Research Project

Wearable low power bluetooth beacon

27.03.2025

Kézia MARCOU - Lïam LOTTE

École des Mines de Saint Etienne - ISMIN speciality

Contents

1. Abstract

The IoT (Internet of Things)[1] market has grown dramatically over the past 10 years. In 2020, the number of connected devices was estimated at 10 billion, driven by the expansion of low-cost mobile networks, the rise of off-the-shelf components[2] and the widespread adoption of smartphones [1].

At the same time, flexible electronics—the ability to manufacture electronic components on flexible materials—has gained traction due to its applications in biomedical technology, security, and IoT wearables. Reflecting this trend, the wearable technology market was valued at USD 854.53 million in 2022 and is projected to reach USD 6,675.99 million by 2030 [2].

One of the key challenges in flexible electronics and embedded systems is power efficiency. In wearable applications, short battery life is generally undesirable. A major issue is that many IoT systems consume significant power due to wireless signal transmission and reception [3]. For example, a well-known microcontroller, the ESP32 by Espressif, requires a minimum current supply of 500mA [4], making it unsuitable for ultra-low-power applications that rely on small batteries or energy harvesting.

To address this issue, certain chips have been developed to acquire and transmit signals using low-power protocols. One of the most widely adopted solutions is Bluetooth Low Energy (BLE), valued for its versatility, low power consumption, and compatibility with a wide range of connected devices, including smartphones and smartwatches. Since these beacons are specifically designed for this purpose, they are significantly more efficient.

The goal of the project explained in this report is to use a Bluetooth beacon (the IN100 from *InPlay*) with an accelerometer (BMA400 from *Botch*) on a flexible substrate to characterize its performance and identify potential issues with this type of device.

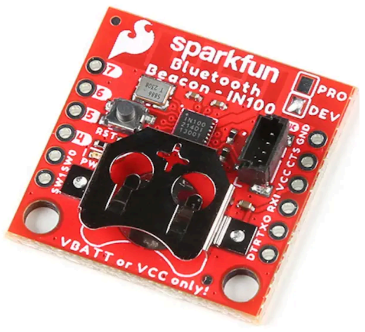

Figure 1: Developer board of the IN100 by Sparkfun

[1]Internet of Things: “Objects with computing devices in them that are able to connect to each other and exchange data using the internet,” according to the Cambridge English Dictionary.

[2]Components that are readily available to everyone, not just major electronics manufacturers, as was the case in the past.

2. Hardware specifications

2.1. Design

2.1.1. Schematic

The schematic design of the circuit is the core of the project. The design was mainly inspired by the IN100 [5] and BMA400 [6] datasheet recommandations and the Sparkfun's datasheets [7], [8] for their development board. As the aim is to make a minimal functional circuit, some features of the Sparkfun's boards were deleted because of their non essential aspects.

The schematic is split per function in the global circuit and is composed of :

- A supply circuit
- A bluetooth antenna
- A bluetooth beacon
- An accelerometer
- A quartz oscillator
- BLE[3] Setup Ports
- Debug Ports

The components are connected together following this schematic :

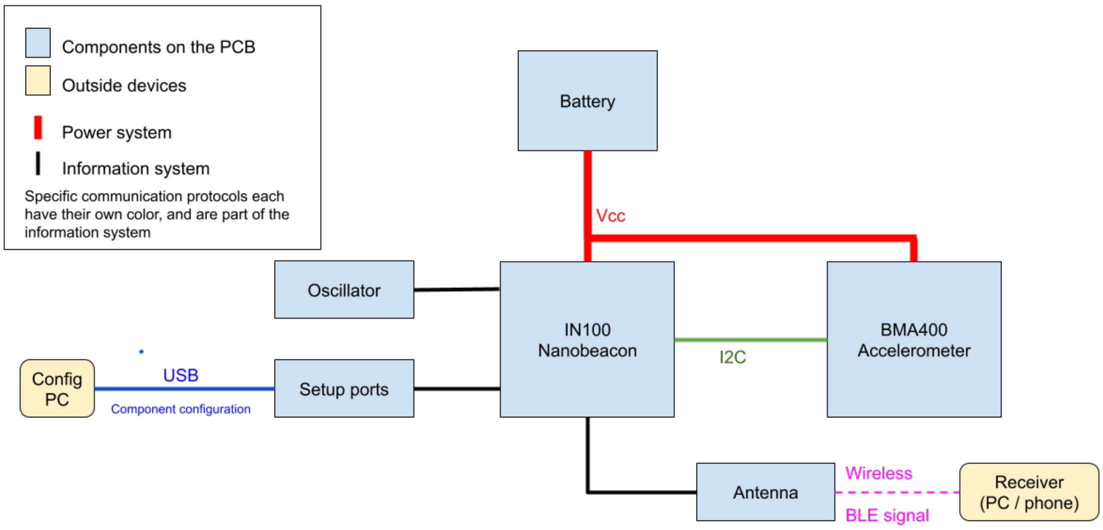

Figure 2: Simplified schematic of the complete circuit

We are going to detail each part of the circuit.

[3]Bluetooth Low Energy, a variation of Bluetooth optimized for low power consumption

- Supply circuit

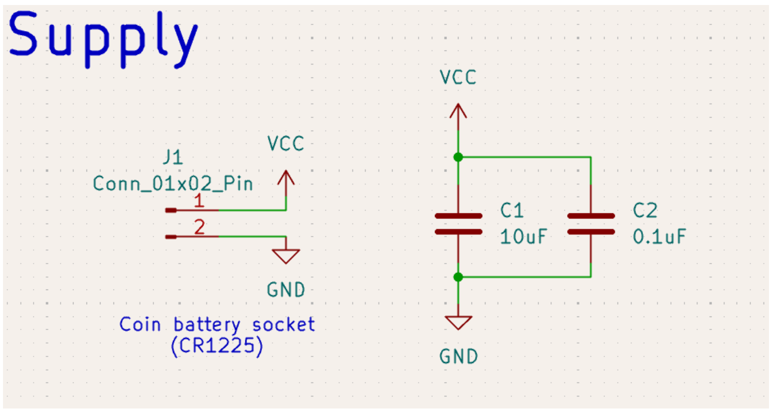

Figure 3: Schematic of the supply circuit

At the end, the power supply will be provided by a coin battery, specifically a CR1225 model. This battery has a voltage of 3V and a maximum continuous discharge of 2mA. While small, this is sufficient to power all the low-power chips. The major advantage of coin batteries is their widespread availability (found in most supermarkets) and their compact form factor (12.5 mm × 2.5 mm).

For now, power efficiency will not be optimized, as the primary goal is to achieve functionality first, with future upgrades focused on improving power efficiency.

- Bluetooth antenna

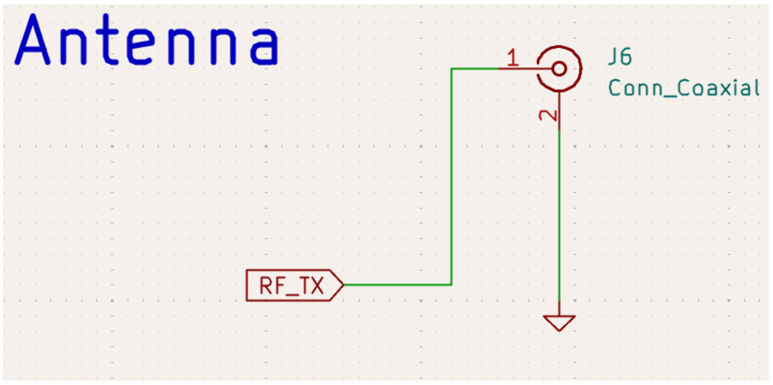

Figure 4: Schematic of the antenna

The Bluetooth antenna is a classic design. The antenna's input signal comes from the Bluetooth beacon. We plan to have two versions of the circuit: one with an antenna engraved on the PCB and another with a connector for an external ceramic antenna. This setup allows for system debugging by determining whether potential issues originate from the antenna or other parts of the system.

- Bluetooth Beacon

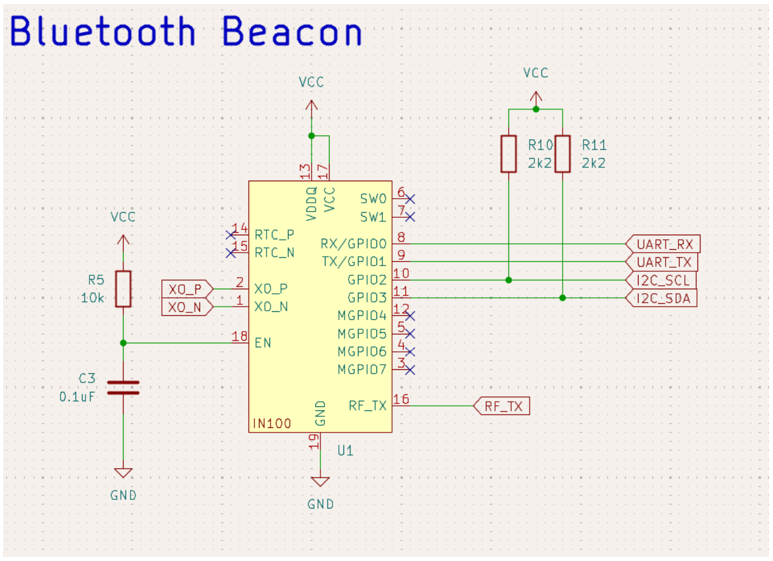

Figure 5: Schematic of the BLE Beacon (IN100)

The Bluetooth beacon is the core component of the circuit, responsible for acquiring and transmitting the required data.

The chip has two power supply input pins (VDDQ and VDD): one for the system's logic and the other for the input/output pins. For simplicity, both are set to the same voltage. We don't need special voltage regulation because the component has its own internal regulator.

The XO_N and XO_P pins are used to connect the clock, which determines the system's operating speed.

The EN (Enable) pin controls whether the chip is turned on or off. Since we want the chip to remain on at all times, we use a pull-up resistor[4] to keep the voltage high on this pin.

A key pair of pins is UART_RX and UART_TX which operate using the UART[5] protocol. These pins are essential for programming and configuring the IN100 chip.

We also utilize the I²C pins, which follow the I²C bus protocol[6], to communicate with the accelerometer sensor. This setup allows for potential expansion, as

[4]A pull-up resistor is a resistor that connects a signal line to a high voltage level (typically VCC) to ensure that the line maintains a default logic high state when no active component is driving it.

[5]UART (Universal Asynchronous Receiver/Transmitter) is a serial communication protocol used for direct, low-latency data exchange between devices.

[6]I²C is a synchronous, two-wire protocol (SDA for data, SCL for clock) that allows a master to communicate with multiple slave devices using unique addresses. It enables efficient, multi-device communication in embedded systems.

multiple sensors can be integrated into the same bus for future applications. The resistors for the EN pin are pull-up resistors.

To improve power efficiency, it would be beneficial to use the switch power pins (SW0 and SW1) to enable the sensors only when the IN100 starts an acquisition. This is especially important for I²C, where the pull-up resistors power. Powering the resistors only when needed could provide significant benefits. However, for the sake of layout simplicity, we have chosen not to implement this option in our first designs.

The chip allows also to use an extern startup 32.768kHz clock (pins RTC_P and RTC_N). As it has an intern and functionnal clock, for simplicity and power efficiency we have chosen not to implement it. All the other pins are GPIO[7] that we do not currently use.

- Accelerometer

The accelerometer is connected to the Bluetooth beacon via the I²C bus of the IN100. Since it operates with its own clock, an external one is not needed.

The power supply comes directly from the VCC provided by the battery. Like the IN100, the BMA400 has its own voltage regulator, which simplifies the circuit by providing two power circuits: one for the GPIO and another for the logic circuit.

The CSB pin allows users to select between the SPI and I²C protocols. Setting it high enables I²C mode, while pulling it low (connecting it to ground) enables SPI mode.

On the other hand, when configured in I²C mode, the SDO pin determines the device address. Connecting it to GND sets the default address (0x14), while connecting it to VDDIO selects the alternative address (0x15). This feature allows more complex systems to use two accelerometers on the same bus. In our case, since we have only one accelerometer, we use the default address.

The INT x pins are used for handling interrupts, but they are not needed for our application.

[7]General Purpose Input Output

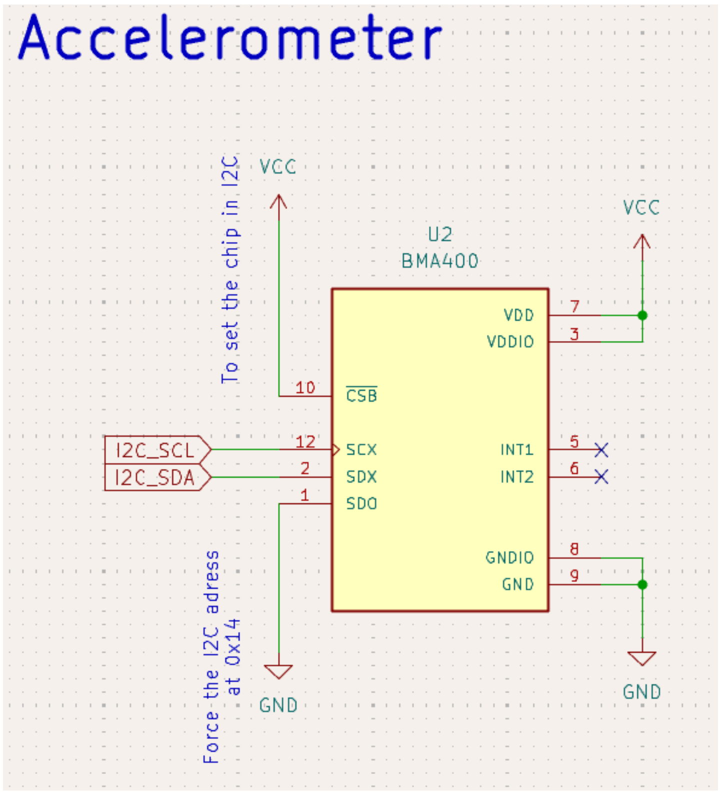

Figure 6: Schematic of the Accelerometer (BMA400)

- Quartz oscillator

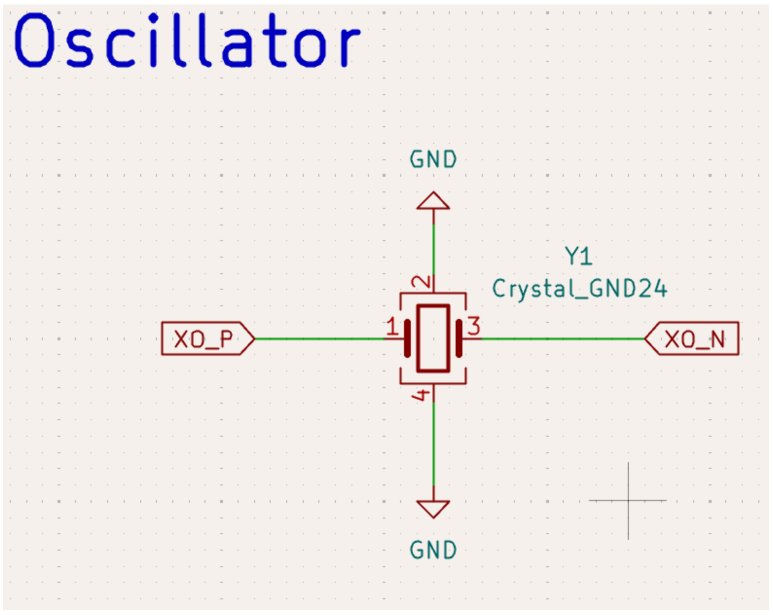

Figure 7: Schematic of the Oscillator

The oscillator for the IN100 must have a frequency between 16 and 48 MHz. We have chosen the typical frequency of 26 MHz.

The oscillator should have a precision of ±40 ppm, meaning the frequency can vary by 1 part per million from the nominal value. At 26 MHz, the maximum error is 1040 Hz. It should also have a load capacitance between 6 and 8 pF and an equivalent series resistance between 60 and 100 Ω.

Based on these specifications, we have selected the CMS 26 MHz oscillator from Murata, which meets our requirements.

- Setup Ports

The setup ports have to be connected to a USB adaptator and allows to setup the IN100 chip thanks to the UART protocol.

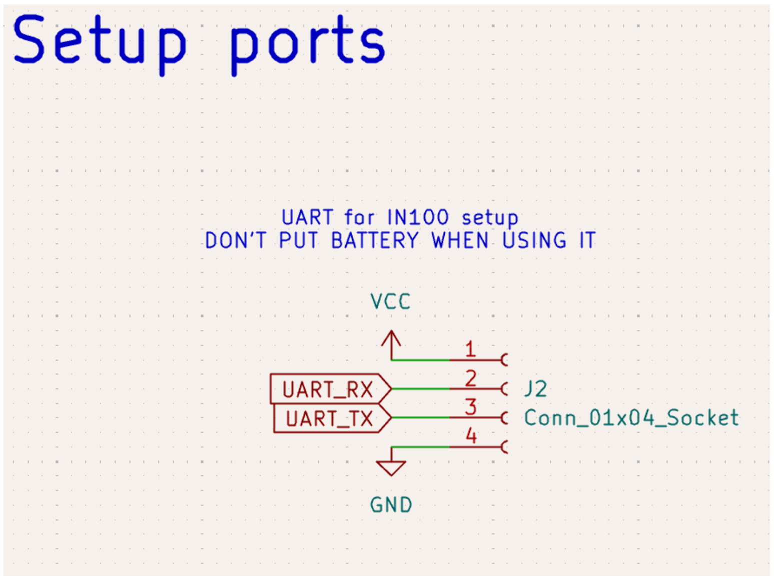

Figure 8: Schematic of Setup Ports

- Debug Ports

The debug ports are here to check the I²C communication and the right behaviour of the oscillator. These test points allow us to easily measure our signals using an oscilloscope or other similar measurement tools.

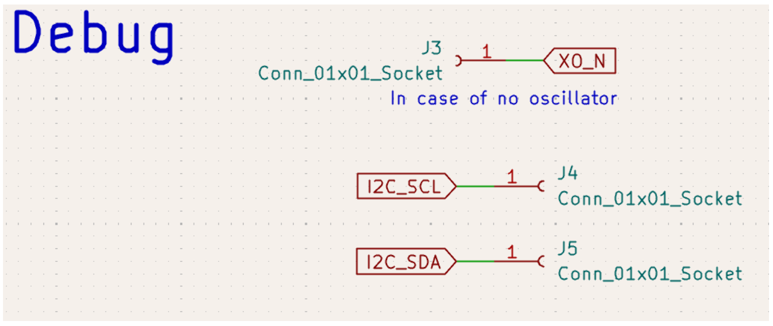

Figure 9: Schematic of Debug Ports

2.1.2. Layout

First Version (V1)

The layout has one important constraint: it must be one-sided only due to the limitations of flexible PCB manufacturing machines. To comply with the netlist generated by the schematic file, we use 0-ohm resistors as jumpers to cross over wires.

All these constraints lead to this first version of the layout.

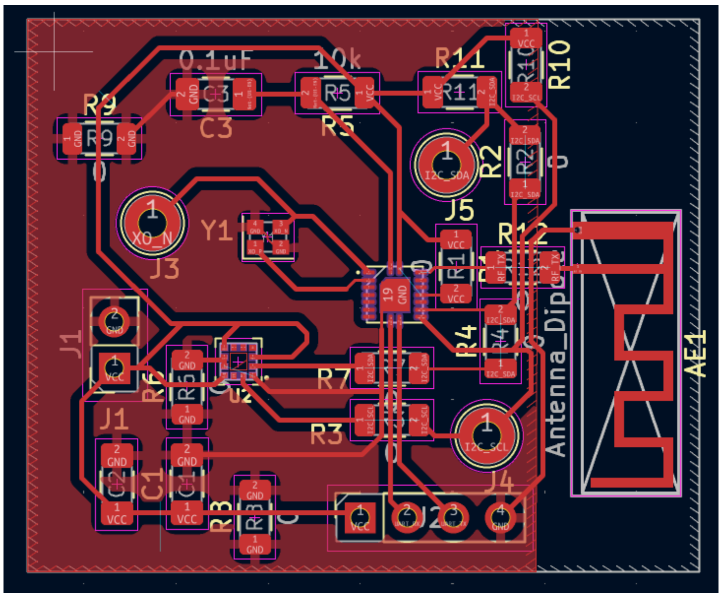

Figure 10: Layout V1

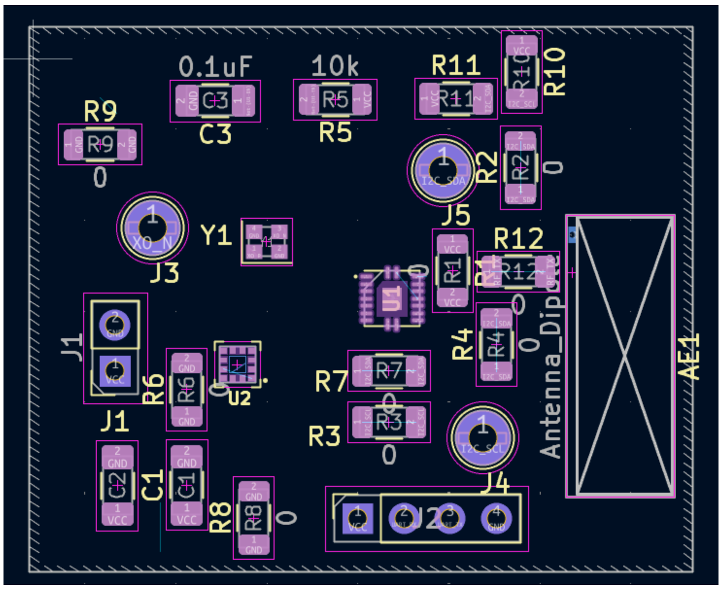

Figure 11: Layout V1 - Components Map

This version has a 35 × 31 mm surface but has some issues. The large number of 0-ohm resistors adds complexity to the PCB, which is rarely beneficial for stability and electrostatics. The main issue is the 0-ohm resistor crossing over two I²C traces that carry the RF signal. Another issue is that, to simplify the circuit, this version does not implement pull-up resistor power consumption optimization by activating them only when needed via the SW0 pin.

However, this is the first version that has been tested and described in bench experiments.

Second Version (V2)

The second layout significantly simplifies the circuit and should offer better performance.

This version is also smaller (35 × 26 mm). It includes only two 0-ohm resistors and does not have any over RF traces. Additionally, the pull-up resistors are supplied through SW0, meaning they are powered only when the IN100 is active.

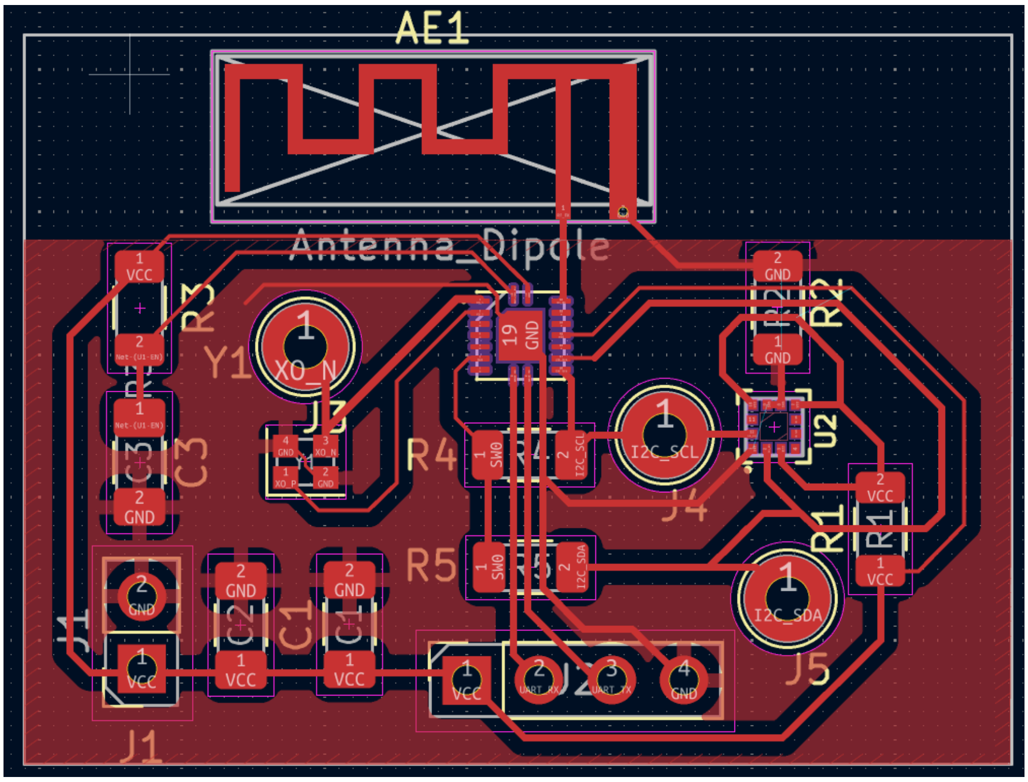

Figure 12: Layout V2

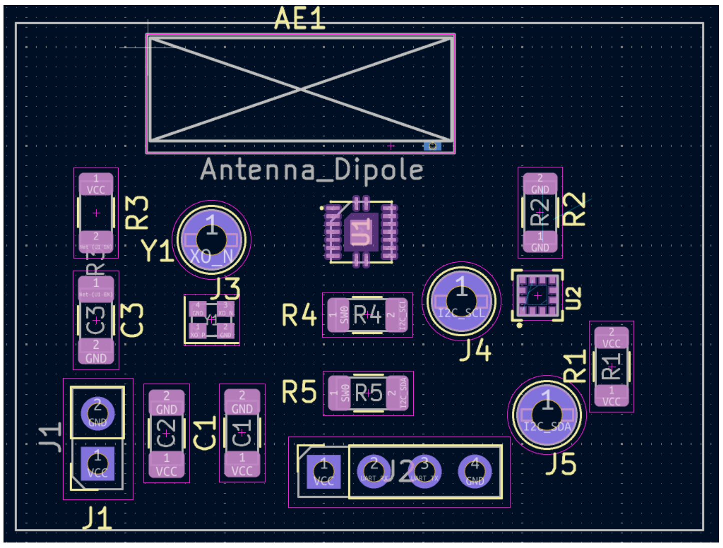

Figure 13: Layout V2 - Components Map

This layout is promising but has not been tested yet because of a lack of time.

2.2. Bluetooth Beacon Configuration

2.2.1. Global behaviour setup

An important factor in the system's power efficiency is the configuration of the Bluetooth beacon. This configuration dictates the I²C signals to that control the accelerometer and the frequency of the BLE signals, which directly affect the power consumption.

The IN100 was configured to acquire acceleration data 1 times per second and transmit it via Bluetooth. After each transmission, the IN100 and the BMA400 should enter sleep mode for 1000 ms.

2.2.2. I²C signals

The I²C signals sent to the BMA400 should be stored in the IN100's memory. This write operation is irreversible.

We want with our I²C signal to :

- Turn the BMA400 in normal mode

```
// Choose to set the ACC_CONFIG0 register
// that allow to configure the accelerometer
tx 0x19

// Write in the ACC_CONFIG0 register to
// set the normal mode
tx 0x2
```

- Wait that the BMA400 turn into normal mode

```
// Wait 1ms
wait 1 4e
```

- Claim the x data from the accelerometer

```
// Claim to the BMA the content of
// the ACC_X_LSB register
tx 0x4
```

- Read the x data from the accelerometer

```
// Wait to receive the 2 bytes
// from ACC_X_LSB (0x4) register and
// the following register which
// is the ACC_X_MSB (0x5) register
rx
rx
```

- Claim the y data from the accelerometer

```
tx 0x6
```

- Read the y data from the accelerometer

```
rx
rx
```

- Claim the z data from the accelerometer

```
tx 0x8
```

- Read the z data from the accelerometer

```
rx
rx
```

- Turn the BMA400 in sleep mode

```
// Choose to set the ACC_CONFIG0 register
// that allow to configure the accelerometer
tx 0x19

// Write in the ACC_CONFIG0 register to
// set the sleep mode
tx 0x0
```

An untested configuration for the IN100 to optimize the system's low power usage is available on GitHub with the low_power_config.cfg file.

2.3. Performances evaluation

2.3.1. Theoretical performances

In this theoretical part, we want to quantify the power needed to supply the system and to know the battery's lifetime. In this part we're going to take the worst cases to make sure the battry is big enough, which means a part of the power will be consumed by pull-up resistors, a thing that can be optimized thanks to the IN100's SW0 and SW1 ports as seen in previous parts.

The IN100 can be switched to sleep mode[8] after each emission and control the BMA400 to switch it to sleep mode too. I we take 1 measures per second, thanks to the example given by InPlay, the manufacturer of the IN100, we can determine a duty cycle of 3,50%. Also thanks to experiments, we determined that the I^2C transmission was 4ms long.

At the moment we can determine the following caracteristics according to datasheets and calculations.

Data	**Value**
Global Circuit Voltage	3V
Global Duty Cycle	3,50%
I^2C Duty Cycle	0.4%

Table 1: Global theoretical electric caracteristics.

Component :	**IN100**
Normal Current Consumption	13.8 μA
Sleep Mode Current Consumption	625 nA
Average Current Consumption	1.06 μA

Table 2: IN100 theoretical electric caracteristics.

Component :	**BMA400**
Normal Current Consumption	14.5 μA
Sleep Mode Current Consumption	160 nA
Average Current Consumption	1.33 μA

Table 3: IN100 theoretical electric caracteristics.

[8]A mode that allow to system to run with less power.

Component :	**I²C Pull Up Resistor**
Resistor Value	10 kΩ
Current Consumption per Resistor	300 μA
Total Current Consumption	600 μA
Average Current Consumption	1,44 μA

Table 4: I²C Pull Up Resistor theoretical electric caracteristics.

Component :	**Battery**
Battery Capacity	50 mAh
Average Current Consumption	3.85 μA
Battery Lifespan	12 974 h 541 days 1,48 years

Table 5: Theoretical battery performances.

2.3.2. On bench performances

- Voltage Limit Test

With our first version of the PCB, we wanted to ensure that it would be robust in relation to the voltage used. So with the IN100 and BMA400, we theoretically ensured an operating voltage between 1.7V (minimum operating voltage of the BMA400) and 3.6V (maximum voltage of the IN100 and BMA400).

By experimentation, we determined that the system always operated between 1.5V and 3.3V. We didn't test higher voltages to avoid damaging our (single) test PCB.

- Current Consumption

With the 1st version of our PCB, we wanted to test the power consumption of our PCB (previously estimated at 3.85 μA). The tests were made with a voltage of 1.8V.

Experience has shown us that spikes in tension are greater than expected, but this can be explained. RF emission was not calculated during our theoretical phase, leading to voltage peaks of 8mA. This is probably due to various disturbances on the transmission line (with the I²C traces behind). Nevertheless, these problems should be reduced with the latest version of the PCB.

So we end up with an average board consumption of 45uA. With a 50mAh battery, this would give us an autonomy of 46 days. Unfortunately, this is insufficient for a number of home automation systems. It would therefore be interesting to observe the performance of a less noisy PCB on its transmission line.

Component :	Battery
Battery Capacity	50 mAh
Average Current Consumption	45 μA
Battery Lifespan	46 days

Table 6: In practice battery performances.

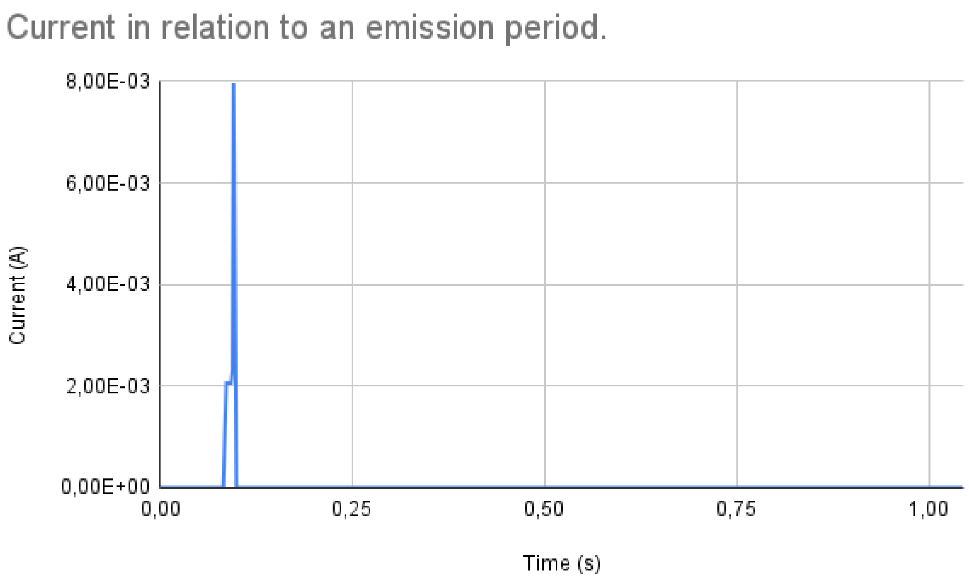

Figure 14: Current in relation to an emission period.

3. Software Reception

A beacon by itself has no value. In order for the project to be complete, the data sent by the beacon needs to be received by another device.

In order to keep the project as simple as possible, the device receiving the data will be a computer. The computer will need to be able to :

- receive data from the device
- decode the data (extract the actual sensor values)
- show the data in a readable manner

3.1. Overview

The application, made using python, will use a multi-threaded approach, which allows us to be able to execute multiple functions at the same time.

It will contain 2 functions :

- a function to read BLE payloads, decode them and write them to a file (csv format)

- a function to read the csv file and update the user interface based on its contents

Here is the full algorithm of the application, where the 2 functions are clearly defined :

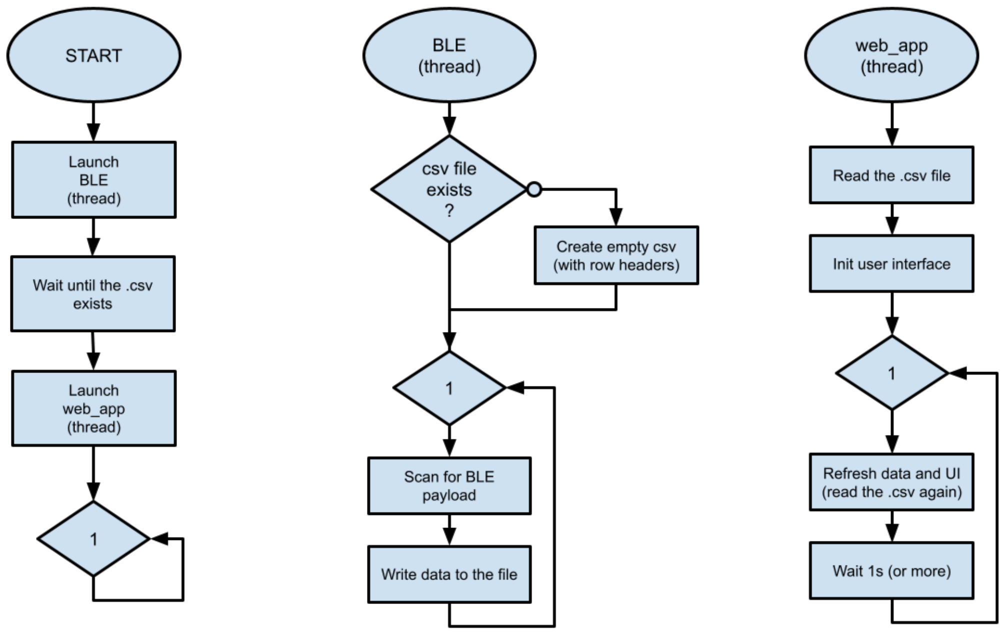

Figure 15: Algorithm of the python application

3.2. Receiving the data

The reception of the data will be done using the bleak python package. It will simply scan for a device with a given name, and read the data it sent.

All this is done through a callback, which can be easily created using the package. The relevant code can be found in the IN100_ble.py file (in the IN100_connect function).

3.3. Decoding the data

Once the data has been received, it is necessary to decode it, as it is contained in a binary format, which cannot be easily read by a human. The python bleak package automatically stores it in a structure, which simplifies the problem but does not solve it entirely. The structure looks like this :

```
AdvertisementData(
  local_name='IN100',
  manufacturer_data={1285: b'i\x9b\x00\xed\x0f\xfa\r'},
  rssi=-54
)
```

The rssi represents the signal strengh (in dB). The higher it is, the better. It is easy to read and can be directly taken from the structure.

On the other hand, the sensor data is not directly accessible, and needs to be decoded. The way to get it is entirely dependent on the IN100's configuration.

With our configuration, the sensor data can be found in the manufacturer data, with the manufacturer ID 1285 (which corresponds to InPlay, the company that makes IN100s). We can access it in the code with :

```
sensor_data = advertising_data.manufacturer_data[1285]
```

However, this data is stored as raw bytes, and needs to be manually processed. The acceleration on each axis is sent as a 12 bit signed integer, which is separated into a LSB (Least Significant Byte[9]) and MSB (Most Significant Byte[9]). The structure of the full message is the following :

Byte index	**Contained data**
0	Acceleration, X axis (LSB)
1	Acceleration, X axis (MSB)
2	Acceleration, Y axis (LSB)
3	Acceleration, Y axis (MSB)
4	Acceleration, Z axis (LSB)
5	Acceleration, Z axis (MSB)

Table 7: Sensor data sent by the IN100

We can then easily extract the acceleration on each axis by transforming the data from the relevant bytes for each axis. This manipulation can be seen in the `get_12bitInt` function in the IN100_ble.py file.

All that remains is to write the data to a file, which is extremely simple. In order to avoid unreadable data, a new csv file is created each day.

3.4. Displaying the data

In order to display the data, we used a python framework called Dash to create a web application. To avoid doing everything by hand, which would take a long time, we started by copying an example application provided by Plotly.

These examples are under a MIT license, which means anyone can use them as long as they keep the original license in the project's files.

After many modifications made in order to adapt the interface to our data (contained in the csv file), we obtained the following interface :

[9]A byte is a unit of digital information that equals 8 bits (a bit can either be 1 or 0), and is the smallest unit of memory that we can manipulate in most computers

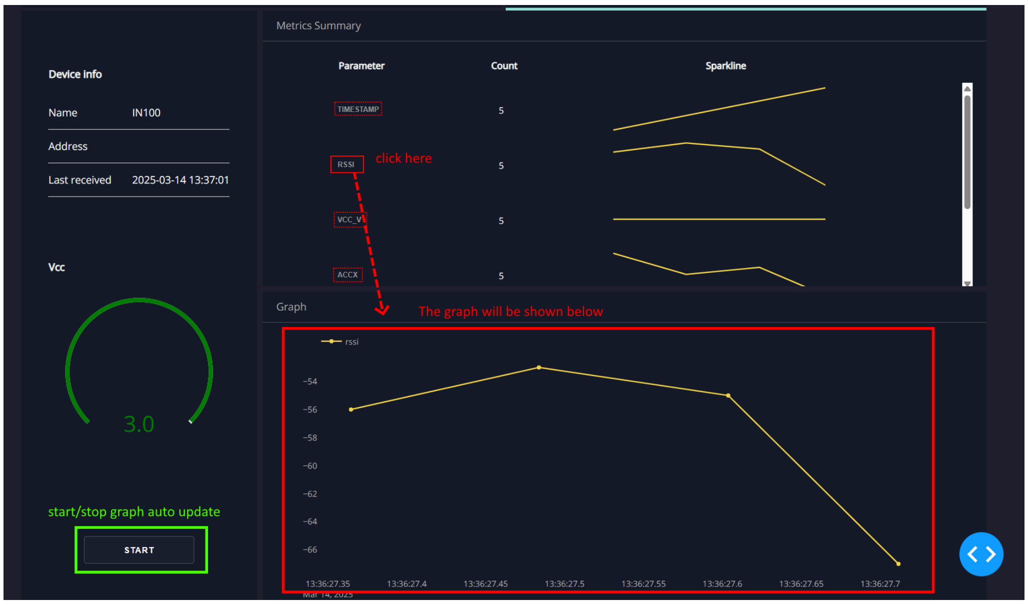

Figure 16: Wep app user interface

This is a very simple interface, which shows a preview of all the available data as graphs, and a precise graph for the selected one on the bottom. An auto-update feature can be turned on and off easily to add newly received data to the graphs.

It is also possible to read a specific measurement by simply hovering the mouse on it, as seen in the image below.

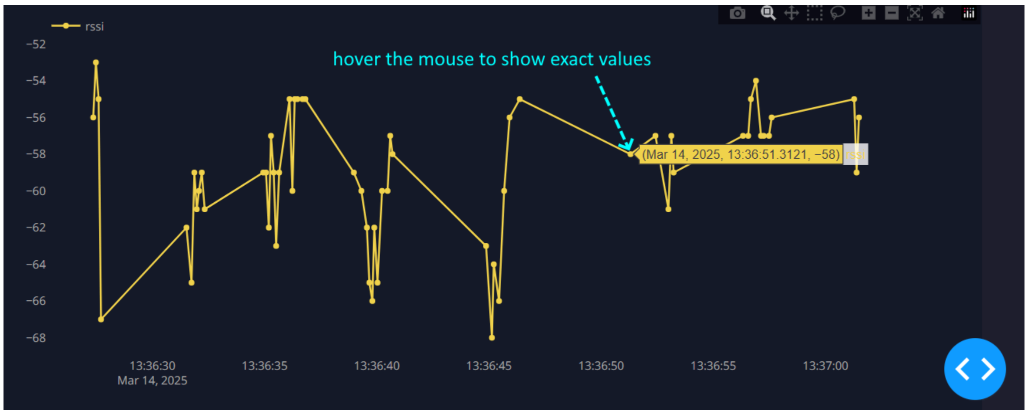

Figure 17: Reading an exact value in the web app graph

Currently, the unit of the acceleration measurements does not correspond to any unit used by scientists, as we are plotting the raw accelerometer data. A calibration of the accelerometer would be needed to get data in a real unit.

4. Conclusion and possible improvements

This project covered every part of the creation of a low power system, from PCB design to software. This was really interesting, as every part had different constraints, and the project went well.

However, due to some delays outside of our control in the making of the PCBs, a lot of time was lost, which resulted in an inability to create a flexible PCB and to test he new version of the PCB that should have greater performances, but the project is functional nonetheless.

5. Appendix

Project Repository : **https://github.com/kez97460/PR_BLE_Wearable**

Bibliography

[1] M. Presser, “The Rise of IoT – Why Today?.” 2016.

[2] R. Bulathsinghala, W. Ding, and R. Dharmasena, “Triboelectric Nanogenerators for Wearable Sensing Applications: A System-Level Analysis.” 2023.

[3] K. Georgiou, S. Xavier-de-Souza, and K. Eder, “The IoT Energy Challenge: A Software Perspective.” 2017.

[4] Espressif Systems, “ESP32-WROOM-32 Datasheet.” 2023.

[5] InPlay, “IN100 Datasheet.”

[6] Botch, “BMA400 Datasheet.” 2025.

[7] Sparkfun, “IN100 dev board.” [Online]. Available at: https://cdn.sparkfun.com/assets/5/a/2/6/2/SparkFun_NanoBeacon_Board_-_IN100.pdf

[8] Sparkfun, “BMA400 dev board.” [Online]. Available at: https://cdn.sparkfun.com/assets/0/2/5/5/b/SparkFun_Triple_Axis_Accelerometer_Breakout_-_BMA400__Qwiic_.pdf