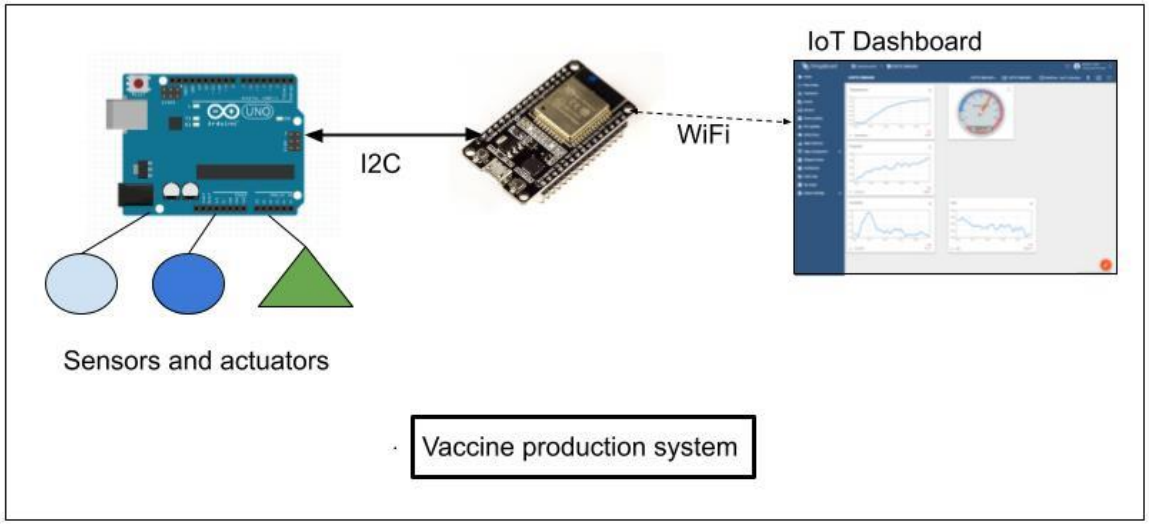
BIOREACTOR CONNECTIVITY OVERVIEW

This is a quick guide to illustrate some of the connectivity functionality that you will need to make use of in the ENGF0001 Bioreactor project. The idea is that you can use this information to experiment with parts of the project and, from there, build up the required features. You'll still need to learn some things yourself but the information here will help you to try stuff out.



Essentially you can think about the system as a series of hops, firstly from an Arduino to the ESP32. Communication between these two devices takes place over a low-level, wired link. From there, the ESP32 is connected to an access point via Wi-Fi and then through a network of servers (the Internet, a network of networks) to a server on which a platform such as Thingsboard runs. On top of this, the communication between the ESP32 and the platform takes place. The Wi-Fi connection has to be established first, but then your program will connect "end-to-end" (it's an abstraction – see lecture material for CS students) from your EPS32 to the remote server. It is independent from the language used for the implementation of the platform.

In the following sections you will implement a basic implementation of the steps required for transferring data from the Arduino Uno to Thingsboard via the ESP32:

1. Establish I2C communication between the Arduino Uno and the ESP32.
2. Connect the ESP32 to a WiFi network.
3. Write code for the ESP32 to send randomly generated values to the Thingsboard demo server at `demo.thingsboard.io`

This will provide you with the end-to-end communications mechanisms, you will need to extend and adapt the materials to deliver your bioreactor software.

I2C COMMUNICATION BETWEEN ARDUINO AND ESP32

There are a number of different protocols that we can choose from for the communication between the Arduino and the ESP32. In this section, we will use I2C (pronounce "I-squared-C) because it is straightforward and it will satisfy the requirements for this part of the bioreactor project. It's up to you what you use.

LEVEL SHIFTING

The Arduino Uno operates at 5V whereas the ESP32 operates at 3.3V. This means that we cannot directly connect the two devices with wires. Connecting the pins on the ESP32 to a 5V signal will damage the board. This is the purpose of the Bidirectional Logic Level Shifter (BLLS) that you have been given. The BLLS converts, or shifts, the current from 5V to 3V, and vice versa. The pins on the Arduino are connected to the high-voltage side and the pins on the ESP32 will be connected to the low-voltage side.

Figure 1 Bidirectional Logic Level Shifter

To connect the 3 devices together, you'll need some jumper wires (4 female-to-male and 4 female-to-female wires). You must connect the ground and voltage from the Arduino to the high-voltage side of the BLLS. Similarly, you must connect the same pins on the low voltage side of the board to the TTGO. For I2C, you also need to connect the data and clock pins via the BLLS. You can use any of the data pins on the BLLS (A1-A4 and B1-B4), but you must match pins on the high- and low-voltage sides: A1 must be connected to B1, A2 must be connected to B2 and so on.

The connections you need are shown in Figure 2 and described in Table 1. You may connect the devices together using a breadboard, if you have one. The meaning of the pin names SDA and SCL will be described later on.

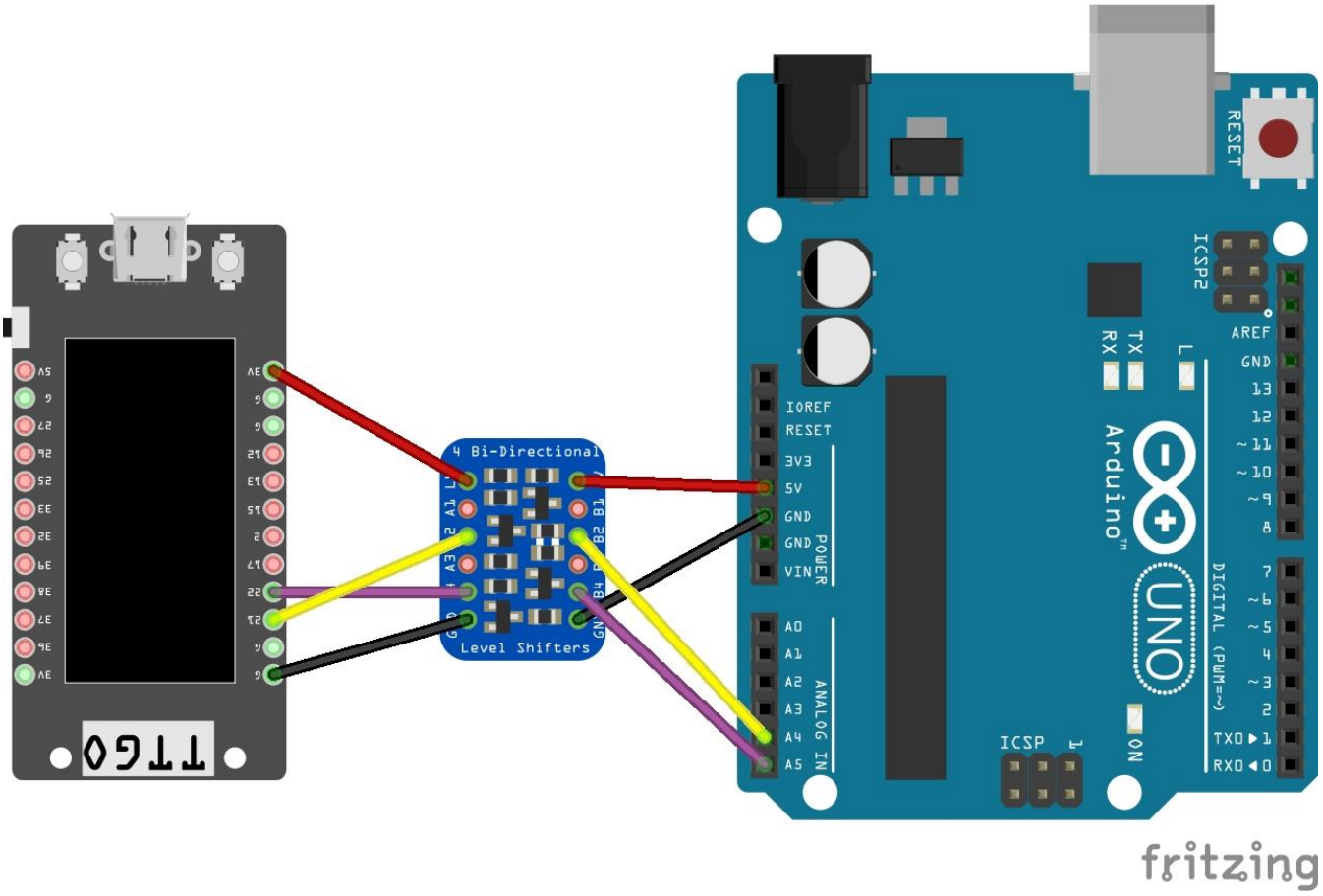


Figure 2 - Arduino, BLLC, TTGO hook-up diagram

Table 1 - Pin mapping

	TTGO	BLLC (3V)	BLLC (5V)	Arduino
Power	3V	LV	HV	5V
Ground	G	GND	GND	GND
SDA	21	A2	B2	A4
SCL	22	A4	B4	A5

I2C

In this section we will describe how to use I2C between the Arduino and the ESP32. I2C is a simple communications protocol involving one device acting as a master and one or more slaves. The master controls when each slave will communicate; slaves cannot communicate with each other. Master and slaves are connected by a bus comprised of 2 wires. The SDA wire is used for transmitting data and the clock (SCL) is used to synchronise communications between the parties. You have already connected the SDA and SCL wires between your devices.

An I2C library, the Wire library, already exists for both the Arduino Uno and the ESP32 so your job is to understand how to use it to accomplish your task.

Assuming, the ESP32, a TTGO, is the master and the Arduino Uno is the slave. The message sequence is:

1. The master requests data from the slave
2. The slave sends data.
3. The master reads the data from the slave.

This is known as the Master Reader / Slave Sender configuration. Follow the specially adapted example below and get this working before you move on.

MASTER READER / SLAVE SENDER

This is the code for the master, the TTGO ESP32, requesting and receiving data from the slave. The code also works for the ESP32 WROVER-E board.

> Note: the example below varies slightly from the example provided with the Wire library documentation. The pins used for I2C communication are explicitly declared.

```
#include <Wire.h>

// Define Slave I2C Address. All slaves must be allocated a
// unique number.

#define SLAVE_ADDR 9

// Define the pins used for SDA and SCL. This is important because
// there is a problem with the TTGO and I2C will not work properly
// unless you do.

#define I2C_SDA 21
#define I2C_SCL 22

void setup() {
  Wire.begin (I2C_SDA, I2C_SCL);      // Configure the pins
  Serial.begin(9600);                 // start serial for output
}

void loop() {
  Wire.requestFrom(SLAVE_ADDR, 6);    // request 6 bytes from slave
                                      // device SLAVE_ADDR

  String received_string = "";
  while (Wire.available()) {  // slave may send less than requested
    char c = Wire.read();     // receive a byte as character
    received_string += c;
  }

  Serial.println(received_string);         // print the character

  delay(500);
}
```

This is the corresponding code for the Arduino slave:

```
// Wire Slave Sender
// by Nicholas Zambetti <http://www.zambetti.com>

// Demonstrates use of the Wire library
// Sends data as an I2C/TWI slave device
// Refer to the "Wire Master Reader" example for use with this

// Created 29 March 2006

// This example code is in the public domain.

// Define Slave I2C Address
#define SLAVE_ADDR 9

#include <Wire.h>

void setup() {
  Wire.begin(SLAVE_ADDR);                // join i2c bus with address #8
  Wire.onRequest(requestEvent);          // register event
}

void loop() {
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() {
  Wire.write("hello "); // respond with message of 6 bytes
  // as expected by master
}
```

You should extend the examples above so that you can send, receive and process sensor data from one or more Arduinos to an ESP32.

Other I2C configurations are possible, such as the Master Writer / Slave Receiver. When you have the first example working, you should think about extending your program so that your ESP32 can also send messages to an Arduino for further processing.

USING WIFI

Before using Thingsboard, you need to write some code for the ESP32 to connect to the WiFi network. There are some examples for using an access point with the ESP32 board included with the Arduino IDE and this is not discussed further here. You will need to do something slightly different if you are connecting to eduroam though, here is some example code that you can start with which connects to an enterprise network like eduroam.

```
#include <WiFi.h>             // WiFi control for ESP32
#include "esp_wpa2.h"   //wpa2 library for connections to Enterprise networks

#define EAP_IDENTITY "insert your user name@ucl.ac.uk here"
#define EAP_PASSWORD "insert your password here"

const char* ssid = "eduroam";
WiFiClient espClient;           // Initialize ThingsBoard client
int status = WL_IDLE_STATUS;    // the Wifi radio's status

void connectWifi() {
  Serial.println();
  Serial.print(F("Connecting to "));
  Serial.println(ssid);

  WiFi.disconnect(true);  //disconnect from wifi to set new wifi connection
  WiFi.mode(WIFI_STA);    //init wifi mode
  esp_wifi_sta_wpa2_ent_set_username((uint8_t *)EAP_IDENTITY, strlen(EAP_IDENTITY));
  esp_wifi_sta_wpa2_ent_set_password((uint8_t *)EAP_PASSWORD, strlen(EAP_PASSWORD));
  esp_wifi_sta_wpa2_ent_enable();

  WiFi.begin(ssid);       //connect to wifi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
  }
  Serial.println("");
  Serial.println(F("WiFi is connected!"));
  Serial.println(F("IP address set: "));
  Serial.println(WiFi.localIP()); //print LAN IP
}

void setup() {
  Serial.begin(115200);
  connectWifi();
}

void loop() {

  // Reconnect to WiFi, if needed
  if (WiFi.status() != WL_CONNECTED) {
    connectWifi();
    return;
  }

}
```

SENDING TELEMETRY DATA TO THINGSBOARD

To complete the last part of the connectivity pathway, you will send some telemetry data to Thingsboard by provisioning the ESP32 on Thingsboard (`demo.thingsboard.io`) and sending some random data to Thingsboard programmatically, using the Arduino Thingsboard library.

SETTING UP THINGSBOARD

Use the Thingsboard demo service at `demo.thingsboard.io` to register your ESP32. Follow the instructions at Getting Started with Thingsboard to do that. Thingsboard will generate a token for your ESP32, keep this safe, you will need it when you connect to Thingsboard programmatically using the Thingsboard API.

USING THE THINGSBOARD LIBRARY

Start by installing the Thingsboard library via the 'Tools -> Manage Libraries' menu in the Arduino IDE. In doing so, the Thingsboard examples are available via 'File -> Examples -> Thingsboard'.

Notes:

1. Install and use Thingsboard library v0.20, not the most recent version. Some library features do not work with the most recent version.
2. Many of the examples feature the ESP8266 rather than the ESP32 and must be adapted. The code below illustrates how to send data to Thingsboard using the API.

```
#include <WiFi.h>              // WiFi control for ESP32
#include <ThingsBoard.h>       // ThingsBoard SDK

// WiFi
#define WIFI_AP_NAME         "YOUR AP NAME"
#define WIFI_PASSWORD        "your ap passwd"


// See https://thingsboard.io/docs/getting-started-guides/helloworld/
#define TOKEN                "your thingsboard token"
#define THINGSBOARD_SERVER   "demo.thingsboard.io"

WiFiClient espClient;            // Initialize ThingsBoard client
ThingsBoard tb(espClient);       // Initialize ThingsBoard instance
int status = WL_IDLE_STATUS;     // the Wifi radio's status
static uint16_t messageCounter = 0; // count values sent

void InitWiFi()
{
  Serial.println("Connecting to AP ...");
  // attempt to connect to WiFi network

  WiFi.begin(WIFI_AP_NAME, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to AP");
}
```

Code continued overleaf:

Code continued here:

```
void reconnect() {
  // Loop until we're reconnected
  status = WiFi.status();
  if ( status != WL_CONNECTED) {
    WiFi.begin(WIFI_AP_NAME, WIFI_PASSWORD);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
    }
    Serial.println("Connected to AP");
  }
}

void setup() {
  Serial.begin(115200);

  WiFi.begin(WIFI_AP_NAME, WIFI_PASSWORD);
  InitWiFi();
}


void loop() {

  delay(1000);

  // Reconnect to WiFi, if needed
  if (WiFi.status() != WL_CONNECTED) {
    reconnect();
    return;
  }

  // Reconnect to ThingsBoard, if needed
  if (!tb.connected()) {

    // Connect to the ThingsBoard
    Serial.print("Connecting to: ");
    Serial.print(THINGSBOARD_SERVER);
    Serial.print(" with token ");
    Serial.println(TOKEN);
    if (!tb.connect(THINGSBOARD_SERVER, TOKEN)) {
      Serial.println("Failed to connect");
      return;
    }
  }


  float r = (float)random(1000)/1000.0;
  messageCounter++;

  Serial.print("Sending data...[");
  Serial.print(messageCounter);
  Serial.print("]: ");
  Serial.println(r);

  // Uploads new telemetry to ThingsBoard using MQTT.
  // See https://thingsboard.io/docs/reference/mqtt-api/#telemetry-upload-api
  // for more details
  tb.sendTelemetryInt("count", messageCounter);
  tb.sendTelemetryFloat("randomVal", r);

  // Process messages
  tb.loop();

}
```