

Full Wireless Communication Device

Project documentation.

Caio Dutra

May 2023

Contents

Preliminaries	1
i Project overviews	1
ii Project objectives	1
1 Introduction	2
1.1 Hardware Requirements	2
1.2 Firmware Requirements	2
1.3 Operation Flow	3
1.4 Limitations and Future Improvements	3
2 Circuitry	4
2.1 Block Diagram	4
2.2 Schematics	5
2.3 Circuit Diagrams	7
2.3.1 Digital and Analog Supplies Circuitry	7
2.3.2 RC Delay Reset Circuitry	7
2.3.3 12Kohm RES	8
2.3.4 RF Antenna Circuitry	8
2.3.5 XTAL Circuitry	8
2.3.6 Drive Current Serial Clock RES	9
2.3.7 Power Circuitry	10
2.3.8 Storage Circuitry	11
2.3.9 Temperature Sensor Circuitry	12
3 Printed Circuit Board	14
3.1 PCB Design	14
3.2 PCB Specifications	14
3.2.1 ESP8266EX Power Supply Design	14
3.2.2 Layers Design	14
3.2.3 Reset Track Design	15
3.2.4 XTAL - Crystal Oscillator Design	15
3.2.5 RF Antenna	15
4 Firmware	17
4.1 Firmware Description	17
4.2 Configurations and Parameters	17
4.3 Firmware Architecture	21
4.4 Features and Algorithms	23
4.5 Interfaces and Protocols	23
References	24

Preliminaries

i Project overviews

The purpose of this project was to develop a firmware solution as part of a job interview for the position of Firmware Engineer at Tractian. The project involved designing and implementing hardware and firmware for data transmissions through a wireless protocol. For this task, the WiFi protocol was chosen. Although it wasn't necessary, it was added to the project the capability of data collection, so there was actual information to be sent later. The device was equipped with a temperature sensor to capture data, which was then stored in a flash EEPROM memory.

ii Project objectives

The main objective of this project is to demonstrate the capabilities of a battery-powered device by successfully transmitting a 500kB file across a challenging 100-meter open-air gap. This task requires efficient wireless communication and optimized power management to ensure reliable and uninterrupted data transmission. By accomplishing this objective, the project aims to showcase the device's ability to overcome distance limitations and deliver data wirelessly, even in challenging environments.

1 Introduction

This project aimed on the full development of a wireless communicating capable device, from scratch all the way up to the final design. Throughout the development process, careful consideration was given to hardware requirements and specifications. The firmware was optimized to ensure accurate temperature data collection and efficient data transmission, while minimizing resource utilization and power consumption. The firmware development encompassed various aspects, including the design of the firmware architecture, algorithm implementation, and integration of communication protocols.

The following sections of this documentation provide detailed insights into the project's hardware and firmware requirements, operation flow, device usage, limitations, and potential future improvements. The schematics and PCB layout diagrams offer a comprehensive view of the hardware design, while the firmware section provides an in-depth analysis of the software implementation.

1.1 Hardware Requirements

The chosen wireless protocol for this project was WiFi, and so, the ESP8266EX microcontroller was selected for its robustness, affordability, widespread acceptance in the firmware market and strong community support. Additionally, this MCU has the capability to work with RF signals at power levels of up to +20dBm in its TX, and sensitivities of -91dBm and -75dBm for 11Mbps and 54Mbps on its RX, respectively. Based on these values, under ideal conditions, signals can travel up to 300m. However, it is important to validate the project concept in real-world scenarios, where noisy environments and obstacles are almost always present.

For the temperature sensor, the TMP35 was chosen due to its low power consumption, ability to be powered off when not in use, good accuracy, and similarity to other widely used components in the market, such as the LM35.

As for the flash EEPROM memories, the MX25R3235F (32Mbit) and AT25SF081 (8Mbit) chips were selected for program and collected data storage, respectively. These ICs were chosen because they meet the size requirements (minimum 2MB for program and 500kB for data) and operate under the serial SPI communication protocol.

1.2 Firmware Requirements

For the firmware, the development was chosen to be carried out using the Arduino IDE, as it works with C++, a language accepted by the chosen microcontroller. Additionally, this IDE provides pre-developed board configurations, which can be helpful during the prototyping phase. Moreover, it makes firmware uploading easier since we can utilize ready-made modules for these tasks, such as the well-known ESP8266 NodeMCU 1.0.

For the WiFi protocol, the "ESP8266WiFi.h" library is used, licensed under the LGPL (Lesser General Public License) Version 2.1. In broad terms, the license can be interpreted as follows:

"Primarily used for software libraries, the GNU LGPL requires that derived works be licensed under the same license, but works that only link to it do not fall under this restriction."

The candidate chose to use this library because they do not have solid knowledge of the WiFi protocol (from the physical layer to the application layer) to develop their own library or subroutines. A full copy of the LGPL Ver 2.1 can be found at [9].

For the communication between the microcontroller and the Flash Data Memory, the SPI communication protocol is used. Also, the it was implemented an algorithm for a linear time-invariant system, namely the Moving Average Filter. Such algorithm is required to ensure that the measured temperature remains in a stable state when it is saved in the flash data memory integrated into the hardware.

1.3 Operation Flow

The final device workflow is simple: it stabilizes the reading/collecting of temperature data, stores it on the flash EEPROM, and then opens communication with a access point, access a localhost webserver url and transmits the collected data. Finally, it erases the last 512,000 samples of temperature and starts the process again.

1.4 Limitations and Future Improvements

To reduce the code complexity, it was assumed that the server the board communicates with via the Access Point (AP) is a "localhost," and therefore, no SSL-HTTPS security configurations were implemented. Additionally, no ACK verification is performed to ensure that the server has indeed received the transmitted data. Furthermore, no HTTP POST messages containing SQL commands for writing to a hypothetical database are implemented. It is assumed that this set of business rules would be on the server side. Also, to avoid further increasing the code length (which has already been extended due to an unnecessary feature - reading temperature with precision and storing it in memory), no routines were implemented to ensure that the data was successfully written to the PCB memory. Finally, the device does not have any interface; it is simply plug and play. If there were any specific requirements, such an interface could be added subsequently.

2 Circuitry

2.1 Block Diagram

The block diagram of the device on the Figure 1 consists of several main components that play essential roles in its operation. Here is a brief overview of each of these parts:

- Microcontroller (ESP8266EX): The ESP8266EX microcontroller is the central component of the device. It is responsible for controlling all operations and functionalities, including wireless communication via WiFi.
- Program Memory (MX25R3235F): The program memory is where the device's firmware is stored.
- Data Memory (AT25SF081): The data memory is used to store the collected data in the device.
- Temperature Sensor (TMP35): The TMP35 temperature sensor measures the ambient temperature.
- RF Antenna (Texas SWRA117D 2.4GHz - Left): The RF antenna plays a crucial role in the wireless communication of the device. It enables the transmission and reception of radio signals, allowing WiFi communication. The RF antenna ensures a stable and reliable connection between the device and the access point (AP), facilitating the transmission of collected data. More information on this type of PCB antenna can be found at [3].

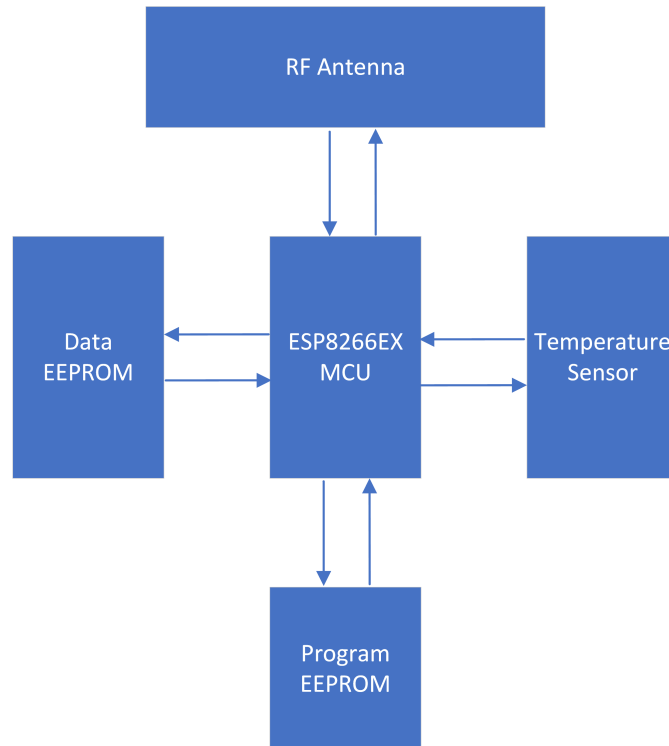


Figure 1: Device's block diagram.

2.2 Schematics

To further delve into the circuit's details, on Figure 2 is the schematic diagram of the circuit. It is also divided into four blocks, each serving a specific purpose in the device's functionality as follows:

- **Power Supply Block:** The power supply block provides a stable +3.3V voltage to the entire circuit. It ensures that all components receive the necessary power to operate efficiently and reliably. The power supply block includes a 3.7V (mAh to be decided, depending on the longevity needed) and a voltage regulators. Decoupling capacitors and other necessary components to regulate and distribute power are contained in the "Main Circuitry" (see Figure 2).
- **Program and Data Memory Block:** This block encompasses the program and data memories of the device. The program memory, represented by the MX25R3235F chip, stores the firmware responsible for controlling the device's operations. The data memory, represented by the AT25SF081 chip, stores collected data for future transmissions. This block includes the necessary connections between the microcontroller and the memory chips.
- **Temperature Sensing Block:** The temperature sensing block includes the temperature sensor component (TMP35) and its associated circuitry. The sensor measures the ambient temperature and then converts it temperature reading into a electrical analog format for processing by the microcontroller. This block is responsible for accurate temperature measurement and data acquisition.
- **Main Block (Microcontroller and Components):** The main block contains the microcontroller (ESP8266EX) and its associated components, in addition to the 2.4GHz RF antenna. The microcontroller serves as the central processing unit of the device, executing firmware instructions and coordinating various operations. This block includes components such as crystal oscillators, capacitors, resistors and inductors; these are the supporting elements necessary for the microcontroller's functionality.

These four blocks work together to ensure the device's proper operation. The power supply block provides the necessary voltage, the program, and data memory block stores and retrieves essential information, the temperature sensing block enables temperature measurement. Last but not least, the main block with the microcontroller controls and manages the device's overall functionality.

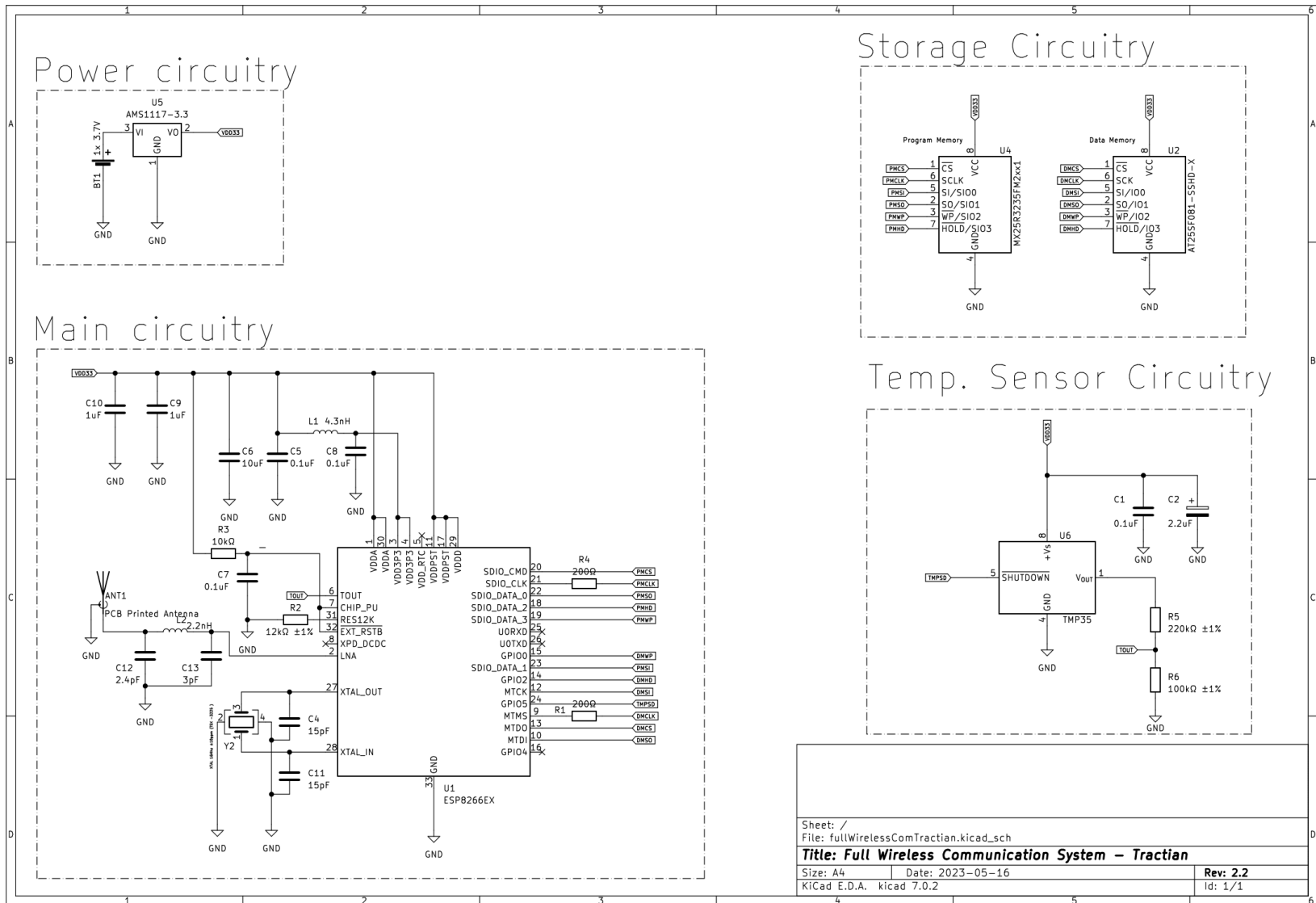


Figure 2: Circuit Schematics.

2.3 Circuit Diagrams

We start highlighting the main points of the Main Circuitry, putting important design requirements for the properly function of the ESP8266EX. Most of the references were obtained on [5] and [4].

2.3.1 Digital and Analog Supplies Circuitry

For the analog power supply, ESP8266EX features five analog pins dedicated to power supply, encompassing Pin1, Pin3, and Pin4, which specifically cater to the power requirements of the internal PA and LNA. Furthermore, Pin29 and Pin30 serve as power inputs for the internal PLL. The acceptable voltage range for these analog power supply pins spans from 2.5 V to 3.6 V [4].

As for the digital power supply, ESP8266EX is equipped with two digital pins, namely Pin11 and Pin17, dedicated to power supply. When it comes to the digital power supply, the inclusion of supplementary filter capacitors is unnecessary. The operational voltage range for the digital power supply pins spans from 1.8 V to 3.3 V [4]. Both Analog and Digital power supply connections can be seen on Figure 3

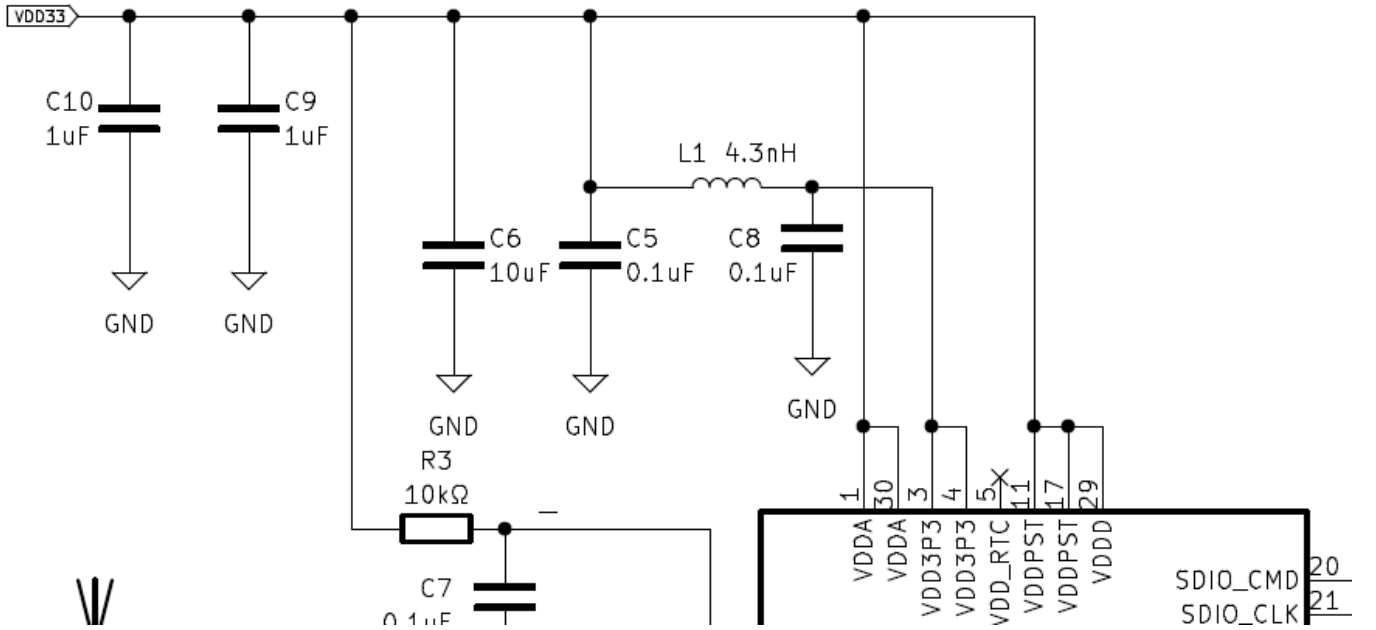


Figure 3: Digital and Analog Supplies Circuitry.

2.3.2 RC Delay Reset Circuitry

The reset pin of ESP8266EX is denoted as Pin32 EXT_RSTB and plays a vital role in the device's operation [4]. This pin possesses an internal pullup resistor and operates on an active low mechanism. As for Pin7 CHIP_EN, it functions as the enable pin for ESP8266EX. When held at a low state, ESP8266EX powers off [4]. Its connections can see on the Figure 2.3.2 shwon in the section (2.3.3) below.

2.3.3 12Kohm RES

An external GND 12k Ω showed on Figure 2.3.2 resistor should be connected with the ERS12K pin (pin 31) [4]. This component must have high accuracy when controlling the bias current. For this, a 12k $\Omega \pm 1\%$ is required.

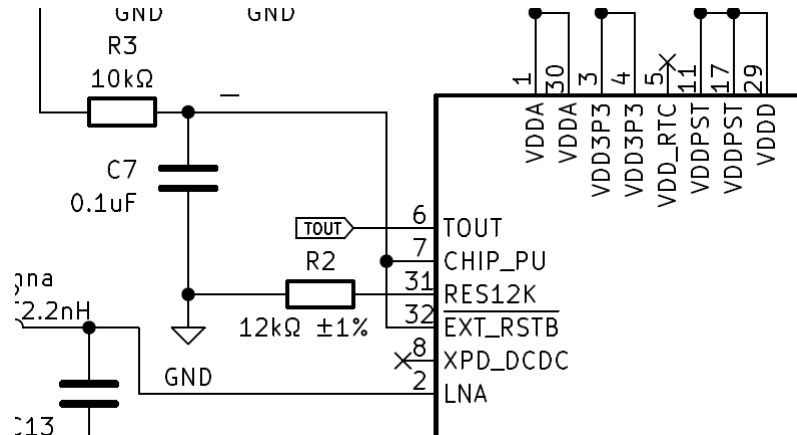


Figure 4: RC Delay Reset circuitry and 12K RES Pin.

2.3.4 RF Antenna Circuitry

For the 2.4GHz RF antenna, in terms of “abstract” circuit itself, the only requirements is that the the impedance at the output terminal of the ESP8266 power amplifier is $39 + j6\Omega$, which necessitates a corresponding matched impedance of $39 - j6\Omega$ (from the antenna to the chip). Moreover, for the RF PCB design, a 50Ω impedance is required on the track design. For such implementation, see the Figure 5 below (also, see the section 3.2.5).

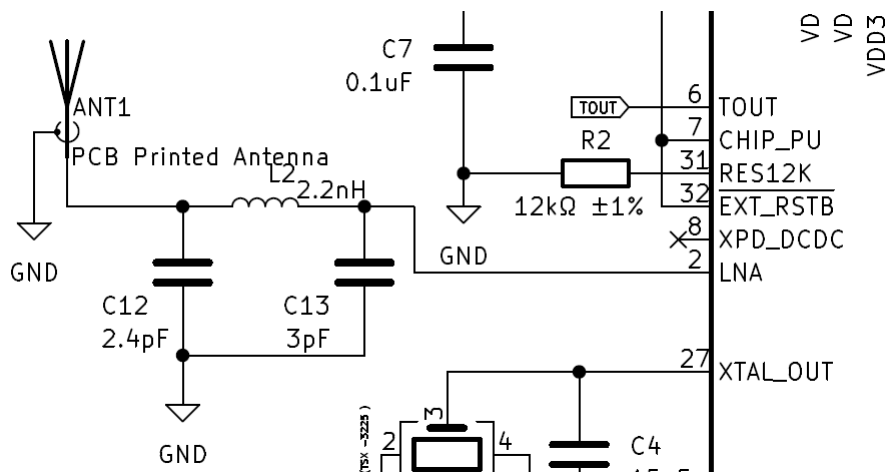


Figure 5: 2.4GHz Antenna circuitry.

2.3.5 XTAL Circuitry

ESP8266EX has the capability to support crystal oscillators with frequencies of 40 MHz, 26 MHz, and 24 MHz. In the circuit design, capacitors C4 and C11 are connected to the ground

and added to the input and output terminals of the crystal oscillator, respectively. The values of these capacitors can be varied within a range of 6 pF to 22 pF - 15pF was chosen. It is crucial that the crystal oscillator used has a precision of ± 10 PPM (parts per million) to maintain accurate timing and synchronization. For the XTAL chip, the FA238 (26Mhz) From Seiko Epson Corporation was chosen [6]. The electrical connection for the crystal oscillator is shown in Figure 6.

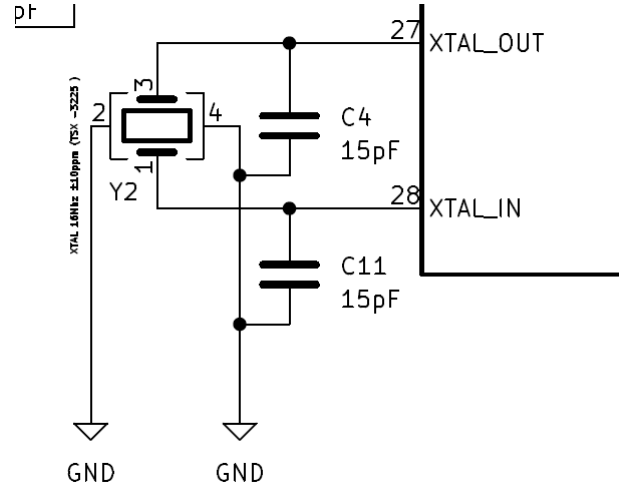


Figure 6: Crystal Oscillator Circuitry.

2.3.6 Drive Current Serial Clock RES

To reduce drive current and eliminates external interruptions, a $200\ \Omega$ is connected to the flashes (Pin21 SD_CLK and Pin9 MTMS). It is recommended a 0402 resistor package [4].

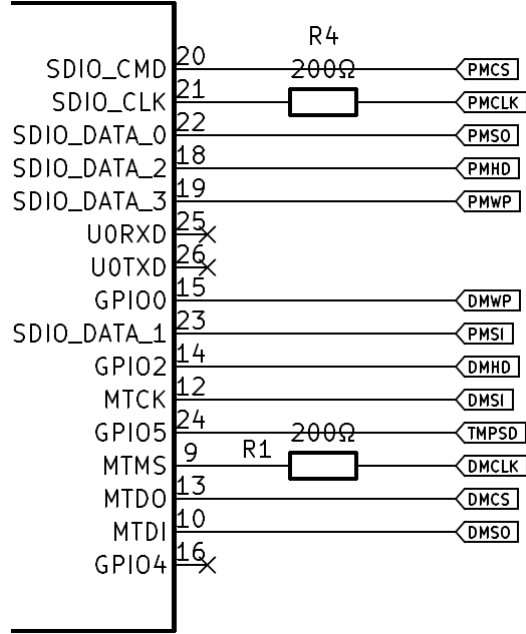


Figure 7: 200Ω RES Drive Current.

As for the Nets connected on each of the pins shown in Figure 7, “P” stands for “Program”, “D” is “Data”, “SI” and “SO” are “Serial IN” (MOSI) and “Serial Out” (MISO) respectively (notice the “IN” “Out” refers to the slave pins). Moreover, “SCLK” means “Serial Clock”, “WP” is “Write Protect”, “HD” refers to “Hold”. Finally, “TEMPSD” stands for “TMP 35 Shutdown”. So, for instance, “DMCLK” refers to the “Data Memory Serial Clock” Net. From this point, the rest of the blocks are described.

2.3.7 Power Circuitry

For the power circuitry, a 3.7V battery is used, along with a voltage regulator. The AMS1117 [2] for 3.3V integrated circuit (IC) was utilized to ensure protection and integrity of the power supply.

Power circuitry

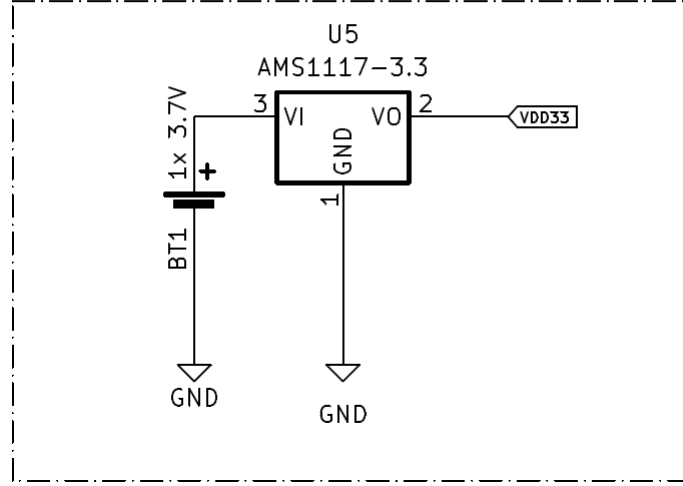


Figure 8: 3.3V Power circuitry.

2.3.8 Storage Circuitry

As for the program memory, since there is no internal memory available, an external flash memory is required for the ESP8266EX to store the user program. It is important to consider that the minimum capacity of the memory should be 512kB for disabled OTA (firmware updates Over The Air) or 1MB for enabled OTA. For firmware programming into the memory, the ESP8266EX supports Standard SPI, Dual SPI, and Quad SPI, which need to be selected prior to the programming process. The chosen memory, MX25R3235F [10], meets all the aforementioned requirements.

For the data memory, the requirements essentially boil down to having sufficient space (min 500kB) and supporting SPI communication. The chosen IC, AT25SF081 [1], fulfills these requirements. Additionally, this IC allows for serial data traffic on its I/Os at speeds of up to 50MHz (85MHz and 104MHz in specific cases). Since the communication with this chip was achieved through bit banging (see section 4.4), with an SCLK oscillation period of $6\mu s$, *i.e.*, a frequency of $f = 166.666\text{kHz}$, operating with this chip is entirely feasible, obviating the need for hardware-based SPI interface. Furthermore, The protection bits are set to zero by default [1], and there is no need to modify these bits. Therefore, the memory can be programmed and erased without any difficulties. Both program and data memory can be seen on Figure 9

Storage Circuitry

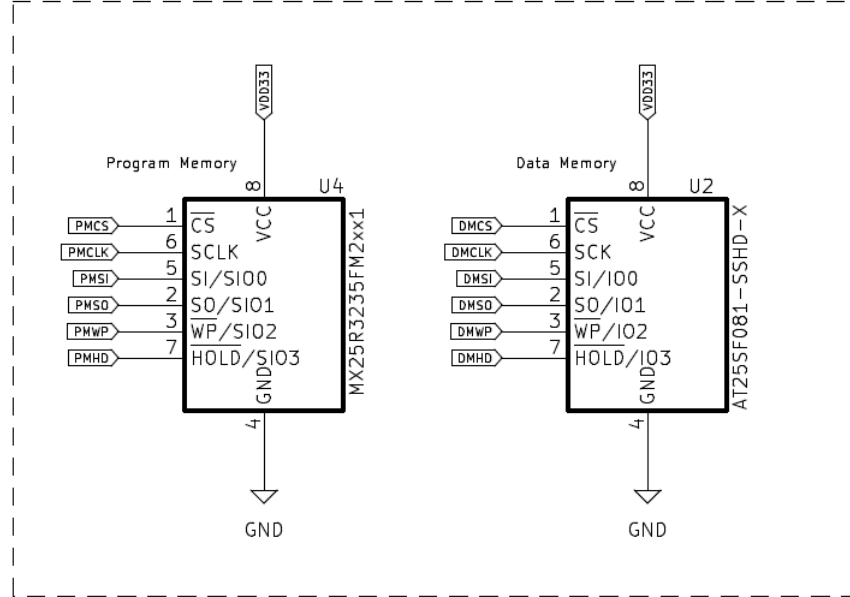


Figure 9: Storage circuitry - Program and Data Memory.

2.3.9 Temperature Sensor Circuitry

The TMP35 temperature sensor [7] provides analog temperature readings, which can be converted into temperature values using appropriate calibration by “voltage divider circuit” shown in the Figure 10. This adjustment is need as the ESP8266EX ADC Input can only receive values between 0 and 1V [5].

Temp. Sensor Circuitry

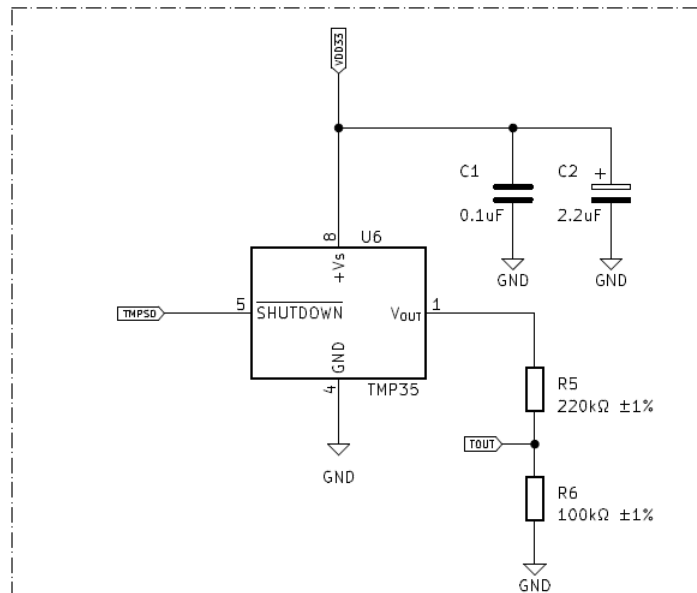


Figure 10: Temperature measuring circuitry.

3 Printed Circuit Board

3.1 PCB Design

3.2 PCB Specifications

In the realm of electronic product development, PCB design plays a crucial role in ensuring the optimal functionality and performance of microcontroller-based systems. This section will highlight some of the key points covered in the microcontroller's hardware design guidelines [4], along with best practices for PCB design. These guidelines provide valuable insights into the recommended approaches for designing the hardware layout, addressing critical aspects such as power supply, signal integrity, component placement, and grounding. By adhering to these guidelines and following established best practices, designers can achieve robust and reliable PCB designs that maximize the microcontroller's potential and minimize potential issues during manufacturing and operation.

3.2.1 ESP8266EX Power Supply Design

Prior to the power traces reaching the analog power-supply pins (Pin1, 3, 4, 28, 29), the inclusion of a 10 μF capacitor, working in conjunction with a 0.1 μF capacitor, becomes necessary. Additionally, an arrangement of a C circuit and an L circuit is advised for the power supplies of Pin3 and Pin4. The C-L-C circuit should be positioned as proximate as possible to the analog power-supply pin [4]. For more information about these components, refer to Figure 3. It is crucial to acknowledge that the power supply pathway can be susceptible to damage as a consequence of abrupt current surges during the transmission of analog signals by ESP8266EX. Consequently, it is imperative to incorporate an additional 10 μF capacitor, packaged in either 0603 or 0805 dimensions, to complement the existing 0.1 μF capacitor [4].

3.2.2 Layers Design

On [4] there are two suggested models regarding the number and dispositions of layers on a PCB that uses ESP8266EX (and by extension, an 2.4GHz RF antenna). The model with four layers was chosen. Here are the specifications:

- The initial layer corresponds to the uppermost part and is designated as the TOP layer, utilized for signal lines and component placement.
- The subsequent layer represents the GND layer, dedicated solely to establishing a comprehensive ground plane without the inclusion of signal lines.
- The following layer constitutes the POWER layer, exclusively reserved for the positioning of power lines. In certain unavoidable situations, it is admissible to incorporate some signal lines within this layer.
- Lastly, the final layer denotes the BOTTOM layer, exclusively allocated for signal line placement. It is discouraged to mount components on this particular layer.

Also, the power tracks should have a minimum size of 15 mils. The value used was 20 mils.

3.2.3 Reset Track Design

The EXT_RSTB (Pin32) possesses an internal pullup resistor and operates on an active low mechanism. To mitigate the possibility of external disturbances causing unwarranted resets, it is advised to maintain a concise PCB trace for EXT_RSTB and incorporate an RC circuit at this pin [4] (see section 2.3.2).

3.2.4 XTAL - Crystal Oscillator Design

To ensure proper functioning of the crystal oscillator, it is recommended to position it as close as possible to the XTAL pins, while considering the length of the traces. However, it is crucial to maintain an appropriate distance between the crystal and the chip to avoid any interference. The recommended distance from the MCU is 0.8 mm. This distance helps prevent any potential disruption caused by the crystal's proximity to the chip. Additionally, it is important to note that there should be no vias present on the input and output traces. This means that the traces should not cross layers, ensuring a clean and uninterrupted signal path for the crystal oscillator. This design consideration helps maintain the integrity of the crystal oscillator's signals and enhances the overall performance of the circuit.

3.2.5 RF Antenna

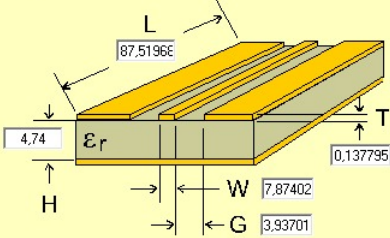
The RF impedance characteristic is 50Ω , which indicates the desired impedance for optimal signal transmission. It is crucial to have a complete ground plane to ensure proper grounding and minimize interference. The RF trace should be kept as short as possible, and it is recommended to have dense ground via stitching surrounding it for isolation purposes. The width of RF lines should be minimized, and additional dense vias should be placed around them. A π -type matching circuitry (see the PCB file .kicad_pcb on the “KiCad Projeto Traction” folder). should be allocated on the RF trace and positioned in close proximity to the RF Pin2. It is important to avoid the use of vias for the RF trace. When routing the RF trace, a 135° angle is preferred, or circular arcs can be utilized for trace bends. Furthermore, the RF antenna should be placed at a distance from high-frequency transmitting devices, such as crystal oscillators, DDR, and specific high-frequency clocks (for example, SPI Clocks). Additionally, it is essential to keep USB ports, USB-to-UART signal chips, and UART signal lines (including traces, vias, test points, headers, etc.) as far away from the antenna as possible. More specific iformations can be seen on [4]. For the circuitry previous to the antenna, see the Figure 5. For the its PCB design, the Texas SWRA117D 2.4GHz - Left [3] was used.

For the 50Ω impedance on track design, the following values were used (Figure 11). The calculation was done using the **AppCad** *software* [8]. Notice that the inputed values are in mils.

AppCAD - [CPW]
 File Calculate Select Parameters Options Help

Coplanar Waveguide

☒ With Groundplane ☐ No Groundplane



Calculate Z0 [F4]

Z0 = **50.0** Ω

Elect Length = **0.317** λ

Elect Length = **114.1** degrees

Elect Length = **155.863** mil (Air Line equiv.)

Delay = **13.206** ps

1.0 Wavelength = **276.146** mil

Vp = **0.562** fraction of c

ε eff = **3.17**

Shape factor = **0.500**

Dielectric: εr = **4.6**

FR-4

Frequency: **2.4** GHz

Length Units: **mils**

Figure 11: Calculation of the 50Ω impedance - Coplanar Waveguide.

4 Firmware

4.1 Firmware Description

The ultimate goal of this firmware is to establish a complete wireless communication between a data collecting station (ESP8266EX) and an access point located 100m away using the WiFi protocol. The hardware collects data on-site, stores it with a size of 500kB, and then sends it to the access point.

For the purpose of data collection, a temperature measuring circuit was implemented on the hardware using a TMP36 IC. A linear time-invariant system, namely the Moving Average Filter, is required to ensure that the measured temperature remains in a stable state when it is saved in the flash data memory integrated into the hardware.

For the WiFi protocol, the library "ESP8266WiFi.h" was used.

For the communication between the microcontroller and the Flash Data Memory, it is used the SPI communication protocol.

4.2 Configurations and Parameters

The code includes various configurations and parameters for an ESP8266 program involving SPI communication, temperature reading, and Wi-Fi transmission. These parameters are the following.

The <ESP8266WiFi.h> library is a header file specific to the ESP8266 platform. It provides the necessary functions and definitions to enable Wi-Fi connectivity and communication on the ESP8266 module. This library allows the program to connect to Wi-Fi networks, perform network-related operations, and establish connections with hosts over the network.

WiFiClient _hostClient instance:

- The WiFiClient _hostClient; creates an instance of the WiFiClient class named _hostClient. The WiFiClient class is part of the ESP8266WiFi library and represents a client connection in a Wi-Fi network. In this case, _hostClient is used to establish a connection with the host specified in the code (HOST). It will be responsible for handling the communication between the ESP8266 and the host, enabling data transmission over the established connection.

By creating an instance of WiFiClient, the program can make use of its methods and functionalities to connect to a host and exchange data.

GPIOs for SPI communication with data memory:

- DMCS: Data Memory Chip/Slave Select, set to 15 (GPIO15).
- DMCLK: Data Memory Serial Clock, set to 14 (GPIO14).

- DMSI: Data Memory SI (slave data input) (MOSI), set to 13 (GPIO13).
- DMSO: Data Memory SO (slave data output) (MISO), set to 12 (GPIO12).
- DMWP: Data Memory Write Protect, set to 0 (GPIO0).
- DMHD: Data Memory Hold, set to 2 (GPIO2).

SPI delay for 8 MHz communication:

- Half_SPI_Period: delay period for SPI communication in microseconds, set to 3. This corresponds to a frequency of approximately 166.666 kHz.

Temperature sensor GPIO definitions:

- Temp_Sensor_ADC_Pin: analog input pin (ADC) for TMP35 sensor output, set to 17.
- Temp_Sensor_Shutdown_Pin: shutdown pin (SHUTDOWN) for TMP35 sensor, set to 5 (GPIO5).

Network settings:

- SSID: Service Set Identifier (SSID) of the Wi-Fi network to be connected, set to "Fake_WiFi_Net_To_C".
- PSSWRD: Password of the Wi-Fi network to be connected, set to "Fake_WiFi_Net_Password".
- HOST: Fake URL to send data, set to "192.13.14.5/temperature_log_files". It is assumed to be used for sending a 500 kB file.
- _httpPort: HTTP port to be used, set to 80.

Arrays and variables related to temperature data reading and sending:

- Temp_Vector: byte array that stores 256 samples of stable temperature measurements.
- Temp_Buffer: buffer to store 50 samples of instantaneous temperature readings.
- _addressBytes: byte array with 3 bytes for memory address.
- _readData: byte array used to read 1 kB of information from data memory and send it to the host.
- ADDRESS: starting memory address for writing, set to 0x000000.
- _packetCount: counter for packets sent to the host.
- _numOfTries: counter for the number of connection attempts to the host before giving up and collecting another set of temperature values.
- p: counter for the 500 kB file (counts the number of 1024-byte packets sent to the host).

- q: counter for the 1024 bytes to be sent in `_readData`.

The actual code for this configurations are shown bellow:

```

1
2 #include <ESP8266WiFi.h>
3
4 WiFiClient _hostClient; // creates an instance of WifiClient for
   the host to be connected
5
6 // Data Memory GPIO's definitions for SPI comm:
7 #define DMCS 15 // Data Memory Chip/Slave Select - Pin 13 on
   ESP8266EX -> (MTDO) GPIO15
8 #define DMCLK 14 // Data Memory Serial Clock - Pin 9 on ESP8266EX
   -> (MTMS) GPIO14
9 #define DMSI 13 // Data Memory SI (slave data input) (MOSI) - Pin
   12 on ESP8266EX -> (MTCK) GPIO13
10 #define DMSO 12 // Data Memory SO (slave data output) (MISO) - Pin
   10 on ESP8266EX -> (MTDI) GPIO12
11 #define DMWP 0 // Data Memory Write Protect - Pin 15 on ESP8266EX
   -> (GPIO0) GPIO0
12 #define DMHD 2 // Data Memory Hold - Pin 14 on ESP8266EX -> (GPIO2)
   GPIO2
13
14 // SPI delay for 8Mhz comm (Flash Data Memory operates up to 8Mhz
   for single IO mode)
15 // For we are using bit banging data, a (way) lower frequency will
   be used.
16 // Acording to Arduino Reference, for 'delayMicroseconds(t);'
   operates very accurately
17 // for a minimum valeu of t = 3 (3us). Being so, The period of the
   SCLK shall be 6us,
18 // which leads to f = 166.666 kHz. That being said, every 3 seconds
   or so 500,000
19 // samples of temperature will be collected. This barely has any
   practical uses,
20 // but the principle of the challenge stands.
21 #define Half_SPI_Period 3 // for f = 8MHz , T = 6us
22
23 // Temperature Sensor GPIO definition
24 #define Temp_Sensor_ADC_Pin 17 // TMP35 Vout - Pin 6 on ESP8266EX
   -> (TOUT) ADC_in | On NodeMCU it is mapped to pin 17
25 #define Temp_Sensor_Shutdown_Pin 5 // TMP35 SHUTDOWN - Pin 24 on
   ESP8266EX -> (GPIO5) GPIO5
26

```

```

27 const char* SSID = "Fake_WiFi_Net_To_Connect"; // service set
    identifier of the AP to be connected
28
29 const char* PSSWRD = "Fake_WiFi_Net_Password"; // net password of
    the AP to be connected
30
31 const char* HOST = "192.13.14.5/temperature_log_files"; // Fake URL
    to send data (500kB file)
32
33 const int _httpPort = 80; // HTTP Port to be used
34
35 // Global Array to store 256 Stable Temperature Readings (later to
    be used on Data Memory Write)
36 byte Temp_Vector[256]; // records 256 sample of stables temp
    measurements to record on a page on Data Memory
37 byte Temp_Buffer[50]; // Buffer to stores 50 samples of
    intantaneous Temperature readings
38
39 // 3Byte Memory Address
40 byte _addressBytes[3];
41
42 // 1024 array for reading from data memory and sending via WiFi (
    total of 500 packets of 1024 bytes to be sent -> 500kB)
43 byte _readData[1024];
44
45 // Memory Address Word
46 unsigned long ADDRESS = 0x000000; // By our definition, Memory
    Write starts at address 0
47 int _packetCount = 0; // count var for Page Program
48
49
50 int _numOfTries = 0; // starts with 0 - count the number of times
    the station tried to connect with to host before it gives up and
    leads to another set of collecting tempereture values
51 // int _connectionSuccess; // indicates if the connection to the
    host was sucessful (1) or not (0)
52
53
54 int p; // count var for 500kB file (counts the number os 1024bytes
    packets sent to the host - total of 512,000 bytes)
55 int q; //count var (parameter of _readData) for 1024 bytes to be
    sent

```

4.3 Firmware Architecture

At a high level, the code operates with the following flowchart to outline its functionality. First, it performs a setup process to initialize necessary variables and configurations. Then, it enters the main loop where it activates the temperature sensor, collects temperature data, and stores it in memory. Afterward, it turns off the temperature sensor. Next, it attempts to establish a connection with the designated Wi-Fi Access Point (AP). If successful, it tries to connect to the host URL, allowing for a maximum of 10 connection attempts. If the connection is established, it sends the collected temperature data to the host. Finally, regardless of the connection outcome, the code disconnects from the host and returns to the beginning of the main loop to repeat the process. This flowchart provides a high-level overview of the code's functionality and the sequential steps it follows to achieve its intended purpose.

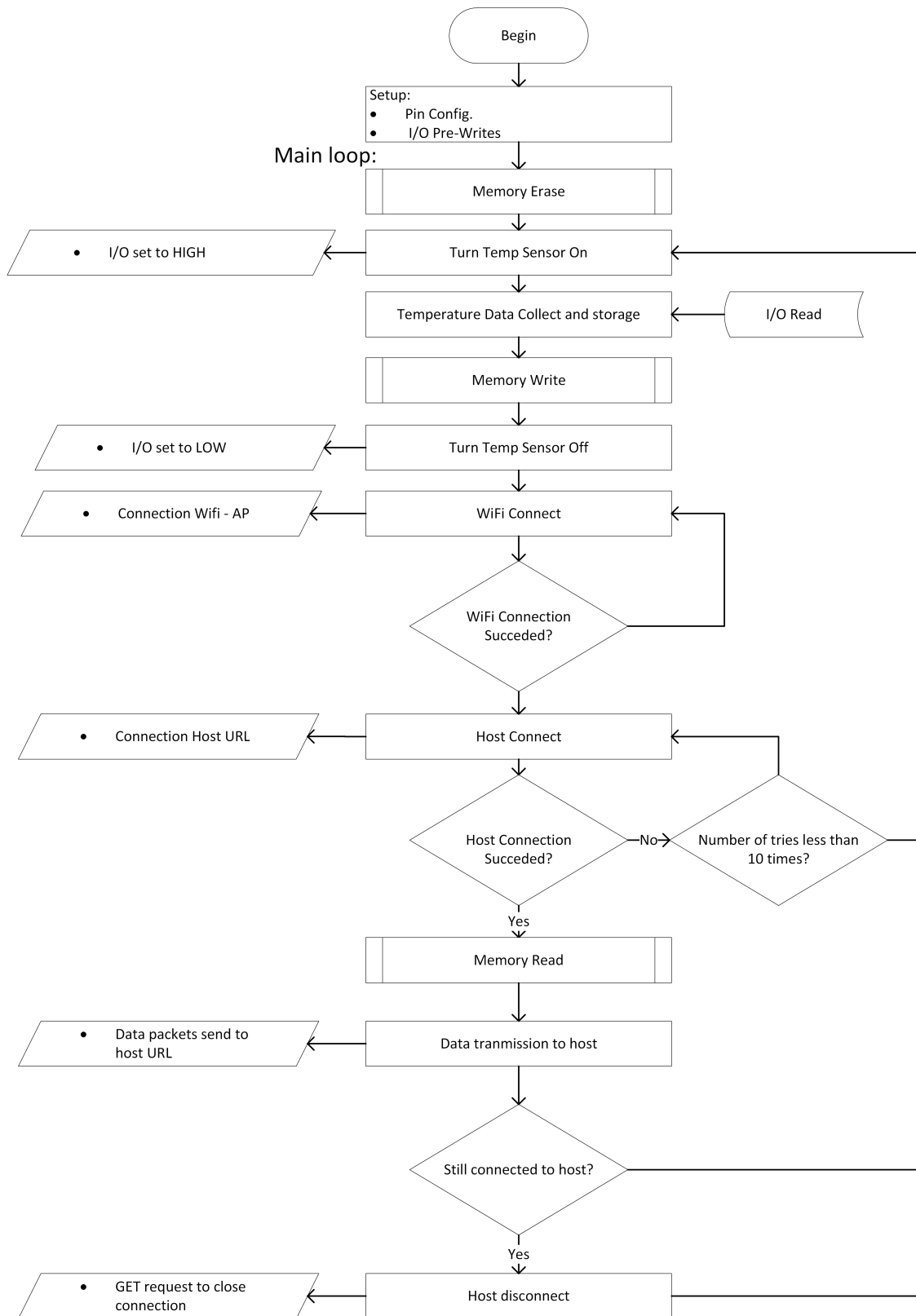


Figure 12: High level overview of the firmware.

4.4 Features and Algorithms

In order to stabilize the instantaneous temperature readings, an Moving Average Filter Algorithm was implemented in the project. This algorithm takes a series of temperature measurements and calculates their average over a specified window of time or number of samples. By smoothing out variations and fluctuations in the temperature data, the Moving Average Filter Algorithm helps to provide a more stable and reliable temperature reading.

Additionally, as part of a possible future implementation, a general 3-byte array generator algorithm was implemented. This algorithm takes a long variable in the form of six nibbles "0xXYZWTR" as input. It then extracts and organizes the individual nibbles of the long variable into a 3-byte array. This allows for efficient storage and manipulation of the long variable in byte-based operations. The 3-byte array representation offers flexibility for future use, such as data transmission, storage, or further processing, depending on the project's requirements.

In addition to the mentioned components, the project also incorporates subroutines for Data Memory operations, including Write, Read, and Erase functionalities. These subroutines enable the program to interact with the Data Memory module connected to the ESP8266. The Write subroutine facilitates the storage of data, such as temperature readings or other relevant information, into the Data Memory. The Read subroutine allows retrieving data from the Data Memory, enabling access to previously stored information. Lastly, the Erase subroutine provides a mechanism to clear or reset the Data Memory, erasing any existing data stored within it. By implementing these subroutines, the project gains the capability to effectively manage and manipulate data in the Data Memory module, facilitating data logging, analysis, or further processing as required.

4.5 Interfaces and Protocols

In the project, a routine for SPI communication using bit banging was implemented. This approach allows direct control over the individual signal lines involved in the SPI protocol. The SPI protocol is widely used for communication between microcontrollers and peripheral devices such as the data memory chip in this case. The implemented routine operates on Mode 0, which is accepted by the data memory chip. Mode 0 means that the clock signal is idle low, and data is sampled on the leading edge of the clock signal.

The SPI communication routine has a clock period of 6 microseconds ($6\mu s$), resulting in a frequency of approximately 166,666 kHz. This frequency falls within the acceptable range supported by the memory IC, ensuring reliable and efficient data transfer. By utilizing bit banging and adhering to the Mode 0 protocol with an appropriate clock frequency, the implemented routine enables effective communication between the ESP8266 and the data memory chip, facilitating data read/write operations and supporting the overall functionality of the project.

References

- [1] *8-Mbit SPI Serial Flash Memory with Dual-I/O and Quad-IO Support - AT25SF081 Datasheet*. URL: <https://pdf1.alldatasheet.com/datasheet-pdf/view/606550/ETC2/AT25SF081.html>.
- [2] *Advanced Monolithic Systems - AMS1117 Datasheet*. URL: <http://www.advanced-monolithic.com/pdf/ds1117.pdf>.
- [3] *Application Note AN043 Small Size 2.4 GHz PCB antenna*. URL: https://www.ti.com/lit/an/swra117d/swra117d.pdf?ts=1684435789781&ref_url=https%253A%252F%252Fwww.google.com%252F.
- [4] *ESP8266 Hardware Design Guidelines*. URL: https://www.espressif.com/sites/default/files/documentation/esp8266_hardware_design_guidelines_en.pdf.
- [5] *ESP8266EX Datasheet*. URL: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.
- [6] *FA-238 Mhz Range Crystal Unit - FA238 Datasheet*. URL: https://br.mouser.com/datasheet/2/137/FA_238_en-932914.pdf.
- [7] *Low Voltage Temperature Sensors TMP35/TMP36/TMP37 - TMP35 Datasheet*. URL: <http://www.advanced-monolithic.com/pdf/ds1117.pdf>.
- [8] *The AppCad Software*. URL: <https://www.broadcom.com/info/wireless/appcad>.
- [9] *The GNU GLPL full text*. URL: <https://github.com/esp8266/Arduino/blob/master/LICENSE>.
- [10] *Ultra low power, 32Mbit [x 1/ x2/ x4] CMOS MXSMIO (Serial Multi I/O Flash Memory - MX25R3235F Datasheet)*. URL: <https://www.macronix.com/Lists/Datasheet/Attachments/8755/MX25R3235F,%20Wide%20Range,%2032Mb,%20v1.8.pdf>.