Topic:	Weather Station CubeIDE
Author:	PARA
Date:	30.11.2021

0. Before you start

This document will give you an overview of the source code for the example package Weather Station. Before you can work with it you need to set up your working environment as explained in the document “examplepackage_cubeide_gettingstarted”. Make sure you have read this document beforehand and executed all the steps to configure your STM32CubeIDE.

This example is only executable on the 4.3- and 5.0-inch display, since the 3.5-inch display has neither a sensor nor an ESP32-WROOM-32 module.

For a more detailed explanation of how the microcontroller works, please refer to the STM32F429 Reference Manual (RM0090) provided by STMicroelectronics.

1. Introduction

In this example package we will use the BME680 environmental sensor to measure pressure, humidity, temperature and air quality. The sensor is not connected directly to the STM32 microcontroller but to the ESP32-WROOM-32 module. The ESP32 is as well a programmable microcontroller, but it also provides WIFI and Bluetooth connectivity. We will program the ESP32 module to retrieve the sensor data from the BME680. The STM32 will ask the ESP32 for this data and get this via UART. Parallel, the ESP32 will act as a Webserver. A WiFi capable device will be able to connect to the board directly and see the sensor data in any web browser.

The Code for the ESP32 Module is provided in the .zip-File on the www.21display.com Website and doesn’t need to be copied in the CubeIDE Project.

2. Additional Requirements for this Example

2.1 Additional required Hardware

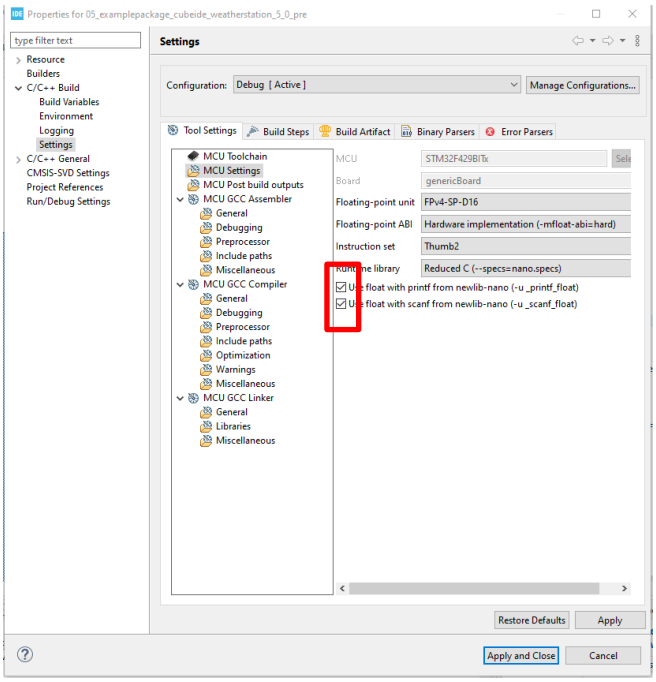
- FT232 USB-to-TTL Serial Converter
- MiniBridge Female 6-pin Connector by ERNI (for an easy connection to the 1.27 mm programming pins of the ESP32 module)

2.2 Additional configurations of CubeIDE

In order to use the provided code, you have to activate 2 settings in CubeIDE Project properties. Click on *Project -> Properties.* Expand C/C++ Build and click on MCU Settings on the Tool Settings tab. Now set the two checkboxes:

- Use float with printf from newlib-nano (-u _printf_float)
- Use float with scanf from newlib-nano (-u _scanf_float)

On the next page you can se a picture of these checkboxes.

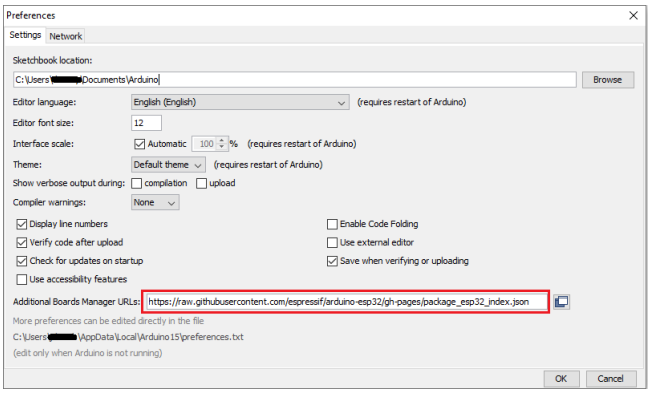


2.3 Arduino IDE – ESP32

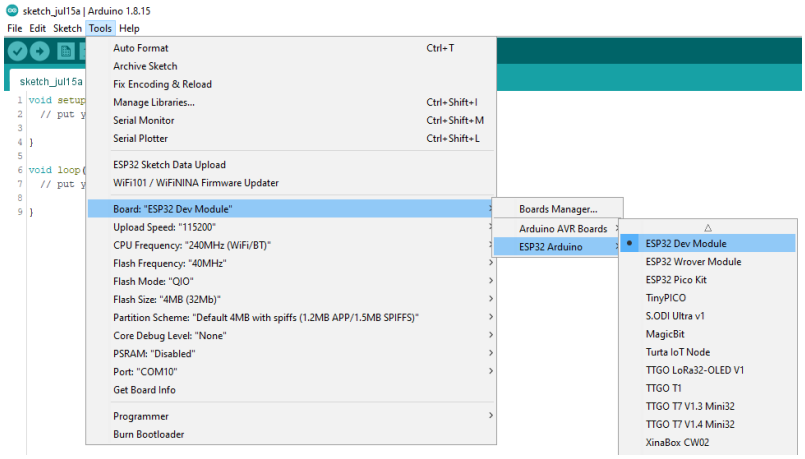
In this example we also program the ESP32 module of our display board. For programming we use the Arduino IDE. Therefore, you need to **download the latest version** of this free software (for this example the version 1.8.15 was used). Some adjustments need to be done, so you can program the ESP32 module with this IDE. Open the Arduino IDE and open the preferences window with *File -> Preferences*. Here, you have to add the URL for the ESP32 board manager.

Write
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

at “Additional Boards Manager URLs:” and click *OK*.



After restarting the IDE you need to open the *Boards Manager* with *Tools -> Board -> Boards Manager…* and install ***esp32*** by Espressif Systems. Now you should be able to select the ESP32 module as your board. Choose “ESP32 Dev Module”, like in the following picture. For this Example we used Version 1.0.6 of the ESP Boards Package.



To host the webpage for external devices, we will store a HTML file and an image on the built-in flash of the ESP32 module. For this you need to install a plugin for your Arduino IDE. An instruction for installing this plugin can be found here: [Install ESP32 Filesystem Uploader in Arduino IDE | Random Nerd Tutorials].
Now you can upload external files to the SPIFFS filesystem of the ESP32 module. This is done by clicking *Tools->ESP32 Sketch Data Upload*. The files, that shall be uploaded, have to be in a directory called “data” in the same location as your Arduino sketch file.

2.4 Arduino IDE – Webserver Libraries

As we are going to implement an asynchronous webserver with the ESP32 module, so that you can monitor the sensor data remotely, two additional libraries are required which are not accessible via the Arduino Library Manager. They have to be added manually.

We need the AsyncTCP library and the ESPAsyncWebServer library by me-no-dev which can be downloaded for free on GitHub. Follow the subsequent links and download each library code as a ZIP file.

GitHub - me-no-dev/AsyncTCP: Async TCP Library for ESP32

GitHub - me-no-dev/ESPAsyncWebServer: Async Web Server for ESP8266 and ESP32

Extract each ZIP file into */”Arduino-installation-folder”/libraries/*. This directory should now contain the two directories *AsyncTCP-master* and *ESPAsyncWebServer-master*.

2.5 Arduino IDE- Bosch BSEC

Bosch provides an API to access its BME680 sensor. The BSEC Software Library. This can be installed inside Arduino IDE. Open the Library Manager and search for “BSEC Software Library” by Bosch Sensortec. Install the latest version. The IDE won’t find the library files at the beginning. At first, you need to modify the *platform.txt* file of the ESP32 board. The file should be located at:
C:\Users*username*\AppData\Local\Arduino15\packages\esp32\hardware\esp32*version_number*\

You need to edit the line which specifies the parameter *recipe.c.combine.pattern*. This should be line 88. Replace this line with:

```
recipe.c.combine.pattern="{compiler.path}{compiler.c.elf.cmd}"
{compiler.c.elf.flags} {compiler.c.elf.extra_flags} -Wl,--start-group
{object_files} "{archive_file_path}" {compiler.c.elf.libs} {build.extra_libs}
{compiler.libraries.ldflags} -Wl,--end-group -Wl,-EL -o
"{build.path}/{build.project_name}.elf"
```

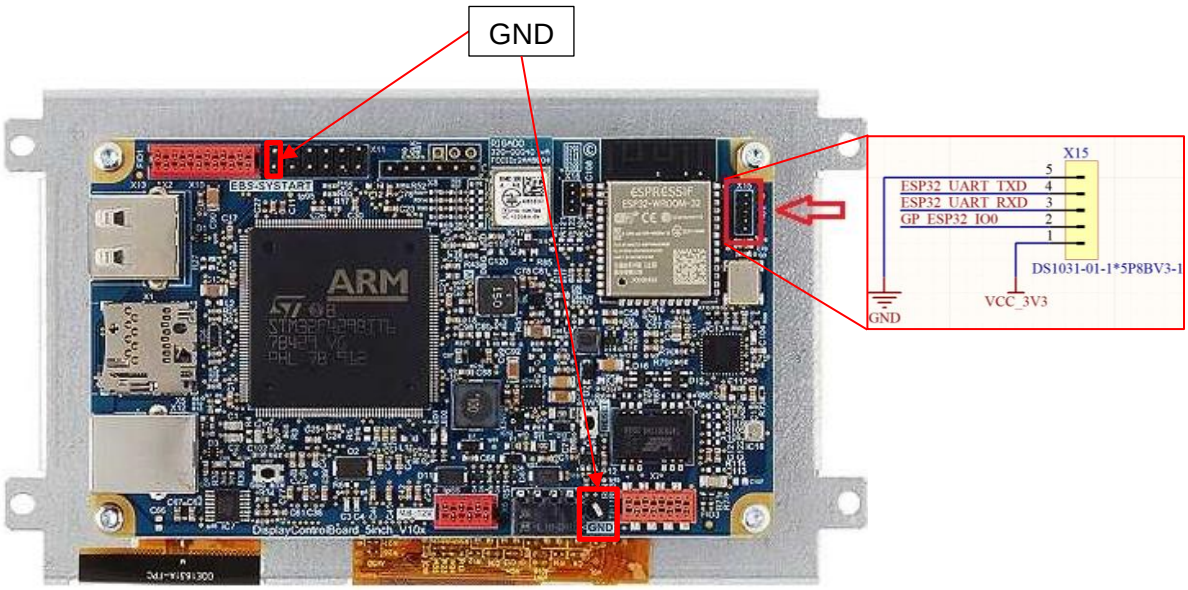
After a restart of your Arduino IDE, everything should be ready to work.

3. Upload Routine

For uploading the whole project, a certain procedure has to be followed. At first, you need to erase the STM32 chip. You can do this within your STM32CubeProgrammer. Open the STM32CubeProgrammer and connect to the display board with the ST-LINK/V2 debugger and with its power supply. Connect to the ST-Link and click on the Erasing & programming button in the left menu. Now click the tab *Erase flash memory* and click *Full chip erase.*

Now you can flash the ESP32 module. For this you need to connect the USB-to-Serial converter with the 1.27mm pin connector to the right of the ESP32. The location and the pin assignments can be found on the pictures below. Pin 1 is marked with a little dot. Therefore, the bottom one is Pin 1. In order to connect the display board correctly to the USB-to-Serial converter, you need to connect three Pins (GND, TX and RX). The other Pins at the converter can be left floating. Note: Connect the RX pin of the connector to the TX pin of the USB-to-Serial converter, the TX pin of the connector to the RX pin of the USB-to-Serial converter and one of the ground pins.

The GP_ESP32_IO0 pin is important to the flashing process since this pin defines the boot mode of the ESP32 module. For flashing, this pin must be connected to ground **before** you turn on the power. This will signal the ESP32 to prepare for an upload. Then you can upload the external data to the SPIFFS and the sketch to the microcontroller in the Arduino-typical way. Between uploading the external data and flashing the sketch you need to turn the power off and on again.



After uploading everything to the ESP32 turn off the power, disconnect the GP_ESP32_IO0 pin from ground and turn the power on again.

Now, you can program and flash the STM32 microcontroller as usual.

4. Explanation of Example Code

4.1 STM32 Code

4.1.1 Main function

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_LTDC_Init();
  MX_I2C1_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_USART6_UART_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  /* Declaration of variables */
  Buffer_e activeBuffer;
  weather_data_struct weatherdata = {0};

  /* Initialize the ringbuffer */
  uart_init();

  /* Backlight enable */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_9, GPIO_PIN_SET);
  if (BACKLIGHT_EN) {
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_SET);
  }
  /* ESP32 enable */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
  /* Display the logos and save active buffer */
  activeBuffer = Display_Startup_Sequence();

  /* Get weatherinterface ip and weather data from esp32 */
  get_ip_address_esp32(&weatherdata);
  get_all_data_esp32(&weatherdata);

  /* Draw the weather interface in the background */
  draw_weather_interface(weatherdata.ip_address, activeBuffer);

  /* Draw the weather data in the foreground */
  draw_weather_data(weatherdata, activeBuffer);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* Get data and display it */
    get_all_data_esp32(&weatherdata);
    draw_weather_data(weatherdata, activeBuffer);
    HAL_Delay(1000);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

We will start our walk through the code at the main function since this gives a perfect overview of the steps that we will take. At the beginning, the hardware abstraction layer (HAL) and the system clocks are initialized. Then all the peripherals are initialized. A few variables are declared for later use.

`uart_init();` initializes the UART functionality and starts the receive interrupt functions.

The 5.0-inch displays need the pin PH6 to be set to high in order to enable the backlight. This is recognized by the define BACKLIGHT_EN which is defined in the global_Display_Touch_HAL.h file. After that, the display startup sequence, consisting of filling the buffers/layers with the color white and displaying our 2 logos, is started.

Now the IP address of the ESP32 and the first set of weather data is retrieved. The weather interface is drawn in the background layer including the IP address. Now the weather data is drawn in the foreground layer and the function jumps into the main loop.

In the main loop, the weather data is retrieved every second and displayed.

4.1.2 draw_weather_interface

```
void draw_weather_interface(char* ip_addr, Buffer_e nextBuffer){
#ifdef DISPLAY_5_INCH
  /* Buffer_1 */
  DMA2D_Fill_Color(0xFFFF, Layer_1, nextBuffer, false);
  DMA2D_Fill_Color(0x7FFF, Layer_2, nextBuffer, false);
  DMA2D_Draw_Image((HDP-Y21Display.width)/2, (VDP-Y21Display.height)/2, Y21Display.width, Y21Display.height,
      0x7F, DMA2D_REPLACE_ALPHA, (uint32_t)Y21Display.data, CM_ARGB8888, Layer_1, nextBuffer, false);
  //temperature box
  DMA2D_Draw_Rectangle(0, 0, HDP*2/5, VDP/2, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Temperature:", 10, 15, 0x8000, &courier_new_23, Layer_1, nextBuffer, false);
  //humidity box
  DMA2D_Draw_Rectangle(0, VDP/2-3, HDP*2/5, VDP/2 -27, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Humidity:", 10, VDP/2 +15, 0x8000, &courier_new_23, Layer_1, nextBuffer, false);
  //pressure box
  DMA2D_Draw_Rectangle(HDP*2/5-3, 0, HDP*3/5+3, VDP*2/3, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Pressure:", HDP*2/5 +10, 15, 0x8000, &courier_new_23, Layer_1, nextBuffer, false);
  DMA2D_write_string("approx. Altitude:", HDP*2/5 +30, VDP/2, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
  //gas resistance box
  DMA2D_Draw_Rectangle(HDP*2/5-3, VDP*2/3-3, HDP*3/5+3, VDP*1/3 -27, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Gas Resistance:", HDP*2/5 +10, VDP*2/3 +15, 0x8000, &courier_new_23, Layer_1, nextBuffer, false);

  //IP-Address
  DMA2D_write_string("IP:", 10, VDP-28, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
  DMA2D_write_string(ip_addr, 50, VDP-28, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
#endif
#ifdef DISPLAY_4_3_INCH
  /* Buffer_1 */
  DMA2D_Fill_Color(0xFFFF, Layer_1, nextBuffer, false);
  DMA2D_Fill_Color(0x7FFF, Layer_2, nextBuffer, false);
  DMA2D_Draw_Image((HDP-Display21_Logo_Y_43Inch.width)/2, (VDP-Display21_Logo_Y_43Inch.height)/2, Display21_Logo_Y_43Inch.width, Display21_Logo_Y_43Inch.height,
      0x7F, DMA2D_REPLACE_ALPHA, (uint32_t)Display21_Logo_Y_43Inch.data, CM_RGB565, Layer_1, nextBuffer, false);
  //temperature box
  DMA2D_Draw_Rectangle(0, 0, HDP*2/5, VDP/2 - 10, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Temperature:", 10, 15, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
  //humidity box
  DMA2D_Draw_Rectangle(0, VDP/2-3-10, HDP*2/5, VDP/2 -27+10, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Humidity:", 10, VDP/2 - 10 +15, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
  //pressure box
  DMA2D_Draw_Rectangle(HDP*2/5-3, 0, HDP*3/5+3, VDP*2/3 - 20, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Pressure:", HDP*2/5 +10, 15, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
  DMA2D_write_string("approx. Altitude:", HDP*2/5 +30, VDP/2-20, 0x8000, &courier_new_14, Layer_1, nextBuffer, false);
  //gas resistance box
  DMA2D_Draw_Rectangle(HDP*2/5-3, VDP*2/3-3-20, HDP*3/5+3, VDP*1/3 -27+20, 0xCF07, 6, Layer_1, nextBuffer);
  DMA2D_write_string("Gas Resistance:", HDP*2/5 +10, VDP*2/3 +15-20, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);

  //IP-Address
  DMA2D_write_string("IP:", 10, VDP-28, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
  DMA2D_write_string(ip_addr, 50, VDP-28, 0x8000, &courier_new_15, Layer_1, nextBuffer, false);
#endif
}
```

This function uses the basic drawing functions, introduced in the example package *Drawing Text and Images*, to create the background for the display. In this example we use the two-layer mode for the display. The background is drawn on Layer 1, since this one lays behind Layer 2. The data is drawn on Layer 2, as you will see later.

Note: The display size determines the size and position of the interface by a `#ifdef`.

4.1.3 get_all_data_esp32

```
47  HAL_StatusTypeDef get_all_data_esp32(weather_data_struct* w_data) {
48
49    HAL_StatusTypeDef ret;
50    uint8_t buf[4];
51    uint8_t cmd = 1;
52    memset(buf,0,4);
53
54    /* Reset ringbuffer to recv new data */
55    reset_ringbuffer(COM3);
56    ret = HAL_UART_Transmit(&huart3, &cmd, 1, 100);
57    if(ret != HAL_OK)
58      return ret;
59    /* Wait shortly for Interrupt filling the buffer */
60    HAL_Delay(200);
61    uart_Receive4Byte(COM3, buf, BIG_ENDIAN_T);
62    float_bytes.bytes[0]=buf[0];
63    float_bytes.bytes[1]=buf[1];
64    float_bytes.bytes[2]=buf[2];
65    float_bytes.bytes[3]=buf[3];
66    w_data->temperature = float_bytes.floatVal;
67    uart_Receive4Byte(COM3, buf, BIG_ENDIAN_T);
68    float_bytes.bytes[0]=buf[0];
69    float_bytes.bytes[1]=buf[1];
70    float_bytes.bytes[2]=buf[2];
71    float_bytes.bytes[3]=buf[3];
72    w_data->humidity = float_bytes.floatVal;
73    uart_Receive4Byte(COM3, buf, BIG_ENDIAN_T);
74    float_bytes.bytes[0]=buf[0];
75    float_bytes.bytes[1]=buf[1];
76    float_bytes.bytes[2]=buf[2];
77    float_bytes.bytes[3]=buf[3];
78    w_data->pressure = float_bytes.floatVal;
79    uart_Receive4Byte(COM3, buf, BIG_ENDIAN_T);
80    float_bytes.bytes[0]=buf[0];
81    float_bytes.bytes[1]=buf[1];
82    float_bytes.bytes[2]=buf[2];
83    float_bytes.bytes[3]=buf[3];
84    w_data->approx_altitude = float_bytes.floatVal;
85    uart_Receive4Byte(COM3, buf, BIG_ENDIAN_T);
86    float_bytes.bytes[0]=buf[0];
87    float_bytes.bytes[1]=buf[1];
88    float_bytes.bytes[2]=buf[2];
89    float_bytes.bytes[3]=buf[3];
90    w_data->gas_resistance = float_bytes.floatVal;
91
92    return HAL_OK;
93  }
```

After the command is sent to the ESP32, we immediately get the response, which is stored in the UART ring buffer until it is read. This reading of the response happens in the function *get_all_data_esp32()*. Since the individual values are floats, we read the four bytes, combine them back to a float by using the *union* data type and store them in a struct for the whole sensor data. This is done for the parameters temperature, humidity, pressure, approximate altitude and gas resistance.

4.1.4 draw_weather_data

```
void draw_weather_data(weather_data_struct w_data, Buffer_e nextBuffer){
  char temp[12] = {0};
#ifdef DISPLAY_5_INCH
  //temperature
  sprintf(temp, "%4.2f %cC", w_data.temperature, 0x7f);
  DMA2D_Draw_FilledRectangle(0x7FFF, 20, VDP*1/8, HDP*2/5-50, VDP*2/8, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP/5 -25*4, VDP/5, 0x8000, &arial_56_deg, Layer_2, nextBuffer, true);

  //humidity
  sprintf(temp, "%2.0f %%", w_data.humidity);
  DMA2D_Draw_FilledRectangle(0x7FFF, 20, VDP*5/8, HDP*2/5-50, VDP*2/8, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP/5 -arial_56.chars->image->width*2, VDP*2/3, 0x8000, &arial_56, Layer_2, nextBuffer, true);

  //pressure
  sprintf(temp, "%6.2f hPa", w_data.pressure);
  DMA2D_Draw_FilledRectangle(0x7FFF, HDP*2/5+20, VDP/5, HDP*3/5-50, VDP*1/6, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP*7/10 -arial_56.chars->image->width*6, VDP/5, 0x8000, &arial_56, Layer_2, nextBuffer, true);

  //approx altitude
  sprintf(temp, "%3.0f m", w_data.approx_altitude);
  DMA2D_Draw_FilledRectangle(0x7FFF, HDP*2/5+20+13*20, VDP/2-10, HDP/4-10, VDP/6-5, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP*2/5 +30+13*20, VDP/2, 0x8000, &courier_new_23, Layer_2, nextBuffer, true);

  //gas resistance
  sprintf(temp, "%5.2f kOhm", w_data.gas_resistance);
  DMA2D_Draw_FilledRectangle(0x7FFF, HDP*2/5+20, VDP*9/12, HDP*2/5, VDP*1/6, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP*7/10 -courier_new_23.chars->image->width*6, VDP*3/4+30, 0x8000, &courier_new_23,
      Layer_2, nextBuffer, true);
#endif
#ifdef DISPLAY_4_3_INCH
  //temperature
  sprintf(temp, "%4.2f %cC", w_data.temperature, 0x7f);
  DMA2D_Draw_FilledRectangle(0x7FFF, 20, VDP*1/8, HDP*2/5-50, VDP*2/8, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP/5 -arial_28_deg.chars->image->width*7, VDP/5, 0x8000, &arial_28_deg, Layer_2, nextBuffer, true);

  //humidity
  sprintf(temp, "%2.0f %%", w_data.humidity);
  DMA2D_Draw_FilledRectangle(0x7FFF, 20, VDP*5/8, HDP*2/5-50, VDP*2/8, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP/5 -arial_28_deg.chars->image->width*2, VDP*2/3, 0x8000, &arial_28_deg, Layer_2, nextBuffer, true);

  //pressure
  sprintf(temp, "%6.2f hPa", w_data.pressure);
  DMA2D_Draw_FilledRectangle(0x7FFF, HDP*2/5+20, VDP/5, HDP*3/5-50, VDP*1/6, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP*7/10 -arial_28_deg.chars->image->width*6, VDP/5, 0x8000, &arial_28_deg, Layer_2, nextBuffer, true);

  //approx altitude
  sprintf(temp, "%3.0f m", w_data.approx_altitude);
  DMA2D_Draw_FilledRectangle(0x7FFF, HDP*3/4, VDP/2-20, HDP/4-10, VDP/6-5, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP*2/5 +courier_new_15.chars->image->width*13, VDP/2 -20, 0x8000, &courier_new_15, Layer_2, nextBuffer, true);

  //gas resistance
  sprintf(temp, "%5.2f kOhm", w_data.gas_resistance);
  DMA2D_Draw_FilledRectangle(0x7FFF, HDP*2/5+20, VDP*9/12, HDP*2/5, VDP*1/6, Layer_2, nextBuffer, true);
  DMA2D_write_string(temp, HDP*7/10 -courier_new_15.chars->image->width*6, VDP*3/4+30-25, 0x8000, &courier_new_15,
      Layer_2, nextBuffer, true);
#endif
}
```

This function draws the retrieved values from the sensor as a string. Note, that this time we draw on Layer 2 which lies on top of Layer 1. The color format used in the two-layer mode is ARGB1555, therefore the color 0x7FFF is white but transparent. This is necessary, because otherwise we would cover the background. Only the strings for the sensor value are opaque.

4.2 ESP32 code

```
float temperature;  // deg Celsius
float humidity;     //%
float pressure;     //hPa
float approx_altitude;  //meters
float gas_resistance;   //kOhm
float breath_voc; //ppm
float iaq_value;
float co2_equivalent; //ppm

// Replace with your own network credentials
const char* apSSID = "Weather Station";
const char* apPASSWORD = "";  // only needed for key protected network
IPAddress apIP(192, 168, 0, 1);

AsyncWebServer server(80);
```

At the beginning of the Arduino code for the ESP32, we declare some variables that are needed. At first, we have the floats for the individual sensor values. Afterwards, we define the parameters for our webserver. The name of the WiFi-network generated by the ESP32 module is here called “Weather Station”. The password string is kept empty, because we don’t

establish a key protected network. But both strings can be changed as you like. The IP address for the webpage is stored in an *IPAddress* data type. We chose 192.168.0.1, but again, you can change this if you want to. At last, we declare an asynchronous Webserver object which listens on port 80.

```
void setup() {
  Serial.begin(115200);
  hspi.begin();
  iaqSensor.begin(15, hspi);

  bsec_virtual_sensor_t sensorList[10] = {
    BSEC_OUTPUT_RAW_TEMPERATURE,  // iaqSensor.rawTemperature
    BSEC_OUTPUT_RAW_PRESSURE, // iaqSensor.pressure
    BSEC_OUTPUT_RAW_HUMIDITY, // iaqSensor.rawHumidity
    BSEC_OUTPUT_RAW_GAS,  // iaqSensor.gasResistance
    BSEC_OUTPUT_IAQ,  // iaqSensor.iaq
    BSEC_OUTPUT_STATIC_IAQ, // iaqSensor.staticIaq
    BSEC_OUTPUT_CO2_EQUIVALENT, // iaqSensor.co2Equivalent
    BSEC_OUTPUT_BREATH_VOC_EQUIVALENT,  // iaqSensor.breathVocEquivalent
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE,  // iaqSensor.temperature
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY, // iaqSensor.humidity
  };
  iaqSensor.updateSubscription(sensorList, 10, BSEC_SAMPLE_RATE_LP);

  SPIFFS.begin();
  WiFi.mode(WIFI_AP);
  WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
  WiFi.softAP(apSSID); //open network; to set up pre-shared key protected network: WiFi.softAP(appSSID, apPASSWORD)
```

Let's take a look at the setup function. It starts with setting up the Serial connection which is Arduino's name for the UART connection to the STM32. Then we initialize the SPI pins, since the BME680 sensor is using this interface to communicate, and the sensor object *iaqSensor*. The SPIFFS filesystem also needs to be started, because the HTML page and an image for the page are stored there and need to be accessed.
Next, we configure our access point with the IP address, we specified before and then open the network with the name stored in *apSSID*. You could also add a password.

```
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/webpage.html", "text/html");
  });
  server.on("/ebssystart_logo", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/ebssystart_logo.jpg", "image/jpg");
  });
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(temperature));
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(humidity));
  });
  server.on("/pressure", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(pressure));
  });
  server.on("/altitude", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(approx_altitude));
  });
  server.on("/gasResistance", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(gas_resistance));
  });

  server.on("/co2equivalent", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(co2_equivalent));
  });
  server.on("/iaq", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(iaq_value));
  });
  server.on("/bVOC", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", String(breath_voc));
  });
  //Start server
  server.begin();
}
```

The second part of the setup function configures the webserver. At first, we define that when you type in the predefined IP address in a web browser, which means a "GET /"-request, the

HTML file in the SPIFFS is send to the user. In the HTML file, a java script will send every second multiple GET requests, asking for a new temperature, humidity, … value. These requests will be answered by sending the respective data. After configuration, we start the server.

Now, let's take a look at the loop function.

At the beginning, the BME680 sensor is asked if there is new data. If not, we start again at the top. If there is new data, we store the values in the respective variables.

```
void loop() {
  if (iaqSensor.run()) { // If new data is available
    return;
  }
  temperature = iaqSensor.temperature;  // deg Celsius
  humidity = iaqSensor.humidity;  //percent
  pressure = iaqSensor.pressure/100.0;  //hPa
  approx_altitude = calcAltitude(pressure, 1013.25);  // meters
  gas_resistance = iaqSensor.gasResistance/1000.0;  //kOhm
  co2_equivalent = iaqSensor.co2Equivalent;  //ppm
  iaq_value = iaqSensor.iaq;
  breath_voc = iaqSensor.breathVocEquivalent;  //ppm
```

The second part of the loop function manages the UART communication with the STM32. If the ESP32 received an input, it responds depending on the value of the byte.

```
  if(Serial.available()>0){
    input = Serial.read();
    switch(input){
      case 0x00:
        Serial.write(0xAA);
        break;
      case 0x01:
        float_bytes.floatVal = temperature;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = humidity;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = pressure;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = approx_altitude;
        Serial.write(float_bytes.bytes, 4);
        float_bytes.floatVal = gas_resistance;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x02:
        float_bytes.floatVal = temperature;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x03:
        float_bytes.floatVal = humidity;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x04:
        float_bytes.floatVal = pressure;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x05:
        float_bytes.floatVal = approx_altitude;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x06:
        float_bytes.floatVal = gas_resistance;
        Serial.write(float_bytes.bytes, 4);
        break;
      case 0x07:
        Serial.println(WiFi.softAPIP());
        break;
    }
  }
}
```

5. Ideas for Exercise Project

Here, we want to give you some suggestions how you could modify our example code and make your own little project.

You could add little icons to the weather interface, which change depending of the values e.g., of the temperature. So that you could graphically indicate, if it is warm, cold, wet, …