




# **81605 Industrial Communication**

## Laboratory Handbook

Version 1.2.0  
SS2025



## Content

1	Introduction .....	4
2	Requirements .....	5
2.1	Hardware .....	5
2.2	Software .....	5
2.2.1	Visual Studio Code .....	6
2.2.2	PlatformIO .....	6
2.2.3	Node Red .....	6
2.2.4	Logic 2 .....	7
2.2.5	Git .....	7
3	Task 1 Modbus RTU .....	8
3.1	Learning objectives .....	8
3.2	Introduction .....	8
3.3	Self-learning Material .....	8
3.4	Preparation .....	8
3.5	Hardware Requirements .....	9
3.6	Modbus Temperature/Humidity Sensor .....	9
3.6.1	Hardware connections .....	11
3.7	Testing the sensor .....	11
3.8	Analyzing the Modbus protocol .....	12
4	Task 2 HTTP Requests .....	15
4.1	Learning objectives .....	15
4.2	Self-learning Material .....	15
4.3	Hardware Requirements .....	15
4.4	ESP32: PlatformIO configuration .....	15
4.4.1	Create a new Project .....	16
4.4.2	Build a Project .....	16
4.4.3	Upload a program .....	17
4.4.4	Monitor a program .....	17
4.4.5	Further information .....	18
4.4.6	POST Requests .....	18
4.4.7	GET Requests .....	20
4.5	HTTP Request Analysis .....	20
5	Task 3 Project Modbus TCP .....	23
5.1	Learning objectives .....	23
5.2	Hardware Requirements .....	23
5.3	Introduction .....	23
5.4	Requirements .....	23
5.4.1	Modbus RTU .....	23
5.4.2	Access point .....	24
5.4.3	Webserver .....	24
5.4.4	Modbus TCP .....	24
5.4.5	Front End .....	24
5.5	Implementation .....	25

5.5.1	Modbus RTU .....	25
5.5.2	HTTP Web Server .....	26
5.5.3	Modbus TCP .....	26
6	Task 4 CAN Bus .....	27
6.1	Learning objectives .....	27
6.2	Self-learning Material .....	27
6.3	Hardware Requirements .....	27
6.4	Introduction .....	27
6.5	Sending CAN Frames .....	28
6.5.1	Hardware connections .....	28
6.5.2	Software .....	28
6.5.3	CAN over Logic Analyzer .....	29
6.5.4	USB CAN Analyzer .....	29
6.6	Receiving CAN Frames .....	31
6.7	Node-RED .....	31
6.7.1	Sending a CAN Messages to Node-RED .....	31
7	Task 5 µProject CAN Joystick .....	33
7.1	Learning objectives .....	33
7.2	Self-learning Material .....	33
7.3	Hardware Requirements .....	33
7.4	Introduction .....	33
7.5	Objectives .....	33
7.6	Implementation details .....	34
7.6.1	CR1301 Joystick .....	34
7.6.2	SAE J1939 .....	34
8	Task 6 IO-Link MQTT .....	36
8.1	Self-learning Material .....	36
8.2	Hardware Requirements .....	36
8.3	Software Requirements .....	36
8.4	Objectives .....	36
8.5	IO-Link Ethernet Setup .....	37
9	Task 7 Arduino IO-Link Device .....	40
9.1	Hardware Requirements .....	40
9.2	Introduction .....	40
9.2.1	IO-Link shield .....	40
9.2.2	IO-Link Library .....	41
9.2.3	IO-Link Master .....	42
	Laboratory Session Log .....	43

# 1 Introduction

The purpose of the practical session of this course is the independent learning of each student. Therefore, it can only make sense to work on the tasks independently. Nevertheless, if a task is organized to be solved in a team or group, each student should independently prepare for the task and not rely on their teammates. The exam won't be done as a team and thus it is of utmost importance that each student dedicates enough time to prepare for the practical part of this course. To pass the practical part of this course, it is necessary to solve all tasks.

Each task may include a section called **Self-Learning**. The student is required to do independent research about the topic they will be working on, and this section will contain information that the student should consult and read to complete their work. As Unique Resource Locators (URLs) can change over time, we provide search engine keywords that can be used to point to relevant online material. In some cases, specific URLs are mentioned in case there is important learning material that the student must read. In the unfortunate case that an URL is not valid or not accessible, please notify the laboratory supervisor.

## Important:

- The most common mistake by students is when skipping or not reading all the instructions provided in this document. Please read this document thoroughly.
- If you are not able to dedicate sufficient time to the preparation of the laboratory sessions and complete successfully the tasks it is recommended that the course is taken in another semester as students tend to underestimate their workload.
- **READ** each page of this document and do not skip sections before asking questions that can be answered by **reading comprehension skills**.
- You have to bring this workbook to every practical session (digitally or physically).
- If the task is completed, the supervisor will sign the task as completed in the respective section of the manual dedicated to keeping track of your progress.
- To complete the whole practice course, you have to hand out the dedicated page to keep track of your progress which is part of this handbook.
- Exam participation is only possible by completing the practical part of this course.

## 2 Requirements

For the practical realization of this course, there are software and hardware requirements that the student shall meet. In addition, there are skills that the student should already know to complete this course:

1. **C/C++.** The student shall be familiar with both C and C++ programming languages. The practical aspect of this course uses external libraries developed primarily in C++ and rarely in C. For this reason, it is highly recommended to improve your C++ programming skills. This course does not teach these languages and it is expected that the student compensates in their spare time for any missing knowledge gaps in the language. Failing to learn C and C++ independently will delay your work as most of the tasks involve C/C++ programming.
2. **Electronics.** The practical sessions use electronic development kits, microcontrollers, and sensors. The student should know how to read electronic schematics, and datasheets and to realize connections with a breadboard and electronic prototyping kits. You are responsible for the hardware you are given and you should take care that you can know how to operate any electronic hardware, or schematic before you make any electrical connections. Failing to operate correctly electronic hardware may damage it and require you to replace it.

### 2.1 Hardware

These laboratory sessions require the use of electronic hardware and sensors for their completion. The hardware will either be given for the completion of the specific task or will be given to the student in advance for its preparation at home and the learning process.

In case any hardware is given to the student, the student is responsible to keep it in the same condition it has received it. In case of damage or loss of material, the student will have to replace it. In addition, when the laboratory supervisor hands out any hardware the student will have to sign a confirmation letter that acknowledges the handout and responsibility of the lease.

### 2.2 Software

The laboratory computers can be used to develop software but it is required that the student install the following software on their private computers to practice at home and prepare properly for the laboratory session. The following software must be installed by the student:

1. Visual Studio Code <https://code.visualstudio.com/download>
2. PlatformIO IDE for VSCode <https://platformio.org/install/ide?install=vscode>
3. Node-RED <https://nodered.org/docs/getting-started/local>
4. Logic 2 <https://www.saleae.com/de/downloads/>
5. Git: <https://git-scm.com/downloads>

In addition, any other software or driver installation for the correct functioning of the hardware will be specified in the different tasks as part of the requirements before the student can begin their task.

### 2.2.1 Visual Studio Code

Visual Studio Code is the main IDE that will be used for any tasks that involve programming. For this reason, the student should learn how to use visual studio, its main features, and how to install extensions. A getting started is provided by Microsoft to learn how to use Visual Studio Code and is highly suggested to be checked out by the students:

<https://code.visualstudio.com/docs/introvideos/basics>

Suggested extensions for software development for this Course:

- C/C++ extension: <https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools>

-Doxygen Document generator:

<https://marketplace.visualstudio.com/items?itemName=cschlosser.doxdocgen>

### 2.2.2 PlatformIO

PlatformIO is a multi-architecture framework for the embedded development of software. For this course, we will be using the [ESP32 Microcontroller](#). The student should learn how to use PlatformIO, more information about this software is provided below:

#### *Getting started*

- <https://docs.platformio.org/en/stable/what-is-platformio.html>  
<https://docs.platformio.org/en/stable/core/quickstart.html>

#### *Tutorials and Examples*

- <https://docs.platformio.org/en/stable/tutorials/index.html>

Project files (i.e. [platformio.ini](#)) should contain the following minimum information for the ESP32 development

```
[env:az-delivery-devkit-v4]
```

```
platform = espressif32
```

```
board = az-delivery-devkit-v4
```

```
framework = Arduino
```

Note: The “board” configuration for this course is “[az-delivery-devkit-v4](#)”, more information about this board will be given in the practical course. In addition, some of the last tasks will require the “[esp32cam](#)” board configuration.

### 2.2.3 Node Red

Node red is a flow-based development tool for visual programming. In this course, node-red is used to develop graphical user interfaces to show data from the practical courses and to connect sensor data and hardware. The student should be familiar with node-red and consult online resources for learning how to use it.

### *Getting started*

<https://nodered.org/docs/getting-started/local>

### *Your first Flow*

<https://nodered.org/docs/tutorials/first-flow>

### *Full documentation*

<https://nodered.org/docs/>

## **2.2.4 Logic 2**

The Logic 2 software allows you to connect a logic analyzer to your computer to analyze. The use of this software will be explained more in detail in the practical course. Some more information about this software can be found here: <https://www.saleae.com/downloads/>

## **2.2.5 Git**

Git is a software version control developed originally by Linus Torvalds for the Linux Kernel. Nowadays Git is used widely to administrate software projects. It is highly recommended that the software that you write for this course be administrated with git and uploaded to the FH Aachen Gitlab server: <https://git.fh-aachen.de>

## 3 Task 1 Modbus RTU

### 3.1 Learning objectives

- Modbus communication protocol
- Practical experience with Modbus RTU
- Modbus data manipulation
- Data visualization with Node-RED

### 3.2 Introduction

Communication protocols are an integral part of any mechatronics system. They allow the exchange of data between different modules or components by using a common language that defines how data is transported and interpreted unambiguously. In this Lab session, we will be working with Modbus, one of the most known and oldest communication protocols in the automation industry.

The student must research more about the Modbus protocol to complete this session. In addition, some of the tasks will involve analyzing the Modbus protocol and identifying the data that is exchanged.

### 3.3 Self-learning Material

- Modbus specification
  - [https://www.modbus.org/docs/Modbus\\_Application\\_Protocol\\_V1\\_1b3.pdf](https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf)
- How to use the Logic analyzer
  - <https://articles.saleae.com/logic-analyzers/how-to-use-a-logic-analyzer>
  - <https://support.saleae.com/user-guide/using-logic>
  - [https://www.youtube.com/watch?v=XhWKOj\\_p9k](https://www.youtube.com/watch?v=XhWKOj_p9k)
- Search Engine Keywords
  - Modbus RTU
  - Modbus RTU Frames
  - Modbus protocol
  - Modbus function codes

### 3.4 Preparation

For this lab session, the following topics should be learned beforehand by the student:

- **RS485 serial communication**
  - <https://www2.htw-dresden.de/~huhle/ArtScienceRS485.pdf>
  - <https://advantech-bb.com/wp-content/uploads/2014/12/RS-422-RS-485-eBook.pdf>
  - [http://www.cromptonusa.com/rs485\\_guide.pdf](http://www.cromptonusa.com/rs485_guide.pdf)
- **Basics of Node-RED**  
Modbus RTU for Node-RED  
<https://stevesnoderedguide.com/node-red-modbus>



Working with messages

<https://nodered.org/docs/user-guide/messages>

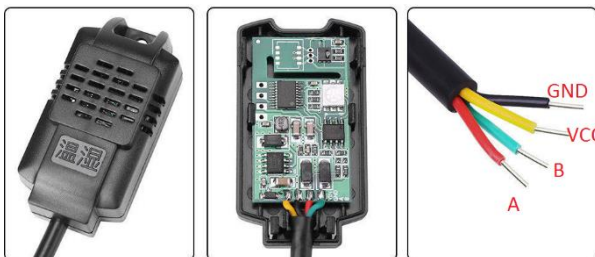
### 3.5 Hardware Requirements

- Modbus Temperature/Humidity sensor
- USB-RS485 converter
  - Driver Download  
<https://fh-aachen.sciebo.de/s/VOe6rZarpkBxydT>
- 2 Banana cables for the power supply of the Modbus sensor
- 8 MHZ 24 Channel Logic analyzer
  - Software and driver
    - <https://www.saleae.com/de/downloads/>

### 3.6 Modbus Temperature/Humidity Sensor

There are different versions of this sensor. They differ in the cabling coloring, pinout, and Modbus registers. Below you will find the pinout for the different versions and a picture of how to differentiate them.

#### Modbus Sensor version A



Color cable	Description
Red	A
Green	B
Yellow	VCC (5 to 36v)
Black	GND

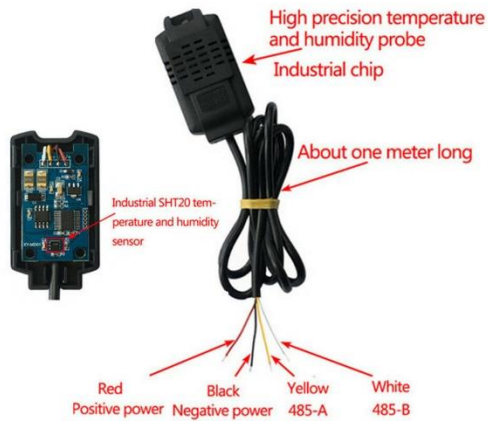
**Slave node address: 1**

**Serial speed: 9600 bits/s**

Table 1 Sensor A Modbus Function Codes

Modbus Function	Register Address	Variable	Size	Decoding
Read Input Register FC04	0x00	Temperature (°C)	2 Bytes	The temperature and Humidity values are given by a factor of 10. Raw Temp: 239 (decimal), Real Temp: 23.9°C Raw Humidity: 983 Real Humidity: 98.3%
	0x01	Relative Humidity (%)	2 Bytes	

## Modbus Sensor version B



Color cable	Description
Red	VCC (5 to 36v)
Black	GND
Yellow	A
White	B

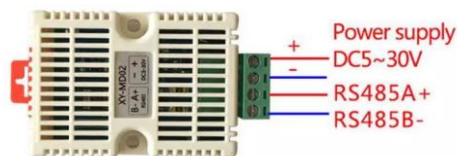
**Slave node address: 1**

**Serial speed: 9600 bits/s**

Table 2 Sensor B Modbus Function Codes

Modbus Function	Register Address	Variable	Size	Decoding
Read Input Register FC04	0x01	Temperature (°C)	2 Bytes	The temperature and Humidity values are given by a factor of 10. Raw Temp: 239 (decimal), Real Temp: 23.9°C Raw Humidity: 983 Real Humidity: 98.3%
	0x02	Relative Humidity (%)	2 Bytes	

## Modbus Sensor version C



**Slave node address: 1**

**Serial speed: 9600 bits/s**

Modbus Function	Register Address	Variable	Size	Decoding
Read Input Register FC04	0x01	Temperature (°C)	2 Bytes	Temperature value = 0x131, converted to a decimal 305, the actual temperature value = 305 / 10 = 30.5 °

				Note: the temperature is a signed hexadecimal number, Temperature value = 0xFF33, converted to a decimal place -205, the actual temperature = -20.5 °;
	0x02	Relative Humidity (%)	2 Bytes	humidity value = 0x222 converted to a decimal number 546, actual humidity value = 546/10 = 54.6%;

### 3.6.1 Hardware connections

For this task, the Modbus temperature sensor will be connected to a computer to analyze the Modbus protocol. To retrieve and send Modbus commands to the sensors (Function codes), the computer requires the USB-RS485 converter. The connections between the USB-RS485 converter and the sensor are as follows

Modbus Sensor	USB-RS485
A	A
B	B

As can be seen from the previous table, the connections are straightforward. Please check the bottom side of the USB adapter to know the RS485 pinout. The ground is not required to be connected as the communication works with a differential signal.

In addition to the communication signals of the RS485 physical layer, the sensor needs a power supply between 9-36 Volts. You can use the integrated 24V power supply from the desk where you are working at the laboratory to connect the power supply of the sensor. Please check which sensor version you have (A or B) as the cable color code is different.

### 3.7 Testing the sensor

The first activity of this task consists of testing that your sensor works and you have connected it correctly. The computer will act as a Modbus Master and the sensor as a Modbus slave. To communicate with the sensor, download and install the software QModMaster from the following link:

<https://sourceforge.net/projects/qmodmaster/>

Be sure to download a stable version and not a BETA version. A recommended version is v0.5.2.3

<https://sourceforge.net/projects/qmodmaster/files/qModMaster-Win32-exe-0.5.2-3.zip/download>

1. Change the communication settings for the Modbus connection via the menu bar Options>Modbus RTU.

To figure out the COM port used in windows you can open your device manager as follows:

- Windows Key + R
- Type "devmgmt.msc" and hit enter
- Search and Check the entry "Ports (COM & LPT)
- The device should be listed as USB-SERIAL CH340

2. Set the Modbus Mode in the main window to RTU

3. Specify the correct Modbus Options according to section 3.6 of this manual.
4. Communicate with the sensor using Commands>> Connect and then Commands>>Read/Write. For this step, you need to check the available Modbus function codes described in section 3.6.

It is important to mention that sensor type **A** might not work correctly with QModMaster. If you get the error "Read Data Failed" try several times until you can get a correct response.

### 3.8 Analyzing the Modbus protocol

After verifying that you can communicate with the sensor you should now analyze the communication to understand how Modbus works. You need to use a logic analyzer and the software Logic 2. External links and information on how to use this software are provided in the Self-Learning section.

To analyze the data, you first need to capture the communication. Connect the logic analyzer as follows:

Logic Analyzer	USB-RS485
CH0	A
GND	GND

Open again QModMaster and read the sensor. Before you read the sensor data, you should start a capture with the Logic software. To analyze the data by creating a new Async Serial analyzer. The analyzer option is located on the right side of the user interface.

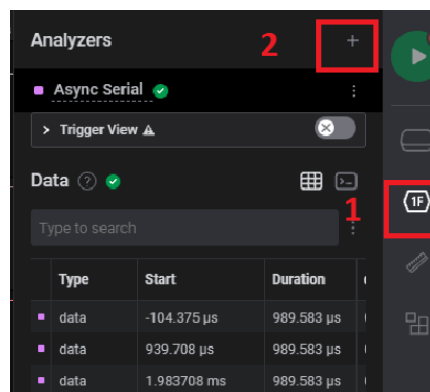


Figure 1 Visual hint on how to add an analyzer

The Async Serial Analyzer should have the following settings

- **Input: Channel:** Channel 0
- **Bit rate:** 9600
- **Bits per frame:** 8
- **Stop bits:** 1
- **Parity bit:** No parity
- **Significant bit:** LSB
- **Signal Inversion:** Non-Inverted
- **Mode:** Normal

**Note:** If you get a Timeout Error, you should reduce the sampling rate from the logic analyzer.

After setting correctly the analyzer you should have a signal with the decoded bytes. These bytes represent the Modbus data frame. Analyze the protocol and complete the following data in hexadecimal format.

*Modbus Master request:*

- **Slave Address:**
- **Function Code:**
- **Start Address:**
- **Number of Registers:**
- **CRC**

*Modbus sensor response:*

- **Slave Address:**
- **Function Code:**
- **Number of Bytes:**
- **Data:**
- **CRC**

Write down the CONVERTED temperature and humidity values you got after reading the values with the logic analyzer:

- **Temperature (°C):**
- **Humidity (%):**

As you have noticed, Modbus requests are based on different function codes. After completing the previous exercise, you should now have a better understanding of the Modbus protocol. Please answer the following:

1. What would be the Modbus transaction to turn on the 2<sup>nd</sup> Coil of a Modbus Slave with address 10 (decimal)? Please specify as well the meaning of each byte of the transaction. The CRC frame can be omitted.

**Modbus TX (Master):**

**Modbus RX (Slave):**

2. The last field in a Modbus RTU frame consists of the cyclic redundancy check frame (CRC).

A) What is the importance of adding a CRC to a Modbus RTU transaction?

B) Why does Modbus TCP not include a CRC frame?



## 4 Task 2 HTTP Requests

### 4.1 Learning objectives

- Analyze Ethernet communication
- HTTP Request methods

### 4.2 Self-learning Material

- HTTP Requests
  - <http://www.steves-internet-guide.com/http-basics/>
  - <https://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340>
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- [HTTP specification](#)
- Search Engine Keywords
  - HTTP Web Server
  - HTTP REST API
  - HTTP specification
  - HTTP Message
  - HTTP Ethernet Frame
- [Wireshark User guide](#)
- PlatformIO
  - <https://docs.platformio.org/en/stable/core/quickstart.html>
  - [https://docs.platformio.org/en/stable/tutorials/esp8266/arduino\\_debugging\\_unit\\_testing.html](https://docs.platformio.org/en/stable/tutorials/esp8266/arduino_debugging_unit_testing.html)
  - <https://docs.platformio.org/en/latest/platforms/esp8266.html>
- ESP32
  - <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html>
  - <https://docs.espressif.com/projects/arduino-esp32/en/latest/>
  - Development Kit Pinout
    - <https://fh-aachen.sciebo.de/s/oxYt5JTQHP95mdX>

### 4.3 Hardware Requirements

- ESP32 Microcontroller
- Computer with a wireless local area network card (WiFi)

#### *Optional*

- ESP32 Debugger. ESP-PROG debuggers are available at the lab to debug easily your code. Ask the lab supervisor in case you want to use one. Instructions on how to install it and use it are provided here: <https://www.hackster.io/brian-lough/use-the-platformio-debugger-on-the-esp32-using-an-esp-prog-f633b6>

### 4.4 ESP32: PlatformIO configuration

In this task, you will be using an ESP32 microcontroller. The microcontroller will be programmed with Visual studio code and the PlatformIO extension. You should verify that you have installed [visual studio code](#) and the PlatformIO [extension](#) for visual studio code.

#### 4.4.1 Create a new Project

Verify that you can communicate with the ESP32 by creating a new sample program in PlatformIO and uploading it. To create a new program, you can take as a reference Figure 2

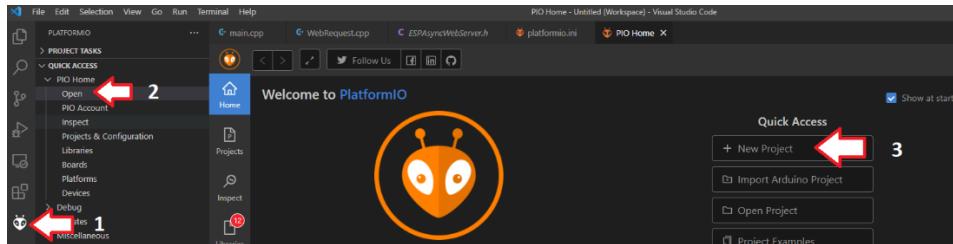


Figure 2 Sequential steps to create a new project

Once you click New Project, you should enter the following settings:

- *Board: AZ-Delivery ESP-32 Dev Kit C V4*
- *Framework: Arduino*

The *Name* can be changed as you wish and will help identify the project name and create a respective folder. The Checkbox *Location* allows you to select where you want to save your project. If you hover over the question mark it will show you the default location.

If this is the first time you create an ESP32 project, it can take a while to install and download the necessary tools. After creating the project you should have a folder structure that looks as in Figure 3. In this case, the name of the project is “MyExample”.

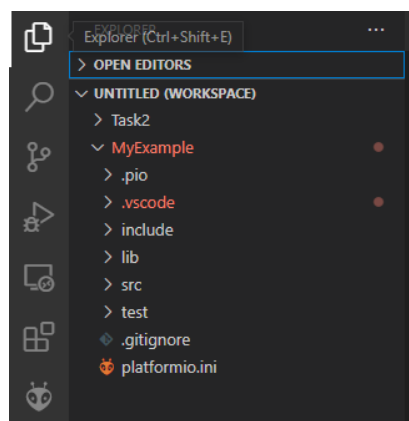


Figure 3 PlatformIO Project Structure

The configuration of the project is saved in the file platformio.ini. More information about this file can be consulted in the documentation from PlatformIO:

<https://docs.platformio.org/en/latest/projectconf/index.html>

#### 4.4.2 Build a Project

To build your project you can use the Command Palette (Ctrl + Shift + P). The Command palette allows you to run any command from visual studio code. This includes extensions such as PlatformIO. In general, to run a command for an extension you should type “NameOfExtension: Command”. For this



case, the command to build a PlatformIO Project is “PlatformIO: Build”. You should get an output similar to as in Figure 4.

```
no dependencies
Building in release mode
Compiling .pio\build\az-delivery-devkit-v4\src\main.cpp.o
Linking .pio\build\az-delivery-devkit-v4\firmware.elf
Retrieving maximum program size .pio\build\az-delivery-devkit-v4\firmware.elf
Checking size .pio\build\az-delivery-devkit-v4\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [ ] 2.5% (used 13224 bytes from 532480 bytes)
Flash: [==] 15.4% (used 201216 bytes from 1310720 bytes)
Building .pio\build\az-delivery-devkit-v4\firmware.bin
esptool.py v3.1
Merged 1 ELF section
===== [SUCCESS] Took 9.31 seconds =====
```

Figure 4 PlatformIO Build output

#### 4.4.3 Upload a program

Similar to building a program you can type the command “PlatformIO: Upload”. This should automatically detect the COM port where your ESP32 is connected. Your output should be similar to as in Figure 5.

```
Compressed 3072 bytes to 128...
Writing at 0x00000000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00000000 in 0.1 seconds (effective 341.4 kbit/s)
Hash of data verified.
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (effective 508.0 kbit/s)
Hash of data verified.
Compressed 208272 bytes to 107284...
Writing at 0x00100000... (14 %)
Writing at 0x001ed6e... (28 %)
Writing at 0x0024410... (42 %)
Writing at 0x002d54a... (57 %)
Writing at 0x0033d3a... (71 %)
Writing at 0x00399b4... (85 %)
Writing at 0x003f72e... (100 %)
Wrote 208272 bytes (107284 compressed) at 0x00100000 in 2.6 seconds (effective 634.7 kbit/s)
Hash of data verified.
```

Figure 5 Successful ESP32 upload

If PlatformIO does not detect the correct COM port you can edit your platformio.ini file and add the option “upload\_port” with the respective com port. Specific information on how to do this is found in the documentation of PlatformIO:

[https://docs.platformio.org/en/stable/projectconf/section\\_env\\_upload.html#upload-speed](https://docs.platformio.org/en/stable/projectconf/section_env_upload.html#upload-speed)

#### 4.4.4 Monitor a program

As the ESP32 does not include an integrated debugger, the easiest method to monitor your application is to print information through its serial port. Add the following lines to the function “setup” in your project file src/main.cpp

```
Serial.begin(115200);
Serial.println("Hello world");
```

This will print the phrase “Hello world” when the ESP32 is initialized. Build and Upload this program. As the baud rate for this example is 115200 you need to change the monitor speed for PlatformIO since by default it's 9600. Modify the platformio.ini file and use the option “monitor\_speed” to set the correct baud rate. More information about how to exactly do this can be found in the documentation from PlatformIO:

[https://docs.platformio.org/en/stable/projectconf/section\\_env\\_monitor.html#monitor-speed](https://docs.platformio.org/en/stable/projectconf/section_env_monitor.html#monitor-speed)

After saving the modification, open the Command Palette and type the command “PlatformIO: Serial Monitor”. After hitting enter, PlatformIO will open the com port for the ESP32 and show the current output. If the output is blank you can verify that the ESP32 prints “Hello World” by pressing the button “RST” located on the front side of the ESP32.

```
--- Available filters and text transformations: colorize, debug, default
--- More details at https://bit.ly/pio-monitor-filters
--- Miniterm on COM3 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:10124
load:0x40080400,len:5828
entry 0x400806a8
hello world
```

Figure 6 Serial Monitor output example

#### 4.4.5 Further information

PlatformIO includes more commands than previously shown here. If you want to learn more in detail about PlatformIO and the development of software applications with the ESP32 you should consult online resources and the official documentation. The next sections will involve the use of the ESP32 and will assume the student has independently learned the basics about the ESP32 and its usage with PlatformIO.

#### 4.4.6 POST Requests

In this activity, you will develop an ESP32 program that processes HTTP POST requests from a computer. You will need the following ESP32 libraries:

- <https://github.com/me-no-dev/ESPAsyncWebServer>
- <https://github.com/me-no-dev/AsyncTCP>

Both of these libraries can be downloaded from Github and installed in a new PlatformIO Project. To install the libraries in your project, download the libraries, unzip the contents, and copy the folder inside of them to the /lib directory in your PlatformIO Project. Your project structure should look like in Figure 7. More information about the manual installation of libraries can be found here:

<https://community.platformio.org/t/trying-to-understand-how-to-install-custom-local-library/3031>

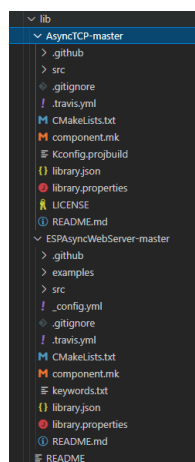


Figure 7 PlatformIO Library installation

To start the project, you can take as a reference the example from:

<https://techtutorialsx.com/2018/10/12/esp32-http-web-server-handling-body-data/>

In this example, they create a Web Server that accepts POST requests with a message body. The function is defined around line 20 `server.on("/post...` reads the message body of the HTTP request and prints it out.

This [example](#) connects to a Wifi Access Point (AP) (lines 12-19). As this requires a router or a computer to act as an AP, using the ESP32 as an AP is simpler. This allows you to connect with any device to the ESP32. You can follow the example from [here](#), and remove the implementation where they connect to an external AP with `WiFi.begin`, `WiFi.status`, and `WiFi.localIP`. In addition, change the SSID and password to a unique name since more than one of your classmates can run the same example and it will be difficult for you to know which Wi-Fi AP is yours.

Once you have done the proper changes build and upload your program. To test that your example works, as mentioned in the reference example, you can use the software, Postman. The software can be downloaded from here:

<https://www.postman.com/downloads/>

Open the software, and click "Create a request" (see Figure 8). Fill in the settings for the request (see Figure 9). The settings should match the IP of the ESP32, which should have been figured out when creating the ESP32 Access Point and printing the IP with the function `WiFi.softAPIP()`. As seen in Figure 9, you are sending a raw string and the ESP32 should reply by printing the HTTP body. For further details check the video from the referenced example:

[https://youtu.be/\\_fLPRvQS6W4](https://youtu.be/_fLPRvQS6W4)

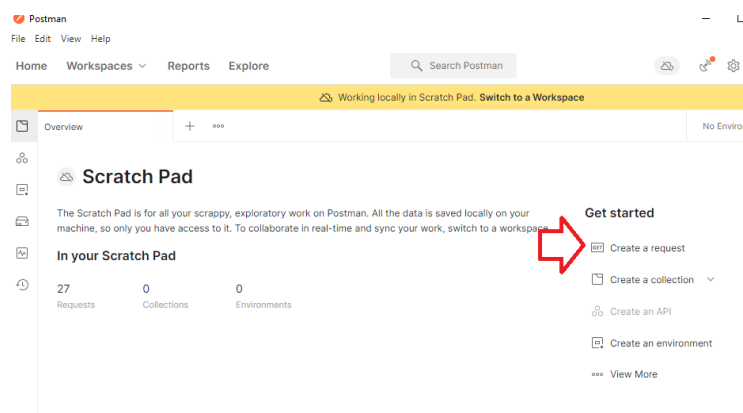


Figure 8 Create a new HTTP request with Postman

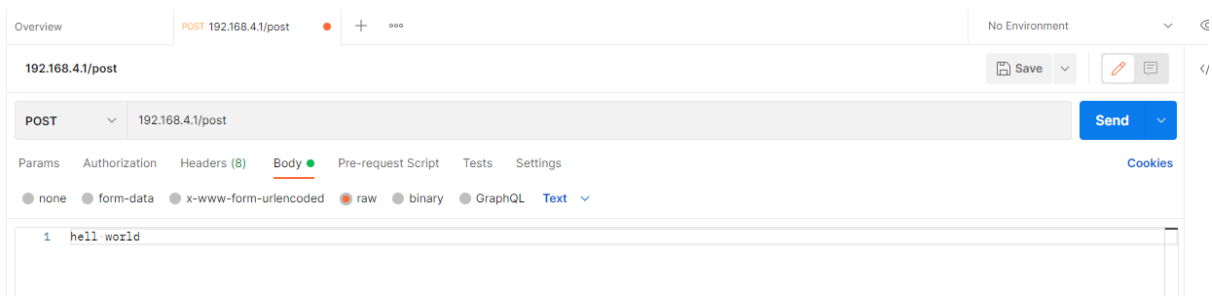


Figure 9 Post request settings

By following the example and modifying the connection with an AP from the ESP32, you should now be familiar with how to process POST requests. You can now continue with the next activity.

#### 4.4.7 GET Requests

Contrary to POST requests, you can retrieve information by using a GET request. Follow the tutorial from:

➤ <https://techtutorialsx.com/2017/12/01/esp32-arduino-asynchronous-http-webserver/>

Similar to the last activity, modify the example to make the ESP32 an AP instead of connecting to an external AP

In this reference example, you are using a Webserver but now instead of having a post request, you have a GET request as seen in the second argument of the function `server.on()`. This function is now using `HTTP_GET`. More information about all the available requests you can use is found [here](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basic_of_HTTP/MIME_types).

The difference with the POST request is that now you need to send information back. In this case, they are sending a string in "text/plain" format. You can get more information about the different data types you can send back in:

[https://developer.mozilla.org/en-US/docs/Web/HTTP/Basic\\_of\\_HTTP/MIME\\_types](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basic_of_HTTP/MIME_types)

To verify that your example works you can use the Postman software and do an HTTP GET request (see Figure 10).

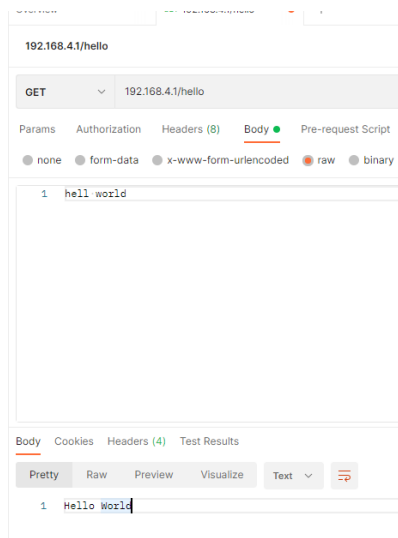


Figure 10 GET request with Postman

#### 4.5 HTTP Request Analysis

The next step is to analyze the HTTP requests that you have programmed so far. You will need to download and install Wireshark: <https://www.wireshark.org/download.html>

After opening Wireshark, select the network interface that is used to connect to the ESP32 (i.e., your Wi-Fi Adapter). If you have multiple W-iFi network interfaces, you will need to identify the correct one. For example, to list the network adapters you can run the command **ipconfig** (Windows) or **ip** (Linux).

Send HTTP requests to the ESP32 for both examples you made (POST, GET) to test the features you developed in the previous steps. This time with Wireshark you should analyze the HTTP messages sent and received between your computer and the ESP32. To filter the messages, you can type "http" in the [filter bar](#) from Wireshark:

1. How Can you identify in Wireshark who is the recipient and sender of the HTTP requests?

2. Analyze the HTTP POST Request and write down the following

HTTP POST request from Computer to ESP32

- **Request Method:**
- **Request URI:**
- **Request Version:**
- **Content-Type**
- **User-Agent:**
- **Host:**
- **Content-Length:**
- **Line-based text data:**

HTTP POST reply from ESP32 to computer

- **Response Version:**
- **Status Code:**
- **Status Code Description:**
- **Content Length:**

3. Analyze the HTTP GET Request and write down the following

HTTP GET request from Computer to ESP32

- **Request Method:**
- **Request URI:**
- **Request Version:**
- **User-Agent:**
- **Host:**

HTTP GET reply from ESP32 to computer

- **Response Version:**
- **Status Code:**
- **Status Code Description:**
- **Content-Length:**
- **Content-Type**
- **Line-based text data:**

4. What is the purpose of the HTTP Status codes?

5. Mention the meaning of the following HTTP status codes.

500

404

200

501

400

401

502



## 5 Task 3 Project Modbus TCP

### 5.1 Learning objectives

- Modbus TCP
- Web server integration
- Modbus RTU
- Data visualization with Node-RED

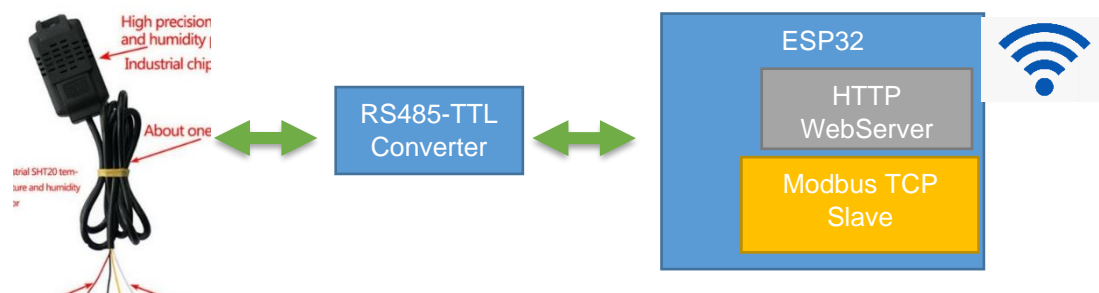
### 5.2 Hardware Requirements

- Modbus Temperature/Humidity sensor
- RS485- 3.3V TTL converter
- ESP32 Microcontroller
- Computer with a wireless local area network card (WiFi)
- Logic Analyzer \*

\* These components are optional and can be used to debug project

### 5.3 Introduction

This task will consist of a small project with the knowledge you have gained from Task 1 and Task 2. The objective of this task is to retrieve the Modbus temperature and humidity sensor data with an ESP32 and make the information available with two interfaces: firstly, an HTTP web server, and secondly a Modbus TCP slave. In the next sections, a description of the requirements are mentioned, and further information about how you can implement this project.



### 5.4 Requirements

#### 5.4.1 Modbus RTU

Temperature and humidity data shall be retrieved from the Modbus sensor you used in task 1. This data should be read by an ESP32 and made available to transmit over a Modbus TCP slave and HTTP web server interface

### 5.4.2 Access point

Both of the interfaces (Webserver, Modbus TCP slave) shall be accessible from an Access Point created from the ESP32. The SSID should be your matriculation number and password Task3\_Modbus

### 5.4.3 Webserver

The data retrieved from the Modbus RTU sensor shall be available through an HTTP web server. The requirements of the web server are the following:

- Retrieval of temperature through an HTTP GET request. URI for this request should be **/temperature**. The http response should include the temperature in a text representation in decimal format (e.g., 23.4 °C).
- Retrieval of the Humidity through an HTTP GET request. URI for this request should be **/humidity**. The http response should include the humidity in a text representation in decimal format (e.g., 100.0 %).

### 5.4.4 Modbus TCP

The second interface that is needed for this task is a Modbus TCP slave. In this interface, you will also use the Modbus Temperature and humidity sensor via the RS485 interface and transmit it over Modbus TCP. The Modbus TCP slave must follow the requirements from section 5.4 of this document. The Slave ID should be 0x01 (1 decimal).

Table 3 Modbus TCP Slave description

Modbus Function	Register Address	Variable	Size	Decoding
Read Input Register FC04	0x01	Temperature (°C)	2 Bytes	Raw value divided over 10
	0x02	Relative Humidity (%)	2 Bytes	

### 5.4.5 Front End

After verifying that both the Modbus TCP Slave and Web Server work, you should now visualize the data in your computer by both methods. You should accomplish this with Node-RED.

The objectives of this part are the following:

- Reading the Temperature and humidity via the “**Modbus-Read**” node and displaying it in a dashboard Tab called **Modbus TCP Slave**. The data should be displayed with a Gauge and a chart for the temperature and humidity.  
- You have already done something similar in Task 1. The only difference is that this time you are using Modbus TCP instead of Modbus RTU.
- Reading the Temperature and humidity data via the Web Server. For this part, you can refer to this [tutorial](#) to know how to create an HTTP request in Node-RED.
- The data you get from the HTTP GET methods for temperature and humidity should be



displayed as well in the dashboard. Create another Tab called **Web Server** to display both temperature and humidity with a Gauge and Chart.

## 5.5 Implementation

In this section, you will find useful information to develop the implementation of this project.

### 5.5.1 Modbus RTU

To retrieve the data from the Modbus sensor you will need to connect an RS485 to TTL converter between the sensor and the ESP32. The hardware connections are described in Table 4.

Use the following Modbus RTU library to communicate between the sensor and the ESP32:

[https://github.com/vChavezB/ModbusMaster\\_FH](https://github.com/vChavezB/ModbusMaster_FH)

Since this library is not in the [Arduino Library Registry](#), you should manually download the library and install it to your PlatformIO Project in the /lib directory.

Table 4 Modbus RTU Connections for Task 3

RS485 - TTL 3.3V	ESP32	Modbus Sensor
A		A
B		B
TX	GPIO 32	
RX	GPIO 35	
GND	GND	
VCC	3.3V	

You can use the example "RS485\_HalfDuplex" from the library you just installed (located in examples/RS485\_Halfduplex) as a reference to start your programming task. This does not mean it will work out of the box, as you will need to understand the code and modify it accordingly example.

The RS485-TTL module you have does not require to use of the RE and DE pins. This means that the code related to *preTransmission*, *postTransmission* are not required.

In addition, you will need to use a second Serial object since the object *Serial* is used for sending data to the computer (e.g., uploading program, debugging). Instead, use *Serial2* as an argument for the method *ModbusMaster::begin()*. Change the default pins of *Serial* with TX as GPIO17 and RX assigned to GPIO16. Information about how to do this is found [here](#).

Remember to select the correct Modbus settings (serial speed, slave address, function code, etc.) to communicate correctly with the sensor in your code. In case you still do not get the data from the sensor correctly you can use a logic analyzer to debug your application. For example, you can sample the TX, RX lines of the Serial2 peripheral or the RS485 line to verify that the sensor is getting the Modbus master request.

Note: If you have a sensor Type A and get an error code while using the function *ModbusMaster::readInputRegister*, reduce the delay between readings from 1 second to 0.2 seconds. The reason is that this sensor sends data over ASCII each second and can interfere with the Modbus protocol.

### 5.5.2 HTTP Web Server

There is no further information for the implementation of the HTTP web server. You can use the knowledge you gain from Task 2 to implement this task. Remember that both temperature and humidity should be represented as a string in decimal format.

You can use the Arduino data type [String](#) or the native C++ [std::string](#) to convert a decimal (float) value to a string representation.

### 5.5.3 Modbus TCP

With this interface, you will retrieve the Modbus Temperature and humidity sensor via the RS485 interface and transmit it over Modbus TCP. You can use the following Modbus library to add the Modbus TCP slave interface to the ESP32

<https://github.com/eModbus/eModbus>

In particular, you can use as a starting point the example "TCPServerAsync". As a small hint, the callback function in the example "TCPServerAsync" gets the requested register address in [line 32](#), and in the [next line](#), the number of word registers to read starts from that specific register address. To add the Modbus TCP data you can simply use the method `ModbusMessage::add`, as seen for example in [line 47](#).

After you have successfully adapted the example, you can test that the Modbus TCP works by using the software QModMaster (<https://sourceforge.net/projects/qmodmaster/>) and connecting your Laptop to the ESP32 access point.


#### Verification with QModMaster

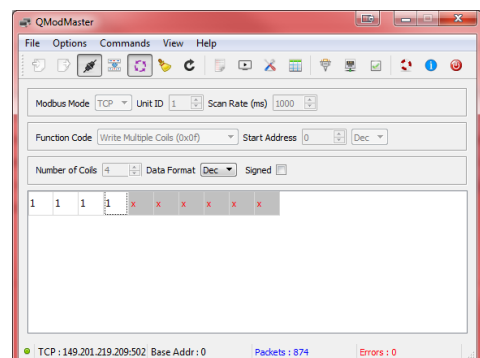
You can use QModMaster as well to verify if the ModbusMasterTCP Slave works.

1. Set the local IP of the Modbus slave you are going to connect (i.e., the ESP32) via the menu bar **Options>>Modbus TCP...**

**Slave IP:** ESP32 IP Address

**Port:** 502

2. Check that the *Modbus Mode* is set to TCP.
3. Click the  connect button or via the Menu bar **Commands>>Connect**



## 6 Task 4 CAN Bus

### 6.1 Learning objectives

- CAN Bus protocol
- Analyzing CAN Bus frames
- Decoding and Encoding of CAN Bus data

### 6.2 Self-learning Material

- Search Engine Keywords
  - Comma separated values
  - C++ String manipulation
  - C++ comma-separated string
  - Node-RED Serial port
  - Node-RED string manipulation
  - CAN Bus protocol
  - CAN bus frames

### 6.3 Hardware Requirements

- MCP2515 Development Board
- ESP32 Microcontroller
- [USB CAN Analyzer](#)
  - Download drivers and software  
<https://fh-aachen.sciebo.de/s/dPE6ZZpRCoXZRww>  
Notes:
    - Driver path: **Driver\driver for USBCAN(CHS40)**
    - Software path: **Program\USB-CAN(V8.00).exe**
- Logic analyzer

### 6.4 Introduction

In this task, you will work with the CAN Bus protocol. This protocol is commonly used in the automotive industry and communication networks with multiple devices talking over the same data line. The CAN Bus consists of a 2-wire differential signal (CAN High/CAN Low), where multiple devices can be connected and processed typically by an electrical control unit (ECU). Each device connected to this network has an assigned priority depending on its device id and allows multiple devices to exchange data in a coordinated matter. This means that the smallest CAN ID receives the highest priority in the data line.

In the following subsections of this task, you will be working with the CAN bus protocol, processing CAN data frames, analyzing them, and visualizing them.

## 6.5 Sending CAN Frames

In this part of the task, you will send CAN frames from your ESP32 to the USB CAN analyzer. With your computer, you will then verify that the frames are received correctly.

### 6.5.1 Hardware connections

The MCP2515 module communicates over a Serial Peripheral Interface (SPI) and notifies of any events through a digital output pin called INT, which stands for interrupt.

Firstly, you should connect the ESP32 to the MCP2515 development board as follows:

*Table 5 HW Connections between the ESP32 and the MCP2515 module*

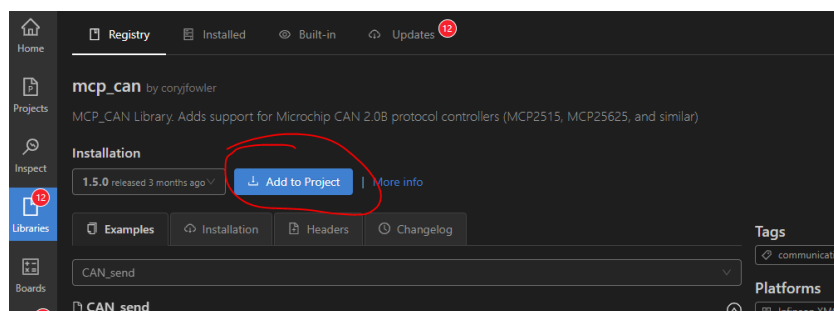
MCP2515	ESP32
INT	GPIO15
SCK	GPIO18
MOSI	GPIO23
MISO	GPIO19
CS	GPIO5
GND	GND
VCC	5V

### 6.5.2 Software

After realizing the hardware connections, you will need to test the USB CAN analyzer and send CAN Frames from the ESP32. Firstly, you need to install the following Library:

[https://github.com/coryjowler/MCP\\_CAN\\_lib](https://github.com/coryjowler/MCP_CAN_lib)

The library can be installed [manually](#) or via the library manager from PlatformIO. To install it via PlatformIO, you can open the Library panel located on the left part of the PlatformIO and search for the library and click “Add to Project” (see Figure 11).



*Figure 11 Installing library via PlatformIO*

You can use the `CAN_send` example as a starting point to send can frames. Change the Hardware CAN settings of this example as necessary. In specific you need to select the correct Chip select Pin (CS).

You will notice that the method `MCP_CAN::begin` has three arguments. The most relevant arguments for setting the MCP2515 module are “speedset” and “clockset”. The `speedset` values are defined in `mcp_can_dfs.h`. Choose the relevant one, depending on the speed you want to communicate over the CAN Bus. The second argument, `clockset`, refers to the actual clock signal that the MCP2515 uses. Available options are defined [here](#). You will have to visually inspect your MCP2515 module and check the value that is on top of the quartz crystal of your module.

### 6.5.3 CAN over Logic Analyzer

Firstly, verify that you can send CAN Frames and you get an output similar to this one through the serial output of the ESP32.

*MCP2515 Initialized Successfully!*

*Error Sending Message...*

This means that the MCP module is communicating with the ESP32 but as you haven't connected any other device over CAN you will notice the error message. However, you can check that the CAN bus is working by using a logic analyzer.

Connect your logic analyzer (not to be confuse with the USB CAN analyzer!) and capture the data over the CAN Low line. Connect the Ground of the Logic analyzer to the Ground of the ESP32. Create a new capture and analyze the CAN data. For this step, you need to add a CAN analyzer. Similar to the last task where you analyzed the Modbus data, you need to set the CAN analyzer in the Logic v2 software.

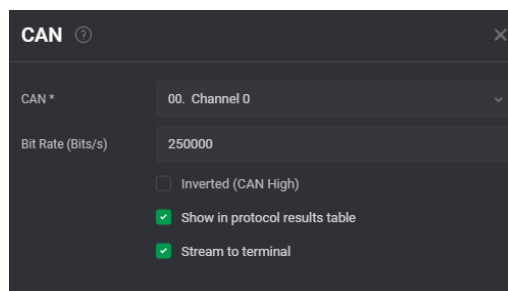


Figure 12 CAN settings for the Logic v2 Software

Write down the parsed data from the CAN\_Send program below in the hexadecimal format:

- **CAN Identifier:**
- **Data:**
- **CRC Value:**

### 6.5.4 USB CAN Analyzer

The USB CAN Analyzer provided for this task allows you to send and receive CAN data frames over USB. You need to install the drivers and software USB-CAN(v8.00) to use the analyzer (refer to section 6.3).

#### 6.5.4.1 Self-Test

Firstly, verify that your USB CAN Analyzer is working correctly. Do not Connect the CAN data lines (CANH,CANL) to your ESP32 yet.

1. Open the software **USB-CAN(V8.00)**.
2. Set the communication over the serial port correctly. Identify your COM port and select the correct one on the upper left side of the program. The baud rate should be set to 2000000, which is the default speed of the Analyzer.

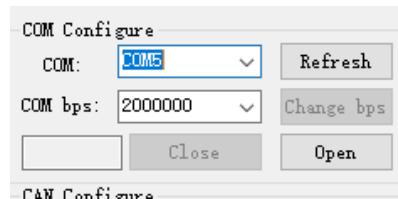


Figure 13 USB CAN Analyzer setup

3. Click "Open"
4. Set the options in the section "CAN Configure" as follows:  
 Mode: Loop back + silent mode  
 Type: Standard Frame  
 CAN bps: 250kbps
5. Click "Set and Start"
6. Click "Send a single Frame". You should receive the same frame you sent and have a similar output as in Figure 14.

Total:	5000	<input type="checkbox"/> Display receive only	<input type="checkbox"/> Statistics	Clear	Pause	Continue	Save	<input type="checkbox"/> Auto save	Exit
No	Direction	Time scale	Frame Type	Frame Format	Frame ID	Data Length	Data(Mouse Left-click Hex->Dec)		
0	Send	22:44:05:600	Data frame	Extended Frame	00000000	8	11	22	33 44 55 66 77 88
1	Receive	22:44:05:605	Data frame	Extended frame	00000000	8	11	22	33 44 55 66 77 88
2	Send	22:44:06:019	Data frame	Extended Frame	00000000	8	11	22	33 44 55 66 77 88
3	Receive	22:44:06:025	Data frame	Extended frame	00000000	8	11	22	33 44 55 66 77 88
4	Send	22:44:06:170	Data frame	Extended Frame	00000000	8	11	22	33 44 55 66 77 88
5	Receive	22:44:06:177	Data frame	Extended frame	00000000	8	11	22	33 44 55 66 77 88

Figure 14 USB CAN Analyzer Loop back mode

If you are not able to receive the same frame you sent over CAN with the USB CAN Analyzer, refer to file "**11. Restore the default configuration description.pdf**" which is part of the USB CAN Analyzer download provided in sciebo. Afterward, try again all the steps of this subsection.

#### 6.5.4.2 Reading CAN data over USB

Verify that you can receive the data from the ESP32 with the USB Analyzer. These connections are straightforward as previously mentioned, CAN consists of a single pair of wires (CAN High, CAN Low). Refer to Table 6 to connect both devices over CAN

Table 6 Hardware connections between the USB CAN analyzer and the MCP2515 module

MCP2515	USB CAN
CAN H	CAN H
CAN L	CAN L

**Important note:** The logic analyzer won't work properly when the USB CAN adapter is also connected. This means that you can't use the logic analyzer while using the USB CAN adapter.

Run again all the steps from the Testing section up until step 3. You will now need to change the "CAN Configure" settings. Select the correct *CAN Bps* according to your "*CAN\_Send*" ESP32 program and change the Mode to "*Normal Mode*". Click "Set and Start", you should now receive the data from the ESP32 over CAN. Once this has been verified you can continue to the next section.

## 6.6 Receiving CAN Frames

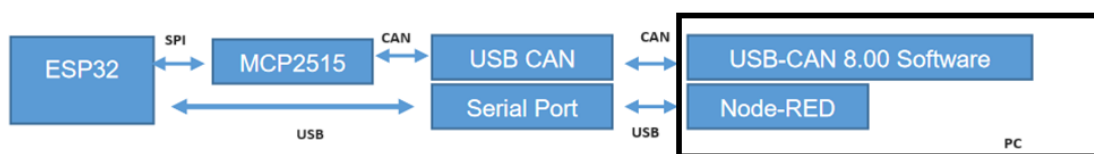
Similar to the last subsection, you will use the USB CAN analyzer and the ESP32. The Hardware connections remain the same but now the ESP32 will receive CAN Frames from the USB analyzer. You can use as a reference the example “*CAN\_receive*” from the Arduino CAN library mentioned in the last subsection. Modify it according to the CAN settings you will use (speed, clock, etc) and hardware pins (CS, INT). The example prints out the information of any CAN frame received and its CAN ID.

*Note:* If you cannot upload the program, try changing the INT pin as it can create a conflict while uploading.

Use again the **USB-CAN(V8.00)** software and follow the same procedure to connect to the analyzer. In this case, you will now send frames from your computer. You can use the “Send a Single Frame” button to send individual frames. Verify that the ESP32 can receive the frames to complete this section.

## 6.7 Node-RED

In this last part of the task, you will use the practical knowledge you have built from the previous two subsections. The ESP32 will receive CAN Frames from the computer via the **USB-CAN(V8.00)** software, process them and send them through its serial port. The data from the serial port will be sent to the computer, processed by Node-RED, and visualized in a dashboard. Moreover, you will send serial data over Node-RED which will be processed by the ESP32 and sent over CAN. The USB analyzer will be used both to send CAN frames to the ESP32 and verify the reception of frames sent over Node-RED.



### 6.7.1 Sending a CAN Messages to Node-RED

In this section, you will process CAN Frames from the USB CAN device and send them to Node-RED over the serial port. You can use the example *CAN\_Recieve* as a reference again. Each time you get CAN frames from the USB CAN into the ESP32 you should send them over the serial port. To ease this step the following format is defined for you to send data over the serial port:

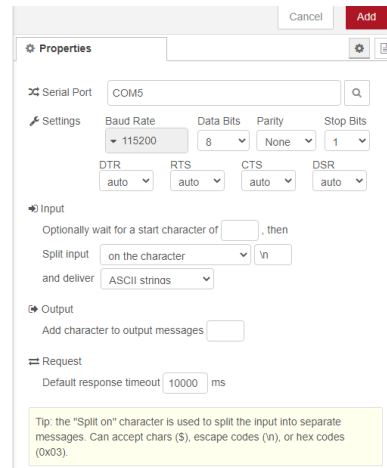
***CAN ID, Length, Data 1, Data 2, Data 3,Data 4, Data 5, Data 6, Data 7 \n***

The first string you send is the CAN ID, next to the number of received bytes and at the end the bytes received. Each element must be separated by a comma and terminated by a newline (\n). An example of this could be

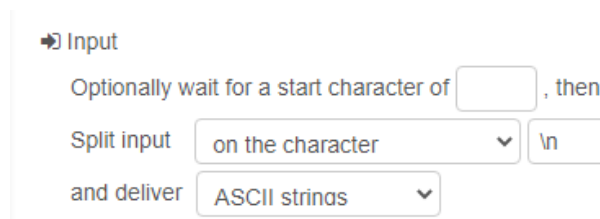
*930, 8, 0,1,2,3,4,5,6,7*

Where the CAN ID is 930, the length is 8 and the rest of the bytes are the received data over CAN.


To receive the data, you must install the node “node-serialport”: <https://flows.nodered.org/node/node-red-node-serialport>. To read data from the ESP32, you can use the node called “serial in”. In this node, you can configure which COM port is used for the ESP32 and its baud rate.



The default configuration will split the messages by using a new line. If it's not set, you can change it in the section "Input" as follows:



The node will now send as part of its message payload each new line sent over the ESP32. You can now use this information to create a dashboard to visualize the data. You can use the "split" node or "csv" node to separate the information. The following information must be displayed: CAN ID, length, and CAN bytes over the dashboard.

More information about how to use these nodes should be consulted online, as the nodes parse differently the information. The third option is to create a custom function to process the string with the function  node. After verifying that this works you can continue with the next section.



## 7 Task 5 µProject CAN Joystick

### 7.1 Learning objectives

- CAN J1939

### 7.2 Self-learning Material

- Search Engine Keywords
  - J1939 Protocol
  - How does J1939 work
  - Introduction to the SAE J1939
- J1939 Protocol
  - <https://copperhilltech.com/blog/sae-j1939-parameter-group-number-pgn-range/>
  - [Book] [A Comprehensive Guide to J1939](#)
  - [PDU Format](#)

### 7.3 Hardware Requirements

- MCP2515 Development board
- ESP32 Microcontroller
- USB CAN Analyzer \*
- Logic analyzer \*
- [CR1301 ifm joystick](#)

\* These components are optional and are intended for debugging your project

### 7.4 Introduction

The main objective of this project is to control the industrial joystick device CR1301 from the company IFM. This joystick is controlled over the CAN Bus and uses the SAE J1939 protocol. The joystick has 6 digital buttons, a directional pad, a digital rotary encoder, and a rotary button.

### 7.5 Objectives

- Track the position of the digital rotary encoder.
- Detect the direction of rotation.
- Use the encoder position to change the color of the circular ring LEDs.



Figure 15 Possible colors for the Joystick selection

## 7.6 Implementation details

### 7.6.1 CR1301 Joystick

#### 7.6.1.1 Pinout

Table 7 CR1301 Wiring color code

Wire	Pin	Description
Gray	1	CAN L
Black	2	CAN H
Brown	3	GND
White	4	9..32 VDC

#### 7.6.1.2 Simulator

Depending on availability, the CR1301 joysticks will need to be shared for more than one group. In case of this, we have prepared a [Simulator](#) that can be used for programming and testing your application. This simulator can be used with your USB Can Analyzer. The USB Can Analyzer will simulate the joystick and you can connect the MCP2515 through CAN. The only difference is that instead of sending the CAN messages to the real device, the simulator will receive them and display them on your computer.

The simulator should be used only for testing. Once you are sure your implementation works you can try using the real device. The purpose of this is to allow the whole group to circumvent the limited amount of CR1301 devices that are in stock.

For further information read the [documentation](#) of the simulator.

### 7.6.2 SAE J1939

The SAE J1939 standard is a protocol that uses the CAN bus to transmit data. The CR1301 joystick implements this protocol and is important to understand how it works. The student is allowed to use

any J1939 library for the project implementation, but it is recommended that they use the following [library](#), which has been tested and verified.



## 8 Task 6 IO-Link MQTT

### 8.1 Self-learning Material

- IO-Link Concepts
  - System description  
[https://io-link.com/share/Downloads/At-a-glance/IO-Link\\_System\\_Description\\_eng\\_2018.pdf](https://io-link.com/share/Downloads/At-a-glance/IO-Link_System_Description_eng_2018.pdf)
  - General introduction  
<https://www.ifm.com/us/en/shared/technologies/io-link/io-link-introduction>

### 8.2 Hardware Requirements

- AL1300 IO-Link Master
- 24V M12 Power supply
- M12 Ethernet Cable
- M12 IO-Link Device Cable
- IO-Link Device (Sensor)

### 8.3 Software Requirements

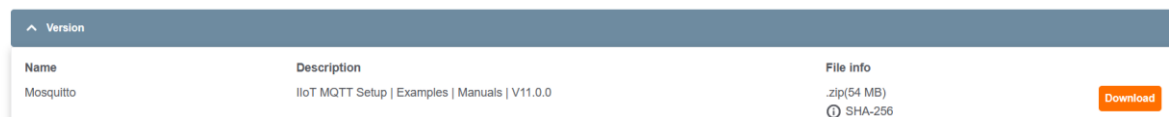
- IFM Moneo: <https://www.ifm.com/de/de/shared/moneo-downloads>

### 8.4 Objectives

In this task you will obtain data from an IO-Link device and read it over an MQTT interface. The data from the sensor then has to be visualized over node-red.

Firstly download the MQTT setup guide from IFM. This can be found in

<https://www.ifm.com/de/en/product/AL1300?tab=documents>



Version		
Name	Description	File info
Mosquitto	IIoT MQTT Setup   Examples   Manuals   V11.0.0	.zip(54 MB) SHA-256

Figure 16 Download page for MQTT setup guide from ifm

Follow the instructions for installation located in **2. PC Tool Setup/ Quick Start - MQTT - web subscribe & MosquittoV2.0.pptx**. For this guide you are expected to have already installed the Moneo ifm software.

Once you have setup Mosquitto you can now create a node-red program that subscribes to the MQTT topic to read the data from the sensor.

## 8.5 IO-Link Ethernet Setup

In case you do not get a connection to the IO-Link Master you can manually set the IP Address. Firstly be sure to connect the Ethernet M12 cable to the IOT port.



Figure 17 IO-Link Master IOT Port

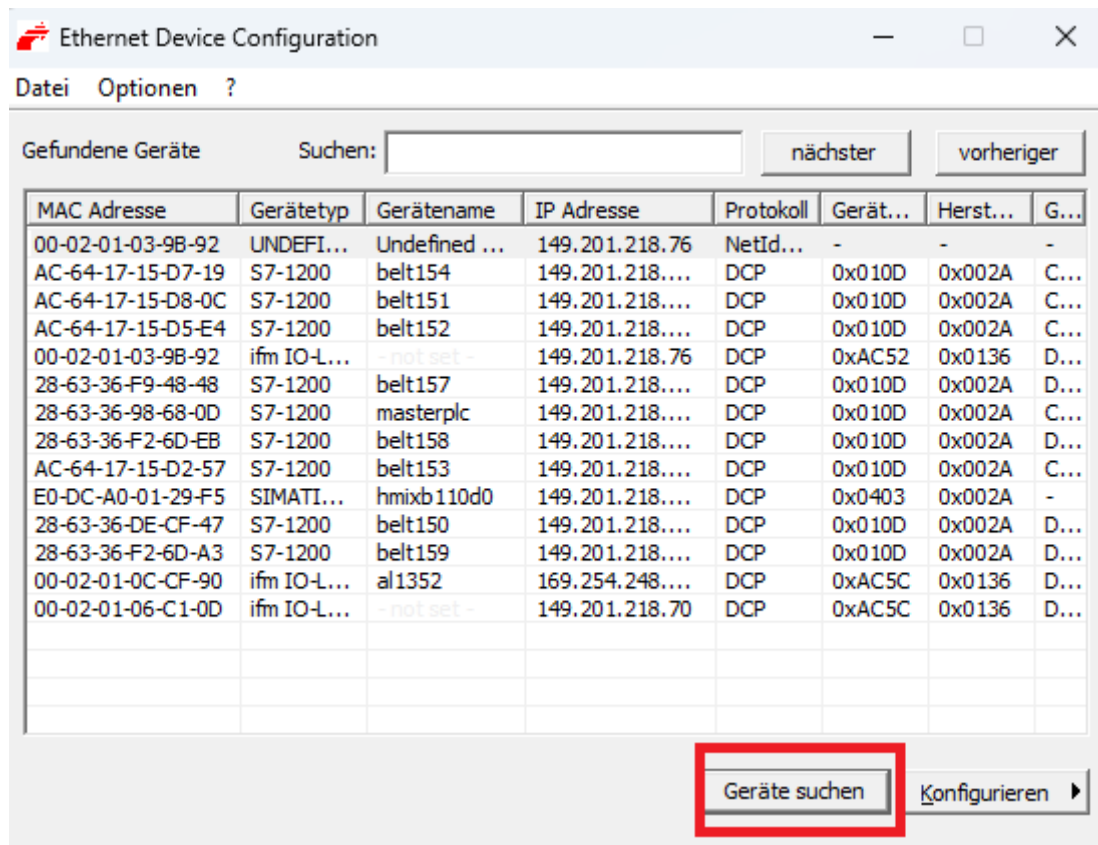
The IO-Link Master's IP Address must be set to the same subnet where the host machine running this PoC is connected to the IO-Link Master.

To configure the IO-Link Master's IP address you can install the Ethernet Device Configuration tool from Hilscher.

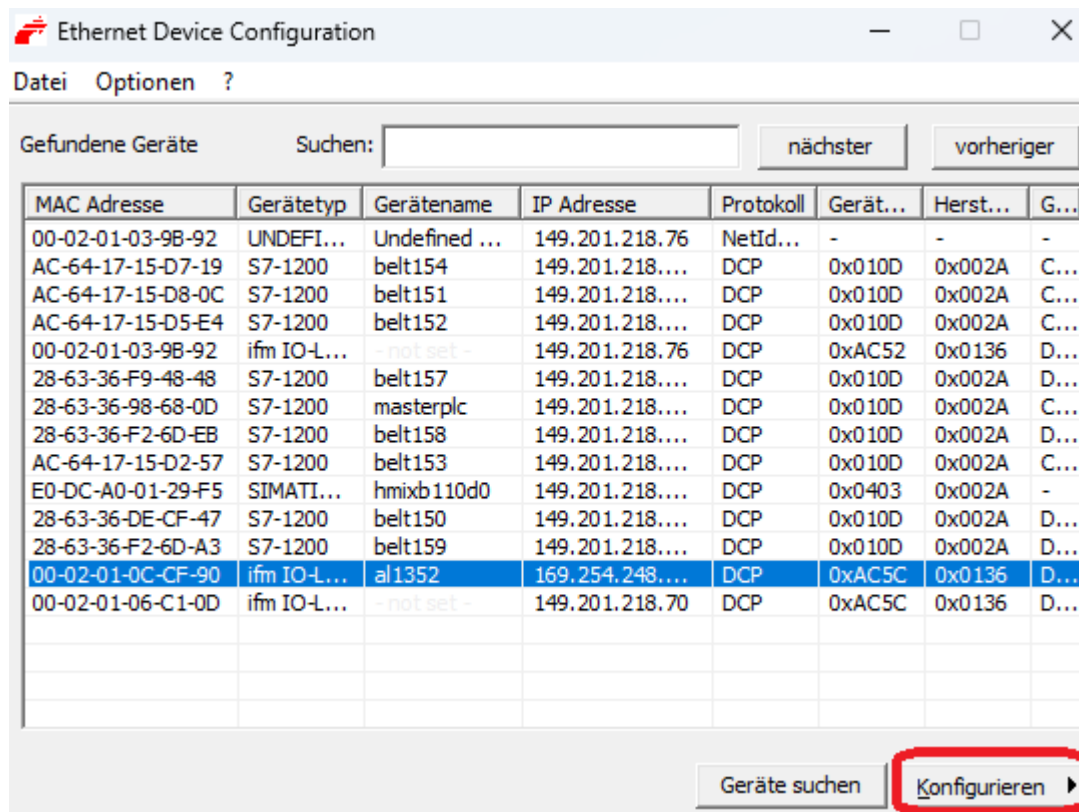
Download link: <https://hilscher.atlassian.net/wiki/spaces/ETHDEVCFG/pages/123076731/V1.0900.3.5506>

After installing the software you can run it and scan for ethernet devices.

1. Connect the IO-Link Master over the M12 Ethernet Interface with the IoT Core Port.
2. Scan the devices.



3. Select your IO-Link Master and click Setup >> Set IP Address



4. Configure IP address and save changes

IP Konfiguration für 00-02-01-0C-CF-90

☒ Statische IP Adresse benutzen

IP Adresse: **1** 169 . 254 . 248 . 100

Subnetzmaske: 255 . 255 . 255 . 0

Standardgateway: 0 . 0 . 0 . 0

☐ IP Adresse per DHCP beziehen

Authentisierungsmethode: Client ID

Client ID:

☐ Einstellungen temporär setzen **2**

**3**

#### Note

*Be sure to set the IP address in the same subnet as your host pc.*

## 9 Task 7 Arduino IO-Link Device

### 9.1 Hardware Requirements

- Arduino IO-Link Device shield
- Arduino Sensor of your choice
- ESP32 Microcontroller
- IO-Link Master

### 9.2 Introduction

You will be given a sensor which you will integrate into this device and create a product that uses this sensor and communicates over IO-Link.

#### 9.2.1 IO-Link shield

The IO-Link shield is the physical layer that sets the appropriate voltage levels for communication with the IO-Link master.

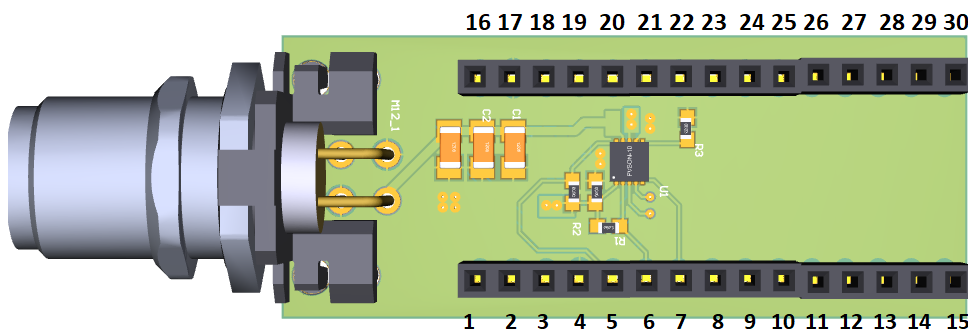


Figure 18 Pinout numbering from top view of Arduino IO-Link shield

Table 8 Required connections between  
IO-Link Shield and ESP32

IO-Link Shield Pin number	ESP32
1	17
2	16
4	GND
5	GPIO5
7	GPIO18

If the IO-Link shield you have has 16 x 2 pins (32 pins) you can follow the same pinout numbering as the only pins that are required are 1 to 7.



### 9.2.2 IO-Link Library

You can use the following IO-Link Arduino library to develop your IO-Link device.

<https://github.com/unref-ptr/lwIOLink>

Check the example located in *examples/lwIOLink\_Demo* and the [header](#) documentation for more information. For additional information about IO-Link concepts and terms, check the self-learning section of Task 6.

Notes:

- If you get the compilation error *"invalid conversion from 'uint8\_t {aka unsigned char}' to 'uart\_port\_t' [-fpermissive]"*, you should update your ESP32 platformio package to at least version 4.4.0 (see the next bullet point).
- Be sure to have the latest ESP32 package for platformio. To update the package run the visual studio command (ctrl + shift + p) *"Platformio:New terminal"*. Then in the new terminal type **pio pkg update**.
- If the communication does not seem to work, change the baud rate of IO-Link to COM2 or COM1 instead of COM3 in the [example code](#).

### 9.2.3 IO-Link Master

You can locally test the IO-Link communication with the WAGO PLCs located at the lab. These PLCs (WAGO 750-8202) include an IO-Link Master module and can be used to debug your application. Refer to file “*WAGO PLC IO-Link Device Setup.pdf*” uploaded in ILIAS. The Computer installed at your desk should be used to connect to the PLC as it already has the required software licenses and connections to the PLC via the green Ethernet cable.

# Laboratory Session Log

**Student:**

**Matriculation Number:**

Activity	Laboratory Supervisor Verification	Date	Remarks
Task 1 Modbus RTU			
Task 2 HTTP Requests			
Task 3 Project Modbus TCP			
Task 4 CAN Bus			
Task 5 CAN Joystick			
Task 6 IO- Link MQTT			
Task 7 Arduino IO-Link Device			