Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea in Ingegneria dell'Automazione

Tesi di laurea

Development of functional safety applications for Autec products. Study of protocols: CANopen, CANopen Safety, FSOE and ProfiSafe

Candidato:
Giovanni Peserico
Matricola 1179223

Relatore:
Ch.mo Prof. Stefano Vitturi

Anno Accademico 2018-2019

Contents

Chapter 1

Introduction

Modern automation systems are strongly based on communication networks, which are nowadays used at all the hierarchical levels. The introduction of industrial networks started in the 1980s, with the so called “Fieldbuses”, that are communication systems specifically conceived for field level data exchange between controllers and sensors/actuators. They have been followed by Real-time Ethernet networks, at the beginning of the 2000s and, some years later, by the industrial wireless networks. Industrial networks are rather different from traditional communication systems typically used for general purpose applications. Indeed, they rely on protocols specifically designed to ensure real-time and deterministic performances, as well as to provide high reliability, since it is essential to guarantee continuity of the production processes, even in the presence of hard environmental conditions, as it is often the case of industrial applications. For example, network operations may be strongly influenced by the magnetic interference, mechanical and thermal stresses, humidity, dusts, vibrations, etc .
In this scenario, in recent years, industrial networks have become of fundamental importance, not only for the communication between the devices at all automation levels, but also for the implementation of intrinsic safety applications that, traditionally, were based on specific and dedicated hardware systems. This opportunity is particularly appealing since also safety applications can be now integrated in the context of factory automation and, more in general, in Industrial Internet of Things (IIoT) and Industry 4.0 systems. This thesis has the principal goal of developing intrinsic safety applications in distributed real-time industrial systems, mainly based on fieldbuses and RTE networks. To achieve this important objective the first part of this elaborate provides an introduction of the principal protocols, such as CANopen Safety, Fail safe Over Ethercat (FSOE) and Profisafe, used for the safety relevant applications in the automation environment, analysing properties, story, the use of them by industry, benefit and drawback of each protocol. To fully understand their working principle the relative consortia networks, such as CAN, Ethercat and Profibus/Profinet, are briefly introduced, but remembering the fact that these protocols can also be implemented on other networks. Understood the main differences of these protocols, with particular attention to the different way of providing safety, it was searched an environment to develop safety application using these protocols. Codesys was identified as this code environment and in particular Codesys Safety SIL2, which allows to work with Safety protocols and it also provides specific tools for the development of safety application and for a SIL2 certification.
Once analysed and studied all these arguments, it was possible to start an applicative activity which provides the practical fully comprehension of them through the development of safety applications.
In particular, since this elaborated is developed with the collaboration of Autec Srl, the applications are developed for this company products, which were previously analysed.

Chapter 2

Protocol used in automation

2.1 Controller Area Network(CAN)

CAN is an International Standardization Organization (ISO) serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus. The specification calls for high-immunity to electrical interference and the ability to self-diagnose and repair data errors. It was developed by BOSCH in 1986 with some important advantages like:

- fixed time response;
- easy wiring based on twisted pair;
- high-immunity to electrical interference;
- ability to self-diagnose and repair data errors with the possibility to isolate a node which highlighted an error without corrupting the network .

All these benefits lead to a widespread use of CAN also in the automation environment. The ISO 11898 architecture defines the lowest two layers of the seven layer OSI/ISO model: the data-link layer and physical layer. The CAN network has a multi-master serial bus for connecting electronic control units, also referred to as nodes, each having a microprocessor and possibly sensors and actuators. Since CAN is a multi-master communication protocol, every node in the system is equal to every other node and any processor can send a message to any other processor. With serial transmission each bit is sent one at a time onto essentially a single wire bus. Also parallel transmissions are permitted, this transmission would involve each bit having its own wire so that all the bits can be sent simultaneously and would give quicker transmission, at the expense of higher cost. The processor decides what messages it wants to transmit and what received messages mean. Messages are sent onto the bus serially when the bus is free. In the case of simultaneous transmissions, by two or more devices, a collision occurs, which is handled by the bitwise arbitration method, which requires all nodes on the CAN network to be synchronized to sample every bit on the CAN network at the same time. The CAN specifications use the terms "dominant" bits and "recessive" bits where dominant is a logical 0 (actively driven to a voltage by the transmitter) and recessive is a logical 1 (passively returned to a voltage by a resistor). The idle state is represented by the recessive level (Logical 1). If one node transmits a dominant bit and another node transmits a recessive bit then there is a collision and the dominant bit "wins". This means there is no delay to the higher-priority message, and the node transmitting the lower priority message automatically attempts to re-transmit six bit clocks after the end of the dominant message. This makes CAN very suitable as a real-time prioritized communications system. The exact voltages for a logical 0 or 1 depend on the physical layer used, but the basic principle of CAN requires that each node listens to the data on the CAN network including the transmitting node(s) itself (themselves). If a logical 1 is transmitted by all transmitting nodes at the same time, then a logical 1 is seen by all of the nodes, including both the transmitting node(s) and receiving node(s). If a logical 0 is transmitted by all transmitting node(s) at the same time, then a

logical 0 is seen by all nodes. If a logical 0 is being transmitted by one or more nodes, and a logical 1 is being transmitted by one or more nodes, then a logical 0 is seen by all nodes including the node(s) transmitting the logical 1. When a node transmits a logical 1 but sees a logical 0, it realizes that there is a contention and it quits transmitting. By using this process, any node that transmits a logical 1 when another node transmits a logical 0 "drops out", losing in such way the arbitration. A node that loses arbitration re-queues its message for later transmission and the CAN frame bit-stream continues without error until only one node is left transmitting. Since the 11 (or 29 for CAN 2.0B) bit identifier is transmitted by all nodes at the start of the CAN frame, the node with the lowest identifier transmits more zeros at the start of the frame, and that is the node that wins the arbitration or has the highest priority.
When the CAN controller receives a message, the received serial bits are stored bit by bit until an entire message has been received and can be acted on by its microcontroller.
All nodes on the CAN network must operate at the same nominal bit rate, but noise, phase shifts, oscillator tolerance and oscillator drift mean that the actual bit rate may not be the same as the nominal bit rate. Since a separate clock signal is not used, a means of synchronizing the nodes is necessary. Synchronization is important during arbitration since all the nodes in must be able to see the data on the bus at the same time to understand which node "wins" the conflict. Moreover it is also important to ensure that variations in oscillator timing between nodes do not cause errors. Synchronization starts with a hard synchronization on the first recessive to dominant transition after a period of bus idle (the start bit). Resynchronization occurs on every recessive to dominant transition during the frame. The CAN controller expects the transition to occur at a multiple of the nominal bit time. If the transition does not occur at the exact time the controller expects it, the controller adjusts the nominal bit time accordingly. The adjustment is accomplished by dividing each bit into a number of time slices called quanta, and assigning some number of quanta to each of the four segments within the bit (Fig. 2.1):

- the Synchronization Segment, which always is one quantum long, is used for synchronization of the clocks. A bit edge is expected to take place here when the data changes on the bus.
- the Propagation Segment, which is needed to compensate for the delay in the bus lines.
- the Phase Segment, which may be shortened or lengthened if necessary to keep the clocks in sync.

A transition that occurs before or after it is expected causes the controller to calculate the time difference and lengthen phase segment 1 or shorten phase segment 2 by this time. This effectively adjusts the timing of the receiver to the transmitter to synchronize them. This resynchronization process is done continuously at every recessive to dominant transition to ensure the transmitter and receiver stay in sync. Continuously resynchronizing reduces errors induced by noise, and allows a receiving node that was synchronized to a node which lost arbitration to resynchronize to the node which won arbitration.

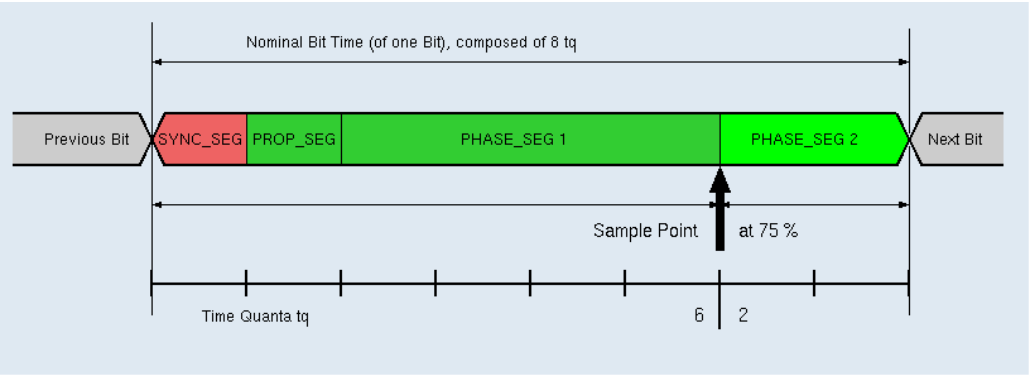


Figure 2.1: Structure of bit timing in CAN

As concerning the message, there is a basic format (Fig. 2.2) composed by:

- 1 bit to denote start of frame(SOF);
- 11 bits to identify the bit used when information is required from another node to determinate the specific node;
- the single remote transmission request(RTR), which must be dominant (0) for data frames and recessive (1) for remote request frames;
- a dominant identifier extension(IDE) bit, which is an extra identification bit that must be dominant (0) for base frame format with 11-bit identifiers;
- a reserved bit for possible use by future standard amendment;
- 4-bit data length code (DLC) which contains the number of bytes of data being transmitted (0–8 bytes).
- Data Field, which are up to 64 bits of application data;
- 16 bits for the cyclic redundancy check (CRC) field;
- Bit for acknowledges (ACK): Every node receiving an accurate message overwrites this recessive bit in the original message with a dominate bit, indicating an error-free message has been sent. Should a receiving node detect an error and leave this bit recessive, it discards the message and the sending node repeats the message after rearbitration. In this way, each node acknowledges the integrity of its data. ACK is 2 bits, one is the acknowledgement bit and the second is a delimiter.
- 7- bit for end-of-frame(EOF) , which marks the end of a CAN frame (message) and disables bitstuffing (when 5 bits of the same logic level occur in succession during normal operation, a bit of the opposite logic level is stuffed into the data), indicating a stuffing error when dominant.
- 7-bit interframe space (IFS), which contains the time required by the controller to move a correctly received frame to its proper position in a message buffer area.

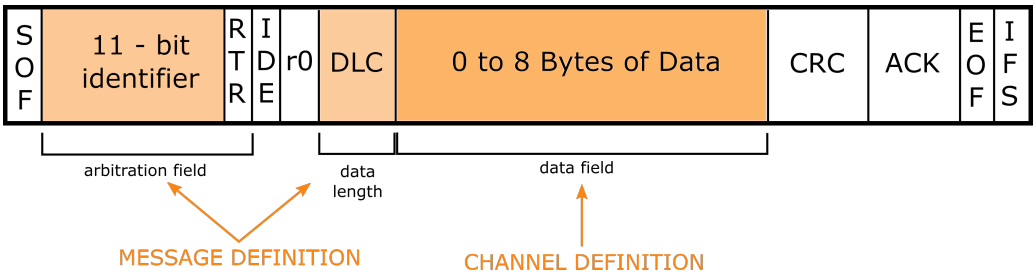


Figure 2.2: Structure of a message in CAN

An extended frame format (Fig. 2.3) is also used in some instances. the Extended CAN message is the same as the Standard message with the addition of:

- The substitute remote request (SRR) bit replaces the RTR bit in the standard message location as a placeholder in the extended format.
- A recessive bit in the identifier extension (IDE) indicates that more identifier bits follow. The 18-bit extension follows IDE.
- Following the RTR and r0 bits, an additional reserve bit has been included ahead of the DLC bit.

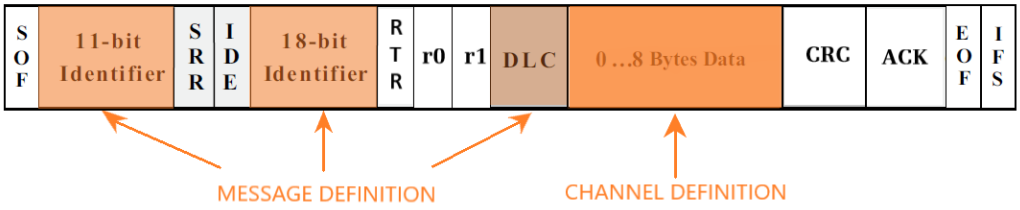


Figure 2.3: Structure of an extended message in CAN

The frame described above can be of four different types:

1. Data Frame: the most common message type, and comprises the Arbitration Field, the Data Field, the CRC Field, and the Acknowledgement Field.
2. Remote Frame: to solicit the transmission of data from another node. The remote frame is similar to the data frame, with two important differences. First, this type of message is explicitly marked as a remote frame by a recessive RTR bit in the arbitration field, and secondly, there is no data.
3. Error Frame: which is a special message that violates the formatting rules of a CAN message. It is transmitted when a node detects an error in a message, and causes all other nodes in the network to send an error frame as well. The original transmitter then automatically retransmits the message. An elaborate system of error counters in the CAN controller ensures that a node cannot tie up a bus by repeatedly transmitting error frames.
4. Overload Frame: similar to the error frame with regard to the format, and it is transmitted by a node that becomes too busy. It is primarily used to provide for an extra delay between messages.

The robustness of CAN may be attributed in part to its abundant error-checking procedures. The CAN protocol incorporates five methods of error checking: three at the message level and two at the bit level. If a message fails any one of these error detection methods, it is not accepted and an error frame is generated from the receiving node. This forces the transmitting node to resend the message until it is received correctly. However, if a faulty node hangs up a bus by continuously repeating an error, its transmit capability is removed by its controller after an error limit is reached.
Error checking at the message level is enforced by the CRC and the ACK slot. The 16-bit CRC contains the checksum of the preceding application data for error detection with a 15-bit checksum and 1-bit delimiter. The ACK field is two bits long and consists of the acknowledge bit and an acknowledge delimiter bit. Also at the message level there is a form check which is done by looking for fields in the message which must always be recessive bits. In particular the bits checked are the SOF, EOF, ACK delimiter, and the CRC delimiter bits and if a dominant bit is detected, an error is generated.
At the bit level, each bit transmitted is monitored by the transmitter of the message. If a data bit (not arbitration bit) is written onto the bus and its opposite is read, an error is generated. The only exceptions to this are with the message identifier field which is used for arbitration, and the acknowledge slot which requires a recessive bit to be overwritten by a dominant bit.
The final method of error detection is with the bit-stuffing rule where after five consecutive bits of the same logic level, if the next bit is not a complement, an error is generated. Stuffing ensures that rising edges are available for on-going synchronization of the network. Stuffing also ensures that a stream of bits is not mistaken for an error frame, or the seven-bit interframe space that signifies the end of a message. Stuffed bits are removed by a receiving node's controller before the data is forwarded to the application. With this logic, an active error frame consists of six dominant bit violating the bit stuffing rule. This is interpreted as an error by all of the CAN nodes which then generate their own error frame. This means that an error frame can be from the original six bits to twelve bits long with all the replies. This error frame is then followed by a delimiter field of eight recessive bits and a bus idle

period before the corrupted message is retransmitted. It is important to note that the retransmitted message still has to contend for arbitration on the bus.
The data link and physical signalling layers, which are normally transparent to a system operator, are included in any controller that implements the CAN protocol, such as TI's TMS320LF2812 3.3-V DSP with integrated CAN controller. Connection to the physical medium is then implemented through a line transceiver such as TI's SN65HVD230 3.3-V CAN transceiver to form a system node. Signalling is differential which is where CAN derives its robust noise immunity and fault tolerance. Balanced differential signalling reduces noise coupling and allows for high signalling rates over twisted-pair cable. Balanced means that the current flowing in each signal line is equal but opposite in direction, resulting in a field-cancelling effect that is a key to low noise emissions. The use of balanced differential receivers and twisted-pair cabling enhances the common-mode rejection and high noise immunity of a CAN bus. The High-Speed ISO 11898 Standard specifications are given for a maximum signal rate of 1 Mbps with a bus length of 40 m with a maximum of 30 nodes. It also recommends a maximum indeterminate stub length of $0.3m$. The cable is specified to be a shielded or unshielded twisted-pair with a 120-Ω characteristic impedance (Zo). The ISO 11898 Standard defines a single line of twisted-pair cable as the network topology, terminated at both ends with 120-Ω resistors, which match the characteristic impedance of the line to prevent signal reflections. According to ISO 11898, placing RL on a node must be avoided because the bus lines lose termination if the node is disconnected from the bus. The CAN standard defines a communication network that links all the nodes connected to a bus and enables them to talk with one another. There may or may not be a central control node, and nodes may be added at any time, even while the network is operating-
Defined the main characteristic of of CAN and introduced its characteristics at data link and physical layers, it is now possible introduce the different application layer supported by the network.

2.2 CANOpen

2.2.1 Introduction

In terms of OSI model, CANopen is a standardized protocol at application layer for distributed automation systems based on CAN (Controller Area Network) at physical and data-link layers. In particular it offers the following performance features:

- Transmission of time-critical process data according to the producer consumer principle;
- Standardized device description (data, parameters, functions, programs) in the form of the so-called "object dictionary". Access to all "objects" of a device with standardized transmission protocol according to the client-server principle
- Standardized services for device monitoring (node guarding/heartbeat), error signalisation (emergency messages) and network coordination ("network management")
- Standardized system services for synchronous operations (synchronization message), central time stamp message
- Standardized help functions for configuring baud rate and device identification number via the bus
- Standardized assignment pattern for message identifiers for simple system configurations in the form of the so-called "predefined connection set"

2.2.2 Structure

At CANopen layer (Fig. 2.4), two devices exchange Communication and Application Objects (COB). These instruments allow to access to objects through a 16-bit index and a 8-bit subindex. As known, since the communication between two devices passes through all their levels, COB_s are inserted in one or more CAN sequence with predefined identificator.

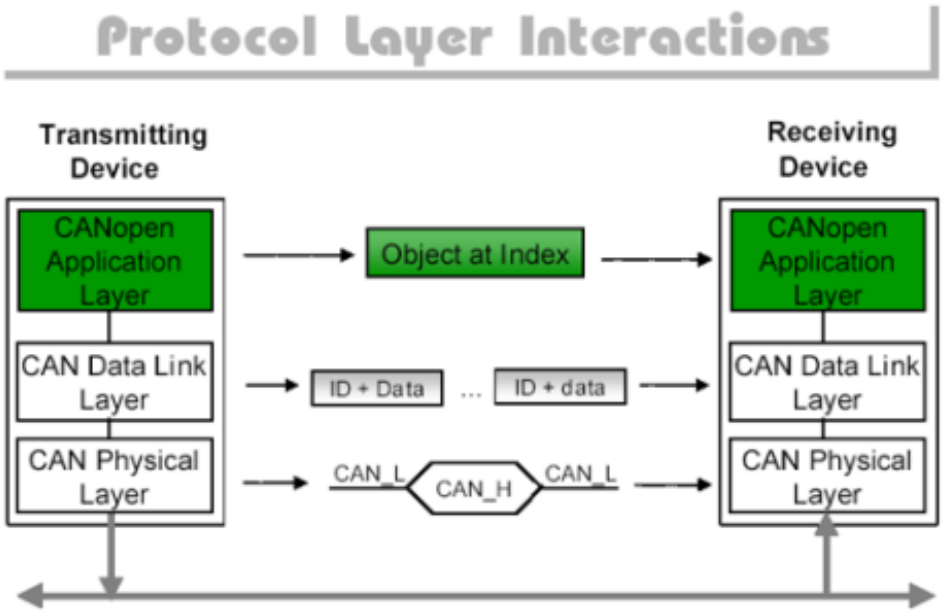


Figure 2.4: Protocol Layer interaction

Once understood in which part of a network CANOpen is inserted it is essential to define how a device with this application protocol is structured (Fig. 2.5). In particular it is mainly composed by three parts:

1. Communication interface, where are defined the communication objects and all the services to transmit them over the bus.
2. Object dictionary, where are defined all types of data and Objects (COB) available for the network.
3. Application program, which defines all the function governing internal mechanisms and for the connection to the hardware interfaces of the process.

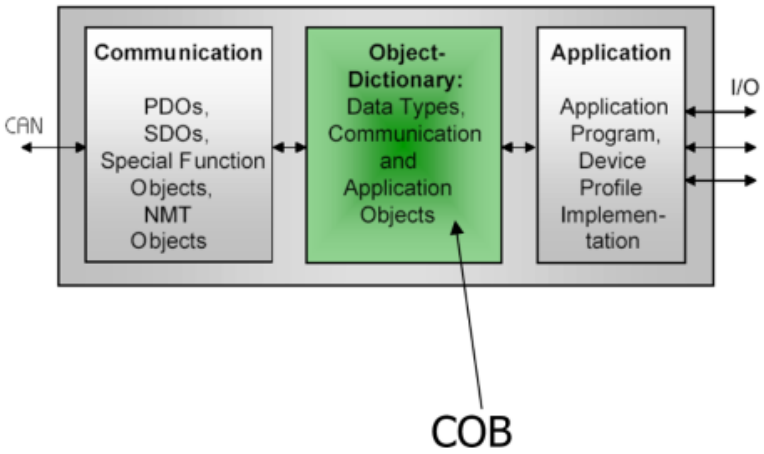


Figure 2.5: CANOpen Device model

The communication Objects defined in the communication interface can be of four different types:

- Network Management
- Service Data Objects
- Process Data Objects
- Predefined messages such as Sync Object, Time stamp Objects and Emergency Objects

In CANopen there are many ways of transmitting data such as synchronous, asynchronous, with or without time constraints. Each type of transmission depends on the specific communication object.
In particular if the SDO is used, then the transmission is not characterized by strong time constrains and it is possible to access to the object dictionary with the index and the subindex.
The SDO transmission uses a Client/Server model and it is also allow to send more than 8 bytes, since it used the segmentation mechanism. Two CAN messages are required to transfer data. Firstly there is the 'SDO request' and secondly the 'SDO response' . The two network nodes, which are involved in this process, are designated as 'SDO Client' and as 'SDO Server'. The server provides the data via its object directory or accepts it. As an SDO transfer is confirmed, each SDO request must be answered, even if the device can provide no relevant data or the request was faulty. This negative response is called "Abort" and includes a long "Abort code" in addition to the 4 bytes. For these characteristics this modality is used to configure devices in a CAN network with messages with low priority.
Instead if PDO is used then the transmission has high speed and high priority. In a PDO message data can be no more than 8 bytes, while the PDO objects are mapped in the Object Dictionary. It is used a Master/slave model where the Master node is the only one device that can allow the initialization of the communication, which is a transmission without confirm (there is no reply to the sender to assure that the message is achieved). With PDO it is possible to have different types of transmission such as synchronous/asynchronous and Cyclic/Acyclic. In particular the synchronization allows the devices to be strictly synchronized with the master clock, while asynchronous transmission does not depend on that clock. The concept of cyclic and acyclic communication are instead linked with the periodicity: in the cyclic one the PDO is sent periodically inside the sync windows, while for the acyclic one it is sent always in the sync period but without fixed interval. It is possible to summarize the way of transmission with PDO in 6 different methods:

1. Synchronous Cyclic: this method is the best one for the data flow and guarantees real-time. The PDO is sent when the Sync object is received and it is possible to set the period.
2. Synchronous Acyclic : this is the cheapest one for the bandwidth, but it is sensitive to error. The PDO is sent after some event and allows the synchronization with the Sync Object but not in a periodic way.
3. Synchronous Remote : this method is based on the Remote Frame. The PDO is sent only after having received the remote frame and data are updated only after the Sync object, which can be sent manually or periodically.
4. Asynchronous Remote : this method work as previous without using the Sync object, but just sending the PDO after the remote frame
5. Asynchronous Manufacturer Profile : this method allows to achieve very good real-time performance with the drawback that the network could saturate. In fact a frame is sent every time an Applicant Event which means for example every time change the value of the PDO and so there could be problem of saturation if it changes too many times.
6. Asynchronous Device Profile: this method is linked with the manufacturer choice.

Predefined messages have specific function, for example SYNC telegram is a periodical broadcast telegram which can be used to transfer synchronized input data and simultaneously activate output data on a system-wide basis. Emergency Object are another example of predefined message, in

particular if a CANopen Slave malfunctions, an emergency message to the fieldbus is sent. Only one emergency message is sent for each error. If the nodes receive an emergency message, they may perform an emergency stop or a predefined action for the specific emergency.
Finally the Network Management Services are used to activate the different system statuses of a CANopen device. When the device has been switched on ('Power On'), an internal system initialization ('initialization') is run. Following a successful initialization, a boot-up telegram responds. The device is now operational and is in the 'preoperational' status. As a slave can be parametrized in this status, it is therefore possible to read and write SDO. The NMT command 'Operational' can be used to switch a CANopen device to the operational status. In this status the process data is active, the PDO communication is running. The NMT command 'Reset Application' is used to restart the CANopen device, whereas the NMT command 'Reset Communication' is used to reset the CANopen communication of the device only. Following both resets, the device is in the initialization status.

2.3 CANOpen Safety

2.3.1 Protocol characteristics

CANopen Safety is a protocol extension of the proven CANopen Standard $(EN50325-4)$.
At the physical layer and the lowest level of communication according to the OSI model, the internationally standardised CAN bus $(ISO11898-1/2)$ is used. CANopen Safety was specified by the international users' and manufacturers association CAN in Automation (CiA) as $DS304$ and transferred into $EN50325-5$.
Therefore, it provides a standardised protocol which allows the user to transfer functional safety information or process data. The basic idea of CANopen Safety is the transmission of process data twice independently in two subsequent CAN messages which make up a CANOpen Safety message, called SRDOs (Safety Relevant Data Objects). The first message contains the process data as in normal PDOs, while the second frame transmits the same data but bit-wise inverted. The two used CAN_{ID_s} are different in minimum in two of the eleven bits. This concept, called serial redundancy, allows the use of the entire 8-byte data field. This means the PDO mapping defined in standard CANopen device and application profiles may be easily adapted to the SRDO mapping.
SRDO$_s$ are transmitted periodically with a fixed cycle time called SCT (safeguard cycle time) and as soon their transmission is interrupted an error is signalled and the safety configuration is imposed. In such way there is the constant monitoring of safety data.
There are two kinds of use for SRDOs: data transmission and data reception,distinguished by the information direction. Devices where the information direction is set to transmit (tx) are SRDO producers and devices where the information direction is set to receive (rx) are called SRDO consumers. Each part of an SRDO has an abundant error checking which is composed by the five, already described, methods typical of CAN which also provide the acknowledgement of each received message. Moreover consuming safety-related device crosschecks the received process data and uses also a running number to detect possible repetition, lost, insertion or wrong sequences.
In addition, there is also a time-out between the transmission of the two CAN messages composing an SRDO which is called SRVT (safety-relevant validation time). There are two timers monitoring the two cited times and in particular used for detecting possible time-out:

1. One monitors the Safety-related object validation time (SRVT)
2. One monitors the safeguard cycle time (SCT).

If one of the two timers expires (Fig. 2.6), the actuating device goes into safe state.

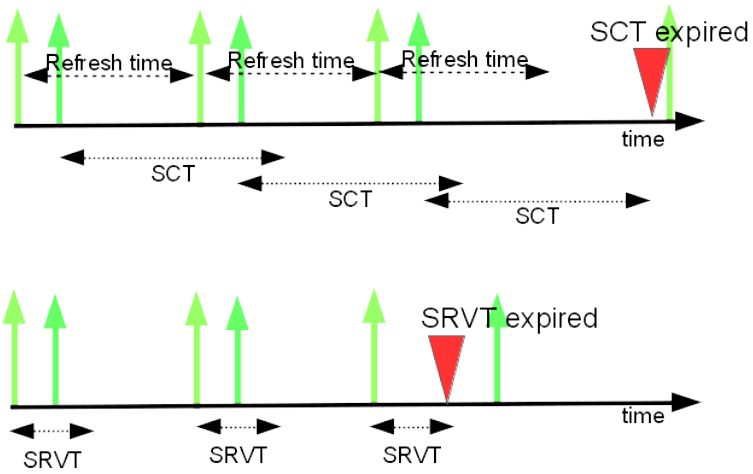


Figure 2.6: CANOpen Safety possible Time-out

The CANopen Safety protocol also defines the GFC (Global Failsafe Command), which consists of two high-priority CAN messages (CAN identifiers 1 and 2), containing no data and which can be sent by all networked nodes. The node that sends the GFC must inform the network of the reason for this GFC transmission via SRDO. In general the GFC is used to speed up the system reaction time. Since it is transmitted event-triggered, it is not considered safe, but when coupled with the transmission of a corresponding SRDO the safety issue is resolved. Due to CANopen application layer compatibility, the safety-relevant communication is limited to transmitting 64 SRDOs. The number of SRDO consumers is not limited. It is assumed that a device with the need of safety-relevant communication may use all other CANopen functionality including PDO (process data object), SDO (service data object), NMT (network management), and Emergency services. SDO access to safety-relevant application and configuration objects is allowed only during the NMT pre-operational state. As the sources (safety inputs) are the origin of safe communication objects (SRDOs), their number is limited to 64. The number of safety controllers is not limited in theory, as CAN allows consumers to listen to the same SRDO(s). As concerning the SRDOs in the object dictionary, they are described by the SRDO communication parameter and the SRDO mapping parameter. The SRDO communication parameter describes the communication capabilities of the SRDO. The SRDO mapping parameter contains information about the content of the SRDOs (device variables). The indices of the corresponding Object Dictionary entries are computed, in general, by the following formulas:

- SRDO communication parameter index = 1300h + SRDO-number
- SRDO mapping parameter index = 1380h + SRDO-number

When a sensor (input) device detects any single failure, it stops transmission of SRDOs, which will be detected by all SRDO consuming devices, which will then transit into safe state. The benefit of the simple SRDO protocol is that no additional CRC is required and that the standardized CANopen profiles can be used without any change. But the simplicity of the CANopen Safety protocol has some drawbacks on the other side. For example, the Safe configuration of application parameters is not standardized by the CiA 304 specification.
Once setted the safe configuration of communication parameters, then it is inserted in the object dictionary and a CRC is calculated and associated to it. The CANopen Safety protocol stack checks periodically, if the CRC sum is still the same. Any inconsistency would lead to stop device operation and to go into safe state. If an amendment is necessary, then, after the change, it is mandatory to confirm the new CRC associated to the new configuration. Otherwise the safe communication does not work. The CANopen Safety Protocol allows safety relevant sensors and actuators to be connected directly with each other. The presence of a safety relevant control unit (e.g. PLC, safety monitor) is possible if required by the application, but not mandatory. Therefore logically comparable safety chains, like those found in standard wired technology, can be realized.

Summing up the safety protocol encompasses procedures to systematically detect faults or errors that could occur during operation. In particular for controlling each error, the CANopen Safety protocol uses different control strategies (also reported in tab 2.1):

- A running-number, to detect possible loss, insertion, repetition and wrong sequence
- 2 timers to detect possible time-out.
- A confirmation of the reception of the message with an acknowledgement for each part (CAN message) of the SRDO.
- The identification of transmit and receive SRDO with double identification of the node: with the id of the first can message and the bit-wise inverted id of the second one
- A CRC checksum to detect possible corruption of data
- The cross-check of the effectiveness bit-wise inversion.

Fault	**Safety Measures**					
	Runn. Nr	Time out	ack mess	rx/tx SRDO	CRC	cross-checking
Corruption			x		x	x
Wrong sequence	x					x
Repetition	x					x
Insertion	x		x	x		x
Loss	x		x			x
Delay		x				
Coupling			x	x		

Table 2.1: Safety measures adopted by CANopen Safety to detect errors and faults

With all the cited measures to obtain a safe transmission, with CANopen Safety a device can reach a SIL-3 (Safety Integrity Level) certification according to the standard EN 61508. Unfortunately there are no tools, useful for the safety certification, supplied by the developers. An example of industrial product, which obtained the SIL-3 certification using CANopen Safety, is the safety encoders from the Optocode series. In particular this kind of product was certified by the German testing authority, TUV Rheinland, and it represents the first certified optical SIL-3 encoders.

2.3.2 Initialization of a safety network with CANopen Safety

As concerning CANopen Safety it is finally presented the general flow chart of the network initialisation process, controlled by a NMT master application or configuration application (Fig. 2.7).

A) In the first step (Step A) all the devices are in the node state PRE-OPERATIONAL which is entered automatically after power–on. In this state the devices are accessible via their default SDO using identifiers that shall been assigned according to the pre-defined connection set. In this step the configuration of device parameters takes place on all nodes which support parameter configuration. Some configuration data might be safety-relevant. So additional measures shall be taken, to ensure the safety function in the network. This is done by a configuration application or tool which resides on the node that is the client for the default SDOs. For devices that support these features, the selection and/or the configuration of PDOs, the mapping of application objects (PDO mapping), the selection and/or the configuration of SRDOs, the mapping of application objects (SRDO mapping), the configuration of additional SDOs and, optionally, the setting of COB-IDs may be performed via the default SDO objects. In many cases, however, a configuration is not necessary, since default values are defined for all application and communication parameters.

B) If the application requires the synchronisation of all or some nodes in the network, the appropriate mechanisms have to be initiated. This step is used to ensure that all nodes except safety nodes are synchronised by the SYNC object before entering the node state OPERTAIONAL (in step E). The first transmission of SYNC object starts within 1 sync cycle after entering the PRE–OPERATIONAL state.

C) In the optional step C, node guarding may be activated (if supported) using the guarding parameters configured in the first step.

D) In step D, safety parameters are checked. In particular, this step covers the following safety relevant configuration entries:

- SRDO numbers(s) used
- Time inspection (refresh time for TX, SCT for RX and the SRVT between two telegrams)
- Information direction
- Mapping parameter

The checksum of the respective configuration entries is defined and associated to the safe configuration. In fact, after the initialization, this CRC sum is periodically checked and in the case of mismatch the safety node stop the transmission of SRDOs and the safety controller enter in safe state.

E) In Step E all the nodes move to the operational state and the initialization procedure is finished.

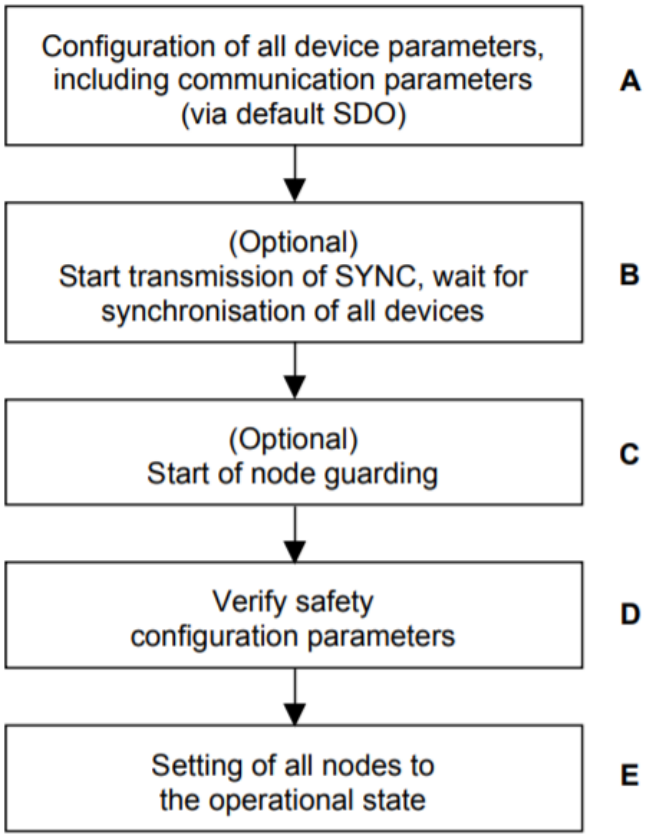


Figure 2.7: Flow chart of the network initialisation process for safety networks with CANopen Safety

2.4 EtherCAT

EtherCAT (Ethernet for Control Automation Technology) is an Ethernet-based fieldbus system, designed by Beckhoff Automation. The EtherCAT protocol is based on a master/slave architecture using a standard ethernet connection. In particular it is Industrial Ethernet and utilizes standard frames and the physical layer as defined in the Ethernet Standard IEEE 802.3. However, it also addresses the specific demands faced in the automation industry, where:

- There are hard real-time requirements with deterministic response times.
- The system is usually made up of many nodes, each only having a small amount of cyclic process data.
- Hardware costs are even more important than in IT and office applications.

The above requirements make using a standard Ethernet (Appendix G) network at the field level practically impossible. If an individual Ethernet telegram is used for each node, the effective data rate sinks significantly for just a few bytes of cyclic process data: the shortest Ethernet telegram is 84 bytes long (including the Inter Frame Gap), of which 46 bytes can be used for process data. For example, if a drive sends 4 bytes of process data for the actual position and status information and receives 4 bytes of data for the target position and control information, the effective data rate for both telegrams sinks to 4/84 = 4.8 %. Additionally, the drive usually has a reaction time that triggers the transmission of the actual values after receiving the target values. At the end, not much of the 100MBit/s transfer rate remains. Protocol stacks, such as those used in the IT world for routing (IP) connection (TCP), require additional overhead for each node and create further delays through the stack runtimes. To overcome these difficulties, EtherCat uses a high performing mode of operation, in which a single frame is usually sufficient to send and receive control data to and from all nodes. The EtherCAT master sends a telegram that passes through each node. Each EtherCAT slave device reads the data addressed to it and inserts its data in the frame as the frame is moving downstream. The frame is delayed only by hardware propagation delay times. The last node in a segment or branch detects an open port and sends the message back to the master using Ethernet technology's full duplex feature. The telegram's maximum effective data rate increases to over 90 %, and due to the utilization of the full duplex feature, the theoretical effective data rate is even greater than 100 MBits/s. The EtherCAT master is the only node within a segment allowed to actively send an EtherCAT frame; all other nodes merely forward frames downstream. This concept prevents unpredictable delays and guarantees real-time capabilities. The master uses a standard Ethernet Media Access Controller (MAC) without an additional communication processor. This allows a master to be implemented on any hardware platform with an available Ethernet port, regardless of which real-time operating system or application software is used. EtherCAT slave devices use an EtherCAT Slave Controller (ESC) to process frames on the fly and entirely in hardware, making network performance predictable and independent of the individual slave device implementation.
As regarding the frame, EtherCAT embeds its payload in a standard Ethernet one (Fig. (2.8)).
The EtherCAT frame is identified with the Identifier (Ox88A4) in the EtherType field. Since the EtherCAT protocol is optimized for short cyclic process data, the use of bulky protocol stacks, such as TCP/IP or UDP/IP, can be eliminated.

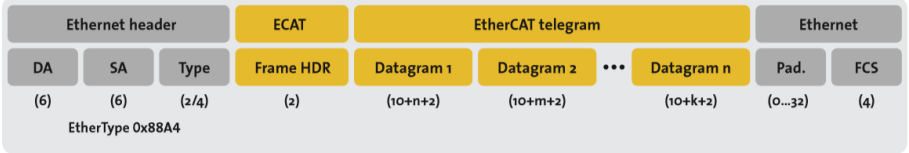


Figure 2.8: EtherCAT in a standard Ethernet frame (according to IEEE 802.3)

To ensure Ethernet IT communication between the nodes, TCP/IP connections can optionally be tunneled through a mailbox channel without impacting real-time data transfer. During startup, the master device configures and maps the process data on the slave devices. Different amounts of data can be exchanged with each slave, from one bit to a few bytes, or even up to kilobytes of data. The EtherCAT frame contains the frame header and one or more datagrams. The datagram header indicates what type of access the master device would like to execute:

- Read, write, or read-write
- Access to a specific slave device through direct addressing, or access to multiple slave devices through logical addressing (implicit addressing)

Logical addressing is used for the cyclical exchange of process data. Each datagram addresses a specific part of the process image in the EtherCAT segment, for which 4 GBytes of address space is available. During network startup, each slave device is assigned one or more addresses in this global address space. If multiple slave devices are assigned addresses in the same area, they can all be addressed with a single datagram. Since the datagrams completely contain all the data access related information, the master device can decide when and which data to access and so a fixed process data structure is not necessary. For example, the master device can use short cycle times to refresh data on the drives, while using a longer cycle time to sample the I/O; This also relieves the master device in comparison to in conventional fieldbus systems, in which the data from each node had to be read individually, sorted with the help of the process controller, and copied into memory. With EtherCAT, the master device only needs to fill a single EtherCAT frame with new output data, and send the frame via automatic Direct Memory Access (DMA) to the MAC controller. When a frame with new input data is received via the MAC controller, the master device can copy the frame again via DMA into the computer's memory – all without the CPU having to actively copy any data. In addition to cyclical data, further datagrams can be used for asynchronous or event driven communication.
Besides to logical addressing, the master device can also address a slave device via its position in the network. This method is used during network boot up to determine the network topology and compare it to the planned topology. After checking the network configuration, the master device can assign each node a configured node address and communicate with the node via this fixed address. This enables targeted access to devices, even when the network topology is changed during operation, for example with Hot Connect Groups. There are two approaches for slave-to-slave communication. A slave device can send data directly to another slave device that is connected further downstream in the network. Since EtherCAT frames can only be processed going forward, this type of direct communication depends on the network's topology, and it is particularly suitable for slave-to-slave communication in a constant machine design (e.g. in printing or packaging machines). In contrast, freely configurable slave-to-slave communication runs through the master device, and requires two bus cycles.
EtherCAT offers a lot of flexibility regarding both the cable types, so each segment can use the exact type of cable that best meets its needs, that the topology: EtherCAT supports almost all the topologies (line, tree, star, or daisy-chain). Finally it is important to highlight that for some tasks an accurate synchronization is required and for this reason EtherCAT uses the allignment of the distribute clocks. In particular the calibration of the clocks in the nodes is completely hardware-based. The time from the first DC slave device is cyclically distributed to all other devices in the system. With this mechanism, the slave device clocks can be precisely adjusted to this reference clock. The resulting jitter in the system is significantly less than 1 μs.
Since the time sent from the reference clock arrives at the slave devices slightly delayed, this propagation delay must be measured and compensated for each slave device in order to ensure synchronicity and simultaneousness. This delay is measured during network startup or, if desired, even continuously during operation, ensuring that the clocks are simultaneous to within much less than 1 µs of each other. If all nodes have the same time information, they can set their output signals simultaneously and affix their input signals with a highly precise timestamp. In motion control applications, cycle accuracy is also important in addition to synchronicity and simultaneousness. In such applications, velocity is typically derived from the measured position, so it is critical that the position

measurements are taken precisely equidistantly (i.e. in exact cycles). Even very small inaccuracies in the position measurement timing can translate to larger inaccuracies in the calculated velocity, especially relative to short cycle times. With EtherCAT, the position measurements are triggered by the precise local clock and not the bus system, leading to much greater accuracy. Additionally, the use of Distributed Clocks also unburdens the master device; since actions such as position measurement are triggered by the local clock instead of when the frame is received, the master device doesn't have such strict requirements for sending frames. This allows the master stack to be implemented in software on standard Ethernet hardware. Even jitter in the range of several microseconds does not modify the accuracy of the Distributed Clocks. Since the accuracy of the clock does not depend on when it's set and so the frame's absolute transmission time becomes irrelevant. The EtherCAT master need only to ensure that the EtherCAT telegram is sent early enough, before the DC signal in the slave devices triggers the output.

2.5 Safety for Ethercat (FSOE)

2.5.1 protocol characteristics

Modern communication systems not only realize the deterministic transfer of control data, they also enable the transfer of safety-critical control data through the same medium. EtherCAT utilizes the protocol Safety over EtherCAT (FSoE : FailSafe over EtherCAT) for this purpose and so allows:

- A single communication system for both control and safety data
- The ability to flexibly modify and expand the safety system architecture
- Pre-certified solutions to simplify safety applications
- Powerful diagnostic capabilities for safety functions
- Seamless integration of the safety design in the machine design
- The ability to use the same development tools for both standard and safety applications

FSoE was developed according to IEC 61508, is TUV certified, and is standardized in IEC 61784-3 to test FSoE slave devices in accordance with the FSoE specification. In fact EtherCAT Technology Group (ETG) implemented some test cases in the tool, which were reviewed by TUV Sud, that conclusively confirmed the qualification of the tool for validation and compliance. The tool is available for all users of Safety over EtherCAT technology and it is also used for the official FSoE conformance tests in the future.

The protocol is based on a master-slave relationship with a single device referred to as FSoE master and several FSoE slaves. The master-slave communication takes place via the so called FSoE Connections, which are established between the master and each slave. A basic Safety Protocol Data Unit (Fig. 2.9) , which is the shortest frame, consist in 6 bytes , which carry 1 byte of safety data.

1 B	1 B	2 B	2 B
CMD	Data	CRC	Conn. ID

Figure 2.9: FSOE basic frame

The first byte contains the command to which a specific state of the FSoE connection corresponds and determines the meaning of the safety data. The Data field is followed by its own CRC and then by two bytes that report the Connection ID, which is an univocal number assigned by the FSoE master to each FSoE slave during the initialization phase. Notably, in order to ensure the lowest error probability, the CRC is elaborated in a different manner with respect to traditional protocols.
Indeed, a device receiving a Safety PDU calculates the CRC on a more complex data structure, which includes 11 fields (tab. 2.2). Such a data structure contains two additional fields that identify the sequence number, which consist in a 16-bit counter from 1 to 65535, re-initialized to 1 when the maximum value is reached. Each safety device manages its own sequence number, which represents the progressive number of the Safety PDU it transmits. The device also maintains the sequence number of the Safety PDU it expects to receive from its partner. Since these two values must coincide, the actual matching is verified by including them in the structure on which the CRC is calculated.

Field Nr.	**Field**
1	received CRC (bit 0-7)
2	received CRC (bit 8-15)
3	ConnId (bit 0-7)
4	ConnId (bit 8-15)
5	Sequence Number (bit 0-7)
6	Sequence Number (bit 8-15)
7	Command
8	SafeData[0]
9	0
10	0
11	0

Table 2.2: Structure of CRC calculation

When more than one byte of safety data is transmitted, a more complex Safety PDU can be used(Fig. 2.10).

1 B	2 B	2 B		2 B	2 B	2B
CMD	$Data_0$	CRC_0	$\ldots$	$Data_i$	CRC_i	$Conn.ID$

Figure 2.10: FSOE extended frame

During normal operation of the protocol, the FSoE master sends a Safety PDU to the addressed slave and waits for the answer. Then, it moves to the next slave of the network. The slave moves from the 'Reset' state to the 'Session' one and then to the 'Connection' one, where the FSoE connection is actually established, upon suitable commands received from the master. Subsequently, in the 'Parameter' state the operational safety parameters are exchanged and, finally, the master sends a command to enter in the 'Data' state. In every state, an immediate transition to 'Reset' may take place due to either a command received from the master or a problem detected by the slave. In each device, a watchdog timer monitors the FSoE communication cycle in order to detect possible delays on the network. If the FSoE master does not receive the answer from a queried slave within a specified time-out, then the watchdog timer of the master triggers the re-initialization of the FSoE connection with that slave. Conversely, if a FSoE slave is not queried by the master within a time-out, then the watchdog timer of the slave forces such device to enter in the reset state. In such a state, the slave exits the operational state and waits to be re-initialized by the master. The FSoE master can handle several slaves by establishing a unique FSoE connection for each of such devices. The communication medium, from the safety point of view, is seen as a black channel (Appendix E). With such an approach, safety applications and standard applications can coexist, sharing the same communication system at the same time.
Summing up the safety protocol encompasses procedures to systematically detect faults or errors that could occur during operation. In particular for controlling each error, the Safety over EtherCAT protocol uses different control strategies (also reported in tab 2.3):

- A session-number, for detecting buffering of a complete startup sequence;
- A unique connection ID and a unique slave address for safely detecting misrouted messages via a unique address relationship.
- A CRC checksum (Tab. 2.2) for detecting message corruption from source to sink.
- A sequence number for detecting interchange, repetition, insertion or loss of whole messages.
- A watchdog, which monitors the FSoE communication cycle in order to detect possible delays

With all these measures, the FSoE standard ensures that the residual error probability can be kept below 10^{-9} which allows to achieve the highest safety performance, referred to as SIL3.

Fault	**Safety Measures**			
	Conn. ID	Seq. Nr	W.dog	CRC
Corruption				x
Interchange	x	x		
Repetition		x		
Insertion		x		
Loss		x	x	
Delay			x	
Misrouting	x			

Table 2.3: Safety measures adopted by FSoE to detect errors and faults

2.6 Profibus and Profinet

2.6.1 Profibus

Profibus is based on universal international standards and it is oriented to the OSI (Open System Interconnection) reference model for international standard ISO 7498. In this model, every layer handles precisely defined tasks. (Layer 1 of this model is the physical layer and defines the physical transmission characteristics, layer 2 is the data link layer and defines the bus access protocol, layer 7 is the application layer and defines the application functions, while layers 3 to 6 are not used).
There are two variations of Profibus in use today:

- Profibus DP (Decentralised Peripherals) is used to operate sensors and actuators via a centralised controller in production automation applications.
- Profibus PA (Process Automation) is used to monitor measuring equipment via a process control system in process automation applications. This variant is designed for use in explosion/hazardous areas. The Physical Layer conforms to IEC 61158-2, which allows power to be delivered over the bus to field instruments, while limiting current flows so that explosive conditions are not created, even if a malfunction occurs. The number of devices attached to a PA segment is limited by this feature.

Profibus DP, which is the most commonly used, is a master/slave token bus protocol for deterministic communication between Profibus masters and their remote I/O slaves, which are peripheral devices. The slaves form 'passive stations' on the network since they do not have bus access rights, and can only acknowledge received messages, or send response messages to the master upon request. It is important to note that all ProfiBus slaves have the same priority, and all network communications are originated by the master. Instead ProfiBus master forms an 'active station' on the network which are subdivided in two different classes :

Class 1 , in which the master handles the normal communication and exchanges data with the slaves assigned to it.

Class 2 , in which the master is a special device primarily used for commissioning slaves and for diagnostic purposes

A token bus protocol is used to regulate the access to the network by the different masters. In particular all masters form a logical ring in which a position with a fixed address assigned to each device. Every node knows the addresses of the previous and the next device, while the last one is followed by the first one.
A special PDU, called token, allows the device holding it to transmit in the network, to interview its slaves or to manage the network (while the other nodes must only listen). This PDU is held only for a fixed time (t_h token holding time), after this period the device holding it must send it to the next node. This working configuration provides some basic function useful to the maintenance of the network (which are described in Appendix F):

- addition of a new station in the logical ring;
- erasure of station from the network;
- initialization of the logical ring;
- management of the loss of the token;

The master-to-master communication takes place only when the token must be exchanged. Instead to communicate with slave/slaves a master operates using a cyclic transfer and in particular it can address to an individual slaves, or to a defined group of slaves (multicast), or it can broadcast a telegram to all connected slaves. Since it is used a periodic polling mechanism between masters and slave then is is also deterministic. The length (and timing) of the I/O data to be transferred from a single slave to a master is predefined in the slave's device data base or GSD file. The

GSD files of each device connected via the network (slaves and class 1 masters only) are compiled into a master parameter record which contains parametrization and configuration data, an address allocation list, and the bus parameters for all connected stations. A master uses this information to set up communication with each slave during startup.
PROFIBUS supports four data transmission services:

1. send data with no acknowledge (SDN);
2. send data with acknowledge (SDA);
3. send and request data with reply (SRD);
4. cyclic send and request data with reply (CSRD).

But PROFIBUS DP supports only SDN and SRD. The SDN is a service used for broadcast from a master station to all other stations on the bus. Conversely, the SRD is based on a real dual relationship between the initiator (master station holding the token) and the responder (slave or master station not holding the token). Most data transfers are defined as SRD which is a message cycle consisting of a request packet and an immediate response. A response can occur as a one byte acknowledgement or a data packet. During a message cycle an addressed station has to respond within a bounded time interval. Since there are different types of message, the structure of the frame depends on its function. In particular the telegram can be:

- With no data (Fig. 2.11), used to coordinate the network and send fuction information;

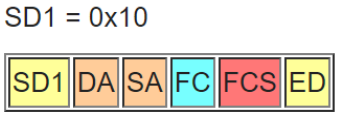


Figure 2.11: Frame structure for a message with no data

- With variable length data (Fig. 2.12), to send messages of variable length;

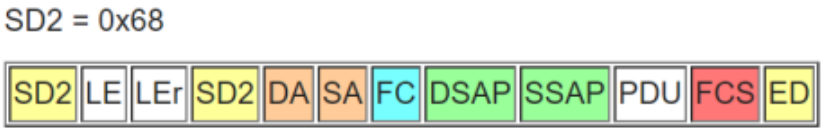


Figure 2.12: Frame structure for a message with variable length data

- With fixed length data (Fig. 2.13), to send messages of fixed length;

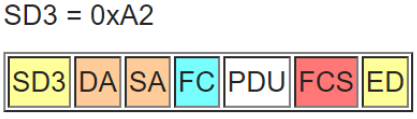


Figure 2.13: Frame structure for a message with fixed length data

- Token (Fig. 2.14), to communicate between masters.

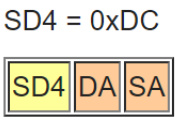


Figure 2.14: Token structure

where the acronym stands for (Tab. 2.4):

SD	Start delimiter
LE	Length of protocol data unit
LEr	Repetition of length of protocol data unit (with an Hamming distance equal to 4)
FC	Function Code
DA	Destination Address
SA	Source Address
DSAP	Destination Service Access Point
SSAP	Source Service Access Point
PDU	Protocol Data Unit
FCS	Frame Checking Sequence, calculated by adding up the bytes within the specified length
ED	End Delimiter

Table 2.4: acronym definition

The Service Access Points(SAP) are internal addresses used at any level, which allow to access to different services. In particular the different services are described in the following table (Tab. 2.5):

Default 0	Cyclical Data Exchange(WriteReaData)
54	Master-to-Master SAP(M-M Communication)
55	Change Station Address(SetSlaveAdd)
56	Read Inputs(RdInp)
57	Read Outputs(RdOutp)
58	Control Commands to a DP Slave(GlobalControl)
59	Read Configuration Data(GetCfg)
60	Read Diagnostic Data(SlveDiagnosis)
61	Send Parametrization Data(SetPrm)
62	CheckConfiguration Data (ChkPrm)

Table 2.5: SAP definition

As concerning the physical level, Profibus provides two different carrier band transmission, a broadband transmission and also the possibility of using the optical fibre. The carrier band transmission are called phase-continuous and phase-coherent. The first one provide a velocity of 1 Mbit/s and a transmission in which the logical level 0 and 1 are identified by different frequencies, 3,75 Mhz and 6,25 Mhz respectively. The transition between the 2 frequencies is gradual and for this reason the name continuous. The Phase-coherent modality use the same methods, but link the transmission velocity to the frequencies. For example for a velocity of 5 Mbit/s the frequencies used are 5 Mhz and 10 Mhz. The both carrier band transmission use a coaxial cable. The broadband method, instead, provides fixed velocity (1, 5 and 10 Mbit/s) and relative fixed frequencies (1.5, 6 and 12 Mhz respec-

tively).
Finally the Optical fibre is the fastest method, but which needs the use of electro-optical converts.

2.6.2 Profinet

Profinet was born as an evolution of Profibus to allow communication between several Fieldbus using industrial Ethernet. The four key functions of PROFINET are:

1. Performance: automation in real-time
2. Safety: safety-related communication with PROFIsafe
3. Diagnostics: high plant availability due to fast commissioning and efficient troubleshooting
4. Investment protection: seamless integration of fieldbus systems

Standard Ethernet communication via TCP(UDP)/IP communication is sufficient for data communication in some cases. But, as already said, in industrial automation requirements regarding time behaviour and isochronous operation cannot be fully satisfied using the TCP/IP channel. Profinet introduced a scalable real-time concept which is a solution for this problem. In particular this concept can be realized with standard network components, such as switches and standard Ethernet controllers. The real-time (RT) communication takes place without TCP/IP information. The transmission of RT data is based on cyclical data exchange using a provider/consumer model. In this type of model there is not a device which controls the access to the bus and so every node is able to start the transmission on the bus if it is not busy (Producer interface), and every node is able to read the bus if someone is transmitting (Consumer interface). To enable enhanced scaling of communication options and, thus, also of determinism in Profinet, real-time classes have been defined for data exchange. These classes involve unsynchronized and synchronized communication. The details are managed by the field devices themselves. Real-time frames are automatically prioritized in Profinet compared to UDP/IP frames, in order to prioritize the transmission of data in switches to prevent RT frames from being delayed by UDP/IP frames.
In particular Profinet differentiates the following classes for RT transmission, which differs in determinism rather than in performances.

- **RT CLASS 1**: Unsynchronized RT communication within a subnet. No special addressing information is required for this communication. The destination node is identified using the Destination Address.
- **RT CLASS 2**: frames can be transmitted via synchronized or unsynchronized communication.
- **RT CLASS 3**: Synchronized communication within a subnet.
- **RT CLASS UDP**: The unsynchronized cross-subnet communication between different subnets requires addressing information via the destination network (IP address).

Cyclic I/O data are transmitted unacknowledged as real-time data between provider and consumer in a parametrizable resolution. They are organized into individual I/O elements (sub-slots). The connection is monitored using a watchdog (time monitoring mechanism).
During data transmission in the frame, the data of a sub-slot are followed by a provider status. This status information is evaluated by the respective consumer of the I/O data. In particular the cycle can be subdivided in 4 different phases:

- RED phase: In this phase only RT Class 3 can be sent at scheduled instants and using a rigid fixed path.
- ORANGE phase: In this phase only RT Class 2 can be sent iat scheduled instants but without a fixed network topology.

- GREEN phase: In this phase only messages that use Ethernet priority defined by IEEE 802.1Q standard can be sent. In particular RT Class1, RT Class2 and TCP and UDP protocols messages are sent.
- YELLOW phase: Characterized by the same traffic of the GREEN phase but with the additional constraint that only the frame that can be sent completely are effectively transmit.

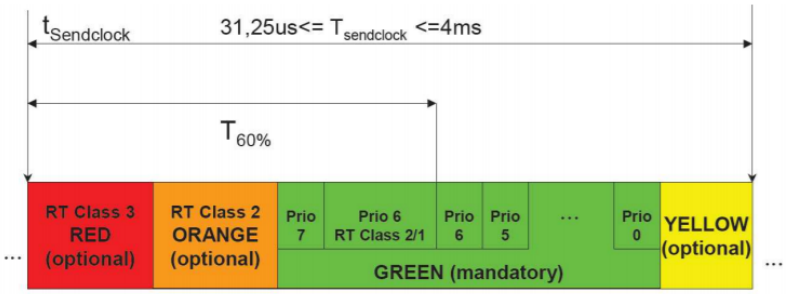


Figure 2.15: Profinet cycle

Instead acyclic data exchange can be used to parametrize and configure IO-Devices or to read out status information. This is accomplished with read/write frames via standard IT services using UDP/IP.
PROFINET provides also an Isochronous data exchange defined as Isochronous Real-Time (IRT). These data exchange cycles are usually in the range of a few hundred microseconds up to a few milliseconds. The difference to real-time communication is essentially the high degree of determinism, so that the start of a network cycle is maintained with high precision. The start of a network cycle can deviate up to 1 µs (jitter). IRT is required, for example, for motion control applications (positioning control processes) because these devices need to be synchronized.
For data exchange with multiple parameters, Multicast Communication Relation (MCR) has been defined. This allows direct data traffic from a provider to multiple nodes (up to all nodes) as direct data exchange. MCRs within a segment are exchanged as RT frames. Cross-segment MCR data follow the data exchange of the RT class. All the PROFINET messages are structured as an ethernet frame and in particular the structure is presented in following figure (Fig. 2.16). Besides at physical level common ethernet specific are used.

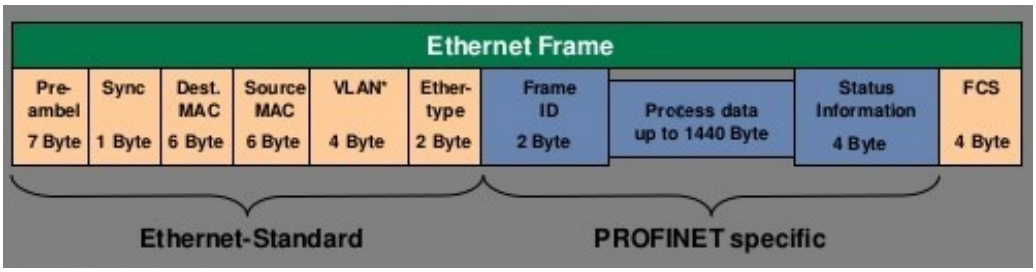


Figure 2.16: Profinet Message

2.7 Profisafe

As seen, PROFIBUS and PROFINET are industrial communication systems providing comprehensive coverage including both factory and process automation. Both protocols are specified in the communication profile family 3 in the International Standards IEC 61158 and IEC 61784-1/-2. The first release of a specification for safety communication was in 1999. Since this event, PROFIsafe

has evolved to become the leading and most comprehensive safety communication technology in the world. PROFIsafe is an International Standard IEC 61784-3-3 since 2007.

2.7.1 protocol characteristics

The PROFIsafe protocol is suitable for both PROFIBUS and PROFINET networks without impacts on these existing fieldbus standards. It is possible to transmit safety messages on the existing standard bus cables in coexistence with standard messages. "Single Channel" approach allows the use of standard PLCs with integrated but logically separated safety processing, but also physical separation of standard and safety communication is supported without any drawback induced by PROFIsafe. The PROFIsafe protocol works equally well over copper wires, fiber optics, wireless communication links, or backplanes, so it does not make any assumptions regarding transmission rates or error detection mechanisms, with a "Black Channel" approach (Appendix E).
Besides PROFIsafe uses the following safety measures :

- The numbering of the PROFIsafe messages (sequence error detection used for "timeliness")
- A time expectation with acknowledgment (timeout error detection used for "timeliness")
- A Codename between sender and receiver ("authentication")
- Data integrity checks (CRC = cyclic redundancy check)

Using the Monitoring Number, a receiver can see whether or not it received the messages within the correct sequence. Besides also the sender can understand if the message was rightly received. In fact the receiver returns a message with the Monitoring Number only as an acknowledgement to the sender. As safety systems are real-time systems, process signals must not only be delivered correctly, but also in time. In case of timeliness errors, the safety system will initiate a safe reaction, e.g. safely stop the movement of a drive. For this purpose, F-Devices utilize a watchdog timer that is restarted whenever a new PROFIsafe message with a new subsequent Monitoring Number arrives. The $1:1$ relationship between the F-Host(safety controller) and a F-Device(safety device) facilitates the detection of misdirected message frames. Sender and receiver must simply have identification (Codename which in PROFIsafe is called 'F-Address') that is unique in the network, and can be used for verifying the authenticity of a PROFIsafe message. A cyclic redundancy check (CRC) plays a key role in detecting corrupted data bits. The necessary probabilistic examination makes use of the definitions within the IEC 61508 that considers the probability of dangerous failures of entire safety functions. A PROFIsafe message that is exchanged between F-Host and its F-Device is carried within the payload of a standard PROFIBUS or PROFINET message. In case of a modular F-Device with several F-Modules, the payload consists of several PROFIsafe messages. The data unit consists of three fields. The first field contains the F-Input or F-Output data using the already-mentioned subset of data types. These data structures of a particular F-Device usually are defined via its associated GSD (General Station Description) file. Normally, factory automation and process automation place different requirements upon a safety system. One deals with short ("bit") signals that must be processed at a very high speed, the other involves longer ("floating point") process values that may take a little more time. PROFIsafe recommends using 1 up to 12/13 bytes F-Input/ Output data for factory automation applications since all F-Hosts are obliged to support at least this data length. However, the measures of PROFIsafe (CRC signature) are laid-out such that data lengths up to a maximum of 123 bytes can be supported. The second field consists of a Control Byte if the SPDU was sent by the F-Host or a Status Byte if it was sent by the F-Device. This information helps synchronizing the sender and receiver of PROFIsafe SPDUs. The third field of a PROFIsafe SPDU is a 32 bit CRC signature. The Monitoring Number is not transmitted within a PROFIsafe SPDU. Both sender and receiver use their own Monitoring Number generators that are synchronized via the Control Byte and Status Byte. Correct synchronization is monitored through the inclusion of the Monitoring Number values into the CRC signature calculation. The generators are based on an efficient pseudo-random number generator. Every connection uses a different seed for the generator, derived from its respective Codename ("F-Address"). Sender and receiver of PROFIsafe SPDUs are located in layers

above the "Black Channel" communication layers and are usually realized in software. Their central functionality is a state machine controlling the regular cyclic processing of PROFIsafe messages and the exceptions such as start-up, power-on/off, CRC error handling etc . . .
Figure(2.17) shows how the PROFIsafe layers interact with the technology part in F-Devices and with the user program in F-Hosts.

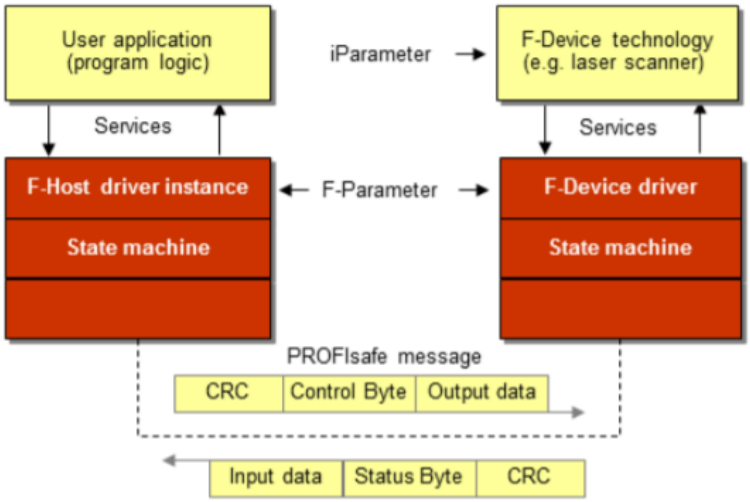


Figure 2.17: PROFIsafe layer structure in F-Host and F-Device

PROFIsafe provides different services for F-hosts or F-devices. For F-Host the main services provide exchange of F-Output and F-Input data. During start-up, or in case of errors, the actual process values are replaced by default fail-safe values (FV). These fail-safe values shall be all "0" to force the receiver into a safe state. PROFIsafe communication errors cause the F-Host driver to switch into a safe state. A safety function is usually not allowed to automatically switch from a safe state to normal operation without human interaction. To inform the user program that an operator intervention and acknowledgement is requested, PROFIsafe provides an additional service "OA-Req". PROFIsafe informs the F-Device about a pending request such that the F-Device can indicate it via a LED. The operator acknowledgement can be passed over from the user program to the F-Host driver via a corresponding service "OA-C".
For F-Device the PROFIsafe services include the corresponding exchange of F-Output and F-Input data, the extra possibility to activate and report fail-safe values, the indicators for the iParameter, which are the technology-specific parameters of a F-Device, handling and for the already mentioned operator request. In case a F-Device needs different iParameters at runtime, another set of services is available. The service "iPar-EN" allows the user program to switch the F-Device into a mode during which it will accept new iParameters. The companion service "iPar-OK" indicates to the user program the readiness to resume normal safety operation.
Following the most important i-parameters are introduced to understand them function and utility:

- F-S/D-Address (short: "F-Address"), it is a unique connection identification for F-Devices/F-Modules within one PROFIsafe island and corresponds to the Codename. The F-Device technology compares this "F-Address" with the locally assigned value of a micro switch or otherwise entered information to ensure the authenticity of the connection.
- F-WD-Time, it specifies a number of milliseconds for a watchdog timer. This timer monitors the reception of the next valid PROFIsafe SPDU.
- F-SIL, it indicates the SIL expected by the user for the particular F-Device. It is compared with the locally stored manufacturer information.
- F-iPar-CRC, it is a signature across all the iParameters within the technology of the F-Device.

- F-Par-CRC, it is a signature across all the F-Parameters. It is used to ensure correct delivery of the F-Parameters.

Additionally, the F-Device technology is able to report F-Device faults to the F-Host driver via the flag "Device-Fault" in the Status Byte. The duration of the demand of a F-Device for a safety reaction shall be long enough to be transmitted by the PROFIsafe communication. A special service informs the technology about new Monitoring Numbers in order to facilitate the realization of this requirement. Diagnostic information from the PROFIsafe layer may be passed over to the technology part via a special service.
PROFIsafe is suitable for safety functions up to SIL-3. In particular the mandatory safety manual of each and every F-Device shall provide information about the SIL_{CL} (claim limit) and the PFH_d (probability of dangerous failure per hour). Besides PROFIsafe provides a specification for test and certification: currently two PI test laboratories are accredited for the PROFIsafe testing. To determine the achieved SIL of a particular safety function, the F-Devices provide the necessary information in their safety manual. In the first step, the least SIL_{CL} (claim limit) of all the safety devices (F-Devices, F-Host) is selected. This determines the maximum achievable SIL of the entire safety function. In some cases system manufacturers may offer system support to upgrade to a higher SIL via redundancy of F-Devices and corresponding system software. In the second step, the PFH_d values are added and the result is checked against the permitted value ranges for a particular SIL. The least SIL value from these two steps determines the achievable SIL.

Chapter 3

AUTEC products

Autec Srl is an important Italian company which produces and sells industrial radio remote controllers. All these instruments need to be implemented with the industrial protocols introduced in the previous chapter. In particular for its products Autec uses principally CANOpen as communication network, but the research and the development of new devices are always prepared to the possibility of changing the protocols to meet the requirements of the customers and the follow to market trends.

3.1 Radio Remote Controller and its safety function

A radio remote controller is a small, usually hand-held, electronic device for controlling another device with radio frequency signals. A controller is composed of two parts:

- a transmitting unit from which the commands are sent;
- a receiver unit which is installed on a device and control it applying the commands received.

These instruments are used in industrial systems with some specifications that are requested from this type of environment and that are different for every kind of device which needs to be controlled. In particular, it is essential that these controllers are resistant and able to work in critical conditions. Moreover, it is important that they are user-friendly with a display and some leds to communicate easily with the user.
Finally it is essential the safety of the user. For this reason, different strategies, that should be certificated, are used to guarantee it. To certificate a radio remote controller, the first step is to demonstrate that it complies with all the standards in force in the market in which the equipment will be used. Depending on the jurisdiction, regulations relevant to radio controls fall under several categories:

- Radio emissions and immunity: these requirements address the risk of interference of the device with other radio devices, and the health risks associated with electromagnetic radiation.
- Functional safety: these requirements are the most complex, and address the risk that the device may malfunction causing dangerous machine behaviour.
- Requirements specific to the lifting machine: these standards may impose special requirements on the system in a wide variety of ways, including safety performance, physical parameters, labelling, etc.
- Electrical safety: these requirements aim to control the risk of electrical shocks and fire, and are common to a wide range of electrical equipment.

The aforementioned regulations can be complex, and can interact with each other. Despite that, they are called 'minimum requirements' and it is necessary a second step to have a 'safe' device. It consists in ensuring that the radio remote controller also complies with the suitable protection level

resulting from the evaluation of the risks.
For radio remote control applications it is essential that the commands are understood correctly at the receiver, and any damage or corruption of the telegram does not result in erroneous machine motion. To that end, each telegram must include some additional error check information so that the receiver can ensure that the telegram was received correctly. Error detection methods are heavily founded in mathematics, and vary in complexity and efficiency. Typically, though, special coding systems are used such that a small change in the input data (i.e. the commands for the RRC system) causes a large change in the telegram. This minimizes the chance that two (or more) errors could cancel each other out, making a damaged telegram to appear as valid. In fact, the number of simultaneous errors that would need to occur in order to defeat an error-detection system is a measure of its effectiveness, and is called Hamming Distance. Besides in a radio remote control application, the communication medium is open. This means that it is not possible to guarantee that the receiver will not be exposed to messages being transmitted by other remote control systems, even located far away. In this case, the use of standard protocols increases the similarity between telegrams on different devices, thus increasing the risk that an overheard telegram from another system may be inadvertently decoded and accepted. Such an occurrence cannot be considered a random event (such as noise damaging a telegram) and on the contrary, it is a systematic risk. Use of proven proprietary telegram protocols helps to protect against interference from other types of systems. But in order to avoid problems with similar systems from the same manufacturer, it is also vital that there is rigorous management of unique (non repeatable) identity codes for each safety radio control system. Once ensured the communication, it also necessary to manage all the possible critical situations that could threaten the safety of the user.
There are too many cases that could be reported, but three important examples are introduced to have an idea of the safety approach.

- The stop button is vital for all industrial environments since it allows the immediate stop of the system in critical circumstances. It needs to be implemented with a redundant strategy to avoid failures and once activated, it should need to be manually reset before the radio control system can be used again.
- The constant control of the device when the controller is pushed on, also when no commands are sent, is another important safety approach. In fact in such way a radio control system is able to constantly check that no errors occur, ensuring that when a command is pushed it will be transmitted or, in case of the detection of an error, transmitting the safe configuration. If this approach is used the controller needs rechargeable batteries since a large amount of power is used. Instead the radio controllers which use standard batteries transmits only when they must transmit a command to reduce the power consumption, but also drastically reducing the safety.
- Finally an essential safety function for a radio remote controller is the protection against unintended movements from standstill (UMFS), that could be due to radio interference. To avoid UMFS it is important that a movement is done only in correspondence of an action over the transmitting unit, in particular a redundant strategy is used, that allows the remote radio controller to transmit only when a button is pushed or a lever is moved.

3.2 Example of AUTEC products

Autec produces many different kind of manufactures, for this reason to not dwell on issues that are far from the goal of this thesis, only the principal devices which use the industrial protocols will be briefly introduced.

3.2.1 Dynamic+ Radio Remote Controller

Autec's DYNAMIC+ consists of both a joystick transmitting units and the corresponding CRD receiver. It is suitable for continuous-current applications typical of hydraulic machines used in construction environments, logistics, transport, infrastructure maintenance and much more. It is able to guarantee advanced diagnostics and to show the machine data on graphic display or LEDs. This product is implemented with CANopen which provides standard communication objects for real-time network configuration and maintenance data. It works with two different frequency bands: 863-870 MHz or 915-928 MHz using bi-directional radio communication with fully-automatic search of working frequency. DYNAMIC+ transmitting units and receiving units, in fact, communicate with each other in "frequency hopping" mode, as they dynamically utilize the working frequencies included in the 863-870 MHz band (or 915-928 MHz for non-European markets). Basically, they constantly change working frequencies, verifying that a frequency is free before using it. In this way they maintain a stable radio link in the presence of interference while simultaneously creating the least possible interference for other radio equipment in the area. The high reliability of the connection allows a fast and accurate response to the proportional controls without the need for frequency mapping, even when radio frequencies are considerably crowded. As regarding the safety, the DYNAMIC+ has been designed according to the most recent standards relating to Functional Safety, such as EN ISO 13849-1 and IEC 62061. The transmitting and receiving units communicate through a unique and univocal proprietary Autec code which is not reproducible. The safety is not developed with only software, but it is principally done with hardware. In particular when the DYNAMIC+ identifies some emergency condition it uses redundant solid state relay, which it is typical for moving machine control.
The safety of DYNAMICS+ is certified, in fact:

- Performance of the STOP function up to PL e cat.4 / SIL 3 classified, according to EN ISO 13849-1 / EN IEC 62061.
- Performance of protection against unintended movements from standstill (UMFS) up to PL d cat.3 / SIL 2 classified, in accordance with EN ISO 13849-1 / EN IEC 6206.

The proportional joystick remote controls of the DYNAMIC+ (Fig. 3.1) have been developed for typical hydraulic and mobile machine applications. They are available in four different models: DJS, DJL, DJR and DJM. They offer up to 12 analog commands and up to 64 digital commands with the option for data feedback, which allows visualization of the information coming from the machine either on a graphic display or via a LED panel.

Figure 3.1: Different type of joystick for DYNAMICS+

The compact receiving unit (Fig. 3.2), 8-30 VDC power supply, with customized cabling (M12 circular connectors or 10-pin reduced plug or cable gland), has a maximum of 12 analog and 64 digital outputs (available via the CANopen interface), 2 STOP outputs, 2 UMFS outputs, 4 programmable MOSFET outputs and 2 CAN outputs. This double CAN output allows simultaneous management of both CANopen and J1939 protocols. The CRD also has a 4-digit display for diagnostics. Usually a standard antenna is internal, but the unit also supports External antenna with 1 m or 3 m extension cord.

Figure 3.2: Different configuration of receiver unit for DYNAMICS+

3.2.2 Air Radio Remote Controller

Remote controls of the AIR Series are ideal for automation, industrial lifting and both operational and self-propelled machinery. It is composed of a transmitting unit, which can be either a joystick (Fig. 3.3.a) or hand-held transmitters (Fig. 3.3.b), and a receiving unit (Fig. 3.3.c). These series of devices have a complete connectivity through CANopen, Profibus, ProfiNet, EtherCat, EtherNet IP and serial interfaces for the control and communication of data. In particular, the controller is principally implemented with CANopen since most of the customers use this protocol. Moreover, this instruments are equipped with a data logger for recording all the radio controls operations. The system of data transmission and control is bi-directional, dual-band and is configurable by the user:

- with automatic frequency search at system start-up (434/915 MHz).
- with completely automatic search through FHSS technology - Frequency Hopping Spread Spectrum (870/915 MHz).

As regarding the safety Autec designs and produces AIR controls with a level of safety that meets even the strictest of the standards. The most important aspects of the remote control (functional, electrical, environmental, radio) reflect state-of-the-art technology for both control and communication. The STOP function of AIR Series models has been certified by TUV Rheinland as compliant up to PL e according to EN ISO 13849-1 and to SIL 3 according to EN IEC 62061. Radio frequency communication is made through a certified and proprietary Autec system which is suitable for safety-critical applications. Each remote system uses its own unique code which cannot be reproduced.
When the AIR controller identifies some emergency condition it switches off the power supply to the devices by opening a electro-mechanical relay which is normally close. Moreover, to ensure the safety of the whole system also in case of failure, this product uses a redundant relay. This approach is typical for industrial production machine.

Finally another two important strengths of these series are

- Reliability: all electronic and mechanical parts are designed, manufactured and tested to withstand heavy use in adverse conditions (e.g. extremes temperature, shock and vibrations, substances such as oils, paints and thinners; even electromagnetic disturbance, dust and water). The AIR Series features casings with IP65 protection. 100% of the radio controls produced are subject to functional testing with specific equipment that ensures proper construction of each part that goes into an Autec system. An effective traceability system allows us to precisely identify the components and activities carried out through the production process to ensure the highest levels of safety and reliability.
- Flexibility: each receiver can be paired with any transmitter in the series, according to the needs of the specific applications. In this way, it is possible to adapt to complex working situations, including the use of multiple machines, maintaining both high reliability and safety. Thanks to the wide configurability of available actuators and displays, the transmitting units can adapt to many application requirements. In addition, with all the different receivers available and the ability to insert additional cards into the expansion slots, the AIR Series offers high configurability and the ability to optimize the output interface with respect to the function required by the machine.

(a) Different type joystick for AIR series

(b) Different type of hand held transmitters for AIR series

(c) Different type of receiver unit for AIR series

Figure 3.3: Principal AIR series products

3.3 New development: ideas and future improvements

As already mentioned, Autec is continuously researching to develop new products to satisfy the market requirements. In particular this thesis would like to propose a new approach for the safety: a complete development with the software through the safety protocols and especially through CANopen Safety. It will be also essential to develop the tests necessary to prove the reliability of the applications and for a further certification of them with TUV Rheinland, as the current products. This aspect may represent a problem, since it may require a long time. Moreover CANopen safety does not provide any tool for certification, as conversely FSOE and Profisafe do. Another interesting improvement would be the introduction of the Industrial Internet of Things (IIoT) approach. In particular, since the Autec products already record the data of the radio operation in some memory, it would be interesting to save these data in a cloud to elaborate them. In this way there would be the possibility of predicting possible failures and consequently doing the right maintenance in advance. This idea introduces another possible problem since when, the data are loaded in a cloud, they are subjected to security attacks. For this reason before introducing this approach it is essential to develop also security applications. In this thesis, only the first activity will be faced and in particular the goal of this work is the development of safety applications, while the IIot activity is an idea for a future improvement.

Chapter 4

Codesys

4.1 Introduction

The development environment Codesys (Controlled Development System) created by the German company 3S-Smart Software Solution allow to implement abstract machine for logical control using one of the languages defined by IEC 61131-3. The development tool is used for two applications:

- To code and run PC controller for automation machine. For these applications Codesys works as compiler of IEC 61131-3 programs, producing a program executable by real time CoDeSys SP RTE.
- To code industrial control system (as PLC). For these applications Codesys is sold with the compiler and with the personalized interface for the specific PLC.

This development environment provides so many tools that it can cover the most different fields, in particular it can range from the soft motion, with an editor for the the motion planning, to specific tasks of safety for disparate systems. This last function is implemented with pre-certified software components that make it much easier for device manufacturers to have their SIL2 or SIL3 controllers certified. Therefore, Codesys Safety consists of components within the programming system and the runtime system, whereas the project planning is completely integrated in the IEC 61131-3 programming environment. Besides different field bus can be used directly in the programming system Codesys and for this purpose, the tool integrates configurators for the most common system such as PROFIBUS, CANopen, EtherCAT, PROFINET and EtherNet/IP. These two important facts are the reasons of the choice of introducing Codesys. In particular for this thesis some safety applications for the Autec products will be developed in the Codesys environment and with CANopen field bus.

4.2 Briefly explanation of how Codesys works

It is possible to download the demo free of charge from the official website, with the restrictions that it is not possible to use all the library and that it has a maximum number of components, or it is possible to buy the license and the full version from the website too. Once installed the development tool it is possible to see all the programs that it includes and in particular Codesys V2.3 which is the most important and which allows the development and simulation of code or control industrial systems according to IEC 61131-3. This packet also includes:

- Configuration, which is useful to configure the servers OPC and DDE, which are the servers supported by Codesys for communication and remote applications.
- ENI interface, which allows to connect the environment CoDeSys with an external database, such that data could be share between more users, programs and computers.

- CoDeSys SP RTE which allows to implement real time execution.
- HMI runtime system, which allows to show graphic visualization created with CoDeSys

A project consists in all the objects useful for the creation of the PLC application. In particular the objects are: POU (program Organization Unit), data defined by the user, visualization part, resources and libraries. Every POU comprises a initial part in which are defined all the data essential for the success of communication and a central part which contains the actual aforementioned program written in one of the IEC 61131-3 languages. Running Codesys, it will load the latest project or, if it is the first time it runs, an example. Once created a new project, which is set in ST and name PLC_PRG by default, the screen is presented as in Fig. 4.1.
It is subdivided in two parts: in the right one, the window link with the element in the left part is showed, while in the left one the Project Browser is showed. This one is made of 4 objects:

- POU: list of the POU (Program Organization Unit), separated in Program (PRG), Function(FC) and Function Blocks (FB), as imposed by regulation IEC 61131-3;
- Data types: list of the type of data defined by the user ;
- Visualizations: list of graphic panels;
- Resource: list of configuration menu .

Moreover, there is the possibility to convert a program from a language to another, to use all the library that support that language and to insert programmed POU such as timers, registers counters etc. A project for the development of a control system can be subdivided in two parts: a program which controls the system and a program that simulates the plant. Obviously they must interact between each other and in particular the control part, once elaborated the outputs, must command the actuators, while the simulator must show the state of the plant coherently with its real operating. By default only the PLC_PRG is run, so it is necessary to modify the system configuration such that both the programs are run.

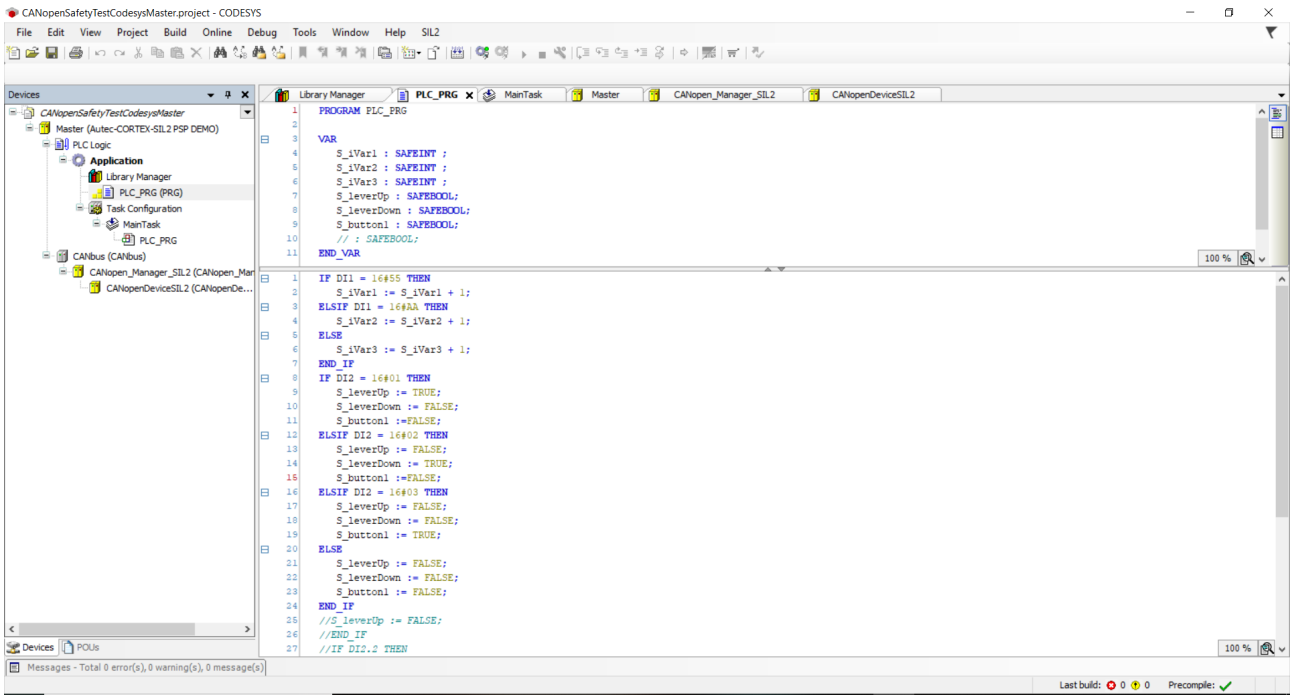


Figure 4.1: Project Browser

Once briefly introduced how Codesys works, it is necessary to explain all the type of data that can be used. In particular Codesys support different standard data which are associated with a memory

space and a particular value:

- Boolean data: which can assume only the value true or false and for which are reserved 8 bit;
- Integer data: which are BYTE, WORD, DWORDS, SINT, USINT, INT, UNIT, DINT and UDINT. For each type of integer is reserved a particular memory space and possible value. (Tab. 4.1)

Type	minimum value	maximum value	memory space
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32768	32767	16 Bit
UINT	0	65535	16 Bit
DINT	-2147483648	2147483647	32 Bit
UDINT	0	4294967295	32 Bit

Table 4.1: Different type of Integer Data

- Real and LReal data: used to represent rational number with floating point. For real data 32 bit of memory are reserved with a lower and upper bound equal to 1.175494351e-38 and 3.402823466e+38 respectively, while fore Lreal data 64 bit of memory are reserved with a lower and upper bound equal to 2.2250738585072014e-308 and 1.79769313486231e+308 respectively.
- String : which can contain all type of character and have an unlimited length(in principle, cince it is limited by physic limits).
- date and time data: TIME ,TIME OF DAY(TOD), DATE and DATE AND TIME (DT) SAVED AS DWord; tie is expressed in millisecond for TIME and TOD, while in second for DATE and DT. To assign this type of data it is necessary to write 't' or 'T' followed by $\#$ and by the time which can be expressed with the following order with day 'd', hours 'h', minutes 'm', seconds 's' and milliseconds 'ms'.
  For example a right assignment is:
  $\boldsymbol{TIME1 = t\#10h38m2s}$
- Addresses: to read or write a defined memory space. In particular the syntax for this type of assignment is $\%$ followed by a prefix for the field (which can be I for input, Q for output and M for memory space) and a prefix for the dimension (which can be X for a bit, B for a byte, W for a word and D for a double word). An example can be:
  $\%IB200 \quad (InputByte200)$.
- Variables: which can be defined either as global (seen by all the program that set up the project) or local (defined only for the program in which they are defined).

4.3 Use of Codesys for safety application

Once introduced the idea of how Codesys works and said that it can range the most different fields, it is possible to focus on the safety function of this development environment. In particular Codesys creates executable safety application code for industrial controllers with integrated code generators on different architectures and it develops safety runtime systems in accordance with IEC 61508. Moreover it offers integrated solutions for practically popular systems. This makes a convenient integrated configuration of Profisafe, FSoE or CANopen Safety in combination with standard field bus systems. Finally there is a cooperation between the Codesys developers and the certification institutions such as TUV SUD and TUV Rheinland, which allows to meet the requirements for certification with more

accuracy and to make realistic estimation for safety projects. In addition there is a separate team exclusively for the development of safety software products. Once the risk analysis for a product is done, it is possible to choose the right Codesys Safety software:

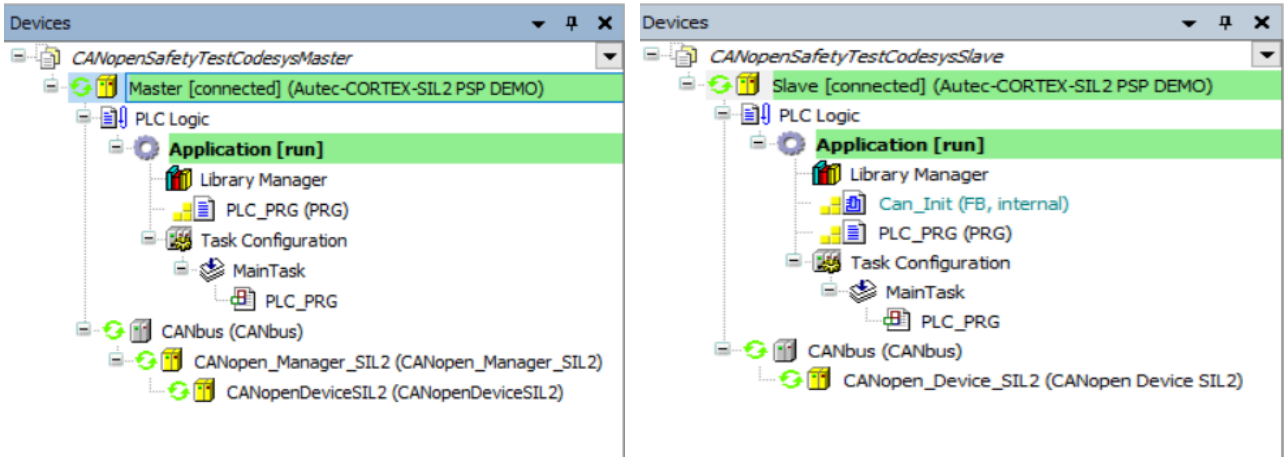
- Codesys Safety, a product for device manufacturers to implement safety-related protection controllers in mechanical engineering pursuant to the Mobile processing machines
- Codesys Safety SIL2, a product for device manufacturers to implement safety-related mobile controllers in accordance with IEC 61508 SIL2 and machine guideline in compliance with IEC 61508 SIL3 and EN13849 PLe.
- Codesys Safety for EtherCAT Safety Module, for device manufacturers and users, based on Beckhoff Automation GmbH's safety logic terminal EL6900 that has already been certified in accordance with IEC 61508 SIL3. The system can be used with any standard controller with CODESYS and CODESYS EtherCAT.

Chapter 5

Application development

5.1 System structure and safety validation operation

Codesys Safety SIL2 was the environment for the application development, since it allows to build safety projects and it provides a series of tools that are useful for using CANopen Safety.
The first activity was characterized by the study of this environment and the development of basic application to understand Codesys characteristics and its operation. After that, two devices with a microcontroller for functional safety were studied since they are the systems over which the applications are run. In particular the microcontroller is Hercules Safety TMS570 MCU by Texas Instrument (Appendix C). Once analysed, they were connected with a PC and, after their drivers installation on the PC, they were connected each other via CANopen , defining one master and one slave. In this context, an experimental session has been carried out to appropriately set the application environment, as well as to configure the evaluation boards. This activity required quite a long time since it implied a series of tests to understand the connection quality, the effective use of CANopen safety, the actual safety performance of the communication system and, finally, all the rules controlling the safety connection. Once established the safe interaction between the two devices and loaded the code on them, the Codesys window device appears as in Fig. (5.1).



(a) Master communication window

(b) Slave communication window

Figure 5.1: Devices communication window

A test network has been implemented as reported in Fig. 5.2. As it can be seen, a radio remote controller was connected to the CANopen network and it was configured to send 4 CANopen messages in which were inserted the Autec telegram. The master node was setted, on Codesys, such that it read all the messages sent by this device and it sent safety messages on the network addressed to the slave evaluation board using the CANopen Safety protocol. Besides, a small actuators, which was able to light some leds according to a specific logic, was connected to the network Configured the slave evaluation board to send a message to this device, the complete designed system architecture was achieved. In particular all the hardware system is also presented in Fig. 5.3
The flow chart of the system is instead presented in the scheme in Fig. 5.4.

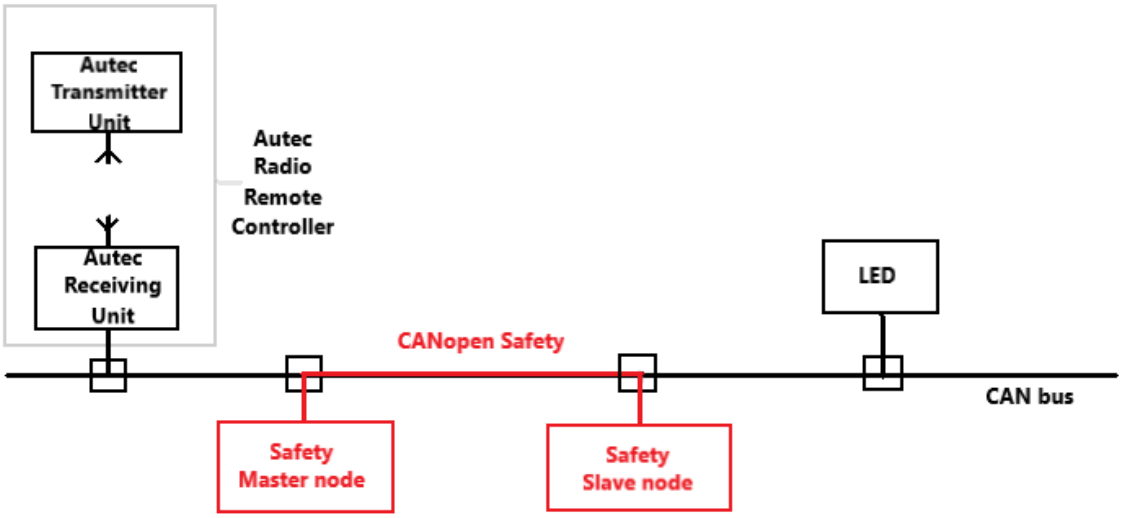


Figure 5.2: Scheme of the hardware system architecture

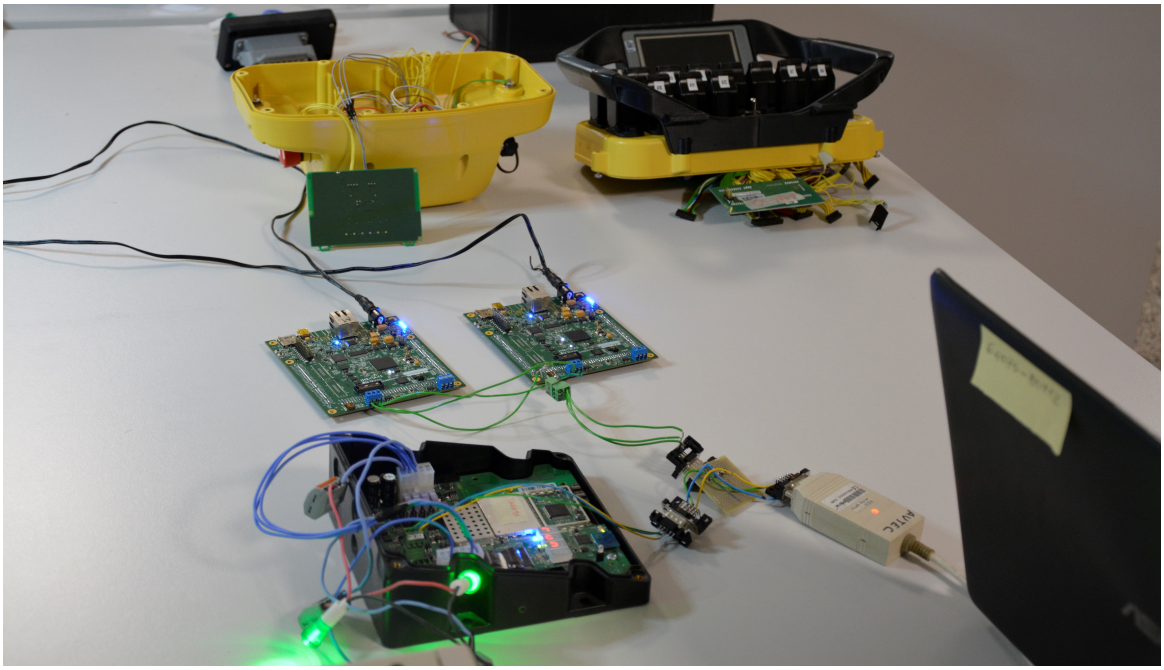


Figure 5.3: Photo of the hardware system architecture

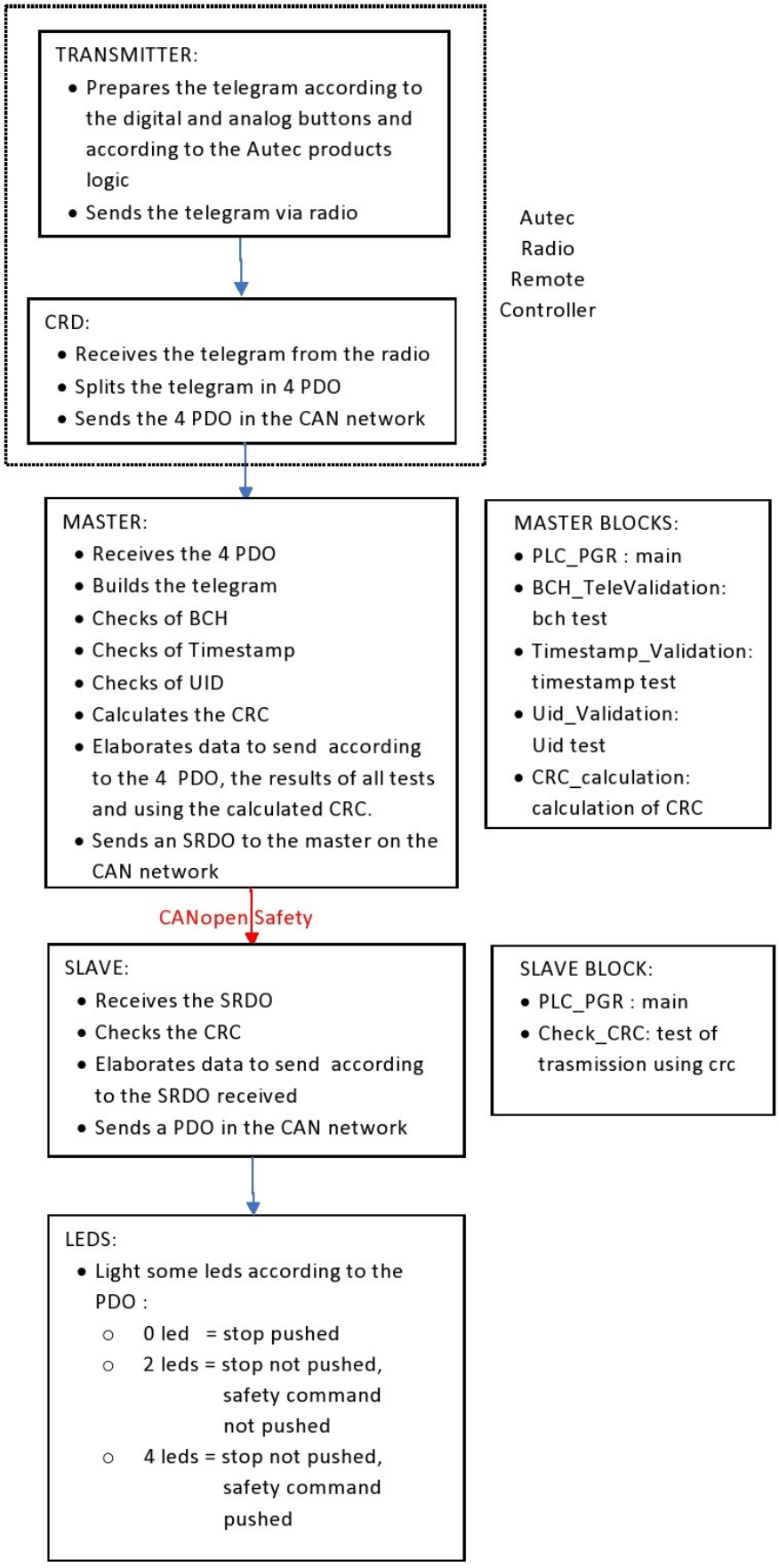


Figure 5.4: Scheme of the flow chart

The network operation is described as follows (Fig. 5.4):

- the radio remote controller split the Autec telegram in 4 PDOs composed by 8 bytes.
  The telegram is composed by 27 bytes of which:
  - 3 bytes for the address of the device.
  - 17 bytes for digital and analog inputs safety messages ,the timestamp and the UID number.
  - 7 bytes used to check the telegram integrity via an algorithm that ensures the correctness of the communication with a defined very high probability.

  In each message it was also sent a byte, which was used as identifier, while the final byte of the last message was set to 0.
- The master evaluation board, once received the 4 PDOs, reconstructed the original telegram only if the 4 identifiers coincided. After that, it used the algorithm for checking if the bits of BCH matched the structure of the message. After the BCH test, the master also checked the timestamp of the telegram, which is a byte calculated to have a time control. In particular a counter is used to have the enumeration of the telegram sent in a similar way to the method used by profisafe and fsoe. The checking algorithm checked that the difference between the value of the bytes of actual timestamp and the value of the expected timestamp is equal to 0 or $+/-1$, detecting possible delay, loss or repetition. Finally, another control was implemented to check that the telegram was sent effectively to the right receiver, simply checking that the UID number of the telegram corresponded to the UID number of the receiver. If all the tests were successful, then the evaluation board prepared the message to transmit, calculating a CRC (cyclic redundancy check) which will be explained in the next section. Once the message was ready, the evaluation board sent it in the CAN network a CANopen Safety message (SRDO) with the bytes of interest. Conversely if any of the tests failed then the evaluation board sents a SRDO in which the bytes configured to signal the error and to impose the safe condition of the machine.
- The slave evaluation board received the SRDO, checked that the transmission was successfully with the code generated by Codesys Safety (which managed the CANopen Safety communication) and with the additional checked of the CRC. After that the board prepared a PDO using the information of the SRDO and sent it in the CAN network such that its information was read by a small electronic board (led unit) which lighted some led according a defined logic.
- Some leds were lighted according to the data sent by the slave to simulate a smart actuator and to have a visual match of the status of the transmission. In particular checking the 4 least significant bits of the safety byte a different number of leds were lighted:
  - 4 leds, if they correspond to the value F (byte =16#*F), which means that the transmission is working and that the safety command was pushed and the stop was not activated;
  - 2 leds, if they correspond to the value C (byte =16#*C), which means that the transmission is right working and that neither the safety command nor the stop button were pushed;
  - 0 led, if they correspond to the value 0 (byte =16#*0), which means that or the stop button was pushed, or an error was detected and so the safety configuration was imposed.

This architecture was designed to manage and transmit safely part of the Autec telegram, the safety commands, in particular, using CANopen safety for the transmission in the CAN network. In fact once checked and verified the message received from the Autec radio remote controller, the safety information was sent through an SRDO, using CANopen Safety, to another device which should simulate the one that usually manages the commands sent by the controller. Moreover the small actuators lightening some leds were inserted in the network to simulate another node in the network and to have a visual match of the status of the transmission.
A future development, as outlined in Fig. 5.5, is expected to implement the safety master in the Autec

receiving unit. In such way the receiving unit would be able to do all the tests, verifying the radio transmission, and then to send the safety information using CANopen Safety. Thus, the communication would result completely safety, since the radio transmission would be seen as a black channel and data would be accepted only after all their complete safety checks.

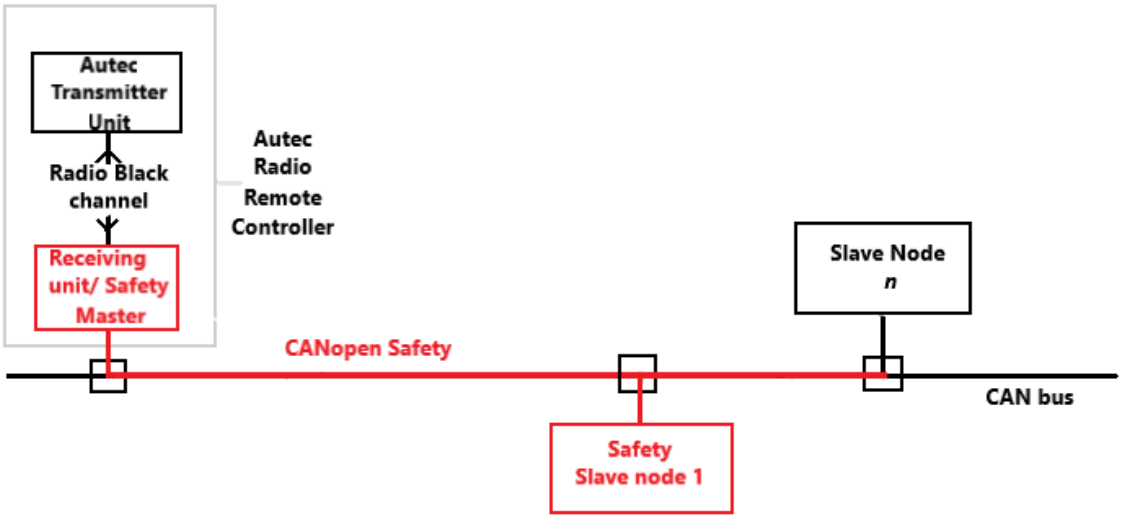


Figure 5.5: Scheme of the future hardware system architecture

All the algorithms were implemented according to the Codesys Safety SIL2 User Manual such that the fully reconstruction of the telegram ensured safety and moreover, it was possible to transmit the safety bytes through CANopen Safety. In particular following the ISO 62061 standard, all the checks of input data and of data pre-transmission corruption were done through the already cited BCH, Timestamp and UID. Also, all the checks of transmission corruption and error were managed by Codesys using the protocol CANopen Safety. Every single rule was checked and all the code analysed to verify that the rule was respected. Regarding the hardware platform, an evaluation board from Texas Instruments including the mentioned Hercules TMS570 safe MCU was used for this work. Although the Hercules TMS570 MCU is a microcontroller designed for safety relevant applications, the evaluation board might not be designed with all procedural requirements and design constraints that are essential for an actual safety relevant hardware platform. However, the check of the correct design of the hardware platform as well as the correct implementation of Codesys Safe SIL 2 runtime environment were not in the scope of this thesis work.
Nevertheless, many different validation tests were carried out to check that the system worked in the expected way in all the operational contexts.
Firstly it was checked that the CANopen Safety worked in the right way, managing and sending all the declared errors. In particular as soon as one of the errors listed in Fig. 5.6 is detected, the safety communication is interrupted and a particular procedure which requires the reboot of the device is needed.

ENUM **ERROR**

Name	Type	Inherited from	Address	Initial	Comment
NO_ERROR	WORD				no error occured
SRDO_RECEIVE_ERROR	WORD				error while receiving SRDO (e.g. old message received)
SRDO_SEND_ERROR	WORD				SRDO couldn't be sent
SRDO_DATA_ERROR	WORD				wrong SRDO data received
INVALID_CONFIGURATION	WORD				inconsistent configuration detected
SRVT_TIMEOUT	WORD				SRDO SRVT timeout detected
SCT_TIMEOUT	WORD				SRDO SCT timeout detected
UNSAFESTACK_NOT_OPERATIONAL	WORD				Unsafe stack is not in operational anymore

Figure 5.6: Different type of error signalled for the safety communication

- The error $SRDO\_RECEIVE\_ERROR$ is generated when the device receives an unexpected SRDO. For example, when it receives an old message . To simulate this error condition, a SRDO was sent in the interval between one SRDO and the correct subsequent one. This was achieved with 'PCAN' tool, which is a hardware and software interface that allows to write in a CAN network messages of variables length and addresses. In such way the device receive an unexpected message and consequently it is signalled the error.
- The error $SRDO\_SEND\_ERROR$ is created when the device sends a wrong SRDO since the message it is sending is corrupted.
- The error $SRDO\_DATA\_ERROR$ is created when the device receives a wrong SRDO, for example when the bit-wise inverted CAN message does not coincide with the expected message calculated from the first part of the SRDO. This error condition was simulated with the PCAN instrument by overwriting the first part of the SRDO.
- The error $SRDO\_INVALID\_CONFIGURATION$ is created when the devices that should communicate are not well configured. In this case the safety communication never starts until the devices are correctly reconfigured.
- The error $SRVT\_TIMEOUT$ is created when there is a delay, exceeding a fixed time, between the first CAN message that compose the SRDO and its subsequent bit-wise inverted CAN message. To impose this condition the CAN network was congested with many high priority messages sent with 'PCAN'. In particular a NMT message was sent every 1 millisecond.
- The error $SCT\_TIMEOUT$ is created when there is a delay, exceeding a fixed time, between two consecutive SRDOs. Since the SRDOs are periodic messages, the device sending them was switched off to create this error. In fact as soon the receiving device does not receive the expected SRDO within the fixed time, it signals this error. This error was also detected when the CAN network was congested, since not even the first part of the SRDO was received within the fixed time.
- The error $UNSAFE\_STACK\_NOT\_OPERATIONAL$ is created when not even the unsafe communication is guaranteed. Also this error was detected when the CAN network was congested. Indeed in this condition it may happen that the devices are not able to communicate. In this case the device signals for the safe communication the above error, while for the unsafe communication it signals the error $MODULE\_NOT\_FOUND$. Once the network is not congested any more, the unsafe communication is recovered, while the safe communication is still no available.

As expected, when one of all the above cited errors is detected, the communication is stopped, the safe configuration is sent, and a human action (the physical reboot of the devices) is necessary to established the transmission again. All these validation tests showed that the CANopen Safety protocol was implemented correctly, signalling errors and managing them in the right way.
After that, some further analyses concerning the timestamp were done, checking the results of the function that controls their correctness. When the transmission through radio misses a packet, the receiving unit sent, in the CAN network to the master, the previous message. For this reason the

timestamp test also checked that there were no more than 9 consecutive misses messages with the same timestamp, which would mean that the transmitter is not able to communicate with the receiver for a time that exceeds a fixed threshold. In this case the elaboration board signals the error and transmits a message to stop the machine in a safety condition. To check if these actions were done effectively, some possible cases were inspected. In particular it was analysed when the transmitter is correctly switched off, when the power supply is suddenly taken off and finally when the transmitter is too far from the receiver and so too many packets are lost. In all these cases all the messages sent by the receiver unit and from the master were tracked and analysed. It was seen that 9 consecutive messages were sent by the CRD and detected from the master with no error. At the 10th consecutive equal message, the master device signalled the error and sent the message to achieve the safe condition.
After that also some tests were made to verify the quality of the function to control the BCH and the UID. In particular with the Radio remote controller it was sent a message with the BCH bits calculated for a configuration which did not correspond to the effective one. When the telegram was inspected by the BCH test, as expected, the error was highlighted by the master and the safety configuration was sent. Finally a message with setted the wrong UID number, and so addressed to another device, was sent with the radio remote controller. Also in this case the function worked correctly, in fact the device signalled the error and it imposed the safety configuration.

5.2 Safety code validation example: function CRC_Calculation

It is now introduced the code of the $CRC\_Calculation$ and CRC_check functions and the safety validation of the code. Firstly it is presented what the CRC is, its goal and this particular calculation. After that, the code and the safety analysis are presented. A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective actions can be taken against data corruption. To compute an n-bit binary CRC, line the bits representing the input in a row, and position the (n + 1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row. The message to transmit is left shifted of n position where n is the length of the CRC (such that the resulting code word is in systematic form). After that it is computed the binary division module 2 between the shifted message and the fixed generator.
This operation is acted with an algorithm that:

1) Computes the bitwise XOR between:
   + the $n+1$ most significant bit of the message
   + the fixed generator.
2) Substitutes the $n+1$ most significant bit with the result of the previous operation.
3) Checks the new message length:
   + If it is composed by more than n bits, then it return to point 1).
   + Otherwise the message is the remainder which is also the searched CRC.

After the calculation of the CRC, the message to transmit is built by appending the calculated CRC to the initial message. The validity of a received message can easily be verified by performing, again, the previous calculation. The remainder should be equal to zero if there are no detectable errors.

Following it is presented an example of a CRC calculation:

$$initial\_message = 0011001000100100$$
$$generator = 10011$$
$$intial\_message\_shifted = 0011001000100100000$$
$$Division:$$
$$110010001001000000$$
$$10011$$
$$\overline{010100001001000000}$$
$$/10011$$
$$\overline{000111001001000000}$$
$$///10011$$
$$\overline{000011111001000000}$$
$$////10011$$
$$\overline{000001100001000000}$$
$$/////10011$$
$$\overline{000000101101000000}$$
$$//////10011$$
$$\overline{000000001011000000}$$
$$////////10011$$
$$\overline{000000000010100000}$$
$$/////////10011$$
$$\overline{000000000000111000}$$
$$//////////10011$$
$$\overline{000000000000011110}$$
$$///////////10011$$
$$\overline{000000000000001101}$$

In general, computation of CRC corresponds to Euclidean division of polynomials over Galois field of dimension 2. In particular for this field only two elements are defined which are nearly always called 0 and 1, while there are two permitted operations: addiction and multiplication which correspond to the XOR operation and the AND operation respectively. Given the original message polynomial $M(x)$ and the degree-n generator polynomial $G(x)$. The bits of $M(x) * x^n$ are the original message with n zeroes added at the end. The CRC 'checksum' is formed by the coefficients of the remainder polynomial $R(x)$ whose degree is strictly less than n. The quotient polynomial $Q(x)$ is of no interest. So resuming it holds:

$$M(x) * x^n = Q(x) * G(x) + R(x)$$

and using the modulo operation, it can be stated that:

$$R(x) = M(x) * x^n mod G(x)$$

In communication, the sender attaches the n bits of R after the original message bits of M, which could be shown to be equivalent to send out $M(x) * x^n - R(x)$ (the codeword.)
The receiver, knowing G(x) and therefore n, separates M from R and repeats the calculation, verifying that the received and computed R are equal. If they are, then the receiver assumes the received message bits are correct.

Explained the BCH goal and its calculation, it is now presented the code and its analysis for a possible safe validation.

```
FUNCTION_BLOCK CRC_Calculation
VAR_INPUT
        S_B1 : SAFEUSINT;
        S_B2 : SAFEUSINT;
        S_B3 : SAFEUSINT;
        S_B4 : SAFEUSINT;
        S_B5 : SAFEUSINT;
        S_B6 : SAFEUSINT;
        S_B7 : SAFEUSINT;
        S_B8 : SAFEUSINT;
END_VAR
VAR_OUTPUT
        S_CRC : SAFEULINT;
        S_ok : SAFEBOOL;
        S_outB1 : SAFEUSINT;
        S_outB2 : SAFEUSINT;
        S_outB3 : SAFEUSINT;
        S_outB4 : SAFEUSINT;
        S_outB5 : SAFEUSINT;
        S_outB6 : SAFEUSINT;
        S_outB7 : SAFEUSINT;
        S_outB8 : SAFEUSINT;
END_VAR
VAR
        S_initial : SAFEULINT;
        S_temp1 : SAFEULINT;
        S_temp2 : SAFEULINT;
        S_temp3 : SAFEULINT;
        S_temp4 : SAFEULINT;
        S_temp5 : SAFEULINT;
        S_temp6: SAFEULINT;
        S_temp7 : SAFEULINT;
        S_temp8 : SAFEULINT;
        S_generator : SAFEULINT;
        S_i : SAFEINT;
        S_trasm : SAFEULINT;
        S_temp : SAFEULINT;
        S_quoz : SAFEULINT;
        S_quot : SAFEULINT;
        S_shift1 : SAFEULINT;
        S_shift2 : SAFEULINT;
        S_shift3 : SAFEULINT;
        S_shift4 : SAFEULINT;
        S_shift5 : SAFEULINT;
        S_shift6 : SAFEULINT;
        S_shift7 : SAFEULINT;
        S_shiftTot :SAFEULINT;
        S_tempQuoz : SAFEULINT;
        S_tempBinDiv :SAFEULINT;
        S_shiftBinDiv:SAFEULINT;
        S_table : ARRAY[0..63] OF SAFEULINT := [
```

```
                16#1                    ,   16#2                    ,
                16#4                    ,   16#8                    ,
                16#10                   ,   16#20                   ,
                16#40                   ,   16#80                   ,
                16#100                  ,   16#200                  ,
                16#400                  ,   16#800                  ,
                16#1000                 ,   16#2000                 ,
                16#4000                 ,   16#8000                 ,
                16#10000                ,   16#20000                ,
                16#40000                ,   16#80000                ,
                16#100000               ,   16#200000               ,
                16#400000               ,   16#800000               ,
                16#1000000              ,   16#2000000              ,
                16#4000000              ,   16#8000000              ,
                16#10000000             ,   16#20000000             ,
                16#40000000             ,   16#80000000             ,
                16#100000000            ,   16#200000000            ,
                16#400000000            ,   16#800000000            ,
                16#1000000000           ,   16#2000000000           ,
                16#4000000000           ,   16#8000000000           ,
                16#10000000000          ,   16#20000000000          ,
                16#40000000000          ,   16#80000000000          ,
                16#100000000000         ,   16#200000000000         ,
                16#400000000000         ,   16#800000000000         ,
                16#1000000000000        ,   16#2000000000000        ,
                16#4000000000000        ,   16#8000000000000        ,
                16#10000000000000       ,   16#20000000000000       ,
                16#40000000000000       ,   16#80000000000000       ,
                16#100000000000000      ,   16#200000000000000      ,
                16#400000000000000      ,   16#800000000000000      ,
                16#1000000000000000     ,   16#2000000000000000     ,
                16#4000000000000000     ,   16#8000000000000000     ];
END_VAR

// CRC_Calculation
// calculate the data to trasmitt(8bytes) and the relative CRC
// b(x) polynomial associated with the data
// g(x) polynomial of the generator of degree g
// generator chosen  16#8005, associated  polynomial : x^16+x^15+x^2+1
// p(x) = b(x) * 2^g;
// p(x) : g(x) = q(x) + r(x) ----> CRC = r(x) doing binary division mod2
// data to trasmitt m(x) = p(x) + r(x)
// no corruption of the data after the trasmission if
// the trasmitted message has a remainder equal to 0 (doing binary
   division mod2)


//Calculation of the safeulint data in a safety way
S_shift1 :=  S_table[56];
S_temp1 := SAFEUSINT_TO_SAFEULINT(S_B1);
S_temp1 := S_temp1*S_shift1;
S_initial := S_temp1;
S_shift2 :=S_table[48];
```

```
104 S_temp2 := SAFEUSINT_TO_SAFEULINT(S_B2);
105 S_temp2 := S_temp2*S_shift2;
106 S_initial := S_initial + S_temp2;
107 S_shift3 := S_table[40];
108 S_temp3 := SAFEUSINT_TO_SAFEULINT(S_B3);
109 S_temp3 := S_temp3*S_shift3;
110 S_initial := S_initial + S_temp3;
111 S_shift4 := S_table[32];
112 S_temp4 := SAFEUSINT_TO_SAFEULINT(S_B4);
113 S_temp4 := S_temp4*S_shift4;
114 S_initial := S_initial + S_temp4;
115 S_shift5 := S_table[24];
116 S_temp5 := SAFEUSINT_TO_SAFEULINT(S_B5);
117 S_temp5 := S_temp5*S_shift5;
118 S_initial := S_initial + S_temp5;
119 S_shift6 := S_table[16];
120 S_temp6 := SAFEUSINT_TO_SAFEULINT(S_B6);
121 S_temp6 := S_temp6*S_shift6;
122 S_initial := S_initial + S_temp6;
123 S_shift7 := S_table[8];
124 S_temp7 := SAFEUSINT_TO_SAFEULINT(S_B7);
125 S_temp7 := S_temp7*S_shift7;
126 S_initial := S_initial + S_temp7;
127 S_temp8 := SAFEUSINT_TO_SAFEULINT(S_B8);
128 S_initial := S_initial + S_temp8;
129
130 // Traslation of the data of g position where g is the degree of the
    generator polynomial
131 S_shiftTot := S_table[15];
132 S_initial := S_initial*S_shiftTot;
133 S_trasm := S_initial;
134 // Calculation of the CRC as the remainder of the binary division mod 2,
    calculation using the safety specification
135 FOR S_i:= 0 TO 48 DO
136         //Selected generator polynomial
137         S_generator := 16#8005;
138         S_tempBinDiv:=16#8000;
139         S_shiftBinDiv := S_table[48-S_i];
140         S_tempBinDiv:=S_tempBinDiv*S_shiftBinDiv;
141         S_tempBinDiv:= S_tempBinDiv AND S_initial;
142         IF NOT (S_tempBinDiv= 0) THEN
143                 S_generator :=S_generator*S_shiftBinDiv;
144                 S_initial := S_initial XOR S_generator;
145         END_IF
146 END_FOR
147 S_CRC:= S_initial;
148 S_trasm := S_trasm + S_CRC;
```

```
FUNCTION_BLOCK Check_CRC
VAR_INPUT
        S_B1 : SAFEUSINT;
        S_B2 : SAFEUSINT;
        S_B3 : SAFEUSINT;
        S_B4 : SAFEUSINT;
        S_B5 : SAFEUSINT;
        S_B6 : SAFEUSINT;
        S_B7 : SAFEUSINT;
        S_B8 : SAFEUSINT;
END_VAR
VAR_OUTPUT
        S_ok:SAFEBOOL;
END_VAR
VAR
        S_initial : SAFEULINT;
        S_temp1 : SAFEULINT;
        S_temp2 : SAFEULINT;
        S_temp3 : SAFEULINT;
        S_temp4 : SAFEULINT;
        S_temp5 : SAFEULINT;
        S_temp6: SAFEULINT;
        S_temp7 : SAFEULINT;
        S_temp8 : SAFEULINT;
        S_generator : SAFEULINT;
        S_i : SAFEINT;
        S_trasm : SAFEULINT;
        S_temp : SAFEULINT;
        S_quoz : SAFEULINT;
        S_quot : SAFEULINT;
        S_shift1 : SAFEULINT;
        S_shift2 : SAFEULINT;
        S_shift3 : SAFEULINT;
        S_shift4 : SAFEULINT;
        S_shift5 : SAFEULINT;
        S_shift6 : SAFEULINT;
        S_shift7 : SAFEULINT;
        S_shiftTot :SAFEULINT;
        S_tempQuoz : SAFEULINT;
        S_tempBinDiv :SAFEULINT;
        S_shiftBinDiv:SAFEULINT;
        S_table : ARRAY[0..63] OF SAFEULINT := [
                16#1                    ,       16#2                    ,
                16#4                    ,       16#8                    ,
                16#10                   ,       16#20                   ,
                16#40                   ,       16#80                   ,
                16#100                  ,       16#200                  ,
                16#400                  ,       16#800                  ,
                16#1000                 ,       16#2000                 ,
                16#4000                 ,       16#8000                 ,
                16#10000                ,       16#20000                ,
                16#40000                ,       16#80000                ,
```

```
                    16#100000             ,     16#200000             ,
                    16#400000             ,     16#800000             ,
                    16#1000000            ,     16#2000000            ,
                    16#4000000            ,     16#8000000            ,
                    16#10000000           ,     16#20000000           ,
                    16#40000000           ,     16#80000000           ,
                    16#100000000          ,     16#200000000          ,
                    16#400000000          ,     16#800000000          ,
                    16#1000000000         ,     16#2000000000         ,
                    16#4000000000         ,     16#8000000000         ,
                    16#10000000000        ,     16#20000000000        ,
                    16#40000000000        ,     16#80000000000        ,
                    16#100000000000       ,     16#200000000000       ,
                    16#400000000000       ,     16#800000000000       ,
                    16#1000000000000      ,     16#2000000000000      ,
                    16#4000000000000      ,     16#8000000000000      ,
                    16#10000000000000     ,     16#20000000000000     ,
                    16#40000000000000     ,     16#80000000000000     ,
                    16#100000000000000    ,     16#200000000000000    ,
                    16#400000000000000    ,     16#800000000000000    ,
                    16#1000000000000000   ,     16#2000000000000000   ,
                    16#4000000000000000   ,     16#8000000000000000   ];
END_VAR

// Check_CRC
// check of the received message
// binary division mod 2 of the received message for the fixed generator
// the message is correct if the remainder is equal to 0

//Calculation of the safeulint data in a safety way
S_shift1 :=  S_table[56];
S_temp1 := SAFEUSINT_TO_SAFEULINT(S_B1);
S_temp1 := S_temp1*S_shift1;
S_initial := S_temp1;
S_shift2 :=S_table[48];
S_temp2 := SAFEUSINT_TO_SAFEULINT(S_B2);
S_temp2 := S_temp2*S_shift2;
S_initial := S_initial + S_temp2;
S_shift3 := S_table[40];
S_temp3 := SAFEUSINT_TO_SAFEULINT(S_B3);
S_temp3 := S_temp3*S_shift3;
S_initial := S_initial + S_temp3;
S_shift4 := S_table[32];
S_temp4 := SAFEUSINT_TO_SAFEULINT(S_B4);
S_temp4 := S_temp4*S_shift4;
S_initial := S_initial + S_temp4;
S_shift5 := S_table[24];
S_temp5 := SAFEUSINT_TO_SAFEULINT(S_B5);
S_temp5 := S_temp5*S_shift5;
S_initial := S_initial + S_temp5;
S_shift6 := S_table[16];
S_temp6 := SAFEUSINT_TO_SAFEULINT(S_B6);
S_temp6 := S_temp6*S_shift6;
```

```
106 S_initial := S_initial + S_temp6;
107 S_shift7 := S_table[8];
108 S_temp7 := SAFEUSINT_TO_SAFEULINT(S_B7);
109 S_temp7 := S_temp7*S_shift7;
110 S_initial := S_initial + S_temp7;
111 S_temp8 := SAFEUSINT_TO_SAFEULINT(S_B8);
112 S_initial := S_initial + S_temp8;
113 FOR S_i:= 0 TO 48 DO
114         //Selected generator polynomial
115         S_generator := 16#8005;
116         S_tempBinDiv:=16#8000;
117         S_shiftBinDiv := S_table[48-S_i];
118         S_tempBinDiv:=S_tempBinDiv*S_shiftBinDiv;
119         S_tempBinDiv:= S_tempBinDiv AND S_initial;
120         IF NOT (S_tempBinDiv= 0) THEN
121                 S_generator := S_generator*S_shiftBinDiv;
122                 S_initial := S_initial XOR S_generator;
123         END_IF
124 END_FOR
125 S_ok:= (S_initial=0);
```

The above code is written in Structured Text (defined by IEC 61131-3, FVL language) respecting all the Codesys Safety SIL2 User Manual at the level extended. Some strong constraints are imposed by the manual. For example, it is not possible to use the function to shift the bits of a variable. To obtain this effect, firstly a table was built in which all the possible power of 2 obtainable using 64 bits were reported, and then the variable was multiplied or divided by the i- element of the table to obtain a left or right i- shift respectively. The possibly exceeded numbers were rejected and everything was managed in a safety way also since safe variables were used. Moreover the table was also used to avoid to use the function EXPT for the power of 2 whose use is not allowed by the manual.
A particular explanation is needed also for the binary division MOD 2, which was implemented in a different way with respect to the original version, because of the constraints imposed by the manual and by the structured text. Indeed, it is not possible to:

- use WHILE cycles
- use FOR cycles with no constant number
- access a single bit of a variable in a cyclic way

then to find the most significant bit an AND operation was executed between the message of interest and a variable which was imposed with all 0 except for the (64-j) bit which is imposed as one. The result is different form 0 if the (64-j) bit was 1 also in the message, while is equal to 0 otherwise. Once found the most significant bit of the message, the divisor is shifted such that its most significant bit has the same position of the one of the message of interest. Then it is done the XOR operation between these two messages and all these operations were repeated until the message was smaller than the divisor and so it coincided with the CRC.
All the constraints that concern the implemented code are reported in the table 5.1 with the corresponding technique used to respect it.

Rule	Technique to respect it
Safe input and output variables not beginning with the prefix $S\_$ are not permitted in FBs	All safe input and output were declared with $S\_$.
Safe, explicitly defined, local variables not beginning with the Prefix $S\_$ are not permitted.	All variables were declared with $S\_$.
Safe data that have not been declared with a SAFE data type are not permitted.	All safe data were declared with safe type.
Arrays are not permitted without explicit range check.	All the arrays were declared with a constant length; all the accesses were done either with constant variables or with a variable which could change to a maximum constant number which was in any case smaller than the fixed limit.
Only libraries whose "SIL2" property is set may be used in the application.	Only libraries whit setted "SIL2" property were used.
POU without a visible description of the functional specification is not permitted.	Every POU was commented with input, output , functional specification description.
Implicit type conversions within expressions are not permitted.	Every conversion was explicit.
FOR loops with no constant number of loop executions per cycle within a state are not permitted.	For loops were used only with constant number of loop.
WHILE loops are not permitted.	While loops were not used.
expressions with a maximum nesting depth greater than 1 are not permitted.	Every expression was subdivided in smaller sub-expressions of depth 1, which was executed one by one.
Calculation of AND or OR with more than one SAFEBOOL output is not permitted.	Every Safe output was calculated independently from the other.
Calculation of a SAFEBOOL output without at least one SAFEBOOL input is not permitted.	every safebool was calculated using safe data.
MOD use is not permitted unless explicitly permitted by the controller manufacturer.	the remainder was calculated subtracting the quotient multiplied for the divisor to the dividend.
EXPT use is not permitted unless explicitly permitted by the controller manufacturer.	since only power of 2 were used, it was built in advance a table with all the possible power of 2 for a ulint.
SHL use is not permitted.	to obtain a left shift of n position, the number was multiplied for the n power of 2.
SHR use is not permitted.	to obtain a right shift of n position, the number was divided for the n power of 2.

Table 5.1: Rules Codesys Safety SIL2 at level extended and their satisfaction

Chapter 6

Conclusion

This thesis introduced the principal characteristics of the most important industrial networks, the relative most important industrial networks' protocols and finally the corresponding safety protocol with all the different techniques to obtain a safety transmission. In such way it is possible to understand how an industrial network works, which requirements are needed and in which way they are achieved. The study of safety protocols allows to understand all the difficulties and all the problem that can arise when it is necessary to ensure the correctness of a transmission with a given probability. The different safety protocol have strategies to ensure the effective safety, which can be completely different and so many methods were analysed. For example, as seen, CANopen Safety provide to consider a message as the effectively message and its bitwise inverted message. Using a CAN message for each part, the 5 methods typical of CAN to check the corruption of the message are used. In such way for a single SRDO there are twice all these controls, besides there is the check of the effectively bit-wise inverted bit and finally there are 2 timers to check possible timeout between the first part and the second one and between two consecutive SRDOs. Instead, for example, FSOE uses a bigger CRC(of 32 bit) calculated with a different algorithm to detect corruption, a session number to verify the correct configuration, a unique connection id to avoid misrouting, a sequence number to detect repetition,loss or insertion and finally a watchdog to detect possible delay and timeout. Without using the protocol methods, also Autec use different way to ensure that the transmission of the radio was successfully in particular an algorithm is used to analyse the data received, besides a CRC is used and the UID is checked. Once understood all the difficulties of ensuring a safety transmission some safety applications were developed and the foundations for their safety validation were laid. But all these operations and all these methods are only a small part of the entire world of safety. In fact there are many aspects which needed to be analysed that go from the safety of the hardware system to software management of all possible failures. For this reason this thesis wants to be only a starting point for the future works that will be done with thePh.D. 'in Alto Apprendistato' always in collaboration with Autec Srl.

Appendix A

Example of CANopen Safety chip

A Structure and Functionality of the CSC(CANopen Safety Chip)

The CSC (CANopen Safety Chip) is composed by a 16-bit microcontroller and by two internal CAN controllers that are used redundantly. Physical connection to the CAN bus is implemented by routing signals from both on-chip CAN controllers to a single CAN transceiver and applicable EMI protection circuitry. The microcontroller operates in single-chip mode and runs with a 16 MHz internal system clock. It provides 10 kByte internal SRAM and 256 kByte Flash memory and it is available a wide selection of on-chip peripheral modules (e.g. ADC, DAC, DMA, Timer, ports, serial synchronous and asynchronous interfaces). The software of the CSC consists in:

- the permanent firmware, which contains the CANopen safety protocol stack with all safety relevant field bus functions, the diagnostic functions for RAM, register, stack, Flash and the watchdog functions with monitoring.
- the variable safety application which is located in the second, variable Flash sector and it is loaded into the Flash by the user at a later time.

The permanent firmware controls all processes in the CSC. Thus the conditions for transfer of an SRDO are verified among others. The test controls whether the SCT has elapsed and whether the data and the configuration are valid and after that the SRDO is assembled and transferred. Upon receipt of an SRDO the received CAN message is examined as well:
- if the SRDO is valid, the safety application will be notified of the receipt in two ways (call of an event function and status in the so called safety relevant RAM);
- if a faulty SRDO was received or if no valid SRDO could be received in the expected time, the safety application will be notified as well. The safety application has to initiate the corresponding error response. Error responses to the SRDO enable targeted action to certain parts of the device which can then be rendered into a secure state. Other functional units not effected by the local error are not influenced. The CSC provides 4 SRDOs to ensure secure data transfer. Two SRDOs are reserved for the transmission of safety relevant data and two are reserved for receiving data. There are 128 bits of data (organized bytewise) available in both send and receive directions. The data structure in the SRDO is static and can be set by the user during programming. While the entry of safety relevant variables in the Object Dictionary can be performed by the user during program development. Beside the CANopen safety services, there are integrated in the CSC the following CANopen standard functions :

- 2 transmit and 2 receive PDOs (process data object), PDO linking, static PDO mapping, synchronous and asynchronous transfer
- 1 SDO Server (service data object), expedited and segmented transfer
- NMT Slave (network management)

- Heartbeat Producer
- Emergency Producer

The CSC is specified and certified for use in devices according to IEC61508 up to SIL3 (safety integrity level). Since the CSC consists of a microcontroller with redundant safety structures, a second shut-off path is required to set the device into a secure state if the microcontroller fails. According to IEC61508 such field bus devices are ranged in the class of highly available devices, which require a degree of diagnostic coverage . The diagnostic test interval must be correspondingly smaller than the safety cycle time, which is the maximum required time between the recognition of a safety relevant event and the electrical initialization of the corresponding safety response. In particular for the CSC the safety cycle time was set to $<= 20ms$. The secure and timely discovery of errors therefore takes on special importance. The diagnostic routines of the CSC are an important part of the permanent firmware, infact they determine directly the time performance and resource requirements of the chip. Time intensive diagnostics include the calculation of a 16-bit CRC across program memory as well as the diagnosis of the RAM. The algorithms used for it determine crucially the usable size of the Flash and RAM. The permanent firmware checks the 5 kByte RAM via a transparent GALPAT test and the 42 kByte program memory (permanent firmware and safety application) via the 16-bit CRC. Errors recognized by the diagnostic function are treated as severe safety related errors. For a sensor device this means transfer of SRDOs is stopped immediately. The CSC is now in an intrinsically secure state that cannot be exited. In an actuator application this will lead to release of the external Watchdog, which sets the actuator into the secure state via the secondary shut-off path. The CSC contains a logical program execution monitor, which is tested by the permanent firmware. The safety application is also integrated in the program process monitoring. Errors that are recognized by the program execution monitor, are considered the same as diagnostic errors and cause a change into an intrinsically secure state. The additional temporal program execution monitor function comes into use when the CSC is implemented in actuators and, so, a Watchdog with an external time base and a time window is used. Errors in the temporal program execution effect the actors via the secondary shut-off path. The user is solely responsible for the safety application. It serves as the general function and data interfaces of the permanent software and is used for additional data processing according to the required functionality. Parameters for initialization of the CANopen stack are handed over in the application. The applied periphery, which is not submitted to the diagnosis of the permanent firmware, has to be diagnosed in the safety application. If there is no safety application running on the CSC, then this will cause an error in the monitoring of the logical program execution and result in a change to an intrinsically secure state. Besides a variable user software enables direct integration of simple applications (e.g. emergency stop device, emergency monitoring relay, safety valve) in the CSC, whereby costs can be minimized. Instead in more complex applications (such as safety related drives, safety light curtains, laser scanners) the CSC can function as dedicated bus interface and communicate with other hosts or microcontrollers. In such cases the safety application provides secure transfer of safety related data to and from the superordinate modules. The interface can be configured freely by the device developer for which is available the entire free CSC periphery. This includes the synchronous and asynchronous (UART) serial interfaces or parallel interfaces via the freely available ports. In particular a total of 70 free port pins are available to the user and some of them have alternative functions. If the chip is used as a bus interface, safety critical errors are sent from the safety application to the superordinate unit as a binary shut-off signal. The superordinate CPU performs the shut-off itself. For example, the CPU can transfer the drive to a secure location with a predefined function and thereby bring the entire system into an intrinsically secure state.

B Structure of the function interface between the permanent firmware and the safety application

Configuration of the permanent firmware (e.g. the CANopen stack), signalization of events (e.g. SRDO received, diagnostic error recognized, send GFC, GFC received etc.) and the call of the

safety application all occur via a program and data interface. Since both software parts are developed separately, they can not be linked with each other via the development environment's linker. Interfaces are required for an alternating call. The interface is defined by a jump table for the function call and data memory for parameter transmission. Data access occurs with fixed addresses that are known by both software parts. This call mechanism via fixed data structures is identified as "Callgate". In particular it makes functions available for the following mechanisms:

1. The permanent firmware calls the safety application functions.
2. The safety application calls permanent firmware functions or CANopen stack functions respectively.
3. Data exchange between the application and the permanent firmware that are components of the Object Dictionary.
4. Data exchange between the application and the permanent firmware that are not components of the Object Dictionary.

The transmission of safety related data in redundant memory occurs over fixed structures that are recognized by both parts of the software. This mechanism enables an efficient use of data and minimization of resource requirements. Recopying the data during the transmission between modules is not required. The main Callgate functions are now introduced:

- Appinitialisation() : This function is called by the permanent firmware in order to initialize the application. Within the function, the safety application initializes its own global and local variables, sets the CANopen node number and defines and registers its own sections of the Object Dictionary.
- AppProcess(): This function is called cyclically by the permanent firmware; the application executes its own cyclical processes in this function.
- AppPdoEvent(): signals the transmission or receipt of a PDO.
- AppSrdoEvent(): This function is called if an SRDO was sent or received without error.
- AppGfcEvent(): signals the receipt of a GFC
- AppSrdoError(): This function is called if an error occurred during the receipt of an SRDO, or if a send-SRDO could not be sent within the refresh time. The safety application confirms the successful processing of the event. If the processing is not confirmed then the permanent firmware will recognize this as a safety error and change to an intrinsically secure state.
- AppStopWatchdog(): This function is called by the permanent firmware if the application's Watchdog can no longer be used (e.g. due to a CAN error or diagnostic error). This

Besides are now introduced the functions for the safety goal:

- CscSetNodeId(): configures the node number to be used
- CscDefineVariable(): defines a variable (for object entries of variable length, e.g. manufacture device name, index 0x1008 according to DS 301); is used for variables that are initialized in the safety application, whereby the object entry is located in the CANopen stack‘s permanent Object Dictionary.
- CscDefineVarTab(): defines a variable block for representation in the Object Dictionary (for objects of a fixed length, e.g process variables)
- CscRegisterOdPart(): registers the user specific part of the Object Dictionary in the CANopen stack of the permanent firmware.
- CscWriteObject(): writes a value in an Object Dictionary entry
- CSCReadObject(): Reads a value from an Object Dictionary entry

- CSCSendEmergency(): sends an emergency message to the CAN bus
- CSCSendGfc(): sends a GFC

The microcontroller provides a series of interrupt sources. All interrupt sources that are not used by the permanent firmware can be used by the safety application. The function interface makes a mechanism available that allows interrupt service routines to be assigned to corresponding interrupt vectors. All unused interrupts are processed by a standard interrupt handler.

C Requirements

The CSC is designed to maintain a security cycle time of 20ms with the resources used. Dure to the time intensive diagnostic routines (see above), not all of the microcontroller's RAM and Flash is available for the CSC. The permanent firmware requires 32 kByte Flash and approximately 2 kByte RAM. Furthermore, 512 Byte RAM are required for the system stack. Approximately 10 kByte Flash and 2.5 kByte RAM are available for the safety application. All safety relevant data that is transferred per SRDO is a component of the permanent firmware and is used by the application as shared memory. The system stack is also used equally by both software parts.

Appendix B

International standard and regulations

In most countries, national laws regulate how people and the environment shall be protected. In Europe for instance, the "Low Voltage Directive", the "EMC Directive", and the "Machinery Directive" are examples of such legislation. The laws in turn refer to approved International Standards. The basic standard for functional safety is the IEC 61508 covering the functional safety of electrical equipment and the basic principles and procedures. It introduces a quantitative approach for calculating the residual probability of so-called safety functions to fail (Safety Integrity Levels - SIL). It is mainly useful for safety device and safety controller developers. The sector standard IEC 62061 describes the specific safety aspects for machinery applications such as those found in factory automation. This standard deals with ready-to-use systems, subsystems, and elements and how to assess safety functions for certain combinations of these. ISO 13849-1 is the successor of the EN 954-1 (withdrawn in 2011) and has a similar scope. However, it introduces a slightly different calculation model (Performance Levels - PL) and covers non-electrical devices such as hydraulic valves, etc. For safety of machinery, the basic terminology and principles for design are defined in ISO 12100 as well as risk assessment and risk reduction. The IEC 60204-1 specifies general requirements and recommendations relating to the electrical equipment of machines. Some of the issues are power supply, protection against electrical shock, emergency stops, conductors and cables, etc. Product standards such as IEC 61496, IEC 61800-5-2, IEC 61131-6 and ISO 10218-1 for example, deal with the requirements for individual product families. The requirements for F-Devices and F-Hosts to provide increased electromagnetic immunity are defined within the generic IEC 61000-6-7 and in sector standard IEC 61326-3-1. Special performance criteria DS ("defined state") allow for incorrect functioning under increased electromagnetic interference conditions above the normally required levels. However, in these cases the equipment under test (EUT) shall go at least into a safe state. The fieldbus standards are specified in IEC 61158 and IEC 61784-1. Real time Ethernet variants are defined in IEC 61784-2. Common parts for installation guidelines are summed up in IEC 61918, whereas profile-specific parts are collected in IEC 61784-5. Common parts for security guidelines are summed up in IEC 62443, whereas profile-specific parts are planned for a future IEC 61784-4.

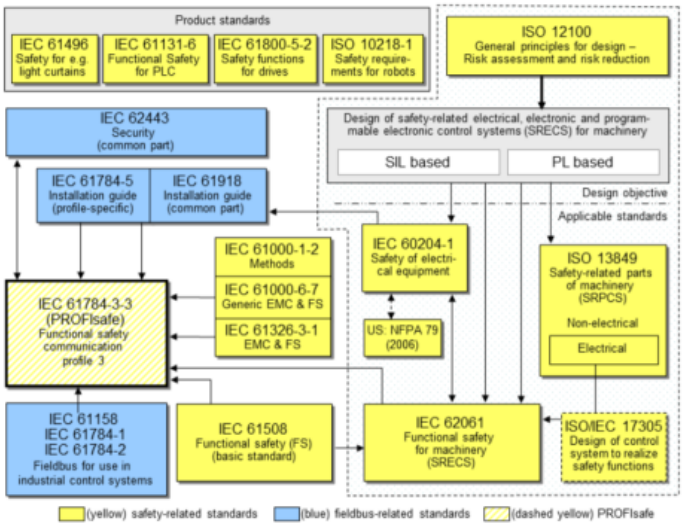


Figure B.1: International fieldbus and safety standards

As instead regarding the regulations of PLC, IEC 61131 provides a standardization for them and in particular:

- IEC 61131-1 define the PLC device;
- IEC 61131-2 define the architecture hardware and software of PLC;
- IEC 61131-3 define the languages to implement sequence control over PLC;
- IEC 61131-4 define the guidelines for the user;
- IEC 61131-5 define the messages service;
- IEC 61131-6 define the communication through the fieldbus;
- IEC 61131-7 define the control coding with fuzzy logic
- IEC 61131-8 define the guidelines to use and develop code languages.

As it can be seen in previous section, Codesys is especially regulated by IEC 61131-3 and so it is interesting to deepen this regulation and in particular the languages that it defines. Since there are many different PLC producers a big complication for a PLC programmer is the differences between the languages. For this reason IEC 61131-3 defined five different standard languages:

- Instruction List(IL): textual language of low level used for small applications or to optimize part of applications;
- Sequential Function Chart(SFC): graphical language used to partition in a sequence of consecutive state the execution of the application;
- Structured TExt(ST): textual structure language of high level, very similar to C. It is usually used to implement the complex part of application which are difficult to implement with a graphic language;
- Ladder Diagram(LD): graphic language which implement a boolean logic using electrical scheme;
- Function Block Diagram(FBD): graphical language, based on the interpretation of the system behaviour through flow of signal between the process elements.

As regarding regulations of remote radio controllers, the most important are:

- EN300-220-2 (radio): Short Range Devices (SRD) operating in the frequency range 25 MHz to 1 000 MHz. Part 2: Harmonised Standard covering the essential requirements of article 3.2 of Directive 2014/53/EU for non specific radio equipment;
- EN301-489-3: EMC standard for radio equipment and services.; EN61000-6-2, -6-3: Electromagnetic Compatibility;
- EN60950: Equipment for safety information technology; IEC 62479:2010: Assessment of the compliance of low power electronic and electrical equipment with the basic restrictions related to human exposure to electromagnetic fields (10 MHz to 300 GHz);
- Comitato Elettrotecnico Italiano (CEI) EN60204-32: Safety equipment
- CEI EN50178: Electronic instrument for power plants ;
- EN13557: Cranes - Controls and Control Stations;

Finally it is presented the most important point of the, already cited, IEC 62061 which is the regulation to follow to certificate the Codesys Safety Sil 2 code according to the manual at the level extended. This regulation is composed by 9 articles:

1. Scope;
2. Normative references;
3. Terms, definitions and abbreviations;
4. Management of functional safety;
5. Requirements for the specification of safety-related control functions;
6. Design and integration of the safety-related electrical control system;
7. Information for use of the machine;
8. Validation of the safety-related electrical control system;
9. Modification of the safety-related electrical control system;

The principal points of the main articles are described following. The article $n_r 4$ affirm that a functional safety plan should bee drawn up and documented for each Safety-Related Electrical Control System (SRECS) design projects. In particular the plan should:

- identify the relevant activities;
- describe the policy and the strategies to fulfil the specified functional safety requirements;
- describe the strategy to achieve functional safety for the application software, development, integration, verification and validation.
- identify persons, departments or other units and resources that are responsible for carrying out and reviewing each of the relevant activities;
- identify or establish the procedures and resources to record and maintain information relevant to the functional safety if the SRECS;
- describe the strategy for configuration management taking into account relevant organization issues, such as authorized persons and internal structures of the organization.
- establish a verification plan
- establish a validation plan.

The main information of article n_r5 concern the all the specification that must be available for a SRECS. In particular:

- the results of the risk assessment for the machine including all safety functions determined to be necessary for the risk reduction process for each specific hazard.
- the machine operating characteristics, including: modes of operation, cycle time, response time performance, environmental conditions, interaction of person with the machine
- all the information relevant to the Safety-Related Control Function(SRCF) which can have a influence on the SRECS design including:
  - a description of the behaviour of the machine that a SRCF is intended to achieve or to prevent;
  - all interfaces between the SRCFs, and between SRCFs and any other function;
  - required fault reaction functions of the SRCF.

The main information of article n_r6 concern the all the steps that must be followed during the development of a SRECS. There are a lot of requirements:

- some general requirements
- for behaviour of the SRECS on detection of a fault in the SRECS
- for systematic safety integrity of the SRECS
- for the selection of safety-related electrical control system
- for safety-related electrical control system (SRECS) and in particular for its design and development, which include all the rule to follow during the designing of hardware and software part of a SRECS.

All the detailed are described in the already cited article of the regulation, which is suggested to be read. In particular as concerning the software design for FVL language this regulation sends to the IEC 61508-3.
For this reason, and since structured text language (which is a FVL language) was used, it is now described the main points of the cited regulation. In particular the regulation affirm that:

- the design method chosen shall possess features that facilitate:
  - abstraction, modularity and other features which control complexity;
  - the expression of:
    1. functionality;
    2. information flow between elements;
    3. sequencing and time related information;
    4. timing constraints;
    5. concurrency and synchronized access to shared resources;
    6. data structures and their properties;
    7. design assumptions and their dependencies;
    8. exception handling;
    9. design assumptions (pre-conditions, post-conditions, invariants);
    10. comments.
  - ability to represent several views of the design including structural and behavioural views;

  - comprehension by developers and others who need to understand the design;
  - verification and validation.
- The software design shall include, commensurate with the required safety integrity level, self-monitoring of control flow and data flow. On failure detection, appropriate actions shall be taken.
- Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate design measures ensure that the failures of non-safety functions cannot adversely affect safety functions.
- Where a pre-existing software element is reused to implement all or part of a safety function, the element shall meet both the requirements below for systematic safety integrity:
  1. meet the requirements of one of the following compliance routes:
     - Route 1S: compliant development. Compliance with the requirements of this standard for the avoidance and control of systematic faults in software;
     - Route 2S: proven in use. Provide evidence that the element is proven in use;
     - Route 3S:assessment of non-compliant development.
  2. provide a safety manual that gives a sufficiently precise and complete description of the element to make possible an assessment of the integrity of a specific safety function that depends wholly or partly on the pre-existing software element.
- The software architecture defines the major elements and subsystems of the software, how they are interconnected, and how the required attributes, particularly safety integrity, will be achieved. It also defines the overall behaviour of the software, and how software elements interface and interact
- The software architecture design shall:
  - select and justify an integrated set of techniques and measures necessary during the software safety lifecycle phases to satisfy the software safety requirements specification at the required safety integrity level. These techniques and measures include software design strategies for both fault tolerance (consistent with the hardware) and fault avoidance, including (where appropriate) redundancy and diversity;
  - be based on a partitioning into elements/subsystems, for each of which the following information shall be provided:
    1. whether the elements/subsystems have been previously verified, and if yes, their verification conditions;
    2. whether each subsystem/element is safety-related or not;
    3. software systematic capability of the subsystem/element.
  - determine all software/hardware interactions and evaluate and detail their significance;
  - use a notation to represent the architecture which is unambiguously defined or restricted to unambiguously defined features;
  - select the design features to be used for maintaining the safety integrity of all data. Such data may include plant input-output data, communications data, operator interface data, maintenance data and internal database data;
  - specify appropriate software architecture integration tests to ensure that the software architecture satisfies the software safety requirements specification at the required safety integrity level.

The principal points of the IEC 61508-3 which could concern the development of the defined application have been highlighted, but the above points are further from resuming the complete regulation. For this reason for further details it is suggested to read the regulation. Finally in the following table (tab . B.1) are resumed al the cited regulations and the relative argument they regulate.

Regulation	Argument
IEC 61508	functional safety of electrical equipment and the basic principles and procedures
IEC 62061	specific safety aspects for machinery applications
ISO 13849-1	Machine Safety with the definition of Performance Levels - PL model
ISO 12100	risk assessment and risk reduction
IEC 60204-1	general requirements and recommendations relating to the electrical equipment of machines
IEC 61496, IEC 61800-5-2 IEC 61131-6 and ISO 10218-1	requirements for individual product families
IEC61000-6-7	requirements for F-Devices and F-Hosts to provide increased electromagnetic immunity
IEC61158 and IEC61784-1	fieldbus standards
IEC 61784-2	Real-time Ethernet variants
IEC 61918	installation guidelines
IEC 62443	security guidelines
IEC 61131	PIC regulations
IEC 61131-1	PLC device
IEC 61131-2	architecture hardware and software of PLC
IEC 61131-3	languages to implement sequence control over PLC
IEC 61131-4	guidelines for the user
IEC 61131-5	messages service
IEC 61131-6	communication through the fieldbus
IEC 61131-7	control coding with fuzzy logic
IEC 61131-8	guidelines to use and develop code languages
EN300-220-2	requirements for non specific radio equipment
EN301-489-3	standard for radio equipment and services
EN61000-6-2, -6-3	Electromagnetic Compatibility
EN60950	Equipment for safety information technology
IEC 62479:2010	Assessment of the compliance of low power electronic and electrical equipment with the basic restrictions related to human exposure to electromagnetic fields
CEI EN60204-32	Safety equipment
CEI EN50178	Electronic instrument for power plants
EN13557	Cranes - Controls and Control Stations

Table B.1: Most important regulations for Protocols, PLC , Remote Controller and for a safety validation

Appendix C

Hercules Safety TMS570 MCU by Texas Instrument

The TMS570LS series is a high performance automotive grade microcontroller family. The safety architecture includes Dual CPUs in lockstep, CPU and Memory Built-In Self Test (BIST) logic, ECC on both the Flash and the data SRAM, parity on peripheral memories, and loop back capability on peripheral IOs. The TMS570LS family integrates the ARM Cortex-R4F Floating Point CPU which offers an efficient 1.6 DMIPS/MHz, and has configurations which can run up to 160 MHz providing more than 250 DMIPS. The TMS570LS series also provides different Flash (1MB or 2MB) and data SRAM (128KB or 160KB) options with single bit error correction and double bit error detection. The TMS570LS devices feature peripherals for real-time control-based applications, including up to 32 nHET timer channels and two 12-bit A to D converters supporting up to 24 inputs. There are multiple communication interfaces including a 2-channel FlexRay, 3 CAN controllers supporting 64 mailboxes each, and 2 LIN/UART controllers. With integrated safety features and a wide choice of communication and control peripherals, the TMS570LS series is an ideal solution for high performance real time control applications with safety critical requirements.
The devices utilize the big-endian format where the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte.
The device memory includes general-purpose SRAM supporting single-cycle read/write accesses in byte, halfword, and word modes. The flash memory on this device is a not volatile, electrically erasable and programmable memory implemented with a 64-bit-wide data bus interface. The flash operates on a 3.3V supply input (same level as I/O supply) for all read, program and erase operations. When in pipeline mode, the flash operates with a system clock frequency of up to 160 MHz.
The device has nine communication interfaces: three MibSPIs, two LIN/SCIs, three DCANs and one FlexRay controller (optional). The SPI provides a convenient method of serial interaction for high-speed communications between similar shift-register type devices. The LIN supports the Local Interconnect standard 2.0 and can be used as a UART in full-duplex mode using the standard Non-Return-to-Zero (NRZ) format. The DCAN supports the CAN 2.0B protocol standard and uses a serial, multi-master communication protocol that efficiently supports distributed real-time control with robust communication rates of up to 1 megabit per second (Mbps). The DCAN is ideal for applications operating in noisy and harsh environments (e.g., automotive and industrial fields) that require reliable serial communication or multiplexed wiring. The FlexRay uses a dual channel serial, fixed time base multimaster communication protocol with communication rates of 10 megabits per second (Mbps) per channel.
A FlexRay Transfer Unit (FTU) enables autonomous transfers of FlexRay data to and from main CPU memory. Transfers are protected by a dedicated, built-in Memory Protection Unit (MPU).
The NHET is an advanced intelligent timer that provides sophisticated timing functions for real-time applications. The timer is software-controlled, using a reduced instruction set, with a specialized timer micromachine and an attached I/O port. The NHET can be used for pulse width modulated outputs, capture or compare inputs, or general-purpose I/O. It is especially well suited for applications requir-

ing multiple sensor information and drive actuators with complex and accurate time pulses. A High End Timer Transfer Unit (HET-TU) provides features to transfer NHET data to or from main memory. A Memory Protection Unit (MPU) is built into the HET-TU to protect against erroneous transfers. The device has two 12-bit-resolution MibADCs with 24 total channels and 64 words of parity protected buffer RAM each. The MibADC channels can be converted individually or can be grouped by software for sequential conversion sequences. Eight channels are shared between the two ADCs. There are three separate groupings, two of which are triggerable by an external event. Each sequence can be converted once when triggered or configured for continuous conversion mode. The frequency-modulated phase-locked loop (FMzPLL) clock module contains a phase-locked loop, a clock-monitor circuit, a clock-enable circuit, and a prescaler. The function of the FMzPLL is to multiply the external frequency reference to a higher frequency for internal use. The FMzPLL provides one of the six possible clock source inputs to the global clock module (GCM). The GCM module provides system clock (HCLK), real-time interrupt clock (RTICLK1), CPU clock (GCLK), NHET clock (VCLK2), DCAN clock (AVCLK1), and peripheral interface clock (VCLK) to all other peripheral modules. The device also has an external clock prescaler (ECP) module that when enabled, outputs a continuous external clock on the ECLK pin. The ECLK frequency is a user-programmable ratio of the peripheral interface clock (VCLK) frequency. The Direct Memory Access Controller (DMA) has 32 DMA requests, 16 Channels/ Control Packets and parity protection on its memory. The DMA provides memory to memory transfer capabilities without CPU interaction. A Memory Protection Unit (MPU) is built into the DMA to protect memory against erroneous transfers. The Error Signaling Module (ESM) monitors all device errors and determines whether an interrupt or external Error pin is triggered when a fault is detected. The External Memory Interface (EMIF) provides a memory extension to asynchronous memories or other slave devices. Several interfaces are implemented to enhance the debugging capabilities of application code. In addition to the built in ARM Cortex-R4F CoreSight debug features, an External Trace Macrocell (ETM) provides instruction and data trace of program execution.
For instrumentation purposes, a RAM Trace Port Module (RTP) is implemented to support high-speed output of RAM accesses by the CPU or any other master. A Direct Memory Module (DMM) gives the ability to write external data into the device memory. Both the RTP and DMM have no or only minimum impact on the program execution time of the application code. A Parameter Overlay Module (POM) can re-route Flash accesses to the EMIF, thus avoiding the reprogramming steps necessary for parameter updates in Flash.

Figure C.1: Hercules Safety TMS570 MCU

Appendix D

Principal differences CAN, EtherCAT and Profinet

	CAN	EtherCAT	Profinet
Band	Up to 1 MBit/s	100 MBit/s	100 MBit/s
Networks lenght	40 m, with repeater in theory infinite	100 m, with hub in theory infinite	100 m, with hub in theory infinite
Data in a packet	8 byte	thorough several sub-telegrams, up to 4 gigabytes	244 bytes
Carrier sense Access	CSMA/BA	CSMA/CD	CSMA/CD
Kind of wire	Twisted Pair	flexible	4-wire shielded copper cable
Terminating resistor	120 Ω	100 Ω	100 Ω
Topology	Bus	All standard type	All standard type
CRC	16 bit	32 bit	32 bit
Prioritisation	Based on ID	None	Real time data have priority

Table D.1: Principal differences between the most important networks

Appendix E

Black channel

Traditionally, if a signal needs to be sent from one controller to another, a hard wired output from one would be connected to the other and would be treated as any other safety input or output on the controller. This method is fine for a small number of signals but can become costly and difficult to modify as the number of signals increase. However, if the same data can be passed using some form of data link the system becomes more flexible and cost effective. Clearly, for this type of link to work, the data must be transferred in a reliable and deterministic manner. The railway industry has been using serial data for safety signalling for many years and this has proven to be very successful. Traditionally in these systems the data is handled by components that are known and considered part of the safety system; hence, their failure modes have been identified and appropriate measures taken to ensure that they meet the safety requirements. Any software installed on the communication components are also safety certified. Therefore, the communication channel is well defined and each configuration has to be designed within the limits of the certified configurations. This type of communication channel is known as a 'White Channel' because the properties of the channel are well defined and known. Each of the components is designed with integrity levels that suit the specific application, so that there is confidence that the data is unlikely to be corrupted by, for example, a software bug and any transmission errors are detectable. The down side of this design philosophy is that migrating to newer technologies can be slow and costly, as well as the components having inflexible architectures. If it is looked at communication in general, the information technology arena has many powerful communication options that are both flexible and relatively low cost. One of the main drivers for this technology is obviously the high take up of the internet. For example, many personal computers can be connected together at high speed using off-the-shelf low cost hubs and switches. The cost is driven down by the high volume of users (unlike the low number of users of white channel communication systems). The down side of using the high volume low cost unit is that it's reliability can be an unknown quantity and the quality of the software inside is also an unknown factor; therefore, there is no way of knowing how a transported packet may be modified if a fault occurred. Communication paths using such unknown components are called 'Black Channel' communication paths. The name black channel comes from the concept of a black box. The intent of both a black box and black channel is that what goes in one end does not see anything between the inlet and outlet as it passes through the device. The difference between the two concepts is that, rather than a black box piece of hardware, it is the network itself that must appear to not be there. The bus system, therefore, does not perform any safety-related tasks but only serves as transmission medium. With a black channel, data from the sending safety system is launched into an unknown communication mechanism. Neither the sender or receiver know the route the data might take, how many nodes will actually be handling the data, how long the data will take to get to the end and how any of the nodes may have interfered with the data before being received by the receiving safety system. Therefore to use a Black Channel for safety related data, the data must have built-in mechanisms to detect any interference and with a confidence level of detection that is suitable for the safety application relying on the data.

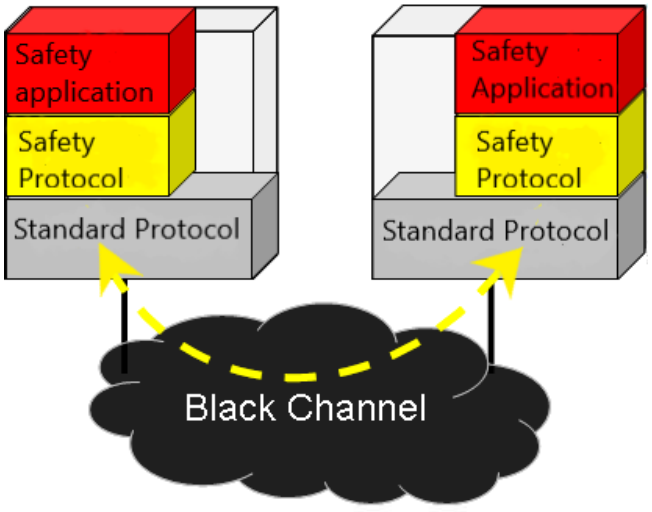


Figure E.1: Black Channel structure

The simplest form of black channel is a point-to-point serial data link. Typical failure modes of this black channel network are:

- Electrical noise (or cross talk) on the cable causing data corruption. Usually detected using a CRC and / or parity check. (The integrity of this will be covered further on in the paper).
- Total loss of data (broken cable). Usually detected by a time-out.
- Cross talk (its own data). Usually the sender receiving its own data that it is sending to the other end. Usually protected against by the sender turning off it's receiver during a send, or simply ignoring (clearing) any received data during the send.
- Cable cross talk (data from another data link). This may be detected if the data is a different type or length than expected.

Appendix F

Token Bus Basic Function

As already seen the token bus configuration is used to regulate the access to the network by the different master. In particular to manage the logical ring some basic function are necessary:

- addition of a new station in the logical ring;
- erasure of a station from the network;
- initialization of the logical ring;
- management of the loss of the token;

A Addition of a new station

Every master device, when it is holding the token, has the duty of periodically checking if a new device wants to enter in the logical ring. In particular this check is done through a frame called 'solicit-successor', which is a frame with the sent in the network. This frame have the function of asking if there is a device with an address number which is include between the current and the next addresses. The master sending this message can obtain:

1. no answer;
2. one valid answer;
3. more than one valid answers;
4. one invalid answer;

In the case n_r1 there is no action to do. In the case n_r2 the device changes the address of its next station and then it sends the token to the new device. The receiver accepts the token and saves the address of device who sent the message to it. In the case n_r3 the master holding the token understands that there were many answers and it starts a mechanism to identify which device should be added to the logical ring. In particular it sends a frame, called 'resolve-contention', which is composed by 4 window. Each window is identify by 2 bits: 00, 01, 10, 11 respectively. The devices which have answered to the first solicit-successor, answers in the window corresponding to the first 2 bit of their address, but only if there is no answer in the window with previous number. If many answer are, again, received, the master keeps to send resolve-contention, which are compare progressively with the 2 more significant bits still not compared. When finally there is only an answer, the master changes its address of its next station and then it sends the token to the new selected device. The receiver accepts the token and saves the address of device who sent the message to it. In the case n_r4 the master understands that there was a collision and so it listens the network.

B Erasure of a station from the network

A master device which wants to exit by the logical ring, waits for the token. When it receives the PDU, it sends a set-successor in which is specified the address of its next device. In such way its previous device can set the address of its new next device.

C Initialization of the logical ring

When the network needs a initialization or the token is lost, there is no activity on the bus. Each device identify a time-out and starts the initialization of the logical ring. This procedure consist in the assignment of the token to a device, which then starts to add new device to the network. The token is assigned to the node which sent in the network the frame claim-token. If many devices sent this frame, a procedure similar to the one for the selection of only one node for the addition of a new station, starts.

D Management of the loss of the token

Every device holding the token has to manage this special PDU until it is sent to the next node. The node sending the token understands if the transmission was successful when it finds activity in the network. If it happens that no activity was found for a time exceeding a fixed period, then the node re-send the token. If the procedure again has not success, then the device understands that its next station was unexpectedly gone out the logical ring and so it sends a frame, called who-follows, which is used to identify the new next device. If no answer is detected, then it tries again. If not even this procedure has success, then it sends a solicit-successor. Finally if no answer is received, then it understands that there was a failure.

Appendix G

Ethernet

Ethernet is a family of computer networking technologies commonly used in local area networks (LAN), metropolitan area networks (MAN) and wide area networks (WAN). It was commercially introduced in 1980 and first standardized in 1983 as IEEE 802.3, and has since retained a good deal of backward compatibility and been refined to support higher bit rates and longer link distances. Over time, Ethernet has largely replaced competing wired LAN technologies such as Token Ring, FDDI and ARCNET.
The original 10BASE5 Ethernet uses coaxial cable as a shared medium, while the newer Ethernet variants use twisted pair and optical fibre links in conjunction with switches. Over the course of its history, Ethernet data transfer rates have been increased from the original 2.94 megabits per second (Mbit/s) to the latest 400 gigabits per second (Gbit/s). The Ethernet standards comprise several wiring and signalling variants of the OSI physical layer in use with Ethernet.
Systems communicating over Ethernet divide a stream of data into shorter pieces called frames. The internal structure of an Ethernet frame (Fig. G.1) is specified in IEEE 802.3. It is composed by:

- the preamble, which consists of a 56-bit (seven-byte) pattern of alternating 1 and 0 bits, allowing devices on the network to easily synchronize their receiver clocks, providing bit-level synchronization.;
- the SFD (start frame delimiter) to provide byte-level synchronization and to mark a new incoming frame (1 byte);
- the destination MAC address (MAC address is a unique identifier(UID) assigned to a network interface controller) (6 bytes);
- the source MAC address (6 bytes);
- an IEEE 802.1Q tag or IEEE 802.1ad tag, which indicates virtual LAN (VLAN) membership and IEEE 802.1p priority. The first two bytes of the tag are called the Tag Protocol IDentifier (TPID) and double as the EtherType field indicating that the frame is either 802.1Q or 802.1ad tagged. (4 bytes);
- the EtherType field, to indicate which protocol is encapsulated in the payload of the frame (2 bytes);
- the payload, which the data of interest to transmit. It is composed at least by 46 bytes and as maximum by 1500 bytes.
- the frame check sequence (FCS), which is a 32 bit CRC that allows detection of corrupted data within the entire frame as received on the receiver side. The FCS value is computed as a function of the protected MAC frame fields: source and destination address, length/type field, MAC client data and padding (that is, all fields except the FCS).

The end of a frame is usually indicated by the end-of-data-stream symbol at the physical layer or by

loss of the carrier signal.

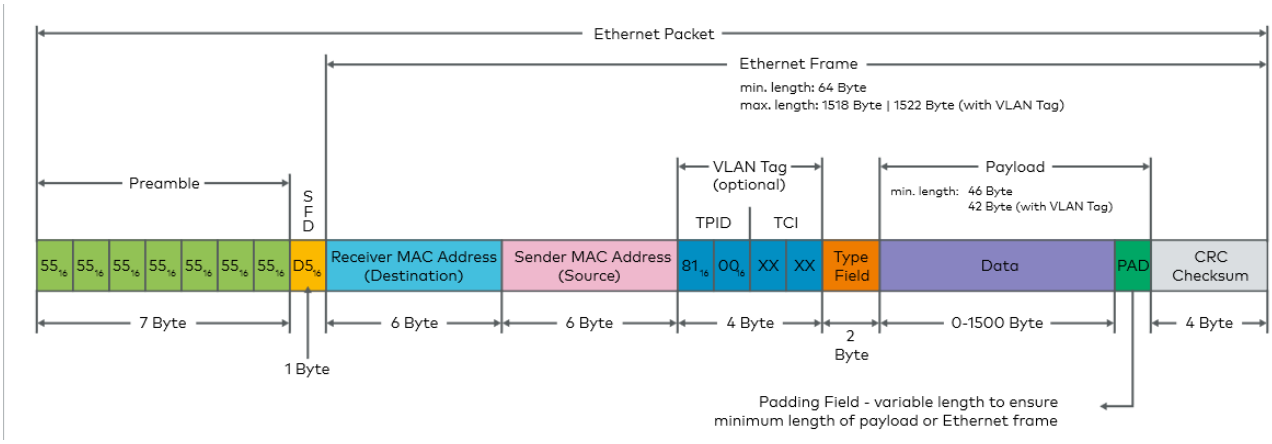


Figure G.1: Ethernet frame structure

Ethernet has evolved to include higher bandwidth, improved medium access control methods, and different physical media. The coaxial cable was replaced with point-to-point links connected by Ethernet repeaters or switches. Despite the evolution of Ethernet technology, all generations of Ethernet use the same frame formats, described above. The original version was based on the idea of computers communicating over a shared coaxial cable acting as a broadcast transmission medium. The method used was similar to those used in radio systems, with the common cable providing the communication channel. A scheme known as carrier sense multiple access with collision detection (CSMA/CD) governed the way the computers shared the channel. In particular a collision happens when two stations attempt to transmit at the same time. With CSMA/CD the devices detect the corrupt transmitted data and then re-transmit according to a previous fixed method (For example they wait random different times and then try to re-transmit). Through the first half of the 1980s, Ethernet's 10BASE5 implementation used a coaxial cable 9.5 mm in diameter. Its successor, 10BASE2, called "thin Ethernet", used the RG-58, which is a type of coaxial cable often used for low-power with an outside diameter of around 5 mm. In a modern Ethernet, the stations do not all share one channel through a shared cable or a simple repeater hub; instead, each station communicates with a switch, which in turn forwards that traffic to the destination station. In this topology, collisions are only possible if station and switch attempt to communicate with each other at the same time, and collisions are limited to this link. Furthermore, the 10BASE-T standard introduced a full duplex mode of operation which became common with Fast Ethernet. In full duplex, switch and station can send and receive simultaneously, and therefore modern Ethernets are completely collision-free.

Bibliography

[1] T. Sauter, S. Soucek, W. Kastner, and D. Dietrich,
"The evolution of factory and building automation," IEEE Ind. Electron. Mag., vol. 5, no. 3, pp. 35–48, 2011

[2] J. P. Thomesse,
“Fieldbus Technologies in Industrial Automation,” Proceedings of the IEEE, vol. 93, no. 6, pp. 1073–1101, June 2005.

[3] J. D. Decotignie
"Ethernet–Based Real–time and Industrial Communications," Proc. IEEE, vol. 93, no. 6, pp. 1102–1117, June 2005.

[4] A. Willig
"Recent and Emerging Topics in Wireless Industrial Communications: a Selection," IEEE Transactions on Industrial Informatics, vol. 4, no. 2, pp. 102–124, May 2008.

[5] M. Wollschlaeger, T. Sauter, and J. Jasperneite,
"The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0," IEEE Industrial Electronics Magazine, vol. 11, no. 1, pp. 17–27, March 2017.

[6] M. Sollfrank, M. F. Pirehgalin and B. Vogel-Heuser,
"Integration of safety aspects in modeling of Networked Control Systems," IEEE 15th International Conference on Industrial Informatics (INDIN), Emden, 2017, pp. 405-412

[7] J. Åkerberg, M. Gidlund, F. Reichenbach and M. Björkman,
"Measurements on an industrial wireless HART network supporting PROFIsafe: A case study," ETFA 2011, Toulouse, 2011, pp. 1-8.

[8] F. Tramarin, S. Vitturi, M. Luvisotto and A. Zanella,
"On the Use of IEEE 802.11n for Industrial Communications," in IEEE Transactions on Industrial Informatics, vol. 12, no. 5, pp. 1877-1886, 2016.

[9] S. Vitturi, A. Morato, A. Cenedese, G. Fadel, F. Tramarin, R. Fantinel.
"An Innovative Algorithmic Safety Strategy for Networked Electrical Drive Systems. 16th International Conference on Industrial Informatics (INDIN18), 2018.

[10] Y. Liu and Y. Song,
"EtherCAT-based functional safety-integrated communication," International Conference on Automatic Control and Artificial Intelligence (ACAI 2012), Xiamen, 2012, pp. 1005-1008

[11] Ing Stefano Maggi,
"Dispensa sul 'Controller Area Network' (CAN)"

[12] CIA 301,
"CANopen: Application layer and communication profile"

[13] Reiner Zitzmann, CIA ,
"CANopen Safety"

[14] Dr. Frank Jungandreas, Klaus Rupprecht, SYS TEC electronic GmbH,
"CANopen safety development solutions"

[15] CAN Newsletter 1/2010,
"CANopen Safety: The forgotten protocol extension"

[16] EtherCAT Technology Group,
"EtherCAT – The Ethernet Fieldbus"

[17] EtherCAT Technology Group,
"The safety solution for EtherCAT"

[18] Dalimir Orfanus, Reidar Indergaard, Gunnar Prytz, Tormod Wien (ABB)/ETFA 2013,
"EtherCAT-based Platform for Distributed Control in High Performance Industrial Applications"

[19] Member of PROFIBUS & PROFINET International,
"PROFIBUS System Description Technology and Application"

[20] Member of PROFIBUS & PROFINET International,
"PROFIsafe System Description Technology and Application"

[21] International Electrical Commission,
IEC 61131, 2003-1,"Programmable controllers", Part 3: Programming languages, Ed. 2

[22] International Electrical Commission,
DIN EN IEC 62061: "Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems" (2005-10)

[23] Alessandro Bonan, Lorenzo Fraccaro, Stefano Bianchin, Marc Cosgrove, Antonio Silvestri
"Safe Paper, Autec Srl"

[24] Autec Srl
"Catalogue: AIR Radiocontrols for automation, industrial lifting and operating machinery"

[25] Autec Srl
"Catalogue: DYNAMIC Radio Remote Controls for Hydraulic and Mobile Machinery"

[26] CODESYS
"Manuale di utilizzo del software Codesys"

[27] CODESYS
"User manual CODESYS Safety SIL2 "

Acknowledgements

This thesis is an important goal of my carrier, which was achieved with a lot of efforts and with the help of some important people. At the same time this thesis is a starting point for the future works that will be done with the Ph.D. 'in Alto Apprendistato' and also for my working carrier. For these reasons I want to thank professor Stefano Vitturi, who gave me this important opportunity and gave me all the assistance I needed. Besides it is essential for me to thank Autec Srl and all the colleagues who worked with me for their helpfulness, their patient and time.
I want to thank, also, my family: my mother and my father who allowed me to achieve this important goal, with emotional and economic support. They always pushed me to not be satisfy of a goal, allowing me to achieve better results. My sister who had supported me, especially when I was younger, teaching me when my parents were working and whom I want to wish all the happiness of the world with the incoming baby and all the supports I can give her as uncle. My girlfriend Giorgia who has supported me for all these 5 years, who has waited for me every time I had to study, being patient and understanding the importance and all the difficulties of these path. I want to thank her since she was always available to talk, laugh and forget all the problems for all the moments together, allowing me to live this path with happiness and lightening the difficulties with her supports.
I want to thank all my friends who helped me to live with a bit more light-heartedness, who laughed together and who were fellows of so many adventures. Finally I want to thank all the people, who I can not cite for time and space, who supported me with a conversation, a smile, a laugh, since all these people, with a small action, have helped me to achieve this goal.