Mong Phan, Xuan Dang, Sami Barbaglia, Hanh Hoang

Ventilation Controller Project

Object-Oriented C-Programming in IoT Devices Project Report

Second-year Hardware Project
School of ICT
Metropolia University of Applied Sciences
12 March 2024

Contents

1 Introduction

In our modern living and working environments, ensuring proper ventilation is crucial for maintaining comfort, health, and safety. Whether it's regulating temperature, managing humidity levels, or controlling CO2 concentrations, effective ventilation plays a pivotal role in creating spaces conducive to productivity and well-being.

The Ventilation Controller Project represents a significant step forward in achieving optimal ventilation control. This project aims to develop a sophisticated ventilation controller system capable of seamlessly integrating both local and remote user interfaces, thereby providing users with convenient and versatile control over their ventilation systems. The primary objective of this project is to design and implement a ventilation controller system equipped with two operational modes: manual and automatic.

In manual mode, the user can directly set the speed of the ventilation fan (0 – 100%) using the local user interface (UI), which includes buttons, rotary knob, and a display. Alternatively, the user can control the fan speed remotely using the web UI by sending MQTT messages. The automatic mode allows user to set the target pressure within the ventilation duct in a range of 0 – 120pa and the system adjusts the fan speed accordingly. This setting is also enabled on both local UI and from an MQTT controller.

In the following documentation, we will provide a comprehensive project overview by delving into various key aspects. Following the introduction, section 2 provides the methods and materials of the ventilation system. Section 3 details the implementation of the system, including hardware setup, software architecture, and integration with external interfaces. Lastly, section 4 summarizes this project by highlighting key achievements, challenges encountered, and opportunities for future development. In Section 5, references are provided for further exploration and validation of the concepts discussed.

Figure 1. below is a picture of the physical parts of the full-scale ventilation project. Figure 2. is a close-up of the RPI Pico with RS-485 interface (the test bench from which the full-scale ventilator is controlled from), circled in red in Figure 1. Figure 3. shows our small-scale version of the ventilator, which we mainly worked on.

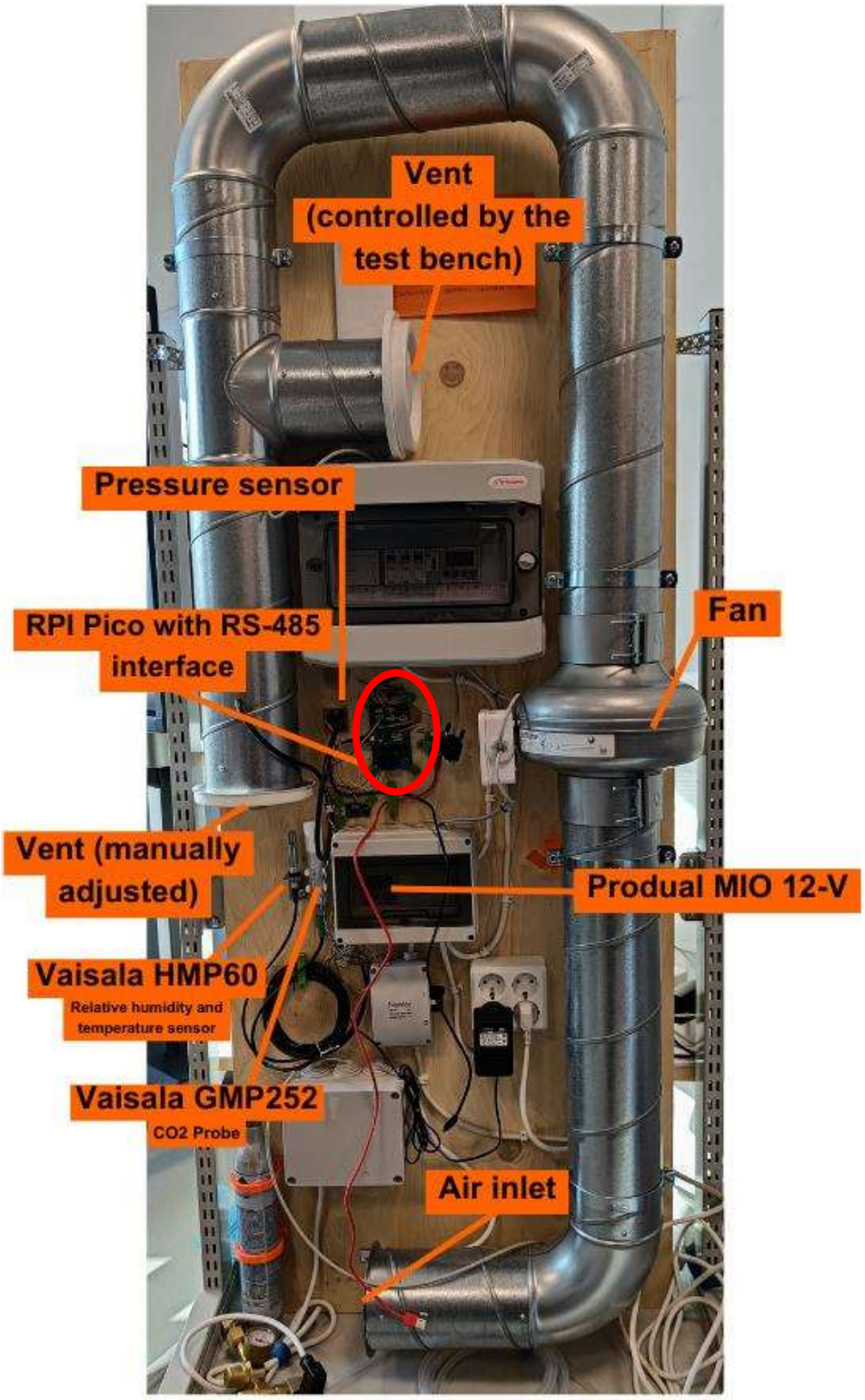


Figure 1: Physical parts of the full-scale ventilator project.

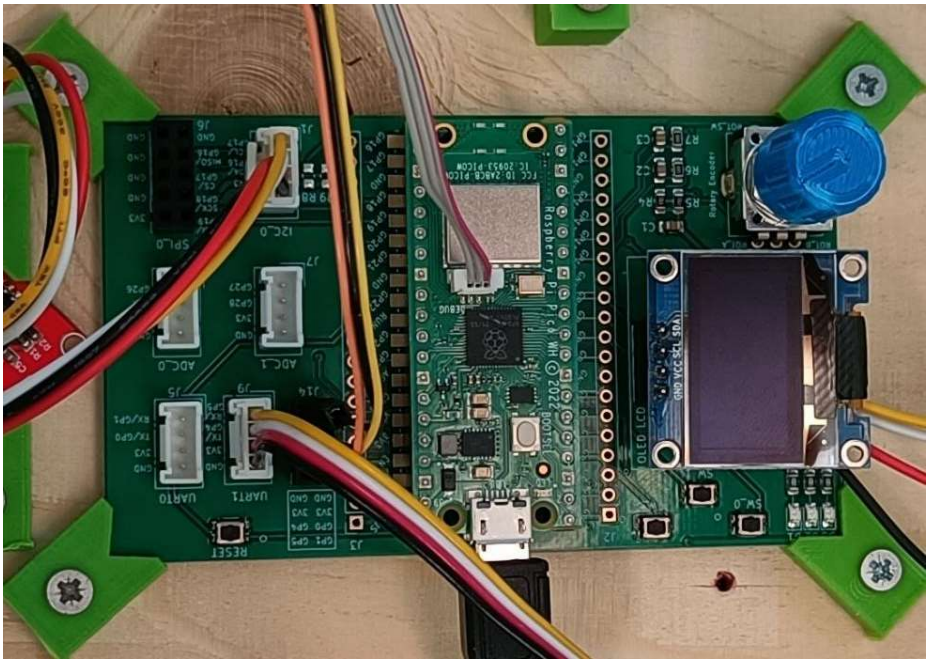


Figure 2: Close up of the controlling unit (RPI Pico with RS-485 interface), circled in red in figure 1.

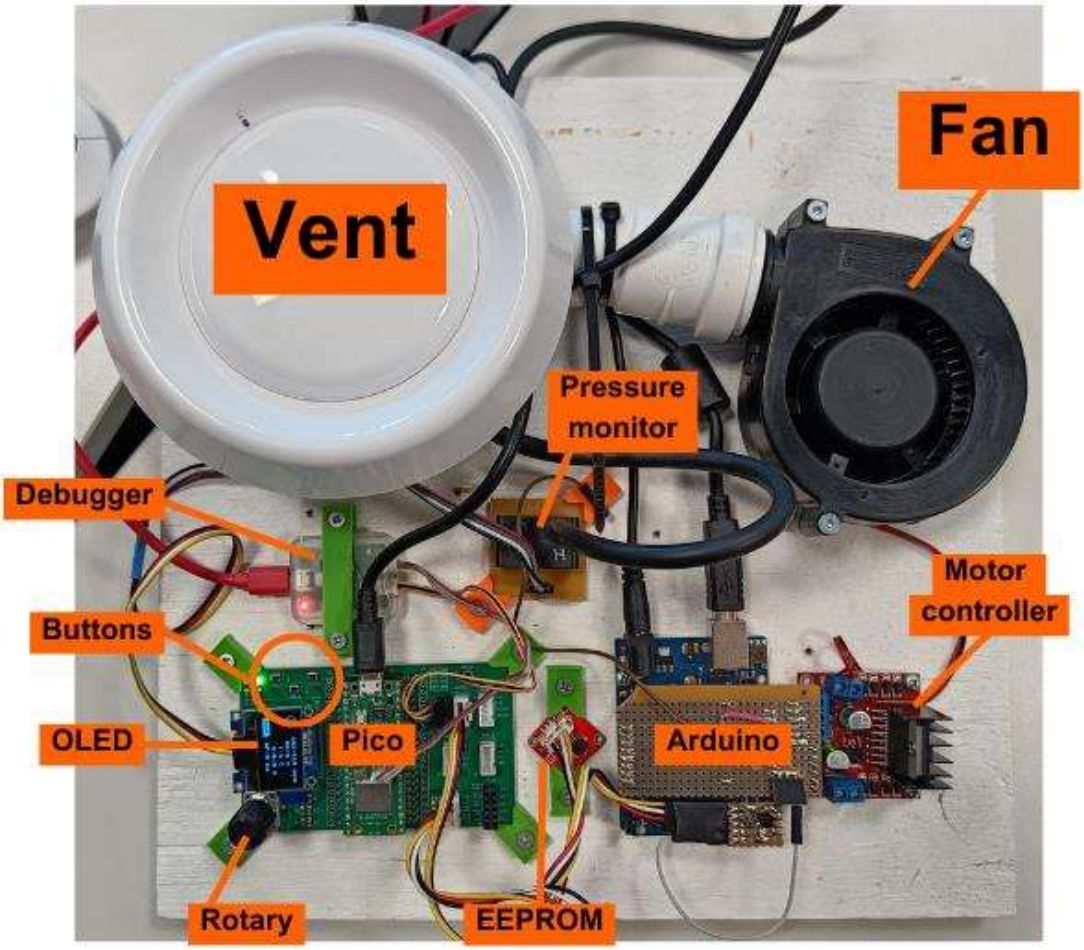


Figure 3: Physical parts of the miniature test system hardware.

2 Methods and Materials

This chapter goes over the software, protocols, and hardware used in the project. The software and protocols section covers C++, Python, Modbus, and MQTT, while the hardware section covers all the physical parts of both the miniature and full-size versions. Figure 4. below illustrates the diagram of our miniature test system.

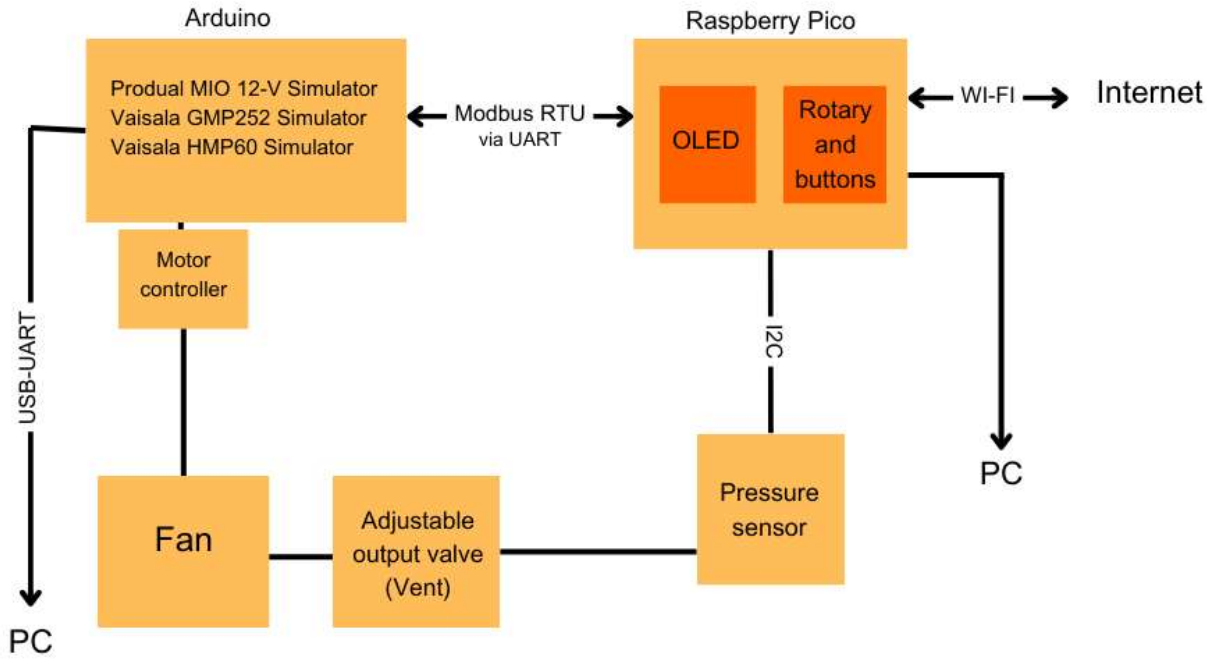


Figure 4: Miniature test system diagram

2.1 Software and Protocols

This section goes over the software and protocols used in the project. It covers the programming languages C++ and Python, as well as messaging/communication protocols MQTT and Modbus RTU.

Table 1. below lists the key software components and features, followed by a short description.

Key Software Components and Features	Description
C++ / CLion	Programming language / IDE. Used for adjusting fan speed based on user input or target pressure in automatic mode.
Python / PyCharm	Programming language / IDE. Used for handling MQTT messages received from the web interface or other remote clients
MQTT	Message Queuing Telemetry Transport. A publish/subscribe, standards-based messaging protocol [1.]. Utilizes PahoMQTT in the project.
Modbus RTU	Modbus Remote Terminal Unit. Client-host relationship model request-response protocol for serial lines [2.].

Table 1. Table of software materials used in the project.

2.1.1 Programming Languages

The project uses two primary programming languages, C++ and Python. The main programming language is C++, which is used for controlling the fan speed based on the user input in manual mode and target pressure in automatic mode. Python is used for handling MQTT messages to and from the MQTT-based web user interface simulator.

2.1.2. Protocols

Effective communication and control are made possible with the messaging/communication protocols, MQTT and Modbus RTU. MQTT is a lightweight and efficient, publish/subscribe standards-based messaging protocol [1.]. Modbus RTU is a client/server data communication protocol for serial lines [2.].

In the project, by utilizing Python, PahoMQTT, and a broker, the system creates two-way communication between the remote UI and the MQTT based ventilation controller. The controller subscribes to a topic to receive commands (setting fan speed or target pressure) and publishes updates on publishing topic to report system's status and display on remote UI. The messages are formatted in JSON for readability and usability. Figure 5. is a screenshot of the Python MQTT based UI simulator used.

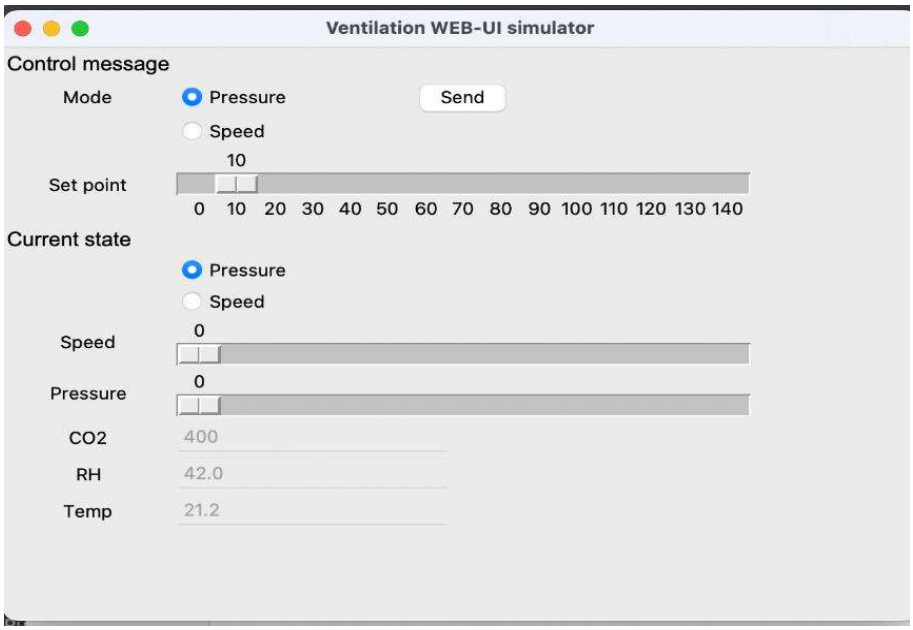


Figure 5: MQTT based ventilation web-UI

Modbus RTU is a client-host relationship model request-response protocol for serial lines [2.]. The Modbus Client object acts as a master (microcontroller Raspberry Pi Pico), which sends requests to read or write to other Modbus Registers instances (sensors and actuators). Message timing requirements are very strict in Modbus RTU, the bytes must be transmitted back-to-back, with only minimal delay allowed [3.]. In the project, UART is used for exchanging data between the microcontroller and sensors and actuators. These include:

- Vaisala GMP252 and HMP60 (simulated by Arduino in miniature version)
- Produal MIO 12-V, a Modbus-controlled IO-devices managing fan speed
- Sensirion SDP610 for pressure measuring

For example, the Modbus writes data to control fan speed according to the automatic or manual settings, and we can monitor the ventilation system in real-time with the data we're reading.

2.2 Hardware

This section goes over the hardware used in the project. This includes Produal MIO 12-V and the OLED screen, parts of the microcontroller, and EEPROM.

Table 2. below names and shortly describes all hardware used both in miniature and full-size versions of the project, but hardware exclusively used on the full-size version will not be further mentioned. Table 3. outlines the development pins in the project. Table 4. lists the Modbus sensors with corresponding Modbus address and register address.

Key Hardware Components and Features	Description
Produal MIO 12-V	Modbus controlled IO-device. The voltage output is connected to the fan speed input, controlling the speed of the fan. Implemented in miniature test system by Arduino. [4.]
Vaisala GMP252	Carbon dioxide (CO2) Probe. Simulated by Arduino in miniature test system.
Vaisala HMP60	Relative humidity and temperature sensor. Simulated by Arduino in the miniature test system.
Sensirion SDP610 – 120Pa pressure sensor	Pressure sensor. Simulated by Arduino in the miniature test system.

Vent	The full-scale project has two vents, one controlled by the test bench and one manually adjusted. The miniature version has one, used for both.
SSD1306 OLED Display	Single-chip OLED driver and display [5.]. Display used in the project. Provides valuable feedback on the status and operation of the ventilation system.
RPI Pico with RS-485 interface	The test bench of the full-size project, where the system is controlled from.
Fan	Fan will operate at high speeds, medium, low, or auto. The speed creates the pressure, in automatic mode the system finds the correct speed for the set pressure.
Arduino	A controller for the ventilation system, handling tasks such as sensor data acquisition, control algorithm execution, and MQTT communication independently. Simulates both Vaisala sensors in the miniature version.
Raspberry Pi Pico	A microcontroller board for the RP2040.
RP2040	A microcontroller chip. Features include a dual ARM Cortex-M0+ @ 133MHz, 264 kB SRAM on the chip, and thirty GPIO pins. [6.]
Raspberry Pi Debug Probe	All-in-one USB-to-debug kit (Arm Serial Wire Debug). Works with OpenOCD. Makes debugging possible without soldering etc. [7.]
EEPROM	An Electrically Erasable Programmable Read-Only Memory. Communicates through I2C.

Table 2. Table of hardware materials used in the project.

Peripherals/Components	**Description/Feature**	**GPIO Pins**

SW_0	Switch to confirm the selection	GPIO9
SW_2	Switch to access main UI on OLED display	GPIO7
ROT_A	Rotary encoder	GPIO10
ROT_B	Rotary encoder	GPIO11
I2C_0 SCL	EEPROM Integrated Circuit Clock	GPIO17
I2C_0 SDA	EEPROM Integrated Circuit Data	GPIO16
UART_1 TX	Modbus Transmitter	GPIO4
UART_1 RX	Modbus Receiver	GPIO5
I2C_1 SCL	Integrated Circuit Clock for I2C OLED display and Pressure sensor	GPIO15
I2C_1 SDA	Integrated Circuit Data for I2C OLED display and Pressure sensor	GPIO14
Debug Probe	For debugging and print logs	GPIO1, 0, GND

Table 3. Development Pins in microcontroller board Raspberry Pi Pico

Modbus sensors and actuators	**Modbus address**	**Register address**	**Baud rate**
Produal MIO 12-V (Fan speed)	1	0	9600
Vaisala GMP252 (CO2 Probe)	240	256	9600
Vaisala HMP60 (Humidity sensor)	241	256	9600

Vaisala HMP60 (Temperature sensor)	241	257	9600

Table 4. Modbus sensors with corresponding Modbus address and register address.

2.2.1 I2C_0 EEPROM

The EEPROM chip used in the project is compatible with AT24C256 which has highest memory address of 32767.

The highest memory address is used to store programming mode, measuring pressure, measuring fan speed. It also stores the input MQTT broker IP and network credentials. Figure 6. showcases Crowtail I2C EEPROM, the non-volatile memory of the system.

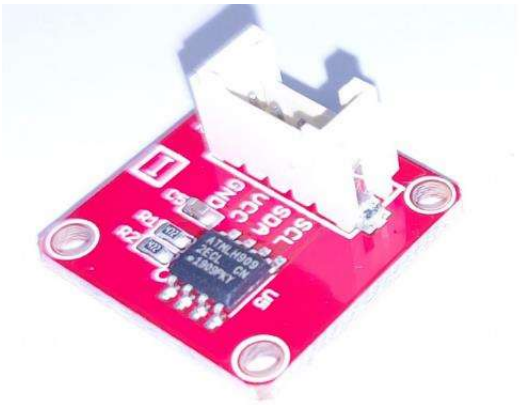


Figure 6: Crowtail I2C EEPROM

2.2.2. I2C_1 on SSD1306 OLED Display and Sensirion SDP610

I2C_1 acts as the communication channel for SSD1306 OLED display and Sensirion SDP610 pressure sensor to the microcontroller, initializing the I2C bus at clock speed of 100kHz. The SD610 pressure sensor (address: 0x40) retrieves its readings through this bus. Figure 7. showcases SSD1306 OLED Display

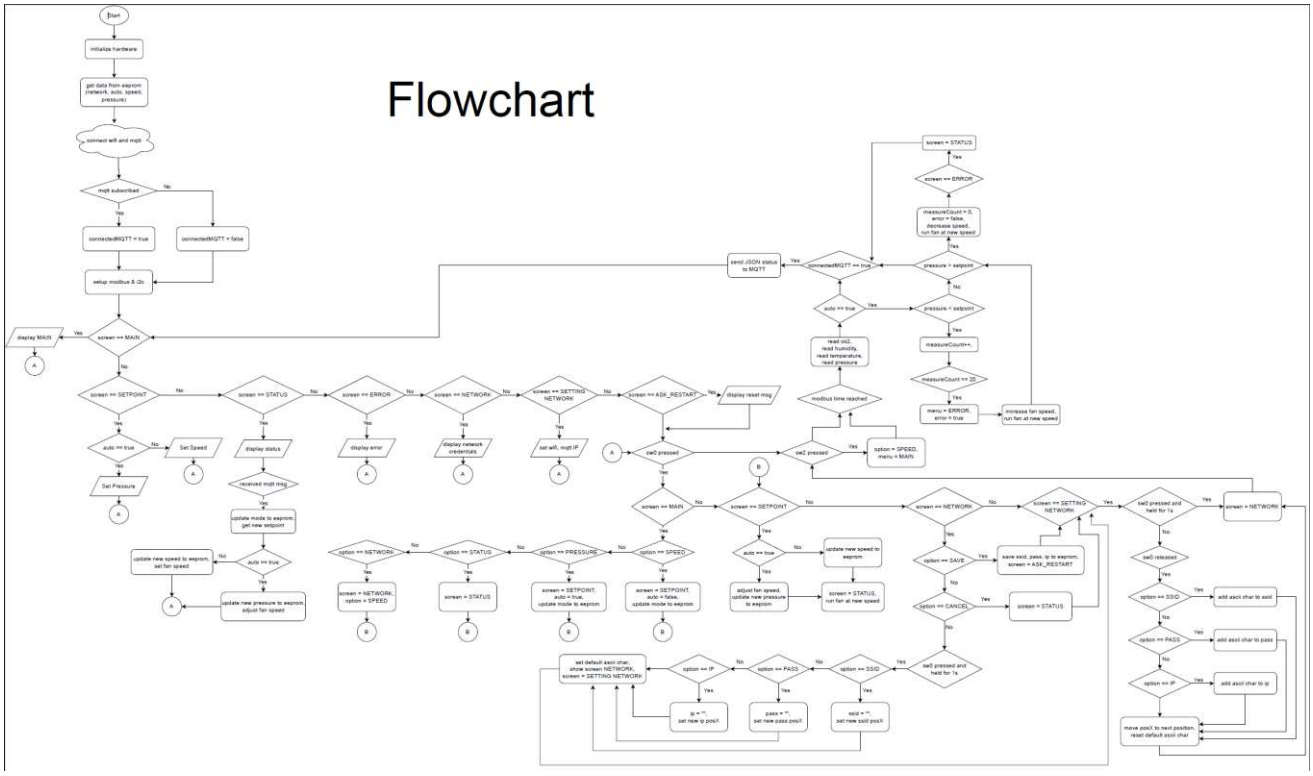
Figure 7: SSD1306 OLED Display

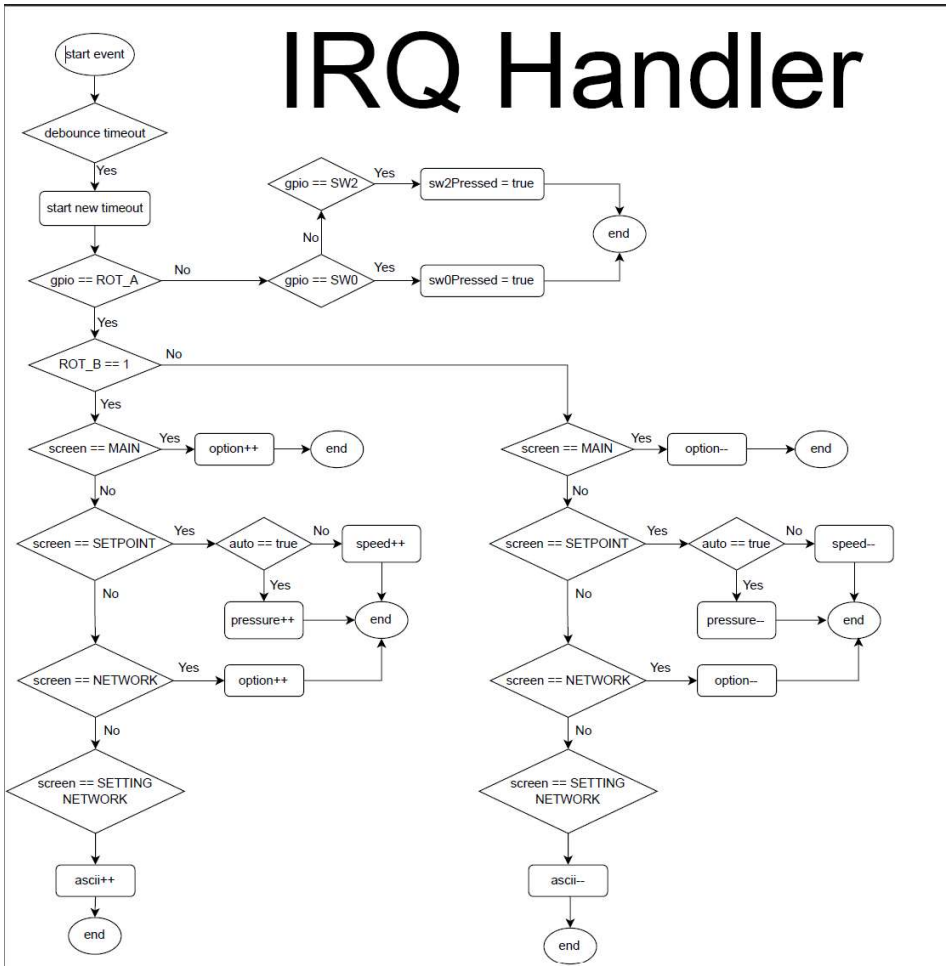
2.2.3. Switch and rotary knob

SW_0 is used for navigating to main menu in the local UI. It's initialized using Button class and operated using hardware interrupts.

Rotary knob ROT_A, ROT_B and ROT_SW are operated and detected using hardware interrupts. ROT_A and ROT_B are used to change different selection in a certain menu while ROT_SW is for confirming the selection.

3 Implementation





The process of implementing an embedded system can be divided into many stages, some of which operate entirely independently of the others and others of which depend on one or more modules that have already been developed. This section will go over how the program is implemented in detail.

The program starts by initializing the hardware followed by getting network credentials, mode, speed, and pressure values from the EEPROM. The device then connects to WiFi and subscribes the MQTT broker. After this step, Pi Pico establishes the connection to peripherals via Modbus and I2C protocols. In the main loop, the device will display the information based on the current screen:

- MAIN screen: shows 4 options to select:
    - Screen to set the fan speed (manual mode).
    - Screen to set the air pressure (auto mode).
    - Screen to show status (sensor's values).
    - Screen to change the WiFi credentials and MQTT broker's IP address.
- SETPOINT screen: whether showing setting the fan speed or air pressure.
    - Set fan speed (manual mode): the fan speed is set from 0% - 100%. After setting this parameter, the fan will start to run with the setpoint value. The microcontroller then measures the air pressure and displays its value on STATUS screen.
    - Set air pressure (auto mode): this setting allows the system to adjust the fan speed to get the desired air pressure which has been set. After confirming the pressure value, the fan will start to run with a relative speed then start to fine tune the speed to get the pressure as close as the setpoint possible.
    - After confirming the setpoint, the LCD displays back the STATUS screen to show all information.
- STATUS screen: where we can monitor co2 level, humidity percentage, temperature, fan speed, and current air pressure.
- ERROR screen: displays the error message when the air pressure failed to reach the target (setpoint) within 1 minute. However, when we press sw0, sw2, or the air pressure backed to normal, then the error is ceased, and LCD shows the STATUS screen.

- NETWORK screen: in this screen, we can change the WiFi's ssid, password, and MQTT broker's IP address. After changing, if user presses "save" button, these values will be stored in the EEPROM, and a message will appear (ASK_RESTART screen) to ask user to reset the device to apply new settings. If "cancel" button is pressed, then the setting will be ignored, then the LCD shows back the STATUS screen.
- SETTING NETWORK screen: This screen shows selecting the ascii characters to form ssid, password, and/or ip address. To enter this screen, the user presses and hold sw0 for at least 1s. The rotary encoder does select a character, then is confirmed by sw0. To exit the editing, sw0 is pressed and held again for at least 1s.
- ASK_RESTART screen: shows a message which asks users to manually reset the device to take new settings.

In the main while loop, the program checks the screen value to display the corresponding one. After this process, the program continues to check whether any button has been pressed which happened in the interrupt handler or not to take an appropriate action. For example, if sw2 is pressed, then the LCD displays MAIN screen, or if sw0 is pressed within MAIN screen, SETPOINT screen, NETWORK screen, or SETTING NETWORK screen then it confirms the selected option or value.

After checking buttons, the program starts to read co2 sensor via Modbus address 240, holding register number 256. Similarly, humidity is read via Modbus address 241 and holding register number 256. The same Modbus address is for reading temperature but in holding register number 257. The read values of humidity and temperature then are divided by 10 to get the correct values. The fan speed is also controlled by Modbus at address 1, holding register number 0. The value for setting fan speed is from 0 – 1000, therefore, the speed's percentage needs to be multiplied by 10.

In the manual mode, the fan speed is set at a fixed value, it is also a setpoint value. The air pressure is then measured and displayed on STATUS screen. By abstracting, the air pressure is set at a fixed value in auto mode, then the program will adjust the fan speed and measure back the air pressure to make sure the current air pressure

meets the target. If the current air pressure greater than the setpoint, then the device will reduce the fan speed and vice versa.

After reading all sensors, the program checks if the device is connected to MQTT broker, then it sends the values to MQTT in JSON format. If the there is no MQTT connection, then the device will skip this step.

The interrupt handler helps to catch rotary encoder, sw0, and sw2 falling edge events for local UI interactions. In case of MQTT successful connection, throughout the whole program, it consistently subscribes to MQTT topics, listening to messages parsed from MQTT broker in a subscribed topic, which contains the commands for ventilation settings (either manual mode with the value of set fan speed or automatic mode with the value of target pressure). Meanwhile during the whole program, it also publishes a message in an interval of 3 seconds, which indicates the current state of ventilation including measurement of speed, pressure, CO2, relative humidity, and temperature. The published message is then shown on MQTT based controller UI's current state session.

3.1 MQTT

nr	a sequence number that is incremented after each message.
speed	The current fan speed (manual mode)
setpoint	The current controller target value, representing either the fan speed (in manual mode) or the target pressure (in automatic mode).
pressure	Current pressure measurement.
auto	a boolean which indicates the controller mode
error	a boolean which is set to true if the controller can't reach the target pressure within reasonable time
CO2	current CO2 ppm value
rh	current relative humidity

temp	measured temperature.

Message Format:

All messages exchanged between the UI and the controller are formatted in JSON for readability and usability. This ensures that commands and status updates are easily interpreted by both systems.

Operation

+ Controller Settings

Use the controller/settings topic to send messages that configure the behavior of the Ventilation System.

Messages must contain the auto field, determining the operating mode, and another field dependent on the mode (speed for manual mode, pressure for automatic mode)

+ Controller Status

The system publishes status messages to the controller/status topic whenever there are changes in settings or sensor readings.

These messages provide real-time information about the system's state, including sensor measurements and operational parameters.

Subscribers to this topic receive updates on fan speed, target values, pressure, operating mode, errors, CO2 levels, humidity, and temperature.

4 Conclusion

In conclusion, the Ventilation Controller Project represents a significant advancement in the field of indoor environmental control, aiming to redefine the standards of ventilation management in modern living and working spaces. Through the development of a sophisticated system capable of integrating local and remote user interfaces, offering manual and automatic operational modes, and prioritizing stability, efficiency, and user experience.

The software aspect of the project is equally crucial, with the implementation of two operating modes - manual and automatic - providing flexibility to users. Through the local user interface and MQTT messages from a web UI, users can adjust ventilation settings seamlessly, ensuring optimal comfort and efficiency.

In the future, improvements for the Ventilation Controller Project may include enhancing remote control capabilities, integrating with smart home systems, incorporating additional sensors, optimizing energy efficiency, refining the user interface, and implementing advanced data analytics.

5 References

1. Amazon Web Services [online]. Year unknown. What is MQTT? Available at: https://aws.amazon.com/what-is/mqtt/ (Accessed: 16 March 2024)

2. Real Time Automation [online]. Year unknown. An Introduction to Modbus RTU. Available at: https://www.rtautomation.com/technologies/modbus-rtu/ (Accessed: 18 March 2024)

3. Länsikangas Keijo, UART and Modbus – Modbus RTU. 25.2.2024 [Unpublished lecture slides]. (Accessed: 16 March 2024)

4. Länsikangas Keijo, Ventilation controller project specification. 2024, [unpublished project assignment]. Metropolia University of Applied Sciences, assignment given 15 February 2024. (Accessed: 12 March 2024)

5. Solomon Systech, SSD1306 Datasheet by DFRobot – General Description. April 2008. Available at: https://www.digikey.com/htmldatasheets/production/2047793/0/0/1/ssd1306.html#pf6 (Accessed: 19 March 2024)

6. Raspberry Pi Documentation. Year unknown. Raspberry Pi Pico W and Pico WH. Available at: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html (Accessed: 14 January 2024)

7. Raspberry Pi. Year unknown. Raspberry Pi Debug Probe. Available at: https://www.raspberrypi.com/products/debug-probe/ (Accessed: 13 January 2024)