POLITECNICO DI TORINO

Computer Engineering Master Degree
Embedded Systems Course

Master Degree Thesis

Creation of a complete data collection system for condition monitoring in industrial environment with IO-Link sensor

Relatori:
Massimo Poncino, Daniele Jahier Pagliari

Company Tutor:
Claudio Chieppa

Candidate:
Jacopo Cerato

Marzo 2020

Acknowledgments

First of all, I would like to thank my supervisor Massimo Poncino for his support during the elaboration of this thesis. I would also like to thank my team and, in particular, Dante Bonino, Carmelo Migliore, Claudio Chieppa and Alberto Brigandì for giving me the opportunity to collaborate with them. Their support during these months and their contribution have been extremely important. Furthermore, I would like to express my gratitude to my parents for giving me the opportunity to pursue a university career and to have supported me economically during all these years. I would also like to thank my friends for making these university years a little lighter. Finally, I would like to thank my beloved girlfriend Giulia for supporting and enduring me in moments of university stress.

Summary

More and more in recent years, there is the need to be able to monitor the environment and the objects that surround us. Just think of the apps that have recently been developed to be able to monitor your home's rooms, or the possibility to switch on and off your heating system from remote in order to control the environment's temperature, or even more the setting of your mobile phone that notifies you when a certain app is consuming a lot, to give you the possibility to switch the app off and reduce power consumption. It is clear that we want to control all the objects that interact with us.
Factories are not immune to this spontaneous process. In fact, the definition of Industry 4.0 has been recently introduced and industrial environments have become increasingly smart. The ability to monitor industrial machinery, in order to, for example, check the production processes' correctness or to comply with safety standards, is becoming a desired in factory environments.
For these reasons, we have chosen to create a complete data collection system which is able to recover data and measurements from the device, store them in a database, and to send them to an MQTT broker, in order to make them accessible also in the cloud.
This thesis work aims to provide a user, even if he is inexperienced, with the possibility of monitoring a machine or an environment, simply by clicking a button on a dashboard. More in details, the developed system will do all the work automatically; this means that, once the button on the dashboard has been pressed, the system will recover the data, requested by the user, from the sensor, store it in the framework database and forward it via MQTT to an online broker.

In recent times, a new IO protocol is spreading in the market, the so called IO-Link protocol. This is a new standard connection technology that has been developed for communication with sensors and actuators, in order to overcome the shortcomings of previous technologies. This is a point to point communication and it requires just one cable in order to establish the connection; the cable is able to provide, using the three wires that it has inside, at the same time the power supply and the digital input/output line. This is the first advantage given by this technology. Even more, the communication is a digital one, so costly shielded cables

are no more required, since digital transmission is not affected by electromagnetic disturbances. Besides, IO-Link drastically reduces the interfaces needed, because it provides a universal physical interface for wiring along with a uniform interface both for configuration and parameter assignment.[10] This is a significant advantage, since problems related to different connection interfaces for digital, analog, and serial data are no more present. For these reasons, IO-Link is really appreciated in industrial environment and this, obviously, affects our device choice.
A typical IO-link system involves the following components:

- IO-Link master;
- IO-Link device;
- Standard cable to connect the device to the master.

The IO-Link master works as a link between the controller to which is connected and the devices that are attached to him. Typically, it has several IO-Link ports, so it can manage more than one device at a time. The IO-Link master is the one that manages the communication between the controller and the device; it is the interface through which the controller has the possibility to set configuration or to send commands to the sensor/actuator and retrieve data from it.
For this project, we choose an evaluation kit, produced by STMicroelecrtonics, that provides us both the IO-Link master and the IO-Link device.

On the controller side, which in our case is a Linux PC, we have choosen to use EdgeX Foundry framework. It is a collection of open source micro-services that communicate through a common API and it is possible to replace or augment them as needed. EdgeX is a useful tool that makes easier to connect with and to control devices, in order to retrieve data from them, to send instruction to them or to move data from the device to the cloud. Microservices of the framework are divided in layers, starting from the Device Services layer, that interacts directly with devices, and arriving at the Export Service layer, that interacts directly with the cloud. Between this two layers, there are the Core Services layer, that is used to store data, commands and all IoT objects connected to the platform, and the Supporting Services layer. All these layers could interact with each other, in order to create a network to manage data, configuration and device events.

In order to realize the data collection system, we need a way to make possible the communication between EdgeX platform and the IO-Link master, which, in turn, will then communicate with the IO-Link device through the standard cable. Physical connection between the controller, in which the framework is hosted, and the IO-Link master is achieved through serial bus. The logical connection is achieved, instead, through a C application that works as a driver for the device. This driver is able to send the correct commands to the master board, in order to retrieve the requested data. Once the data requested to the device are sent to the controller, the driver is able to store them in the database in order to make them accessible for futures uses and to have a sort of history of the system. Then, every time a data comes in the framework, it is also forwarded to the cloud through MQTT protocol, exploiting features provided by the Export Services layer. Finally, to make the use of the system more "user friendly", a dashboard has been designed. This dashboard interacts with the system through API REST. It provides the user several buttons and every time a button is pressed an API REST starts; when data are available, they are displayed on the screen, in such a way that the user could check them without accessing the database in which they are stored or without connecting to the MQTT broker.

In conclusion, we could say that the systems works fine and all our desired are respected. In addition, a study on power consumption has been done. We have verified that the IO-Link device, even when it is stressed with subsequent commands, never requires more than 80 mA. This fact places the device in the "low-power" category; this is an advantage, since companies are always looking for low-consumption objects and, furthermore, it is possible to supply the device with batteries.

Table of contents

Chapter 1

Introduction

1.1 Background

More and more in recent years we hear about Internet of Things and Industry 4.0.

Internet of Things is a new terminology recently introduced referred to the extension of internet to devices and concrete places[2]. It is the expression used, for some years now, to define the network of equipment and devices, other than computers, connected to the Internet: they could be fitness sensors, vehicles, radios, air conditioning systems, but also bulbs, cameras, furnishings, household appliances, containers for the transport of goods. In short, any type of electronic device, equipped with a software that allows it to exchange data with other connected devices, could be connected to the Internet.
The main purpose of the Internet of Things concept is to monitor and control, to exchange information and perform consequent operations. Several examples of our everyday life could express the IoT concept: IoT is, for example, roller shutters that rise in the morning when your alarm rings. IoT is your house that switches on lights as soon as it hears you entering. These IoT examples give you the opportunity to combine the real world and the virtual one, by connecting objects to the network. Internet of things allows to put "things" on the network, exactly like personal computer, tablet and smartphone made with people. It is estimated that in 2020, IoT objects will be 20,4 billions[3].
Internet of things development and its application to the most variety of objects, have been possible thanks to the discipline that deals with the creation of sensors. These devices are designed to collect specific data according to a predetermined purpose: for example, there are devices that detect data relating to room temperature, air quality, environmental noise level or the movement of a machine. These sensors allow, therefore, to detect information and transform it into digital data that could

be shared through the Internet.

Together with the concept of the Internet of Things, the theme of Industry 4.0 has spread. The name Industry 4.0 was conceived as the fourth industrial revolution, the process that is leading to fully automated and interconnected industrial production; industries, in fact, are getting smarter and Internet of Things is deeply involved in this. It is evident, in fact, that in companies there are more and more connected objects. They are used, for example, to monitor the production process in order to make it more efficient, to monitor machinery behavior in such a way that one can promptly intervene in case of breakdowns or malfunctions. With Industry 4.0 concept, interactions within men, machines and systems are strongly enhanced. IoT is a primary technological component in Industry 4.0 projects, to make machines and production lines more intelligent, through integration of sensors, actuators and Edge computing components. Operative data, selected and summarized by Edge systems, are made available for further elaborations, like for example filling dashboards for industrial monitoring or feed external machine learning systems and AI useful for predictive maintenance or optimize production.
Industrial Internet of Things could be divided in three main areas of use:

- Smart factory
- Smart logistics
- Smart lifecycle

The first one includes production progress control, work safety, maintenance, material handling, quality control, waste management. Second one could be composed by traceability / monitoring of the supply chain via RFId (Radio-Frequency Identification) and sensor tags, cold chain monitoring, safety management in complex logistics centers, fleet management (e.g. via GPS / GPRS). Last one includes, instead, improvement of the new product development process (e.g. through data from previous versions of connected products), end of life management, supplier management in the new product development phase.[4]
So, IIoT can generate revolutionary operational efficiencies and offer completely new business models to any reality that deals with the production and/or transport of physical goods.

Along with previous two topics, one new IO protocol is spreading in the market: it is the so called IO-Link protocol. It is a new standard that is particularly appreciated in industrial environments, since it provides several benefits with respect to previous communication protocols. It is used to connect sensors or actuators to a controller system and it requires just one cable for the connection: the cable in fact is able to provide the device, at the same time, both power supply and data transaction line. Furthermore, the transmission is a digital one, so the cable does not need to be shielded, since digital signals are not affected by electromagnetic disturbances. Since the transmission is a digital one, the quality of measuring sensors is obviously enhanced because just one conversion from analog to digital is required. For these reasons, this innovative technology is a good solution for condition monitoring of both environment and machines. The reader can find an accurate description of IO-Link in chapter two.

1.2 Motivations

Taking into account topics discussed in previous section, we will now analyze the motivations that led to the realization of this thesis work that was carried out in Concept Reply company, described in section 1.7.
Considering IO-Link technology interesting and that there was a market need, the company decided to deepen its competence on this new communication protocol. Main interesting features of IO-Link that moved our choice are the fact that this technology increases signal quality of measuring sensors, drastically reduces the interfaces needed, increases machine availability and provides advanced diagnostic functionalities. The reader can find a detailed section on these previous topic at the end of chapter two.
Since the company is very attentive to industries needs and considering the fact that the so-called condition monitoring is an increasingly requested feature, the idea of a complete data collection system, based on IO-Link device, is conceived.
The main goal of this thesis work is to create a complete data collection system that is able to collect data from an IO-Link device, to store them for analysis or future consultations and to spread them on the network.
Main challenge of this project is trying to integrate the IO-Link device, and all what concerns it, starting from command up to data and measurements, with an open source IoT framework, named EdgeX Foundry. Integration between these two elements should result in a complete system able to collect, store and forward data autonomously any time the user makes a request to the sensorized board.

1.3 Approach and results

Taking into account what stated in previous sections, the device used to perform condition monitoring should be an IO-Link one. In order to work correctly, an IO-Link device requires a IO-Link master board.
For these reasons we chose a development kit produced by STMicroelectronics. This

kit includes a master board, described in chapter four, a sensorized device, described in chapter three, and an adapter board for ST-Link, used to flash firmware on the IO-Link device.
The device communicates with the master board through an M12 standard cable, that is able to provide both power supply and data line. You can have evidence of this by looking the picture below:

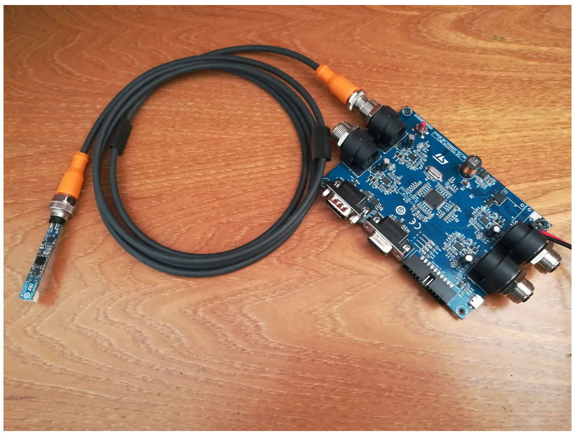


Figure 1.1. IO-Link connection between master and device.

The master board, in turn, communicates with the PC, that works as the IO-Link system controller, with an RS485-USB cable as you can see in the picture below. A bench power supply is used to supply the master board.

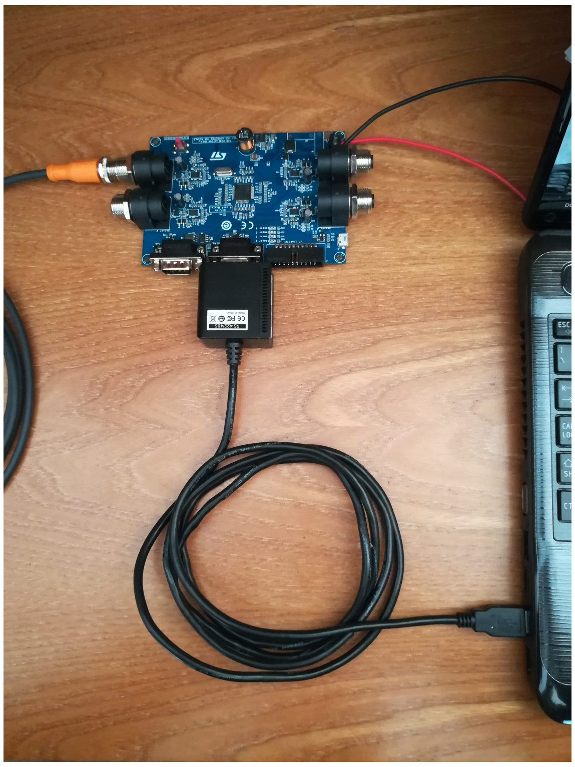


Figure 1.2. Connection between master board and PC.

So, the overall connection between the device and the PC should be like the one showed in the following picture. It is evident that, just two cables are required in order to establish the communication between the IO-Link device and the controller of the system, which in our case is a common Linux computer. This is obviously, a big advantage, since costs and complexity are strongly reduced with respect to non IO-Link solutions.

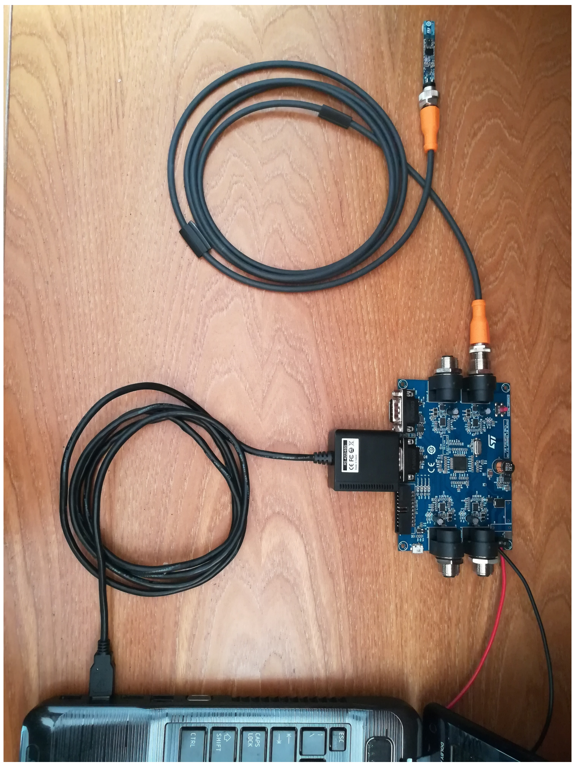


Figure 1.3. Overall connection between device and PC.

Another key feature of this data collection system is the use of EdgeX Foundry platform. It is an open source framework, based on microservices, and we chose to integrate it in our work since it is suitable for Industrial Internet of Things. You can find a detailed description of this open source platform in chapter five.
So the main idea of this thesis work is trying to integrate the IO-Link system with the EdgeX framework, in order to create an automatic system able to collect data from the device, to store them permanently and to forward them to the cloud.
Therefore, main results achieved are:

- Communication between framework, master board and device achieved.
- Data recovery from the IO-Link device achieved.
- Persistent data storage achieved.
- Data forwarding achieved.

Results are anyway better discussed in chapter seven and eight.

1.4 A typical usage scenario

Data collection system scenario is reported as follows:

1. The user wants to monitor industrial machinery or, more in general, the environment in which the machine works.

2. The user requests the device for a certain data. It could be: pressure, humidity, temperature, acceleration, vibration.

3. The system programs the master board in order to communicate with the device, then sends to the device the command corresponding to the requested resource.

4. The device responds with the target data.

5. The system stores the data in the database for future use.

6. The system forwards the data, through MQTT protocol, to the cloud.

1.5 Structure of this thesis

This section briefly describes the thesis plan in order to give the reader an overview on what is going to read.

First part of this document is used to describe all the components involved in this thesis work; chapters from two to five make up this part. In particular, in chapter two the reader can find the description of the IO-Link protocol, which is the protocol used by the sensorized device to communicate with the master board.
Chapter three provides you a description of the STEVAL-IDP005V1, a board, produced by STMicroelectronics, on which are embedded several sensors, that could be used to monitor industrial environment or machinery.
In chapter four, the reader can find the description of the STEVAL-IDP004V1; it is the master board used to interface the pc to the sensorized device. Hardware architecture and firmware descriptions are provided.
Last chapter of this first part is the one related to the framework used to collect data from the STEVAL-IDP005V1, to store them and to forward them to the cloud. This framework is an open source one, called EdgeX Foundry, and the reader can find information on it in chapter number five.

Second section of this paper, composed by chapters six, seven and eight, is related to the implementation of the data collection system. More in details, chapter

six describes the data collection system in its entirely, starting from the low level to the higher level. Particular attention is given to the device driver used in order to make the device interact with the framework.
Chapter seven provides the user visibility on testing and validation of the system. All tests performed and device costs are discussed.
Last chapter, number eight, analyzes results achieved with this thesis work, provides a study made in order to verify the possibility to supply the system with batteries and, finally, treats future steps and related works.

1.6 The company

This degree thesis was written to conclude the Master's degree course in Computer Engineering at Polytechnic of Turin, at the end of six months of work spent at Reply, in particular in Concept Reply business unit.
Concept Reply is the hardware and software development partner of the Reply group specialized in IoT innovation. It offers solutions in the Smart Infrastructure, Industrial IoT and Connected Vehicle sectors; in particular, the business unit where I work is dedicated to Industrial IoT and Edge Computing. The company covers several aspects: from hardware design and development to software implementation in embedded environments, to edge computing software or cloud applications.

Chapter 2

IO-Link: overview, behavior, features

2.1 IO-Link overview

During recent years, IO-Link is becoming really popular in industry environment.
It is a connection technology that has been developed for communication with sensors and actuators, in order to overcome the shortcomings of previous technologies.
IO-Link is not a fieldbus based technology, because it is a point-to-point communication and it requires just one cable, with three wires inside, in order to establish the communication.
The communication is a digital one, so the cable do not require to be shielded, since digital transmission is not affected by electromagnetic disturbances.
A typical IO-Link system involves the following components[5]:

- IO-Link master;
- IO-Link device;
- Standard cable to connect the device to the master.

The IO-Link master works as a link between the controller to which is connected and the devices that are attached to him. Typically, it has several IO-Link ports, so it can manage more than one device at a time. Obviously, just one device at a time could be connected to a port, but there is no restriction on which port to use to connect a device.
The IO-Link master is the one that manages the communication between the controller and the device; it is the interface through which the controller has the possibility to set configuration or to send commands to the sensor/actuator and retrieve data from it.

2.2 IO-Link interface

As previously said, IO-Link is a point-to-point connection that provides both energy supply and signal transmission.
In order to satisfy an higher grade of protection, to make possible installation in industrial environment, an M12 plug connector has been chosen. The master typically has a 5 pin M12 socket, and the pin assignment is the following[5]:

- Pin 1: 24V;
- Pin 3: 0V;
- Pin 4: Switching and communication line (C/Q).

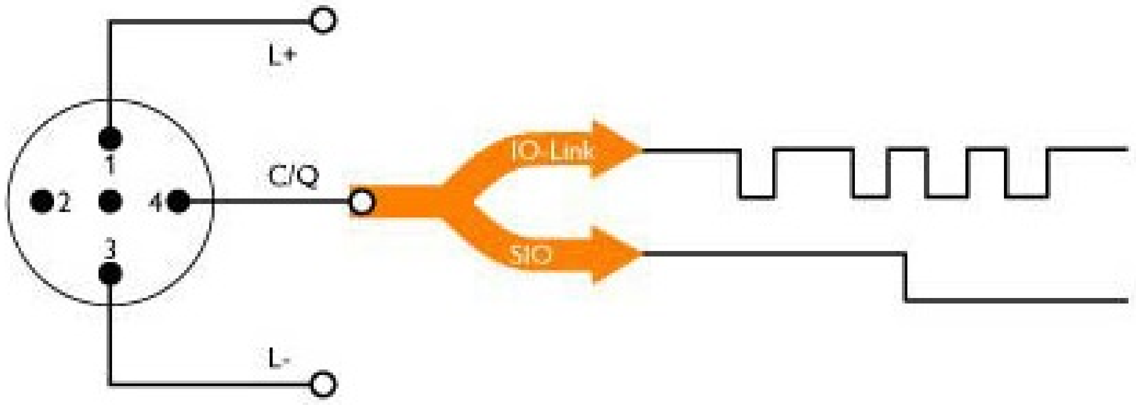


Figure 2.1. Pin assignment of IO-Link device. Taken from [6]

Pin 2 and pin 5 could have different functions depending on the type of the port. In fact, two port classes are defined for the IO-Link master, port class A and port class B:

Port Class A (Type A)

This is the first port type; in this case it is up to the manufacturers to define functions for pin 2 and pin 5, since these functions are not specified by the standard. Normally, pin 5 is left unconnected, while pin 2 provides an additional digital channel.[5]

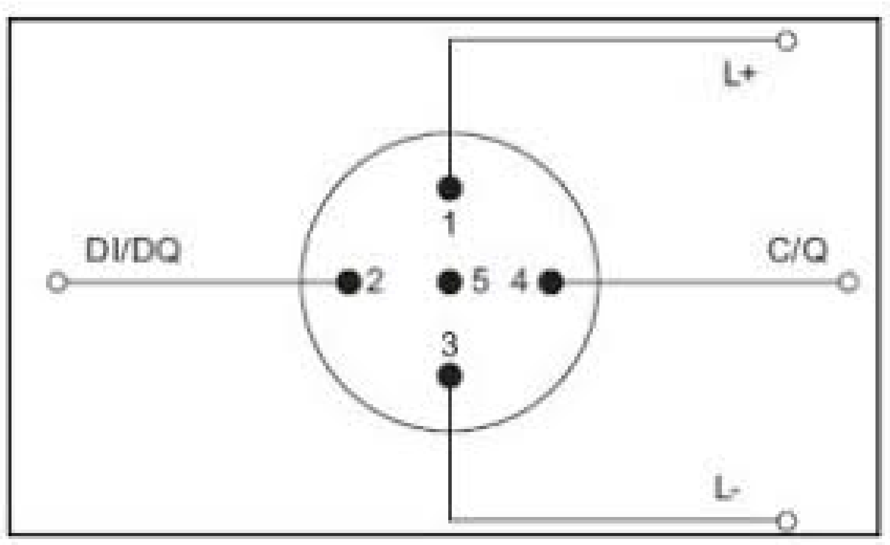


Figure 2.2. Pin assignment port class A. Taken from[5]

Port Class B (Type B)

In some cases, users need to use devices which require more power than normal. For this reason, port class B has been defined; in fact, type B port exploits pin 2 and pin 5 to give additional supply voltage to the device.
Obviously, in this case a 5-wire cable is needed, instead of the typical 3-wire cable, in order to provide this additional supply voltage.[5]

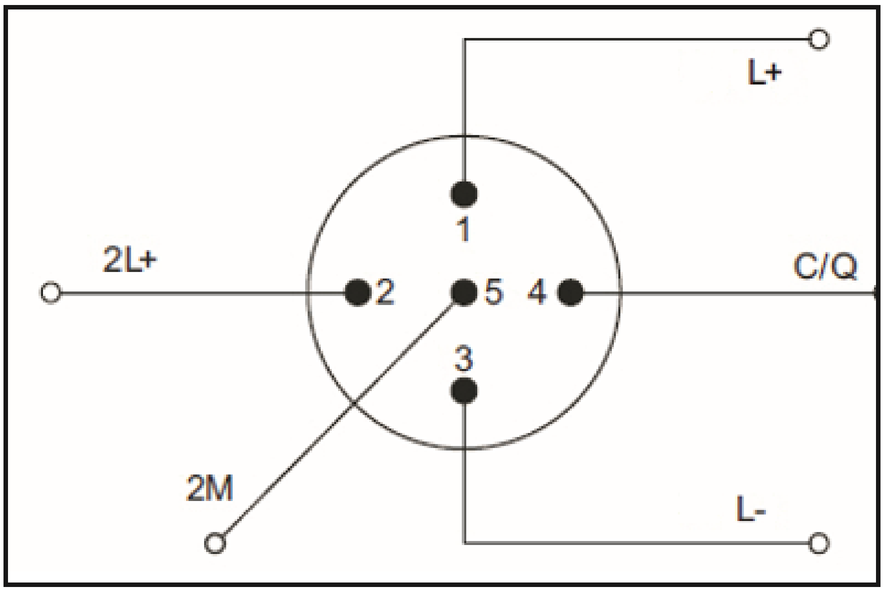


Figure 2.3. Pin assignment port class B. Taken from[5]

Connecting cable

Connecting cable could be of two type, depending on the master's port class. For type A port, 3-wire cable is sufficient, while a 5-wire cable is required for type B port.
The cable is at most 20 meter long, shielding is not needed and also the material cable doesn't matter.[5]

2.3 IO-Link protocol

To analyze the IO-Link protocol, we have to focus on three main topics: communication ports, data types and transmission speed.

The IO-Link ports of the master can be configured in four different ways[5]:

- **IO-Link mode**:
  This mode configures the port for a bi-directional communication, so the port is used for IO-Link communication.

- **DI mode**:
  DI mode configures the port as a digital input.

- **DQ mode**:
  DQ mode configures the port as a digital output.

- **Deactivated**:
  Deactivated mode just simply deactivates the port and could be applied to unused ports.

In IO-Link communication, there are four possible data types: process data, value status data, device data and event data.
More in details, we could analyze each of them:

- **Process data**:
  Process data are information read by the device and transmitted to the master, like, for example, distance readings of laser measuring sensors.
  Process data of devices are cyclic data that are transmitted in a data frame; the size of the process data is specified by the device.
  Depending on the device, 0 to 32 bytes of process data are possible (for each input and output).[5]

- **Value status data**:
  Value status data are the ones transmitted along with process data, in order to tell the master if the latter are valid or not.[5]

- **Device data**:
  Device data are the ones used to send parameter or configuration to the device, or to retrieve from it diagnostic information.
  It is the master that requests the transmission of device data, and obviously them could be written to or read from the device.[5]

- **Events data**:
  Every times an event occurs, for example a short-circuit or overheating, events data are transmitted from the device to the master; the master then forward data to the controller, in order to make the controller handle the event.
  Transmission of events data does not affect transmission of other data.[5]

Now that we have analyzed data types, it is time to focus on transmission speed.
For IO-Link mode, three transmission rates are specified:

- COM1 = 4.8 kilobaud

- COM2 = 38.4 kilobaud

- COM3 = 230.4 kilobaud

For each IO-Link device only one of the previous three transmission rates are defined; the master is, instead, capable to communicate with devices using each transmission rates and it adapts itself according to the transmission rate requested by the device.[5]

2.4 IO-Link advantages

IO-Link is becoming really popular in Industry 4.0 and the reason of that is that it provides a wide number of benefits with respect to previous standards.

In the following we will analyze some of those advantages.

- **IO-Link increases machines' availability**

  One of the main troubles in industry environment is represented by the possibility to replace diagnostics devices. The problem is that when a replacement is needed, it takes time, since the new device has to be correctly set with rights parameters and configurations.
  With IO-Link this problem is no more present, since when a new device is installed, all parameters and configuration of the replaced device are automatically transferred to the new installed sensor or actuator. This action is performed either by the IO-Link master or by the system controller.
  In this way, human errors are minimized and device replacement do not require specialized users anymore.
  Another advantage is that the time to replace a device or to restart the system is significantly reduced.[7]

- **IO-Link provides advanced diagnostic functionalities**

  IO-Link provides users with visibility into errors and status of each device. This means that users can now observe not only what the sensor is detecting, but also performances: valuable information for machine efficiency. Furthermore, advanced diagnostic features allow users to identify a malfunctioning sensor and locate problems without stopping the line of the machine.
  Combination of real-time and historical data, made possible by a system with IO-Link, not only reduces diagnostic operations' complexity, but also optimizes the maintenance program of the machines, allowing the user to save costs and to maximize efficiency over the long term.[8]

- **IO-Link increases signal quality of measuring sensors**

  We have previously seen that IO-Link technology provides digital transmission. This is a big advantage, since digital communications are not affected by electromagnetic disturbances and, therefore, this means lower costs whit respect to analog technologies, because costly shielded cables could be avoided. Furthermore, IO-Link requires just one conversion; in fact, starting from the device, for the measured value is required only one conversion from analog to digital. The digital signal then reaches the IO-Link master that, in turn, forwards it to the controller. In this way problem related to conversion, like, for example, conversion losses, are no more present and the signal could keep an high quality.[9]

- **IO-Link drastically reduces the interfaces needed**

  Up to now, one of the problems of modern automation systems is given by the great variety of physical interfaces present. The reason of this is that sensors and actuators, from many manufacturers, are connected to the controller via different physical interfaces for digital, analog, serial data.
  Obviously, for the reason just mentioned, a variety of wiring and cables comes in the automation system.
  Moreover, in order to assign parameters to these sensors and actuators and for configuration management, a wide variety of interfaces and tool are required. IO-Link, instead, provides a universal physical interface for wiring and a uniform interface for configuration and parameter assignment. This represents a significant advantage for users; in fact, wiring and connection methods are reduced to a single method for both sensors and actuators side and automation system side.[10]

Chapter 3

STEVAL-IDP005V1: overview, features, components

3.1 STEVAL-IDP005V1 overview

The STEVAL-IDP005V1 is an industrial sensor board, designed by STMicroelectronics.

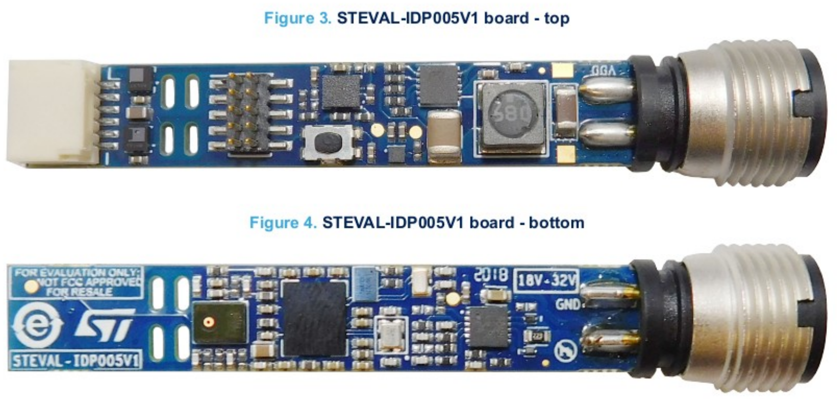


Figure 3. STEVAL-IDP005V1 board - top
Figure 4. STEVAL-IDP005V1 board - bottom
FOR EVALUATION ONLY: NOT FCC APPROVED FOR RESALE
STEVAL - IDP005V1
2018
18V-32V
GND

Figure 3.1. Overview picture of the board. Taken from[1]

The micro-controller of the board is a high end ARM Cortex-M4 32-bit. It is used to run the processing and analysis firmware for the board's sensors.
On the board we can find the following sensors[1]:

- an iNEMO six Degrees of Freedom accelerometer and gyroscope;

- a relative humidity and temperature sensor;
- a digital microphone;
- a barometric pressure sensor.

One of the most interesting features lies in its limited dimensions, as you can see in the picture below.

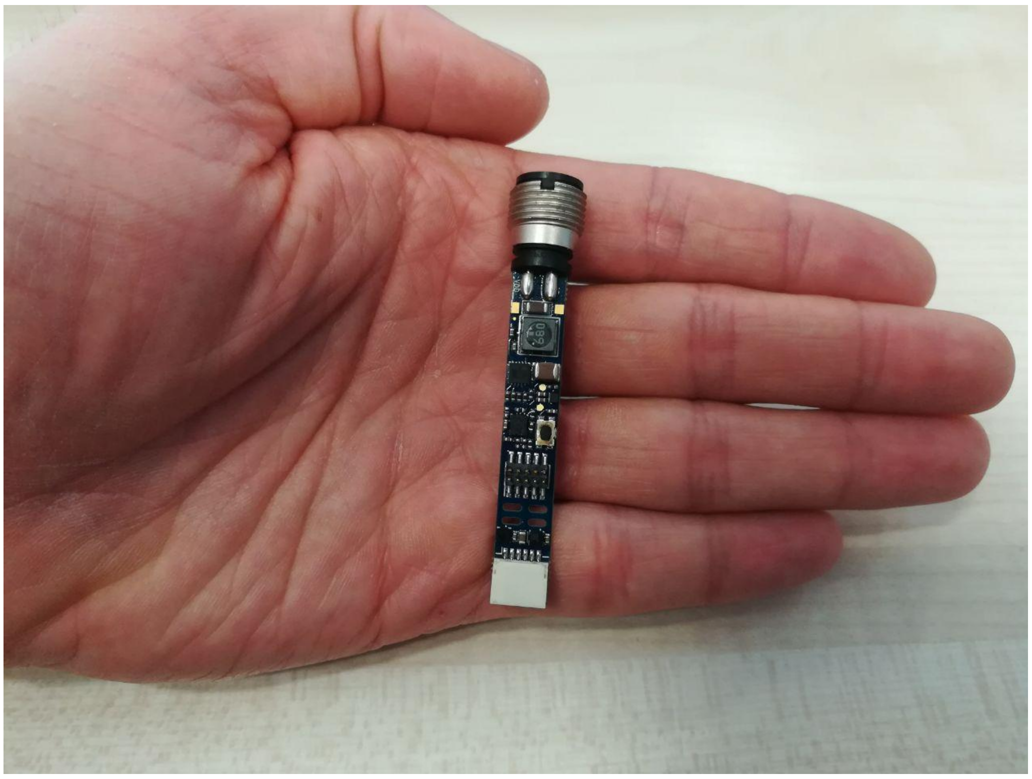


Figure 3.2. Dimensions with respect to my hand

This makes the board available to install in almost all industrial environment.

The sensor platform comes complete with EEPROM for data storage, an IO-Link PHY device and power management based on a step-down switching regulator and LDO regulator.[1]
Sensor data results can be transmitted through one of the following serial communication channels[1]:

- IO-Link: connection with an external STEVAL-IDP004V1 IO-Link master multi-port evaluation board, discussed in next chapter, is required.
- UART: display the data using a common terminal emulator, like TeraTerm, through the UART communication channel.

For the scope of this work, we choose to use the IO-Link channel to transmit data, because IO-Link is becoming widely used in companies and in Industry 4.0 projects and provides several advantages, as described in chapter one.

3.2 STEVAL-IDP005V1 Hardware architecture

In this section you can find the list of the main hardware components along with the top side and bottom side picture of the board. The two pictures are useful in order to have a match between the list of the components and the board in which they are integrated.

Top side components[1]:

- JP1 - IO-Link 4-position M12 A-coded connector
- J1 - SWD connector
- J2 - Auxiliary connector
- SW1 - Reset button
- L1 - Shielded power inductor
- U1 - L6984 step-down switching regulator
- U2 - LDK220 LDO
- U4 - ISM330DLC 3D accelerometer and 3D gyroscope
- U6 - HTS221 humidity and temperature sensor
- U8 - LPS22HB pressure sensor

Bottom side components[1]:

- U3 - L6362A IO-Link communication transceiver

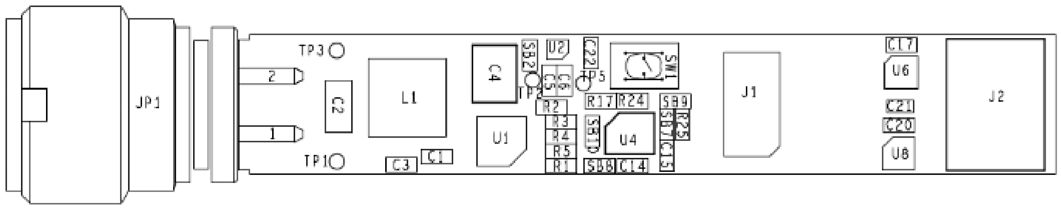


Figure 3.3. STEVAL-IDP005V1 top side components. Taken from[1]

- U7 - MP34DT05-A digital microphone
- U9 - M95M01-DF 1-Mbit serial SPI bus EEPROM
- U10 - STM32F469AI ARM Cortex-M4 32-bit MCU
- Y1 - 32.768 kHz crystal
- Y2 - 24 MHz crystal

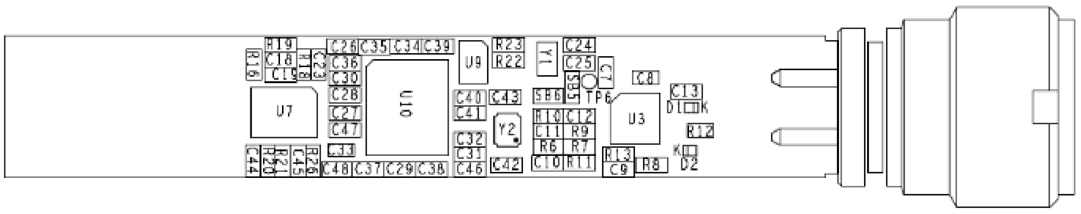


Figure 3.4. STEVAL-IDP005V1 bottom side components. Taken from[1]

Five main functional subsystems could be identified inside the board:

- Microcontroller
- Memory
- MEMS Sensors
- Wired connectivity
- Power management

Next sections analyze each subsystem in details.

3.2.1 Microcontroller[1]

The STEVAL-IDP005V1 embeds an STM32F469AI microcontroller. It is based on the ARM Cortex-M4 32-bit, an high performance RISC core that could operate at a frequency of up to 180 MHz.
The microcontroller features up to twenty-one communication interfaces: multiple I^2C, multiple UART and USART, multiple SPI, multiple CAN, one SAI (serial audio interface) and one SDIO interface are available.
Up to 2 Mbytes of flash memory and up to 384 Kbytes of SRAM are embedded in the device.
The microcontroller could correctly work in the $-40°$C to 105°C temperature range.

3.2.2 Memory[1]

The STEVAL-IDP005V1 embeds a non volatile memory of 1-Mbits. It is an EEPROM, that means electrically erasable programmable memory.
The SPI bus is used to access the memory.
It can work correctly in the $-40°$C to 85°C temperature range.

3.2.3 MEMS Sensors[1]

Several sensors are embedded in the STEVAL-IDP005V1. They are fundamental in order to detect environmental parameters or to detect vibration.

On the board we can find the following sensors:

- ISM330DLC 3D accelerometer and 3D gyroscope
- HTS221 humidity and temperature sensor
- LPS22HB pressure sensor

ISM330DLC

It is a system-in-package designed for Industry 4.0 applications. Inside the package we can find a high performance 3D digital accelerometer and a 3D digital gyroscope.

In order to furtherly increase robustness and stability, manufacturers choose to implement on the same silicon die both sensing elements of the accelerometer and of the gyroscope.

It can operate in an extended temperature range, from $-40°$C up to 85°C. This, in

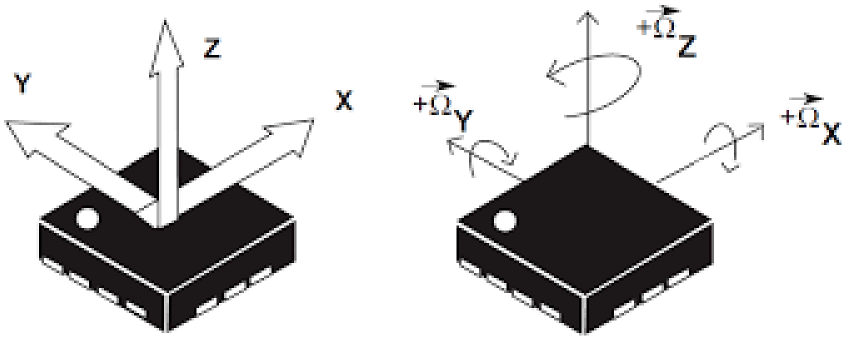


Figure 3.5. Orientation axes for a 3D accelerometer and 3D gyroscope. Taken from[11]

addition to the high shock survivability of the system, makes it optimal for Industry 4.0 applications.

HTS221

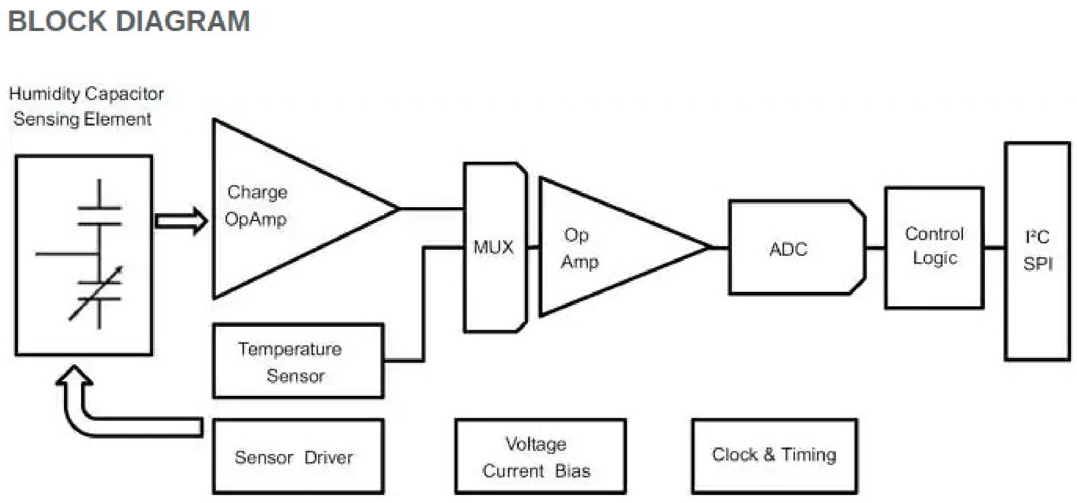


Figure 3.6. Block Diagram of HTS221 sensor. Taken from[12]

The HTS221 is an ultra compact sensor for relative humidity and temperature. In order to provide, through digital serial interfaces, measurement information, the sensor includes a mixed signal ASIC and a sensing element.

The sensing element consists of a polymer dielectric planar capacitor structure capable of detecting relative humidity variations and is manufactured using a dedicated process developed by ST.[1]

The sensor could detect humidity in a range of 20 to 80% relative humidity with an accuracy of ±3.5%.It can measure temperature in a range of 15°C to 40°C with an accuracy of ±0.5°C.

The HTS221 could operate in an extended temperature range, from −40°C up to 120°C.

LPS22HB

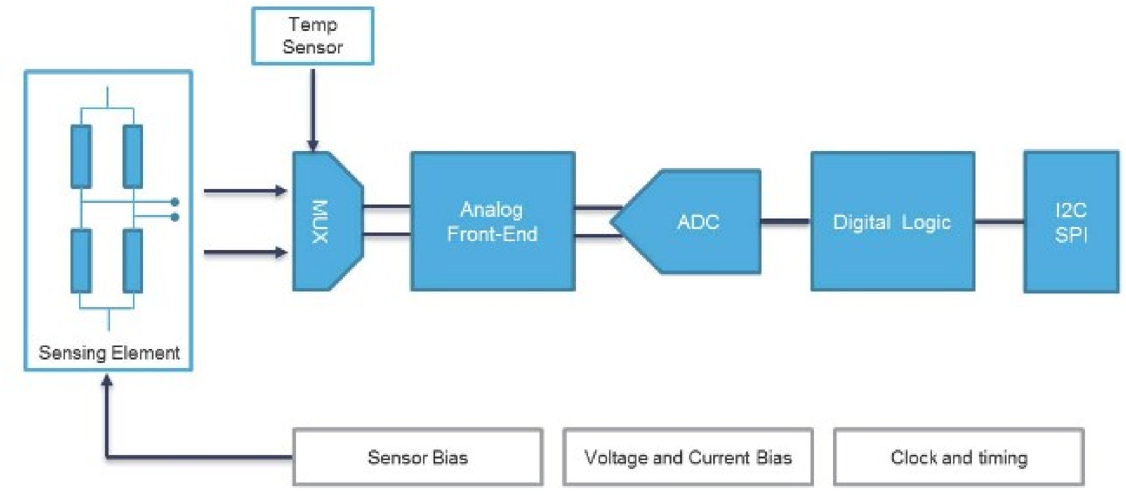


Figure 3.7. Block Diagram of LPS22HB sensor. Taken from[13]

The LPS22HB is an ultra-compact piezoresistive absolute pressure sensor which functions as a digital output barometer.[1] Again, the device is composed by two main parts; the first one is the sensing element, used to detect pressure, and the second is an IC interface, that is used in order to communicate from the sensing element to the application, using I^2C or SPI.

The sensing element is able to dececct absolute pressure; it consists of a suspended membrane, manufactured using a dedicated ST process.

The LGA package of the LPS22HB is holed; the reason of this is that, in this way, external pressure is able to reach the sensing element.

The LPS22HB could operate in an extended temperature range, from −40°C up to 85°C.

It could measure pressure in an absolute pressure range that goes from 260 to 1260 hPa.

3.2.4 Wired connectivity

Several information about IO-Link protocol have been provided to you in the first chapter of this document. Now we analyze how the IO-Link protocol is integrated in the STEVAL-IDP005V1.

The STEVAL-IDP005V1 includes an IO-Link transceiver device, the L6362A, which supports COM1, COM2 and COM3 modes.
The IC can interface a sensor node to a master unit using both the Serial Data Communication Interface (SDCI), based on IO-Link protocol, and the Standard I/O mode (SIO).[1]
The IC is protected against reverse polarity across VCC, GND, OUTH, OUTL and I/O pins, output short-circuit, overvoltage and impulse voltage withstand.

3.2.5 Power management

The STEVAL-IDP005V1 power management stage is composed by two main components:

- L6984 step-down switching regulator;
- LDK220 Low-DropOut regulator.

The first one is a step-down switching regulator. It could operate with input voltage between 4.5 V and 36 V. It is able to provide 3.3 V output voltage without the need for an external resistor divisor and it could supply up to 400 mA DC output current.
The switching frequency could be adjusted from 250 kHz to 600 kHz.
The output voltage could be adjusted starting from 0.9 V.
The L6984 has low consumption mode and low noise mode.
It could work fine in the temperature range that stands form −40°C to 150°C.[14]

The second one is a low drop voltage regulator. It is able to provide up to 200 mA output current and the input voltage could be between 2.5 V and 3.2 V.
Depending on the requirements, the output voltage could be adjusted with step of 100 mV in the range 1.2 V to 12 V.

The LDK220 could operate in an extended temperature range, from -40°C up to 125°C.[15]

Chapter 4

STEVAL-IDP004V1: overview, features, components

4.1 STEVAL-IDP004V1: Overview

In chapter one, we have seen that, in order to connect the IO-Link device with the higher level controller, we need an IO-Link master. The STEVAL-IDP004V1 is the master board, provided by STMicroelectronics, included in the IO-Link evaluation kit for condition monitoring in factory automation systems.

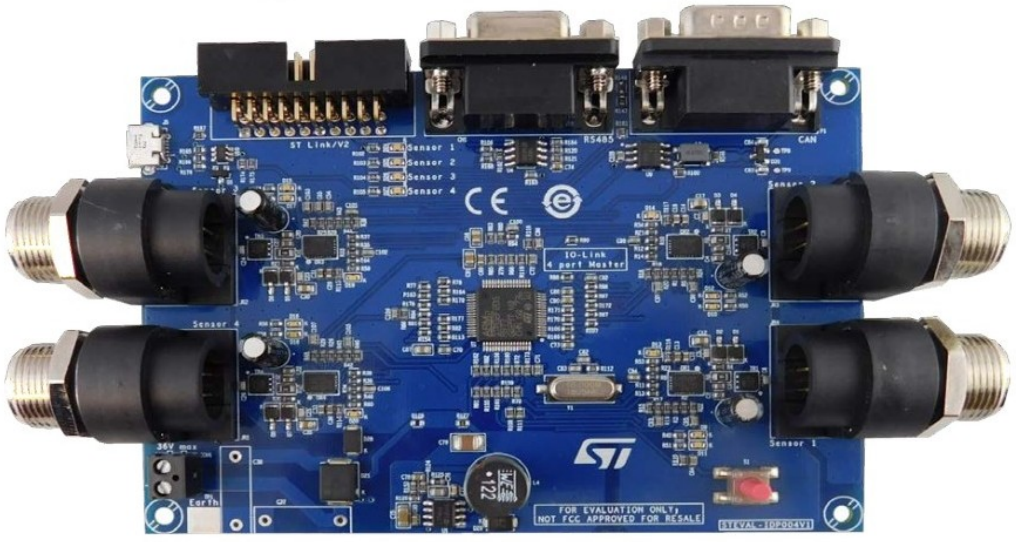


Figure 4.1. Overview of the master evaluation board. Taken from [16]

This master board is the physical link between the sensor device and the controller; through the STEVAL-IDP004V1, the controller could communicate with the STEVAL-IDP005V1 sensor board, analyzed in chapter two, in order to retrieve measurements data or status data from it and to send command to it.

The master board can have up to four device connected, each of them with possible different settings. The master is able to store configuration for each device, so if one of them will have to be substituted, it is not necessary to set again the configuration for the new installed device, because the master sets directly the new device with configurations of the older device.

4.2 STEVAL-IDP004V1: Hardware architecture

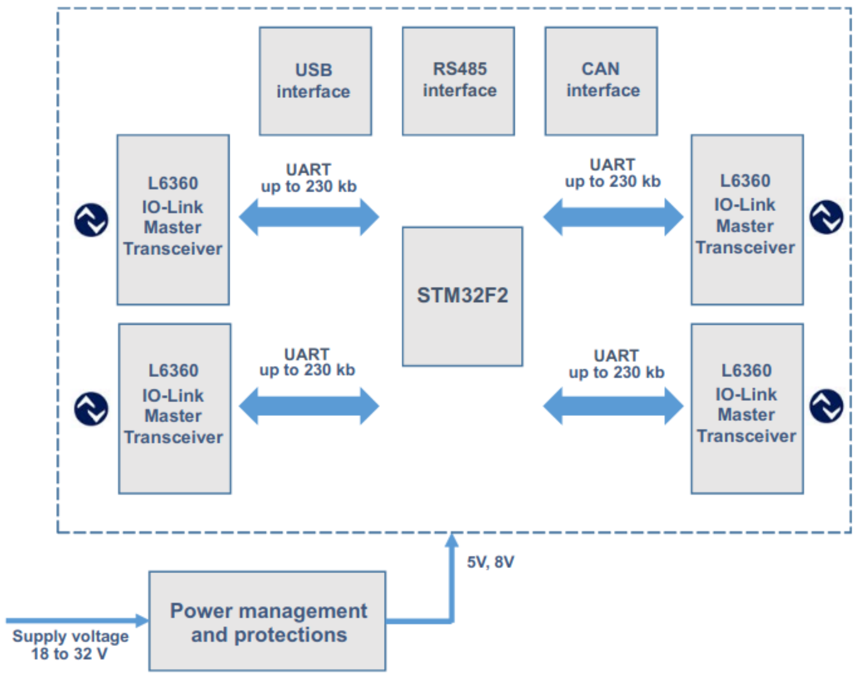

USB interface
RS485 interface
CAN interface
L6360 IO-Link Master Transceiver
UART up to 230 kb
STM32F2
L6360 IO-Link Master Transceiver
UART up to 230 kb
L6360 IO-Link Master Transceiver
UART up to 230 kb
L6360 IO-Link Master Transceiver
UART up to 230 kb
5V, 8V
Supply voltage 18 to 32 V
Power management and protections

Figure 4.2. STEVAL-IDP004V1 block diagram. Taken from [16]

The power management and protection block is used to adapt the input voltage, which is in the range 18 to 32 V, to the 8 V voltage required by the board.
The USB interface and the RS-485 interface are used in order to achieve the communication with the controller, which could be the PC, like in this thesis work, or another control unit. The CAN transceiver, instead, is used to implement the interface with the other communication bus.

The IO-Link interface is implemented using four L6360 master ICs. The L6360 is a programmable transceiver that embeds several registers used to configure different IC settings like output stage configuration, output current limitation threshold protection, de-bounce time in receive mode, LED to signal malfunctions. [16]

4.3 STEVAL-IDP004V1: Software description

The master board comes with a firmware able to manage the communication with the PC via RS-485 interface and to manage the communication with the STEVAL-IDP005V1 device via IO-Link interface.
In the firmware kit we could identify three source files that implement functions mentioned above:

- Master_Settings.c;
- PC_Communication_RS485.c;
- Master_DeviceCOMM.c.

First one is the driver used to properly configure the master board IO-Link transceivers, in order to make them communicate in the correct manner with the device. It provides all the routines needed to evaluate master's status (programmed or not) and to write the correct value in configurable registers, with the possibility to do this either in random way (setting one register at a time), or in sequential mode (providing all register value to the board in just one time).

Second one is like a wrapper since it is the one that manages the communication with the PC. It is in charge of decode user command and, depending on what it has received, it wakes up routines and functions provided by the other two drivers. It provides functions able to transmit and receive data via RS-485, to print messages on the PC screen, to begin a new operation and to close a old one.

Last one is the driver that manages the communication between the master board and the device. It provides functions able to set the master either in transmission mode or in receiver mode; it also makes available routines to send commands or data to the device and to manage the IO-Link transmission.

Chapter 5

EdgeX Foundry: the open software framework for IoT

5.1 EdgeX Foundry: Introduction

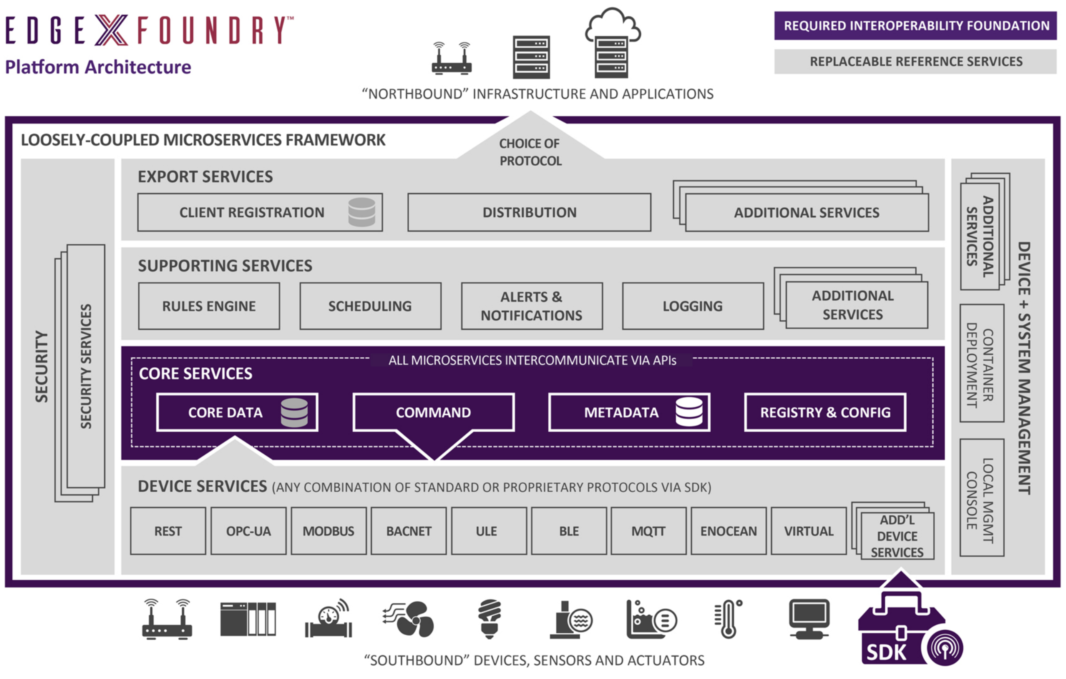


Figure 5.1. Overview of the framework's architecture. Taken from [17]

The EdgeX Foundry is a collection of open source micro-services.
These services communicate through a common API and it is possible to augment or replace them as needed.
In other words, EdgeX Foundry is a vendor-neutral open source platform at the edge of the network, that interacts with several devices, sensors and other IoT objects.
EdgeX is a useful tool that makes easier to connect with and to control devices, in order to retrieve data from them, to send instruction to them or to move data from the device to the cloud.
In this framework we could identify two main sides, the north and the south one. The north side refers to the cloud in which data are stored, analyzed and transformed into information. The south side, instead, is the part of the system that includes the IoT objects and the part of the network that communicates with them.
The framework is divided in several layers, each of them with a specific scope. There are in total six layers, four of them are Service Layers, while the others two are System Services.

5.2 EdgeX Foundry: Service Layers

Four service layers are present in EdgeX Foundry. They are structured in order to create a link between the device and the cloud. Each layer could interact with the others, in order to create a network to manage data, configuration and device events.

5.2.1 Device Services Layer

The Device Services Layer is the first level of the structure, starting from the devices.
Device services are the ones that interacts directly with the devices and that allow to collect data from devices and to send them configuration or input commands. Devices could be actuators or sensors, like in our case of study, or even more complex systems like, for example, lights, hydraulic pumps, irrigation systems.

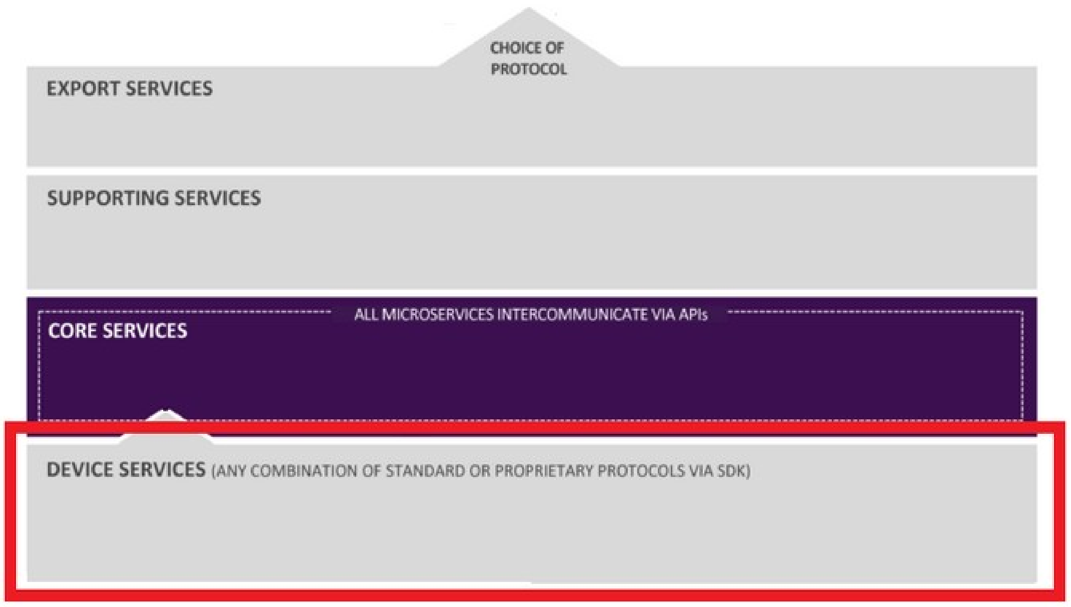

CHOICE OF PROTOCOL
EXPORT SERVICES
SUPPORTING SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)

Figure 5.2. Core services layer. Taken from [17]

Device services could manage more than one device at the same time and one specific device service could serve all devices of the same type, for which the device service was created.
Device services have to be implemented in such a way that they could communicate with physical devices using the native protocol of each device. For example, in our case of study the device uses the serial line to communicate with the controller; in this case the device service had to be implemented in such a way that it was capable to read and write the serial channel.
Once data have been acquired from the device using its native protocol, the Device Services Layer transforms them into a format suitable for EdgeX and, after the conversion, sends them to other micro-services of others framework's layers.
In the same manner, the Device Services Layer must be able to translate commands coming from others EdgeX micro-services and to forward the command to the device.

5.2.2 Core Services Layer

The second layer, starting from the bottom, of the framework architecture is the Core Services Layer.

All the layers over the Core Services Layer belong to the north side, seen in paragraph 4.1, while layer and micro-services under the Core Services Layer belong to the south side, seen in paragraph 4.1. So, Core Services Layer is the link between the north and the south side, in other words the link between the cloud and the devices. Four main elements could be identified in this layer.

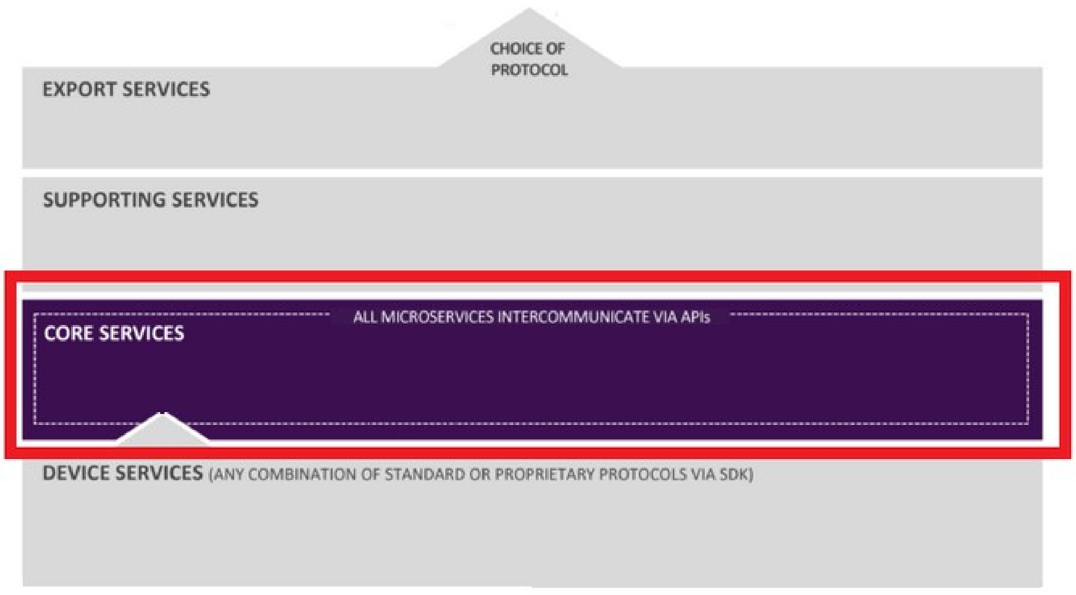

CHOICE OF PROTOCOL
EXPORT SERVICES
SUPPORTING SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)

Figure 5.3. Core services layer. Taken from [17]

The first one is the Core Data. It is a service that is used as a permanent repository for all the data collected from the devices; in practice it is like a database for IoT objects' data.

Second one is the Command service. This is a service useful to facilitate and to control commands coming from upper layers that have to be forwarded to IoT objects.

Third one is the Metadata service. Metadata acts like a repository for devices connected to EdgeX. Inside Metadata are registered all IoT objects connected to the framework, along with the type of resources these devices can provide and the list of their available commands. Metadata makes the user capable to add new devices to EdgeX and to associate them with the correct device service.

Last one is the Registry and Configuration service. This is used as a repository for values to be used during initialization.

5.2.3 Supporting Services Layer

This layer is the one that manages the communication of events and is designed to give edge smartness and analytics skills. In fact, the Supporting Services Layer embeds logging microservices, in order to make the user capable of monitoring each service and, so, how each component is behaving and how components interact with each other.
Alerts and notifications services are also included in this layer; this is a powerful tool, since notifications or alerts could be sent to an external system in case of device or service malfunctions. Alerts and notifications could be sent via email, REST callback to an external system or person.

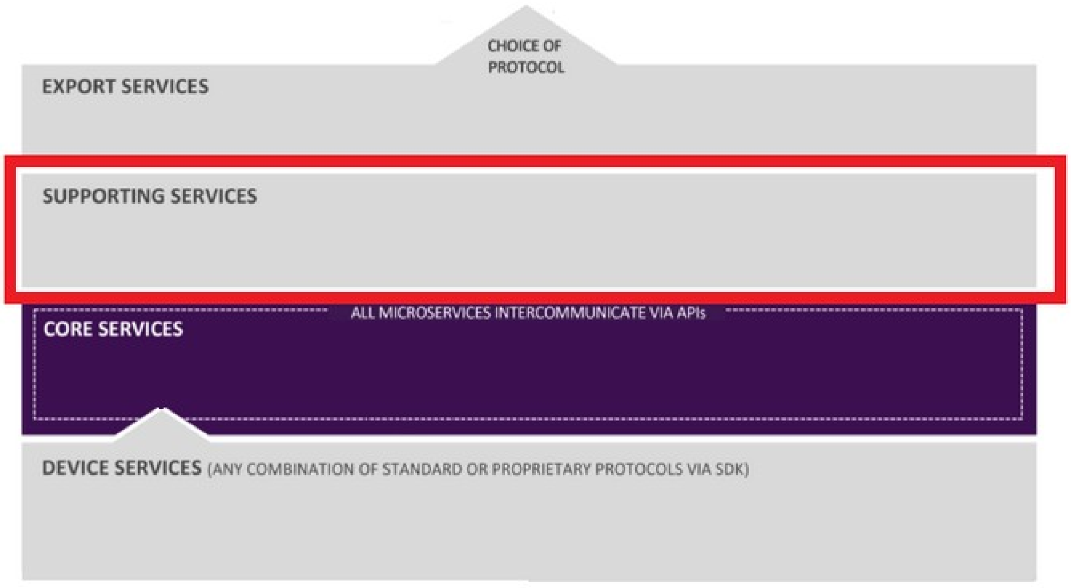

CHOICE OF PROTOCOL
EXPORT SERVICES
SUPPORTING SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)

Figure 5.4. Core services layer. Taken from [17]

Furthermore, in the Supporting Services Layer we could find the Rules Engine and the Scheduling microservices. The first one is used in order to automatically trigger some resources, such as device activation, depending on how the administrator defines the rule.
Scheduling microservices are instead used to delete some data from the core data, after they have been processed and transmitted. This last operation could be automatically performed by setting a timer.

5.2.4 Export Services Layer

This layer is used in order to send data, collected from the south side, to the north side. It is possible to register several clients to receive data from EdgeX; this works in a simple manner: every time a data reaches the Core Data service it is automatically forwarded to each client registered in the Export Service.
During registration phase, it is possible also to define the protocol to be used to export the data. For this thesis work, we have choosen to use MQTT protocol.

5.2.5 System Management and Security Services Layer

System management services are all the services that helps with the installation, initialization, starting and stopping of other microservices, of the BIOS or the operating system.
Security services are related to the protection of sensitive data. These could be device data or commands, secret password or access keys.

Chapter 6

Data collection system: from the device to the cloud

6.1 Overview

In chapter three we have seen that the master board comes with a firmware that allow the communication with the pc. To do that, it is possible to run a common terminal emulator, like Tera Term, PuTTY or Minicom, and to start sending commands to the board. Anyway, this communication takes place only between the board and the controller (the pc in our case of study) and does not involve the framework EdgeX.

In chapter four we have analyzed the structure of EdgeX Foundry and we have seen the layers of the framework and how they work.
Since we have clear the structure of the framework, it is obvious that, if we want to establish a communication between the sensor board and EdgeX, we have to create a sort of driver for the target device.
This driver is included in the Device Services Layer, because this is the platform part that interacts directly with the device.

To build EdgeX device services and to add devices to the framework, two Software Development Kit are available; the difference between these two SDK lies only in the fact that one is for C language, while the other one is for Go language. For our work we choose the C SDK, that means that the device driver had to be written in C language. The choice between the two version was made taking into account the fact that I studied C language during university years.

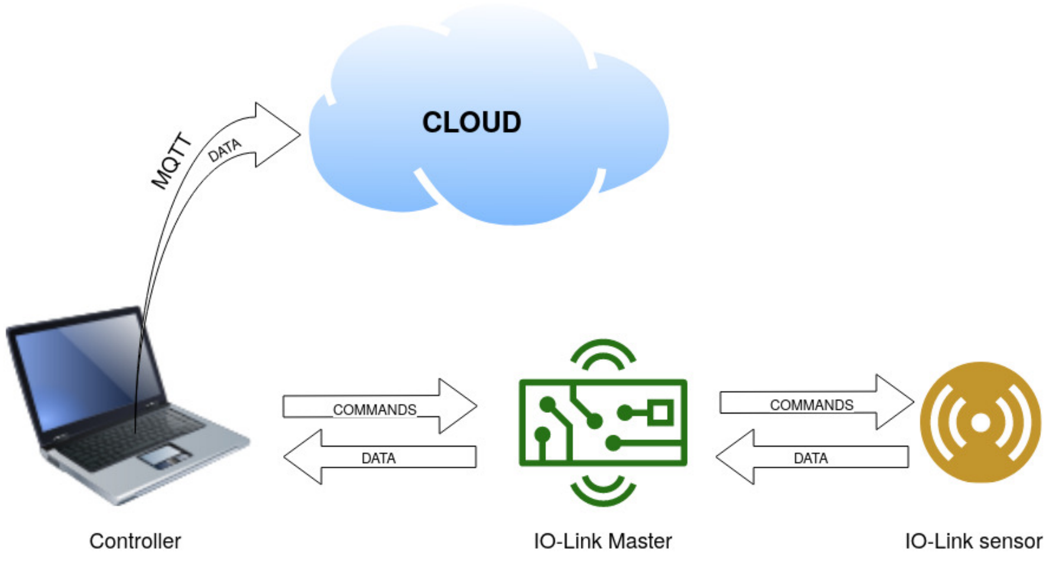


Figure 6.1. Data collection system overview

In order to perfectly integrate a new device with EdgeX Foundry, three main files are required.
One is the C file that specifies what the device service should be able to do and defines the actions required in order to communicate commands or retrieve data from the device.
Another one is the device profile file; it is a yaml file that specifies device's properties and commands that could be sent to the device. Only one device profile is required for all the objects it describes.
Last file needed is a TOML file, which is a configuration file for the device.

Once the device driver is ready, composed by the three components just mentioned, the system is able to take measurements and data from the device and to import them in Edgex platform; in order to make this system complete, we need to export data to the higher level of the network. This is performed exploiting the EdgeX Export service and using MQTT protocol for the communication.

Last component of the system is the dashboard we design in order to give a more user friendly tool to the customer. In fact, even if the user is not an expert one, he could retrieve data and measurements from the device in such a simple way, by simply clicking on a button on the screen.
In the following sections we will analyze all these components.

6.2 Device Profile

The device profile is the file that describes our device properties.
It defines the resources of the sensor, i.e. the physical quantities that we could retrieve from the board, their value type and their unit of measure.
Furthermore, it defines also the exactly command that we could send to the device through API Rest, the exact path on which making the request and the type of responses expected.
It is a YAML file, so it is human readable and it is simple to manipulate and modify.

In the following it is reported the code for the STEVAL-IDP005V1 device profile.

```yaml
name: "ser_dev"
manufacturer: "ST"
model: "STeval"
description: "environmental sensor"
labels:
  - "sensor"

deviceResources:
  -
  name: TEMP
  description: "environmental temperature"
  attributes:
    { parameter: "temp" }
  properties:
    value:
      { type: "Float32", readWrite: "R", floatEncoding:
          "eNotation" }
    units:
      { type: "string", readWrite: "R", defaultValue: "deg" }
  -
  name: ENV_HUM
  description: "environmental humidity"
  attributes:
    { parameter: "hum" }
  properties:
    value:
      { type: "Float32", readWrite: "R", floatEncoding:
          "eNotation" }
    units:
      { type: "string", readWrite: "R", defaultValue: "%" }
  -
  name: ENV_PRESS
  description: "environmental pressure"
  attributes:
    { parameter: "pressure" }
  properties:
    value:
      { type: "Float32", readWrite: "R", floatEncoding:
          "eNotation" }
    units:
      { type: "string", readWrite: "R", defaultValue: "mBar" }
  -
  name: ALL_ENV
  description: "all environment values"
  attributes:
    { parameter: "all_env" }
  properties:
    value:
      { type: "String", readWrite: "R" }
    units:
      { type: "string", readWrite: "R", defaultValue: "environment
          measures" }
  -
  name: FFT
  description: "vibration values in m/s2"
  attributes:
    { parameter: "vibration" }
  properties:
    value:
      { type: "String", readWrite: "R" }
    units:
      { type: "string", readWrite: "R", defaultValue: "text
          vibration" }
  -
  name: TDM
  description: "rms acceleration x y z, peak acceleration x y z"
  attributes:
    { parameter: "acceleration" }
  properties:
    value:
      { type: "String", readWrite: "R" }
    units:
      { type: "string", readWrite: "R", defaultValue: "text" }

coreCommands:
  -
  name: FFT
  get:
    path: "/api/v1/device/{deviceId}/FFT"
    responses:
      -
      code: "200"
      description: "Get sensor measured accelerometer FFT"
      expectedValues: ["FFT"]
      -
      code: "503"
      description: "service unavailable"
      expectedValues: []
  -
  name: TDM
  get:
    path: "/api/v1/device/{deviceId}/TDM"
    responses:
      -
      code: "200"
      description: "Get sensor measured accelerometer RMS data"
      expectedValues: ["TDM"]
      -
      code: "503"
      description: "service unavailable"
      expectedValues: []
  -
  name: ALL_ENV
  get:
    path: "/api/v1/device/{deviceId}/ALL_ENV"
    responses:
      -
      code: "200"
      description: "Get sensor measured environment data"
      expectedValues: ["ALL_ENV"]
      -
      code: "503"
      description: "service unavailable"
      expectedValues: []
  -
  name: TEMP
  get:
    path: "/api/v1/device/{deviceId}/TEMP"
    responses:
      -
      code: "200"
      description: "Get sensor measurred temperature data"
      expectedValues: ["TEMP"]
      -
      code: "503"
      description: "service unavailable"
      expectedValues: []
  -
  name: ENV_HUM
  get:
    path: "/api/v1/device/{deviceId}/ENV_HUM"
```

```
127      responses:
128        -
129        code: "200"
130        description: "Get sensor measured humidity data"
131        expectedValues: ["ENV_HUM"]
132        -
133        code: "503"
134        description: "service unavailable"
135        expectedValues: []
136    -
137    name: ENV_PRESS
138    get:
139      path: "/api/v1/device/{deviceId}/ENV_PRESS"
140      responses:
141        -
142        code: "200"
143        description: "Get sensor measured pressure data"
144        expectedValues: ["ENV_PRESS"]
145        -
146        code: "503"
147        description: "service unavailable"
148        expectedValues: []
```

In the following we will analyze the above code.

- **Line 1 - 7:** This first part is just an introduction and a description of the device.

- **Line 7 - 69:** In these lines we can find the list of all the resources that the device makes available. For each resource, we can find the description, its attributes and its properties. Properties are referred to the resource value and units; in value field it is specified the type of data, like for example Float or String, and if this resource is available in reading or writing mode. In units field, instead, it is defined the unit of measure of the resource.
  In this project, the device makes available five type of resources. First three are referred to environment monitoring, more in details they are temperature, humidity and pressure. These resources are foundamental in cases where the machine to be monitored must work in environment with defined parameters, for example because high temperature or humidity value could infect the machine. Humidity, pressure and temperature could also be measured in the same moment, in order to have a complete control on the environment.
  Last two resources are referred to motion parameters; the first one is able to obtain from the device peak acceleration value and root mean square value,

both in m/s^2 and both for the three axes x, y, z. Second one is used to analyze vibration parameters by processing accelerometer data with a Fast Fourier Transformation algorithm. These last two resources are useful to monitor machine acceleration and vibration, in order to detect a wrong behavior on it and to perform predictive maintenance.

- **Line 70 - 148:** In these lines we can find the list of all the commands that it is possible to send to the device in order to retrieve resources from it. For each available command, we can find the type of the command (PUT, GET, POST), the path at which the command is reachable and the type of responses that the command could send. If the command works fine, the response code is 200 and the expected value is the value of the resource asked by that command; if something went wrong, the response code is 503 and no expected value is present. In order to make the system correctly work, it is important to have one command available for each resource.

6.3 Device Configuration

The device configuration file is a TOML file that is used by EdgeX Foundry to correctly set the environment in order to make the driver work in the desired way.
In the following the reader will find the code of the device configuration; below the code a short explanation is provided.

```
[Service]
  Host = "127.0.0.1"
  Port = 49090
  Timeout = 5000
  ConnectRetries = 3
  Labels = [ "Serial" ]
  StartupMsg = "Serial device started"
  CheckInterval = "10s"

[Clients]
  [Clients.Data]
    Host = "127.0.0.1"
    Port = 48080

  [Clients.Metadata]
    Host = "127.0.0.1"
    Port = 48081

[Device]
  DataTransform = true
  Discovery = false
```

```
22     InitCmd = ""
23     InitCmdArgs = ""
24     MaxCmdOps = 128
25     MaxCmdResultLen = 256
26     RemoveCmd = ""
27     RemoveCmdArgs = ""
28     ProfilesDir = ""
29     SendReadingsOnChanged = true
30
31   [[DeviceList]]
32     Name = "Serial Device"
33     Profile = "ser_dev"
34     Description = "environmental sensor device"
35   [DeviceList.Protocols]
36     [DeviceList.Protocols.Other]
37       Address = "/dev/ttyUSB0"
38       Baudrate = "230400"
39       Parity = "2"
40       NR_Data = "8"
41       Stopbit = "1"
```

- **Line 1 - 8:** This first part is used to define device service parameters, such as the host and the port at which the service is reachable, the timeout and the number of connection retries to perform to start the service.
- **Line 10 - 17:** In these lines there is the list of the clients with which the device has to communicate. For each client, it is specified again the host and the port to connect through. For this thesis work, the IO-Link device service must interact with the Core Data and with the Core Metadata service.
- **Line 19 - 29:** These are default parameters, like the maximum number of command operands or the maximum length of the command result. Most of them must be left empty.
- **Line 31 - 41:** In these lines parameters useful to access the device are provided; in this project, the Io-Link sensor communicates with the pc using serial interface. For this reason, in the ”DeviceList.Protocols”, we have specified the address, which in this case is the name of the serial port, the Baudrate, the parity bit value (2 means no parity), the number of data bits and stop bit.

In this file it is also possible to make the system work autonomously, in the sense that resources are no more requested by the user of the system, but they are retrieved autonomously. To do this some line of configuration have to be added to the

TOML file. Based on the type of resource we want to automatically retrieve it is sufficient to add to the configuration file, in the device list section, the information about auto events. This is such composed:

```
 [[DeviceList.AutoEvents]]
   Resource = "ResourceName"
   OnChange = could be either true or false
   Frequency = "frequency expressed in seconds(i.e., 10s)"
```

Obviously it is possible to add each type of resource in the auto events section.
For this thesis work, temperature value has been tested with auto events execution and the result was good. Anyway we choose to avoid this configuration, since our goal was to demonstrate that the system could react correctly to user inputs.

6.4 Device Driver

In this section we will analyze the device driver file. This is the main part of this project work, since it is the part related to the connection and the communication with the device. Using function defined in this C file, in fact, it is possible to open the serial port, to set the serial port in the correct way, to set the master board with right configurations, to send commands to the device and to retrieve data and measurements from it.
Three main function could be analyzed in this file: first one is the one in charge of finding the correct communication protocol, second one is the one in charge of setting the serial port with correct parameters and the last one is the function that sends commands to the device and retrieves data from it.

In the section below, the reader could find the code of the function able to detect the connection protocol:

```
static const edgex_protocols *findprotocol
(const edgex_protocols *prots, const char *name)
{

  const edgex_protocols *result = prots;
  int i=0;
  while (result != NULL)
  {
    if (strcmp (result->name, name) == 0)
    {
      return result;
    }
    i++;
    result = result->next;
  }

  return NULL;
}
```

This first function, named findprotocol, is the one used by the driver in order to find the correct protocol in the device configuration. It works in such a simple way: it takes the list of device protocols and the name of the protocol to be found and it starts looping on the list comparing the name of the current list object with the name of the protocol needed. In case the two names match, the loop breaks and the function returns the correct protocol, otherwise the loop will continue with the following element of the list. If no protocol is found, the function returns a NULL pointer.

Now we focus on the function able to correctly set the serial port:

```
static const int setSerialPort
(iot_logger_t *lc, const edgex_protocols *protocols)
{

  int addr;
  memset(&conf, 0, sizeof(conf));

  const edgex_protocols *p = findprotocol (protocols, "Other");
  if (p == NULL) {
    iot_log_error (lc, "No Serial protocol in device address");
    return -1;
  }

  const char *name = findinpairs (p->properties, "Address");

  if (name == NULL || strlen (name) == 0) {
    iot_log_error (lc, "No Address property in Serial protocol");
    return -1;

  } else{

    addr = open(name, O_RDWR);

  }

  const char *Baud = findinpairs (p->properties, "Baudrate");

  if (Baud == NULL || strlen (Baud) == 0) {

    iot_log_error (lc, "No Baudrate property in Serial protocol");
    return -1;
  }

  if(strcmp(Baud, "230400") == 0){
    if(cfsetispeed(&conf, B230400) < 0 )
      perror("not able to set input speed\n");
    else
      cfsetospeed(&conf, B230400);
  }

  if(tcsetattr(addr, TCSANOW, &conf)<0)
    perror("tcsetattr failed\n");

  tcflush(addr, TCIFLUSH);
  sleep(1);

  return addr;
}
```

This is the function used to open and set the serial port. First of all, it searches for the serial protocol using the "findprotocol" function previously described: if the "findprotocol" returns a NULL pointer, the function prints an error and exit with code -1. Otherwise, it starts searching for the address name inside the protocol by using a function, "findinpairs", that works similar to the "findprotocol". Once the address name is available, the function either exit with code -1, in case the name is NULL, or open the serial port. After the serial port has been opened, the function looks for the Baudrate property inside the protocol and then set it for the serial port just opened. Finally, the function returns the integer value, that corresponds to the serial port, used to access it in reading and writing mode.

Once findprotocol and setSerialPort functions have been analyzed it is time to focus on the main function that handles the communication between the controller and the device. This is the function able to set the master board and to get data from the device:

```
 static bool serial_get_handler
(
void *impl,
const char *devname,
const edgex_protocols *protocols,
uint32_t nreadings,
const edgex_device_commandrequest *requests,
edgex_device_commandresult *readings
)
{
  serial_driver *driver = (serial_driver *) impl;

  if(id_port == -1){
    id_port = setSerialPort(driver->lc, protocols);
    if(id_port == -1){
      return false;
    }
    program_master_node();
  }

  if(id_port != -1){

    for (uint32_t g = 0; g < nreadings; g++){

      const char *param = findinpairs (requests[g].attributes,
          "parameter");
      if (param == NULL) {
        iot_log_error (driver->lc, "No parameter attribute in GET
            request");
        return false;
      }

    if(strcmp(param, "temp") == 0) {
      ask_for_temperature(readings, g);
      return true;
    }
    else{
      if(strcmp(param, "hum") == 0){
        ask_for_humidity(readings, g);
        return true;
      }
      else{
        if(strcmp(param, "pressure") == 0){
          ask_for_pressure(readings, g);
          return true;
        }
        else{
          if(strcmp(param, "acceleration") == 0){
            ask_for_acceleration(readings, g);
            return true;
          }
          else{
            if(strcmp(param, "vibration") == 0){
              ask_for_vibration(readings, g);
              return true;
            }
55             else{
56               if(strcmp(param, "all_env") == 0){
57                 ask_for_env_parameters(readings, g);
58                 return true;
59               }
60               else
61                 return false;
62             }
63           }
64         }
65       }
66     }
67
68   }
69   }
70
71
72   return true;
73 }
```

We could divide this function in two main parts, depending on the value of the id_port variable, that represents the integer returned by the open function.

- **Line 13 - 19:** This is the first part, that corresponds to the case in which id_port is -1. If id_port is equal to -1, it means that the serial port has not been opened yet; in this case the function shall execute the setSerialPort process, in order to correctly open and set the serial port. After this, the serial port has been opened and configured with right parameters for the communication, but the master board is not programmed yet and it does not know to which of its four IO-Link node has to interact. So, the first set of commands has to be sent to the master board, by calling the function named ”program_master_node()”, in such a way that the serial communication is established and the node corresponding to the device is programmed. This is the very first section that shall be executed, during the start up of the system. It is like an initialization for the master board and it is executed just the first time a resource is asked to the device.
  When this section ends, the communication between the controller and the STEVAL-IDP004V1 has started and the master node is programmed to correctly communicate with the device attached to it.
  For completeness the code of the program_master_node() is reported in the next page:

```c
void program_master_node() {

char *buf = malloc(30*sizeof(char));

strcpy(buf, "START\n");
write(id_port, buf, 6);
sleep(3);

strcpy(buf, "MASTER\n");
write(id_port, buf, 7);
sleep(3);

strcpy(buf, "0\n");
write(id_port, buf, 2);
sleep(3);

strcpy(buf, "WR_S\n");
write(id_port, buf, 5);
sleep(3);

strcpy(buf, "096,248,033,122,122,122,122,\n");
for(int k=0; k<28; k++) {
  write(id_port, buf+k, 1);
  usleep(1000);
}

write(id_port, "\r", 1);
sleep(3);

free(buf);

}
```

As you can see, this function simply writes on the serial port the correct commands sequence in order to start communication with the master board and to program the node to which the IO-Link device is attached. The most critical part is the one in which register values are written. In this case, the function has to write one character at a time with an interval of 1 ms between one and the other; the reason of this lies in the fact that writing operation, performed by the PC, is too fast and the master is not capable to set registers correctly. By writing one character at a time, all works fine.
No reading operation are performed in this function, since, in programming phase, reading master responses is out of interest.

In the following two pictures the reader can see the master board before and after the programming phase.

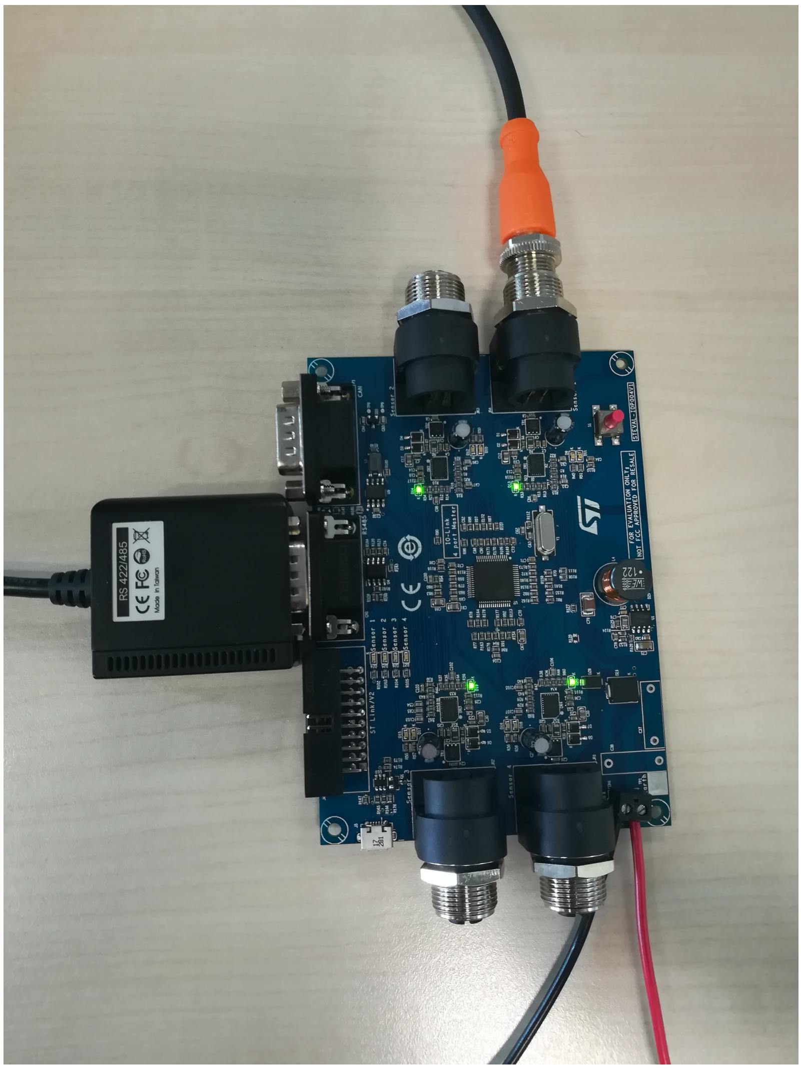


Figure 6.2. Master board view before programming phase

It is evident that, when the board is not programmed, only LED referred to power supply for each node are switched on; instead, after the programming phase, the node with the IO-Link cable attached to it has the LED, referred to the transceiver status, switched on.
Once the transceiver LED starts blinking, it means that the master board is

ready to communicate with the IO-Link sensor through the M12 cable and the controller could start asking resources to the device. From this moment the LED referred to the target node will continue blinking until the reset button is pressed or the power supply is switched off; this means that it is possible to retrieve data from the sensorized board for a long time without doing again the master configuration. This, obviously, speeds up data measurements. Furthermore, even in the case that the IO-Link device must be replaced, as we have seen in previous chapters, there is no need to re-configure the master node: it is sufficient to remove the device and to place a new one and the system will continue working correctly.

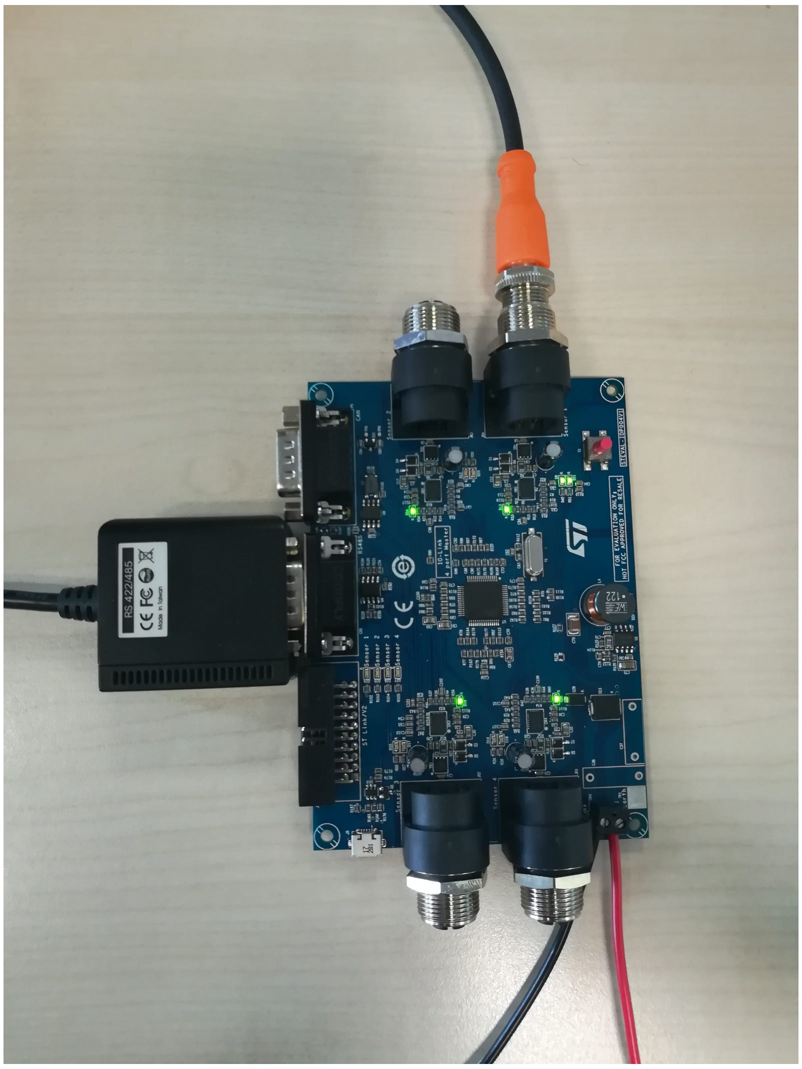


Figure 6.3. Master board view after programming phase

- **Line 21 - 69:** This is the second part of the serial_get_handler function, that corresponds to the case in which id_port is not equal to -1. If the id_port value is different from -1, it means that the serial communication has already been established. In this case, the configuration part described above is no more needed and the driver could directly send commands to the sensor.
  Inside this section, the main part is composed by an "if then else" decision tree used do discriminate between different parameter values. In fact, once parameter name has been retrieved by the "findinpairs" function, it is compared with all parameters name available for this device. For each type of resource requested, the driver has a specific function used to send commands to the device, through the master board.
  In the following we will analyze the function used to retrieve temperature data from the device.

```c
void ask_for_temperature(edgex_device_commandresult *readings, int index) {

char *buf = malloc(2000*sizeof(char));
char *ptr = buf; //ptr points to the first location of buf, used to free buf at the end if buf pointer will be modified
char *tmp = malloc(200*sizeof(char));
double temp_value;
int i;
strcpy(buf, "COMMAND END\n");
write(id_port, buf, 12);
sleep(3);

strcpy(buf, "DEVICE\n");
write(id_port, buf, 7);
sleep(3);

strcpy(buf, "0\n");
write(id_port, buf, 2);
sleep(3);

stpcpy(buf, "MSR\n");
write(id_port, buf, 4);
sleep(3);

tcflush(id_port, TCIFLUSH);

stpcpy(buf, "ENV\n");
write(id_port, buf, 4);
sleep(3);

30      i = read(id_port,buf,2000);
31
32
33      do{
34      sscanf(buf, "%s", tmp);
35      buf += strlen(tmp) + 1;
36
37      } while(strcmp(tmp, "from:") != 0);
38
39      sscanf(buf, "%*s %*s %*s %s", tmp);
40      temp_value = atof(tmp);
41
42      readings[index].type = Float32;
43      readings[index].value.f32_result = temp_value;
44
45      free(tmp);
46      free(ptr);
47
48      return;
49      }
```

The previous code show the behavior of the function used to ask temperature value to the device. It is a simple sequence of writing on the serial port and waiting few seconds to make possible the master board and the device to elaborate command received. Reading operation is performed just once, at the end of the writing sequence. In line 24 the reader could find the tcflush function call; this is used in order to discard all what has been written on the serial port, either by the controller or by the master board, that has not been read yet. The reason of this is that when the driver reads from the serial port, it is interested only in the response to the resource request and not to everything that has been written on the serial port. After this function has read the device data, it saves it in readings structure, that is a variable read by the framework to store data in the database.
Function to retrieve other resources from the IO-Link device are quite similar to the one just analyzed, so they are omitted in this treatise, in order to not make the reader boring.

In conclusion we could say that the behavior of the device driver, after the system initialization, is quite simple: it is based on writing a command for the device on the serial line, sleeping for some seconds in order to allow the device to process the command received and to send back the response, then writing the new command. Once all commands have been sent from the controller to the device, the

driver read the sensor's answer, retrieve data and measurements from it and send them to the EdgeX framework.
Even if it is not difficult to understand how does the driver works, its realization was not so easy. The main reason lies in the fact that the communication between the PC and the board requires an high precise synchronization, otherwise the master will reset itself and the transaction will fail. So, before we found the correct solution, several days of tests have been spent, trying to set different values of sleep between two sequential command write and looking for the best communication flow. In fact, different values of seconds for the sleep call have been tried, because certain commands work fine with some waiting time while other commands work better with other sleep value. The most critical phase is the one in which master board registers have to be programmed; in fact, in these step the communication failed every time. The solution was found by writing one character at a time, with a sleep of 1 ms between two subsequent char. All other commands, instead, work fine by setting a sleep of three seconds between one and the following.

6.5 Data Exporting

In order to make a complete data collection system, the cloud must be included in the work. In fact, data are more useful if they are available not only where they are collected, but also in remote places; the reason of this, lies in the fact that, maybe, someone wants to monitor the industrial machine or the environment in which it works even if he is not in the same place of the system controller. Furthermore, by exporting data to a cloud service it is possible to have multiple storing location and post processing operations could be performed without affecting controller's performances.

For this thesis work, we choose to use MQTT protocol to send data. MQTT stands for Message Queue Telemetry Transport and it is a messaging protocol designed to be used in systems in which bandwidth is limited or unreliable.[18] It is a publish-subscribe protocol based on a broker. The broker could be seen as the server of the MQTT communication; in fact, it is in charge of routing all the messages between clients. Each client could subscribe to a certain *topic*, in order to receive all messages published on that topic by other clients. The broker monitors each topic and, when a message is published on one of these, it forwards the message to all clients registered to that target topic. Connection messages and acknowledges are exchanged between client and broker before communication starts. In the following picture the reader could find the typical MQTT message flow.

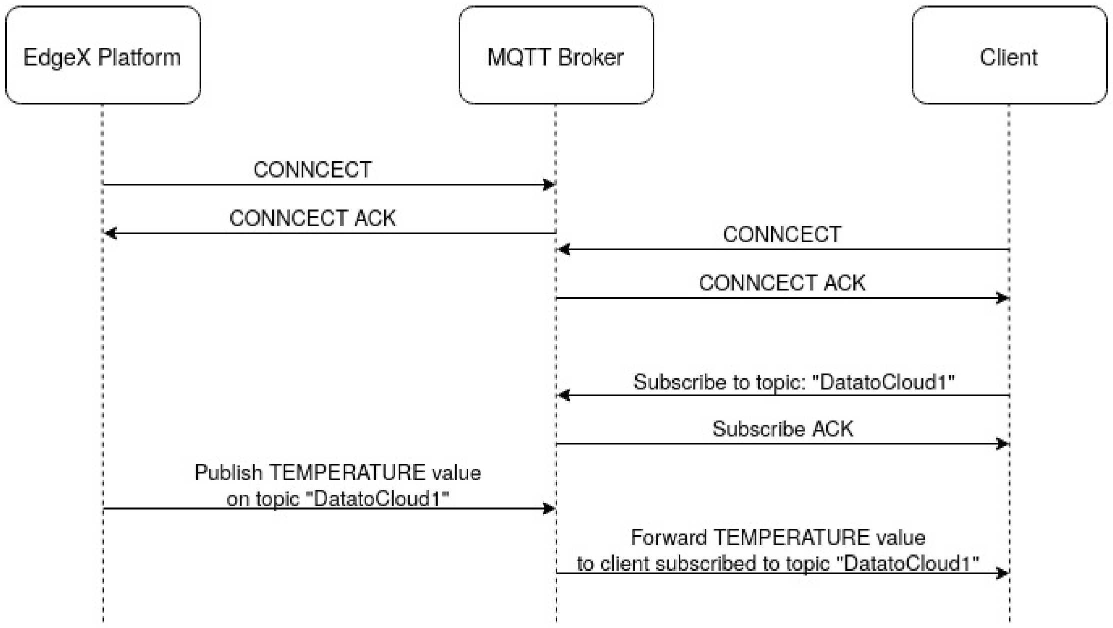
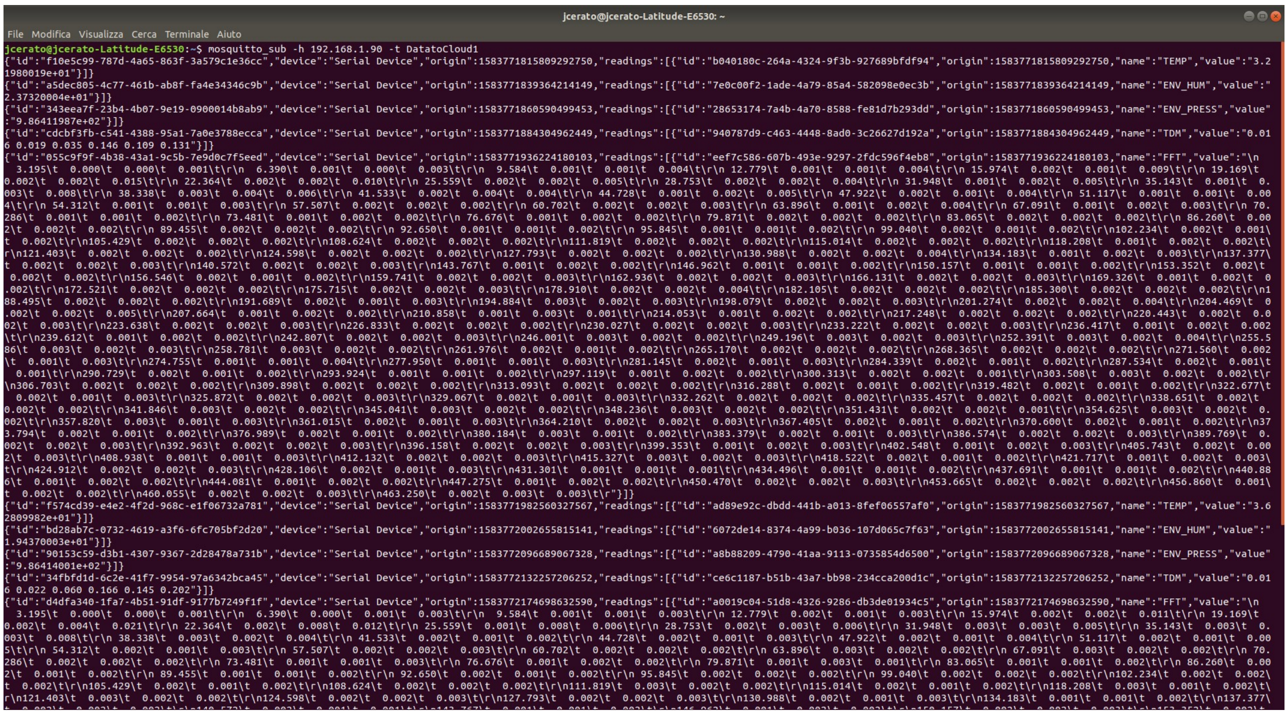

EdgeX Platform
MQTT Broker
Client
CONNCECT
CONNCECT ACK
CONNCECT
CONNCECT ACK
Subscribe to topic: "DatatoCloud1"
Subscribe ACK
Publish TEMPERATURE value on topic "DatatoCloud1"
Forward TEMPERATURE value to client subscribed to topic "DatatoCloud1"

Figure 6.4. MQTT message flow

In order to forward data to an MQTT broker it has been exploited the EdgeX Foundry^{TM} Export Client service. Every time a data reaches the Core Data microservice, the Export Client service will send the data to all the clients registered to it. This is a very powerful tool, since data could be forwarded to multiple clients with multiple protocols in the same time. In order to use this feature of the platform, we need to register the MQTT broker as a client to which export data.
For this project work, we use Eclipse Mosquitto^{TM} because it is one of the most popular open source MQTT broker that could be directly installed on the controller machine. The registration to the Export service is performed through the HTTP REST POST method. In the POST body it is possible to specify the protocol through which send data to the client, in this case is MQTT, the host name and the port through which the broker is reachable, the topic on which to publish the data and, finally, the login parameters to connect to the broker, such as username and password. If the POST was successful, it is possible to connect to the broker and subscribe to the topic in order to receive data from EdgeX platform.
In the picture below the reader can verify the correct behavior of the system; it is evident, in fact, that, by subscribing to the correct host address and the correct topic, it is possible to receive data and measurements from the device. More in details, it is demonstrated that all type of data have been received: temperature, humidity, pressure, acceleration and vibration.
All data are available in JSON format; in readings field it is possible to find the

name of the resource and its value.


jcerato@jcerato-Latitude-E6530: ~
File Modifica Visualizza Cerca Terminale Aiuto

```
jcerato@jcerato-Latitude-E6530:~$ mosquitto_sub -h 192.168.1.90 -t DatatoCloud1
{"id":"f10e5c99-787d-4a65-863f-3a579c1e36cc","device":"Serial Device","origin":1583771815809292750,"readings":[{"id":"b040180c-264a-4324-9f3b-927689bfdf94","origin":1583771815809292750,"name":"TEMP","value":"3.2
1980019e+01"}]}
{"id":"a5dec805-4c77-461b-ab8f-fa4e34346c9b","device":"Serial Device","origin":1583771839364214149,"readings":[{"id":"7e0c00f2-1ade-4a79-85a4-582098e0ec3b","origin":1583771839364214149,"name":"ENV_HUM","value":"
2.37320004e+01"}]}
{"id":"343eea7f-23b4-4b07-9e19-0900014b8ab9","device":"Serial Device","origin":1583771860590499453,"readings":[{"id":"28653174-7a4b-4a70-8588-fe81d7b293dd","origin":1583771860590499453,"name":"ENV_PRESS","value"
:"9.86411987e+02"}]}
{"id":"cdcbf3fb-c541-4388-95a1-7a0e3788ecca","device":"Serial Device","origin":1583771884304962449,"readings":[{"id":"940787d9-c463-4448-8ad0-3c26627d192a","origin":1583771884304962449,"name":"TDM","value":"0.01
6 0.019 0.035 0.146 0.109 0.131"}]}
{"id":"055c9f9f-4b38-43a1-9c5b-7e9d0c7f5eed","device":"Serial Device","origin":1583771936224180103,"readings":[{"id":"eef7c586-607b-493e-9297-2fdc596f4eb8","origin":1583771936224180103,"name":"FFT","value":"\n
  3.195\t  0.000\t  0.000\t  0.001\t\r\n  6.390\t  0.001\t  0.000\t  0.003\t\r\n  9.584\t  0.001\t  0.001\t  0.004\t\r\n 12.779\t  0.001\t  0.001\t  0.004\t\r\n 15.974\t  0.002\t  0.001\t  0.009\t\r\n 19.169\t
0.002\t  0.002\t  0.015\t\r\n 22.364\t  0.002\t  0.002\t  0.010\t\r\n 25.559\t  0.002\t  0.002\t  0.005\t\r\n 28.753\t  0.002\t  0.002\t  0.004\t\r\n 31.948\t  0.001\t  0.002\t  0.005\t\r\n 35.143\t  0.001\t  0.
003\t  0.008\t\r\n 38.338\t  0.003\t  0.004\t  0.006\t\r\n 41.533\t  0.002\t  0.004\t  0.004\t\r\n 44.728\t  0.001\t  0.002\t  0.005\t\r\n 47.922\t  0.002\t  0.001\t  0.004\t\r\n 51.117\t  0.001\t  0.001\t  0.00
4\t\r\n 54.312\t  0.001\t  0.001\t  0.003\t\r\n 57.507\t  0.002\t  0.002\t  0.002\t\r\n 60.702\t  0.002\t  0.002\t  0.003\t\r\n 63.896\t  0.001\t  0.002\t  0.004\t\r\n 67.091\t  0.001\t  0.002\t  0.003\t\r\n 70.
286\t  0.001\t  0.001\t  0.002\t\r\n 73.481\t  0.001\t  0.002\t  0.002\t\r\n 76.676\t  0.001\t  0.002\t  0.002\t\r\n 79.871\t  0.002\t  0.002\t  0.002\t\r\n 83.065\t  0.002\t  0.002\t  0.002\t\r\n 86.260\t  0.00
2\t  0.002\t  0.002\t\r\n 89.455\t  0.002\t  0.002\t  0.002\t\r\n 92.650\t  0.001\t  0.001\t  0.002\t\r\n 95.845\t  0.001\t  0.001\t  0.002\t\r\n 99.040\t  0.002\t  0.001\t  0.002\t\r\n102.234\t  0.002\t  0.001\
t  0.002\t\r\n105.429\t  0.002\t  0.002\t  0.002\t\r\n108.624\t  0.002\t  0.002\t  0.002\t\r\n111.819\t  0.002\t  0.002\t  0.002\t\r\n115.014\t  0.002\t  0.002\t  0.002\t\r\n118.208\t  0.001\t  0.002\t  0.002\t\
r\n121.403\t  0.002\t  0.002\t  0.002\t\r\n124.598\t  0.002\t  0.002\t  0.002\t\r\n127.793\t  0.002\t  0.002\t  0.002\t\r\n130.988\t  0.002\t  0.002\t  0.004\t\r\n134.183\t  0.001\t  0.002\t  0.003\t\r\n137.377\
t  0.002\t  0.002\t  0.003\t\r\n140.572\t  0.002\t  0.002\t  0.003\t\r\n143.767\t  0.001\t  0.002\t  0.002\t\r\n146.962\t  0.001\t  0.001\t  0.002\t\r\n150.157\t  0.001\t  0.001\t  0.002\t\r\n153.352\t  0.002\t
 0.002\t  0.002\t\r\n156.546\t  0.002\t  0.001\t  0.002\t\r\n159.741\t  0.002\t  0.002\t  0.003\t\r\n162.936\t  0.002\t  0.002\t  0.003\t\r\n166.131\t  0.002\t  0.002\t  0.003\t\r\n169.326\t  0.001\t  0.002\t  0
.002\t\r\n172.521\t  0.002\t  0.002\t  0.002\t\r\n175.715\t  0.002\t  0.002\t  0.003\t\r\n178.910\t  0.002\t  0.002\t  0.004\t\r\n182.105\t  0.002\t  0.002\t  0.002\t\r\n185.300\t  0.002\t  0.002\t  0.002\t\r\n1
88.495\t  0.002\t  0.002\t  0.002\t\r\n191.689\t  0.002\t  0.001\t  0.003\t\r\n194.884\t  0.003\t  0.002\t  0.003\t\r\n198.079\t  0.002\t  0.002\t  0.003\t\r\n201.274\t  0.002\t  0.002\t  0.004\t\r\n204.469\t  0
.002\t  0.002\t  0.005\t\r\n207.664\t  0.001\t  0.002\t  0.002\t\r\n210.858\t  0.001\t  0.003\t  0.001\t\r\n214.053\t  0.001\t  0.002\t  0.002\t\r\n217.248\t  0.002\t  0.002\t  0.002\t\r\n220.443\t  0.002\t  0.0
02\t  0.003\t\r\n223.638\t  0.002\t  0.002\t  0.003\t\r\n226.833\t  0.002\t  0.002\t  0.002\t\r\n230.027\t  0.002\t  0.002\t  0.003\t\r\n233.222\t  0.002\t  0.002\t  0.003\t\r\n236.417\t  0.001\t  0.002\t  0.002
\t\r\n239.612\t  0.001\t  0.002\t  0.002\t\r\n242.807\t  0.002\t  0.002\t  0.003\t\r\n246.001\t  0.003\t  0.002\t  0.002\t\r\n249.196\t  0.003\t  0.002\t  0.003\t\r\n252.391\t  0.003\t  0.002\t  0.004\t\r\n255.5
86\t  0.003\t  0.002\t  0.003\t\r\n258.781\t  0.003\t  0.002\t  0.002\t\r\n261.976\t  0.002\t  0.001\t  0.002\t\r\n265.170\t  0.002\t  0.002\t  0.002\t\r\n268.365\t  0.002\t  0.002\t  0.002\t\r\n271.560\t  0.002
\t  0.001\t  0.003\t\r\n274.755\t  0.001\t  0.001\t  0.004\t\r\n277.950\t  0.001\t  0.001\t  0.003\t\r\n281.145\t  0.002\t  0.001\t  0.003\t\r\n284.339\t  0.002\t  0.001\t  0.002\t\r\n287.534\t  0.002\t  0.001\t
  0.001\t\r\n290.729\t  0.002\t  0.001\t  0.002\t\r\n293.924\t  0.001\t  0.001\t  0.002\t\r\n297.119\t  0.001\t  0.002\t  0.002\t\r\n300.313\t  0.002\t  0.002\t  0.001\t\r\n303.508\t  0.003\t  0.002\t  0.002\t\r
\n306.703\t  0.002\t  0.002\t  0.002\t\r\n309.898\t  0.002\t  0.002\t  0.002\t\r\n313.093\t  0.002\t  0.002\t  0.002\t\r\n316.288\t  0.002\t  0.001\t  0.002\t\r\n319.482\t  0.002\t  0.001\t  0.002\t\r\n322.677\t
  0.002\t  0.001\t  0.003\t\r\n325.872\t  0.002\t  0.002\t  0.003\t\r\n329.067\t  0.002\t  0.001\t  0.003\t\r\n332.262\t  0.002\t  0.002\t  0.002\t\r\n335.457\t  0.002\t  0.002\t  0.002\t\r\n338.651\t  0.002\t
0.002\t  0.002\t\r\n341.846\t  0.003\t  0.002\t  0.002\t\r\n345.041\t  0.003\t  0.002\t  0.002\t\r\n348.236\t  0.003\t  0.002\t  0.002\t\r\n351.431\t  0.002\t  0.002\t  0.001\t\r\n354.625\t  0.003\t  0.002\t  0.
002\t\r\n357.820\t  0.003\t  0.001\t  0.003\t\r\n361.015\t  0.002\t  0.001\t  0.003\t\r\n364.210\t  0.002\t  0.002\t  0.003\t\r\n367.405\t  0.002\t  0.001\t  0.002\t\r\n370.600\t  0.002\t  0.001\t  0.002\t\r\n37
3.794\t  0.002\t  0.001\t  0.002\t\r\n376.989\t  0.002\t  0.001\t  0.002\t\r\n380.184\t  0.003\t  0.001\t  0.002\t\r\n383.379\t  0.002\t  0.001\t  0.003\t\r\n386.574\t  0.002\t  0.002\t  0.003\t\r\n389.769\t  0.
002\t  0.002\t  0.003\t\r\n392.963\t  0.002\t  0.002\t  0.003\t\r\n396.158\t  0.002\t  0.002\t  0.003\t\r\n399.353\t  0.001\t  0.002\t  0.003\t\r\n402.548\t  0.001\t  0.002\t  0.003\t\r\n405.743\t  0.002\t  0.00
2\t  0.003\t\r\n408.938\t  0.001\t  0.001\t  0.003\t\r\n412.132\t  0.002\t  0.002\t  0.003\t\r\n415.327\t  0.003\t  0.002\t  0.003\t\r\n418.522\t  0.002\t  0.001\t  0.002\t\r\n421.717\t  0.001\t  0.002\t  0.003\
t\r\n424.912\t  0.002\t  0.002\t  0.003\t\r\n428.106\t  0.002\t  0.001\t  0.003\t\r\n431.301\t  0.001\t  0.001\t  0.001\t\r\n434.496\t  0.001\t  0.001\t  0.002\t\r\n437.691\t  0.001\t  0.001\t  0.002\t\r\n440.88
6\t  0.001\t  0.002\t  0.002\t\r\n444.081\t  0.001\t  0.002\t  0.002\t\r\n447.275\t  0.001\t  0.002\t  0.002\t\r\n450.470\t  0.002\t  0.002\t  0.003\t\r\n453.665\t  0.002\t  0.002\t  0.002\t\r\n456.860\t  0.001\
t  0.002\t  0.002\t\r\n460.055\t  0.002\t  0.002\t  0.003\t\r\n463.250\t  0.002\t  0.003\t  0.003\t\r"}]}
{"id":"f574cd39-e4e2-4f2d-968c-e1f06732a781","device":"Serial Device","origin":1583771982560327567,"readings":[{"id":"ad89e92c-dbdd-441b-a013-8fef06557af0","origin":1583771982560327567,"name":"TEMP","value":"3.6
2809982e+01"}]}
{"id":"bd28ab7c-0732-4619-a3f6-6fc705bf2d20","device":"Serial Device","origin":1583772002655815141,"readings":[{"id":"6072de14-8374-4a99-b036-107d065c7f63","origin":1583772002655815141,"name":"ENV_HUM","value":"
1.94370003e+01"}]}
{"id":"90153c59-d3b1-4307-9367-2d28478a731b","device":"Serial Device","origin":1583772096689067328,"readings":[{"id":"a8b88209-4790-41aa-9113-0735854d6500","origin":1583772096689067328,"name":"ENV_PRESS","value"
:"9.86414001e+02"}]}
{"id":"34fbfd1d-6c2e-41f7-9954-97a6342bca45","device":"Serial Device","origin":1583772132257206252,"readings":[{"id":"ce6c1187-b51b-43a7-bb98-234cca200d1c","origin":1583772132257206252,"name":"TDM","value":"0.01
6 0.022 0.060 0.166 0.145 0.202"}]}
{"id":"d4dfa340-1fa7-4b51-91df-9177b7249f1f","device":"Serial Device","origin":1583772174698632590,"readings":[{"id":"a0019c04-51d8-4326-9286-db3de01934c5","origin":1583772174698632590,"name":"FFT","value":"\n
  3.195\t  0.000\t  0.000\t  0.001\t\r\n  6.390\t  0.000\t  0.001\t  0.003\t\r\n  9.584\t  0.001\t  0.001\t  0.003\t\r\n 12.779\t  0.002\t  0.001\t  0.003\t\r\n 15.974\t  0.002\t  0.002\t  0.011\t\r\n 19.169\t
0.002\t  0.004\t  0.021\t\r\n 22.364\t  0.002\t  0.008\t  0.012\t\r\n 25.559\t  0.001\t  0.008\t  0.006\t\r\n 28.753\t  0.002\t  0.003\t  0.006\t\r\n 31.948\t  0.003\t  0.003\t  0.005\t\r\n 35.143\t  0.003\t  0.
003\t  0.008\t\r\n 38.338\t  0.003\t  0.002\t  0.004\t\r\n 41.533\t  0.002\t  0.001\t  0.002\t\r\n 44.728\t  0.002\t  0.001\t  0.003\t\r\n 47.922\t  0.002\t  0.001\t  0.004\t\r\n 51.117\t  0.002\t  0.001\t  0.00
5\t\r\n 54.312\t  0.002\t  0.001\t  0.003\t\r\n 57.507\t  0.002\t  0.002\t  0.003\t\r\n 60.702\t  0.002\t  0.002\t  0.002\t\r\n 63.896\t  0.003\t  0.002\t  0.002\t\r\n 67.091\t  0.003\t  0.002\t  0.002\t\r\n 70.
286\t  0.002\t  0.002\t  0.002\t\r\n 73.481\t  0.001\t  0.001\t  0.003\t\r\n 76.676\t  0.001\t  0.002\t  0.002\t\r\n 79.871\t  0.001\t  0.003\t  0.001\t\r\n 83.065\t  0.001\t  0.001\t  0.002\t\r\n 86.260\t  0.00
2\t  0.001\t  0.002\t\r\n 89.455\t  0.001\t  0.001\t  0.002\t\r\n 92.650\t  0.002\t  0.001\t  0.002\t\r\n 95.845\t  0.002\t  0.002\t  0.002\t\r\n 99.040\t  0.002\t  0.002\t  0.002\t\r\n102.234\t  0.002\t  0.002\
t  0.002\t\r\n105.429\t  0.002\t  0.001\t  0.002\t\r\n108.624\t  0.002\t  0.002\t  0.002\t\r\n111.819\t  0.003\t  0.002\t  0.002\t\r\n115.014\t  0.002\t  0.001\t  0.002\t\r\n118.208\t  0.003\t  0.001\t  0.002\t\
r\n121.403\t  0.003\t  0.002\t  0.002\t\r\n124.598\t  0.002\t  0.002\t  0.003\t\r\n127.793\t  0.002\t  0.002\t  0.003\t\r\n130.988\t  0.002\t  0.001\t  0.003\t\r\n134.183\t  0.001\t  0.001\t  0.002\t\r\n137.377\
```

Figure 6.5. MQTT received messages

6.6 The dashboard

All the features of this collection data system could be used also without this dashboard. Anyway, this requires to be an expert user and, moreover, it is a quite tricky usage. For these reasons, we thought of creating a user interface to be provided to the user to make the use of the system more intuitive and simpler.

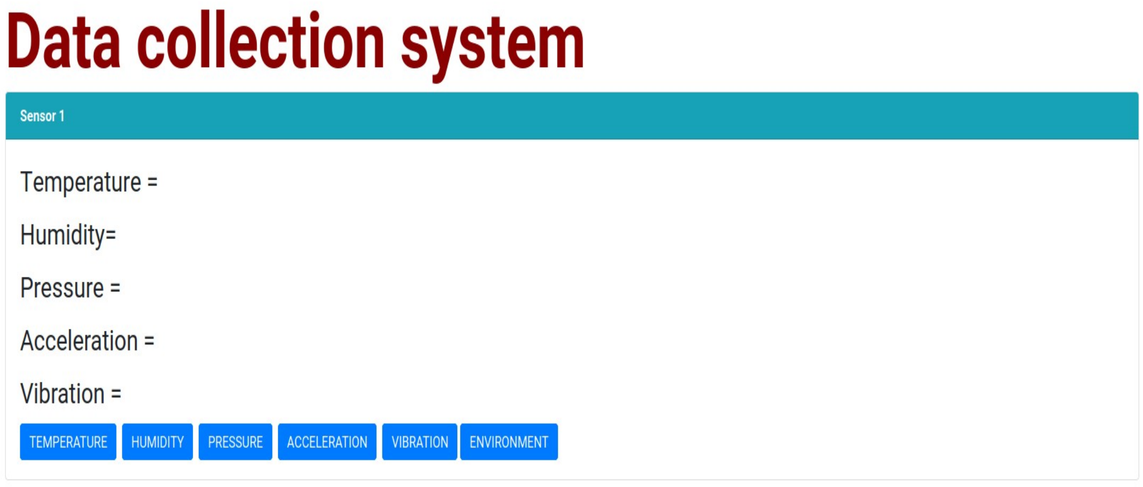


Figure 6.6. Dashboard view

As you can see in the picture above, this UI provides the user a ”user-friendly” way to collect data for condition monitoring. By simply clicking on a button, the corresponding resource will be printed on the screen in a few seconds.
This dashboard was made using Angular, an open source framework for developing web applications. The user just have to click on a button and the system will do everything autonomously. If it is the first execution of the system, the master board will be programmed; if, instead, the system has already been initialized, it will just retrieve the data from the sensorized device.
The dashboard exploits the Angular HttpClient injectable class, designed in order to perform HTTP requests. This class provides several methods like GET, PUT, POST, DELETE to make HTTP requests. For this project work we use the GET method, since we have to retrieve data from the device and, as seen in previous chapters, this is performed through HTTP REST API; the GET method ”constructs an observable that, when subscribed, causes the configured GET request to execute on the server”[19]. So, the main component of the dashboard is made in such a way that, when it detects a button click, it subscribes to the get request correspondent to the button resource and it retrieves the data from the device. Once the device sends back the requested data, it is visible on the dashboard. In the same time, the system will automatically store the data in the database and send it to the cloud, via MQTT.

Chapter 7

Testing and validation

7.1 Testing

Once the implementation phase has reached the end, several hours of tests have been spent to verify the correct behavior of the data collection system.
The tests performed allowed us to verify that:

- The system is able to correctly program the master board IO-Link tansceiver node, in order to make it communicate with the IO-Link device
- The system is able to send commands to the IO-Link device in order to retrieve data and measurements from it
- All data of interest could be retrieved from the device: temperature, humidity, pressure, acceleration in time domain and vibration analysis values
- The system is able to store device data in the database
- The system is able to forward data outside using MQTT protocol
- The dashboard provided to the user is able to show data retrieved from the device

7.2 Device Cost

As previously stated, the EdgeX Foundry IoT framework is an open source one, so there are no costs for its use. This obviously is a big advantage of this data collection system.
With respect to the two devices involved in this project we could say that the most expensive is the master board. It has a cost of around 150 Euros.

The IO-Link device, instead, has a cost of about 120 Euros and it is sent with all the components needed to program it. It is possible to find in the kit also all tool needed to use the sensorized board without the master board, exploiting the capability to communicate via service UART.
In conclusion, the total price is around 270 Euros, but this is referred to just one of both two products. It is expected that total cost decreases in case of mass production of the system.

Chapter 8

Conclusions

8.1 Power supply

In these months of implementation and use of the data collection system we realize that devices, even when stressed with continuous commands, never require more than 80 mA at 24V. For this reason, I think that it is possible to supply the system also with batteries. If we take, for example, a commercial battery, that is possible to find on the web, with 24 V output and 12 Ah of charge, we could make some calculation to verify if the system could work autonomously.
So, we know that the device requires 80 mA @ 24 V:

$$80mA * 24V = 1920mW$$

The battery is able to provide:

$$12000mAh * 24V = 288000mWh$$

Now, since this is not an ideal battery, let we assume that it has an efficiency (discharge) factor of 95%:

$$288000mWh * 0.95 = 273600mWh$$

In conclusion we could define the total hours that the battery is able to supply the system:

$$273600mWh/1920mW = 142.5h$$

So the system could work for 142 hours continuously. This means that if the system has to monitor the environment all day and all night, the battery shall be recharged around every six days. If we assume, instead, that the system has to monitor machinery or environment for ten hours at a day, the battery charge is capable to supply it for more than two weeks.

8.2 Results

Reached the end of this thesis project, I am satisfied because, during this months, I had the possibility to improve my skills and to challenge myself with technologies and tools that I didn't know before. In particular, I acquired experience on EdgeX Foundry platform and on Docker and Docker Compose technologies, because the open source framework is based on containers. Furthermore, I learned how IO-Link protocol works and I had the opportunity to manage devices and to work with them. Finally, I also learned how to design a dashboard, using Angular framework, and how does the MQTT protocol work. Of course, I think all these knowledges will become useful for my future working career.

About this work, all components involved in the collection data system works in the proper and desired manner. This means that we have been able to design and realize this system in its entirely.
The driver is able to program master node in order to correctly communicate with the IO-link device; we made several tests and this process works every time. After the programming phase, the system is ready to send commands to the device; this operation is performed, again, automatically by the system and it behaves correctly. Every time the controller make a request for a certain data, the data will become available in few seconds. In fact, once the device sends back a certain data to the controller, it is displayed on the dashboard and, in the same time, it is stored in the MongoDB database and forwarded to each client registered to the MQTT topic on which messages are published by the system.

Results reached with this thesis work, demonstrate that integration between IO-Link devices and EdgeX Foundry platform, in order to perform environment or machinery monitoring, it is possible. The data collection system is consistent and works fine and the data flow from the device to the cloud never crashes during this period of tests.
Obviously, this is just a prototype and improvements or extensions could be applied to it in order to use it in production scenarios. The reader could find some hints on how to improve the system in future steps section.

8.3 Future steps

This section describes both possible future steps, that could be performed in order to improve the data collection system, and works related to it.

Firmware update

During this months in which we use the master board and the device, several firmware issues have appeared. In fact, for example, if there is not a perfect synchronization in the communication between the controller and the master board, the latter resets itself and communication flow starts again from the beginning. This issue could be solved by modifying the firmware of the master board.
Another problem we encountered is the fact that the device is not able to send to the controller raw accelerometers data. This is a software matter and it is possible to solve it by modifying the firmware for the IO-Link device.

More than one device

All this thesis work was performed using just one IO-Link device. Obviously, in order to improve the data collection system, a good idea could be to involve more than one device. The master board is capable to have four devices attached to it and it would be interesting trying to exploit all of these four.
With more than one IO-Link devices it is possible to better monitor the environment or, even more, thanks to cables long up to 20 meters, it is possible to monitor more than one environment or several industrial machinery.
In order to use up to four devices, little modifications have to be applied to the driver in order to adapt it to manage communications with and resources retrieval from several sensors. In particular, attention has to be applied on the selection of the node on which to send commands, since it is critical the case in which one wants to monitor machine vibrations and it gets instead vibrations of a device mounted on a wall to monitor environment temperature and humidity.
In the following picture the reader can find a schematic of the system with four devices available.

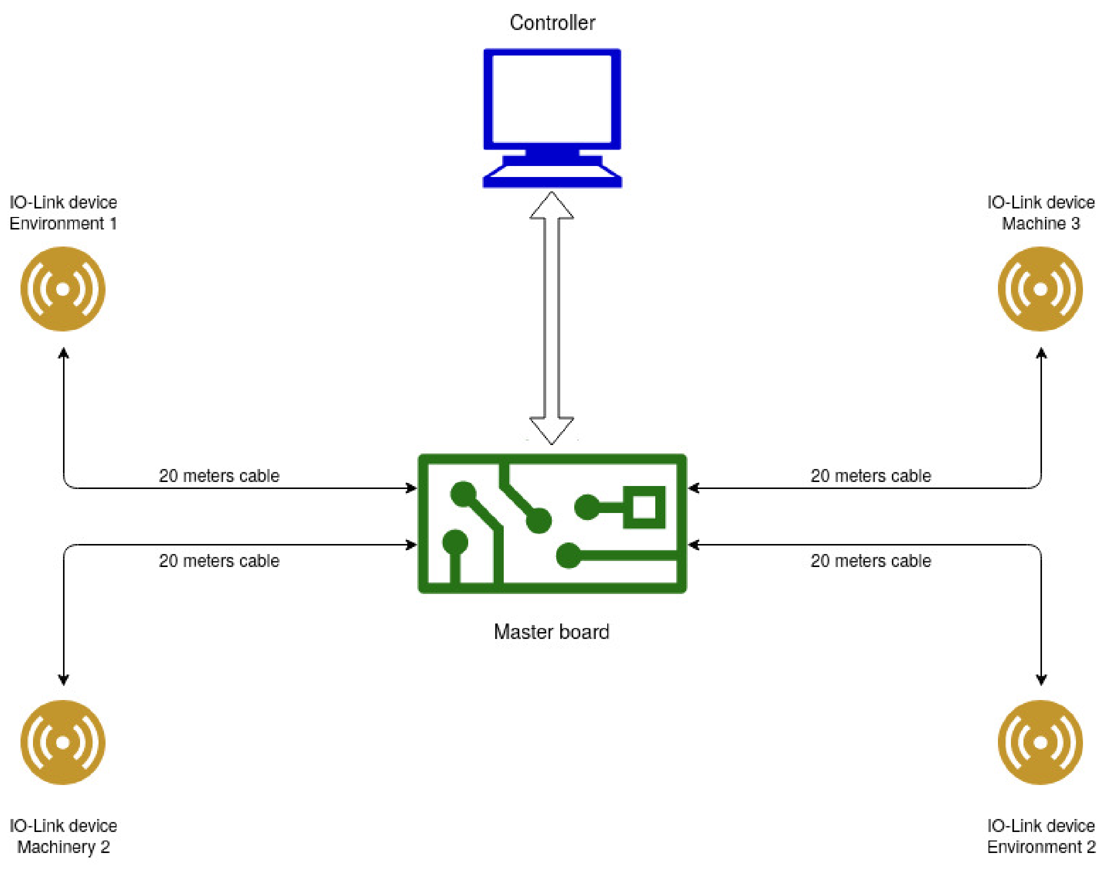


Figure 8.1. System overview with four IO-Link devices

Security implementation

Security is an increasingly important topic in the digital world. For that reason, also in the data collection system, communication flow could be secure. In fact it is reasonable that some industry do not want to have their production data stolen or hacked.
Data from the device to the database are exchanged on top of HTTPS, therefore this branch of the communication is already secure.
One interesting improvement that could be applied to the system is to make the MQTT transmission, between the framework and the cloud, secure. A first level of security could be introduced by setting username and password to publish or subscribe to a certain topic. A second layer of security could be achieved by using encryption of data.

Machine Learning

During recent times, machine learning is becoming really popular. Since the data collection system is able to make a lot of measurements both on environment parameters and on accelerometers parameters, machine learning algorithms could be applied to these data. This could be a useful improvement, since by exploiting machine learning algorithms it is possible to do predictive maintenance on monitored machinery. For example, by analyzing raw accelerometers data, it is possible to detect when a certain machine is going to break due to strong vibrations. In the same manner, it is possible to detect when a specific component is becoming too hot, by analyzing temperature measurements obtained with the data collection system.

Figure 8.2. Machine learning representation. Taken from [20]

Bibliography

[1] User manual. https://www.st.com/content/ccc/resource/technical/document/user_manual/group1/41/68/d4/a7/8e/62/4d/bb/DM00518672/files/DM00518672.pdf/jcr:content/translations/en.DM00518672.pdf.

[2] Internet delle cose. https://it.wikipedia.org/wiki/Internet_delle_cose.

[3] L'internet delle cose (iot): cos'e' e come rivoluzionera' prodotti e servizi. https://www.zerounoweb.it/analytics/big-data/internet-of-things-iot-come-funziona/.

[4] Giovanni Miragliotta. Industrial internet of things: definizione, applicazioni e diffusione. https://blog.osservatori.net/it_it/industrial-iot-definizione-applicazioni.

[5] IO-Link community. Io-link system description - technology and application. https://io-link.com/share/Downloads/At-a-glance/IO-Link_System_Description_eng_2018.pdf.

[6] Jurgen Reiser. Overview: How does the io-link work? https://blog.wika.com/knowhow/overview-io-link/.

[7] Io-link advantages. https://www.balluff.com/en/de/industries-and-solutions/solutions-and-technologies/io-link/io-link-benefits/.

[8] Io-link - what it is and 5 key advantages. https://www.bannerengineering.com/us/en/company/expert-insights/io-link.html.

[9] Did you know...? https://io-link.com/en/Technology/DidYouKnow/DidYouKnow_MeasuringSensors.php.

[10] Did you know...? https://io-link.com/en/Technology/DidYouKnow/DidYouKnow_ReducesInterfaces.php.

[11] Lsm6ds3: always-on 3d accelerometer and 3d gyroscope. https://www.st.com/content/ccc/resource/technical/document/application_note/12/98/b4/44/a5/bf/4e/c5/DM00157511.pdf/files/DM00157511.pdf/jcr:content/translations/en.DM00157511.pdf.

[12] Stmicroelectronics hts221 capacitive digital humidity sensor. https://www.mouser.it/new/stmicroelectronics/stm-hts221-sensor/.

[13] Lps22hb. http://www.st.com/resource/en/datasheet/lps22hb.pdf.

[14] L6984. https://www.st.com/en/power-management/l6984.html.

[15] Ldk220. https://www.st.com/en/power-management/ldk220.html.

[16] Io-link solution based on steval-idp004v2 master evaluation board and steval-idp003v1 kit. https://www.st.com/content/ccc/resource/technical/document/application_note/group0/10/d0/2b/48/af/7a/4f/11/DM00402784/files/DM00402784.pdf/jcr:content/translations/en.DM00402784.pdf.

[17] Edgex foundry documentation. https://docs.edgexfoundry.org/1.2/.

[18] Mqtt. https://en.wikipedia.org/wiki/MQTT.

[19] Angular - httpclient. https://angular.io/api/common/http/HttpClient#get.

[20] Donato Ceccomancini. La rinascita del machine learning e' gia' qui. https://www.bitmat.it/blog/news/76033/la-rinascita-del-machine-learning-gia.