PARVATHA REDDY BABUL REDDY
VISVODAYA INSTITUTE OF TECHNOLOGY AND SCIENCE's

VITSPACE

For more Materials
☺ Click the link given below ☺

vitspace.netlify.com

Scroll to read your material

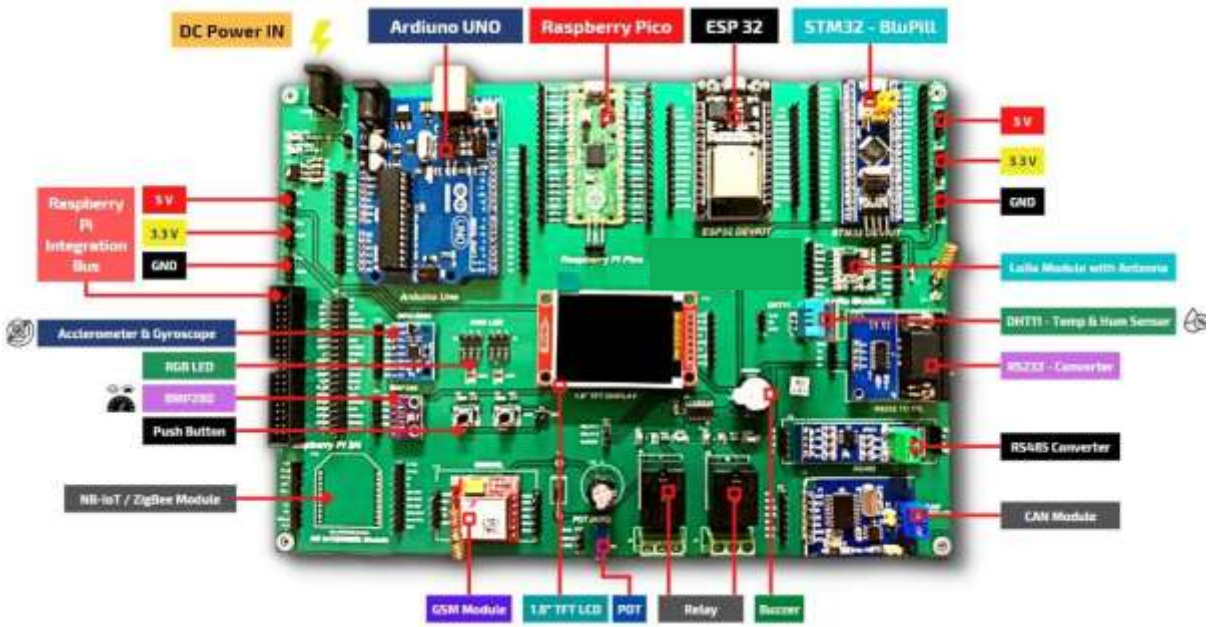
Contents

Preface

We have designed a new Variant of Multi-controller IoT Kit, which is ready to use with a various Wireless protocol's like Wi-Fi, BLE, LoRa and Embedded AI/ ML with onboard sensors and its help Engineers to Learn, Develop and Implement from Concept to Idea Applications.

Introduction to Hardware

Hello! Welcome to the starter guide of IoT Experience Kit based on C, C++, MicroPython & AI/ML. This guide is a course based on the IoT Experience Kit developed by EdGate Technologies. In this course, we will use Arduino UNO, Raspberry Pi Pico, ESP32, STM32 as the controller, along with the Various Sensors, Wired Communication & Wireless Communication Protocols, to learn how to Program, Communicate & Integrate with Various IDEs and Cloud Servers. In the first lesson of this series, I will give you a detailed introduction to the main characters of this series: IoT Experience Kit Hardware's.



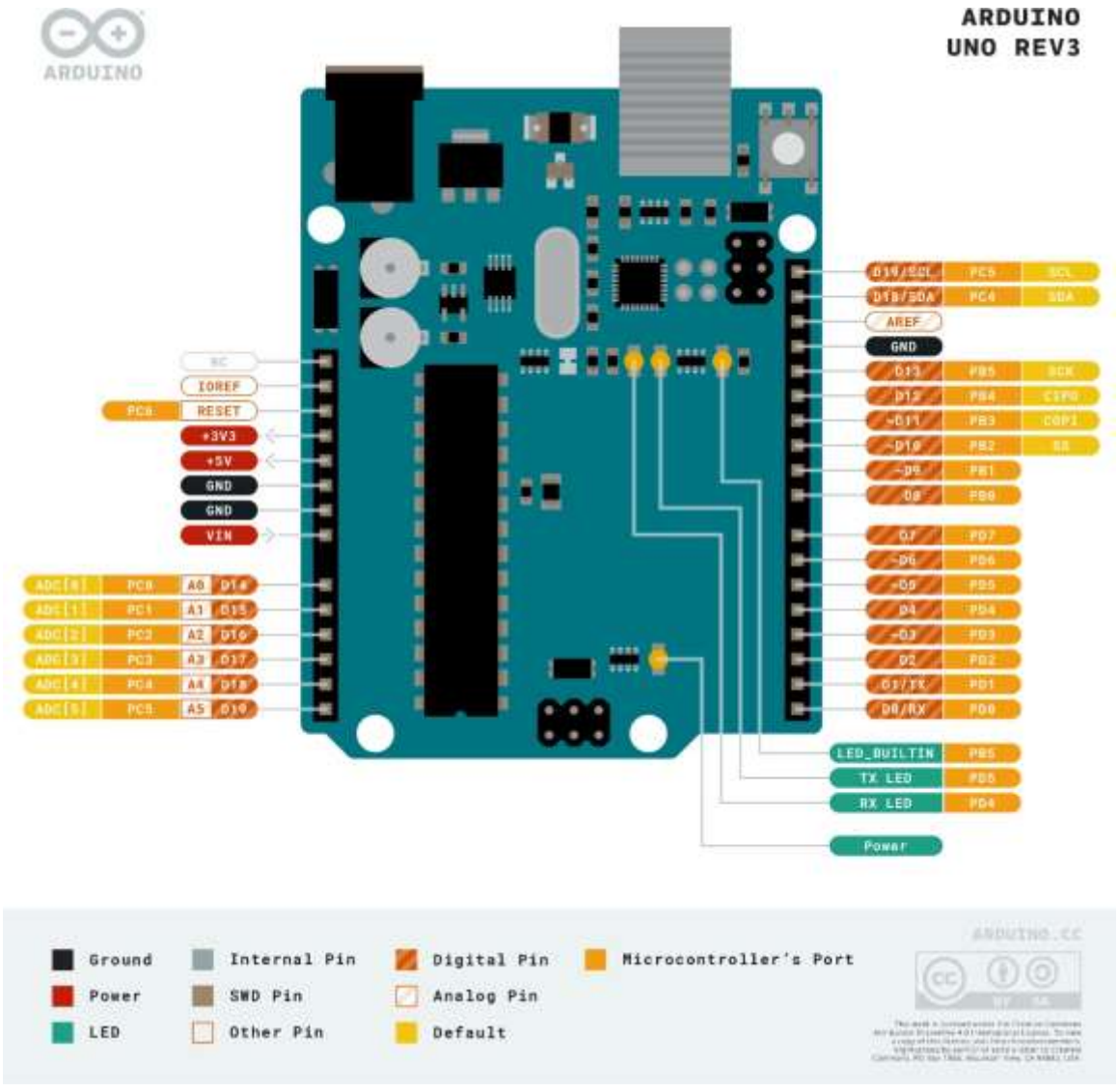
Microcontroller: Arduino UNO

Introduction of Arduino UNO:

The Arduino UNO is the best board to get started with electronics and coding. If this is your first experience tinkering with the platform, the UNO is the most robust board you can start playing with. The UNO is the most used and documented board of the whole Arduino family.

Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

GPIO Pin Outs:



Microcontroller: Raspberry Pi Pico

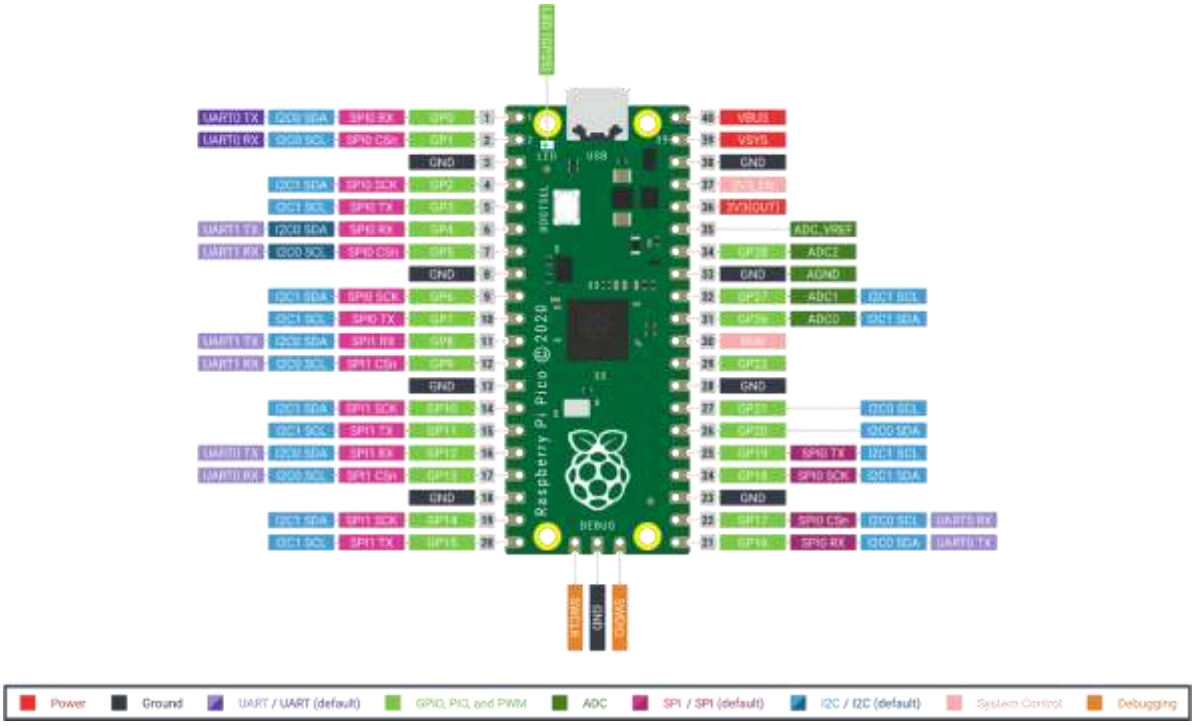
Introduction of Pico:

No one expected that Raspberry Pi, the most popular single-board computer maker in the world, would suddenly release a microcontroller of its own. What's more surprising is that Raspberry Pi Pico does not base its design on the common ESP32 or SAMD21, but instead a brand new microcontroller chip: the RP2040 microcontroller. The RP2040 microcontroller is a microcontroller chip independently designed by Raspberry Pi, and is powered by a dual-core ARM Cortex-M0+ processor that runs up to 133Mhz.

GPIO Pins:

In addition to the powerful new chip, the board of Raspberry Pi Pico exposes 26 multifunction GPIO pins, including 2 * SPI, 2 * I2C, 2 * UART, 3 * 12 -bit ADC, and 16 controllable PWM channels. I will explain the functions of these pins in later chapters.

In addition to these GPIO pins, Pico also has eight ground pins and a series of power pins. In this course, however, we will not utilize them as we will not be performing tedious wiring with breadboards or DuPont wires when building projects.



Microcontroller: ESP32

Introduction of ESP32:

The ESP32 is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs. All embedded memory, external memory and peripherals are located on the data bus and/or the instruction bus of these CPUs.

ESP32 can perform as a complete standalone system or as a slave device to a host MCU, reducing communication stack overhead on the main application processor. ESP32 can interface with other systems to provide Wi-Fi and Bluetooth functionality through its SPI / SDIO or I2C / UART interfaces.

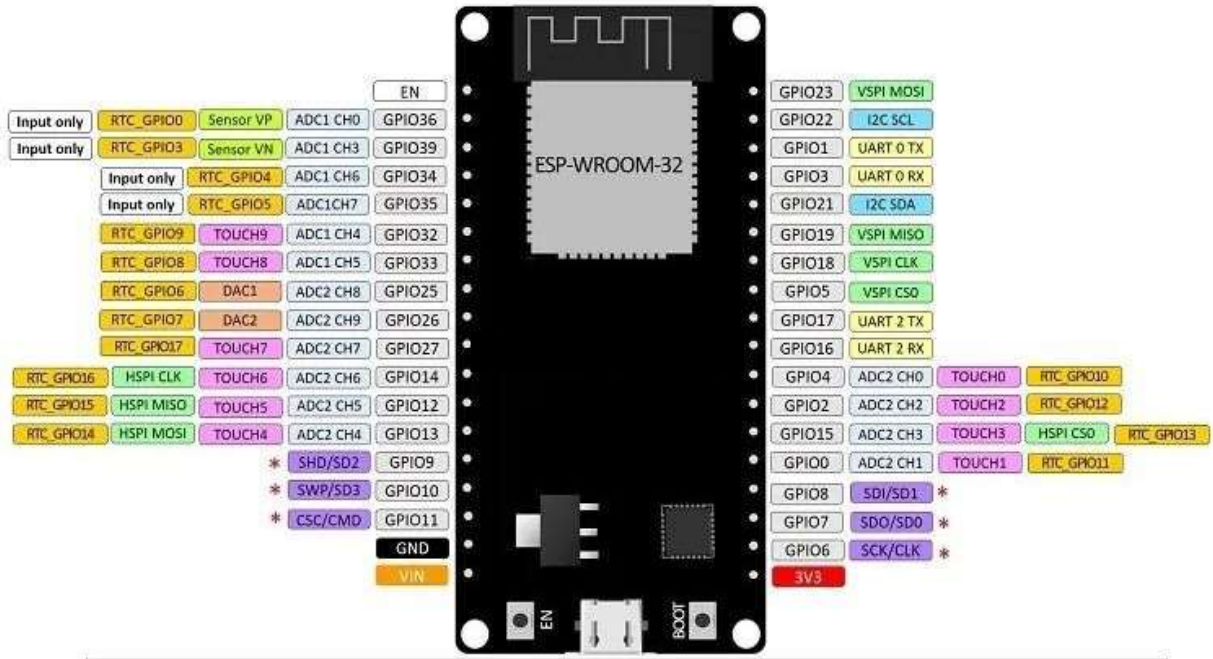
The two CPUs are named "PRO_CPU" and "APP_CPU" (for "protocol" and "application"), however, for most purposes the two CPUs are interchangeable.

GPIO Pins:

The ESP32 peripherals include:

- 18 Analog-to-Digital Converter (ADC) channels
- 3 SPI interfaces
- 3 UART interfaces
- 2 I2C interfaces
- 16 PWM output channels
- 2 Digital-to-Analog Converters (DAC)
- 2 I2S interfaces
- 10 Capacitive sensing GPIOs

The ADC (analog to digital converter) and DAC (digital to analog converter) features are assigned to specific static pins. However, you can decide which pins are UART, I2C, SPI, PWM, etc – you just need to assign them in the code. This is possible due to the ESP32 chip's multiplexing feature.



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

Microcontroller: STM32

Introduction of STM32:

The STM32 family of 32-bit microcontrollers based on the Arm® Cortex®-M processor is designed to offer new degrees of freedom to MCU users. It offers products combining very high performance, real-time capabilities, digital signal processing, low-power / low-voltage operation, and connectivity, while maintaining full integration and ease of development.

The unparalleled range of STM32 microcontrollers, based on an industry-standard core, comes with a vast choice of tools and software to support project development, making this family of products ideal for both small projects and end-to-end platforms.

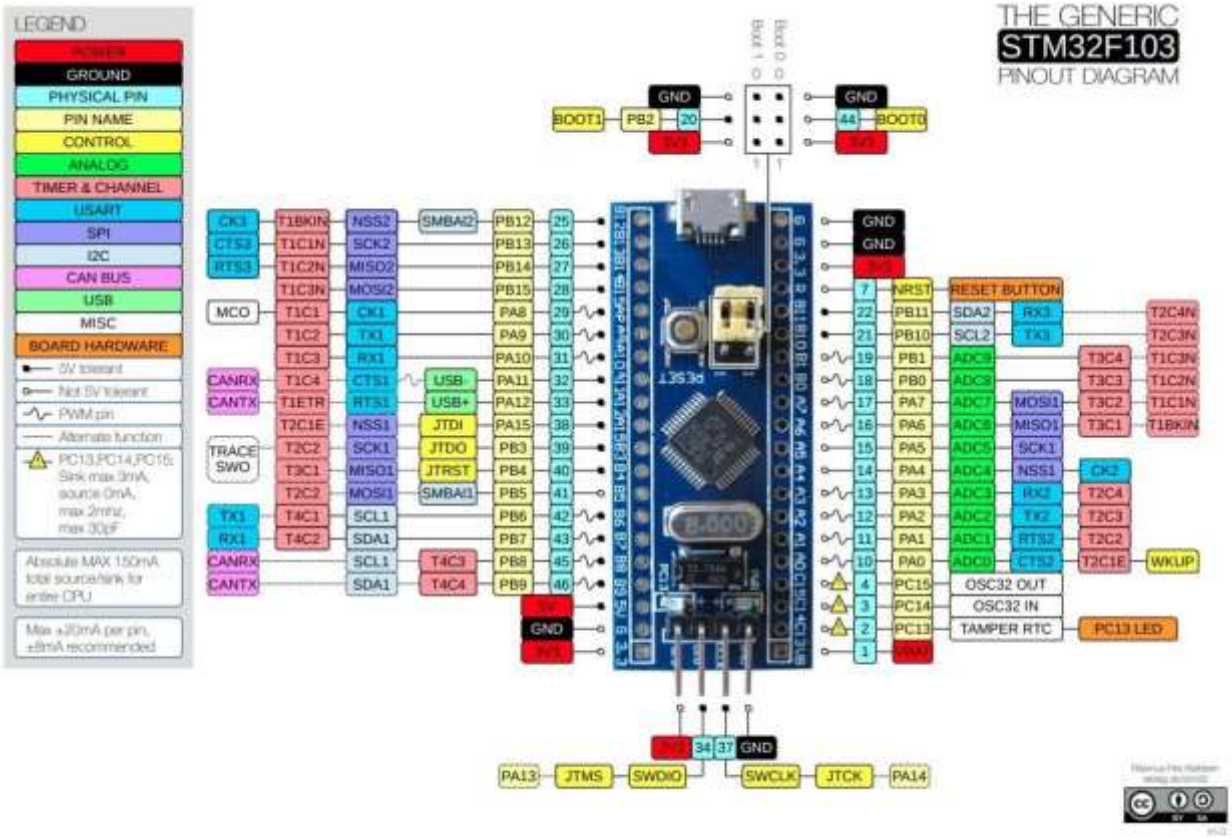
GPIO

The Blue Pill is a 32-bit Arduino compatible development board that features the STM32F103C8T6, a member of the STM32 family of ARM Cortex-M3 core microcontrollers. This board aims to bring the 32-bit ARM Cortex microcontrollers to the hobbyist market with the Arduino style form factor.

Powering your Blue Pill:

There are three ways of powering your Blue Pill development board:

- Using the built-in USB micro connector.
- Supplying 5V to the 5V pin as external supply.
- Supplying 3.3V directly to the 3.3V pin.



Sensors

Sensors	Description
	**DHT 11 :** The DHT11 is a commonly used **Temperature and humidity** sensor that comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data.
	**MPU6050**: MPU6050 sensor module is complete 6-axis Motion Tracking Device. It combines **3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor** all in small package. Also, it has additional feature of on-chip Temperature sensor. It has I2C bus interface to communicate with the microcontrollers.
	**BMP280 Barometric Pressure and Altitude Sensor** The BMP280 Breakout has been designed to be used in indoor/outdoor navigation, weather forecasting, home automation, and even personal health and wellness monitoring. And to measure **barometric pressure, and temperature** readings all without taking up too much space.

Wired Communication Modules

Modules	Description
	**RS232 to TTL Serial Interface Module** This RS232 to TTL Serial Interface Module is a board with the MAX3232 transceiver integrated circuit (IC). It facilitates serial communication between TTL and RS232 ports by providing the necessary electrical signal conversion.
	**MAX485 TTL To RS485 Module:** The MAX485 TTL To RS485 Module converts TTL signal to RS485 for long range, high data rate error prone differential communication. Digital communications networks implementing the EIA-485 standard can be used effectively over long distances and in electrically noisy environments. Multiple receivers may be connected to such a network in a linear, multi-drop configuration. These characteristics make such networks useful in industrial environments and similar applications.

	**MCP2515 CAN Bus Module:** It contains CAN Controller MCP2515 and TJA1050 which is a high speed CAN trans-receiver. This module can be esaily interfaced to any microcontroller via SPI Interface. It can also be interfaced to Arduino and Raspberry PI.

Wireless Communication Modules

Modules	Description
	**LoRa Transceiver** RFM series wireless communication module is designed to work at 865 - 867 MHz ISM band global free application. It is designed for low-power, high-bandwidth wireless digital communications module. The modules have a low cost, small, stable, product consistency and other characteristics, the highest spatial rate of up to 300Kbps, can be widely used in daily life and activities of the occasion requires a wireless connection can also be used for industrial control, access control, attendance, surveillance and security industries.
	**SIM800 GSM Module:** Sim800L Module is low cost, low form factor GSM module based on Simcoms SIM800L chipset. Sim800L module supports quad-band GSM and GPRS network. This breakout board is perfect for application where size and cost is a constraint. Sim800l gsm module also supports quad band which means that it can work anywhere in the world. This low cost module is perfect for launching your next IoT project. Using this module you can almost make your own cellphone.

Display:

Modules	Descriptions
	**TFT LCD Display:** This lovely little display breakout is the best way to add a small, colorful and bright display to any project. Since the display uses 4 - wire SPI to communicate and has its own pixel-addressable frame buffer, it can be used with every kind of microcontroller. Even a very small one with low memory and few pins available!

Introduction to Software

Installing Arduino IDE

The first thing you need is the Arduino IDE. If your computer doesn't have Arduino IDE installed, then visit the official Arduino download page and download the installation file for your preferred operating system

Link: https://www.arduino.cc/en/software

Installing the Microcontroller Board in Arduino IDE

1. ESP 32

There's an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE and its programming language. We'll show you how to install the ESP32 board in Arduino IDE.

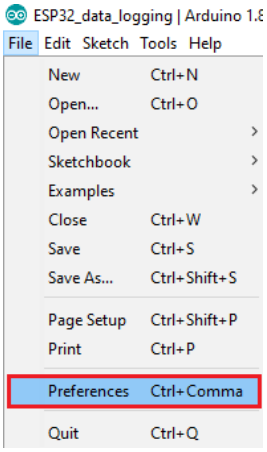
Prerequisites: Arduino IDE Instaled

Before starting this installation procedure, make sure you have the latest version of the Arduino IDE installed in your computer. If you don't, uninstall it and install it again. Otherwise, it may not work.

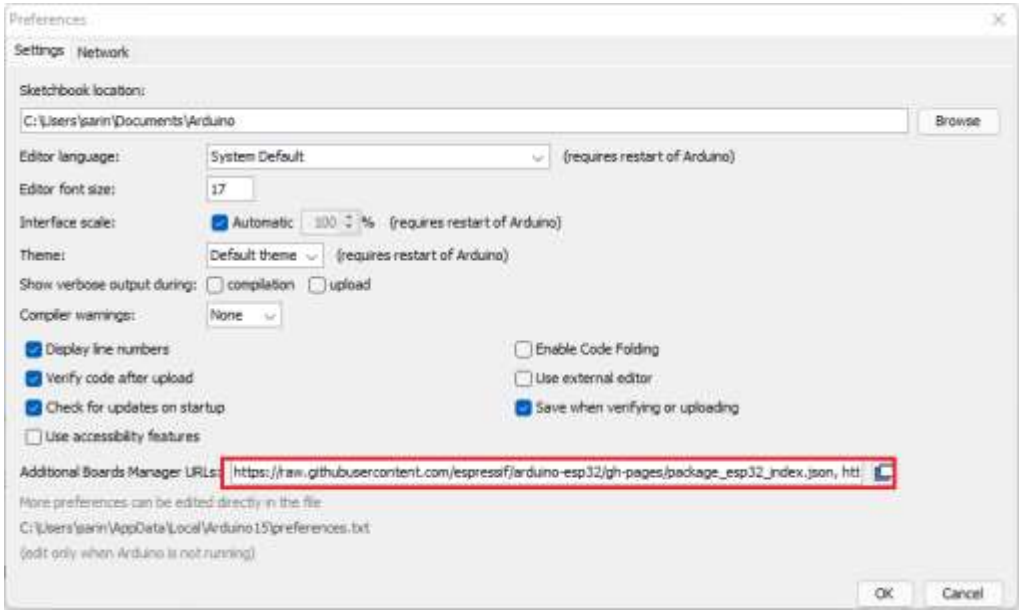
Instaling ESP32 Add-on in Arduino IDE

To install the ESP32 board in your Arduino IDE, follow these next instructions:

i. In your Arduino IDE, go to File> Preferences

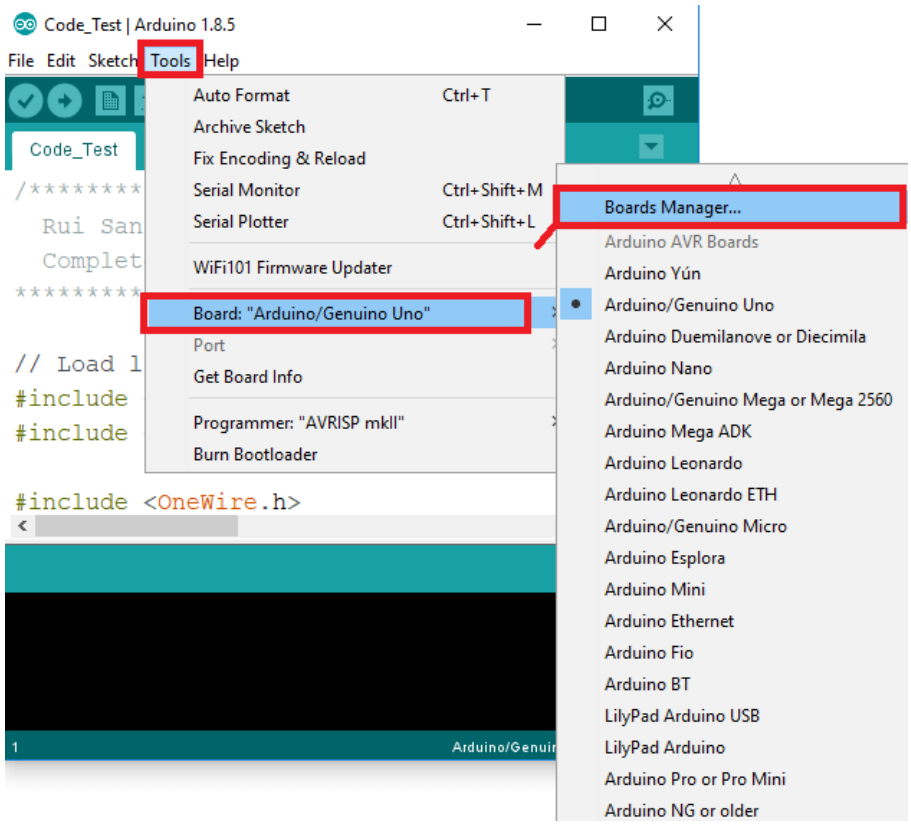


ii. Enter the following into the "Additional Board Manager URLs" field: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

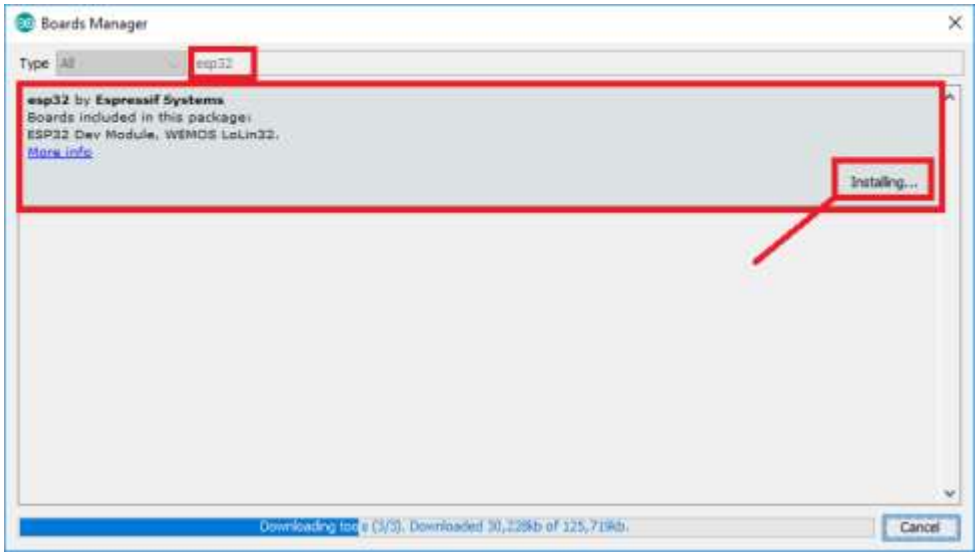


Then, click the "OK" button:

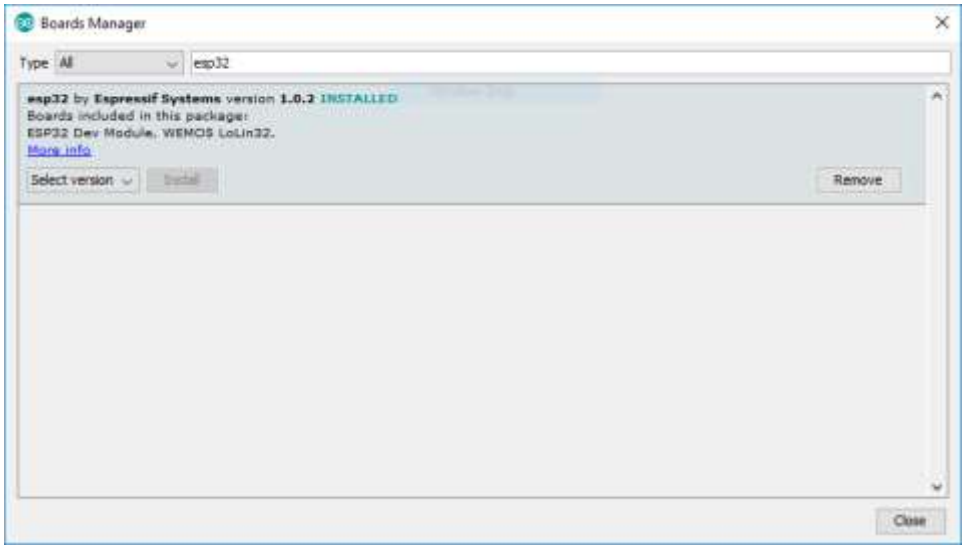
iii. Open the Boards Manager. Go to **Tools > Board > Boards Manager...**



iv. Search for ESP32 and press install button for the **"ESP32 by Espressif Systems":**



v. That's it. It should be installed after a few seconds



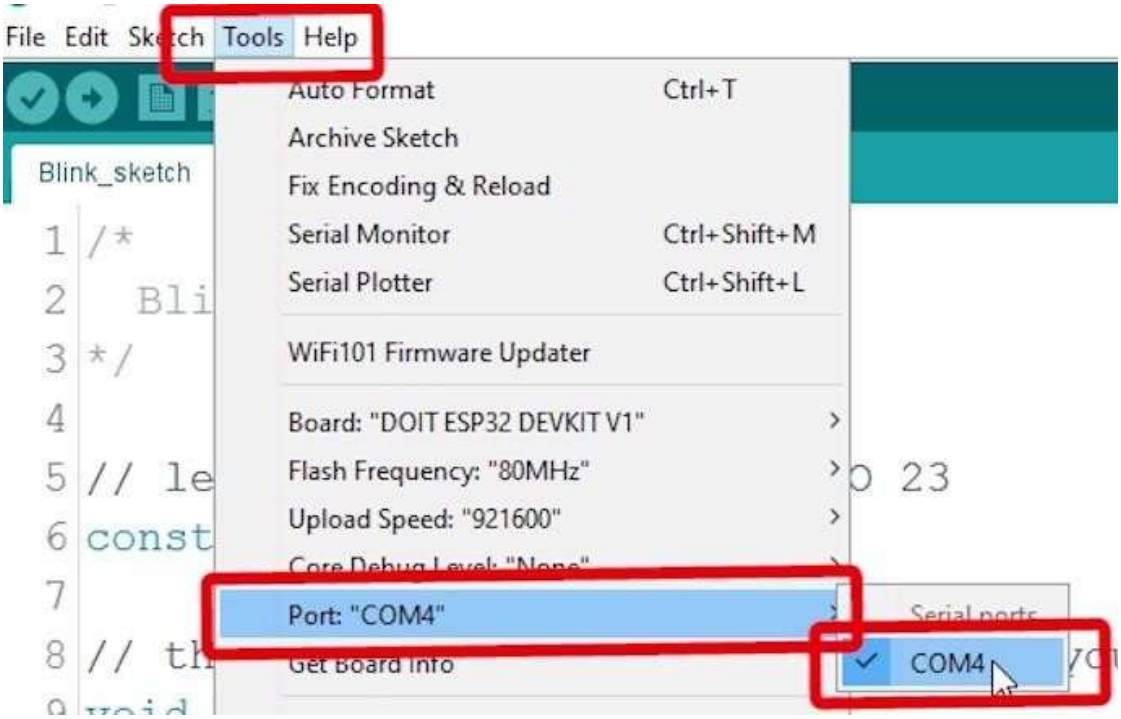
.

Testing the Instalation

Plug the ESP32 board to your computer. With your Arduino IDE open, follow these steps:

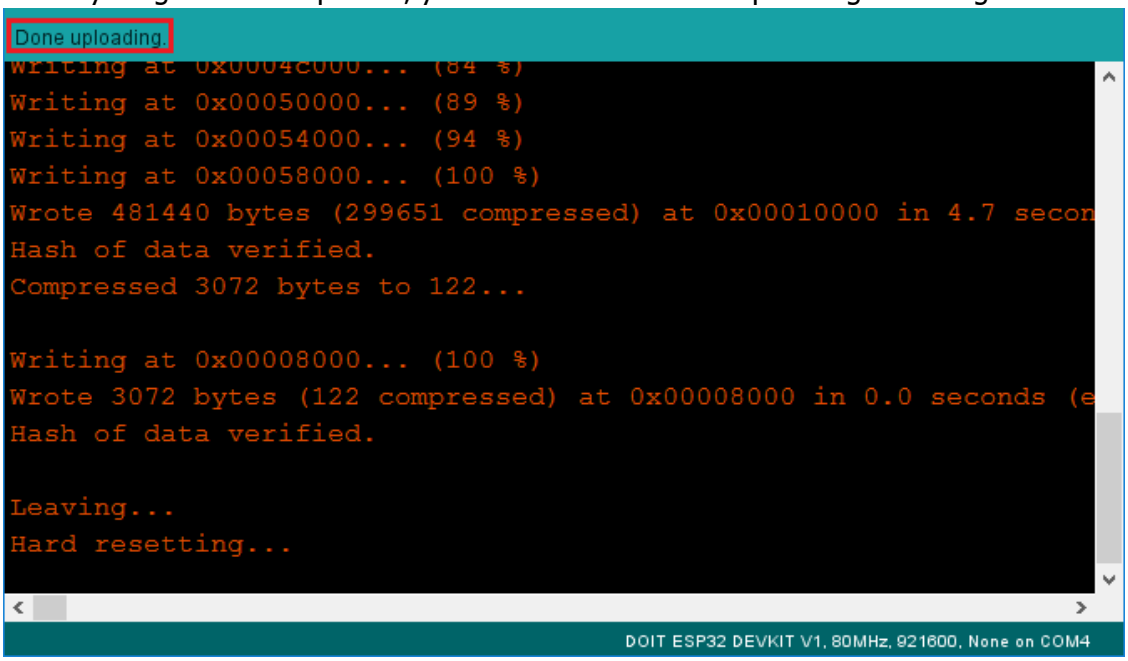
1. Select your Board in Tools > Board menu (in my case it's the DOIT ESP32 DEVKIT V1)

2. 2. Select the Port (if you don't see the COM Port in your Arduino IDE, you need to install the CP210x USB to UART Bridge VCP Drivers)



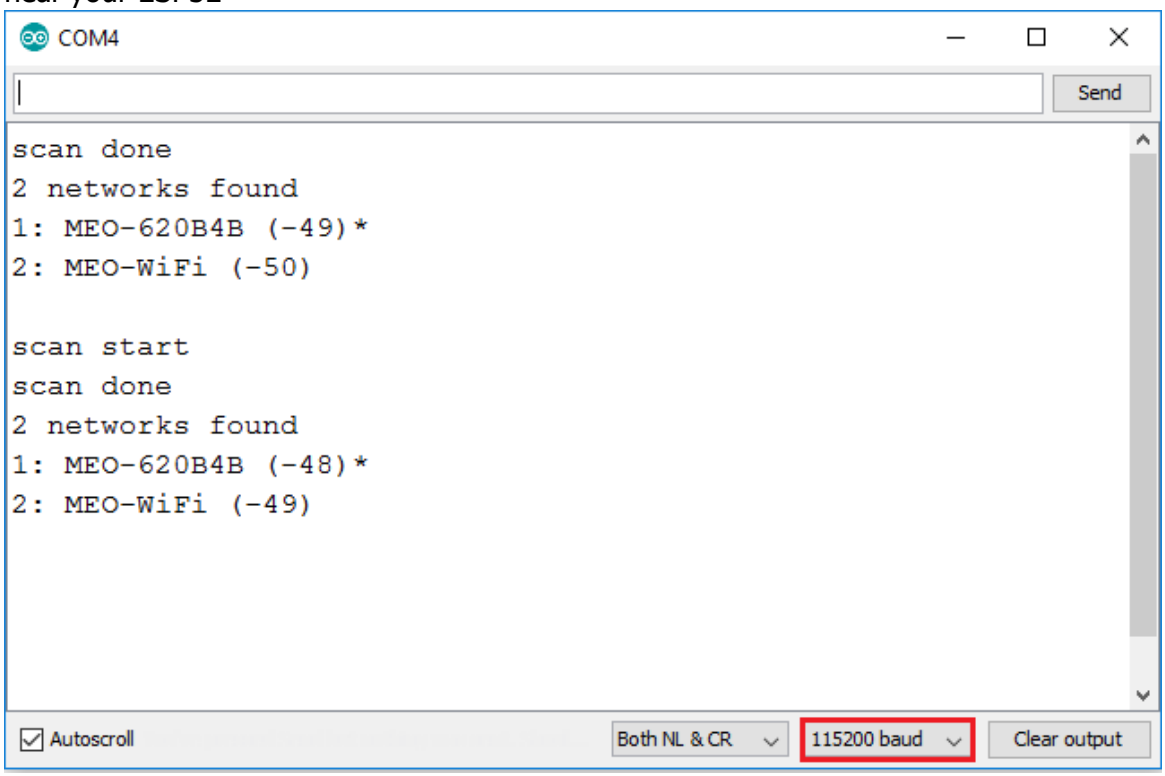
3. Open the following example under File > Examples > WiFi (ESP32) > WiFiScan
4. A new sketch opens in your Arduino IDE
5. Press the Upload button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board.

6. If everything went as expected, you should see a "Done uploading." message.



7. Open the Arduino IDE Serial Monitor at a baud rate of 115200

8. Press the ESP32 on-board Enable button and you should see the networks available near your ESP32

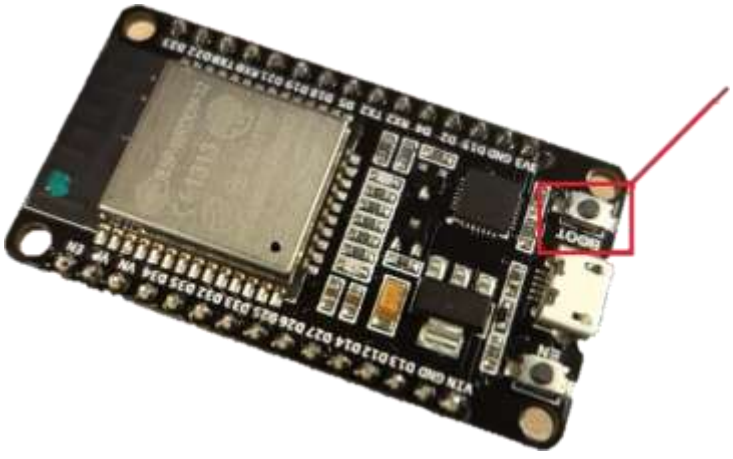


Troubleshooting

If you try to upload a new sketch to your ESP32 and you get this error message "A fatal error occurred: Failed to connect to ESP32: Timed out… Connecting…". It means that your ESP32 is not in flashing/uploading mode.

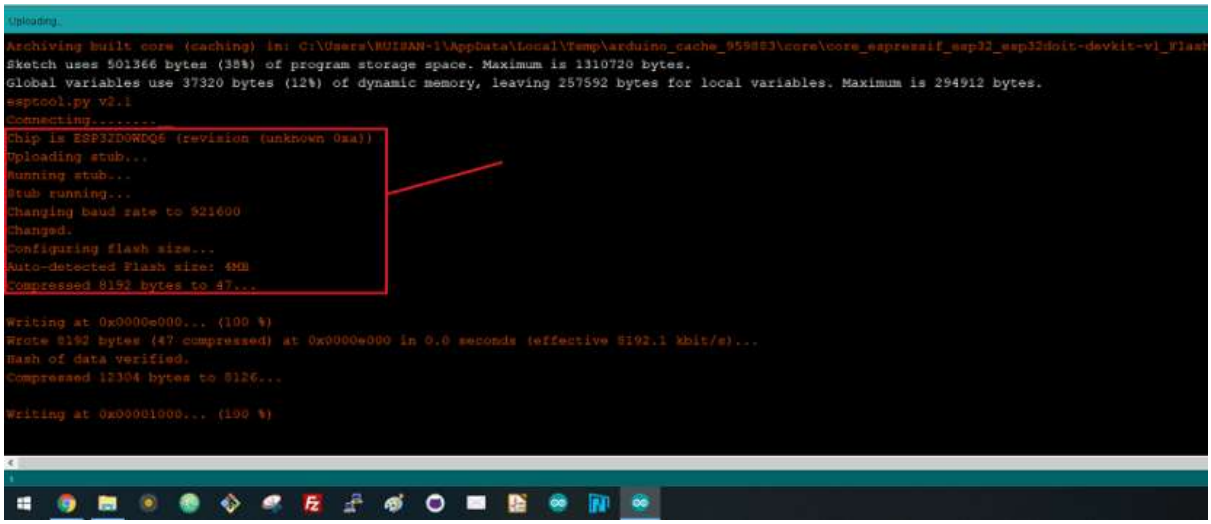
Having the right board name and COM por selected, follow these steps:

- Hold-down the "BOOT" button in your ESP32 board



- Press the "Upload" button in the Arduino IDE to upload your sketch:

- After you see the "Connecting…." message in your Arduino IDE, release the finger from the "BOOT" button:

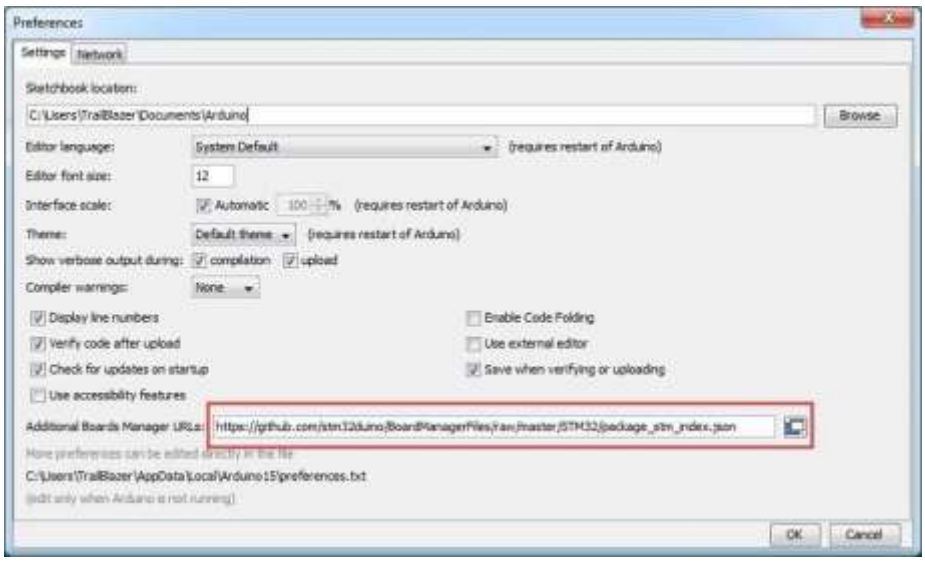


That's it. Your ESP32 should have the new sketch running. Press the "ENABLE" button to restart the ESP32 and run the new uploaded sketch.

2. STM 32 Blu - Pill

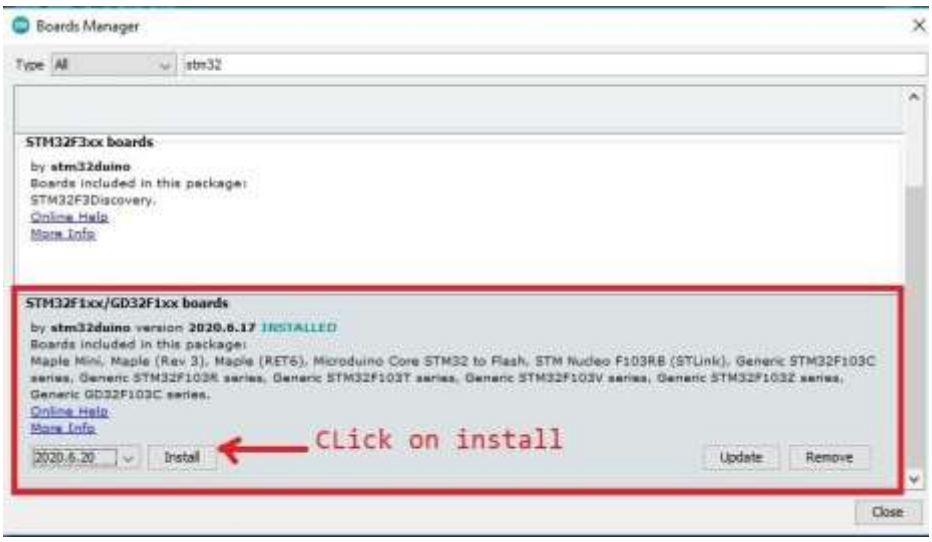
Configuring Arduino IDE to Program STM32F103C8T6 Blue Pil

I am sure you already have Arduino IDE installed on your PC (or Laptop). If not, then install it first. After than open your Arduino IDE and select File -> Preferences. You will find a tab called "Additional Boards Manager URLs". Copy the following link and paste it there.
Link: http://dan.drown.org/stm32duino/package_STM32duino_index.json



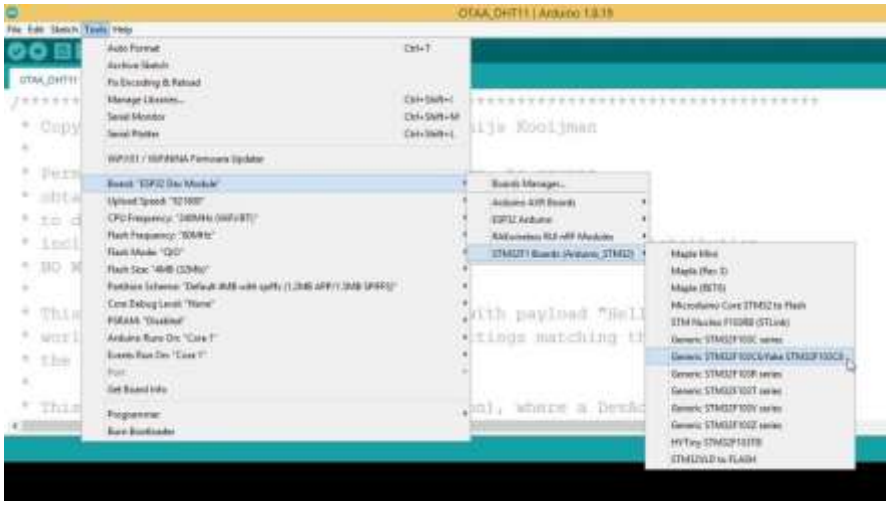
Note: If you already have some URLs in this section, you can add more by separating them with comma. If you have worked with ESP8266 Boards, then you might be familiar with this process already. After adding the **URL,** click on OK.

Now, go to Tools -> Board -> Board Manager… option and search for "stm32". You will get a result like "STM32 Cores by STMicroelectronics". Install the latest version. At the time of preparing this tutorial, the latest version is 1.8.0.

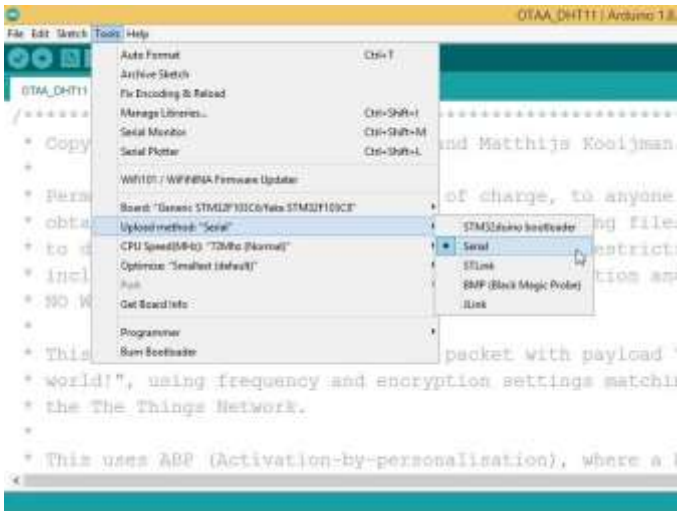


This will take some time as it will download and install some of the necessary files and tools. (I said some because, you have to download another tool from STMicroelectronics for this to work).

Now you can select the board from Tools -> Board -> Generic STM32F1 series. Once you select this board, a bunch of options will appear below for customizing your board type. The first important option is "Board part number". Make sure that "BluePill F103C8" is selected.



The other important options are "U(S)ART support", make it as "Enabled (generic 'Serial')" and "Upload method", make its as " Serial". You can leave the remaining options as their default values.



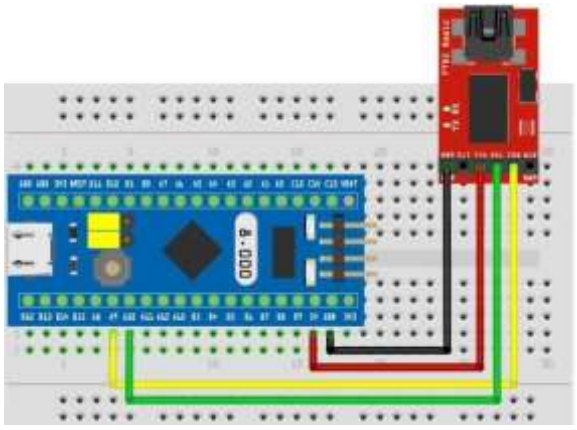
Connections:

For the purpose of easy representation, I am using an FTDI like USB to Serial Converter in Fritzing Software to show the connections.

The connections should be as follows:

STM32 Blue Pill	FTDI Programmer
5V	VCC
GND	GND

A9	RX
A10	TX



Instalation of STM32CubeProgrammer:

This section describes the requirements and the procedure for the use of the STM32CubeProgrammer software. The setup also offers optional installation of the "STM32 trusted package creator" tool, used to create secure firmware files for secure firmware install
and update. For more information, check STM32 Trusted Package Creator tool software description (UM2238), available on https://www.st.com/en/development-tools/stm32cubeprog.html

To install the STM32CubeProgrammer tool, you need to download and extract the zip package and execute **SetupSTM32CubeProgrammer_win64.exe**, which guides you through the installation process.

Test Program for STM32F103C8T6 Blue Pil Board

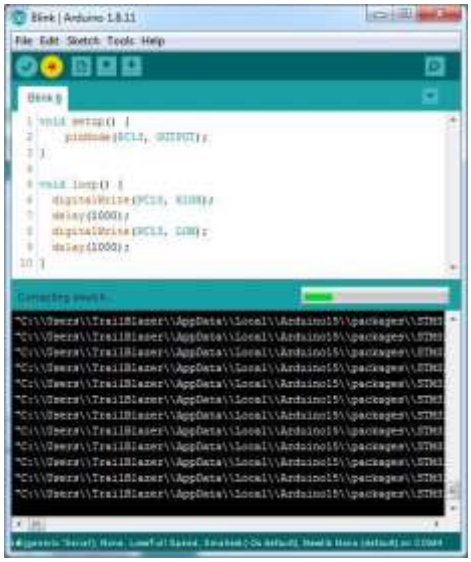
Make sure that you made the necessary changes to the Arduino IDE as mentioned in the previous section (selecting the correct board, etc.). Once that is done, make the connection between FTDI Programmer (i.e. USB to Serial Converter) and STM32 Board as mentioned before.

Now, before connecting the FTDI to the PC, make sure that STM32 Blue Pill Board is in "Programming Mode" i.e. connect the BOOT0 pin to HIGH. After that, connect the FTDI to the

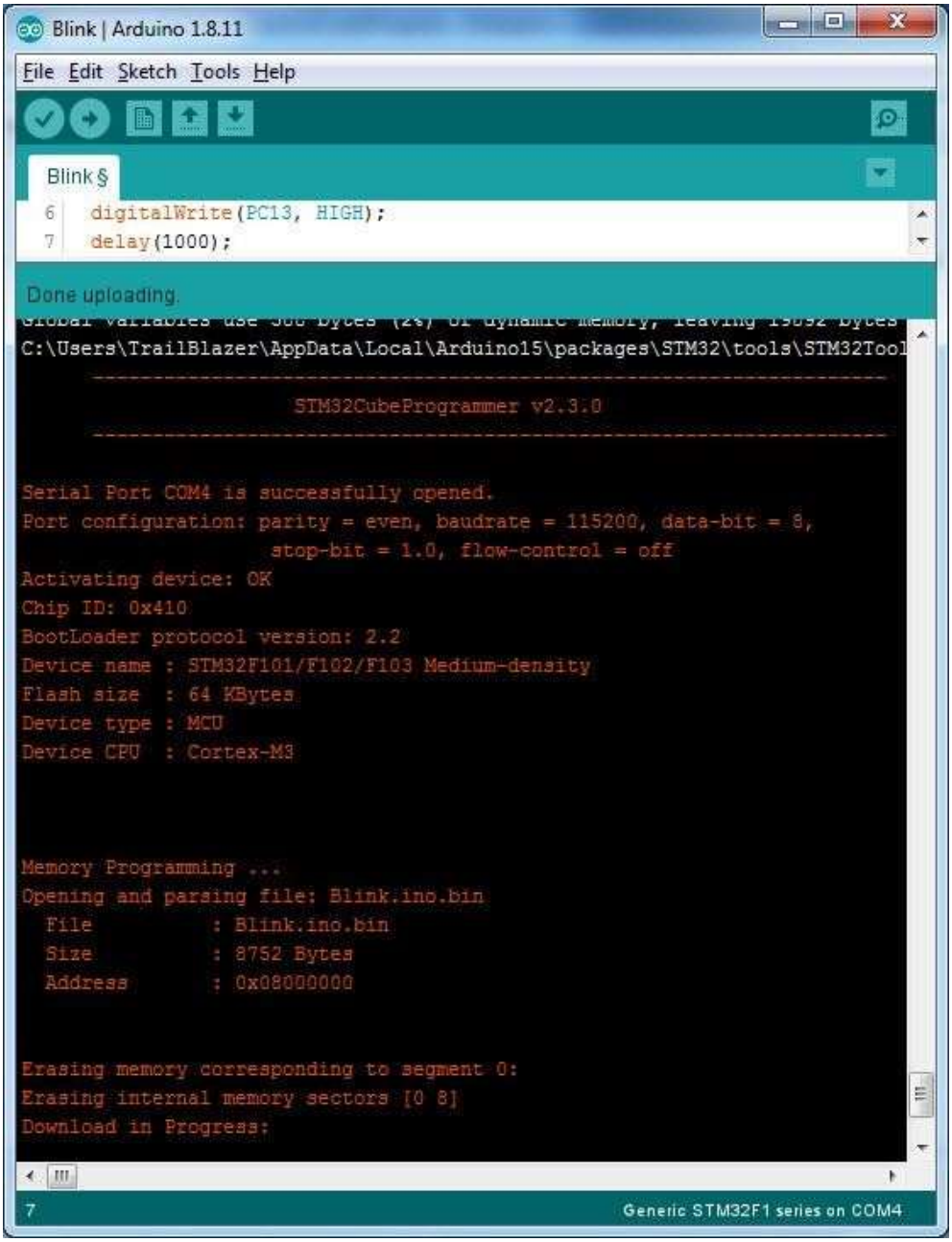
PC or Laptop. A COM port will be assigned to the programmer and select the same COM port in the Arduino IDE.

Write the Blinky program as follows. It is similar to the Arduino Blinky sketch but instead if LED_BUILTIN, I have used PC13 as the LED is connected to that pin of the MCU.

After this, you can click on Upload and the IDE will start compiling the code. It will take some time for compiling.



Once the compilation is successful, it will automatically invoke the STM32CubeProgrammer tool. If everything goes well, the IDE will successfully program the STM32 Board.



It will automatically reset the MCU and you can notice the LED blinking. Don't forget to move the BOOT0 pins back to LOW position so that the next time you power-on the board, it will start running the previously uploaded program.

Installing Thonny IDE for Raspberry Pi PICO

To build interesting projects with Raspberry Pi Pico, you need to learn how to program it first. Raspberry Pi Pico supports two types of programming languages: C++ and MicroPython.

C++ is a general-purpose language developed on the basis of the C language. It is often used in the development of ESP32 and Arduino, so most microcontroller users are familiar with this language. However, to make things simple in this course, we will use another language to program
Pico: MicroPython.

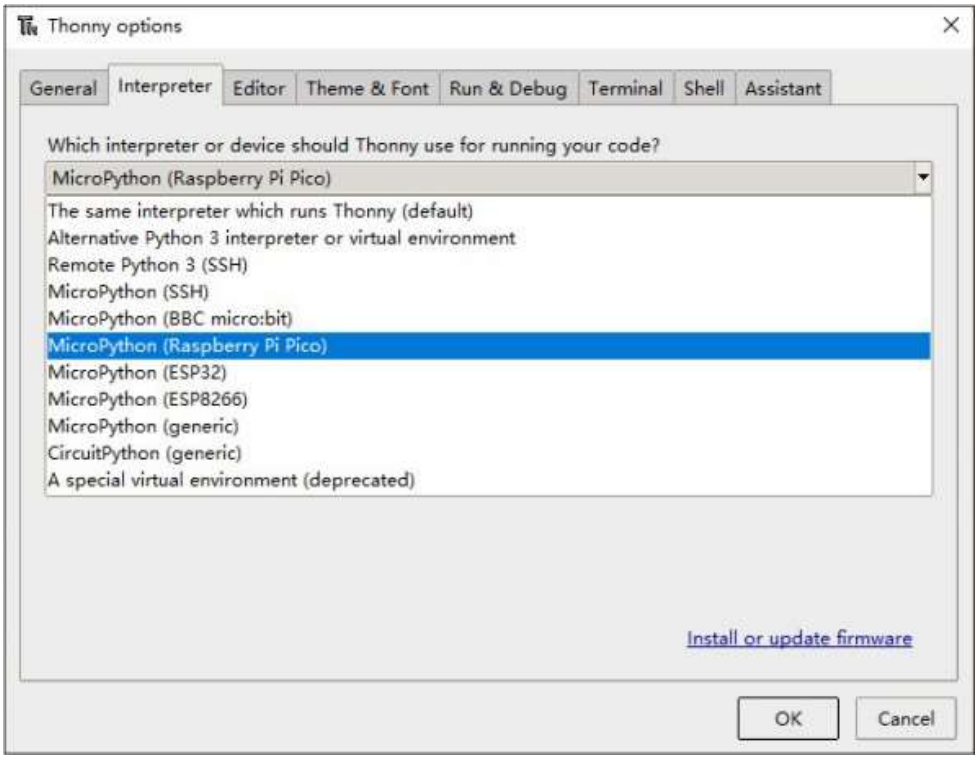
a. Install Thonny

The installation of Thonny is very simple. Since Python 3.7 is built into Thonny, all you need is a simple installer and you're ready to start learning to program. First, click Thonny.org to enter the download page, and select the installation package of Thonny to download in the upper right corner of the page according to your operating system. After downloading the installation package, double-click to open it and follow the on-screen instructions to install it.

After the installation, open Thonny. You should be directly greeted by the main interface.

The main interface of Thonny is simple, and can be divided into the following four parts:

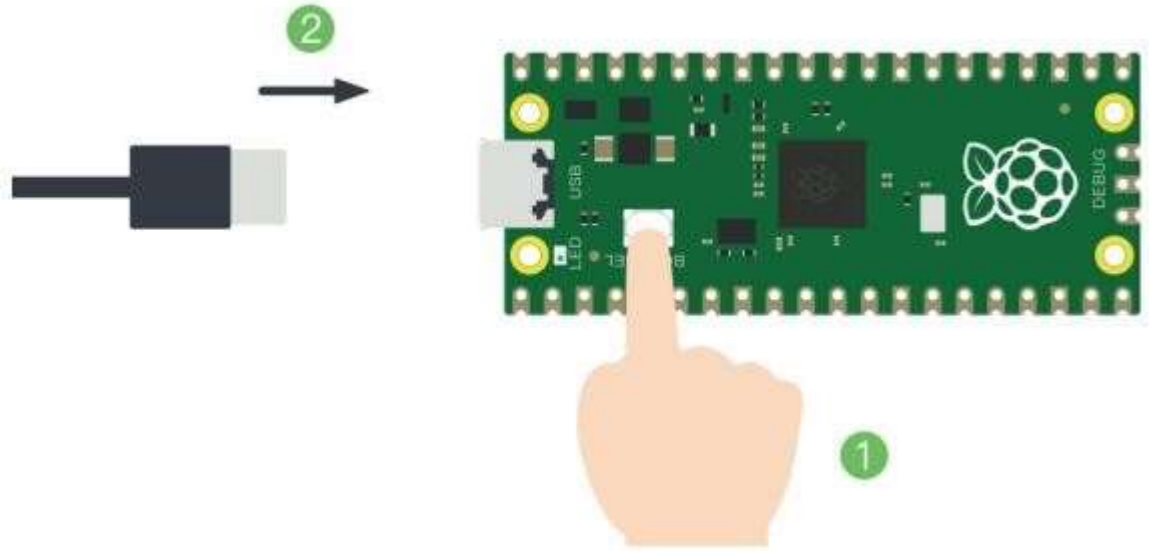
1.Toolbar: the toolbar offers icon-based quick-access to commonly used program functions,
such as new, open, save, run the current script, stop, etc.
2.Script Area: the script area is the core area of Thonny, where your Python programs are
written.
3. Python Shell: the Shell allows you to run console commands and also provides information
about running programs.
4.Interpreter: the interpreter allows you to change the interpreter version used to run your
programs. Click on "Python 3.7.9", look for "MicroPython (Raspberry Pi Pico)" in the menu,
click "OK", and switch it to Pico's exclusive interpreter.We can also switch languages in
"Theme & Font".



b. Install MicroPython for Pico

In addition to installing Thonny on the computer, we also need to install the MicroPython firmware on the Pico itself.The MicroPython firmware download link is preloaded inside Pico, so we can directly use this link to enter the download interface.

First, hold down the "BOOTSEL" button on Pico; then, while still holding it down, connect Pico to a computer using a USB cable.



At this point, a drive called RPI-RP2 should show up as a removable device on your computer.
Navigate to it to find this file: INDEX.HTM.

Open INDEX.HTM to be taken to the download page. Drop down and click on the MicroPython tab where you can find information about Pico and MicroPython. Click on the "Download UF2.file" button in this tab to download the MicroPython firmware.

Finally, drag the downloaded MicroPython firmware into the RPI-RP2 directory to complete the firmware installation.

Test Program: Helo World!

First, connect Pico to the computer using a USB cable; then, open Thonny, and click on the "restart back-end process" button on the toolbar. If you successfully connect Pico to your computer, you will see the MicroPython version information and device name returned by Pico in the Shell area.

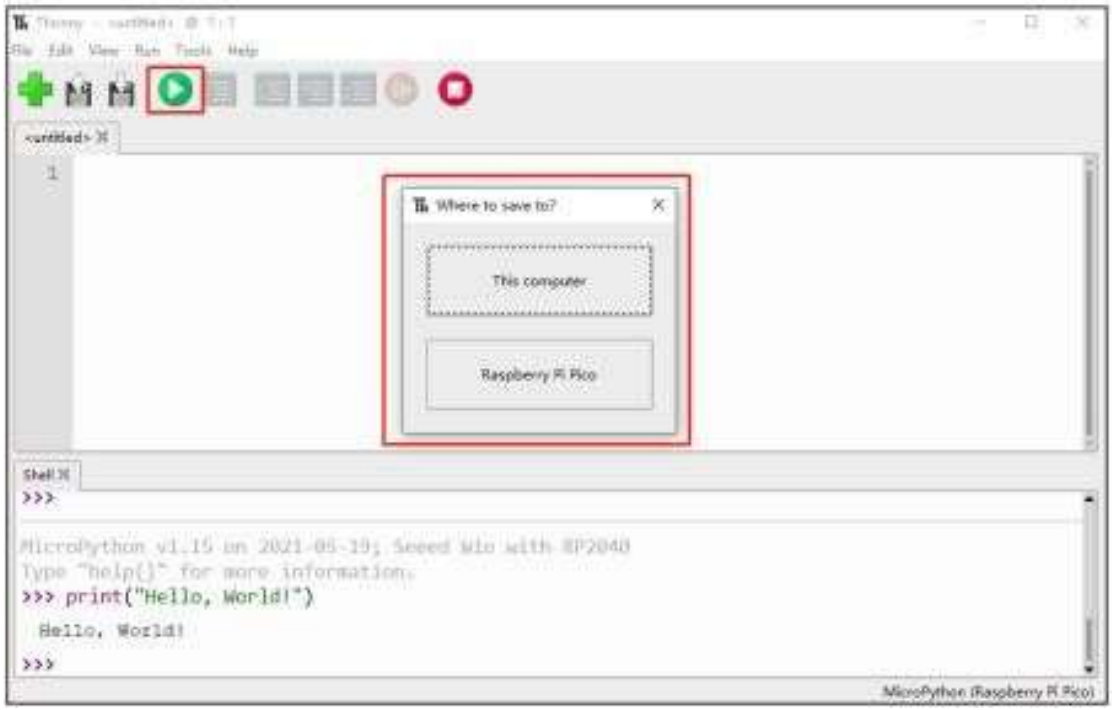
After connected, click on the Shell area and type the instruction:

```
print("Hello, World!")
```

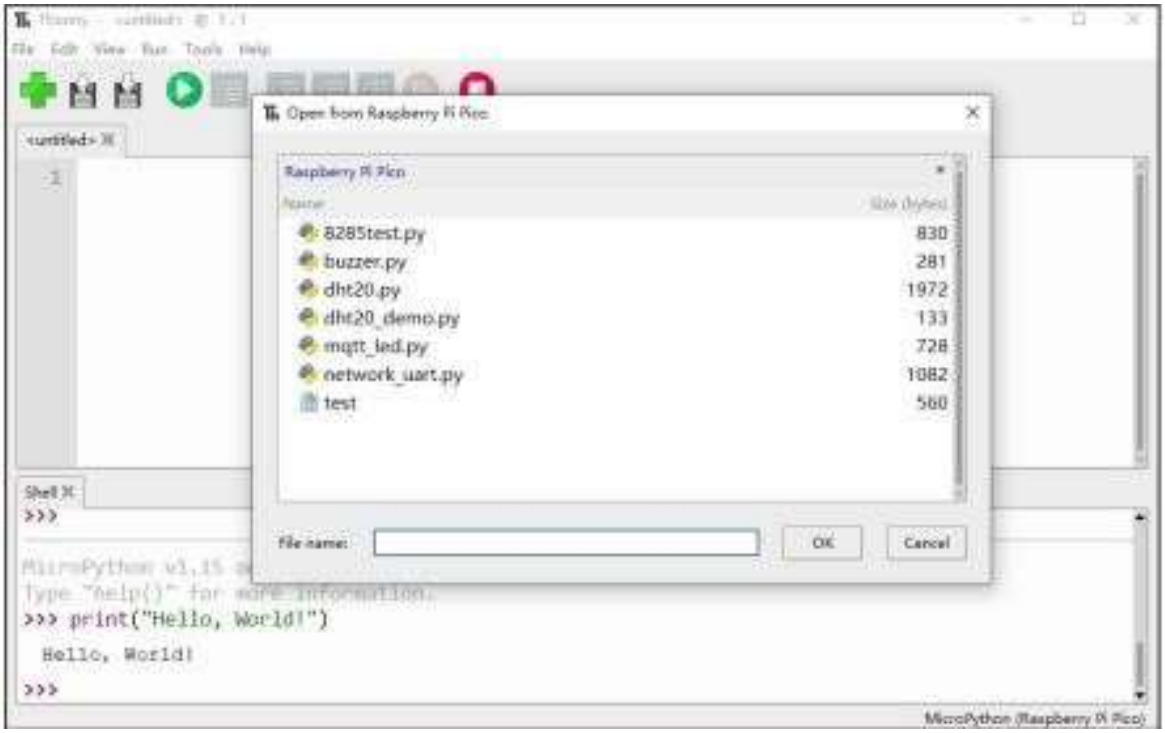
print() is a common function used as a printout. When you need to output some data to the Shell, you can use this function. When you press ENTER after you type the instruction, you'll see the execution result of the input instruction is displayed in the Python Shell area, which means, the message "Hello, World!" is printed out in the Python Shell area.

Now let's try to type the same instruction in the script area. When you press ENTER this time, nothing happens — except that a new, blank line is created in the script.

To make your program in the script work, you'll have to click the "Run current script" button in the toolbar.
If you are running a new program that has never been saved, then when you click the "Run current script" button, Thonny automatically shows you a menu with two options — save the program to "This computer" or "Raspberry Pi Pico". You can choose where to store the program according to your own needs. Once your program is saved, you'll see the execution result of the program in the Shell area.



The next time you want to open your saved program, click the "open" button on the toolbar to find the corresponding program.


Open from Raspberry Pi Pico
Raspberry Pi Pico
8285test.py 830
buzzer.py 281
dht20.py 1972
dht20_demo.py 133
mqtt_led.py 728
network_uart.py 1082
test 560
File name:
OK
Cancel
>>> print("Hello, World!")
Hello, World!
>>>
MicroPython (Raspberry Pi Pico)

Develop & Code (Basics)

1. Blink a RGB LED

Hardware Interfacing Instruction:

Arduino UNO	**Arduino Uno**	**LED**		Refer Page No:49
	D5	R		
	D4	G		
	D3	B		
RPI PICO	**PICO**	**LED**		Refer Page No:53
	GP11	R		
	GP12	G		
	GP13	B		
ESP32	**ESP32**	**LED**		Refer Page No:56
	GPIO27	R		
	GPIO26	G		
	GPIO25	B		
STM32	**STM32**	**LED**		Refer Page No:59
	GPIO13	R		
	GPIO15	G		
	GPIO15	B		

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Above in Introduction to Software	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:63
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:64
ESP32	Refer Example code in Section B - Codes	Refer Page No:63
STM32	Refer Example code in Section B - Codes	Refer Page No:63

2. Pushbutton with an LED

Hardware Interfacing Instruction:

Arduino UNO	**Arduino Uno** / **I/Os** D13 / R (LED) D4 / Pushbutton	Refer Page No:49
RPI PICO	**PICO** / **I/Os** GP26 / R (LED) GP27 / Pushbutton	Refer Page No:53
ESP32	**ESP32** / **I/Os** GPIO 2 / R (LED) GPIO 16 / Pushbutton	Refer Page No:56
STM32	**STM32** / **I/Os** GPIO 2 / R (LED) GPIO 16 / Pushbutton	Refer Page No:60

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Above in Introduction to Software	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:65
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:65
ESP32	Refer Example code in Section B - Codes	Refer Page No:65
STM32	Refer Example code in Section B - Codes	Refer Page No:65

3. DHT 11 Interfacing with Arduino UNO

Hardware Interfacing Instruction:

Arduino UNO	**Arduino Uno**	**DHT 11**		Refer Page No:50
	D2	OUT		
	3V3	3.3		
	GND	GND		
RPI PICO	**Rpi PICO**	**DHT 11**		Refer Page No:54
	GP28	OUT		
	3.3V	3.3		
	GND	GND		
ESP32	**ESP32**	**DHT 11**		Refer Page No:57
	GPIO 4	OUT		
	3.3V	3.3		
	GND	GND		
STM32	**STM32**	**DHT 11**		Refer Page No:60
	PA 0	OUT		
	3.3V	3.3		
	GND	GND		

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Above in Introduction to Software	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

Library Installation

Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:66
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:66
ESP32	Refer Example code in Section B - Codes	Refer Page No:66
STM32	Refer Example code in Section B - Codes	Refer Page No:66

4. Interfacing Relay with AC Appliances

Hardware Interfacing Instruction:

Arduino UNO	**Arduino Uno** / **Relay** D7 / Relay 1	Refer Page No:50
RPI PICO	**PICO** / **Relay** GP17 / Relay 1	Refer Page No:54
ESP32	**ESP32** / **Relay** GPIO23 / Relay 1	Refer Page No:57
STM32	**STM32** / **Relay** PB2 / Relay 1	Refer Page No:61

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Above in Introduction to Software	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:67
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:67
ESP32	Refer Example code in Section B - Codes	Refer Page No:67
STM32	Refer Example code in Section B - Codes	Refer Page No:67

5. Interfacing PoT for Reading Analog Value

Hardware Interfacing Instruction:

Arduino UNO	**Arduino Uno** | **PoT** A0 | PoT	Refer Page No:51
RPI PICO		Refer Page No:55
ESP32	**ESP32** | **PoT** GP35 | PoT	Refer Page No:58
STM32	**STM32** | **PoT** PA0 | PoT	Refer Page No:61

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Above in Introduction to Software	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:68
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:68
ESP32	Refer Example code in Section B - Codes	Refer Page No:68
STM32	Refer Example code in Section B - Codes	Refer Page No:68

6. Interfacing BMP280 Sensor

Hardware Interfacing Instruction:

Arduino UNO

Arduino Uno	BMP280
3.3v	VCC 3.3v
GND	GND
A5	SCL
A4	SDA

Refer Page No:51

ESP32

ESP32	BMP280
3.3v	VCC 3.3v
GND	GND
GPIO 22	SCL
GPIO 21	SDA

Refer Page No:58

PICO

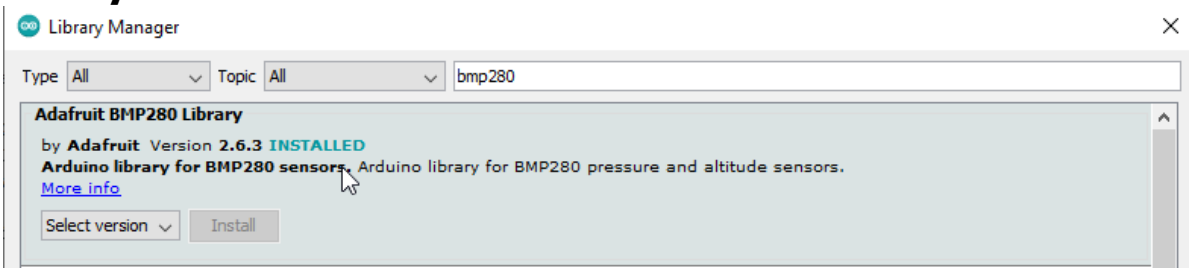
PICO	BMP280
3.3v	VCC 3.3v
GND	GND
GP10	SCL
GP11	SDA

STM32

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Example code in GitHub	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

Library Installation



Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:69
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:70
ESP32	Refer Example code in Section B - Codes	Refer Page No:69
STM32	Refer Example code in Section B - Codes	Refer Page No:69

7. Interfacing of MPU6050

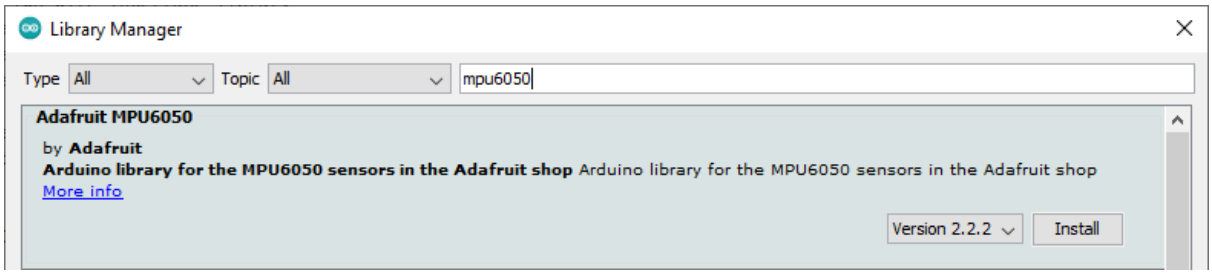
Hardware Interfacing Instruction:

Arduino UNO	**Arduino Uno** / **MPU6050** 3.3v / VCC 3.3v GND / GND A5 / SCL A4 / SDA	Refer Page No:52
ESP32	**ESP32** / **MPU6050** 3.3v / VCC 3.3v GND / GND GPIO 22 / SCL GPIO 21 / SDA	
PICO	**PICO** / **MPU6050** 3.3v / VCC 3.3v GND / GND GP10 / SCL GP11 / SDA	
STM32		

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Example code in GitHub	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

Library Installation



Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:72
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:74
ESP32	Refer Example code in Section B - Codes	Refer Page No:72
STM32	Refer Example code in Section B - Codes	Refer Page No:72

8. Interfacing of TFT Screen

Hardware Interfacing Instruction:

Arduino UNO

Arduino Uno	TFT 1.8’
5v	VCC 5 v
GND	GND
D10	CS
D8	RST
D9	A0
D11	SDA
D13	SCK
3.3V	LED

ESP32

ESP32	TFT 1.8’
5v	VCC 5 v
GND	GND
GPIO 12	CS
GPIO 14	RST
GPIO 13	A0
GPIO 21	SDA
GPIO 22	SCK
3.3v	LED

PICO

PICO	TFT 1.8’
5v & 3.3v	VCC 5 v
GND	GND
GP18	CS
GP17	RST
GP16	A0
GP11	SDA
GP10	SCK
3.3v	LED

STM32

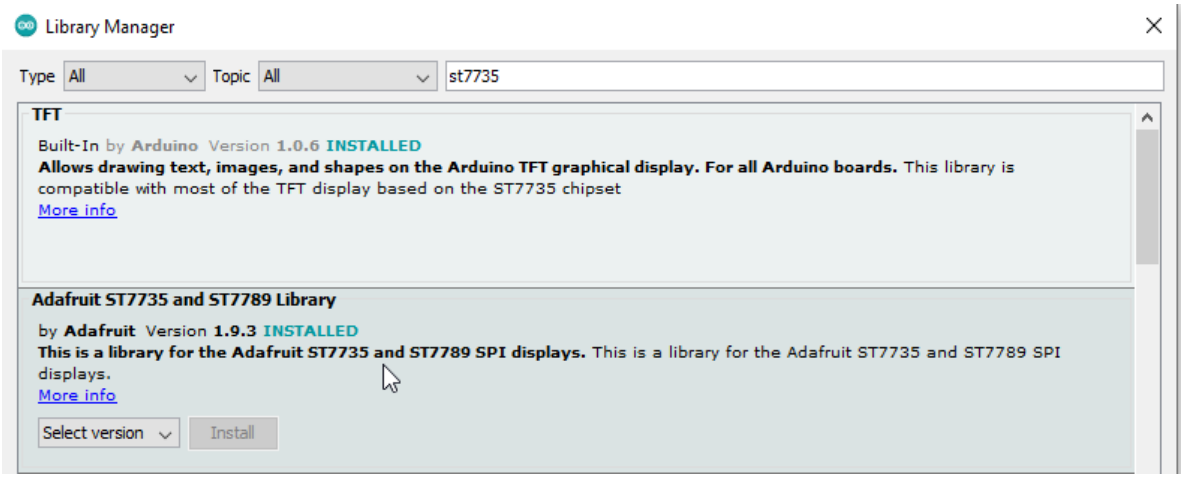
PICO	TFT 1.8’
5v & 3.3v	VCC 5 v
GND	GND
PA2	CS
PA0	RST
PA1	A0
PA7	SDA
PA5	SCK
3.3v	LED

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
RPI PICO	Refer Example code in GitHub	Refer Page No:24
ESP32	Refer Above in Introduction to Software	Refer Page No:13
STM32	Refer Above in Introduction to Software	Refer Page No:19

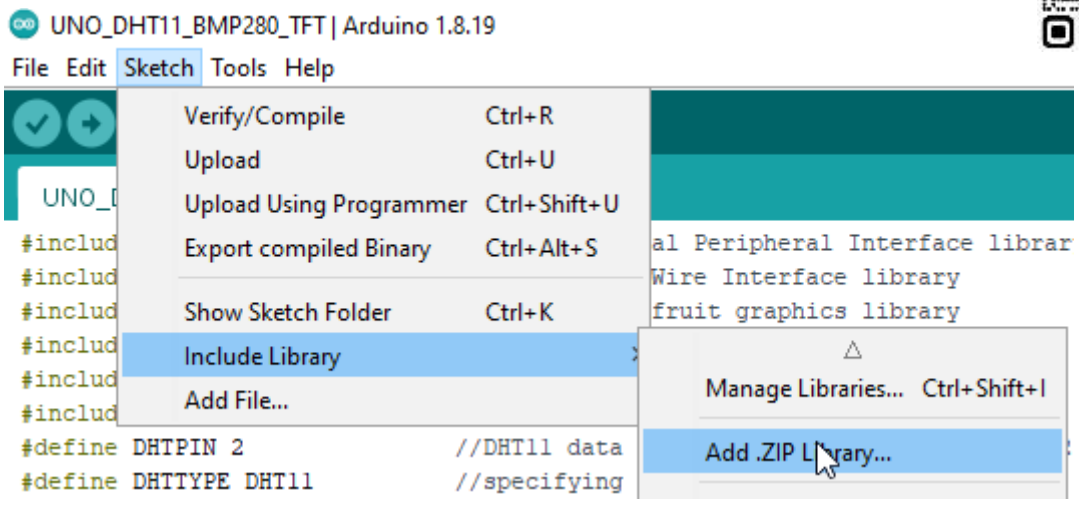
Library Installation

ESP32 Library



STM32

Scan & Install the Library by External File developed by IobiT Solutions



Code

Arduino UNO	Refer Example code in Section B - Codes	Refer Page No:74
RPI PICO	Refer Example code in Section B - Codes	Refer Page No:80
ESP32	Refer Example code in Section B - Codes	Refer Page No:75
STM32	Refer Example code in Section B - Codes	Refer Page No:79

IoT Architecture

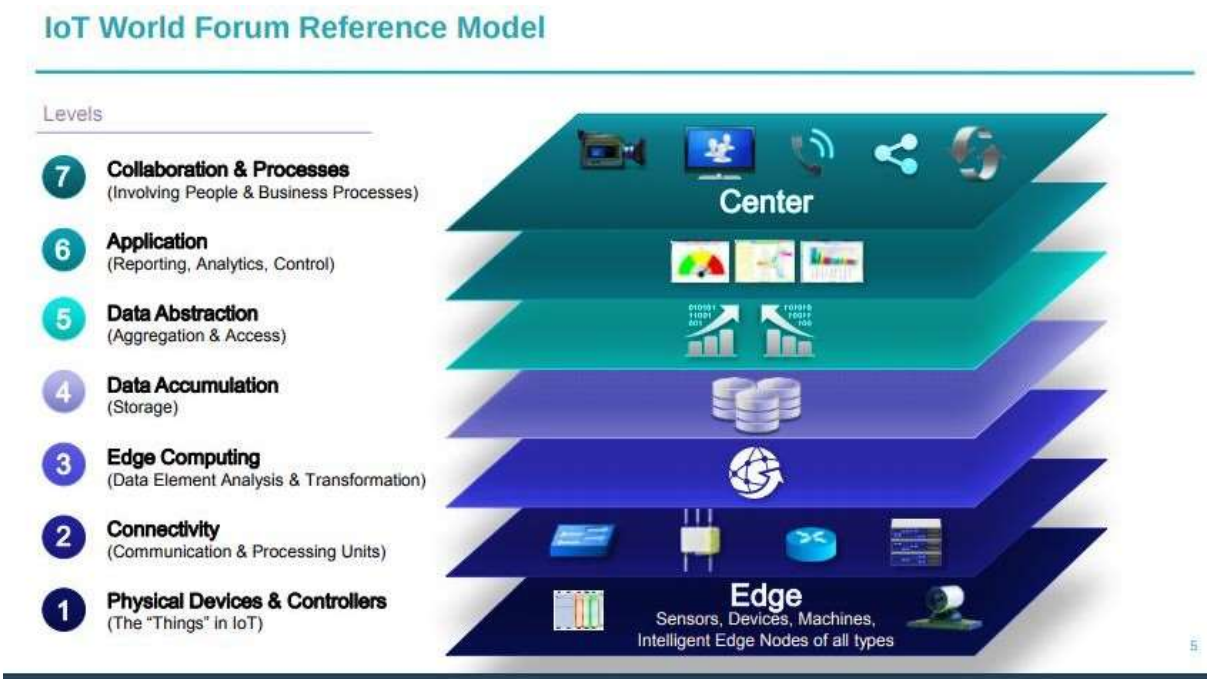
IoT solutions have become a regular part of our lives. From the smartwatch on your wrist to industrial enterprises, connected devices are everywhere. Having *things* work for us is no longer sci-fi fantasy.

You tap the screen of your smartphone or say a word, and get immediate results. A door automatically opens, a coffee machine starts grinding beans to make a perfect cup of espresso while you receive analytical reports based on fresh data from sensors miles away.

But between your command and tasks fulfilled, there lies a large and mostly invisible infrastructure, that involves multiple elements and interactions. This article describes IoT — the Internet of Things — through its architecture, layer to layer. Let's peek behind the curtain to see how everyday magic works.

Major IoT building blocks and layers

Before we go any further, it's worth pointing out that there is no single, agreed -upon IoT architecture. It varies in complexity and number of architectural layers depending on a particular business task.

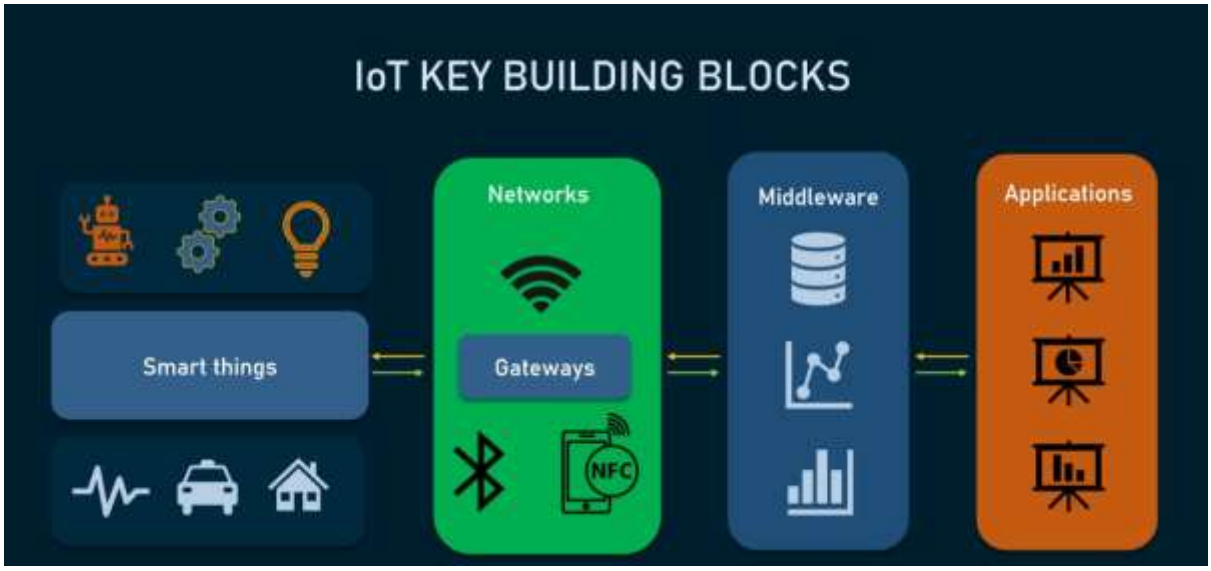


But no matter the use case and number of layers, the key building blocks of any IoT structure are always the same, namely:

- **Smart things;**
- **Networks** and **Gateways** enabling low-power devices (which is often the case in IoT) to enter the big Internet;
- the **Middleware** or **IoT platforms** providing data storage spaces and advanced computing engines along with analytical capabilities; and
- **Applications**, allowing end users to benefit from IoT and manipulate the physical world.

These elements make up the backbone of any IoT system upon which effective, multi-layered architecture can be developed. Most commonly, these layers are:

- The **perception layer** hosting smart things;
- The **connectivity or transport** layer transferring data from the physical layer to the cloud and vice versa via networks and gateways;
- The **processing layer** employing IoT platforms to accumulate and manage all data streams; and
- The **application layer** delivering solutions like analytics, reporting, and device control to end users.



Besides the most essential components, the article also describes three additional layers:

- The edge or fog computing layer performing data pre-processing close to the edge, where IoT things collect new information. Typically, edgy computing occurs on gateways;
- The business layer where businesses make decisions based on the data; and
- The security layer encompassing all other layers.

Perception layer: converting analog signals into digital data and vice versa

The initial stage of any IoT system embraces a wide range of "things" or endpoint devices that act as a bridge between the real and digital worlds. They vary in form and size, from tiny silicon chips to large vehicles. By their functions, IoT things can be divided into the following large groups.

Sensors such as probes, gauges, meters, and others. They collect physical parameters like temperature or humidity, turn them into electrical signals, and send them to the IoT system. IoT sensors are typically small and consume little power.

Actuators, translating electrical signals from the IoT system into physical actions. Actuators are used in motor controllers, lasers, robotic arms.

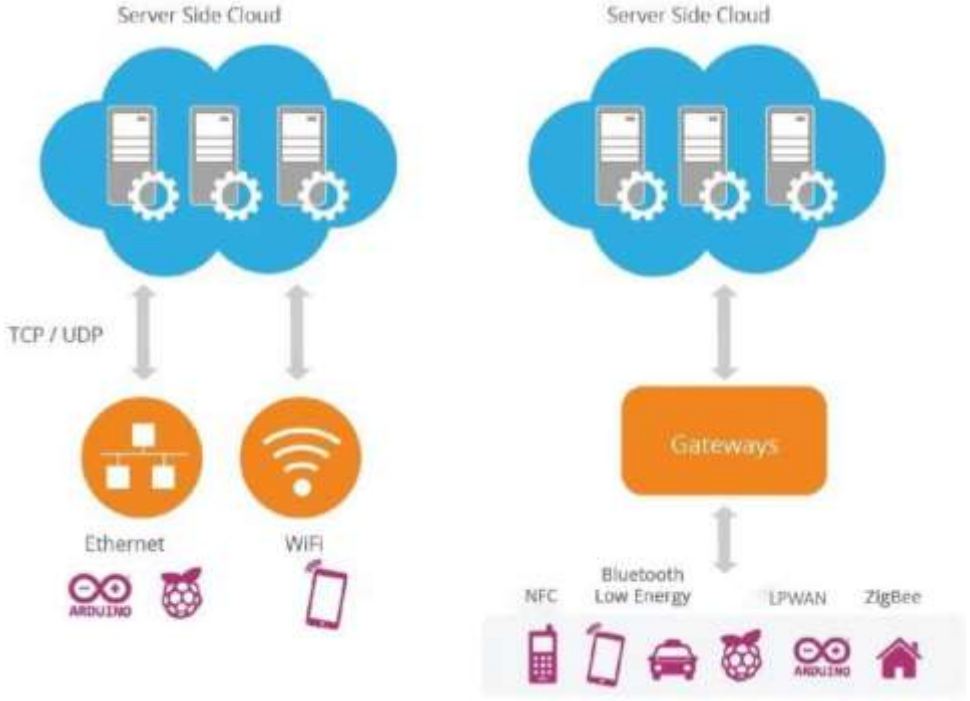
Machines and devices connected to sensors and actuators or having them as integral parts.

It's important to note that the architecture puts no restriction on the scope of its components or their location. The edge-side layer can include just a few "things" physically placed in one room or myriads of sensors and devices distributed across the world.

Connectivity layer: enabling data transmission

The second level is in charge of all communications across devices, networks, and cloud services that make up the IoT infrastructure. The connectivity between the physical layer and the cloud is achieved in two ways:

- directly, using TCP or UDP/IP stack;
- via gateways — hardware or software modules performing translation between different protocols as well as encryption and decryption of IoT data.



The communications between devices and cloud services or gateways involve different networking technologies.

Ethernet connects stationary or fixed IoT devices like security and video cameras, permanently installed industrial equipment, and gaming consoles.

WiFi, the most popular technology of wireless networking, is a great fit for data-intensive IoT solutions that are easy to recharge and operate within a small area. A good example of use is smart home devices connected to the electrical grid.

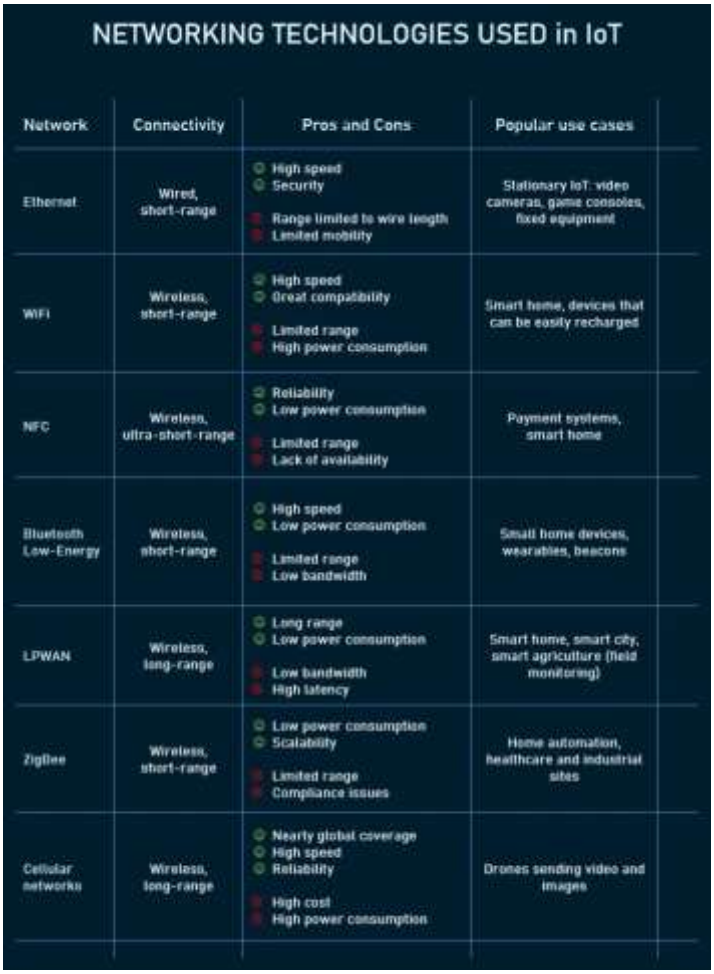
NFC (Near Field Communication) enables simple and safe data sharing between two devices over a distance of 4 inches (10 cm) or less.

Bluetooth is widely used by wearables for short-range communications. To meet the needs of low-power IoT devices, the Bluetooth Low-Energy (BLE) standard was designed. It transfers only small portions of data and doesn't work for large files.

LPWAN (Low-power Wide-area Network) was created specifically for IoT devices. It provides long-range wireless connectivity on low power consumption with a battery life of 10+ years. Sending data periodically in small portions, the technology meets the requirements of smart cities, smart buildings, and smart agriculture (field monitoring).

ZigBee is a low-power wireless network for carrying small data packages over short distances. The outstanding thing about ZigBee is that it can handle up to 65,000 nodes. Created specifically for home automation, it also works for low-power devices in industrial, scientific, and medical sites.

Cellular networks offer reliable data transfer and nearly global coverage. There are two cellular standards developed specifically for IoT things. LTE-M (Long Term Evolution for Machines) enables devices to communicate directly with the cloud and exchange high volumes of data. NB-IoT or Narrowband IoT uses low-frequency channels to send small data packages.



Once parts of the IoT solution are networked, they still need messaging protocols to share data across devices and with the cloud. The most popular protocols used in the IoT ecosystems are:

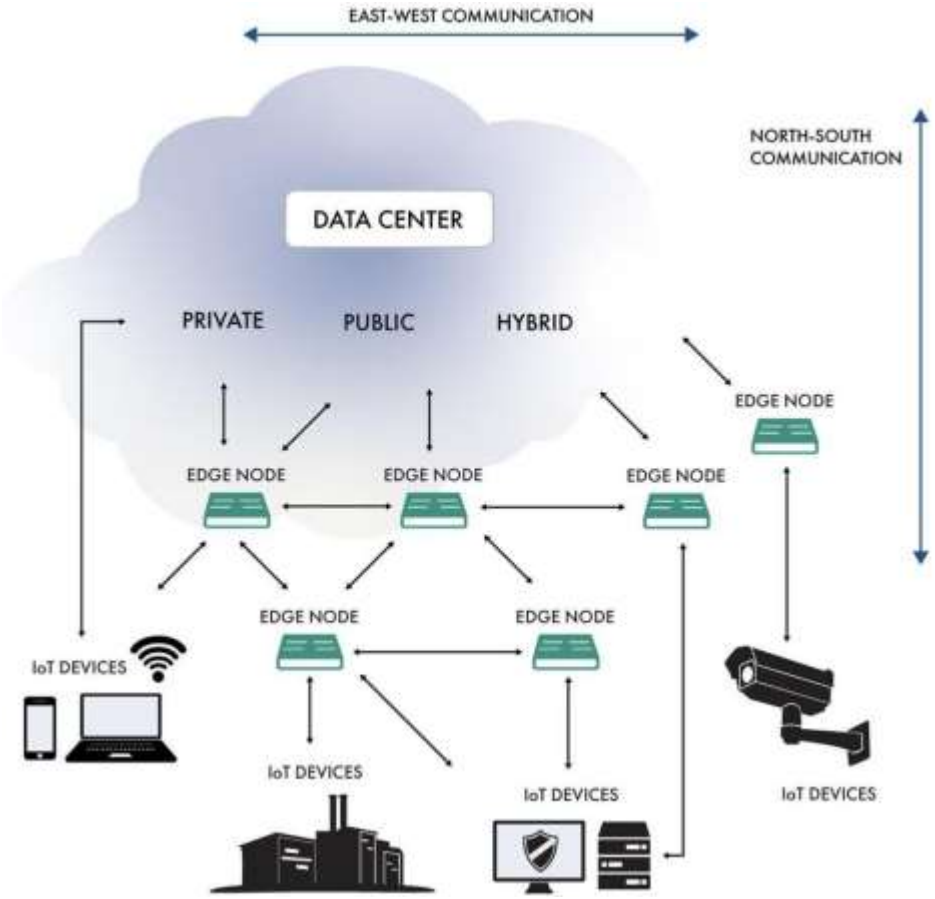
- **DDS** (the Data Distribution Service) which directly connects IoT things to each other and to applications addressing the requirements of real-time systems;

- **AMQP** (the Advanced Message Queuing Protocol) aiming at peer-to-peer data exchange between servers;
- **CoAP** (the Constrained Application Protocol), a software protocol designed for constrained devices — end nodes limited in memory and power (for example, wireless sensors). It feels much like HTTP but uses fewer resources;
- **MQTT** (the Message Queue Telemetry Transport), a lightweight messaging protocol built on top of TCP/IP stack for centralized data collection from low-powered devices.

Edge or fog computing layer: reducing system latency

This level is essential for enabling IoT systems to meet the speed, security, and scale requirements of the 5th generation mobile network or 5G. The new wireless standard promises faster speeds, lower latency, and the ability to handle many more connected devices, than the current 4G standard.

The idea behind edge or fog computing is to process and store information as early and as close to its sources as possible. This approach allows for analyzing and transforming high volumes of real-time data locally, at the edge of the networks. Thus, you save the time and other resources that otherwise would be needed to send all data to cloud services. The result is reduced system latency that leads to real-time responses and enhanced performance.



Edge computing occurs on gateways, local servers, or other edge nodes scattered across the network. At this level, data can be:

- Evaluated to determine if it needs further processing at higher levels,
- Formatted for further processing,
- Decoded,
- Filtered, and
- Redirected to an additional destination

To sum up, the first three layers see data in motion, as it is constantly moving and altering. Only on hitting the next level, is data finally at rest and available for use by consumer applications.

Processing layer: making raw data useful

The processing layer accumulates, stores, and processes data that comes from the previous layer. All these tasks are commonly handled via IoT platforms and include two major stages.

Data accumulation stage

The real-time data is captured via an API and put at rest to meet the requirements of non-real-time applications. The data accumulation component stage works as a transit hub between event-based data generation and query-based data consumption.

Among other things, the stage defines whether data is relevant to the business requirements and where it should be placed. It saves data to a wide range of storage solutions, from data lakes capable of holding unstructured data like images and video streams to event stores and telemetry databases. The total goal is to sort out a large amount of diverse data and store it in the most efficient way.

Data abstraction stage

Here, data preparation is finalized so that consumer applications can use it to generate insights. The entire process involves the following steps:

- Combining data from different sources, both iot and non-iot, including ERM, ERP, and CRM systems;
- Reconciling multiple data formats; and
- Aggregating data in one place or making it accessible regardless of location through data virtualization.
- Similarly, data collected at the application layer is reformatted here for sending to the physical level so that devices can "understand" it.

Together, the data accumulation and abstraction stages veil details of the hardware, enhancing the interoperability of smart devices. What's more, they let software developers focus on solving particular business tasks — rather than on delving into the specifications of devices from different vendors.

Application layer: addressing business requirements

At this layer, information is analyzed by software to give answers to key business questions. There are hundreds of IoT applications that vary in complexity and function, using different technology stacks and operating systems. Some examples are:

- Device monitoring and control software,

- Mobile apps for simple interactions,
- Business intelligence services, and
- Analytic solutions using machine learning.

Currently, applications can be built right on top of IoT platforms that offer software development infrastructure with ready-to-use instruments for data mining, advanced analytics, and data visualization. Otherwise, IoT applications use APIs to integrate with middleware.

Business layer: implementing data-driven solutions

The information generated at the previous layers brings value if only it results in problem-solving solution and achieving business goals. New data must initiate collaboration between stakeholders who in turn introduce new processes to enhance productivity.

The decision-making usually involves more than one person working with more than one software solution. For this reason, the business layer is defined as a separate stage, higher than a single application layer.

Security layer: preventing data breaches

It goes without saying that there should be a security layer covering all the above-mentioned layers. IoT security is a broad topic worthy of a separate article. Here we'll only point out the basic features of the safe architecture across different levels.

Device security. Modern manufacturers of IoT devices typically integrate security features both in the hardware and firmware installed on it. This includes

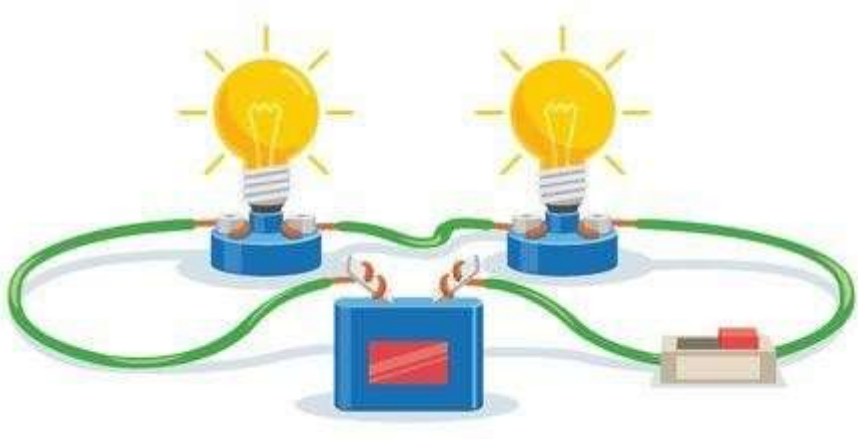
- Embedded TPM (Trusted Platform Module) chips with cryptographic keys for authentication and protection of endpoint devices;
- A secure boot process that prevents unauthorized code from running on a powered-up device;
- Updating security patches on a regular basis; and
- Physical protection like metal shields to block physical access to the device.

Connection security. Whether data is being sent over devices, networks, or applications, it should be encrypted. Otherwise, sensitive information can be read by anybody who intercepts information in transit. IoT-centric messaging protocols like MQTT, AMQP, and DDS may use standard Transport Layer Security (TSL) cryptographic protocol to ensure end-to-end data protection.

Cloud security. Data at rest stored in the cloud must be encrypted as well to mitigate risks of exposing sensitive information to intruders. Cloud security also involves authentication and authorization mechanisms to limit access to the IoT applications. Another important security method is device identity management to verify the device's credibility before allowing it to connect to the cloud.

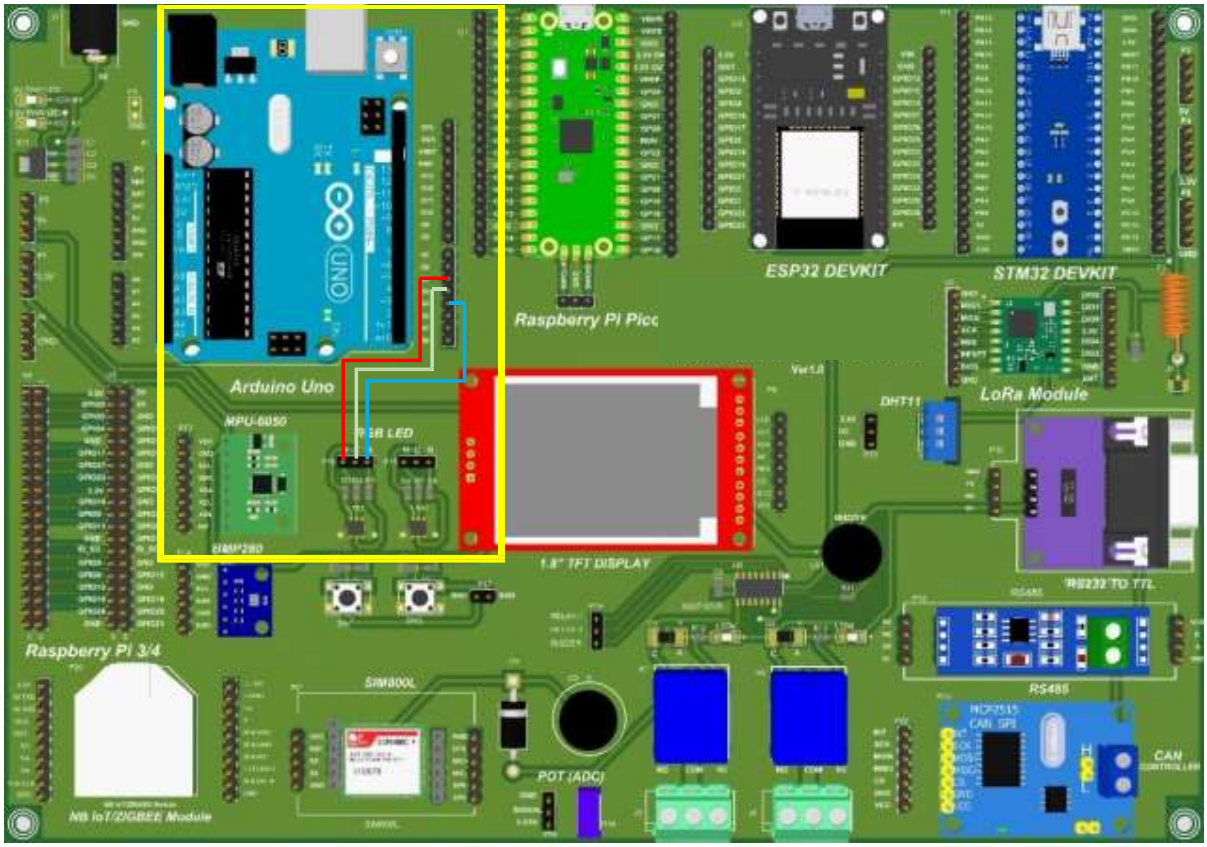
The good news is that IoT solutions from large providers like Microsoft, AWS, or Cisco come with pre-built protection measures including end-to-end data encryption, device authentication, and access control. However, it always pays to ensure that security is tight at all levels, from the tiniest devices to complex analytical systems.

Section A – Hardware Interfacing Instructions

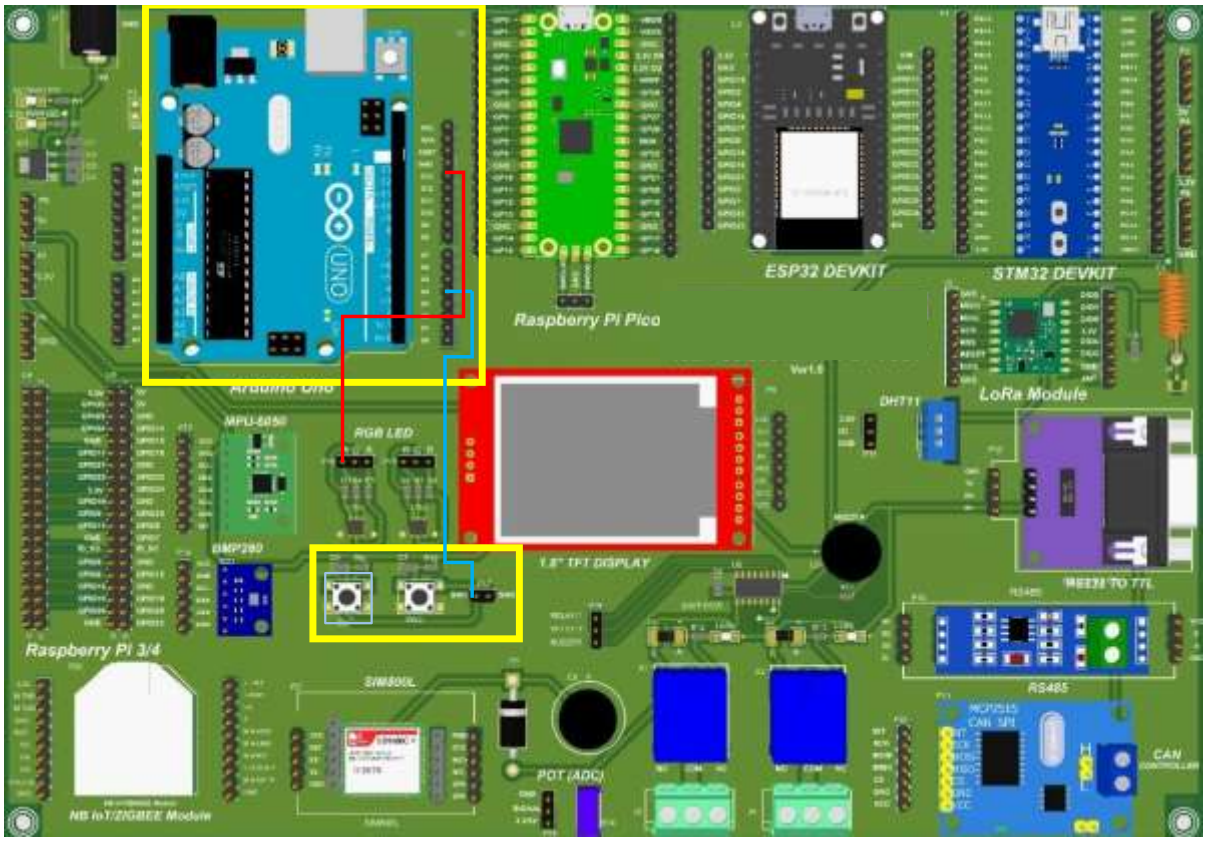


(Arduino UNO)

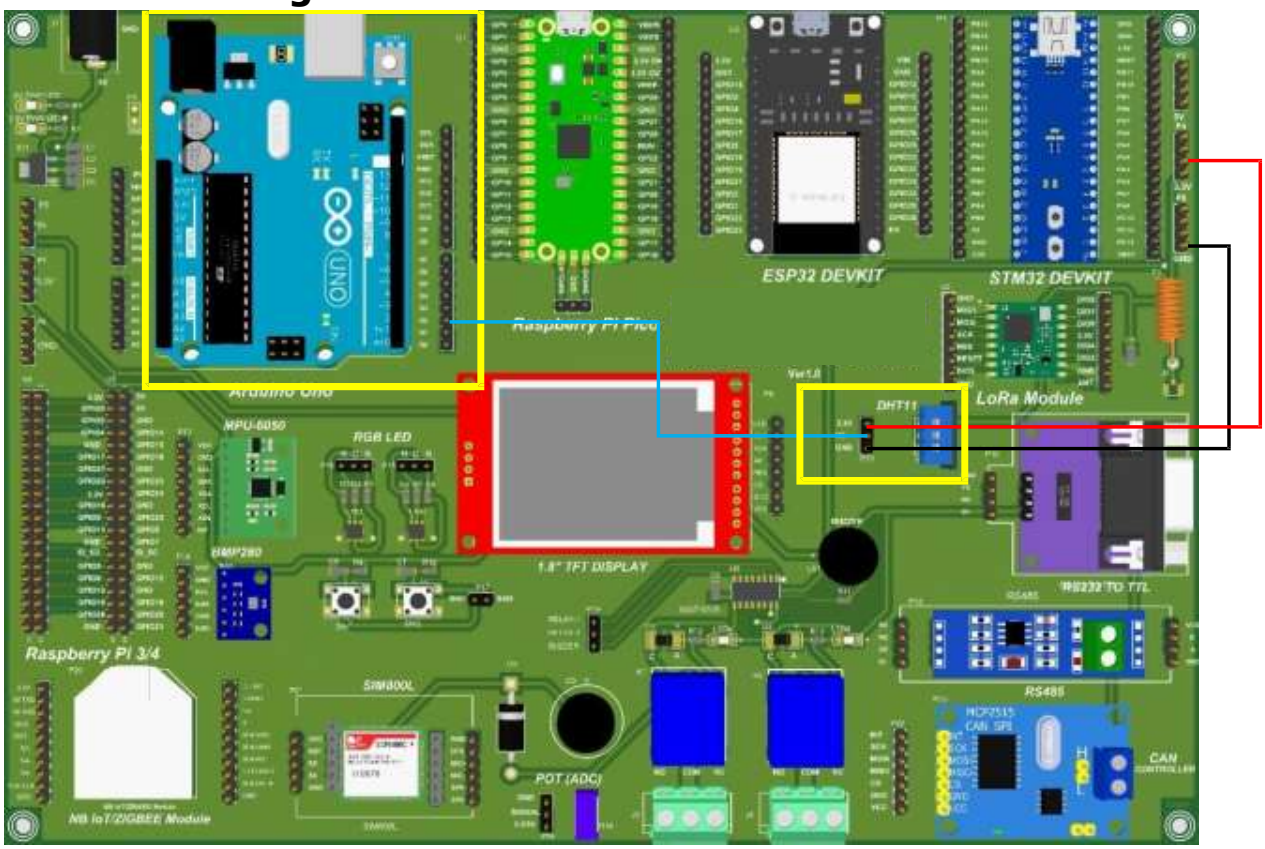
1. Blink a RGB LED



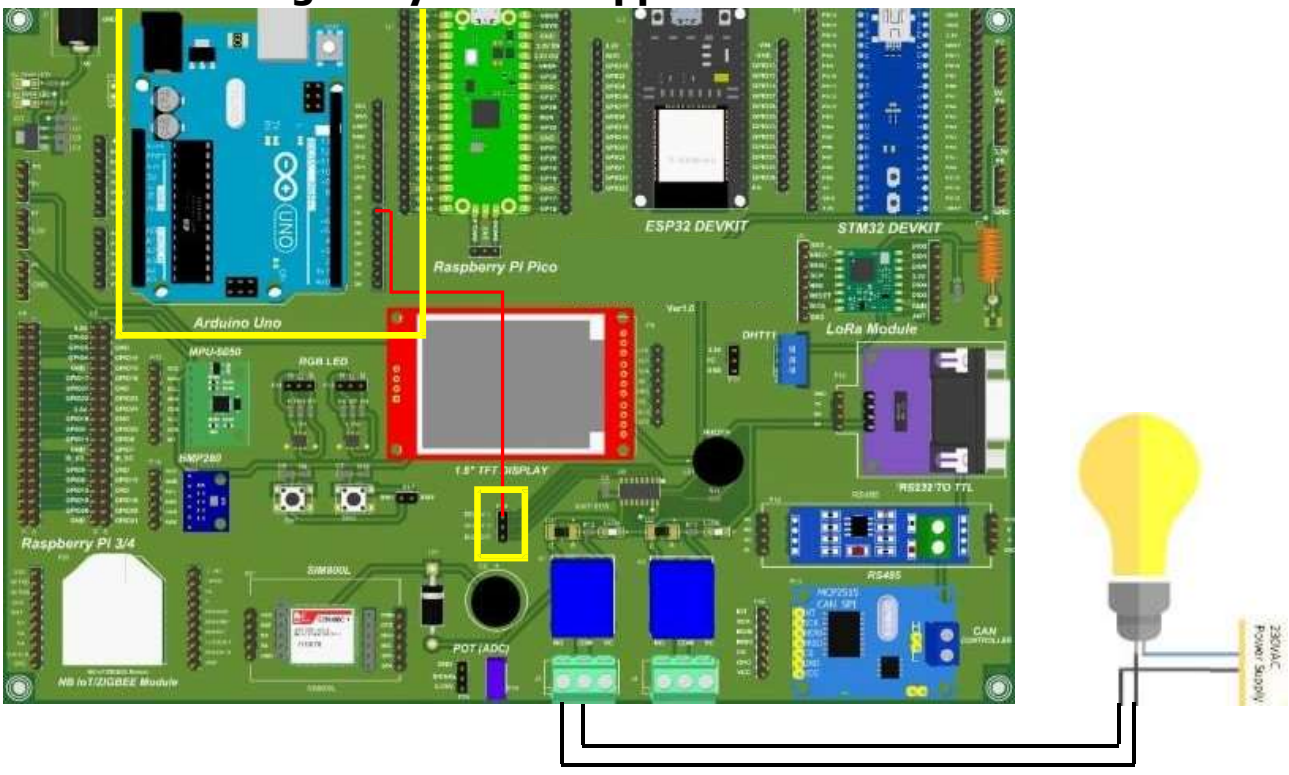
2. Pushbutton with an LED



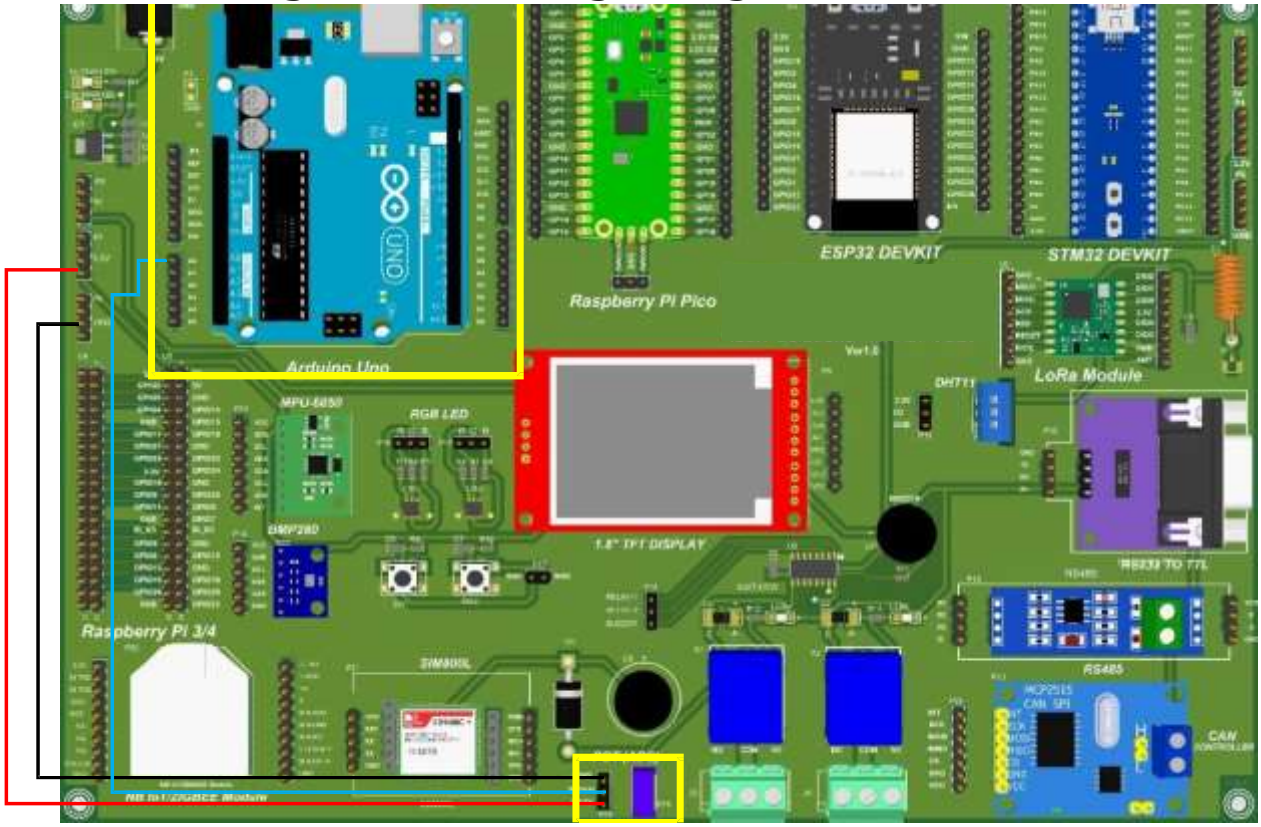
3. Interfacing DHT11



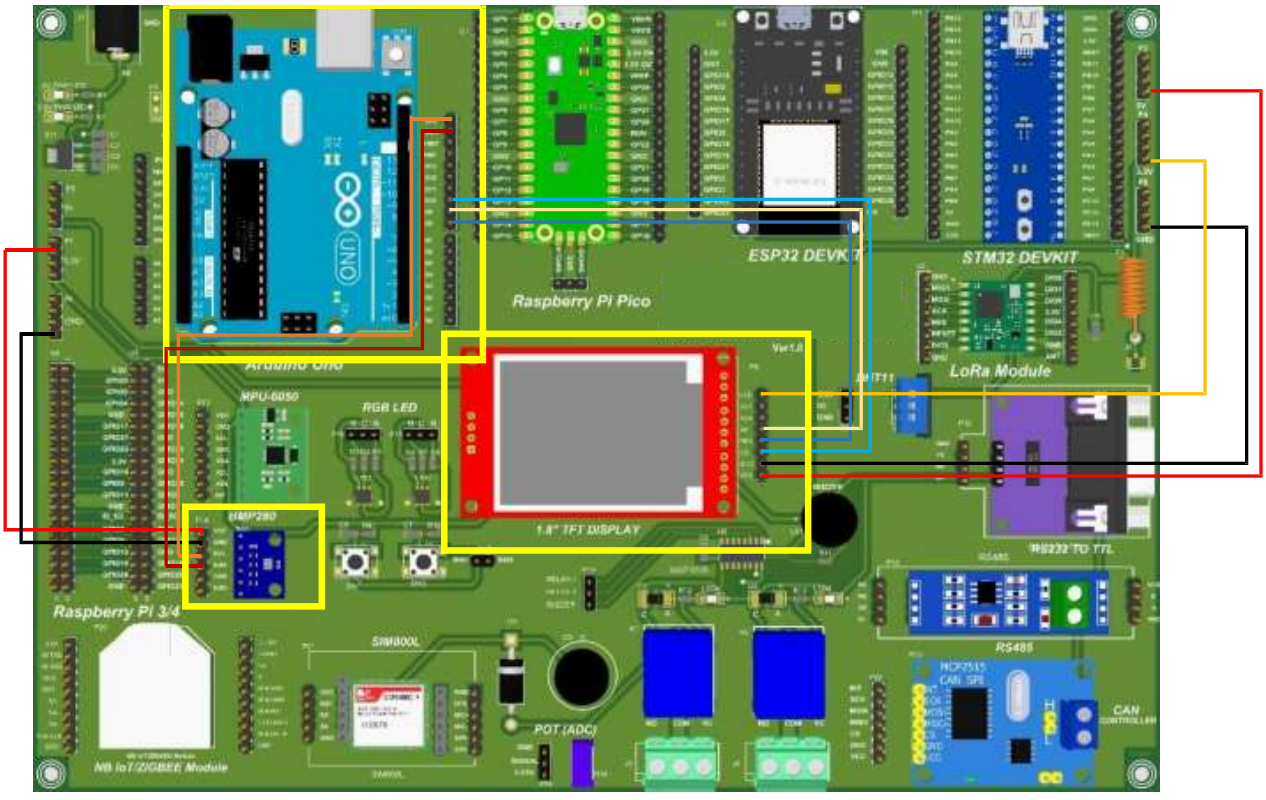
4. Interfacing Relay with AC Appliances



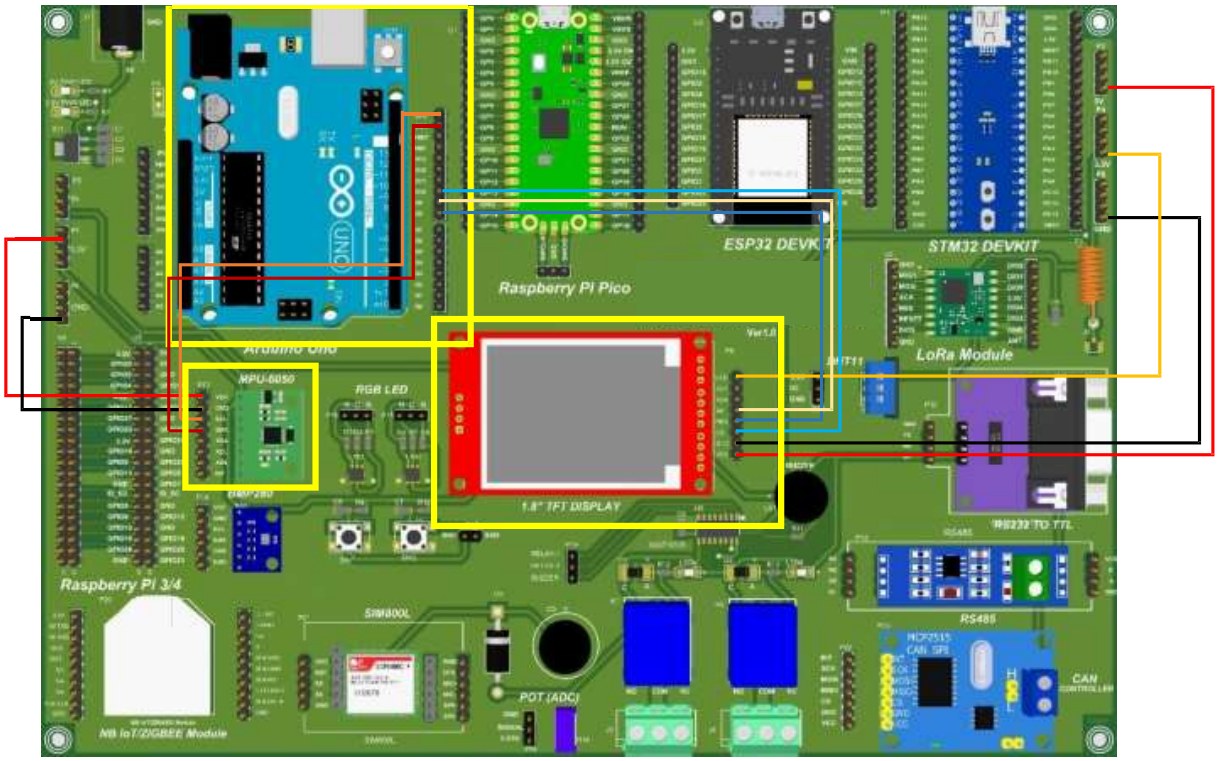
5. Interfacing PoT for Reading Analog Value



6. Interfacing TFT Display with BMP280 Sensor

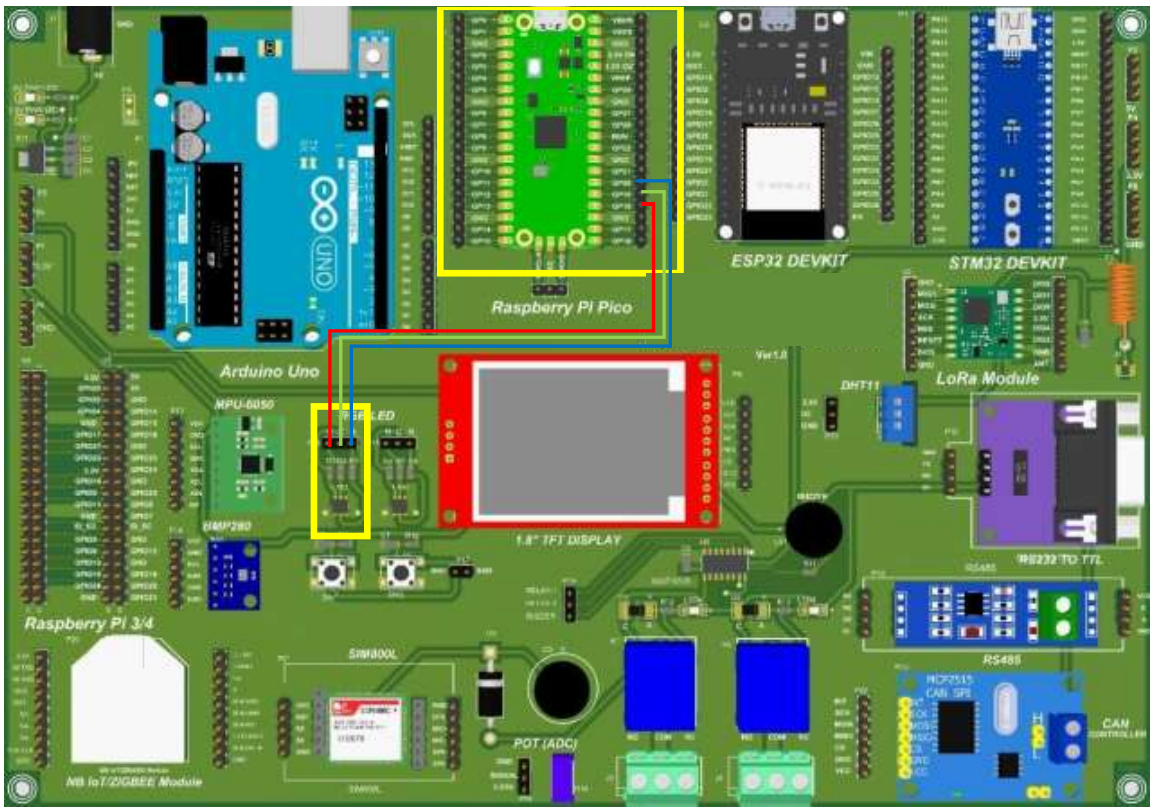


7. Interfacing TFT Display with MPU6050 Sensor

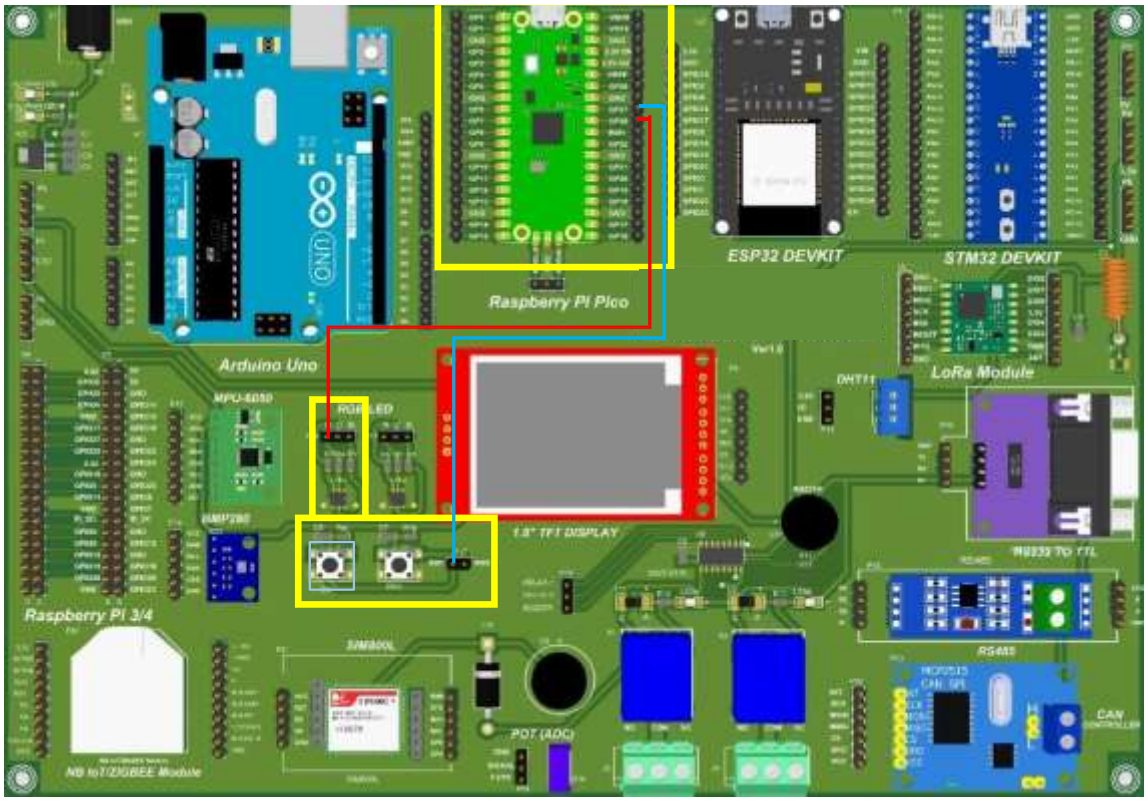


(RPI - PICO)

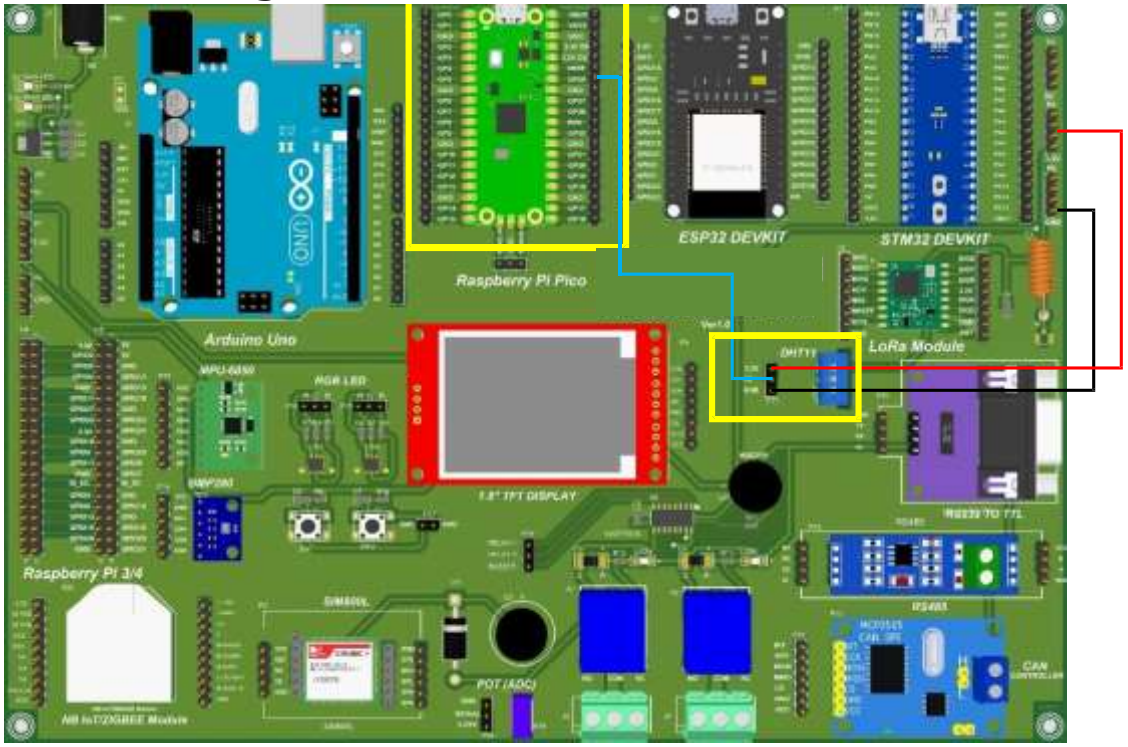
1. Blink a RGB LED



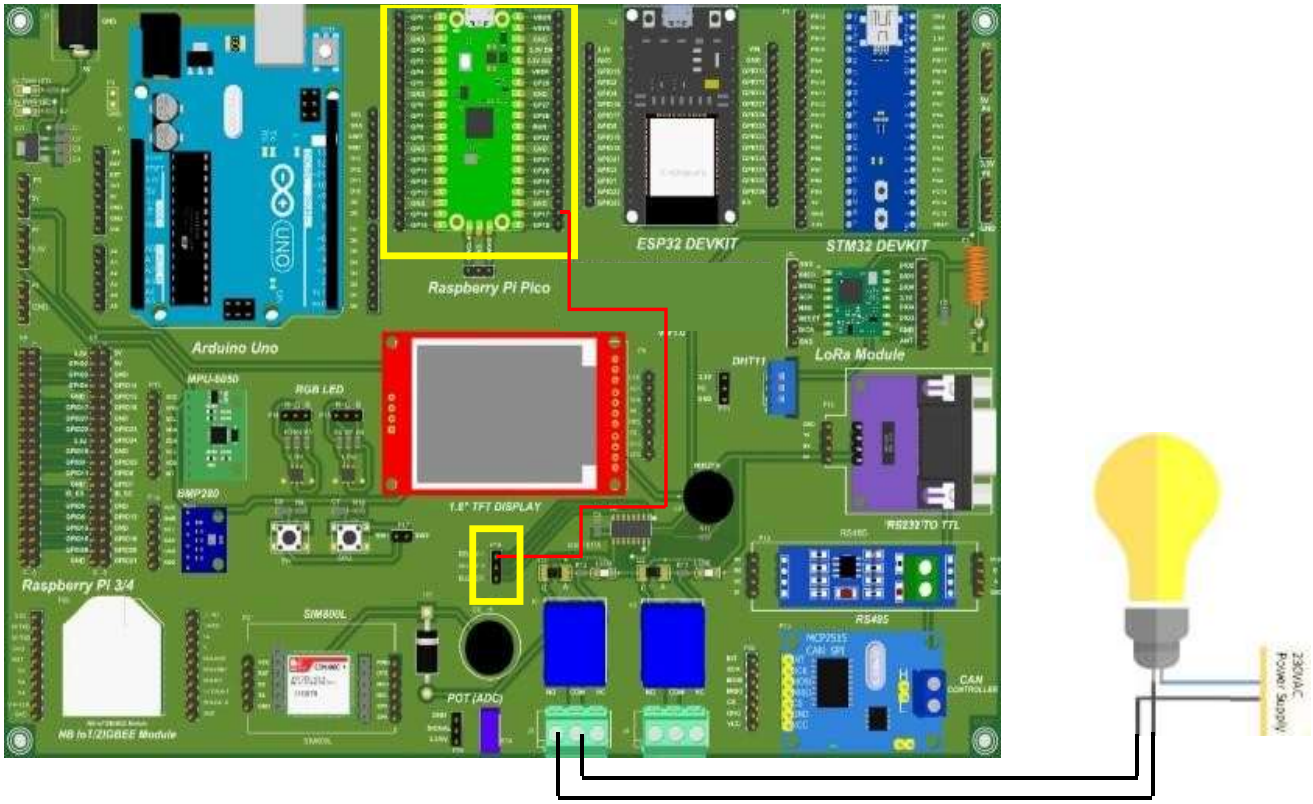
2. Pushbutton with an LED



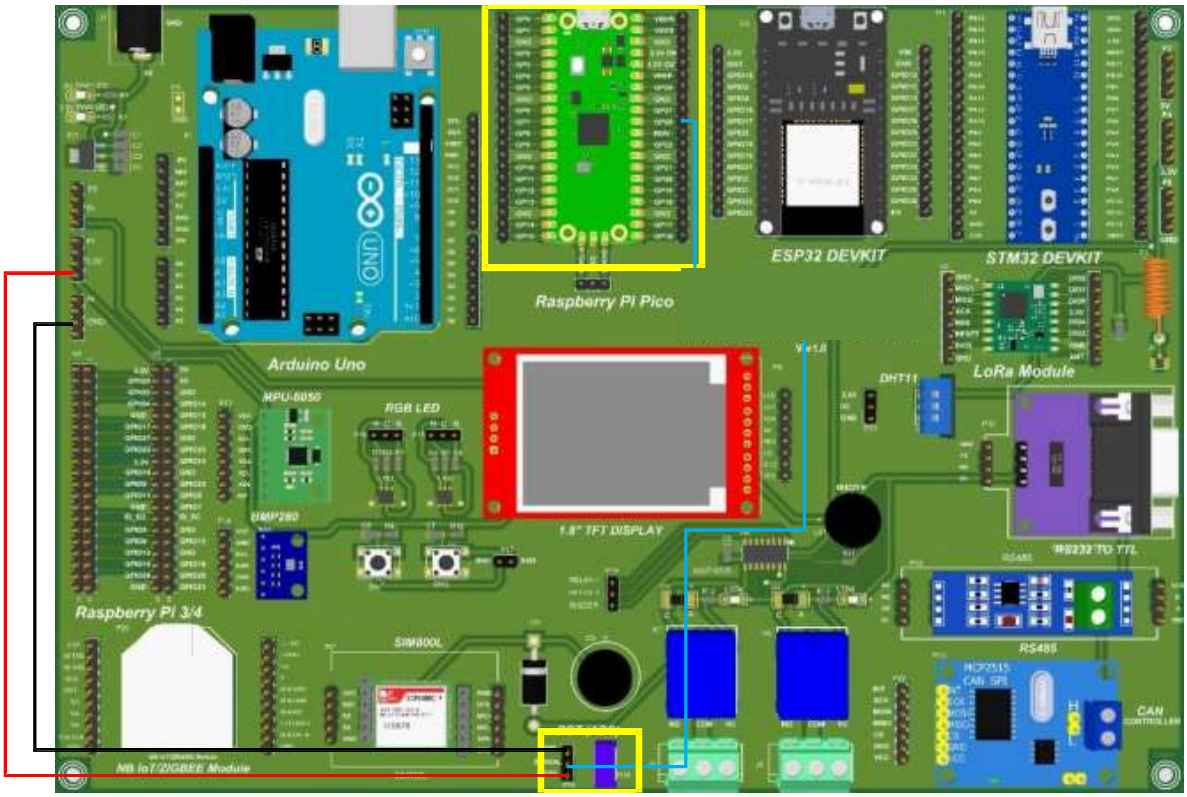
3. Interfacing DHT11



4. Interfacing Relay with AC Appliances

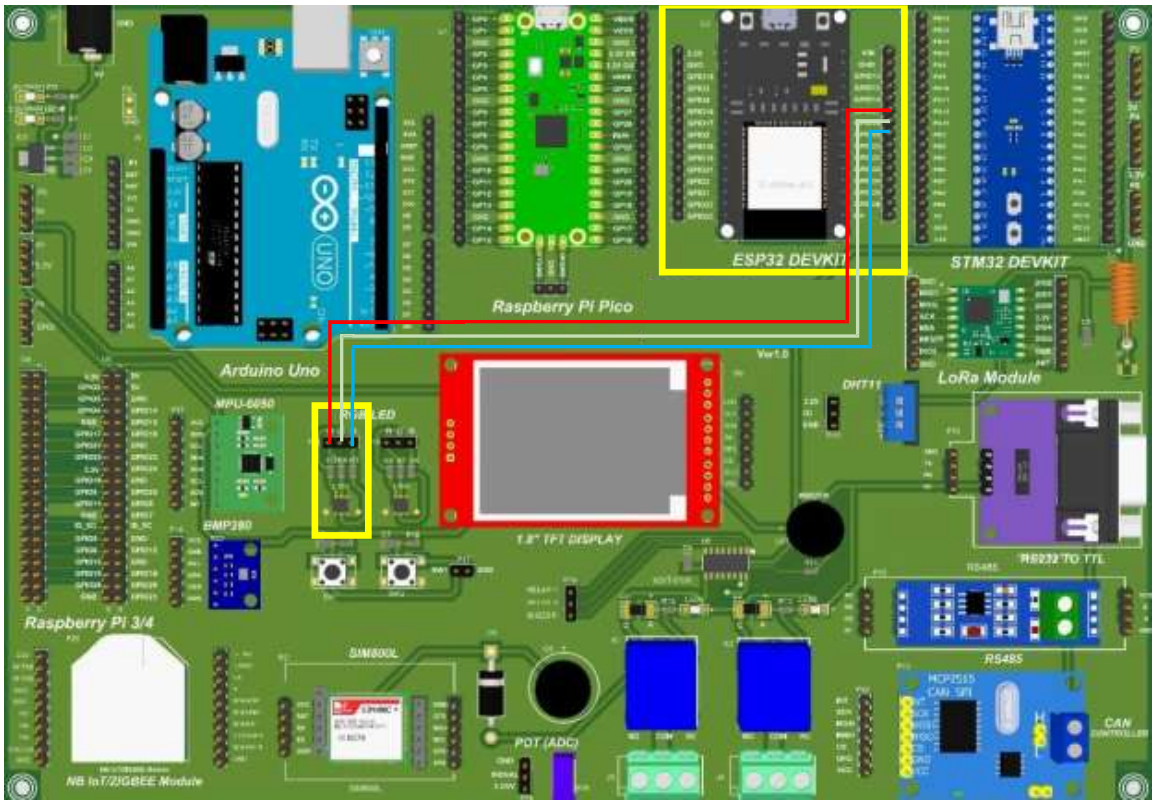


5. Interfacing PoT for Reading Analog Value

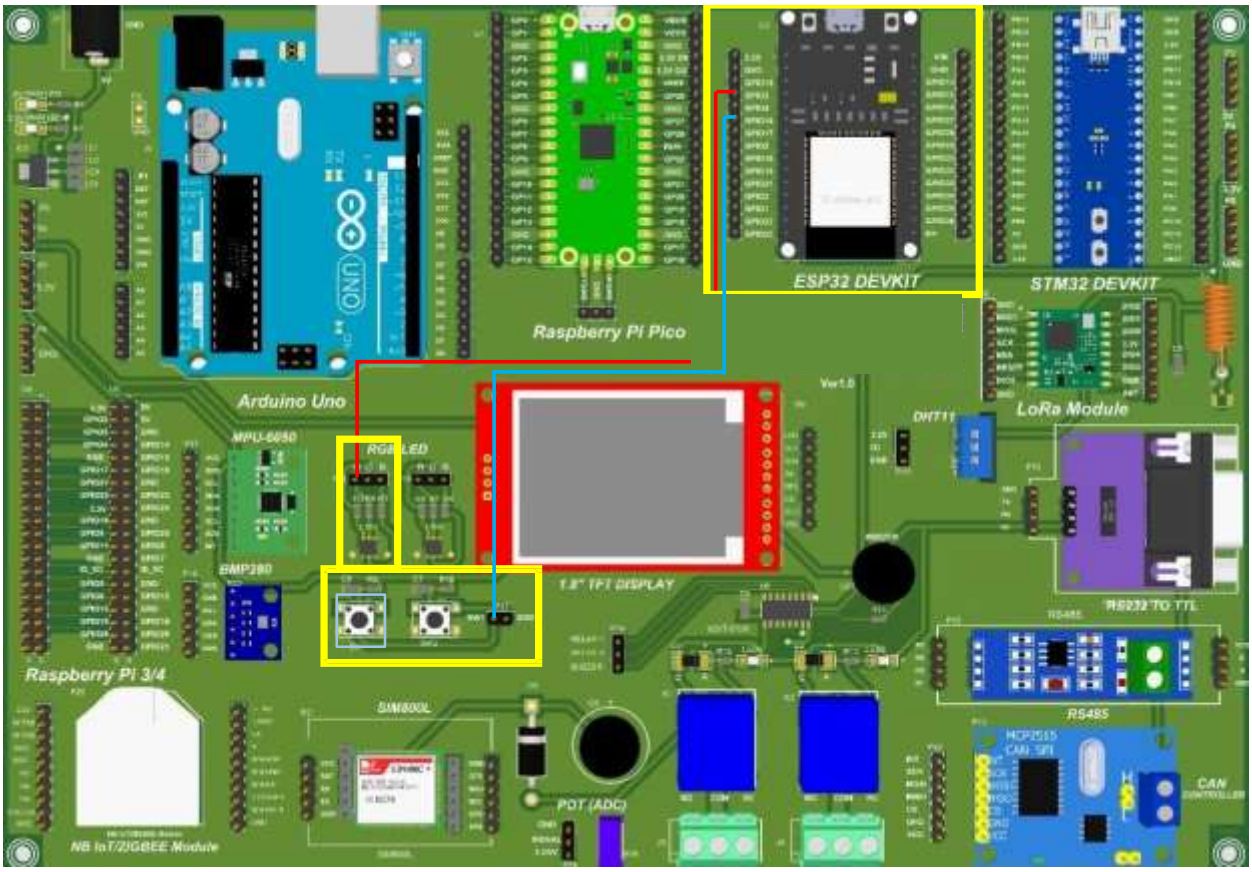


(ESP32)

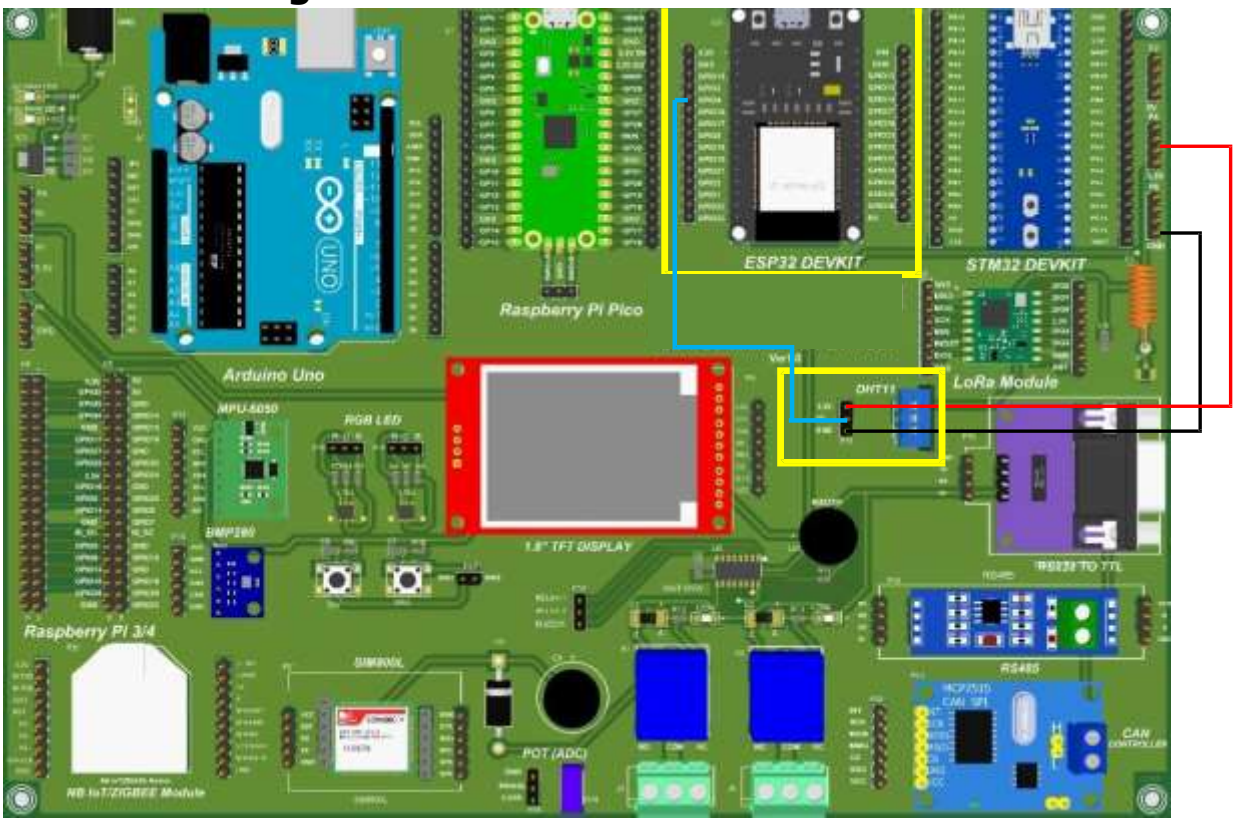
1. Blink a RGB LED



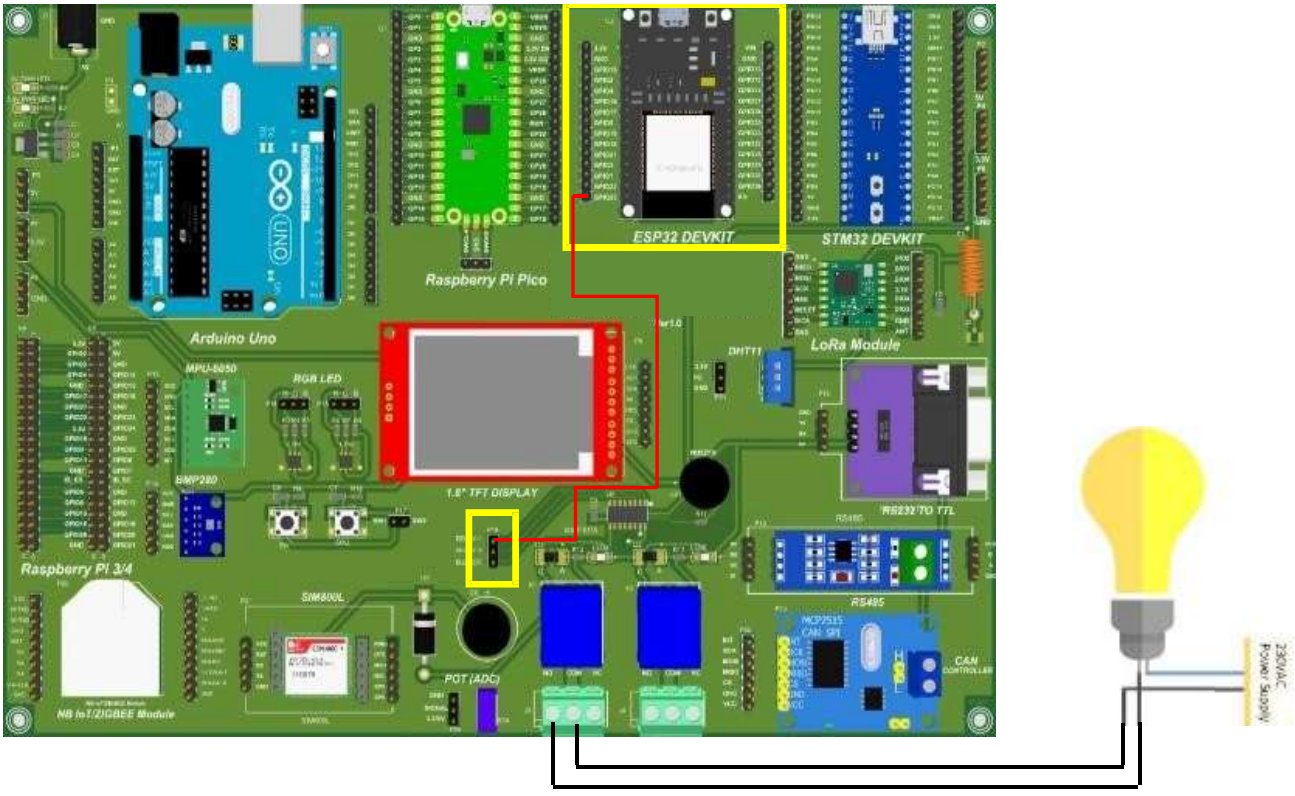
2. Pushbutton with an LED



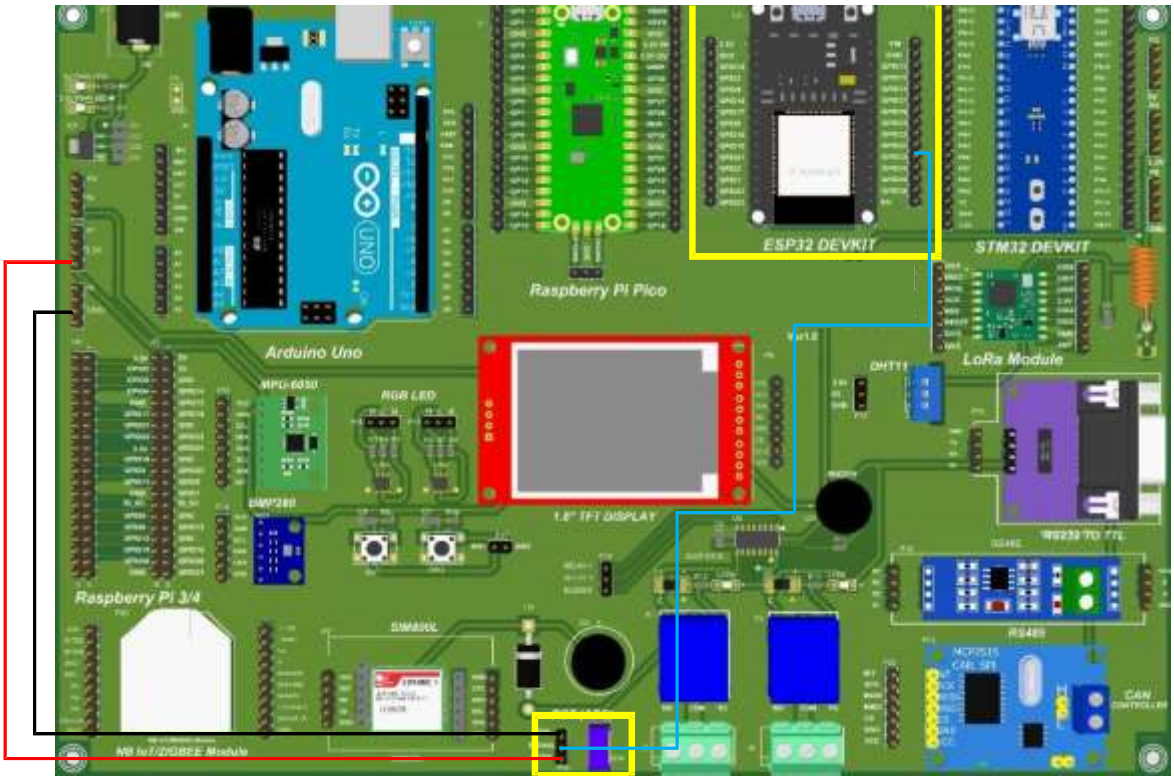
3. Interfacing DHT11



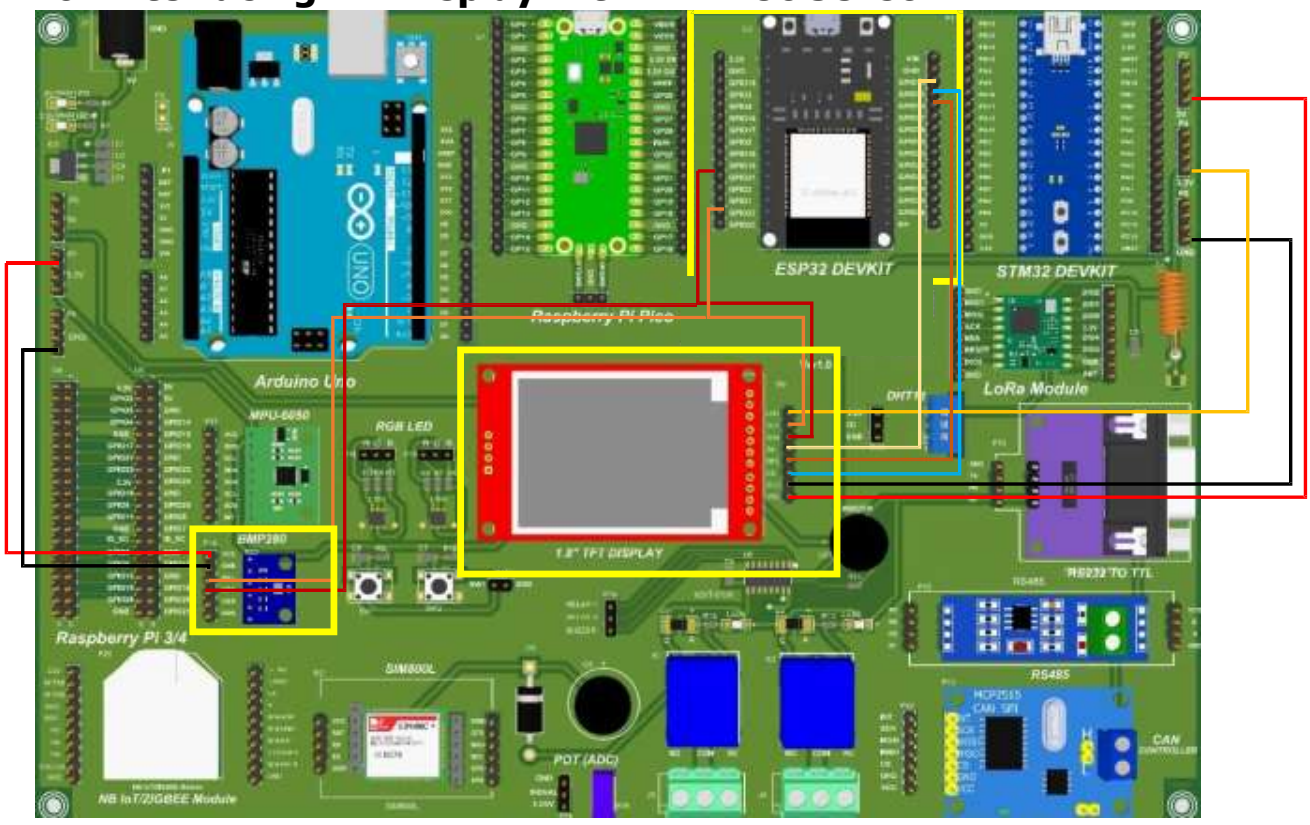
4. Interfacing Relay with AC Appliances



5. Interfacing PoT for Reading Analog Value

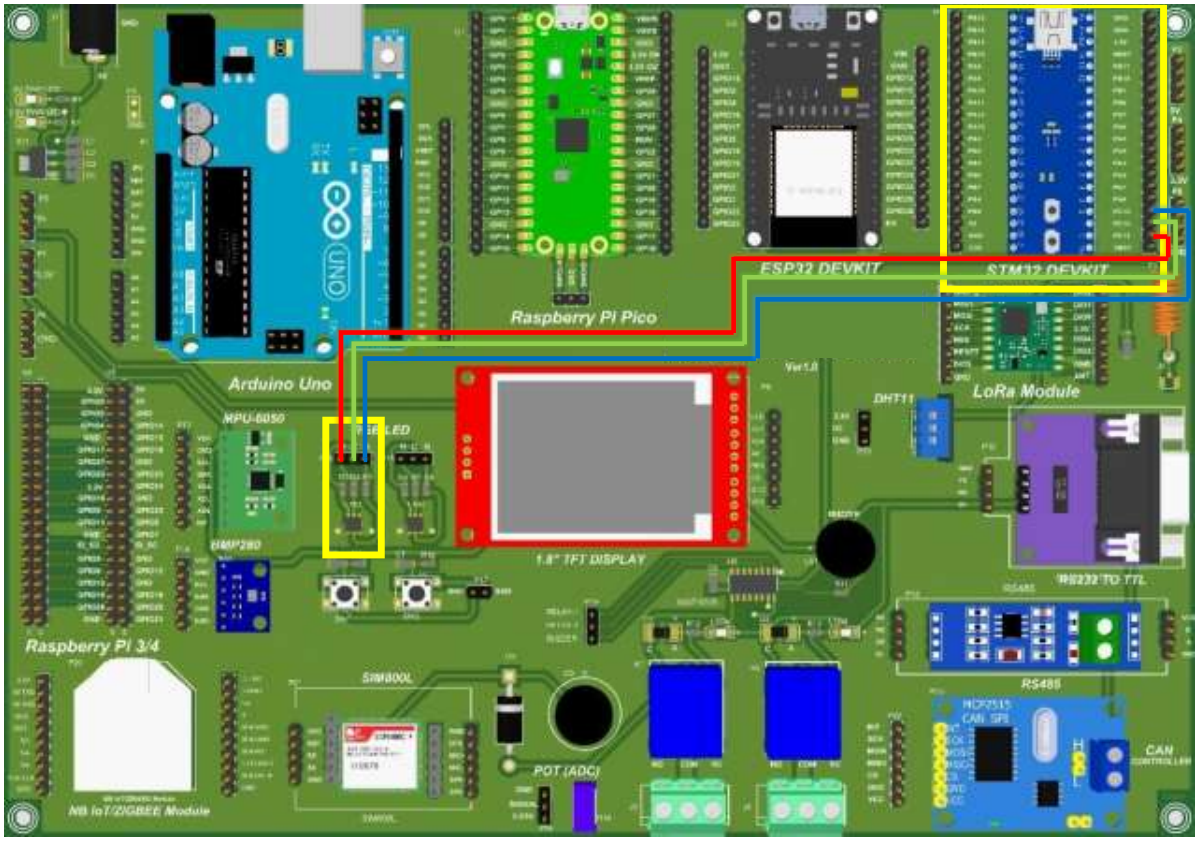


6. Interfacing TFT Display with BMP280 Sensor

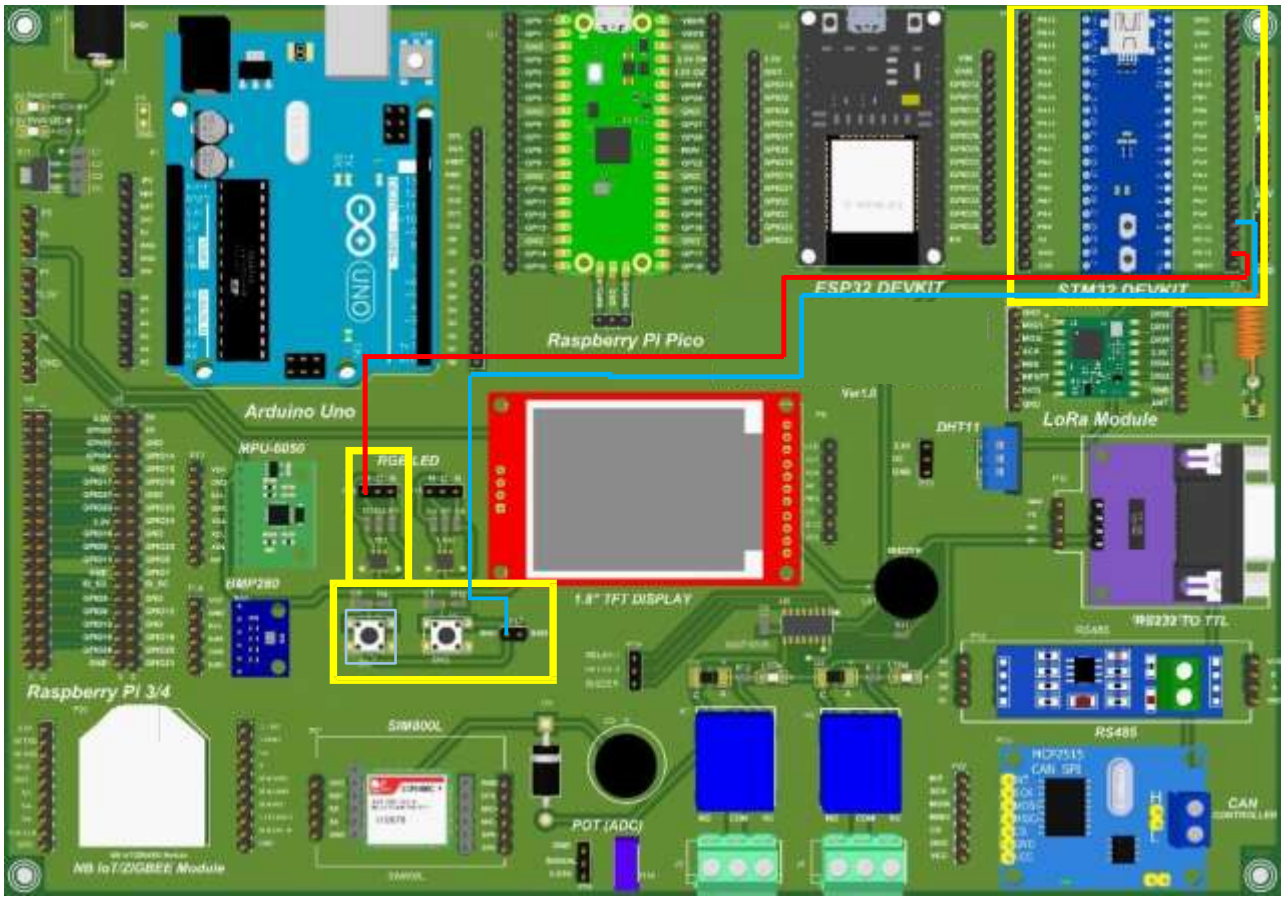


(STM32)

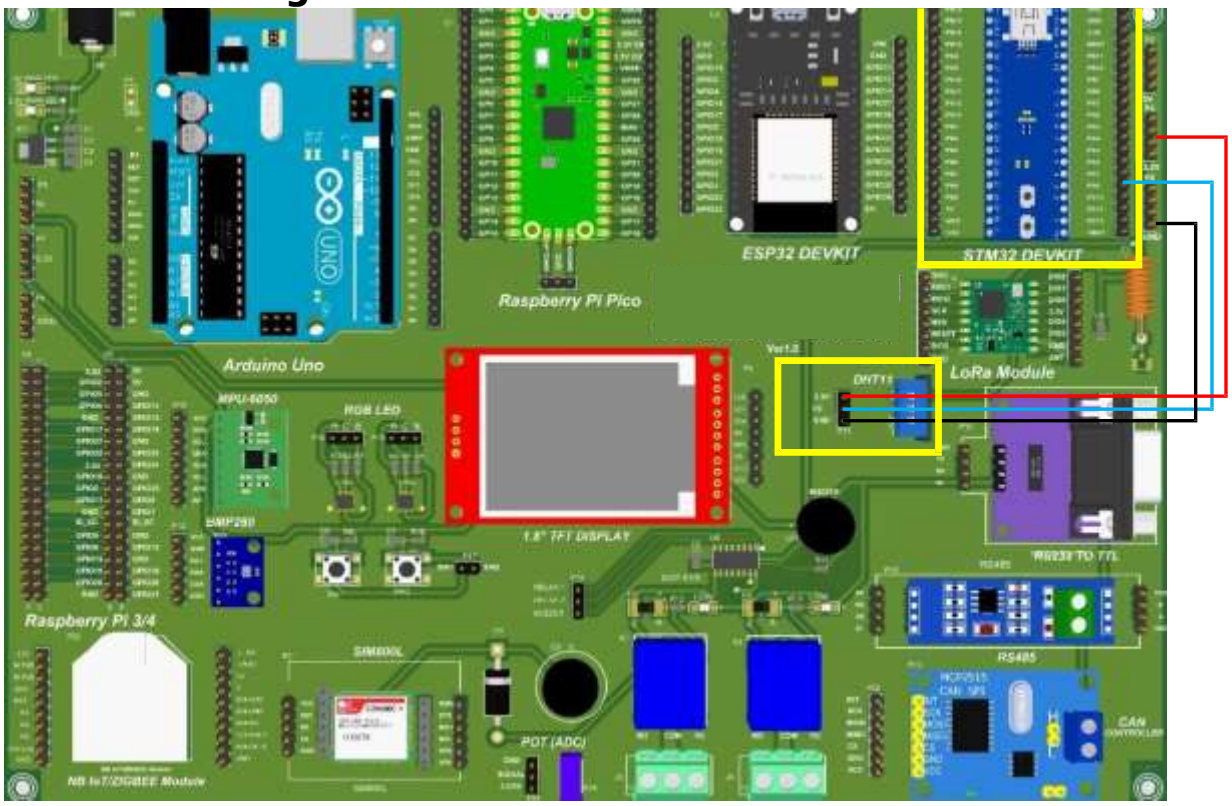
1. Blink a RGB LED



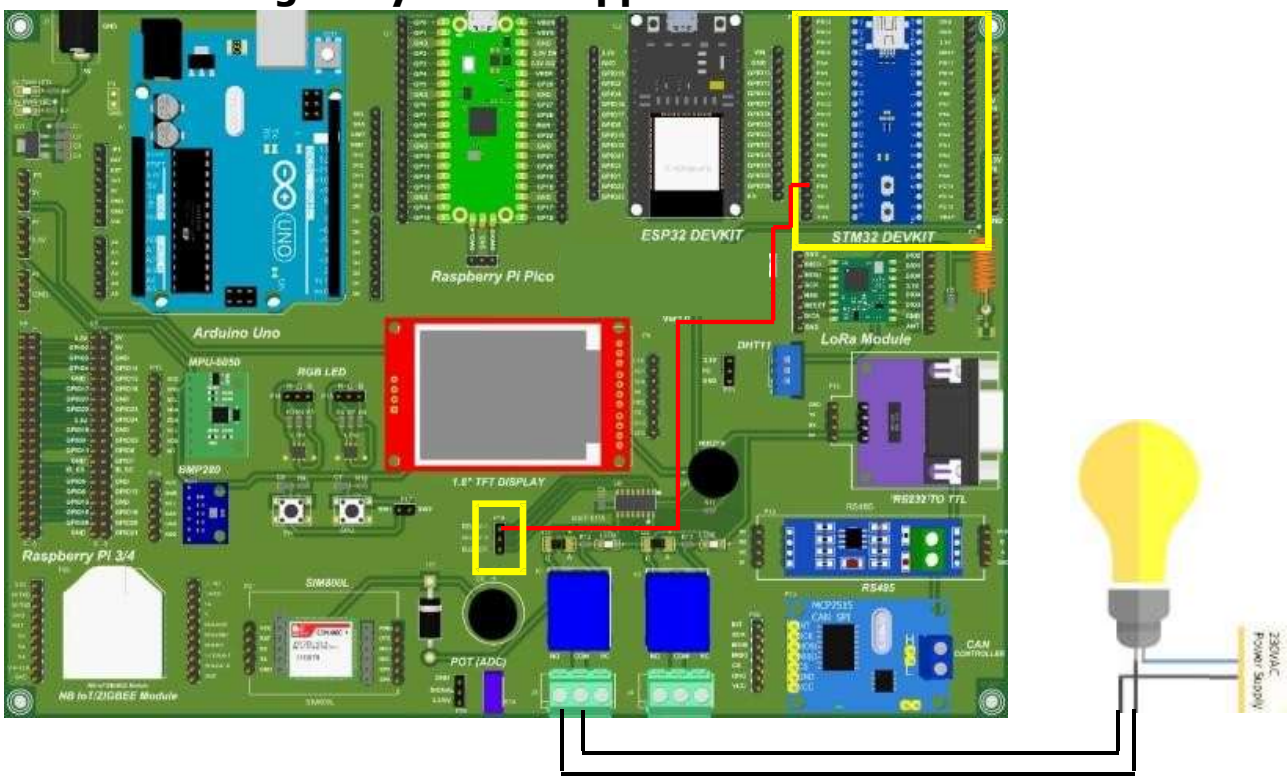
2. Pushbutton with an LED



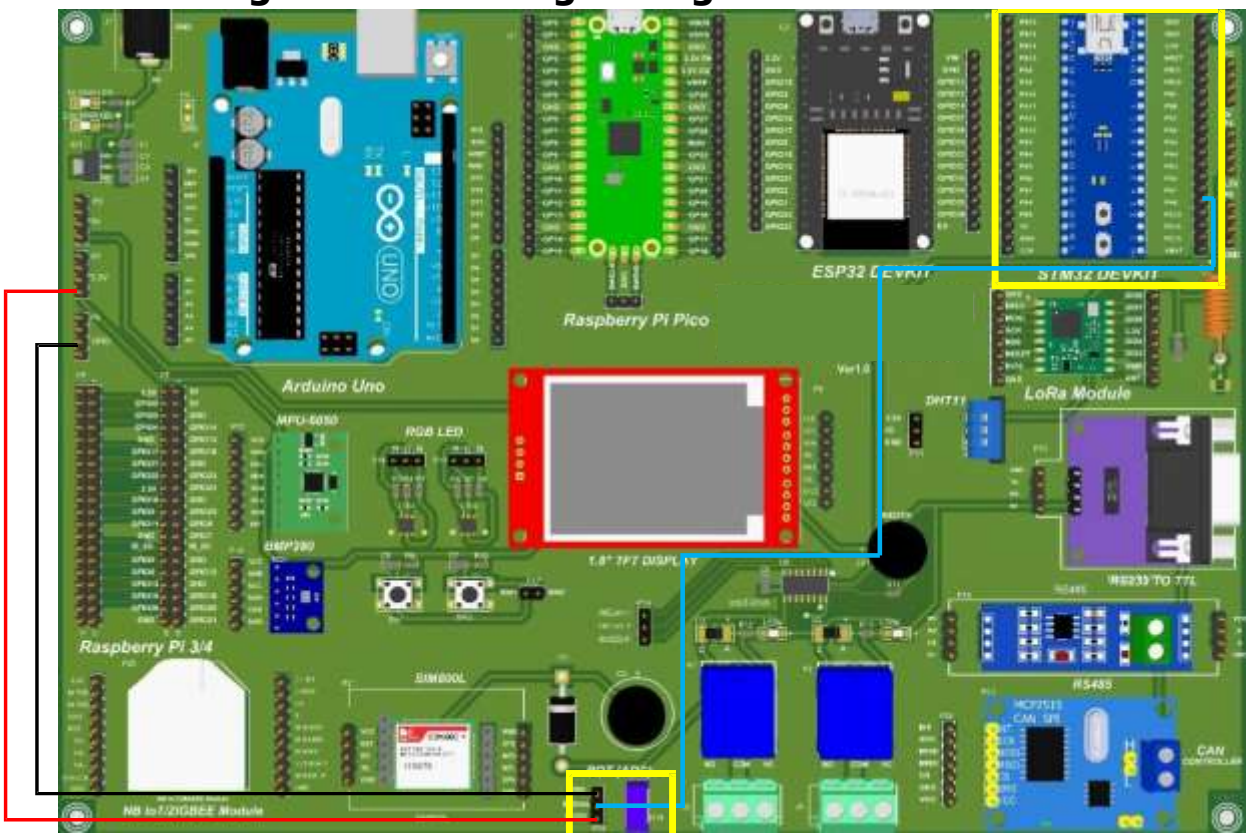
3. Interfacing DHT11



4. Interfacing Relay with AC Appliances



5. Interfacing PoT for Reading Analog Value



Section B – Codes

(Basic, Intermediate, Advanced)

Basics

Blink a RGB LED

(Arduino UNO | ESP32 | STM32)

LED	Uno	ESP32	STM32
R	D5	GPIO25	PC13
G	D4	GPIO26	PC14
B	D3	GPIO27	PC15

Refer Here for Different

```
// Prepared by IobiT Solutions
/*Copyrights reserved by Iobit Solutions*/

#define LED1RED 5
#define LED1BLUE 3
#define LED1GREEN 4
void setup() {
 // put your setup code here, to run once:
 pinMode(LED1RED, OUTPUT);
 pinMode(LED1BLUE, OUTPUT);
 pinMode(LED1GREEN, OUTPUT);
  Serial.begin(9600);
}
void loop() {
 // put your main code here, to run repeatedly:
 digitalWrite(LED1RED, HIGH);
 Serial.println("LED1RED ");
 delay(1000);
 digitalWrite(LED1RED, LOW);
  Serial.println("LED1RED");
 delay(1000);

 digitalWrite(LED1BLUE, HIGH);
 Serial.println("LED1BLUE");
 delay(1000);
 digitalWrite(LED1BLUE, LOW);
 Serial.println("LED1BLUE");
 delay(1000);

 digitalWrite(LED1GREEN, HIGH);
 Serial.println("LED1GREEN");
 delay(1000);
 digitalWrite(LED1GREEN, LOW);
 Serial.println("LED1GREEN");
 delay(1000);

}
```

Note: This code is common for (Arduino UNO | ESP32 | STM32) Refer the Pin details above mentioned in each section of **Hardware Interfacing Instruction**

www.edgate.in

RGB Blink Program for Rpi PICO

```
from machine import Pin
import utime


red = Pin(11, Pin.OUT)
green = Pin(12, Pin.OUT)
blue = Pin(13, Pin.OUT)


while True:
    red.value(0)
    green.value(0)
    blue.value(0)
    utime.sleep(2)

    red.value(1)
    green.value(0)
    blue.value(0)
    utime.sleep(2)

    red.value(0)
    green.value(0)
    blue.value(0)
    utime.sleep(2)

    red.value(0)
    green.value(1)
    blue.value(0)
    utime.sleep(2)

    red.value(0)
    green.value(0)
    blue.value(0)
    utime.sleep(2)

    red.value(0)
    green.value(0)
    blue.value(1)
    utime.sleep(2)
```

Pushbutton with an LED (Arduino UNO | ESP32 | STM32)

```
#define BUTTON_PIN 4
#define LED_PIN   13

// The below are variables, which can be changed
int button_state = 0;  // variable for reading the button status

void setup() {
 // initialize the LED pin as an output:
 pinMode(LED_PIN, OUTPUT);
 // initialize the button pin as an pull-up input:
 // the pull-up input pin will be HIGH when the button is open and LOW when the button is pressed.
 pinMode(BUTTON_PIN, INPUT);
 Serial.begin(9600);
}

void loop() {
 // read the state of the button value:
 button_state = digitalRead(BUTTON_PIN);
 Serial.print("BUTTON VALUE : ");
 Serial.println(button_state);
 delay(100);

 // control LED according to the state of button
 if (button_state == LOW)      // if button is pressed
   digitalWrite(LED_PIN, HIGH); // turn on LED
 else                          // otherwise, button is not pressing
   digitalWrite(LED_PIN, LOW);  // turn off LED
}
```

Refer Here for

I/Os	Uno	ESP32	STM32
R (LED)	13	GPIO2	PC13
PB	4	GPIO16	PC15

Note: This code is common for (Arduino UNO | ESP32 | STM32) Refer the Pin details above mentioned in each section of **Hardware Interfacing Instruction**

Pushbutton with an LED for Rpi PICO

```
from machine import Pin
import utime as time

led = Pin(25, Pin.OUT)
sensor = Pin(28, Pin.IN)

while True:
   time.sleep(1)
   print(sensor.value())
   if sensor.value()==0:
      led.value(1)
   else:
      led.value(0)
```

Interfacing of DHT 11
(Arduino UNO | ESP32 | STM32)

I/Os	Uno	ESP32	STM32
DHT 11	2	4	PA0

```
#include "DHT.h"
#define DHTPIN 2     // Digital pin connected to the DHT sensor

#define DHTTYPE DHT11  // DHT 11

// Initialize DHT sensor.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
 Serial.begin(9600);
 Serial.println("DHTxx test!");
 dht.begin();
}

void loop() {
 delay(2000);  // Wait a few seconds between measurements.
 // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
 float h = dht.readHumidity();
 // Read temperature as Celsius (the default)
 float t = dht.readTemperature();

 // Check if any reads failed and exit early (to try again).
 if (isnan(h) || isnan(t) ) {
  Serial.println("Failed to read from DHT sensor!");
  return;
 }
 Serial.print("Humidity: ");
 Serial.print(h);
 Serial.print("% Temperature: ");
 Serial.print(t);
 Serial.print("°C");

}
```

Interfacing of DHT 11 Sensor for Rpi PICO

Import Libraries form IobiT Solutions Git Page

```
from machine import Pin, I2C
import utime as time
import sys
sys.path.append('Your Library Installed Path with Brackets & Quotes')
from DHT import DHT11, InvalidChecksum
while True:
   time.sleep(5)
   pin = Pin(28, Pin.OUT, Pin.PULL_DOWN)
   sensor = DHT11(pin)
   t = (sensor.temperature)
   h = (sensor.humidity)
   print("Temperature: {}".format(sensor.temperature))
   print("Humidity: {}".format(sensor.humidity))
```

Interfacing Relay with AC Appliances

(Arduino UNO | ESP32 | STM32)

I/Os	Uno	ESP32	STM32
Relay 1	7	23	PB2

```
#define RELAY1 7

void setup() {
  // put your setup code here, to run once:
  pinMode(RELAY1, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(RELAY1, HIGH);   // turn the RELAY1 on (HIGH is the voltage level)
  Serial.println("RELAY1");
  delay(2000);                  // wait for a second
  digitalWrite(RELAY1, LOW);    // turn the RELAY1 off by making the voltage LOW
  Serial.println("RELAY1");
  delay(2000);                  // wait for a second
}
```

Interfacing of Relay for Rpi PICO

Import Libraries form IobiT Solutions Git Page

```
from machine import Pin, Timer
relay_pin1 = Pin(17, Pin.OUT)
relay_pin2 = Pin(27, Pin.OUT)
timer = Timer()

def relay(timer):
   relay_pin1.toggle()
   relay_pin2.toggle()

timer.init(freq=2.0, mode=Timer.PERIODIC, callback=relay)
```

Interfacing PoT for Reading Analog Value

(Arduino UNO | ESP32 | STM32)

I/Os	Uno	ESP32	STM32
PoT	A0	35	PA0

```
// Potentiometer is connected to GPIO 36 (Analog ADC1_CH6)

const int potPin = A0;

// variable for storing the potentiometer value
int potValue = 0;

void setup() {
  Serial.begin(9600);
  delay(1000);
}

void loop() {
  // Reading potentiometer value
  potValue = analogRead(potPin);
  Serial.print("potValue: ");
  Serial.println(potValue);
  delay(1000);
}
```

Interfacing of Relay for Rpi PICO

Import Libraries form IobiT Solutions Git Page

```
from machine import ADC, Pin
import time

adc = ADC(Pin(26))

while True:
    print(adc.read_u16())
    time.sleep(1)
```

Interfacing BMP280 Sensor

(Arduino UNO | ESP32 | STM32)

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>
#define BMP_SCK  (13)
#define BMP_MISO (12)
#define BMP_MOSI (11)
#define BMP_CS   (10)
Adafruit_BMP280 bmp; // I2C by defalut
//Adafruit_BMP280 bmp(BMP_CS); // hardware SPI
//Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO,  BMP_SCK);

void setup() {
  Serial.begin(9600);
  while ( !Serial ) delay(100);  // wait for native usb
  Serial.println(F("BMP280 test"));
  unsigned status;
  //status = bmp.begin(BMP280_ADDRESS_ALT, BMP280_CHIPID);
  status = bmp.begin(0X76);
  if (!status) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "
                     "try a different address!"));
    Serial.print("SensorID was: 0x"); Serial.println(bmp.sensorID(),16);
    Serial.print("       ID of 0xFF probably means a bad address, a BMP 180 or BMP 085 \n");
    Serial.print("  ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("       ID of 0x60 represents a BME 280.\n");
    Serial.print("       ID of 0x61 represents a BME 680.\n");
    while (1) delay(10);
  }
  /* Default settings from datasheet. */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,    /* Operating Mode. */
                  Adafruit_BMP280::SAMPLING_X2,    /* Temp. oversampling */
                  Adafruit_BMP280::SAMPLING_X16,   /* Pressure oversampling */
                  Adafruit_BMP280::FILTER_X16,     /* Filtering. */
                  Adafruit_BMP280::STANDBY_MS_500);/* Standby time. */
}
void loop() {
    Serial.print(F("Temperature = "));
    Serial.print(bmp.readTemperature());
    Serial.println(" *C");

    Serial.print(F("Pressure = "));
    Serial.print(bmp.readPressure());
    Serial.println(" Pa");
    Serial.print(F("Approx altitude = "));
    Serial.print(bmp.readAltitude(1013.25)); /* Adjusted to local forecast! */
    Serial.println(" m");
    Serial.println();
    delay(2000);
}
```

Interfacing with BMP280 for Rpi PICO

```
# https://github.com/dafvid/micropython-bmp280

# import the required libraries
from bmp280 import *
from machine import Pin, I2C
import utime
from time import sleep

i2c = I2C(0, sda = Pin(0), scl = Pin(1), freq = 400000)
# Caliberation error in pressure
# use it according to your situation
# it helps in calibrating altitude .It is optional, else put ERROR = 0
ERROR = -3 # hPa

# declare pins for I2C communication
sclPin = Pin(1)# serial clock pin
sdaPin = Pin(0) # serial data pin

# Initiate I2C
i2c_object = I2C(0,          # positional argument - I2C id
            scl = sclPin,  # named argument - serial clock pin
            sda = sdaPin,  # named argument - serial data pin
            freq = 1000000) # named argument - i2c frequency

# scan i2c port for available devices
result = I2C.scan(i2c_object)
print("I2C scan result : ", result) # 118 in decimal is same as 0x76 in hexadecimal
if result != []:
   print("I2C connection successfull")
else:
   print("No devices found !")


# create a BMP 280 object
bmp280_object = BMP280(i2c_object,
                addr = 0x76, # change it
                use_case = BMP280_CASE_WEATHER)

# configure the sensor
# These configuration settings give most accurate values in my case
# tweak them according to your own requirements

bmp280_object.power_mode =  BMP280_POWER_NORMAL
bmp280_object.oversample = BMP280_OS_HIGH
bmp280_object.temp_os = BMP280_TEMP_OS_8
bmp280_object.press_os = BMP280_TEMP_OS_4
bmp280_object.standby = BMP280_STANDBY_250
bmp280_object.iir = BMP280_IIR_FILTER_2

print("BMP Object created successfully !")
utime.sleep(2)# change it as per requirement
print("\n")

# Function for calculation altitude from pressure and temperature values
# because altitude() method is not present in the Library
```

```
def altitude_HYP(hPa, temperature):
    # Hypsometric Equation (Max Altitude < 11 Km above sea level)
    temperature = temperature
    local_pressure = hPa
    sea_level_pressure = 1013.25# hPa
    pressure_ratio = sea_level_pressure/local_pressure # sea level pressure = 1013.25 hPa
    h = (((pressure_ratio**(1/5.257)) - 1) * temperature ) / 0.0065
    return h

# altitude from international barometric formula, given in BMP180 datasheet
def altitude_IBF(pressure):
    local_pressure = pressure   # Unit : hPa
    sea_level_pressure = 1013.25# Unit : hPa
    pressure_ratio = local_pressure / sea_level_pressure
    altitude = 44330*(1-(pressure_ratio**(1/5.255)))
    return altitude
   while True:
    # accquire temperature value in celcius
    temperature_c = bmp280_object.temperature # degree celcius


    # convert celcius to kelvin
    temperature_k = temperature_c + 273.15


    # accquire pressure value
    pressure = bmp280_object.pressure # pascal


    # convert pascal to hectopascal (hPa)
    # 1 hPa = 100 Pa
    # Therefore 1 Pa = 0.01 hPa
    pressure_hPa = ( pressure * 0.01 ) + ERROR # hPa
        # accquire altitude values from HYPSOMETRIC formula
    h = altitude_HYP(pressure_hPa, temperature_k)
        # accquire altitude values from International Barometric Formula
    altitude = altitude_IBF(pressure_hPa)
    press = "{:.2f}".format(pressure_hPa)
    h_alti = "{:.2f}".format(h)
    i_alti = "{:.2f}".format(altitude)
    print("Temperature : ",temperature_c," Degree Celcius")
    print("Pressure : ",pressure," Pascal (Pa)")
    print("Pressure : ",press," hectopascal (hPa) or millibar (mb)")
    print("Altitude (Hypsometric Formula) : ", h_alti , " meter")
    print("Altitude (International Barometric Formula) : ", i_alti , " meter")
    print("\n")
    utime.sleep(1.5)
```

Interfacing MPU6050 Sensor

(Arduino UNO | ESP32 | STM32)

```
// Basic demo for accelerometer readings from Adafruit MPU6050

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
Adafruit_MPU6050 mpu;
void setup(void) {
  Serial.begin(9600);
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit MPU6050 test!");

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  Serial.print("Accelerometer range set to: ");
  switch (mpu.getAccelerometerRange()) {
  case MPU6050_RANGE_2_G:
    Serial.println("+-2G");
    break;
  case MPU6050_RANGE_4_G:
    Serial.println("+-4G");
    break;
  case MPU6050_RANGE_8_G:
    Serial.println("+-8G");
    break;
  case MPU6050_RANGE_16_G:
    Serial.println("+-16G");
    break;
  }
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  Serial.print("Gyro range set to: ");
  switch (mpu.getGyroRange()) {
  case MPU6050_RANGE_250_DEG:
    Serial.println("+- 250 deg/s");
    break;
  case MPU6050_RANGE_500_DEG:
    Serial.println("+- 500 deg/s");
    break;
  case MPU6050_RANGE_1000_DEG:
    Serial.println("+- 1000 deg/s");
    break;
  case MPU6050_RANGE_2000_DEG:
    Serial.println("+- 2000 deg/s");
    break;
  }
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
```

```
  Serial.print("Filter bandwidth set to: ");
  switch (mpu.getFilterBandwidth()) {
  case MPU6050_BAND_260_HZ:
   Serial.println("260 Hz");
   break;
  case MPU6050_BAND_184_HZ:
   Serial.println("184 Hz");
   break;
  case MPU6050_BAND_94_HZ:
   Serial.println("94 Hz");
   break;
  case MPU6050_BAND_44_HZ:
   Serial.println("44 Hz");
   break;
  case MPU6050_BAND_21_HZ:
   Serial.println("21 Hz");
   break;
  case MPU6050_BAND_10_HZ:
   Serial.println("10 Hz");
   break;
  case MPU6050_BAND_5_HZ:
   Serial.println("5 Hz");
   break;
  }
  Serial.println("");
  delay(100);
}
void loop() {
  /* Get new sensor events with the readings */
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  /* Print out the values */
  Serial.print("Acceleration X: ");
  Serial.print(a.acceleration.x);
  Serial.print(", Y: ");
  Serial.print(a.acceleration.y);
  Serial.print(", Z: ");
  Serial.print(a.acceleration.z);
  Serial.println(" m/s^2");

  Serial.print("Rotation X: ");
  Serial.print(g.gyro.x);
  Serial.print(", Y: ");
  Serial.print(g.gyro.y);
  Serial.print(", Z: ");
  Serial.print(g.gyro.z);
  Serial.println(" rad/s");

  Serial.print("Temperature: ");
  Serial.print(temp.temperature);
  Serial.println(" degC");

  Serial.println("");
  delay(500);
}
```

Interfacing with MPU6050 for Rpi PICO

```
# Blog: https://www.iobit.in
# Date: May 18th, 2022

#import sys
#sys.path.append('/MYLIB/MPU6050LIB')
from IMU import MPU6050
import time
from machine import Pin, I2C

i2c = I2C(0, sda=Pin(0), scl=Pin(1), freq=400000)
imu = MPU6050(i2c)

while True:
    # Following print shows original data get from libary. You can uncomment to see raw data
    #print(imu.accel.xyz,imu.gyro.xyz,imu.temperature,end='\r')


    # Following rows round values get for a more pretty print:
    ax=round(imu.accel.x,2)
    ay=round(imu.accel.y,2)
    az=round(imu.accel.z,2)
    gx=round(imu.gyro.x)
    gy=round(imu.gyro.y)
    gz=round(imu.gyro.z)
    tem=round(imu.temperature,2)
    print(ax,"\t",ay,"\t",az,"\t",gx,"\t",gy,"\t",gz,"\t",tem,"        ",end="\r")


    # Following sleep statement makes values enought stable to be seen and
    # read by a human from shell
    time.sleep(0.2)
```

Interfacing of TFT Screen

(Arduino UNO)

```
// include TFT and SPI libraries
#include <TFT.h>
#include <SPI.h>
// pin definition for Arduino UNO
#define cs   10
#define dc   9
#define rst  8
// create an instance of the library
TFT TFTscreen = TFT(cs, dc, rst);
void setup() {
  //initialize the library
  TFTscreen.begin();
  // clear the screen with a black background
  TFTscreen.background(0, 0, 0);
  //set the text size
  TFTscreen.setTextSize(2);
}
void loop() {
  //generate a random color
  int redRandom = random(0, 255);
  int greenRandom = random(0, 255);
  int blueRandom = random(0, 255);
```

```
  // set a random font color
  TFTscreen.stroke(redRandom, greenRandom, blueRandom);

  // print Hello, World! in the middle of the screen
  TFTscreen.text("Hello, World!", 6, 57);

  // wait 200 miliseconds until change to next color
  delay(200);
}
```

ESP32

```
//*************************************************************************************
//include library code
#include <Adafruit_GFX.h>
#include <Adafruit_ST7735.h>
#include <SPI.h>
//*************************************************************************************
//define pins of TFT screen
#define  TFT_CS     12
#define TFT_RST    14
#define  TFT_DC     13
#define TFT_SCLK   22
#define TFT_MOSI   21
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK,
TFT_RST); float p = 3.1415926;
//*************************************************************************************
void setup(void) {
  Serial.begin(115200);//initialise serial communication at 115200 bps
  Serial.print("ST7735 TFT grafics test");
  tft.initR(INITR_BLACKTAB);//initialize a ST7735S chip, black tab
  Serial.println("Initializing...");
  uint16_t time = millis();
  tft.fillScreen(ST7735_BLACK);
  time = millis() - time;
  Serial.println(time, DEC);
  delay(500);
  tft.fillScreen(ST7735_BLACK); //large block of text
  testdrawtext("HelloIobiT. ", ST7735_WHITE);
  delay(1000);

  // tft print function!
  tftPrintTest();
  delay(4000);

  // a single pixel
  tft.drawPixel(tft.width()/2, tft.height()/2, ST7735_GREEN);
  delay(500);

  // line draw test
  testlines(ST7735_YELLOW);
  delay(500);

  // optimized lines
  testfastlines(ST7735_RED, ST7735_BLUE);
  delay(500);
```

```
  testdrawrects(ST7735_GREEN);
  delay(500);

  testfillrects(ST7735_YELLOW, ST7735_MAGENTA);
  delay(500);

  tft.fillScreen(ST7735_BLACK);
  testfillcircles(10, ST7735_BLUE);
  testdrawcircles(10, ST7735_WHITE);
  delay(500);

  testroundrects();
  delay(500);

  testtriangles();
  delay(500);

  mediabuttons();
  delay(500);

  Serial.println("done");
  delay(1000);
}

void loop() {
  tft.invertDisplay(true);
  delay(500);
  tft.invertDisplay(false);
  delay(500);
}

void testlines(uint16_t color) {
  tft.fillScreen(ST7735_BLACK);
  for (int16_t x=0; x < tft.width(); x+=6) {
    tft.drawLine(0, 0, x, tft.height()-1, color);
  }
  for (int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(0, 0, tft.width()-1, y, color);
  }

  tft.fillScreen(ST7735_BLACK);
  for (int16_t x=0; x < tft.width(); x+=6) {
    tft.drawLine(tft.width()-1, 0, x, tft.height()-1, color);
  }
  for (int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(tft.width()-1, 0, 0, y, color);
  }

  tft.fillScreen(ST7735_BLACK);
  for (int16_t x=0; x < tft.width(); x+=6) {
    tft.drawLine(0, tft.height()-1, x, 0, color);
  }
  for (int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(0, tft.height()-1, tft.width()-1, y, color);
  }
```

```
  tft.fillScreen(ST7735_BLACK);
  for(int16_t x=0; x < tft.width(); x+=6) {
    tft.drawLine(tft.width()-1, tft.height()-1, x, 0, color);
  }
  for(int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(tft.width()-1, tft.height()-1, 0, y, color);
  }
}

void testdrawtext(char *text, uint16_t color) {
  tft.setCursor(0, 0);
  tft.setTextColor(color);
  tft.setTextWrap(true);
  tft.print(text);
}

void testfastlines(uint16_t color1, uint16_t color2) {
  tft.fillScreen(ST7735_BLACK);
  for(int16_t y=0; y < tft.height(); y+=5) {
    tft.drawFastHLine(0, y, tft.width(), color1);
  }
  for(int16_t x=0; x < tft.width(); x+=5) {
    tft.drawFastVLine(x, 0, tft.height(), color2);
  }
}

void testdrawrects(uint16_t color) {
  tft.fillScreen(ST7735_BLACK);
  for(int16_t x=0; x < tft.width(); x+=6) {
    tft.drawRect(tft.width()/2 -x/2, tft.height()/2 -x/2 , x, x, color);
  }
}

void testfillrects(uint16_t color1, uint16_t color2) {
  tft.fillScreen(ST7735_BLACK);
  for(int16_t x=tft.width()-1; x > 6; x-=6) {
    tft.fillRect(tft.width()/2 -x/2, tft.height()/2 -x/2 , x, x, color1);
    tft.drawRect(tft.width()/2 -x/2, tft.height()/2 -x/2 , x, x, color2);
  }
}

void testfillcircles(uint8_t radius, uint16_t color) {
  for(int16_t x=radius; x < tft.width(); x+=radius*2) {
    for(int16_t y=radius; y < tft.height(); y+=radius*2) {
      tft.fillCircle(x, y, radius, color);
    }
  }
}

void testdrawcircles(uint8_t radius, uint16_t color) {
  for(int16_t x=0; x < tft.width()+radius; x+=radius*2){
    for (int16_t y=0; y < tft.height()+radius; y+=radius*2) {
      tft.drawCircle(x, y, radius, color);
    }
  }
}
```

```
void testtriangles() {
  tft.fillScreen(ST7735_BLACK);
  int color= 0xF800;
  int t;
  int w = tft.width()/2;
  int x = tft.height()-1;
  int y = 0;
  int z = tft.width();
  for(t= 0 ; t <= 15; t++) {
    tft.drawTriangle(w, y, y, x, z, x, color);
    x-=4;
    y+=4;
    z-=4;
    color+=100;
  }
}

void testroundrects() {
  tft.fillScreen(ST7735_BLACK);
  int color= 100;
  int i;
  int t;
  for(t= 0 ; t <= 4; t+=1) {
    int x = 0;
    int y = 0;
    int w = tft.width()-2;
    int h = tft.height()-2;
    for(i= 0 ; i <= 16; i+=1) {
      tft.drawRoundRect(x, y, w, h, 5, color);
      x+=2;
      y+=3;
      w-=4;
      h-=6;
      color+=1100;
    }
    color+=100;
  }
}

void tftPrintTest() {
  tft.setTextWrap(false);
  tft.fillScreen(ST7735_BLACK);
  tft.setCursor(0, 30);
  tft.setTextColor(ST7735_RED);
  tft.setTextSize(1);
  tft.println("Hello World!");
  tft.setTextColor(ST7735_YELLOW);
  tft.setTextSize(2);
  tft.println("Hello World!");
  tft.setTextColor(ST7735_GREEN);
  tft.setTextSize(3);
  tft.println("Hello World!");
  tft.setTextColor(ST7735_BLUE);
  tft.setTextSize(4);
  tft.print(1234.567);
  delay(1500);
  tft.setCursor(0, 0);
```

```
  tft.fillScreen(ST7735_BLACK);
  tft.setTextColor(ST7735_WHITE);
  tft.setTextSize(0);
  tft.println("Hello World!");
  tft.setTextSize(1);
  tft.setTextColor(ST7735_GREEN);
  tft.print(p, 6);
  tft.println(" Want pi?");
  tft.println(" ");
  tft.print(8675309, HEX); // print 8,675,309 out in HEX!
  tft.println(" Print HEX!");
  tft.println(" ");
  tft.setTextColor(ST7735_WHITE);
  tft.println("Sketch has been");
  tft.println("running for: ");
  tft.setTextColor(ST7735_MAGENTA);
  tft.print(millis() / 1000);
  tft.setTextColor(ST7735_WHITE);
  tft.print(" seconds.");
}

void mediabuttons() {
  // play
  tft.fillScreen(ST7735_BLACK);
  tft.fillRoundRect(25, 10, 78, 60, 8, ST7735_WHITE);
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST7735_RED);
  delay(500);
  // pause
  tft.fillRoundRect(25, 90, 78, 60, 8, ST7735_WHITE);
  tft.fillRoundRect(39, 98, 20, 45, 5, ST7735_GREEN);
  tft.fillRoundRect(69, 98, 20, 45, 5, ST7735_GREEN);
  delay(500);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST7735_BLUE);
  delay(50);
  // pause color
  tft.fillRoundRect(39, 98, 20, 45, 5, ST7735_RED);
  tft.fillRoundRect(69, 98, 20, 45, 5, ST7735_RED);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST7735_GREEN);
}
```

STM32

```
#define SCR_WD  128
#define SCR_HT  160
#include <SPI.h>
#include <Adafruit_GFX.h>

#define  TFT_CS   PA2
#define  TFT_DC   PA1
#define TFT_RST PA0
#include <Arduino_ST7735_STM.h>


Arduino_ST7735 lcd = Arduino_ST7735(TFT_DC, TFT_RST, TFT_CS);

void setup()
```

```
{
  Serial.begin(9600);
  lcd.init();
}

void loop(void)
{
  for(uint8_t rot = 0; rot < 4; rot++) {
    testText(rot);
    delay(2000);
  }
}

unsigned long testText(int rot)
{
  lcd.setRotation(rot);
  lcd.fillScreen(BLACK);
  lcd.setCursor(0, 0);
  lcd.setTextColor(BLUE);
  lcd.setTextSize(1);
  lcd.println("Hello World!");
  lcd.setTextColor(WHITE);
  lcd.print("Rotation= ");
  lcd.println(rot);
  lcd.setTextColor(YELLOW);
  lcd.setTextSize(2);
  lcd.println(1234.56);
  lcd.setTextColor(RED);
  lcd.setTextSize(3);
  lcd.println(0xDEAD, HEX);
  lcd.println();
  lcd.setTextColor(GREEN);
  lcd.setTextSize(4);
  lcd.println("Hello");
}
```

Interfacing TFT Screen with Rpi PICO

```
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI,Pin
import time
import math
spi = SPI(1, baudrate=20000000, polarity=0, phase=0,
        sck=Pin(10), mosi=Pin(11), miso=None)
tft=TFT(spi,16,17,18)
tft.initr()
tft.rgb(True)

def testlines(color):
    tft.fill(TFT.BLACK)
    for x in range(0, tft.size()[0], 6):
        tft.line((0,0),(x, tft.size()[1] - 1), color)
    for y in range(0, tft.size()[1], 6):
        tft.line((0,0),(tft.size()[0] - 1, y), color)

    tft.fill(TFT.BLACK)
```

```
    for x in range(0, tft.size()[0], 6):
        tft.line((tft.size()[0] - 1, 0), (x, tft.size()[1] - 1), color)
    for y in range(0, tft.size()[1], 6):
        tft.line((tft.size()[0] - 1, 0), (0, y), color)

    tft.fill(TFT.BLACK)
    for x in range(0, tft.size()[0], 6):
        tft.line((0, tft.size()[1] - 1), (x, 0), color)
    for y in range(0, tft.size()[1], 6):
        tft.line((0, tft.size()[1] - 1), (tft.size()[0] - 1,y), color)

    tft.fill(TFT.BLACK)
    for x in range(0, tft.size()[0], 6):
        tft.line((tft.size()[0] - 1, tft.size()[1] - 1), (x, 0), color)
    for y in range(0, tft.size()[1], 6):
        tft.line((tft.size()[0] - 1, tft.size()[1] - 1), (0, y), color)

def testfastlines(color1, color2):
    tft.fill(TFT.BLACK)
    for y in range(0, tft.size()[1], 5):
        tft.hline((0,y), tft.size()[0], color1)
    for x in range(0, tft.size()[0], 5):
        tft.vline((x,0), tft.size()[1], color2)

def testdrawrects(color):
    tft.fill(TFT.BLACK);
    for x in range(0,tft.size()[0],6):
        tft.rect((tft.size()[0]//2 - x//2, tft.size()[1]//2 - x/2), (x, x), color)

def testfillrects(color1, color2):
    tft.fill(TFT.BLACK);
    for x in range(tft.size()[0],0,-6):
        tft.fillrect((tft.size()[0]//2 - x//2, tft.size()[1]//2 - x/2), (x, x), color1)
        tft.rect((tft.size()[0]//2 - x//2, tft.size()[1]//2 - x/2), (x, x), color2)


def testfillcircles(radius, color):
    for x in range(radius, tft.size()[0], radius * 2):
        for y in range(radius, tft.size()[1], radius * 2):
            tft.fillcircle((x, y), radius, color)

def testdrawcircles(radius, color):
    for x in range(0, tft.size()[0] + radius, radius * 2):
        for y in range(0, tft.size()[1] + radius, radius * 2):
            tft.circle((x, y), radius, color)

def testtriangles():
    tft.fill(TFT.BLACK);
    color = 0xF800
    w = tft.size()[0] // 2
    x = tft.size()[1] - 1
    y = 0
    z = tft.size()[0]
    for t in range(0, 15):
        tft.line((w, y), (y, x), color)
        tft.line((y, x), (z, x), color)
        tft.line((z, x), (w, y), color)
```

```
        x -= 4
        y += 4
        z -= 4
        color+= 100

def testroundrects():
    tft.fill(TFT.BLACK);
    color= 100
    for t in range(5):
        x = 0
        y = 0
        w = tft.size()[0] - 2
        h = tft.size()[1] - 2
        for i in range(17):
            tft.rect((x, y), (w, h), color)
            x += 2
            y += 3
            w -= 4
            h -= 6
            color+= 1100
        color+= 100

def tftprinttest():
    tft.fill(TFT.BLACK);
    v = 30
    tft.text((0, v), "Hello World!", TFT.RED, sysfont, 1, nowrap=True)
    v += sysfont["Height"]
    tft.text((0, v), "Hello World!", TFT.YELLOW, sysfont, 2, nowrap=True)
    v += sysfont["Height"]* 2
    tft.text((0, v), "Hello World!", TFT.GREEN, sysfont, 3, nowrap=True)
    v += sysfont["Height"]* 3
    tft.text((0, v), str(1234.567), TFT.BLUE, sysfont, 4, nowrap=True)
    time.sleep_ms(1500)
    tft.fill(TFT.BLACK);
    v = 0
    tft.text((0, v), "Hello World!", TFT.RED, sysfont)
    v += sysfont["Height"]
    tft.text((0, v), str(math.pi), TFT.GREEN, sysfont)
    v += sysfont["Height"]
    tft.text((0, v), " Want pi?", TFT.GREEN, sysfont)
    v += sysfont["Height"]* 2
    tft.text((0, v), hex(8675309), TFT.GREEN, sysfont)
    v += sysfont["Height"]
    tft.text((0, v), " Print HEX!", TFT.GREEN, sysfont)
    v += sysfont["Height"]* 2
    tft.text((0, v), "Sketch has been", TFT.WHITE, sysfont)
    v += sysfont["Height"]
    tft.text((0, v), "running for:", TFT.WHITE, sysfont)
    v += sysfont["Height"]
    tft.text((0, v), str(time.ticks_ms()/1000), TFT.PURPLE, sysfont)
    v += sysfont["Height"]
    tft.text((0, v), " seconds.", TFT.WHITE, sysfont)

def test_main():
    tft.fill(TFT.BLACK)
    tft.text((0, 0), "Hello IobiT ", TFT.WHITE, sysfont, 1)
    time.sleep_ms(1000)
```

```
    tftprinttest()
    time.sleep_ms(4000)

    testlines(TFT.YELLOW)
    time.sleep_ms(500)

    testfastlines(TFT.RED, TFT.BLUE)
    time.sleep_ms(500)

    testdrawrects(TFT.GREEN)
    time.sleep_ms(500)

    testfillrects(TFT.YELLOW, TFT.PURPLE)
    time.sleep_ms(500)

    tft.fill(TFT.BLACK)
    testfillcircles(10, TFT.BLUE)
    testdrawcircles(10, TFT.WHITE)
    time.sleep_ms(500)

    testroundrects()
    time.sleep_ms(500)

    testtriangles()
    time.sleep_ms(500)

test_main()
```

Experience IoT

Bluetooth Communication

The ESP32 has native Bluetooth support (version 4.2). This means that it can interact with Bluetooth devices such as keyboards, mice and cell phones. Let us review what Bluetooth means for us. Bluetooth is a wireless communication protocol/technology that provides data transfer over a radio signal. Let us assume that ESP32 is one end of the connection and any other Bluetooth device can be at the other.

For security purposes, arbitrary Bluetooth devices can't simply be "used" without some explicit authorization. For example, it would be very wrong if I could bring my Bluetooth headset near your phone and start listening to your calls. To achieve security, a process called "pairing" needs to be performed. This achieves a level of trust between the two Bluetooth devices such that they subsequently allow connection without having to be re-paired.

1. Programming to send "HELLO WORLD" via Bluetooth Communication

Hardware Interfacing Instruction:

ESP32	

Setup your IDE

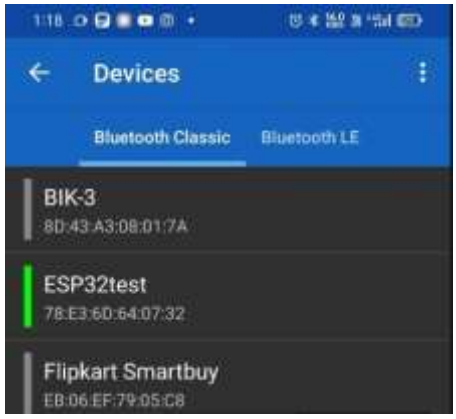
ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

Code

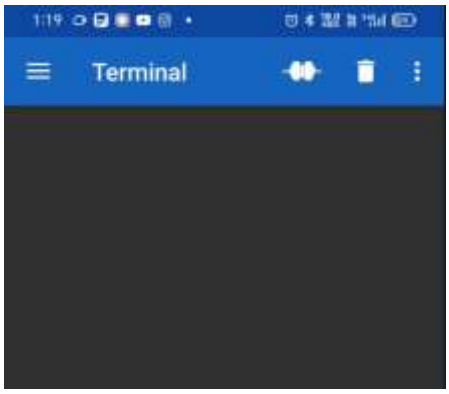
```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()){
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

To Experience the Outcome

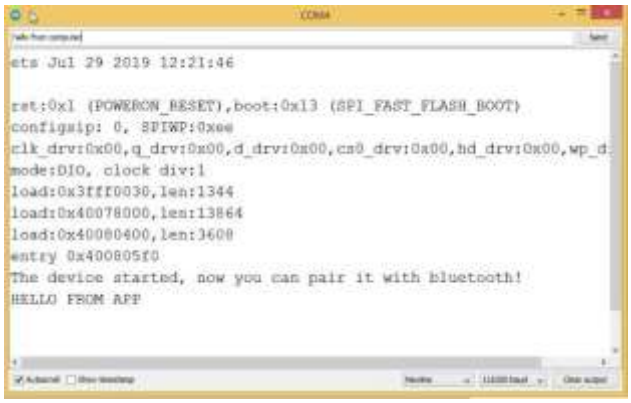
- Download & install the apk file of "Bluetooth Serial" form Google Play store
- Open the Mobile Application and Turn ON the Bluetooth in your Mobile to Scan List the ESP32 Devices



- Make Sure it is Connected



- Send the Data from the Command Line



2. Programming to send "HELLO WORLD" via Bluetooth Communication

Hardware Interfacing Instruction:

ESP32		
Buzzer	GPIO 2	

Setup your IDE

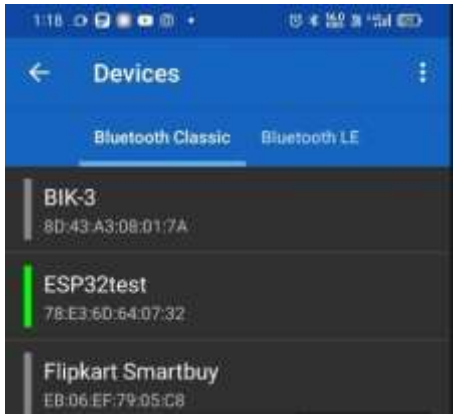
ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

Code

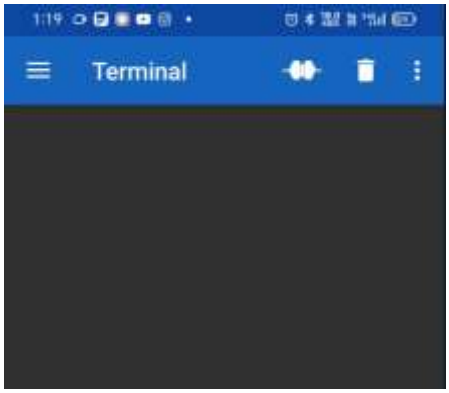
```
#include "BluetoothSerial.h"
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif
#define BUZZER 2

BluetoothSerial SerialBT;
// Handle received and sent messages
String message = "";
char incomingChar;

void setup() {
  pinMode(BUZZER, OUTPUT);
  Serial.begin(115200);
  SerialBT.begin("ESP32"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");
}
void loop() {
 if (SerialBT.available()){
    char incomingChar = SerialBT.read();
    if (incomingChar != '\n'){
      message += String(incomingChar);
    }
    else{
      message = "";
    }
    Serial.write(incomingChar);
  }
  // Check received message and control output accordingly
  if (message =="on"){
    digitalWrite(BUZZER, HIGH);
  }
  else if (message =="off"){
    digitalWrite(BUZZER, LOW);
  }
  delay(20);
}
```

To Experience the Outcome

- Download & install the apk file of "Bluetooth Serial" form Google Play store
- Open the Mobile Application and Turn ON the Bluetooth in your Mobile to Scan List the ESP32 Devices



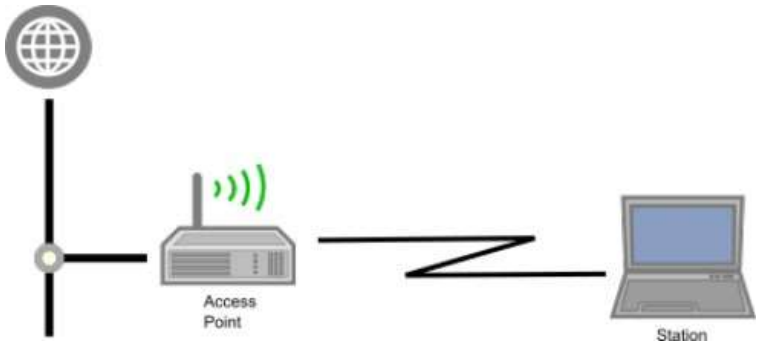
- Make Sure it is Connected



- Send the Data from the Command Line ON & OFF from Mobile

WiFi Communication

When working with a WiFi oriented device, it is important that we have at least some understanding of the concepts related to WiFi. At a high level, WiFi is the ability to participate in TCP/IP connections over a wireless communication link. WiFi is specifically the set of protocols described in the IEEE 802.11 Wireless LAN architecture. Within this story, a device called a Wireless Access Point (access point or AP) act as the hub of all communications. Typically it is connected to (or acts as) as TCP/IP router to the rest of the TCP/IP network. For example, in your home, you are likely to have a WiFi access point connected to your modem (cable or DSL). WiFi connections are then formed to the access point (through devices called stations) and TCP/IP traffic flows through the access point to the Internet.



An ESP32 device can play the role of an Access Point, a Station or both at the same time. Very commonly, the access point also has a network connection to the Internet and acts as a bridge between the wireless network and the broader TCP/IP network that is the Internet. A collection of stations that wish to communicate with each other is termed a Basic Service Set (BSS). The common configuration is what is known as an Infrastructure BSS.

In this mode, all communications inbound and outbound from an individual station are routed through the access point. A station must associate itself with an access point in order to participate in the story. A station may only be associated with a single access point at any one time. Each participant in the network has a unique identifier called the MAC address. This is a 48bit value.

When we have multiple access points within wireless range, the station needs to know with which one to connect. Each access point has a network identifier called the **BSSID** (or more commonly just SSID). **SSID** is service set identifier. It is a 32-character value that represents the target of packets of information sent over the network.

1. Experiencing to Interface MPU6050 Sensor and send data to Blynk Cloud App

Hardware Interfacing Instruction:

ESP32	MPU6050
3.3v	VCC 3.3v
GND	GND
GPIO 22	SCL
GPIO 21	SDA

Setup your IDE

ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

Library Installation

Code

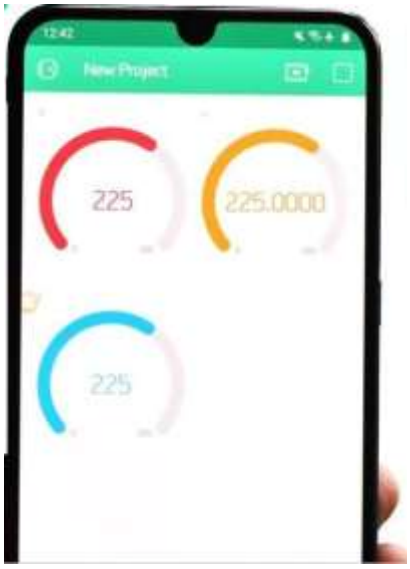
```
#include<Wire.h>

#define BLYNK_TEMPLATE_ID "Enter Blyk ID"
#define BLYNK_DEVICE_NAME "ESP32 MPU6050"
#define BLYNK_AUTH_TOKEN "Enter Auth Token"


#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Enter Wifi SSID";  // Enter your Wifi Username
char pass[] = "Enter Wifi Pass";  // Enter your Wifi password

Adafruit_MPU6050 mpu;
BlynkTimer timer;

void setup(){
 Serial.begin(9600);
```

```
while (!Serial)
  delay(10); // will pause Zero, Leonardo, etc until serial console opens

Serial.println("Adafruit MPU6050 test!");

// Try to initialize!
if (!mpu.begin()) {
  Serial.println("Failed to find MPU6050 chip");
  while (1) {
    delay(10);
  }
}
Serial.println("MPU6050 Found!");

mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
Serial.print("Accelerometer range set to: ");
switch (mpu.getAccelerometerRange()) {
case MPU6050_RANGE_2_G:
  Serial.println("+-2G");
  break;
case MPU6050_RANGE_4_G:
  Serial.println("+-4G");
  break;
case MPU6050_RANGE_8_G:
  Serial.println("+-8G");
  break;
case MPU6050_RANGE_16_G:
  Serial.println("+-16G");
  break;
}
mpu.setGyroRange(MPU6050_RANGE_500_DEG);
Serial.print("Gyro range set to: ");
switch (mpu.getGyroRange()) {
case MPU6050_RANGE_250_DEG:
  Serial.println("+- 250 deg/s");
  break;
case MPU6050_RANGE_500_DEG:
  Serial.println("+- 500 deg/s");
  break;
case MPU6050_RANGE_1000_DEG:
  Serial.println("+- 1000 deg/s");
  break;
case MPU6050_RANGE_2000_DEG:
  Serial.println("+- 2000 deg/s");
  break;
}

mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
Serial.print("Filter bandwidth set to: ");
switch (mpu.getFilterBandwidth()) {
case MPU6050_BAND_260_HZ:
  Serial.println("260 Hz");
  break;
case MPU6050_BAND_184_HZ:
  Serial.println("184 Hz");
  break;
case MPU6050_BAND_94_HZ:
```

```
   Serial.println("94Hz");
   break;
  case MPU6050_BAND_44_HZ:
   Serial.println("44Hz");
   break;
  case MPU6050_BAND_21_HZ:
   Serial.println("21Hz");
   break;
  case MPU6050_BAND_10_HZ:
   Serial.println("10Hz");
   break;
  case MPU6050_BAND_5_HZ:
   Serial.println("5 Hz");
   break;
  }

  Serial.println("");
  delay(100);
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  timer.setInterval(2500L, sendSensor);
 }
void loop(){
Blynk.run();
timer.run();
}

void sendSensor(){
   /* Get new sensor events with the readings */
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  /* Print out the values */
  Serial.print("Acceleration X: ");
  float xx = a.acceleration.x;
  Serial.print(a.acceleration.x);
  Serial.print(", Y: ");
  float yy = a.acceleration.y;
  Serial.print(a.acceleration.y);
  Serial.print(", Z: ");
  float zz = a.acceleration.z;
  Serial.print(a.acceleration.z);
  Serial.println(" m/s^2");
  Serial.println("");
  Blynk.virtualWrite(V2, xx);
  Blynk.virtualWrite(V3, yy);
  Blynk.virtualWrite(V4, zz);
  delay(500);
}
```

To Experience the Outcome

Refer Annexure A – Creating a Blynk IoT Environment



2. Experiencing to Control RGB LED using Blynk

Hardware Interfacing Instruction:

ESP32	MPU6050
3.3v	VCC 3.3v
GND	GND
GPIO 22	SCL
GPIO 21	SDA

Setup your IDE

ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

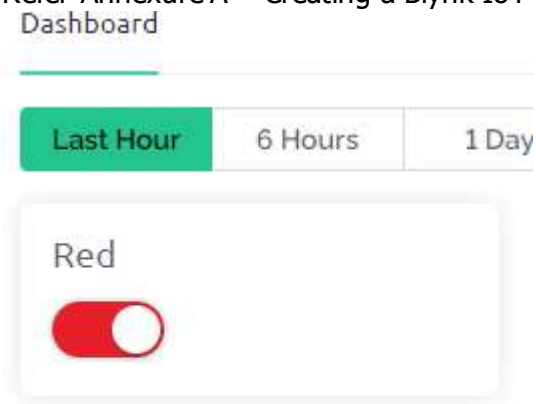
Library Installation

Code

```
//Control LED Using Blynk 2.0/Blynk IOT

#define BLYNK_TEMPLATE_ID "Enter Blyk ID"
#define BLYNK_DEVICE_NAME "LED CONTROL"
#define BLYNK_AUTH_TOKEN "Enter Auth Token"


#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>



char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Type Your SSID";  // Enter your Wifi Username
char pass[] = "Type your Pass";  // Enter your Wifi password

int ledpin = 2;
void setup()
{
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  pinMode(ledpin,OUTPUT);
}

void loop()
{
  Blynk.run();
}
```

To Experience the Outcome

Refer Annexure A – Creating a Blynk IoT Environment



MQTT Protocol

The MQ Telemetry Transport (MQTT) is a protocol for publish and subscribe style messaging. It was originally invented by IBM as part of the MQSeries family of products but since has become an industry standard governed by the Oasis standards group. The latest specification version is 3.1.1.

Being Pub/Sub, this means that there is a broker (an MQTT Broker) to which subscribers can register their subscriptions and publishers can submit their publications. Publications and subscriptions agree on the topics to be used to link the messages together. A client can be a publisher, a subscriber or both.

The value of MQTT is that it can be used to deliver data from an application running on one machine to an application running on another. Immediately we seem to see an overlap between MQTT and REST calls but there are some major differences. In a REST environment, when you form a connection from a client to a server, the server must be available in order for the client to deliver the data. With MQTT that is not necessarily the case. The client can publish a message which can then be held by the broker until such time as the receiving application comes on-line to retrieve it. This is a store and forward mechanism.

Every published message must have a topic associated with it that is used to determine which subscribers would be interested in receiving a copy.

The structure of a topic is broken into topic levels separated by a "/". Subscribers can include wild cards in their topic selections of copies of messages that they would like to receive:

- + – Single topic level wild-card

eg. a/+/c
would subscribe to a/b/c and a/x/c.

- # – Multi topic level wild-card

eg. a/#
would subscribe to a/.<anything>

MQTT is commonly implemented on top of TCP/IP. Clients connect to the broker (not to each other) over a TCP connection.

There is a quality of service requested by a client. This is encoded in the QoS field:

- QoS=0 – Send at most once. This can lose messages. At most once means perhaps never.
- QoS=1 – Send at least once. This means that the message will be delivered. Saying this another way, a message will not be discarded or lost. However, duplicates can arrive … i.e. the message can be delivered twice or more.
- QoS=2 – Send exactly once. This means that the message will not be lost and will be delivered once and once only.

MQTT also has the capability to buffer messages for subsequent delivery. For example, if a client subscriber is not currently connected, a message can be queued or stored for delivery to the client when it eventually re-connects. We call a client that is not connected an off-line client. For a subscription, we have the choice to deliver all the queued messages for a client

or just the last message. To understand the difference, we can imagine a published message that says "I sold your stock for price" … we want all such messages sent to the client because they are all of interest. However if we think of a published message of "Today's forecast is sunny and warm" then there may be no need for old messages and only the current weather forecast is of interest to us. During publishing we can declare that a message is eligible for retention and this is called the "retained message". When a client subscribes, it can ask to receive the last retained message immediately … so even if a subscription takes place after a previous publication, it can still receive data immediately.

Clients make their status known to the broker so the broker can tell if a client is connected. This is achieved via a keep-alive/heartbeat. When forming a connection to a broker, the client provides a keep-alive interval (in seconds). If the broker hasn't received a message from the client in this interval then the broker can disconnect the client assuming it to have been lost/disconnected. If the keep-alive interval is set to 0, then there will be no validation from the broker.

If a client connection is lost because of a network disconnection, the broker can detect that occurrence. This is where we get morbid. We define this as the client having "died". In the real world, when someone dies, there may be a last "will and testament" which are the desired instructions of what the person wanted to happen when they die. MQTT has a similar concept. A client can register a message to be published in the event of the clients death. This is remembered by the broker and in the event of the client dieing, the broker will perform the role of the attorney and publish the last registered "will and testament" message on behalf of the deceased client.

The default port number for an MQTT broker is 1883

Type	Name	Notes
int	cmd	
struct mg_str	payload	The payload of a received message.
uint8_t	connack_ret_code	Used on MG_EV_MQTT_CONNACK.
uint16_t	message_id	Used on MG_EV_MQTT_PUBLISH.
char *	topic	The topic on which the message was received.

Experience to send the DHT11 data via MQTT protocol and control the Appliance via MQTT protocol

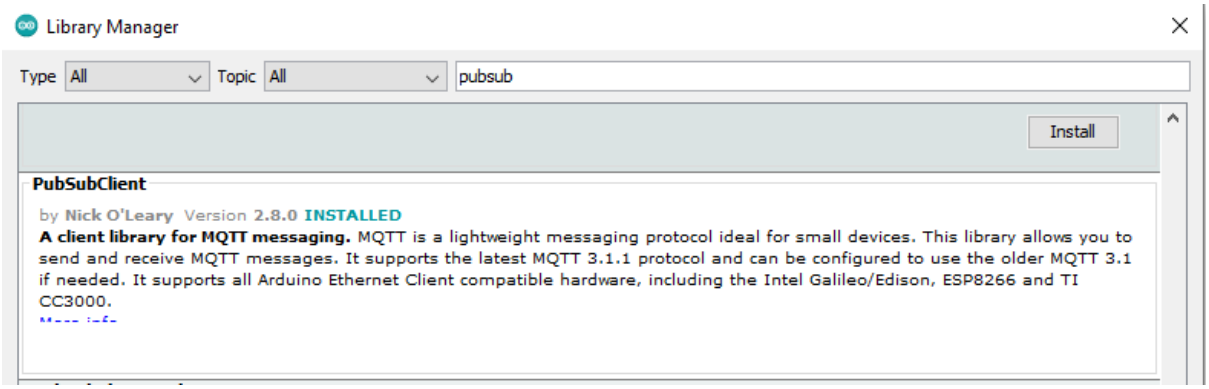
Hardware Interfacing Instruction:

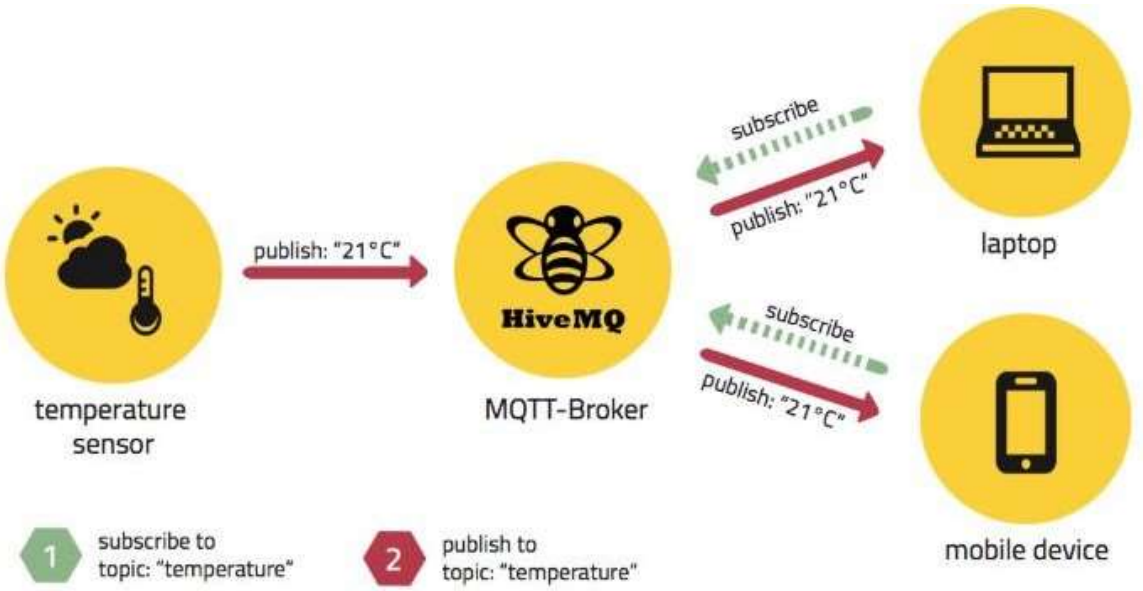
ESP32	MPU6050		
3.3v	VCC 3.3v		
GND	GND		
GPIO 22	SCL		
GPIO 21	SDA		

Setup your IDE

ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

Library Installation





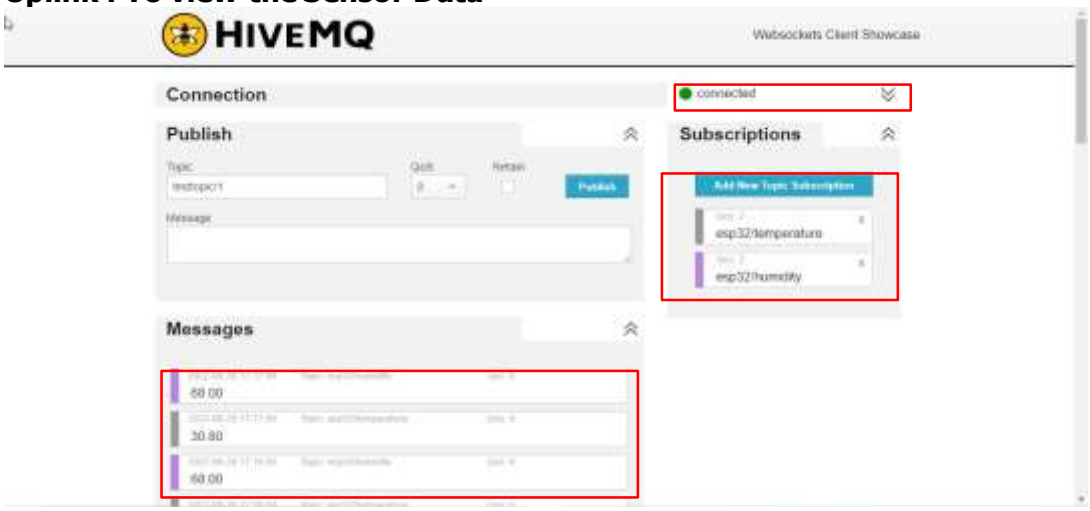
Code

```
#include <Arduino.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"

WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

#define DHTPIN 4    // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);

const char* ssid = "Types your SSID";
const char* password = "Your Password";

char *mqttServer = "broker.hivemq.com";
int mqttPort = 1883;

void setupMQTT() {
  mqttClient.setServer(mqttServer, mqttPort);
  mqttClient.setCallback(callback);
}

void reconnect() {
  Serial.println("Connecting to MQTT Broker...");
  while (!mqttClient.connected()) {
     Serial.println("Reconnecting to MQTT Broker..");
     String clientId = "ESP32Client-";
     clientId += String(random(0xffff), HEX);


     if (mqttClient.connect(clientId.c_str())) {
       Serial.println("Connected.");
       // subscribe to topic
       mqttClient.subscribe("esp32/message");
     }
  }
}

void setup() {
  Serial.begin(9600);
  Serial.println("DHT11 test!");
  dht.begin();
  WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
    }
    Serial.println("");
    Serial.println("Connected to Wi-Fi");
   pinMode(2, OUTPUT);
   setupMQTT();
}

void loop() {
  if (!mqttClient.connected())
```

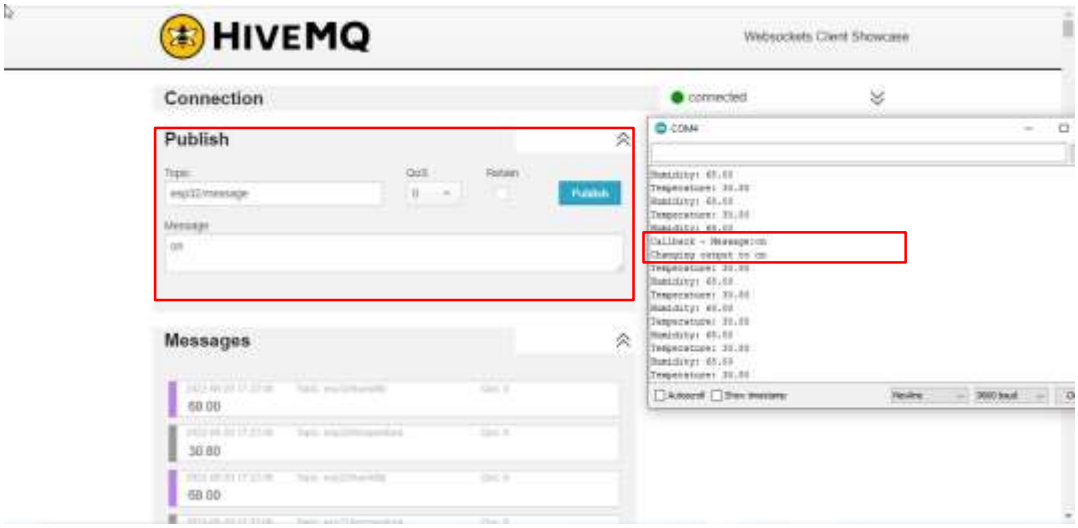
```
  reconnect();
 mqttClient.loop();
 long now = millis();
 long previous_time = 0;

 if (now - previous_time > 1000) {
  previous_time = now;

  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(h) || isnan(t)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
 }


  char tempString[8];
  dtostrf(t, 1, 2, tempString);
  Serial.print("Temperature: ");
  Serial.println(tempString);
  mqttClient.publish("esp32/temperature", tempString);




  char humString[8];
  dtostrf(h, 1, 2, humString);
  Serial.print("Humidity: ");
  Serial.println(humString);
  mqttClient.publish("esp32/humidity", humString);
  delay(2000);


 }
}

 void callback(char* topic, byte* message, unsigned int length) {
 Serial.print("Callback - ");
 Serial.print("Message:");
 String messageTemp;
 for (int i = 0; i < length; i++) {
  Serial.print((char)message[i]);
  messageTemp += (char)message[i];
 }
 Serial.println();
 if (String(topic) == "esp32/message") {
  Serial.print("Changing output to ");
  if(messageTemp == "on"){
   Serial.println("on");
   digitalWrite(2, HIGH);
  }
  else if(messageTemp == "off"){
   Serial.println("off");
   digitalWrite(2, LOW);
  }
 }
}
```

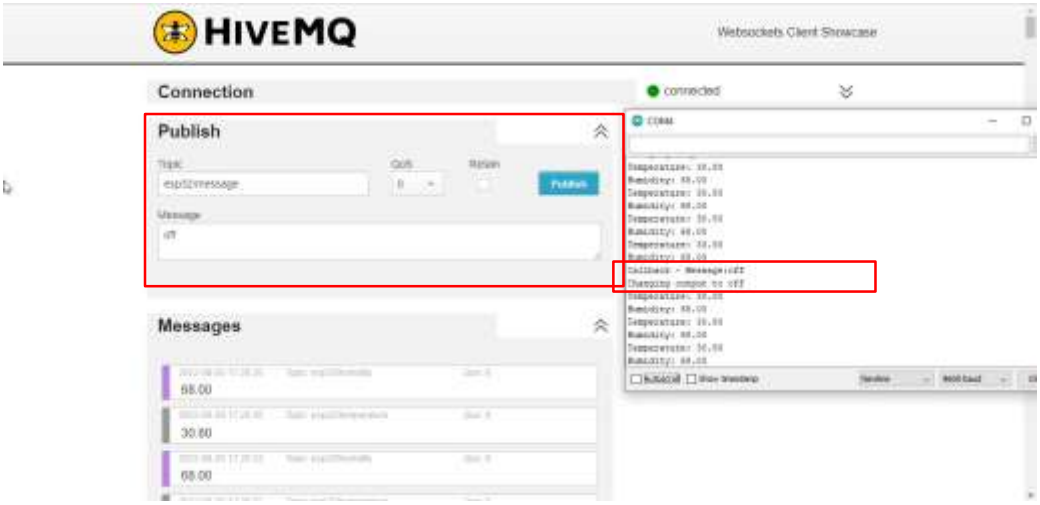
To Experience the Outcome

Uplink : To view the Sensor Data



Downlink: To On & Off the relay





GSM Protocol

GSM Networks: Why they're so popular for cellular IoT

For years, GSM networks were the gold standard of cellular communications. Now known as 2G, GSM technology was a tremendous improvement over the first generation mobile networks, helping to transition cellular technology from analog to digital.

In its early days, the GSM network (now known as 2G) was revolutionary technology, the first to support streaming data for mobile devices with built-in encryption and 50 kilobit-per-second speeds that were blindingly fast at the time. But 2G was surpassed by other standards long ago — the current 4G protocol speed runs between 200MB–1 gigabit per second, and 5G networks operate starting at 1 gigabit per second, running up to 1,000 times faster than 4G.

While 2G technology seems antiquated in today's consumer smartphone world, it remains widely used in Internet of Things (IoT) applications. GSM IoT (EC-GSM-IoT) is a low power wide area network (LPWAN) technology designed to provide long-range, low complexity cellular communications for IoT devices in a way that conserves energy. Most devices that rely on GSM IoT today are battery-powered cellular IoT devices at the edge, such as smart meters, embedded sensors, and asset trackers. But as many countries sunset their 2G networks, IoT designers and managers will need to shift devices to newer standards.

What is a GSM Network?

The global system for mobile communications, more commonly referred to as GSM, rose to prominence in the 1990s and became the most popular cellular network standard on a global scale. By the 2010s, GSM operated in more than 193 countries and achieved more than 90% of market share, including mobile network operators (MNOs) such as AT&T, T-Mobile, and Pine Cellular.

GSM got its start in 1982, when the European Conference of Postal and Telecommunications Administrations created a group dedicated to creating a new mobile technology that could reach across Europe. The standard didn't emerge on commercial markets until 1991, when mobile telephones started to become more common. GSM includes now-expected features like caller ID and call waiting, conferencing, and SMS (text messaging).

The other major network standard that falls under the 2G heading is Code Division Multiple Access (CDMA), used by Verizon, Sprint, and Virgin Mobile. CDMA works by multiplexing — it does not assign a particular frequency to each user, but rather distributes that data over the entire available frequency range, allowing more users to communicate at the same time. There is a limit to how many users multiplexing can accommodate, but it does increase the number of connections that can be made.

How Are GSM Networks Structured?

A GSM network takes a geographical area in need of coverage and divides it up into hexagonal segments, or cells. The size of each cell varies based on its number of users. Densely populated areas might have cells of a few blocks, while rural areas might have cells of up to several miles in radius. As devices move from cell to cell, calls and data are switched from tower to tower, a process referred to as a "handoff."

At the center of each cell is a transmitter and receiver. GSM also differs from CDMA in that data takes turns being transmitted rather than all going at once.

Now let's look at some of the hardware components involved in a GSM network:

Mobile Station
The mobile station provides an access point for connecting to the network. It consists of a device equipped with a SIM card or chip that links the device with a subscriber identity using the international mobile subscriber identity (IMSI) number. The SIM can then link the device to a nearby base station subsystem (BSS).

Base Station Subsystem (BSS)
The base station subsystem (BSS) includes two components: the base transceiver station (BTS) and the base station controller (BSC). These two stations allow components made by different suppliers to communicate with one another.

BASE TRANSCEIVER STATION (BTS)
The BTS is, in effect, the radio transceiver at the center of each cell and serves several important functions such as encoding, encrypting, synchronizing time and frequency, decoding and decrypting, and uplink channel measurements.

BASE CONTROLLER STATION (BCS)
The base controller station manages the radio signals from multiple BTS transceivers, deciding the "turn order" that GSM uses. It also manages frequency hopping, performs traffic concentration, manages power consumption, reallocates frequencies among BTSs, and measures time delays on signals received from the mobile station.

Network and Switching Subsystem (NSS)
The network and switching subsystem (NSS) is in charge of user authentication, roaming features, and security encryption. It essentially refers to the most important components of a 2G core network. In earlier days of GSM, the NSS used the home location register (HLR), message service center (MSC), authentication center (AuC), and visitor location register (VLR) to enable connection-oriented voice calls. Once the GPRS core network was introduced, the NSS also became a part of data connections.

Operations Support System (OSS)
The OSS refers to the functioning of the operations and maintenance center (OCM) and the BSC. The OCM is connected to the BSC and allows operators to monitor network function, providing cost effective support by centralizing information to help with troubleshooting. Carriers use the OSS to manage:

- Subscription services
- Security
- Network configuration
- Performance monitoring
- Maintenance tasks

Why is GSM Still Relevant?

Three decades after its initial introduction, GSM remains an active technology in IoT. Telecommunications leaders have found ways to keep using GSM, applying upgrades to networks to extend their usefulness. Here's why some IoT leaders are still relying on it:

Global Coverage

GSM still has wide coverage in the United States and even wider globally. This opens up Cellular IoT applications to more geographical locations.

Low Bandwidth Requirements

Cellular IoT devices are often transmitting very small amounts of data in short bursts. The higher processing power of newer networks is not needed, and less processing power means less energy consumption — and more infrequent battery changes (or charges) for IoT devices.

Flexibility

GSM networks are generally more compatible with a wider variety of devices, so using them as a base for cellular IoT often gives you greater interoperability between devices and gateways.

Lower Costs

2G cellular modules are less pricey than their 4G counterparts, and data for 2G devices is also more affordable per megabyte.

The Future of GSM and 2G

While 2G is a useful technology for low bandwidth IoT applications, many carriers have made business decisions to end support and allocate 2G spectrum to other areas. Bandwidth isn't infinite, and carriers must constantly balance capacity with customer demand. MNOs in South Africa, India, Ecuador, and many locations in Asia and the Middle East have already shut down their 2G GSM and CDMA networks. In Europe, a 2G sunset is not on the calendar until 2025, while in the U.S. some carriers have already discontinued service and others plan for a sunset in 2022.

With 2G going away, IoT designers should look at other options as backup connectivity for their devices — for example, NB-IoT and Cat-M1 provide excellent LPWAN functionality and have growing coverage areas around the world.

Cellular IoT with Hologram

Whether they require 2G or a higher bandwidth standard, all IoT devices need a dependable source of connectivity. Hologram's IoT SIM card offers seamless, global coverage for IoT devices with access to LTE/4G/3G/2G technologies. With our Hyper eUICC-enabled SIMs, you'll gain access to new connectivity partnerships without any additional carrier negotiations, integrations, or hardware swaps.

Control Relay using GSM – Sim800L through SMS

Hardware Interfacing Instruction:

ESP32	SIM 800L
VCC	VCC
GND	GND
GPIO 4	Tx
GPIO 2	Rx

Setup your IDE

ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

Code

```
//sender phone number with country code
const String PHONE = "+91YourNumber";

//GSM Module RX pin to ESP32 Pin 2
//GSM Module TX pin to ESP32 Pin 4
#define rxPin 4
#define txPin 2
#define BAUD_RATE 9600
HardwareSerial sim800(1);

#define RELAY_1 17
#define RELAY_2 15

String smsStatus,senderNumber,receivedDate,msg;
boolean isReply = false;

void setup() {
  pinMode(RELAY_1, OUTPUT); //Relay 1
  pinMode(RELAY_2, OUTPUT); //Relay 2

  digitalWrite(RELAY_1, LOW);
  digitalWrite(RELAY_2, LOW);
  //delay(7000);

  Serial.begin(9600);
  Serial.println("esp32 serial initialize");

  sim800.begin(BAUD_RATE, SERIAL_8N1, rxPin, txPin);
  Serial.println("SIM800L serial initialize");

  smsStatus = "";
  senderNumber="";
  receivedDate="";
  msg="";

  sim800.print("AT+CMGF=1\r"); //SMS text mode
```

```
  delay(1000);
}

void loop(){
while(sim800.available()){
  parseData(sim800.readString());
}
while(Serial.available()) {
  sim800.println(Serial.readString());
}
} //main loop ends


void parseData(String buff){
  Serial.println(buff);

  unsigned int len, index;
  //Remove sent "AT Command" from the response string.
  index = buff.indexOf("\r");
  buff.remove(0, index+2);
  buff.trim();
  if(buff != "OK"){
    index = buff.indexOf(":");
    String cmd = buff.substring(0, index);
    cmd.trim();


    buff.remove(0, index+2);


    if(cmd == "+CMTI"){
      //get newly arrived memory location and store it in temp
      index = buff.indexOf(",");
      String temp = buff.substring(index+1, buff.length());
      temp = "AT+CMGR=" + temp + "\r";
      //get the message stored at memory location "temp"
      sim800.println(temp);
    }
    else if(cmd == "+CMGR"){
      extractSms(buff);




      if(senderNumber == PHONE){
        doAction();
      }
    }
  }
  else{
  //The result of AT Command is "OK"
  }
}

void extractSms(String buff){
  unsigned int index;
  index = buff.indexOf(",");
  smsStatus = buff.substring(1, index-1);
  buff.remove(0, index+2);


  senderNumber = buff.substring(0, 13);
```

```
    buff.remove(0,19);

    receivedDate = buff.substring(0, 20);
    buff.remove(0,buff.indexOf("\r"));
    buff.trim();


    index=buff.indexOf("\n\r");
    buff = buff.substring(0, index);
    buff.trim();
    msg = buff;
    buff = "";
    msg.toLowerCase();
}
void doAction(){
  if(msg == "relay1 off"){
    digitalWrite(RELAY_1, LOW);
    Reply("Relay 1 has been OFF");
  }
  else if(msg == "relay1 on"){
    digitalWrite(RELAY_1, HIGH);
    Reply("Relay 1 has been ON");
  }
  else if(msg == "relay2 off"){
    digitalWrite(RELAY_2, LOW);
    Reply("Relay 2 has been OFF");
  }
  else if(msg == "relay2 on"){
    digitalWrite(RELAY_2, HIGH);
    Reply("Relay 2 has been ON");
  }


  smsStatus = "";
  senderNumber="";
  receivedDate="";
  msg="";
}
void Reply(String text)
{
    sim800.print("AT+CMGF=1\r");
    delay(1000);
    //sim800.print("AT+CMGS=\""+PHONE+"\"\r");
    //delay(1000);
  sim800.print ("AT+CSMP=17,167,0,0\r");// Configuring TEXT mode

  sim800.print("AT+CMGS=\"");// Send the SMS number
  sim800.print(PHONE);
  sim800.println("\"");
  delay(500);
    sim800.print(text);
    delay(100);
    sim800.write(0x1A); //ascii code for ctrl-26 //sim800.println((char)26); //ascii code for ctrl-26
    delay(1000);
    Serial.println("SMS Sent Successfully.");
}
```

LoRa / LoRaWAN Protocol

Introducing LoRa

What is LoRa?

LoRa is a wireless data communication technology that uses a radio modulation technique that can be generated by Semtech LoRa transceiver chips.
This modulation technique allows long range communication of small amounts of data (which means a low bandwidth), high immunity to interference, while minimizing power consumption. So, it allows long distance communication with low power requirements.

LoRa Frequencies

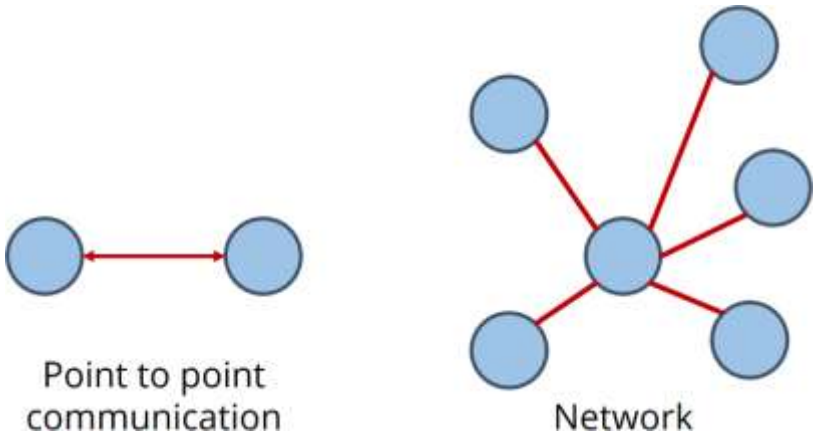
LoRa uses unlicensed frequencies that are available worldwide. These are the most widely used frequencies:

- 865 – 867 for India
- 868 MHz for Europe
- 915 MHz for North America
- 433 MHz band for Asia

Because these bands are unlicensed, anyone can freely use them without paying or having to get a license

LoRa Topologies

You can use LoRa in:

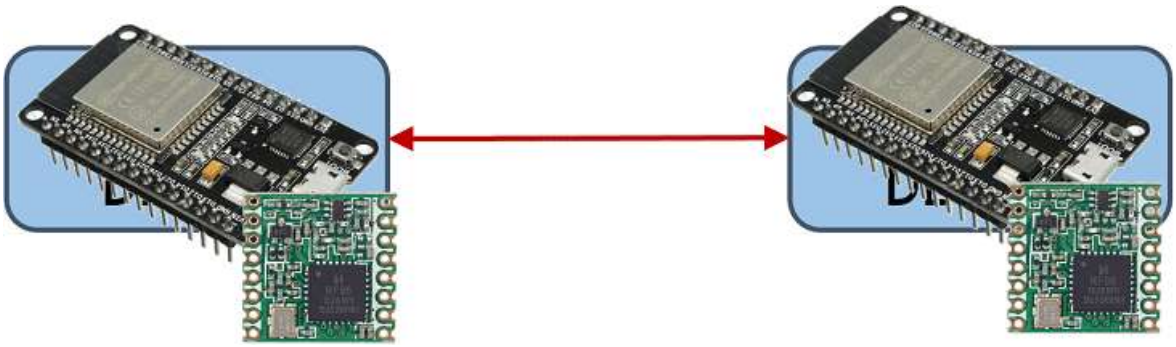


- Point to point communication
- Or build a LoRa network (using LoRaWAN for example)

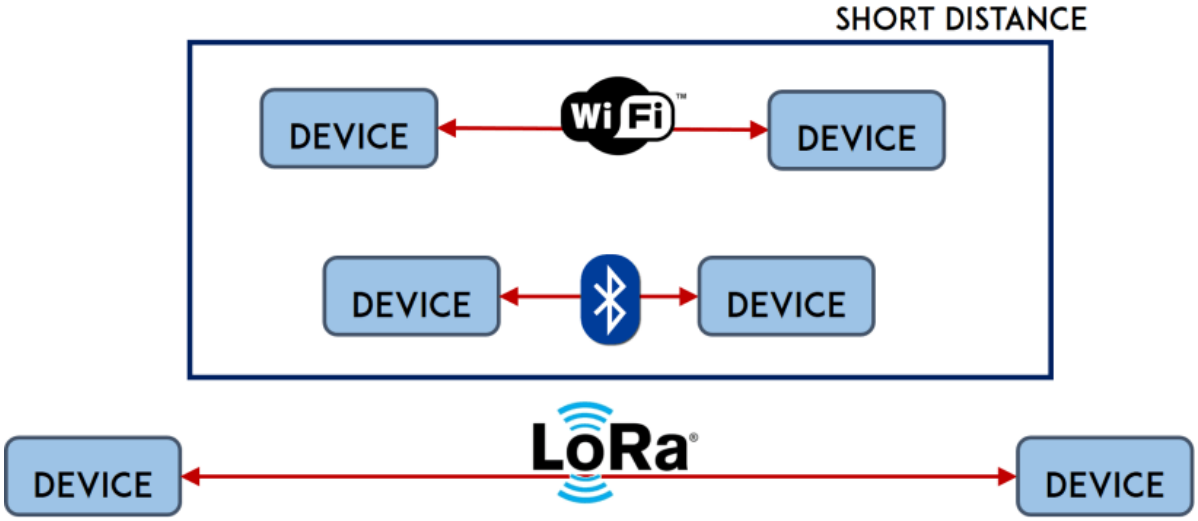
Point to Point Communication

In point to point communication, two LoRa enabled devices talk with each other using RF signals.

For example, this is useful to exchange data between two ESP32 boards equipped with LoRa transceiver chips that are relatively far from each other or in environments without Wi-Fi coverage.



Unlike Wi-Fi or Bluetooth that only support short distance communication, two LoRa devices with a proper antenna can exchange data over a long distance



You can easily configure your ESP32 with a LoRa chip to transmit and receive data reliably at more than 200 meters distance (you can get better results depending on your enviroment and LoRa settings). There are also other LoRa solutions that easily have a range of more than 10Km.

LoRa Applications

LoRa long range and low power features, makes it perfect for battery-operated sensors and low-power applications in:

- Internet of Things (IoT)
- Smart home
- Machine-to-machine communication
- And much more…

So, LoRa is a good choice for sensor nodes running on a coil cell or solar powered, that transmit small amounts of data.

Keep in mind that LoRa is not suitable for projects that:

- Require high data-rate transmission;
- Need very frequent transmissions;
- Or are in highly populated networks.

LoRaWAN

You can also build a LoRa network using LoRaWAN.

The LoRaWAN protocol is a Low Power Wide Area Network (LPWAN) specification derived from LoRa technology standardized by the LoRa Alliance. We won't explore LoRaWAN in this tutorial, but for more information you can check the LoRa Alliance and The Things Network websites.

ABP vs OTAA Activation Method

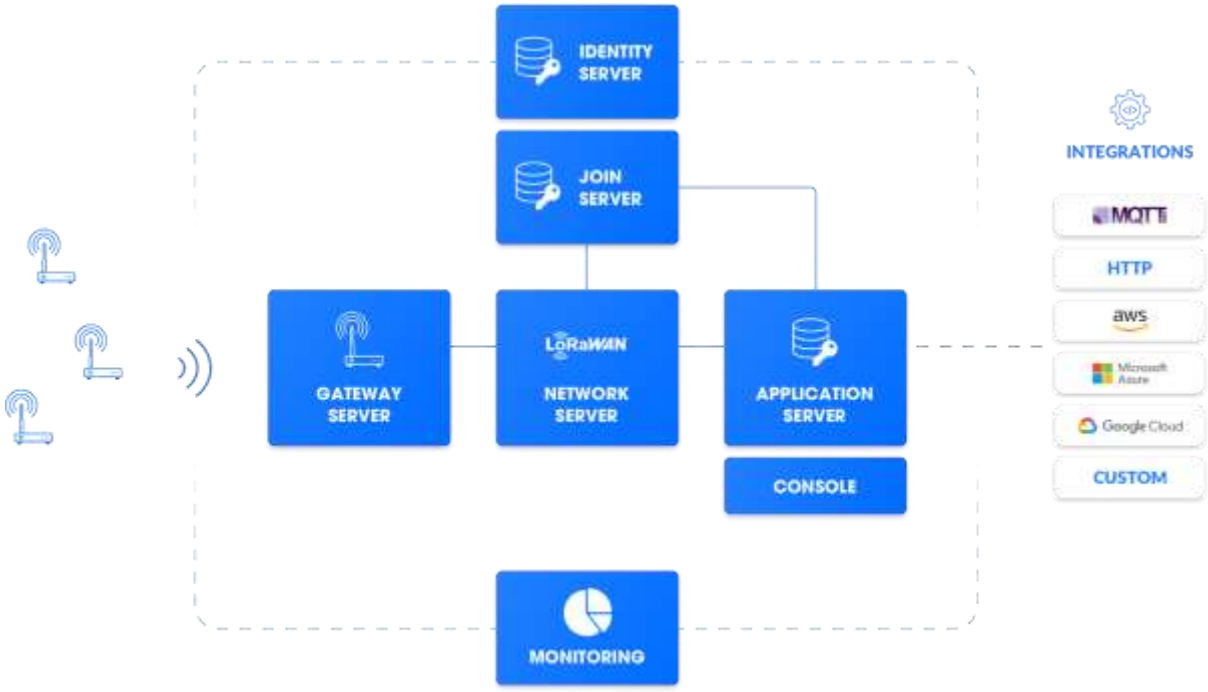
Every end device must be registered with a network before sending and receiving messages. This procedure is known as activation. There are two activation methods available:

Over-The-Air-Activation (OTAA) - The most secure and recommended activation method for end devices. Devices perform a join procedure with the network, during which a dynamic device address is assigned and security keys are negotiated with the device.

Activation By Personalization (ABP) - requires hardcoding the device address as well as the security keys in the device. ABP is less secure than OTAA and also has the downside that devices can not switch network providers without manually changing keys in the device.

The join procedure for LoRaWAN 1.0.x and 1.1 is slightly different. The following two sections describe the join procedure for LoRaWAN 1.0.x and 1.1 separately.

LoRaWAN Architecture:



Experiencing the DHT 11 with LoRaWAN

Hardware Interfacing Instruction:

For LoRa Module

ESP32	LoRa	DHT 11	LED R
GPIO 15	NSS		
GPIO 12	MISO		
GPIO 14	SCK		
GPIO 13	MOSI		
GPIO 17	RESET		
GPIO 4	DIO 0		
GPIO 33	DIO 1		
GPIO 32	DIO 2		
GPIO 27		OUT	
GPIO 02			LED R

Setup your IDE

ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

Library

Refer Annexure B of Setting up the LoRa Environment & Annexure C setting up the TTN

Code

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

#include "DHT.h"

#define DHTPIN 27    // Digital pin connected to the DHT sensor
#define LED 2

#define DHTTYPE DHT11   // DHT 11

// Initialize DHT sensor.

DHT dht(DHTPIN, DHTTYPE);

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const u1_t PROGMEM APPEUI[8]={0x63, 0x89, 0x25, 0x85, 0x69, 0x23, 0x55, 0x15 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}
// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const u1_t PROGMEM DEVEUI[8]={ 0xB2, 0x0A, 0x05, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}
```

```
// LoRaWAN end-device address (DevAddr)
static const u1_t PROGMEM APPKEY[16] = { 0x7F, 0xF4, 0x4E, 0x49, 0x17, 0xFB, 0x85, 0x6F, 0x60, 0x61, 0x3A,
0x19, 0x98, 0x1F, 0x95, 0x03 };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

uint8_t Data;
static uint8_t mydata[2] = {0x00,0x00};
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 30;

void dhttemp()
{
  delay(2000);
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(h) || isnan(t)) {
   Serial.println(F("Failed to read from DHT sensor!"));
   return;
  }
  Serial.print(F("% Temperature: "));
  Serial.print(t);
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.println();

  mydata[0] = t;
  mydata[1] = h;

  delay(2000);//Delay of 1 second for ease of viewing
}

// Pin mapping
const lmic_pinmap lmic_pins = {
   .nss = 15,
   .rxtx = LMIC_UNUSED_PIN,
   .rst = 17,
   .dio = {4, 33, 32},
};

void onEvent (ev_t ev) {
   Serial.print(os_getTime());
   Serial.print(": ");
   switch(ev) {
      case EV_SCAN_TIMEOUT:
         Serial.println(F("EV_SCAN_TIMEOUT"));
         break;
      case EV_BEACON_FOUND:
         Serial.println(F("EV_BEACON_FOUND"));
         break;
      case EV_BEACON_MISSED:
         Serial.println(F("EV_BEACON_MISSED"));
         break;
      case EV_BEACON_TRACKED:
         Serial.println(F("EV_BEACON_TRACKED"));
```

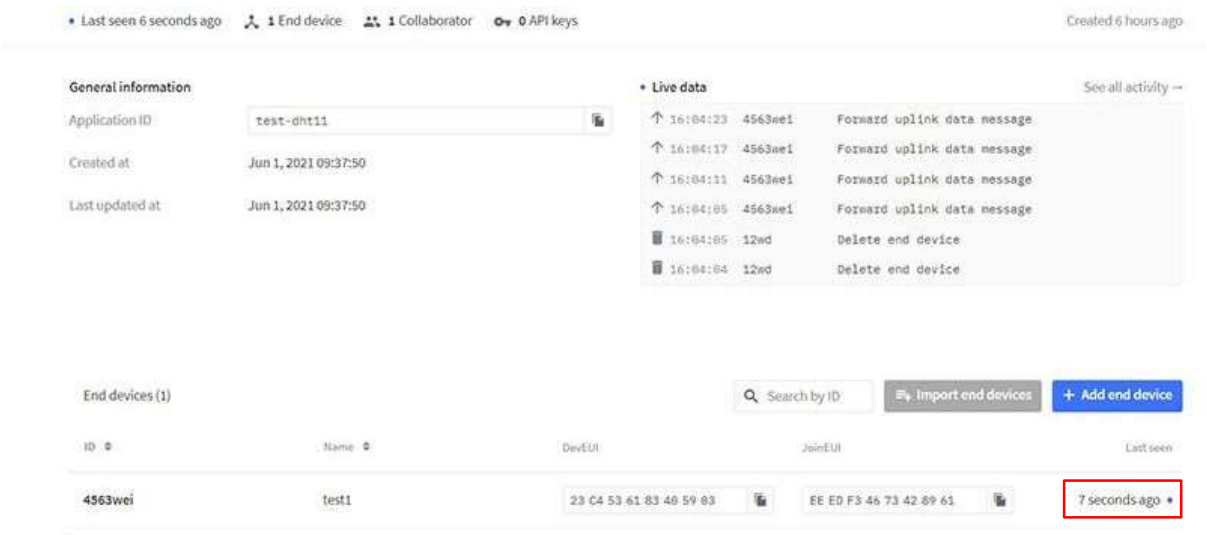
```
    break;
  case EV_JOINING:
    Serial.println(F("EV_JOINING"));
    break;
  case EV_JOINED:
    Serial.println(F("EV_JOINED"));
    break;
  case EV_RFU1:
    Serial.println(F("EV_RFU1"));
    break;
  case EV_JOIN_FAILED:
    Serial.println(F("EV_JOIN_FAILED"));
    break;
  case EV_REJOIN_FAILED:
    Serial.println(F("EV_REJOIN_FAILED"));
    break;
  case EV_TXCOMPLETE:
    Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
    if (LMIC.txrxFlags & TXRX_ACK)
     Serial.println(F("Received ack"));
    if (LMIC.dataLen) {
     Serial.println(F("Received "));
     for (int i = 0; i < LMIC.dataLen; i++)
     {
      if (LMIC.frame[LMIC.dataBeg + i] < 0x10)
      {
        Data = (LMIC.frame[LMIC.dataBeg + i]);
      }
     }
     Serial.print("Downlink Data : ");
     Serial.println(Data);
     if (Data == 1)
     {
      digitalWrite(LED, HIGH);
      Serial.println("LED on");
     }
     else
     {
      digitalWrite(LED, LOW);
      Serial.println("LED off");
     }
    }
    // Schedule next transmission
    os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
    break;
  case EV_LOST_TSYNC:
    Serial.println(F("EV_LOST_TSYNC"));
    break;
  case EV_RESET:
    Serial.println(F("EV_RESET"));
    break;
  case EV_RXCOMPLETE:
    // data received in ping slot
    Serial.println(F("EV_RXCOMPLETE"));
    break;
  case EV_LINK_DEAD:
    Serial.println(F("EV_LINK_DEAD"));
```

```
            break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE"));
            break;
        default:
            Serial.println(F("Unknown event"));
            break;
    }
}

void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND)
    {
        Serial.println(F("OP_TXRXPEND, not sending"));
    }
    else
    {
        dhttemp();
        // Prepare upstream data transmission at the next possible time.
        LMIC_setTxData2(1, mydata, sizeof(mydata), 0);
        Serial.println(F("Packet queued"));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

void setup() {


    pinMode(LED, OUTPUT);
    Serial.begin(9600);
    dht.begin();
    Serial.println(F("Starting"));

    #ifdef VCC_ENABLE
    // For Pinoccio Scout boards
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
    #endif

    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();

      // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;

    // Set data rate and transmit power for uplink (note: txpow seems to be ignored by the library)
    LMIC_setDrTxpow(DR_SF7,14);

    // Start job
    do_send(&sendjob);
}
```

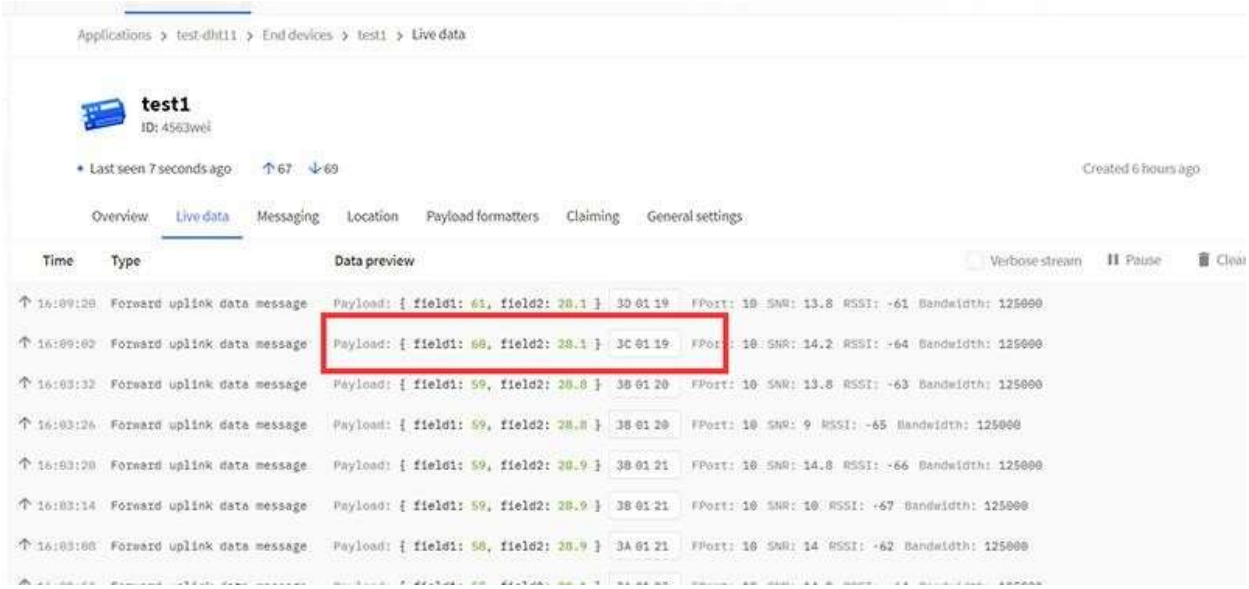
```
void loop() {
  os_runloop_once();
}
```

To Experience the Outcome

Uplink



As the picture, the original payload is **3C0119**, the date after payload formatter decoded are 60 and 28.1, it mean that the **temperature is 28.1 °C and humidity is 60%.**



To Decode this data write a Payload in Payload Formatter in End Device section

```
function Decoder(bytes,port) {
var decoded ={};

if (bytes[0] +bytes [1] + bytes [2] +bytes [3] ==bytes[4]) {
var humidity =bytes [0] + bytes [1] /10;
var scaleValue = bytes[3] & 0xEF;
var signValue = bytes[3] & 0x80;
var temperature = bytes [2] + scaleValue / 10

if (signValue)
        temperature =-temperature

decoded field1= temperature;
decoded field2 =humidity;
}
return decoded;
```

Controller to Controller Communication

1. Sending DHT 11 sensor Data form Arduino UNO to Blynk through ESP32

Hardware Interfacing Instruction:

ESP32	Uno	DHT 11
GPIO 17 (Tx2)	D2 (Rx)	
GPIO 3 (Rx2)	D3 (Tx)	
	D8	OUT

Setup your IDE

ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**

Library

Refer Annexure A for Setting a Blynk IoT Platform

Code

For Arduino

```
#include "DHT.h"
#include <SoftwareSerial.h>

SoftwareSerial esp(2,3);// (Rx, Tx)
#define DHTPIN 8    // Digital pin connected to the DHT sensor

// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11

DHT dht(DHTPIN, DHTTYPE);

int sdata1 = 0;// humidity
int sdata2 = 0;// temperature
String cdata;// complete data, consisting of sensors values

void setup() {
  Serial.begin(9600);
  esp.begin(9600);
  Serial.println("DHTxx test!");

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  if(esp.available()== 0 )
{
  delay(2000);
```

```
 // Reading temperature or humidity takes about 250 milliseconds!
 // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
 float h = dht.readHumidity();
 // Read temperature as Celsius (the default)
 float t = dht.readTemperature();
 // Check if any reads failed and exit early (to try again).
 if (isnan(h) || isnan(t)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
 }
  sdata1 = h;
  sdata2 = t;
  cdata = cdata + sdata1+","+sdata2;// comma will be used a delimeter
  Serial.println(cdata);
  esp.println(cdata);
  delay(500);
  cdata = "";
}
}
```

For ESP32

```
#define BLYNK_TEMPLATE_ID "TMPLGEUxwyI5"
#define BLYNK_DEVICE_NAME "DHT11"
#define BLYNK_AUTH_TOKEN "GCbkm8zDg28JNH7WXRK5VAqaD6SsS4N9"

#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
//#include <SoftwareSerial.h>

char auth[] = BLYNK_AUTH_TOKEN;

char ssid[] = "SSID"; // type your wifi name
char pass[] = "Password!"; // type your wifi password

BlynkTimer timer;

#define RXD2 16
#define TXD2 17

String  myString; // complete message from Arduino, which consists of sensors data
char rdata; // received characters
int firstVal, secondVal,thirdVal; // sensors

//void myTimerEvent()
//{
// // You can send any value at any time.
// // Please don't send more that 10 values per second.
// Blynk.virtualWrite(V1, millis() / 1000);
//
//}

void setup()
```

```
{
 // Debug console
 Serial.begin(9600);
 Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
 Blynk.begin(auth, ssid, pass);
 timer.setInterval(3000L,sensorvalue1);
 timer.setInterval(3000L,sensorvalue2);

}

void loop()
{
  if (Serial2.available() ==0)
  {
   Blynk.run();
   timer.run();// Initiates BlynkTimer
   }

 if (Serial2.available() > 0 )
 {
  rdata = Serial2.read();
  myString = myString+ rdata;
  Serial.println(rdata);


  if( rdata == '\n')
  {


  String l = getValue(myString, ',', 0);
  String m = getValue(myString, ',', 1);


  firstVal  =  l.toInt();
  secondVal =  m.toInt();




  myString = "";// end new code

  }
 }


}


void sensorvalue1()
{
 int sdata1 = firstVal;// humidity value
 Blynk.virtualWrite(V2, sdata1);
 Serial.print("data1:");
 Serial.println(sdata1);


}
void sensorvalue2()
{
int sdata2 = secondVal;// temperature value
 Blynk.virtualWrite(V3, sdata2);
 Serial.print("data2:");
 Serial.println(sdata2);
```

```
String getValue(String data, char separator, int index)
{
  int found = 0;
  int strIndex[] = { 0, -1 };
  int maxIndex = data.length() - 1;


  for (int i = 0; i <= maxIndex && found <= index; i++) {
    if (data.charAt(i) == separator || i == maxIndex) {
      found++;
      strIndex[0] = strIndex[1]+ 1;
      strIndex[1] = (i == maxIndex) ? i+1 : i;
    }
  }
  return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

2. Sending DHT 11 sensor Data form STM32 to Blynk through ESP32

Hardware Interfacing Instruction:

ESP32	STM32	DHT 11
GPIO 17 (Tx2)	PA3 (Rx)	
GPIO 3 (Rx2)	PA2 (Tx)	
	PA0	OUT

Setup your IDE

ESP32	**Refer Above in Introduction to Software**	**Refer Page No:13**
STM32	Refer Above in Introduction to Software	

Library

Refer Annexure A for Setting a Blynk IoT Platform

Code

For STM32

```
#include "DHT.h"
#define DHTPIN PA0    // Digital pin connected to the DHT sensor

// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11

DHT dht(DHTPIN, DHTTYPE);

int sdata1 = 0;// humidity
int sdata2 = 0;// temperature
```

```
String cdata; // complete data, consisting of sensors values

void setup() {
  Serial.begin(9600);
  Serial2.begin(9600); // PA2 - TX PA3 -RX
  Serial.println("DHTxx test!");
  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  if(Serial2.available() == 0 )
{
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();


  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }


// Serial.print("Humidity: ");
// Serial.print(h);
// Serial.print("% Temperature: ");
// Serial.print(t);
// Serial.println("°C");

   sdata1 = h;
   sdata2 = t;
   cdata = cdata + sdata1+","+sdata2; // comma will be used a delimeter
   Serial.println(cdata);
   Serial2.println(cdata);
   delay(500);
   cdata = "";

}
}
```

For ESP32

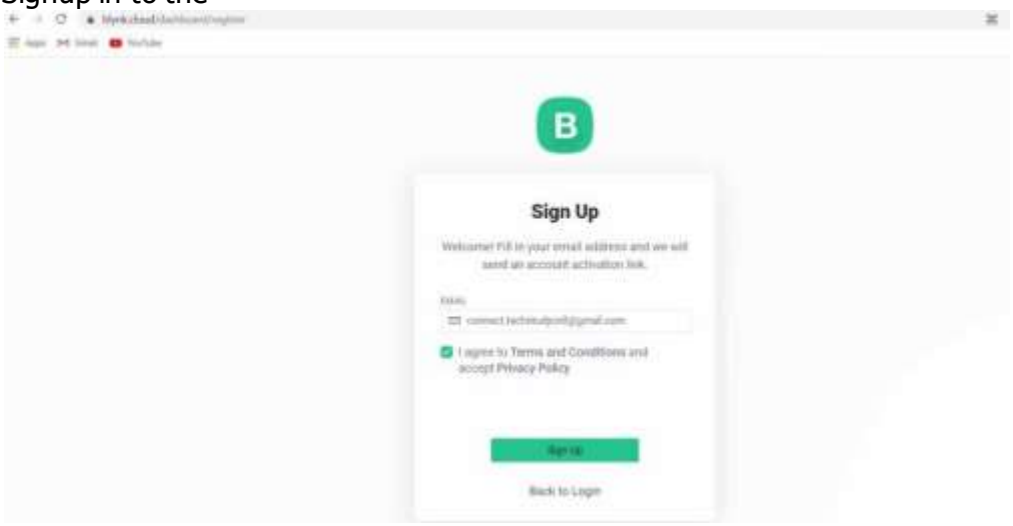
```
#define BLYNK_TEMPLATE_ID "TMPLGEUxwyI5"
#define BLYNK_DEVICE_NAME "DHT11"
#define BLYNK_AUTH_TOKEN "GCbkm8zDg28JNH7WXRK5VAqaD6SsS4N9"

#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
```

```
//#include<SoftwareSerial.h>

char auth[] = BLYNK_AUTH_TOKEN;

char ssid[] = "SSID"; // type your wifi name
char pass[] = "Password!"; // type your wifi password

BlynkTimer timer;

#define RXD2 16
#define TXD2 17

String  myString; // complete message from Arduino, which consists of sensors data
char rdata; // received characters
int firstVal, secondVal,thirdVal; // sensors

//void myTimerEvent()
//{
// // You can send any value at any time.
// // Please don't send more that 10 values per second.
//  Blynk.virtualWrite(V1, millis() / 1000);
//
//}

void setup()
{
  // Debug console
  Serial.begin(9600);
  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
  Blynk.begin(auth, ssid, pass);
  timer.setInterval(3000L,sensorvalue1);
  timer.setInterval(3000L,sensorvalue2);

}

void loop()
{
   if (Serial2.available() ==0 )
   {
    Blynk.run();
    timer.run();// Initiates BlynkTimer
    }

  if (Serial2.available() > 0 )
  {
   rdata = Serial2.read();
   myString = myString+ rdata;
   Serial.println(rdata);


   if( rdata == '\n')
   {


   String l = getValue(myString, ',', 0);
   String m = getValue(myString, ',', 1);


   firstVal  =  l.toInt();
   secondVal =  m.toInt();
```

```
    myString = "";// end new code

    }
   }


}


void sensorvalue1()
{
  int sdata1 = firstVal;// humidity value
  Blynk.virtualWrite(V2, sdata1);
  Serial.print("data1: ");
  Serial.println(sdata1);


}
void sensorvalue2()
{
int sdata2 = secondVal;// temperature value
  Blynk.virtualWrite(V3, sdata2);
  Serial.print("data2 : ");
  Serial.println(sdata2);


}


String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;


    for (int i = 0; i <= maxIndex && found <= index; i++){
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1]+ 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

Annexure A – Creating a Blynk IoT Environment

1. Open the URL https://blynk.cloud/
2. Signup in to the



3. Create Template in Blynk IoT Cloud

4. After login to Blynk IoT account, first you have create a template. In the dashboard click on the Templates from the left side menu.

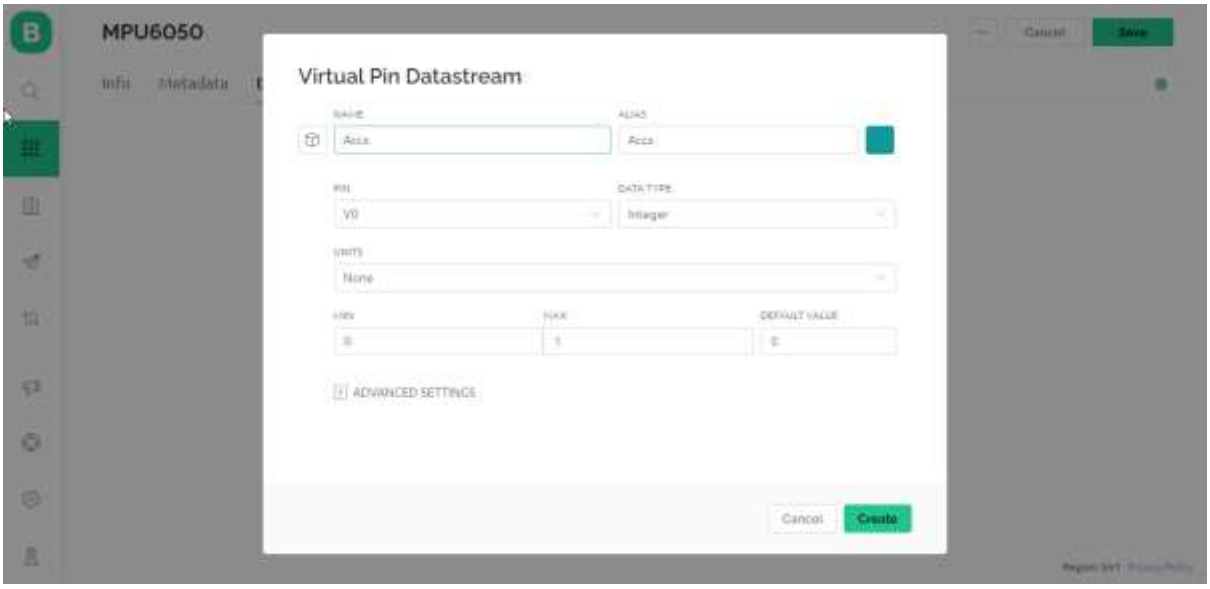
5. Then click on New Template.

6. Then, you have to enter a name for that template.

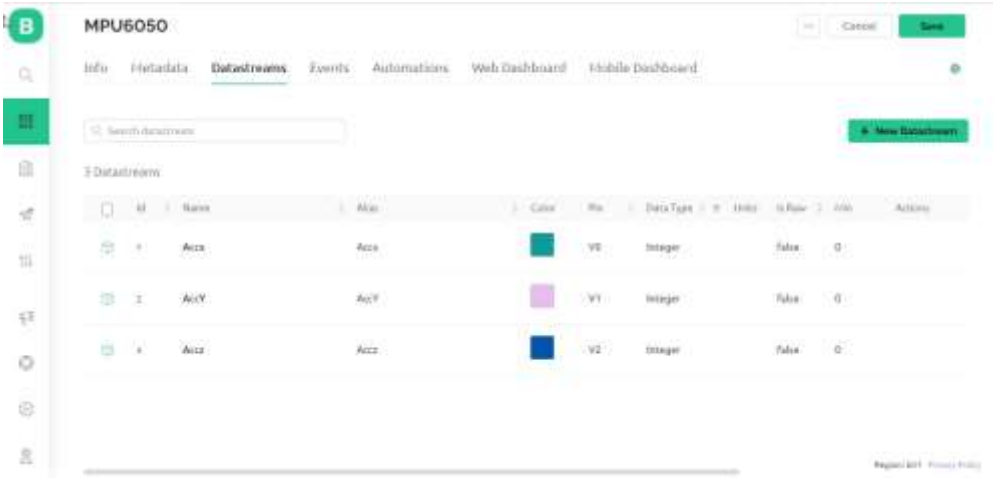
7. Select the microcontroller or board you will use in the project from the Hardware dropdown menu.

8. The connection type should be WiFi.
9. You can enter small description for the template. This field is optional.
10. Now click on Done.
11. Create Datastreams in the Blynk IoT template

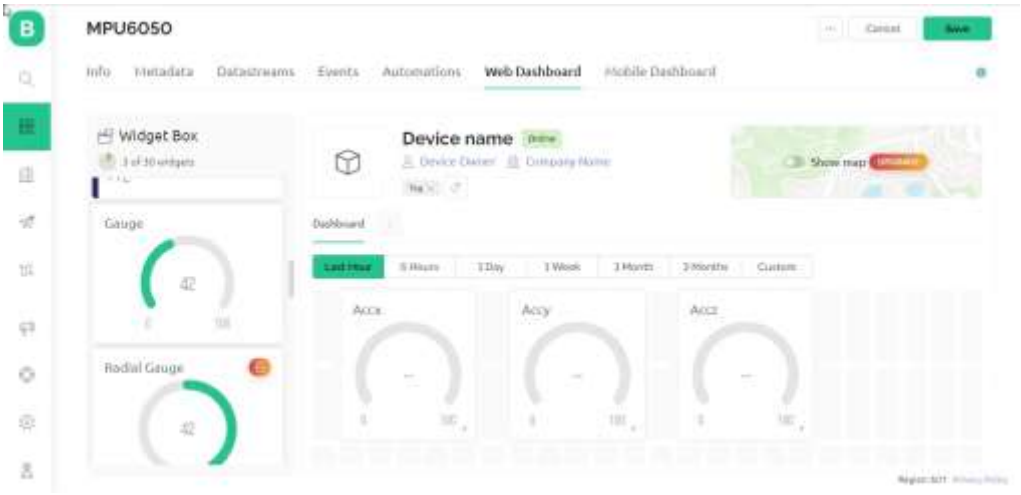
12. After creating the template, go to Datastreams tab.
13. Then click on "New Datastream" and select Virtual Pin.



14. Now, you have to enter a name for this Datastream.
15. Select the Virtual Pin and Datatype from the dropdown menu as per the requirement.
16. Here I will control a relay, so I have selected Integer, For the Temperature you have to select Double datatype and related UNIT.
17. You can also set the MAX, MIN, and default value for the datastream.
18. After that click on Create.

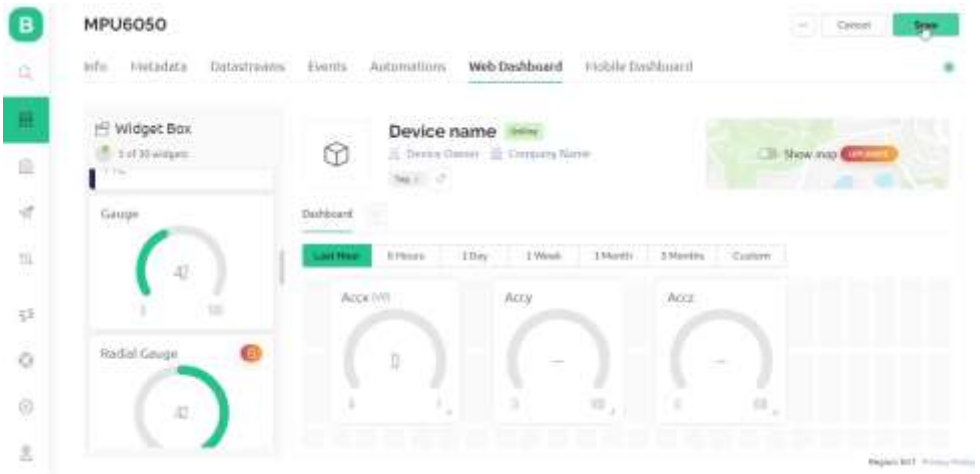


Create Web Dashboard in the Blynk IoT template

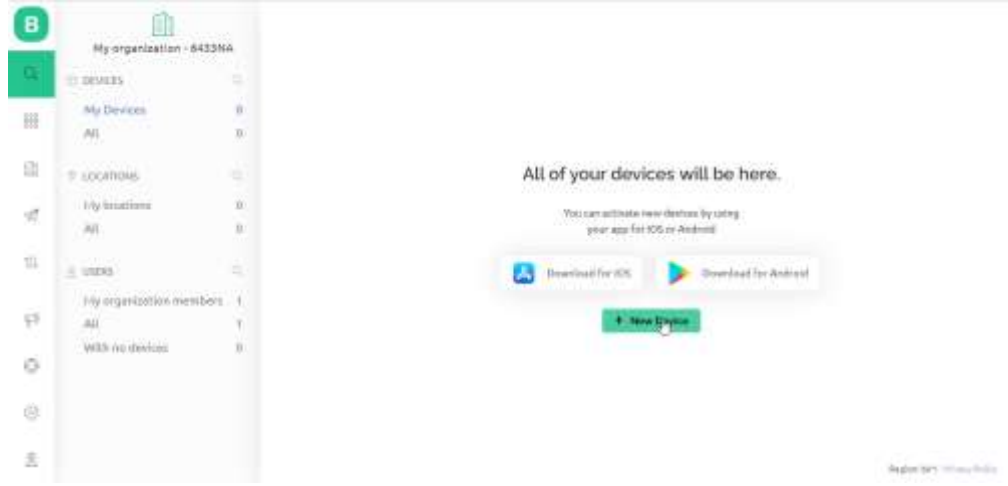


19. Now, go to the Web Dashboard tab to add widgets.
20. In the dashboard, you can easily drag and drop any widgets from the left side.
21. Here I have added 3 Gauge widgets to View Values.
22. Now, if you hover over the widget, you will find a button with a gear icon. Click on the button to go to setting.

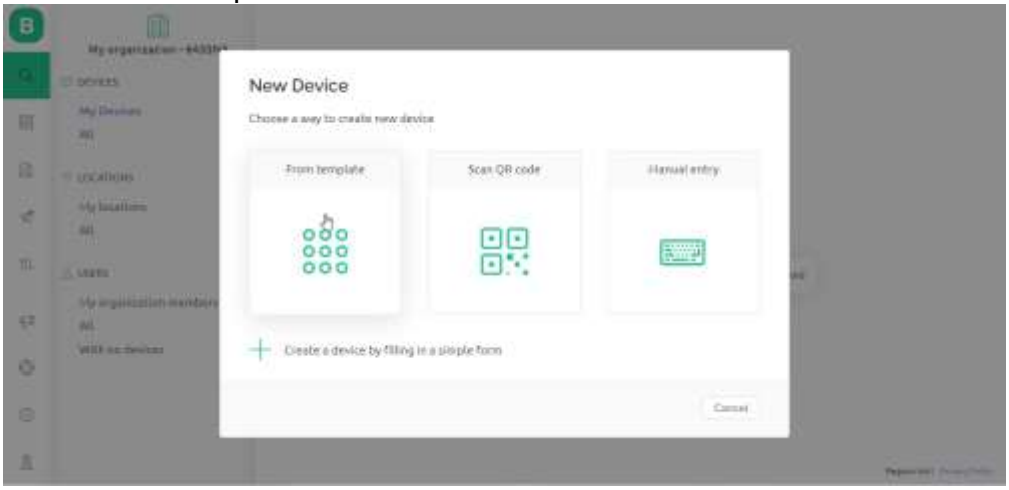
23. Click on Save



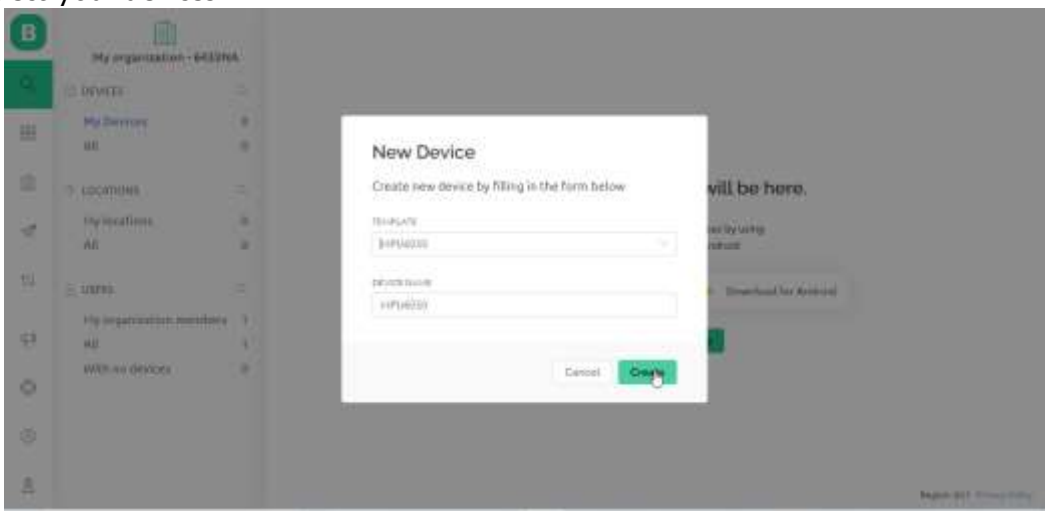
24. Go to Organization click on "New Device"



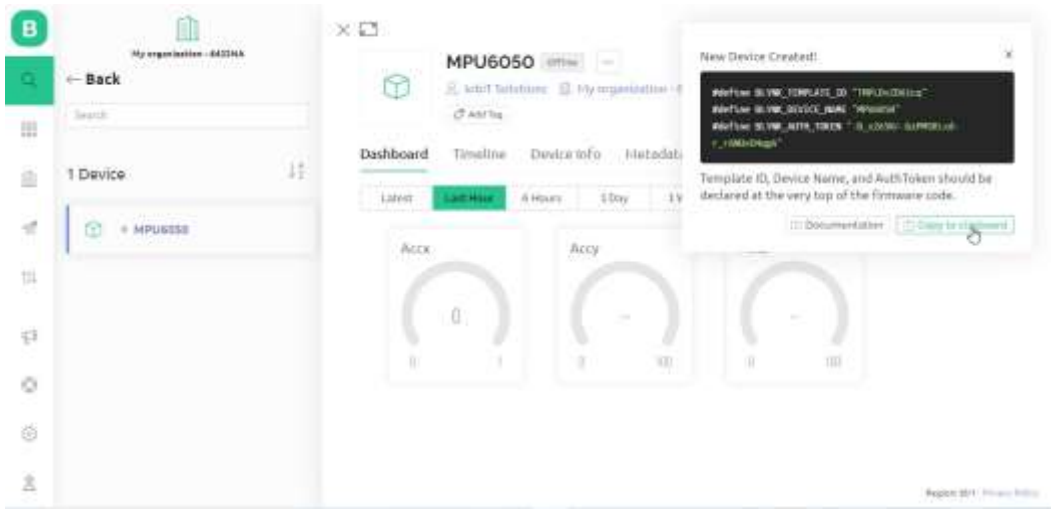
25. Choose from Template



26. Select your devices



27. In the code, you just have to update the BLYNK_TEMPLATE_ID, BLYNK_DEVICE_NAME & BLYNK_AUTH_TOKEN.



Annexure B - Setting Up the LoRa Environment

Hardware Interfacing Instruction:

ESP32	**Refer Annexure C – 3 Interfacing DHT11**	
Pin Configuration	**ESP32** / **LoRa** GPIO 15 / NSS GPIO 12 / MISO GPIO 14 / SCK GPIO 13 / MOSI GPIO 17 / RESET GPIO 4 / DIO 0 GPIO 33 / DIO 1 GPIO 32 / DIO 2	

Setup your IDE

Arduino UNO	Plug and Select the COM Port in Tools Menu and Install your Code to Controller	
ESP32	Refer Above in Introduction to Software	Refer Page No:13

Library Installation

- Download LMIC Library from **https://www.arduinolibraries.info/libraries/mcci-lo-ra-wan-lmic-library**
- Current Version using for this Demo is **MCCI_LoRaWAN_LMIC_library-4.0.0.zip**
- Extract the zip as Folder

- Install Libraries ..\MCCI_LoRaWAN_LMIC_library-4.0.0\project_config and open lmic_project_config.h and make changes as follows for Indian Frequency and save it.

```
// project-specific definitions
//#define CFG_eu868 1
//#define CFG_us915 1
//#define CFG_au915 1
//#define CFG_as923 1
// #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP    /* for as923-JP */
//#define CFG_kr920 1
#define CFG_in866 1
#define CFG_sx1276_radio 1
//#define LMIC_USE_INTERRUPTS
```

- Go to Installed Libraries location …..\MCCI_LoRaWAN_LMIC_library-4.0.0\src\hal and open hal.cpp and find below line

Before Change	After Change

`static void hal_spi_init () {`
`  SPI.begin();`
`}`
	`static void hal_spi_init () {`
	`  SPI.begin(14,12,13,15);`
	`}`

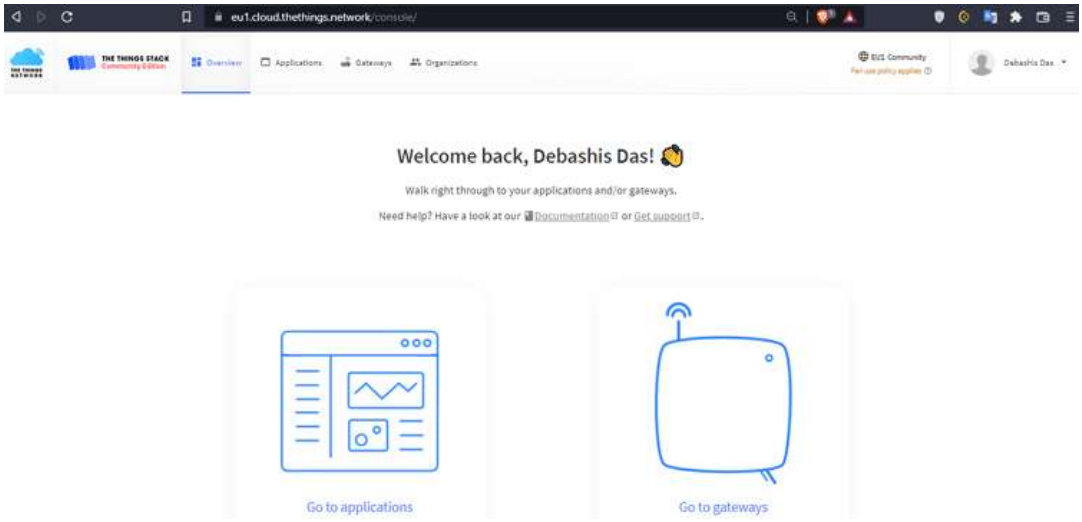
Note: The Pin definition for SPI Pins were as follows CLK – 14, MISO – 12, MOSI – 13, NSS – 15

- Make changes in SPI.begin() as follows and save it
- Copy and paste this folder MCCI_LoRaWAN_LMIC_library-4.0.0 in followinglocation ….\Documents\Arduino\libraries

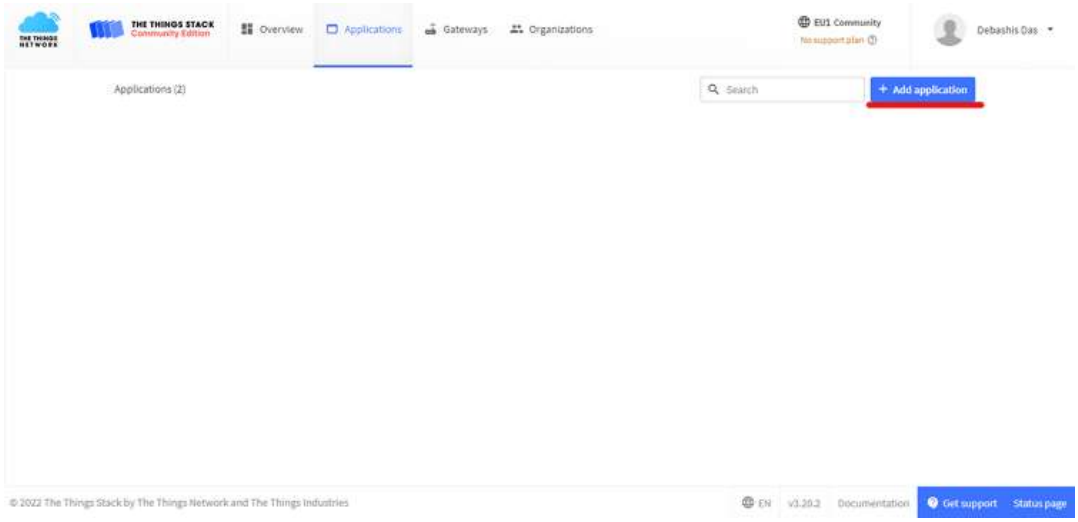
Annexure C - Setting Up the End Nodes on the TTN Network

To transmit data to the TTN network, you need to first set up your gateway, and once that is done you need to set up your end nodes properly in order for the device to work properly. For that, you need to make a TTN account or log in to your existing TTN account. Once you have successfully logged in you need to go enter the TTN Console. If you have done everything correctly, you will be presented with the screen shown in the image below-

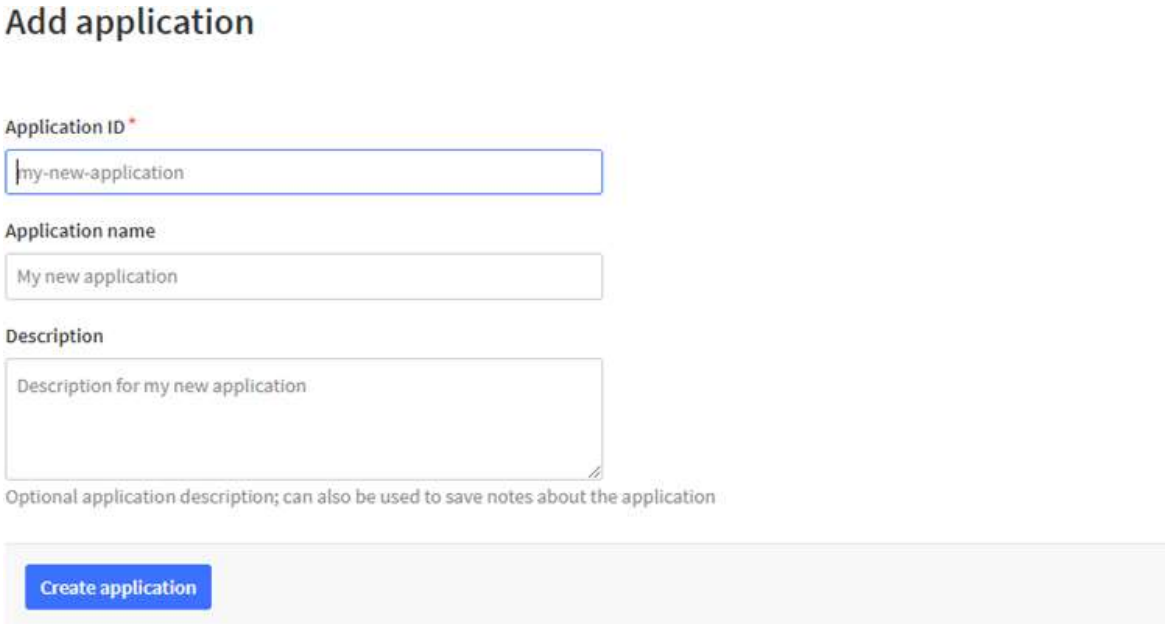
Once we are here we need to select a cluster, if you want to know more about the cluster you can click on the more information button. For our case, we will choose europe1.



Once you do this you will be presented with the screen that is shown above. Now click on "Go to application".

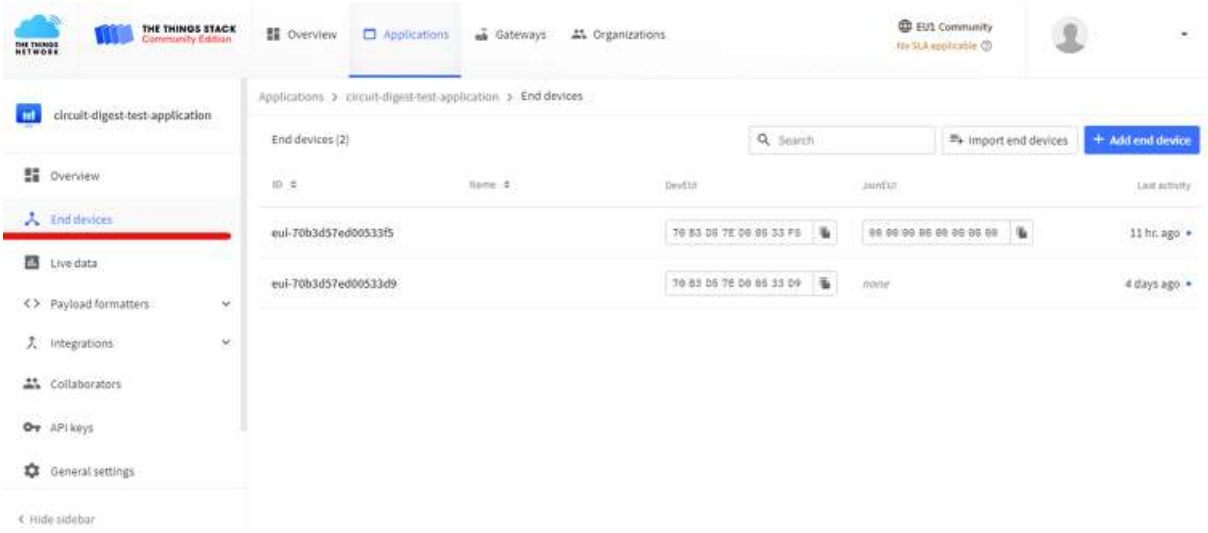


You will be presented with the screen shown above, here you need to click on the Add application button to create your first application.

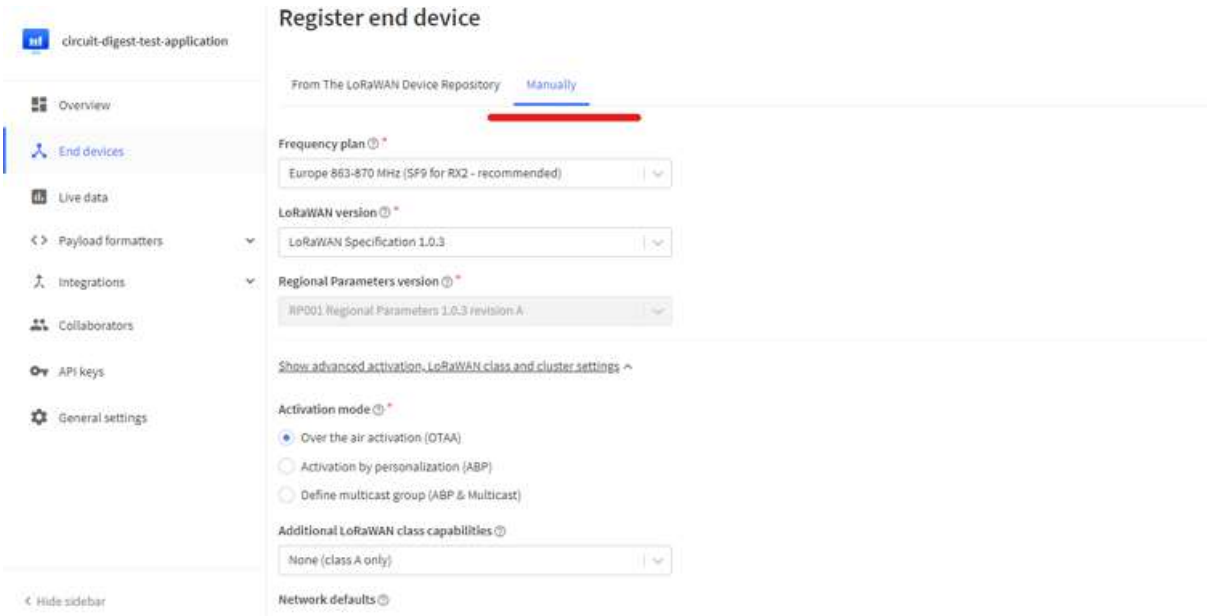


Now you need to create a new application by providing it with a unique Application ID and name. Once that is done the Description part is optional and you can click on the create application button.

Once done, you will be presented with the screen that is shown above, now you need to create End Nodes for your application, as mentioned earlier, end nodes are the devices that send data to the application server.

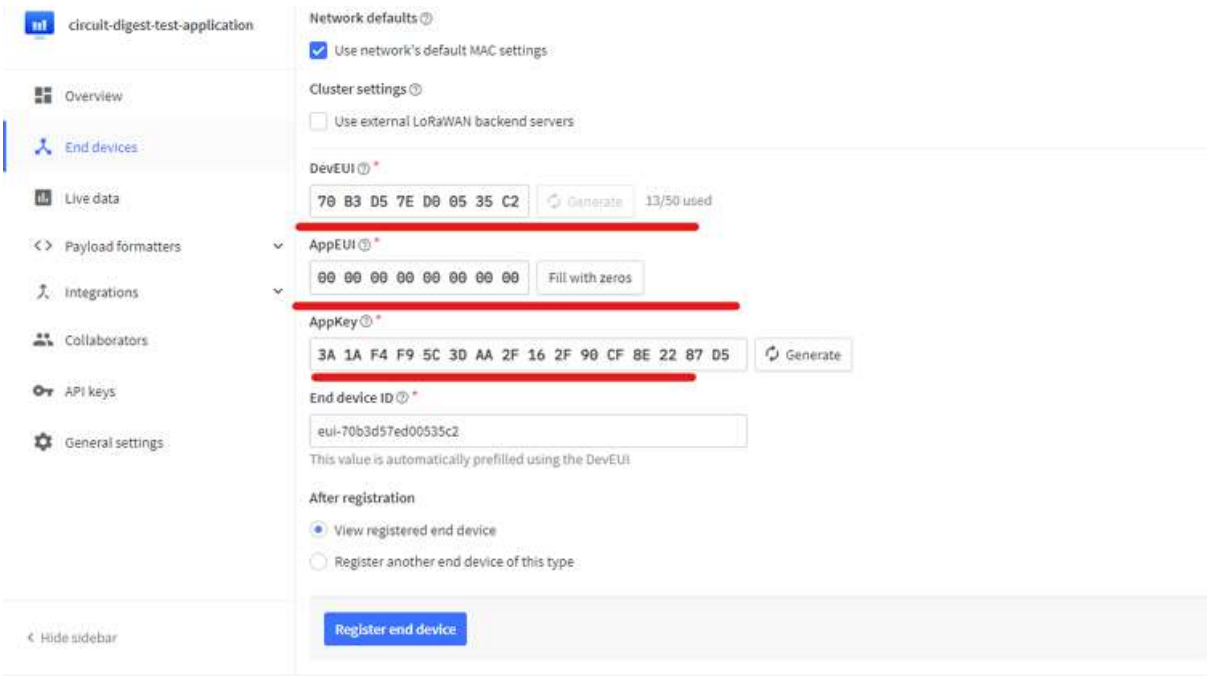


Now click on the newly created Application and click on the end node button on the left and click on add application button on the right. As you can see from this demonstration we have already created two end nodes. One is to test the ABP method and the other one is to test the **OTAA method.**



Now once you click on the add end node button, you will be presented with the screen that is shown above. Now click on it manually and follow along. First, we select the frequency plan,

the LoRaWAN Specification, and the Activation Mode and for our first example, we are configuring it as OTAA.



Next, all you need to do is **generate the keys for your end nodes** and you are all done. Now click on the Register end device button to complete the process. Please keep in mind that your **DeviceEUI, AppKey, and AppEUI** are the three most important keys that you need to be careful with. Now all that is done, you can click on the Register end device button to complete the process.

Note: Now if you look at the code carefully you will find placeholders for the keys that we copied earlier. Paste in our keys according to MSB or LSB format. If you check out the comments above you can see which is which.

Now we are all set on the TTN side and finally move on to the coding part.