Weather Station

Developed by:

E.Karamanliev

S.Yordanov

A.Asenov

BOSCH - Engineering Center Sofia

Career Camp vol.3

Table of contents

Chapter 1

Overview

This is part of the documentation for the "Weather Station" project.

The project aims to create a functioning weather station using the STM32F429I_DISCO board, a couple of modules and for visualization we use TouchGFX.

Functionality

The Weather Station developed by our team has the following funcitons and features:

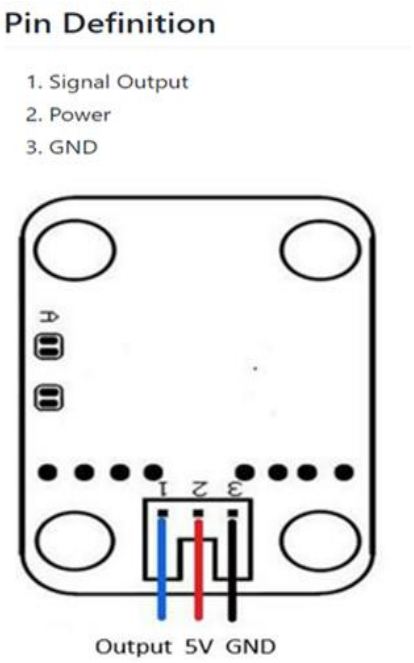
1. Monitoring of different environmental parameters.
    - Temperature outside and inside.
    - Pressure
    - Humidity
    - Ambient Light
    - Carbon Monoxide
2. User Interface (using TouchGFX).
    - An easy-to-understand UI is included in this project with menus to navigate data and settings.
3. Command Line Interface.
    - This system allows a developer to directly influence the UI and Data on the board.
    - It can help with debugging and gathering data. More in the *“Getting Started section”*.
4. Statistics throughout the day and over a period.
    - daily
    - weekly
    - monthly
5. Alarms for high and dangerous readings.

Chapter 2

Module specification

Carbon Monoxide Gas Sensor MQ7 - The MQ7 is a simple-to-use Carbon Monoxide (CO) sensor suitable for sensing CO concentrations in the air. It can detect CO-gas concentrations anywhere from 20 to 2000ppm.

- Power supply needs: **5V**
- Interface type: **Analog**
- Pin Definition:
  - 1- **Output**
  - 2- **VCC**
  - 3- **GND**
- High sensitivity to **carbon monoxide**
- Fast response
- Stable and long life
- Size: **40x20mm**

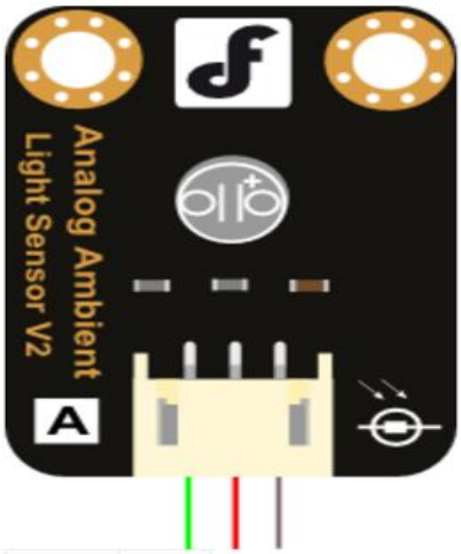


*Before we start working with the gas sensor we need to power it properly using **5V** and **GND** from the microcontroller. Then we need to read the returned analog value using the **ADC**, but since the maximum we can read with it is **3V**, we need to make a voltage divider whose coefficient we use to correctly calculate the return value.*

Ambient Light Sensor SKU_DFR0026 - This module can help you to detect the light density of your enviroment and reflect this information back via an analog voltage signal to your controller.

- Supply Voltage: **3.3V** to **5V**
- Illumination range : **1 Lux** to **6000 Lux**
- Responsive time : **15us**
- Interface: **Analog**
- Size: **22x30mm**

PinOut



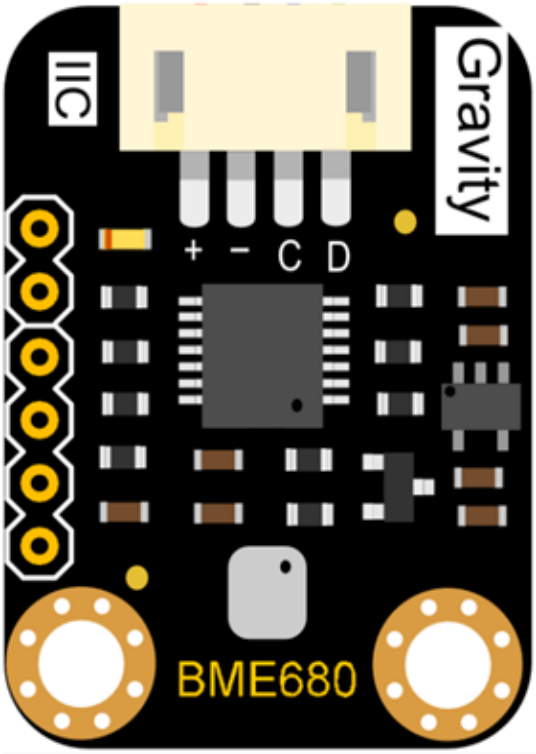
Color	Pin
GREEN	S
RED	VCC
BLACK	GND

*We **power** this sensor in the same way, being careful not to connect the wires incorrectly. The sensor also returns an analog value that we read using an **ADC** (different channel) and convert **the value** into **a percentage.***

BME680 Environmental Sensor -This is a low power gas, pressure, temperature & humidity sensor based on ***BOSCH BME680*** sensor. It is a 4-in-1 multi-functional MEMS environmental sensor which integrates VOC (Volatile Organic Compounds) sensor, temperature sensor, humidity sensor and barometer.

- Input Voltage: **3.3V~5.0V**
- Operating Current: **5mA** (*25mA in VOC*)
- Wire Connector: **Gravity I2C**
- Connector in Reserve: **SPI**
- Temperature Measurement Range: **-40℃ 85℃**
- Temperature Measurement Precision: **±1.0℃**(*0~65 ℃*
- Humidity Measurement Range: **0-100**% *r.H*
- Humidity Measurement Precision: **±3**% r.H.(***20-80%*** *r.H.* ***25 ℃***
- Atmospheric Pressure Measurement Range: **300-1100**hPa
- Atmospheric Pressure Measurement Precision: **±0.6**hPa(***300-1100****hPa,0~65 ℃*
- Module Size: **30 × 22**(*mm*) / **1.18 x0.87**(*inches*).

Board Overview

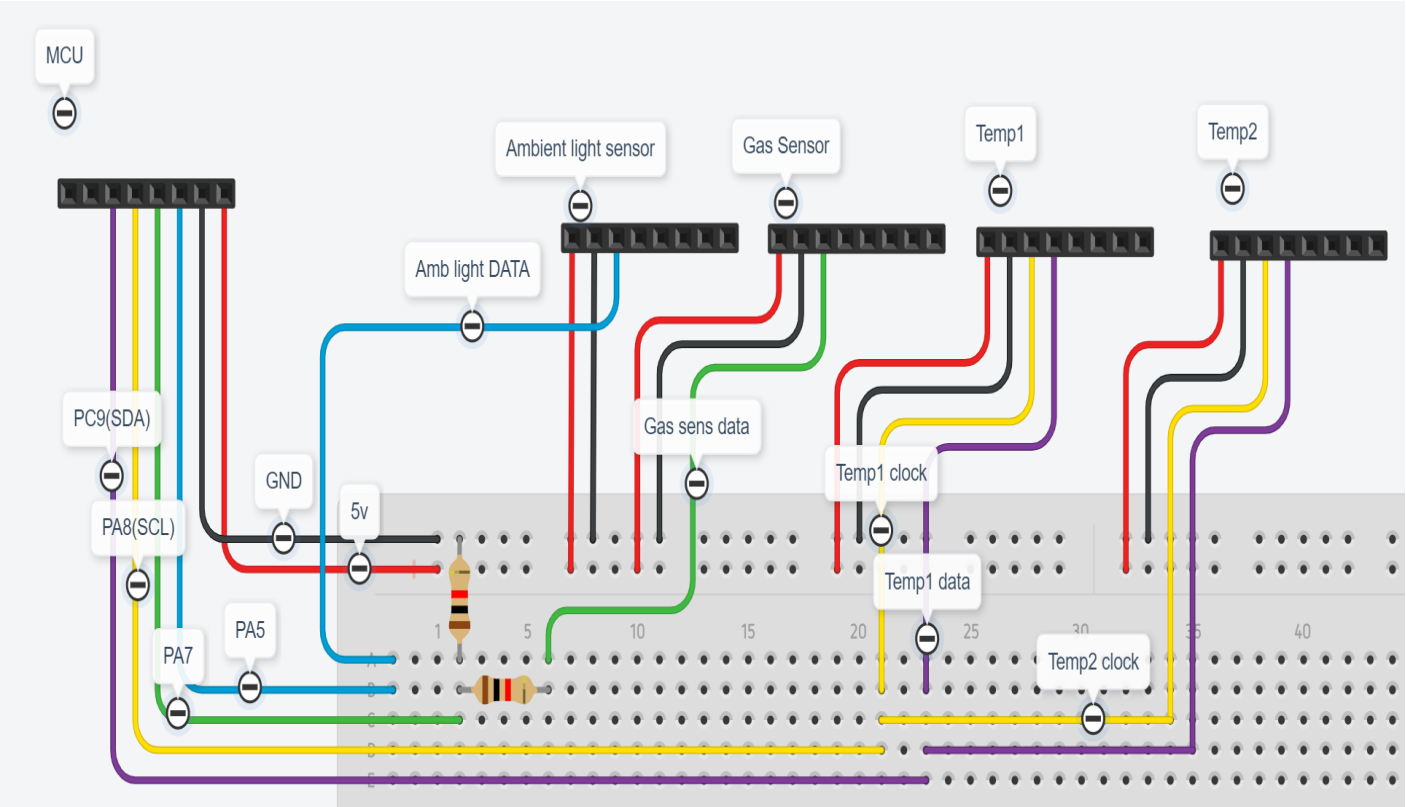


Num	Label	Description
+	VCC	Power Input(3.3~5.0V)
-	GND	Power Ground(0V)
C	SCL	I2C Clock Signal
D	SDA	I2C Data Signal

*After successfully powering the sensor, we have to choose what type of communication we will use **I2C** or **SPI** (the microcontroller supports both types of communication). In our case, we have chosen **I2C,** which means we need **2 additional buses** for clock and for data. We have initialized pins **PA8(SCL)** and **PC9(SDA)** to which we connect the wires from the sensor correctly.*

***NOTE:** We can connect more than one sensor, but its address must be different.*

Connection diagram



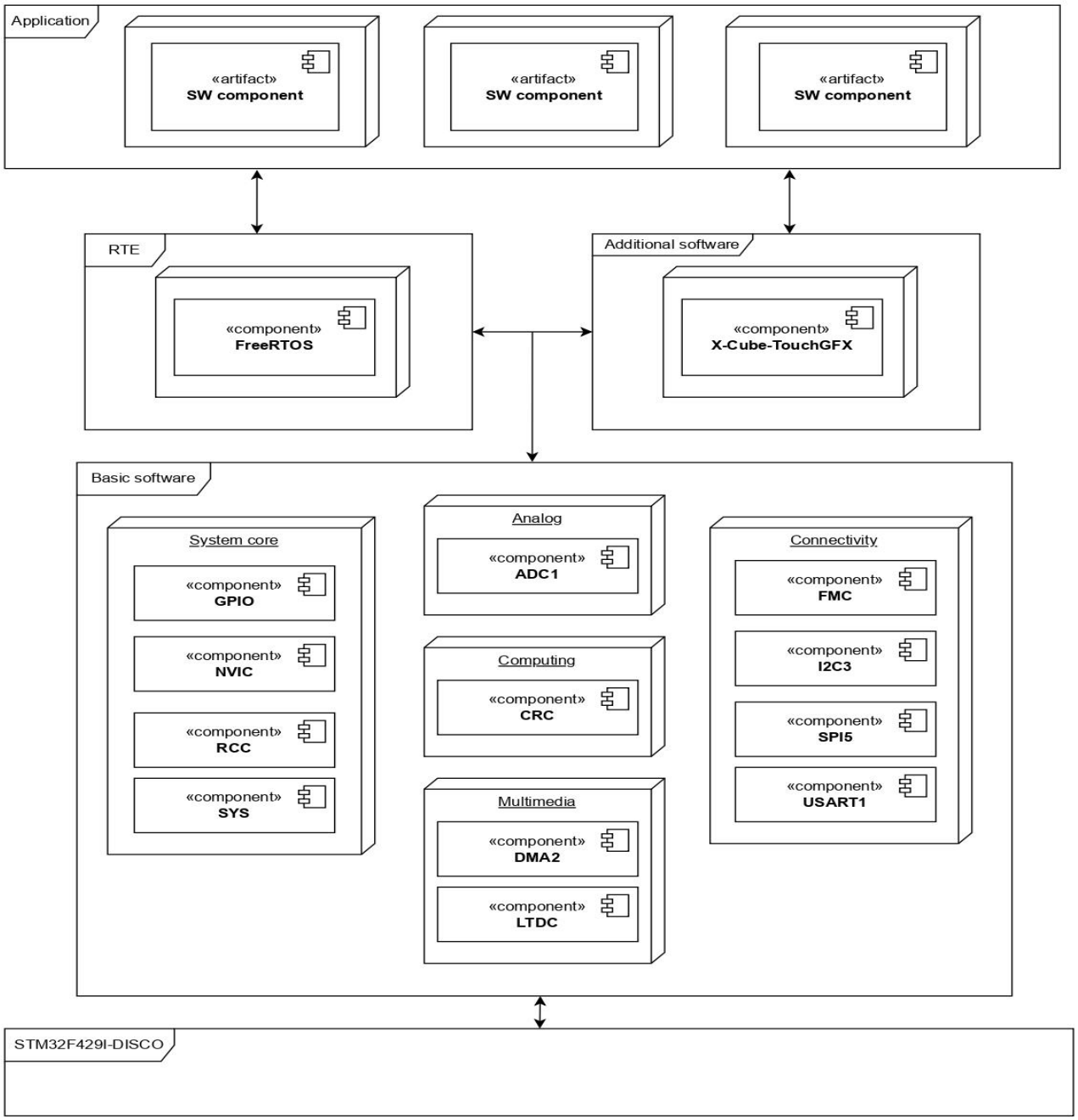
In the end, everything should look like this.

NOTE: Any connection changes must be made with the power off, otherwise unexpected conditions may occur.

The final step is to initialize all the pins needed for the Weather Station to work! In order to do so we will utilize the tools available in the STM32CubeIDE! From there we will enable a UART Communication channel and an ADC Channel with pins: PA5 & PA7

Chapter 3

Software architecture



Architecture describes the conceptual model of software. At the top is the user's code all algorithms, implementations, etc. written by us for the system to

function properly. The operating system used by the system is FreeRTOS which allows multiple programs to execute at the same time. For an interface providing the necessary information to the user in a clear and understandable way, we used TouchGFX. All the components presented below represent the initialized modules without which data collection, processing and presentation would not be possible.

- **GPIO** - stands for general purpose input/output. It is a type of pin found on an integrated circuit that does not have a specific function. While most pins have a dedicated purpose, such as sending a signal to a certain component, the function of a GPIO pin is customizable and can be controlled by the software.

- **NVIC** - The NVIC provides a fast response to interrupt requests, allowing an application to quickly serve incoming events. An interrupt is handled without waiting for the completion of a long instructions sequence. (We use it in CLI to generate an interrupt when the command is entered and needs to be send back through UART)

- **RCC** - The RCC peripheral is used to control the internal peripherals, as well as the reset signals and clock distribution. The RCC gets several internal (LSI, HSI and CSI) and external (LSE and HSE) clocks. They are used as clock sources for the hardware blocks, either directly or indirectly, via the four PLLs (PLL1, PLL2, PLL3 and PLL4) that allow to achieve high frequencies.

- **ADC1** - The analog-to-digital converters allows the microcontroller to accept an analog value like a sensor output and convert the signal into the digital domain.

- **CRC** - The cyclic redundancy check (CRC) is a powerful and easily implemented technique to obtain data reliability. It is used to detect errors in data transmission or storage integrity check, without making corrections when errors are detected.

- **LTDC** - The LTDC on the STM32 microcontrollers is an on-chip LCD display controller that provides up to 24-bit parallel digital RGB signals to interface with various display panels.

- **FMC** - The FMC allows to interface with static-memory mapped external devices such as SRAM, NOR Flash, NAND Flash, SDRAM.

- **I2C3** - is a two-wire serial communication system used between integrated circuits. The I2C is a multi-master, multi-slave, synchronous, bidirectional, half-duplex serial communication bus. It is used for communication between MCU, BME680 and Touchscreen.

- **SPI5** - Serial Peripheral Interface (SPI) is a synchronous serial communication interface, used to communicate between microcontrollers and peripheral devices such as sensors, displays (ili9341) and memory.

- **USART1** - The USART peripheral is used to interconnect STM32 MPU devices with other systems, typically via RS232 or RS485 protocols. In addition, the USART can be used for smartcard interfacing or SPI master/slave operation and supports the Synchronous mode.

Software documentation

FreeRTOS - We will mention some of the features we use and a brief description of them and how they interact.

Create a new task and add it to the list of tasks that are ready to run.

- ```
  BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                          const char * const pcName,
                      const configSTACK_DEPTH_TYPE uxStackDepth,
                          void *pvParameters,
                          UBaseType_t uxPriority,
                          TaskHandle_t *pxCreatedTask
                            );
  ```

Starts the RTOS scheduler. After calling the RTOS kernel has control over which tasks are executed and when.

- ```
  void vTaskStartScheduler( void );
  ```

Creates a binary semaphore, and returns a handle by which the semaphore can be referenced.

- ```
  SemaphoreHandle_t xSemaphoreCreateBinary( void );
  ```

Macro to release a semaphore.

- ```
  xSemaphoreGive( SemaphoreHandle_t xSemaphore );
  ```

Macro to obtain a semaphore.

- **xSemaphoreTake(SemaphoreHandle_t xSemaphore,**
  TickType_t xTicksToWait);

NOTE: *We use semaphores to prevent conflicts during I2C communication and ensure proper sequence of actions.*

Creates a new queue and returns a handle by which the queue can be referenced.

- **QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength,**
  UBaseType_t uxItemSize);

Post an item on a queue.

- **BaseType_t xQueueSend(**
  QueueHandle_t xQueue,
  const void * pvItemToQueue,
  TickType_t xTicksToWait);

Receive an item from a queue.

- **BaseType_t xQueueReceive(**
  QueueHandle_t xQueue,
  void *pvBuffer,
  TickType_t xTicksToWait);

Return the number of free spaces in a queue.

- **UBaseType_t uxQueueSpacesAvailable(QueueHandle_t xQueue);**

NOTE: *We use queues to transfer data between tasks and UI.*

BME680_SensorAPI - We use the library to read and configure the sensor.

Code organization and user functions – All STM32CubeIDE automatically generated functions are exported in separate source file and a corresponding header file.

For reading analog data from MQ7 and Ambient light sensor we use 2 functions:

- void ADC_Select_CH5(SensData_t *);
- void ADC_Select_CH7(SensData_t *);

The functions are intended to change the pin from which ADC reads and write the information in the corresponding structure passed as a parameter.

Command Line Interface

The CLI utilizes its own task to receive and respond to commands.

It will respond either when a command has been entered (by receiving the '\r' symbol) or by exceeding the string limit that is allowed, which is a maximum length of 24 letters.

The CLI works by the following principle:

command received -> check if command is valid -> send to appropriate handler.

Let's check out the code!

In the header file, since we use function pointers to navigate, to move correctly through the arrays of functions and avoid entering inaccessible memory, we use defines.

Our main function is the CLIHandler function.

From there we check if the string is too long and clear the string for the next message.

```
if (*msgIDX >= MAX_ALLOWED_STRING) {
        (error[ERR_MSG_LONG])(huart1);
        clearMSG(msg, msgIDX);
}
```

Then we check for the "**help**" command which displays the current selected menu.

```
if (*msgIDX > 0 && strncmp(msg, "help", *msgIDX) == 0) {
(menu[selectedMenu])(huart1);
clearMSG(msg, msgIDX);
```

This is an example of the default "***help***" menu.

```
void DisplayHelpMenu(UART_HandleTypeDef *huart1) {
        char msg[40];
        sprintf(msg, "\n\rHELP MENU\n\r");
        HAL_UART_Transmit(huart1, (uint8_t*) msg, strlen(msg), 1000);
        sprintf(msg, "1. Data Menu\n\r");
        HAL_UART_Transmit(huart1, (uint8_t*) msg, strlen(msg), 1000);
        sprintf(msg, "2. Date and Time\n\r");
        HAL_UART_Transmit(huart1, (uint8_t*) msg, strlen(msg), 1000);
        sprintf(msg, "3. Alarms Menu\n\r");
        HAL_UART_Transmit(huart1, (uint8_t*) msg, strlen(msg), 1000);
}
```

From there we can navigate the menus by inputting the corresponding number. In our case that would be 1, 2 or 3.

NOTE: *The CLI accepts only number inputs. The only exception being the "help" menu which can be accessed by typing "h" or "help".*

After our next input, if it is not the "help" command, we go to our selected command handler, which takes our input and displays the appropriate action.

```
else {
      (handler[selectedMenu])(&dateTime, SelectedSensor, huart1, msg);
      clearMSG(msg, msgIDX);
```

An example of "**help menu**" handler:

```
void CommandMenuHandler(DateTime_t *dateTime, SensData_t *SelectedSensor,
                UART_HandleTypeDef *huart1, char *msgRec) {
        switch (msgRec[0]) {
        case '1':
                selectedMenu = DISPLAY_DATA_MENU;
                (menu[selectedMenu])(huart1);
                break;
        case '2':
                selectedMenu = DISPLAY_DATETIME;
                (menu[selectedMenu])(huart1);
                break;
        case '3':
                selectedMenu = DISPLAY_ALARMS_MENU;
                (menu[selectedMenu])(huart1);
                break;
        default:
                (error[ERR_MSG_NF])(huart1);
                break;
        }

}
```

From there we further navigate and use commands to display what we have available. For example, we can navigate to the "***Display Data***" menu and from there get the "***Pressure***" value from the data.

Example of "***Get Pressure***" command:

```
void GetPressure(SensData_t *SelectedSensor, char *msg) {
        sprintf(msg, "\n\r Pressure: %dhPa\n\r", SelectedSensor->pressure);
}
```

This is the main functionality of the **CLI.**

However, we also support changing elements of the **UI**!

This is supported in the "***Date Time***" handler, where we setup the **CLI** to wait for an **Input**.

```
case '3':
        changed = SET_DATE_DAY;
        sprintf(msg, "\n\r ENTER DAY:\n\r");
        HAL_UART_Transmit(huart1, (uint8_t*) msg, strlen(msg), 1000);
        clearMSG(msg, NULL);
        break;
```

Then, in the main **CLI** we take the Input and set the value in our structure.

```
if(changed!=0)
{
        int Input = atoi(msg);
        (command[changed])(&dateTime,Input,huart1);
        changed = 0;
        finished = 1;
        clearMSG(msg, msgIDX);
}
```

NOTE: The "atoi" function is from "stdlib.h" library and it is used to convert our input into a number.

Once the input is finished, we take our "**finished**" flag and send it to the **UI model** to indicate a change.

Start the project

Once the components are connected correctly, we are ready to test out the program!

Firstly, you will need a program to flash the board with the code.

Our personal recommendation is to use the **STM32IDE.**

Second, you will have to import the project into the IDE. Then we can finally flash our code on the board.

NOTE: *There may be issues upon importing, linked with directories and file locating. A simple fix would be to* **clean the project** -> **build** -> **no errors** -> **flash**.

If everything is done correctly you should see the screen on the board display the **UI**!

Feel free to interact and see how our **Weather Station** works!

You can use the touchscreen to navigate the menus and settings, but you can also use the

Command Line Interface!

To access the **CLI** you have to know to which port on your **PC** you have connected the board. That is as simple as opening **device manager** on windows -> **Ports**.

Now we can use the built-in console by our **stm32IDE** or something like **PuTTy**.

Once the console is open we need to set the **frequency** and correct **COM** port.

Upon connecting the user can see the list of available commands by typing: **HELP**

Chapter 4

Tests

NOTE: *All the tests are conducted within the hardware specifications provided by the manufacturer!*

Errors and bugs found during testing:

- In case of disturbances or disconnections of any of the BME680 sensors I2C bus goes into BUSY mode which can only be recovered with power reset.

- Auto-dim – we do not have the ability to control it via PWM due to hardware reasons. (We have implemented a software solution)

- I2C conflict between Touchscreen and BME680 (Problem solved with semaphores)

- MQ7 gas sensor calibration – for normal operation of the sensor, according to the datasheet, it must be preheated for a period of 48 hours, the heating cycles adjusted and its resistance calculated in an environment with a CO concentration of 100ppm, which we cannot guarantee.

In order to be able to connect a second BME680 sensor to I2C , we need to change its address and the manufacturer did not provide us software option for this. The address has been changed in hardware by our intervention.

The remaining 2 sensors can be disconnected and an alert will appear for this action.

Alert codes

*Temperature Inside

UNPLUGGED	101
LOW TEMPERATURE	102
HIGH TEMPERATURE	103

*Temperature Outside

UNPLUGGED	201
LOW TEMPERATURE	202
HIGH TEMPERATURE	203

*Humidity

UNPLUGGED	301
LOW HUMIDITY	302
HIGH HUMIDITY	303

*Pressure

UNPLUGGED	401

*Ambient light

UNPLUGGED	501

*Gas sensor

UNPLUGGED	601
HIGH CO	603