

An improvement on Arduino-based system for environmental monitoring in the CMS detector

*D. Spessot**

Delft University of Technology, The Netherlands

Abstract

This paper studies the possibility for an upgrade of the existing implementation of the sensor network deployed for environmental monitoring at the Compact Muon Solenoid detector (CMS). The proposed implementation introduces a more reliable error handling protocol and a higher reliability, made possible by the integration of the measurements into the PLC-based Tracker control system.

*Corresponding author: davide.spessot@mail.polimi.it

Contents

1	Introduction	3
2	Requirements	5
3	Design	5
3.1	Hardware architecture	5
3.2	Current software architecture	5
3.3	Proposed software architecture	6
3.4	Environment guide	15
4	Test plans	15
5	Project issues and further improvements	16
6	Conclusions	16

1 Introduction

The CMS Si-Tracker has strong cooling requirements ($>80\text{kW}$) that are more critical as the irradiation of the detector and the material damage due to it accumulate with time. The cooling system is running continuously, however, as it crosses parts of the CMS detector and the Tracker, it can be partially or completely off while the coolant is circulating adding to the complexity. In these conditions, a strict environmental control is necessary in order to avoid critical elements in the assembly to reach the dew point and water to be formed on miniature electronic structures with catastrophic effects. The environmental control is realized by deploying a network of sensors (pressure, humidity, temperature). These sensors are organized in two nets that fulfill different roles:

- the main sensor net, used for the real-time control of the detector. This network is a PLC-based, fail-safe environmental control, based on a limited (~ 600) environmental sensors with weight put on their reliability, radiation tolerance and redundancy attached to the Detector Safety System (DSS).
- the secondary sensor net, used for less critical monitoring and signal analysis. The sensors of this network are not designed to be radiation tolerant and are relatively inexpensive, for this reason their number is higher than in the main network.

The secondary sensor network is the focus of this paper.

The monitoring system is built upon the previous implementation [1], that consists in a sensor set connected to Arduinos provided with an Ethernet board (Fig. 1). In the proposed system the previous sensor set is kept, together with the HW and most of the SW architecture of the sensors to μC connection. The μC to higher level connection, however, is changed (Fig. 2). This connection was previously implemented with an **online part** and an **offline part**. In the online part, the Arduino uploads the sensor data to its own website. In offline part, the machine tk-webmonitor of the CMS cluster runs a series of scripts that uploads and processes the data, in order to be visualized in a GUI. Now, this whole system is replaced by a direct Ethernet connection between μC and PLC, using the Settimino library [3]. In other words, in the proposed system the sensor data is read by the Arduinos and written directly in the PLC memory. A SCADA system then processes the sensor data further.

The software was developed with the help of GitLab/CERN services, and the source is now available through the following link: <https://gitlab.cern.ch/cms-ardPLC/PLC-arduino-CMS>.

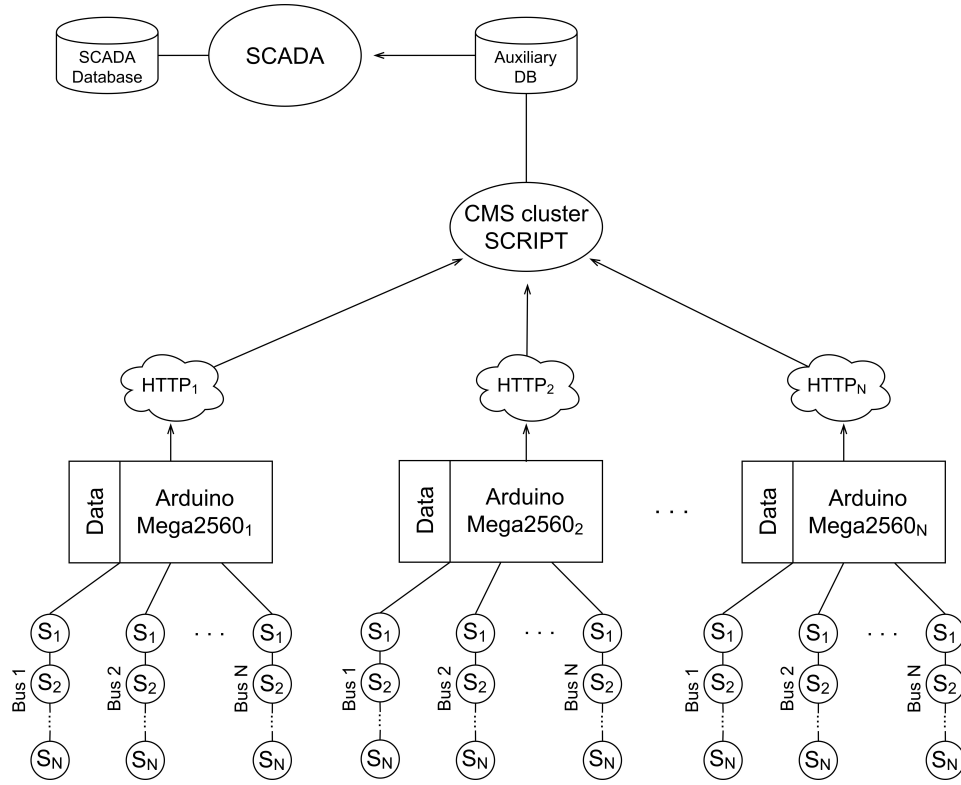


Fig. 1: Diagram of the original system.

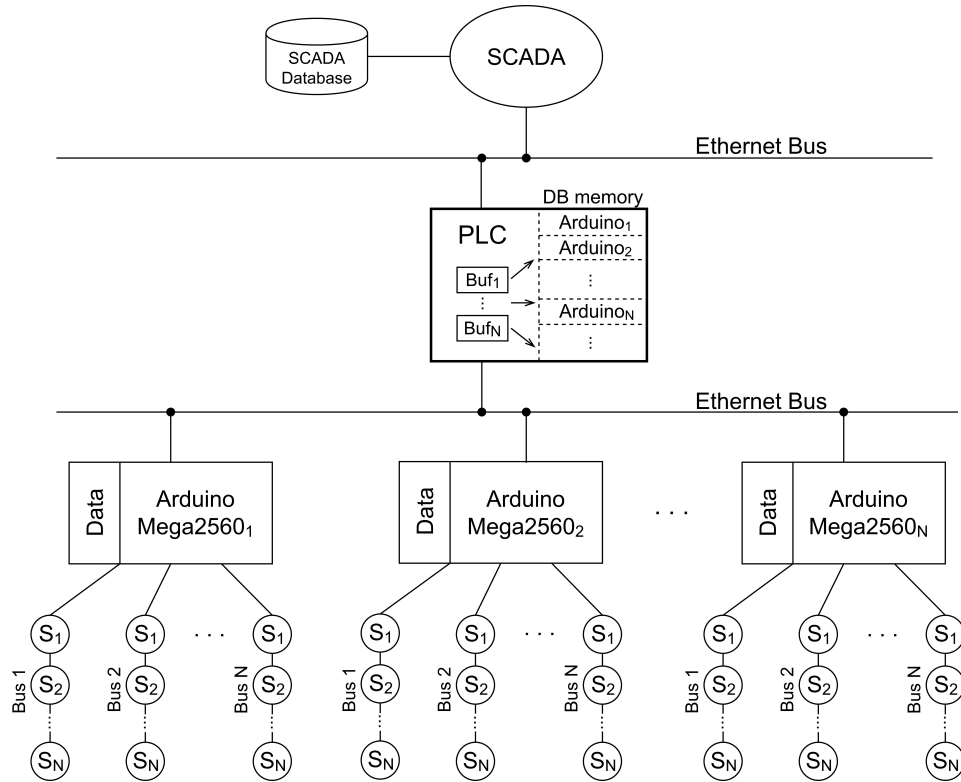


Fig. 2: Diagram of the proposed system.

2 Requirements

In addition to the new uC to PLC direct connection, that is the main focus of the project, the new firmware is required to fulfill some functions that are missing in the current implementation:

- Provide a scalable, re-configurable and maintainable environment. The system should be fully tunable from the PLC side, without the need of reprogramming the firmware.
- Detect any sensor disconnection after the first scan and reconnect the sensor when it becomes available again;
- Errors and exceptions should raise a flag in the PLC memory and, in case of critical errors, the system should fail safely;
- Avoid dynamic memory allocation;
- Provide a maximum sampling frequency of at least $0.1Hz$.

3 Design

3.1 Hardware architecture

The data acquisition system is the same as described in the original paper [1]. To summarize, 13 Mega2560 Arduino boards are hosted inside 2 crates of the X2N37 and X2F38 racks in the UXC. A W5100-based Ethernet shield is used to link every Arduino board with the network switch contained in the crates. The Mega2560 and the Ethernet shield are plugged into the opposite sides of a custom made Connector Board. This board hosts the sensors buses.

There are two kinds of sensors that are currently deployed in the system: the first, DS18B20, is a digital temperature sensor and the second, HIH4030, is an analogue humidity sensor, that has to be coupled with the DS2438 battery controller chip for digital conversion. The DS2438 chip also contains an internal temperature sensor, that is also used for temperature monitoring.

3.2 Current software architecture

3.2.1 Overview

The current software is compiled through the help of the Arduino Makefile [2], instead of using the built-in IDE. The makefile is supposedly portable and supports every major OS, but there are some complications that make it more challenging to compile it on Windows.

The core is based around three libraries: Webduino, OneWire and Ethernet/Ethernet2. The Webduino library [4] is a web server library for the Arduino platform. The Ethernet/Ethernet2 libraries [6] are used to enable Ethernet communication through the external shield. The difference between the two is that Ethernet supports Arduino shields with the W5100 embedded Ethernet controller, while Ethernet2 supports the W5500 version. The custom-made library owKam, that contains the definition of the main classes and functions used for storing and processing the sensor data. All the sensor data is structured in two linked lists as it will explained in the following section. The OneWire library [5] is used in order to communicate with the sensors, since all the sensors support 1-Wire bus connection. OneWire's functions are called by several owKam functions. These functions are a modified version of the DallasTemperature library [7] functions.

3.2.2 High level description of the current software

In the setup() routine, first the digital pins are scanned consecutively for new sensors. When a new sensor is found, its address is read. Depending of the first byte of the address the software is able to recognize whether the sensor is a temperature sensor (DS18B20) or a humidity sensor (HIH4030+DS2438). Two dynamically allocated linked lists are used to store the sensor data, comprised of sensor address, pin

number, values, error counter. One list contains only temperature sensors data and the other only humidity sensors data. If the sensor has been recognized, its address will be saved in a dynamically allocated node at the tail of the corresponding list. Lastly, the resolution of all the temperature sensors is set.

The `loop()` function's role is to communicate with the sensors, request temperature conversion, wait for the conversion and read the data. Then the data has to be processed in HTML form, in order to be read by the CMS cluster.

Whenever a DS2438 device appears disconnected, the error counter of the list node associated to it is increased by one and its value fields are set to '-100'. When it becomes visible again, the error counter is decreased by one. The same is not true for the DS18B20, since the increase counter instruction for the error counter is missing, making it effectively devoid of exception handling.

After all of this, the DS2438 output has to be further processed. This is because its humidity output is encoded as two voltage values, the outputs of its two ADCs. In order to calculate the actual humidity and dew point, a conversion algorithm is performed [8].

As a final note, all the sensor values are produced by the sensor in fixed point, and immediately converted, stored and processed in floating point representation.

3.3 Proposed software architecture

3.3.1 Overview

As a guideline for software development, it was decided to keep the original software architecture where possible, adapting it to the new requirements where necessary. In practice, this meant that the library used for sensor communication, `owKam`, was kept, although modified where necessary. The structure of the read sensors routine was also kept.

The first step in the development of the proposed software was to tidy up the project environment, identifying and removing unused functions, declarations and libraries and adding comments and descriptions to classes and functions. Due the many versions of the source coexisting in the main folder, the most recent version was kept, and all the others were removed.

The second step was removing the code dependency to the Arduino makefile and reverting it to the original IDE, in order to make it more portable for a Windows-based machine, that is the standard for the current CERN computing network.

The `Webduino` library, before used for the communication between the uCs and the CMS cluster, and all the related classes and functions were removed from the code and their role taken by the `Settimino` library [3], that enables direct Arduino/PLC communication. The main read/write routines of this library are the twin functions `ReadArea()` and `WriteArea()`. These functions take as input a void pointer to an array, that works as a buffer during the communication with the PLC. `WriteArea()` takes this buffer as the source of the data to write in the PLC, while `ReadArea()` uses it as its destination.

The library `logging` has been developed, in order to properly encode and decode the content of the communication buffer, considering also the difference in endianness between the Arduino (small-endian) and the PLC (big-endian).

The header `constants_main.h` contains all the parameters defining the execution of the firmware, like the data blocks, the maximum number of sensors, the size of the communication buffer, the resolution of the DS18B20 devices.

The library `owKam`, as explained before, was updated. In the proposed version it introduces 2 classes:

- `newSensor` is the abstraction of a sensor. It contains a sensor address, the data type of the contained data, a single 16 bit sensor value, an 8 bit flag variable, a 32 bit timestamp, the pin number where the sensor is connected, a pointer to the OneWire bus associated with said pin. All the sensor data waiting to be transferred to the PLC memory will be stored in a linked list of objects of this class;
- `owPin` represents the digital pins hosting a OneWire bus. It contains pin number, pointer to OneWire bus and the number of sensors found on the pin divided in total, temperature and humidity.

The last developed library is `PLCcommunication`, with its classes:

- `staticslot` contains the data read from the PLC, consisting of the parameters necessary for execution. The role of these parameters will be further explained in the communication protocol paragraph;
- `ArduinoDBData` contains all the data that will be written in the PLC memory in the next cycle. This data contains parameters such as the number of sensors, the handshake and sampling timestamps. Moreover, it contains the pointer to the linked list of `newSensor` objects.

As a final note, the firmware makes use of the Atmel 2560 timer interrupt [9] in order to schedule its periodic tasks and provide a stable execution frequency.

3.3.2 Communication protocol

The communication protocol between PLC and Arduino will make use of the S7 Telegram protocol, as explained in the Settimino protocol documentation [3].

The reading and writing routines are organized by allocating the PLC DB memory into three main areas:

- Static DB, written by the PLC and read by the Arduinos and is used to configure their execution parameters.
- Dynamic DB, written by the Arduinos and read by the PLC, contains all the sensor data. Every uC has a dynamic memory space associated, so the entire system's dynamic memory is considerably bigger than the static memory, and spans across several DBs.
- Error DB, written by the Arduinos and read by the PLC, contains the data of the Arduinos that experienced a fatal error during execution and need further user input to solve the problem.

The **static DB memory** (Table 1) contains:

- A single public variable, the total number of Arduinos in the system;
- A private space for every Arduino, containing:
 - MAC address of the Ethernet shield;
 - DB number and byte offset of where the Arduino should begin writing;
 - the maximum amount of free slots available to the Arduino to fill with sensor data;
 - the sampling time, expressed in milliseconds;
 - a 2 bytes space for flags (Table 2).

Static MEM - DB700		
	Memory description	Datatype
Slot 1 (Arduino 1)	Number of slots	INT
	MAC Address ₁	ARRAY [1..6] of Byte
	DB number ₁	INT
	DB offset ₁	INT
	Max. Entries ₁	INT
	Sampling Time ₁	DINT
	Flags ₁	INT
Slot N (Arduino N)
	MAC Address _N	ARRAY [1..6] of Byte
	DB number _N	INT
	DB offset _N	INT
	Max. Entries _N	INT
	Sampling Time _N	DINT
	Flags _N	INT

Table 1: Fixed PLC memory variables declaration

BIN	Meaning
0b0000000000000001	Reset request
0b0000000000000010	Reset acknowledge
0b0000000000000100	Sensor buses rescan request
0b0000000000001000	Sensor buses rescan acknowledge
0b0000000000010000	Allocated memory insufficient for all the sensors found on buses
0b0000000000100000	New sensors found during rescan

Table 2: Static memory flags.

All the variables defined in the static DB memory have to be initialized to the correct value by the PLC before the Arduinos are allowed to start their execution. The only exception to this are the flag variables, where only the first and third bit should be changed by the PLC.

The **dynamic DB memory** (Table 3) allocated for a single Arduino follows the following structure:

- A header, that contains:
 - The slot index of the current Arduino, as introduced in the static slot. This parameter is written as a mean for the PLC to recognize the sensor data;

- The number of sensors currently online;
- A handshake variable, that is 1 when the data is not ready and 0 when the data is ready;
- Two timestamp fields, one signifying the beginning of the sensor reading and the other the end. For both of them, the unit is milliseconds.
- A space for every sensor entry, that contains:
 - The sensor ID;
 - The Arduino digital pin number where the sensor is connected;
 - The sensor value type. If the value is produced by the DS18B20 it's '1', if instead is produced by the DS2438, depending on whether it's the temperature, VAD or VDD value, is '2', '3' or '4';
 - The sensor value, in fixed point representation according to the specifications of the sensors datasheet;
 - A 2 bytes space for flags (Table 4).

Dynamic MEM - DBX		
	Memory description	Datatype
Header	Slot n°	INT
	n° of sensors	INT
	Handshake	INT
	Timestamp 0	UNSIGNED LONG
	Timestamp 1	UNSIGNED LONG
Sensor 1	Sensor ID ₁	ARRAY [1..8] of Byte
	Pin number ₁	
	Value type ₁	
	Sensor value ₁	
	Flags ₁	
Sensor N
	Sensor ID _N	ARRAY [1..8] of Byte
	Pin number _N	
	Value type _N	
	Sensor value _N	
	Flags _N	

Table 3: Dynamic PLC memory variables declaration

BIN	Meaning
0b00000001	Data ready
0b00000010	Incorrect value conversion due to insufficient time
0b00000100	Sensor is unconnected
0b00001000	DS18B20 temperature precision incorrect
0b00100000	Sensor class uninitialized (firmware error)

Table 4: Dynamic memory flags.

All the variables defined in the dynamic DB are written by the Arduino and read by the PLC. The "data ready" flag, however, can be used by the PLC as a token, and put to zero after the new values have been consumed.

The **error DB memory** (Table 5) contains:

- A variable, indicating the number of failed Arduinos in the system;
- A flag for requesting a reset to all faulty Arduinos if set to 1;
- A private space for every Arduino, containing:
 - MAC address of the Ethernet shield;
 - Number of sensors connected to the Arduino;
 - A 2 bytes space for flags (Table 6).;

This area of memory works as a circular buffer, such that if the memory is insufficient for saving the information of all the faulty Arduinos the information of the Arduinos added first will be overwritten.

Error MEM - DB799		
	Memory description	Datatype
Slot 1 (Arduino 1)	Number of faulty Arduinos	INT
	Reset request	INT
	MAC Address ₁	ARRAY [1..6] of Byte
	n° of sensors ₁	INT
	Flags ₁	INT
Slot N (Arduino N)
	MAC Address _N	ARRAY [1..6] of Byte
	n° of sensors _N	INT
	Flags _N	INT

Table 5: Error PLC memory variables declaration

BIN	Meaning
0b00000000000000000001	The static PLC DB was not correctly defined
0b00000000000000000010	The Arduino was not found in the static DB
0b00000000000000000100	The Arduino was found 2 or more times in the static DB
0b000000000000010000	The dynamic slot given is ill defined and doesn't correspond to the static slot definition
0b0000000000010000	In the dynamic DB there is not enough space for all the sensors found

Table 6: Error memory flags.

The contents of the error DB memory are all written by the Arduinos and read by the PLC. The only exception is the "reset request variable".

The flags considered in the static, dynamic and error DB areas, unfortunately can't cover errors caused by a failure in the S7 protocol, for obvious reasons (if no communication is possible, the error codes can't be written in the PLC memory). The proposed firmware makes possible to find and debug these errors via serial connection. In fact by connecting serially, the user can get access of the variable used internally to store every error code.

This error variable is an extension of the error returned by the Settimino library, as it adds more error codes to the error variable returned by Settimino functions (Table 7).

HEX	Meaning
0x0000	No error
0x0001	TCP Connection error
0x0002	Connection reset by the peer
0x0003	A timeout occurred waiting a reply
0x0004	Ethernet driver returned an error sending the data
0x0005	Ethernet driver returned an error receiving the data
0x0006	ISO connection failed
0x0007	ISO PDU negotiation failed
0x0008	Malformed PDU supplied
0x0010	Static PLC DB is smaller than definition
0x0020	Static DB does not contain the Arduino's MAC
0x0030	Static DB contains multiple copies of the Arduino's MAC
0x0040	The dynamic slot given is ill defined
0x0050	In the dynamic DB there is not enough space for all the sensors found
0x0100	Invalid PDU received
0x0200	Error sending a PDU
0x0300	Error during data read
0x0400	Error during data write
0x0500	The PLC reported an error for this function
0x0600	The buffer supplied is too small
0x1000	One or more sensor read request faster than conversion time.
0x2000	One or more sensor has disconnected.
0x3000	One or more sensor has 0x0010 and 0x0020 error.
0x4000	One or more DS18B20 sensors resolution is incorrect

Table 7: System's error codes.

3.3.3 High level description

The execution flow of the `setup()` is similar to that of the original architecture, in the way the sensors are consecutively scanned and their value saved. The only difference is that now only one statically allocated list is used to store the sensor information, instead of two.

After this, the static DB memory is read, and its correctness of definition is checked. Then the sensors resolution is set for all the temperature sensors. If the definition of the static DB is incorrect, the Arduino will increase the number of faulty Arduinos variable in the error DB and write its own address together with its data in a slot. Then it will wait for a reset request.

The `loop()` function is also similar to the original version of the code, except for the addition of the the periodic read of the static DB, and for the fact that all the periodic tasks are scheduled by the corresponding interrupt routines. Two interrupt service routines are defined, one for managing the sensor sampling, and one for managing the read static DB routine. Every sensor sampling cycle is followed by a write operation to the dynamic DB area, and all the PLC communication functions are wrapped in a error checking condition that deals with errors and updates the corresponding communication flags in the PLC memory, according to the protocol described in the previous section.

3.3.4 Data structures

The main data structure introduced by the proposed software is the linked list of `newSensor` objects that stores all the sensor information. The nodes of the list structure are statically defined as a vector (Fig. 3). In order to represent a physical sensor that produces more than one value it's necessary to use as many

`newSensor` objects as the number of values produced by the sensor. We need to do this because a universal sensor representation is a requirement of this project, as it will make the design of the SCADA system much simpler. In our sensor set the HIH4030+DS2438 sensor produces 3 values, one representing temperature and two of them encoding the humidity in the form of voltage value. For this reason, 3 `newSensor` objects are introduced for such sensor. The DS18B20, instead, produces only one temperature value, so one object is sufficient.

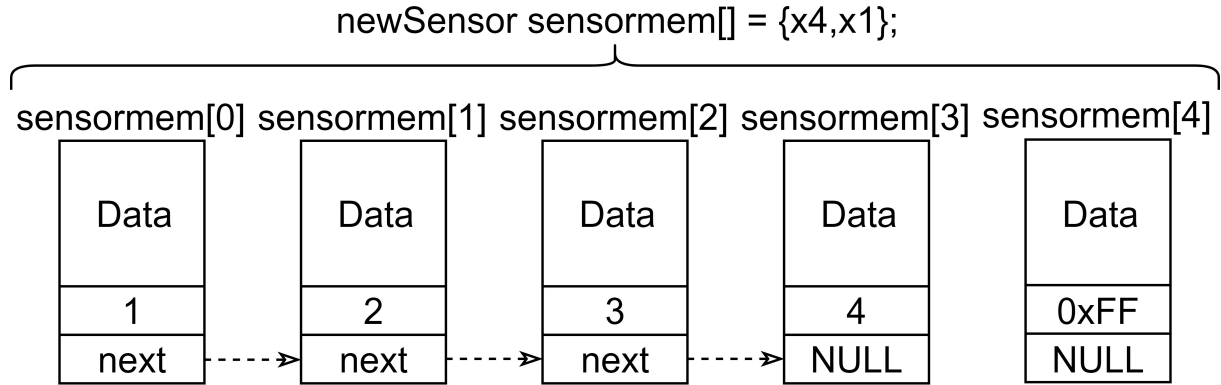


Fig. 3: Linked list in case of a bus containing one DS18B20 sensor and one HIH4030+DS2438 sensor. The type variable is shown separately from the data to highlight the structure.

The last important data structure is the communication buffer used for the write operation in the dynamic slot of the PLC.

The Settimino's `WriteArea()` function requires as an input a void pointer to an array, whose data, during the function call, will be copied in the PLC memory. This array is the communication buffer. The write function accepts a buffer of any dimension as an input, but since the size of the sensor data is variable, it's impossible to allocate a big dimension array and complete the write in one `WriteArea()` function call. For this reason, the data has to be divided into smaller packages to feed as input for the `WriteArea()` function. In the current implementation, it was decided to choose as packet dimension the size of the S7 data packet (240 bytes). In this way, we ensure optimal write speed by using the totality of the available packets size. Another advantage is that the communication buffer can be set arbitrarily, depending on the available RAM, as long as it's a multiple of the S7 data packet.

An algorithm was implemented to perform the packaging, that requires another auxiliary buffer of the size of 16 bytes. First of all (Fig. 4) the sensor data structures are encoded one by one in the communication buffer. When the encoding operation reaches the limit of the communication buffer, the encoding operation halts and the excess bytes are stored in the auxiliary buffer.

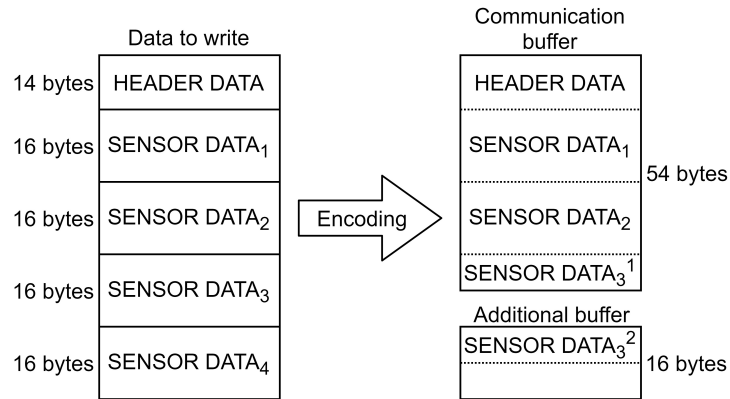


Fig. 4: Phase 1 of the write operation.

Then (Fig. 5) the function `WriteArea()` is called and the content of the communication buffer is written in the PLC memory.

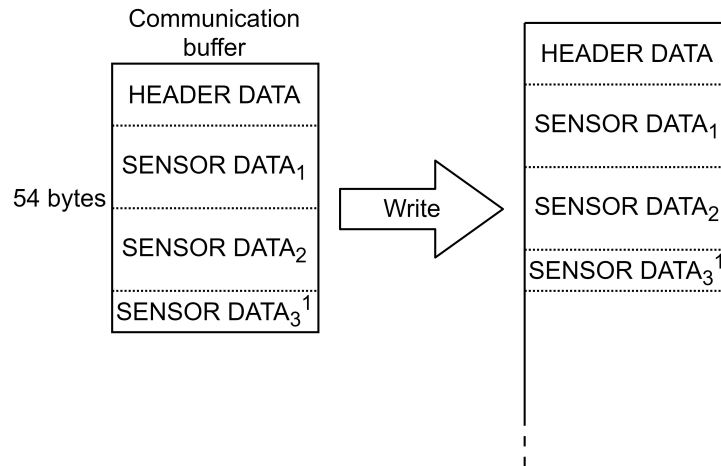


Fig. 5: Phase 2 of the write operation.

In the next phase (Fig. 6), the content of the communication buffer is reset, and the bytes contained in the auxiliary buffer are copied into it. The encoding continues and phase 1 and 2 are repeated until the data to encode is exhausted.

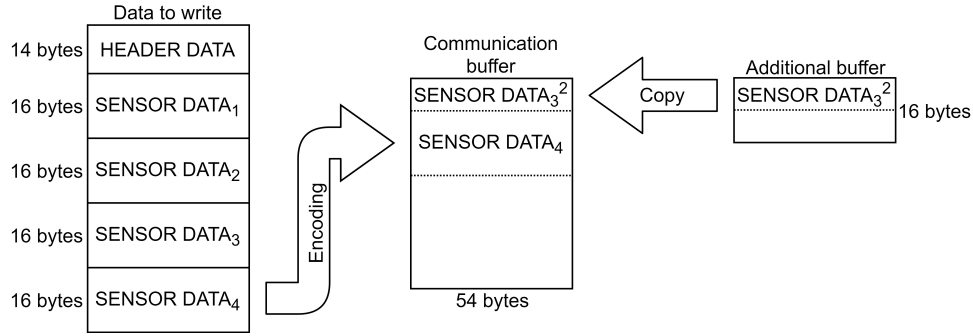


Fig. 6: Phase 3 of the write operation.

Finally (Fig. 7), the last WriteArea() function call is performed, and the last encoded data is written in the PLC dynamic memory.

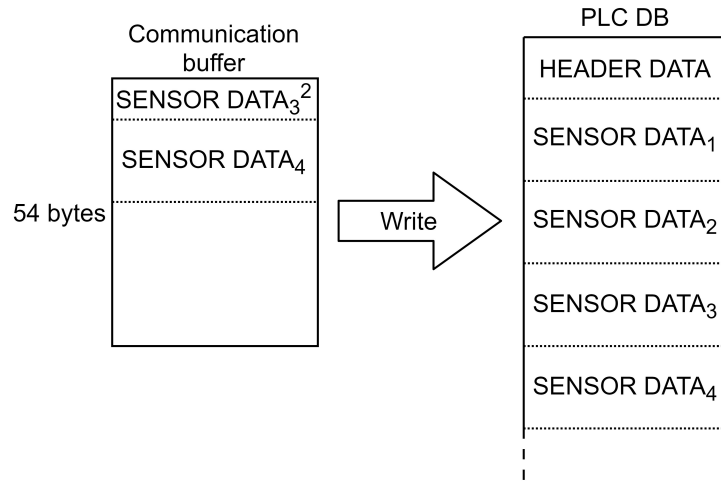


Fig. 7: Phase 4 of the write operation.

3.3.5 Features

The current firmware has several features and error handling routines:

- Two serial debug modes. In the first one all the important system data are reported, like static DB data, sensor data, number of sensors connected, IP addresses. In the second one, instead, only the system errors variable is printed;
- Ability by the PLC to ask for a complete sensors rescan request. This enables the user to add new sensors during execution or debug sensors disconnections;
- Sensors sampling time set by the PLC in milliseconds;
- General reset request for all the faulty Arduinos listed in the error DB;
- DS18B20 resolution error recovery. In case of a DS18B20 disconnection, the system re-initializes its resolution as soon as it comes back online;
- Definition correctness check for Static and Dynamic slots.

3.4 Environment guide

3.4.1 Setup on Arduino

- Add in `firmware_main.h` the complete list of Arduino serial numbers and their MAC address according to the template, and change in `constants_main.h` the serial number definition according to the serial number of the current Arduino;
- In `constants_main.h`, line 18, change the static data update period if necessary;
- In `constants_main.h`, line 21 and 22, change the static slot and error slot numbers if necessary;
- In `firmware_main.h`, line 51 and 52, change the digital pins that will be used as buses according to the template;
- In `firmware_main.h`, line 54 and 55, insert the IP addresses of the PLC and the Arduino.

3.4.2 Setup on PLC

- Define the Data Blocks according to the communication protocol;
- Initialize the dynamic and error blocks to zero and the static block with all the parameters necessary for the Arduino execution;

3.4.3 During execution

- Change the static slot variables according to necessity. The minimum stable sampling period is around 1500 ms, as will be shown in the next section;
- Reset to zero the dynamic slot memory and the error slot after every system reset.

4 Test plans

The developed features and error handling routines were tested on a prototype system with two physical sensors (one of each kind) on the same OneWire bus. Moreover, the firmware's sampling time was analyzed with MATLAB, in order to check for its properties.

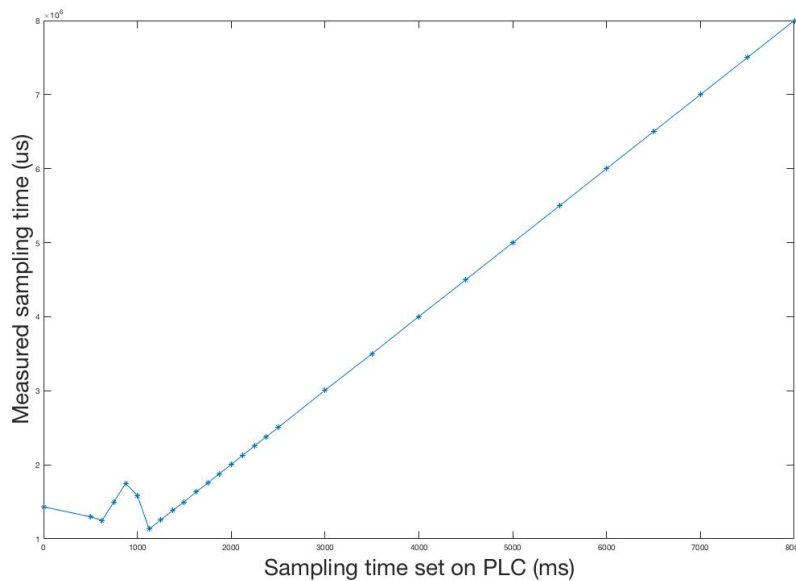


Fig. 8: Measured sampling time as a function of the sampling time set on the PLC.

As we can see from the graph Fig. 8, the sampling time is very stable for every value above 1500 ms, but below that it starts to lose its linearity. This is because the sampling routine itself has an execution time of around one second, so any value below that will saturate the sampling frequency. This phenomenon was observed with a system consisting of two sensors, so it can accentuate in case of a bigger system, causing the maximum frequency of the system to be lower. Even in this case, however the effect is not expected to contradict the requirement of a sampling frequency of 0.1Hz, because most of the sampling routine's execution time is constituted by wait operations, as the Arduino waits for the sensor conversions.

5 Project issues and further improvements

The biggest issue with the project is that it has not been tested with the final system. Moreover, the testing didn't go beyond just two sensors, one for type on a single bus. This is because in the time period the firmware was developed, the CMS temperature control system and the LHC were online. Therefore another round of testing is recommended, before applying the firmware on the final system.

6 Conclusions

In conclusion, the proposed firmware enables a direct communication between PLC and Arduino, while being able to meet, during testing, all the system requirements. Moreover, it provides an environment where the sensor set can easily be extended. Because of this, the system can easily be modified, and applied for monitoring applications outside the original project objectives.

References

- [1] F. Palmonari, E. Butz, A. Kaminskiy, Arduino for monitoring in the CMS experiment (Github link: <https://github.com/palmogit/P5-arduino-Firmware>).
- [2] Arduino makefile (<https://github.com/sudar/Arduino-Makefile>).
- [3] D. Nardella, Settimino library (<http://settimino.sourceforge.net/>).
- [4] Ben Combee, Ran Talbott, Christopher Lee, Martin Lormes, Francisco M Cuenca-Acuna, Webduino library (<https://github.com/sirleech/Webduino>).
- [5] Jim Studt, OneWire library (https://www.pjrc.com/teensy/td_libs_OneWire.html).
- [6] Arduino, Ethernet library (<https://github.com/arduino-libraries/Ethernet>).
- [7] Miles Burton, DallasTemperature library (<https://github.com/milesburton/Arduino-Temperature-Control-Library>).
- [8] Dew point algorithm used by the current firmware (http://irtfweb.ifa.hawaii.edu/~tcs3/tcs3/Misc/Dewpoint_Calculation_Humidity_Sensor_E.pdf).
- [9] Atmel 2560 datasheet (<http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561-datasheet.pdf>)

Appendices

The screenshot shows the SIMATIC Manager interface for a SIMATIC 300 Station. The main window displays the 'Variable Declaration' table for a static block. The table lists variables from address 0.0 to 42.0, including slots for Arduino IDs, BN, BO, and various sensors and flags. The status bar at the bottom indicates the PLC is in RUN mode.

Address	Name	Type	Initial value	Actual value	Comment
0.0	NumberOfSlots	INT	0	16	
2.0	SlotData[1].ArduinoID[1]	BYTE	B#16#0	B#16#2C	
3.0	SlotData[1].ArduinoID[2]	BYTE	B#16#0	B#16#F7	
4.0	SlotData[1].ArduinoID[3]	BYTE	B#16#0	B#16#F1	
5.0	SlotData[1].ArduinoID[4]	BYTE	B#16#0	B#16#08	
6.0	SlotData[1].ArduinoID[5]	BYTE	B#16#0	B#16#14	
7.0	SlotData[1].ArduinoID[6]	BYTE	B#16#0	B#16#CE	
8.0	SlotData[1].ArduinoBN	INT	0	702	DE
10.0	SlotData[1].ArduinoBO	INT	0	0	By
12.0	SlotData[1].MaxSensors	INT	0	128	Ma
14.0	SlotData[1].CycleTime	DINT	L#0	L#2000	Re
18.0	SlotData[1].Flags	INT	0	0	
20.0	SlotData[2].ArduinoID[1]	BYTE	B#16#0	B#16#2C	
21.0	SlotData[2].ArduinoID[2]	BYTE	B#16#0	B#16#F7	
22.0	SlotData[2].ArduinoID[3]	BYTE	B#16#0	B#16#F1	
23.0	SlotData[2].ArduinoID[4]	BYTE	B#16#0	B#16#08	
24.0	SlotData[2].ArduinoID[5]	BYTE	B#16#0	B#16#14	
25.0	SlotData[2].ArduinoID[6]	BYTE	B#16#0	B#16#DA	
26.0	SlotData[2].ArduinoBN	INT	0	701	DE
28.0	SlotData[2].ArduinoBO	INT	0	0	By
30.0	SlotData[2].MaxSensors	INT	0	128	Ma
32.0	SlotData[2].CycleTime	DINT	L#0	L#500	Re
36.0	SlotData[2].Flags	INT	0	0	
38.0	SlotData[3].ArduinoID[1]	BYTE	B#16#0	B#16#2C	
39.0	SlotData[3].ArduinoID[2]	BYTE	B#16#0	B#16#F7	
40.0	SlotData[3].ArduinoID[3]	BYTE	B#16#0	B#16#F1	
41.0	SlotData[3].ArduinoID[4]	BYTE	B#16#0	B#16#08	
42.0	SlotData[3].ArduinoID[5]	BYTE	B#16#0	B#16#14	

Press F1 to get Help. RUN Abs < 5.2 Rd

Fig. .1: PLC definition of static block.

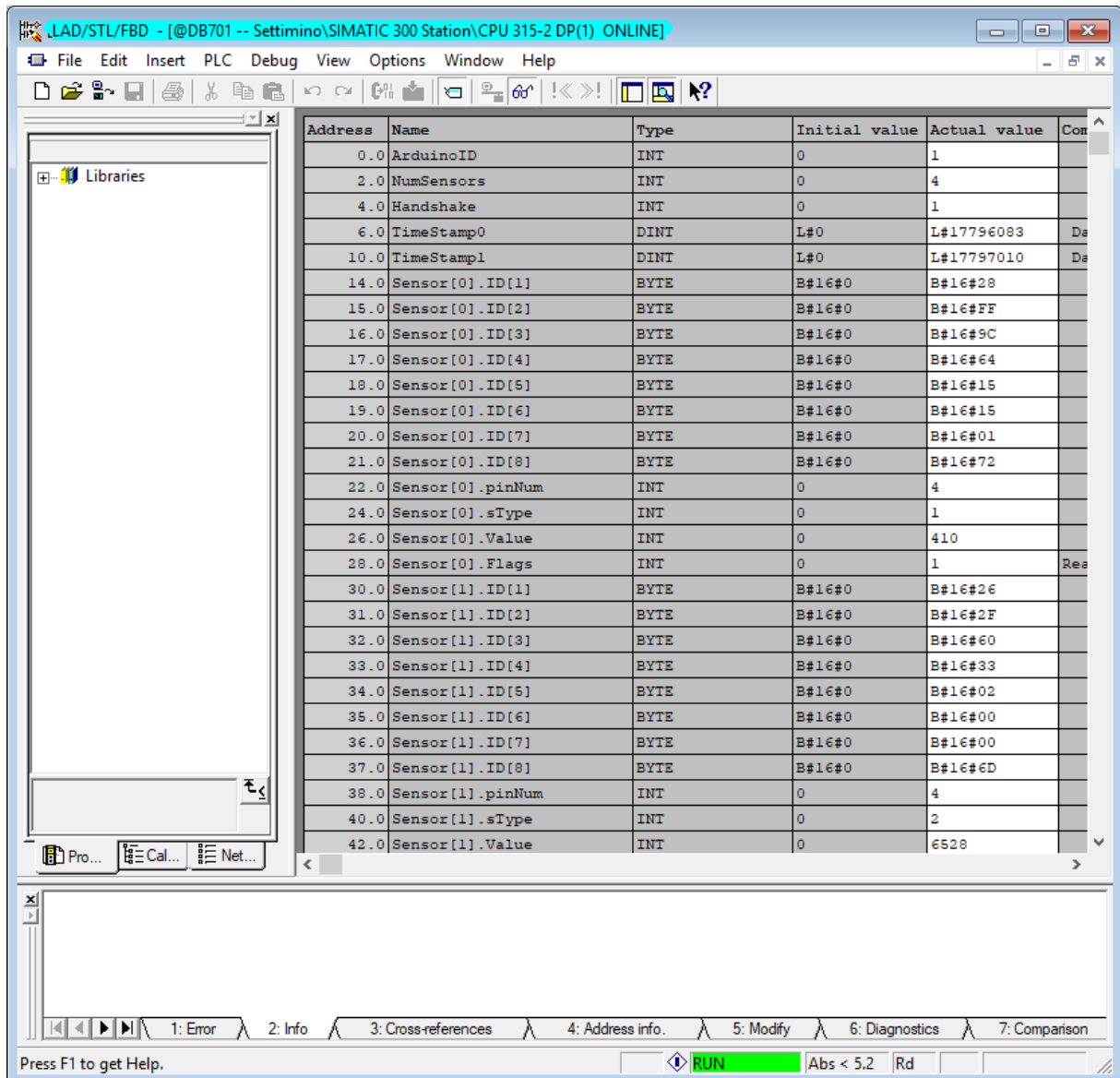


Fig. .2: PLC definition of dynamic block.

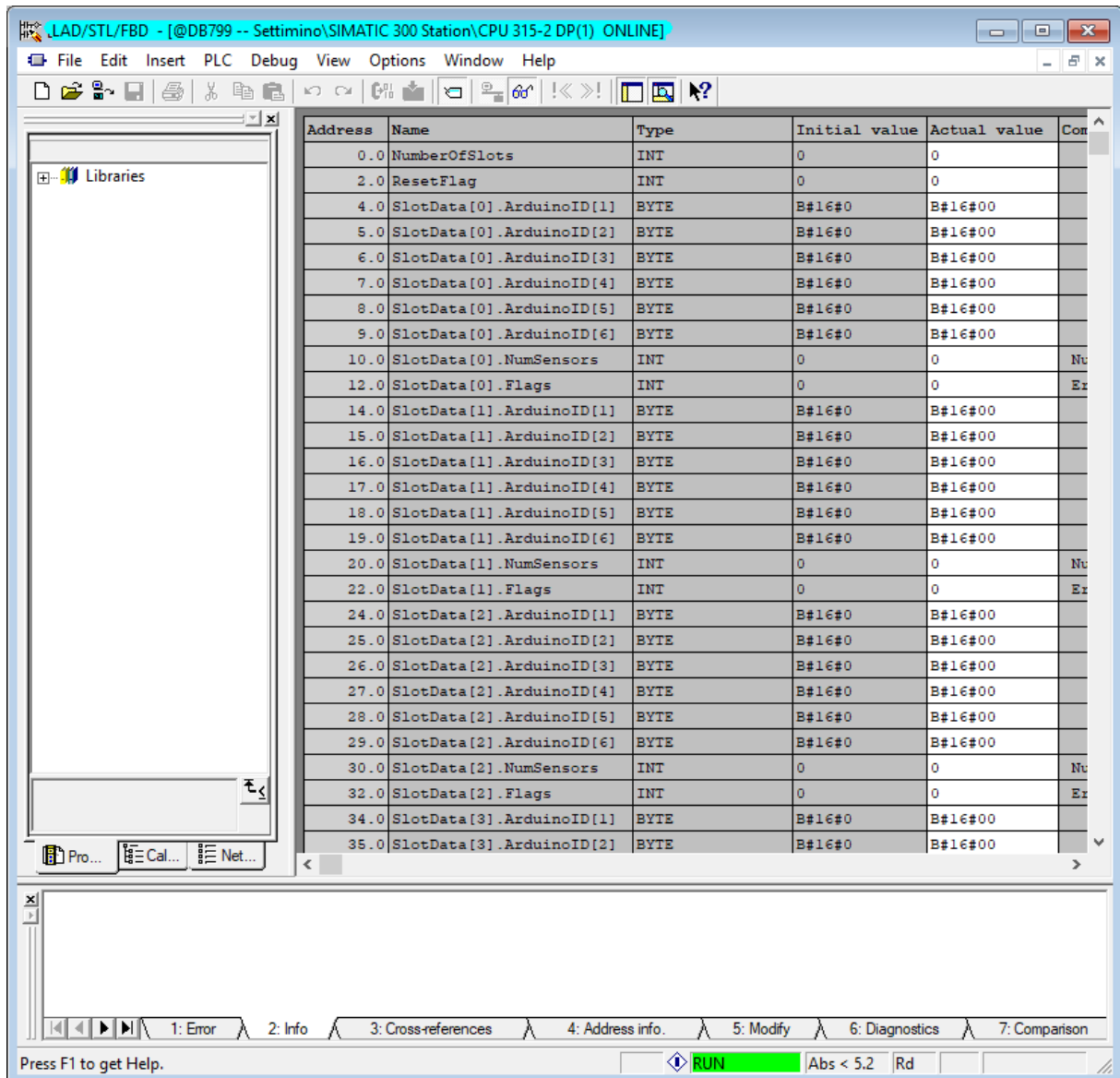


Fig. .3: PLC definition of error block.

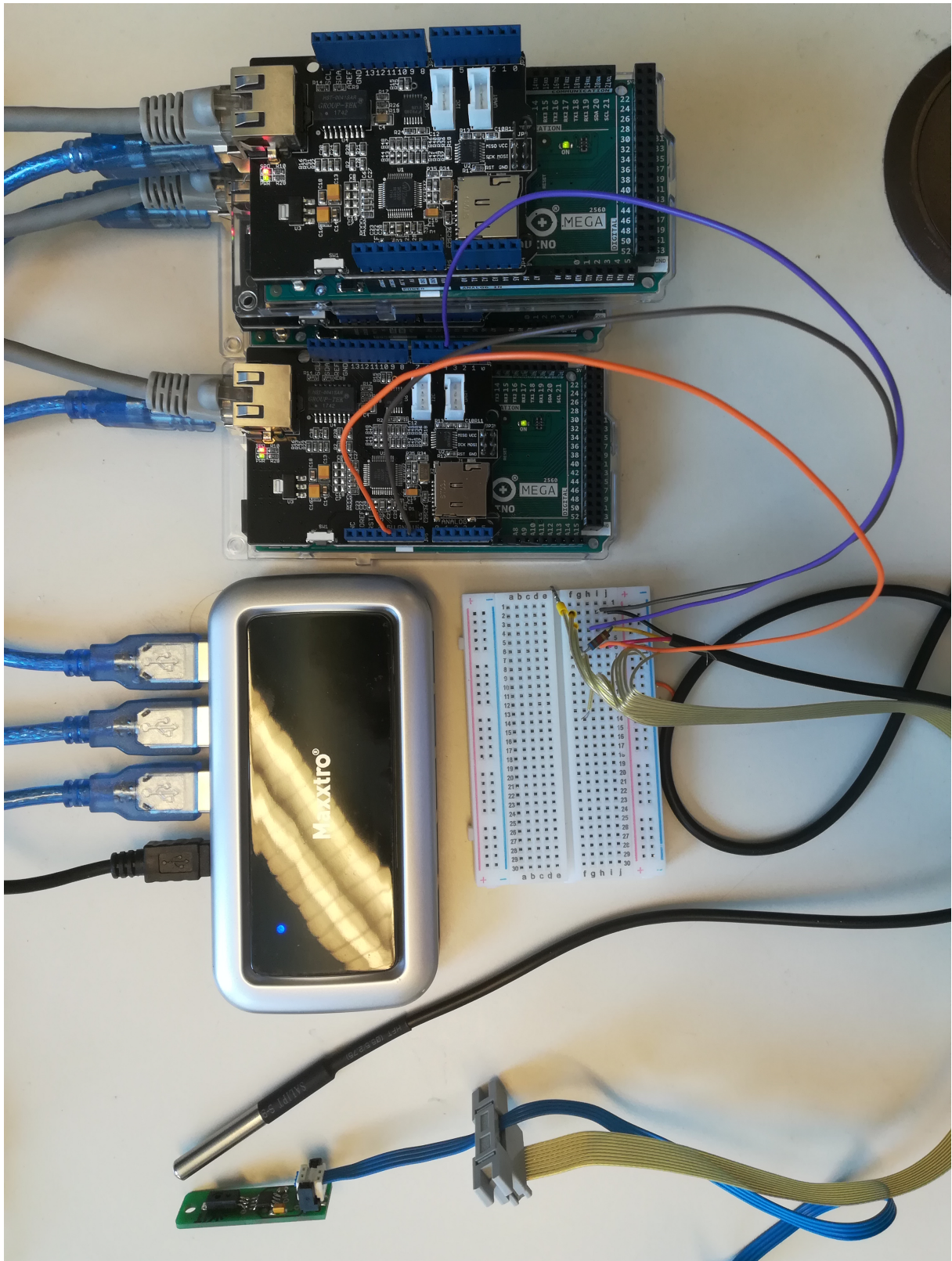


Fig. .4: Photo of the hardware testing setup: sensors, Arduinos and USB switch.

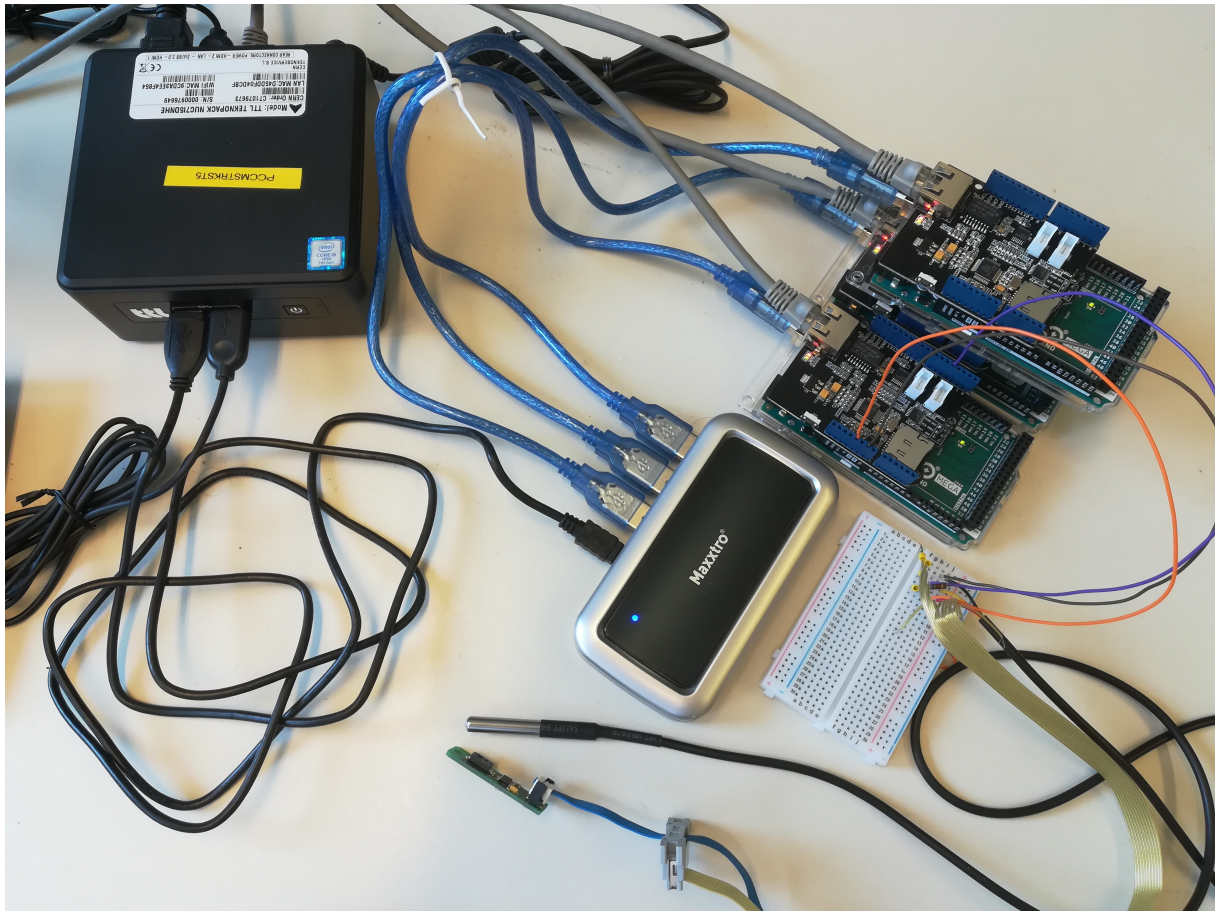


Fig. .5: Details of the USB and Ethernet connection.



Fig. .6: Simatic S7-300 PLC used for testing.