly in the following chapters: HPIB (IEEE 488) and RS 232. However with the improved performance of electronic applications, leading to the development of Information Technology as we know it today, these network quickly became outdated and, although the RS 232 and HPIB standards are still used today, their utilisation must really be put into perspective.

Today, networks represent an everyday aspect of Information Technology.

ETHERNET is the most widely used network in industrial contexts. However, there are many applications where ETHERNET cannot be used, for example, for the transfer of simple information, in real time, between PLCs and computers, or worse still between sensors and a PLC. This is where the new industrial local area networks come into their own.

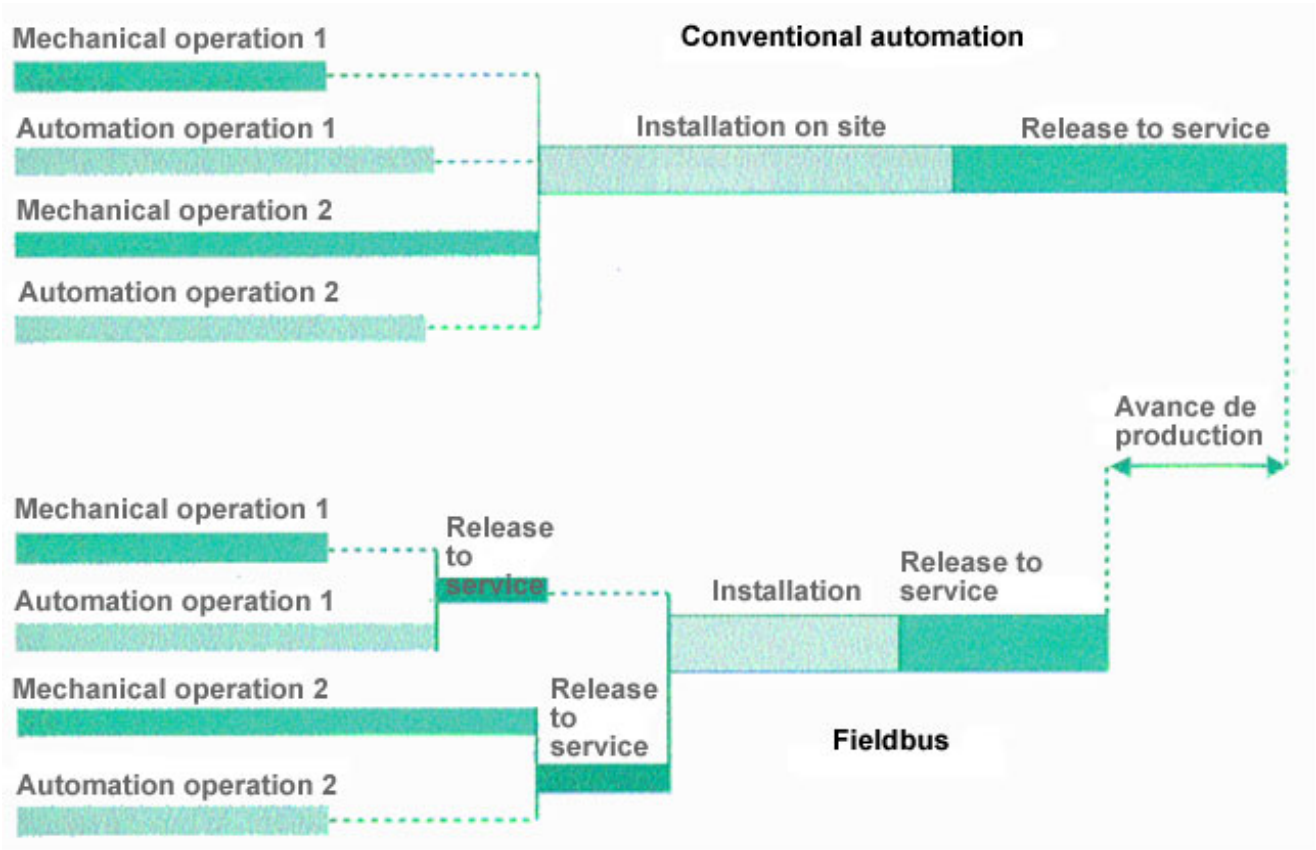


Figure 104 : Example of costs minimisation through implementation of networks

In general, when using a network, we are looking for a simple and transparent approach to communications, reduced manufacturing costs (insofar as less wiring is required and the development costs are often lower) and, above all, a standard, available for all industrial manufacturing applications.

And this is where the main problem lies with local area networks: the multiplication of standards often makes the choice complex, and then the only things that count are the application and hardware available. Indeed, the manufacturers of PLCs are often also the manufacturers of LANs, hence a perfect compatibility between machines, and by extension an incompatibility with machines made by other manufacturers.

Fortunately, certain systems seem effectively to be becoming standards, that is to say standards by force of circumstance. And there will not be many winners at this little game, even though CAN would seem to have become the standard in terms of on-board networks (in the automotive and aeronautical sectors). All the same, the future is not clear.

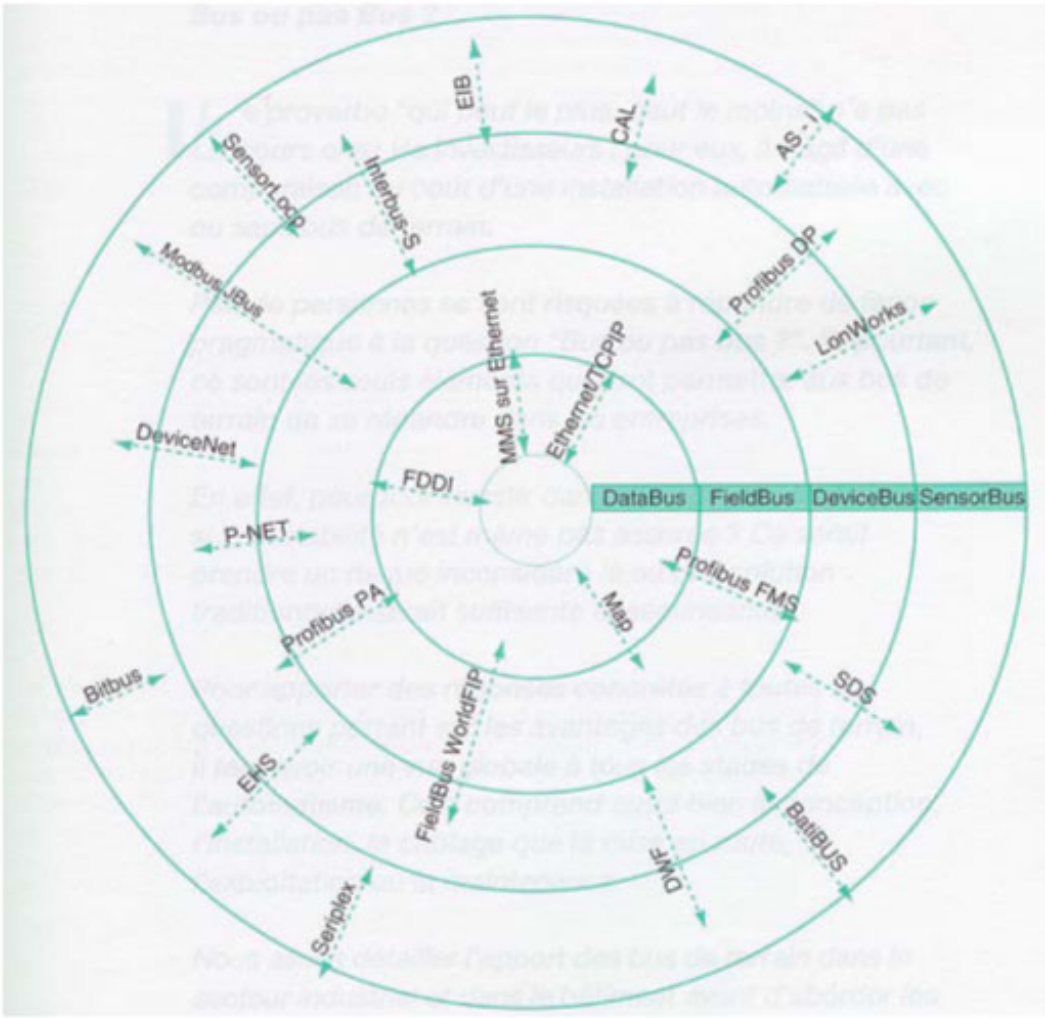


Figure 105 : The different types of network

We can therefore hope that, like in the IT sector, we will see not one but a set of standardised networks, accessible to all manufacturers, appear in the forthcoming years.

The best known networks and preferred application areas are:

- **DATABUS:** networks responsible for transferring large amounts of data over long distances (+1000km), without the notion of real time.
- **FIELDBUS:** networks that make it possible to control the structures, that is a relatively small amount of data to be circulated over relatively large distances (1km), with a notion of real-time event.
- **DEVICEBUS:** local networks that carry small data flows over short distances (100m) in real time.
- **SENSORBUS:** local networks that carry events over very short distances (10 to 100m) in real time.

6.1. THE INFORMATION TRANSMISSION TECHNIQUES

As we have seen there are many problems when it comes to transmitting an item of information. We therefore tend to use tried and tested methods, derived from the telecommunications sector. But technological developments have seriously put into question this not particularly innovative view of things.

Before examining the methods used to transmit a signal in baseband or in bandshift (we will see these terms later on), it should be understood how a connection is made between two machines, whether they are close or distant.

As we have seen earlier on, noise is a very serious enemy of transmission. In the next chapter we will see that this is not the only enemy and also that it can be combated.

6.1.1. Connection techniques

Connecting two machines together would seem to be simple at first glance. There are three connection methods:

- connection with one wire,
- connection with two wires and ground,
- connection with two differential wires.

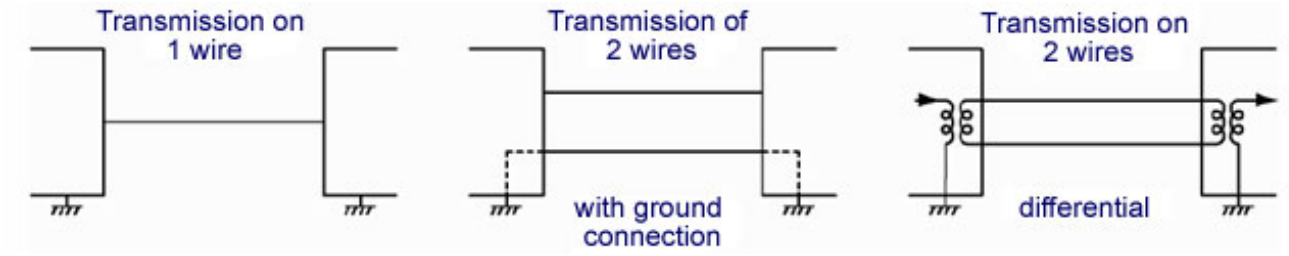


Figure 106: The connection techniques

6.1.1.1. Single-wire connection

In the case of transmission on a single wire, we play on the fact that the ground is the same for both machines, which can only be true for extremely short distances (on the same electronics board for example).

This type of connection cannot be used for long distances. For example, assuming that chassis ground and earth are connected, if we installed this type of wiring between Europe and the USA, the potential difference between the two earths would exceed 100V.

Furthermore, this type of connection would by extremely sensitive to noise.

6.1.1.2. Two-wire connection with chassis ground

The immediate temptation for tackling the problem of single-wire connections would be to add a second connection to the chassis ground, both to solve the problem of the difference of potential between the two connections to earth, and also to protect the signal, for example by providing shielding.

This technique has its advantages (data protection, for example) and many drawbacks such as noise on the ground (in the case of long links), the intercontinental difference of potential which could create strong currents in the wires or quite simply a risk of lightning on the line which could destroy both the transmitter and the receiver.

6.1.1.3. Differential two-wire connection

To effectively combat noise, we virtually always use a solution that consists of transmitting in differential mode. Differential mode consists of sending the information on 2 lines.

It is important that you understand that the information is not redundant, it is split. We thus create a "positive logic" channel and a "negative logic" channel, the information being the subtraction of these two signals.

In the case of disturbance, both wires in the link are affected nearly at the same time by the disturbance and with nearly the same degree of noise power.

If, at the departure of the line, we have an information signal of amplitude 2A. At the line's arrival we therefore have, on the one hand on the first channel the data (A) and noise (B) "half-signal" and on the other channel , the same "half-signal" but, this time, inverted (-A) and noise (B). By subcontracting the signals of the two channels we obtain the output O:

$$O = (A + B) - (-A + B) = A + B + A - B = 2 \square A$$

As we no longer transmit the ground, the information is floating, without a reference. On arrival, the subtraction of the 2 signals is made, with the local ground reference, which fixes the signal's potential at its final destination, which eliminates the problem of the difference of potential between the grounds.

This type of link is used in a large number of medium- and large-scale networks (ETHERNET) or in networks exposed to an extremely noisy environment (CAN).

Now that we know the techniques that can be used to connect two machines together to transmit information, we must look at the modifications that must be made to the data to allow them to circulate on the line.

We have seen that differential transmissions use transformers. It is therefore essential that the signals transmitted should have a null continuous component, which is not the case with conventional "binary" signals. We will therefore have to modify these signals so that they can be transmitted, and also limit the spectrum of the signals and thus attempt to increase the number of channels in a link.

For many years, we used analog methods – modulations – to transmit several signals on one line. Now that IT and digital systems function at high speeds, we have naturally turned to digital transmission techniques.

Starting from their, we have designed systems capable of compressing the signal on ever-smaller frequency ranges, and these methods are grouped together under the name of baseband transmission.

6.1.2. Baseband transmission

The principle of baseband transmission consists of modifying the signal's spectrum, without shifting it into another frequency range, by playing on amplitude parameters or by associating different signals.

6.1.2.1. Polarities

We say of a code that it is unipolar when the coding of the information involves one electrical level (in addition to ground). We say that a code is bipolar when the coding of the information involves two electrical levels to code the information (ground can be used as a 3rd level). Bipolar codes generally eliminate the continuous component from the transmitted signal.

Examples:

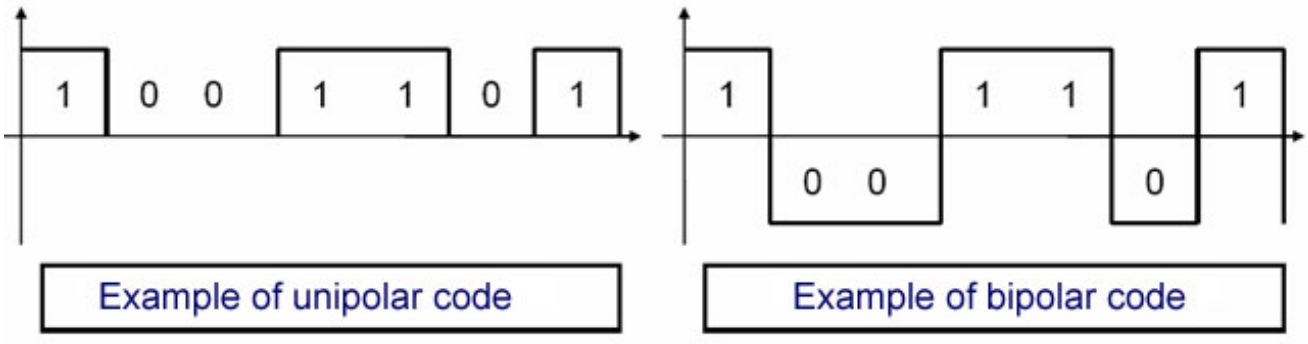


Figure 107: Polarities

Return to zero

The transmission systems also act on the way of coding a signal in terms, not of the voltage level, but of the change over time. The code that we call binary ('0' = 0V and '1' = +5V) is in fact a unipolar (the level of the voltages), NRZ (organisation in time) code.

NRZ means Non Return to Zero, that is to say: which does not change state over the duration of a bit.

It is in fact the contrary of RZ code (Return to Zero) which automatically creates a return to zero state over the duration of a bit.

Example:

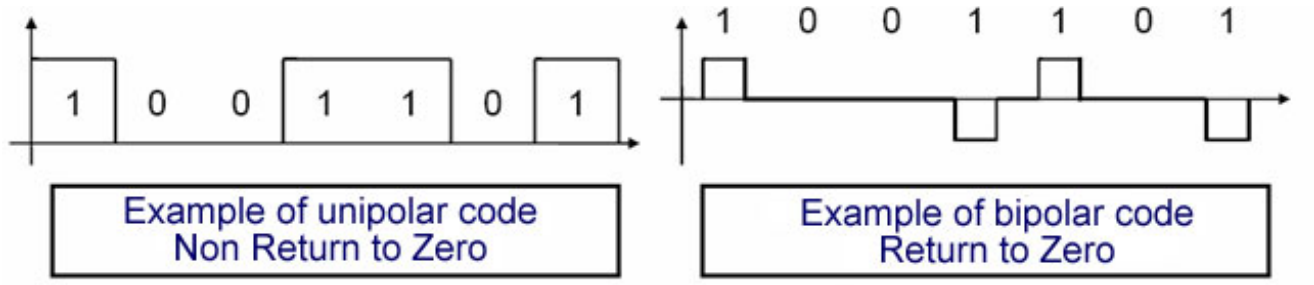


Figure 108: Return to Zero

We can also mention NRZI codes (Non Return to Zero Inverted). These codes generate changes in the signal's level at the output from the coder on a given level of the input signal. For example, the NRZI-S code (Non Return to Zero Inverted on Space) changes the state of the output signal whenever the bit presented on the input to the coder is at '0' (zero being considered to be a space).

6.1.2.2. Asynchronous codes

The main problem with asynchronous codes is, in fact, suggested in their name: the absence of a clock.

The respective clocks of the transmitter and of the receiver not being, by definition, the same they beat at a rhythm that is close but different. So we must manage to regularly resynchronise the two machines.

It is the edges on the data signal that are used to reset the receiver clock with respect to the transmitter clock.

But, the larger the amount of data being transmitted, the greater the probability of having a long "blank", that is to say an absence of edge.

That is why it is extremely rare for asynchronous codes to be used unless the messages to be transmitted are small (even if there are a lot of small messages in succession). We tend to use synchronous codes once large amounts of data have to be transmitted.

6.1.2.3. Synchronous codes

The principle of synchronous codes is based on the mixing of the clock information with the data signal to create a signal that has an easily usable clock and from which the data can easily be extracted.

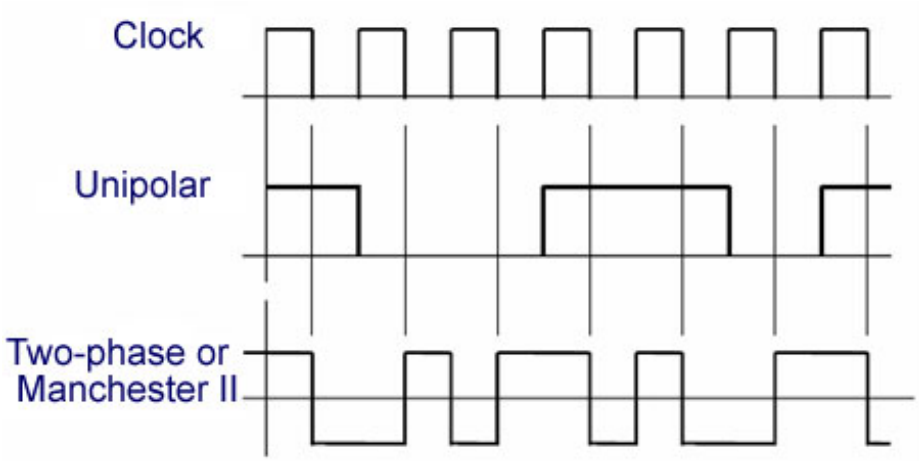


Figure 109: Synchronous code

Manchester II code

This code is quite frequently used in networks; it is a two-phase, bipolar code. The principle of Manchester coding consists of translating the '1' as a falling edge and the '0' as a rising edge. To achieve this, we use the EXCLUSIVE NOR logic function between the clock and the data signal.

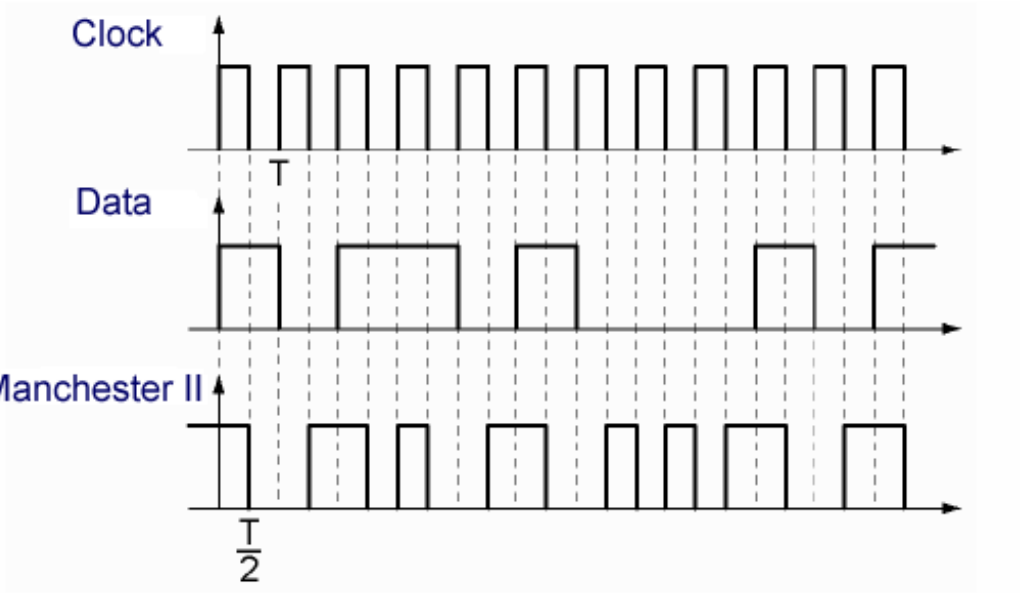


Figure 110: Manchester II code

This signal may seem quite complicated to analyse, but in fact it is very simple. All the odd numbers of times T/2 (T/2, 3T/2, 5T/2, …), there is systematically an edge. If this edge is rising, it is a logic '0', if the edge is falling, it is a logic '1'.

Occasionally, there are fronts on the signal after an even number of half-periods. This phenomenon occurs when there are consecutively two bits with the same value. Indeed, in order to be able to have two falling edges, there must absolutely be a rising edge between the two.

Besides the fact that the signal coded with Manchester does not have a continuous component, it considerably broadens the spectrum of the NRZ signal to the extent where it occupies a frequency band situated between 2F/3 and 3F/2 (F is the frequency of the transmission clock). Its spectrum therefore inevitably contains one component at the clock's frequency F.

As we have seen, this type of code is extremely simple to make, but its decoding, on the other hand, is slightly more difficult. You must re-create a clock (the complicated part of the work) before passing this clock through the same EXCLUSIVE OR gate to recover the data.

The clock can only be regenerated by eliminating the transitions at the multiple times of T, and only keeping the edges at T/2. Originally, a system based on non-resettable flip-flops was used (before being replaced by digital techniques) to eliminate the edges at T. Then a PLL was used, with the signal extracted from the flip-flops, to re-construct a clock with the same frequency as the transmitter clock and in phase with it. This clock was then used to recover the data. The "slowness" of the PLL's synchronisation then required long salvoes of data to enable the receiver to synchronise itself. For Ethernet (which uses Manchester II coding), there are 7 bytes used as a salvo.

Miller code

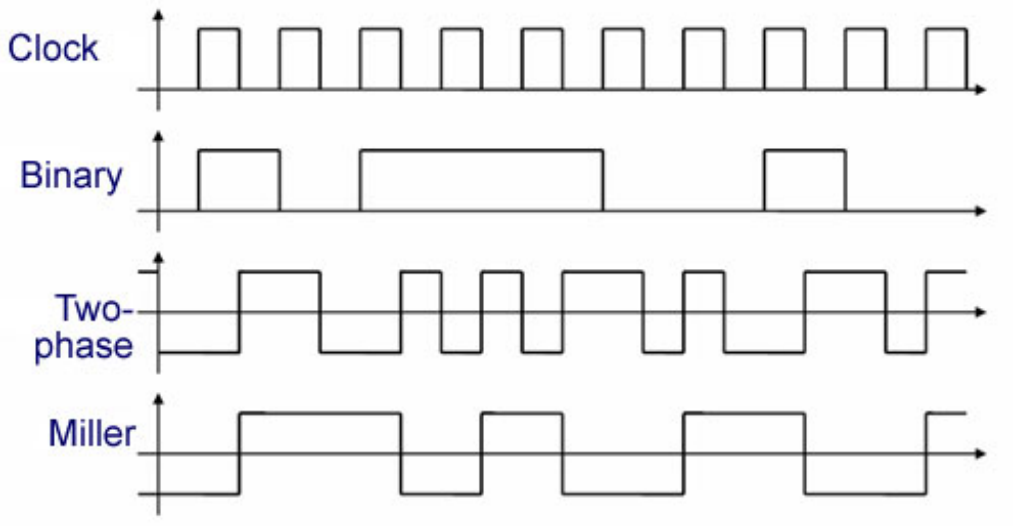


Figure 111: Miller code

Miller code is a code derived from a conventional two-phase code; it meets the need to transmit a signal in a narrower band. To make Miller code, we pass the two-phase code through a division-by-two flip-flop. Decoding is of course much more complicated in this case.

HDB3 code

HDB3 code is the code used in telephony (not at user level, but at the level of telephone exchanges and international calls). HDB3 code is a bipolar, return to zero, alternating (the bits at '1' are alternately positive and then negative), asynchronous code. It makes it possible to transmit extremely long frames, without any possibility of desynchronisation of the transmitter and receiver.

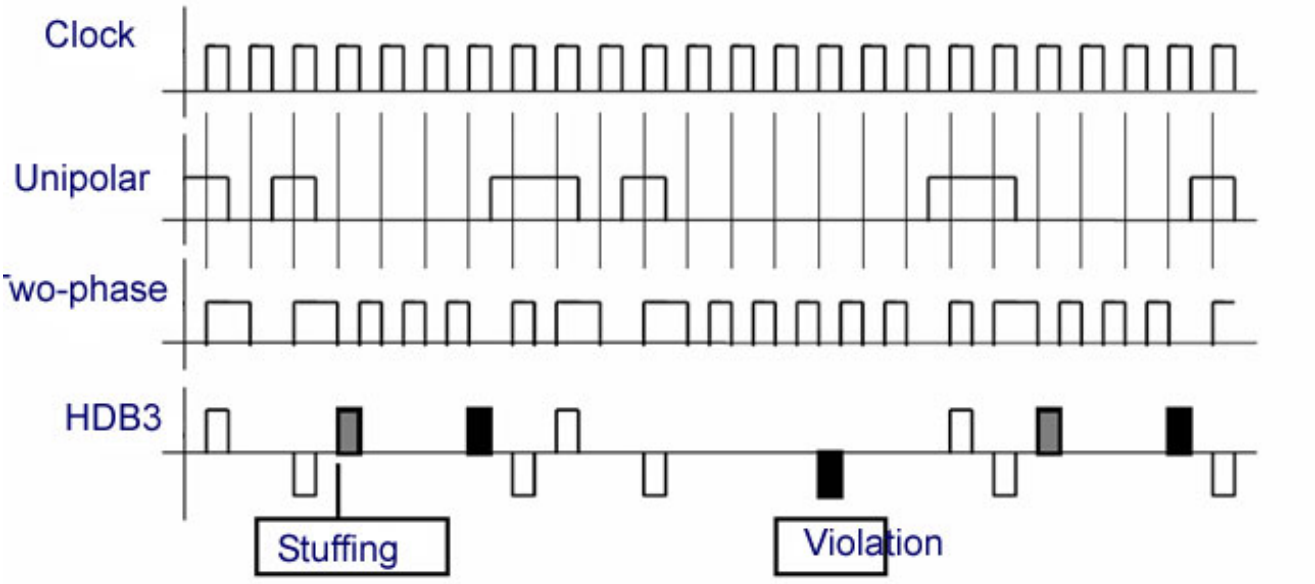


Figure 112: HDB3 code

The principle used here is called bit stuffing, it consists (in the case of HDB3code) of replacing bits at '0' with bits fictitiously at '1'. We then use the term replacement bit stuffing. In the case of other networks, there is no replacement, but the addition of a bit at '1'. We then use the term adding bit stuffing.

In HDB3 code, if a series of more than 3 consecutive zeros appears on the data line (hence the 3 in HDB3), we replace the fourth '0' with a '1' which we call violation bit.

In order to be able to distinguish immediately between a violation bit and a bit really at '1', the violation bit systematically has the same polarity as the last bit at '1' (whether it is a bit really at '1' or a stuffing bit). This violation bit regulates the alternation.

However, this non-compliance of alternation could, in the case of very long series of '0', generate the appearance of a DC voltage (the violation bits all being in the same direction). To compensate for them, we therefore introduce so-called stuffing bits to force alternations.

Stuffing bits respect the alternation rule. We insert them in the transmission if the total number of bits at '1' since the last violation bit is even. So, in the case of long series of '0', the stuffing bits eliminate the direct component by alternating.
Decoding of this system is simple, it consists of identifying the violation bits (which is easy as they do not respect the alternation rule) then, eliminate the stuffing bits, you just have to know that a violation bit is preceded by a series of three '0'…

6.1.3. Band shift transmission

Band shift transmission consists of processing the information to modify its frequency and thus shift its spectrum to a given place. This operation is called modulation.

Indeed, if several people "talk" at the same time and in the same place, it is quite difficult to distinguish between what one person says and what the other people are saying. In the case of radio, a large number of transmitters "talk" at the same time but, fortunately for our ears, not at the same frequencies thanks to the modulations.

Furthermore, the voice cannot be transmitted well over long distances (the atmosphere tending to damp it very quickly). The modulation techniques therefore make it possible both to find the frequency that is best suited to transmission in the atmosphere, and at the same time authorise several transmitters to talk at the same time.

Although not very widely used in the world of wired networks, modulation has recently made a remarkable entry into the world of networks with the arrival of wireless technologies.

Because this transmission technique is quite specific to the world of telecommunications, I will just limit myself to a quick presentation without elaborating or going into the details.

There are two families of analog modulations. One modifies the amplitude of a high frequency signal according to another signal containing the information. We then use the term **amplitude modulation**.

The other one does not modify the amplitude, but the phase or the frequency of the carrier signal according to the information signal. We then use the term **angle modulation**.

In general amplitude modulations are not suited to logic signals because they are not able to transmit an edge suitably (we have seen that the fundamental element in the transmission of digital signals is the data signal's edge). Furthermore, amplitude modulations are very sensitive to noise.

As for angle modulations, they are much more complex to implement but, on the other hand, they do offer a large number of advantages such as good immunity to noise, or the ability to transmit relatively steep edges. They are therefore much more widely used than amplitude modulations to transmit digital signals.

6.1.3.1. Amplitude modulations.

Amplitude modulations are essentially used in so-called telluric wave ranges (frequencies that propagate over very long distances due to a ground effect).

Amplitude modulations are based on the principle of the multiplication of 2 signals. On the one hand, an information signal called the modulating signal (f) and a "high frequency" signal called carrier (p). And we recover a modulated signal (s) at the output from the modulator.

Mathematical expressions:

In our example f and p are sinusoidal signals:

$$p(t) = A\cos(\omega_0 t)$$
$$f(t) = B\cos(\omega' t)$$
$$s(t) = AB\cos(\omega_0 t) * \cos(\omega' t)$$
$$s(t) = \frac{AB}{2}\left[\cos(\omega_0 + \omega')t + \cos(\omega_0 - \omega')t\right]$$

We can see that s consists of two sinusoidal signals. If we look at the spectrum of s,

given that:

$$\cos(\omega t) = \frac{e^{j\omega t} + e^{-j\omega t}}{2} \quad \text{et} \quad e^{j\omega t} \xrightarrow{TF} \delta(\omega)$$

We then find the spectrum of s :

$$s(t) = \frac{AB}{4}\left[\delta(\omega_0 + \omega') + \delta(\omega_0 - \omega') + \delta(-\omega_0 + \omega') + \delta(-\omega_0 - \omega')\right]$$

This translates into the following frequency representation:

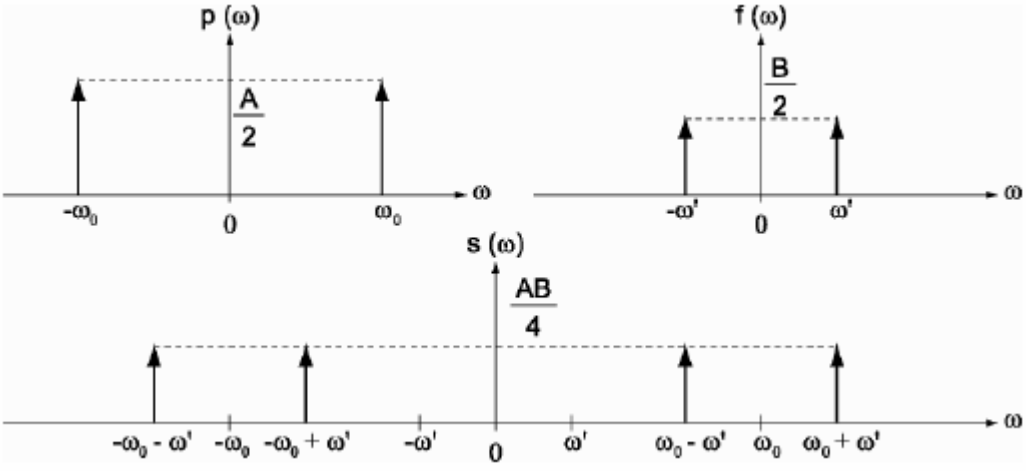


Figure 113: Frequency representation of amplitude modulations

Let us now consider that the information signal is no longer a sinusoidal but a polychromatic signal (which is made up of a spectrum with several lines).

We can then consider that the Fourrier transform of the signal f(t) is F(ω), we then find the expression of S(ω).

$$s(t) = f(t) * p(t)$$
$$p(t) = A\cos(\omega_0 t)$$
$$\Leftrightarrow s(t) = f(t) * \frac{A}{2}(e^{j\omega_0 t} + e^{-j\omega_0 t})$$
$$\Leftrightarrow s(t) = \frac{A}{2} f(t) * e^{j\omega_0 t} + \frac{A}{2} f(t) * e^{-j\omega_0 t}$$
$$s(t) \xrightarrow{TF} S(\omega)$$
$$\Leftrightarrow S(\omega) = \frac{A}{2}[F(\omega + \omega_0) + F(\omega - \omega_0)]$$

Which gives the following spectrum:

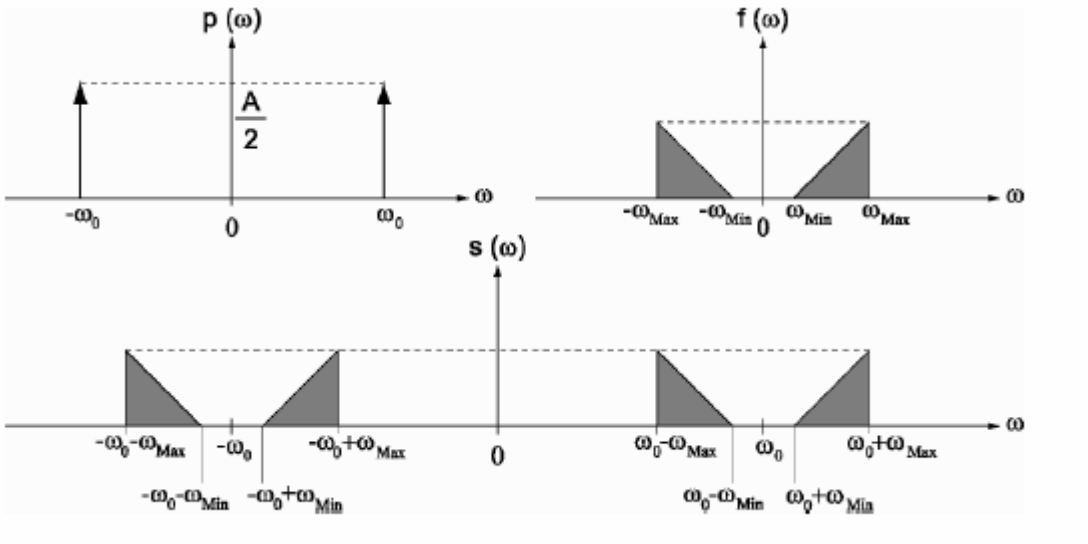


Figure 114: Amplitude modulation spectrum

6.1.3.2. The different amplitude modulations

There are three major types of amplitude modulation.

Amplitude modulation with carrier

Amplitude modulation with carrier makes it possible to transmit the information signal while keeping an image of the carrier.

The spectrum amplitude modulation with carrier is as follows:

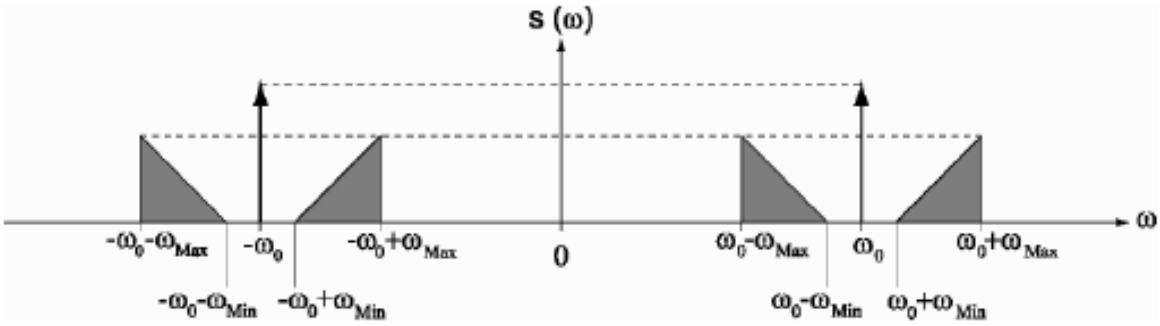


Figure 115: Spectrum of the amplitude modulation with carrier

You can make an amplitude modulation with carrier by inserting a direct component in the information signal.

We can therefore write:

$$f(t) = B[m + \cos(\omega t)] \text{ et } p(t) = A\cos(\omega_0 t)$$

$$s(t) = AB[m + \cos(\omega t)]\cos(\omega_0 t)$$

$$s(t) = AB\left\{m\cos(\omega_0 t) + \frac{1}{2}\left[\cos(\omega_0 + \omega)t + \cos(\omega_0 - \omega)t\right]\right\}$$

Where m is called the modulation index.

If we now consider the absolute value of m, we can describe three types of modulation:

- If $|m| = 0$, we have a modulation without carrier,
- If $|m| < 1$, we have an under-modulation,
- If $|m| > 1$, we have an over-modulation.

In the next chapter we are going to examine the case of modulations without a carrier.

When we make a modulation with carrier, we exclusively use modulation indexes m higher than 1.

Below 1, as the information signal passes below zero volts, we can no longer use an envelope detection for the demodulation of the signal and only a synchronous detection makes it possible to demodulate the signal.

But this type of demodulation is also used for modulations without a carrier.

In this case, why consume energy to transmit a carrier that you do not physically need?

The time representation of the signals is then variable, as a function of m. In the case of an over-modulation (index m higher than 1), the information signal is always higher than 0.

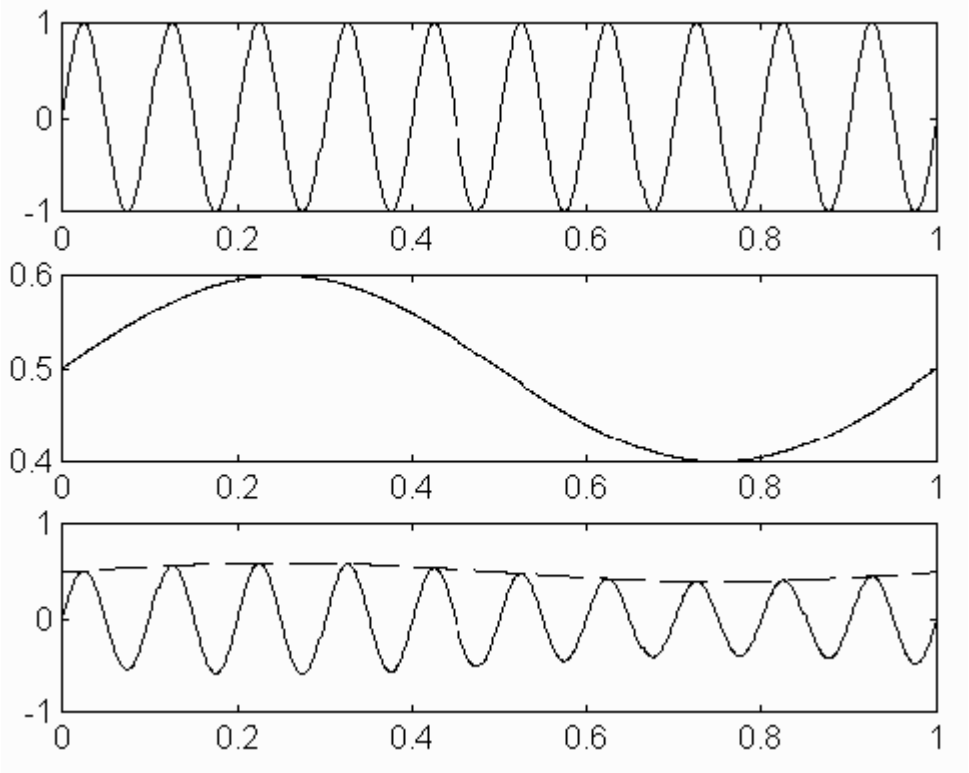


Figure 116 : Amplitude modulation with carrier (m>1)

But the main advantage of this type of modulation lies in the simplicity of the demodulation by envelope detection, and the main defect lies in their very low transmission efficiency (generally lower than 50%). As soon as a coherent demodulation is required, we will therefore prefer amplitude modulations without carriers.

Amplitude modulations without carrier

Modulations without carrier make it possible to save the power that we provide to the carrier and which makes the transmission efficiency fall enormously. In an amplitude modulation without carrier, we are confronted with the need to reconstitute a carrier in order to demodulate the signal. But we regularly have an inversion of the carrier's phase (whenever the information signal passes through zero volts).

Using a time representation, we can easily distinguish between modulations with and without a carrier.

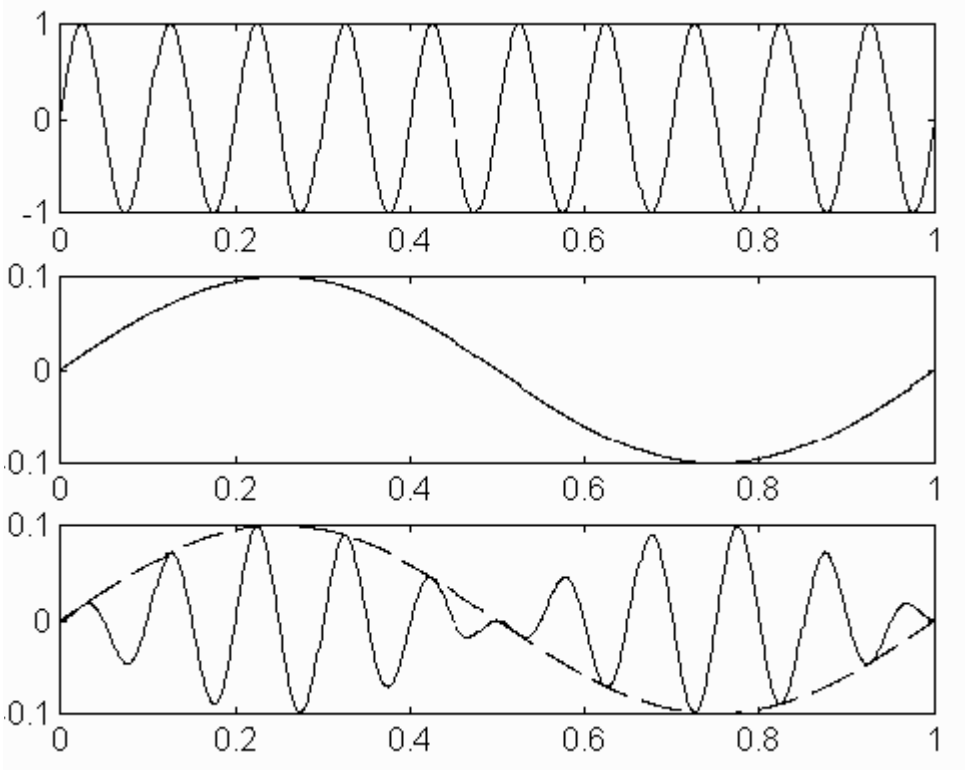


Figure 117 : Amplitude modulation without a carrier

The extra costs due to the demodulation of this signal, places this modulation in the same price range as angle modulations which, as we will see later on, are much more resistant to disturbances. We therefore tend not to use amplitude modulation without a carrier, and prefer angle modulations.

Narrow-band amplitude modulations

Narrow-band modulations represent a relatively recent transmission technique. The best known of these types of modulation is SSB modulation (Single Side Band). It has a modulation efficiency of 100% (all of the energy is used by the information signal).

Nevertheless, it is extremely complex to demodulate, which means it is impossible to use in local area networks.

The principle of SSB modulation consists of eliminating the carrier and, at the same time, half of the information signal's spectrum.

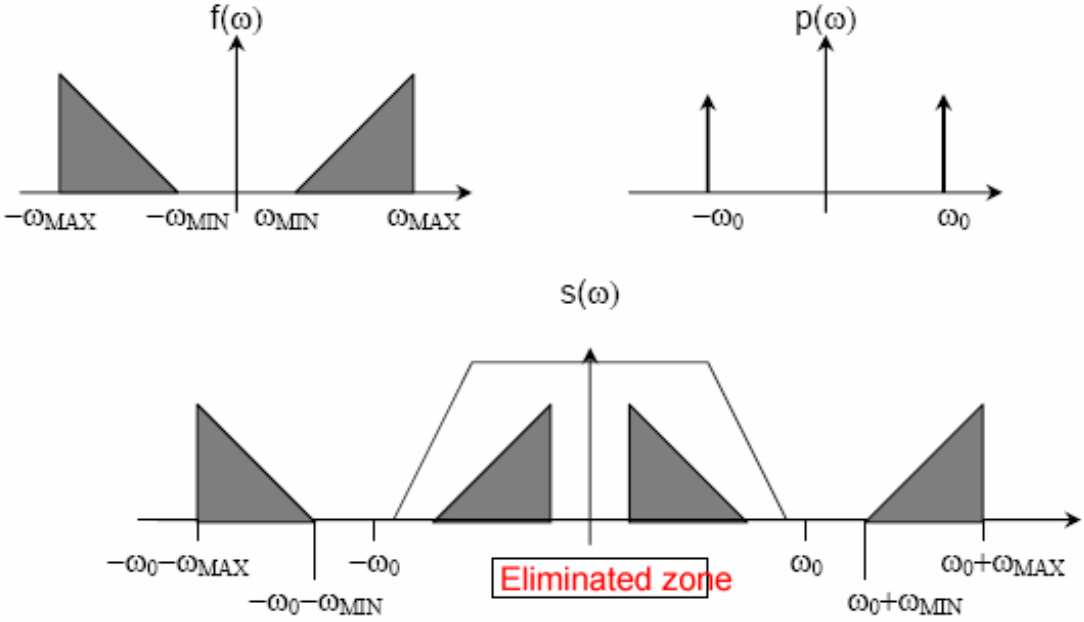


Figure 118: Narrow-band amplitude modulations

Remarks on amplitude modulations

Amplitude modulations are completely absent from the world of industrial local area networks, the edges of the digital signals not being particularly well suited to the utilisation of amplitude modulations. Their use is generally limited to low quality transmissions or to transmissions over very long distances. Furthermore, their over-sensitivity to external disturbances makes them dangerous. Indeed, they do not protect the transmitted signals, they may even make them a bit more sensitive, as the slightest noise can deform the modulated signal's envelope. So we generally prefer to use angle modulations when we want to transmit digital data. It should also be noted that in the area of industrial LANs it is extremely rare for modulations (angle or amplitude) to be used.

6.1.3.3. Angle modulations

The other method used for shifting the spectrum of the information signal consists of modifying the phase or the instantaneous frequency of a carrier signal as a function of the amplitude of the information signal.
Angle modulations are very commonly used for digital signals because they ensure a good transmission of the edges as well as a good immunity to noise (because the amplitude of the signal transmitted does not contain any information). Contrary to the previous paragraphs on amplitude modulations where we have studied the transmission of analog signals (therefore, by extension digital signals), we are going to discuss here a much narrower area and only examine the principle digital angle modulations. There are two types of angle modulation, one acts on the phase (**PSK**), and the other on the frequency (**FSK**).

FSK

FSK (FREQUENCE SHIFT KEYING) associates a carrier frequency with each logic level. The principle of FSK modulation consists of using two sources with frequency (f_1 and f_2) to generate a signal that is the frequency representation of the digital signal.

We define the modulation index of an FSK by: $X = \frac{|f_2 - f_1|}{R}$.

Where R is the maximum frequency of the data signal and f_1 and f_2 are the frequencies of the carrier respectively for a logic '0' and a logic '1'.

Generally, we try to produce modulation indexes that are quite close to 0.66 (optimum spectral occupation).

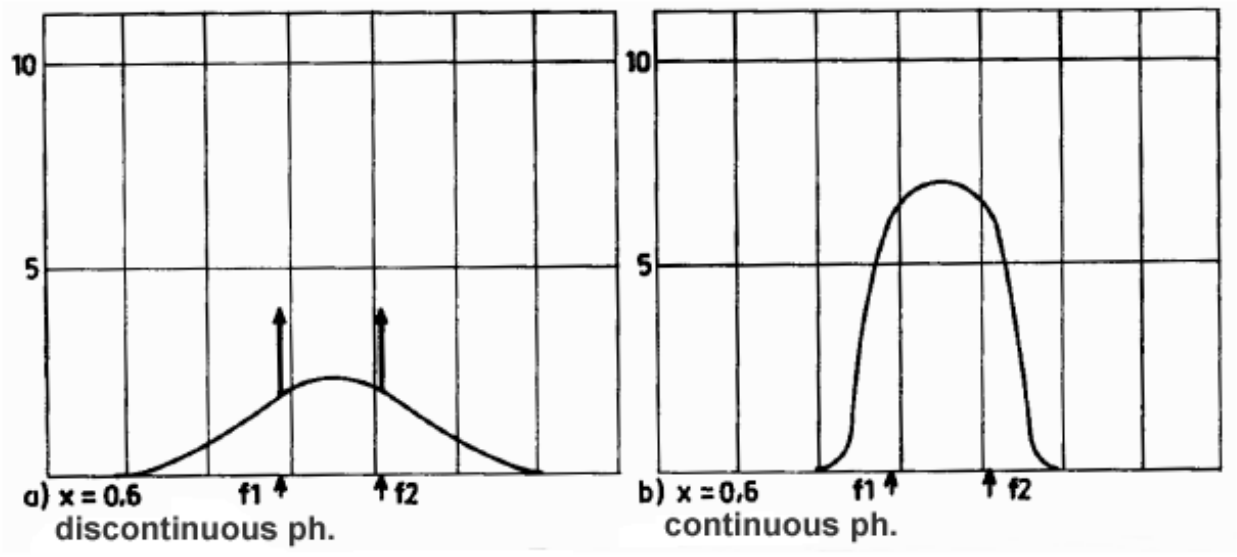


Figure 119 : FSK with a modulation index of 0.6

But often the transmission channel used is quite narrow which is equivalent to saying that f_1 and f_2 are imposed and often close to one another. We therefore play on the upper frequency of the information signal to obtain the desired index.

You must note that it is not advisable to reduce or increase the distance (when that is possible) between f_2 and f_1 for reasons linked to the capacity to discriminate between these two values (when they are too close) or to avoid too great a degree of spectral occupation (when they are too far apart).

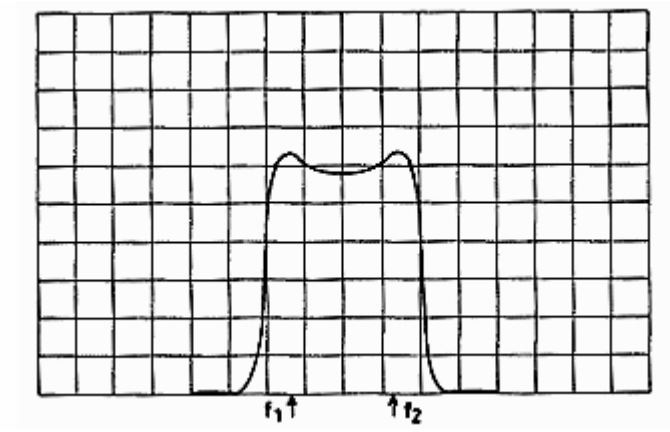


Figure 120 : FSK with a modulation index of 0.66 with direct phase

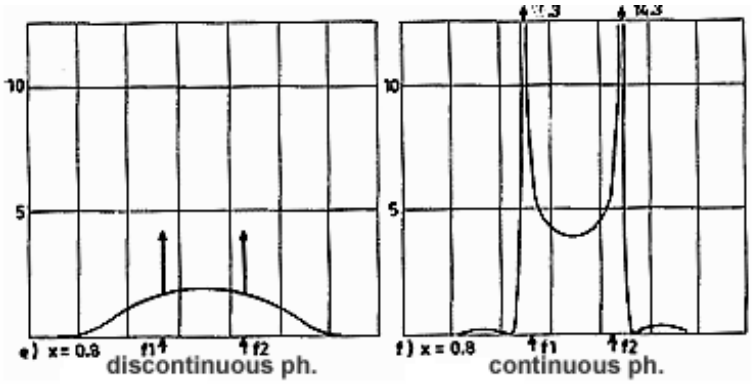


Figure 121 : FSK with a modulation index of 0.8

Once again we must distinguish here between two families of FSK, on the one hand continuous phase FSKs, that is to say without a phase jump when passing from f_1 to f_2 (and vice versa), or discontinuous phase FSKs where phase jumps may occur.

As shown in the previous examples, the continuous phase FSKs have a much better spectrum (reduced width) than their discontinuous phase equivalents. Of course, continuous phase FSKs are much more complex to make than discontinuous phase FSKs.

PSK

PSK (Phase Shift Keying) is another form of angle modulation. It acts by shifting phase. Once again, there are a large number of PSKs using different techniques to translate the digital code transmitted. Some standard PSKs code the logic '1' as a signal with a phase of 180° and the '0' as a signal with a null phase. These are absolute phase values.

Other PSKs (DPSK for Differential PSK) code the phase in relation to the previous value. So the logic '1' and logic '0' correspond to phase shifts respectively of +90° and -90° with respect to its previous signal phase.

There are also PSKs that code several bits at the same time. For example with two bits, we are going to establish four phase shifts as a function of the value of the two bits. For example "00" corresponds to a null phase shift, "01" to a phase shift of 90°, etc.

Caution: the greater the number of bits that the PSK codes at a time (and thus increase the flow on the link), the more difficult it will be to discriminate between two phases, and therefore the more complicated it will be to make the demodulator.

6.1.3.4. Reminders on analog modulations

Below we can see, after the clock, another form of modulation: **PWM**.

As it is not used in networks, I will not comment on this.

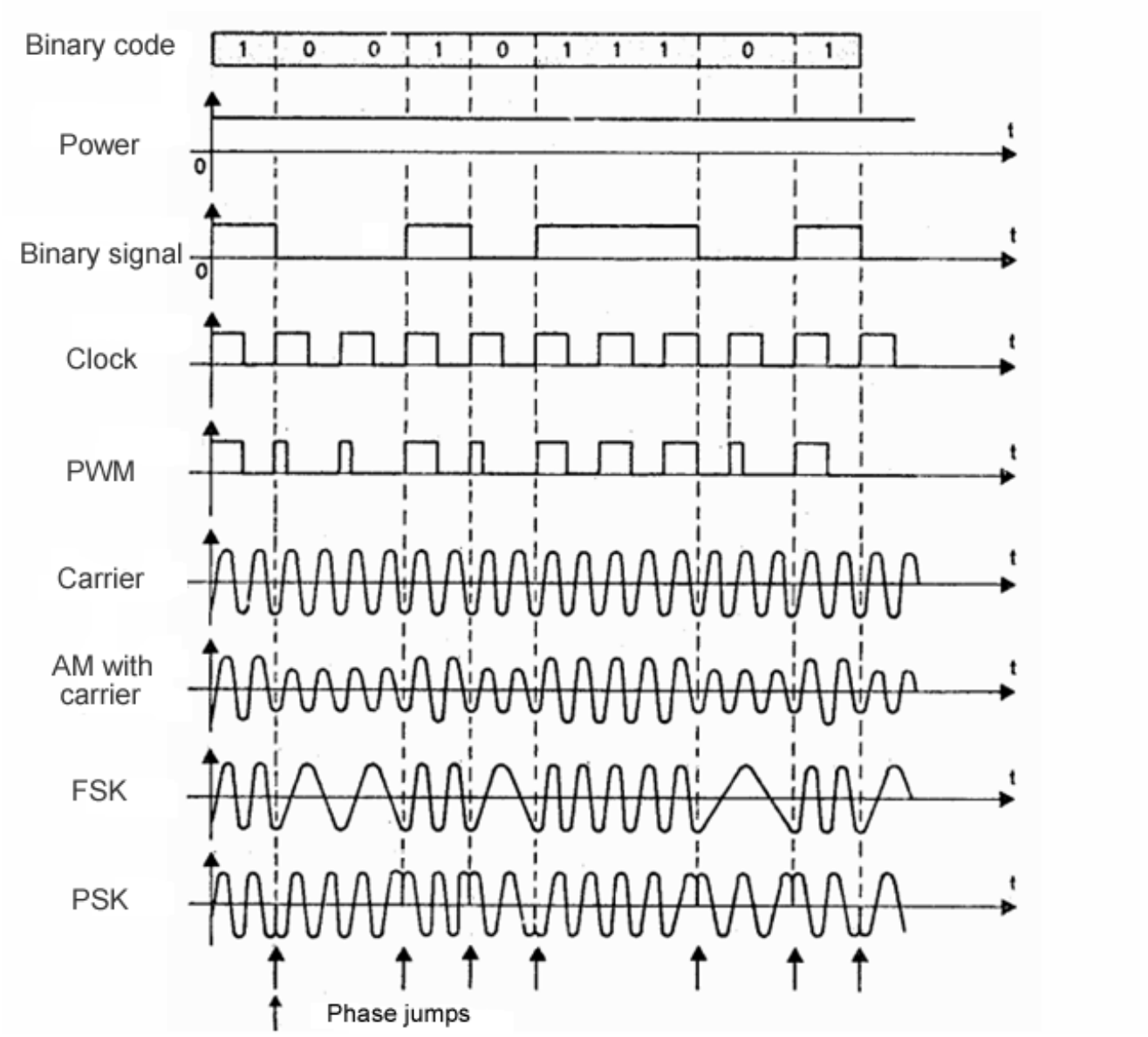


Figure 122 : Reminder on analog modulations

In this chapter, still in the framework of data transmission, we are going to look at how we prepare these data with a view to transmission where, thanks to appropriate coding, we can improve the quality, speed or amount of data transmitted.

6.1.4. Information coding

Information coding is used in transmissions, whether in baseband or in band shift, to prevent errors from being transmitted. As an error is always unacceptable, in order to avoid them we must provide, in addition to the data, a code that will in the worst case detect the errors and in the best case correct them

6.1.4.1. Parity coding

The simplest code for transferring information is coding of the parity. This method, the most basic one, was one of the first ones to be used. Its principle is quite simple, you have to add a supplementary bit to the data transmitted. In the case of an even parity, this bit allows the message always to be composed of an even number of bits at '1', and in the case of an odd parity, always of an odd number of bits at '1'. This coding is achieved by means of the EXCLUSIVE OR function for the even parity and EXCLUSIVE NOR for the odd parity.

At the time of decoding, we will use the same logic function as was used for the coding to validate the message. For example with even parity, we will perform an EXCLUSIVE OR function between the n bits of the message and the parity bit. If the result equals '0', the message is considered to be true, if the result equals '1' it is considered to be false. However, this result must be taken with caution, because a parity code can only detect an odd number of errors. It cannot detect two errors. We therefore consider that simple parity codes can only detect a single error.

We can also look at the number of control bits transmitted per message. In the case of a simple parity code, we generally consider that it is dangerous to exceed a proportion of 1/8, that is to say one control bit for eight data bits.

A development in the area of parity codes consists of introducing coding redundancy. Thus, we can envisage, still using a parity control, finding the incorrect bit. Let us consider sending a message of 5 words of 5 bits each, we can then code the message in the following way. On the transmission side, we place an even parity bit at the end of each line, and we do the same things with the columns. The parity bits created for each column are called transversal, and those created for each line are called longitudinal parity.

The signals transmitted are:
100111 110000 101000
010111 100010 001010

The signals received are:
100111 110000 101***1***00
010111 100010 001010

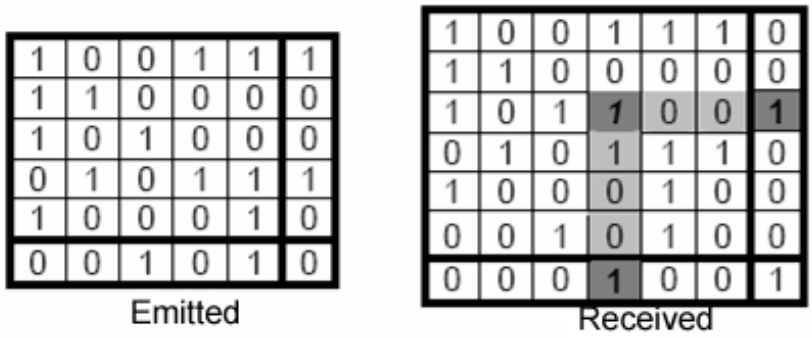


Figure 123: Parity coding

On reception, we recover the message that was transmitted and we apply a longitudinal and transversal parity control to it again. If an error has slipped into the code, Without fail, a longitudinal parity and a transversal parity are at '1'. And that means there is an error in the transmission.

In our example, by verifying the control values, we can see that there are two invalid control codes, one in one line, and the other in one column. So we know precisely where the error is located. This code therefore makes it possible to correct the error because it is placed at the intersection of the two lines and columns at fault. However, if on reception, there are two or three errors, there will always systematically be at least one transversal or longitudinal parity bit at '1'. We can therefore detect up to three errors, but we can only correct a single error.

The proportion of control bits with respect to the data bits, at the time of transmission, passes (in our example) from 1/5 (0.2) for a simple parity to 11/25 (0.44) for the composed parities.

6.1.4.2. Redundant coding

Redundant coding increases the number of bits in the message to make it possible to detect any errors and possibly correct them. The best known code is HAMMING code, and even though it is hardly used now, it was the code used in the first digital transmissions.

Example:

Let us imagine that we want to transmit one 4-bit word. The HAMMING code tells us that for n bits of information, we need k control bits to correct **one** error. We can express k as a function of n using the following formula:

$$2_k \geq n+k+1$$

We can solve this equation by iteration. For n = 4, we have k = 3.

That means that we are going to transmit 7 bits (m = k + n). We must now code the k bits with a view to transmission.

To do this, we are going to code all the message's error possibilities in a table. For a message of m bits, there are m + 1 error possibilities. This table is made up of k (= 3) columns. Hence the previous formula (m + 1 = n + k + 1 and 2_k represent the number of identifiable lines with k columns).

Error on the bit	e_3	e_2	e_1
None	0	0	0
1st bit (m_1)	0	0	1
2nd bit (m_2)	0	1	0
3rd bit (m_3)	0	1	1
4th bit (m_4)	1	0	0
5th bit (m_5)	1	0	1
6th bit (m_6)	1	1	0
7th bit (m_7)	1	1	1

We now take the equations of e_1, e_2 and e_3, and we find:

$e_1 = m_1 \oplus m_3 \oplus m_5 \oplus m_7$

$e_2 = m_2 \oplus m_3 \oplus m_6 \oplus m_7$

$e = m \oplus m \oplus m \oplus m$

The $\oplus$ sign means a modulo 2 addition which can be translated by $1 \oplus 1 = 0$. $\oplus$ is the Exclusive Or operator.

If we look carefully, we will see that the terms m_1, m_2 and m_4 only appear once in the equations. We can therefore use them as the Hamming coding bits (to simplify the equations).

The message transmitted then being made up as follows:

m_7	m_6	m_5	m4	m_3	m_2	m_1
n_4	n_3	n_2	k_3	n_1	k_2	k_1

To complete the creation of our coding we just have to fill out the table, that is to say find the values of k_1, k_2 and k_3 that make e_1, e_2 and e_3 null. Indeed, as the message has not yet been transmitted, it is not supposed to contain any errors.

To find the value of m_1, m_2 and m_4, you just have to write:

$k_1 = m_1 = e_1 \oplus m_3 \oplus m_5 \oplus m_7$

$k_2 = m_2 = e_2 \oplus m_3 \oplus m_6 \oplus m_7$

$k_3 = m_4 = e_3 \oplus m_5 \oplus m_6 \oplus m_7$

As e_1, e_2 and e_3 are null, we find:

$k_1 = m_3 \oplus m_5 \oplus m_7$

$k_2 = m_3 \oplus m_6 \oplus m_7$

$k_3 = m_5 \oplus m_6 \oplus m_7$

Which gives the following table:

m_7	m_6	m_5	m_4	m_3	m_2	m_1		
n_4	n_3	n_2	k_3	n_1	k_2	k_1	n =	k =
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	3
0	0	1	1	0	0	1	2	5
0	0	1	1	1	1	0	3	6
0	1	0	1	0	1	0	4	6
0	1	0	1	1	0	1	5	5
0	1	1	0	0	1	1	6	3
0	1	1	0	1	0	0	7	0
1	0	0	1	0	1	1	8	7
1	0	0	1	1	0	0	9	4
1	0	1	0	0	1	0	10	2
1	0	1	0	1	0	1	11	1
1	1	0	0	0	0	1	12	1
1	1	0	0	1	1	0	13	2
1	1	1	1	0	0	0	14	4
1	1	1	1	1	1	1	15	7

On reception, you just have to calculate the values of e_1, e_2 and e_3 to determine whether there is an error and where it is situated. We are then capable of transmitting a message while detecting and correcting any errors.

So, if the message to be sent is 1100, we have $k_3 = 0$, $k_2 = 0$ and $k_1 = 1$ the message transmitted is therefore 1100001. But if on the way an error appears on the 3rd bit, the message received becomes 1100101, and by using the Hamming decoder we find:

$e_1 = m_1 \oplus m_3 \oplus m_5 \oplus m_7 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$

$e_2 = m_2 \oplus m_3 \oplus m_6 \oplus m_7 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$

$e_3 = m_4 \oplus m_5 \oplus m_6 \oplus m_7 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$

We then obtain

e_3	e_2	e_1		decimal
0	1	1		3

We therefore know where the error is situated because e_1, e_2 and e_3 indicate an error on the 3rd bit. We can therefore correct the code received and say the error-free code is 1100001 and therefore that the message is 1100.

With this Hamming code, we therefore know how to correct an error on one bit, however we can also detect up to three errors, but without being able to correct them.

As for the proportion of control bits, we can see that the more n is great, the more k increases, but not in a linear way. For example, if to detect 3 errors, we need 3 control bits for 4 data bits, with 112 data bits, we just need to use 7 control bits.

Caution: do not forget that the probability of an error occurring increases linearly with the number of bits transmitted. For a transmission of 119 bits, the risk of error is therefore 17 times greater than for a transmission of 7 bits. So, for such an amount of data, it would be reasonable to use a more powerful Hamming code, that is to say one that is capable of detecting more errors.

6.1.4.3. CRC codes

CRC or CYCLICAL REDUNDANCY CHECK codes are based on a series of arithmetic operations.

Everything starts with the elaboration of the polynomial form of the binary message. For example, 110100, that is to say $1 . 2^5 + 1 . 2^4 + 0 . 2^3 + 1 . 2^2 + 0 . 2^1 + 0 . 2^0$ is written in its polynomial form $x^5+x^4+x^2$.

We name this polynomial P(x). Then we apply the following formula:

$$\begin{array}{r|l} P(x) \cdot x^v & G(x) \\ \hline & Q(x) \\ R(x) & \end{array}$$

Where G(x) is a polynomial defined by the network's protocol, known by the transmitter and by the receiver and of degree v. Q(x) is the quotient of the division of $P(x).x_v$ by G(x) and R(x) is therefore the remainder of the division.

Since G(x) is of degree v, R(x) is necessarily of a degree lower than or equal to v. It is this remainder R(x) that is going to be transmitted in addition to P(x) to the receiver.

On arrival, we therefore know R(x) (the remainder transmitted with the message), v (which is obtained by analysing G(x)), G(x) (known by definition) and P(x) (the message transmitted). We will then redo the same calculation as the one done by the transmitter and compare the remainders of the two divisions.

If there is a difference between the two remainders, we are sure that an error has slipped into the transmission.

G(x) is defined by the transmission standard used. Notification V41 of the CCITT (International Telegraph and Telephone Consultative Committee) standard defines G(x) as:

$$\mathbf{G(x) = x^{16} + x^{12} + x^5 + 1}$$

For the CAN network, the code G(x) equals $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$

And this is where we see how effective CRC code is because, in the case of the CAN bus, it makes it possible to detect the following errors:

- All the salvoes of errors including an odd number of terms
- All the salvoes of errors including at least 17 bits.
- 99.998% of the salvoes of errors of more than 16 bits.

This explains why CRC coding is very widely used for media that are not very reliable, and also in most networks.

Concerning the proportions of the number of data bits and of the number of control bits, we can see, for example, that the Ethernet network uses a 32-bit CRC code which allows it to ensure the safety of its data frame which can obtain up to 1526 bytes, giving a ratio of 4/1526 (0.0026).

More generally, CRC codes on 64 bits are used by data compression software (ZIP or RAR) to validate files that can be as big as several tens of Megabytes.

You must note, however, that the CRC codes that we have presented to you do not make it possible to correct the errors, likewise, they do not provide any communications confidentiality despite their great similarity with military or civil encryption codes. There are nevertheless CRC codes that correct the errors, but we are not going to examine them here.

6.1.5. Correction of transmission errors.

In the world of networks, the codes are only there to detect transmission errors, even if they are capable of correcting them. The transfer capacity of modern networks, and the sensitivity of the data, are such that that we prefer to retransmit a packet if it is likely to contain errors rather than make more errors due to dubious corrections.

6.1.6. Multiplexing

The goal of transmissions has always been to transmit more data in the smallest possible space. At present, with the multiplication of radio networks, and the boom in telephony, we are suffering from a severe lack of space. But the world of transmissions is governed by an inviolable law:

In order to transmit several signals on one channel, you must be capable of extracting them all.

This therefore means you must shift the signals to be transmitted either in time or in frequency. This technique, called multiplexing, works together on two axes: on the one hand the time axis and on the other the frequency axis. In both cases, this means dividing the authorised band into short intervals which will constitute as many communication channels as there are intervals.

6.1.6.1. Frequency-division multiplexing

Frequency-division multiplexing consists of dividing the frequency band you have at your disposal into intervals of the size of the communication channel.

This technique was very widely used in telecommunications at a time when the performance of digital systems was not as high as it is today.

The human voice, as seen from the telecoms perspective, is a signal occupying a band between 300 and 3400Hz (which is somewhat restrictive as, in general, it is considered that speech occupies a band between 20Hz and 20KHz). Thanks to this restriction on the frequency band, and to frequency-division multiplexing, we can place about 1,800 communications simultaneously on a single line.

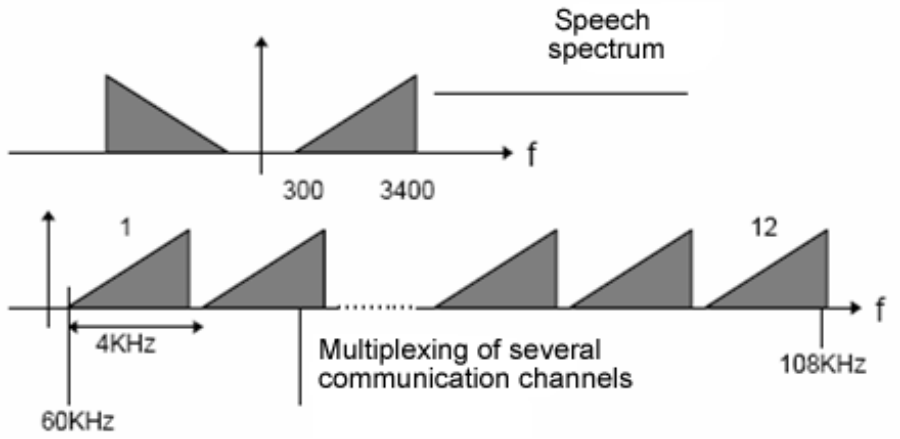


Figure 124: Example of 12-channel frequency-division multiplexing

Nevertheless, we cannot obtain 1,800-channel multiplexing directly, we have to proceed in steps.

Firstly, we go from the phone to the building's (or the street's, as the case may be) telephone junction box, and until then your line is personal, you are the only person to use it. On arrival at the first junction box, your line is multiplexed into 12 channels.

This 12-channel line goes to the central exchange where it will be multiplexed again with other channels of the same size to create 144 channels on one line. This line is then assembled with others of the same size to form a line with 1,728 channels.

It must be said, however, that as you are not the holder of a fixed frequency, and therefore that you use the first frequency available at the level of the building's junction box, in addition to the communication channels there are systematically added coding channels, which make it possible to send all the conversations to their respective destinations.

In general, multiplexing operations are represented as follows:

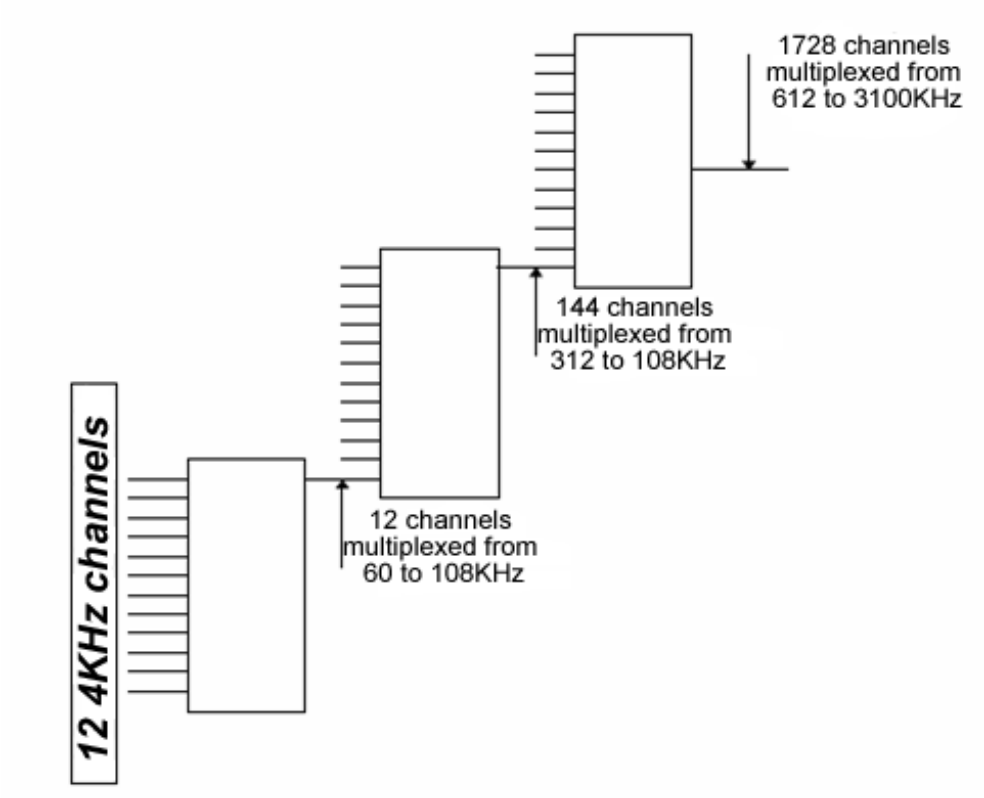


Figure 125: Representation of multiplexing operations

6.1.6.2. Time-division multiplexing

This multiplexing method was created with the development of digital technologies. Less costly than frequency-division multiplexing, which requires a vast number of very finely tuned carriers and SSB modulations (therefore quite complex to make), digital multiplexing only uses a single clock, coupled with high-frequency digital components.

It is therefore with the integration of digital components that high-speed digital telephone exchanges are made, thus opening up the way to time-division multiplexing (reminder: it was in 1996, that the whole FRANCE TELECOM communication system switched over to digital operation).

The principle of time-division multiplexing consists of dividing speech or data into small basic units (samples), which will be digitised to form digital data packets which are in turn transmitted over a short time interval called time slot. We then add to them the bits from another packet transmitted by another source. All these packets form a frame. At the time of the next frame, we place the next packet in the space that is made available to us.

We therefore have a discontinuous link with the person we are talking with. But, if these small bits of speech are transmitted at very high speed, this phenomenon becomes inaudible (this is the sampling principle).

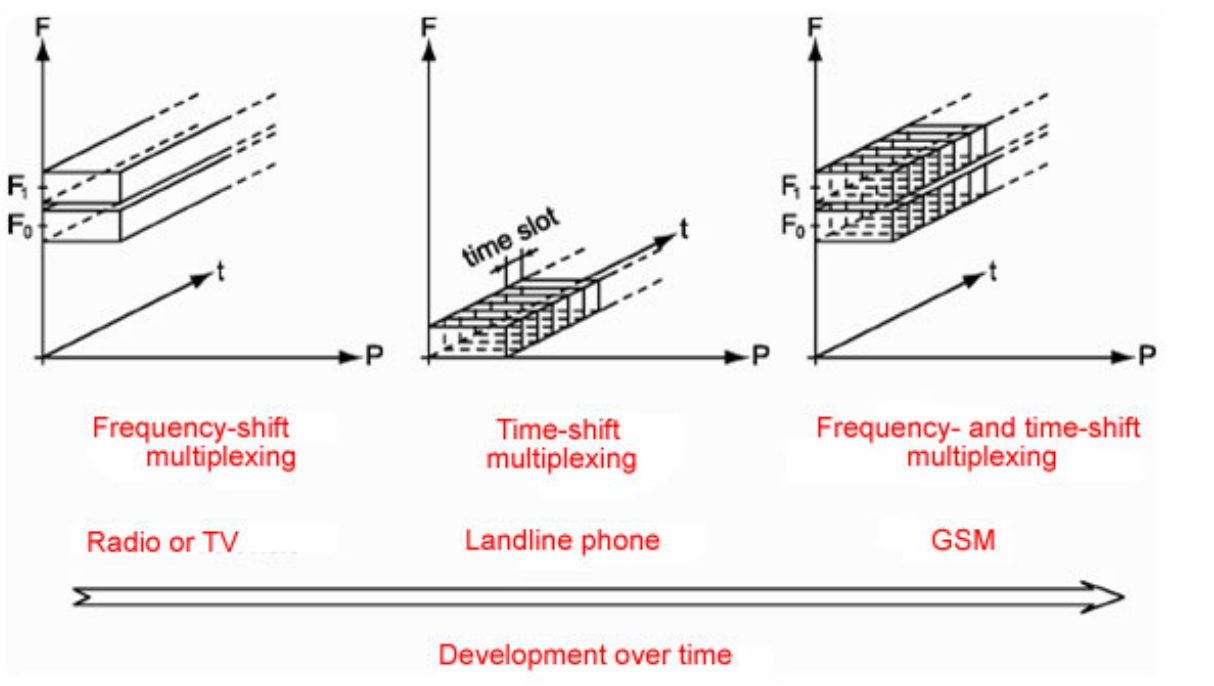


Figure 126: Time-division multiplexing

You must note that nowadays a third form of multiplexing is used, this is the technology used in UMTS systems, this is called power multiplexing. In the illustration above, one axis is not used (the P axis for power), this is the axis used for this type of multiplexing.

The concept of this completely digital modulation is that it mixes a very high-frequency pseudo-random code with the information to be transmitted, then on reception it uses correlation systems to extract the information.

Using this technique, we manage to extract, from amongst several other signals transmitted at the same time and at the same frequency, a signal simply through the fact of knowing its pseudo-random code.

This technique is also used (but in this case for protection purposes) in the GPS concept.

6.1.7. Transmissions vocabulary

This paragraph provides a brief glossary of terms that are commonly used in transmissions.

FULL DUPLEX: transmission method enabling two machines to talk to each other at the same time (not necessarily on the same line).

HALF DUPLEX : transmission method enabling two machines to talk to each other on the same line, but not at the same time.

SIMPLEX : transmission method whereby one and only one machine talks on one line, without any possibility of another machine talking.

MODEM : (Modulator-Demodulator), tool used to dialogue on a phone line and that conforms to the standard formats of the telephone communication standards.

UART : (Universal Asynchronous Receiver Transmitter) an electronic component making it possible to receive and transmit data on a series link by managing the parity control.

BAUD : transmission unit representing the number of events transmitted in one second. It differs from bits per second due to the fact that a measurement in bits per second can only be applied to binary signals. In the case of a transmission of a 2-state signal, one baud is equal to one bit per second.

INTERCONNECTION: there are several types of connections, enabling a certain diversity of wiring systems, making it simpler to adapt the network to the topology of the site. We will mention two here: "point-to-point" connections and distributed connections.

The "point-to-point" connection consists of establishing links between two machines only and multipoint (or distributed) connection consists of using a single line to connect all the machines.

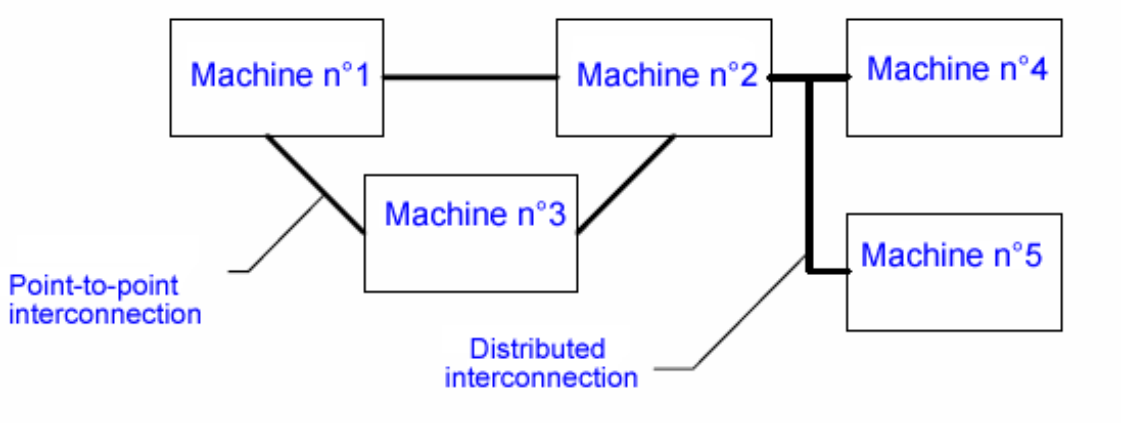


Figure 127: Interconnection

REAL TIME: this notion is an important aspect in local area networks. The notion of real time in fact means the ability of a system to respond within a given time. Let us take the example of an automatic drilling bench: we usually have a sensor making it possible to detect the breaking of the drill bit. In the case of a break, we must react relatively quickly, either by diverting the parts to other drilling units, or by stopping the line so that the defective part can be replaced. It is therefore important that the information should arrive on the central computer before a new part is presented for drilling. So, the order given in response to the broken drill bit must be processed in a limited time. This limit time serves as the basis for the notion of real time.

DETERMINISTIC NETWORK: a network is said to be deterministic when the access method is not left to chance (the order in which the machines talk is determined). This contrasts in general with the random access networks.

NODES: a machine's connection point with the network. This term expresses the number of elements connected to a network. Caution: we can place a repeater between two nodes, this will not be counted as it is not active with respect to the transfer of information.

WATCHDOG: device that is totally independent and generally embedded directly in the chip of electronic components, it is used to automatically restart the machine in the event of a failure of its software.

BIG ENDIAN : the most significant bit is transmitted first and the least significant bit at the end.

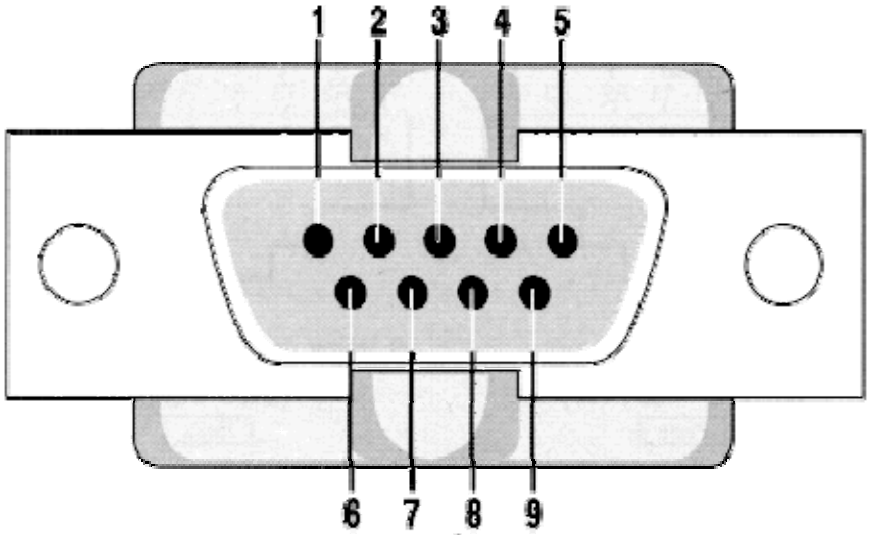
SMALL ENDIAN : reverse of BIG ENDIAN, this consists of transmitting the LSB first and then the MSB at the end.

6.2. THE FIRST COMPUTER NETWORKS

6.2.1. The RS232 standard

RS232 is a way of making two machines communicate with each other, but the term 'network' cannot be applied to it in any way. This is absolutely true but, as we must begin somewhere, let us begin with this system because, I am sure, you have heard of it (perhaps under the name of 'serial link').

The RS232 standard makes it possible to connect two and only two machines together. The information to be transmitted will pass between these two machines via a 3- or 9-wire serial link. The signals present on these wires, to which ground must be added, are as follows:



Pin	Signal		
1	DCD	Data Carrier Detect	Input
2	RD	Received Data	Input
3	TD	Transmitted Data	Output
4	DTR	Data Terminal Ready	Output
5		Signal Ground	
6	DSR	Data Set Ready	Input
7	RTS	Request To Send	Output
8	CTS	Clear To Send	Input
9	RI	Ring Indicator	Input

Figure 128: The RS232 standard

6.2.1.1. Presentation.

The RS 232 standard is a serial transmission protocol that enables full duplex, half duplex and even simplex transmissions. The signals transmitted are coded on ±12V with the logic "1" equivalent to a -12V level. Communication is governed by hardware parameters which must be identical on both the machines communicating with each other:

- speed: this can be chosen, according to the versions of the standard, between 75 bauds and several hundred kilo bauds (typical value: 9600 bauds),
- number of data bits: this can vary between 4 and 8 bits,
- number of stop bits: this can be 1, 1.5 or 2.
- type of parity control. This must be chosen from among several possibilities:

  - no parity control;

  - even parity (the message contains an even number of '1');

  - odd parity (the message contains an odd number of '1').

Furthermore, once the hardware parameters have been defined identically on the two machines, to allow them to communicate, you must choose a transmission mode, that is to say define the flow control process used for the serial link.

6.2.1.2. Flow control

The notion of flow control must be understood as a means implemented to ensure that, for two machines connected to each other using a serial link with a fixed transmission speed and different processing speeds on the machines, the speed is aligned on the slower processing capacity.

Even if nowadays the processing capacity of modern computers makes the information processing time seem ridiculous with respect to the duration of a bit, there are still cases where this control is essential.

There are three possibilities for flow control:

- software flow control.
- hardware flow control.
- no flow control.

Hardware flow control

Hardware flow control consists of using supplementary signals to "regulate" the exchanges, that is to say that the machines communicating with each other are going to transmit control data on supplementary channels.

There are several hardware flow control methods that use a greater or smaller number of signals.

The general case (as described by the RS232 standard) uses seven signals called **CTS** (Clear to Send), **RTS** (Request to Send), **DTR** (Data Set Ready), **TD** (Transmit Data), **RD** (Received Data), **DCD** (Data Carrier Detected) to which we add a ground.

In the wiring, we associate these signals in pairs: **DTR** with **DSR** (and **DCD**), **RTS** with **CTS** and **TD** with **RD**. We then need a 7-wire connection.

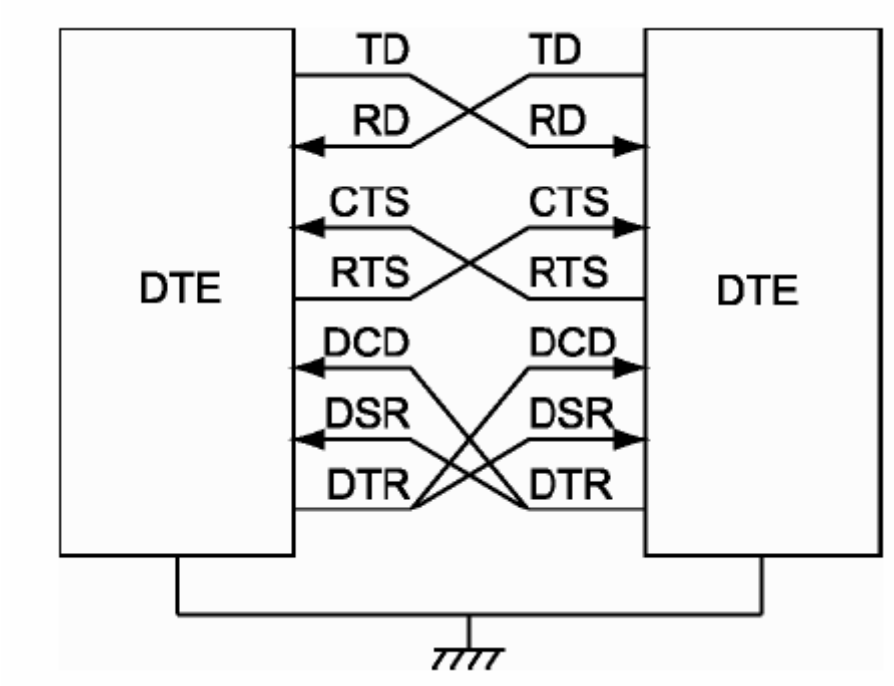


Figure 129 : 7-wire serial link

Amongst the other wiring possibilities, we have 5-wire (partial flow control) and 3-wire connections (no hardware flow control).

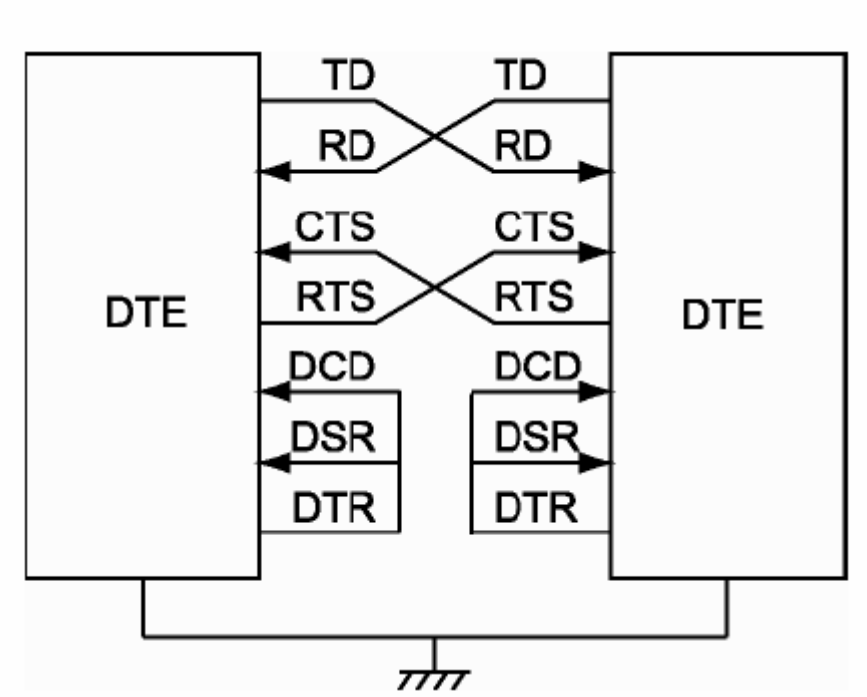


Figure 130 : 5-wire serial link

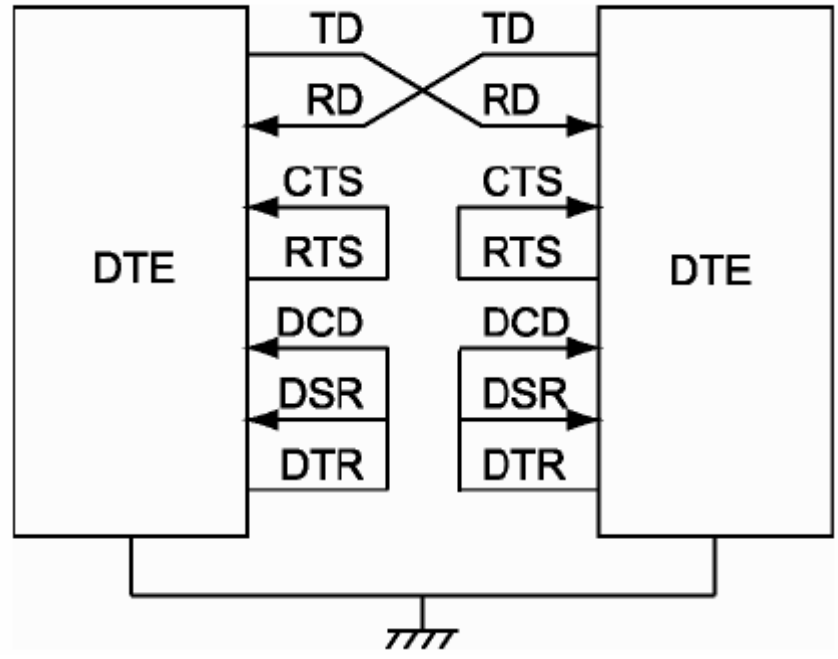


Figure 131 : 3-wire serial link

Software flow control

We can also use a software protocol to control the flow of exchanges. Here, the connection no longer needs the RTS, CTS, DTR, DSR and DCD signals, and we can therefore limit ourselves to a 3-wire connection.

It is at their software level that the machines are going to control their data exchanges. The protocol (because in this case it is a protocol) is called Xon/Xoff. It is limited to exchanges in **ASCII**. It is impossible to use this protocol to send binary information.

Furthermore, this protocol requires a **FULL DUPLEX** link.

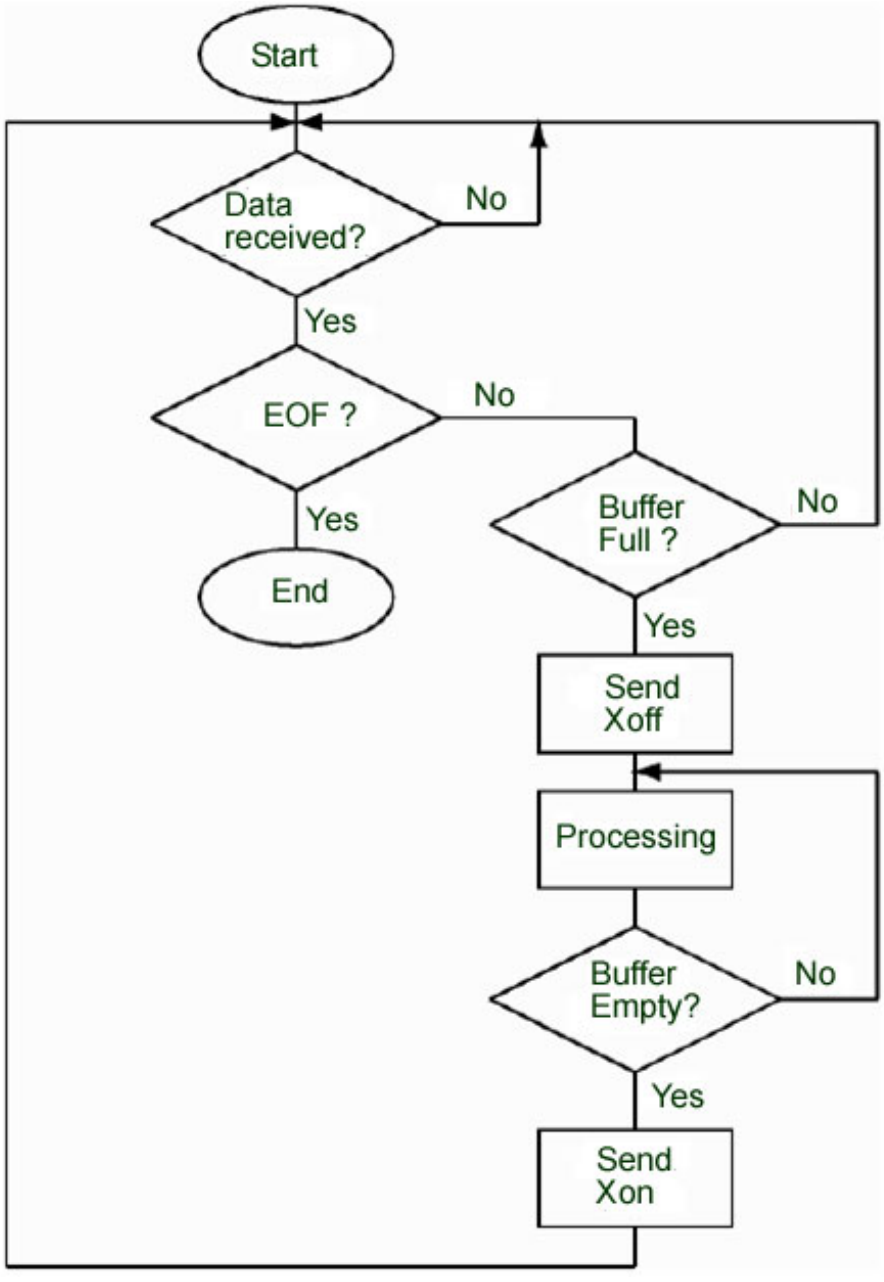


Figure 132 : Concept of software flow control for the receiver

The ASCII codes used to control the exchanges are Xon, which is ASCII code 17 (0x11) or CTRL Q and Xoff which is ASCII code 19 (0x13) or CTRL S.

The operating principle for this flow control is based on a simple automation concept.

From the receiver machine's viewpoint which, at the origin, is naturally ready to receive data, each byte received is stored in a reception buffer.

When this reception buffer reaches a certain occupation percentage (generally 80%), the receiver machine sends the Xoff character to the transmitter. It will then process the data it has received so it can empty the reception buffer.

Once a second threshold is reached (generally 50%), it sends the transmitter the Xon character so that the transfer of data can resume.

On the transmitter side, reception of the Xoff blocks transmission until an Xon is received.

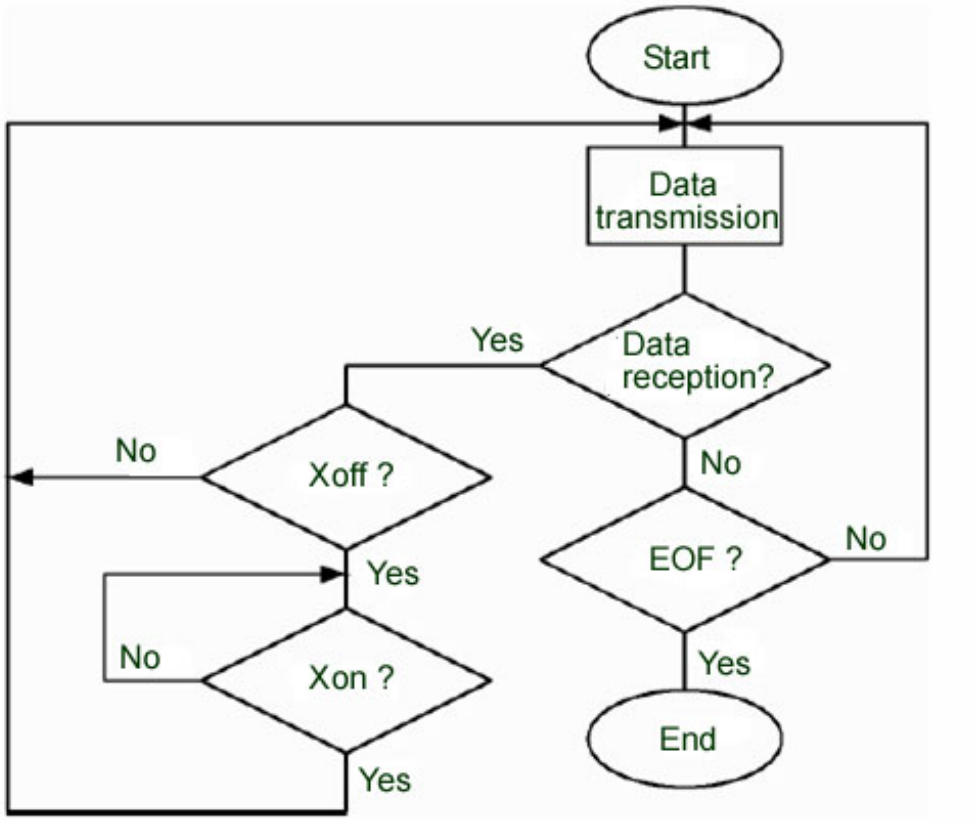


Figure 133 : Concept of software flow control for the transmitter

Application

So a transmission of 7 data bits with an even parity control and 2 stop bits at 9600 bauds represents a transmission of 11 bits in all, 7 of which are useful. That is (9600*7) / 11 = 6110 useful bits /s.

Once these points have been defined, we can transmit the data using a UART. The frames then take the following form:

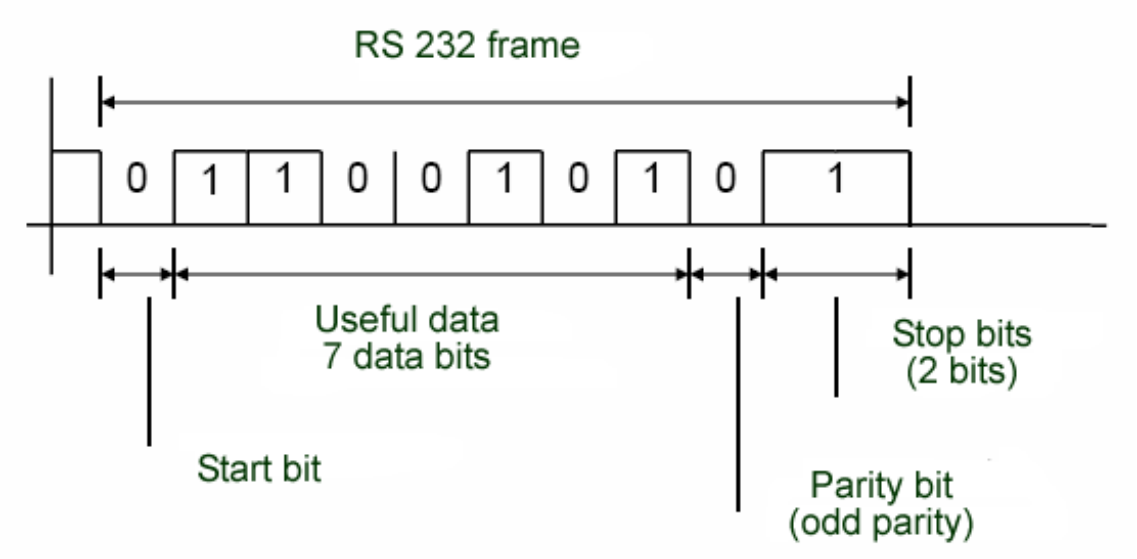


Figure 134: Application of software flow control

The data is received by a UART's internal clock, the multiple of the transmission frequency (internal clock = N* transmission clock) to allow it to synchronise with the transmitted signal.

A signal falling edge is used to define the beginning of the start bit (which equals ‘0’ whereas the unoccupied line or the stop is at ‘1’), we then have counting of the internal clocks edges. As we have defined the transmission speed, the receiver knows how many internal clock edges (N) represent one transmitted bit, at N/2, the UART knows the data is present on the line. We do the same thing with the other bits in the transmission at 3N/2, 5N/2, etc.

We then reconstitute the message without it being necessary to transmit the clock. However, this is only valid for short messages, otherwise the desynchronisation of the clocks could pose problems that could lead to a loss of data. To avoid this type of malfunction, you just have to resynchronise the reception clock on each rising edge and/or on each falling edge of the data signal.

The RS232 standard is now reserved for basic uses. Certain developments (RS422, RS423 and RS485) of the standard enable much higher speeds between many machines. However, they are not installed "as standard" on modern computers, which is a severe handicap for their development.

6.2.2. The IEEE 488 bus

The IEEE 488 bus was the first instrumentation bus to be standardised. Born from the HPIB bus (HEWLETT PACKARD INSTRUMENTATION BUS), this bus is exclusively dedicated to the automation of measuring chains.

It is used to transfer data, bi-directionally, on an 8-bit bus by means of an asynchronous HALF DUPLEX dialogue protocol. The machines are connected to each other by means of a 16-wire cable that is connected to the distributed link on the IEEE connector.

The bus is obligatorily managed by a controller (often the data processing unit). On this bus, all the machines (including the controller) can alternately be the TALKER (transmitter) or LISTENER (receiver). There must not be more than one transmitter at a time, but there can be several receivers.

You can connect up to 15 machines (including the controller) and achieve speeds as high as 1Mbyte a second over a total line length not exceeding 15m (that is 1m between each machine). No segment between two machines may be more than 2m long.

There are three types of signal on the lines (the numbers in brackets represent the pin number on the IEC connector):

- The data signals from D0 (1) to D4 (4) and from D5 (14) to D8 (17),
- The control signals REN (5), IFC (10), ATN (12), SRQ (11) and EOI (6),
- The flow control signals DAV (7), NRFD (8), NDAC (9),

All the other wires are connected to ground.

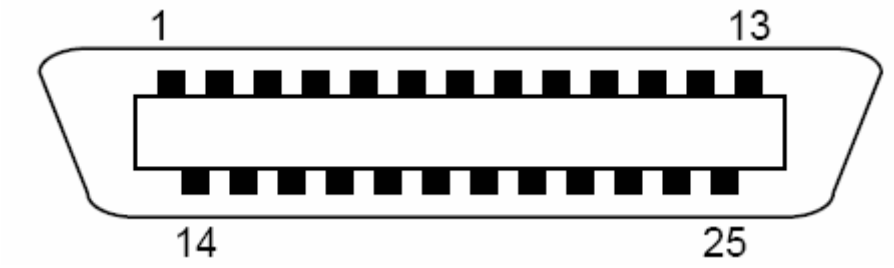


Figure 135 : IEEE 488 connector (IEC model)

6.2.2.1. Machine control signals

The REN signal (Remote ENable)

REN enables the bus to take control of a machine. In the case where the REN signal is at "0", the instruments (measuring or display) are accessible via their front faces. If REN is at 1, the instruments are controlled by the bus and their front face is deactivated.

You can then command them by sending data. The format of the messages is then an ASCII code, such as defined by the manufacturer of the machine.

The IFC signal (InterFace Clear)

IFC allows the bus controller to initialise all the machines connected to it. When IFC is at 1, all the machines stop. The line is freed and so the data control signals remain free and allow the controller to initialise the machines.

The SRQ signal (Service ReQuest)

This signal (optional) is used by the machines (certain machines only) to notify the controller of the need to give new orders. This line is shared by all the machines so the controller must, when SRQ passes to 1, decode the address of the requesting machine.

The EOI signal (End Or Identify).

The EOI signal allows the controller to determine which machine is asking for assistance by means of SRQ (identification).

These signals are transmitted with a ground to improve immunity to noise. The grounds are placed on pins 13 and from 18 to 25.

The ATN signal (ATtentioN).

ATN allows the bus controller to talk. When ATN is at 1, the controller also becomes the transmitter of operating orders. That is to say it no longer supplies data but gives commands. When ATN is at 0, the controller is a machine like the others, transmitter or receiver according to the programming and the signals on the bus are then data.

6.2.2.2. HAND-SHAKE (flow control):

Use of an asynchronous protocol imposes a set of command signals.

The following signals are used:

- ***The DAV signal (DAta Valid)***

DAV allows the transmitter to notify to the receiver(s) that the data presented on the bus are valid and can be acquired.

- ***The NRFD signal (Not Ready For Data)***

NRFD allows a receiver to notify to the transmitter that it cannot acquire an item of data that it could present on the bus.

- ***The NDAC signal (No Data ACcepted)***

NDAC allows a receiver to notify to the transmitter that it has not yet acquired the data presented on the bus.

Transmission timing chart

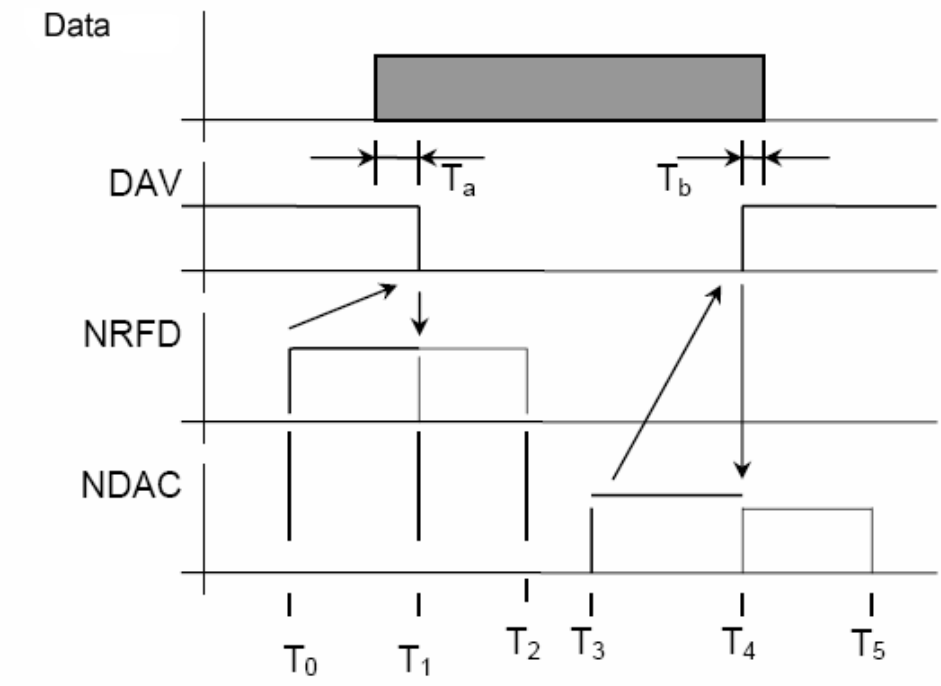


Figure 136: Transmission timing chart

- Before T_0, the receiver indicates that it is not able to receive data.
- At T_0, the receiver indicates that it is ready to receive.
- At T1, the data have been present since time T_a on the line. The transmitter then indicates to the receiver(s) that the data are valid.
- From T_2 to T_3, the receiver can acquire data.
- At T_3, the data have been acquired and the receiver notifies this to the transmitter.
- At T_4, the transmitter announces that the data are no longer valid (even though they remain valid for time T_b).
- At T_5, the cycle can start over again.

6.3. NETWORK ORGANISATION

As you can see, networks are complex things, requiring a large amount of technology. You must remember that no two networks are identical, there are therefore an impressive quantity of them, each with its preferred application area, hence the definition of several levels of specification in the form of a pyramid named **CNIM**.

This pretty representation is in the process of disintegrating due to the ever-wider use of the Internet. A large number of companies having drastically reduced their network offering in this respect. Increasingly, the common solution has become Ethernet + TCP/IP, the pyramid then being limited to two layers. An upper layer (Ethernet +TCP/IP) then underneath that, networks of sensors.

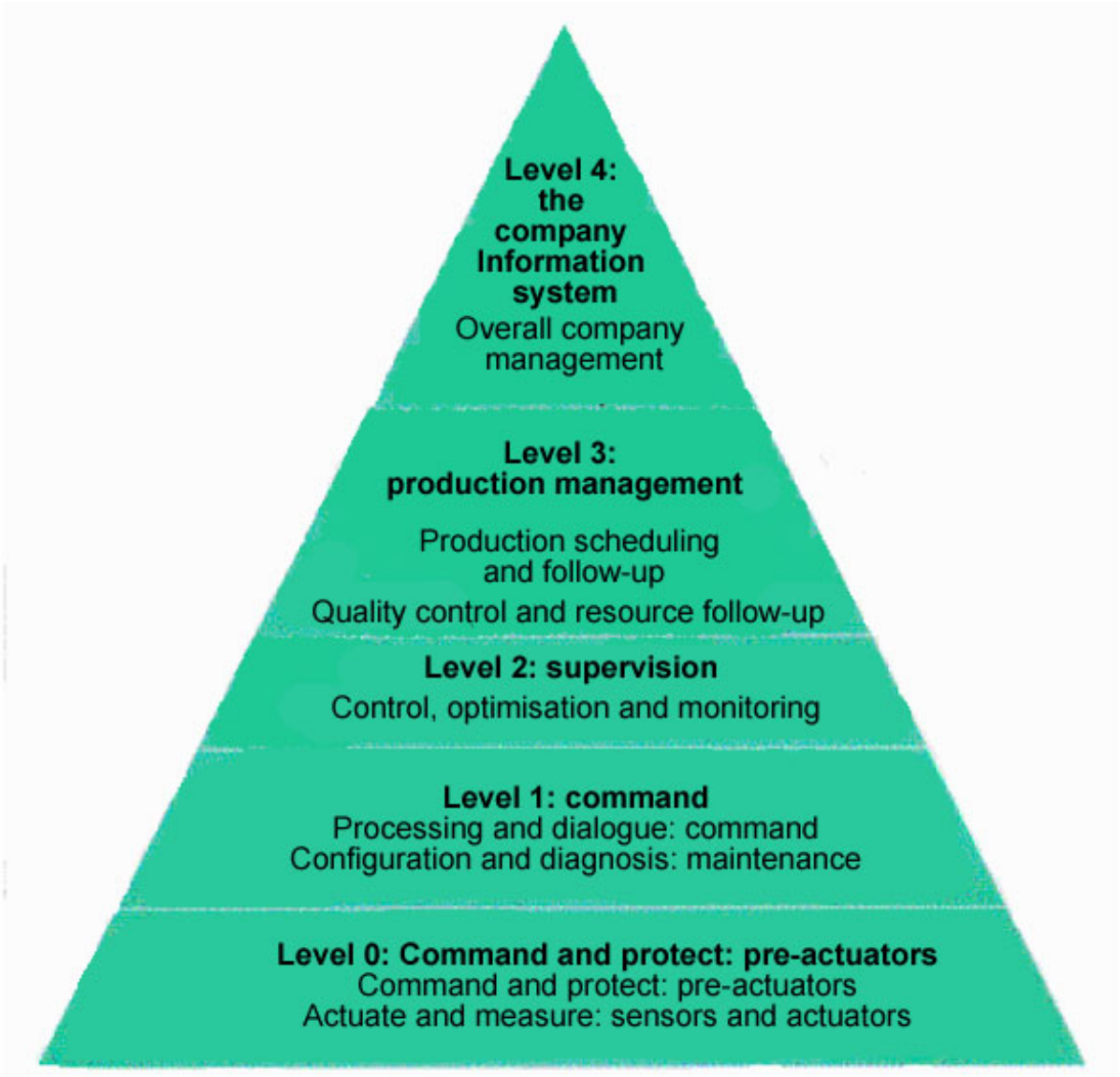


Figure 137 :CNIM pyramid

However, in a collective effort to achieve standardisation, the OSI standard (OPEN SYSTEM INTERCONNECTION) has been defined. This standard, based on a functional breakdown of the elements in a network, defines 7 layers each characterising a programming or wiring network.

6.3.1. The OSI standard

Launched in 1977 and accepted in 1978, the ISO standard defines 7 levels of specification presented each as a layer superposed over the previous layer.

6.3.1.1. The 7 layers of the OSI standard

Generally, we use epistolary analogies to present this notion of layers, so let us go along with the tradition and imagine that we want to send a job application letter. Nearly unknown to it, we define a set of parameters around the information to be sent, making it possible to send the data from us to a manager.

Let us proceed in steps.

- We write the letter (application),
- We include a header so that the destination can recognise us (presentation),
- Once we have written the letter, we put it in an envelope (physical).
- Then we write the destination's address on the envelope (transport).
- We stick a stamp on the envelope (session).
- We place the envelope in a letter box (network),
- It is sent in a transparent way to the destination (link).

There in 7 lines I have defined the principle of all networks, we start from an application (what we want to transmit), we add a presentation, etc. and in the end the message is sent.

You can understand therefore that we have had to add a set of elements around the text, which do not represent a great deal of information for us, but which will allow the letter to reach the destination.

We have therefore added successive layers of information, hence the idea of a definition of these additions that takes the **form of layers**.

But the notion of layer masks a double meaning:

- A layer provides a stable base, it enables good communications between the lower level and the higher level (a bit like bricks in a wall, where you must be able to rest on the bricks that have already been laid). We then have as dialogue rule the possibility of communicating with the adjacent levels (vertical dialogue). We call this function **the service**.

- The other notion is linked to the utilisation of the layer, that is to say the coherence with the layer on the same level on the machine with which you are communicating. To go back to our brick analogies, it is the coherence between two bricks side by side in the same row. If one is bigger than the other, you may well get a surprise. We then have as dialogue rule the possibility of communicating with a layer on the same level (horizontal dialogue). We call this function **the protocol.**

The Physical layer (layer 1)

The physical layer represents the lowest level of specification, it defines a network's electrical and mechanical specifications. We thus define the type of connection (full duplex, half duplex or simplex), the type of link (serial or parallel), the medium (radio, coaxial cable, twisted pair link, etc.).

The link layer (layer 2)

The link layer makes it possible to manage access to the line and the transfer of information between two adjacent machines. It manages the connection and disconnection processes, it detects the errors, manages addressing and defines the frame formalism to adapt it to the physical support. Here there are two very important sub-layers: **the MAC sub-layer** (Medium Access Control) which is used to manage access to the network, manage or avoid conflicts; and the **LLC sub-layer** (Logic Link Control) which deals with the service with the higher layers.

The network layer (layer 3)

The network layer makes it possible to route the information from one network to another, or even through a set of networks.

The transport layer (layer 4)

The transport layer is used to control the flow of data in a network. It manages the control of transmission errors and the reliability of the link.

The session layer (layer 5)

The function of the session layer is to connect up the services available in the two machines, by making the network's lower layers transparent.

The presentation layer (layer 6)

The presentation layer manages the presentation of the data in a syntactic way (grammatical). Among other things, it enables the coding and decoding of the information (confidentiality or compression). Its role is in fact to make heterogeneous machines compatible (for example, dialogue between a Mac and a PC).

The application layer (layer 7)

The application layer is the network's highest layer. It encompasses all the applications that are going to be used by the network. In general, this is the layer that will serve as the user's interface. We can distinguish between two types of applications: applications in connected mode (where the connection must be maintained) and applications in unconnected mode (where the connection is intermittent, such as INTERNET for example or e-mail).

We use layers 1, 2 and 3 often and we call them the **lower layers** (using industrial IT) in opposition to the others, which we call the **upper layers**.

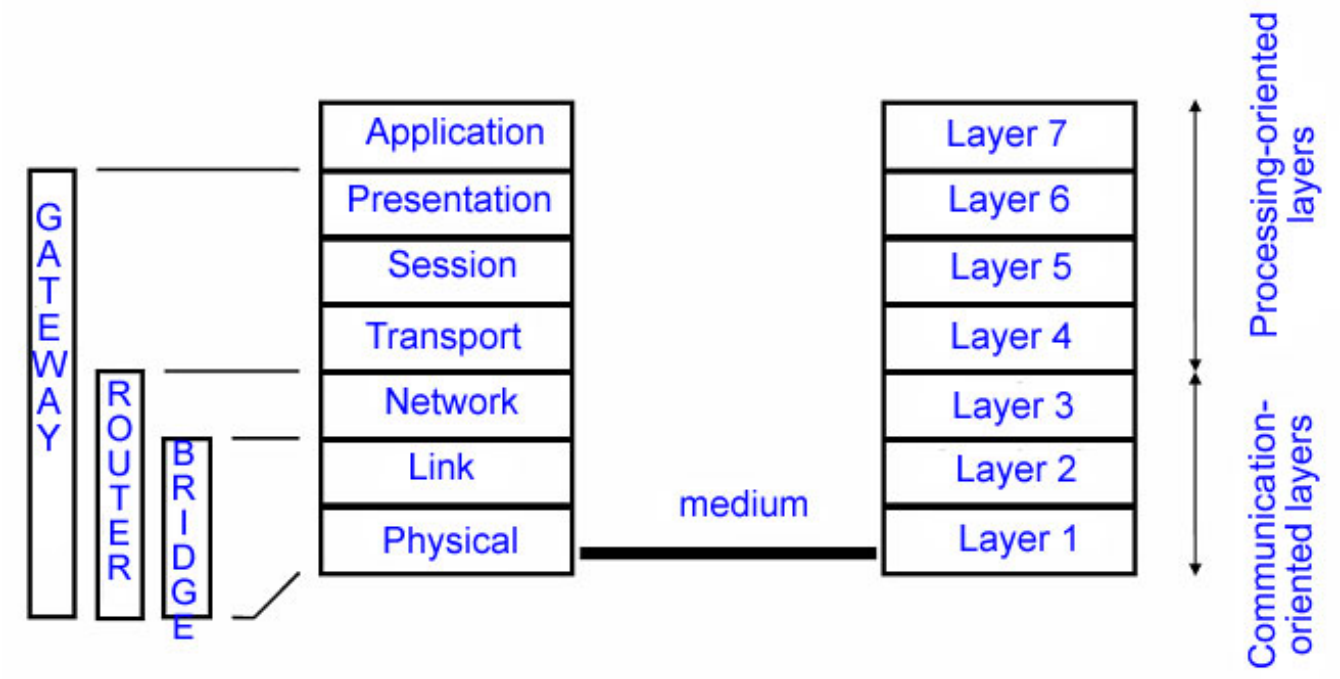


Figure 138 : Representation of a network organisation (OSI)

The definition of a network's OSI layers in the overall sense, leads us to examine the influence of these layers in our applications. For example, the use of the IEEE 488 bus means that layers 1, 2 and 7 are used. And that means that layers 3, 4, 5 and 6 do not exist.

This point is extremely important because it permits us to say that if the OSI "model" allows us to define all the networks, in no way are the networks forced to use all of the 7 layers in the OSI model.

6.3.1.2. Data encapsulation

The phenomenon of layers is not simply theoretical, it has a representation on the physical level, we call this phenomenon **encapsulation**.

Encapsulation is the addition, layer by layer, of data in addition to the data provided by the higher level to authorise the correct decoding of the information.

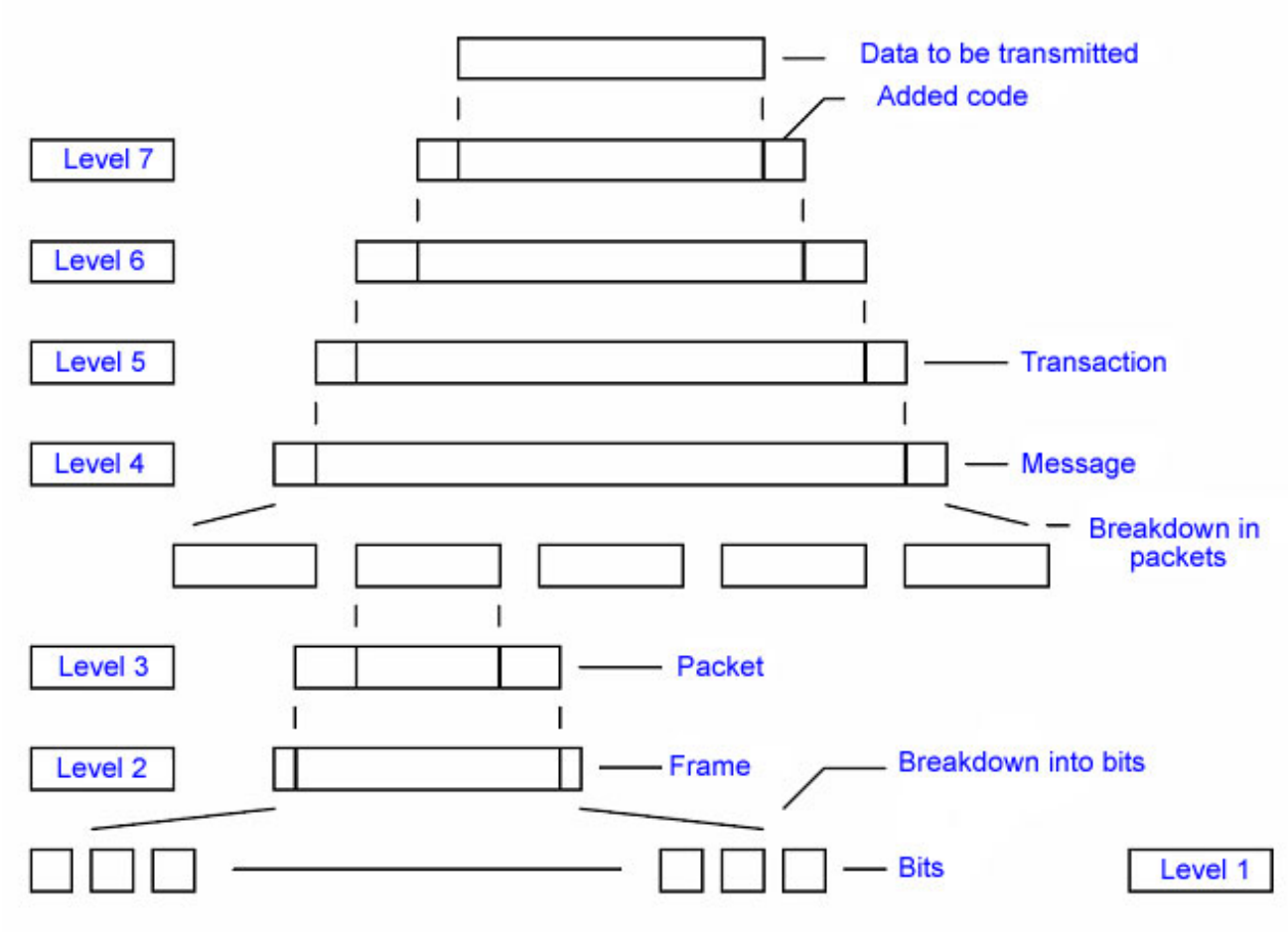


Figure 139 : Representation of data encapsulation

6.3.2. Frames and packets

The terms **frames** and **packets** represent very important elements in transmissions. Let us go back to the example of the letter, we saw that we added supplementary parameters to the text to be transmitted, such as the destination's address, the sender's address, etc.

In the electronic reality of networks, we find more or less the same thing, if the text represents n bits, supplementary terms will have to be added so that the transmission can be made.

6.3.2.1. Principle of the basic frame

A frame represents the smallest unit that can be understood by a network, that is to say the least number of bits required for the message to be transmitted. So, let us analyse how frames are made in networks.

If we take our letter example again, we can see that our message can be summarised as follows:

Destination's address	Sender's address	Text of the message

Unfortunately, this basic frame is too simple for modern networks. Indeed, if the message does not have a constant length, there is a serious risk that we will not know where it ends and where the next one begins.

So we have to add another field to define the size of the text. But that might not be sufficient, we must also be sure that each machine reading the message understands that if it finds its address in the text of the message, that this is only a coincidence.

So, in general, we add a code at the beginning of the frame making it possible to identify the beginning, we call this code **the marker**. We also add a field for the error control.

Which gives:

Start marker	Destination's address	Sender's address	Number of bytes	Text of the message	Control code

6.3.2.2. Principle of the packet

We have seen in the previous chapter that we add elements to the text to be transmitted. But often, the frames have a limited format, to allow the others to talk (upper limit) or for technological reasons that we will examine later on (lower limit).

If in the case of the lower limit, we generally resolve the problem by adding meaningless characters, in the case of the upper limit we must often break the text down into bits with a restricted format, thus enabling a transfer that complies with the standards; this breakdown constitutes the creation of packets and the message extracts are then called packets (for certain networks, the term **datagrams** is used).

The OSI pyramid therefore has a defect: it seriously increases the size of the information to be transmitted.

However, it does have the advantage of delivering the data, without errors and in an understandable way, which is the least you can expect.

6.4. THE PHYSICAL LAYER

The physical layer makes it possible to define all of the network's physical characteristics (hence its name). In fact it makes it possible to make the match with the medium used for the transmission by fixing its utilisation limits (speed, length, etc.).

6.4.1. Network topology

A network's topology is the way the machines must be connected. There are three basic methods for establishing communications.

Token ring

The machines are placed in a ring to form a closed loop, made with point-to-point links.

With this type of structure, as each machine constitutes a link in the transmission, the failure of one machine may cause the failure of the network. To avoid this, there is a device that short-circuits the connection to the machine which also makes it possible to install a new machine or remove a machine from the ring without leading to the complete shutdown of the others.

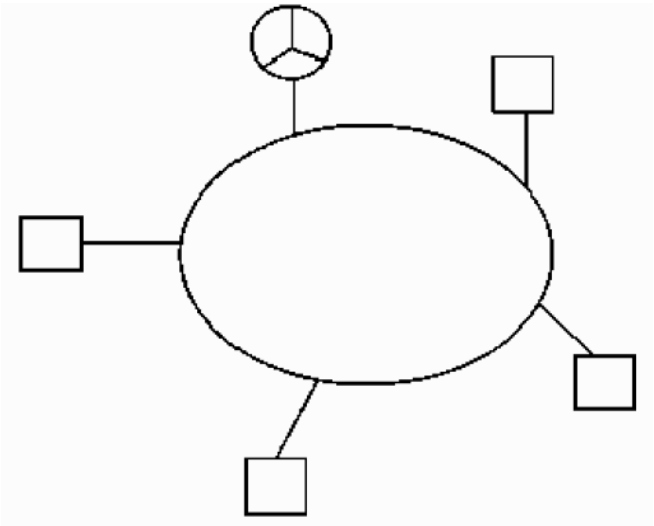


Figure 140 : Token ring network

Star

The machines are distributed in a star around another machine called the concentrator, connected to the latter by a point-to-point link. This topology makes it possible to create small independent islands.

The installation or removal of a machine (with the exception of the router) is not subject to any special conditions. Only the failure of the concentrator can cause the network to fail, with the failure sometimes extending beyond the island.

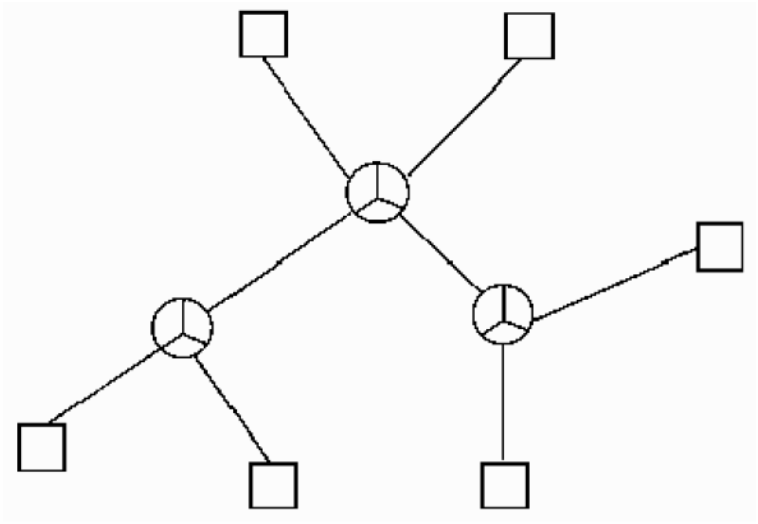


Figure 141 : Star network

Bus

The machines are distributed all along the line, like memories on a microprocessor bus, connected to the router by a multipoint link. Like with the star, the installation or removal of a machine (with the exception of the router) is not subject to any special conditions, however, just one machine can cause the failure of the whole network.

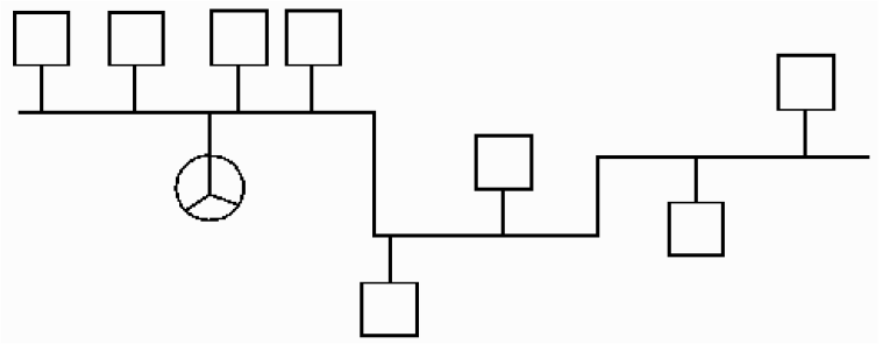


Figure 142 : Bus network

Besides these basic methods, there are also more complex functions enabling more varied forms, which are therefore better suited to the architectures of buildings or utilisation conditions. These forms (back-bone, tree structure and meshed network) are respectively derived from the bus, star and ring.

Back-bone

The back-bone is a line to which all the other structures are connected (like ribs that are articulated around the spine that transmits the nervous information).

Tree-structure

The tree-structure is the image of a tree where the number of branches increases as you move further away from the trunk.

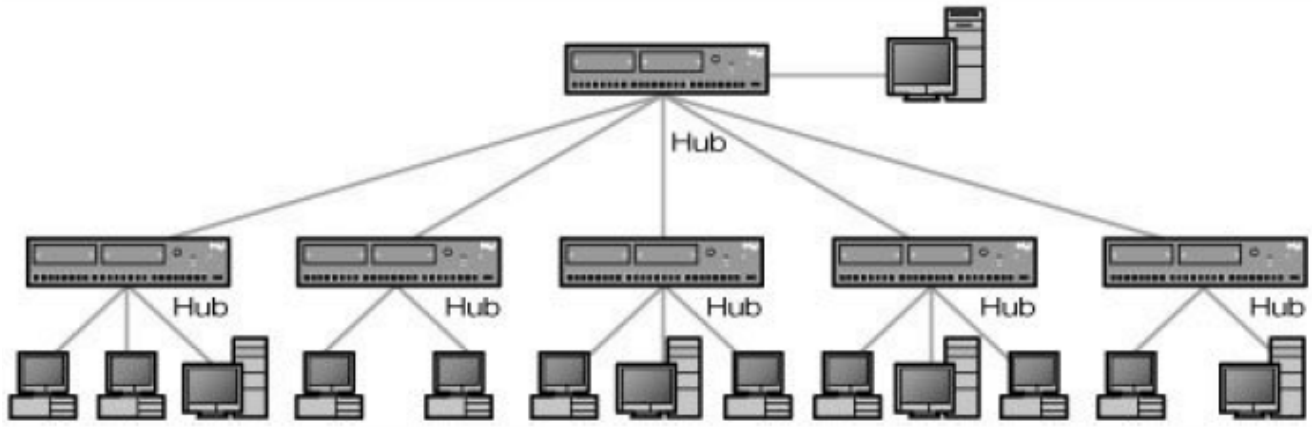


Figure 143 : A network tree-structure

Meshed network

This structure uses multiple point-to-point links providing several communication channels for going from one point to another (link redundancy).

The meshed network is the most complex structure to use, but also the most dependable because a failed link does not prevent the machines that are connected together from communicating, as the data can then take a different path.

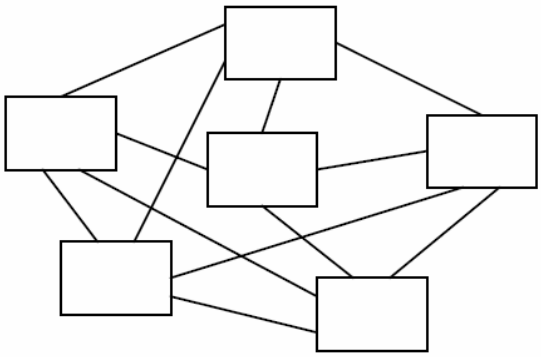


Figure 144 : Meshed network

All these topologies each have one (or more) specific defects: there is a risk of the ring opening, the star is very difficult to reconfigure, the bus creates a risk of collisions and, lastly, the meshed network requires the use of a **routing protocol**.

We can use several different connection topologies for a global network but, generally, we use meshed networks to cover a whole territory (city, country), a star or token ring network locally in a company (with the star network being preferred). Lastly, at the level of local area networks, the bus topology is very widely used.

It is not rare to see companies (IBM, HP, etc.) use a star network as the structure for a building and then create small islands with token ring networks. On this subject, let us wring the neck of some preconceived ideas:

- Token ring networks are often much faster than star or bus networks. It is true that the latter can become "saturated", that is to say they have a capacity for dialogue that decreases when the number of machines communicating with each other increases, whereas a token ring network always has the same level of performance,

- Star or bus networks sometimes used shared resources (distributed between several machines) which makes them as sensitive as token rings with respect to the failure of one machine,

- No type of network can automatically be recommended for an application and only a prior study can lead us to say that one is better than another.

6.5. THE MEDIUM

The medium is the physical link between the machines, this is therefore a technological aspect of networks that we are going to examine here.

There is a great variety of media for carrying an item of information, for example there are: **cable links** which we will see later on, and also **radio, infrared, ultrasound links, etc**.

For example, the field bus used by EUROCOPTER to test the TIGRE helicopter for obvious reasons uses a radio link to communicate with the sensors placed on its rotors.

However, it must be acknowledged that it is essentially wire links on the one hand and **optical fibres** on the other hand that are used.

We generally use a twisted pair to make small networks (in a building). Indeed, it is often simpler to accept a certain error rate than to use much more expensive media.

It should be noted that coaxial links are gradually disappearing.

6.5.1. Remark on propagation speeds

The propagation speed in a copper electrical wire is more or less of the order of 220,000 km/s, which corresponds to the formula:

$$V_\varphi = \frac{c}{\sqrt{\varepsilon \cdot \mu}}$$

Where ε and μ are respectively the conductivity and the permittivity of the medium, and c is the speed of light in a vacuum (299,792 km/s).

It is this formula, transformed into $V_\varphi = \frac{c}{n}$ that makes it possible to obtain the propagation speed of a light wave in an optical fibre.

We define n as the medium's index. In optical fibres, the medium's index is of the order of 1.5, and that gives a propagation speed of the order of 200,000 km/s. It can also be noted that in water, light reaches 75% of c and that it is comprised between 50 and 60% of c in glass.

You can therefore see that the information propagates faster in copper than in optical fibre.

6.5.2. Twisted pairs

The generic name "**twisted pair**" does not cover a uniform set of qualities; they are classified according to an accumulation of three standards:

The standard for frequencies

Here we are talking about the category of twisted pairs. There are 7 (or 8) categories of link with twisted pairs.

Category	Maximum Frequency	Speed	Utilisation area
Cat. 1	Telephone cable		No longer used
Cat. 2			
Cat. 3	16 MHz	10 Mbits/s	10 BASE-T
Cat. 4	20 MHz	16 Mbits/s	Token Ring or 100BASE-T4
Cat. 5	100 MHz	100 Mbits/s	100BASE-TX or ATM
Cat. 5^e	100 MHz	1 Gbit/s	Ethernet Gigabit or higher
Cat. 6	200 MHz	1 Gbit/s et +	
Cat. 7	600 MHz	1 Gbit/s et +	

Table 9: The standards for the frequencies

The standard for conductor cross-sections

This standard is better known under the name AWG (American Wire Gauge), it uses a number to give the diameter of the conductor and also its linear resistance. Anyway that was the original intention, unfortunately there seem to be some divergences.

However, we can give a line's cross-section and resistivity starting from the AWG reference:

$$diameter = 0.32'' \times 2^{\frac{AWG}{6}}$$

$$r = \mu \times 2^{\frac{AWG}{6}}$$

where μ = 3.06824 10-4 Ω/m

This standard, which was originally used for guitar strings (going up one key is equivalent to taking a string with AWG+1), is now dedicated to electrical wires. There are a very large number of tables (not all very coherent) to avoid having to do tedious and complicated calculations.

AWG	Ø : 1/10 mm	Ω / m
36	1.270	1.257
38	1.008	1.995
40	0.800	3.167
42	0.635	5.027
44	0.504	7.980

The standard for protections

Although in the vast majority of cases, the simple fact of using a differential transmission system is amply sufficient for protecting the information circulating through a twisted pair, there is a standard that describes the protection provided for a line. This classification uses a protection index.

UTP twisted pairs do not provide any particular protection, they are therefore theoretically more sensitive to disturbances than the other twisted pairs.

With FTP, an aluminium screen (generally not connected to ground) adds a shield with respect to external disturbances, in general it is an intermediate solution between the UTP and the S-FTP.

Unshielded cable	UTP	Unshielded Twisted Pair	
Foiled cable	FTP	Foiled Twisted Pair	
Shielded-foiled cable	S-FTP	Shielded-Foiled Twisted Pair	
Double-shielded cable	S-STP	Shielded-Shielded Twisted Pair	

Table 10: Cable protection standards

With S-FTP (sometimes called STP for Shielded Twisted Pair), in addition to the FTP screen, there is a braid connected to ground which plays the role of Faraday cage.

Lastly with S-STP which is the *nec plus ultra* of twisted pair connections, each pair is individually shielded and the cable as a whole is shielded too. These transmission lines are almost perfectly insulated from all disturbances, but they are horrendously expensive…

For all twisted pairs of the UTP, FTP or S-FTP type, there is a sly form of disturbance, crosstalk, that is to say the effect of one pair on another.

Another trick is used for twisted pairs therefore, in addition to their protection, to reduce the effect of crosstalk: the pairs' twisting frequency.

In a 2-pair link, pairs 1 and 2 are twisted at different frequencies (in general there is a factor of 2). In a 4-pair link, pairs 1 and 3 have the same frequency, as do pairs 2 and 4, but these two frequencies are different.

In general, the manufacturers provide tables summarising the characteristics of their links.

Protection index	F_{max}	Category	Attenuation
UTP or FTP	200 MHz	5th	19.8 dB/100m at 100 MHz
S-FTP	300 MHz	5th	19.8 dB/100m at 100 MHz
UTP, FTP or S-FTP	400 MHz	6	27 dB/100m at 200 MHz
S-STP	750 MHz	7	49.2 dB/100m at 600 MHz
S-STP	1200 MHz	8	64 dB/100 m at 1200 MHz

6.5.3. Optical fibres

There are two major types of optical fibres, single-mode fibres, where the light wave passes through the core of the fibre, and multimode fibres (index jump or gradient index), where the wave is reflected off the walls of the fibre.

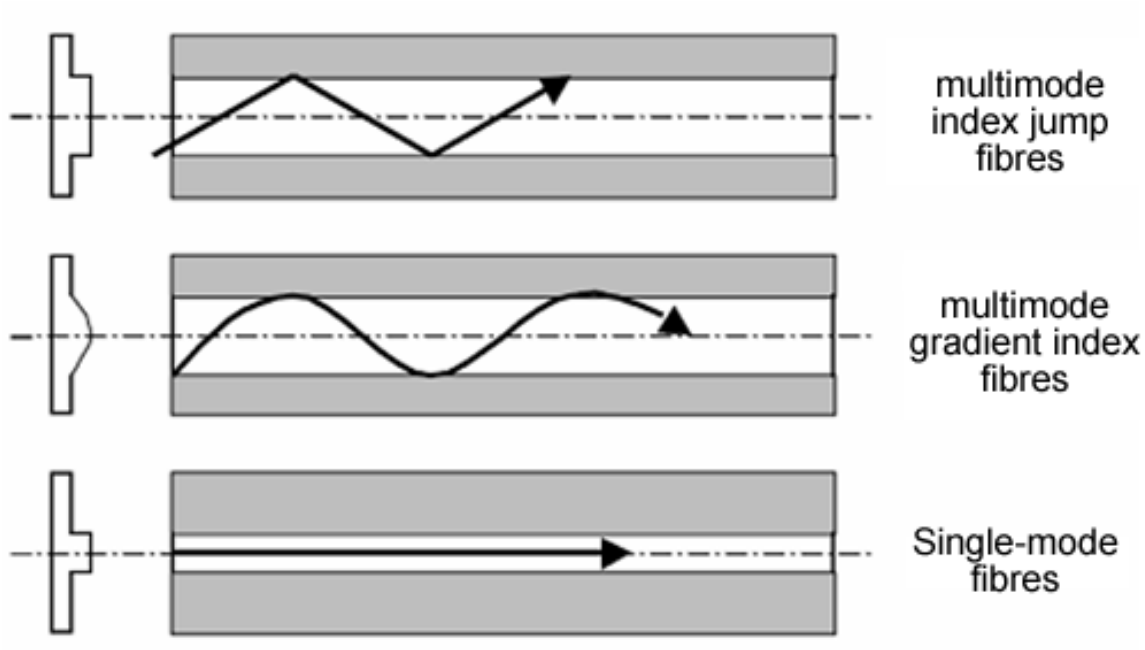


Figure 145 : The two types of optical fibre

Optical fibres appeared at the beginning of the nineteen sixties, but it was only in the seventies that they entered the area of networks, thanks to a better mastery of silicon and its doping agents which made it possible to obtain attenuations of the order of 20dB/km (instead of 1000dB/km originally).

In the eighties, the single-mode fibres arrived and line attenuation was raised to 2dB/km.

Nowadays, single-mode fibres provide attenuations of 0.2 to 0.3dB/km.

It is considered that index jump optical fibres (virtually abandoned these days) offer speeds of the order of 50 Mbits/s, whereas gradient index fibres make it possible to achieve speeds of 1Gbit/s. As for single-mode fibres, they achieve speeds of 40Gbit/s on distances going from 3km for standard single-mode fibres (G 652) up to 25km for "True Wave" fibres (G 655).

Optical fibres work in the near infrared (wavelength of 800 to 1600 nm). They are made of silicon oxide (SiO_2) with a very low OH^- ion density (these ions have an unfortunate tendency to absorb radiation in the near infrared). We then dope the core of the fibre with Germanium or phosphorus, which makes it possible to increase the core's index slightly. We also dope the sheath with boron or fluorine to reduce its index slightly.

The core of an optical fibre has an index (n_1) of about 1.5 for a diameter of the order of 200µm for index jump fibres, 62.5µm for gradient index fibres and 10µm for single-mode fibres.

The sheath has an index (n_2) very close to that of the core ($\frac{n2}{n1} = 0,99$), for an external fibre diameter of 380µm for index jump fibres and of 125µm for gradient index or single-mode fibres.

The whole is wrapped in an acrylic envelope which absorbs the mechanical shocks. It is the optical fibre alone that gives the transmission line its longitudinal mechanical properties, the plastic envelope only serves to protect it against shearing. In theory, a fibre with a diameter of 29mm should be able to withstand the weight of 216 elephants (1,300 tonnes).

Optical fibres are not, however, completely free from defects, even if they are perfectly immune to electromagnetic disturbances, they do have a tendency to diffuse the light, that is to say create fuzzy spots, and another problem is that they do not propagate all wavelengths at the same speed, which tends to deform the signals transmitted. Lastly, their natural attenuation limits the transmission distances.

6.5.4. The elements making up the physical layer

The other fundamental part of layer 1 that makes it possible to have connections without any analysis of the protocols is the **HUB**.

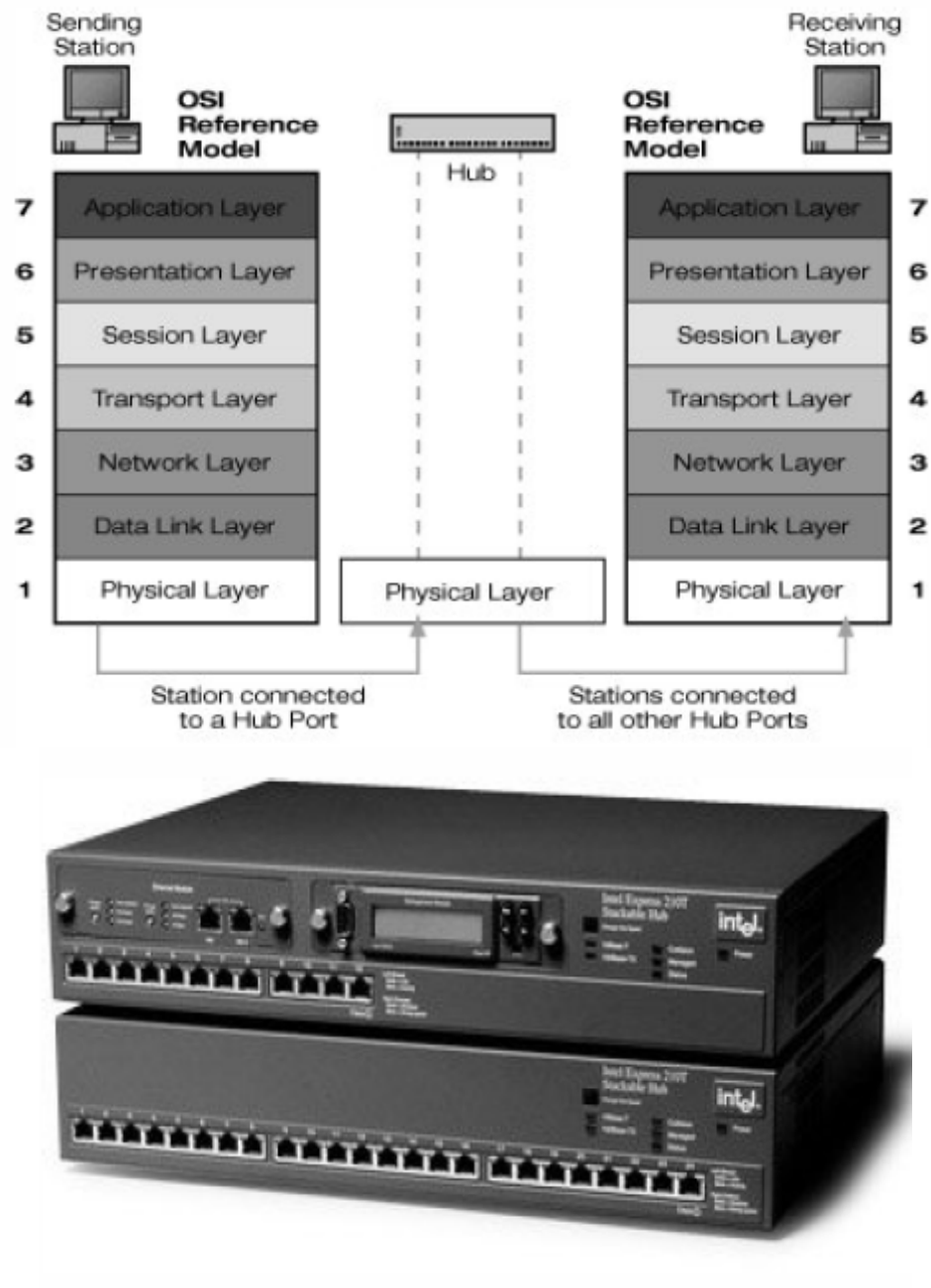


Figure 146 : The HUB

6.6. ETHERNET

Ethernet is the best known standard in the world of networks, even though it is one of the standards on the market with the lowest performance. Its worldwide use has made it an ideal tool for all companies. It was created in 1980 by Bob Metcalf (the founder of 3Com).

It is a low-cost interface that makes it possible to connect machines via a bus topology with a view to sharing resources

Although, strictly speaking, it does not really belong to the world of industrial local area networks, Ethernet nevertheless remains a company local area network, due to the fact that it is no longer rare to see PLCs connected to Ethernet or to the Internet via Ethernet (example: web server).

Ethernet only uses the two lowest layers of the OSI pyramid. The physical layer makes it possible to connect up via 3 families of media: twisted pair (UTP); coaxial cable (large or fine) and optical fibres. The access layer to the medium uses a standardised process (IEEE 802.3).

But what characterises this network the most is the incredible number of applications that have been developed "on" it. Indeed, a large number of protocols, such as TCP/IP, PROFIBUS, FIELDBUS, etc., have been grafted on to it

Definitively, Ethernet is a universal support, which explains why it is so popular.

We are therefore going to look at Ethernet's standard frame, and then the connections used. Lastly, we will examine the following chapters: ‘TCP/IP and PROFIBUS protocol’.

This frame has undergone many changes since it was first put in place by the progenitors of the network. Originally, the length of the frame was coded in a specific field. Since then it has been transformed into a definition of the encapsulated data type.

6.6.1. Fundamental frame

The fundamental Ethernet frame consists of 6 fields:

- The start field (preamble),
- The Destination Address Field,
- The Source Address Field,
- The length or type field,
- The data field,

- the CRC field.

	Ethernet Header				
Preamble / SFD	Destination Address	Source Address	Type / Length	Data	Frame Check Sequence
7 + 1 bytes	6 bytes	6 bytes	2 bytes	46 – 1500 bytes	4 bytes

6.6.1.1. The preamble

The preamble is made up of 7 bytes making it possible to regenerate the sender's clock, and then the start delimiter. The first 7 bytes are made up of an alternation of 1s and 0s, forming the hexadecimal code AA_H. Whereas the start byte forms the word AB_H (only the last bit changes).

6.6.1.2. The destination and source addresses

The addresses are made up of 2 zones of 3 bytes each. The first one makes it possible to define the name of the manufacturer (it is provided by a regulation organisation) whereas the 3 LSB bytes code the board's serial number. These addresses are generally called the MAC addresses (due to the name of the layer where they are used). A specific feature of this addressing is that it does not use odd addresses to talk with individual machines. These addresses are strictly reserved for multiple transmissions (multicast and broadcast).

6.6.1.3. The type

As the Ethernet frames encapsulate a large number of other protocols, we use this field to indicate the type of information that is encapsulated. For example the IP packets are coded $0800_{H.}$ See the table of values below.

6.6.1.4. The data field

The data field is used by the higher layers to place data. This field must contain at least 46 bytes. If a frame does not contain enough data to fill the space, padding bits are used to fill the space.

6.6.1.5. Table of Ethertypes.

Type (Hex)	Function
0 – 05DC	Size of the data field
0600	XEROX
0800	Internet Protocol
0805	X25 version 3
0806	Address Resolution Protocol
0BAD	Banyan System
0BAF	Banyan Vines
8035	Reverse ARP
8037	Novell Netware IPX (New)
809B	Ethertalk (Appletalk)
80D5	IBM SNA Services over Ethernet
80F3	Apple Talk ARP
8137	Novell Netware IPX (Old)
8138	Reserved for Novell, Inc.
814C	SNMP over Ethernet
8191	NetBIOS/NetBEUI (PC)
817D	XTP
81D6	Artisoft Lantastic
81D7	Artisoft Lantastic
8203-8205	QNX Software Systms Ltd.
86DD	IP version 6
AAAA	DECNET (VAX)

Figure 147 : Extracts from the Standardised Ethertypes table

6.6.2.

6.6.3. Ethernet's Physical layer

Originally, Ethernet was designed to be used exclusively with 50 Ω coaxial cable, with a bus topology, therefore with termination impedances.

Because of its characteristics, it was able to transmit the information signal at 10 Mbaud in baseband over a distance of 500 m. Hence its wiring name: 10BASE-5.

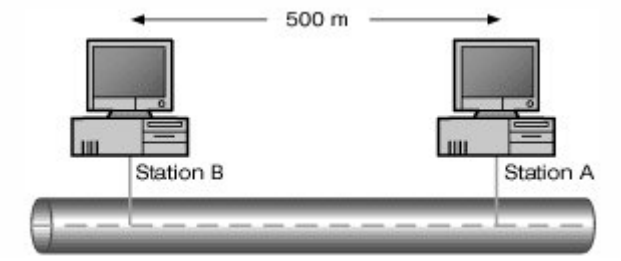


Figure 148: Transmission with coax cable

By breaking down 10BASE-5 we get **10** Mbaud in **BASE** band over **5**00 m.

The use of 10BASE-5 wiring requires the installation of a "rigid" main cable (the Back Bone), to which we connect transceivers or line-extender amplifiers with vampire taps (enabling connections without breaking the line). This forms an MAU: Media Attachment Unit.

This large cable must have a length that is an odd multiple of 3.4 m (23.4 m, 70.2 m, 117 m, 163.8, etc.) without, however, exceeding 500 m that is to say 491 m or 21 segments of coaxial cable. Its attenuation must not exceed 8.5 dB for 500 m at 10 MHz and its resistivity must be lower than 10 MΩ/m.

But the development of the techniques and above all the real need to lower the costs of wiring has made it possible to put in place other versions such as the 10BASE-2 that uses fine coaxial cable which is much less costly, with the coaxial cables connected directly at the level of each machine by Tee-connectors. However, the distance between machines is reduced to 200 m and it is prohibited to connect a machine via a strand to the Tee-connector.

This wiring is called fine coaxial (or yellow cable) wiring, its attenuation shoots up to 4.6 dB for 100 m at 10 MHz, it is also propagates the electromagnetic wave more slowly than the large coaxial cable.

These types of wiring always have the drawback of not allowing the installation (or withdrawal) of machines without interrupting the communications and the failure of one element in the network affects all the other elements.

The need to use a more advanced type of wiring soon made itself felt, hence the implementation by the IEEE committee of twisted pair wirings. This technology makes it possible to use unshielded twisted wires, that is to say an extremely economical type of wiring, using a star technology instead of a bus and the now famous **RJ45 connector**.

We can then make a connection with the machines via an interface (AUI for Attachment Unit Interface), made up of a CANON DB15 type connector.

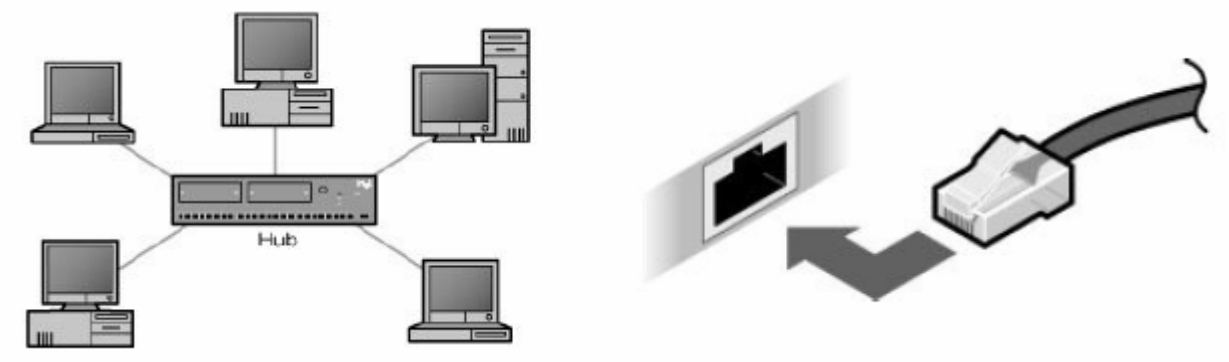


Figure 149 : Ethernet star network with connection to a HUB using an RJ45 connector

Twisted pair cables have a characteristic impedance of 120Ω, they are even slower than the fine coaxial cable with respect to the propagation of electromagnetic signals. Lastly, it imposes an attenuation of 11.5dB for 100m at 10MHz. They also have other defects such as crosstalk (influence of one strand on the other), which must not exceed 26dB at 10MHz.

This wiring method is called **10BASE-T**. As it uses 2 pairs of twisted wire, one for transmission, and the other for reception, it enables a **FULL DUPLEX** transfer of the information.

However, the improvements made to Ethernet do not stop there as there are operating modes that use optical fibre, these are the 100BASE-FL modes. In general, these modes use Back-bone type topologies (made of optical fibre) to which the various forms of network (token ring, bus or star) are connected.

The utilisation of star wirings or Back-bone structures imposes the use of a Hub (to transmit the information in the star) or of a Switch (to modify the speeds, or even the protocols linked to broadcasting methods) respectively. As long as a routing does not have to be established, you can keep the Ethernet protocols.

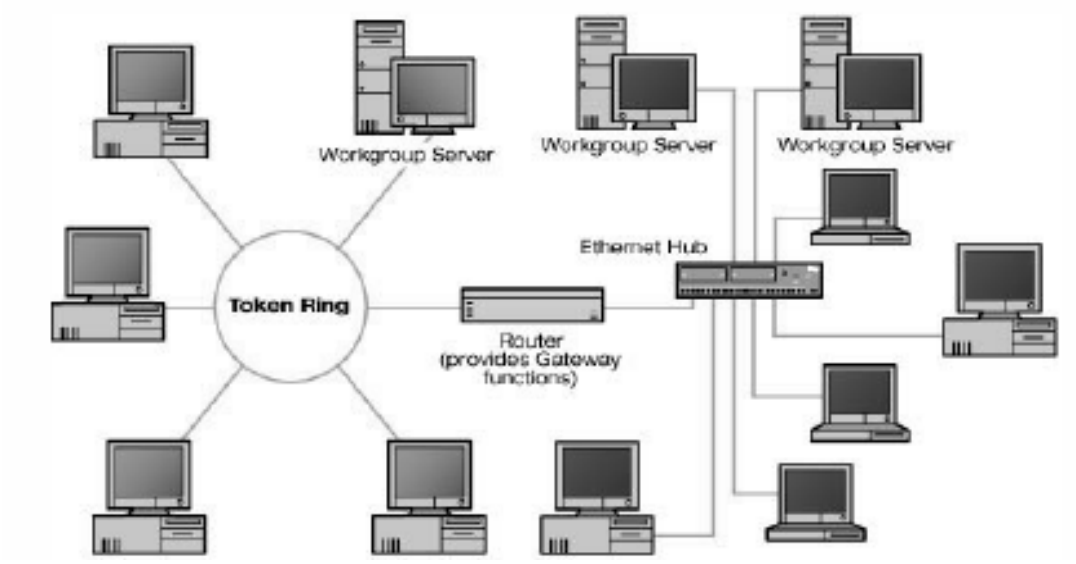


Figure 150 : Example of network routing

However, as soon as you have segmentation into networks and sub-networks, it is essential to include a layer-3 or even a layer-4 protocol; the TCP/IP protocol being the most widely used.

6.6.4. The Ethernet link layer

The link layer is based on the frame described earlier, because of its size it enables an easy encapsulation of messages from the higher layers. So, a standardised IEEE 802.3 frame coming directly from the OSI pyramid's application layer can be presented as follows:

		Ethernet Header			Data Field	
PA	SFD	DA	SA	Type	Data Packet	FCS
7	1	6	6	2	46 – 1500 bytes	4

Figure 151 : Conventional IEEE 802.3 standardised Ethernet II frame

Whereas the utilisation of encapsulations makes it possible to further breakdown the frames received:

		Ethernet Header			Data Field				
PA	SFD	DA	SA	Lenght	802.2 LLC Header			Data Packet	FCS
					DSAP	SSAP	Ctrl		
7	1	6	6	2	1	1	1	43 – 1497 bytes	4

Figure 152 : Ethernet frame with an LLC encapsulation

As you can see, we have placed information in the data field that comes from the higher layers (here the LLC sub-layer).

As this information addresses elements of the machine that are not pointed to be MAC addresses, the data encapsulated on the data field of the MAC frame therefore starts with the definition of the destination service and of the source service (in our case, this concerns the addressing of the SAP in the LLC layer entering into communication). We therefore reduce the size of the data field by as much.

Still in the framework of the protocol, we must examine the addressing principles a bit more closely. Thus, the 6 bytes in the address field are broken down into 2 times 3 bytes. The first byte (or the MSB byte) makes it possible to identify the manufacturer.

It must also be possible to direct the data correctly, that is to say establish which service the data present on the bus must be sent to. So we have reconverted the length field into a

field of 2 bytes coding the type of information transmitted. For example, an IP field is referenced 0800_{16}.

The first byte (or the MSB byte) makes it possible to identify the manufacturer. All countries identify the manufacturers in the same way, but this code is not transmitted by the routing algorithms. It is therefore only known locally. The manufacturers can therefore use a given address twice (or more) provided they do not sell those boards in the same geographical zone, hence a certain risk if you buy network boards in two different countries that are made by the same manufacturer.

The second byte (or the LSB byte) is used to identify the board. This byte must imperatively be even.

But even if the address is even, there are frames with odd addresses that circulate on the network (this can be seen using a spy). These are frames intended for groups of boards (MULTICAST), or even for all the boards (BROADCAST).

The BROADCAST address is quite easy to remember because it only consists of 1s (in binary) which in hexadecimal gives "FF FF FF FF FF FF". All the machines are supposed to answer to this broadcast address if the content of the message concerns them. We will see how this functions in the framework of TCP/IP utilisation.

As for the MULTICAST addresses, there are millions of them (approximately eight million), they are obtained by an algorithm from the addresses of the machines concerned by the message. This algorithm, related to the CRC calculations, makes it possible to re-create automatically an address common to a set of machines concerned by the multicast. By applying the same calculation, each machine will be able to find its own address.

The utilisation of MULTICAST commands is more limited than the utilisation of the BROADCAST commands. However, it is not rare to see this type of frame circulating when using the higher layers.

6.7. THE DATA TRANSMISSION PROTOCOLS

We have three fundamental transmission protocols:

- **MODBUS**, this has nearly disappeared nowadays,
- **PROFIBUS**, this is one of the most commonly used protocols for connecting PLCs in a network
- **TCP/IP**, this is very often used to enable a computer to communicate with PLCs.

6.7.1. TCP/IP

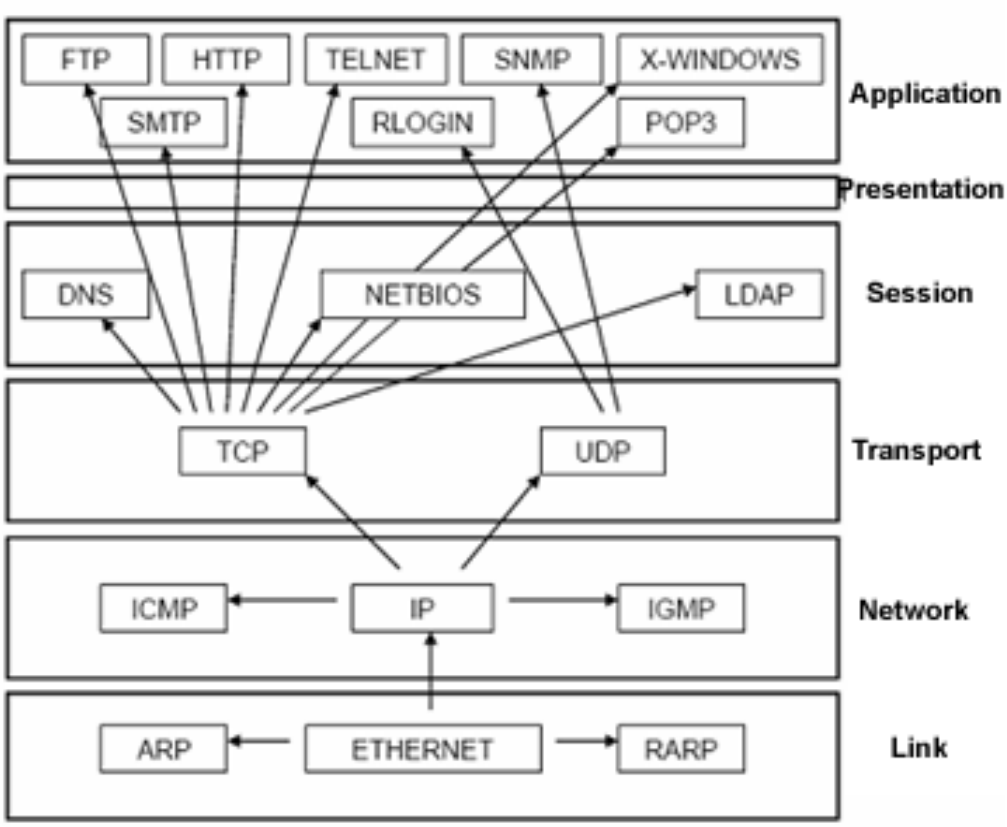


Figure 153 : Presentation of the OSI model adapted for some elements in the TCP/IP suite

6.7.2. The IP protocol

The role of the IP protocol is to route the information, that is to say allow the information to be carried from one network to another passing through several routers, and therefore go beyond layer 2 to cross the router's wall and reach its target by taking the shortest path. To do this, we use a new addressing format, the **IP address**.

The IP address is made up, as we will see later, of 32 bits, but to simplify its utilisation (we may find this simplification a bit too much "IT oriented", we have grouped these terms by byte and listed the address in dotted decimal (4 decimal numbers lower than 255 and separated by decimal points).

This address is found encapsulated in the Ethernet frame (in the first bytes of the IEEE 802.3 frame's data field). It is placed inside the IP header.

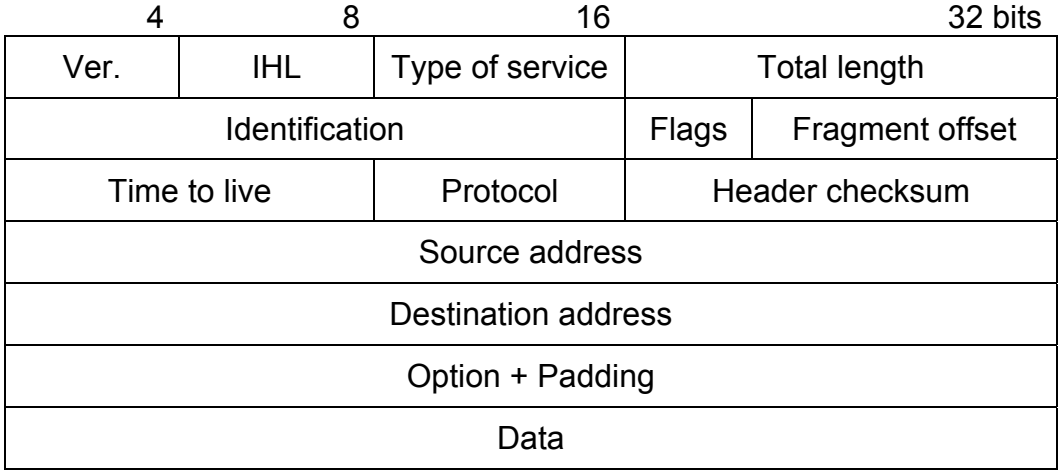


Figure 154: IP header

The IP address is defined by a set of 4 bytes, which makes it possible to define 2^{32} addresses (4,300 billion nodes). These addresses are arranged in 5 classes, depending on the value of the address's first bits.

- class A makes it possible to create 126 networks of 2^{24} machines (16 million), using addresses 1.0.0.0 to 126.255.255.255.
- class B makes it possible to create 16384 networks of 2^{16} machines (65 536), using addresses 128.1.0.0 to 191.255.255.255.
- class C makes it possible to create 221 networks de 2^{56} machines(2 million), using addresses 192.0.1.0 to 223.255.255.255.
- class D allows a single IP frame IP to address several machines (MULTICAST but in IP), it uses the addresses comprised between 224.0.0.0 and 239.255.255.255.
- Lastly class E is reserved for future uses, it nevertheless uses the addresses comprised between 240.0.0.0 and 247.255.255.255.

This distribution also makes it possible to define not only a "tree structure" with the network's number, but also a hierarchical organisation.

At the level of class A, all the machines are routers or gateways interconnected between a small number of networks. At this level, we form the interconnections between the major intercontinental networks.

At the level of class C, we have a small number of computers connected to a large number of subnetworks (subnetworks controlled by the class A and B routers).
Here we are in the "general public" domain with lots of little networks to which the users' machines are connected.

We therefore obtain the following representation of IP addresses:

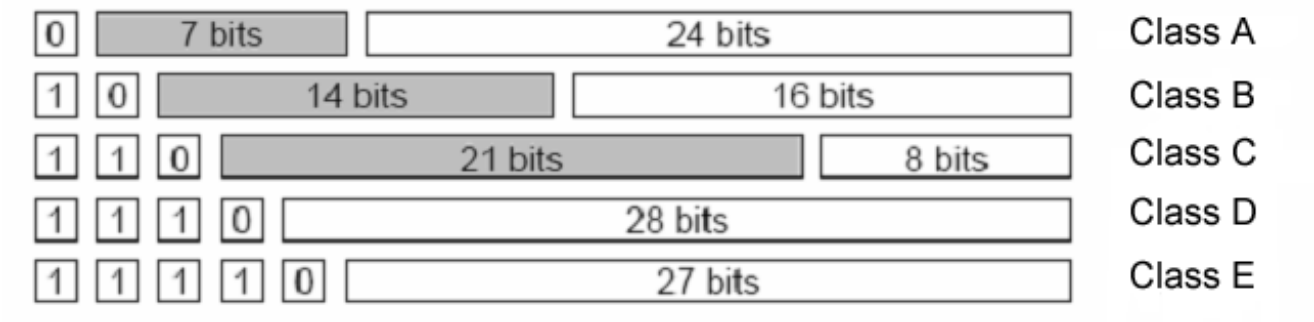


Figure 155: Representation of IP addresses

In grey, the coding of the network's "name", in white the coding of the machine's "name".

To allow a machine to identify itself correctly in its environment, another parameter besides its IP address is essential, this is the sub-network mask.

This value, made up of 4 bytes allows it to define the address of the network to which it is connected and, by deduction, know its "name" on the network.

For example, for a machine with a class C IP address:

IP	192	10	15	20
& Mask	255	255	255	0
= Network	192	10	15	0

IP	192	10	15	20
+ Mask	255	255	255	0
= Machine	255	255	255	20

Caution: the numbers given above are base 10, but to better understand the operation presented in this example we must translate the numbers into hexadecimal.

IP	C0	A	F	15
& Mask	FF	FF	FF	0
= Network	C0	A	F	0

IP	C0	A	F	15
+ Mask	FF	FF	FF	0
= Machine	FF	FF	FF	15

Now let us analyse an IP header line by line.

4	8	16	32 bits
Ver.	IHL	Type of service	Total length
Identification		Flags	Fragment offset
Time to live	Protocol	Header checksum	
Source address			
Destination address			
Option + Padding			
Data			

Figure 156: IP header

The first 4 bits define the IP version number, and in general this number is 4 (IPv4).

The **IHL field** (Ip Header Length) gives the number of 32-bit words contained in the header (the data are not included in this), but including the options and Padding.

The **TOS field** (Type Of Service) is in fact made up of 5 subsets

- The first 3 bits form a set that codes the message's priority.
  - 000 : routine (normal)
  - 001 : priority
  - 010 : immediate
  - 011 : flash
  - 101 : critical
  - 110 : internetwork control
  - 111 : network control
- The following bit is at 0.
- The last 4 bits are used to describe the service requested. They are exclusive (only one bit can be validated for a frame)
  - D : Minimise the delay, used for small messages.

- T : Maximise the speed, this option is of course used once there is a large amount of data to be transmitted.
- R : Maximise network management.
- C : Minimise the cost, not used in general.

The **TLF field** (Total Length Field) is used to fix the total size of the IP packet (we therefore know the size of the data field's header). This field thus makes it possible to eliminate the padding terms used to bring up small IP frames to the minimum format of the Ethernet frame (46 bytes).

The **identification field** gives a unique number to each of a machine's datagrams which is incremented by 1 after each transmission. It thus makes it possible to know the order number of each datagram transmitted. In the case of a datagram being fragmented, the identification number is duplicated so that it is the same on all the fragments of any given message.

The 3 Flag bits make it possible to control the fragmentations.

- "More Fragments" is used to indicate that other fragments of the same datagram are going to follow. The last fragment of the datagram necessarily has this bit at "0" (No more Fragments).
- "Don't Fragment" is used to prohibit fragmentation of the datagram (this may lead to errors insofar as the maximum size of the message authorised by the network's lower layer is smaller than the size of the datagram).
- The last flag bit is not used and is therefore at 0.

The **Fragment Offset field** makes it possible to define the fragment's "order number", as each fragment can be routed independently from the others, it can pass via a shorter path than its predecessor and therefore arrive at its destination before it. The target machine then deals with recomposing the datagram by putting the fragments (which have the same identity) back in the right order (increasing order of Fragment Offsets).

Caution: fragmentation is a double-edged blade ways because if a fragment gets lost, the whole message will have to be retransmitted.

The **TTL field** (Time To Live) is used to set the maximum number of router's that a datagram can go through. This field is initialised at a certain value to begin with. Then each router goes through decrements. When it arrives at 0, the datagram is rejected, and the sender is notified. Contrary to what you might think, this method is not coercive but, rather, makes it possible to eliminate the packets that have got lost on the network.

The **protocol field** is used to define what type of service (TCP, UDP, etc.) the IP's data field uses to encapsulate its data. You should note certain values of this field (1 for ICMP, 6 for TCP and 17 for UDP)

The **HCS field** (Header Check Sum) contains a CRC code that makes it possible to validate the IP header and exclusively the IP header encapsulated in the Ethernet frame. It should be noted that the actual HCS field is itself included in the header, therefore in the control field, and this is why we consider it equals zero for the calculation.

Lastly, we present the sender's and the destination's IP addresses then, before stacking the data, we systematically leave 32 bits free for the definition of options (if there are any) or for later use.

6.7.3. The IP frame's options

The IP frame may contain options. They are used for the debugging of networks.
However, they remain optional although they must be implemented by all the elements in a network. The options are described by a single byte.

The byte's first bit is the copy indicator, it is used to signify whether the information concerning this option must be copied for each of the fragments that may exist (bit at 1) or not (bit at 0).

The following 2 bits define the option class:

- 00 : control class
- 01 : reserved for future use
- 10 : debugging and measuring
- 11 : reserved for future use

The last 5 bits define the option.

The following options should be noted in particular:

- Option 7 is used to record the path taken,

- Options 9 and 3 are used to fix for option 9, the whole path that the message must take, and for option 3 (loose source route) the points that a packet must obligatorily pass through.

- Option 4 (class 2) is used to create a time-dating for the packets. A time information is added to the frame each time it passes through a router.

As the options only use a single byte for their definition, we systematically add padding bytes to complete the 4-byte packet started by the option's definition. We use the term PADDING for this.

Copy	Class	Number	Value	Name			Reference
0	0	0	0	EOOL	-	End of Options List	[RFC791,JBP]
0	0	1	1	NOP	-	No Operation	[RFC791,JBP]
1	0	2	130	SEC	-	Security	[RFC1108]
1	0	3	131	LSR	-	Loose Source Route	[RFC791,JBP]
0	2	4	68	TS	-	Time Stamp	[RFC791,JBP]
1	0	5	133	E-SEC	-	Extended Security	[RFC1108]
1	0	6	134	CIPSO	-	Commercial Security	[???]
0	0	7	7	RR	-	Record Route	[RFC791,JBP]
1	0	8	136	SID	-	Stream ID	[RFC791,JBP]
1	0	9	137	SSR	-	Strict Source Route	[RFC791,JBP]
0	0	10	10	ZSU	-	Experimental Measurement	[ZSu]
0	0	11	11	MTUP	-	MTU Probe	[RFC1191]
0	0	12	12	MTUR	-	MTU Reply	[RFC1191]
1	2	13	205	FINN	-	Experimental Flow Control	[Finn]
1	0	14	142	VISA	-	Expermental Access Control	[Estrin]
0	0	15	15	ENCODE	-	???	[VerSteeg]
1	0	16	144	IMITD	-	IMI Traffic Descriptor	[Lee]
1	0	17	145	EIP	-	Extended Internet Protocol	[RFC1385]
0	2	18	82	TR	-	Traceroute	[RFC1393]
1	0	19	147	ADDEXT	-	Address Extension	[Ullmann IPv7]
1	0	20	148	RTRALT	-	Router Alert	[RFC2113]
1	0	21	149	SDB	-	Selective Directed Broadcast	[Graff]
1	0	22	150	NSAPA	-	NSAP Addresses	[Carpenter]
1	0	23	151	DPS	-	Dynamic Packet State	[Malis]
1	0	24	152	UMP	-	Upstream Multicast Pkt.	[Farinacci]

Figure 157: IP frame options

6.7.4. IP and Ethernet

The utilisation of the IP protocol on the Internet means that two problems must be defined: on the one hand where and how to store the data and the IP frame's identification (in the physical sense); and on the other hand, how do we associate the functioning of the two protocols (in the logic sense).

When sending a frame

The IP protocol and the Ethernet protocol are linked to each other by an exchange table called the ARP table (for Address Resolution Protocol). The purpose of this table is to associate an IP address with a MAC address. This is done using a very simple procedure.

The three parameters that must be defined for all the machines using IPs are respectively: IP addresses, sub-network mask and IP address of its gateway (which must be in the same physical network as the machine).

When sending an IP frame, the source machine compares the IP address of the target machine with the address of its sub-network. Then, if it does not recognise the address of its own network, it knows that only the gateway will be able to send its message to the Web, it will therefore attempt to contact its gateway. To do this, it will use the same procedure as the one that allows it to contact a machine on the same sub-network.

If the target is on the same sub-network, considering that the source machine has remained inactive for quite a long time, it must therefore associate an address at the level of the MAC address with the IP address of its target (IP is not a network, remember what I said about it, only the MAC layer provides access to the physical link). In order to identify the address of its target, the source machine sends an ARP Request frame.

The ARP frame uses the data field of the local area network's frame to present its header and places the hexadecimal 0806 code in the encapsulated frame's type definition field.

<table>
<tr><td colspan="2">16</td><td>32 bits</td></tr>
<tr><td colspan="2">Hardware Type</td><td>Protocol Type</td></tr>
<tr><td>HLen (8)</td><td>Plen (8)</td><td>Operation</td></tr>
<tr><td colspan="3">Sender Hardware Address</td></tr>
<tr><td colspan="3">Sender Protocol Address</td></tr>
<tr><td colspan="3">Target Hardware Address</td></tr>
<tr><td colspan="3">Target Protocol Address</td></tr>
</table>

Figure 158 : ARP header

ARP header

The ARP Request frame then sends a message to all the machines (BROADCAST ALL) asking them to answer the sender if their IP address is present in the frame. The machine concerned will answer using an ARP Reply frame directly addressed to the sender of the request.

The ARP header is made up of two types of field, on the one hand fields oriented towards the physical layer (HARDWARE), and on the other hand the fields oriented towards the IP layer (PROTOCOL). Everything begins with 4 bytes respectively defining the type of MAC layer used for the first two (Ethernet is characterised by a 1), and for the last two the type of network layer used (we use the 0x0800 code for IP).

The Hlen and Plen fields define respectively the size in bytes of the addresses of the MAC layer (6 for Ethernet) and of those of the network layer (4 for IP).

The OP field indicates whether it is an ARP request or reply (1 for a request and 2 for a reply) or a RARP command (Reverse Address Resolution Protocol).

The fields given after that are respectively the source's MAC address, then come the target's MAC and network addresses.

When making the request, the field corresponding to the target's MAC address is left empty. When giving the reply, it is the machine that was the target that becomes the source so it is the latter that provides the source address and target address values at the level of the network layer, like at the level of the MAC layer, as the target machine is then the machine that sent the request.

These data are then stored in a dynamic table (which clears itself if we do not refresh it after 30 seconds). So, we automatically clear the input data in the table if they are no longer used.

6.7.5. The PING command

The **PING** command comes from Packet INternet Groper, it is used to test the reply from a target machine to a call from a source machine. This call is an echo that must be returned by the target machine.

The message sent is therefore made up of an IP header with the address of the source machine and of the target machine, then we find, encapsulated in the IP data field the ICMP header which has a type and a code at 0, then follows a code identifying the source machine and the sequence number.

The message returned by the target is also made up of an IP header and of an ICMP header where the type equals 8 and the code equals 0, the identification field and the sequence number are those of the echo request (the message sent).

6.7.6. IP operation

6.7.6.1. Identification of the local addresses

When a machine wishes to communicate with another one, it must use a physical network support to transmit its data. Let us take the example of a fictitious network where 3 machines are in control.

These 3 machines are in fact a router and 2 computers. Each of them has its own IP address and, apart from the router which uses 2 types of network, the 2 computers exclusively use the Ethernet network. We therefore have the following structure:

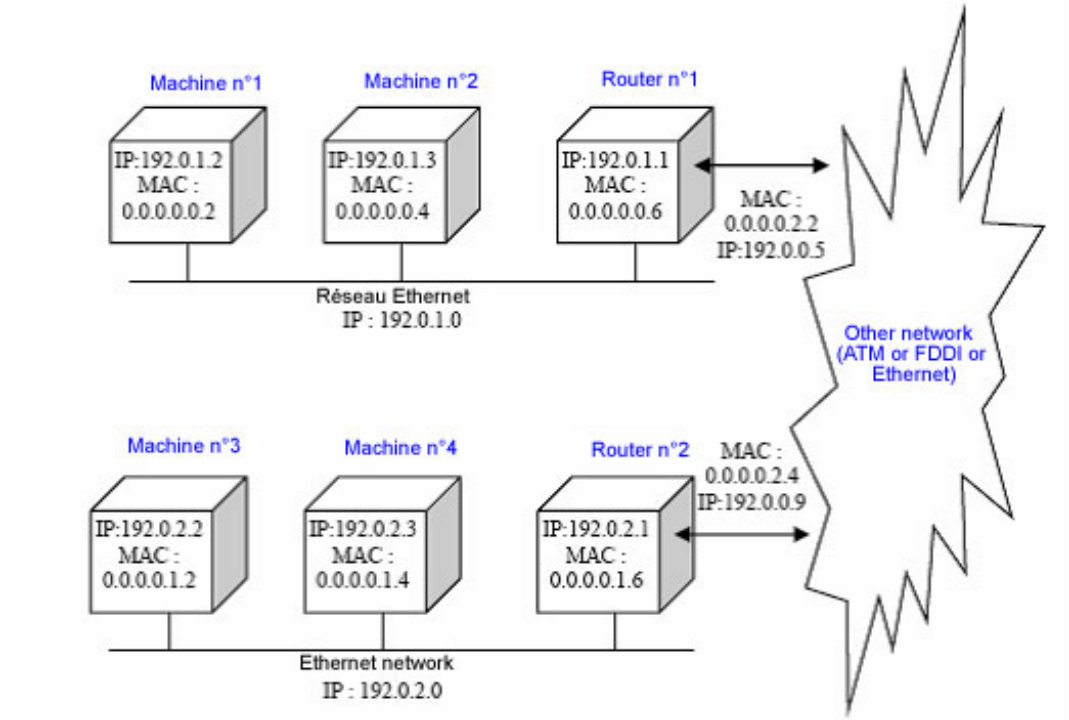


Figure 159: Ethernet network structure

Let us look at two cases, the first one where Machine No.1 wants to talk with Machine No.2, and another one where Machine 2 wants to talk with Machine 4. After that, we will examine an example with even more networks.

In our first case, we are going to imagine that a user on Machine No.1 wants to communicate with Machine No.2, and as the latter machine has been inactive for a very long time, it does not know its neighbours.

To communicate, it needs the target machine's physical address (if the latter is on the same network as itself) or the physical address of the router if the target machine is not on its local area network.

To define its network's address, the source machine performs a logic AND between its IP address and the mask. We obtain:

192	0	1	2
255	255	255	0
192	0	1	0

The network is therefore 192.0.1.0. As the target's address is 192.0.1.3, we also look for its network:

192	0	1	3
255	255	255	0
192	0	1	0

The two machines are therefore in the same network. The source machine will therefore ask all the other machines in the network which one's **IP address is 192.0.1.3**.

This request is made by sending an ARP request in BROADCAST ALL mode, that is to say to the address of all the machines. We therefore obtain an ARP frame containing the following information:

Sender Hardware Address	0 0 0 0 0 2
Sender Protocol Address	192.0.1.2
Target Hardware Address	FF FF FF FF FF FF
Target Protocol Address	192.0.1.3

Then an ARP reply frame comes from the target, answering:

Sender Hardware Address	0 0 0 0 0 4
Sender Protocol Address	192.0.1.3
Target Hardware Address	0 0 0 0 0 2
Target Protocol Address	192.0.1.2

The 2 machines now know each other and communicate via the Ethernet network, without having to re-send any ARP commands. As long as these 2 machines continue to talk with each other, they will keep locally the association of the IP and MAC addresses, in the ARP

table. Furthermore, a routing table now allows them to know the route they must take to talk with each other.

Now let us take the second case: Machine No.2 wants to talk with Machine No.4, so we start the procedure over again:

Definition of the source machine's network:

192	0	1	3
255	255	255	0
192	0	1	0

Definition of the target machine's network:

192	0	2	3
255	255	255	0
192	0	2	0

As the networks are different, the source machine knows that it cannot talk directly with its target, it must therefore absolutely communicate with its router to obtain the transfer of information to the target.

As the machine has not communicated with the router for a long time, it must associate the router's MAC with the latter's IP address. To do that, it sends an ARP request in BROADCAST ALL mode.

Sender Hardware Address	0 0 0 0 0 4
Sender Protocol Address	192.0.1.3
Target Hardware Address	FF FF FF FF FF FF
Target Protocol Address	192.0.1.1

Then the router returns an ARP reply frame:

Sender Hardware Address	0 0 0 0 0 6
Sender Protocol Address	192.0.1.1
Target Hardware Address	0 0 0 0 0 4
Target Protocol Address	192.0.1.3

Machine No.2 now knows how to talk with its router. It is now up to this machine to establish the end of communication. There are several possibilities for doing that: static routing (the path to be taken has been set by an administrator) or dynamic routing (the routers must find each other without any outside help).

But, whatever the routing method used, the principle stays the same, the router like the other machines, checks the presence of the target network in its routing table. And if it is not present, it consults the other machines using the route exchange protocols (such as the IRDP protocol for Internet Router Discovery Protocol).

6.7.7. Routing of the IP packets

6.7.7.1. The RIP protocol

The main protocol used by routers is the **RIP protocol** (for Routing Information Protocol). This is the protocol that allows a router to define automatically and dynamically (that is to say without any outside intervention) the shortest path to be taken to reach a target.

The routing information is not centralised but is distributed locally. Each router has its own routing table called the RIP table. There is no routing information centralisation node in the network, nor is there any router that knows all the available networks.

The definition of the shortest path involves the utilisation of a distance measurement, based on the HOP (a HOP corresponds to the passing of a router). The distance is therefore not real but fictitious, indeed, it does not matter whether a network measures several hundreds of kilometres, what counts is that you must use the least number of routers possible to transfer an item of information.

The RIP table of routers therefore stores 3 pieces of information:

- The IP number of the destination network (R).
- The IP number of the next router making it possible to access it (P).
- the path's total distance in HOPs (H).

Regularly (every 30 seconds approximately), the routers send out their RIP tables. Fortunately, there are 2 safeguards on these exchanges, which on a network as vast as the Internet could pose serious problems of saturation. The first one is based on the fact that the tables are propagated with a limitation to 15 HOPs of the maximum accessible distance. The second one is based on the fact that only the shortest path is memorised.

6.7.7.2. Distribution and constitution of the RIP tables

Now, let us imagine the following situation:

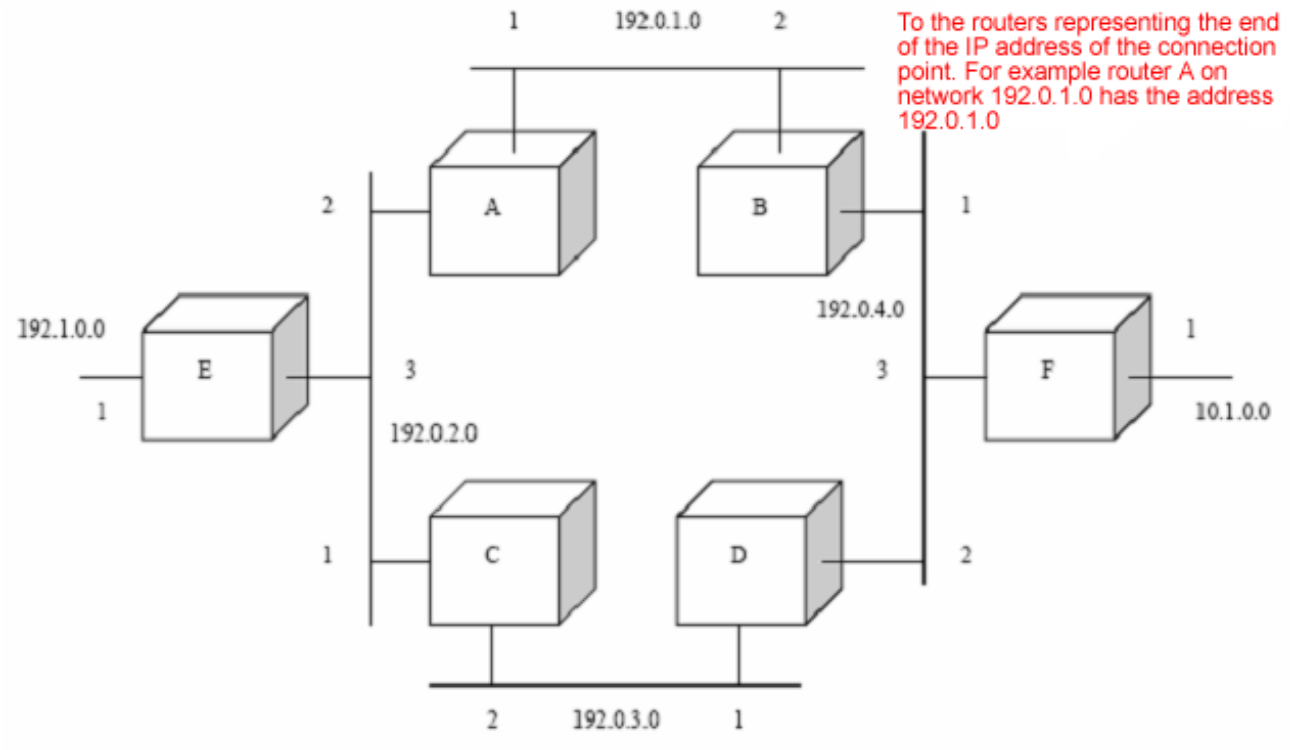


Figure 160: Network with routers

The numbers 1, 2 and 3 placed on the wires connected to the routers represent the end of the connection point's IP address.

For example, router A on network 192.0.1.0 has the address 192.0.1.1

If we analyse router A, we obtain the following RIP table:

Network	Gateway	HOP	Remark
192.0.1.0		0	Other path 4 HOPs away (eliminated)
192.0.2.0		0	Other path 4 HOPs away (eliminated)
192.0.3.0	192.0.2.1	1	Other path 2 HOPs away (eliminated)
192.0.4.0	192.0.1.2	1	Other path 2 HOPs away (eliminated)
192.1.0.0	192.0.2.3	1	Other path 4 HOPs away (eliminated)
10.1.0.0	192.0.1.2	2	Other path 3 HOPs away (eliminated)

As for router B, it has the following table:

Network	Gateway	HOP	Remark
192.0.1.0		0	Other path 4 HOPs away (eliminated)
192.0.4.0		0	Other path 4 HOPs away (eliminated)
192.0.3.0	192.0.4.2	1	Other path 2 HOPs away (eliminated)
192.0.2.0	192.0.1.1	1	Other path 2 HOPs away (eliminated)
192.1.0.0	192.0.1.1	2	Other path 4 HOPs away (eliminated)
10.1.0.0	192.0.4.3	1	Other path 3 HOPs away (eliminated)

These tables are maintained dynamically, that means for example that if network 192.0.3.0 fails (a link is broken, for instance), as soon as router C (or router D) tries to transmit on that network, it will detect an error and will update its routing table and, by propagation, the other networks' tables.

In our example above, router D will have the following information:

Network	Gateway	HOP
192.0.1.0	192.0.4.1	1
192.0.2.0	192.0.3.2	1
192.0.3.0		0
192.0.4.0		0
192.1.0.0	192.0.3.2	2
10.1.0.0	192.0.4.3	1

Before

Network	Gateway	HOP
192.0.1.0	192.0.4.1	1
192.0.2.0	192.0.4.1	2
192.0.3.0		0
192.0.4.0		0
192.1.0.0	192.0.4.1	3
10.1.0.0	192.0.4.3	1

After

The creation of these routing tables is done by propagation from the other networks' tables (by incrementing the distances), via exchanges of RIP frames.

For example, let us image the initialisation of A's RIP table. We consider that A initiates the propagation.

- Naturally, A knows the 2 networks to which it is connected. It therefore sends the information that makes up its routing table in "BROADCAST" mode to all the machines in the 2 networks to which it is connected (only the routers will retain this information stamped RIP).

- B receives A's table, it attempts to update it:
  - For network 192.0.1.0, B already knows it and knows it is connected to it.
  - However for network 192.0.2.0, as B does not know it, it therefore enters A's address as "Next HOP".
- B uses its own knowledge to complete it:
  - Addition of 192.0.4.0 in the table
- B waits for a random short time and re-distributes its table (as A)
- C receives A's table, it attempts to update it:
  - For network 192.0.2.0, C already knows it and knows it is connected to it.
  - However for network 192.0.1.0, as C does not know it, it therefore enters A's address as "Next HOP"..
- C uses its own knowledge to complete it:
  - Addition of 192.0.3.0 in the table.
- C waits for a random short time and re-distributes its table (as A)

I am ignoring E's router on purpose because it does not add anything to the case we are examining. The random time is used in the case where network 192.0.2.0 has to avoid a collision between the return from C's table and E's table.

For A, which is going to receive, depending on the random values, the frame from B or C first of all, it is therefore going to update its table by including the 2 networks (192.0.3.0 from C and 192.0.4.0 from B). The respective tables of B and C continue to propagate themselves.

Router D, like A, will receive at the same time as A the information from B and C, it will therefore update its table as well and also propagate it after a short time.

At this point the 4 networks 192.0 X 0 are all known, and to better understand the path taken, we are going to summarise these results in the form of a table.

Step	Router A		Router B		Router C		Router D	
	Network address	HOP	Network address	HOP	Network address	HOP	Network address	HOP
0	192.0.1.0	0	192.0.1.0	0	192.0.2.0	0	192.0.3.0	0
	192.0.2.0		192.0.4.0		192.0.3.0		192.0.4.0	
A propagates its RIP table								
1			192.0.2.0	1	192.0.1.0	1		
B propagates its RIP table								
2	192.0.4.0	1					192.0.1.0	2
C propagates its RIP table								
3	192.0.3.0	2					192.0.2.0	1
We then have the following tables								
After step 3	192.0.1.0	0	192.0.1.0	0	192.0.1.0	1	192.0.1.0	2
	192.0.2.0	0	192.0.2.0	1	192.0.2.0	0	192.0.2.0	1
	192.0.3.0	2			192.0.3.0	0	192.0.3.0	0
	192.0.4.0	1	192.0.4.0	0			192.0.4.0	0
D now propagates its RIP table								
4			192.0.3.0	2	192.0.4.0	2		
We then have the following tables								
After step 4	192.0.1.0	0	192.0.1.0	0	192.0.1.0	1	192.0.1.0	2
	192.0.2.0	0	192.0.2.0	1	192.0.2.0	0	192.0.2.0	1
	192.0.3.0	2	192.0.3.0	2	192.0.3.0	0	192.0.3.0	0
	192.0.4.0	1	192.0.4.0	0	192.0.4.0	2	192.0.4.0	0
A, B and C are going to propagate their tables, but all the networks are already known, and the paths are already the shortest, so they will therefore not be followed by re-propagations.								

As the distances have to be minimised, after the third step, all of the network's elements know the shortest path to reach their target. If we add the routers E and F, propagation will take a bit longer but well lead to the same results.

6.7.7.3. The RIP frame

The RIP frame is encapsulated in an IP frame. It is made up of a variable number of fields. However, the RIP frame is made up of at least 24 bytes. The next frame is in bytes.

1	1	2	2	2	4	8	4
C	V	ZERO	AFI	ZERO	ADDRESS	ZERO	ETRIC

The ZERO fields are "empty" fields (filled with zeros)

C Command	Defines the message's propagation direction. This may be a request (a router asks another one to propagate its table) or a "reply" which is either a regular update or an extraordinary update (case where a link in the network is faulty).
V Version	Gives the version number of the RIP protocol used
AFI Address Family Identifier	Makes it possible to use RIP with the network protocols other than IP. We will limit ourselves to the IP protocol, so this field contains the hexadecimal word "0002".
ADDRESS	IP address of the accessible network
METRIC	Distance to the target network in HOPs

An RIP frame may contain up to 25 occurrences of the ADDRESS and METRIC fields, thus making it possible (for each frame) to give the position of 25 routers.

The METRIC field is used to give the number of HOPs between the source router and its target. The number of HOPs is limited to 15 (and cannot be less than 1), however, it is possible to have the value 16 in the METRIC field and this means that the network is inaccessible.

As for the IP address of the gateway to be contacted for the routing, it is in the IP frame encapsulating the RIP frame.

6.7.8. The MODBUS protocol

The MODBUS protocol consists of the definition of exchange frames.

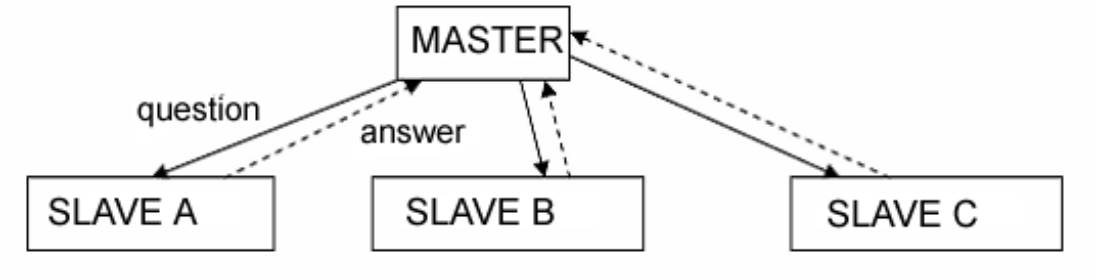


Figure 161: MODBUS protocol

The master sends a **question** and waits for an **answer**.

Two slaves cannot talk with each other.

The master-slave dialogue can be outlined in the form of a succession of point-to-point links.

6.7.8.1. Principle of MODBUS exchanges

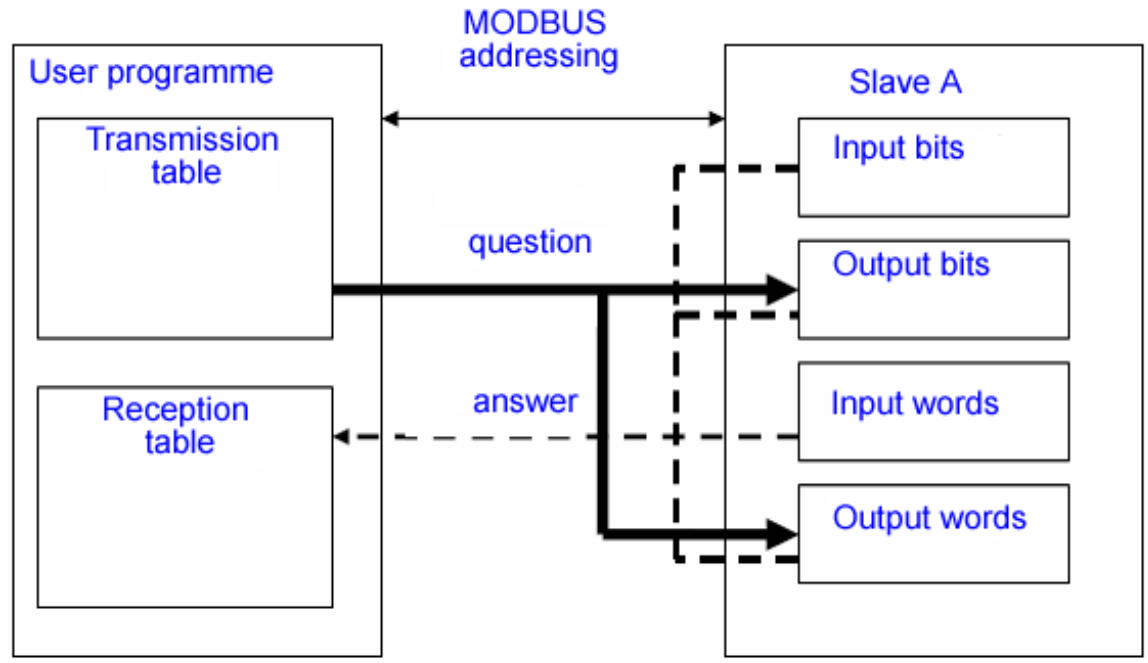


Figure 162 : Principle of data exchanges using MODBUS

6.7.8.2. Addressing

The bus's subscribers are identified by addresses assigned by the user.

Each subscriber's address is independent from its physical location.

The addresses go from 1 to 64 and do not obligatorily have to be assigned in sequence.

Two subscribers cannot have the same address.

6.7.8.3. Exchange from master to one slave

The master interrogates a slave with a unique number on the network and waits for the answer from that slave:

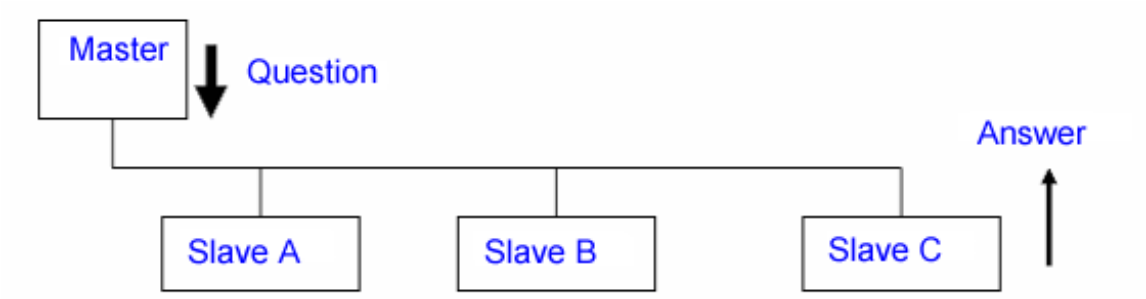


Figure 163 : Exchange for Master to one slave (MODBUS)

6.7.8.4. Exchange from master to all the slaves

The master sends a message to all the slaves in the network, they execute the order contained in the message without sending an answer.

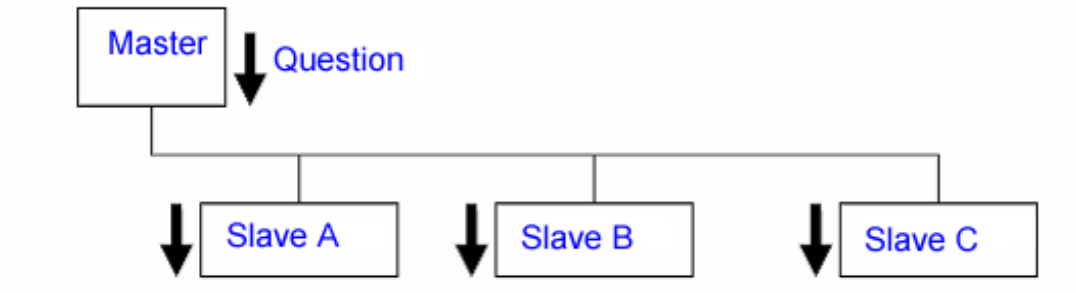


Figure 164 : Exchange from master to all the slaves (MODBUS)

6.7.8.5. Question/answer exchange frame

The question

It contains a function code indicating to the slave being addressed what type of action is requested.

The data contain additional information that the slave needs to execute that function.

The control bytes field allows the slave to check the integrity of the content of the question.

Slave No.	Function code	Specific information concerning the request	Control word
1 byte	1 byte	n bytes	2 bytes

The answer

Slave No.	Function code	Data received	Control word
1 byte	1 byte	n bytes	2 bytes

If an error appears, the function code is modified to indicate that the answer is incorrect.

The data then contain a code (exception code) making it possible to know the type of error.

The control field allows the master to confirm that the message is valid.

Slave No.	Function code	Exception code	Control word
1 byte	1 byte	1 byte	2 bytes

6.7.8.6. General frame format

Two types of coding can be used to communicate on a Modbus network.

All the equipment present in the network must be configured according to the same type.

ASCII type: each byte making up a frame is coded with 2 ASCII characters (2 times 8 bits).

START	Address	Function	Data	LRC	END
1 character	2 characters	2 characters	n characters	2 characters	2 "CR LF" characters

LRC : this is the sum in hexadecimal modulo 256 of the content of the frame outside the delimiters, complemented to 2 and sent in ASCII.

RTU type (Remote Terminal Unit): each byte making up a frame is coded on 2 hexadecimal characters (2 times 4 bits).

START	Address	Function	Data	CRC	END
Silence	1 byte	1 byte	n bytes	2 bytes	Silence

The maximum size of the data is 256 bytes.

ASCII mode makes it possible to have intervals of more than one second between the different characters without generating errors, whereas RTU mode enables a higher rate for the same transmission speed.

All of the information contained in the message is expressed in hexadecimal.

The master addresses the slave whose address is given in the field provided for that purpose.

The function code indicates to the slave what type of action is to be performed. For example: reading of a register, function code $(03)_{HEX}$, writing in a register, function code $(10)_{HEX}$.

The data field is coded on n words in hexadecimal from 00 to FF, or on n bytes.

Depending on the function code, the data field contains various items of additional information allowing the slave to decode the message (see example below).

In the case of RTU mode, the CRC error check field (Cyclical Redundancy Check) contains a value coded on 16 bits.

Note: the parity check may, in certain cases, be deleted because other exchange checks are implemented (case of the CRC control also called Checksum)

The slave returns its answer; it places its own address in the address field so that the master can identify it.

It then uses the function field to indicate whether the answer contains an error. For a normal answer, the slave uses the same function code as that of the message sent by the

master, otherwise it returns an error code corresponding to the original code with its MSB at 1.

The data field contains various items of information depending on the function code.

The error check field contains on a value coded on 16 bits. This value is the result of a CRC (Cyclical Redundancy Check) calculated from a message.

6.7.8.7. Transmission medium

Each byte making up a message is sent in RTU mode in the following way:

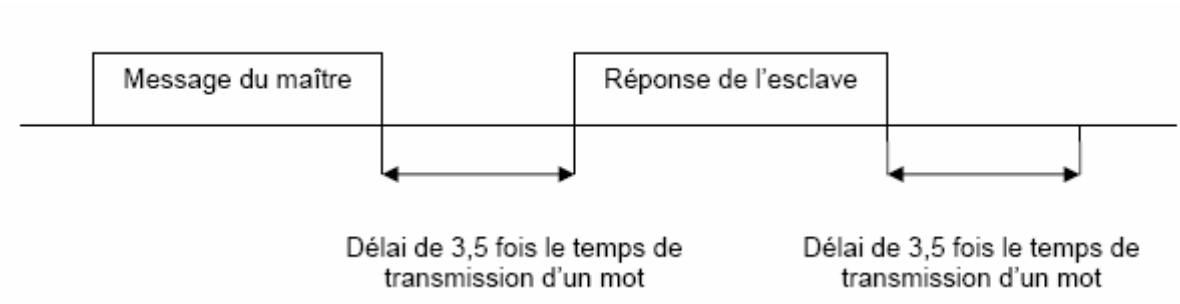
Without a parity check

Start	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Stop

With a parity check

Start	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Stop

There must be a silence before and after each message equivalent to 3.5 times the transmission time for a word.



The whole message must be transmitted continuously. If there is a silence of more than 1.5 times the transmission time for one word during the transmission, the destination of the message will consider that the next item of information it receives will be the address of the start of a new message.

The MODBUS protocol only defines the structure of the messages and their exchange mode.

We can use any RS 232, RS 422 or RS 485 transmission medium, but the RS 485 link is the most widely used because it authorises "multipoint" mode.

Example of an exchange between master and slave

Frame sent by the master: 04 03 00 02 0001 25 CA

- Slave address: 04
- Function code 03 = reading of a register
- No. of the start of reading register: MSB : 00 and LSB : 02
- Number of the reading register: MSB : 00 and LSB : 01
- CRC : 25 CA

Answer from the slave with error : 04 83 02 01 31

- Slave address: 04
- Function code: reading with MSB = 1 : 83
- Error code (register No.) : 02
- CRC: 01 31

Answer from the slave without error: 04 03 02 02 58 B8 DE

- Slave address: 04
- Function code: register reading: 03
- Number of data bytes: 02
- Data in register 0002 : MSB 02 and LSB : 58
- CRC : B8 DE

6.7.9. The PROFIBUS protocol

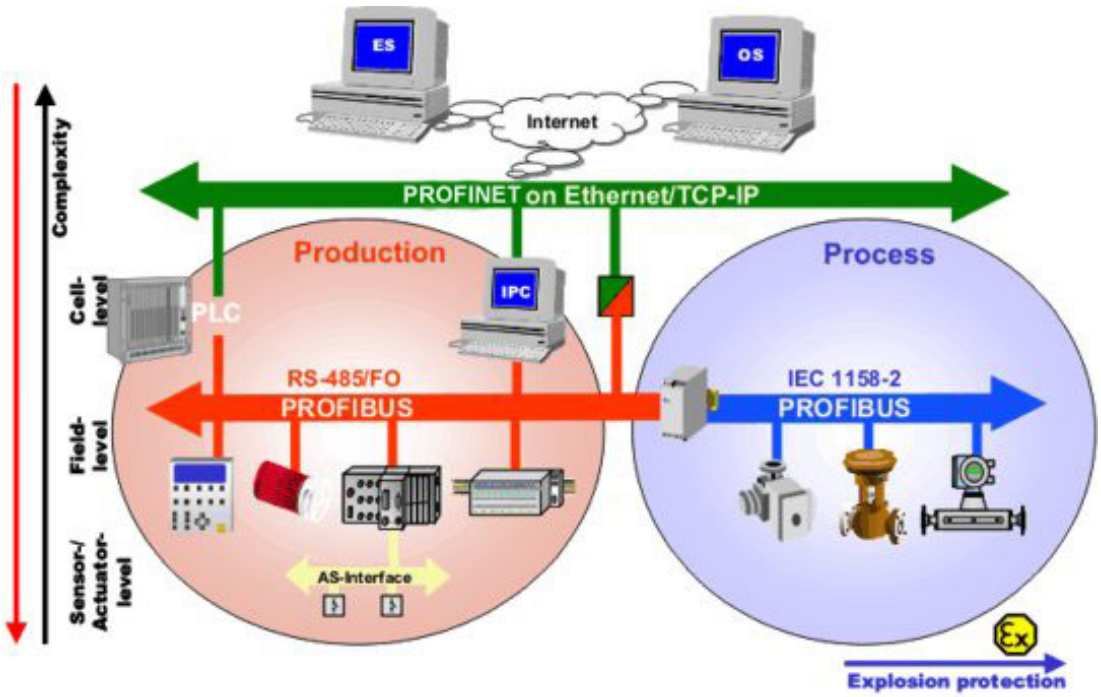


Figure 165 : Architecture of a communication in PROFIBUS

This protocol is frequently used these days because it allows us to make measuring instruments and actuators communicate with PLCs on the same bus.

Developed by SIEMENS, the Profibus DP protocol (decentralised peripheral) is designed to recover the values of remote input and output variables.

The dialogue is based on the "Master-Slave" principle with passing of a token on the bus, which enables each master station (active) in turn to interrogate slave stations (passive) or to dialogue with other master stations.

The maximum transmission flow on the RS485 bus can be as high as 12 Mbps today.

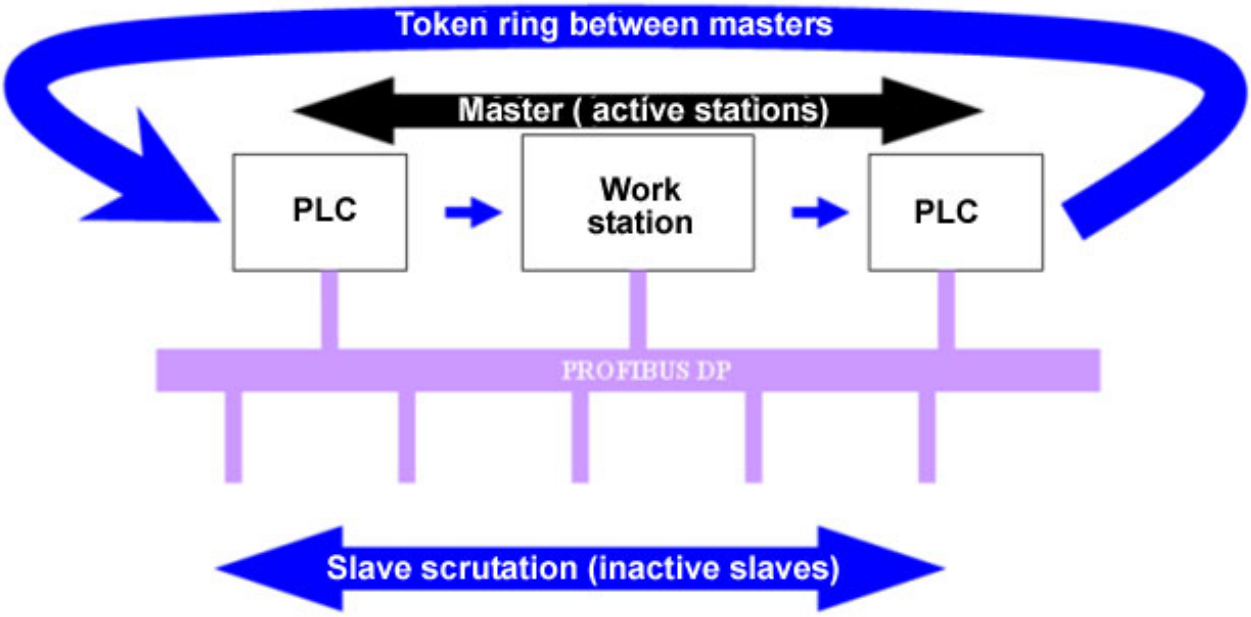


Figure 166 : The Profibus principle

One of the best known is the **PROFIBUS DP**:

6.7.9.1. Profibus DP

The PROFIBUS DP (**D**ecentralized **P**eripheral) field bus is derived from the European standard EN 50170. It was specially designed for communication between Control/Command systems and the distributed Input/output peripherals.

It makes it possible to exchange up to 244 bytes with 127 stations at a speed comprised between 9.6Kbit/s and 12 Mbit/s.

The corresponding communication functions are defined by the basic DP functions (version DP-V0).

Other versions have been developed (**DP-V1** and **DP-V2**) to meet specific requirements with very precise characteristics:

- **DP-V0** ensures the basic DP functions, including the exchange of cyclic data, as well as station, module and channel diagnosis.
- **DP-V1** adds the process-oriented services to that, particularly the acyclic communication designed for parameter setting, and the implementation, visualisation and processing of alarms on intelligent field devices. This enables on-line access to the stations, by means of engineering tools.
- **DP-V2** integrates other improvements mainly designed for drive applications such as isochronal communication and direct exchanges between slaves (Data Exchange Broadcast).

There are three types of equipment for Profibus DP networks:

- DP slave
- Class 1 DP master (DPM1)
- Class 2 DP master (DPM2)

DP slave

This is a peripheral item of equipment (I/O block, variator, HMI, valve...) which, at its input, collects information and, at its output, returns the information to the controller.

Class 1 DP master (DPM1)

This is a cell controller (API, PC, VME calculator) that exchanges information with the remote DP slaves in a parameter-configured message cycle.

Class 2 DP master (DPM2)

This is a programming, configuration or control device, that is used to configure the DP network when it is put into service, and then to operate or monitor it.

Extended communication functions between DPM1 and the DP slaves

Acyclic communication between DPM1 master and DP slaves requires an access point to the SAP 51 supplementary service.

In a service sequence, DPM1 established an MSAC_C1 link with the slave. Once MSAC_C1 has been established, DPM1 can make the cyclic data transmission via MSCY_C1, and acyclic data transmission via MSAC_C1.

For reading and writing, there are two functions (DDLM_Read and DDLM_Write) that make it possible to read or write access any of the slave's data blocks, using the layer 2 SRD service. After transmission of a read/write request, the master interrogates the slave with SRD telegrams until it receives the corresponding DLM_Read/Write reply.

Data block addressing assumes that the DP slaves are of modular design or can be structured in logic functional modules. Addressing relies on identifiers (input or output, data type...) which make up the slave's configuration, it is also subject to a check by DPM1 when the network is started up.

With this model all the read or write validated data blocs are also considered as belonging to the modules.

These blocks can be addressed by slot number and by index. The slot number identifies the module (the number 0 is reserved for the equipment itself) and the index number identifies the data blocks attached to the module.

The read/write request's length field makes it possible to read or write portions of the data block. If access to the block is successful, the DP slave returns a positive reply, otherwise it gives a negative reply indicating the class of problem.

The Profibus-DP basic functions also enable the DP slaves to spontaneously repatriate the events to the master, accompanied by a diagnosis. The DDLM_Alarm function ensures this flow control and makes it possible to explicitly acknowledge the alarms received from the DP slave.

Extended communication functions between DPM2 and DP slaves

The DP extended functions allow one or more diagnosis or control (DPM2) tools to perform acyclic read/write services on any of the DP slave's data blocks.

The DDLM_Initiate service establishes an MSAC_C2 link in connected mode for the transmission of user data.

Transmission is then carried out by means of the DLM_Read and DLM_Write services.

The master can then insert Idle_PDU monitoring telegrams during the transmission pauses. MSAC_C2 thus has an automatic time check of the connections. The monitoring interval is indicated by DDLM_Initiate when the link is established. If the link checker detects a fault, that link is automatically released on the master side and on the slave side.

6.7.10. The HART protocol

6.7.10.1. What is HART?

HART Communication is a bi-directional industrial field communication protocol used to communicate between intelligent field instruments and host systems. HART is the global standard for smart process instrumentation and the majority of smart field devices installed in plants worldwide are HART-enabled.

The global installed base of HART-enabled devices is the largest of all communication protocols at more than 20 million. HART technology is easy to use and very reliable.

There are several reasons to have a host communicate with a field instrument. These include:

- Device Configuration or re-configuration
- Device Diagnostics
- Device Troubleshooting
- Reading the values of additional measurements provided by the device
- Device Health and Status
- And much more!

A host system can be a Distributed Control System, PLC, Asset Management System, Safety System or a handheld device.

HART is a master-slave field communications protocol developed in the late 1980's to facilitate communication with Smart field devices. HART stands for Highway Addressable Remote Transducer. The HART protocol makes use of the Bell 202 Frequency Shift Keying (FSK) standard to superimpose digital communication signals at a low level on top of the 4-20mA.

This enables two-way field communication to take place and makes it possible for additional information beyond just the normal process variable to be communicated to/from a smart field instrument. The HART protocol communicates at 1200 bps without interrupting the 4-20mA signal and allows a host application (master) to get two or more digital updates per second from a field device. As the digital FSK signal is phase continuous, there is no interference with the 4-20mA signal.

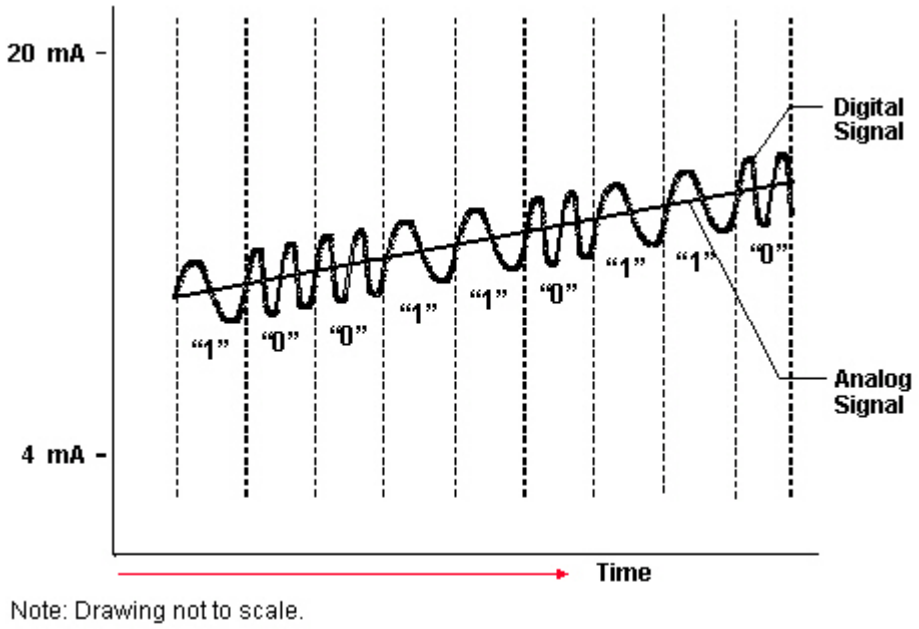


Figure 167: HART protocol

HART is a master/slave protocol which means that a field (slave) device only speaks when spoken to by a master. The HART protocol can be used in various modes for communicating information to/from smart field instruments and central control or monitoring systems.

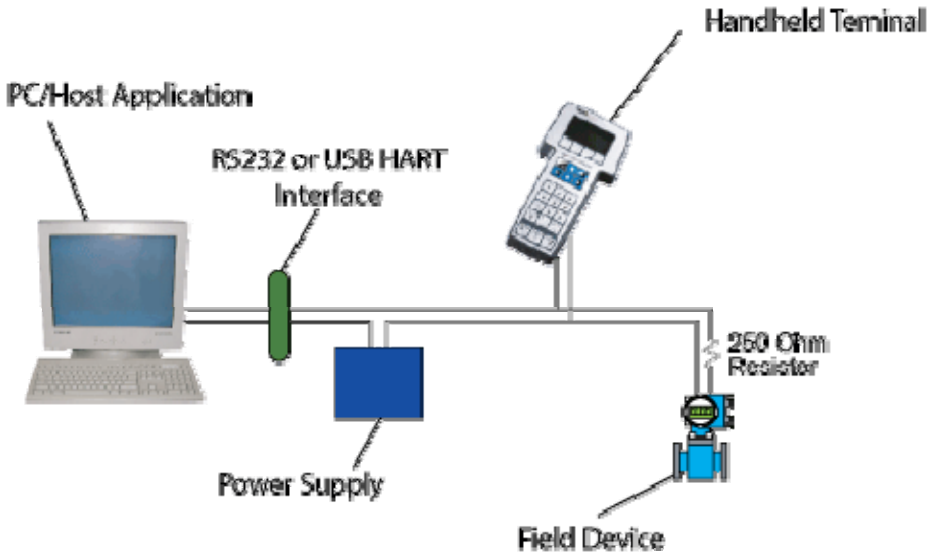


Figure 168 : Application of a communication in HART

HART provides for up to two masters (primary and secondary). This allows secondary masters such as handheld communicators to be used without interfering with communications to/from the primary master, i.e. control/monitoring system.

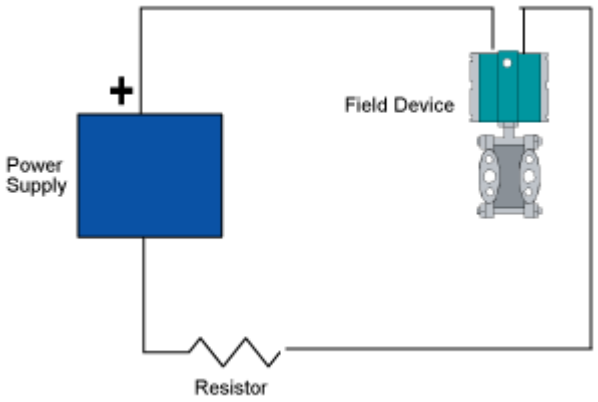


Figure 169 : Configuration as a point-to-point network

The HART protocol permits all digital communication with field devices in either point-to-point or multidrop network configurations.

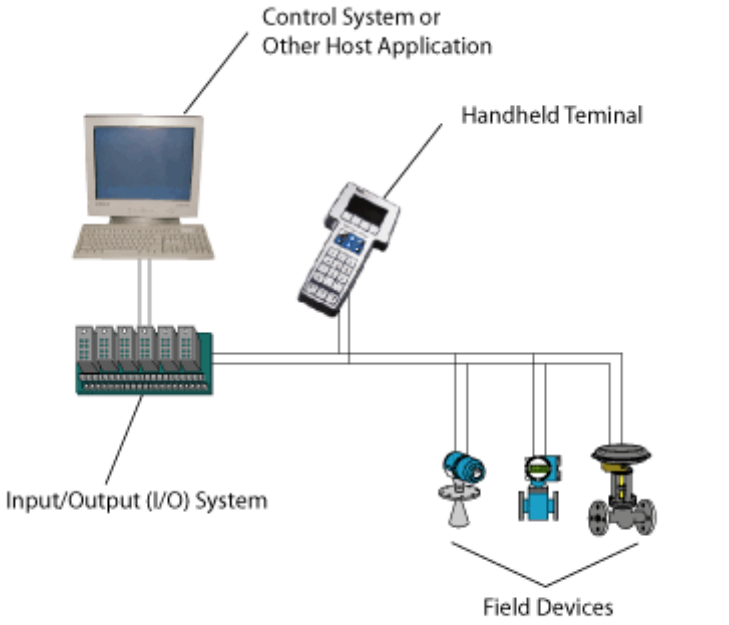


Figure 170 : Configuration as a multidrop network

6.7.10.2. How does HART function?

HART communication occurs between two HART-enabled devices, typically a field device and a control or monitoring system. Communication occurs using standard instrumentation grade wire and using standard wiring and termination practices.

HART provides two simultaneous communication channels: the 4-20mA Analog signal and a digital signal. The 4-20mA signal communicates the primary measured value (in the case of a field instrument) using the 4-20mA current loop - the fastest and most reliable industry

standard. Additional device information is communicated using a digital signal that is superimposed on the Analog signal. The digital signal contains information from the device including device status, diagnostics, additional measured or calculated values, etc.

Together, the two communication channels provide a complete field communication solution that is easy to use and configure, is low-cost and is very robust.

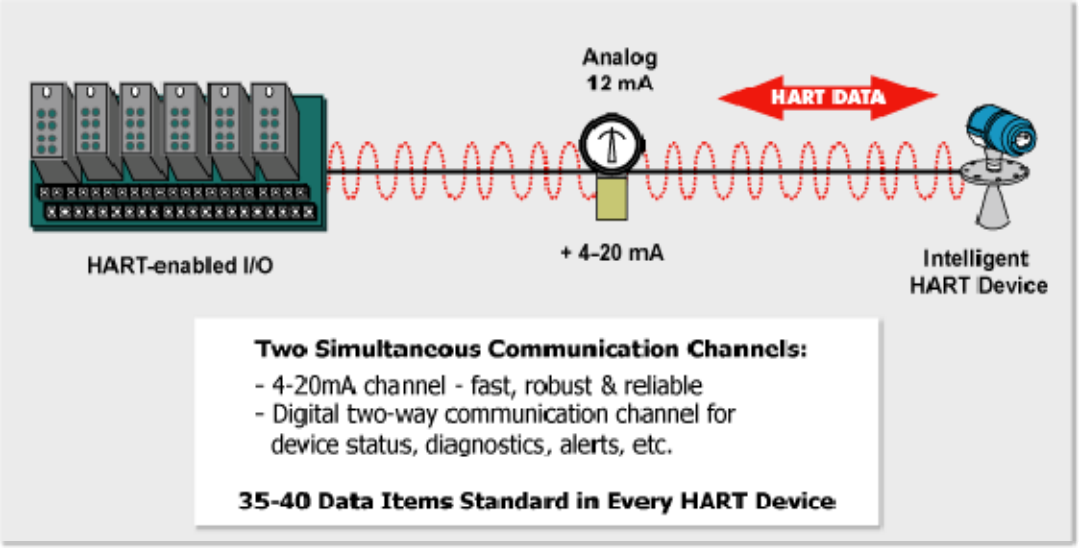


Figure 171 : HART protocol operating mode

HART Data—Overview

- Digital data: 35-40 valuable data items standard in every HART device
- Device identification: device tag, supplier, device type and revision, device serial number
- Calibration data: upper and lower range values, upper and lower sensor limits, PV damping, last calibration date
- Process variables: primary variable plus secondary measurements and multivariable parameters
- Status/diagnostic alerts: device malfunction, configuration change, power fail restart, loop current fixed or saturated, primary or secondary variable out of limits, communication error, plus more

HART Commands

HART devices communicate via commands. The command set includes three classes: universal, common practice, and device specific.

Universal	All devices using the HART Protocol must recognize and support the universal commands. Universal commands provide access to information useful in normal operations. For example: read primary variable and units, read manufacturer and device type, read current output and percent of range and read sensor serial number and limits.
Common Practice	Common practice commands provide functions implemented by many, but not necessarily all, HART communication devices. The HART specifications recommend devices to support these commands when applicable. Examples of common practice commands are: read a selection of up to four dynamic variables, write damping time constant, write transmitter range, set fixed output current and perform self-test.
Device Specific	Device specific commands represent functions that are unique to each field device. These commands access setup and calibration information, as well as information about the construction of the device. Information on device-specific commands is available from device manufacturers or in the Field Device Specification document. Examples of device specific commands: read or write sensor type, start, stop or clear totalizer, read or write alarm relay set point, etc.

6.7.10.3. The HART specifications

The HART Field Communications Protocol Specifications is a set of documents that define the HART Field Communications Protocol.

Designed to compliment traditional 4-20mA analog signalling, the HART Protocol supports two way digital communications for process measurement and control devices.

Applications include remote process variable interrogation, cyclical access to process data, parameter setting and diagnostics. This document defines the specification documents that comprise the HART Field Communications Protocol. Specification of the HART protocol is based largely on the OSI 7-Layer Communication Model:

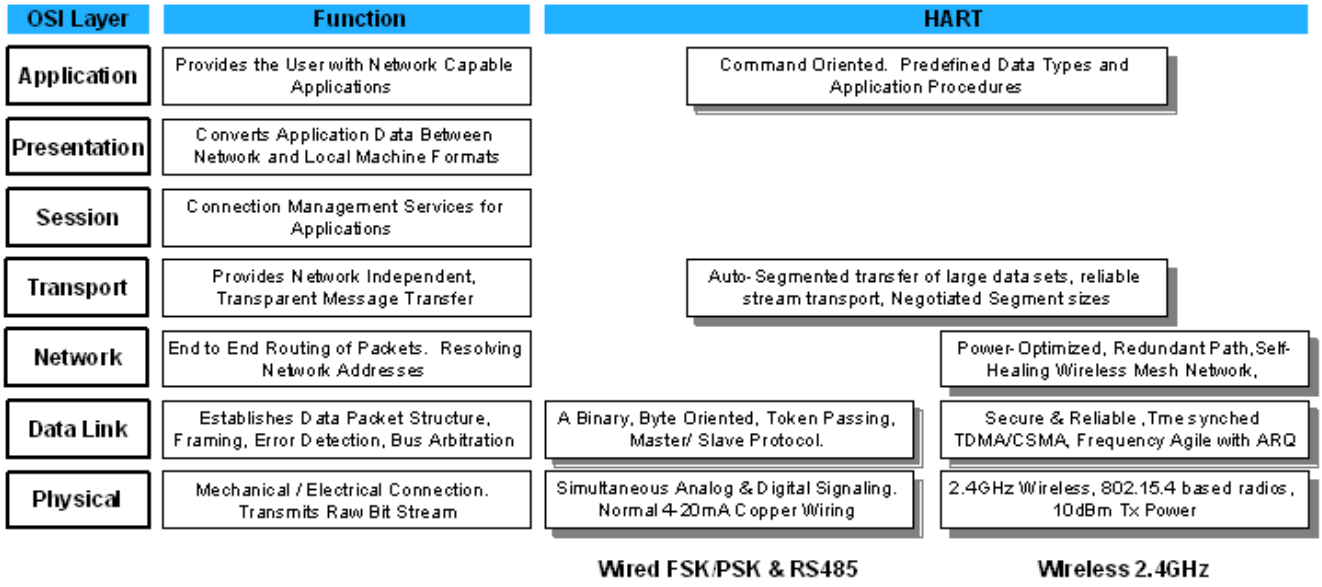


Figure 172: HART specifications

The Application Layer defines the commands, responses, data types and status reporting supported by the Protocol. In addition, there are certain conventions in HART (for example how to trim the loop current) that are also considered part of the Application Layer.

While the Command Summary, Common Tables and Command Response Code Specifications all establish mandatory Application Layer practices (e.g. data types, common definitions of data items, and procedures), the Universal Commands specify the minimum Application Layer content of all HART compatible devices.

6.7.10.4. What is contained in HART data?

There are several types of data or information that can be communicated from a HART-enabled device.

This includes:

- Device data
- Supplier data
- Measurement data
- Calibration data

When this data is integrated with control, asset management or safety systems, you are able to improve plant operations, lower cost and increase plant availability.

The following is a summary of data items available for communication between HART Devices and a Host.

Process Variable Values

- Primary Process Variable (analog) - 4-20 ma current signal continuously transmitted to host
- Primary Process Variable (digital) - Digital value in engineering units, IEEE floating point, up to 24 bit resolution
- Percent Range - Primary Process Variable expressed as percent of calibrated range
- Loop Current - Loop current value in milliamps
- Secondary Process Variable 1 - Digital value in engineering units available from multivariable devices
- Secondary Process Variable 2 - Digital value in engineering units available from multivariable devices
- Secondary Process Variable 3 - Digital value in engineering units available from multivariable devices

Commands From Host to Device

- Set Primary Variable Units
- Set Upper Range
- Set Lower Range

- Set Damping Value
- Set Message
- Set Tag
- Set Date
- Set Descriptor
- Perform Loop Test - Force loop current to specific value
- Initiate Self Test - Start device self test
- Get More Status Available Information

Status and Diagnostic Alerts

- Device Malfunction - Indicates device self-diagnostic has detected a problem in device operation
- Configuration Changed - Indicates device configuration has been changed
- Cold Start - Indicates device has gone through power cycle
- More Status Available- Indicates additional devices status data available
- Primary Variable Analog Output Fixed - Indicates device in fixed current mode
- Primary Variable Analog Output Saturated - Indicates 4-20mA signal is saturated
- Secondary Variable Out of Limits - Indicates secondary variable value outside the sensor limits
- Primary Variable Out of Limits - Indicates primary variable value outside the sensor limits

Device Identification

- Instrument Tag - User defined, up to 8 characters
- Descriptor - User defined, up to 16 characters
- Manufacturer Name (Code) - Code established by HCF and set by manufacturer

- Device Type and Revision - Set by manufacturer
- Device Serial Number - Set by manufacturer
- Sensor Serial Number - Set by manufacturer

Calibration Information for 4-20mA Transmission of Primary Process Variable

- Date - Date of last calibration, set by user
- Upper Range Value - Primary Variable Value in engineering units for 20mA point, set by user
- Lower Range Value - Primary Variable Value in engineering units for 4mA point, set by user
- Upper Sensor Limit - Set by manufacturer
- Lower Sensor Limit - Set by manufacturer
- Sensor Minimum Span - Set by manufacturer
- PV Damping - Primary Process Variable Damping Factor, set by user
- Message - Scratch pad message area (32 characters), set by user
- Loop Current Transfer Function - Relationship between Primary Variable digital value and 4-20mA current signal
- Loop Current Alarm Action - Loop current action on device failure (upscale/downscale)
- Write Protect Status - Device write-protect indicator

7. THE WIRELESS NETWORK

7.1. INTRODUCTION

7.1.1. What is a wireless network?

A ***wireless network*** is, as suggested by its name, a network in which at least one of the two terminals (laptop computer, PDA, etc.) can communicate without a wire connection.

Thanks to wireless networks, a user can stay connected while moving around in a more or less extensive geometrical perimeter, and that is why we sometimes here the term "mobility".

Remark concerning the term "wireless" networks: you will often hear people use the term Wi-Fi.

Wireless networks are based on a link using radio-electric waves (radio and infrared) instead of the usual wires. There are several technologies differentiated on the one hand by the transmission frequency used and the speed and range of the transmissions.

Wireless networks make it possible to very easily connect pieces of equipment that are ten meters to some kilometres apart. Furthermore, the installation of such networks does not require the very costly construction of infrastructure as is the case with wire networks (digging of trenches to route the cables, installing wiring, cable troughs and connectors in buildings), which explains the rapid development of this type of technology.

However, there is the problem of the regulations relative to radio-electric transmissions. Indeed, radio-electric transmissions are used for a large number of applications (military, scientific, amateur, ...), but are sensitive to interferences, which is why regulations are required in each country in order to define the frequency bands and power ratings at which it is possible to transmit for each category of use.

Furthermore radio waves are difficult to confine in a restricted geographical surface, it is therefore easy for an eavesdropper to listen in on the network if the information circulates in unencrypted form (this is the default case). It is therefore necessary to put in place the necessary measures to ensure the confidentiality of the data transmitted on wireless networks.

7.1.2. The categories of wireless networks

We usually make a distinction between several categories of wireless network, according to the geographical perimeter offering connectivity (called *coverage zone*):

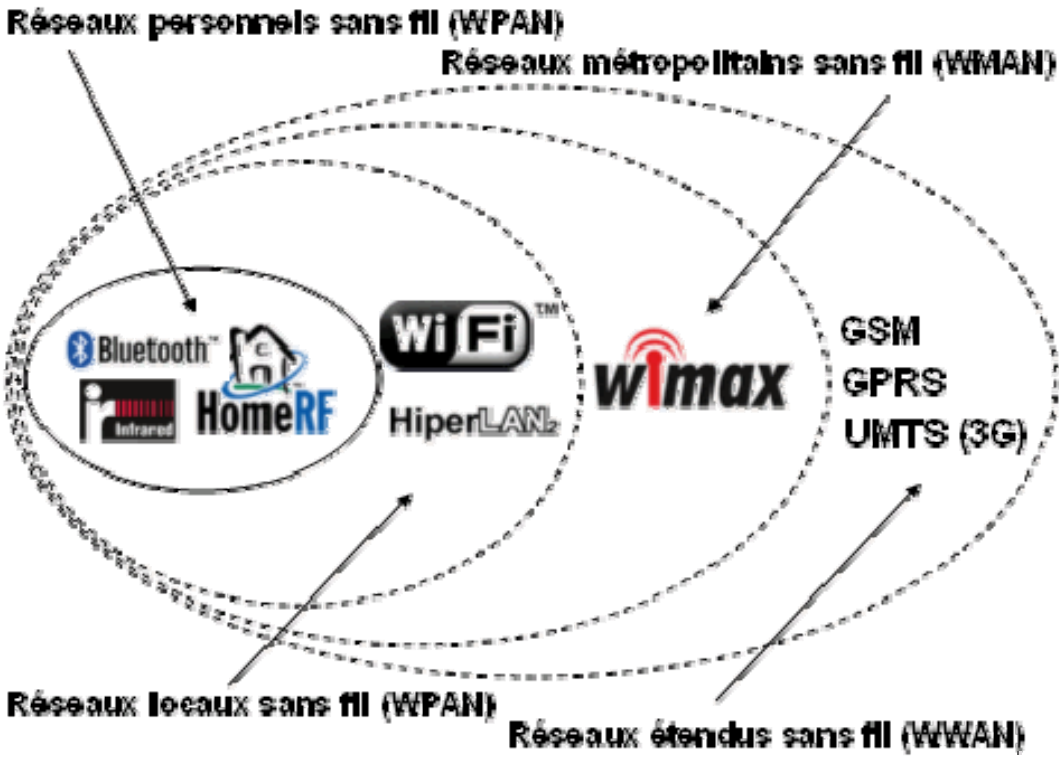


Figure 173 : The various categories of wireless network

7.2. Wi-Fi

7.2.1. Presentation

IEEE 802.11 (*ISO/IEC 8802-11*) is an international standard describing the characteristics of a wireless local area network (*WLAN*). The name **Wi-Fi** (contraction of *Wireless Fidelity*, sometimes incorrectly written *WiFi*) initially corresponded to the name given to the certification issued by the Wi-Fi Alliance, formerly WECA (*Wireless Ethernet Compatibility Alliance*), the organisation responsible for maintaining interoperability between the hardware that meets the 802.11 standard. Through a misuse of language (and for marketing purposes) the name of the standard is now confused with the name of the certification. So a Wi-Fi network is in reality a network that meets the 802.11 standard. The hardware certified by the Wi-Fi Alliance can use the following logo:

Thanks to Wi-Fi, it is possible to create high-speed wireless local area networks provided that the ***computer*** to be connected is not too far from the access point. In practice, Wi-Fi makes it possible to connect ***laptop computers***, ***desktop computers***, ***Personal Digital Assistants (PDA)*** or any type of peripheral to a high-speed link (11 Mbps or higher) over a radius of some tens of meters indoors (generally between twenty and fifty meters) and several hundreds of meters in an open environment.

So, operators are beginning to irrigate zones with a high concentration of users (stations, airports, hotels, trains, etc.) with wireless networks. These access zones are called "**hot spots**".

The 802.11 standard focuses on defining the low layers of the ***OSI model*** for a wireless link using electromagnetic waves, that is to say:

- the ***physical layer*** (sometimes noted *PHY layer*), proposing three types of information coding.
- the ***data link layer***, made up of two sub-layers: the **Logical Link Control**, or **LLC** and the **Media Access Control**, or **MAC.**

The physical layer defines the modulation of the radio-electrical waves and the data transmission signalling characteristics, whereas the *data link* layer defines the interface between the machine's bus and the physical layer, in particular an access method close to that used in the ***Ethernet*** standard and the rules relative to communication between the various stations. The 802.11 standard in fact proposes three physical layers, defining alternative transmission modes:

<table>
<tr><td rowspan="2">Data link layer
(MAC)</td><td colspan="3">802.2</td></tr>
<tr><td colspan="3">802.11</td></tr>
<tr><td>Physical layer
(PHY)</td><td>DSSS</td><td>FHSS</td><td>Infrared</td></tr>
</table>

It is possible to use any high-level protocol on a Wi-Fi network in the same as it is possible on an ***Ethernet*** network.

7.2.2. The different Wi-Fi standards

The *IEEE 802.11* standard is in fact the initial standard offering speeds of 1 or 2 Mbps. Revisions have been applied to the original standard to optimise the speed (that is the case of the 802.11a, 802.11b and 802.11g standards, called the physical 802.11 standards) or to clarify the requirements and ensure greater security or better interoperability. Here is a table presenting the various revisions of the 802.11 standard and their meaning:

Standard	Name	Description
802.11a	Wifi5	The 802.11a standard (known as *WiFi 5*) makes it possible to obtain a high speed (54 Mbps theoretically, 30 Mbps in reality). The 802.11a standard specifies 8 radio channels in the 5 GHz frequency band.
802.11b	Wifi	The 802.11b standard is currently the most widely used standard. It proposes a theoretical speed of 11 Mbps (6 Mbps in reality) with a range of up to 300 meters in an open environment. The frequency range used is 2.4 GHz band, with 3 radio channels available.
802.11c	802.11 bridge to 802.1d	The 802.11c standard is of no interest for the general public. It is simply a modification of the 802.1d standard making it possible to establish a bridge with the 802.11 frames (*data link* level).
802.11d	Internationalisation	The 802.11d standard is a supplement to the 802.11 standard whose goal is to enable the international utilisation of 802.11 local area networks. It consists of enabling different items of equipment to exchange information on the frequency bands and power levels authorised in the hardware's country of origin.

Standard	Name	Description
802.11e	Improved quality of service	The 802.11e standard aims to provide possibilities in the area of the quality of service at the level of the *data link* layer. The goal of this standard is therefore to define the requirements of the various packets in terms of passband and transmission times in such a way as to allow a better transmission of voice and video in particular.
802.11f	Roaming	The 802.11f standard is a recommendation intended for access point retailers to ensure a better level of product interoperability. This proposes *Inter-Access point roaming protocol* allowing an itinerant user to change access point in a transparent way when travelling, whatever the access point brands present in the network infrastructure. This possibility is called *roaming*)
802.11g		The 802.11g standard offers a high speed (54 Mbps in theory, 30 Mbps in reality) on the 2.4 GHz frequency band. The 802.11g standard has bottom-up compatibility with the 802.11b standard, which means that hardware that complies with the 802.11g standard can operate with 802.11b
802.11h		The *802.11h* standard aims to come close to the 802.11 standard in the European standard (HiperLAN 2, hence the *h* in 802.11h) and conform to the European regulation in the area of frequencies and power saving.
802.11i		The purpose of the *802.11i* standard is to improve transmission security (management and distribution of ciphers, encryption and authentication). This standard is based on the *AES* (*Advanced Encryption Standard*) and proposes an encryption of communications for transmissions using the 802.11a, 802.11b and 802.11g. technologies

Standard	Name	Description
802.11Ir		The *802.11r* standard was drawn up for the utilisation of infrared signals. This standard is now technically outdated.
802.11j		The *802.11j* standard is the Japanese equivalent of the European 802.11h standard.

Table 11: The various WIFI strandards

It is interesting to note the existence of a standard called "*802.11b+*". It is a proprietary standard that proposes improvements in terms of speeds. This standard, however, suffers from shortcomings in terms of interoperability guarantees in that it is not an IEEE standard.

7.2.3. Ranges and speeds

The 802.11a, 802.11b and 802.11g standards, called "*physical standards*", correspond to revisions of the 802.11 standard and propose operating modes making it possible to obtain different speeds according to the range.

Standard	Frequency band	Speed	Range
WiFi a (802.11a)	5 GHz	54 Mbit/s	10 m
WiFi B (802.11b)	2.4 GHz	11 Mbit/s	100 m
WiFi G (802.11b)	2.4 GHz	54 Mbit/s	100 m

Table 12: Wi-Fi ranges and speeds

7.2.3.1. The 802.11a standard

The 802.11a standard makes it possible to obtain a theoretical speed of 54 Mbps, that is five times higher than 802.11b, for a range of about only thirty meters. The 802.11a standard is based on an *Orthogonal Frequency Division Multiplexing* (OFDM) type coding on the 5 GHz frequency band and using 8 non-overlapping channels.
So, 802.11a equipment is not compatible with 802.11b equipment. However, hardware including 802.11a and 802.11b does exist, we then use the term **dual band** hardware.

Theoretical speed (indoors)	Range
54 Mbits/s	10 m
48 Mbits/s	17 m
36 Mbits/s	25 m
24 Mbits/s	30 m
12 Mbits/s	50 m
6 Mbits/s	70 m

Table 13: The 802.11a standard

7.2.3.2. The 802.11b standard

The 802.11b standard makes it possible to obtain a theoretical speed of 11 Mbps, for a range of about fifty meters indoors and up to 200 meters outdoors (and even more with directional antennas).

Theoretical speed	Range (indoors)	Range (outdoors)
11 Mbits/s	50 m	200 m
5.5 Mbits/s	75 m	300 m
2 Mbits/s	100 m	400 m
1 Mbit/s	150 m	500 m

Table 14: The 802.11b standard

7.2.3.3. The 802.11g standard

The 802.11g standard makes it possible to obtain a theoretical speed of 54 Mbps for ranges equivalent to those of the 802.11b standard. Furthermore, insofar as the 802.11g standard uses the 2.4GHz frequency band with OFDM coding, this standard is compatible with 802.11b hardware, with the exception of certain old items of equipment.

Theoretical speed	Range (indoors)	Range (outdoors)
54 Mbits/s	27 m	75 m
48 Mbits/s	29 m	100 m
36 Mbits/s	30 m	120 m
24 Mbit/s	42 m	140 m
18 Mbit/s	55 m	180 m
12 Mbit/s	64 m	250 m
9 Mbit/s	75 m	350 m
6 Mbit/s	90 m	400 m

Table 15: The 802.11g standard

8. LIST OF FIGURES

9. LIST OF TABLES