Table of Contents

Contents

List of Figures

Abstract

This work provides the development of a low-cost automatic weather station and air quality monitor system capable of measuring meteorological and air quality data in a specific region. Self-feeding by using LiPo batteries and solar panels to fully operate the system. The weather station and air quality monitor system uses two microcontroller systems: Arduino Pro Mini for gathering sensor data (DHT22 – Temperature and Humidity and SDS011 particulate matter for measuring both PM10 and PM2.5 particles) and sending by request to ESP32 development board with LTE capability via I2C communication. Then the data is sent to LAMP server (PHP, MySQL) with POST method and displayed on an Android application via GET method. The data is updated every 10 minutes automatically until 5PM and afterwards it's set to update hourly due to maintaining battery life during the night.

The sensor data can be accessed from anywhere.

Keywords: Mobile Data, Lithium Polymer battery, Air Quality, Internet Of Things, Remote Monitoring.

Introduction

Internet of Things is a novel paradigm combining telecommunications and any kind of device or applications using sensors, tags, microcontrollers and ARM processors. This paper proposes an implementation of weather monitoring system using Internet of Things (IoT). An Arduino based implementation is proposed to monitor PM2.5, PM 10, temperature, humidity of the Particulate Matter pollutants available. Internet of Things is playing a leading role in providing solutions to many applications with the support of software, internet and embedded systems. There are various IoT devices available in the market ranging from micro controllers to microprocessors. There are many technologies developed for weather monitoring using IoT devices. Mid-air contamination is the one of the major problems in this society not only does it effect the food, animals but it's also highly effected to human beings, also causes several diseases which leads to death. It's one of the major issues among several natural disasters so I have to refine all of the contamination from the nature to lead a peaceful life, so I have made these by utilizing new technologies. To study about moisture, sulphur oxide, automobile monoxide and dioxide and also implementing solutions to solve all this contamination.

Problem statement

Seamless data monitoring for different types of sensors both air quality and environmental/climate sensors while trying to maintain low battery consumption and to connect multiple sensors to the ESP32 Microcontroller and to power the sensors. To also make sure there is communication between the weather station and the backend. Making sure the end user can see all of the sensor data with ease. To be alarmed when there is change in weather conditions, low air quality/high pollution, and most importantly battery voltage. All of that integrated in one Android application.

Outline of the thesis

In this thesis, the following topics will be covered, as presented below:

- Overview of the technologies and tools used, such as: Arduino IDE, C/C++, ESP32, Arduino Pro Mini, DHT22, PHP, MySQL, Android Studio, Java, Mobile Data, LTE, I2C Communications, Solar, LiPo battery, EasyEDA.
- My view of the problem and the way I implemented all the forementioned technologies to my aid.

Overview of Technologies and Tools

Arduino IDE

Arduino IDE was created by Arduino Software for alongside usage of the Single-board microcontroller Arduino, which was created by the same company. The software itself is written in C, C++ and Java. It includes a code editor with features such as text cutting and pasting, searching and replacing text, automatic indenting, brace matching, and syntax highlighting, and provides simple *one-click* mechanisms to compile and upload programs to an Arduino board. It also contains a message area, a text console, a toolbar with buttons for common functions and a hierarchy of operation menus.

Figure 1 Arduino Logo

The Arduino IDE supports the languages C and C++ using special rules of code structuring.

C

Figure 2 C Logo

C is a general-purpose computer programming language. It was created in the 1970s by Dennis Ritchie, and remains very widely used and influential. By design, C's features cleanly reflect the capabilities of the targeted CPUs. It has found lasting use in operating systems, device drivers, protocol stacks, though decreasingly for application software. C is commonly used on computer architectures that range from the largest supercomputers to the smallest microcontrollers and embedded systems.

A successor to the programming language B, C was originally developed at Bell Labs by Ritchie between 1972 and 1973 to construct utilities running on Unix. It was applied to re-implementing the kernel of the Unix operating system. During the 1980s, C gradually gained popularity. It has become one of the most widely used programming languages, with C compilers available for almost all modern computer architectures and operating systems. C has been standardized by ANSI since 1989 (ANSI C) and by the International Organization for Standardization (ISO).

C is an imperative procedural language supporting structured programming, lexical variable scope, and recursion, with a static type system. It was designed to be compiled to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant C program written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with few changes to its source code.

Since 2000, C has consistently ranked among the top two languages in the TIOBE index, a measure of the popularity of programming languages.

The traditional "Hello World" program can be written in C as:

```
1.  #include <stdio.h>
2.  int main() {
3.      printf("Hello, World!");
4.      return 0;
5.  }
6.
```

C++

Figure 3 C++ Logo

C++ is a general-purpose programming language created by Danish computer scientist Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Embarcadero, Oracle, and IBM, so it is available on many platforms.

C++ was designed with systems programming and embedded, resource-constrained software and large systems in mind, with performance, efficiency, and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, including desktop applications, video games, servers (e.g. e-commerce, web search, or databases), and performance-critical applications (e.g. telephone switches or space probes).

The traditional "Hello World" program can be written in C++ as:

```
1.  #include <iostream>
2.
3.  int main() {
4.      std::cout << "Hello World!";
5.      return 0;
6.  }
7.
```

ESP32

ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series employs either a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations, Xtensa LX7 dual-core microprocessor or a single-core RISC-V microprocessor and includes built-in antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management modules. ESP32 is created and developed by Espressif Systems, a Shanghai-based Chinese company, and is manufactured by TSMC using their 40 nm process. It is a successor to the ESP8266 microcontroller.

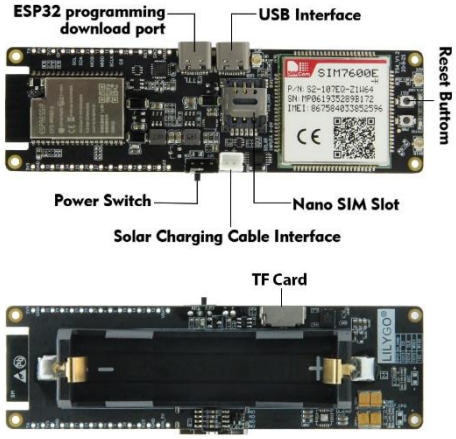
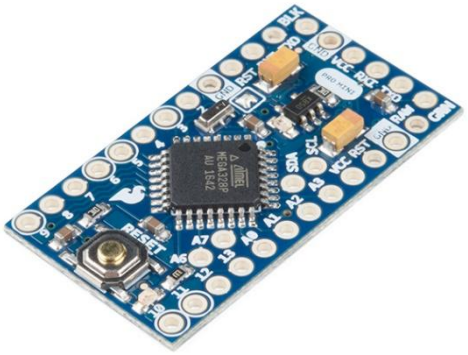


Figure 4 Lilygo TTGO T-SIM7600G

Arduino Pro Mini



This board was developed for applications and installations where space is premium and projects are made as permanent set ups. Small, available in 3.3 V and 5 V versions, powered by ATmega328P. The **Arduino Pro Mini** 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, an on-board resonator, a reset button, and holes for mounting pin headers. A six pin header can be connected to an FTDI cable or Sparkfun breakout board to provide USB power and communication to the board.
The Arduino Pro Mini is intended for semi-permanent installation in objects or exhibitions. The board comes without pre-mounted headers, allowing the use of various types of connectors or direct soldering of wires.The pin layout is compatible with the Arduino Mini.

Figure 5 Arduino Pro Mini Board

Communication

The Arduino Pro Mini has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328P provides UART TTL serial communication, which is available on digital pins 0 (RX) and 1 (TX). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board via a USB connection.
A SoftwareSerial library allows for serial communication on any of the Pro Mini's digital pins. The ATmega328P also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus.

I2C Communications

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.
Like UART communication, I2C only uses two wires to transmit data between devices:

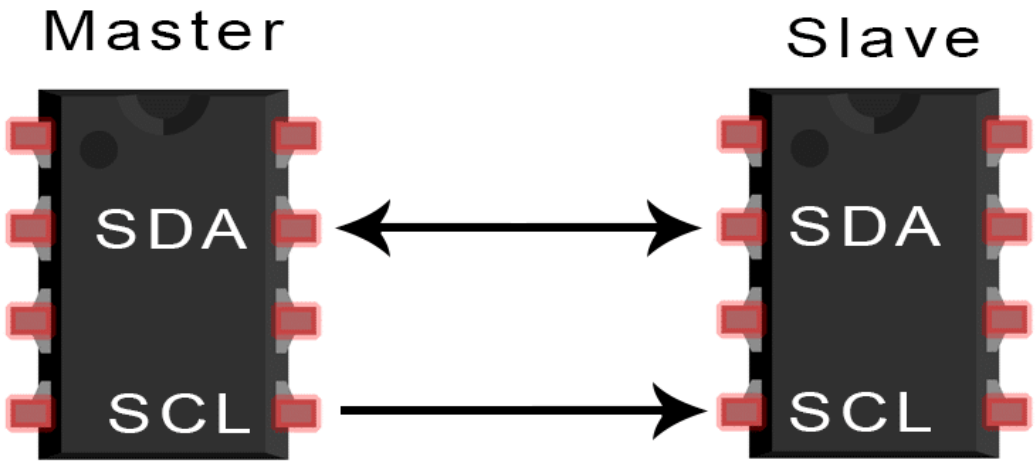


Figure 6 Graphically shown how I2C works

The I2C protocol involves using two lines to send and receive data: a serial clock pin **(SCL)** that the Arduino Controller board pulses at a regular interval, and a serial data pin **(SDA)** over which data is sent between the two devices. As the clock line changes from low to high (known as the rising edge of the clock pulse), a single bit of information - that will form in sequence the address of a specific device and a a command or data - is transferred from the board to the I2C device over the SDA line. When this information is sent - bit after bit -, the called upon device executes the request and transmits it's data back - if required - to the board over the same line using the clock signal still generated by the Controller on SCL as timing.
Because the I2C protocol allows for each enabled device to have it's own unique address, and as both controller and peripheral devices to take turns communicating over a single line, it is possible

for your Arduino board to communicate (in turn) with many devices, or other boards, while using just two pins of your microcontroller.

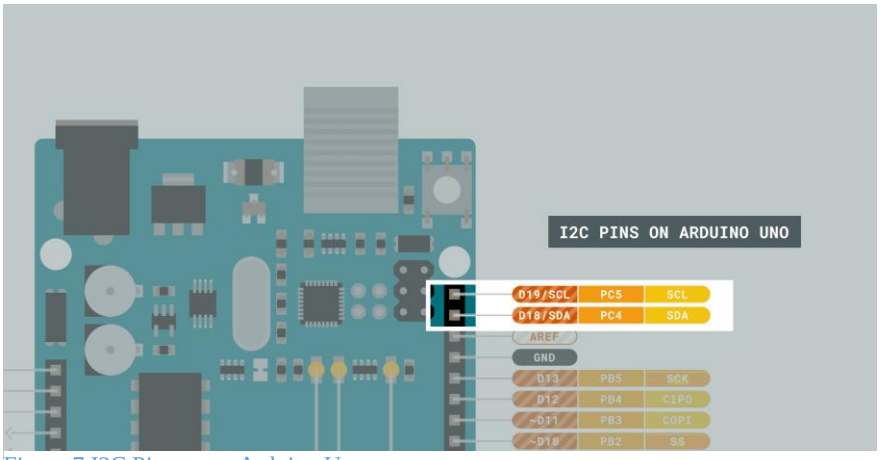


Figure 7 I2C Pins on an Arduino Uno

Mobile Data

Mobile data is the marketing term for wireless Internet access via mobile networks. Access to the network can be made through a portable modem, wireless modem, or a tablet/smartphone (possibly tethered) or other mobile device. The first wireless Internet access became available in 1991 as part of the second generation (2G) of mobile phone technology. Higher speeds became available in 2001 and 2006 as part of the third (3G) and fourth (4G) generations. In 2011, 90% of the world's population lived in areas with 2G coverage, while 45% lived in areas with 2G and 3G coverage.

Generations

Roughly every ten years, new mobile network technology and infrastructure involving a change in the fundamental nature of the service, non-backwards-compatible transmission technology, higher peak data rates, new frequency bands, and/or wider channel frequency bandwidth in Hertz, becomes available. These transitions are referred to as generations. The first mobile data services became available during the second generation (2G).
The generations go as following:

- 2G, from 1991 (GSM CSD, CDPD, GSM GPRS and GSM EDGE)
- 3G, from 2001 (UMTS W-CDMA, UMTS HSPA, UMTS TDD, CDMA2000, GSM EDGE-Evolution)
- 4G, from 2006 (HSPA+, LTE, LTE-Advanced, MBWA)
- 5G, from 2018 (HSPA+, 5G)

The generation my project works on is 4G, or specifically LTE.

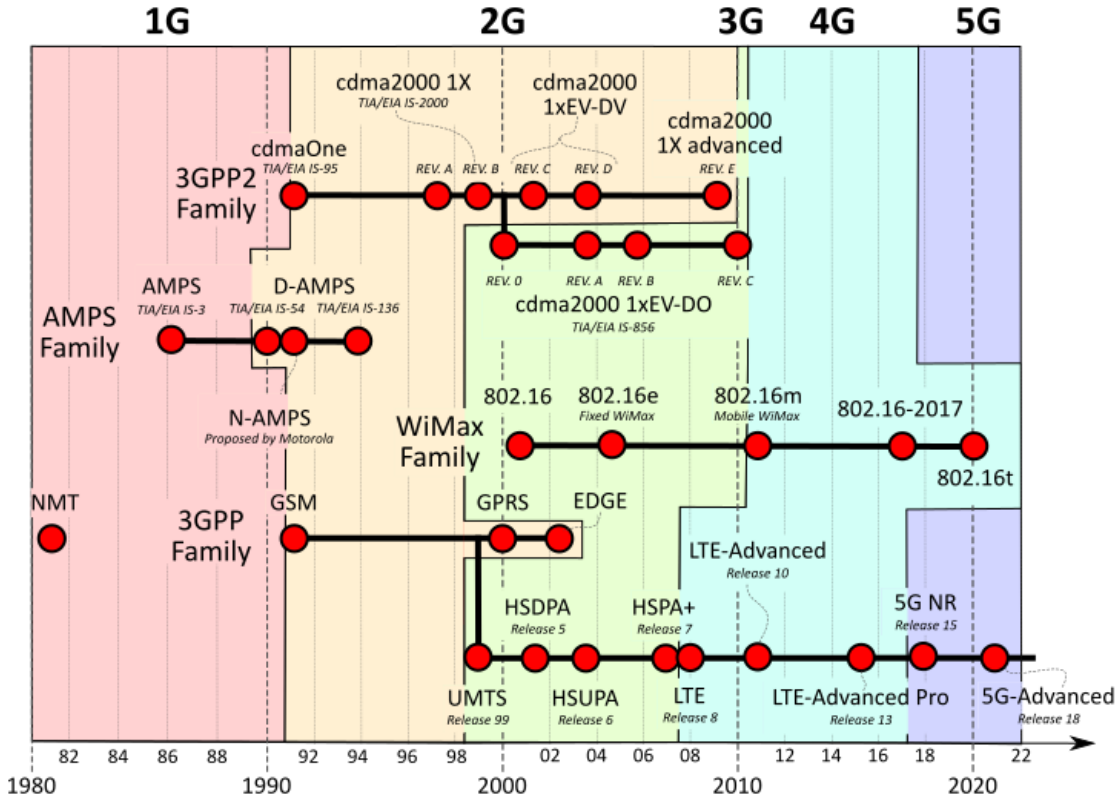


Figure 8 Cellular network standards and generation timeline

4G(LTE)

4G is the fourth generation of broadband cellular network technology, succeeding 3G, and preceding 5G. A 4G system must provide capabilities defined by ITU in IMT Advanced. Potential and current applications include amended mobile web access, IP telephony, gaming services, high-definition mobile TV, video conferencing, and 3D television.

Figure 9 LTE Logo

However, in December 2010 the ITU expanded its definition of 4G to include Long Term Evolution (LTE), Worldwide Interoperability for Microwave Access (WiMAX), and Evolved High Speed Packet Access (HSPA+).

LTE stands for Long-Term Evolution and is a registered trademark owned by ETSI (European Telecommunications Standards Institute) for the wireless data communications technology and a development of the GSM/UMTS standards. However, other nations and companies do play an

active role in the LTE project. The goal of LTE was to increase the capacity and speed of wireless data networks using new DSP (digital signal processing) techniques and modulations that were developed around the turn of the millennium.

LiPo battery

A **lithium polymer battery**, or more correctly **lithium-ion polymer battery** (abbreviated as **LiPo**, **LIP**, **Li-poly**, **lithium-poly** and others), is a rechargeable battery of lithium-ion technology using a polymer electrolyte instead of a liquid electrolyte. High conductivity semisolid (gel) polymers form this electrolyte. These batteries provide higher specific energy than other lithium battery types and are used in applications where weight is a critical feature, such as mobile devices, radio-controlled aircraft and some electric vehicles.

Applications

The vast majority of commercial Li-ion batteries are used in consumer electronics and electric vehicles. Such devices include:

- Portable devices- these include mobile phones and smartphones, laptops and tablets, digital cameras and camcorders, electronic cigarettes, handheld game consoles and torches (flashlights).
- Power tools: Li-ion batteries are used in tools such as cordless drills, sanders, saws, and a variety of garden equipment including whipper-snippers and hedge trimmers.
- Electric vehicles: electric vehicle batteries are used in electric cars, hybrid vehicles, electric motorcycles and scooters, electric bicycles, personal transporters and advanced electric wheelchairs. Also radio-controlled models, model aircraft, aircraft, and the Mars Curiosity rover.

More niche uses include backup power in telecommunications applications. Lithium-ion batteries are also frequently discussed as a potential option for grid energy storage, although they are not yet cost-competitive at scale.

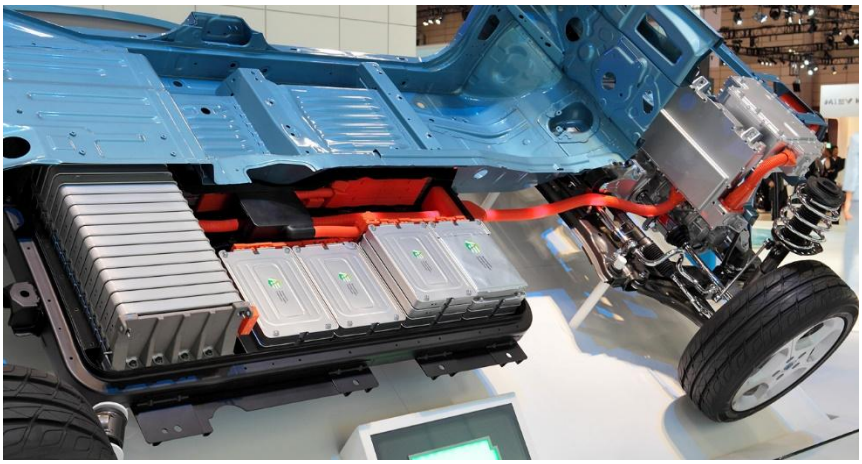


Figure 10 Nissan Leaf's lithium-ion battery pack

Solar Power

Solar power is the conversion of energy from sunlight into electricity, either directly using photovoltaics (PV), indirectly using concentrated solar power, or a combination. Photovoltaic cells convert light into an electric current using the photovoltaic effect[1]. Concentrated solar power systems use lenses or mirrors and solar tracking systems to focus a large area of sunlight to a hot spot, often to drive a steam turbine.

Figure 11 A solar photovoltaic system array on a rooftop

As of 2021, solar generates 4% of the world's electricity, compared to 1% in 2015 when the Paris Agreement to limit climate change was signed. Along with onshore wind, the cheapest levelised cost of electricity is utility-scale solar. The International Energy Agency said in 2021 that under its "Net Zero by 2050" scenario solar power would contribute about 20% of worldwide energy consumption, and solar would be the world's largest source of electricity.

Solar Cell

A solar cell, or photovoltaic cell, is an electronic device that converts the energy of light directly into electricity by the photovoltaic effect, which is a physical and chemical phenomenon. It is a form of photoelectric cell, defined as a device whose electrical characteristics, such as current, voltage, or resistance, vary when exposed to light. Individual solar cell devices are often the electrical building blocks of photovoltaic modules, known colloquially as solar panels.

Figure 12 Solar cell

Solar cells are described as being photovoltaic, irrespective of whether the source is sunlight or an artificial light. In addition to producing energy, they can be used as a photodetector (for example infrared detectors), detecting light or other electromagnetic radiation near the visible range, or measuring light intensity.

The operation of a photovoltaic (PV) cell requires three basic attributes:

- The absorption of light, generating excitons (bound electron-hole pairs), unbound electron-hole pairs (via excitons), or plasmons.
- The separation of charge carriers of opposite types.
- The separate extraction of those carriers to an external circuit.

In contrast, a solar thermal collector supplies heat by absorbing sunlight, for the purpose of either direct heating or indirect electrical power generation from heat. A "photoelectrolytic cell" (photoelectrochemical cell), on the other hand, refers either to a type of photovoltaic cell (like that developed by Edmond Becquerel and modern dye-sensitized solar cells), or to a device that splits water directly into hydrogen and oxygen using only solar illumination.

[1] The photovoltaic effect is the generation of voltage and electric current in a material upon exposure to light. It is a physical and chemical phenomenon.

DHT22 Sensor

The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use but requires careful timing to grab data.
Technical details of the sensor:

- Low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 0-100% humidity readings with 2-5% accuracy
- No more than 0.5Hz sampling rate (once every 2 seconds)
- 4 pins (2.5mm spacing)
- Weight: 2.4g

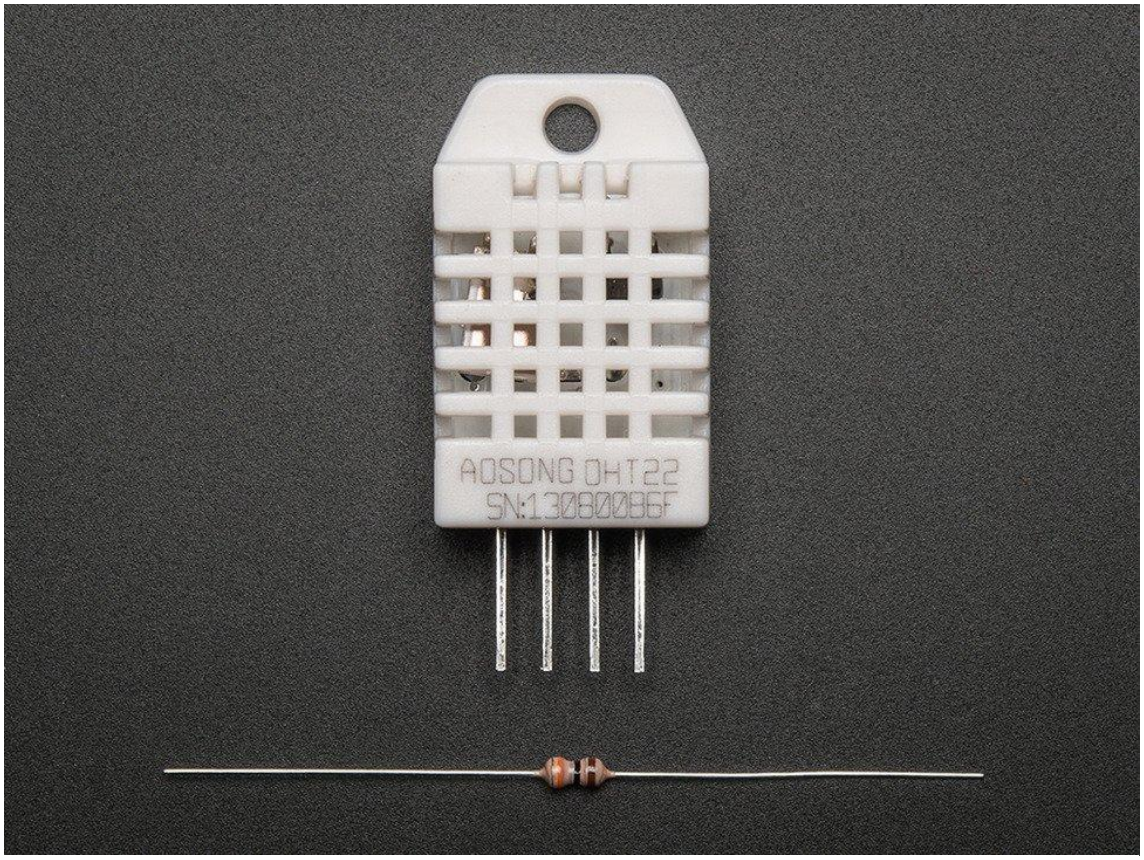


Figure 13 A DHT Sensor with a 4.7K - 10K resistor

SDS 011 Sensor

Figure 14 Nova PM SDS 011 Sensor

The SDS 011 Sensor is a quite recent Air Quality Sensor developed by Nova Fitness, a spin-off from the university of Jinan (in Shandong).
With its size, it is probably one of the best sensor in terms of accuracy: While other sensors tend to focus on shrinking the sensor size, the SDS 011 has opted for a size tradeoff allowing it to use a larger fan. And the larger the fan, the better the quality.
Technical details of the sensor:

- Output: PM2.5, PM10
- Measuring Range: 0.0-999µg/m3
- Input Voltage: 5V
- Maximum Current: 100mA
- Sleep Current: 2mA
- Response Time: 1 second
- Particle Diameter Resolution: ≤0.3µm
- Relative Error: 10%

The SDS011 humidity working range is 0-70%. This is good, since above 70% humidity, the readings become unreliable.

EasyEDA

EasyEDA is a web-based EDA tool suite that enables hardware engineers to design, simulate, share - publicly and privately - and discuss schematics, simulations and printed circuit boards. Other features include the creation of a bill of materials, Gerber files and pick and place files and documentary outputs in PDF, PNG and SVG formats.

EasyEDA allows the creation and editing of schematic diagrams, SPICE simulation of mixed analogue and digital circuits and the creation and editing of printed circuit board layouts and, optionally, the manufacture of printed circuit boards.

Subscription-free membership is offered for public plus a limited number of private projects. The number of private projects can be increased by contributing high quality public projects, schematic symbols, and PCB footprints and/or by paying a monthly subscription.

Figure 15 EasyEDA Logo

Registered users can download Gerber files from the tool free of charge; but for a fee, EasyEDA offers a PCB fabrication service. This service is also able to accept Gerber file inputs from third party tools . I used EasyEDA to design my project schematics.

PHP

PHP is a general-purpose scripting language geared toward web development. It was originally created by Danish-Canadian programmer Rasmus Lerdorf in 1994. The PHP reference implementation is now produced by The PHP Group. PHP originally stood for Personal Home Page,but it now stands for the recursive initialism PHP: Hypertext Preprocessor.

Figure 16 PHP Logo

PHP code is usually processed on a web server by a PHP interpreter implemented as a module, a daemon or as a Common Gateway Interface (CGI) executable. On a web server, the result of the interpreted and executed PHP code – which may be any type of data, such as generated HTML or binary image data – would form the whole or part of an HTTP response. Various web template systems, web content management systems, and web frameworks exist which can be employed to orchestrate or facilitate the generation of that response. Additionally, PHP can be used for many programming tasks outside the web context, such as standalone graphical applications and robotic drone control. PHP code can also be directly executed from the command line.

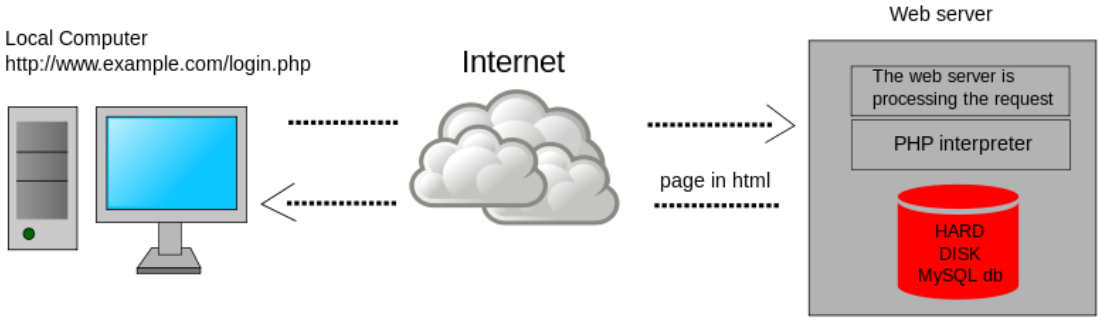


Figure 17 Example of server-side scripting (PHP and MySQL)

The traditional "Hello World" program can be written in PHP as:

```
1.  <!DOCTYPE html>
2.  <html>
3.  <body>

4.  <h1>My first PHP page</h1>

5.  <?php
6.  echo "Hello World!";
7.  ?>

8.  </body>
9.  </html>
```

MySQL

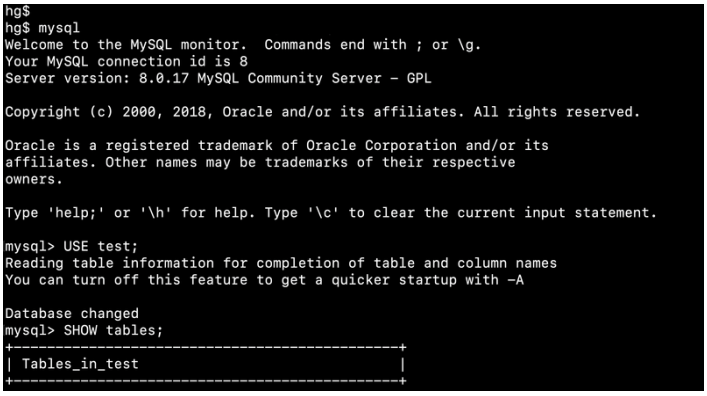


Figure 18 Screenshot of the default MySQL command-line banner and prompt

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter My, and "SQL", the abbreviation for Structured Query Language. A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

MySQL has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often, MySQL is used with other programs to implement applications that need relational database capability. MySQL is a component of the LAMP web application software stack (and others), which is an acronym for Linux, Apache, MySQL, Perl/PHP/Python. MySQL is used by many database-driven web applications, including Drupal, Joomla, phpBB, and WordPress. MySQL is also used by many popular websites, including Facebook, Flickr, MediaWiki, Twitter, and YouTube.

Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.

Android Studio was announced on May 16, 2013, at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0.

On May 7, 2019, Kotlin replaced Java as Google's preferred language for Android app development. Java is still supported, as is C++.

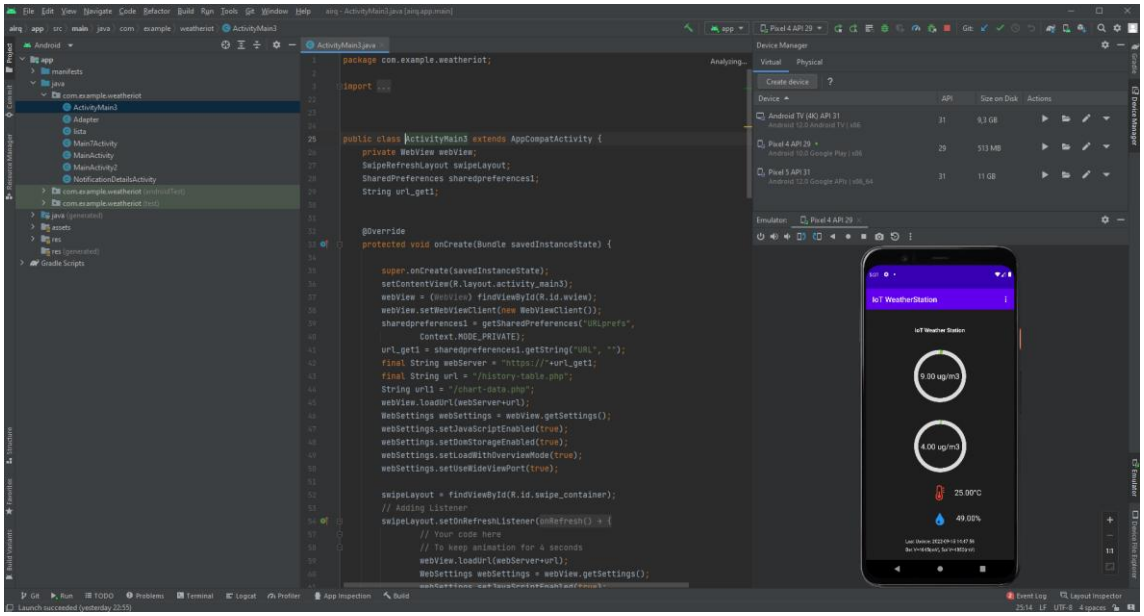


Figure 19 Screenshot of my application in Android Studio, running on an emulator.

Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere (WORA),meaning that compiled Java code can run on all platforms that support Java without the need to recompile. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019, Java was one of the most popular programming languages in use according to GitHub, particularly for client–server web applications, with a reported 9 million developers. The traditional "Hello World" program can be written in Java as:

Figure 20 Java Logo

```
1.  public class HelloWorldApp {
2.      public static void main(String[] args) {
3.          System.out.println("Hello World!"); // Prints the string to the console.
4.      }
5.  }
6.
```

The Java programming language requires the presence of a software platform in order for compiled programs to be executed.

Oracle supplies the Java platform for use with Java. The Android SDK is an alternative software platform, used primarily for developing Android applications with its own GUI system.

Android

The Java language is a key pillar in Android, an open source mobile operating system. Although Android, built on the Linux kernel, is written largely in C, the Android SDK uses the Java language as the basis for Android applications but does not use any of its standard GUI, SE, ME or other established Java standards. The bytecode language supported by the Android SDK is incompatible with Java bytecode and runs on its own virtual machine, optimized for low-memory devices such as smartphones and tablet computers. Depending on the Android version, the bytecode is either interpreted by the Dalvik virtual machine or compiled into native code by the Android Runtime.

Android does not provide the full Java SE standard library, although the Android SDK does include an independent implementation of a large subset of it. It supports Java 6 and some Java 7 features, offering an implementation compatible with the standard library (Apache Harmony).

Problem solving

In this thesis I'm going to discuss about my attempt at building a portable battery and solar powered weather station with an Android application as a front-end. In order to approach the project I did the following:

- Develop the hardware and firmware for the weather station system.
- Design the back-end that communicates between the weather station system and the front-end of the Android application.
- Testing the functionality of communication between the three end points.
- Optimizing the system so it's power efficient.
- Available for future upgrades.

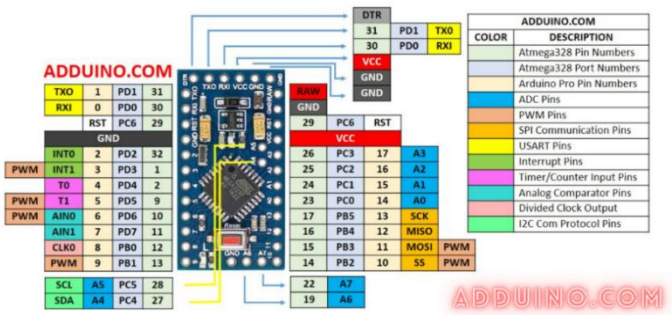
Theoretical approach

Hardware Development

My initial plan was to use the TTGO T-SIM7000G, ESP32 development board and DHT22 and SDS011 sensors. But the development board has only one available GPIO (General Purpose Input Output), because the other ones are reserved for the on board modules as the LTE Modem and the SD Card reader, and their power pins. The other problem that I had is that when the ESP32 is battery powered, it can't supply 5V which is needed for the SDS011 sensor, that needs more than 5V. That's why I used another microcontroller system, which is the Arduino Pro Mini. The Arduino Pro Mini works on 5V and more. The Arduino uses the Atmega 328 microcontroller that is by definition 5V device, on the other hand the ESP32 microcontroller is only 3.3V. To solve that I used a commercially available boost converter MT3608 which steps up the battery voltage to exactly 5.25V which is used to power the Arduino Pro Mini and the sensors connected to the Arduino. To reduce the power consumption, the sensors are switched on when data is being transmitted between them and the Arduino, and also the Arduino is woken up from deep sleep. When the Arduino is awake it transmits the sensor data through I2C communication on command from the ESP32. Then the sensor data is sent by the onboard LTE Modem to my "cloud". Then the data is accessed by the Android application from my "cloud".

Connections on the ESP32 board

So I can understand how the whole system functions, firstly I would like to explain the principle of sensor function which are connected to the Arduino Pro Mini microcontroller system.



Functionality of the DHT22 sensor

Ok now let's see how these sensors actually work. They consist of a humidity sensing component, a NTC temperature sensor (or thermistor) and an IC on the back side of the sensor. For measuring humidity they use the humidity sensing component which has two electrodes with moisture holding substrate between them. So as the humidity changes, the conductivity of the substrate changes or the resistance between these electrodes changes. This change in resistance is measured and processed by the IC which makes it ready to be read by a microcontroller. On the other hand, for measuring temperature these sensors use a NTC temperature sensor or a thermistor.

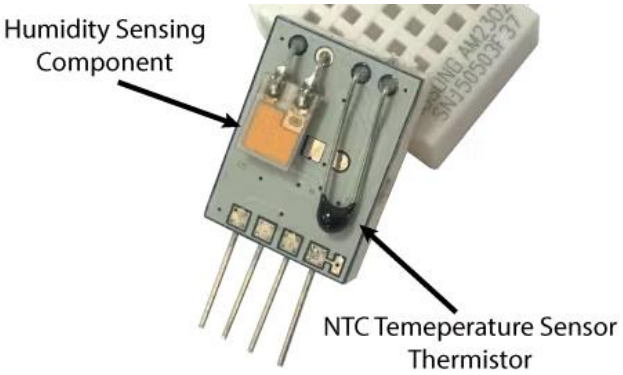


Figure 22 Inside of the sensor

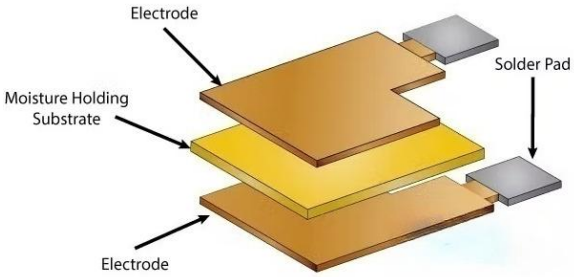


Figure 21 Working principle of humidity sensor

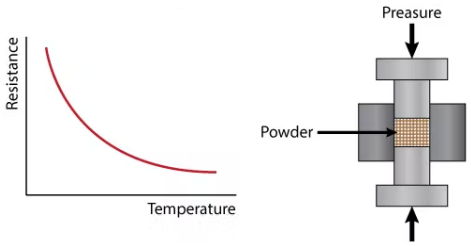
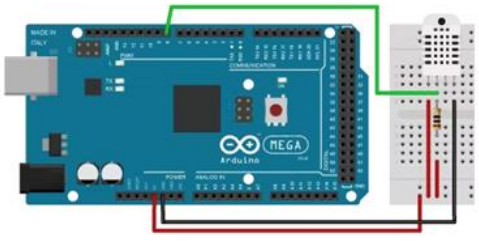


Figure 23 Working principle of NTC Thermistor

A thermistor is actually a variable resistor that changes its resistance with change of the temperature. These sensors are made by sintering of semiconductive materials such as ceramics or polymers in order to provide larger changes in the resistance with just small changes in temperature. The term "NTC" means "Negative Temperature Coefficient", which means that the resistance decreases with increase of the temperature.

The DHTxx sensors have four pins, VCC, GND, data pin and a not connected pin which has no usage. A pull-up resistor from 5K to 10K Ohms is required to keep the data line high and in order to enable the communication between the sensor and the Arduino Board. There are some versions of these sensors that come with a breakout boards with built-in pull-up resistor and they have just 3 pins.



Communication and signal

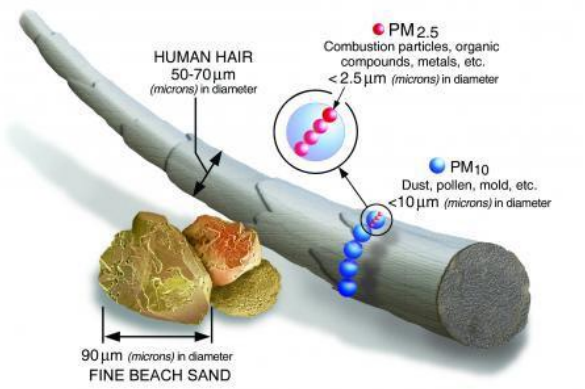
Single-bus data is used for communication between MCU[2] and DHT22, it costs 5mS for single time communication. Data is comprised of integral and decimal part, the following is the formula for data. DHT22 send out higher data bit firstly! DATA=8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data+8 bit check-sum If the data transmission is right, check-sum should be the last 8 bit of "8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data". When MCU send start signal, DHT22 change from low-power-consumption-mode to running-mode. When MCU finishes sending the start signal, DHT22 will send response signal of 40-bit data that reflect the relative humidity and temperature information to MCU. Without start signal from MCU, DHT22 will not give response signal to MCU. One start signal for one time's response data that reflect the relative humidity and temperature information from DHT22. DHT22 will change to low-power-consumption-mode when data collecting finish if it don't receive start signal from MCU again.

The particulate matter (PM), what is, and how does it get into the air?

So, to understand pollution or air contamination, I must study the particles that are related to that, that are also known as particulate matter. Looking at the graphs on the previous section I can observe that they mentioned PM2.5 and PM10. Let's give a quick overview of that.

PM stands for particulate matter (also called particle pollution): the term for a mixture of solid particles and liquid droplets found in the air. Some particles, such as dust, dirt, soot, or smoke, are large or dark enough to be seen with the naked eye. Others are so small they can only be detected using an electron microscope.

Particles come in a wide range of sizes. Particles less than or equal to 10 micrometers in diameter are so small that they can get into the lungs, potentially causing serious health problems. Ten micrometers is less than the width of a single human hair.



beach sand

Particle pollution includes:

- **Coarse dust particles (PM10)**: inhalable particles, with diameters that are generally 10 micrometers and smaller. Sources include crushing or grinding operations and dust stirred up by vehicles on roads.

[2] MCU – Micro Controller Unit

- **Fine particles (PM2.5)** : fine inhalable particles, with diameters that are generally 2.5 micrometers and smaller. Fine particles are produced from all types of combustion, including motor vehicles, power plants, residential wood burning, forest fires, agricultural burning, and some industrial processes.

Why is important care about those particulate matters?

As described by GERARDO ALVARADO Z. in his work at Chile University, studies of episodes of high air pollution in the Meuse Valley (Belgium) in 1930, Donora (Pennsylvania) in 1948 and London in 1952 have been the first documented sources that related mortality with particle contamination (Préndez, 1993). Advances in the investigation of the effects of air pollution on people's health have determined that health risks are caused by inhalable particles, depending on their penetration and deposition in different sections of the respiratory system, and the Biological response to deposited materials.

The thickest particles, about 5 μm, are filtered by the joint action of the cilia of the nasal passage and the mucosa that covers the nasal cavity and the trachea. Particles with a diameter between 0.5 and 5 μm can be deposited in the bronchi and even in the pulmonary alveoli, however, they are eliminated by the cilia of bronchi and bronchioles after a few hours. Particles smaller than 0.5 μm can penetrate deeply until they are deposited in the pulmonary alveoli, remaining from weeks to years, since there is no mucociliary transport mechanism that facilitates elimination.

The Particle Sensor — SDS011

Air Quality monitoring is well known and established science which started back in the 80's. At that time, the technology was quite limited, and the solution used to quantify the air pollution complex, cumbersome and really expensive.

Fortunately, nowadays, with the most recent and modern technologies, the solutions used for Air Quality monitoring are becoming not only more precise but also faster at measuring. Devices are becoming smaller, and cost much more affordable than ever before.

In this article I will focus on a particle sensor, that can detect the amount of dust in the air. While the first generation was just able to detect the amount of opacity, most recent sensors as the SDS011 from INOVAFIT, a spin-off from the University of Jinan (in Shandong), can now detect PM2.5 and PM10.

The SDS011 use the PCB as one side of the casing, allowing to reduce its cost. The receptor diode is mounted on the PCB side (this is mandatory as any noise between the diode and the LNA should be avoided). The emitter laser is mounted on the plastic box and connected to the PCB via a flexible wire.

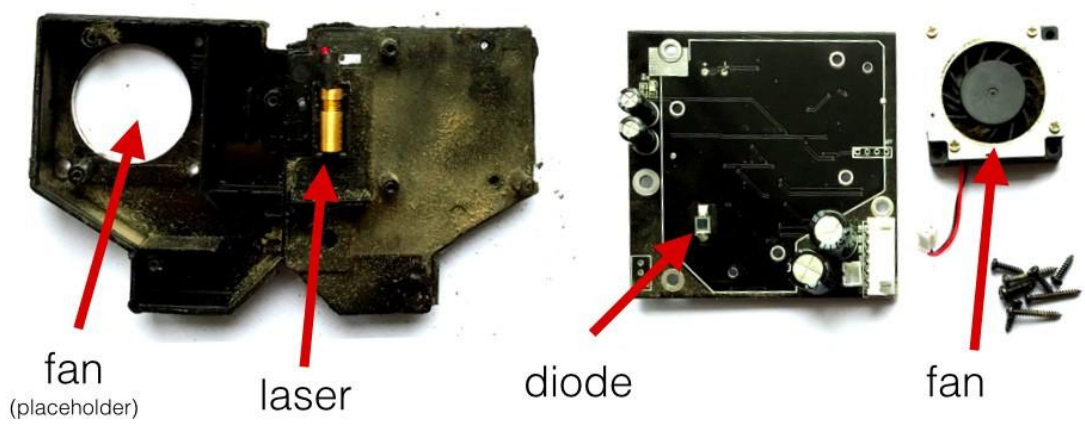


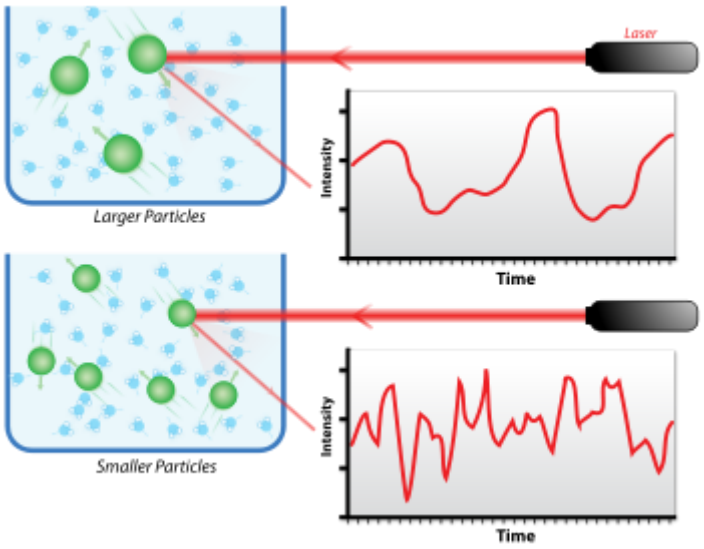
Figure 26 The inside of the SDS011 Sensor

Laser Scattering Principle: Light scattering can be induced when particles go through the detecting area. The scattered light is transformed into electrical signals and these signals will be amplified and processed. The number and diameter of particles can be obtained by analysis because the signal waveform has certain relations with the particles diameter.

But how the SDS011 can capture those particles?

As commented before, the principle used by SDS011 is light scattering or better, Dynamic Light Scattering (DLS), that is a technique in physics that can be used to determine the size distribution profile of small particles in suspension or polymers in solution. In the scope of DLS, temporal fluctuations are usually analyzed by means of the intensity or photon auto-correlation function (also known as photon correlation spectroscopy or quasi-elastic light scattering). In the time domain analysis, the autocorrelation function (ACF) usually decays starting from zero delay time, and faster dynamics due to smaller particles lead to faster decorrelation of scattered intensity trace. It has been shown that the intensity ACF is the Fourier transform of the power spectrum, and therefore the DLS measurements can be equally well performed in the spectral domain.

Below a hypothetical dynamic light scattering of two samples: Larger particles (like PM10) on the top and smaller particles (as PM2.5) on the bottom:



And looking inside my sensor, I can see how the light scatter principle is implemented:

Figure 27 Hypothetical dynamic light scattering

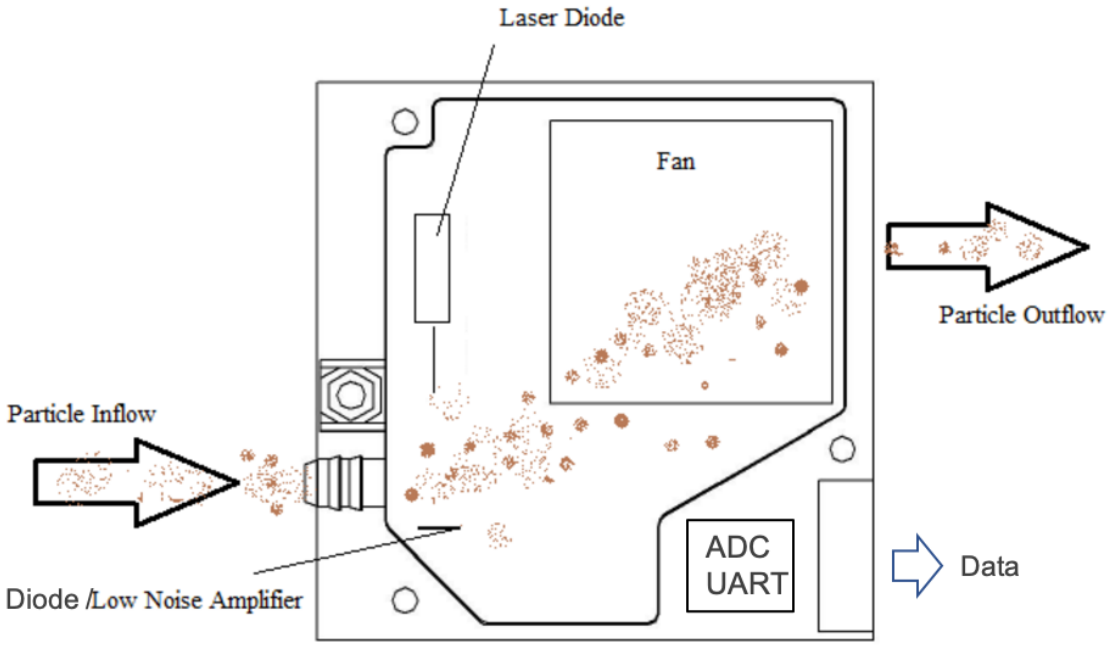


Figure 28 Light Scatter Principle

The electrical signal captured on the diode goes to Low Noise Amplifier and from that to be converted to digital signal thru an ADC and to outside via an UART.

Operating characteristics of the Arduino Pro Mini

It’s good to mention the operating characteristics of the Arduino as it’s very important for the functionality of the sensors. All of the sensors are connected to it as most of the sensors have an operating voltage of 5V only, which the Arduino is by default a 5V microcontroller system. It can be powered with an USB to Serial (FTDI)[3] cable or breakout board connected to its six pin header, or with a regulated 3.3V or 5V supply (depending on the model) on the Vcc pin. There is a voltage regulator on board so it can accept voltage up to 12VDC. If you're supplying unregulated power to the board, be sure to connect to the "RAW" pin on not VCC. The power pins are as follows:

Figure 29 USB to Serial (FTDI) breakout board

- **RAW** For supplying a raw voltage to the board.
- **VCC** The regulated 3.3 or 5 volt supply.
- **GND** Ground pins.

Input and Output

Each of the 14 digital pins on the Pro Mini can be used as an input or output, using pinMode,digitalWrite, and digitalRead functions. They operate at 3.3 or 5 volts (depending on the model). Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

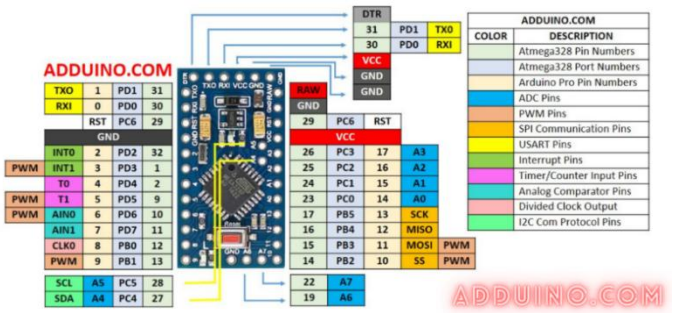


Figure 30 Arduino Pro Mini Pin Diagram

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the TX-0 and RX-1 pins of the six pin header.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt function for details.

[3] FTDI – USB to Serial convertor

- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Pro Mini has 8 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). Four of them are on the headers on the edge of the board; two (inputs 4 and 5) on holes in the interior of the board. The analog inputs measure from ground to VCC. Additionally, some pins have specialized functionality:

- I2C: A4 (SDA) and A5 (SCL). Support I2C (TWI) communication using the Wire library.

There is another pin on the board:

- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Because the ESP32 T-SIM7000G development board when battery powered can only supply maximum of 4.2V on the VBAT pin, and the Arduino needs 5V of operating voltage supplied, I use a boost convertor that boosts the voltage to 5.25V so I can supply unregulated voltage to the RAW voltage pin on the Arduino Pro Mini. The Arduino then regulates the voltage using its on-board voltage regulator, that regulates the RAW input voltage of 5.25V to 5V. Because of that I used a boost (step-up) convertor MT3608.

Figure 31 DC to DC Step Up Boost Converter MT3608 Module Pinout

A boost converter (step-up converter) is a DC-to-DC power converter that steps up voltage (while stepping down current) from its input (supply) to its output (load). It is a class of switched-mode power supply (SMPS) containing at least two semiconductors (a diode and a transistor) and at least one energy storage element: a capacitor, inductor, or the two in combination. To reduce voltage ripple, filters made of capacitors (sometimes in combination with inductors) are normally added to such a converter's output (load-side filter) and input (supply-side filter).

In order to study how it works, I will divide it in two stages. The ON and OFF stages. In the ON part, the switch is closed as I can see in the next figure where the diode is open because the cathode voltage is higher than the anode. The key principle that drives the boost converter is the tendency of an inductor to resist changes in current by creating and destroying a magnetic field. In a boost converter, the output voltage is always higher than the input voltage. When the switch is closed, current flows through the inductor in clockwise direction and the inductor stores some energy by generating a magnetic field. Polarity of the left side of the inductor is positive. So in this case I obtain the current through the inductor using formulas.

When the switch is opened, current will be reduced as the impedance is higher. The magnetic field previously created will be destroyed to maintain the current towards the load. Thus the polarity will be reversed (means left side of inductor will be negative now). As a result, two sources will be in series causing a higher voltage to charge the capacitor through the diode D. In this case the voltage across the inductor is the difference between the output voltage and the input. So once again using the next figure formulas I obtain the current of the OFF part depending on the duty cycle.

Now if I want to obtain the output depending on the input and the duty cycle of the PWM all I have to do is to make the sum of the On and Off current equal to 0. That means that the On current is equal to the Off current.

So I've obtain that the output is depending of the duty cycle disproportionate. So the bigger the Duty cycle gets, the higher will be the output. The duty cycle of the PWM can have values between 0 and 1. So the only possible output will be equal or higher than the input. That's why this configuration is called step up converter.

Because there is a buyable and very cheap boost convertor module, I bought one instead of making one myself. The text above was just for theoretical purposes.

You can notice that the **MT3608 is an IC**, and the module is a circuit built around the IC to make it work as an adjustable converter.

Pinout for the MT3608 module is:

- **IN+** Here I connect the red wire from the battery (or the power source), this is VCC or VIN **(2V - 24V)**
- **IN-** Here I connect the black wire from the battery (or the power source), this is ground, **GND** or V—
- **OUT+** Here I connect the positive voltage of the power distribution circuit or a component powered

- **OUT-** Here I connect the ground of the power distribution circuit or a component powered

To adjust the voltage I have to do a couple of steps.

- **Connect the converter with the battery** or other power source.
- **Set the multimeter to read the voltage** and connect the output of the converter to it. Now you can already see the voltage on the output.
- **Adjust the trimmer** (here 100k Ohm) with a tiny screwdriver until the voltage is set to the desired output.
- **Connect the device/module you want to power** instead of the multimeter.

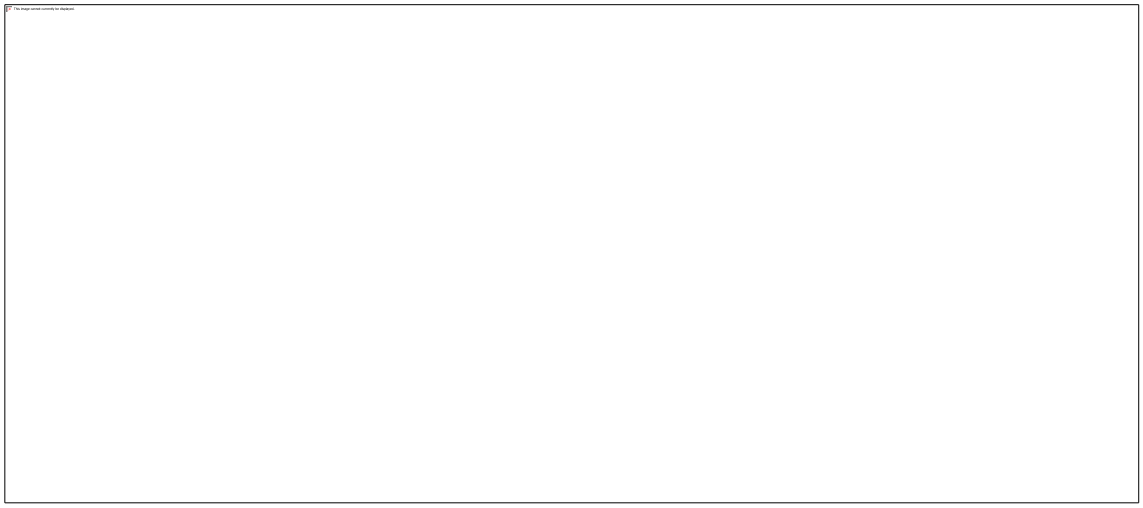


Figure 32 Side view of the MT3608 Boost Converter Module

Using MOSFET as a switch to toggle ON/OFF state of the SDS011 Nova Particulate Matter Sensor for reduced power consumption

To reduce the power consumption I used a MOSFET to turn the SDS011 sensor On and Off, in which way it turns on only when needed, as the whole sensor device consumes a lot of power on when constantly turned on.

The SDS011 sensor requires 70mA at +5V which is over the max current that an Arduino Pro Mini pin can source. If you consult the datasheet for the ATmega328, the max current per pin is 40mA. Going over the maximum current limit can damage the microcontroller.

N-Channel MOSFET

A MOSFET can be used for amplifying or switching signals – in this example, I'll be using it as a switch. It consists of 3 terminals: gate, source, and drain (pinout is below). The N-channel

MOSFET is a voltage-controlled device. There are two types of N-channel MOSFETs: enhancement- and depletion-type. An enhancement type MOSFET is normally off when the gate-source voltage is 0V, so a voltage must be applied to the gate for current to flow through the drain-source channel. A depletion-type MOSFET is normally on when the gate-source voltage is 0V, and thus current flows through the drain-source channel until a positive voltage is applied at the gate.

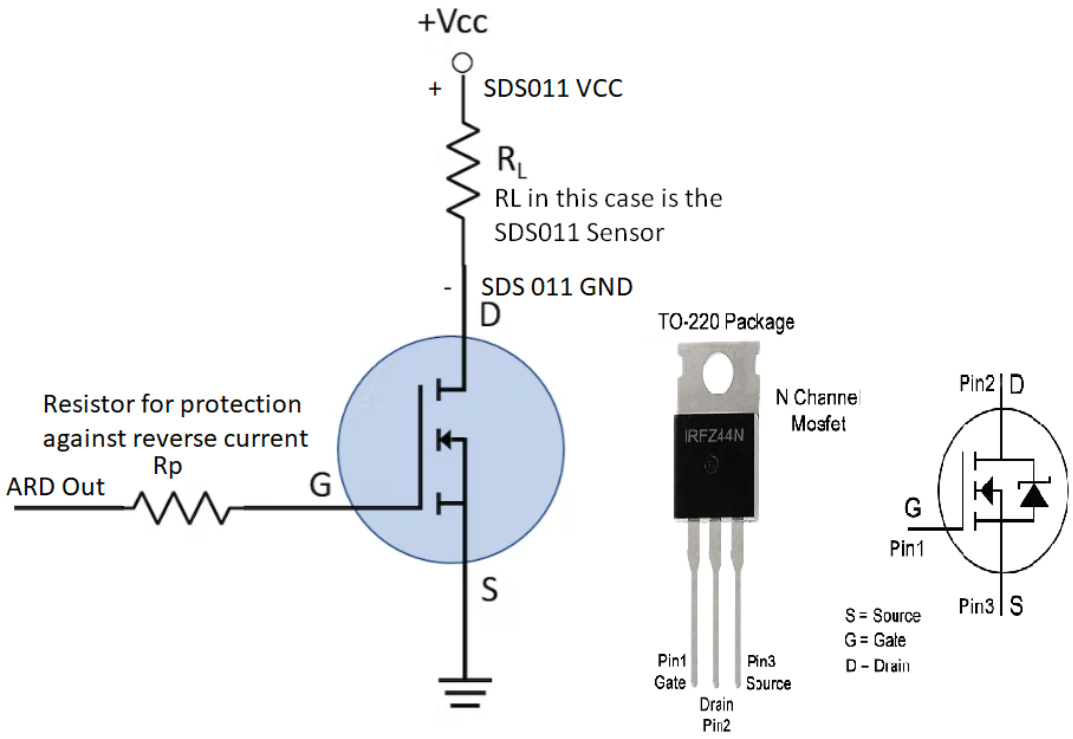


Figure 33 Schematic for connecting the MOSFET to the SDS011 Sensor and the Arduino

Reducing Arduino Power Consumption

When it comes to portable electronics, one of the most important features is how to maximize the battery life. The ATmega328P, used on the Arduino Uno Pro Mini are actually quite power hungry. The Arduino Pro Mini draws around 15mA minimum, which doesn't sound like much but as you'll see in this guide, I can drastically bring that down with just a few tricks. I can reduce the supply current to less than 10uA with a couple hardware and software tricks. Specifically I will be talking about saving power with software.

Inside the ATmega328P, lies a series of circuits that work together to offload work from the processor, and each of these draws some amount of power. The Arduino's analogWrite() function, for example, doesn't have the processor create a PWM signal by counting the clock cycles itself.

Instead, the Arduino uses one of the built in timers to count clock cycles and send an interrupt request to the processor. From there, the processor stops what it's doing and handles the interrupt by switching the pin's state. By offloading some of the work, the microcontroller is able to do multiple things at the same time. Some of the other circuitry built into the ATmega328P include:

- 3 timers
- Watchdog timer
- Brown-out detect
- Analog to digital conversion

Each of these independent components need power to work, and, unless you manually disable them, they will continue to draw power. The brown-out detection actively monitors the system voltage to ensure it doesn't drop below its threshold. If it does, the controller powers down until the voltage is increased above that threshold. The analog to digital converter (ADC) does just as the name suggests, it take the analog voltage (which can be any value from 0V up to VCC) and converts it to a digital value that the microcontroller can use (0-1023 for 10-bit converters). Of a project doesn't need to use the ADC, disabling it will cut down on the power draw drastically.

But what if you still need the ADC? Thankfully there are registers where you can disable some of these circuits with software. Using software allows you to enable the circuits you need, when you need them, and, when you're done, you can disable them again. All of the registers are well documented in the datasheet for the ATmega328p, but, there are libraries out there that are free to use.

The Power down mode (SLEEP_MODE_PWR_DOWN).

When you put your Arduino to sleep it turns off all unnecessary components, reducing the power consumption of the MCU (Microcontroller Unit). In this mode the only way you can wake it up is the use of an external influence (e.g. I give it a nudge to wake up).

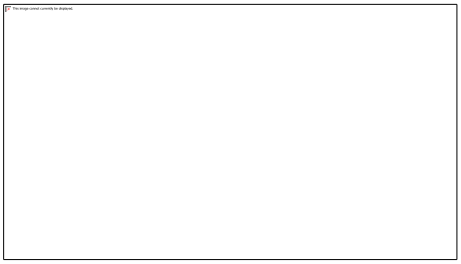


Figure 34 Sleeping Arduino

Before I go into the code to put an Arduino to sleep I need to understand the interrupt concept. The best way to describe it is ; You are working on something you really need to concentrate on. You wear headphones blasting your music loud to drown out your surroundings . You are so concentrated on this that the outside world is lost to you. The only way to get your attention is by giving you a nudge. After you receive this nudge you pay attention to what the interruption is about, and after dealing with it you put the music back on and continue with your task.

Most true Arduino's have a couple of pins that do just that. The Pro Mini in my case has only one interrupt pin to wake the Arduino from deep sleep, and I put the Arduino to sleep by I2C communication by the ESP32 on command.

How I2C works

How is it possible, a communication between so many devices with just to wires? Well each device has a preset ID or a unique device address so the master can choose with which devices will be communicating.

The two wires, or lines are called Serial Clock (or SCL) and Serial Data (or SDA). The SCL line is the clock signal which synchronize the data transfer between the devices on the I2C bus and it's generated by the master device. The other line is the SDA line which carries the data.

The two lines are "open-drain" which means that pull up resistors needs to be attached to them so that the lines are high because the devices on the I2C bus are active low. Commonly used values for the resistors are from 2K for higher speeds at about 400 kbps, to 10K for lower speed at about 100 kbps.

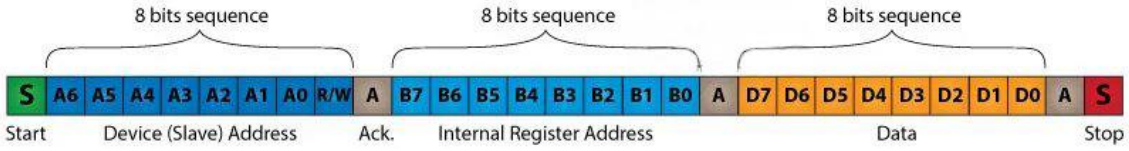


Figure 35 Visual representation of an I2C communication

The data signal is transferred in sequences of 8 bits. So after a special start condition occurs comes the first 8 bits sequence which indicates the address of the slave to which the data is being sent. After each 8 bits sequence follows a bit called Acknowledge. After the first Acknowledge bit in most cases comes another addressing sequence but this time for the internal registers of the slave device. Right after the addressing sequences follows the data sequences as many until the data is completely sent and it ends with a special stop condition.

Let's take even closer look at these events. The start condition occurs when data line drops low while the clock line is still high. After this the clock starts and each data bit is transferred during each clock pulse.

The device addressing sequence stars with the most significant bit (MSB) first and ends with the least significant bit (LSB) and it's actually composed of 7 bits because the 8th bit is used for indicating whether the master will write to the slave (logic low) or read from it (logic high).

The next bit AKC/ NACK is used by the slave device to indicate whether it has successfully received the previous sequence of bits. So at this time the master device hands the control of the SDA line over to the slave device and if the slave device has successfully received the previous

sequence it will pull the SDA line down to the condition called Acknowledge. If the slave does not pull the SDA line down, the condition is called Not Acknowledge, and means that it didn't successfully received the previous sequence which can be caused by several reasons. For example, the slave might be busy, might not understand the received data or command, cannot receive any more data and so on. In such a case the master device decides how it will proceed.

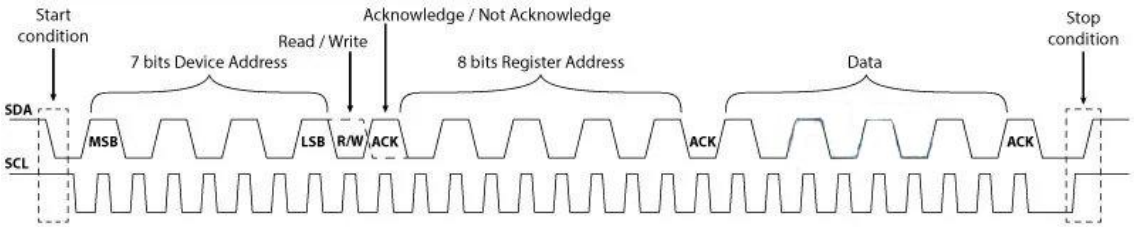


Figure 36 Negotiation between the I2C Slave and Slave, and vice versa

Arduino Wire Library

The Arduino has a built-in library for working with I2C called the Wire Library. It makes it very easy to communicate on the I2C bus, and it can configure the Arduino to become either a master or a slave.

The Wire library has several useful functions for working with I2C.

- begin() – This initiates the library and sets up the Arduino to be either master or slave.
- requestFrom() – This function is used by the master to request data from a slave.
- beginTransmission() – This function is used by the master to send data to a specified slave.
- endTransmission() – This function is used by the master to end a transmission started with the beginTransmission function.
- write() – Used by both master and slave to send data on the I2C bus.
- available() – Used by both master and slave to determine the number of bytes in the data they are receiving.
- read() – Reads a byte of data from the I2C bus.
- SetClock() – Used by the master to set a specific clock frequency.
- onReceive() – Used by the slave to specify a function that is called when data is received from the master.

- onRequest() – Used by the slave to specify a function that is called when the master has requested data.

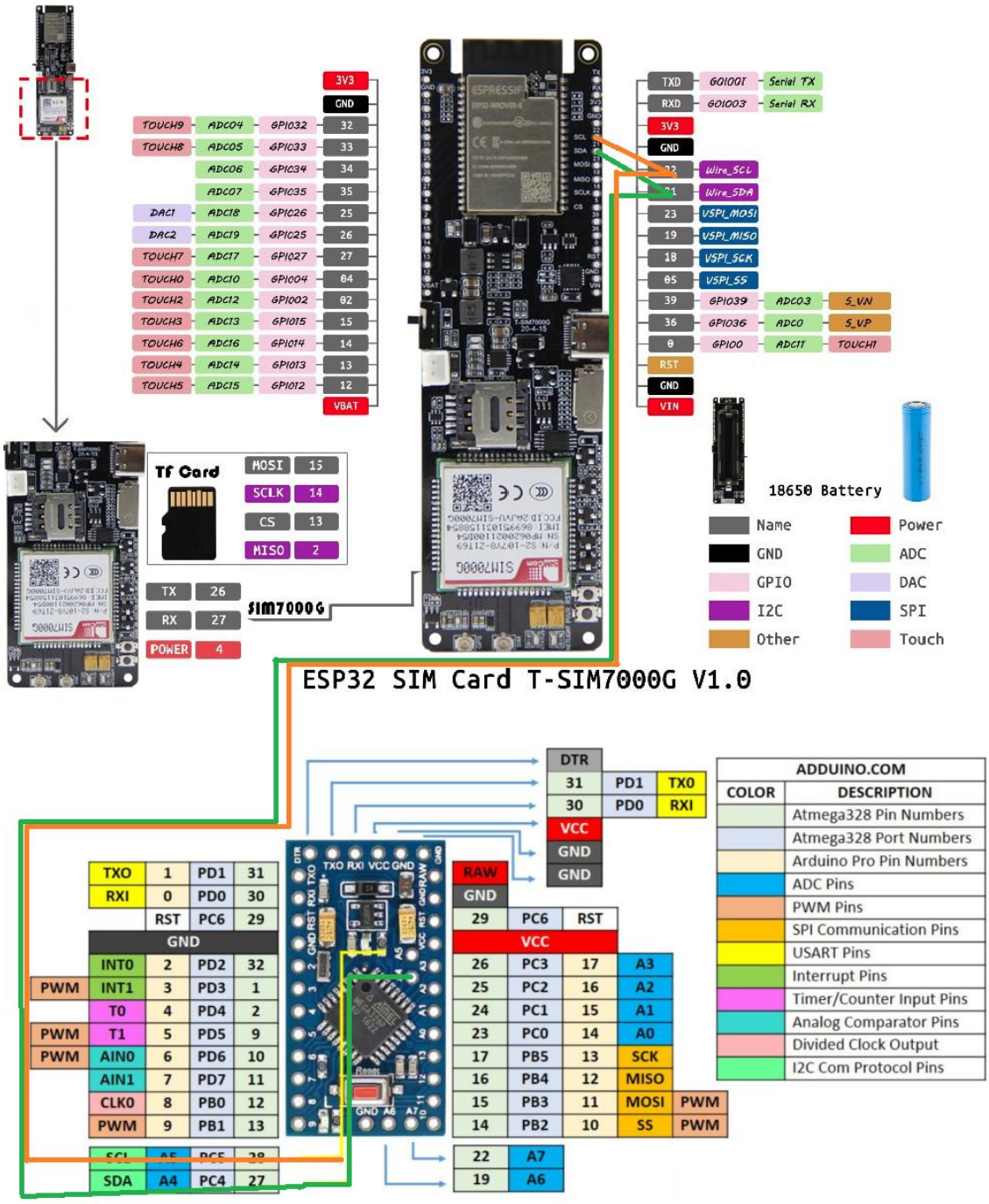


Figure 37 Arduino Pro Mini as an I2C Slave and ESP32 as an I2C Master

ESP32 as an I2C Master, Arduino demo code explanation

```
1.   // Include Arduino Wire library for I2C
2.  #include <Wire.h>
3.
4.  // Define Slave I2C Address
5.  #define SLAVE_ADDR 9
6.
7.  // Define Slave answer size
8.  #define ANSWERSIZE 5
9.
10. void setup() {
11.
12.   // Initialize I2C communications as Master
13.   Wire.begin();
14.
15.   // Setup serial monitor
16.   Serial.begin(9600);
17.   Serial.println("I2C Master Demonstration");
18. }
19.
20. void loop() {
21.   delay(50);
22.   Serial.println("Write data to slave");
23.
24.   // Write a charatre to the Slave
25.   Wire.beginTransmission(SLAVE_ADDR);
26.   Wire.write(0);
27.   Wire.endTransmission();
28.
29.   Serial.println("Receive data");
30.
31.   // Read response from Slave
32.   // Read back 5 characters
33.   Wire.requestFrom(SLAVE_ADDR,ANSWERSIZE);
34.
35.   // Add characters to string
36.   String response = "";
37.   while (Wire.available()) {
38.       char b = Wire.read();
39.       response += b;
40.   }
41.
42.   // Print to Serial Monitor
43.   Serial.println(response);
44. }
```

As with all I2C sketches, I start by including the Wire library.

Next I define a few constants to represent the I2C address of the slave and the number of bytes of data that I expect to retrieve from it.

In the Setup I initialize the I2C communications as a master. I know it is a master as there is no address parameter in the begin function. I also setup a serial monitor and print a line of text to it.

Now to the Loop.

I start with a tiny time delay, mostly to slow things down enough so that I can read the display on the serial monitor.

Next I use the *beginTransmission* function to send data to the slave. In this case the data I send is just a number zero. I finish sending with a call to the *endTransmission* function.

Next I request some data back from the slave using the *requestFrom* function.

After that I formulate a response string by reading the data, a byte at a time, from the slave.

I print the details of what I are doing and of the data I receive to the serial monitor. And then I finish the Loop and do it all over again.

Arduino Pro Mini as an I2C Slave, Arduino demo code explanation

```
1.  // Include Arduino Wire library for I2C
2.  #include <Wire.h>
3.   // Define Slave I2C Address
4.  #define SLAVE_ADDR 9
5.   // Define Slave answer size
6.  #define ANSWERSIZE 5
7.   // Define string with response to Master
8.  String answer = "Hello";
9.
10. void setup() {
11.  // Initialize I2C communications as Slave
12.   Wire.begin(SLAVE_ADDR);
13.   // Function to run when data requested from master
14.   Wire.onRequest(requestEvent);
15.   // Function to run when data received from master
16.   Wire.onReceive(receiveEvent);
17.   // Setup Serial Monitor
18.   Serial.begin(9600);
19.   Serial.println("I2C Slave Demonstration");
20. }
21.
22. void receiveEvent() {
23.   // Read while data received
24.   while (0 < Wire.available()) {
25.     byte x = Wire.read();
26.   }
27.   // Print to Serial Monitor
28.   Serial.println("Receive event");
29. }
30.
31. void requestEvent() {
32.   // Setup byte variable in the correct size
33.   byte response[ANSWERSIZE];
34.   // Format answer as array
35.   for (byte i=0;i<ANSWERSIZE;i++) {
36.     response[i] = (byte)answer.charAt(i);
37.   }
38.   // Send response back to Master
39.   Wire.write(response,sizeof(response));
40.   // Print to Serial Monitor
41.   Serial.println("Request event");
42. }
43.
44. void loop() {
45.  // Time delay in loop
46.   delay(50);
47. }
```

Once again I start by including the Wire library. As with the previous sketch I also define the I2C address for the slave, as well as the number of bytes I are planning to send back to the master.

Next I define the string that I are going to send back to the master, in this case just the word "Hello". If you decide to change this make sure that you adjust the *ANSWERSIZE* constant in both sketches to be correct.

In the Setup I initialize the connection to the I2C bus with a *begin* function. Take note of the different way I do this, as this is a slave I specify the I2C address I are going to be using. By doing this the Wire library knows I want to operate in slave mode.

Now I need to define the names of the functions that I will call when two events occur – a data request received from the master and data received from the master. I also setup and print to the serial monitor.

The function *receiveEvent* is called when I receive data from the master. In this function I read data while the data is available and assign it to a byte (remember, the data will be received one byte at a time).

The *requestEvent* function is called whenever I get a request for data from the master. I need to send my string "Hello" back to the master. As I need to send the data one byte at a time I divide the characters in "Hello" into individual items in an array and then send them one-by-one.

I report all of my progress in both functions to the serial monitor.

The Loop in this sketch just adds a time delay, which matches the one used in the master sketch.

ESP32 Timer Wake Up from Deep Sleep

The ESP32 can go into deep sleep mode, and then wake up at predefined periods of time. This feature is specially useful if you are running projects that require time stamping or daily tasks, while maintaining low power consumption.

The ESP32 RTC controller has a built-in timer you can use to wake up the ESP32 after a predefined amount of time.

Enabling the ESP32 to wake up after a predefined amount of time is very straightforward. In the Arduino IDE, you just have to specify the sleep time in microseconds in the following function:

```
1.  esp_sleep_enable_timer_wakeup(time_in_us)
```

Code

To program the ESP32 I used Arduino IDE, with the ESP32 add-on installed.

The library comes with a predefined deep sleep with timer sketch in the library. To access it, open Arduino IDE, and go to **File** > **Examples** > **ESP32** > **Deep Sleep**, and open the **TimerWakeUp** sketch.

```
1.   /*
2.  Simple Deep Sleep with Timer Wake Up
3.  =====================================
4.  ESP32 offers a deep sleep mode for effective power
5.  saving as power is an important factor for IoT
6.  applications. In this mode CPUs, most of the RAM,
7.  and all the digital peripherals which are clocked
8.  from APB_CLK are powered off. The only parts of
9.  the chip which can still be powered on are:
10. RTC controller, RTC peripherals ,and RTC memories
11.
12. This code displays the most basic deep sleep with
13. a timer to wake it up and how to store data in
14. RTC memory to use it over reboots
15.
16. This code is under Public Domain License.
17.
18. Author:
19. Pranav Cherukupalli <cherukupallip@gmail.com>
20. */
21.
22. #define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds to seconds
    */
23. #define TIME_TO_SLEEP  5        /* Time ESP32 will go to sleep (in seconds) */
24.
25. RTC_DATA_ATTR int bootCount = 0;
26.
27. /*
28. Method to print the reason by which ESP32
29. has been awaken from sleep
30. */
31. void print_wakeup_reason(){
32.   esp_sleep_wakeup_cause_t wakeup_reason;
33.
34.   wakeup_reason = esp_sleep_get_wakeup_cause();
35.
36.   switch(wakeup_reason)
37.   {
38.     case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal
    using RTC_IO"); break;
39.     case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal
    using RTC_CNTL"); break;
40.     case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
41.     case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad");
    break;
42.     case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program");
    break;
43.     default : Serial.printf("Wakeup was not caused by deep sleep:
    %d\n",wakeup_reason); break;
44.   }
45. }
46.
47. void setup(){
48.   Serial.begin(115200);
49.   delay(1000); //Take some time to open up the Serial Monitor
50.
51.   //Increment boot number and print it every reboot
52.   ++bootCount;
53.   Serial.println("Boot number: " + String(bootCount));
54.
55.   //Print the wakeup reason for ESP32
```

```
  print_wakeup_reason();

  /*
  First I configure the wake up source
  I set my ESP32 to wake up every 5 seconds
  */
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
  " Seconds");

  /*
  Next I decide what all peripherals to shut down/keep on
  By default, ESP32 will automatically power down the peripherals
  not needed by the wakeup source, but if you want to be a poweruser
  this is for you. Read in detail at the API docs
  http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep_sleep.html
  Left the line commented as an example of how to configure peripherals.
  The line below turns off all RTC peripherals in deep sleep.
  */
  //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
  //Serial.println("Configured all RTC Peripherals to be powered down in sleep");

  /*
  Now that I have setup a wake cause and if needed setup the
  peripherals state in deep sleep, I can now start going to
  deep sleep.
  In the case that no wake up sources were provided but deep
  sleep was started, it will sleep forever unless hardware
  reset occurs.
  */
  Serial.println("Going to sleep now");
  delay(1000);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){
  //This is not going to be called
}
```

Let's take a look at this code. The first comment describes what is powered off during deep sleep with timer wake up:

> **In this mode CPUs, most of the RAM, and all the digital peripherals which are clocked from APB_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals, and RTC memories.**

When you use timer wake up, the parts that will be powered on are RTC controller, RTC peripherals, and RTC memories.

Define the Sleep Time

These first two lines of code define the period of time the ESP32 will be sleeping.

```
#define uS_TO_S_FACTOR 1000000 /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP 540 /* Time ESP32 will go to sleep (in seconds) */
```

This example uses a conversion factor from microseconds to seconds, so that you can set the sleep time in the TIME_TO_SLEEP variable in seconds. In this case, the example will put the ESP32 into deep sleep mode for 540 seconds or 9 minutes.

Wake Up Reason

Then, the code defines the *print_wakeup_reason()* function, that prints the reason by which the ESP32 has been awaken from sleep.

The setup()

In the setup() is where the code should be located. In deep sleep, the sketch never reaches the loop() statement. So, all the sketch is written in the setup().

Then, the code calls the print_wakeup_reason() function, but you can call any function you want to perform a desired task. For example, you may want to wake up your ESP32 once a day to read a value from a sensor.

Next, the code defines the wake up source by using the following function:

```
1.  esp_sleep_enable_timer_wakeup(time_in_us)
```

This function accepts as argument the time to sleep in microseconds as I've seen previously.

In my case, I have the following:

```
1.  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
```

Then, after all the tasks are performed, the ESP32 goes to sleep by calling the following function:

```
1.  esp_deep_sleep_start()
```

The loop()

The loop() section is empty, because the ESP32 will go to sleep before reaching this part of the code.

Every 540 seconds, the ESP32 wakes up, prints a message on the serial monitor, and goes to deep sleep again.

PNP Transistor as a switch

PNP transistor works same as NPN for a switching operation, but the current flows from the base. This type of switching is used for negative ground configurations. For the PNP transistor, the base terminal is always negatively biased with respect to the emitter.

In this switching, base current flows when the base voltage is more negative. Simply, a low voltage or more negative voltage makes the transistor to short circuit otherwise, it will be open circuit.

In this connection, load is connected to the transistor switching output with a reference point. When the transistor is turned ON, current flows from the source through transistor to the load and finally to the ground.

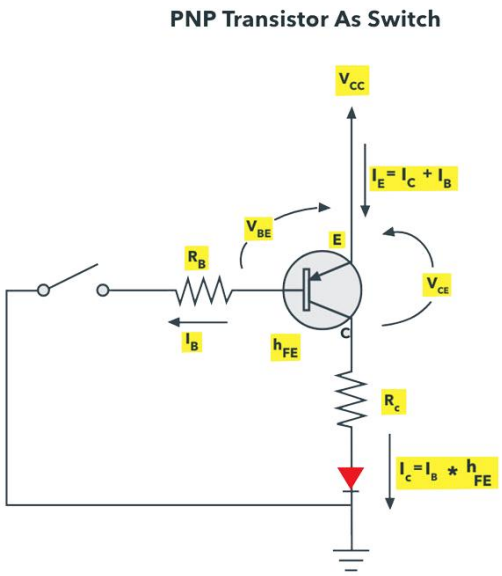


Figure 38 PNP Transistor As Switch

The PNP transistor wakes the Arduino up from deep sleep with 3.3V HIGH logic level shifted to 5V logic level because the Arduino needs 5V logic level

HTTP Client using SIM7000G LTE included on the ESP32 development board.

SIM7000G enables GPRS connectivity to embedded applications. I can implement HTTP Client protocol using SIM7000G HTTP function AT Commands.

The Hypertext Transfer Protocol (HTTP) is a standard application layer protocol which functions as a request response protocol in between server and client.

It is widely used in IoT (Internet of Things) embedded applications, where every sensor is connected to a server and I have access to control them over the internet.

The GSM/GPRS module uses USART communication to communicate with microcontroller or PC terminal. AT commands are used to configure the module in different modes and to perform various functions like calling, posting data to a site, etc.

Sending sensor mock data from a remote server using SIM7000G as a HTTP Client.

```
1.   #include <SoftwareSerial.h>
2.  /* Create object named SIM900 of the class SoftwareSerial */
3.  SoftwareSerial SIM900(8, 7);
4.  void setup() {
5.    SIM900.begin(9600); /* Define baud rate for software serial communication */
6.    Serial.begin(9600); /* Define baud rate for serial communication */
7.  }
8.
9.  void loop() {
10.   Serial.println("HTTP post method :");
11.   Serial.print("AT\\r\\n");
12.   SIM900.println("AT"); /* Check Communication */
13.   delay(5000);
14.   ShowSerialData();   /* Print response on the serial monitor */
15.   delay(5000);
16.   /* Configure bearer profile 1 */
17.   Serial.print("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"\\r\\n");
18.   SIM900.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");       /* Connection type
    GPRS */
19.   delay(5000);
20.   ShowSerialData();
21.   delay(5000);
22.   Serial.print("AT+SAPBR=3,1,\"APN\",\"TATA.DOCOMO.INTERNET\"\\r\\n");
23.   SIM900.println("AT+SAPBR=3,1,\"APN\",\"TATA.DOCOMO.INTERNET\"");     /* APN of
    the provider */
24.   delay(5000);
25.   ShowSerialData();
26.   delay(5000);
27.   Serial.print("AT+SAPBR=1,1\\r\\n");
28.   SIM900.println("AT+SAPBR=1,1");         /* Open GPRS context */
29.   delay(5000);
30.   ShowSerialData();
31.   delay(5000);
32.   Serial.print("AT+SAPBR=2,1\\r\\n");
33.   SIM900.println("AT+SAPBR=2,1");         /* Query the GPRS context */
34.   delay(5000);
35.   ShowSerialData();
36.   delay(5000);
37.   Serial.print("AT+HTTPINIT\\r\\n");
38.   SIM900.println("AT+HTTPINIT");          /* Initialize HTTP service */
39.   delay(5000);
40.   ShowSerialData();
41.   delay(5000);
42.   Serial.print("AT+HTTPPARA=\"CID\",1\\r\\n");
43.   SIM900.println("AT+HTTPPARA=\"CID\",1");         /* Set parameters for HTTP
    session */
44.   delay(5000);
45.   ShowSerialData();
46.   delay(5000);
47.   Serial.print("AT+HTTPPARA=\"URL\",\"api.thingspeak.com/update\"\\r\\n");
48.   SIM900.println("AT+HTTPPARA=\"URL\",\"api.thingspeak.com/update\""); /* Set
    parameters for HTTP session */
49.   delay(5000);
50.   ShowSerialData();
51.   delay(5000);
52.   Serial.print("AT+HTTPDATA=33,10000\\r\\n");
53.   SIM900.println("AT+HTTPDATA=33,10000");/* POST data of size 33 Bytes with
    maximum latency time of 10seconds for inputting the data*/
54.   delay(2000);
55.   ShowSerialData();
```

```
56.   delay(2000);
57.
    Serial.print("api_key=C7JFHZY54GLCJY38temp=35&hum=50&pm25=8&pm10=10&vbat=4900&vsol=
    5500");     /* Data to be sent */
58.
    SIM900.println("api_key=C7JFHZY54GLCJY38temp=35&hum=50&pm25=8&pm10=10&vbat=4900&vso
    l=5500");
59.   delay(5000);
60.   ShowSerialData();
61.   delay(5000);
62.   Serial.print("AT+HTTPACTION=1\\r\\n");
63.   SIM900.println("AT+HTTPACTION=1");      /* Start POST session */
64.   delay(5000);
65.   ShowSerialData();
66.   delay(5000);
67.   Serial.print("AT+HTTPTERM\\r\\n");
68.   SIM900.println("AT+HTTPTERM");          /* Terminate HTTP service */
69.   delay(5000);
70.   ShowSerialData();
71.   delay(5000);
72.   Serial.print("AT+SAPBR=0,1\\r\\n");
73.   SIM900.println("AT+SAPBR=0,1"); /* Close GPRS context */
74.   delay(5000);
75.   ShowSerialData();
76.   delay(5000);
77. }
78.
79. void ShowSerialData()
80. {
81.   while(SIM900.available()!=0)  /* If data is available on serial port */
82.   Serial.write(char (SIM900.read()));     /* Print character received on to the
    serial monitor */
83. }
84.
```

You can see successful transmission of the data field sent to the server highlighted in red box in the image shown above. The +HTTPACTION : 1,200,4 tells us that data has be sent successfully. 200 is the code for success. If the transmission had failed, I would get 600 (instead of 200) or some other code depending on type of failure. For this I used a library for sending the sensor data through the LTE modem, instead of using AT Commands because it's easier to read what I'm sending to the server.

```
1.   // Your GPRS credentials (leave empty, if not needed)
2.  const char apn[]      = "data.lycamobile.mk"; // APN (example:
    internet.vodafone.pt) use https://wiki.apnchanger.org
3.  const char gprsUser[] = "lmmk"; // GPRS User
4.  const char gprsPass[] = "plus"; // GPRS Password
5.
6.  // SIM card PIN (leave empty, if not defined)
7.  const char simPIN[]   = "";
8.
9.  // Server details
10. // The server variable can be just a domain name or it can have a subdomain. It
    depends on the service you are using
11. const char server[] = "daffer.mk"; // domain name: example.com, maker.ifttt.com,
    etc
12. const char resource[] = "/post-data.php";         // resource path, for example:
    /post-data.php
13. const int  port = 80;                            // server port number
14.
```

```
15. // Keep this API Key value to be compatible with the PHP code provided in the
    project page.
16. // If you change the apiKeyValue value, the PHP file /post-data.php also needs to
    have the same key
17. String apiKeyValue = "C7JFHZY54GLCJY38";
18.
19. #define UART_BAUD   9600
20.
21.
22. // Set serial for debug console (to Serial Monitor, default speed 115200)
23. #define SerialMon Serial
24. // Set serial for AT commands (to SIM800 module)
25. #define SerialAT Serial1
26.
27. // Configure TinyGSM library
28. #define TINY_GSM_MODEM_SIM7000
29. #define TINY_GSM_RX_BUFFER 1024 // Set RX buffer to 1Kb
30.
31. // Define the serial console for debug prints, if needed
32. //#define DUMP_AT_COMMANDS
33.
34. #include <TinyGsmClient.h>
35.
36. #ifdef DUMP_AT_COMMANDS
37.   #include <StreamDebugger.h>
38.   StreamDebugger debugger(SerialAT, SerialMon);
39.   TinyGsm modem(debugger);
40. #else
41.   TinyGsm modem(SerialAT);
42. #endif
43.
44. // TinyGSM Client for Internet connection
45. TinyGsmClient client(modem);
46.
47. SerialMon.begin(9600);
48.
49. pinMode(4, OUTPUT);
50.   digitalWrite(4, LOW); //modem powerOn on Low Logic
51.   delay(1000);
52.   digitalWrite(4, HIGH);//modem powerOn oFF HIGH Logic
53.
54.
55. String name = modem.getModemName();
56.   delay(500);
57.   SerialMon.println("Modem Name: " + name);
58.
59.
60.
61.   String modemInfo = modem.getModemInfo();
62.   delay(500);
63.   SerialMon.println("Modem Info: " + modemInfo);
64.
65.
66.   // Set GSM module baud rate and UART pins
67.   SerialAT.begin(UART_BAUD, SERIAL_8N1, 26, 27);
68.   delay(3000);
69.
70.   // Restart SIM800 module, it takes quite some time
71.   // To skip it, call init() instead of restart()
72.   SerialMon.println("Initializing modem...");
73.   modem.restart();
74.   // use modem.init() if you don't need the complete restart
75.
76.   // Unlock your SIM card with a PIN if needed
77.   if (strlen(simPIN) && modem.getSimStatus() != 3 ) {
78.     modem.simUnlock(simPIN);
79.   }
80.
```

```
81.
82. SerialMon.print("Connecting to APN: ");
83.   SerialMon.print(apn);
84.   if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
85.     SerialMon.println(" fail");
86.   }
87.   else {
88.     SerialMon.println(" OK");
89.
90.     SerialMon.print("Connecting to ");
91.     SerialMon.print(server);
92.     if (!client.connect(server, port)) {
93.       SerialMon.println(" fail");
94.     }
95.     else {
96.       SerialMon.println(" OK");
97.
98.       // Making an HTTP POST request
99.       SerialMon.println("Performing HTTP POST request...");
100.
101.             String httpRequestData =
    "api_key=C7JFHZY54GLCJY38temp=35&hum=50&pm25=8&pm10=10&vbat=4900&vsol=5500";
102.
103.  SerialMon.println(httpRequestData);
104.       //String httpRequestData =
    "api_key=C7JFHZY54GLCJY38temp=35&hum=50&pm25=8&pm10=10&vbat=4900&vsol=5500";
105.       client.print(String("POST ") + resource + " HTTP/1.1\r\n");
106.       client.print(String("Host: ") + server + "\r\n");
107.       client.println("Connection: close");
108.       client.println("Content-Type: application/x-www-form-urlencoded");
109.       client.print("Content-Length: ");
110.       client.println(httpRequestData.length());
111.       client.println();
112.       client.println(httpRequestData);
113.
114.       unsigned long timeout = millis();
115.       while (client.connected() && millis() - timeout < 10000L) {
116.         // Print available data (HTTP response from server)
117.         while (client.available()) {
118.           char c = client.read();
119.           SerialMon.print(c);
120.           timeout = millis();
121.         }
122.       }
123.       SerialMon.println();
124.
125.       // Close client and disconnect
126.       client.stop();
127.       SerialMon.println(F("Server disconnected"));
128.       modem.gprsDisconnect();
129.       SerialMon.println(F("GPRS disconnected"));
130.       delay(1000);
131.     }
132.   }
133. }
134.
135.   sendCommand(6, 4);
136.
```

SIM Card with data plan

To use the TTGO T-Call ESP32 SIM800L board, you need a nano SIM card with a data plan. I recommend using a SIM card with a prepaid or monthly plan, so that you know exactly how much you'll spend.

APN Details

To connect your SIM card to the internet, you need to have your phone plan provider APN details. You need the domain name, username and a password.

In my case, I'm using Lycamobile Macedonia.

How the Code Works

Insert GPRS APN credentials in the following variables:

```
const char apn[]      = ""; // APN (example: data.lycamobile.mk) use https://wiki.apnchanger.org
const char gprsUser[] = " lmmk "; // GPRS User
const char gprsPass[] = " plus "; // GPRS Password
```

In my case, the APN is data.lycamobile.mk .

You also need to type the server details in the following variables. It is my own server domain or any other server that you want to publish data to.

```
const char server[] = "daffer.mk"; // domain name: example.com, maker.ifttt.com, etc
const char resource[] = "/post-data.php";   // resource path, for example: /post-data.php
const int port = 80;                        // server port number
```

The apiKeyValue is just a random string that you can modify. It's used for security reasons, so only anyone that knows your API key can publish data to your database.

```
SerialMon.print("Connecting to APN: ");
SerialMon.print(apn);
if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
  SerialMon.println(" fail");
}
else {
  SerialMon.println(" OK");

  SerialMon.print("Connecting to ");
  SerialMon.print(server);
  if (!client.connect(server, port)) {
    SerialMon.println(" fail");
  }
  else {
    SerialMon.println(" OK");
```

Prepare the message data to be sent by HTTP POST Request

```
String httpRequestData = "api_key=" + apiKeyValue + "&value1=" + String(bme.readTemperature())
```

```
2.  + "&value2=" + String(bme.readHumidity()) + "&value3=" +
    String(bme.readPressure()/100.0F) + "";
```

Basically, I create a string with the API key value and all the sensor readings.

The following lines make the POST request.

```
1.  client.print(String("POST ") + resource + " HTTP/1.1\r\n");
2.  client.print(String("Host: ") + server + "\r\n");
3.  client.println("Connection: close");
4.  client.println("Content-Type: application/x-www-form-urlencoded");
5.  client.print("Content-Length: ");
6.  client.println(httpRequestData.length());
7.  client.println();
8.  client.println(httpRequestData);
9.
10. unsigned long timeout = millis();
11. while (client.connected() && millis() - timeout < 10000L) {
12.   // Print available data (HTTP response from server)
13.   while (client.available()) {
14.     char c = client.read();
15.     SerialMon.print(c);
16.     timeout = millis();
17.   }
18. }
```

Finally, close the connection, and disconnect from the internet.

```
1.  client.stop();
2.  SerialMon.println(F("Server disconnected"));
3.  modem.gprsDisconnect();
4.  SerialMon.println(F("GPRS disconnected"));
```

PHP GET/POST request

PHP GET/POST request tutorial shows how to generate and process GET and POST requests in PHP. I use plain PHP and Symfony, Slim, and Laravel frameworks.

HTTP

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP protocol is the foundation of data communication for the World Wide Web.

HTTP GET

The HTTP GET method requests a representation of the specified resource.

GET requests:

- should only be used to request a resource
- parameters are displayed in the URL
- can be cached
- can be bookmarked
- should never be used when dealing with sensitive data
- have length limits

HTTP POST

The HTTP POST method sends data to the server. It is often used when uploading a file or when submitting a completed web form.

POST requests:

- should be used to create a resource
- parameters are not displayed in the URL
- are never cached
- do not remain in the browser history
- cannot be bookmarked
- can be used when dealing with sensitive data
- have no length limits

PHP $_GET and $_POST

PHP provides the $_GET and $_POST superglobals. The $_GET is an associative array of variables passed to the current script via the URL parameters (query string). The $_POST is an associative array of variables passed to the current script via the HTTP POST method when using

application/x-www-form-urlencoded or multipart/form-data as the HTTP Content-Type in the request.

PHP GET request

In the following example, I generate a GET request with curl tool and process the request in plain PHP.

```
1.   <?php
2.
3.  $name = $_GET['name'];
4.
5.  if ($name == null) {
6.      $name = 'guest';
7.  }
8.
9.  $message = $_GET['message'];
10.
11. if ($message == null) {
12.     $message = 'hello there';
13. }
14.
15. echo "$name says: $message";
```

The example retrieves the name and message parameters from the $_GET variable.

```
1.  $ php -S localhost:8000 get_req.php
```

I start the server.

```
1.  $ curl 'localhost:8000/?name=Lucia&message=Cau'
2.  Lucia says: Cau
3.  $ curl 'localhost:8000/?name=Lucia'
4.  Lucia says: hello there
```

I send two GET requests with curl.

PHP POST request

In the following example, I generate a POST request with curl tool and process the request in plain PHP.

```
1.   <?php
2.
3.  $name = $_POST['name'];
4.
5.  if ($name == null) {
6.      $name = 'guest';
```

```
7.  }
8.
9.  $message = $_POST['message'];
10.
11. if ($message == null) {
12.     $message = 'hello there';
13. }
14.
15.
16. echo "$name says: $message";
```

The example retrieves the name and message parameters from the $_POST variable.

```
1.  $ php -S localhost:8000 post_req.php
```

I start the server.

```
1.  $ curl -d "name=Lucia&message=Cau" localhost:8000
2.  Lucia says: Cau
```

I send a POST request with curl.

Android Explanation

Set up a RequestQueue

This text walks you through the explicit steps of creating a RequestQueue, to allow you to supply a custom behavior.
This lesson also describes the recommended practice of creating a RequestQueue as a singleton, which makes the RequestQueue last the lifetime of your app.

Set up a network and cache

```
1.   // Start the queue
2.  requestQueue.start();
3.
4.  String url = "http://www.example.com";
5.
6.  // Formulate the request and handle the response.
7.  StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
8.          new Response.Listener<String>() {
9.      @Override
10.     public void onResponse(String response) {
11.         // Do something with the response
12.     }
13. },
14.     new Response.ErrorListener() {
15.         @Override
16.         public void onErrorResponse(VolleyError error) {
17.             // Handle error
```

```
18.     }
19. });
20.
21. // Add the request to the RequestQueue.
22. requestQueue.add(stringRequest);
```

Make a standard request

This describes how to use the common request types that Volley supports:

- StringRequest. Specify a URL and receive a raw string in response.
- JsonObjectRequest and JsonArrayRequest (both subclasses of JsonRequest). Specify a URL and get a JSON object or array (respectively) in response.

Request JSON

Volley provides the following classes for JSON requests:

- JsonArrayRequest: A request for retrieving a JSONArray response body at a given URL.
- JsonObjectRequest: A request for retrieving a JSONObject response body at a given URL, allowing for an optional JSONObject to be passed in as part of the request body.

Both classes are based on the common base class JsonRequest. You use them following the same basic pattern you use for other types of requests. For example, this snippet fetches a JSON feed and displays it as text in the UI:

```
1.   String url = "http://my-json-feed";
2.
3.  JsonObjectRequest jsonObjectRequest = new JsonObjectRequest
4.          (Request.Method.GET, url, null, new Response.Listener<JSONObject>() {
5.
6.      @Override
7.      public void onResponse(JSONObject response) {
8.          textView.setText("Response: " + response.toString());
9.      }
10. }, new Response.ErrorListener() {
11.
12.     @Override
13.     public void onErrorResponse(VolleyError error) {
14.         // TODO: Handle error
15.
16.     }
17. });
18.
19. // Access the RequestQueue through your singleton class.
20. MySingleton.getInstance(this).addToRequestQueue(jsonObjectRequest);
```

Understanding App Permissions

By default, an Android app starts with zero permissions granted to it. When the app needs to use any of the protected features of the device (sending network requests, accessing the camera, sending an SMS, etc) it must obtain the appropriate permission from the user to do so.

Before Marshmallow, permissions were handled at install-time and specified in the AndroidManifest.xml within the project. Full list of permissions can be found here. After Marshmallow, permissions must now be requested at runtime before being used. There are a number of libraries available to make runtime permissions easier. If you to get started quickly, check out my guide on managing runtime permissions with PermissionsDispatcher.

Permissions were much simpler before Marshmallow (API 23). All permissions were handled at install-time. When a user went to install an app from the Google Play Store, the user was presented a list of permissions that the app required (some people referred to this as a "wall of permissions". The user could either accept **all** the permissions and continue to install the app or decide not to install the app. It was an all or nothing approach. There was no way to grant only certain permissions to the app and no way for the user to revoke certain permissions after the app was installed.

For an app developer, permissions were very simple. To request one of the many permissions, simply specify it in the AndroidManifest.xml:
For example, an application that needs to read the user's contacts would add the following to it's AndroidManifest.xml:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >

    <uses-permission android:name="android.permission.READ_CONTACTS" />
    ...
</manifest>
```

Marshmallow brought large changes to the permissions model. It introduced the concept of runtime permissions. These are permissions that are requested while the app is running (instead of before the app is installed). These permission can then be allowed or denied by the user. For approved permissions, these can also be revoked at a later time.

This means there are a couple more things to consider when working with permissions for a Marshmallow app. Keep in mind that your targetSdkVersion must be >= 23 and your emulator / device must be running Marshmallow to see the new permissions model. If this isn't the case, see the backwards compatibility section to understand how permissions will behave on your configuration.

When you need to add a new permission, first check if the permission is considered a PROTECTION_NORMAL permission. In Marshmallow, Google has designated certain permissions to be "safe" and called these "Normal Permissions". These are things like ACCESS_NETWORK_STATE, INTERNET, etc. which can't do much harm. Normal permissions are automatically granted at install time and **never** prompt the user asking for permission.

Important: Normal Permissions must be added to the AndroidManifest:

```
1.   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2.      package="com.android.app.myapp" >
3.
4.      <uses-permission android:name="android.permission.INTERNET" />
5.      ...
6.  </manifest>
```

If the permission you need to add isn't listed under the normal permissions, you'll need to deal with "Runtime Permissions". Runtime permissions are permissions that are requested as they are needed while the app is running. These permissions will show a dialog to the user, similar to the following one:

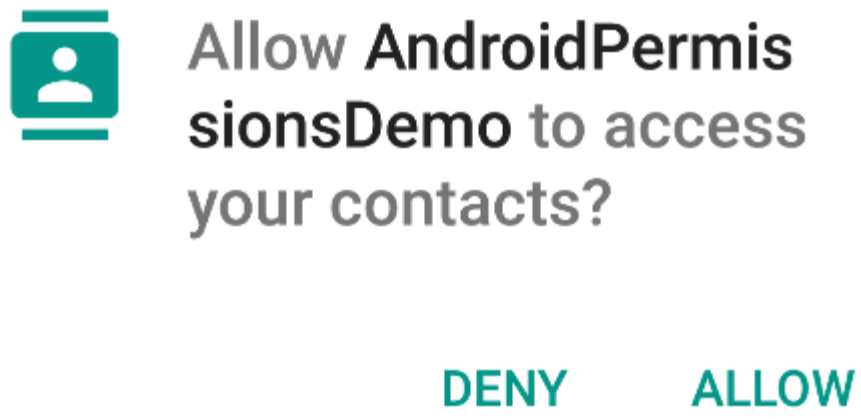


Figure 39 Example of permissions dialog

The first step when adding a "Runtime Permission" is to add it to the AndroidManifest:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.codepath.androidpermissionsdemo" >

   <uses-permission android:name="android.permission.READ_CONTACTS" />
   ...
</manifest>
```

Next, you'll need to initiate the permission request and handle the result. The following code shows how to do this in the context of an Activity, but this is also possible from within a Fragment.

```
// MainActivity.java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        // In an actual app, you'd want to request a permission when the user
performs an action
        // that requires that permission.
        getPermissionToReadUserContacts();
    }

    // Identifier for the permission request
    private static final int READ_CONTACTS_PERMISSIONS_REQUEST = 1;

    // Called when the user is performing an action which requires the app to
read the
    // user's contacts
    public void getPermissionToReadUserContacts() {
        // 1) Use the support library version
ContextCompat.checkSelfPermission(...) to avoid
        // checking the build version since Context.checkSelfPermission(...)
is only available
        // in Marshmallow
        // 2) Always check for permission (even if permission has already
been granted)
        // since the user can revoke permissions at any time through Settings
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_CONTACTS)
                != PackageManager.PERMISSION_GRANTED) {

            // The permission is NOT already granted.
            // Check if the user has been asked about this permission already
and denied
            // it. If so, I want to give more explanation about why the
permission is needed.
            if (shouldShowRequestPermissionRationale(
                    Manifest.permission.READ_CONTACTS)) {
                // Show my own UI to explain to the user why I need to read
the contacts
                // before actually requesting the permission and showing the
default UI
            }

            // Fire off an async request to actually get the permission
            // This will show the standard permission request dialog UI
            requestPermissions(new
String[]{Manifest.permission.READ_CONTACTS},
                    READ_CONTACTS_PERMISSIONS_REQUEST);
        }
    }

    // Callback with the request from calling requestPermissions(...)
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           @NonNull String permissions[],
                                           @NonNull int[] grantResults) {
        // Make sure it's my original READ_CONTACTS request
        if (requestCode == READ_CONTACTS_PERMISSIONS_REQUEST) {
            if (grantResults.length == 1 &&
                    grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Read Contacts permission granted",
Toast.LENGTH_SHORT).show();
```

```
            } else {
                // showRationale = false if user clicks Never Ask Again,
otherwise true
                boolean showRationale = shouldShowRequestPermissionRationale(
this, Manifest.permission.READ_CONTACTS);

                if (showRationale) {
                   // do something here to handle degraded mode
                } else {
                   Toast.makeText(this, "Read Contacts permission denied",
Toast.LENGTH_SHORT).show();
                }
            }
        } else {
            super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
        }
    }
}
```

Notifications in Android

Notifications provide short, timely information about events in your app while it's not in use. This page teaches you how to create a notification with various features for Android 4.0 (API level 14) and higher. For an introduction to how notifications appear on Android, see the Notifications Overview. For sample code that uses notifications, see the People sample.

Notice that the code on this page uses the NotificationCompat APIs from the Android support library. These APIs allow you to add features available only on newer versions of Android while still providing compatibility back to Android 4.0 (API level 14). However, some new features such as the inline reply action result in a no-op on older versions.

Create a Notification

A notification in its most basic and compact form (also known as collapsed form) displays an icon, a title, and a small amount of content text. In this section, you'll learn how to create a notification that the user can click on to launch an activity in your app.

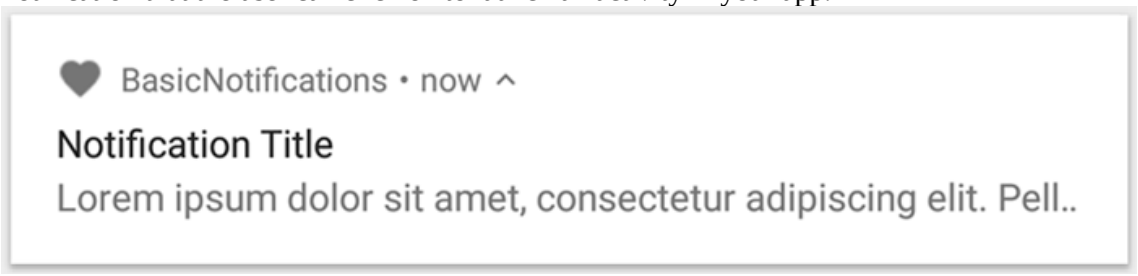


Figure 40 A notification with a title and text

Set the notification content

To get started, you need to set the notification's content and channel using a NotificationCompat.Builder object. The following example shows how to create a notification with the following:

- A small icon, set by setSmallIcon(). This is the only user-visible content that's required.
- A title, set by setContentTitle().
- The body text, set by setContentText().
- The notification priority, set by setPriority(). The priority determines how intrusive the notification should be on Android 7.1 and lower. (For Android 8.0 and higher, you must instead set the channel importance—shown in the next section.)

```
1.  NotificationCompat.Builder builder = new NotificationCompat.Builder(this,
    CHANNEL_ID)
            .setSmallIcon(R.drawable.notification_icon)
            .setContentTitle(textTitle)
            .setContentText(textContent)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Notice that the NotificationCompat.Builder constructor requires that you provide a channel ID. This is required for compatibility with Android 8.0 (API level 26) and higher, but is ignored by older versions.

By default, the notification's text content is truncated to fit one line. If you want your notification to be longer, you can enable an expandable notification by adding a style template with setStyle().

Create a channel and set the importance

Before you can deliver the notification on Android 8.0 and higher, you must register your app's notification channel with the system by passing an instance of NotificationChannel to createNotificationChannel(). Because you must create the notification channel before posting any notifications on Android 8.0 and higher, you should execute this code as soon as your app starts. It's safe to call this repeatedly because creating an existing notification channel performs no operation.

Notice that the NotificationChannel constructor requires an importance, using one of the constants from the NotificationManager class. This parameter determines how to interrupt the user for any notification that belongs to this channel—though you must also set the priority with setPriority() to support Android 7.1 and lower (as shown above).

Although you must set the notification importance/priority as shown here, the system does not guarantee the alert behavior you'll get. In some cases the system might change the importance level based other factors, and the user can always redefine what the importance level is for a given channel.

Android Runnables

In java the interface **Runnable** in basically an abstraction for an executable command.
Runnable, an interface that comes from the Java API, and meant to specify and encapsulate code that is intended to be executed by a Java thread instance or any other class that handles this Runnable.
It’s an interface and is defined in the java.lang package and has an abstract method called run() that I have to implement.

```
1.   package java.lang;
2.  public interface Runnable{
3.      public abstract void run();
4.  }
```

Usage

The Runnable interface is used extensively to execute code in Threads.
The Runnable interface has one abstract method called run(): public abstract void run ()

Normally you implement that method whenever your class implements Runnable interface. The method will then get called to start executing the active part of the class' code.
Let's say I want to use Runnable as my threading mechanism. Well first implement the interface. Then provide the implementation for the run() method:

```
1.   public class MyRunnable implements Runnable {
2.      public void run(){
3.          Log.d("Generic","Running in the Thread " +
4.          Thread.currentThread().getId());
5.      }
6.  }
```

With that, my Runnable subclass can now be passed to Thread and is executed independently in the concurrent line of execution:

```
1.  Thread thread = new Thread(new MyRunnable());
2.  thread.start();
```

In the following code, I basically created the Runnable subclass so that it implements the run() method and can be passed and executed by a thread.
Runnable has alot of them so I list only the ones I used:

Thread - A concurrent unit of execution.This class implements Runnable interface.The Thread class makes use of the Runnable's abstract run() method by calling it's(the Thread's) concrete run() method. That run() method then calls the Runnable's abstract run() method internally.
TimerTask - An abstract class used to describe a task that should run at a specified time.Maybe once,Maybe recurringly.TimerTask implements Runnable interface.The tasks done by the TimerTask do get specified in the run() method.

Android Webview

Android WebView component is a full-fledged browser implemented as a View subclass to embed it into my android application.

Importance Of Android WebView

For HTML code that is limited in terms of scope, I can implement the static method fromHtml() that belongs to the HTML Utility class for parsing HTML-formatted string and displaying it in a TextView. TextView can render simple formatting like styles (bold, italic, etc.), font faces (serif, sans serif, etc.), colors, links, and so forth. However, when it comes to complex formatting and larger scope in terms of HTML, then TextView fails to handle it well. For example browsing Facebook won't be possible through a TextView. In such cases, WebView will be the more appropriate widget, as it can handle a much wider range of

HTML tags. **WebView** can also handle CSS and JavaScript, which Html.fromHtml() would simply ignore. WebView can also assist with common browsing metaphors, such as history list of visited URLs to support backwards and forwards navigation. Still WebView comes with its own set of cons such as it's a much more expensive widget to use, in terms of memory consumption than a TextView. The reason for this increased memory is because WebView is powered by **WebKit/Blink** that are open source Web rendering engine to power content in browsers like Chrome.

Computer Application Description

As this is a prototype, there is no public release of the android application.
When the application is launched this would be the home screen. The first two values are the PM2.5 and PM10 sensor values.
Then next to the thermometer symbol there is the Temperature value, followed by humidity value.
On the bottom the app informs me of the last update that happened between the sensors and the server of which this Android app is pulling the info from.
Afterwads there is the battery voltage value and solar voltage value.

On the top right there is a triple dot extra menu in which I set the sensor backend URL and see the Measured Data History.

The user side of things is made to be very simple and easy to use.

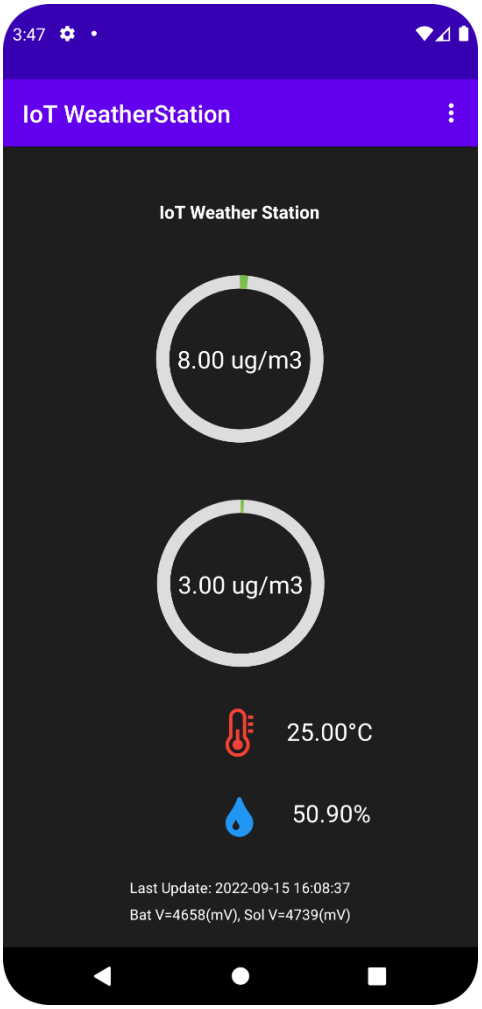


Figure 41 Application Home Screen

Appendices

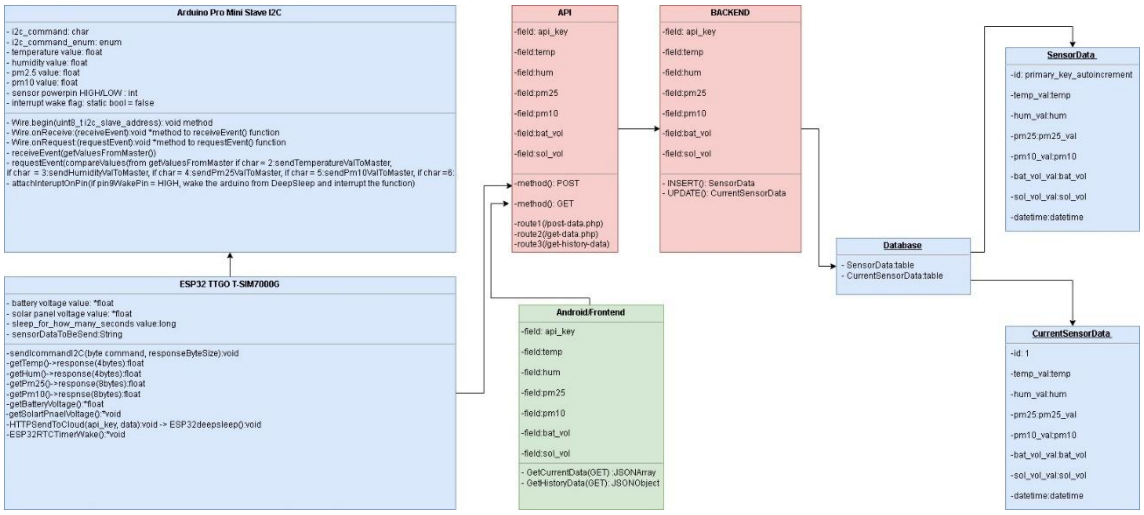


Figure 42 UML Diagram of the whole system

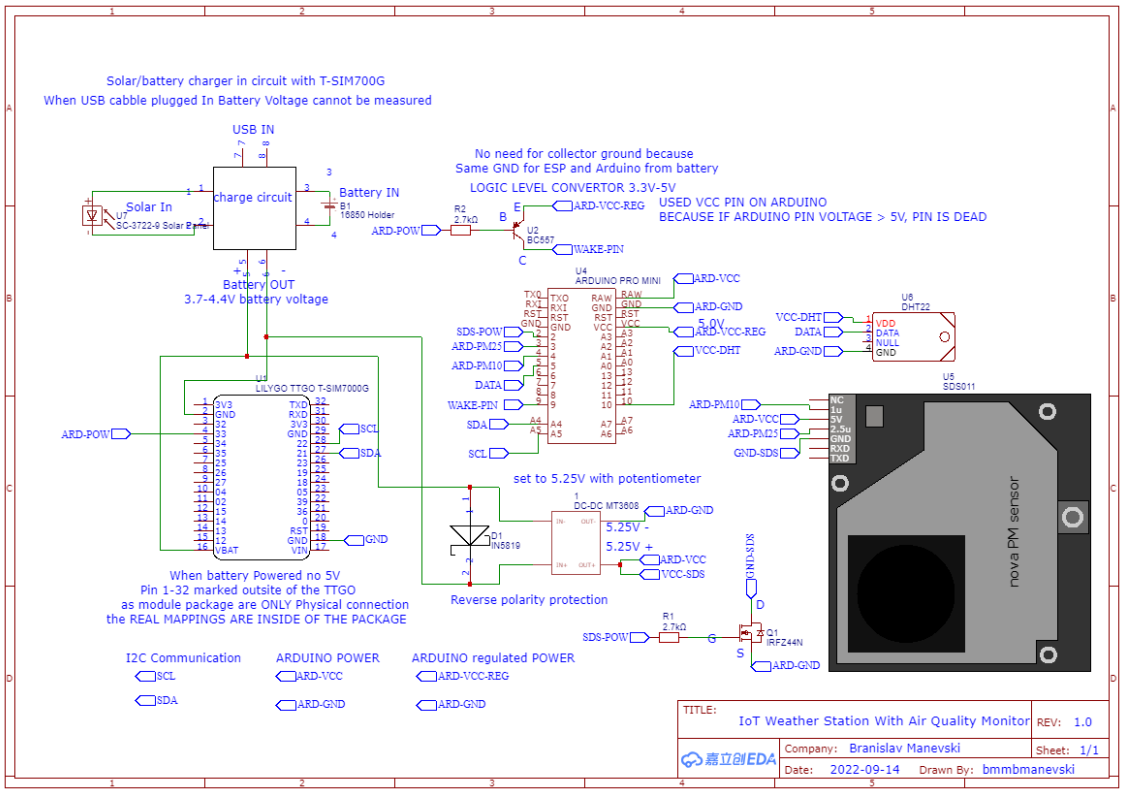


Figure 43 Hardware schematics of the whole System

Arduino Pro Mini Sensor Slave I2C Code

```
1.   #include <Arduino.h>
2.  #include "Wire.h"
3.  #include <DHT.h>
4.  #include <avr/sleep.h>
5.  #include <avr/power.h>
6.
7.  int pin2 = 9;
8.  static bool test = false;
9.
10.
11. /***************************************************
12.  *  Name:        pin2Interrupt
13.  *
14.  *  Returns:     Nothing.
15.  *
16.  *  Parameters:  None.
17.  *
18.  *  Description: Service routine for pin2 interrupt
19.  *
20.  ***************************************************/
21. void pin2Interrupt(void)
22. {
23.   /* This will bring us back from sleep. */
24.
25.   /* I detach the interrupt to stop it from
26.    * continuously firing while the interrupt pin
27.    * is low.
28.    */
29.   detachInterrupt(0);
30. }
31.
32.
33. /***************************************************
34.  *  Name:        enterSleep
35.  *
36.  *  Returns:     Nothing.
37.  *
38.  *  Parameters:  None.
39.  *
40.  *  Description: Enters the arduino into sleep mode.
41.  *
42.  ***************************************************/
43. void enterSleep(void)
44. {
45.
46.   /* Setup pin2 as an interrupt and attach handler. */
47.   attachInterrupt(0, pin2Interrupt, HIGH);
48.   delay(100);
49.
50.   set_sleep_mode(SLEEP_MODE_PWR_DOWN);
51.
52.   sleep_enable();
53.
54.   sleep_mode();
55.
56.   /* The program will continue from here. */
57.
58.   /* First thing to do is disable sleep. */
59.   sleep_disable();
60. }
61.
62.
63.
64. // dht vars
65. const unsigned int dhtpin = 5;
```

```
66. const unsigned int dhtPow = 10;
67. const unsigned int sdsPow = 2;
68. const unsigned int pm25pin = 3;
69. const unsigned int pm10pin = 4;
70.
71. #define DHTTYPE DHT22 // DHT 22 (AM2302)
72. DHT dht(dhtpin, DHTTYPE);
73.
74. unsigned long previousDhtSensorPolling = 0;
75. unsigned long dhtSensorPollingInterval = 2000;
76. float temperature = 0.0;
77. float humidity = 0.0;
78. float pm25 = 0.0;
79. float pm10 = 0.0;
80.
81. // i2c vars
82. const byte SLAVE_ADDR = 0x64;
83. const byte NUM_BYTES = 4;
84. const byte NUM_BYTES_SDS = 8;
85. // command example found on http://www.gammon.com.au/i2c
86. enum {
87.     CMD_ID = 1,
88.     CMD_READ_TEMP  = 2,
89.     CMD_READ_HUMIDITY = 3,
90.     CMD_READ_PM25 = 4,
91.     CMD_READ_PM10 = 5,
92.     CMD_POWER_DOWN = 6
93.   };
94. char command;
95.
96. void requestEvent() {
97.     switch (command)
98.      {
99.       case CMD_ID:
100.          Wire.write (0x64);
101.          break;   // send my ID
102.        case CMD_READ_TEMP: {
103.          byte *tempByte = (byte *)&temperature;
104.          for (byte i=0; i<NUM_BYTES; i++) {
105.              Wire.write(tempByte[i]);
106.          }
107.          break;
108.        }
109.        case CMD_READ_HUMIDITY: {
110.          byte *humidByte = (byte *)&humidity;
111.          for (byte i=0; i<NUM_BYTES; i++) {
112.              Wire.write(humidByte[i]);
113.          }
114.          break;
115.          }
116.        case CMD_READ_PM25: {
117.          byte *pm25Byte = (byte *)&pm25;
118.          for (byte i=0; i<NUM_BYTES_SDS; i++) {
119.              Wire.write(pm25Byte[i]);
120.          }
121.          break;
122.          }
123.        case CMD_READ_PM10: {
124.          byte *pm10Byte = (byte *)&pm10;
125.          for (byte i=0; i<NUM_BYTES_SDS; i++) {
126.              Wire.write(pm10Byte[i]);
127.          }
128.          break;
129.          }
130.          case CMD_POWER_DOWN : {
131.            test = true;
132.          }
133.        default : {
```

```
       }
      }
}



void receiveEvent (uint8_t howMany)
{
  command = Wire.read();

    // remember command for when I get request
} // end of receiveEvent

void setup() {
  Serial.begin(9600);
  pinMode(dhtPow, OUTPUT);
  pinMode(sdsPow, OUTPUT);
  pinMode(pm25pin, INPUT);
  pinMode(pm10pin, INPUT);
  pinMode(pin2, INPUT);

  // setup i2c
  command = 0;
  Wire.begin(SLAVE_ADDR);
  Wire.onReceive (receiveEvent);
  Wire.onRequest(requestEvent);
  // setup dht.begin();
  dht.begin();
}
int seconds=0;
void loop() {
    //if(seconds == 30)

      digitalWrite(dhtPow, HIGH);
      digitalWrite(sdsPow, HIGH);
      delay(1000);
      seconds++;
      Serial.print("Awake for ");
      Serial.print(seconds, DEC);
      Serial.println(" second");
      unsigned long currentMillis = millis();
      if(currentMillis - previousDhtSensorPolling >= dhtSensorPollingInterval) {
        previousDhtSensorPolling = currentMillis;

        temperature = dht.readTemperature();
        humidity = dht.readHumidity();
        float pm10_mom = pulseIn(pm10pin, HIGH, 1500000) / 1000 - 2;
        float pm25_mom = pulseIn(pm25pin, HIGH, 1500000) / 1000 - 2;
        pm10 = pm10_mom;
        pm25 = pm25_mom;
          if(pm10_mom <= 0 || pm25_mom <= 0){
            digitalWrite(sdsPow, LOW);
            return;
          }
          if (isnan(humidity) || isnan(temperature)) {
            digitalWrite(dhtPow, LOW);
            return;
          }
    }
    if(test == true)
    {
      digitalWrite(dhtPow, LOW);
      digitalWrite(sdsPow, LOW);
      test = false;
      Serial.println("Entering sleep");
      delay(200);
      seconds = 0;
```

```
202.         enterSleep();
203.       } else {
204.       }
205.   }
```

ESP32 Sensor Data I2C Master Sender to Cloud

```
1.   // Your GPRS credentials (leave empty, if not needed)
2.  const char apn[]      = "data.lycamobile.mk"; // APN (example:
    internet.vodafone.pt) use https://wiki.apnchanger.org
3.  const char gprsUser[] = "lmmk"; // GPRS User
4.  const char gprsPass[] = "plus"; // GPRS Password
5.
6.  // SIM card PIN (leave empty, if not defined)
7.  const char simPIN[]   = "";
8.
9.  // Server details
10. // The server variable can be just a domain name or it can have a subdomain. It
    depends on the service you are using
11. const char server[] = "daffer.mk"; // domain name: example.com, maker.ifttt.com,
    etc
12. const char resource[] = "/post-data.php";         // resource path, for example:
    /post-data.php
13. const int  port = 80;                             // server port number
14.
15. // Keep this API Key value to be compatible with the PHP code provided in the
    project page.
16. // If you change the apiKeyValue value, the PHP file /post-data.php also needs to
    have the same key
17. String apiKeyValue = "JUlzGXNaj7BJe";
18.
19. // TTGO T-Call pins
20. #define UART_BAUD   9600
21.
22.
23. // Set serial for debug console (to Serial Monitor, default speed 115200)
24. #define SerialMon Serial
25. // Set serial for AT commands (to SIM800 module)
26. #define SerialAT Serial1
27.
28. // Configure TinyGSM library
29. #define TINY_GSM_MODEM_SIM7000
30. #define TINY_GSM_RX_BUFFER 1024 // Set RX buffer to 1Kb
31.
32. // Define the serial console for debug prints, if needed
33. //#define DUMP_AT_COMMANDS
34.
35. #include <Wire.h>
36.
37.
38. // Define Slave I2C Address
39. #define SLAVE_ADDR 0x64
40. #define TEMP_REGISTER 1
41. #define HUM_REGISTER 2
42. #define PM25_REGISTER 3
43. #define PM10_REGISTER 4
44.
45. // Define Slave answer size
46. #define ANSWERSIZE_DHT 8
47. #define ANSWERSIZE_SDS 32
48.
49. #define ARDUINO_POWER_PIN 33
50.
51. #define PIN_ADC_BAT 35
52. #define PIN_ADC_SOLAR 36
```

```
53. #define ADC_BATTERY_LEVEL_SAMPLES 100
54.
55. uint16_t v_bat = 0;
56. uint16_t v_solar = 0;
57.
58.
59.
60.
61. #include <TinyGsmClient.h>
62.
63. #ifdef DUMP_AT_COMMANDS
64.   #include <StreamDebugger.h>
65.   StreamDebugger debugger(SerialAT, SerialMon);
66.   TinyGsm modem(debugger);
67. #else
68.   TinyGsm modem(SerialAT);
69. #endif
70.
71. // TinyGSM Client for Internet connection
72. TinyGsmClient client(modem);
73.
74. #define uS_TO_S_FACTOR 1000000ULL  /* Conversion factor for micro seconds to
    seconds */
75. #define TIME_TO_SLEEP  540        /* Time ESP32 will go to sleep (in seconds) */
76.
77. RTC_DATA_ATTR int bootCount = 0;
78.
79. /*
80. Method to print the reason by which ESP32
81. has been awaken from sleep
82. */
83. void print_wakeup_reason(){
84.   esp_sleep_wakeup_cause_t wakeup_reason;
85.
86.   wakeup_reason = esp_sleep_get_wakeup_cause();
87.
88.   switch(wakeup_reason)
89.   {
90.     case ESP_SLEEP_WAKEUP_EXT0 : SerialMon.println("Wakeup caused by external
    signal using RTC_IO"); break;
91.     case ESP_SLEEP_WAKEUP_EXT1 : SerialMon.println("Wakeup caused by external
    signal using RTC_CNTL"); break;
92.     case ESP_SLEEP_WAKEUP_TIMER : SerialMon.println("Wakeup caused by timer");
    break;
93.     case ESP_SLEEP_WAKEUP_TOUCHPAD : SerialMon.println("Wakeup caused by
    touchpad"); break;
94.     case ESP_SLEEP_WAKEUP_ULP : SerialMon.println("Wakeup caused by ULP program");
    break;
95.     default : Serial.printf("Wakeup was not caused by deep sleep:
    %d\n",wakeup_reason); break;
96.     delay(500);
97.   }
98. }
99.
100.  void read_adc_bat(uint16_t *voltage)
101.  {
102.    uint32_t in = 0;
103.    for (int i = 0; i < ADC_BATTERY_LEVEL_SAMPLES; i++)
104.    {
105.      in += (uint32_t)analogRead(PIN_ADC_BAT);
106.    }
107.    in = (int)in / ADC_BATTERY_LEVEL_SAMPLES;
108.
109.    uint16_t bat_mv = ((float)in / 4096) * 3600 * 2;
110.
111.    *voltage = bat_mv;
112.  }
113.
```

```
114.  void read_adc_solar(uint16_t *voltage)
115.  {
116.    uint32_t in = 0;
117.    for (int i = 0; i < ADC_BATTERY_LEVEL_SAMPLES; i++)
118.    {
119.      in += (uint32_t)analogRead(PIN_ADC_SOLAR);
120.    }
121.    in = (int)in / ADC_BATTERY_LEVEL_SAMPLES;
122.
123.    uint16_t bat_mv = ((float)in / 4096) * 3600 * 2;
124.
125.    *voltage = bat_mv;
126.  }
127.
128.
129.
130.  void sendCommand (const byte cmd, const int responseSize)
131.  {
132.    Wire.beginTransmission(SLAVE_ADDR);
133.    Wire.write(cmd);
134.    Wire.endTransmission();
135.
136.    Wire.requestFrom(SLAVE_ADDR, responseSize);
137.  }  // end of sendCommand
138.
139.  union {
140.   uint8_t b_temp[4];
141.   float f_temp;
142.  } temp_data;
143.
144.  union {
145.   uint8_t b_hum[4];
146.   float f_hum;
147.  } hum_data;
148.
149.  union {
150.   uint8_t b_pm25[8];
151.   float f_pm25;
152.  } pm25_data;
153.
154.  union {
155.   uint8_t b_pm10[8];
156.   float f_pm10;
157.  } pm10_data;
158.
159.  float getTemp() {
160.    sendCommand(2, 4);
161.    while(Wire.available()){
162.      for(uint8_t i = 0; i < 4; i++){
163.        temp_data.b_temp[i] = Wire.read();
164.      }
165.    }
166.    return temp_data.f_temp;
167.  }
168.
169.  float getHum() {
170.    sendCommand(3, 4);
171.    while(Wire.available()){
172.      for(uint8_t i = 0; i < 4; i++){
173.        hum_data.b_hum[i] = Wire.read();
174.      }
175.    }
176.    return hum_data.f_hum;
177.  }
178.
179.  float getPm25() {
180.    sendCommand(4, 8);
181.    while(Wire.available()){
```

```
182.      for(uint8_t i = 0; i < 8; i++){
183.        pm25_data.b_pm25[i] = Wire.read();
184.      }
185.    }
186.    return pm25_data.f_pm25;
187.  }
188.
189.  float getPm10() {
190.    sendCommand(5, 8);
191.    while(Wire.available()){
192.      for(uint8_t i = 0; i < 8; i++){
193.        pm10_data.b_pm10[i] = Wire.read();
194.      }
195.    }
196.    return pm10_data.f_pm10;
197.  }
198.
199.
200.
201.  void setup(){
202.    pinMode(ARDUINO_POWER_PIN, OUTPUT);
203.    digitalWrite(ARDUINO_POWER_PIN, HIGH);
204.    Wire.begin();
205.    SerialMon.begin(9600);
206.    delay(1000); //Take some time to open up the Serial Monitor
207.
208.    //Increment boot number and print it every reboot
209.    ++bootCount;
210.    SerialMon.println("Boot number: " + String(bootCount));
211.
212.    //Print the wakeup reason for ESP32
213.    print_wakeup_reason();
214.
215.    pinMode(4, OUTPUT);
216.    digitalWrite(4, LOW);
217.    delay(1000);
218.    digitalWrite(4, HIGH);
219.
220.
221.
222.    pinMode(PIN_ADC_BAT, INPUT);
223.    pinMode(PIN_ADC_SOLAR, INPUT);
224.
225.
226.
227.    String name = modem.getModemName();
228.    delay(500);
229.    SerialMon.println("Modem Name: " + name);
230.
231.
232.
233.    String modemInfo = modem.getModemInfo();
234.    delay(500);
235.    SerialMon.println("Modem Info: " + modemInfo);
236.
237.
238.    // Set GSM module baud rate and UART pins
239.    SerialAT.begin(UART_BAUD, SERIAL_8N1, 26, 27);
240.    delay(3000);
241.
242.    // Restart SIM800 module, it takes quite some time
243.    // To skip it, call init() instead of restart()
244.    SerialMon.println("Initializing modem...");
245.    modem.restart();
246.    // use modem.init() if you don't need the complete restart
247.
248.    // Unlock your SIM card with a PIN if needed
249.    if (strlen(simPIN) && modem.getSimStatus() != 3 ) {
```

```
250.      modem.simUnlock(simPIN);
251.    }
252.
253.
254.    int   year3    = 0;
255.    int   month3   = 0;
256.    int   day3     = 0;
257.    int   hour3    = 0;
258.    int   min3     = 0;
259.    int   sec3     = 0;
260.    float timezone = 0;
261.    for (int8_t i = 5; i; i--) {
262.      SerialMon.println("Requesting current network time");
263.      if (modem.getNetworkTime(&year3, &month3, &day3, &hour3, &min3, &sec3,
264.                               &timezone)) {
265.          SerialMon.println(hour3);
266.
267.      }
268.    }
269.
270.    /*
271.    First I configure the wake up source
272.    I set my ESP32 to wake up every 5 seconds
273.    */
274.    if(hour3 >= 17){
275.       esp_sleep_enable_timer_wakeup(3600 * uS_TO_S_FACTOR);
276.    }else{
277.        esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
278.    }
279.
280.    SerialMon.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
281.    " Seconds");
282.
283.    read_adc_bat(&v_bat);
284.    read_adc_solar(&v_solar);
285.    SerialMon.println("Get sensor data");
286.    String temp = String(getTemp());
287.    String hum = String(getHum());
288.    String pm25 = String(getPm25());
289.    String pm10 = String(getPm10());
290.    String v_sol_str = String(v_solar);
291.    String v_bat_str = String(v_bat);
292.    SerialMon.println(temp+hum+pm25+pm10+v_sol_str+v_bat_str);
293.
294.    /*
295.    Next I decide what all peripherals to shut down/keep on
296.    By default, ESP32 will automatically power down the peripherals
297.    not needed by the wakeup source, but if you want to be a poweruser
298.    this is for you. Read in detail at the API docs
299.    http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep_sleep.html
300.    Left the line commented as an example of how to configure peripherals.
301.    The line below turns off all RTC peripherals in deep sleep.
302.    */
303.    //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
304.    //Serial.println("Configured all RTC Peripherals to be powered down in sleep");
305.
306.    /*
307.    Now that I have setup a wake cause and if needed setup the
308.    peripherals state in deep sleep, I can now start going to
309.    deep sleep.
310.    In the case that no wake up sources were provided but deep
311.    sleep was started, it will sleep forever unless hardware
312.    reset occurs.
313.    */
314.    SerialMon.print("Connecting to APN: ");
315.    SerialMon.print(apn);
316.    if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
317.      SerialMon.println(" fail");
```

```
318.     }
319.     else {
320.       SerialMon.println(" OK");
321.
322.       SerialMon.print("Connecting to ");
323.       SerialMon.print(server);
324.       if (!client.connect(server, port)) {
325.         SerialMon.println(" fail");
326.       }
327.       else {
328.         SerialMon.println(" OK");
329.
330.         // Making an HTTP POST request
331.         SerialMon.println("Performing HTTP POST request...");
332.         // Prepare your HTTP POST request data (Temperature in Celsius degrees)
333.         // Prepare your HTTP POST request data (Temperature in Fahrenheit degrees)
334.         //String httpRequestData = "api_key=" + apiKeyValue + "&value1=" +
    String(1.8 * bme.readTemperature() + 32)
335.         //                       + "&value2=" + String(bme.readHumidity()) +
    "&value3=" + String(bme.readPressure()/100.0F) + "";
336.
337.         // You can comment the httpRequestData variable above
338.         // then, use the httpRequestData variable below (for testing purposes
    without the BME280 sensor)
339.         //String httpRequestData =
    "api_key=tPmAT5Ab3j7F9&value1=24.75&value2=49.54&value3=1005.14";
340.
341.         String httpRequestData =
    "api_key="+apiKeyValue+"&temp="+temp+"&hum="+hum+"&pm25="+pm25+"&pm10="+pm10+"&vbat
    ="+v_bat_str+"&vsol="+v_sol_str;
342.         SerialMon.println(httpRequestData);
343.         //String httpRequestData =
    "api_key=JUlzGXNaj7BJe&temp=35&hum=50&pm25=8&pm10=10&vbat=4900&vsol=5500";
344.         client.print(String("POST ") + resource + " HTTP/1.1\r\n");
345.         client.print(String("Host: ") + server + "\r\n");
346.         client.println("Connection: close");
347.         client.println("Content-Type: application/x-www-form-urlencoded");
348.         client.print("Content-Length: ");
349.         client.println(httpRequestData.length());
350.         client.println();
351.         client.println(httpRequestData);
352.
353.         unsigned long timeout = millis();
354.         while (client.connected() && millis() - timeout < 10000L) {
355.           // Print available data (HTTP response from server)
356.           while (client.available()) {
357.             char c = client.read();
358.             SerialMon.print(c);
359.             timeout = millis();
360.           }
361.         }
362.         SerialMon.println();
363.
364.         // Close client and disconnect
365.         client.stop();
366.         SerialMon.println(F("Server disconnected"));
367.         modem.gprsDisconnect();
368.         SerialMon.println(F("GPRS disconnected"));
369.         delay(1000);
370.       }
371.     }
372.     sendCommand(6, 4);
373.     digitalWrite(ARDUINO_POWER_PIN,LOW);
374.     delay(5000);
375.     SerialMon.println("Going to sleep now");
376.     SerialMon.flush();
377.     esp_deep_sleep_start();
378.     SerialMon.println("This will never be printed");
```

```
379.  }
380.
381.  void loop(){
382.    //This is not going to be called
383.  }
```

PHP Code

API route for posting data

```
1.   <?php
2.  include '../private_dir/config.php';
3.
4.
5.  // $api_key_value = getKey();
6.
7.  $db_con = new Connection();
8.
9.  $api_key = $temp = $hum = $pm25 = $pm10 = $vbat = $vsol = "";
10.
11. if ($_SERVER["REQUEST_METHOD"] == "POST") {
12.     $api_key = test_input($_POST["api_key"]);
13.     if(strcmp(getKey(),$api_key)==0) {
14.         $temp = test_input($_POST["temp"]);
15.         $hum = test_input($_POST["hum"]);
16.         $pm25 = test_input($_POST["pm25"]);
17.         $pm10 = test_input($_POST["pm10"]);
18.         $vbat = test_input($_POST["vbat"]);
19.         $vsol = test_input($_POST["vsol"]);
20.
21.     // Create connection
22.         $conn = new mysqli($db_con->getHost(), $db_con->getUsername(), $db_con-
    >getPWD(), $db_con->getDBName());
23.
24.         // Check connection
25.         if ($conn->connect_error) {
26.             die("Connection failed: " . $conn->connect_error);
27.         }
28.
29.          $sql = "INSERT INTO SensorData (temp_val, hum_val, pm25_val, pm10_val,
    vbat_val, vsol_val)
30.         VALUES ('" . $temp . "', '" . $hum . "', '" . $pm25 . "', '" . $pm10 . "',
    '" . $vbat . "' , '" . $vsol . "')";
31.
32.         $sql1 = "UPDATE CurrentSensorData SET temp_val = '" . $temp . "', hum_val =
    '" . $hum . "', pm25_val = '" . $pm25 . "', pm10_val = '" . $pm10 . "' , vbat_val =
    '" . $vbat . "' , vsol_val = '" . $vsol . "' WHERE id=1";
33.
34.         if ($conn->query($sql) === TRUE) {
35.             echo "New record created successfully";
36.         }
37.         else {
38.             echo "Error: " . $sql . "<br>" . $conn->error;
39.         }
40.
41.         if ($conn->query($sql1) === TRUE) {
42.             echo "New record created successfully";
43.         }
44.         else {
45.             echo "Error: " . $sql1 . "<br>" . $conn->error;
46.         }
47.
48.         $conn->close();
```

```
49.     }
50.
51.     else {
52.         echo "Wrong API Key provided.";
53.     }
54.
55. }
56. else {
57.     echo "No data posted with HTTP POST.";
58. }
59.
60. function test_input($data) {
61.     $data = trim($data);
62.     $data = stripslashes($data);
63.     $data = htmlspecialchars($data);
64.     return $data;
65. }
66. ?>
```

API route for getting current data

```
1.   <?php
2.  include '../private_dir/config.php';
3.  $db_con = new Connection();
4.  $api_key = $temp = $hum = $pm25 = $pm10 = $vbat = $vsol = "";
5.
6.  if ($_SERVER["REQUEST_METHOD"] == "GET") {
7.          $api_key = test_input($_GET["api_key"]);
8.          if(strcmp(getKey(),$api_key)==0) {
9.          // Create connection
10.         $conn = new mysqli($db_con->getHost(), $db_con->getUsername(), $db_con-
    >getPWD(), $db_con->getDBName());
11.         // Check connection
12.         if ($conn->connect_error) {
13.             die("Connection failed: " . $conn->connect_error);
14.         }
15.
16.         $sql1 = "SELECT * FROM  CurrentSensorData";
17.         $result=mysqli_query($conn,$sql1);
18.
19.         $_ResultSet = array();
20.         while ($row = mysqli_fetch_assoc($result)) {
21.            $_ResultSet[] = $row;
22.         }
23.            header("Content-type: application/json; charset=utf-8");
24.            echo json_encode($_ResultSet);
25.
26.         $conn->close();
27.     }
28. }
29. else {
30.     echo "Error";
31. }
32.
33. function test_input($data) {
34.     $data = trim($data);
35.     $data = stripslashes($data);
36.     $data = htmlspecialchars($data);
37.     return $data;
38. }
39. ?>
```

API route for getting all measured data

```
1.   <?php
2.  include '../private_dir/config.php';
3.  $db_con = new Connection();
4.  $api_key = $temp = $hum = $pm25 = $pm10 = $vbat = $vsol = "";
5.
6.  if ($_SERVER["REQUEST_METHOD"] == "GET") {
7.          $api_key = test_input($_GET["api_key"]);
8.          if(strcmp(getKey(),$api_key)==0) {
9.          // Create connection
10.         $conn = new mysqli($db_con->getHost(), $db_con->getUsername(), $db_con-
    >getPWD(), $db_con->getDBName());
11.         // Check connection
12.         if ($conn->connect_error) {
13.             die("Connection failed: " . $conn->connect_error);
14.         }
15.
16.         $sql1 = "SELECT * FROM SensorData ORDER BY reading_time DESC";
17.         $result=mysqli_query($conn,$sql1);
18.
19.         $_ResultSet = array();
20.         while ($row = mysqli_fetch_assoc($result)) {
21.            $_ResultSet[] = $row;
22.         }
23.            echo json_encode($_ResultSet);
24.
25.         $conn->close();
26.     }
27. }
28. else {
29.     echo "Error";
30. }
31.
32. function test_input($data) {
33.     $data = trim($data);
34.     $data = stripslashes($data);
35.     $data = htmlspecialchars($data);
36.     return $data;
37. }
38. ?>
```

Chart data

```
1.   <?php
2.  include '../private_dir/config.php';
3.  $db_con = new Connection();
4.  $api_key = $temp = $hum = $pm25 = $pm10 = $vbat = $vsol = "";
5.          // Create connection
6.          //getters
7.  $conn = new mysqli($db_con->getHost(), $db_con->getUsername(), $db_con->getPWD(),
    $db_con->getDBName());
8.  // Check connection
9.  if ($conn->connect_error) {
10.     die("Connection failed: " . $conn->connect_error);
11. }
12.
13. $sql = "SELECT id, temp_val, hum_val, pm10_val, pm25_val, vbat_val, vsol_val,
    reading_time FROM SensorData order by reading_time desc limit 40";
14.
15. $result = $conn->query($sql);
16.
17. while ($data = $result->fetch_assoc()){
18.     $sensor_data[] = $data;
19. }
20.
21. $readings_time = array_column($sensor_data, 'reading_time');
22.
23. // ******* Uncomment to convert readings time array to your timezone ********
24. /*$i = 0;
25. foreach ($readings_time as $reading){
26.     // Uncomment to set timezone to - 1 hour (you can change 1 to any number)
27.     $readings_time[$i] = date("Y-m-d H:i:s", strtotime("$reading - 1 hours"));
28.     // Uncomment to set timezone to + 4 hours (you can change 4 to any number)
29.     //$readings_time[$i] = date("Y-m-d H:i:s", strtotime("$reading + 4 hours"));
30.     $i += 1;
31. }*/
32.
33. $value1 = json_encode(array_reverse(array_column($sensor_data, 'temp_val')),
    JSON_NUMERIC_CHECK);
34. $value2 = json_encode(array_reverse(array_column($sensor_data, 'hum_val')),
    JSON_NUMERIC_CHECK);
35. $value3 = json_encode(array_reverse(array_column($sensor_data, 'pm10_val')),
    JSON_NUMERIC_CHECK);
36. $value4 = json_encode(array_reverse(array_column($sensor_data, 'pm25_val')),
    JSON_NUMERIC_CHECK);
37. $value5 = json_encode(array_reverse(array_column($sensor_data, 'vbat_val')),
    JSON_NUMERIC_CHECK);
38. $value6 = json_encode(array_reverse(array_column($sensor_data, 'vsol_val')),
    JSON_NUMERIC_CHECK);
39. $reading_time = json_encode(array_reverse($readings_time), JSON_NUMERIC_CHECK);
40.
41. /*echo $value1;
42. echo $value2;
43. echo $value3;
44. echo $reading_time;*/
45.
46. $result->free();
47. $conn->close();
48. ?>
49.
50. <!DOCTYPE html>
51. <html>
52. <meta name="viewport" content="width=device-width, initial-scale=1">
53.   <script src="https://code.highcharts.com/highcharts.js"></script>
54.   <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
55.   <style>
```

```
    body {
      min-width: 310px;
      max-width: 1280px;
      height: 500px;
      margin: 0 auto;
    }
    h2 {
      font-family: Arial;
      font-size: 2.5rem;
      text-align: center;
    }
  </style>
  <body>
    <h2>Air Quality Monitor Graph</h2>
    <div id="chart">
    <div id="chart-temperature" class="container"></div>
    <div id="chart-humidity" class="container"></div>
    <div id="chart-pm10" class="container"></div>
    <div id="chart-pm25" class="container"></div>
    <div id="chart-vbat" class="container"></div>
    <div id="chart-vsol" class="container"></div>
    </div>
<script>

var value1 = <?php echo $value1; ?>;
var value2 = <?php echo $value2; ?>;
var value3 = <?php echo $value3; ?>;
var value4 = <?php echo $value4; ?>;
var value5 = <?php echo $value5; ?>;
var value6 = <?php echo $value6; ?>;
var reading_time = <?php echo $reading_time; ?>;

var chartT = new Highcharts.Chart({
  chart:{ renderTo : 'chart-temperature' },
  title: { text: 'Temperature' },
  series: [{
    showInLegend: false,
    data: value1
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#059e8a' }
    },
    xAxis: {
      type: 'datetime',
      categories: reading_time
    },
    yAxis: {
      title: { text: 'Temperature (°C)' }
      //title: { text: 'Temperature (Fahrenheit)' }
    },
    credits: { enabled: false }
  });

  var chartH = new Highcharts.Chart({
    chart:{ renderTo:'chart-humidity' },
    title: { text: 'Relative Humidity' },
    series: [{
      showInLegend: false,
      data: value2
    }],
    plotOptions: {
      line: { animation: false,
        dataLabels: { enabled: true }
      }
    },
```

```
    xAxis: {
      type: 'datetime',
      //dateTimeLabelFormats: { second: '%H:%M:%S' },
      categories: reading_time
    },
    yAxis: {
      title: { text: 'Relative Humidity (%)' }
    },
    credits: { enabled: false }
  });

  var chartP = new Highcharts.Chart({
    chart:{ renderTo:'chart-pm10' },
    title: { text: 'PM10' },
    series: [{
      showInLegend: false,
      data: value3
    }],
    plotOptions: {
      line: { animation: false,
        dataLabels: { enabled: true }
      },
      series: { color: '#18009c' }
    },
    xAxis: {
      type: 'datetime',
      categories: reading_time
    },
    yAxis: {
      title: { text: 'PM10 (ug/m3)' }
    },
    credits: { enabled: false }
  });

  var chartQ = new Highcharts.Chart({
    chart:{ renderTo:'chart-pm25' },
    title: { text: 'PM2.5' },
    series: [{
      showInLegend: false,
      data: value4
    }],
    plotOptions: {
      line: { animation: false,
        dataLabels: { enabled: true }
      },
      series: { color: '#18029c' }
    },
    xAxis: {
      type: 'datetime',
      categories: reading_time
    },
    yAxis: {
      title: { text: 'PM2.5 (ug/m3)' }
    },
    credits: { enabled: false }
  });

  var vbat = new Highcharts.Chart({
    chart:{ renderTo:'chart-vbat' },
    title: { text: 'Battery Voltage mV' },
    series: [{
      showInLegend: false,
      data: value5
    }],
    plotOptions: {
      line: { animation: false,
        dataLabels: { enabled: true }
      },
```

```
192.      series: { color: '#ff936a' }
193.    },
194.    xAxis: {
195.      type: 'datetime',
196.      categories: reading_time
197.    },
198.    yAxis: {
199.      title: { text: 'Battery Voltage mV' }
200.    },
201.    credits: { enabled: false }
202.  });
203.
204.  var vsol = new Highcharts.Chart({
205.    chart:{ renderTo:'chart-vsol' },
206.    title: { text: 'Solar Panel Voltage (mV)' },
207.    series: [{
208.      showInLegend: false,
209.      data: value6
210.    }],
211.    plotOptions: {
212.      line: { animation: false,
213.        dataLabels: { enabled: true }
214.      },
215.      series: { color: '#a3ffb4' }
216.    },
217.    xAxis: {
218.      type: 'datetime',
219.      categories: reading_time
220.    },
221.    yAxis: {
222.      title: { text: 'Solar Panel Voltage (mV)' }
223.    },
224.    credits: { enabled: false }
225.  });
226.
227.  </script>
228.
229.  <script>
230.          $(document).ready(function(){
231.              $('body').find('img[src$="https://cdn.000webhost.com/000webhost/logo/
    footer-powered-by-000webhost-
    white2.png"]').parent().closest('a').closest('div').remove();
232.          });
233.  </script>
234.  <script>
235.  function doRefresh(){
236.      $('#chart-temperature').load("#chart-temperature");
237.      $('#chart-humidity').load("#chart-humidity");
238.      $('#chart-pm25').load("#chart-pm25");
239.      $('#chart-pm10').load("#chart-pm10");
240.      $('#chart-vbat').load("#chart-vbat");
241.      $('#chart-vsol').load("#chart-vsol");
242.
243.  }
244.  $(function() {
245.      setInterval(doRefresh, 5000);
246.  });
247.  </script>
248.  </body>
249.  </html>
250.
```

Conclusion

The weather conditions are one of the most important factors affecting human life in several areas. Therefore, knowing the exact weather changes has an effective role in improving and facilitating human life, as well as preserving it. Despite great technological development in monitoring weather conditions through satellites, there remains an obstacle to getting accurate results in a small area, so the ground and water weather stations are a very important component in studying and predicting weather conditions. In this work, I tried to design an electronic system to monitor changes of weather conditions that have a direct impact on agricultural production, transportation of goods, traffic ... etc. in isolated and difficult areas where there are few power lines and the ability to access the Internet. For a start, my project was introduced, which is a weather station integrated with the technology of Internet of things, which is self-feeding through solar panels, and after that I identified the most important types of weather stations known and the features of each one from the other and the most important elements of weather such as temperature, humidity, pressure ... Etc.

My project incorporates multiple spheres of programming, such as microcontroller programming for the Arduino, database management and programming, and .NET implementation for the web server API. As well as hardware soldering and hardware debugging and analysis, as well as IoT communication protocols. And a whole lot of research.

This was something new for me as I have never tried making an IoT project, but always had the interest, and I decided to dive in for the special occasion of graduating.

After choosing how my weather station will be, I compared the microcontrollers available in the market to choose a microcontroller that meets the needs of my project under certain conditions. After that, I chose the sensors and other components that I used in my project, as well as the components of the solar energy circuit and the calculation of the amount of energy consumption of my system and the duration of its operation without recharging the batteries.

Finally, a practical experiment was carried out for all the components of the system, represented by the weather station, the gateway and website, and a comparison of the information sent and received. The results were good, although it is only a prototype.

This work allowed me to test and improve our theoretical and practical knowledge in electronics, electrical engineering and computer science. In addition to that, use several tools such as: EasyEDA, Apache, Programming languages: C / C ++, JavaScript, PHP, SQL ... etc.

Future Work

To Improve my project in the future, I suggest:

- Making a system to monitor large areas by connecting multiple weather stations on one network.
- Using of several types of communication methods
- Using industrial sensors to get better results.
- Using artificial intelligence to add weather predictions.
- Multiple languages support

Bibliography

Marcelo Rovai (29 August, 2019) Sensing the Air Quality. [Online]

United States Environmental Protection Agency (EPA) (18 July, 2022) [Online]

Alex The Giant on learn.sparkfun.com [Online]

Ab Kurk (26 January, 2018) A guide to putting your Arduino to sleep [Online]

Dejan on How To Mechatronics, How I2C Communication Works? Arduino and I2C Tutorial [Online]

Ravi Teja (9 April, 2021) Working of transistor as a Switch [Online]

Multiple contributors (10 May, 2022), Sending HTML Form Data in ASP.NET Web API [Online]

Jan Bodnar (13 July, 2022) PHP Get/Post Requests [Online]

Electronoobs (1 February, 2016) DC to DC Boost Converter [Online]

Robottronic, DC-DC Boost Converter MT3608 [Online]

Ejshea (18 May, 2019), Connecting an N-Channel MOSFET [Online]

[4]

[4] I may have missed some references as there was a lot of research going on.