



ENGINEERING PROJECT FINAL REPORT

Siemens Project

SIEMENS

Acknowledgments

We extend our heartfelt gratitude to Siemens for their invaluable support throughout this project. Our sincere thanks go to Siemens' Digital Industry Process Automation Technology and Innovation Connectivity Edge (DI PA TI CE) department for their guidance and resources that significantly contributed to the success of our endeavour. We would like to express our deep appreciation to Christoph Weiler and Yves Jung for their exceptional administrative support, which played a crucial role in facilitating the various aspects of this project. A special mention of gratitude goes to Jochen Balduf for his immense support and mentorship. His guidance and dedication, from providing internships to imparting invaluable insights into industry embedded programming and company organization, have been truly invaluable to our growth. Lastly, we would like to acknowledge and thank Télécom Physique Strasbourg, our esteemed institution, for providing us this opportunity.

Abstract

The report details a comprehensive engineering project conducted by four students from the Télécom Physique Strasbourg engineering school in collaboration with Siemens. The project aimed to develop a system for precise fluid level and density determination in tanks using pressure sensors and communication protocols. The report encompasses various aspects, from project overview to technical implementation and testing. The project's core objective was to implement a communication protocol between pressure sensors to calculate higher process values. It involved the development of firmware for pressure sensors, leveraging components like STM32 microcontrollers, embOS, and emNet TCP/IP stacks. A specific focus was on the development of a lightweight data protocol (XCom) for efficient packet exchange between boards, along with a configuration webpage to input system details. Furthermore, the report delves into the project's work organization, detailing the timeline, used tools, task assignments, and daily organizational strategies employed throughout the project's duration. The team adapted and evolved their project plan over time, managing tasks, conducting meetings, and dividing work among team members effectively. Overall, the report encapsulates the comprehensive process of designing, implementing, testing, and refining a complex engineering solution involving hardware, firmware, and communication protocols for precise fluid measurement in tanks.

Table of contents

1. Introduction.....	5
2. Team presentation	6
3. Project presentation.....	7
3.1 Pressure measurement sensors	7
3.2 Fundamental physics basics	7
3.3 Requirements	9
3.4 Use components.....	9
4. Solution implemented.....	10
4.1 Board's firmware	10
4.1.1 Global system	10
4.1.2 Use embOS features.....	10
4.1.3 Firmware architecture.....	11
4.1.4 Data protocol.....	13
4.1.5 Configuration webpage	15
4.1.6 Time synchronization	18
4.2 Monitoring application.....	19
4.3 Wireshark plugin	21
4.4 Client and server test application.....	22
5. Results	23
5.1 Time synchronization result	23
5.2 Webpage configuration result.....	24
5.3 Monitoring application result.....	24
6. Work organization	25
6.1 Timeline	25
6.2 Tools	26
6.3 Task assignment	27
6.4 Daily organization	27
7. Conclusion	28

Abbreviations

Short	Description
FW	Firmware
SW	Software
HW	Hardware
MCU	Microcontroller
OS	Operating System
IDE	Integrated Development Environment
TS	Time Synchronization

1. Introduction

This report represents the culmination of a vast engineering project undertaken during our course at Télécom Physique Strasbourg. As a team of four students specializing in IT and networks, with a focus on the Internet of Things, we were privileged to collaborate on a significant project titled *Lightweight IP-based Communication for Field Devices*, commissioned by Siemens. Our primary objective throughout this project was to gain practical insights into the realm of engineering within the IT domain. Over the course of our second and third academic years, we immersed ourselves in this project, not just to enhance our technical acumen but also to sharpen our soft skills, critical for professional growth. This report encapsulates the culmination of our efforts, presenting a comprehensive overview of our engineering project, its technical facets, the implemented solutions, and the invaluable lessons learned throughout this enriching experience.

2. Team presentation

This project was carried out by a group of 4 students from the Télécom Physique Strasbourg engineering school in in the IT and networks field, and in the networks and Internet of Things specialization. The group was made up of Thomas Dumond, Ethan Huret, Nicolas Cipolla and Louis Duval. The project was undertaken for Siemens' Digital Industry Process Automation Technology and Innovation Connectivity Edge (DI PA TI CE) department. The project was supervised by Jochen Balduf, Christoph Weiler and Yves Jung. On the school side, the project was supervised by Quentin Bramas.

3. Project presentation

The goal of this project is to determine with precision the level and the density of a fluid in a tank using several pressure sensors. Based on the individual pressure value of the sensors the level and the density should be calculated. Therefore a cyclic measurement and data exchange between the sensors is required. The mission of this project is to implement a communication protocol between the sensors and the calculation of the higher process values.

3.1 Pressure measurement sensors

Siemens is developing a wide range of pressure measurement sensors. They differ at least by the internal functionality and features, used materials, housing, measurement range and accuracy and communication physics. Independent from this there are three kinds to measure the pressure:

- Relative pressure (rel. Pressure)
- Absolute pressure (abs. Pressure)
- Differential pressure (diff. Pressure)

The relative pressure measurement sensor is measuring the pressure at one point against the atmosphere. These kinds of sensors could be used only if the atmosphere pressure does not have any effect on the final measurement result.

The absolute pressure measurement sensor is measuring the pressure at one point against a vacuum reference. As a result, in the case of a tank with atmosphere for P_0 the additional pressure of the atmosphere is also measured, like it is shown on [Figure 2](#).

Finally, the differential pressure measurement sensor is measuring the pressure difference between two measurement points. This kind of sensor could be used to measure the flow of a medium within a pipe or for tank application. Therefore diaphragm seal and oil filled capillary pipes, or open pipes to the process application are used. Due to external impacts like solar radiation or clogging of the pipes the level of the tank could not be measured correctly. To avoid this the differential pressure should be performed by single absolute pressure measurement devices.

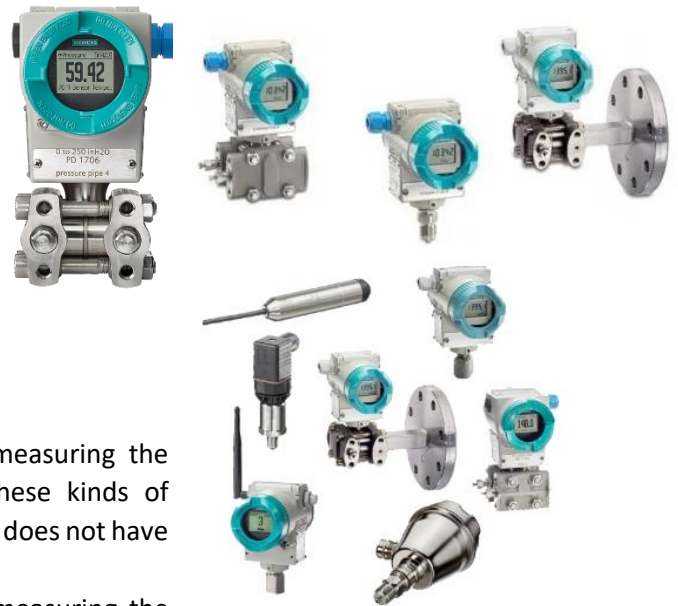


Figure 1 – Sensors examples

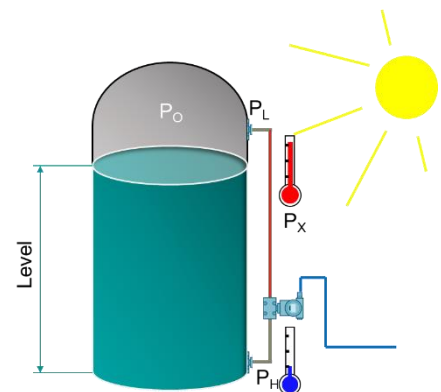


Figure 2 – Tank example

3.2 Fundamental physics basics

There are three possibilities to calculate the tank's level:

• If the density ρ of the fluid and the gas pressure P_O in the tank are known, one pressure sensor P_H is needed to measure the level of the tank:

- $P_H = P_M + P_O$ with P_H for high pressure, P_M for hydrostatic pressure of the medium and for P_O overpressure of the tank

The fundamental hydrostatic law gives us:

- $P_M = \rho \cdot g \cdot l$
- $l = \frac{P_H - P_O}{\rho g}$ with ρ the density of the fluid and P_O the gas pressure in the tank as static known values

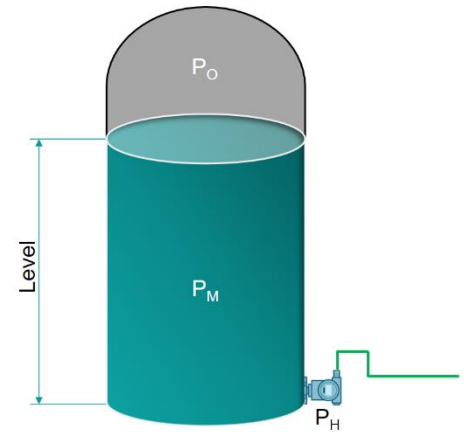


Figure 3 – Tank measurement level case 1

• If the over pressure P_O of the tank is unknown, two pressure sensors are needed to be able to measure the level of tank. The density ρ of the fluid must be also known as mentioned before.

- $P_H = P_M + P_L$ with P_H for high pressure, P_M for hydrostatic pressure of the medium and for $P_L = P_O$ for low pressure

The fundamental hydrostatic law gives us:

- $l = \frac{P_H - P_L}{\rho g}$ with ρ the density of the fluid

This kind of measurement is the first use case to replace existing solutions using one differential pressure measurement sensor. Using diaphragm seals at the bottom and top of the tank to measure the difference of the overpressure P_O and total pressure P_H .

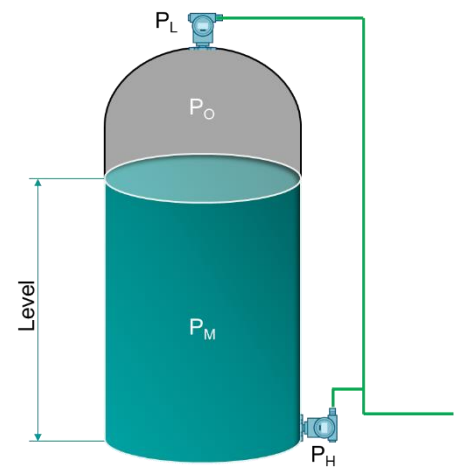


Figure 4 – Tank measurement level case 2

• To improve these new measurement concept, the density of the medium could be also measured if a third sensor come in place:

- P_H for high pressure
- P_D for density pressure
- $P_L = P_O$ for low pressure

The fundamental hydrostatic law gives us:

- $\rho = \frac{P_H - P_D}{gh}$ with g the gravity and h the distance between P_D sensor and the bottom of the tank
- $l = \frac{P_H - P_L}{\rho g}$ with ρ the density of the fluid

Moreover, we can have the level of the fluid with another formula:

- $l_1 = \frac{P_{M1}}{\rho g}$ with $P_{M1} = P_H - P_D$
- $l_2 = \frac{P_{M2}}{\rho g}$ with $P_{M2} = P_D - P_L$
- $l = l_1 + l_2$

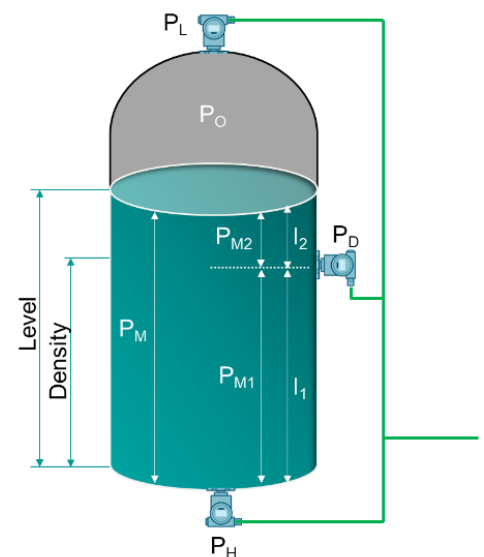


Figure 5 - Tank measurement level case 3

3.3 Requirements

The project focuses on implementing a communication protocol and calculating higher process values. The project covers the implementation of a firmware (FW) for the pressure sensors to perform these different tasks. The first requirement is that all the sensors must have the same FW.

Secondly, as mentioned in Fundamental physics basics, the project must include three different application cases:

- 1) with one sensor (P_H)
- 2) with two sensors (P_H with P_D or P_L)
- 3) with three sensors (P_H with P_D and P_L)

The first two cases require additional information (gas pressure, density) and this information must be able to be entered directly by the user. Another point not to be overlooked is that the sensors must have synchronised pressure measurement cycles. Finally, there are a few constraints to which the project must be limited, such as the fact that no pressure sensor is available for the project and that a simulation must be undertaken, or the fact that no external memory system must be used.

3.4 Use components

In order to carry out this project, software (SW) and hardware (HW) components are used:

- *STM32F429ZI eval-board*: STM32 microcontroller (MCU) to simulate pressure sensors.
- *Segger embOS*: pre-emptive Real Time Operating System (RTOS) for the base of the project FW.
- *Segger emNet*: IPv4/IPv6 TCP/IP stack to manage network in the FW.
- *Segger Embedded Studio*: all-in-one Integrated Development Environment (IDE) to program and flash the FW to the STM32 MCU.

The embOS and emNet were specified because they are also used in Siemens' products.

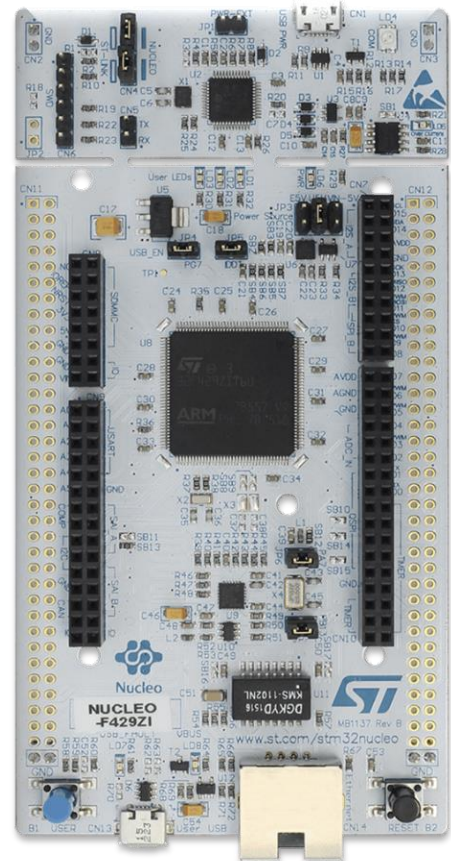


Figure 6 - STM32F429ZI

4. Solution implemented

4.1 Board's firmware

4.1.1 Global system

The system architecture is based on a primary/secondary architecture. This is C language FW compiled with GCC. Each eval card has its own IPv4 address, which is set/configured at device start-up via DHCP requests. MAC addresses can be selected using jumpers on the underside of the board. There are four different positions giving four different MAC addresses. This is a workaround that allows each board to have the same firmware image. Normally, additional non-volatile memory contains all device instance-specific configuration. This type of memory is not provided by eval cards.

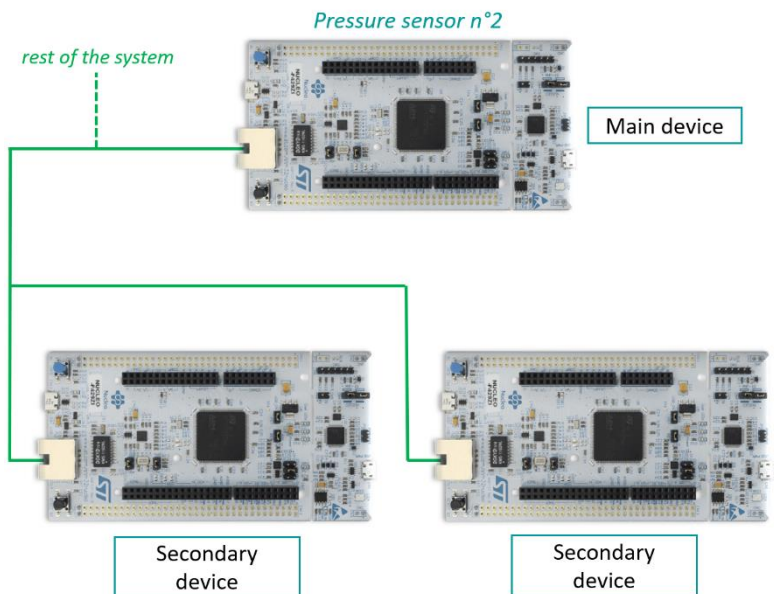


Figure 7 - Main/Secondary architecture

4.1.2 Use embOS features

As mentioned in Use components, the FW uses *embOS* as operating system and some features that is offering. This project uses *tasks* to separate each part of the application. A task is an independent unit of work that runs concurrently with other tasks. It has its own stack, execution context, and function. It is created during system initialization and can be started, suspended, resumed, or deleted during runtime. A *scheduler* is responsible of the task switching and decide which task runs. For that, it uses *preemption*, where a higher-priority task can interrupt the execution of a lower-priority task if necessary. It also uses *task events* to switch between these application parts. It is a digital signal which wake up a sleeping/waiting task. It is based on *event flags*, which are 32-bit variable where each bit represents a different condition. For example, a task (no. 1) can wait for one or more specific event flags, and another task (no. 2) can signal one or more of these specific event flags by setting the corresponding bits. Then, the scheduler will stop the task no. 2 and start the task no. 1. Finally, it uses *mailboxes* to exchange data between application parts. Mailboxes are *queue of messages* for inter-task communication and data exchange. It is linked to only one task.

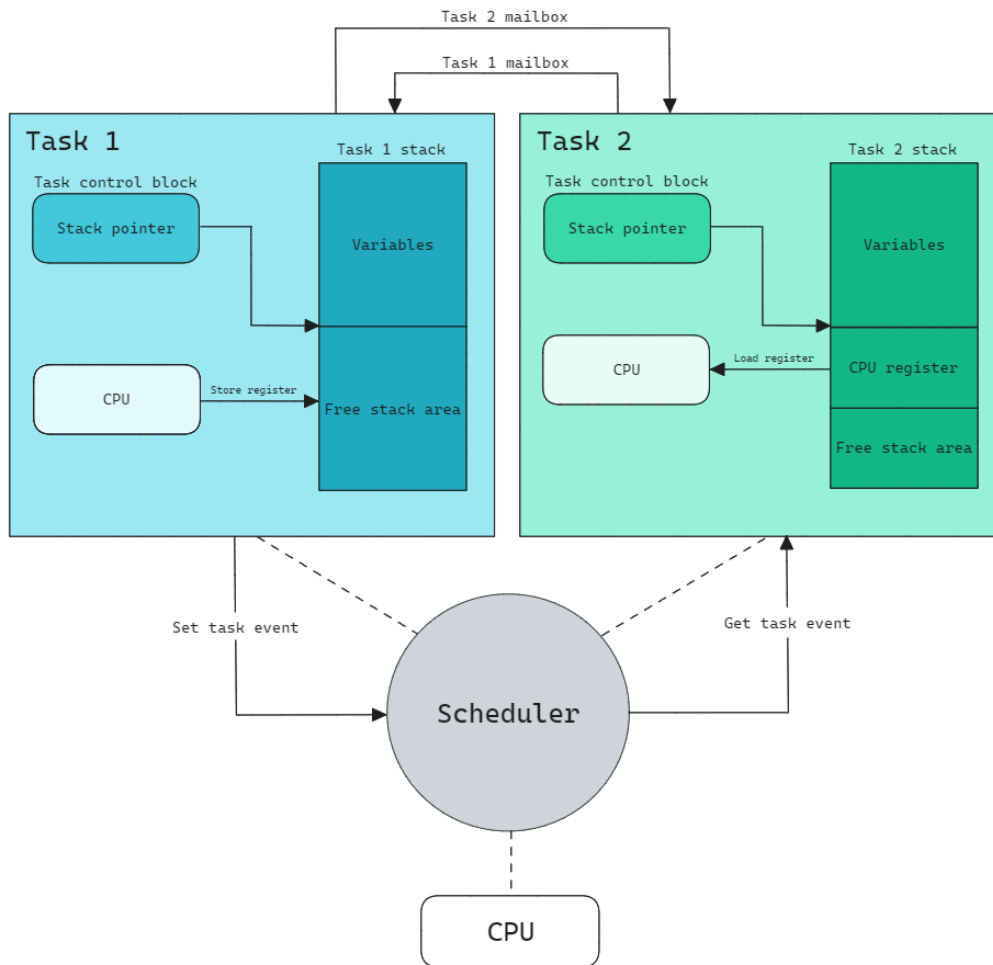


Figure 8 - embOS features: task, task event and mailbox

4.1.3 Firmware architecture

This project is divided into 4 different parts, which are each implemented as separate embOS task. Before these 4 tasks, the HW is initialised, then the first task, called Main, and its mailbox are created, and launched when the OS starts.

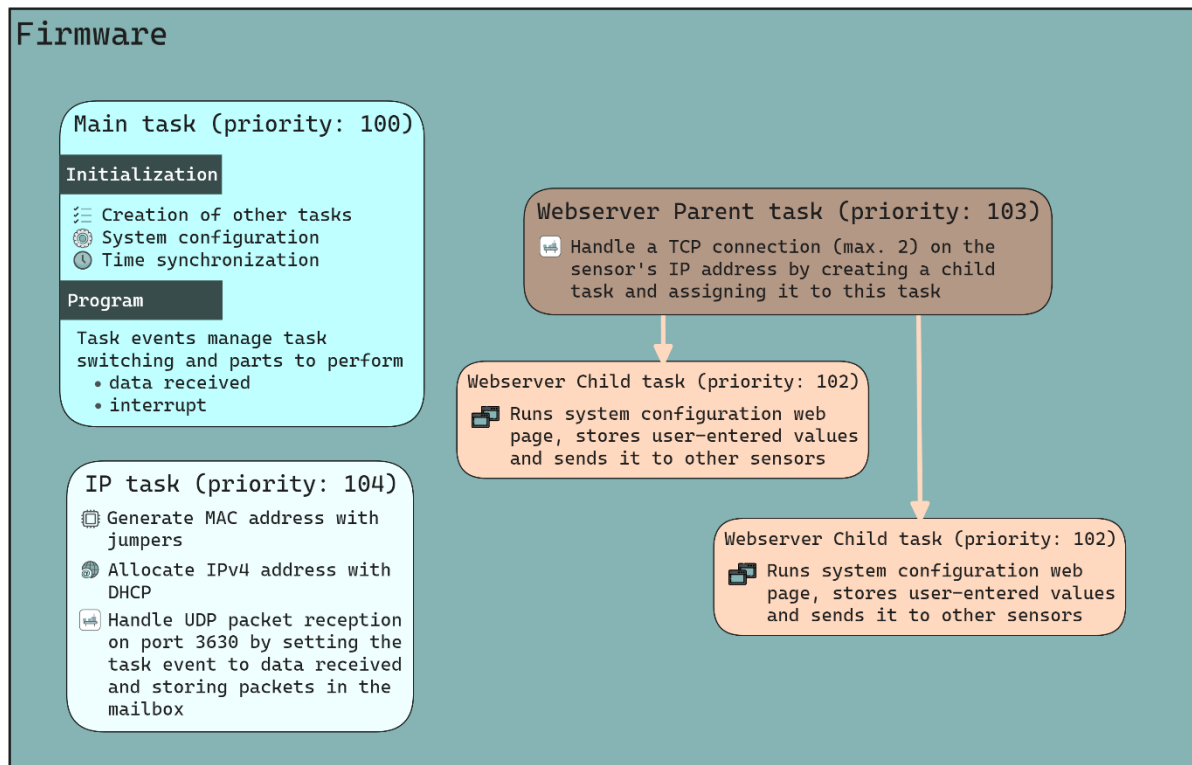


Figure 9 - Firmware architecture

The *Main* task is divided into two parts, an initialisation part, and an application part.

The *initialisation* part sets the system variables to zero and sets the callback functions (functions passed as parameters in another function). It is also responsible for the configuration part. It waits to receive a complete system configuration before moving on. Finally, it performs the initialisation of time synchronisation (TS), which starts the clocks of each board at the same time.

The *application* part is based on task events and interrupts.

- *Task events* allow the main task to wait for data reception. When a packet is received, the Main task handles it using the *XCom Decode module* of the Data Protocol and the callback, contained in XCom Decode, sets a particular bit of the event flag according to packet type. The same procedure is used with interrupts, to switch from the interrupts to the main task.

- *Interrupts* are initialised after performing time synchronization. This represents a function performed every n millisecond, which uses a task event to signal to main task what to do. There are 3 interrupts, one for pressure measurement, one for level calculation and one for periodically reperform time synchronisation. Moreover, one of the configuration variables is whether the board is the main or not, otherwise it will not launch time synchronization and level calculation interrupt.

The *IP* task is responsible for allocating the board's IP address using the DHCP protocol, and for setting the MAC address using jumpers. In addition, on initialization, it sets a callback for the reception of UDP packets on port 3630, which corresponds to the Data Protocol. This callback stores the packet in the main task mailbox and sets the *EVENT_RECEIVEDDATA* task event. This task has the highest priority, as it must be launched as soon as a packet arrives.

The *Webserver Parent* task is waiting for a TCP connection. When a TCP connection is initiated, it creates a child task and links the TCP connection to this task.

The *Webserver Child* task corresponds to this child task. It runs the Configuration Webpage and retrieves the values entered by the user. Once the user has entered all the information, it sets the event flag to *EVENT_USERSETCONF*.

4.1.4 Data protocol

Another part of this project is to exchange data between the boards, especially for certain operations such as time synchronization or configuration, but also and above all to exchange process values such as pressure and level values. The easiest way to do this is to create a protocol from scratch. This solution has the benefits of being very lightweight, of being precisely adapted to the needs of the project and of allowing features to be added easily.

4.1.4.1 Protocol structure

The protocol is called *XCom*. It is made up of 4 main types, representing the different parts of the application:

- Time synchronisation
- Configuration
- Pressure measurement
- Level calculation

Cross communication protocol									
message ID 64 bits	length 32 bits	type 8 bits	payload						
		0x01 (time synchro)	type 8 bits	timestamp 64 bits	offset 32 bits				
			0x01 (initialization)						
			0x02 (cycle)						
		0x02 (configuration)	type 8 bits	payload					
			0x01 (send tank info)	diameter 32 bits	diameter unit 8 bits	volume 32 bits	volume unit 8 bits	height 32 bits	height unit 8 bits
			0x02 (get tank info)						
			0x03 (sensor info)	number 8 bits	IP address 32 bits	role 8 bits	height 32 bits	height unit 8 bits	
			0x04 (get sensor info)						
			0x05 (fluid information)	density 64 bits	density unit 8 bits	gas pressure 32 bits	pressure unit 8 bits		
			0x06 (get fluid info)						
			0x07 (cycle information)	cycle time 32 bits	time unit 8 bits	frame loss 8 bits			
		0x03 (pressure)	type 8 bits	payload					
			0x01 (send pressure)	pressure 64 bits	pressure unit 8 bits	quality code 8 bits			
			0x02 (get pressure)						
		0x04 (level)	type 8 bits	payload					
			0x01 (send level)	level 32 bits	level unit 8 bits				
			0x02 (get level)						

Figure 10 - XCom protocol architecture

Each main type has subtypes. In the case of Time synchronization, this corresponds to calculating latency for *cycle* subtype and starting interrupts for *init* subtype. In the case of Configuration, this corresponds to the different groups of values linked to an item such as the tank or the fluid. In the case of Pressure and Level, there are two subtypes, a *Get* subtype, where for a received packet of this type, the board returns its value (pressure or level), and a *Send* subtype, where the board sends its value.

4.1.4.2 Protocol module

The protocol implements 3 different modules.

The *Init* module creates the XCom structure according to the type with a simple function that requires the packet subtype values.

```

1  int XCom_Init_TS(XCom_Frame_t *frame, uint8_t type, uint64_t timestamp,
2      uint32_t offset);
3
4  int XCom_Init_Conf_Send_Tank(XCom_Frame_t *frame, uint32_t diameterValue,
5      uint8_t diameterUnit, uint32_t volumeValue, uint8_t volumeUnit,
6      uint32_t heigthValue, uint8_t heigthUnit);
7
8  int XCom_Init_Conf_Get_Tank(XCom_Frame_t *frame);
9
10 int XCom_Init_Conf_Send_Sensor(XCom_Frame_t *frame, uint8_t number,
11     uint32_t ip, uint8_t role, uint32_t heigthValue, uint8_t heigthUnit);
12
13 int XCom_Init_Conf_Get_Sensor(XCom_Frame_t *frame);
14
15 int XCom_Init_Conf_Send_Fluid(XCom_Frame_t *frame, uint32_t densityValue,
16     uint8_t densityUnit, uint32_t gasPressureValue, uint8_t gasPressureUnit);
17
18 int XCom_Init_Conf_Get_Fluid(XCom_Frame_t *frame);
19
20 int XCom_Init_Conf_Cycle(XCom_Frame_t *frame, uint32_t timeValue,
21     uint8_t timeUnit, uint8_t frameLoss);
22
23 int XCom_Init_Pressure(XCom_Frame_t *frame, uint8_t type,
24     uint32_t pressureValue, uint8_t pressureUnit, uint8_t qualityCode);
25
26 int XCom_Init_Level(XCom_Frame_t *frame, uint8_t type, uint32_t levelValue,
27     uint8_t levelUnit);

```

Figure 11 - XCom Init function prototype

The *Encode* module transforms the XCom structure into an 8-bit array in network endian format.

```

1  int XCom_encode(XCom_Frame_t frame, size_t length, uint8_t *data);

```

Figure 12 - XCom Encode function prototype

The *Decode* module transforms an 8-bit array in network endian format into an XCom structure.

```

1  int XCom_decode(uint8_t *data, size_t *length, XCom_Frame_t *frame);

```

Figure 13 - XCom Decode function prototype

4.1.4.3 Packet exchange

To exchange packets, the first step is to use the Init module to create the XCom structure of the desired type. This is followed by the Encode module, which converts the structure to network format.

The last step is to send the packet by using the `IP_UDP_Send` function of emNet, with the previous built array in argument.

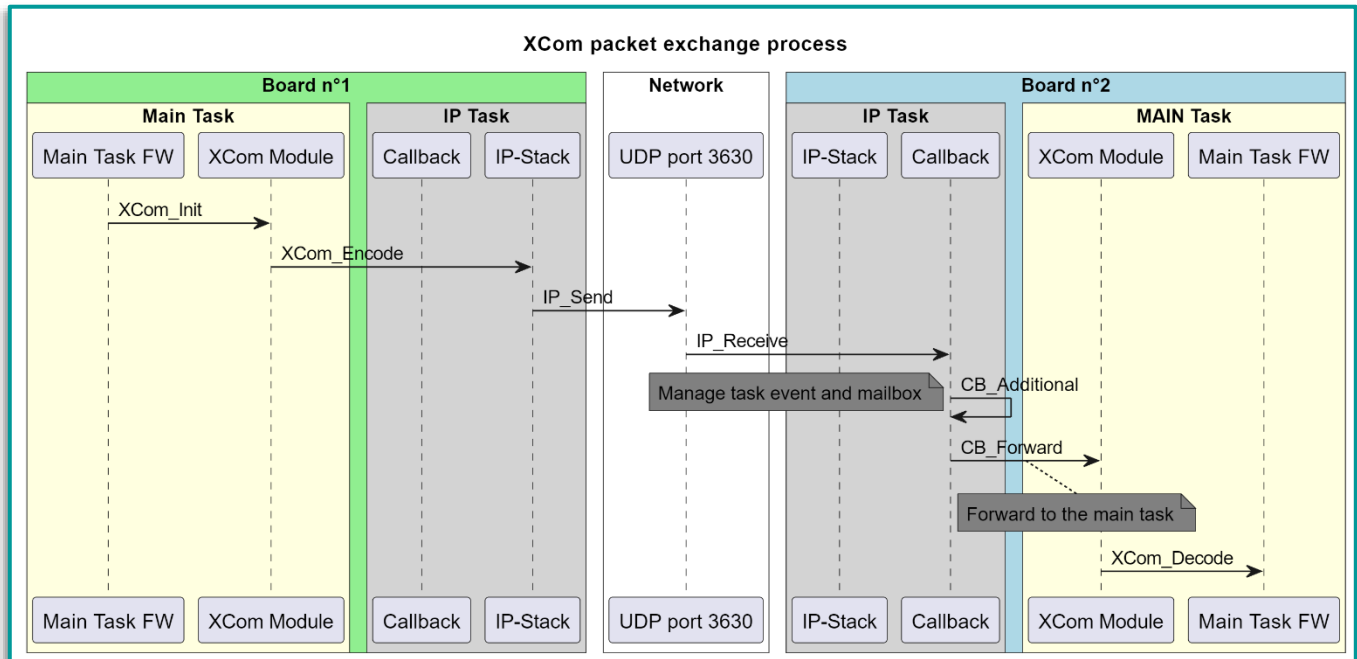


Figure 14 - XCom packet exchange process

Reception is handled by the IP task which, with the callback on port 3630 of UDP. The callback stores the packet in the main task's mailbox and sets the event flag to `EVENT_DATA RECEIVED`. The main task is then woken up and proceeds to decode the packet contained in its mailbox. The decode module has callbacks according to the package subtype, allowing switching to other parts of the application such as Time synchronization or Configuration.

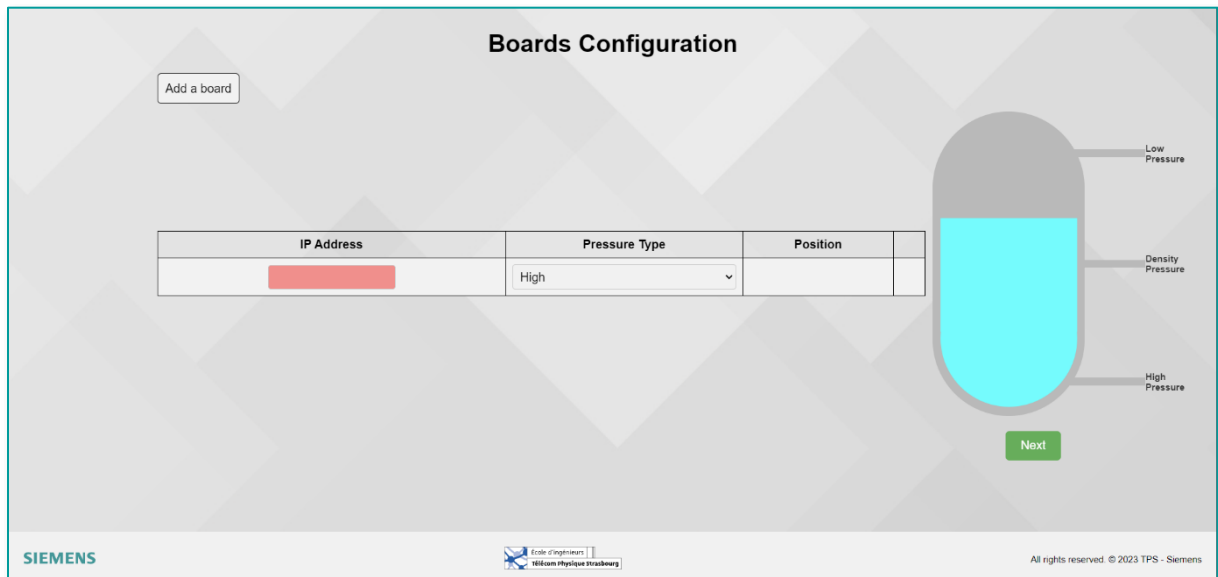
4.1.5 Configuration webpage

As mentioned in Requirements, the boards need additional information from the user. Moreover, the user must also configure the system, which means describing the number of sensors, their positions on the tank, etc. To enable the user to enter this information, a web server with a wizard-based interface on the board's web page to streamline user interaction and simplify the configuration process is implemented, with a task is dedicated to run this web server between the board and the user's pc. Each board runs this web server, so the user can configure the system via any board using its IP address. Once the user has entered the configuration, the board checks the validity of the information entered and sends it to the other boards.

The configuration is done via different wizard pages :

- The first page (Figure 15) retrieves the measurement scenario and for each board its IP address, the pressure type that the board measure and its position
- The second page (Figure 16) retrieves information about the tank, its volume, its diameter, and its height
- The third page (Figure 17) retrieves information's about the fluid inside the tank, its density, and the gas pressure at the top of the tank
- The fourth page (Figure 18) retrieves systems constraints, the wanted update cycle for measurement and the number of allowed packet loss before considering a connection loss

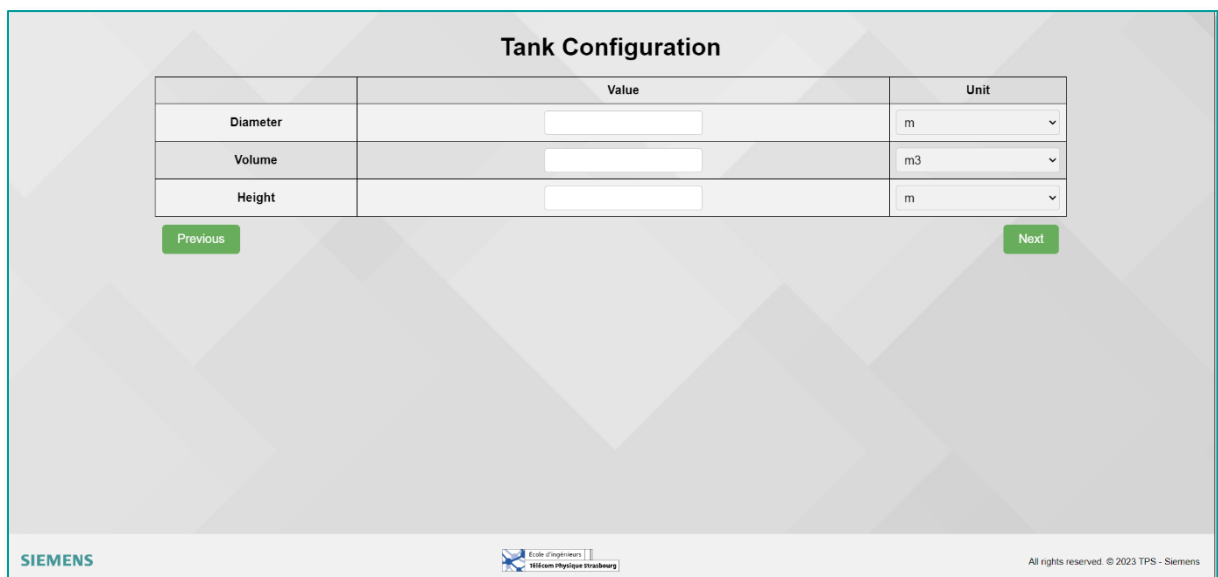
- The last page ([Figure 19](#)) retrieves the IP address of the device that runs the Monitoring application, which is not mandatory



The screenshot shows the 'Boards Configuration' page of a Siemens wizard. At the top left is a button labeled 'Add a board'. Below it is a table with three columns: 'IP Address', 'Pressure Type', and 'Position'. The 'IP Address' cell contains a red rectangular input field. The 'Pressure Type' cell has a dropdown menu with 'High' selected. The 'Position' cell is empty. To the right of the table is a diagram of a tank with three pressure ports labeled 'Low Pressure', 'Density Pressure', and 'High Pressure'. A green 'Next' button is located below the diagram. The footer contains the Siemens logo, a university logo (École d'ingénieurs | Wilcom Physique Strasbourg), and the text 'All rights reserved. © 2023 TPS - Siemens'.

IP Address	Pressure Type	Position
<input type="text"/>	High	

Figure 15 - First page of the Wizard concerning the boards configuration



The screenshot shows the 'Tank Configuration' page of a Siemens wizard. It features a table with three rows: 'Diameter', 'Volume', and 'Height'. Each row has a 'Value' column with an input field and a 'Unit' column with a dropdown menu. The units are 'm', 'm3', and 'm' respectively. Below the table are 'Previous' and 'Next' buttons. The footer contains the Siemens logo, a university logo (École d'ingénieurs | Wilcom Physique Strasbourg), and the text 'All rights reserved. © 2023 TPS - Siemens'.

	Value	Unit
Diameter	<input type="text"/>	m
Volume	<input type="text"/>	m3
Height	<input type="text"/>	m

Figure 16 - Second page of the Wizard concerning the tank configuration

Fluid Configuration

Default density :	Water
Default gas pressure :	Atmospheric pressure

Previous Next

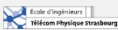
SIEMENS  All rights reserved. © 2023 TPS - Siemens

Figure 17 - Third page of the Wizard concerning the fluid configuration

Warnings Configuration

	Value
Wanted cycle of communication in ms	40
The maximum of acceptable packet loss before saying that the connection is lost	5

Previous Next


SIEMENS  All rights reserved. © 2023 TPS - Siemens

Figure 18 - Fourth page of the Wizard concerning the communication configuration

Debug Configuration	
IP Address of the test application	IP
	<input type="text"/>

Previous Submit

SIEMENS Ecole d'ingénieurs | HES-SO Valais Suisse All rights reserved. © 2023 TPS - Siemens

Figure 19 - Final page of the Wizard concerning the monitoring application

4.1.6 Time synchronization

To ensure that measurements are synchronized, the boards must initially start their pressure measurement cycle simultaneously. Furthermore, to maintain clock consistency and eliminate any discrepancies, the process is cyclically repeated to skip clock differences.

Each board must start its pressure measurement cycle at the same time. To do that, the time synchronization protocol is used to synchronized them. It consists in 2 steps:

- The first step consists in measuring the latency between the main and the secondary board. The main board starts a timer and sends a packet *XCOM_TS_Cycle_Packet* which is used as a ping pong packet. The secondary board receives the packet and answers directly to the main board with the same packet. The main board, when receiving the answer, stops its timer and calculates the latency. This step could be repeated many times to have a mean of the latency between the boards.
- The second step consists in synchronizing the clocks with the new timestamp calculated according to the latency measured. The main board takes the maximum latency, multiplies it by 4 to give a margin, and sends a *XCOM_TS_Init_Packet* packet with this value minus the corresponding latency divided by 2 to each board.

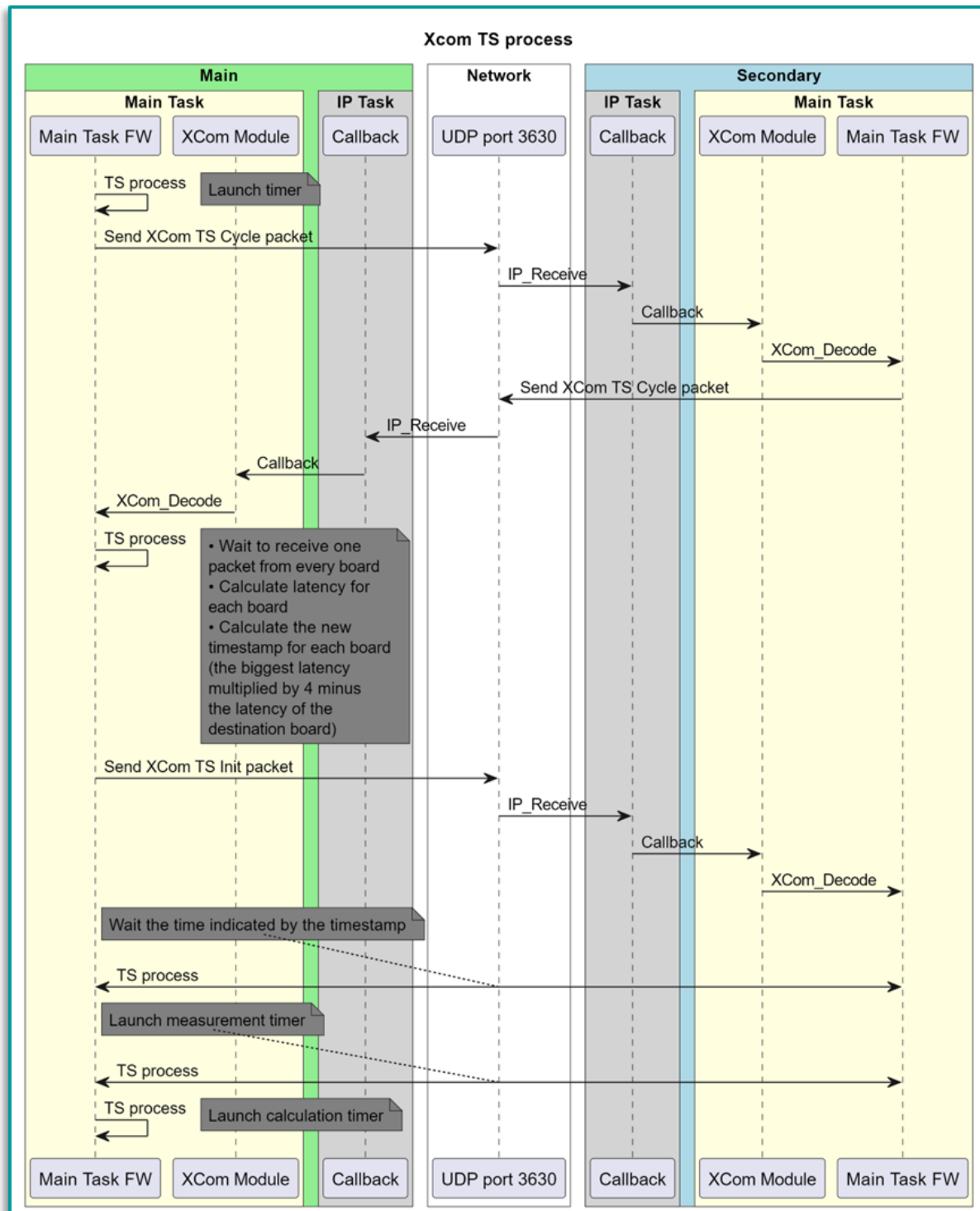


Figure 20 - Illustration of Time Synchronization

4.2 Monitoring application

The monitoring application can be used to test and monitor sensor operation. This is a web application written in python. It has three pages: Send, Read and Test.

- The first page allows you to send a pressure value to a board, using its IP, the value, and the unit of the value:

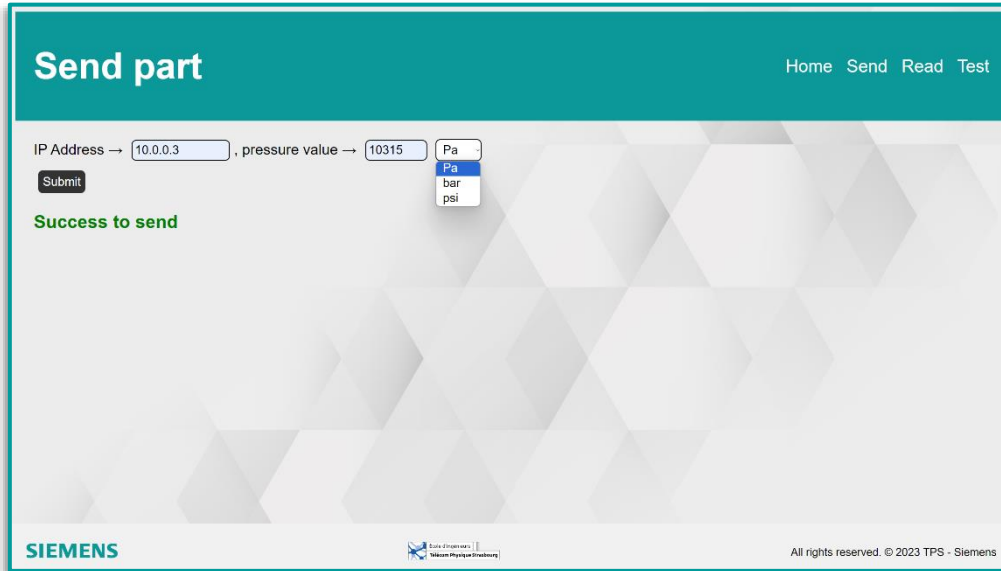


Figure 21 - Send page of Monitoring application

- The second page allows you to retrieve a board's pressure value using its IP:

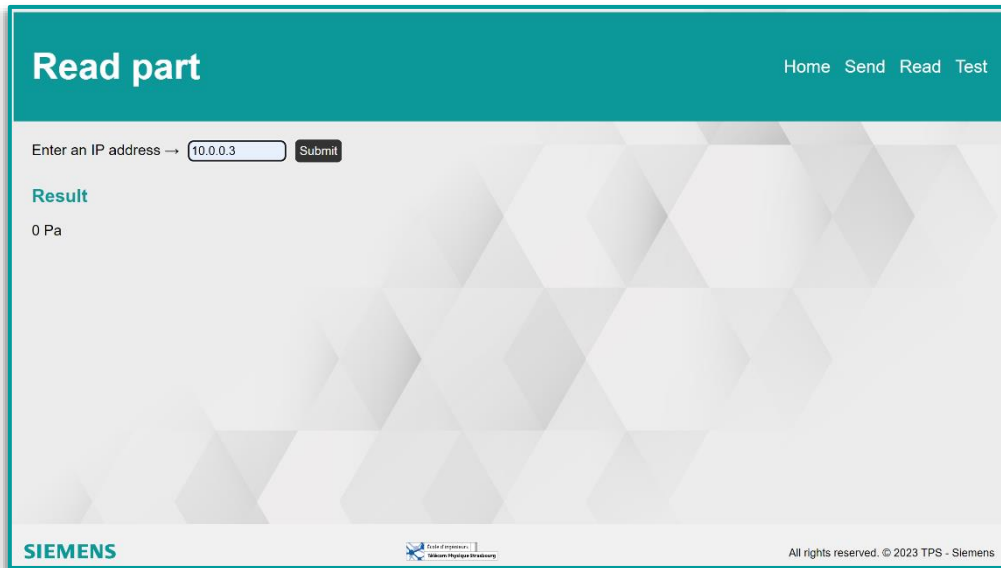


Figure 22 - Read page of Monitoring application

- The last page allows to test the system, by entering the pressure value of each sensor in the same way as on the first page. These pressures are sent to the boards. The application retrieves the level value calculated by the main board and compares it with value that should be obtained using the system configuration retrieved from a board (tank and fluid information):

Figure 23 - Test page of Monitoring application

This application uses Python as its language. It uses the Python libraries: *flask* to run a web application, *request*, *ping3* and *socket*. The XCom protocol has been reprogrammed in Python to use it in this application. To contact the board, it uses an *unused random port* as source port and the *XCom port* (3630) as destination port. The board responds using the XCom port as source port and the same application source port as destination port.

4.3 Wireshark plugin

During the project, there were major debugging phases and Wireshark was often used to observe which packets were exchanged between boards. However, boards communicate via the XCom protocol, which Wireshark cannot interpret.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.21	192.168.0.26	UDP	68	54044 → cs-remote-db(3630) Len=26
2	0.000582	192.168.0.21	192.168.0.26	UDP	56	54045 → cs-remote-db(3630) Len=14
3	0.003243	192.168.0.26	192.168.0.21	UDP	71	cs-remote-db(3630) → cs-remote-db(3630) Len=29
4	0.004095	192.168.0.21	192.168.0.26	UDP	56	54046 → cs-remote-db(3630) Len=14
5	0.007436	192.168.0.26	192.168.0.21	UDP	68	cs-remote-db(3630) → cs-remote-db(3630) Len=26
6	0.008079	192.168.0.21	192.168.0.26	UDP	56	54047 → cs-remote-db(3630) Len=14
7	0.011632	192.168.0.26	192.168.0.21	UDP	66	cs-remote-db(3630) → cs-remote-db(3630) Len=24
8	0.013441	192.168.0.21	192.168.0.26	UDP	62	54048 → cs-remote-db(3630) Len=20
9	0.013880	192.168.0.21	192.168.0.26	UDP	56	54049 → cs-remote-db(3630) Len=14
10	0.016878	192.168.0.26	192.168.0.21	UDP	62	cs-remote-db(3630) → cs-remote-db(3630) Len=20
11	0.017677	192.168.0.21	192.168.0.26	UDP	56	54050 → cs-remote-db(3630) Len=14
12	0.021049	192.168.0.26	192.168.0.21	UDP	61	cs-remote-db(3630) → cs-remote-db(3630) Len=19

> Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) > Ethernet II, Src: IBM_ff:ff:00 (00:22:00:ff:ff:00), Dst: CompalIn_f1:b2:51 (08:8f:c3:f1:b2:51) > Internet Protocol Version 4, Src: 192.168.0.26 (192.168.0.26), Dst: 192.168.0.21 (192.168.0.21) > User Datagram Protocol, Src Port: cs-remote-db (3630), Dst Port: cs-remote-db (3630) > Data (24 bytes) Data: 000000000000000020000000180205000000101000000001 [Length: 24]	
--	--

Figure 24 - Wireshark non-interpreted XCom packets

A Wireshark plugin called dissector has been implemented to translate packets according to their type and sub-type and to highlight the packet values.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.21	192.168.0.26	XCOM	68	Time Syncro
2	0.000582	192.168.0.21	192.168.0.26	XCOM	56	Configuration
3	0.003243	192.168.0.26	192.168.0.21	XCOM	71	Configuration
4	0.004095	192.168.0.21	192.168.0.26	XCOM	56	Configuration
5	0.007436	192.168.0.26	192.168.0.21	XCOM	68	Configuration
6	0.008079	192.168.0.21	192.168.0.26	XCOM	56	Configuration
7	0.011632	192.168.0.26	192.168.0.21	XCOM	66	Configuration
8	0.013441	192.168.0.21	192.168.0.26	XCOM	62	Configuration
9	0.013880	192.168.0.21	192.168.0.26	XCOM	56	Pressure
10	0.016878	192.168.0.26	192.168.0.21	XCOM	62	Pressure
11	0.017677	192.168.0.21	192.168.0.26	XCOM	56	Level
12	0.021049	192.168.0.26	192.168.0.21	XCOM	61	Level

```

> Frame 10: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
> Ethernet II, Src: IBM_ff:ff:00 (00:22:00:ff:ff:00), Dst: CompalIn_f1:b2:51 (08:8f:c3:f1:b2:51)
> Internet Protocol Version 4, Src: 192.168.0.26 (192.168.0.26), Dst: 192.168.0.21 (192.168.0.21)
> User Datagram Protocol, Src Port: cs-remote-db (3630), Dst Port: cs-remote-db (3630)
< Xcom Protocol
  - MSG ID: 3
  - Length: 20
  - Type: Pressure (0x03)
  < Field Pressure
    - Type: XCOM_SEND_PRESSURE (0x01)
    - Unit: 0x284b
    - Value: 0
    - quality Code: QUALITY_CODE_SIMULATE (0x02)

```

Figure 25 - Wireshark interpreted XCom packets

4.4 Client and server test application

During this project, the fact of having only one board per person was sometimes limiting for testing different parts of the FW, notably time synchronization. So, two C programs were developed. One is called the Client test application, which simulates a secondary board, and the other is the Server test application, used to simulate the main board. These two programs are designed to communicate with a main board in the case of the first one, and a secondary board in the case of the second one. They can be used to check, for example, that the latency calculation between boards is without error or, more simply, that the XCom protocol and its module are correctly implemented.

5. Results

We tested each part of the solution in varying degrees of depth. The following parts focus on the most important and difficult part of the solution.

5.1 Time synchronization result

We tested that time synchronization allowed the pressure to be measured at the same time on each sensor. To do this, we looked at the logs of each board (main and secondary) and checked that the pressure values sent by the secondary boards arrived in the same cycle as the main one (i.e. one cycle is considered between two level calculations).

- For the latency calculation, the packets are received and sent by each board.

Main board

```
10:807 Main Task - Init clock
10:808 Main Task - Packet sent
10:809 IP_Task - Packet received
10:809 Main Task - Event : data received
10:810 Main Task - TS packet received
10:810 Main Task - Cycle packet received
10:810 Main Task - Packet sent
10:810 IP_Task - Packet received
10:810 Main Task - Event : data received
10:810 Main Task - TS packet received
10:810 Main Task - Cycle packet received
10:812 Main Task - Packet sent
10:815 IP_Task - Packet received
10:815 Main Task - Event : data received
10:815 Main Task - TS packet received
10:815 Main Task - Cycle packet received
10:816 Main Task - Packet sent
```

Secondary board

```
12:865 Main Task - Init clock
12:867 IP_Task - Packet received
12:868 Main Task - Event : data received
12:868 Main Task - TS packet received
12:868 Main Task - TS Cycle packet received
12:868 Main Task - Packet sent
12:868 IP_Task - Packet received
12:868 Main Task - Event : data received
12:868 Main Task - TS packet received
12:869 Main Task - TS Cycle packet received
12:869 Main Task - Packet sent
12:873 IP_Task - Packet received
12:873 Main Task - Event : data received
12:873 Main Task - TS packet received
12:873 Main Task - TS Cycle packet received
12:873 Main Task - Packet sent
12:875 IP_Task - Packet received
12:875 Main Task - Event : data received
12:875 Main Task - TS packet received
12:875 Main Task - TS Init packet received
```

- As regards interrupt start-up, this is correct.

```
10:816 Main Task - Init latency interrupt
10:816 Main Task - Init calculation interrupt
10:816 Main Task - Wait before launching measurement
10:820 Interrupt - Init measurement
```

```
12:876 Main Task - Wait before launching measurement
12:879 Interrupt - Init measurement
```

- Finally, the pressure measurement interruption starts at the same time because the packet of the secondary pressure value received by the hand is contained in the same measurement cycle and it is also visible that it arrives just after the pressure measurement of the board hand because of the latency of the network.


```
10:856 Main Task - Event : level calculation
10:861 Main Task - Event : pressure measurement
10:865 IP_Task - Packet received
10:865 Main Task - Event : data received
10:896 Main Task - Event : level calculation
10:900 Main Task - Event : pressure measurement
10:900 IP_Task - Packet received
10:900 Main Task - Event : data received
10:936 Main Task - Event : level calculation
```

```
12:919 Main Task - Event : pressure measurement
12:919 Main Task - Packet sent
12:959 Main Task - Event : pressure measurement
12:959 Main Task - Packet sent
```

5.2 Webpage configuration result

There are no apparent bugs when it comes to entering values on the webpage. It is possible for two configurations on two different boards to collide simultaneously, but we did not manage to get this on several tests, as the network latency would have to be high. Furthermore, the values retrieved and sent/received via XCom are completely correct. However, we did find a bug that should be avoided if the user enters a pingable IP address such as 1.1.1.1 or 8.8.8.8. To resolve this, it would be possible to force the user to enter an IP address from the desired subnet, that of the DHCP. Lastly, some fields are redundant, such as the volume in the case of tank information, which should be removed and calculated via boards with height and diameter values.

5.3 Monitoring application result

For the monitoring application, we tested the 3 different pages with one, two or three sensors. On the Send and Read page, there were no apparent bugs, the values were set by the boards and returned to the monitoring application. However, for the test page, there is an error, probably from the monitoring application, where the configuration is not properly retrieved. The application requests the configuration from one of the boards entered by the user, and the board returns the configuration with the correct values (visible via WireShark), but the application retrieves them as 0. The same applies to the level. This is a curious problem because the request/answer system is the same for the pressure value, but it works.

6. Work organization

For this project lasting over a year, we had to define a specific organization and work method, as well as deadlines to respect.

6.1 Timeline

Initially, we worked with Siemens to define a timeline for the various parts of the project and their duration. To do this, we used a Gantt chart.

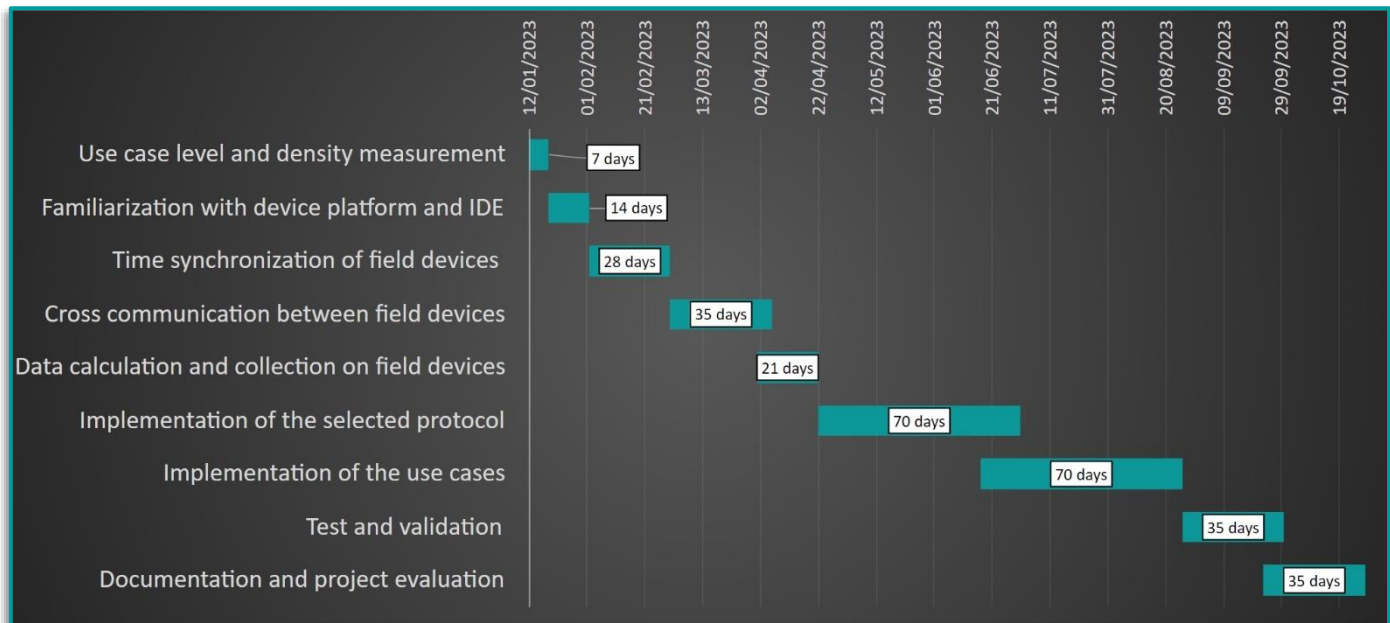


Figure 26 - Initial Gantt chart

Now that the project has come to an end, it is interesting to look at the final Gantt chart, which we did change considerably over the year.

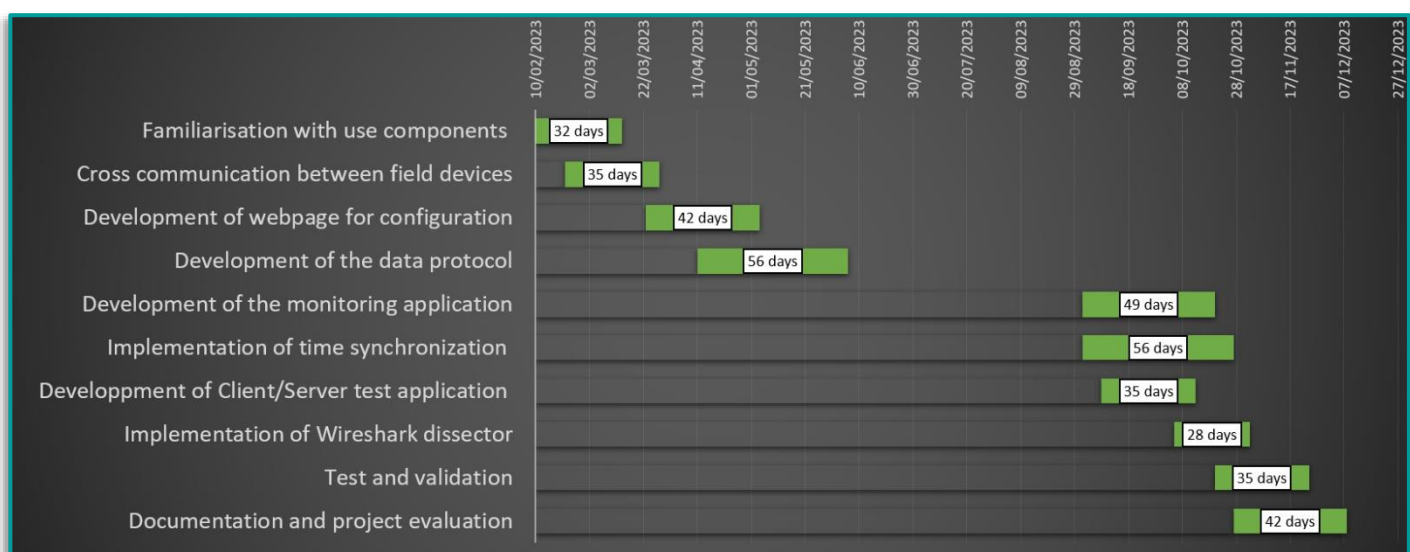


Figure 27 - Final Gantt chart

There is a great discrepancy between the two, as other parts of the project have obviously been added, such as the development of the monitoring application. In addition, the summer period was included in the project at the outset. We still tried to follow the initial timeline as closely as possible, but sometimes we realized that it was necessary to do one part before another, as in the case of the implementation of the time synchronization and the data protocol, because in order to do time synchronization, it was necessary to already have the data protocol implemented. But in the end, we managed to complete the project almost in its entirety, although we would have liked to have given more time to the testing and validation phase, particularly in correcting the last errors. And, of course, we would have liked to test the project under real conditions on real sensors.

6.2 Tools

To conduct this project, we were able to use tools provided by Siemens to facilitate organization and teamwork. Initially, we used Microsoft Teams with a channel, where we could exchange between members and supervisors, but also store all types of documents, including project documentation such as project reviews, monthly reports, project architecture diagrams etc...

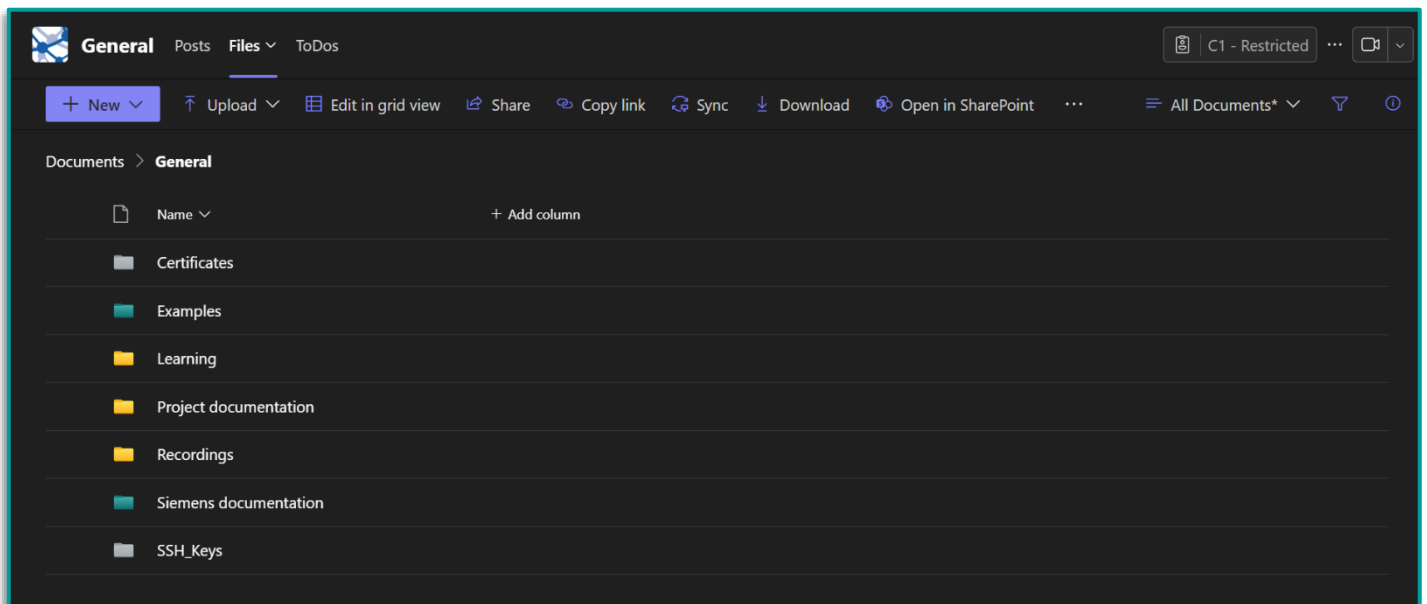


Figure 28 - Teams channel

Teams was a great asset regarding the project documentation because it allowed us to work simultaneously on the office suite such as PowerPoint or Word.

Regarding the programming part, Siemens gave us access to their Git servers. We were able to have repositories for each part of the project. This made it easier to work as a team and also to group parts together. For example, we have a repository for the data protocol and one for the board's FW, and we placed the data protocol one as a submodule in the FW one. Using Git allowed us to divide the parts. We worked with one branch per part : a branch for time synchronization, a branch for the FW architecture with tasks, tasks events and mailboxes etc...

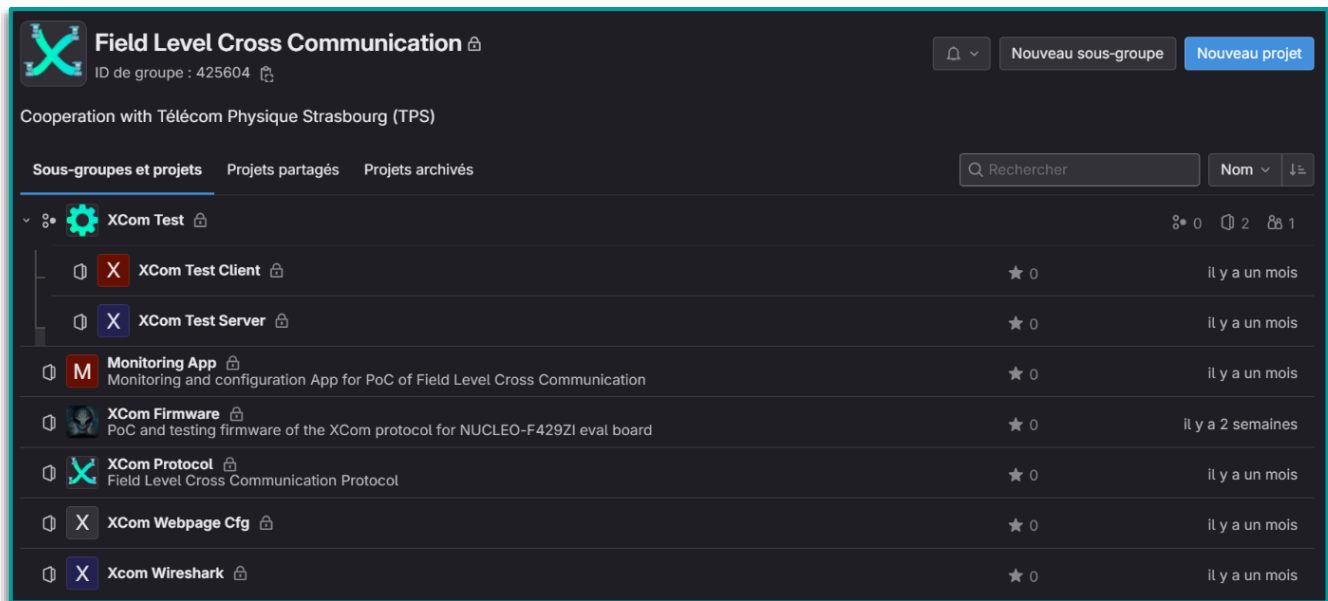


Figure 29 - Git repositories

6.3 Task assignment

Initially, we organized ourselves in pairs over the first 6 months with Thomas Dumond and Ethan Huret the number 1 pair, and Louis Duval and Nicolas Cipolla the number 2 pair. At the very beginning of the project, we all worked on familiarization with the components used. After this step, pair number 1 focused on the configuration web page and the data protocol, and pair number two investigated the possibilities for the FW architecture with embOS and time synchronization methods.

Secondly, Nicolas Cipolla went on mobility and could therefore no longer work on the project, so we redistributed the tasks on a basis of one task per person. Ethan and Thomas worked on time synchronization, Ethan worked on the monitoring application, Thomas worked on the Client and Server test applications, and Louis worked on the Wireshark dissector. As for testing and documentation, they were done by everyone.

Finally, it is difficult to quantify the number of hours spent on the project, however, it is possible to estimate a range of between 200 and 400 hours of work per person.

6.4 Daily organization

Regarding our daily organization, we had, for example, one Teams meeting per week with Jochen Balduf to present the progress of the project. We also had debugging meetings, when necessary, with also him, who could provide us with his experience on the problems we encountered. In the first phase of the project, the work was often done by the pairs together in person. So, we also had meetings with at least one person from each pair to explain to each other what the progress was, the blocking points, etc. Then in the second phase, the Teams meetings allowed everyone to present their work. Regarding the production of the regular/monthly engineering project documentation, we usually did the work (report and presentation) a week in advance, which sometimes allowed us to make a first presentation in front of our Siemens supervisors.

7. Conclusion

In conclusion, our journey through this project has been an enriching experience, offering a blend of technical exploration and skill development. We ventured into the intricacies of engineering, delving deep into the realm of communications protocols, hardware integration, and software development. One of the key achievements was the design of a lightweight communications protocol, XCom, adapted to the precise needs of the project. This protocol not only facilitated efficient data exchange, but also illustrated our ability to adapt and innovate in the technical field. The spirit of collaboration among team members, coupled with the unwavering support and guidance of Siemens mentors, highlighted the importance of effective teamwork and mentoring in overcoming business challenges. Our exploration of firmware architecture, time synchronization and more generally industrial embedded programming demonstrates our ability to adapt and our problem-solving skills. Identifying and fixing bugs highlighted the importance of meticulous testing and validation processes to ensure robust solutions. Additionally, the project timeline, task assignment, and use of tools such as Microsoft Teams and Git repositories have highlighted the importance of effective project management methodologies in large-scale engineering projects. On a personal level, this project was the cornerstone of our academic journey, fostering not only technical expertise, but also the development of crucial soft skills. Effective communication, adaptability and resilience were essential to overcoming challenges and achieving our project objectives. Looking forward, the lessons learned, and skills acquired during this project will undoubtedly serve as a solid foundation for our future endeavours in the field of engineering. In conclusion, we express our gratitude to everyone who contributed to this project.