Final Degree Project

Bachelor’s Degree in Industrial Technology Engineering

Wipy board based HART-Wifi gateway for monitoring industrial sensors in a mobile app

REPORT

Author: Gerard Altés Gaspar
Director: Manuel Moreno Eguílaz
Summon: April 2017

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona
Abstract

This project develops a method of wireless communication between a mobile device and a HART modem through the programming in Micropython of the microcontroller based board WiPy in order to obtain the values of different HART sensors. For this, a wireless communication is established through the WiFi router between the WiPy and the client, and a direct link between the WiPy and the HART modem, in this way, the WiPy performs a data gateway function between them. The query is done through the client by entering the address of the sensor and the obtained response is composed by the measured value and the units, which are displayed on the mobile device.
Summary

ABSTRACT ......................................................................................................................... 1

SUMMARY ............................................................................................................................ 2

1. GLOSSARY ........................................................................................................................... 5

2. PREFACE .................................................................................................................................. 7
   2.1. Origin of the project........................................................................................................... 7
   2.2. Motivation.......................................................................................................................... 7
   2.3. Previous requirements....................................................................................................... 7

3. INTRODUCTION ...................................................................................................................... 8
   3.1. Project objectives................................................................................................................ 8
   3.2. Scope of the project.............................................................................................................. 8
   3.3. Background......................................................................................................................... 8
       3.3.1. Highway Addressable Remote Transducer (HART) Protocol.................................... 8
           3.3.1.1. How it Works?........................................................................................................... 9
           3.3.1.2. WirelessHART....................................................................................................... 12
           3.3.1.3. WiFi and HART.................................................................................................... 13
       3.3.2. MicroPython............................................................................................................... 13

4. DEVELOPMENT OF PROPOSALS ......................................................................................... 15
   4.1. Proposals.......................................................................................................................... 15
       4.1.1. First Proposal: WiPy 1.0 + Blynk............................................................................... 15
           4.1.1.1. Components........................................................................................................... 16
           4.1.1.1.1 WiPy 1.0 .............................................................................................................. 16
           4.1.1.1.2 Blynk.................................................................................................................. 17
           4.1.1.2. Issues..................................................................................................................... 18
       4.1.2. Second Proposal: WiPy 1.0 as a data gateway ............................................................... 18
           4.1.2.1. Issues..................................................................................................................... 18
       4.1.3. Third proposal: WiPy 2.0 + Blynk ............................................................................... 19
           4.1.3.1. Components........................................................................................................... 19
           4.1.3.1.1 WiPy 2.0 .............................................................................................................. 19
4.1.3.1.2 ........................................Blynk .................................................. 20
4.1.3.2. Issues ........................................................................................................ 20
4.1.4. Fourth proposal: WiPy 2.0 as a data gateway ............................................ 20

4.1.4.1. Components .............................................................................................. 21

4.1.4.1.1 ........................................ Wipy 2.0 .......................................................... 21
4.1.4.1.1.1 Run main program on WiPy ................................................................. 21
4.1.4.1.1.2 Run main program on client (PC or mobile device) ......................... 21

4.1.4.1.2 ........................................ Tkinter (Python) .................................................. 22

4.1.4.1.3 ........................................ SL4A and P4A .................................................. 22

4.1.5. Modem HART ............................................................................................. 22

4.1.5.1. InLink-TC™ HART Modem Module ......................................................... 22

4.2. WiPy 2.0 configuration ..................................................................................... 23

4.2.1.1. Connecting to power supply ................................................................. 23
4.2.1.2. Firmware update .................................................................................... 24
4.2.1.3. Connecting PC to WiPy ....................................................................... 25
4.2.1.4. Open REPL on PC ............................................................................. 25
4.2.1.5. FTP Connection .................................................................................. 25
4.2.1.6. Connect to WiFi Router ..................................................................... 27
4.2.1.7. Save Boot ............................................................................................ 29

5. RESULTS .................................................................................................................. 31

5.1. Files .................................................................................................................... 31

5.1.1. boot.py ....................................................................................................... 31
5.1.2. main.py ..................................................................................................... 31
5.1.3. gateway.py ................................................................................................. 31

5.1.3.1. UART ..................................................................................................... 31

5.1.4. client_GUI.py ............................................................................................. 32
5.1.5. client_wo_GUI.py ...................................................................................... 32

5.2. Test .................................................................................................................... 32

6. PLANNING ............................................................................................................. 34

7. BUDGET ................................................................................................................ 35

CONCLUSIONS ....................................................................................................... 36
1. Glossary

FTP (File Transfer Protocol): standardized file transfer protocol for sending files between computers with any operating system.

HART (Highway Addressable Remote Transducer): open protocol commonly used in control systems, used for remote configuration and monitoring data field instruments.

Telnet (Telecommunication Network): network protocol that allows you to manage another computer remotely as if we were sitting in front of it.

Python: interpreted programming language whose philosophy emphasizes a syntax that favours readable code.

UART (Universal Asynchronous Receiver-Transmitter): a piece of hardware, generally part of an integrated circuit, used for serial communications via the serial port of a computer or peripheral.
2. Preface

2.1. Origin of the project

This Final degree project was published in the ETSEIB repository for project proposals with the title “HART-Wifi gateway for industrial monitoring using Python” supervised by the Department of Electronics and arouse my personal interest.

2.2. Motivation

For personal interest, the subject of work should be linked to the environment of electronics and industrial control and also to the programming of a microcontroller to have a first experience with these components.

The possibility of developing future projects with the same microcontroller also makes it attractive.

2.3. Previous requirements

The initial knowledge about HART communication was insufficient. Therefore, a search and analysis of the fundamentals of HART communication is carried out in order to understand the origin of the data as well as the digital format of the sensor queries via the HART modem.

On the other hand, it has been analyzed the operation of the microcontroller based board (WiPy) and the communication methods needed to perform the required work such as FTP and Telnet protocols and communication through UARTs and WLAN.
3. Introduction

3.1. Project objectives

The main objective of this project is to program the microcontroller board (WiPy) to perform the communication between a HART modem and a mobile device through the WiFi connection available in the WiPy.

In order to achieve this main objective it must be possible to establish a stable connection between the mobile device and the WiPy to have an interface in the mobile device in order to send the address of the sensor to the WiPy and to establish a communication between the WiPy and the HART modem to send the request and the response with the corresponding values.

3.2. Scope of the project

In order to comply with the aforementioned objective, the following elements must be developed:

- Establish a wireless connection (WiFi) between a mobile device and the WiPy
- Create or use an existing interface to interact with the WiPy and the HART modem through the mobile device
- Program the WiPy to establish a communication connection between the WiPy and the HART modem

3.3. Background

3.3.1. Highway Addressable Remote Transducer (HART) Protocol

The HART protocol is the global standard communication across analogue wires between smart devices and control or monitoring system, by sending and receiving (bi-directional communication) digital information. [1]

Currently, all HART protocol rights belong to the HART Communication Foundation, but in fact, this protocol was developed by Rosemount Inc in the 1980's and transferred in the early 1990's to encourage its use, so it would be freely available to anyone. [2]
3.3.1.1. How it Works?

HART is called a hybrid protocol because it combines analog and digital communication. It can communicate a single variable using a 4-20 mA analog signal meanwhile it also communicates additional information of a digital signal. To achieve this, the HART protocol uses the Bell 202 standard frequency shift key (FSK) to superimpose the digital communication signals in a low intensity level above 4-20 mA. [3]

The FSK is a digital modulation system in which each bit or group of bits produces a certain variation of the frequency of the carrier wave.

The Bell 202 FSK uses a 1200 Hz tone for mark (typically a binary 1) and 2200 Hz for space (typically a binary 0).

The HART Protocol communicates at 1200 bps (1200Hz), allowing a host application (master) to get two or more digital updates per second.

This modulated signal has an average value of zero, so does not affect the 4-20mA signal. It is removed by standard analogue input circuit filtering. The ability to carry this added digital information is the basis for HART’s key benefits. [2]

![Image of Frequency Shift Keying (FSK)](source: FieldComm Group Webpage)

Fig. 3.1. Frequency Shift Keying (FSK). Source: FieldComm Group Webpage.
HART protocol allows the connection of up to two masters (primary and secondary). By using secondary masters, such as handheld communicators, you can communicate with the sensor without interfering with communications to / from the primary master (control system). [1]

Fig. 3.2. Frequency Shift Keying (FSK) during tests. Source: Own source.

Fig. 3.3. HART network example. Source: FieldComm Group Webpage.
To establish the communication between the master (host) and the slave device, a request must be generated. This request always has the same message structure (see Fig. 3.4).

Each message contains, among others, the address of the sensor to consult, to ensure that it is received by the correct device. This address is unique for each sensor. [2]

Fig. 3.4. HART Sensor Message Structure. Source: HART Field Communications Protocol: a technical overview (Bowden, Romilly).

The next part of the message after the Address field is the command. This part contains what kind of information the master (host) is requesting from the slave device. There are different commands that can be structured in the three classes, as shown in Fig. 3.5. [4]

<table>
<thead>
<tr>
<th>Universal Commands</th>
<th>Common Practice Commands</th>
<th>Device Specific Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognized and supported by all HART devices. Provide access to information useful in normal operations (e.g., read primary variable and units)</td>
<td>Implemented by many, but not necessarily all, HART Communication devices.</td>
<td>Functions that are unique to each field device</td>
</tr>
<tr>
<td>- Read manufacturer and device type</td>
<td>- Read selection of up to four dynamic variables</td>
<td>- Read or write low-flow cut-off</td>
</tr>
<tr>
<td>- Read primary variable (PV) and units</td>
<td>- Write damping time constant</td>
<td>- Start, stop, or clear totalizer</td>
</tr>
<tr>
<td>- Read current output and percent of range</td>
<td>- Write device range values</td>
<td>- Read or write density calibration factor</td>
</tr>
<tr>
<td>• Read up to four pre-defined dynamic variables</td>
<td>• Calibrate (set zero, set span)</td>
<td>• Choose PV (mass, flow, or density)</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>• Read or write eight-character tag, 16-character descriptor, date</td>
<td>• Set fixed output current</td>
<td>• Read or write materials or construction information</td>
</tr>
<tr>
<td>• Read or write 32-character message</td>
<td>• Perform self-test</td>
<td>• Trim sensor calibration</td>
</tr>
<tr>
<td>• Read device range values, units, and damping time constant</td>
<td>• Perform master reset</td>
<td>• PID enable</td>
</tr>
<tr>
<td>Read or write final assembly number</td>
<td>• Trim PV zero</td>
<td>• Write PID set point</td>
</tr>
<tr>
<td>Write polling address</td>
<td>• Write PV unit</td>
<td>Valve characterization</td>
</tr>
<tr>
<td></td>
<td>Trim DAC zero and gain</td>
<td>Valve set point</td>
</tr>
<tr>
<td></td>
<td>Write transfer function (square root/linear)</td>
<td>Travel limits</td>
</tr>
<tr>
<td></td>
<td>Write sensor serial number</td>
<td>User units</td>
</tr>
<tr>
<td></td>
<td>Read or write dynamic variable assignments</td>
<td>Local display information</td>
</tr>
</tbody>
</table>

Fig. 3.5. HART Protocol Commands. Source: FieldComm Group Webpage.

Beside this, each message has a checksum to allow detection of errors during transmission. Every slave device reply message will contain the field "Device Status", indicating the correct operating condition or, failing this, the error. [2]

### 3.3.1.2. WirelessHART

WirelessHART is a wireless sensor networking technology that works by using the HART protocol. This protocol supports operation in the 2.4 GHz ISM band using IEEE 802.15.4 (technical standard which defines the operation of low-rate wireless personal area networks) standard radios.

Through this protocol, it is possible to access the different devices at a distance. This fact allows to cover the need for technification and improvements of the industrial processes and their control, as well as the plant expansions and the new regulatory requirements and safety level demands. [5]
3.3.1.3. WiFi and HART

WiFi connections are based on IEEE 802.11 standards. Therefore, WiFi does not share the same protocol with WirelessHART.

The Wi-Fi industry has ambitions to make Wi-Fi the dominant wireless technology for the Internet of Things (IoT), It is more IP-friendly than any of the alternative wireless technologies, supports higher rates, and the chip industry backs it with high volumes and low prices. [6]

Using WiFi connection to interact with the Modem HART allows the use of WiFi networks already available or the installation of a new network, exclusive for this communication, at low cost.

3.3.2. MicroPython

MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments.

MicroPython is compact enough to fit and run within just 256k of code space and 16k of RAM.

The goal of MicroPython is to achieve the highest compatibility with normal Python to be able to use code already tested in the computer to the microcontroller. This way it is not necessary to learn a new language, as long as the programming in Python (known as CPython) is already known.

While running on the microcontroller, it is possible to get interactive messages (REPL "read-eval-print loop") to execute commands immediately using single user inputs, or to execute and import scripts already incorporated to the file system. The REPL has a history, tabulation, auto-indent and paste mode to facilitate the user experience.

Micropython includes the main Python libraries but, as an extra, it has its own microcontroller (machine) modules to access the low level hardware, thus gaining access to the I/O pins and the different types of microcontroller interfaces (I²C, I²S, UART, SPI, etc.) among others.

Since MicroPython is available under the MIT licence (free software license originating in the Massachusetts Institute of Technology), it can be used and adapted freely for both personal and educational uses as well as commercial applications, and it is also possible to obtain extension modules made by third parties.

MicroPython is developed in open-source in GitHub which is a web-based Git repository.
hosting service (Git is a control version system (VCS) for tracking changes in computer files and coordinating work in those files among multiple people) and the source code is available in the GitHub page and on the download page. [7]
4. Development of proposals

This part of the project has been the more tedious since it uses a hardware (WiPy) that has never before been used in projects of the department of electronic engineering. It is for this reason that, as the project progresses, the initial proposal could not meet the needs and it had been necessary to look for alternatives with which the objective of the project could be achieved in time.

4.1. Proposals

Through the following sections the different elements of hardware and software tested, both the initial proposal and the different alternatives will be presented.

4.1.1. First Proposal: WiPy 1.0 + Blynk

The initial proposal was to program the microcontroller based board WiPy 1.0 to connect with the Blynk App installed in the Client mobile device. In this way, it can represent the response provided by the HART modem.

The data flow in this proposal would be as follows:

1. Enter the address of the desired sensor in the Blynk App Widget Terminal
2. Via the internet connection of the mobile device (client) it is sent to the Blynk App server, which then transmits it to the WiPy
3. Once the sensor address is received by the WiPy, the address is processed and the message structure is generated, with the value-reading command, so that the HART modem can interpret it
4. WiPy sends the request via UART to the HART modem
5. The HART modem modulates the current loop to add the request message
6. The sensor responds to the request with a message with the value and the units
7. The HART modem demodulates the current loop, gets the response and sends it through the UART to the WiPy
8. The WiPy processes the response message and sends the value to the client's mobile device via WiFi by the Blynk server
9. The Blynk App represents, by means of a Gauge or Display, the value of the sensor

Another part of the programming focuses on establishing the UART connection between the WiPy and the HART modem.

4.1.1.1. Components

4.1.1.1.1 WiPy 1.0

At the beginning of the project the microcontroller based board WiPy 1.0 is available presenting the following main characteristics (Fig. 4.1.1):

<table>
<thead>
<tr>
<th>Mechanical</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Size: 25mm x 45mm (1.0” x 1.77&quot;)</td>
<td>• UART (x2), SPI, I2C, I2S, SD card</td>
</tr>
<tr>
<td>Processing</td>
<td>• Analog channels: 3×12 bit ADCs</td>
</tr>
<tr>
<td>• Dual processor + WiFi radio SoC</td>
<td>• Timers: 4×16 bit with PWM and input capture.</td>
</tr>
<tr>
<td>• MCU: Texas Instrument CC3200,</td>
<td>• 3v3 output capable of sourcing up to 250mA</td>
</tr>
<tr>
<td>Cortex-M4 @ 80MHz</td>
<td></td>
</tr>
<tr>
<td>• Network processor handles the</td>
<td>Hash and encryption engines</td>
</tr>
<tr>
<td>WiFi connectivity and the IP</td>
<td>• SHA</td>
</tr>
<tr>
<td>stack.</td>
<td>• MD5</td>
</tr>
<tr>
<td>• Main processor entirely free to</td>
<td>• DES</td>
</tr>
<tr>
<td>run the user application.</td>
<td>• AES</td>
</tr>
<tr>
<td>Memory</td>
<td>Networking</td>
</tr>
<tr>
<td>• RAM: 256 KBytes</td>
<td>• WiFi: 802.11b/g/n 16Mbps</td>
</tr>
<tr>
<td>• Flash: 2 MBytes</td>
<td>RTC</td>
</tr>
<tr>
<td>• GPIO: Up to 25</td>
<td>DMA</td>
</tr>
<tr>
<td></td>
<td>Security</td>
</tr>
</tbody>
</table>

Fig. 4.1.1. WiPy 1.0 Main characteristics. Source: pycom.io.

Through this hardware, it is intended to establish an internet connection through the WiFi
SoC (System-on-Chip) available on it and a router. Through this internet connection, the address of the sensor, sent by the Client, (in this case a mobile device or a PC) will be received and processed to send the request through a UART to the HART modem, which will respond with the sensor value. This reading will be processed by the WiPy and sent to the Client as shown in Fig. 4.1.2.

![Hardware structure diagram](image)

**Fig. 4.1.2. Hardware structure diagram. Source: Own source.**

4.1.1.1.2 Blynk

Blynk is a Platform with iOS and Android apps to control Arduino, Raspberry Pi and other different types of microcontroller based boards over the internet.

![Blynk graphic interface example](image)

**Fig. 4.1.3. Blynk graphic interface example. Source: blynk.cc.**

It is a digital control panel where it is possible to create a graphical interface for simply dragging and dropping widgets.
There is an existing BlynkLib.py library for WiPy 1.0 that allows WiPy to connect to the Blynk App through the application's own server to interact via the WiPy Pins assignment (either Analog, Digital or Virtual) to the widgets of the application. In this way, you can interact with the WiPy by sending the address of the sensor and graphically displaying the reading obtained from the sensor.

To configure a dashboard it is necessary to sign in and create a new project which will be assigned an Authorization Token. This token will enable the WiPy connection to the Blynk App server. Once the project has been created, the different widgets can be added and linked to the WiPy Pins. A simple example applicable to WiPy 1.0, where the LED is linked to the Digital Pin GP25, consists of define a button and assign the Digital Pin 25. When this is pressed, the LED of the microcontroller based board will be activated.

It must be taken into account that each widget consumes energy units of the App. Initially free 1000 units are available, that allow to use 3 or 4 widgets. But if more are needed, it is necessary to make a purchase of energy. [8]

4.1.1.2. Issues

This initial proposal was discarded since WiPy 1.0 does not support Float Numbers (IEEE-754) and did not allow the conversion of the HART modem response to a decimal value.

As an alternative, it was proposed to treat WiPy 1.0 as a simple data gateway and program in Python an application running on the mobile device.

4.1.2. Second Proposal: WiPy 1.0 as a data gateway

This proposal continues with the same operating structure as the previous one but differs from it in that the WiPy does not handle the response of the HART modem but sends it directly (gateway) to the mobile device which, through the use of a Python application, processes the response and displays the value on the screen.

It is not possible to process the answer message using the Blynk application, as it only supports a real number as an input to display in the graphical interface.

4.1.2.1. Issues

Due to a failure in WiPy 1.0 for unknown reasons the microcontroller becomes useless and the new version of the microcontroller based board, the WiPy 2.0 was ordered.
4.1.3. Third proposal: WiPy 2.0 + Blynk

4.1.3.1. Components

4.1.3.1.1 WiPy 2.0

This microcontroller based board, as its name indicates, is the updated version to the WiPy 1.0 and presents the following specifications (Fig. 4.1.4):

<table>
<thead>
<tr>
<th>WiPy Features</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Powerful CPU, BLE and state of the art WiFi radio 1km WiFi range</td>
<td>• 2 x UART, 2 x SPI, I2C, I2S, micro SD card</td>
</tr>
<tr>
<td>• MicroPython enabled, the Linux of IoT for fast deployment Fits in a standard breadboard (with headers)</td>
<td>• Analog channels: 8x12 bit ADCs</td>
</tr>
<tr>
<td>• Ultra-low power usage: a fraction compared to other connected micro controllers</td>
<td>• Timers: 4x16 bit with PWM and input capture</td>
</tr>
<tr>
<td></td>
<td>• DMA on all peripherals</td>
</tr>
<tr>
<td></td>
<td>• GPIO: Up to 24</td>
</tr>
<tr>
<td></td>
<td><strong>Hash / encryption</strong></td>
</tr>
<tr>
<td></td>
<td>• SHA, MD5, DES, AES</td>
</tr>
<tr>
<td><strong>Processing</strong></td>
<td><strong>WiFi</strong></td>
</tr>
<tr>
<td>• Espressif ESP32 chipset</td>
<td>• 802.11b/g/n 16mbps</td>
</tr>
<tr>
<td>• Dual processor and WiFi radio system on chip</td>
<td><strong>Bluetooth</strong></td>
</tr>
<tr>
<td>• Network processor handles the WiFi connectivity and the IPv6 stack</td>
<td>• Low energy and classic</td>
</tr>
<tr>
<td>• Main processor is entirely free to run the user application</td>
<td><strong>RTC</strong></td>
</tr>
<tr>
<td>• An extra ULP-coprocessor that can monitor GPIOs, the ADC channels and control most of the internal peripherals during deep-sleep mode while only consuming 25uA</td>
<td>• Running at 32KHz</td>
</tr>
<tr>
<td></td>
<td><strong>Power</strong></td>
</tr>
<tr>
<td></td>
<td>• 3.3V to 5.5V</td>
</tr>
<tr>
<td></td>
<td>• 3V3 output capable of sourcing up to 500mA</td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td><strong>Security and Certifications</strong></td>
</tr>
<tr>
<td>• Size: 42mm x 20mm x 3.5mm (excluding headers)</td>
<td>• SSL/TLS support</td>
</tr>
<tr>
<td>• Operating temperature: -40 to 85 degrees celsius</td>
<td>• WPA Enterprise security</td>
</tr>
<tr>
<td>• 3V3 ultra low noise switching regulator</td>
<td>• FCC – 2AJMTWIPY2R</td>
</tr>
<tr>
<td>• ESP32 dual core micro controller</td>
<td>• CE 0700</td>
</tr>
<tr>
<td></td>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td></td>
<td>• RAM: 512KB</td>
</tr>
<tr>
<td></td>
<td>• External flash 4MB</td>
</tr>
</tbody>
</table>
and WiFi / Bluetooth 4.2 radio
- Reset switch
- RGB heart beat LED
- 32Mbit flash memory
- RF switch
- U.FL connector
- High performance SMD antenna

Quick Verification
- For easy and fast debugging use the interactive shell that is accessible through telnet or one of the serial ports

Easy Upload
- Upload your scripts and any other files you want to the WiPy via the FTP server

Locally or remotely
- Reset the WiPy (you can do it locally or remotely via Telnet)

- Hardware floating point acceleration
- Python multi-threading

Fig. 4.1.4. WiPy 2.0 Main characteristics. Source: pycom.io.

4.1.3.1.2 Blynk

4.1.3.2. Issues

The WiPy firmware is in development stage, and updates are released periodically, which is why it does not have any of the modules used in the Blynk library (e.g. the Timers module). Since modifying the Blynk library exceeds the time and scope of the project, an alternative is to use the WiPy as a data gateway, like the second proposal, between the HART modem and an application developed in Python for use on the PC or on the mobile device through a script capable of running Python on Android. This proposal is explained in the following point.

4.1.4. Fourth proposal: WiPy 2.0 as a data gateway

The new WiPy 2.0 based microcontroller has better processing specifications and does support float numbers, according to the IEEE-754 standard. However, as it is in the development phase, it still does not have the necessary modules to run the connection with the Blynk App. That is why the above alternative of using the WiPy as data gateway resumes.
In this case the data flow sequence would be:

1. The client enters the address of the sensor

2. The client sends, through a network socket (internal endpoint for sending or receiving data at a single node in a computer network), the data packet to the WiPy.

   *The use of the network socket limits the independence of the client since it must be connected to the same network as the WiPy in order to establish a communication.*

3. The WiPy receives the data packet and processes it.

4. The WiPy sends the request via UART to the HART modem.

5. The HART modem modulates the current loop to add the request message.

6. The sensor responds to the request with the value and the unit.

7. The HART modem demodulates the current loop, gets the response and sends it through the UART.

8. WiPy processes the response and sends it to the client via socket network.

9. The client application processes the response and shows the value and the unit.

### 4.1.4.1. Components

#### 4.1.4.1.1 Wipy 2.0

Running the main program on the microcontroller allows the client to have minimal code and can execute the program without processing limit problems.

#### 4.1.4.1.2 Run main program on client (PC or mobile device)

Running the main application on the client allows the WiPy code to be unique and definitive, thus avoiding having to reprogram the WiPy before the need to add functionalities such as, for example, new commands. If adding new commands was necessary, the new code could be edited comfortably on a PC or on the Mobile Device itself.

Since this way is standardized and prevents the development of the program in WiPy, where the transfer and editing of programs is not trivial, it is decided to design the main program to run on the client.
4.1.4.1.2 Tkinter (Python)

Tkinter is Python's de-facto standard to the Graphical User Interface (GUI) toolkit package that enables rapid development of GUI applications. By using tkinter is possible to develop a simple and friendly application that is executable on the PC and allows to interact with the WiPy easily.

In this way, the same programming language is shared, thus simplifying the connections between the two devices, WiPy and Client. [9]

Tkinter is a script engine usually part of Python. However, Tcl/Tk often is not. In the standard Python available in Android this modules are not included. To solve this problem is necessary to use Scripting Layer for Android (SL4A) and Python for Android (P4A), through which it is possible to run a program with a graphical interface by adapting the PC program. Anyway a program without graphical interface has been added, executable in the ordinary Python for Android to expedite the use.

4.1.4.1.3 SL4A and P4A

The Scripting Layer for Android (SL4A) is a library that helps write scripts targeting the Android Platform. SL4A supports a number of scripting languages like Perl, Python, JRuby, PHP etc. [10]

To achieve this, it is necessary to install the program P4A built to run python on Android devices SL4A.

4.1.5. Modem HART

4.1.5.1. InLink-TC™ HART Modem Module

InLink-TC is a commercial HART protocol modem module that modulates the serial data of the logic level in a HART loop and demodulates the received HART data by providing a serial bit stream.

InLink-TC interfaces to standard 3.3V to 5V logic and only requires three control lines; Transmit (TXD), receive (RXD) and request to send (RTS). [11]
4.2. WiPy 2.0 configuration

4.2.1. Connecting to power supply

The WiPy 2.0 + Expansion Board pack is available. The acquisition of the Expansion Board is highly recommended as it facilitates the power supply of the WiPy and the connection to the different Pins, as well as the use of an SD card. It also enables the firmware update through the USB connection to the PC.

If the Expansion Board is available, the correct fitting is important because if the connection is mistakenly reversed, it could cause irreparable damage to the microcontroller. Under WiPy 2.0 are the labels of each Pin, you must make the pin of "Vin" with the one of the Expansion board (see Fig. 4.2.1).

Fig. 4.2.1. Expansion Board and WiPy 2.0 connection. Source: pycom.io.
Once the set is in place, it is allowed to connect via USB to the PC.

4.2.1.2. Firmware update

The manufacturer’s recommendation is to upgrade to the latest firmware, as it is a microcontroller in the development phase and should still incorporate improvements. It is advisable to check periodically that the latest firmware version is available. Updates announcements are posted on the Pycom forum (https://forum.pycom.io/category/1/announcements).

To perform the firmware update you must proceed to download the file available at https://www.pycom.io/support/supportdownloads/ for Windows, Linux or OS X

Once it has been downloaded and installed, it will be executed, and will announce that a new upgrade is available. The following steps are simple. It will only be necessary to choose the type of device that is available, follow the connection instructions of the jumper cable and choose the port on which it is connected (this depends on each computer but usually there is always only one port to choose from) (see Fig. 4.2.2).

![Firmware Updating](https://example.com/firmware-updating.png)

**Warning:** Note that to perform the update, a UART connection called "uart" is created using the P1 and P2 pins of the WiPy. It is very important to remember that if the user needs to create a new UART connection this is not called "uart" or make use of pins P1 and P2 since it would overwrite the already created UART and this generates problems of operation in the microcontroller.
4.2.1.3. Connecting PC to WiPy

When the WiPy receives power supply, a new WiFi network will be available. You will proceed to connect to it using the security key www.pycom.io. Once the connection is established, it is possible to execute code and transfer files to WiPy.

4.2.1.4. Open REPL on PC

In order to interact with the device it is necessary to open the REPL through a telnet connection (Telecommunication Network), a network protocol that allows to manage another equipment remotely, this action can be done in two ways.

The first one, by installing and configuring the device in the Pymkr program (https://www.pycom.io/solutions/pymakr/) as indicated in the "Quickstart Guide" for WiPy (https://docs.pycom.io/pycom_esp32/pycom_esp32/getstarted.html#connecting-your-board-using-pymakr).

The second, simpler way, is to open the Windows / Linux / OS X command interpreter and run the command "telnet 192.168.4.1" for Windows and "$ telnet 192.168.4.1" for Linux and OS X. This command sets a Telnet connection between the PC and the WiPy, that has default IP 192.168.4.1.

Once the command is executed, the WiPy will request the credentials that by default are login: micro and password: python.

This way the connection with the WiPy is already available and any micropython command can be executed or the compilation process of programs previously transferred can be seen.

![Executing micropython in the WiPy via telnet. Source: Own source](image)

4.2.1.5. FTP Connection

To transfer the files to the WiPy it is needed to establish an FTP (File Transfer Protocol) connection, a standardized file transfer protocol to send files between computers with any operating system. [12]
To perform this action, as recommended by the manufacturer, it is necessary to install the program FileZilla ([https://filezilla-project.org/](https://filezilla-project.org/)). Once the program is installed and executed, open the Site Manager (Ctrl + S) and create a new configuration, since we cannot make the connection through the "Quickconnect".

In the **General** tab the IP of the WiPy must be entered in the Host and the set the Port 21 (the port must be entered manually each time the connection is established) make sure that encryption is set to: Only use plain FTP (insecure). In the **Transfer Settings** tab set the transfer mode in Passive and limit the max number of connections to one [13], otherwise FileZilla will try to open a second command connection when retrieving and saving files, and for simplicity and reduces code size, only one command and One data connections are possible (see Fig. 4.2.4).

![FTP connection settings](image)

**Fig. 4.2.4.** FTP connection settings. Source: Own source.

Once the FTP connection is established, you can access the directories and files contained in the WiPy and transfer new files.
Warning: Keep in mind that when the WiPy is connected to the router’s WLAN, it will assign a new IP that must be known to establish the connection again (both on telnet and FTP). (See how to get the new IP address in 4.2.1.6 Connect to WiFi Router)

4.2.1.6. Connect to WiFi Router

To get the full potential of WiPy we must be able to access it, not only when connected to the PC but also from anywhere else.

To achieve this goal, a connection with the WiFi router must be established. So, one of the options is to replace the boot.py file with the one in the attachments by adding the name and password of the WiFi network to the list of known networks. Remember that this file is executed every time the WiPy is restarted.
Fig. 4.2.6. Generic WiFi connection code. Source: Own source.

To transfer the file using FileZilla, it is as simple as opening the directory where the `main.py` file is contained in our PC and dragging it to the WiPy /flash directory and allowing to overwrite the file (see Fig. 4.2.7). To execute the file we can restart the WiPy, either through the button or the telnet connection with the command `machine.reset()`.

Fig. 4.2.7. Screen capture overwriting `main.py` file. Source: Own source.
Warning: Note that when the WiPy is connected to the WLAN of the router, it will assign a new IP address to the WiPy that is needed to know to establish the connection again (both in telnet and FTP). It is possible to know the new IP address by accessing the network map or with the help of a connection visualization program (for example: Wireless Network Watcher v2.05 Copyright © 2011 - 2017 Nir Sofer [http://www.nirsoft.net/utils/wireless_network_watcher.html])

Fig. 4.2.8. Screen capture of the Wireless Network Watcher. Source: Own source.

In this case, to establish the telnet connection again it is necessary to replace the IP address 192.168.4.1 with the new one assigned by the router (192.168.1.41 in the example in Fig. 4.2.8).

For the FTP connection in FileZilla it is possible to configure a new site just like the one already configured only by replacing the host address (Reminder: Enter Port 21 each time)

4.2.1.7. Save Boot

If you power up normally, or press the reset button, the WiPy will boot into standard mode; the boot.py file will be executed first, then main.py will run.

But if there is an error in the executed code of the transferred files and the possibility of connecting directly to the WiPy is lost, it has a recovery mode.

You can override the normal boot sequence by pulling P12 (G28 in the Expansion Board) up (connect it to the 3V3 output pin) during reset. This procedure also allows going back in time
to old firmware versions. Three different firmware versions, which are: the factory firmware plus 2 OTA images.

![Boot modes and safe boot](image)

Fig.4.2.9. Boot modes and safe boot. Source: WiPy 2.0 Pinout.
5. Results

5.1. Files

This project consists of five main files to execute the communications between the client and the HART modem, boot.py, main.py, gateway.py, client_GUI.py and client_wo_GUI.py, all available in the attachments.

5.1.1. boot.py

Connects the WiPy to the WiFi Router, it is a generic program.

5.1.2. main.py

This file is executed automatically after boot.py. It contains the command to execute the file gateway.py.

5.1.3. gateway.py

This is the main program and it contains the necessary code to establish the UART connection to the HART modem and the communication to the customer via sockets.

5.1.3.1. UART

UART connection with the HART modem is defined. This connection meets the conditions of the variables indicated in HART modem specifications shown in Fig. 5.1

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud Rate</td>
<td>1200 bd</td>
</tr>
<tr>
<td>Start Bit</td>
<td>1 bit</td>
</tr>
<tr>
<td>Parity Bit</td>
<td>1 bit (Odd)</td>
</tr>
<tr>
<td>Stop Bit</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

Fig. 5.1. HART modem UART Specifications. Source: Own source.

When the UART is defined, the I/O pins must be linked to the transmission (Tx) and Receive (Rx) as well as the Ready-to-Send (RTS). These input are assigned to the P3, P4 and P5 Pins (G24, G11, G12 in the Expansion Board), respectively.
Once these pins are known, the connection to the HART modem can be made by connecting Tx-Rx, Rx-Tx and RTS-RTS as well as GND and 3V3 power (see Fig. 5.3)

![WiPy-HART modem connection](image)

**Fig. 5.3.** WiPy-HART modem connection. Source: Own source.

### 5.1.4. client(GUI).py

The client(GUI).py file processes the response received from the WiPy, which contains the sensor response message. The program processes this message and returns the value of the sensor as well as the unit of measurement of the sensor through a graphical interface (GUI), developed by the Tkinter module.

### 5.1.5. client_no_GUI.py

This file performs the same process as the client(GUI).py but it does not have a graphical user interface so it can run without installing SL4A and P4A since it uses the python terminal to enter the addresses and display the values and units.

### 5.2. Test

The program test is performed by a HART sensor emulator because the price of a HART
sensor is very high. This emulator simulates two sensors, one of pressure and one of temperature.

The test consists of configuring these sensors to a specific value and obtaining the two values in the client, each one by entering the address of the corresponding sensor (see Fig. 5.4).

![TEST 1](image1)

![TEST 2](image2)

Fig. 5.4. Test Results. Source: Own source.
6. Planning
# 7. Budget

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Price (€)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiPy 1.0</td>
<td>1</td>
<td>19.95</td>
<td>19.95</td>
</tr>
<tr>
<td>Expansion Board</td>
<td>1</td>
<td>16.00</td>
<td>16.00</td>
</tr>
<tr>
<td>WiPy 2.0</td>
<td>1</td>
<td>19.95</td>
<td>19.95</td>
</tr>
<tr>
<td>InLink-TC™ HART Modem Module</td>
<td>1</td>
<td>89.25</td>
<td>89.25</td>
</tr>
<tr>
<td>InLink- Evaluation Board</td>
<td>1</td>
<td>29.45</td>
<td>29.45</td>
</tr>
<tr>
<td>PC amortization</td>
<td>15%</td>
<td>600.00</td>
<td>90.00</td>
</tr>
<tr>
<td>HART Sensors emulator amortization</td>
<td>10%</td>
<td>600.00</td>
<td>60.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>324,60 EUR</strong></td>
</tr>
</tbody>
</table>
Conclusions

By using the WiPy it has been possible to establish a wireless connection between the client and the HART sensors, thus achieving the objective of the project. It is a stable but slow connection, since to receive a response it takes about 7 seconds when the HART protocol allows up to three requests per second. The bottleneck is located in the HART modem, which needs to receive bytes with high separation times (0.5s).

Possibly this point can be optimized to reduce the query time since if it were a critical sensor maybe 7 seconds would be too much time between readings.

As being the first experience with the WiPy in UPC, during development of the project they have presented a series of adversities with software and hardware WiPy 1.0 that have been solved but have delayed the project considerably.

The use of the WiFi network and the WiPy allows to adapt all the HART sensors to a method of Wireless communication without having to allocate great economic resources, since the sensors and the WiFi network already installed can be used.

Some possible advances in the project would be the inclusion of new commands in the program to be able to interact more specifically with the sensors.
Bibliography

Bibliographical references


