Politecnico Di Torino

Master of Science in Electronic Engineering

A Project Report on

Design of electromagnetic sensors for civil applications

Supervisors:
Prof. Luciano Scaltrito
Prof. Sergio Ferrero

Candidate:
Kartik Upadhyay
264248

ACADEMIC YEAR 2021-2022

ACKNOWLEDGMENT

Firstly, I would like to express my deepest gratitude to both of my project guide, **Prof. Luciano Scaltrito** and **Prof. Sergio Ferrero** for their persistent support, guidance, and encouragement during the whole process of my project work.

Secondly, I would like to express my special gratitude to **Mr. Giorgio Damosso**, **Mr. Massimiliano Messere, Mr. Gianluca Melis, Mr. Matteo Manachino, Mr. Andrea Porceddu,** and **Mr. Francesco Perrucci** of **Microla Optoelectronics srl** for providing opportunity, guidance and support.

Ultimately, I would like to especially thank *my parents* **Mr. Tarun Upadhyay** & **Dr. Namrata Upadhyay** and *my brother* **Dr. Nirved Upadhyay**.

ABSTRACT

The aim of project "Design of electromagnetic sensors for civil applications" is to create a prototype device which makes increase the traceability of transportation of materials supplied at the construction sites. I have created two versions of prototype device:

1. Base Version,

2. Premium Version.

Base version is consist of two hardware boards, whereas, Premium version consist of three hardware boards. In both versions, we have TAG - PROVINO board and RECEIVER ESTERNO board but, in premium version we have additional board called CASSAFORMA board.

TAG -PROVINO board is placed inside the material box which needs to be traced. It's job is to collect all the data and store it and transfer it at appropriate time. It consist of microcontroller, memory, temperature sensor, shock sensor, light sensor(Photodiode), wake up system, communication block and non-rechargeable battery.

RECEIVER ESTERNO board is used collect all the data wirelessly from TAG - PROVINO board and supply it to our mobile device whenever required. It consist of microcontroller, wake up generator system, communication block, rechargeable battery and Bluetooth.

CASSAFORMA board is part of premium version because it has GPS module which increases cost of complete project. It consist of microcontroller, GPS module, communication block and rechargeable battery.

Keywords: IoT, Digitization, Sensor, PCB, Civil application.

CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

List of Symbols	Terms
IoT	Internet of Things
GPS	Global Position System
EEPROM	Electrically Erasable Programmable Read-Only Memory
PCB	Printed Circuit Board
BJT	Bipolar Junction Transistor
R	Resister
L	Inductance
C	Capacitor
AC	Alternative Current
AM	Amplitude Modulation
EM	Electro-Magnetic
Tx	Transmit
Rx	Receive
ADC	Analog to Digital Convertor
GNSS	Global Navigation Satellite System
IDE	Integrated Development Environment
GPIO	General Port Input Output
API	Application Programming Interface
HAL	Hardware Abstraction Layer
LL	Low Level
SRAM	Static Random Access Memory

Chapter 1

Introduction

The influence of technologies is growing day by day. Nowadays, we can find smart-devices anywhere and everywhere like vehicles, streets, and farms etc. These devices help us in increasing accountability, improve decision making and reduce resource wastage. My Thesis project has same goals. To develop a system through which we can trace and also certify the quality of material supply at construction sites by digitizing whole process. A system which will help make robust supply chains by increasing their reliability and making whole process traceable and more secure.

We need to build a module which can be fitted inside the box of materials supplied, which keep trace of location of the box, which can send us an alert if box is being opened or if it is being tempered with its material. Our system needs to be working on ultra-low power because battery will be the sole source of power inside the box with no rechargeable option. Our module needs to store all of its data inside itself till the time we can collect it. Our system must also include a receiver device, through which we can collect all the data. Our system needs to have communication ability internally as well as externally. Our external receiver needs a sub-block of wake-up generator to generate a unique signal to wake up the device inside the material box.

Based on the project requirement, we have created two version of our system:

1. Base Version: It consists of two devices, shown in Figure 1.1. Device which will be placed inside the material box is called TAG - PROVINO board and another device which we use to collect the data gathered by TAG-PROVINO device called RECEIVER ESTERNO board.

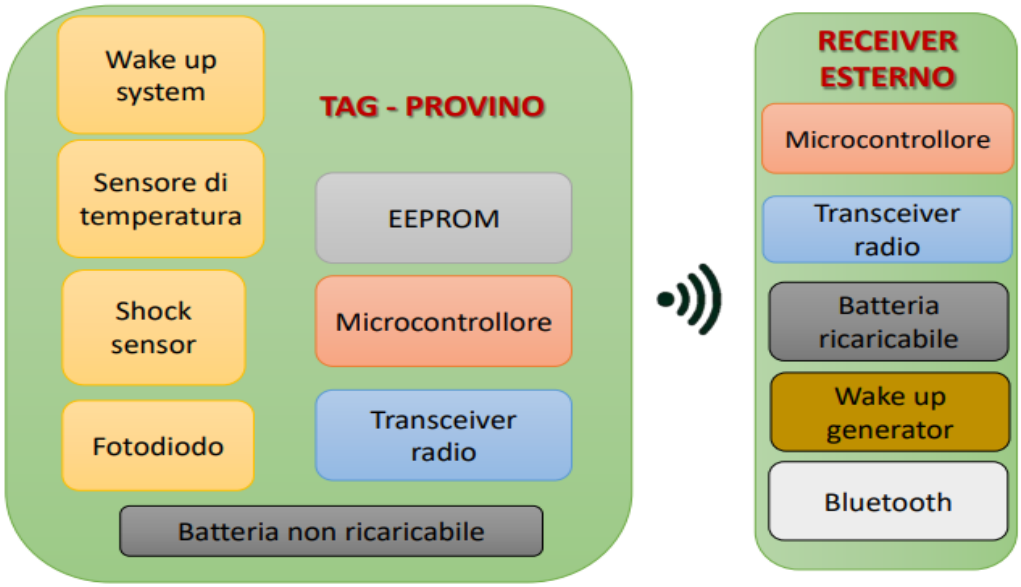


Figure 1.1: Base Version

2. Premium Version: Premium version consists of three devices, shown in Figure 1.2. In addition of two devices which are available in base version, we also have another device called CASSAFORMA, which consist of GPS module.

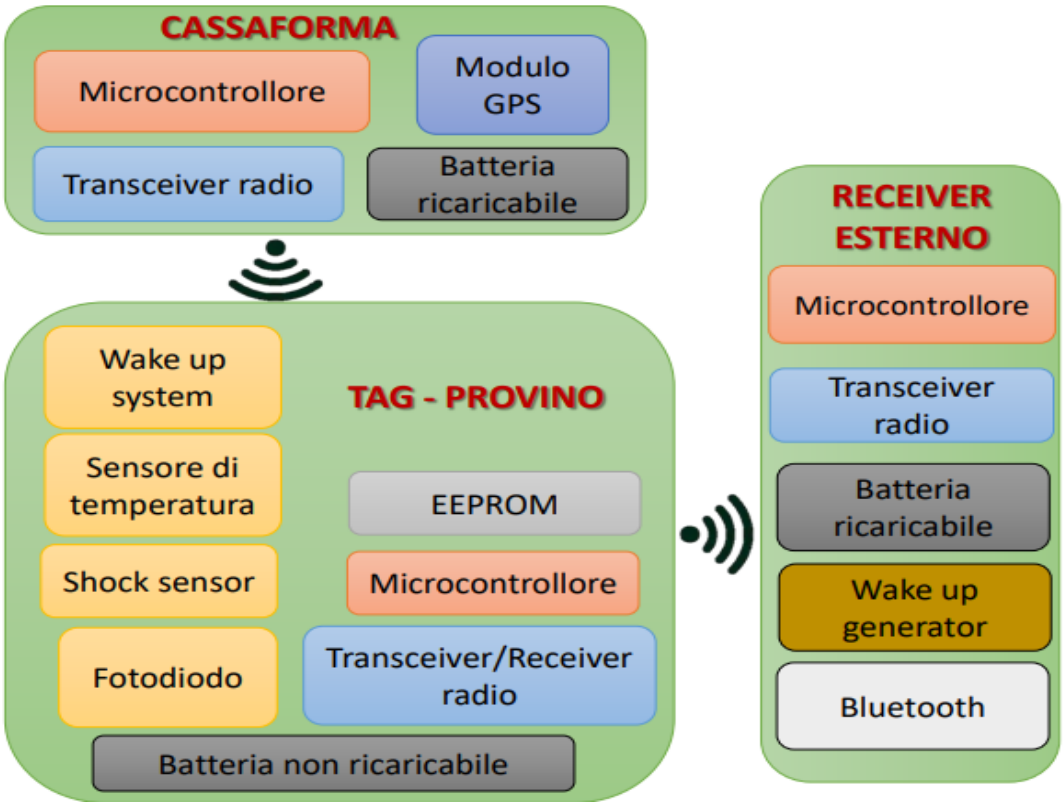


Figure 1.2: Premium Version

We created two version of our system based on the customer requirements. First, Base version is a low cost solution. It gathers data only from temperature sensor, Shock sensor(Vibration sensor) and Photodiode(Light Sensor). Second, Premium version is more expensive solution. It consist of all the features on base version. In addition, It has separate board for GPS module, which is the most expensive component of the whole project.

In the TAG- PROVINO board, Wake up system represents a module dedicated to wake up microcontroller from the deep sleep mode. Temperature Sensor is module to collect the value of temperature inside the material box. Shock sensor represents a module dedicated to identify if material box is tempered. Photodiode(Light sensor) represents a module to notify if material box is being opened. EEPROM represents the memory to all the data. Microcontroller is the brain which performs all the function. Transceiver/Receiver radio represents a module dedicated for the communication process. A non-rechargeable battery is required to power all the modules.

In the RECEIVER ESTERNO board, Microcontroller is the brain which performs all the function. Transceiver/Receiver radio represents a module dedicated for the communication

process. A rechargeable battery module is required to charge the battery. Wake up generator represents a module dedicated to send EM waves to wake up microcontroller present in TAG-PROVINO board from the deep sleep mode. Bluetooth represents a module dedicated for the communication from the board to the external world.

In the CASSAFORMA board, Microcontroller is the brain which performs all the function. GPS module is used to detect the location of the material box. Transceiver/Receiver radio represents a module dedicated for the communication process. A rechargeable battery module is required to charge the battery.

Further details about the Hardware configuration and software configuration of all three boards will be described in Chapter 2 and Chapter 3 respectively.

Chapter 2

Electrical Design

In this chapter, we will focus on the hardware part of the project. It consist of several parts like designing wake up generator and wake up receiver modules. The choice of microcontroller in the project and interfacing it with various sensors like temperature sensor, vibration sensor, photodiode and GPS module.

In section 2.1, Wake up generator is described and characteristics are Colpitts oscillator, software simulation and bread board prototype circuit.

In section 2.2, Wake up receiver is described and characteristics are LC circuit, receiver circuit software simulation and bread board prototype circuit.

In section 2.3, various experiments performed between bread board prototype of Wake up generator and Wake up receiver are described with their results.

In section 2.4, process of PCB designing of Wake up generator and Wake up receiver are described.

In section 2.5, various experiments performed between PCB prototype of Wake up generator and Wake up receiver are described with their results.

In section 2.6, choice of microcontroller is described along with the process of PCB designing of custom microcontroller board.

In section 2.7, Based on choice of microcontroller, structure of internal temperature sensor is described.

In section 2.8, Shock sensor with its interfacing with microcontroller is described.

In section 2.9, Light sensor with its interfacing with microcontroller is described.

In section 2.10, GPS module with its interfacing with microcontroller is described.

In section 2.11, process of PCB designing of all three boards: TAG - PROVINO board, RECEIVER EXTERNO board and CASSAFORMA board are described.

2.1 Wake up generator:

Low power consumption is the key constrain in our project as the TAG board will be placed inside the material box. That is why, it is decided that TAG board will be operating in deep-sleep mode when not performing some tasks, which is majority of its lifetime. To collect

the data from the TAG board, we will need to wake up TAG from deep sleep mode using external interrupt signal from outside the material box.

For this particular purpose, To wake up TAG board from deep sleep mode "wake up receiver" sub-module is added, which is similar to passive antenna. When it receives an external wireless signal antenna generates an interrupt on the wake up pin of microcontroller which wakes up the whole TAG board from deep sleep and enable it to perform required task.

In order to generate a signal to wake up TAG board from outside the material box, sub-module "Wake up generator" is added in RECEIVER EXTERNO board. Wake up generator operates in such a way that, when a particular designated switch is pressed in RECEIVER EXTERNO board it generates wire-less EM signal above 1 MHz of frequency with enough power to activate antenna in wake up receiver to generate interrupt at wake up pin in the TAG board.

To design wake up generator which produces wireless EM sinusoidal signal which oscillates at/of the frequency of approximate 1 MHz, there are several possible options of oscillator like RC phase shift oscillator, Hartley oscillator, Colpitts oscillator and many more. Out of all the oscillator, we decided to use Colpitts oscillator because of following reasons:

1. It generates sinusoidal signals of very high frequencies.
2. Stability of frequency is higher at both high and low temperatures.
3. Frequency can be varied by using both the variable capacitors.
4. Less number of components are sufficient, hence it is cost efficient.
5. The amplitude of the output remains constant over a fixed frequency range.
6. It consist of two capacitor and one inductance, out of which one inductance can be used as antenna to not only to generate but transmit our signal.

The circuit diagram of Colpitt's oscillator using BJT is shown in Figure 2.1. It consists of an R-C coupled amplifier using an n-p-n transistor in common emitter configuration. R1 and R2 are two resistors which form a voltage divider bias to the transistor. The Colpitts LC tank circuit consists of a single inductor and two capacitors. A resistor R3 is connected in the circuit which stabilizes the circuit against temperature variations. The coupling capacitor C5 blocks dc and provides an ac path from the collector to the tank circuit.

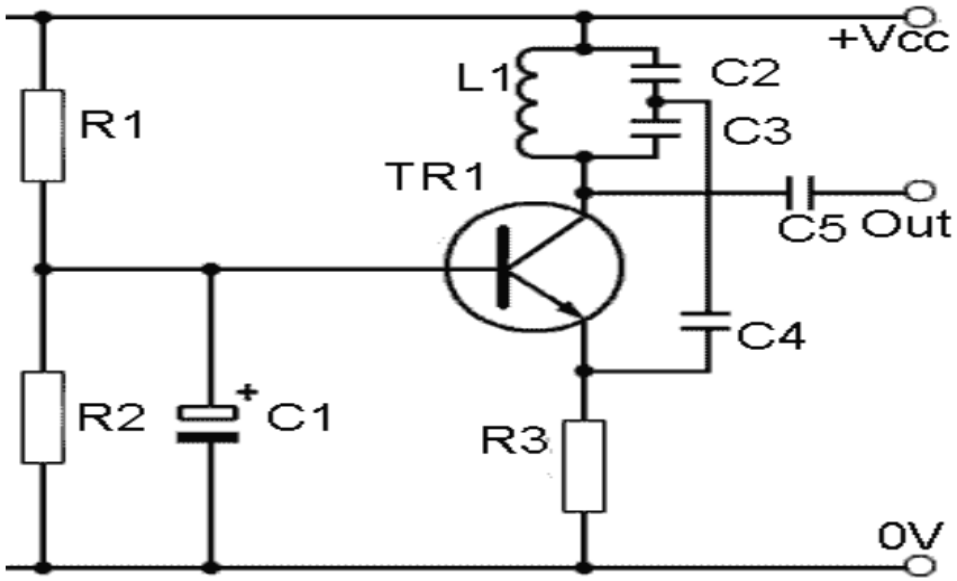


Figure 2.1: Colpitt's Oscillator

The feedback network (tank circuit) consists of two capacitors C2 and C3 (in series) which placed across a common inductor L1. The centre of the two capacitors is tapped and connected across collector. The feedback network (C2, C3 and L1) determines the frequency of oscillation of the oscillator. The two series capacitors C2, and C3 form the potential divider led for providing the feedback voltage.

Whenever power supply is switched on, the capacitors C2 and C3 start charging and after the capacitors get fully charged, the capacitors starts discharging through the inductor L1 in the circuit causing damped harmonic oscillations in the tank circuit. Thus, an AC voltage is produced across C1 & C2 by the oscillatory current in the tank circuit. While these capacitors get fully discharged, the electrostatic energy stored in the capacitors get transferred in the form of magnetic flux to the inductor and thus inductor gets charged.

Similarly, when the inductor starts discharging, the capacitors start charging again and this process of energy charging and discharging capacitors and inductor continues causing the generation of oscillations and the frequency of these oscillations can be determined by using the resonant frequency of the tank circuit consisting of inductor and capacitors. This tank circuit is considered as the energy reservoir or energy storage. This is because of frequent energy charging and discharging of the inductor, capacitors that is a part of LC network forming the tank circuit.

The resonant frequency is given by:

$$f_0 = 1/(2\Pi\sqrt{(L1*C)})$$

$$C = (C2*C3)/((C2+C3))$$

Where f_0 is the resonant frequency

L1 represents the self inductance of the coil.

C is the equivalent capacitance of series combination of C2 and C3 of the tank circuit

First, we simulated the colpitts circuit and measure its output which can be seen in figure 2.2 and figure 2.3 respectively.

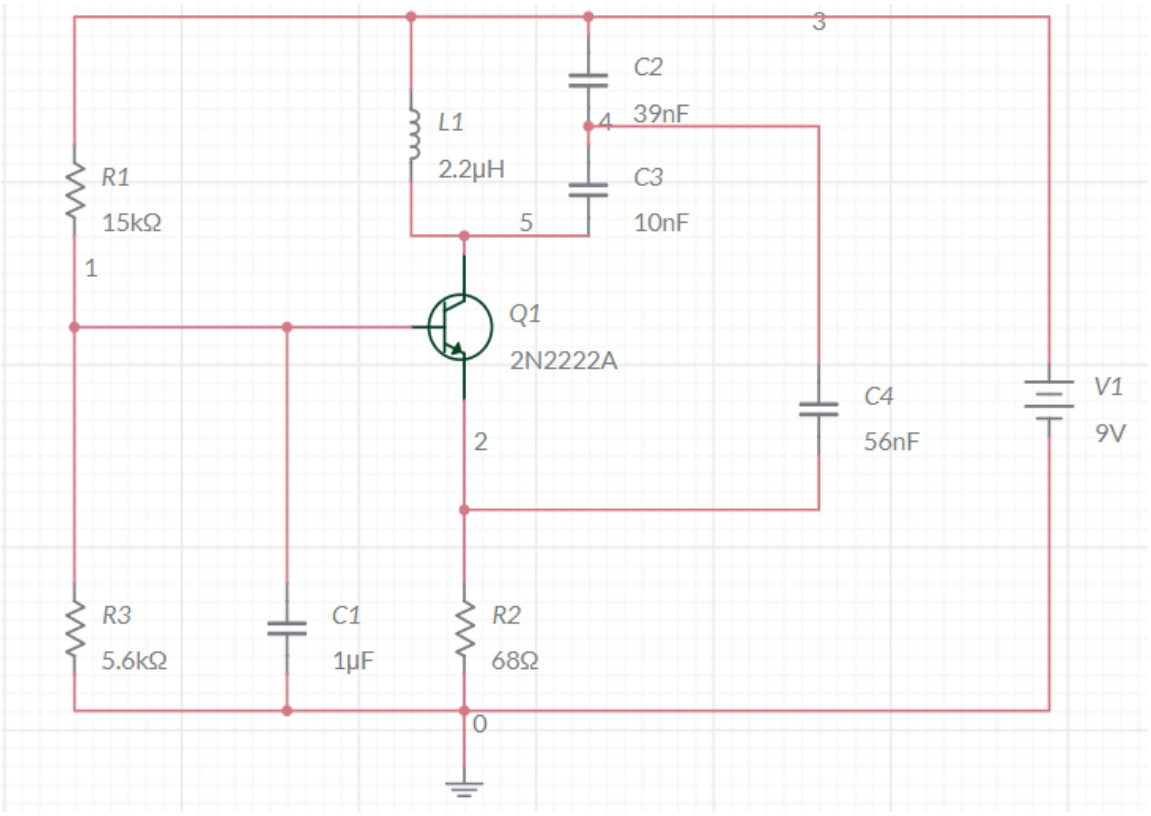


Figure 2.2: Transmitter Circuit Simulation

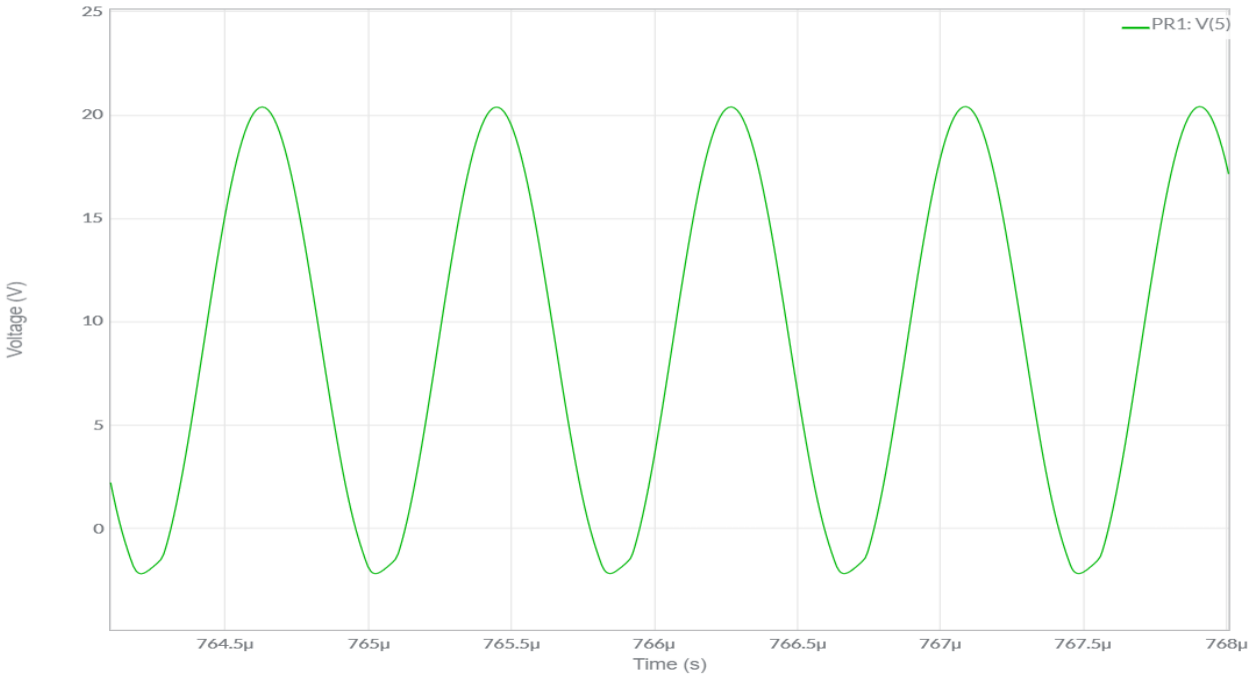


Figure 2.3: Transmitter Simulation Output

Simulation provided us with the output as sinusoidal wave with the frequencies of 1.2 MHz

After the simulation, we made bread board prototype of same circuit which can be seen in figure 2.4 and its output in figure 2.5.

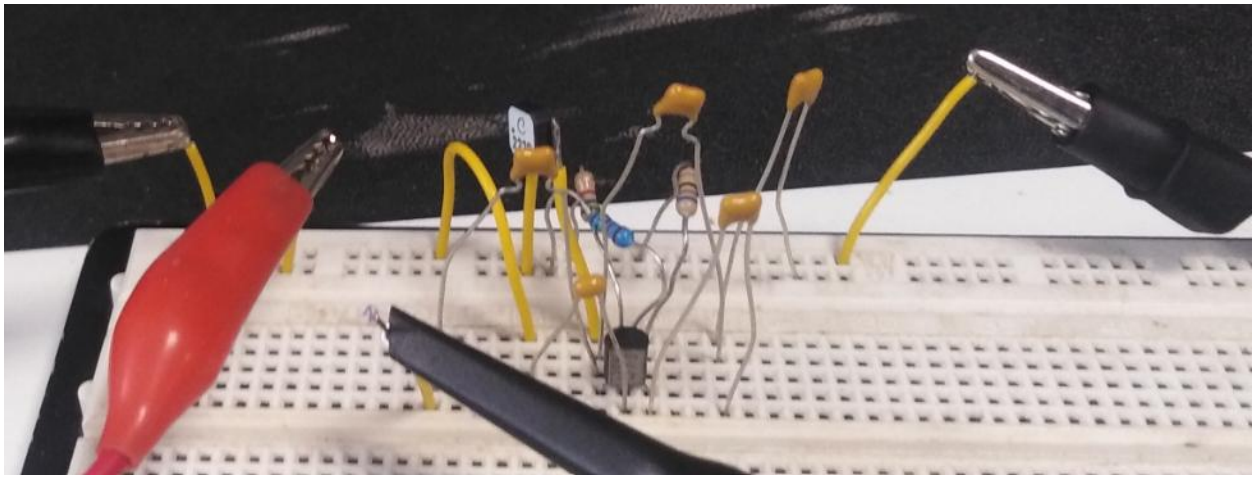


Figure 2.4: Transmitter Prototype Circuit

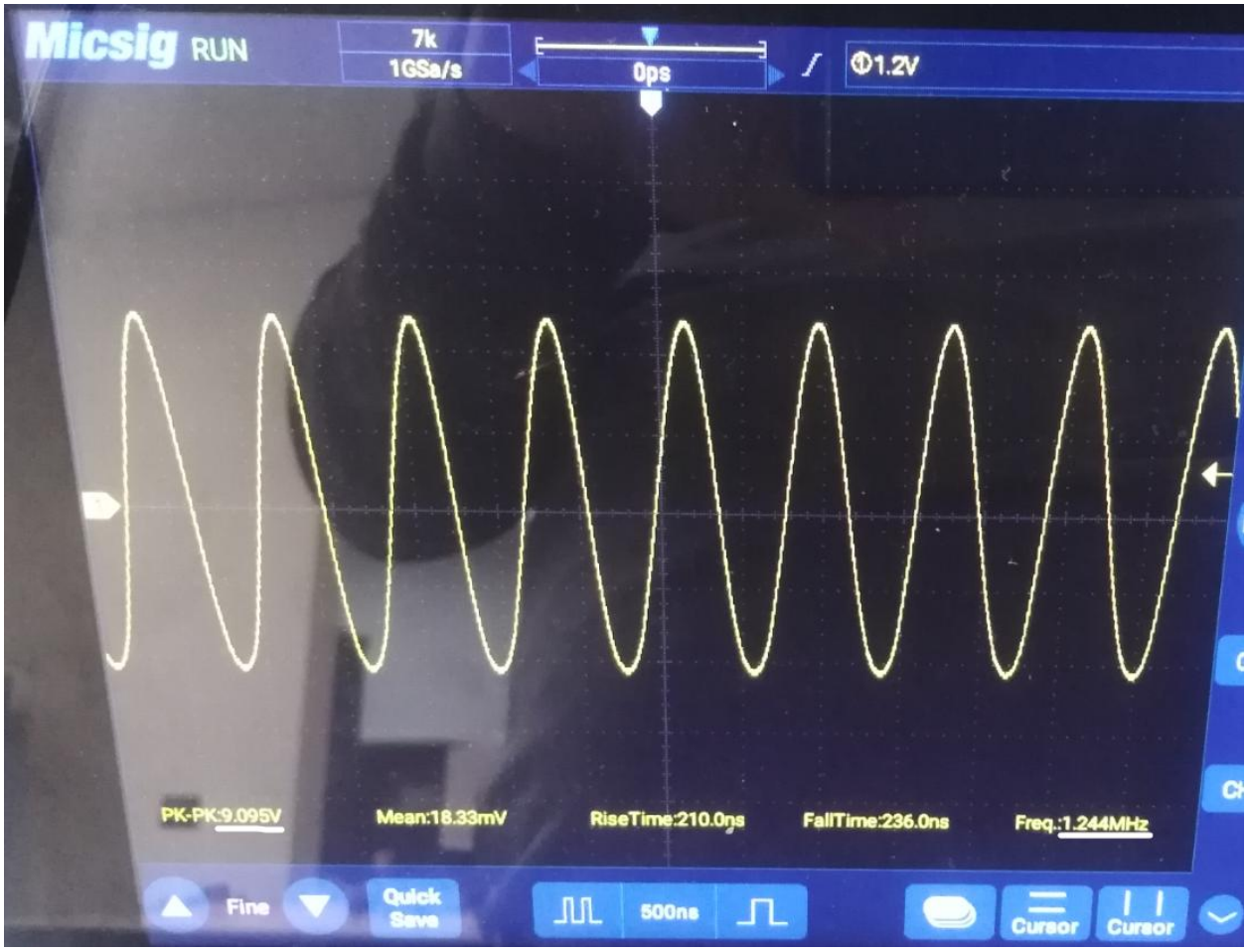


Figure 2.5: Output Wave

bread board prototype provided us with the output as sinusoidal wave with the frequencies of 1.2 MHz with 9 volts peak to peak.

2.2 wake up receiver:

To wake up TAG board from deep sleep mode "wake up receiver" sub-module is added, which is similar to passive antenna. When it receives an external wireless signal antenna generates an interrupt on the wake up pin of microcontroller which wakes up the whole TAG board from deep sleep and enables it to perform the required task.
In the wake up receiver circuit, we have used simple LC circuit to receive external signal and generate interrupt at the wake up pin of the microcontroller.

LC circuit can be seen in figure 2.6:

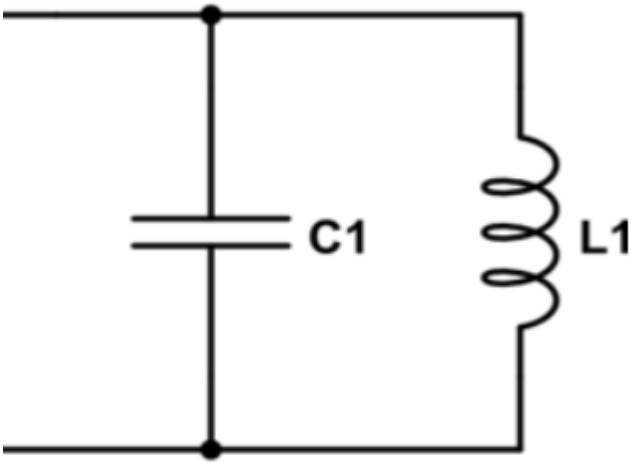


Figure 2.6: Wake up receiver circuit

After matching LC, below circuit act as an AM envelope detector. Zener diode limits the voltage flows in the circuit.
Simulation in figure 2.7 and output in figure 2.8.

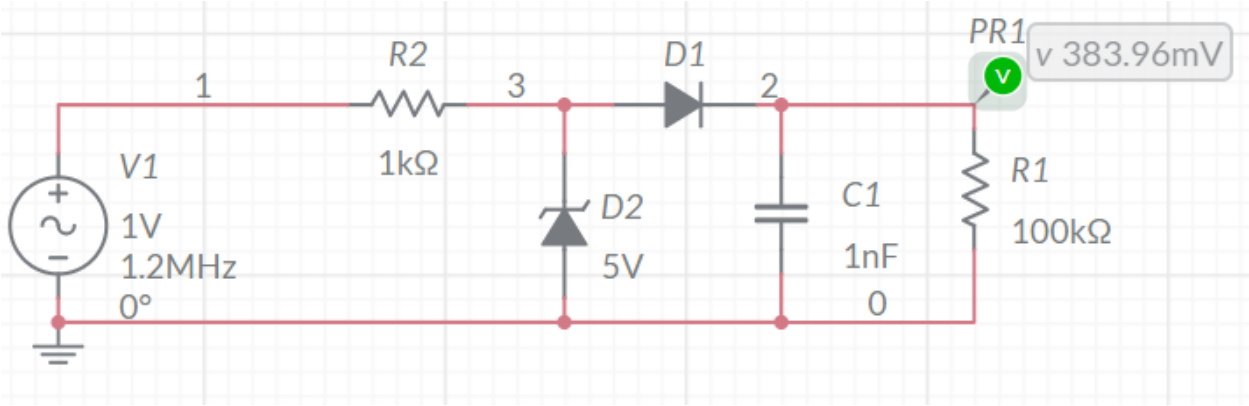


Figure 2.7: Receiver Circuit Simulation

In below figure, we can see voltage across R1:

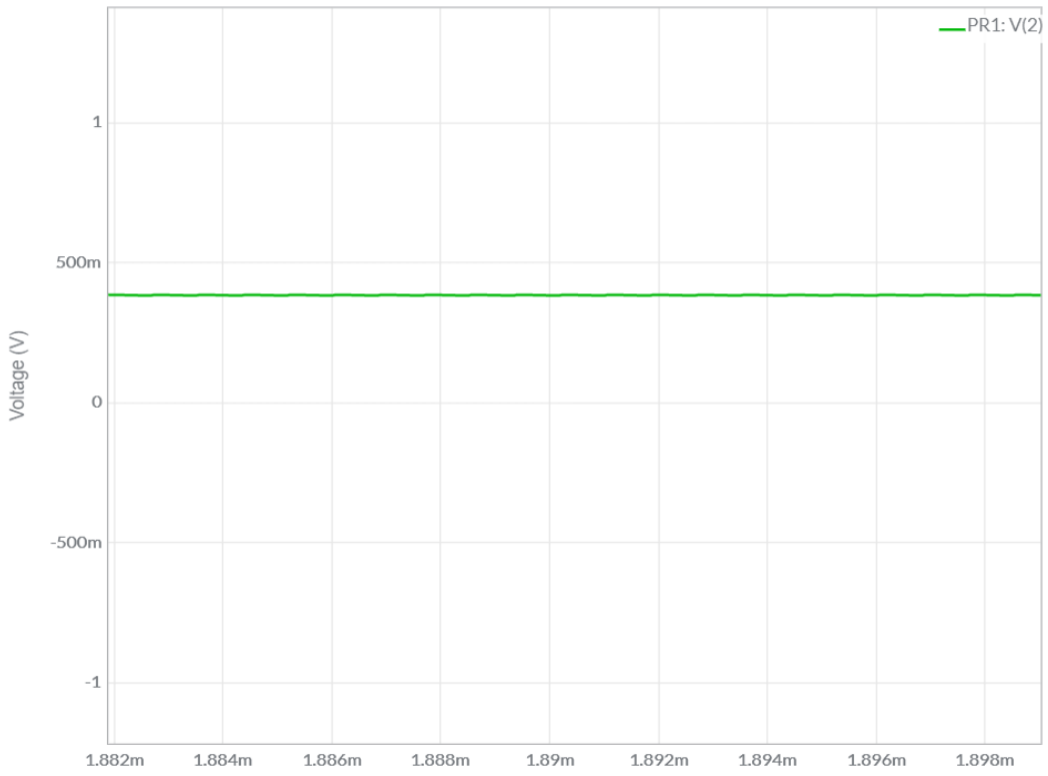


Figure 2.8: Receiver Circuit Simulation Output

2.3 Prototype testing:

2.3.1 Experiment 1 :

To verify our prototype, we have generated EM signal of 1.24 MHz using Colpitts oscillator and noted voltage at wake up receiver circuit by varying distance between transmitter and receiver shown in Figure 2.9.

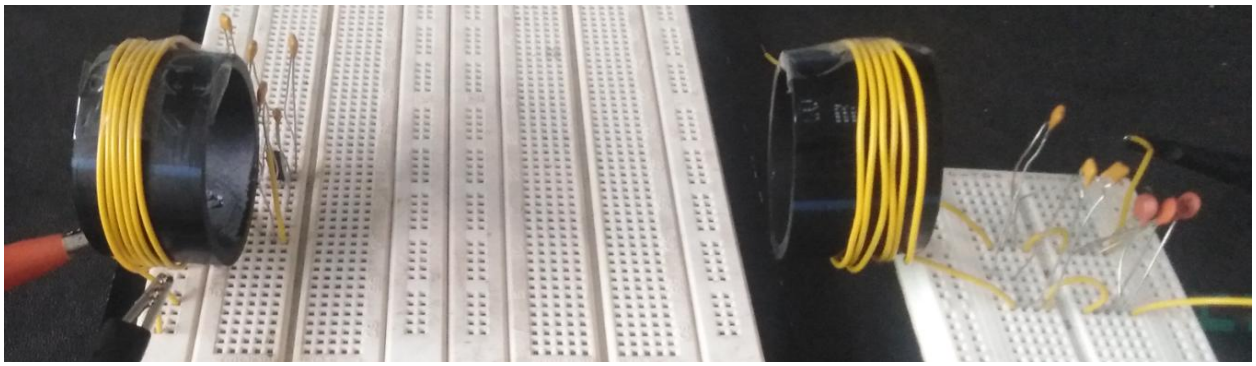


Figure 2.9: Transmitter and receiver Prototype Circuit

Results are shown in Table 1:

Table 2.1 Voltage received at TAG at different distance between TAG and OUTSIDER

S. N.	Distance(cm)	Voltage(Vpk-pk)
1	2.2	9.2
2	3	8.8
3	4	8.2
4	5	6.9
5	7	4.8
6	8.5	3.2
7	10	2.4
8	12	1.4
9	15	0.98
10	17	0.65
11	19	0.49
12	21	0.32
13	23	0.32
14	25	0.24
15	28	0.16
16	30	0.16

2.3.2 Experiment 2 :

To check the attenuation of signal at receiving end by using cement brick as medium in between transmitter and receiver, we have generated EM signal of 1.24 MHz using Colpitts oscillator and noted voltage at wake up receiver circuit by varying distance between transmitter and receiver without and with placing brick shown in Figure 2.10 and in Figure 2.10 respectively.

Figure 2.10: TAG and OUTSIDER without brick

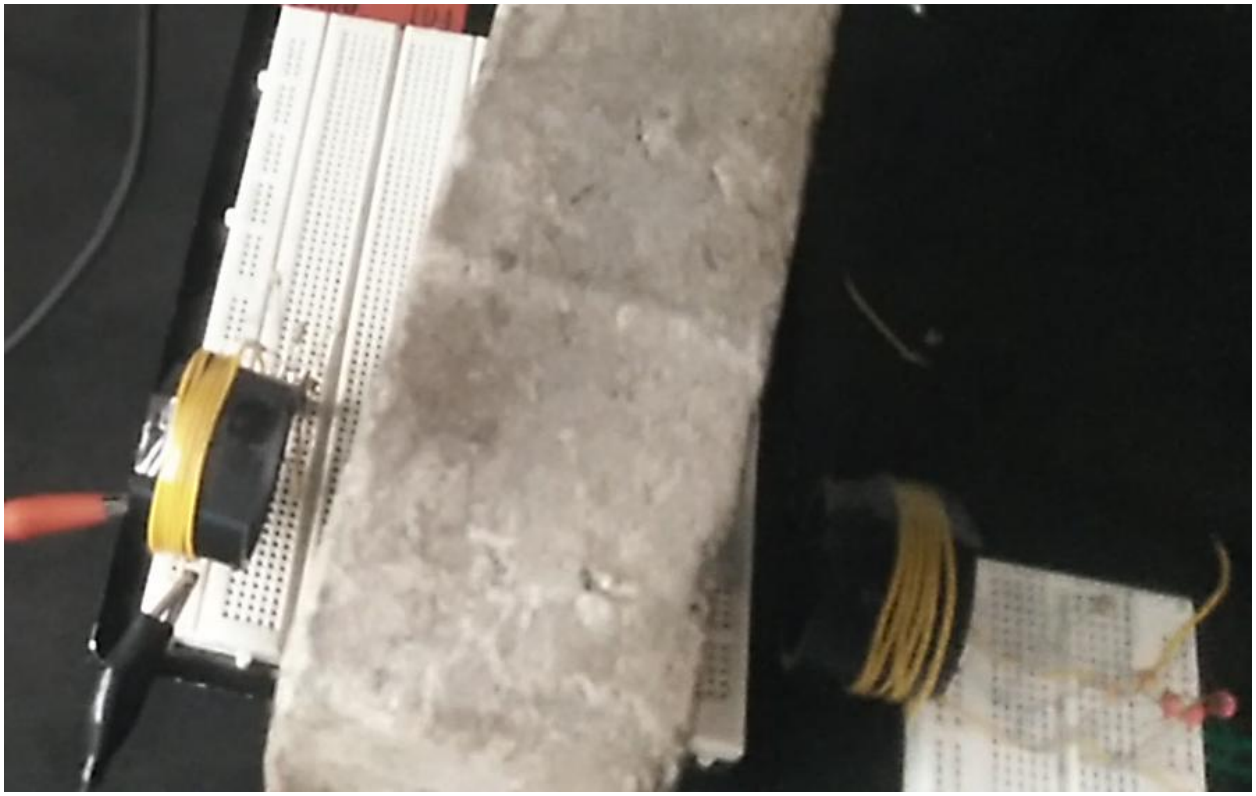


Figure 2.11: TAG and OUTSIDER with brick

Table 2.2 Voltage received at TAG at different distance between TAG and OUTSIDER directly and using cement brick

S.N.	Distance(cm)	Voltage(direct)(Vpk-pk)	Voltage(Brick)(Vpk-pk)
1	10.5	2.2	2.2
2	15.5	0.78	0.78
3	13	1.3	1.3

2.3.2 Outcome of experiments

After performing wireless voltage transfer experiments at varying distance and obtaining above result, we can confirm that cement brick does not produce any attenuation at resonant frequency of 1.2 MHz.

2.4 PCB design of Wake up generator and receiver:

After being satisfied by the results of the above experiments, we decided to build Prototype PCB of wake up generator and wake up receiver.

PCB was designed by using OrCAD's PCB Editor software. Schematic was designed by using OrCAD's Allegro software.

Schematics are following:

1. Schematic of Wake up generator aka Colpitts oscillator shown in Figure 2.12:

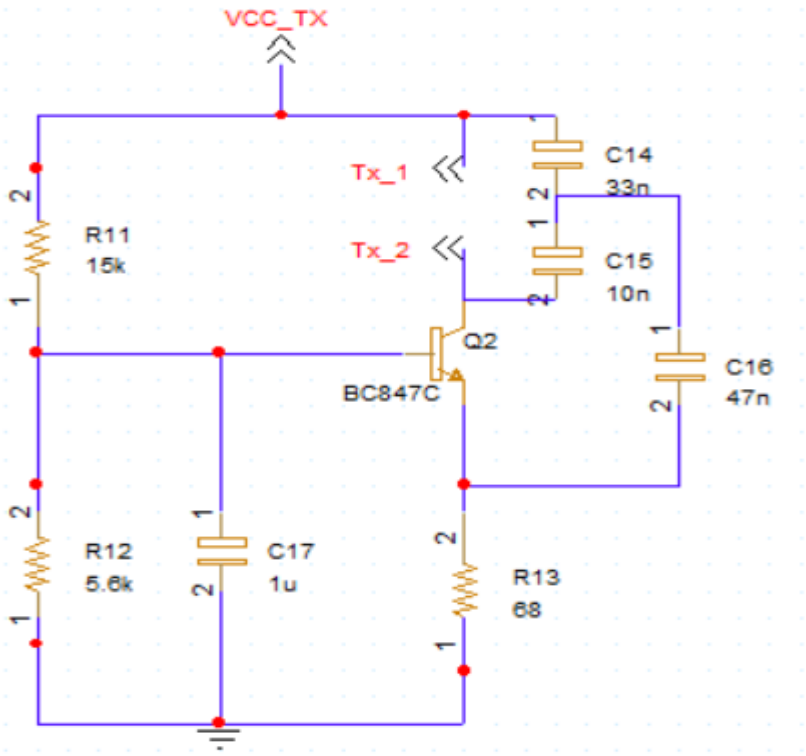


Figure 2.12: Colpitt's Oscillator

2. Schematic of Wake up receiver aka LC circuit shown in Figure 2.13:

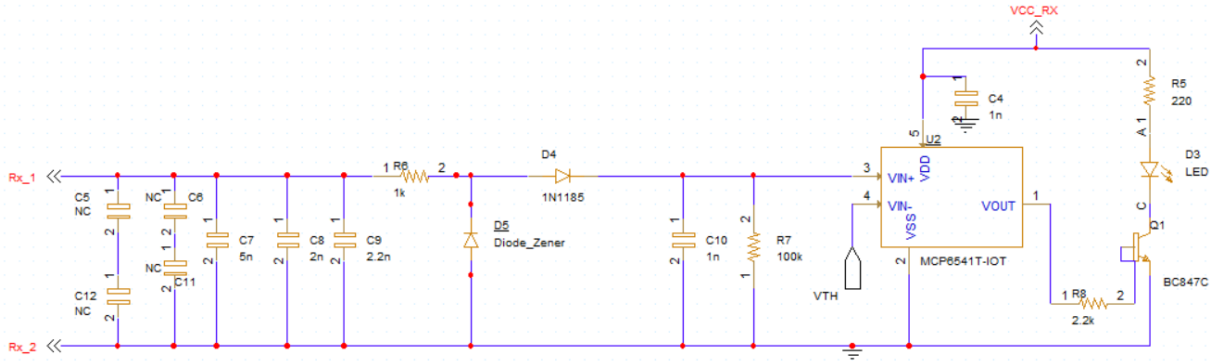


Figure 2.13: LC circuit

3. Schematic of power supply for Wake up generator and Wake up receiver shown in Figure 2.14:

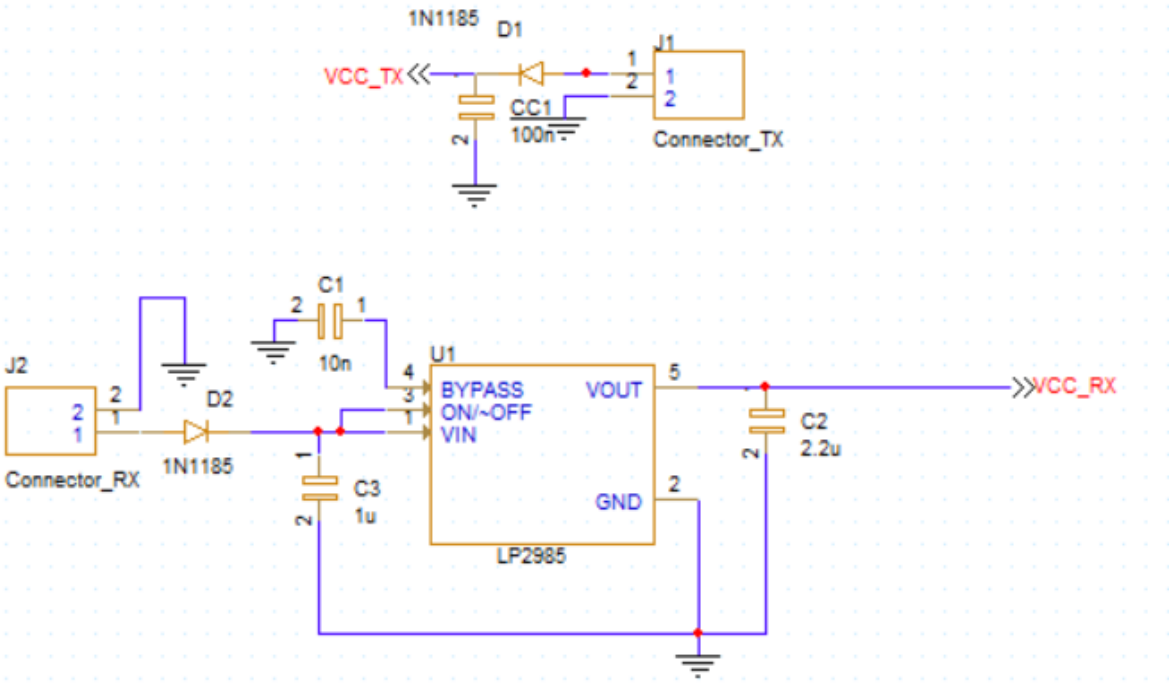


Figure 2.14: Power Supply of wake up generator circuit

To save the cost, prototype PCB was designed in such a way that wake up generator was on one side of PCB and wake up receiver was on the another side. We ordered several PCBs so that one PCB can be used as wake up generator and another as wake up receiver.
PCB design can be seen in following Figures 2.15, 2.16 and 2.17:

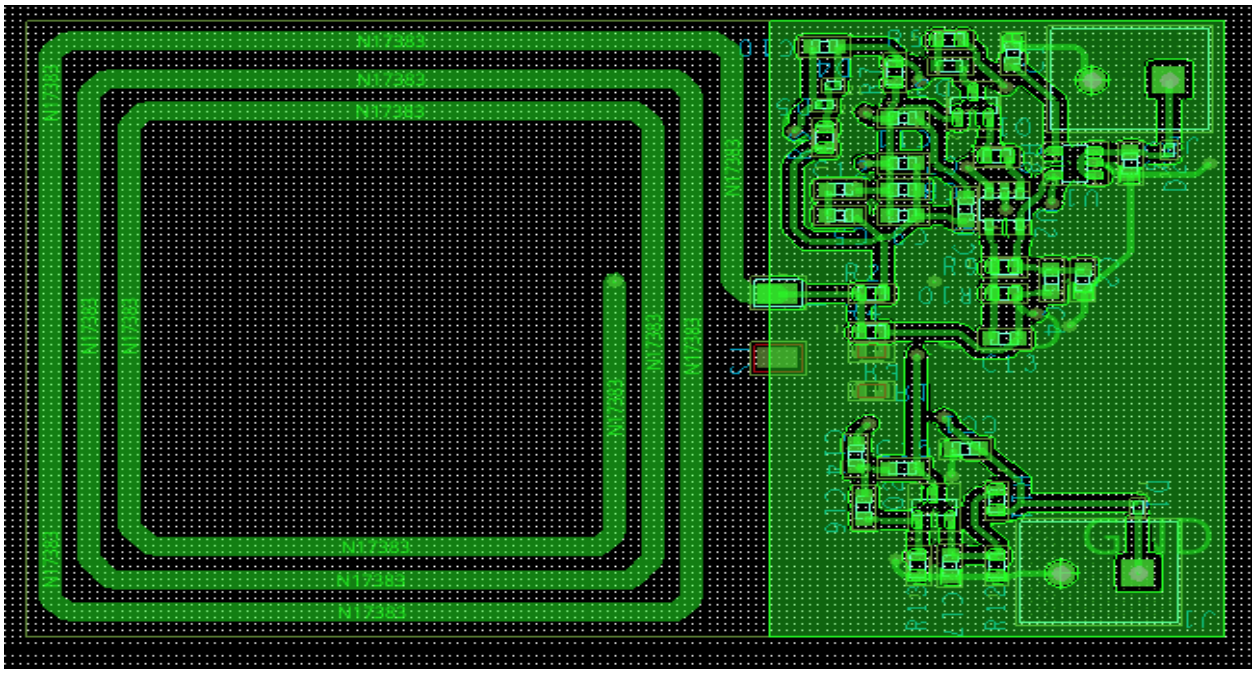


Figure 2.15: Top layer of wake up generator circuit

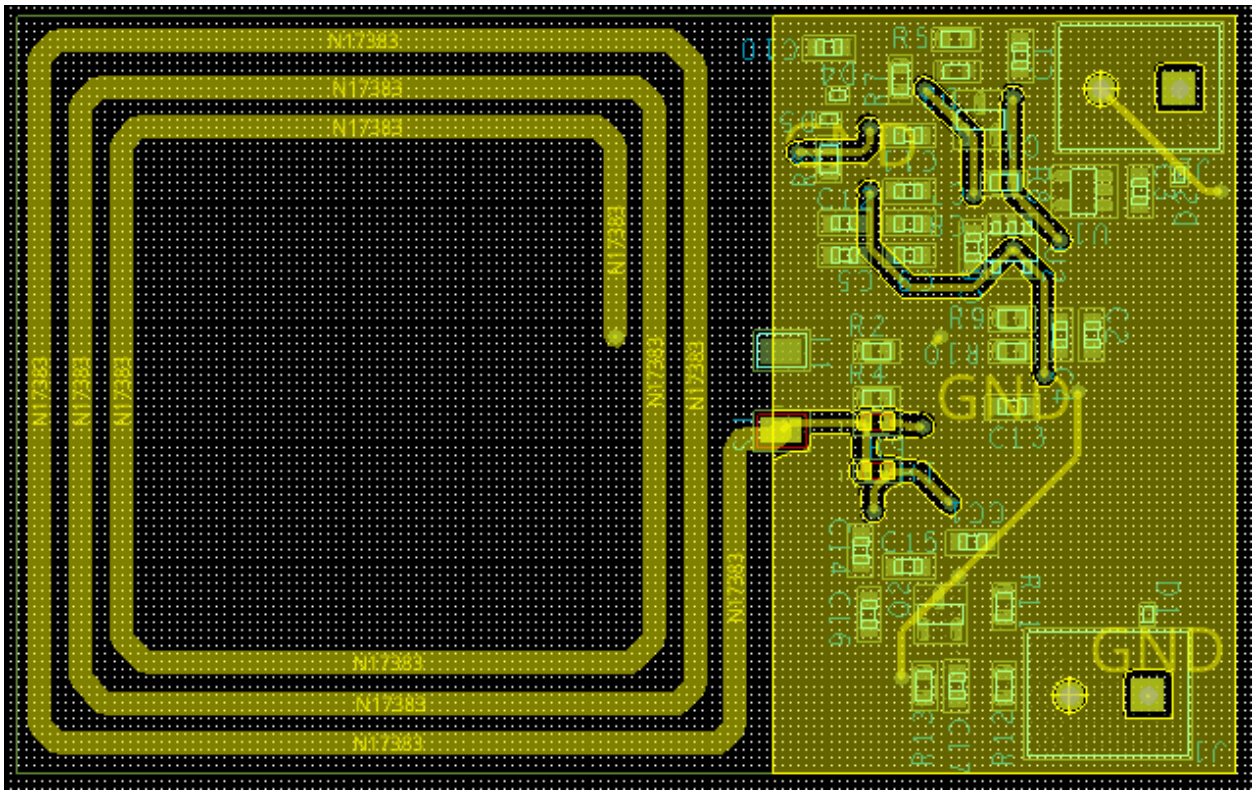


Figure 2.16: Bottom layer of wake up generator circuit

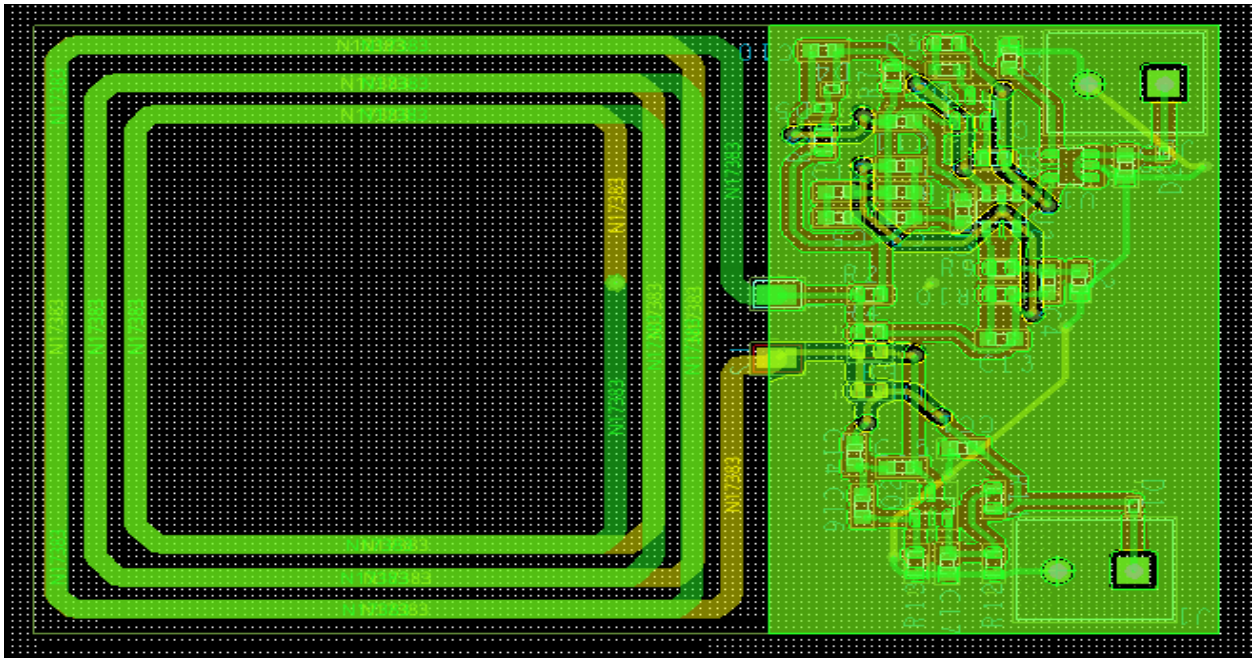


Figure 2.17: Complete Layout of wake up generator circuit

For extensive testing purpose, we have made two version of same PCB:

The PCB with bigger inductance, whose inductance dimensions are 45x40mm called Grande and another one whose inductance dimensions are 33x28mm called Piccolo shown in Figure 2.18:

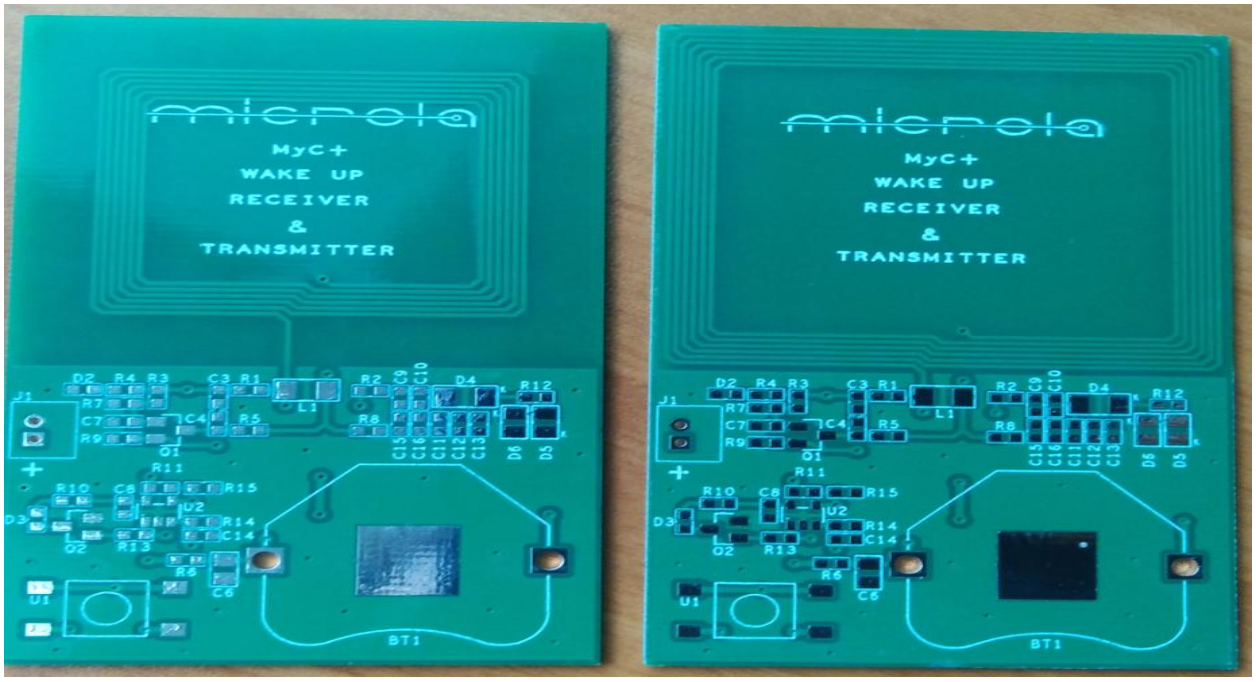


Figure 2.18: (a) Piccolo, (b) Grande

In the grande to grande communication, we found resonance at 39nF in series with 10nF and maximum distance of between Tx and Rx is 10cm shown in Table 3.

In grande to piccolo communication, we found resonance at 10nF(~9) in parallel with 2.2nF(~2) & 470pF, which is equivalent of 12.6nF(~11.4) and maximum distance of between Tx and Rx is 7.5cm.

Table 2.3 Maximum distance between Communication of varies configuration of Grande and Piccolo

S. No.	TX	RX	Max Distance(cm)
1	P	P	7
2	P	G	5
3	G	P	7.5
4	G	G	10

2.5 PCB Prototype testing:

we performed following experiments:

2.5.1. Direct EM transfer

we have performed TX to RX communication shown in Figure 2.19 and checked what is maximum distance possible between grande and piccolo, which is 7.5 cm.

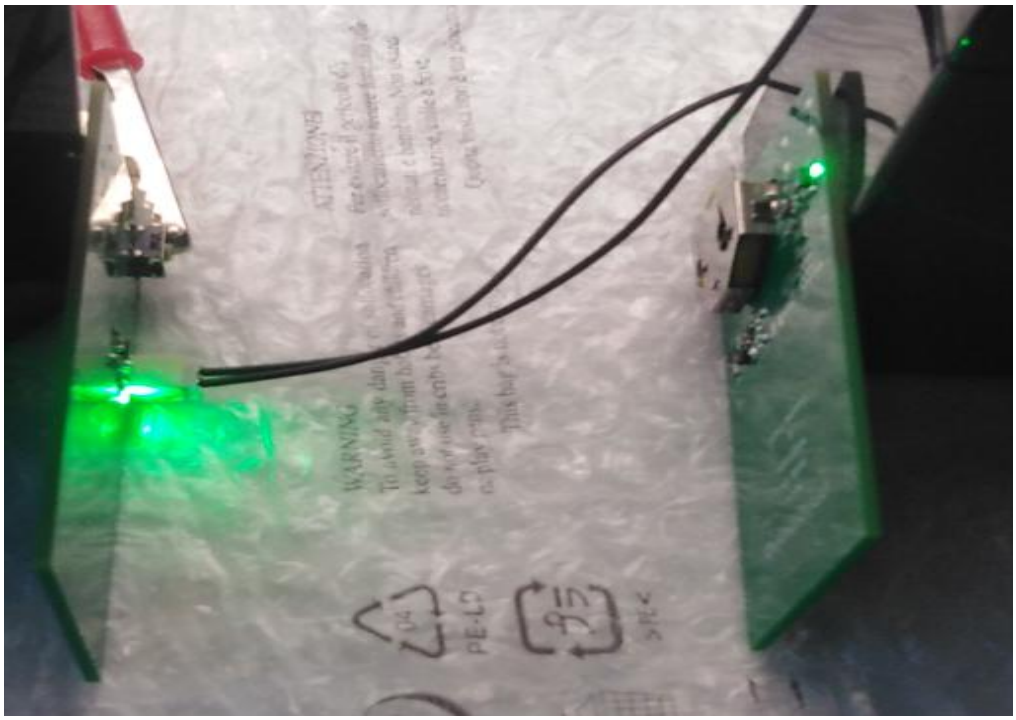


Figure 2.19: Grande and Piccolo communication

2.5.2. Dry Brick

we have performed the same experiment as above but this time placing dry brick between TX and RX to check if there is any attenuation shown in Figure 2.20:

Figure 2.20: Grande and Piccolo communication with Dry Brick in-between

Outcome: No attenuation

2.5.3. Wet Brick

we have performed the same experiment as above but this time placing wet brick between TX and RX to check if there is any attenuation shown in Figure 2.21:

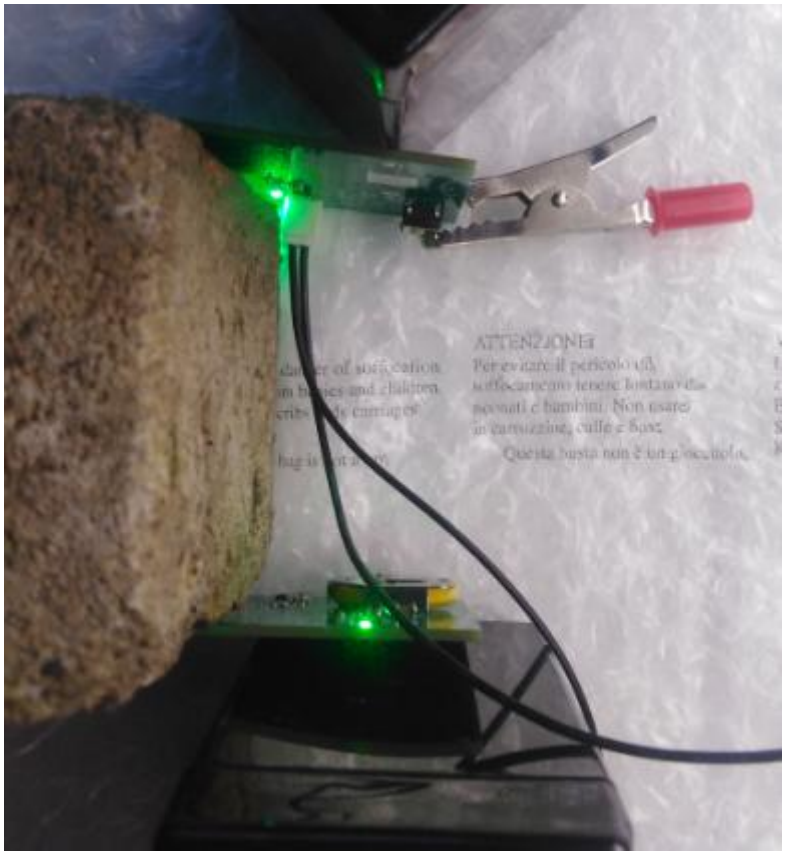


Figure 2.21: Grande and Piccolo communication with Wet Brick in-between

Outcome: No attenuation

2.5.4. Outcome of experiments

Outcome of our experiments is that we did not get any attenuation between TX and RX by either dry or wet brick. at given distance and frequency above.

2.6 Choice of Microcontroller:

For the reliability of any project, choice of component is critically important. Microcontroller is brain of the project. It is responsible for correctly executing all the task required. Out of several choices, we initially chose STM32WB55RGv6 because of several reasons:

1. It is dual-core processor, one core for regular task and another core specifically designed for communication purpose.

2. It is possible to put it in ultra low power mode which consume power in the range of nano-Amps.
3. It supports several communication protocols like Bluetooth LE 5.0, zigbee as well as IEEE 802.15.4.
4. It has internal analog temperature sensor, due to which we don't have to put external temperature sensor which save area as well as cost.
5. ST microelectronics provide rich development environment and community support.
6. Nucleo board which consist of same microcontroller, which helps in testing with interfacing with all other modules.

Nucleo board can be seen below in Figure 2.22:

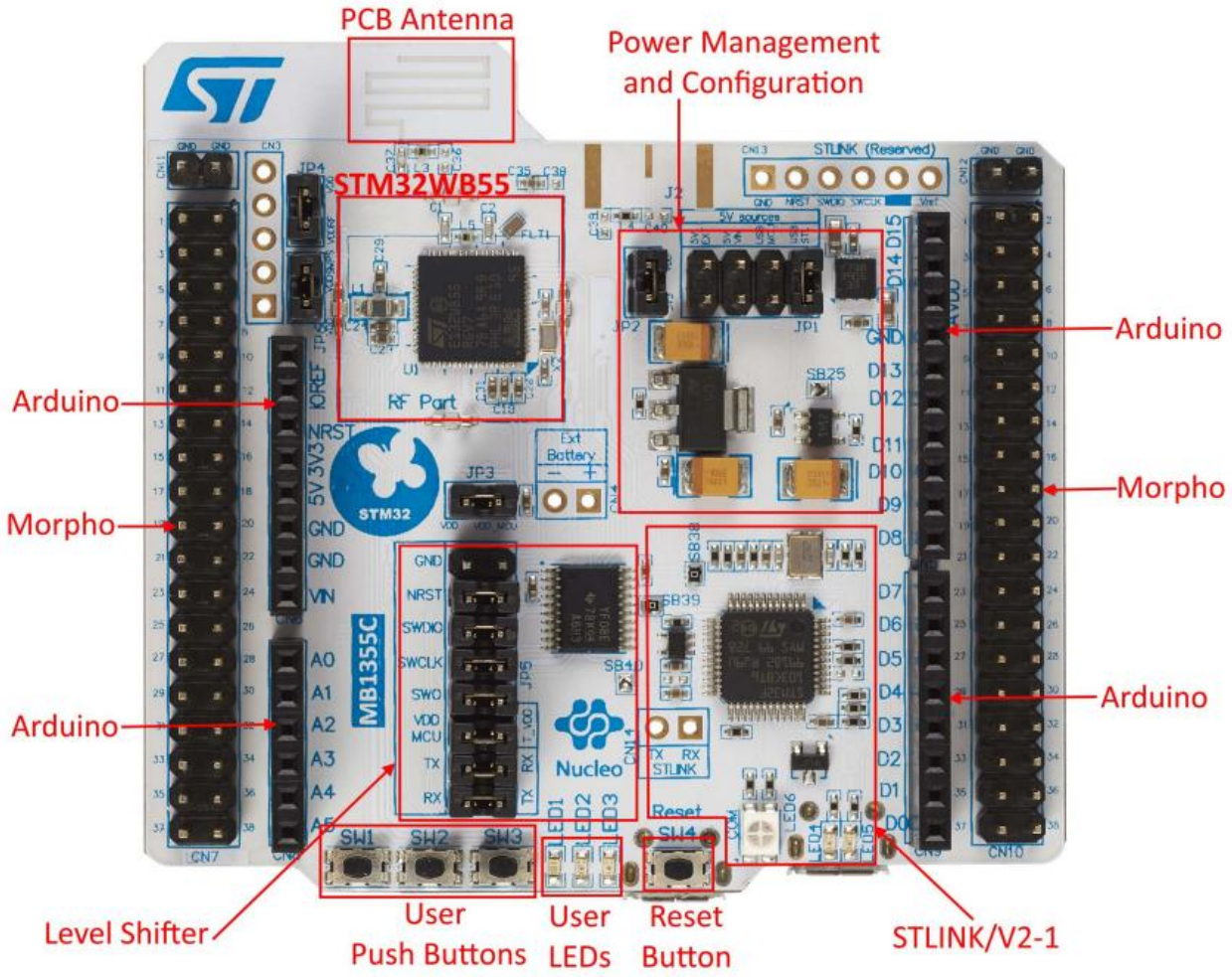


Figure 2.22: STM32WB55 Nucleo Board

Schematic of custom STM32WB55RGv6 board are shown in Figure 2.23, 2.24, 2.25, 2.26 and 2.27:

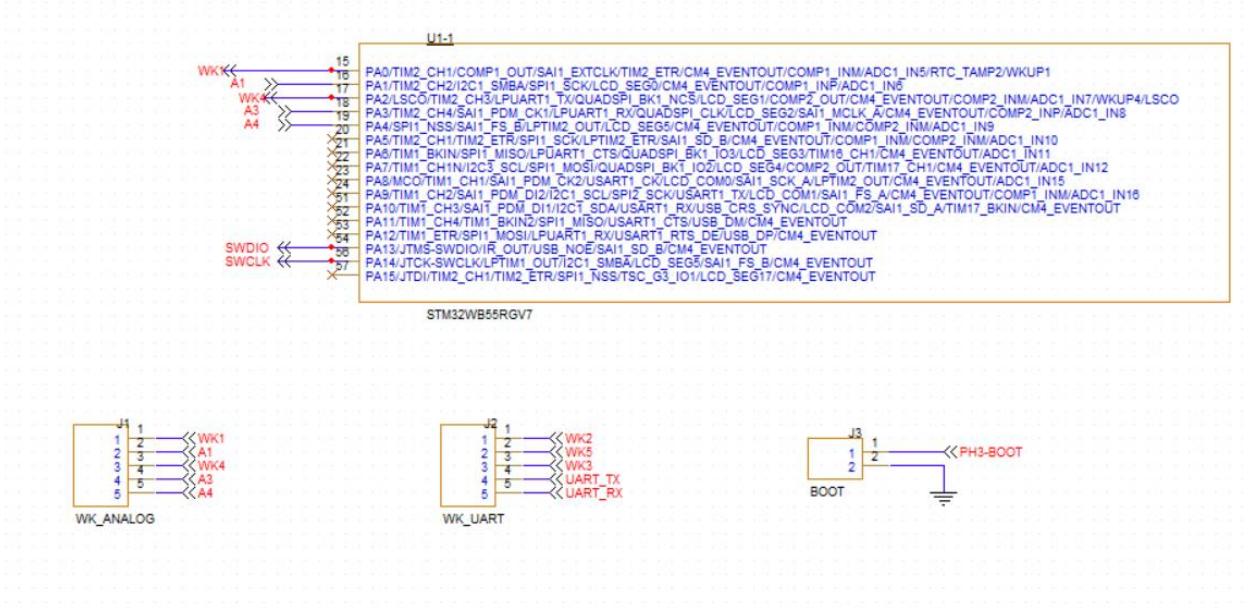


Figure 2.23: Port A of custom board

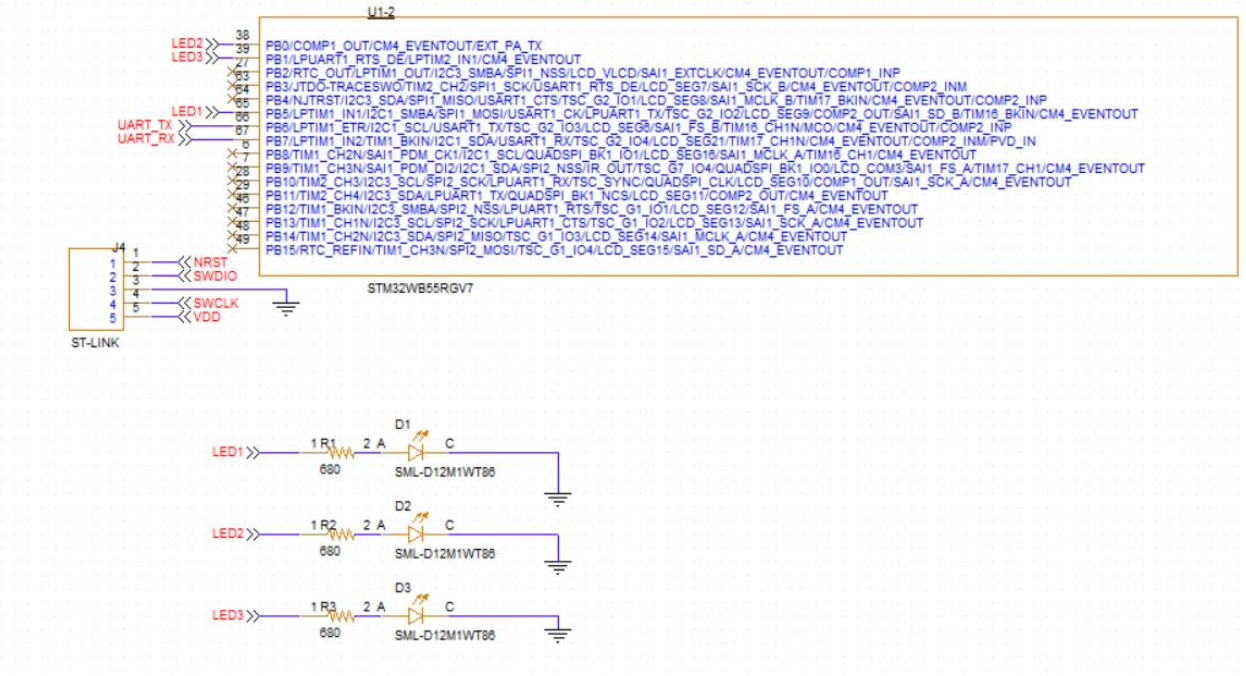


Figure 2.24: Port B of custom board

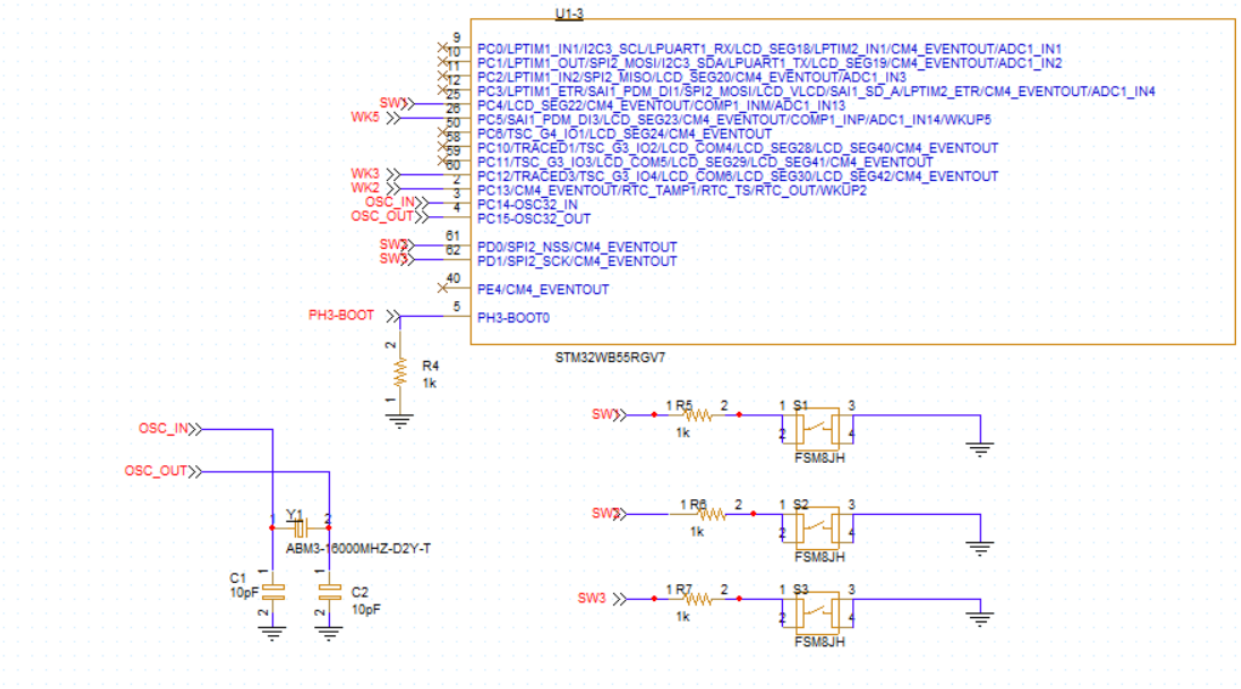


Figure 2.25: Port C, D and E of custom board

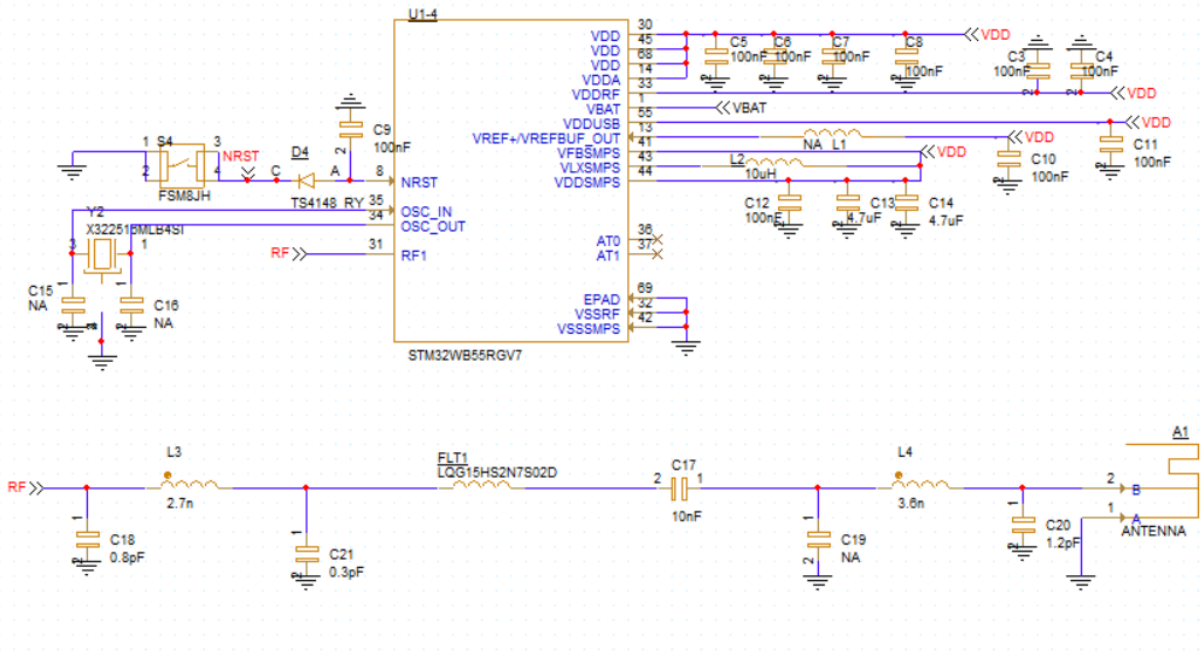


Figure 2.26: Power and Ground Pins of custom board

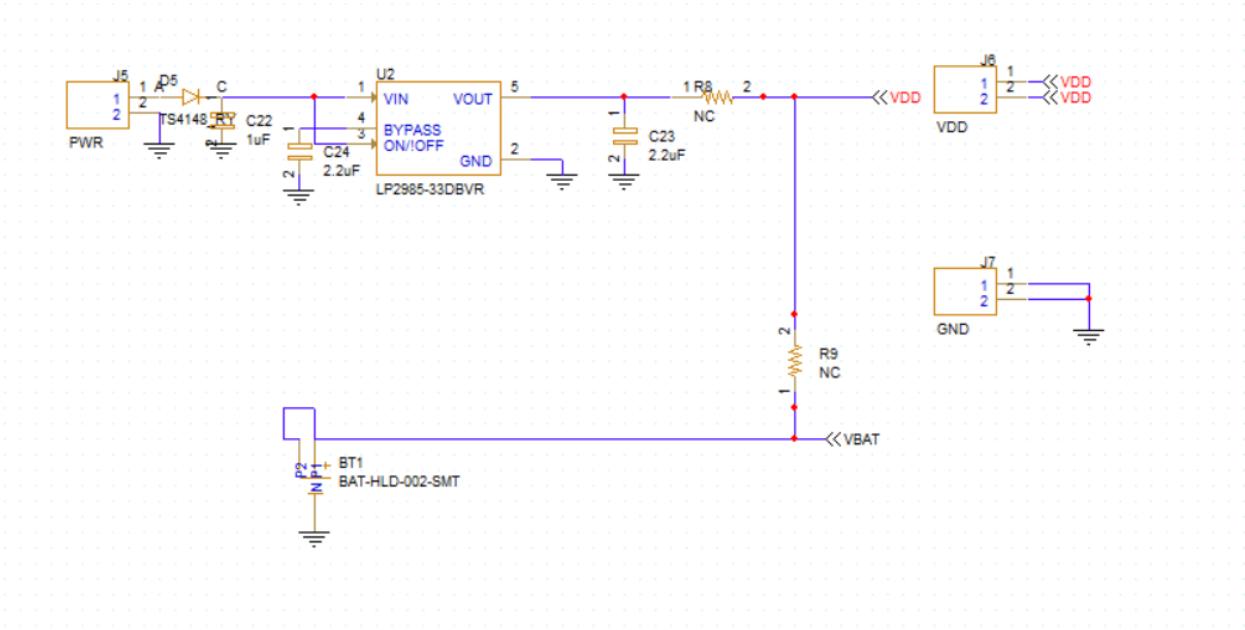


Figure 2.27: Power Supply and Battery Holder of custom board

PCB design can be seen below shown in Figure 2.28, 2.29 and 2.30:

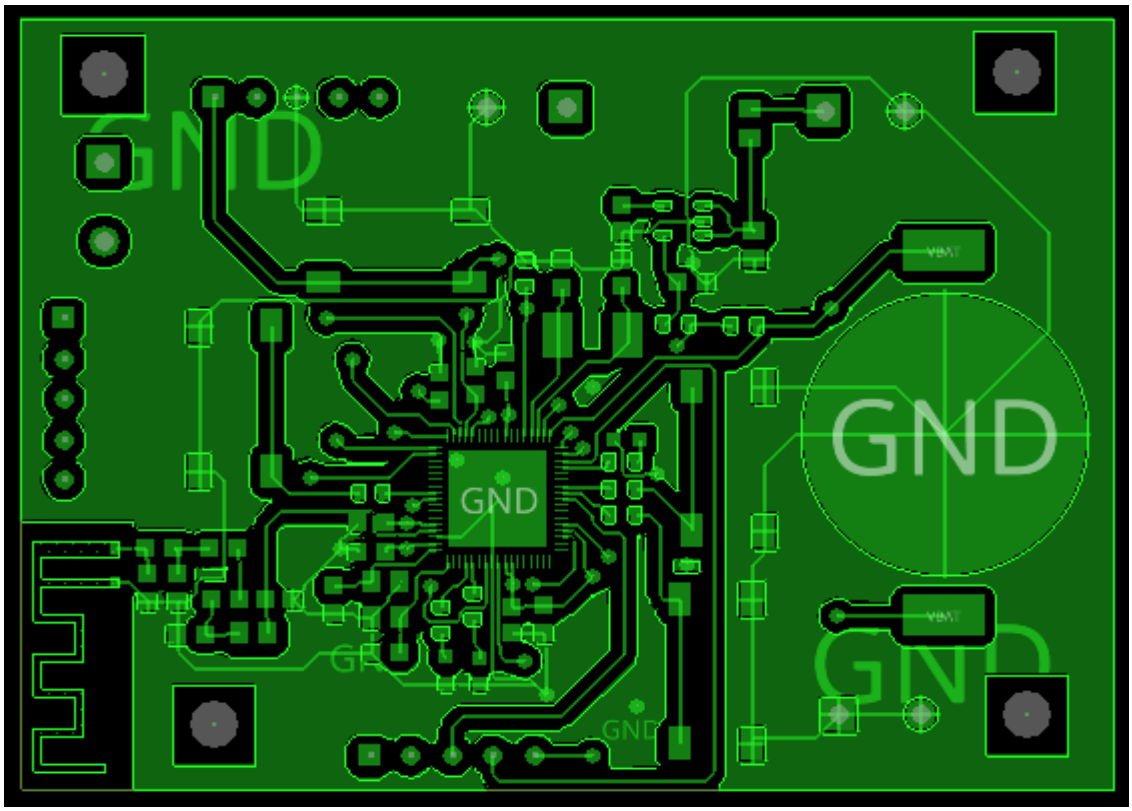


Figure 2.28: TOP Layer of custom board

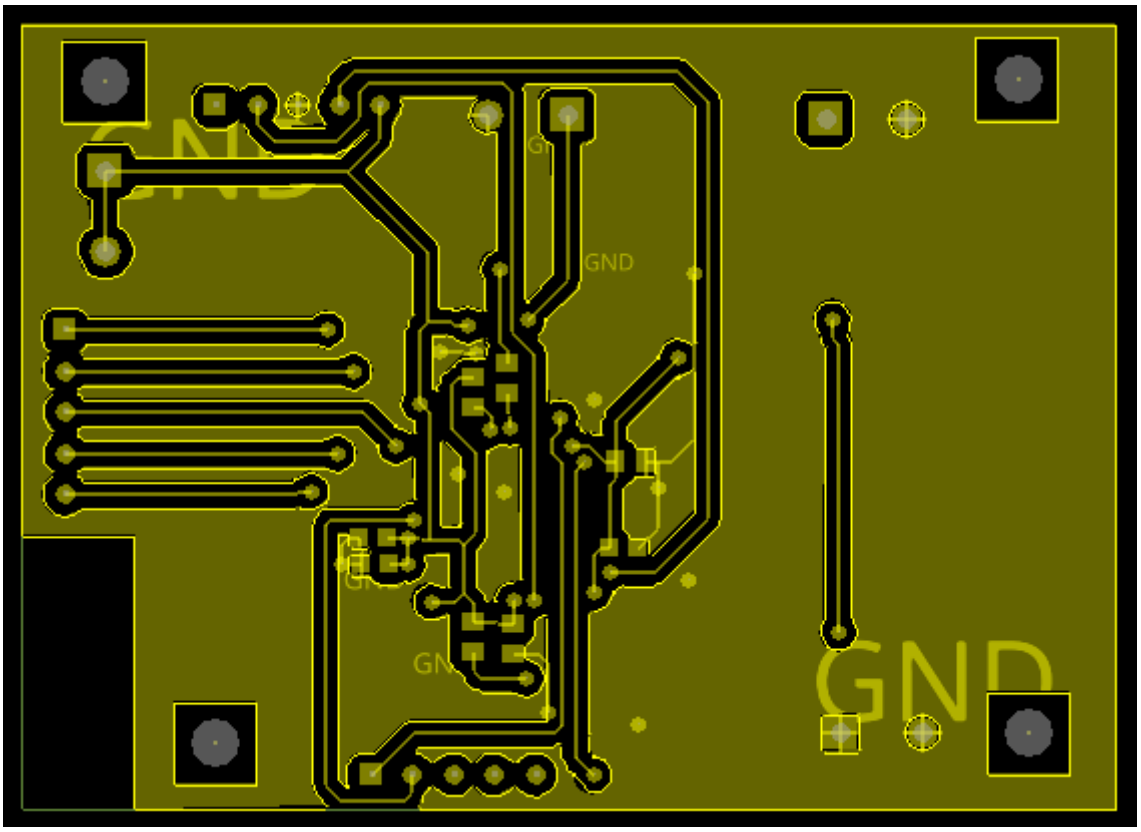


Figure 2.29: BOTTOM Layer of custom board

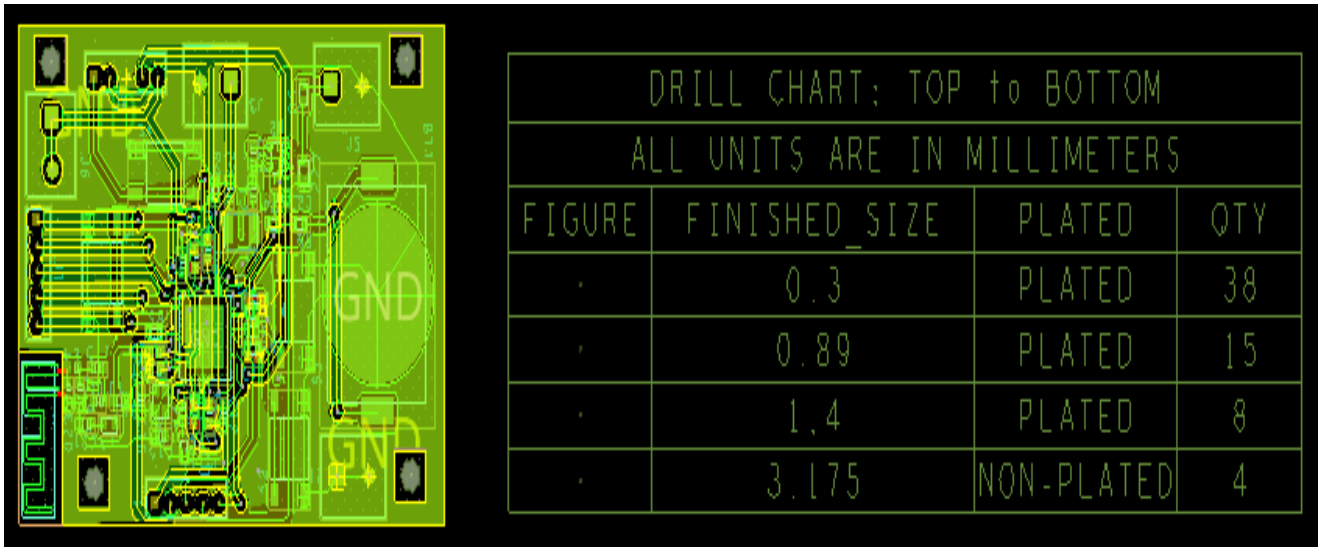


Figure 2.30: Complete view with Drill Table of custom board

Our custom PBC is shown in Figure 2.31.

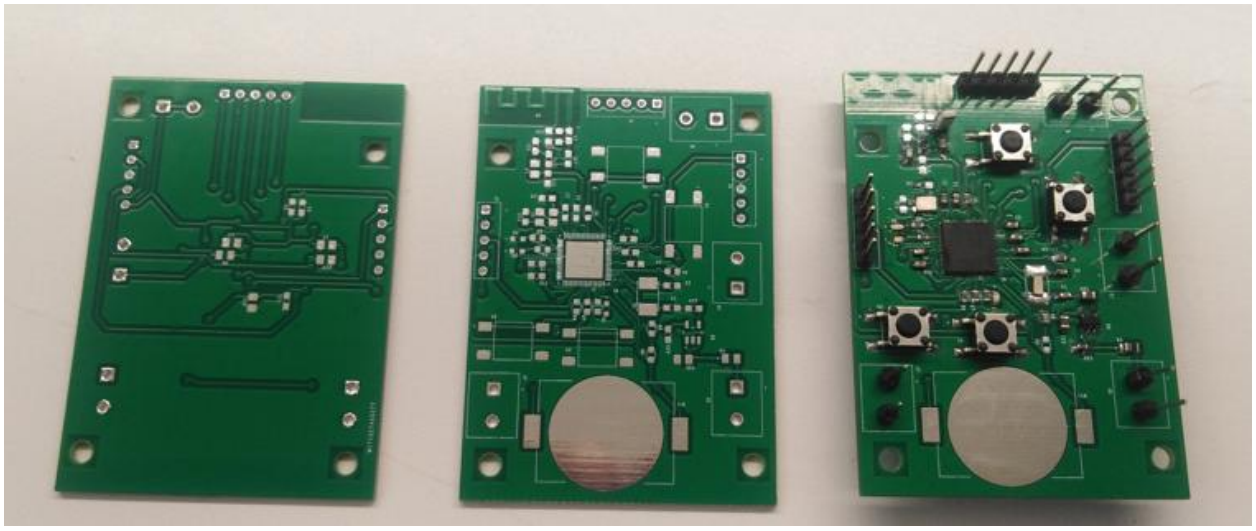


Figure 2.31: (a) Bottom side, (b) Top side, (c) PCB with mounted components

2.7 Internal Temperature sensor:

STM32WB55RGV6 Nucleo board contains several a successive approximation ADCs, which has up to 16 external channels and three internal channels: internal reference voltages and temperature sensor.

The temperature sensor (TS) generates a voltage V_{TS} that varies linearly with temperature. The offset of this line varies from chip to chip due to process variation. The temperature sensor is internally connected to the ADC1_IN17 input channel is shown in Figure 2.32, which is used to convert the sensor output voltage into a digital value. It support the temperature range –40 to 125 °C.

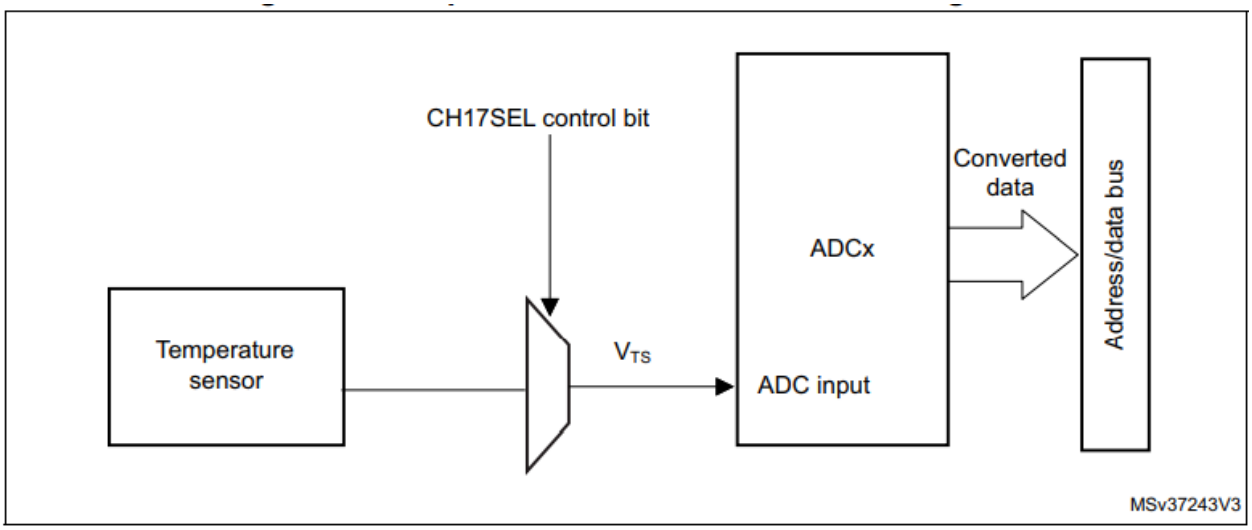


Figure 2.32: Internal TS connected with ADC

2.8 Shock Sensor

The spring loaded vibration switches are non-directional vibration trigger switches with low sensitivity. Inside the sensor is a stiff spring wound around a long metal pin. The switch is moved when the spring touches the center pole to establish a connection. When there is movement, the two pins act as a closed switch. Switch stays open when everything is still.

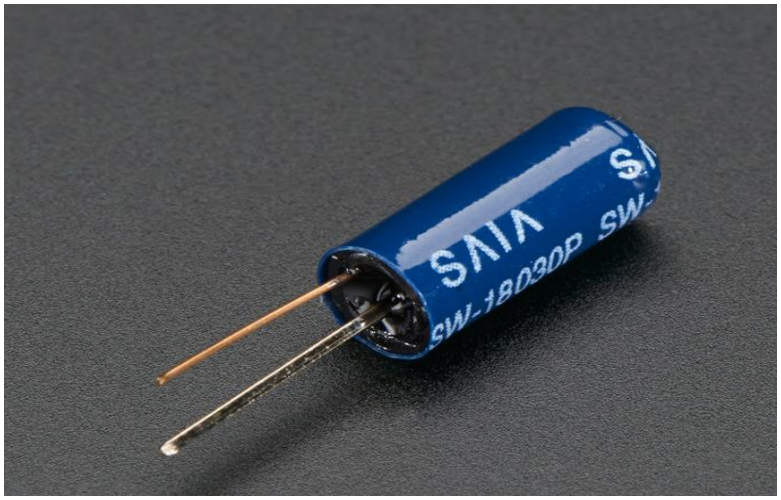


Figure 2.33: Vibration Sensor

One pin of Shock or Vibration sensor will be connected to one of the wake up pin of our processor, which is further connected to ground(To avoid floating of pin). Another pin is connected to VDD.

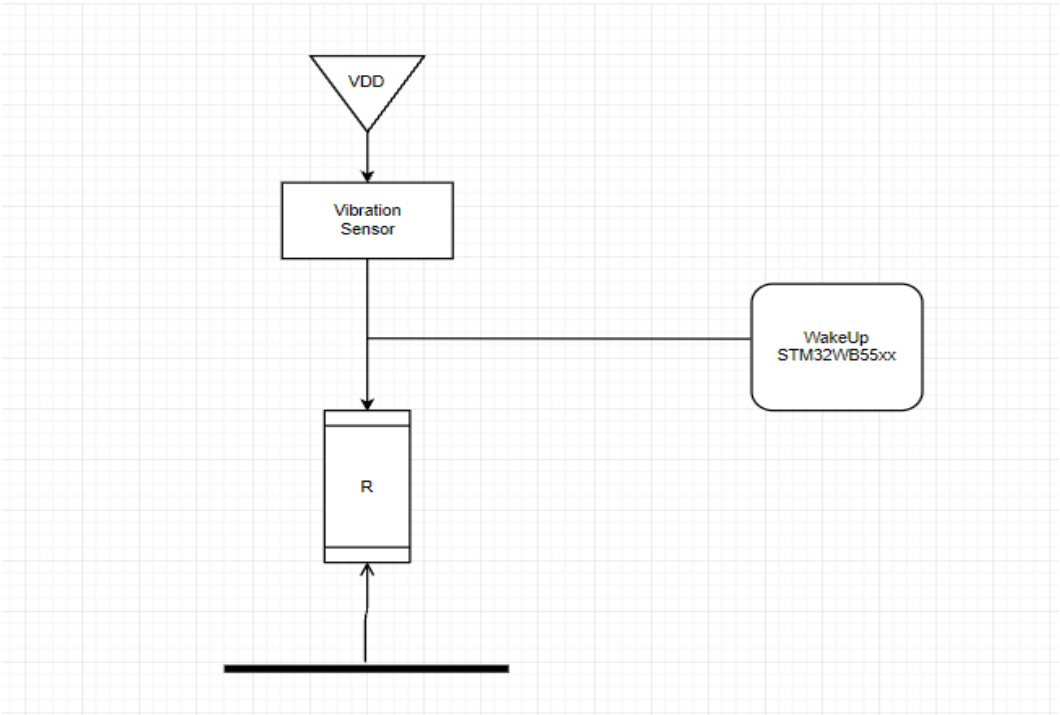


Figure 2.34: Configuration for Vibration sensor

2.9 Photodiode

OSRAM SFH2440 High-Speed PIN Photodiode is a high linearity photodiode with 9.4nA / lx spectral sensitivity and excellent IR suppression. This diode comes in a dual in-line plastic (DIL) package and operates in a temperature range of -40ºC to 100ºC. The SFH2440 photodiode has a low temperature coefficient of spectral sensitivity, and the spectral sensitivity is made to order to human eye sensitivity. This photodiode is apt for high speed applications like mobile phone, regulation of air conditioning, and bio-monitoring.

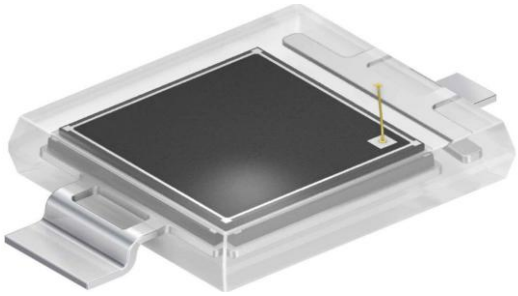


Figure 2.35: Photodiode

Photodiodes or light sensors, work by creating a pair of electron holes when a photon of sufficient energy hits the diode. This mecha nism is also called as inner photoelectric effect. If the absorption arises in the depletion region junction, then the carriers are removed from the junction by the inbuilt electric field of the depletion region.

One pin of Photodiode or light sensor will be connected to one of the wake up pin of our processor, which is further connected to ground(To avoid floating of pin). Another pin is connected to VDD.

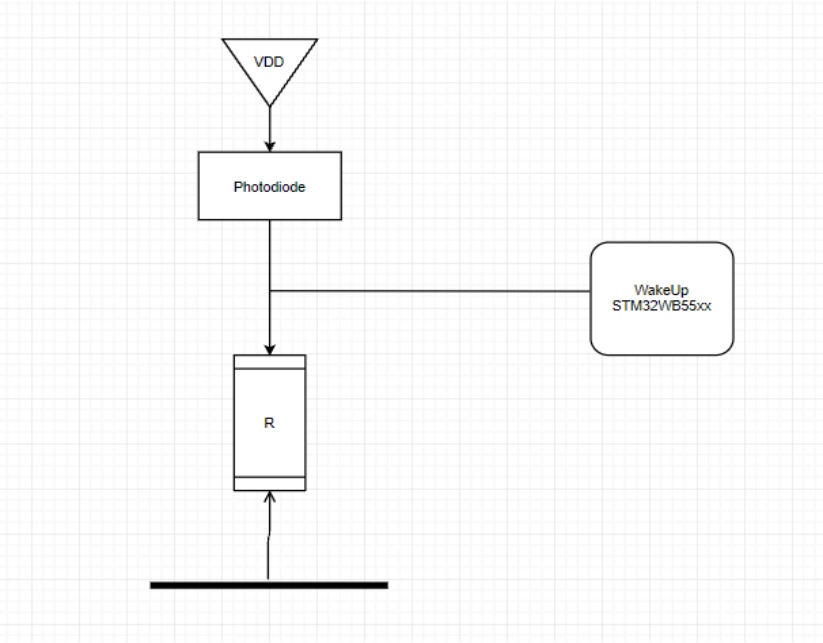


Figure 2.36: Configuration for Photodiode

2.10 GPS module

The u-blox concurrent SAM-M8Q GNSS patch antenna module is shown in Figure 2.37, which benefits from the exceptional performance of the u-blox M8 multi-GNSS engine. The SAM-M8Q module provides high sensitivity and minimal acquisition times in an ultra-compact form factor.

Figure 2.37: GPS Module

2.11 Designing of boards

Schematics of TAG board are shown in Figure 2.38, Figure 2.39 and Figure 2.40:

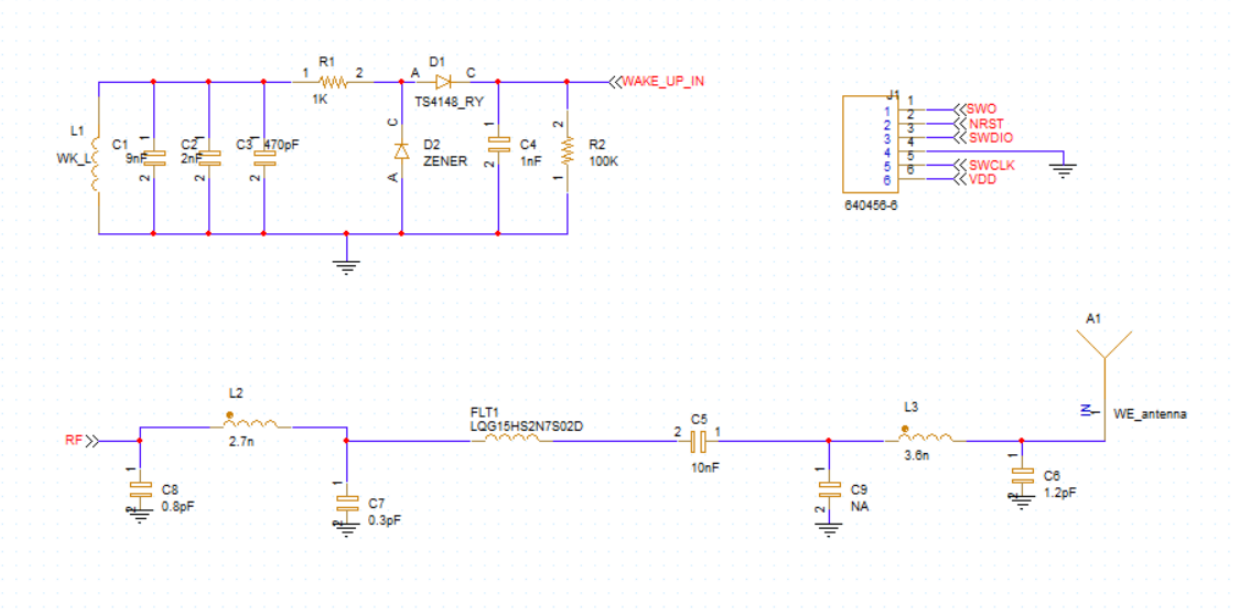


Figure 2.38: Wake-up receiver and RF antenna of TAG board

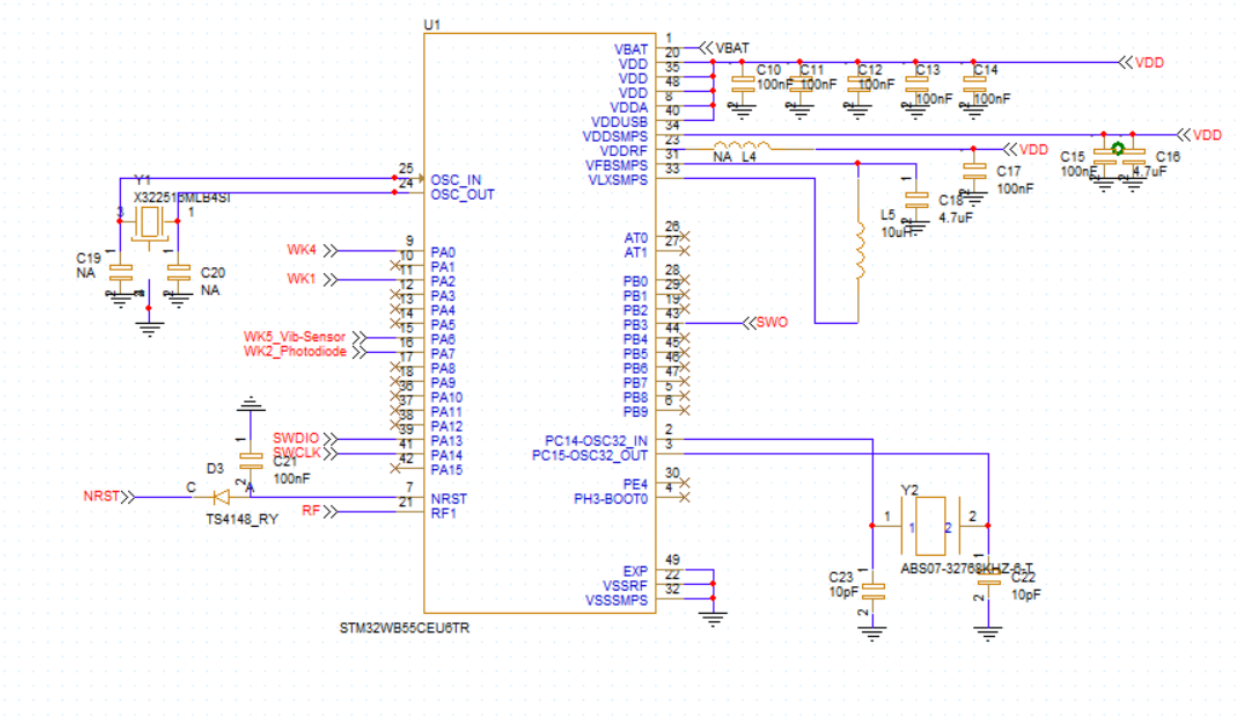


Figure 2.39: STM32WB55CGU6

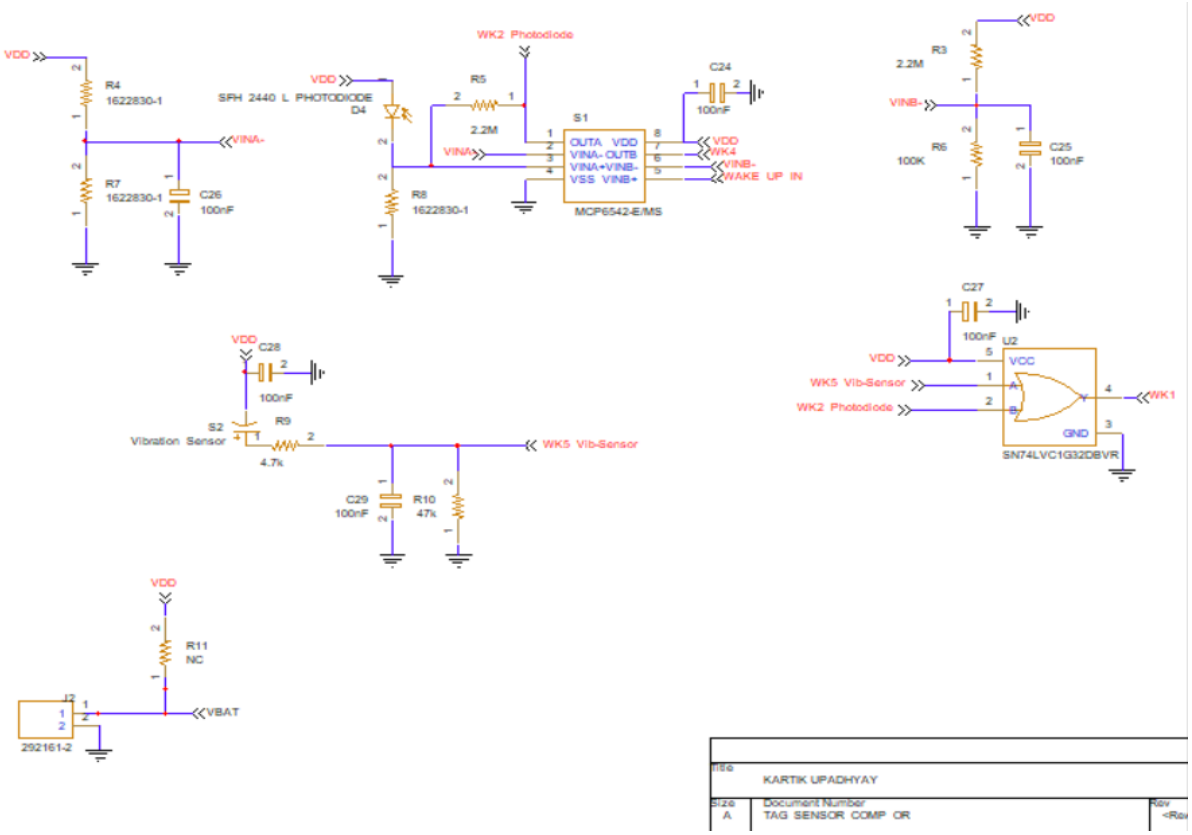


Figure 2.40: Vibration sensor and Photodiode

PCB design of TAG board are shown in Figure 2.41, Figure 2.42 and Figure 2.43:

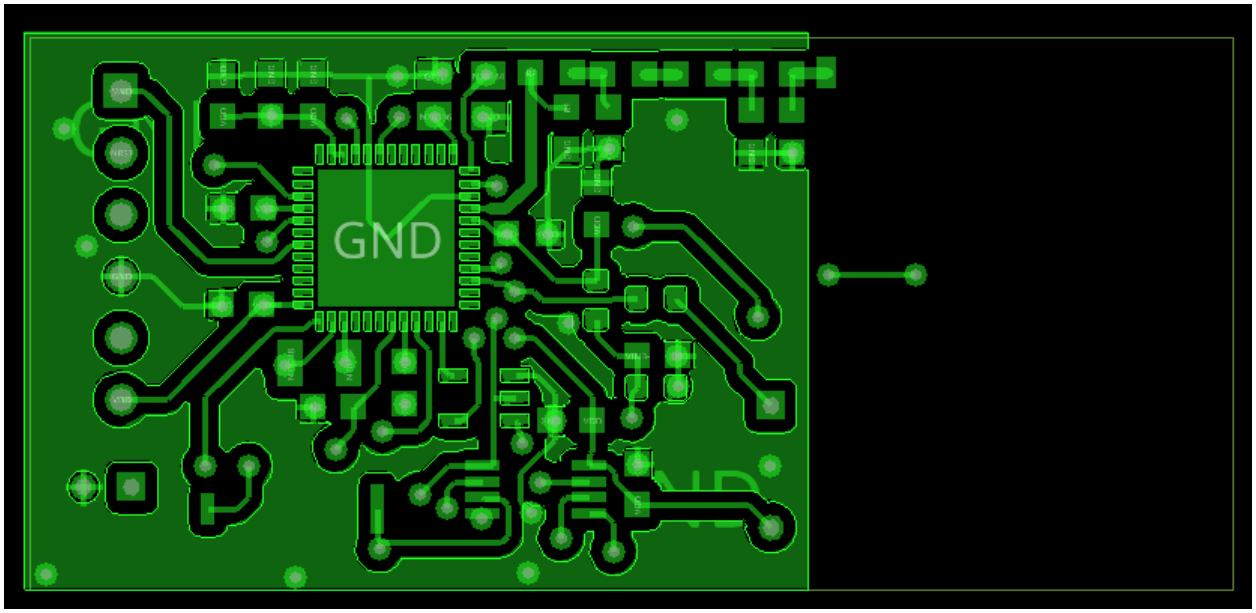


Figure 2.41: TOP Layer of TAG board

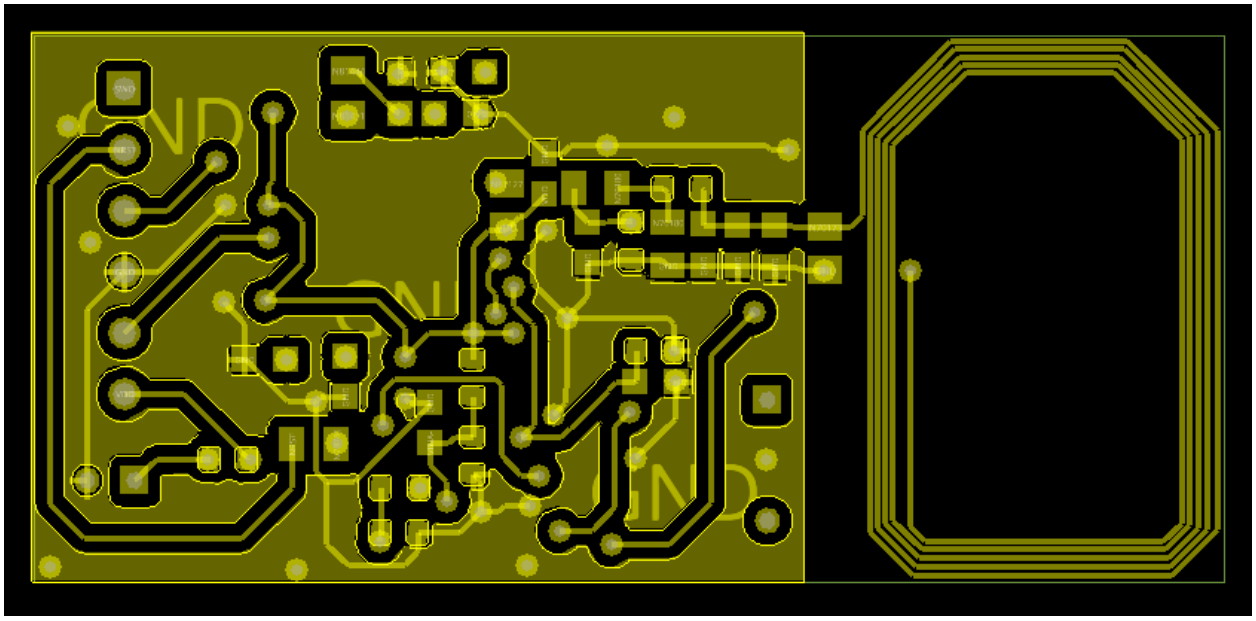
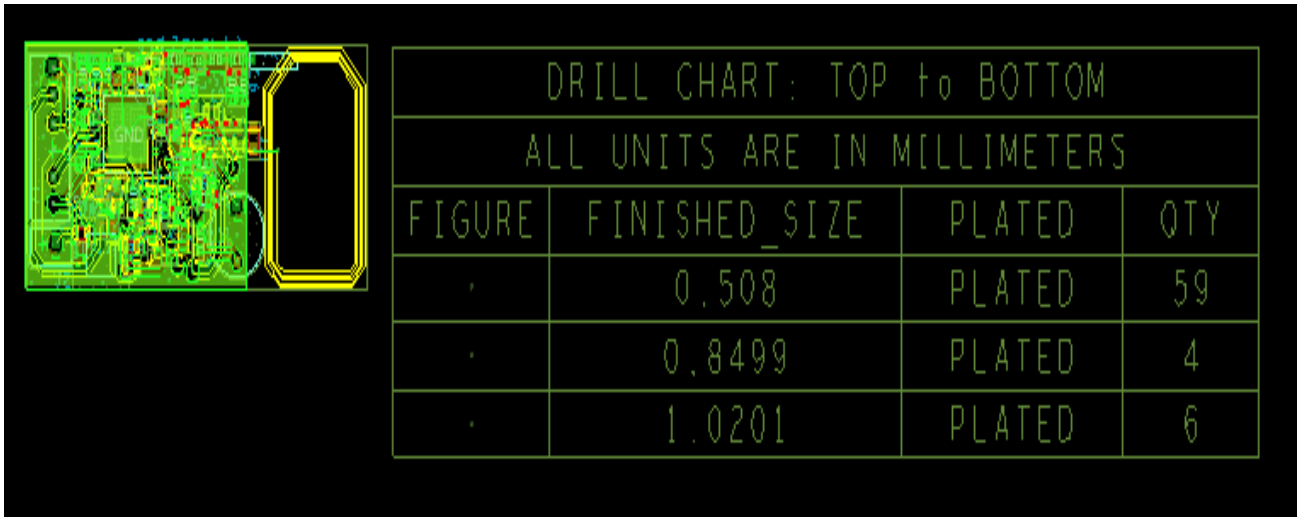


Figure 2.42: BOTTOM Layer of TAG board



DRILL CHART: TOP to BOTTOM			
ALL UNITS ARE IN MILLIMETERS			
FIGURE	FINISHED_SIZE	PLATED	QTY
.	0.508	PLATED	59
.	0.8499	PLATED	4
.	1.0201	PLATED	6

Figure 2.43: Complete view with Drill Table of TAG board

Final PCB of TAG board is shown in Figure 2.44 and Figure 2.45:

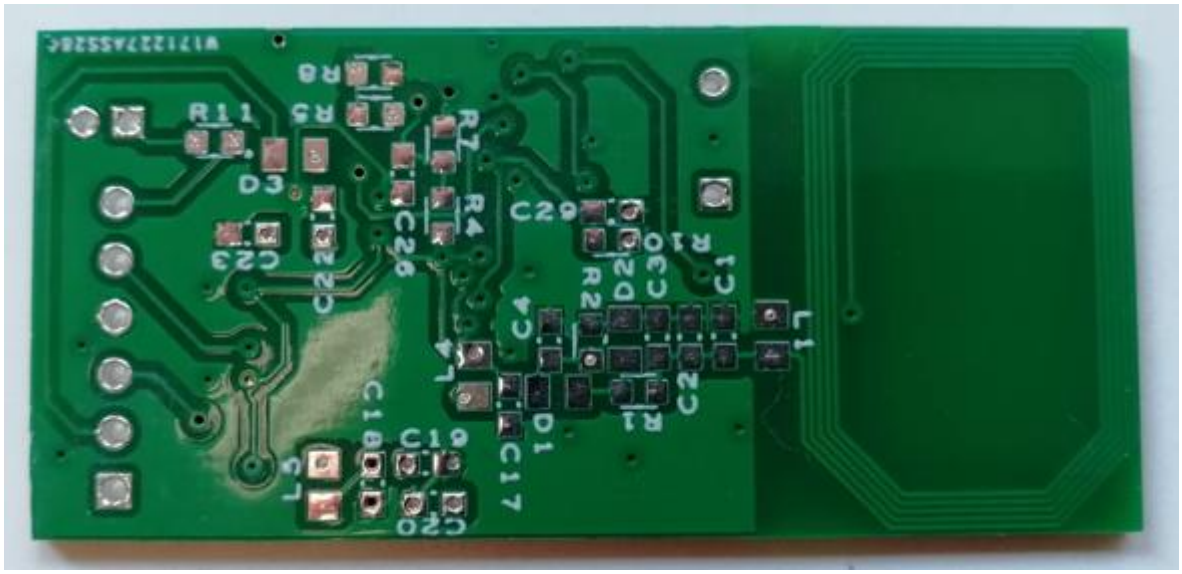


Figure 2.44: Top side of TAG board

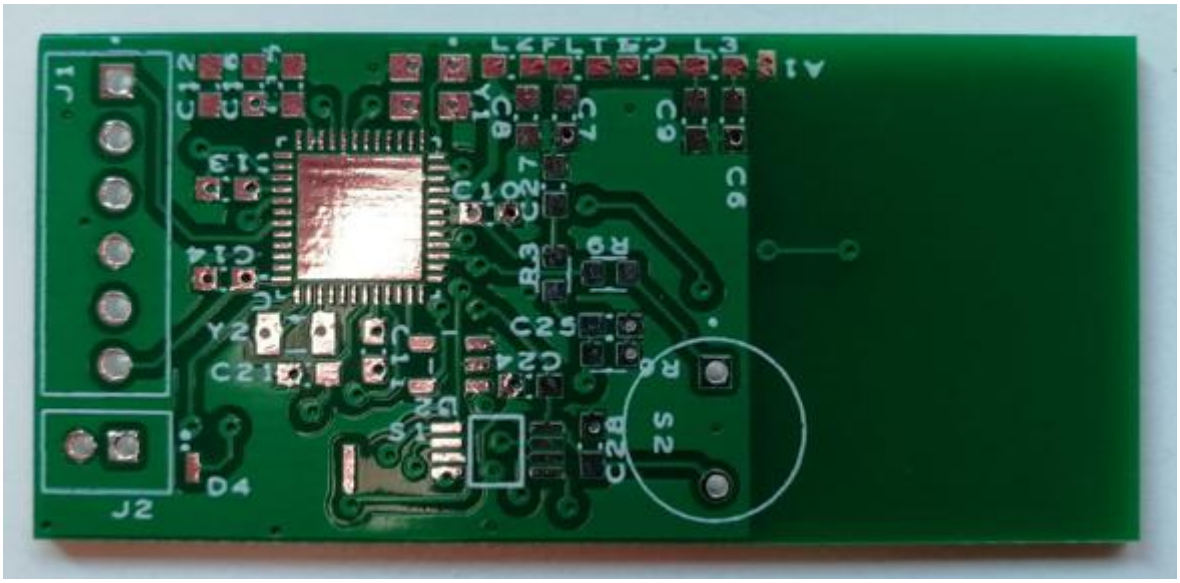


Figure 2.45: Bottom side of TAG board

Schematic of RECEIVER EXTERNO board are shown in Figure 2.46, Figure 2.47 and Figure 2.48:

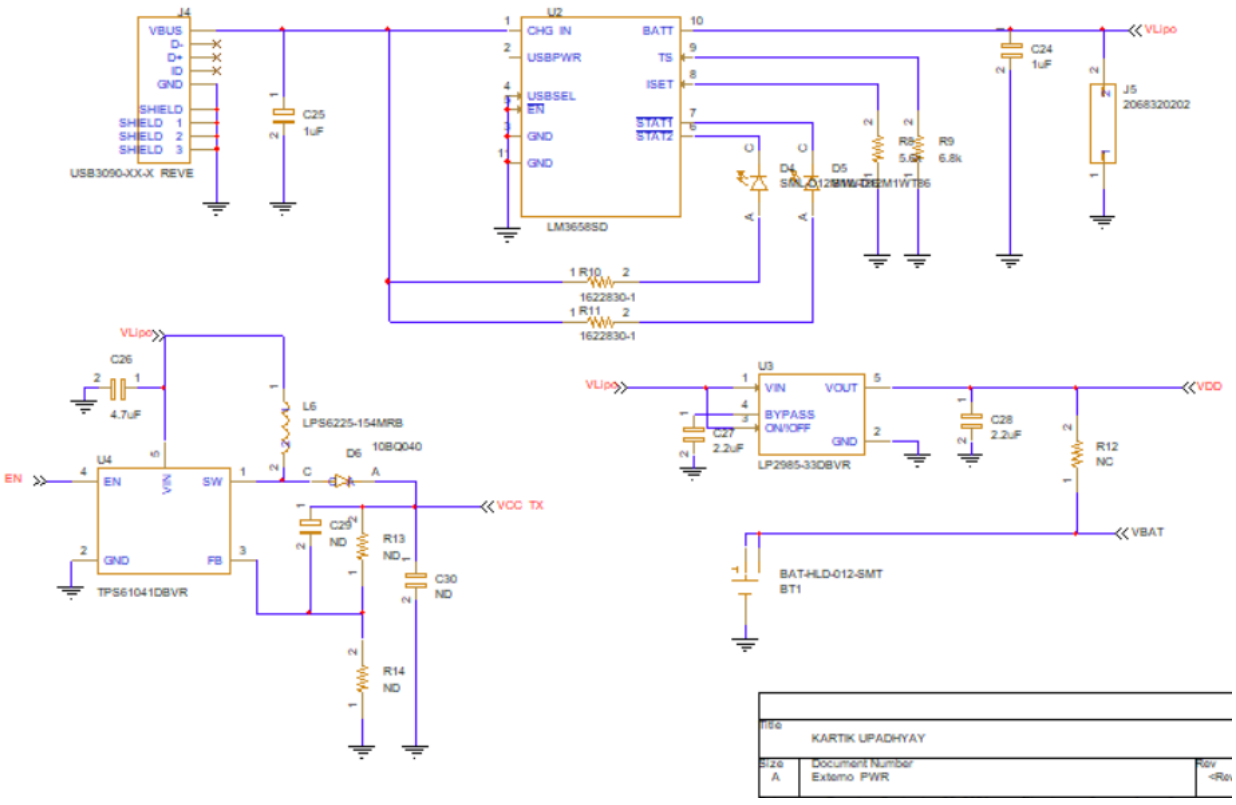


Figure 2.46: Power Supply of RECEIVER EXTERNO board

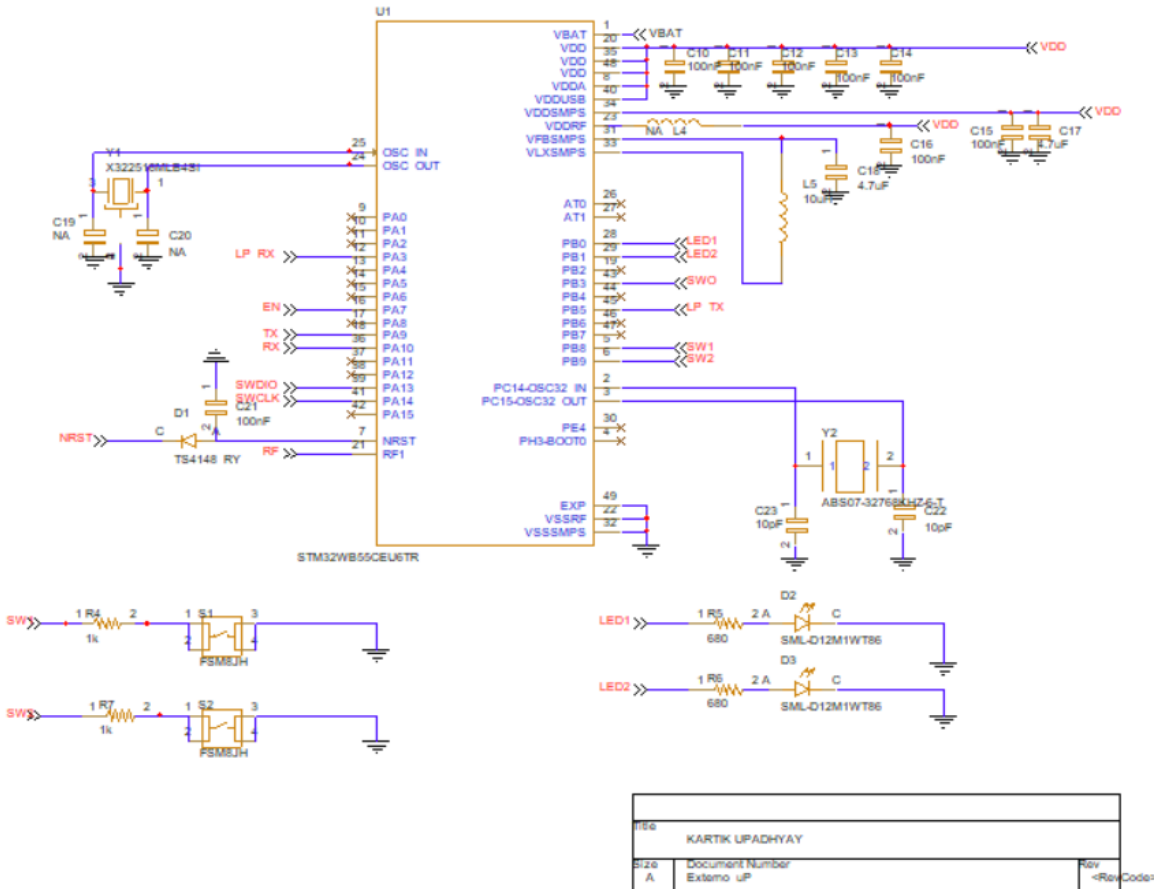


Figure 2.47: STM32WB55CGU6

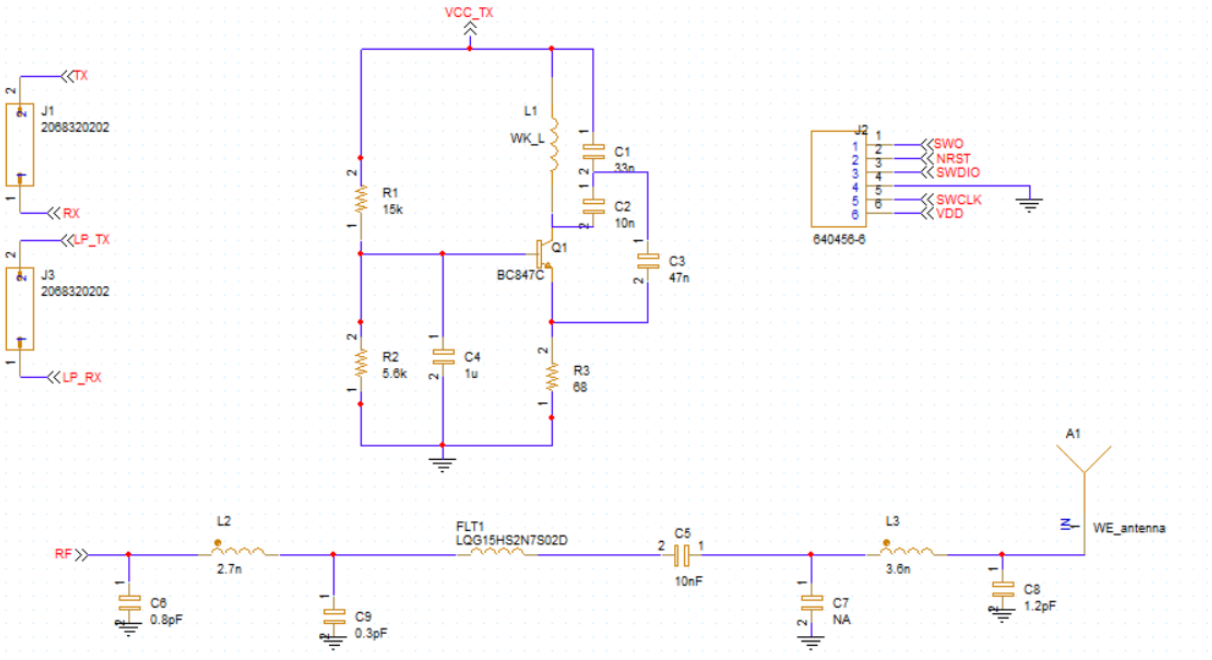


Figure 2.48: Wake-up generator and RF of RECEIVER EXTERNO board

PCB design of RECEIVER EXTERNO board are shown in Figure 2.49, Figure 2.50 and Figure 2.51:

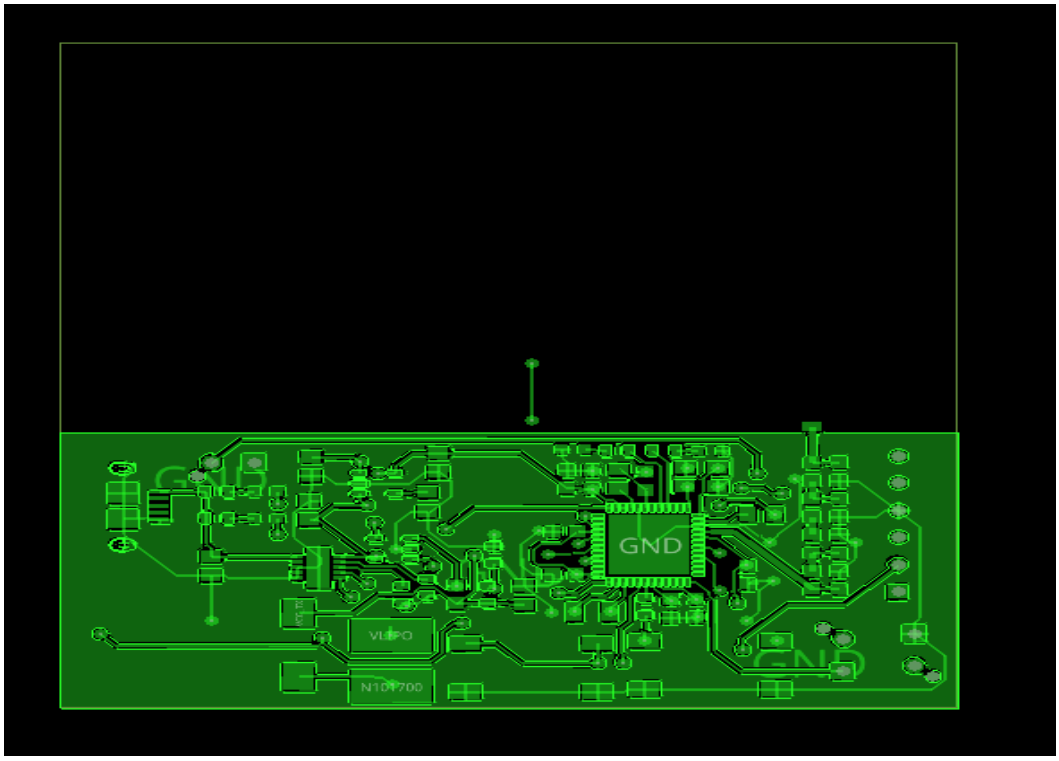


Figure 2.49: TOP Layer of RECEIVER EXTERNO board

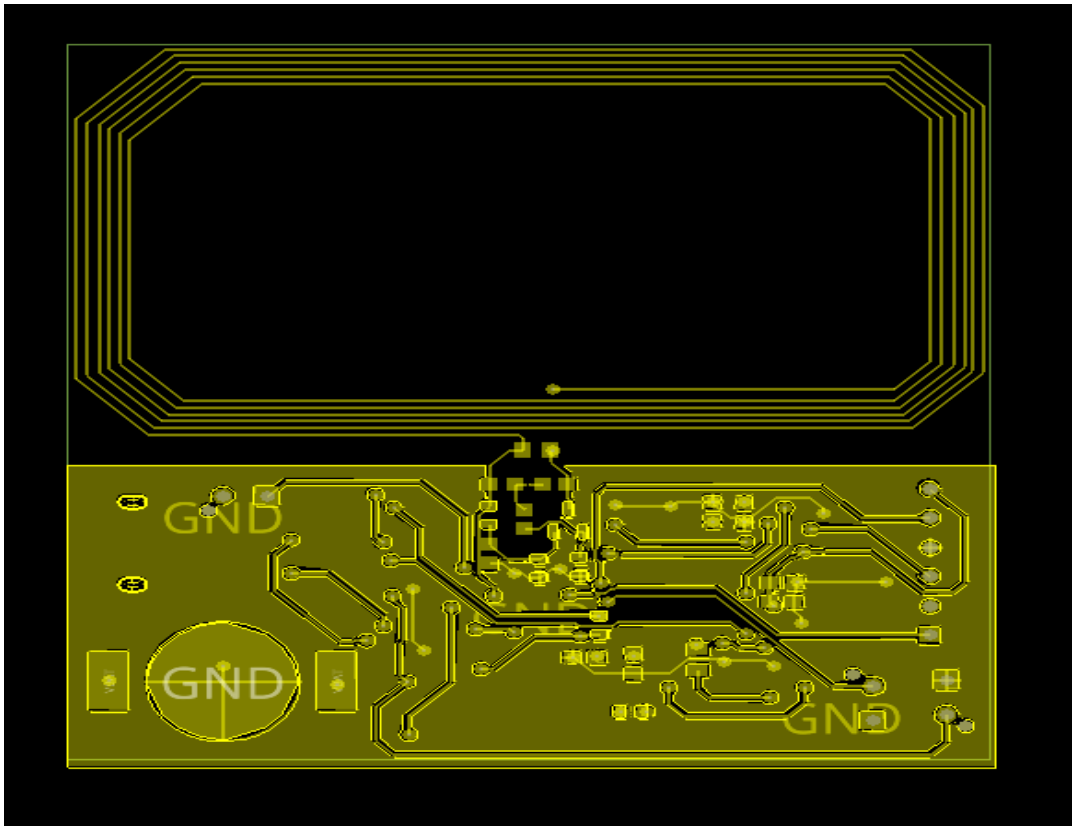


Figure 2.50: BOTTOM Layer of RECEIVER EXTERNO board

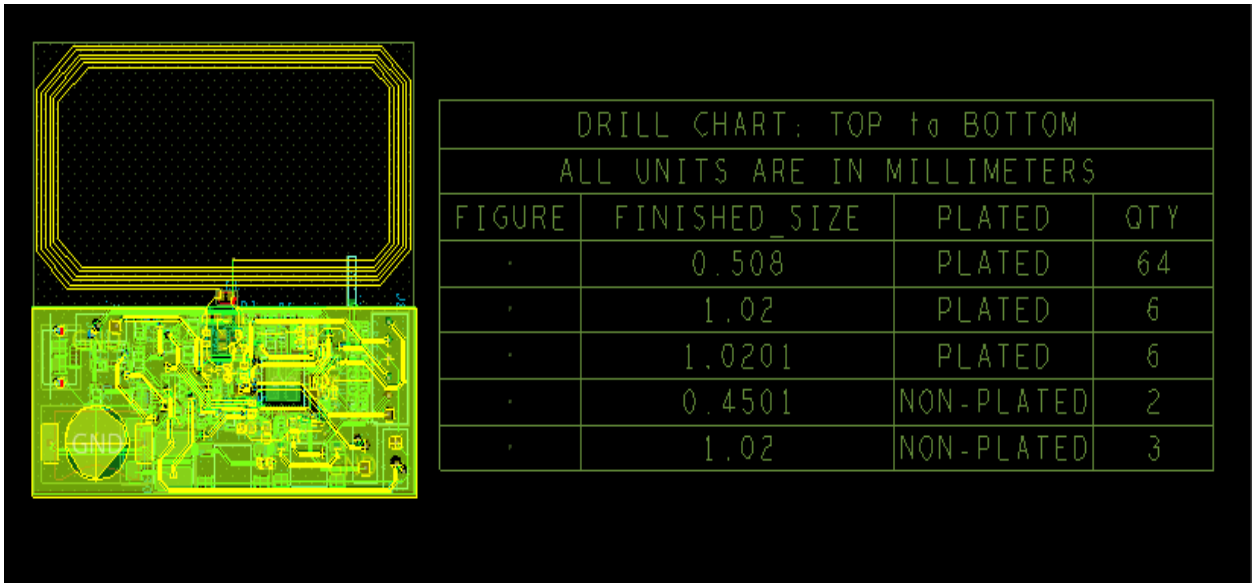


Figure 2.51: Complete view with Drill Table of RECEIVER EXTERNO board

PCB of RECEIVER EXTERNO board are shown in Figure 2.52 and Figure 2.53:

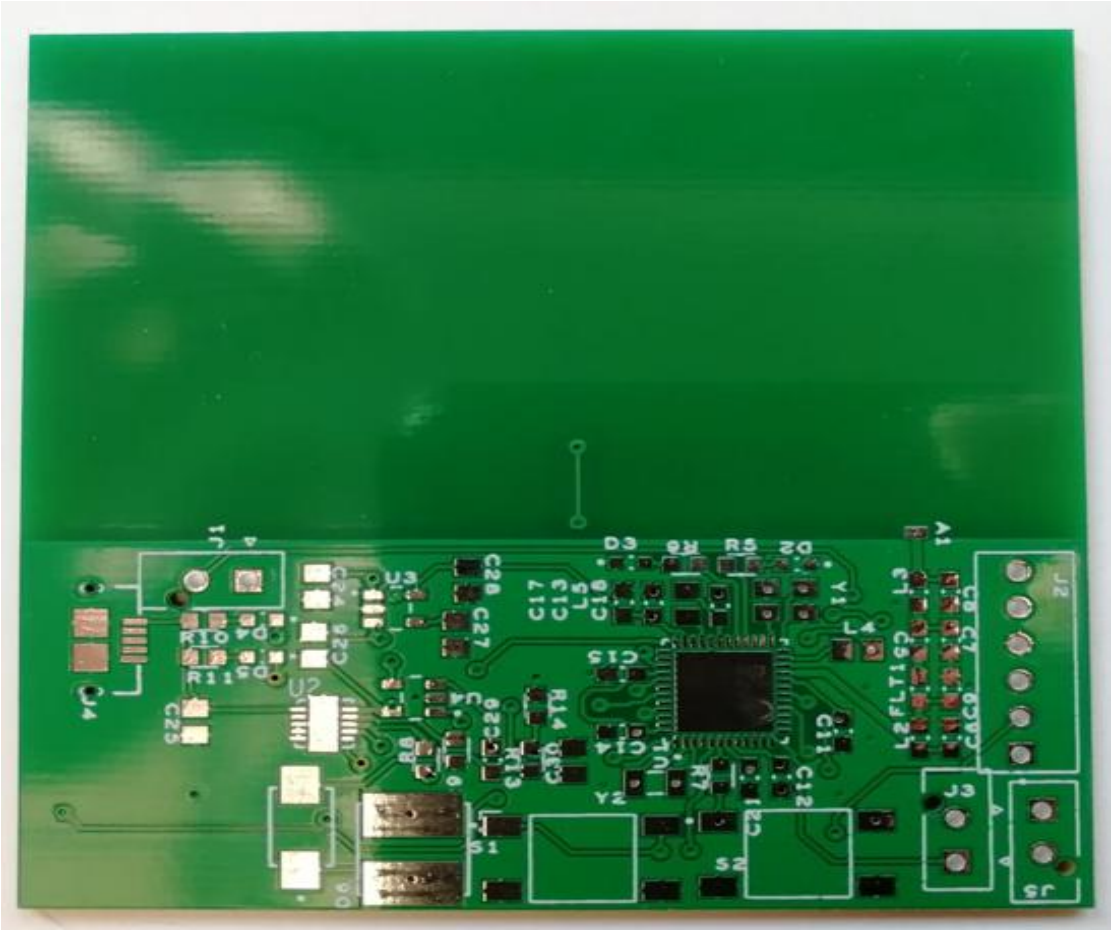


Figure 2.52: Top side of RECEIVER EXTERNO board

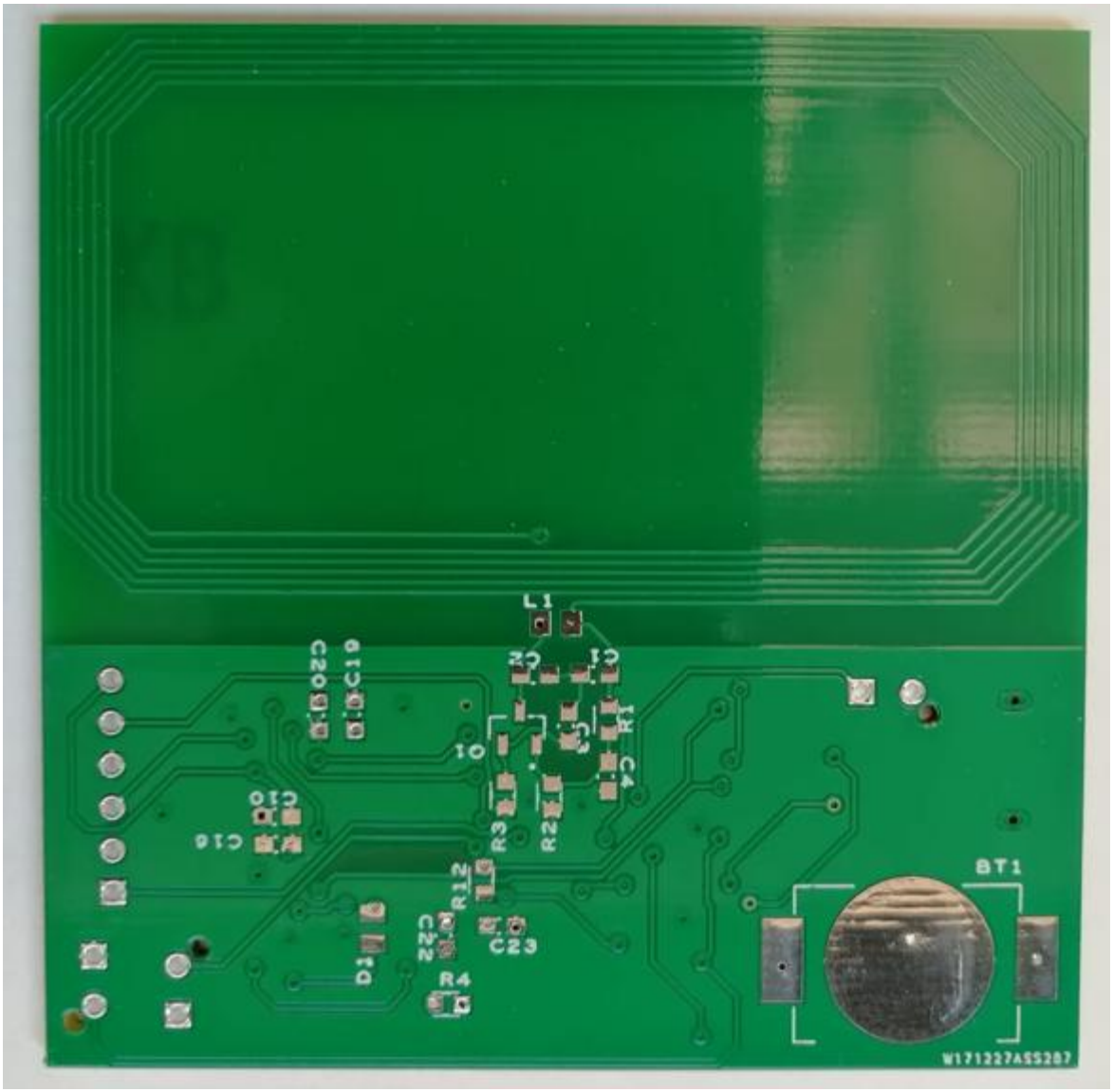


Figure 2.53: Bottom side of RECEIVER EXTERNO board

Schematic of CASAFORMA board are shown in Figure 2.54, Figure 2.55 and Figure 2.56:

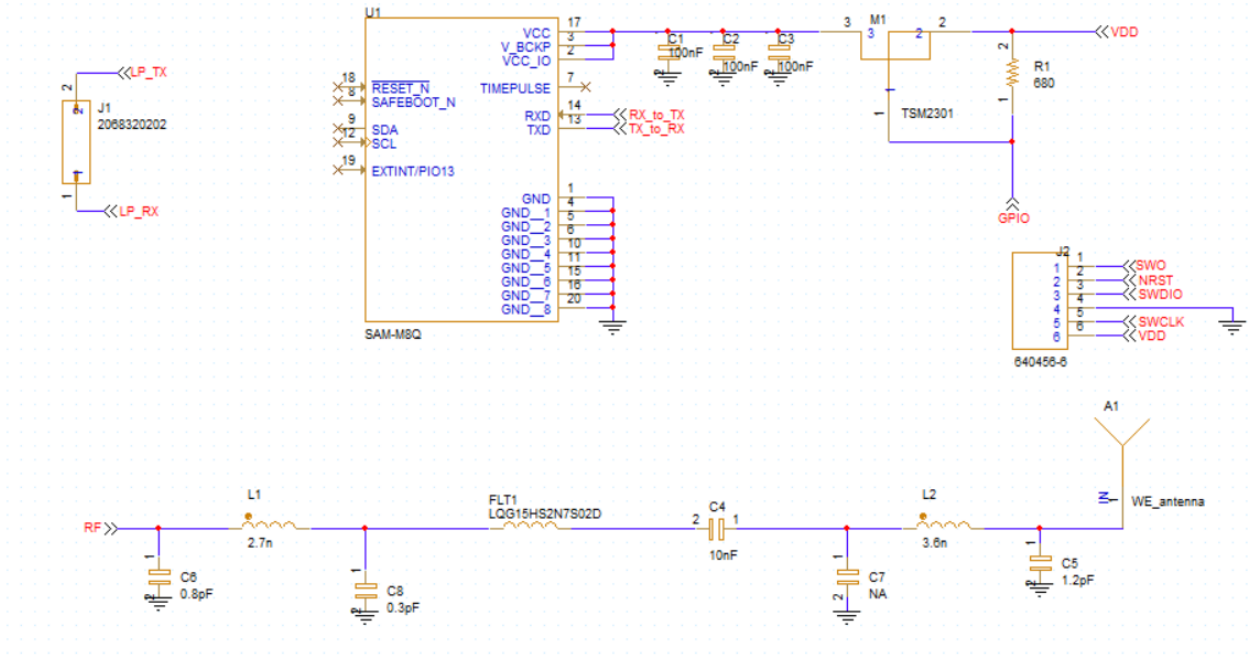
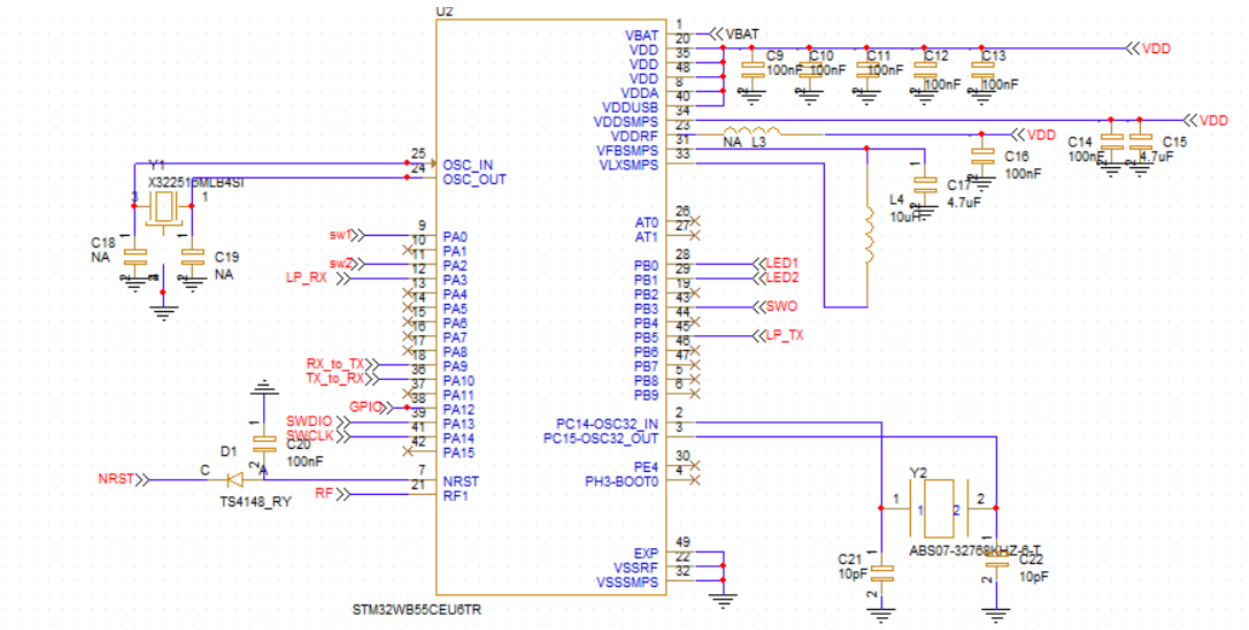


Figure 2.54: GPS module and RF of CASAFORMA board



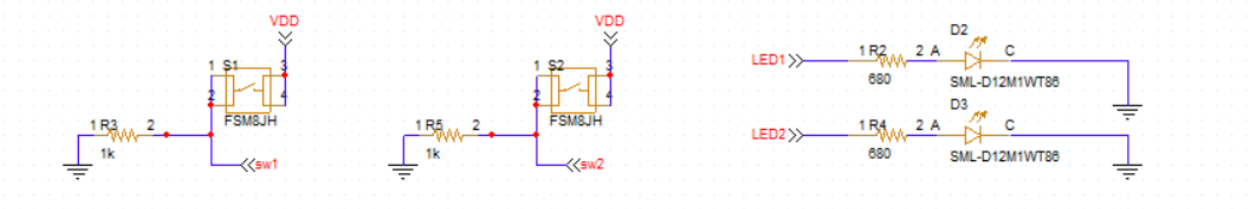


Figure 2.55: STM32WB55CGU6

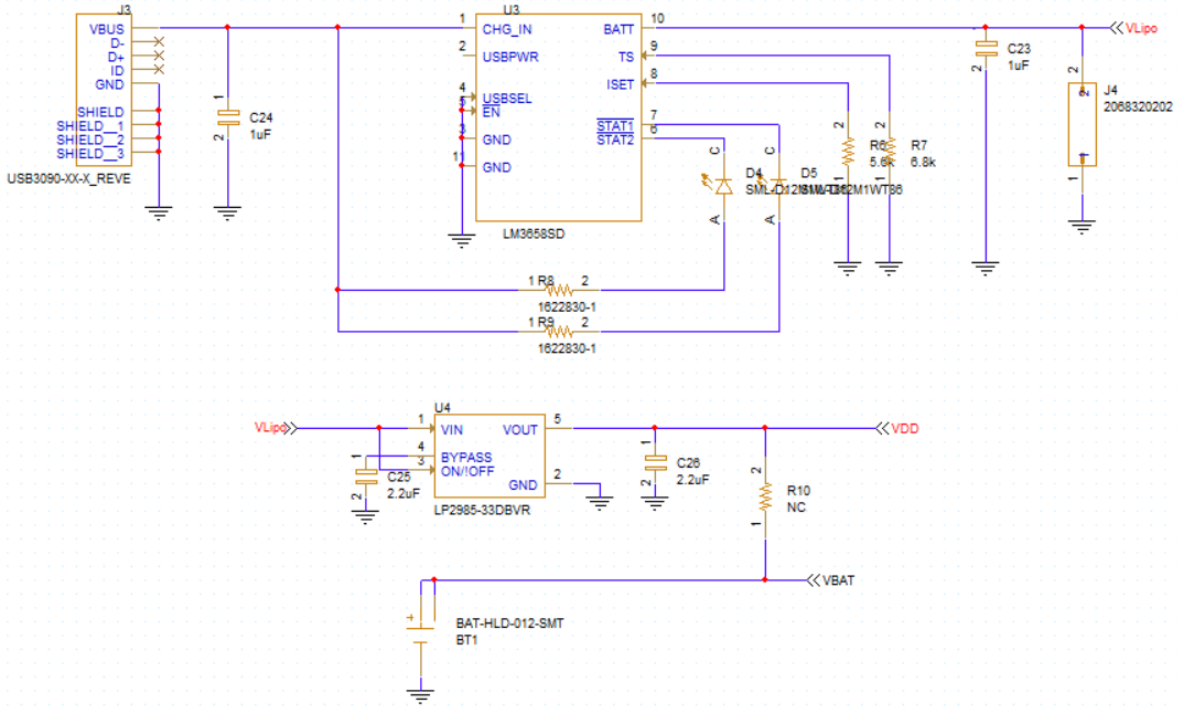


Figure 2.56: Power Supply of CASAFORMA board

PCB design of CASAFORMA board are shown in Figure 2.57 and Figure 2.58:

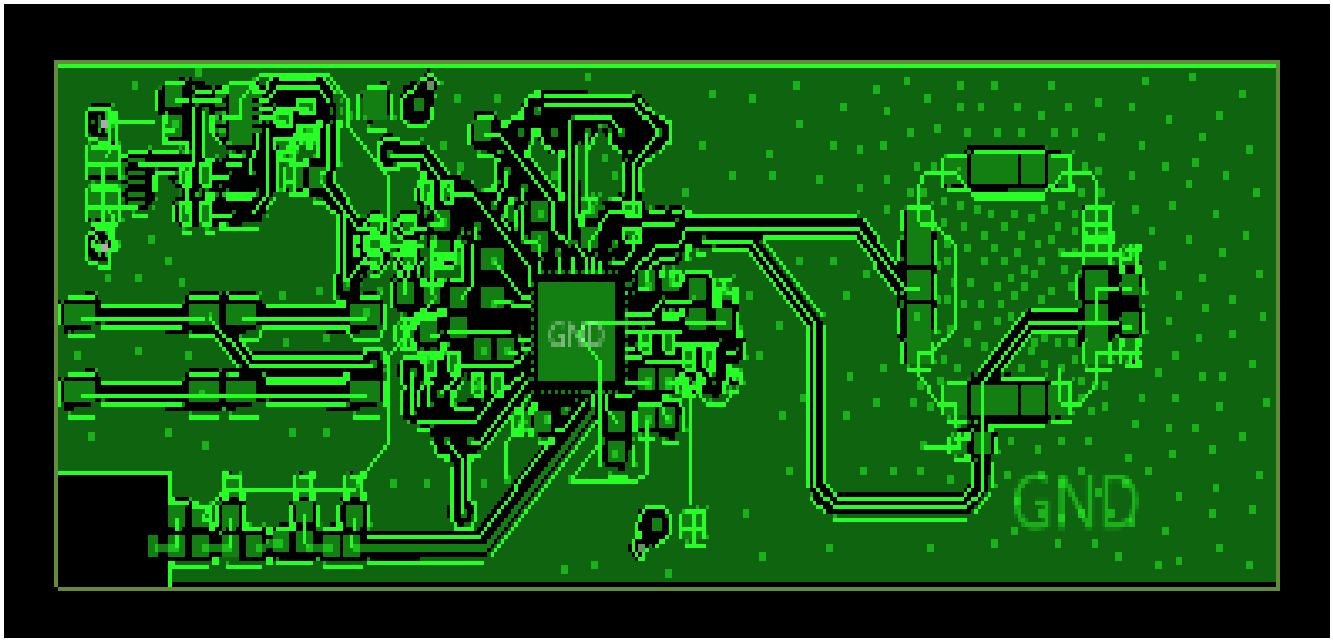


Figure 2.57: TOP Layer of CASAFORMA board

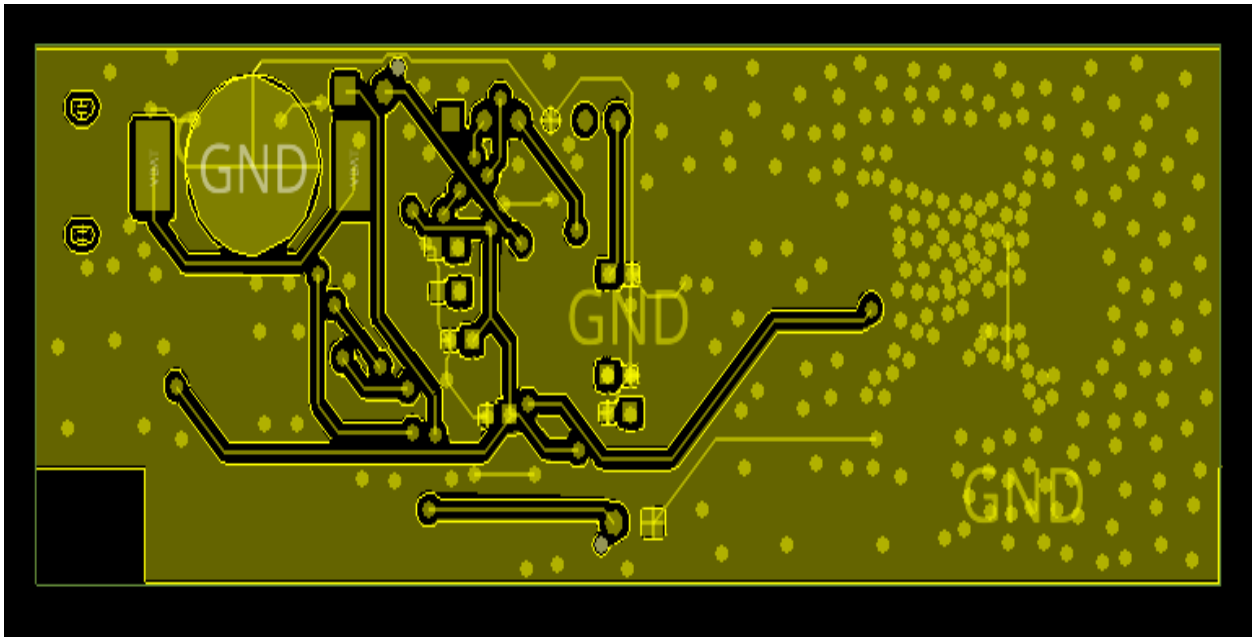


Figure 2.58: BOTTOM Layer of CASAFORMA board

PCB of CASAFORMA board are shown in Figure 2.59 and Figure 2.60:

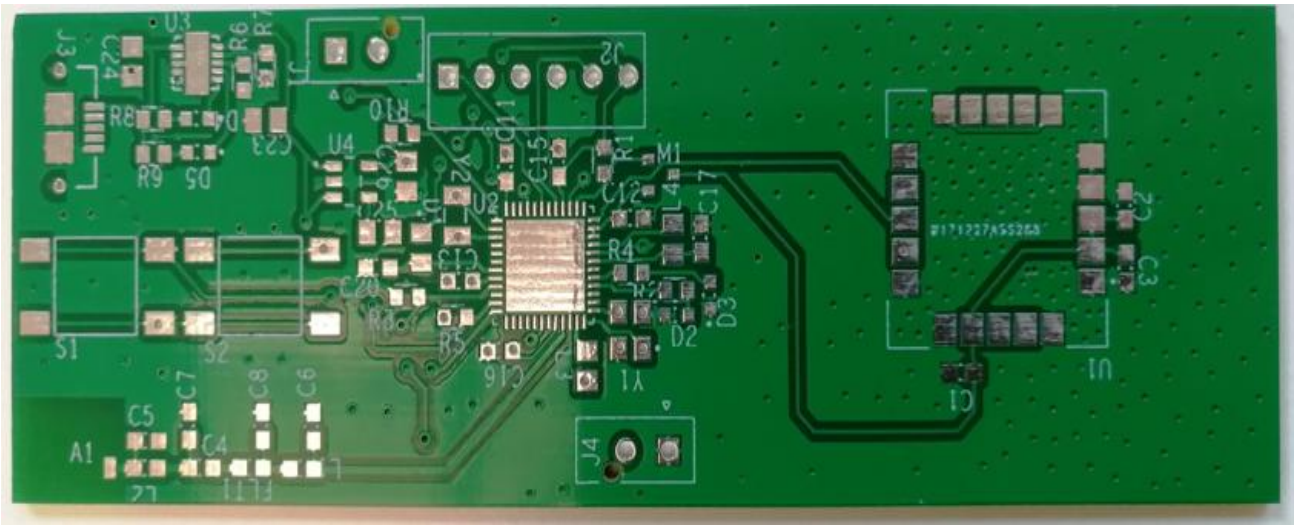


Figure 2.59: Top side of CASAFORMA board

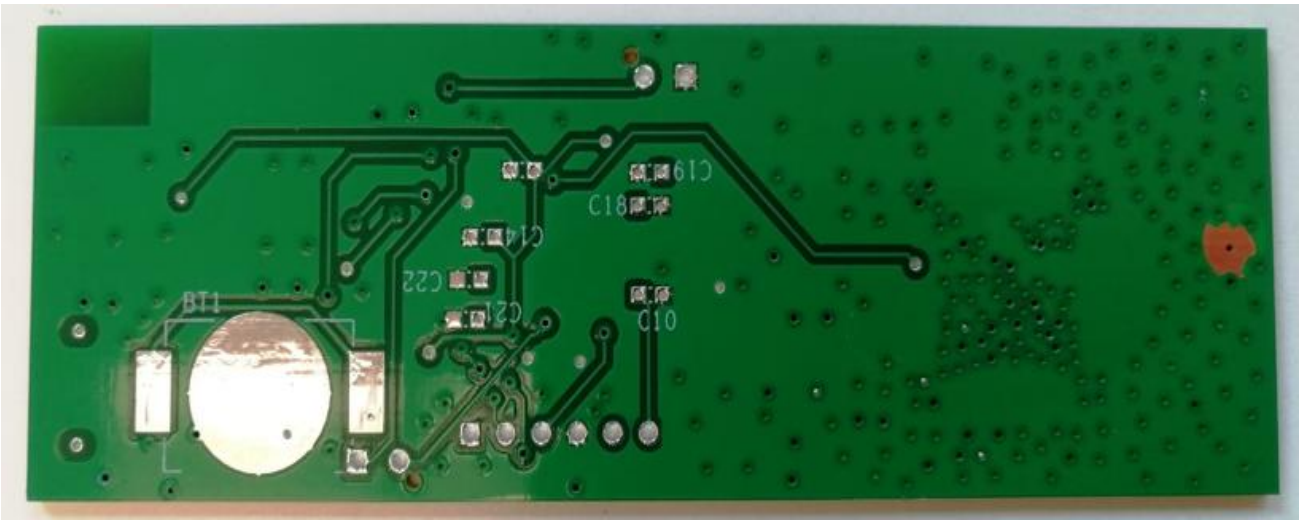


Figure 2.60: Bottom side of CASAFORMA board

Chapter 3

Software Design

3.1 STMicroelectronics

STMicroelectronics idea to drastically improvise developer efficiency and productivity by reducing amount to time, cost and effort by introducing several software chain tools like STM32CubeMX, STM32CubeIDE, STM21CubeMonitorRF etc. To maximize the utilization of resources available given like graphical software configuration tool, Hardware Abstraction Layer(HAL), Low Layer APIs(LL) to bare board knowledge of each resource of microcontroller.

3.1.1 Graphical software tool:

While using its IDE, To initialize we used graphical software configuration tool that allows to set varies parameters like GPIO, clock frequency setting etc, which generates C initialization code using graphical wizards, as reported in Figure 3.1.

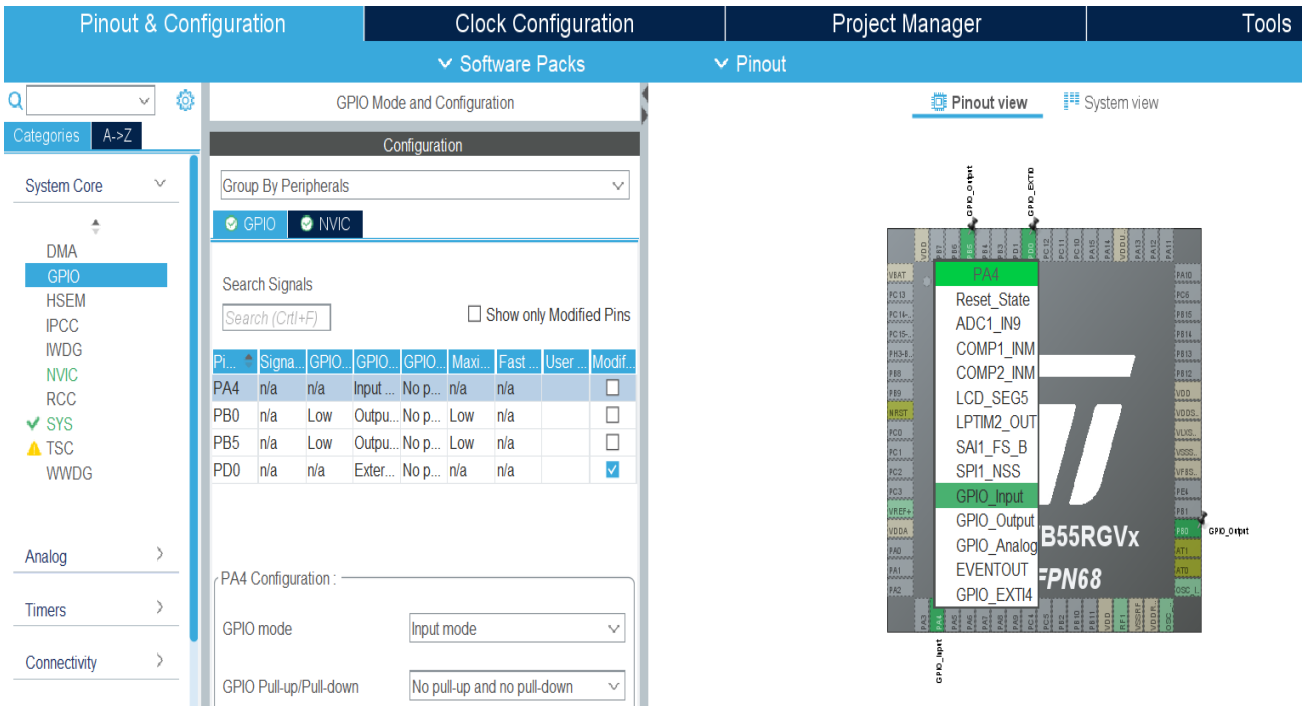


Figure 3.1: Graphical software configuration tool

3.1.2 Hardware Abstraction Layer(HAL)

The STM32Cube Hardware Abstraction Layer, STM32 abstraction layer embedded software making sure to increase portability of code across the STM32 family. Hardware Abstraction Layer APIs are accessible for all the common peripheral features as well as extension in case of specific peripheral features.

The HAL drivers are made to offer a large set of drivers and to easily interact with the upper layers of the application. Each driver consists of a set of functions covering the most common peripheral features. Each driver is designed using a common API that standardizes driver structure, functions, and parameter names.

HAL GPIO Examples:
HAL_GPIO_ReadPin()
HAL_GPIO_WritePin()
HAL_GPIO_TogglePin()
HAL_GPIO_LockPin()

HAL Detailed Function Description, as reported in Figure 3.2.
HAL_GPIO_WritePin() :

Function name

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

Function description

Set or clear the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..F) to select the GPIO peripheral for STM32WBxx family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
  - GPIO_PIN_RESET: to clear the port pin
  - GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Figure 3.2: Function Description

Using STM32 HAL device drivers can be beneficial in many conditions and it assist in reduce development time. Especially for "Proof of Concept" projects. It is inefficient to spend a lot of time making a entire software stack for a specific target, then it proves that the main idea requires improvement or the target itself is not working well enough. In some cases, the moderately high-level API offered by HAL may have more options available than needed. Therefore, it may use more memory space and due to the overhead of the built-in library, it may execute some tasks slightly slower.

3.1.3 Low Layer APIs(LL)

The low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The low-layer (LL) drivers are only available for a set of peripherals.

The low-layer drivers provide hardware services based on the available features of the STM32 peripheral functions. These services accurately reflect the capabilities of the hardware and provide one-time operations that must be invoked according to the programming model described in the microcontroller line reference manual. As a consequence, LL driver don't perform any processing and don't need any extra memory resources to store their states, counters or data pointers: all operations are performed by changing the contents of the associated peripheral registers.

There are low-level hardware drivers for almost all the hardware peripherals in the STM32 microcontrollers. Including Timers, ADC, USART, I2C, USB, DAC, Comparators, etc.

LL drivers can be used and optimized more at the register level to improve memory usage or execution speed. However, the final application will not be easy to port across multiple targets.

LL GPIO Examples:
LL_GPIO_SetPinMode()
LL_GPIO_GetPinMode()
LL_GPIO_SetPinOutputType()
LL_GPIO_SetPinSpeed()

LL Detailed Function Description, as reported in Figure 3.3.
LL_GPIO_SetPinMode()

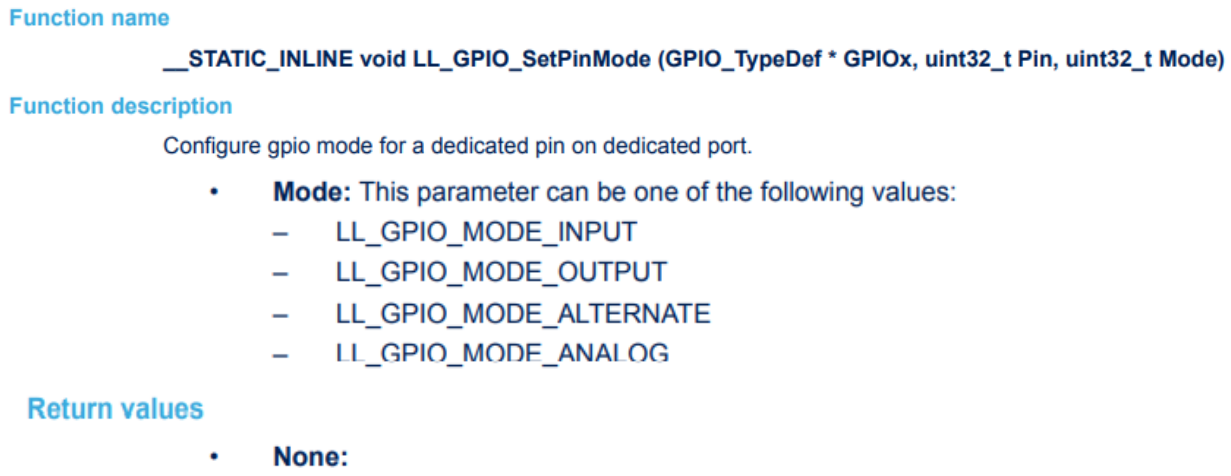


Figure 3.3: Function Description

3.1.4 Bare Board

Bare board programming requires in depth knowledge of each register and each of its bit. It is most optimized form of programming but it has no portability of code across the STM32 family. It is very particular hardware specific. It is much closer than HAL and LL APIs.

It requires bit level manipulation of each register.
Example of Bare board instructions:
GPIOx_MODER
GPIOx_OTYPER
GPIOx_OSPEEDR
GPIOx_ODR

Bare board Detailed Function Description, as reported in Figure 3.4.
GPIOx_MODER Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODE[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

Figure 3.4: MODER Register

3.2 Code for Deep-sleep mode and measuring current

Micro-controller STM32WB55RGV6 has several power modes. Like, run & low-power run modes, sleep & low-power sleep modes, stop0, stop1 & stop2 modes, standby mode and shutdown mode.
When we put our micro-controller in low power mode, we prefer to put it in Standby mode due to following reason: It has lowest current consumption in which we can retain our data in SRAM2a. In shutdown mode, it has even less current consumption than standby mode but we cannot retain memory(SRAM1, SRAM2). Other modes like stop modes or sleep mode have higher current consumption.
Putting in standby mode using HAL command:

```
HAL_PWR_EnterSTANDBYMode();
```

Current consumption: 0.59 mA
Putting in standby mode using register level coding:

```
SCB->SCR= 0x4;
```

```
PWR-> CR3 |= 0x200;
PWR->CR1=0x00004433;
PWR->C2CR1=0x33;
```

Current consumption: 0.1 uA

We have put our microcontroller at different modes and check its current consumption, as reported in Table 3.1:
Voltage = 3.3V Operation frequency = 16 MHz

Table 3.1: Current consumption at different modes

S. No.	Operating Mode	I
1	Normal Mode	1.31 mA
2	Stop 2 Mode	0.71 mA
3	Stop 1 Mode	0.60 mA
4	Stop 0 Mode	0.25 mA
5	Standby Mode	0.1 uA
6	Shutdown Mode	*BOR

*BOR= Below Operating Range of Ammeter used
In Shutdown Mode, current consumption was in the range of neno-Amps, which was out of range of ammeter used.
Complete code is given at Appendix A

3.3 Current measurement at different frequency

STM32WB55RGV6 Nucleo board contains several clock like MSI(Multi-Speed Internal), HSI(High Speed Internal), HSE(High Speed External) etc. We have checked current consumption of microcontroller and current consumption of whole Nucleo board at several different frequency and by using different clocks and results are reported in Table 3.2:

Table 3.2: Current consumption at different frequency at normal mode using MSI clock

S. No.	Frequency	MCU 1(mA)	BOARD(mA)
1	48MHz	1.16	8.90
2	32M	1.30	7.15
3	24M	1.20	5.85
4	16M	1.1	4.6
5	8M	1.08	3.4
6	4M	1.08	2.8
7	2M	1.08	2.54
8	1M	0.90	2.30
9	800K	1.05	2.27
10	400K	1.00	2.10

11	200K	0.95	2.23
12	100K	0.85	1.89
	Current consumption while using HSI clock		
13	16M	1.216	4.6
	Current consumption while using HSE clock		
14	32M	1.22	6.84

3.4 . Internal temp sensor process and comparison with normal temperature

STM32WB55RGV6 Nucleo board contains several a successive approximation ADCs, which has up to 16 external channels and three internal channels: internal reference voltages and temperature sensor.

The temperature sensor is internally connected to the ADC1_IN17 input channel, which is used to convert the sensor output voltage into a digital value.
Formula to convert value obtained from ADC to ^{o}C is following:

$$\text{Temperature (in °C)} = \frac{\text{TS\_CAL2\_TEMP} - \text{TS\_CAL1\_TEMP}}{\text{TS\_CAL2} - \text{TS\_CAL1}} \times (\text{TS\_DATA} - \text{TS\_CAL1}) + 30\ °C$$

Where:

TS_CAL2 is the temperature sensor calibration value acquired at TS_CAL2_TEMP.
TS_CAL1 is the temperature sensor calibration value acquired at TS_CAL1_TEMP.
TS_DATA is the actual temperature sensor output value converted by ADC.
Refer to the device datasheet for more information about TS_CAL1 and TS_CAL2 calibration points.

To improve the accuracy of the temperature sensor measurement, each device is individually factory-calibrated by ST. The temperature sensor factory calibration data are stored in the system memory area, accessible in read-only mode, as reported in Table 3.3.

Table 3.3: Calibration Address

Calibration value name	Description	Memory address
TS_CAL1	TS ADC raw data acquired at a temperature of 30 °C (± 5 °C), $V_{DDA} = V_{REF+}$ = 3.0 V (± 10 mV)	0x1FFF 75A8 - 0x1FFF 75A9
TS_CAL2	TS ADC raw data acquired at a temperature of 130 °C (± 5 °C), $V_{DDA} = V_{REF+}$ = 3.0 V (± 10 mV)	0x1FFF 75CA - 0x1FFF 75CB

The value of TS_CAL1 is 1034 and value of TS_CAL2 is 1384, in our case given in system memory address given above.

The experiment to measure Temperature measurement by using internal temperature sensor and comparing it with external device is reported in Table 3.4:

Table 3.4: Temperature using internal sensor Vs External device

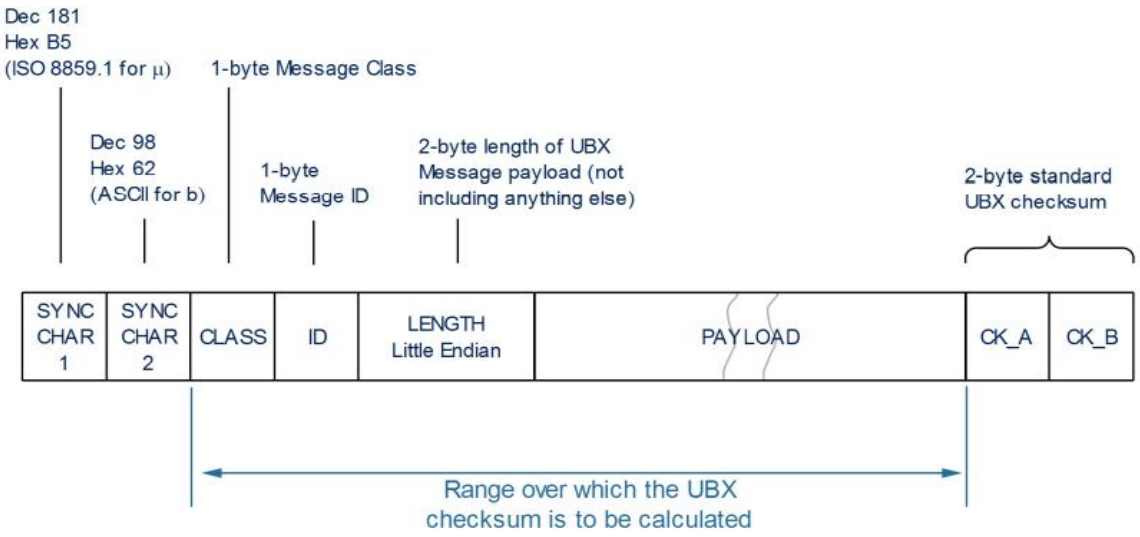
S. No.	Temperature(Internal Sensor) ^{o}C	Temperature(By External device) ^{o}C
1	13.34	13.6
2	14.24	14.1
3	16.4	14.9
4	17.16	15.1
5	18.3	16.3
6	19.0	16.6
7	19.62	17.2
8	20.34	19.4
9	21.5	20.2
10	23.22	21.3
11	24.5	22.1
12	25.28	23.8
13	25.91	24.0
14	26.54	25.4
15	27.85	25.6

Ultimately, they were stabilize at 27.85 and 25.6 respectively, which indicates that there is difference of 1 to 3 ^{o}C. This is because of our microcontroller is performing some task and it is an active component due to which there will be some variation of temperature. It can be compensated by varying the given formula.

Complete code is given at Appendix B

3.5 GPS module

When we interface microcontroller with the GPS module in UART configuration. Large number of data is being received by the microcontroller. Frame after frame is received. Structure of frame is reported in Figure 3.5:



- Every **Frame** starts with a 2-byte Preamble consisting of two synchronization characters: 0xB5 0x62.
- A 1-byte Message **Class** field follows. A Class is a group of messages that are related to each other.
- A 1-byte Message **ID** field defines the message that is to follow.
- A 2-byte **Length** field follows. The length is defined as being that of the payload only. It does not include the Preamble, Message Class, Message ID, Length, or Cyclic Redundancy Check (CRC) fields. The number format of the length field is a Little-Endian unsigned 16-bit integer.
- The **Payload** field contains a variable number of bytes.
- The two 1-byte **CK_A** and **CK_B** fields hold a 16-bit checksum whose calculation is defined below. This concludes the Frame.

Figure 3.5: Frame Structure

Some example of frames that we obtain are following:
$xxRMC,time,status,lat,NS,lon,EW,spd,cog,date,mv,mvEW,posMode,navStatus*cs<CR><LF>
$GPRMC,083559.00,A,4717.11437,N,00833.91522,E,0.004,77.52,091202,,,A,V*57
$xxGLL,lat,NS,lon,EW,time,status,posMode*cs<CR><LF>
$GPGLL,4717.11634,N,00833.91297,E,124923.00,A,A*6E

Message	UBX-NAV-POSLLH					
Description	Geodetic position solution					
Firmware	Supported on: • u-blox 8 / u-blox M8 protocol versions 15, 15.01, 16, 17, 18, 19, 19.1, 19.2, 20, 20.01, 20.1, 20.2, 20.3, 22, 22.01, 23 and 23.01					
Type	Periodic/Polled					
Comment	See important comments concerning validity of position given in section Navigation Output Filters. This message outputs the Geodetic position in the currently selected ellipsoid. The default is the WGS84 Ellipsoid, but can be changed with the message UBX-CFG-DAT.					
Message Structure	Header	Class	ID	Length (Bytes)	Payload	Checksum
	0xB5 0x62	0x01	0x02	20	see below	CK_A CK_B
Payload Contents:						

Byte Offset	Number Format	Scaling	Name	Unit	Description
0	U4	-	iTOW	ms	GPS time of week of the navigation epoch. See the description of iTOW for details.
4	I4	1e-7	lon	deg	Longitude
8	I4	1e-7	lat	deg	Latitude
12	I4	-	height	mm	Height above ellipsoid

Figure 3.6: Longitude & Latitude position in frame

Hence, we need to make a filter of header and class and obtain the correct value of longitude and latitude from large chunk of data based on Figure 3.6.

Complete code is given at Appendix C.

3.6 Bluetooth communication

The RF subsystem is composed of an RF analog front end, BLE and 802.15.4 digital MAC blocks as well as of a dedicated Arm® Cortex®-M0+ microcontroller (called CPU2), plus proprietary peripherals. The RF subsystem performs all of the BLE and 802.15.4 low layer stack, reducing the interaction with the CPU1 to high level exchanges.

STM32WB is a ultra low power Bluetooth low energy (BLE) single-mode network processor that complies with Bluetooth specification v5.2 and supporting master or slave roles.

Bluetooth Low Energy (BLE) wireless technology was developed by the Bluetooth Special Interest Group (SIG) to achieve a very low energy standard operating with a coin cell battery for numerous years. Classic Bluetooth technology has been developed as the standard for wireless communication to replace cables for connecting portable and / or fixed electronics, but it cannot provide the ultimate in battery life due to the relatively complex connection-based behavior and relatively complex procedures for connection. Low-power Bluetooth devices consume only part

of the power of standard Bluetooth products, allowing to connect coin-operated devices to wireless Bluetooth enabled devices.

To be precise, low-power Bluetooth technology is designed to transmit very small packets of data simultaneously, consuming significantly less power than mainstream / enhanced data / high-speed (BR / EDR / HS) devices.

The Bluetooth low energy stack consists of two components:

[1] Controller [2] Host

The Controller includes the physical layer and the link layer. The Host includes the logical link control and adaptation protocol (L2CAP), the security manager (SM), the attribute protocol (ATT), generic attribute profile (GATT) and the generic access profile (GAP). The interface between the two components is called host controller interface (HCI), as reported in Figure 3.7.

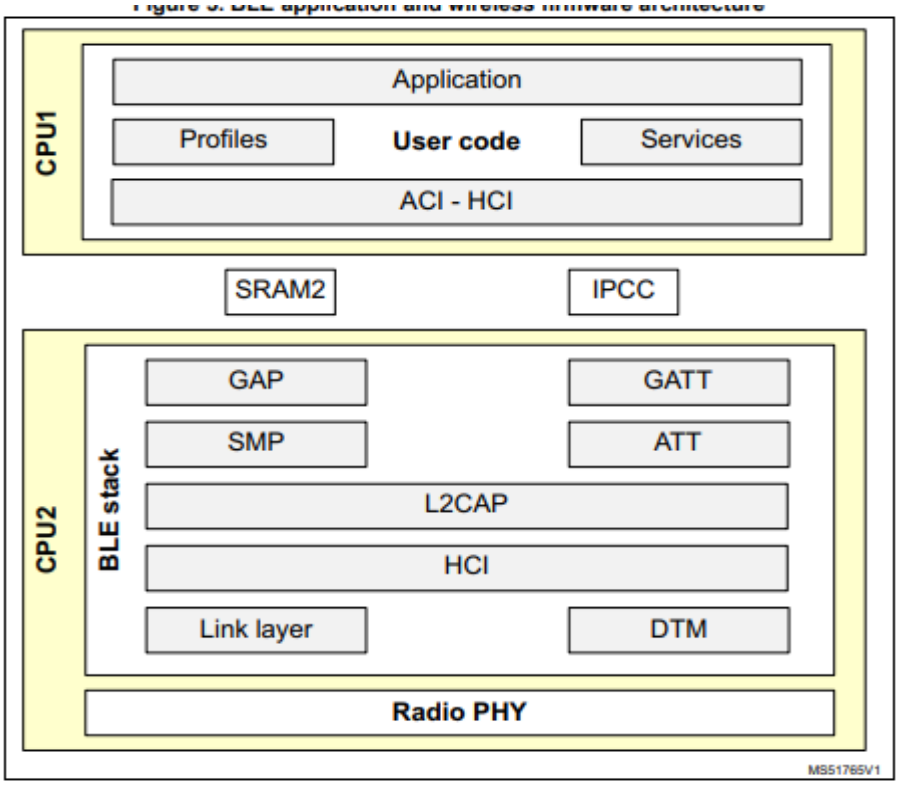


Figure 3.7: BLE application and wireless firmware architecture

Setting up a Bluetooth devices involves first “pairing” the two devices and establishing a connection. The Bluetooth Client component sends the connection request and the Bluetooth Server component accepts the request.

Client/server protocol, as reported in Figure 3.8, forms the basis of data exchange in BLE applications. Server (BLE peripheral) provides data upon request from a client (central device). Server data stored in so-called Attribute Table, which contains a series of record (attributes) of various types.

GATT comes into play when a connection is established. It defines data exchange between two BLE devices. It adds a data model and hierarchy on top of the ATT (by means of concepts called services and characteristics)

A service is a container for logically related data items (e.g. Device Information Service – various information about the device). It characteristics are logically related data items within one service (e.g. Serial Number String and Manufacturer Name String from the Device Information Service). A characteristic consists of a type, a value, some properties, permissions and optionally descriptors. Descriptors either provides additional details or allows configuration of behavior related to the characteristic (e.g. turn on notifications)

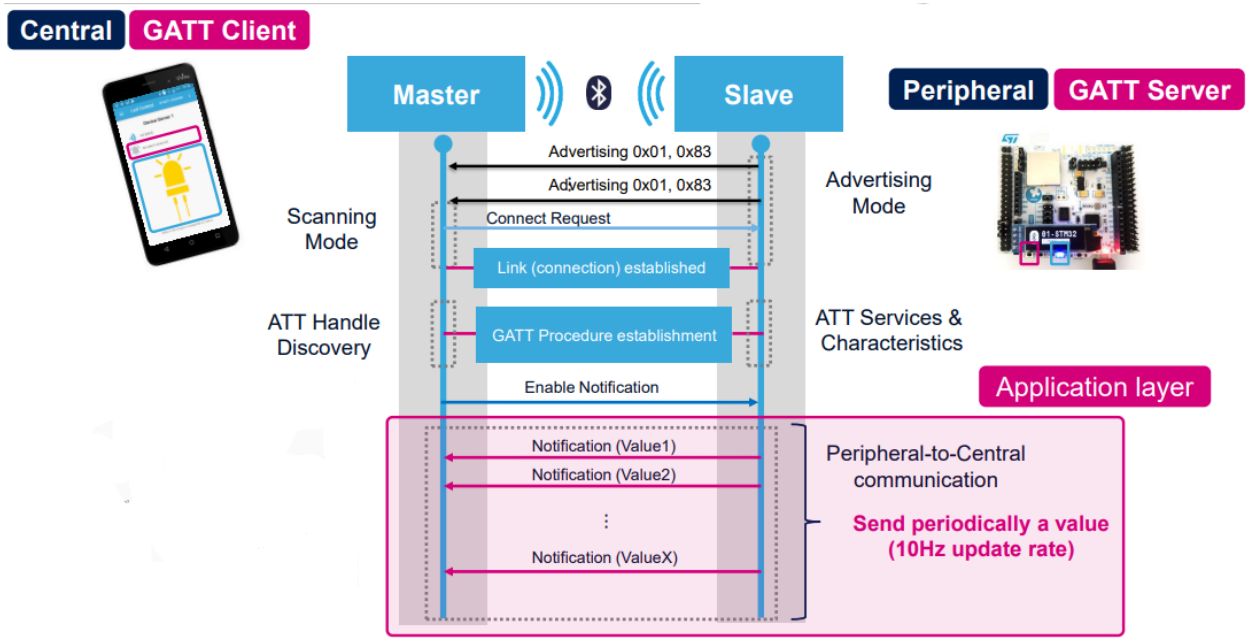


Figure 3.8: BLE communication process

Our project requires performing full duplex communication between client and server. RECEIVER EXTERNO board act as an client, which ask for data whenever we require. Where, on the other hand, TAG - PROVINO board act as a server. It stores all the data inside microcontroller. It periodically advertise and whenever contacted by the client, it executes required task of sending particular data or reset timer etc.

We have performed both ways communication:

1. Sending data from client to sever: We have use STBle app in our mobile phone, which act as a client and have established communication with Nucleo board, which behaves as a server. Through app we have controlled the LED on the Nucleo board, as reported in Figure 3.9.

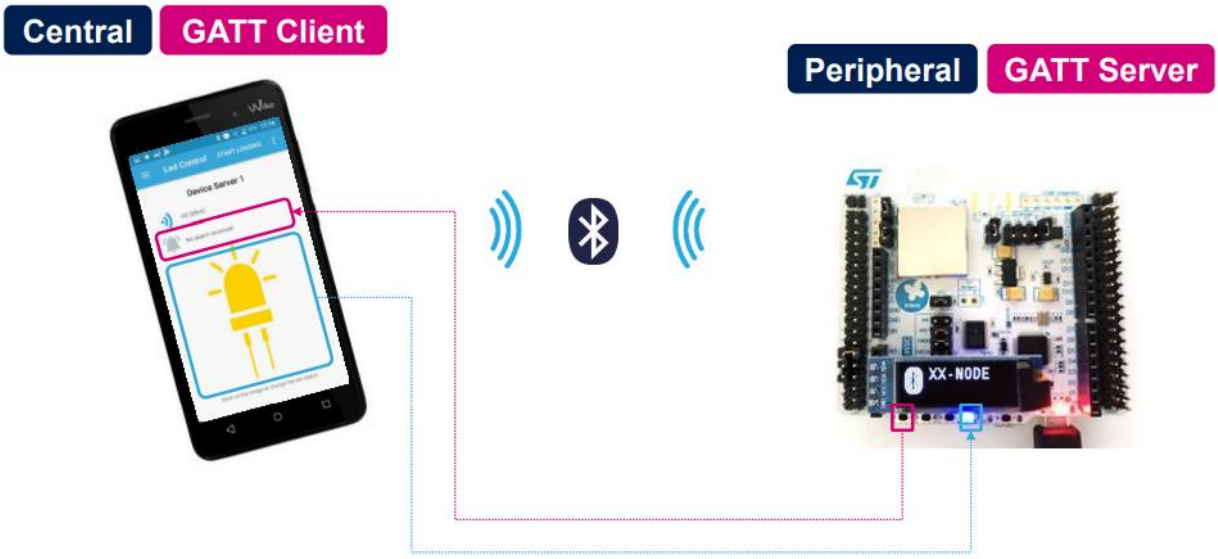


Figure 3.9: Client to Server Communication

2. Sending data from sever to client: In this mode, same as above mobile app act as a client and Nucleo board behaves as a server and we receive temperature value from Nucleo board on mobile app, as reported in Figure 3.10.

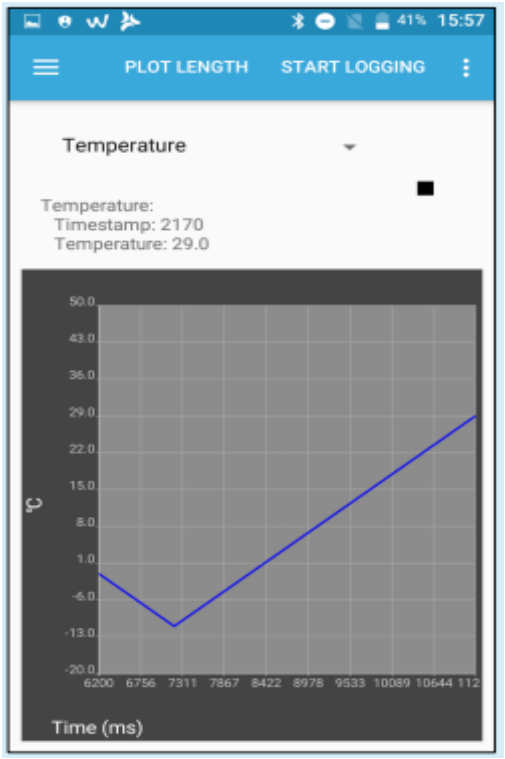


Figure 3.10: Data received by APP

Chapter 4

Conclusion

We have final Prototype of all three device: TAG-Provino board, Receiver Esterno board, and Cassaforma board.

First device, TAG-Provino board is collecting data from temperature sensor and saving it inside microcontroller. The microcontroller consist of two cores: Arm® Cortex®-M4 core and Arm Cortex®-M0+ core, where Arm® Cortex®-M4 is primarily responsible for performing the function and Arm Cortex®-M0+ is dedicated for real time radio layer. When not performing any task, the microcontroller goes into standby mode which consumes less than 2 uA of power. Wake up receiver helps system to wake up from standby mode into run mode. Microcontroller also saves data with timestamp of event of opening of material box or tempering of its material detected by the Shock sensor and the Photodiode. The size of TAG-Provino board is only 5 x 2.4 mm.

Figure 4.1: TAG-Privono board

Second device, CASSAFORMA board consist of same microcontroller as TAG-Provino board. So, it brings same ultra low power consumption ability in this board as well. GPS module requires special attention while designing board as no other component except three capacitors should be placed near it. Also, large number of Vias are needed to be placed right beneath it and no other component should be present below the GPS module. Location data is periodically collected from the GPS module and stored in the microcontroller. Further, Location data is transmitted to the TAG-Provino board.

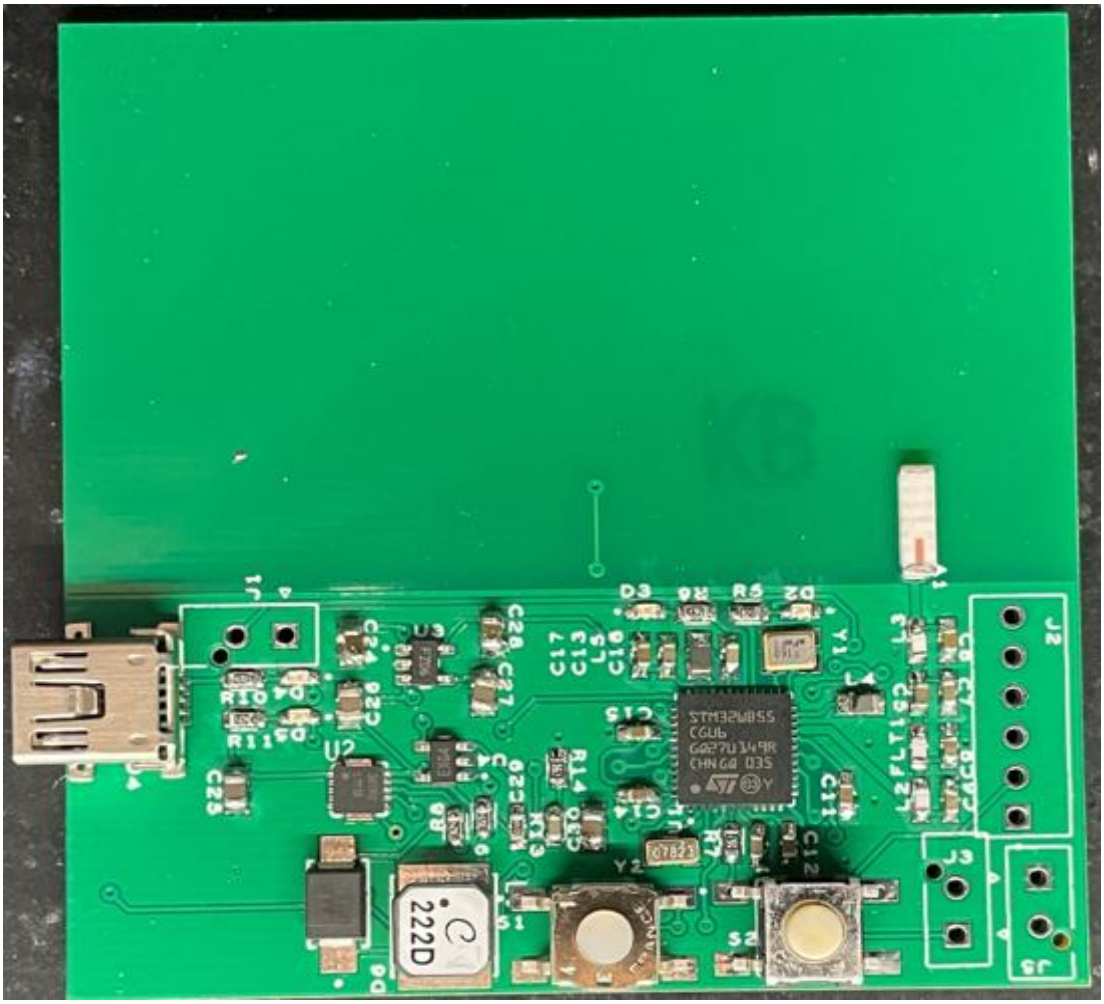


Figure 4.2: Receiver Esterno board

Third device, RECEIVER ESTERNO board has similar function of that of a Gateway. It act as a bridge between whole system and external world. It consist of the wake up generator, which generates electro-magnetic signal of more than 1 MHz to wake up TAG-Provino board from the deep sleep mode. The communication among the devices and calibration of the time is responsibility of this board. Not only communication among the boards but communication with external device is done through this board that is why it has dual functionality.

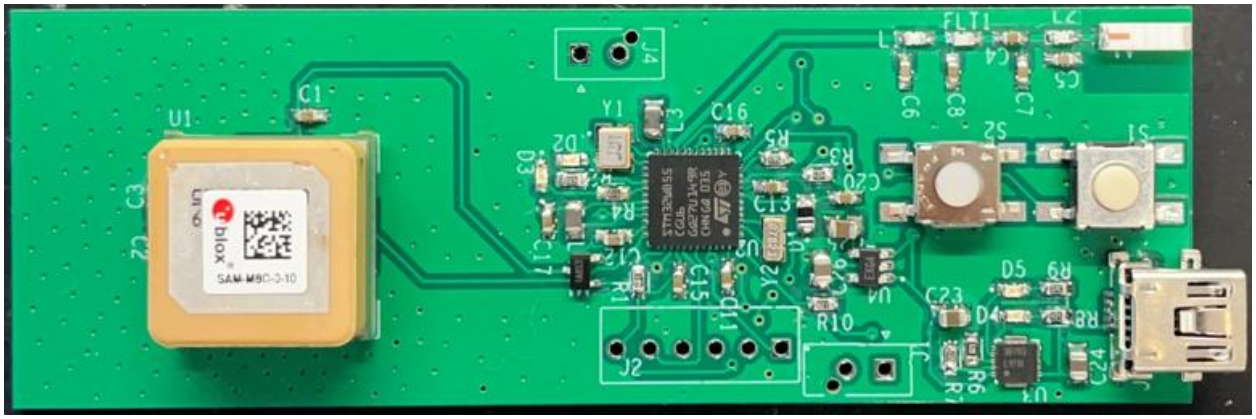


Figure 4.3: Cassaforma board

APPENDIX

Appendix A

Current consumption measurement

```
#include "main.h"
void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{

  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();

  HAL_Delay(2000);

  while (1)
  {
  }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Macro to configure the PLL multiplication factor */
  __HAL_RCC_PLL_PLLM_CONFIG(RCC_PLLM_DIV4);
  /** Macro to configure the PLL clock source */
  __HAL_RCC_PLL_PLLSOURCE_CONFIG(RCC_PLLSOURCE_MSI);
  /** Configure LSE Drive Capability  */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
  /** Configure the main internal regulator output voltage */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure. */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSE
                                   |RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.MSICalibrationValue = RCC_MSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_11;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /** Configure the SYSCLKSource, HCLK, PCLK1 and PCLK2 clocks dividers */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK4|RCC_CLOCKTYPE_HCLK2
                              |RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.AHBCLK2Divider = RCC_SYSCLK_DIV2;
  RCC_ClkInitStruct.AHBCLK4Divider = RCC_SYSCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the peripherals clocks */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_SMPS|RCC_PERIPHCLK_USART1
                              |RCC_PERIPHCLK_USB;
  PeriphClkInitStruct.PLLSAI1.PLLN = 8;
  PeriphClkInitStruct.PLLSAI1.PLLP = RCC_PLLP_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLQ = RCC_PLLQ_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLR = RCC_PLLR_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_USBCLK;
  PeriphClkInitStruct.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInitStruct.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
  PeriphClkInitStruct.SmpsClockSelection = RCC_SMPSCLKSOURCE_HSI;
  PeriphClkInitStruct.SmpsDivSelection = RCC_SMPSCLKDIV_RANGE0;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Enable MSI Auto calibration */
  HAL_RCCEx_EnableMSIPLLMode();
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LD2_Pin|LD3_Pin|LD1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : LD2_Pin LD3_Pin LD1_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|LD3_Pin|LD1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : B2_Pin B3_Pin */
  GPIO_InitStruct.Pin = B2_Pin|B3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}
```

Appendix B

Temperature measurement

```
#include "main.h"
#include <stdio.h>

#define TEMP130_CAL_VALUE              ((uint16_t*)((uint32_t)0x1FFF75CA))
#define TEMP30_CAL_VALUE               ((uint16_t*)((uint32_t)0x1FFF75A8))
#define TEMP130                                                    130.0f
#define TEMP30                                                     30.0f
float CALIBRATION_REFERENCE_VOLTAGE = 3.3F;
float REFERENCE_VOLTAGE = 3.0F;

float temperature;
float sensorValue;
float div;                // REFERENCE_VOLTAGE/CALIBRATION_REFERENCE_VOLTAGE;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);



int main(void)
{
  HAL_Init();

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

  while (1)
  {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, 1);
        HAL_Delay(1000);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, 0);
        HAL_Delay(1000);
        HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
        HAL_ADC_Start_IT(&hadc1);
        HAL_Delay(100);
```

```
  }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Macro to configure the PLL multiplication factor */
  __HAL_RCC_PLL_PLLM_CONFIG(RCC_PLLM_DIV1);
  /** Macro to configure the PLL clock source */
  __HAL_RCC_PLL_PLLSOURCE_CONFIG(RCC_PLLSOURCE_MSI);
  /** Configure LSE Drive Capability */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
  /** Configure the main internal regulator output voltage */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure. */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSE
                                   |RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.MSICalibrationValue = RCC_MSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_8;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure the SYSCLKSource, HCLK, PCLK1 and PCLK2 clocks dividers */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK4|RCC_CLOCKTYPE_HCLK2
                              |RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.AHBCLK2Divider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLK4Divider = RCC_SYSCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the peripherals clocks */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_SMPS|RCC_PERIPHCLK_USART1
                              |RCC_PERIPHCLK_USB|RCC_PERIPHCLK_ADC;
  PeriphClkInitStruct.PLLSAI1.PLLN = 6;
  PeriphClkInitStruct.PLLSAI1.PLLP = RCC_PLLP_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLQ = RCC_PLLQ_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLR = RCC_PLLR_DIV2;
```

```
  PeriphClkInitStruct.PLLSAI1.PLLSAI1ClockOut =
RCC_PLLSAI1_USBCLK|RCC_PLLSAI1_ADCCLK;
  PeriphClkInitStruct.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInitStruct.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
  PeriphClkInitStruct.AdcClockSelection = RCC_ADCCLKSOURCE_PLLSAI1;
  PeriphClkInitStruct.SmpsClockSelection = RCC_SMPSCLKSOURCE_HSI;
  PeriphClkInitStruct.SmpsDivSelection = RCC_SMPSCLKDIV_RANGE0;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Enable MSI Auto calibration
  */
  HAL_RCCEx_EnableMSIPLLMode();
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LD2_Pin|LD3_Pin|LD1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC4 */
  GPIO_InitStruct.Pin = GPIO_PIN_4;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin LD3_Pin LD1_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|LD3_Pin|LD1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : B2_Pin B3_Pin */
  GPIO_InitStruct.Pin = B2_Pin|B3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI4_IRQn);

}

/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
```

```
{
      if(HAL_ADC_PollForConversion(&hadc1, 100) == HAL_OK)
        {
        div= CALIBRATION_REFERENCE_VOLTAGE/REFERENCE_VOLTAGE;
        adcCalTemp30C = (float)(*TEMP30_CAL_VALUE);
        adcCalTemp130C = (float)(*TEMP130_CAL_VALUE);

        sensorValue = (float)HAL_ADC_GetValue(&hadc1);
        HAL_ADC_Stop(&hadc1);

        temperature = (float)((sensorValue * div ) - adcCalTemp30C)/(adcCalTemp130C
- adcCalTemp30C) * (130.0F - 30.0F) + 30.0F ;

        }
       else
        {
        temperature = -273;
        }

}
```

Appendix C

GPS coordinate measurement

```
#include "main.h"
#include "string.h"

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);

uint8_t Rx_data[70];
uint64_t Sdata[70];
uint8_t LL[20];

int main(void)
{

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

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

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();

  HAL_UART_Receive_IT(&huart1, Rx_data, 70);
  HAL_Delay(800);
```

```
  while (1)
  {

      HAL_UART_Receive_IT(&huart1, Rx_data, 70);
      HAL_Delay(500);

  }

}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Macro to configure the PLL multiplication factor */
  __HAL_RCC_PLL_PLLM_CONFIG(RCC_PLLM_DIV1);
  /** Macro to configure the PLL clock source */
  __HAL_RCC_PLL_PLLSOURCE_CONFIG(RCC_PLLSOURCE_MSI);
  /** Configure LSE Drive Capability */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
  /** Configure the main internal regulator output voltage */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure. */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSE
                                   |RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.MSICalibrationValue = RCC_MSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure the SYSCLKSource, HCLK, PCLK1 and PCLK2 clocks dividers */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK4|RCC_CLOCKTYPE_HCLK2
                              |RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.AHBCLK2Divider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLK4Divider = RCC_SYSCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /** Initializes the peripherals clocks */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_SMPS|RCC_PERIPHCLK_USART1
                              |RCC_PERIPHCLK_USB;
  PeriphClkInitStruct.PLLSAI1.PLLN = 24;
  PeriphClkInitStruct.PLLSAI1.PLLP = RCC_PLLP_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLQ = RCC_PLLQ_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLR = RCC_PLLR_DIV2;
  PeriphClkInitStruct.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_USBCLK;
  PeriphClkInitStruct.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInitStruct.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
  PeriphClkInitStruct.SmpsClockSelection = RCC_SMPSCLKSOURCE_HSI;
  PeriphClkInitStruct.SmpsDivSelection = RCC_SMPSCLKDIV_RANGE0;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN Smps */

  /* USER CODE END Smps */
  /** Enable MSI Auto calibration */
  HAL_RCCEx_EnableMSIPLLMode();
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_7B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.Init.ClockPrescaler = UART_PRESCALER_DIV1;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetTxFifoThreshold(&huart1, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetRxFifoThreshold(&huart1, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_DisableFifoMode(&huart1) != HAL_OK)
  {
    Error_Handler();
  }

}
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LD2_Pin|LD3_Pin|LD1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin LD3_Pin LD1_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|LD3_Pin|LD1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : B2_Pin B3_Pin */
  GPIO_InitStruct.Pin = B2_Pin|B3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
const unsigned char GLL_HEADER[2] = { 0xF0, 0x01 };
const unsigned char BLL_HEADER[7] = { 36,71,78,82,77,67,44 };
unsigned char comp[7], pos[7];
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){

	__NOP();
	int a=0,b=0,c=0,j=0, k=0;

	for(a; a<70;a++){

		if(BLL_HEADER[b]==Rx_data[a])
		{
		comp[c]=BLL_HEADER[b];
		pos[c]=a;
		c++;
		b++;
		if(c==7){
			k= a;
			}
		}

}
	k=k+8;//+9
	for ( j=0; j<20; j++){
	LL[j] = Rx_data[k];
```

```
        k++;
        }

}
```

REFERENCES

[1] RM0434 Reference manual link: https://www.st.com/resource/en/reference_manual/dm00318631-multiprotocol-wireless-32bit-mcu-armbased-cortexm4-with-fpu-bluetooth-lowenergy-and-802154-radio-solution-stmicroelectronics.pdf

[2] UM2435 User manual link: https://www.st.com/resource/en/user_manual/dm00517423-bluetooth-low-energy-and-802154-nucleo-pack-based-on-stm32wb-series-microcontrollers-stmicroelectronics.pdf

[3] STM32CubeIDE software link: https://www.st.com/en/development-tools/stm32cubeide.html

[4] Electronic – Reading internal temperature sensor STM32 link: https://itectec.com/electrical/electronic-reading-internal-temperature-sensor-stm32/

[5] Youtube video link: https://www.youtube.com/watch?v=CpBZjWBAtpw&pp=sAQA

[6] Calibration link: https://stackoverflow.com/questions/56493262/how-to-measure-know-precisely-the-adc-reference-voltage-on-stm32l052k6t6

[7] VREF formula link: https://community.st.com/s/question/0D50X00009XkYIWSA3/stm32-how-to-use-vref-to-calculate-the-actual-vdda-value

[8] Controllerstech link: https://controllerstech.com/low-power-modes-in-stm32/

[9] Youtube link: https://www.youtube.com/watch?v=UtkszckecV8&pp=sAQA

[10] Colpitts link: https://www.elprocus.com/colpitts-oscillator-circuit-working-and-applications/

[11] Circuit link: https://learnabout-electronics.org/Oscillators/osc24.php

[12] Colpitt's oscillator calculator link: https://www.calctown.com/calculators/colpitt-oscillator

[13] Inductor calculator link: https://www.allaboutcircuits.com/tools/coil-inductance-calculator/

[14] Online Simulator link: https://www.multisim.com/

[15] Receiver Circuit link: https://electronics.stackexchange.com/questions/216681/build-remote-wireless-wake-up-with-attiny85