KU LEUVEN

IoT Systems Project Report

Monitoring of Stress in Musicians

ELICT: ELECFS
19/12/2024
Master in Engineering Technology
Academic year 2024-2025

TABLE OF CONTENTS

1 INTRODUCTION

The Brussels Philharmonic Orchestra is currently conducting a research on stress monitoring of musicians.
Our task is to provide a way of measuring biometrics of the musicians. We achieved this by providing a complete IoT-ecosystem allowing measurements, low-power communication and connection with the cloud. The solution will be used by researchers without electronic knowledge, therefore special attention was given to assure ease of use of the sensor platform. An easy to understand GUI was created, allowing seamless measurements and data analysis.

Figure 1.1: The Brussels Philharmonic Orchestra

To design a functioning IoT solution that fulfils the requirements of the research team, we had to first turn these requirements into specifications. This was done in collaboration of all the teams during the first sessions. In this chapter, these specifications will be motivated.

2.1 Requirements

The following requirements were created based on the task description and the information from the meeting with Damien Buisseret. He is the researcher overseeing the project.

- We have to monitor the stress level of the musicians. The proposed metrics for this was heart rate.
- The posture of the musicians should also be monitored, using an EMG sensor placed on the musicians' backs.
- A sensor should be measure the sound intensity at all participants, both musicians and director.
- The boards containing the sensors and wireless modules should not hinder the musicians in any way.
- The solution should autonomously measure for at least a whole practice. The average time is of 5 to 8 hours.
- The GSR and temperature sensor evolve slowly, it is thus sufficient to measure them every 5 minutes, for 30 seconds.
- For the EMG sensor (posture), a higher measurement frequency is required.

2.2 Specifications

Based on the requirements, we created the following specifications.

2.2.1 General

- The wireless communication between sensor modules and gateway will be based on LoRa.
- The sensor modules will be worn in a custom-made harness, carrying the electronics, battery, and cabling. A picture of the prototype can be found in figure 6.2. The design ensures movement freedom and flexibility in term of body types.

- The autonomy of the modules will be at least 8 hours.
- The GSR and temperature sensor will be measured out every 10 minutes.
- The gateway is connected to the cloud by NB-IoT.
- In order to prevent collisions, the modules will communicate with the gateway through Time Division Multiplexing. Each node will get a specific timing to communicate with the gateway.

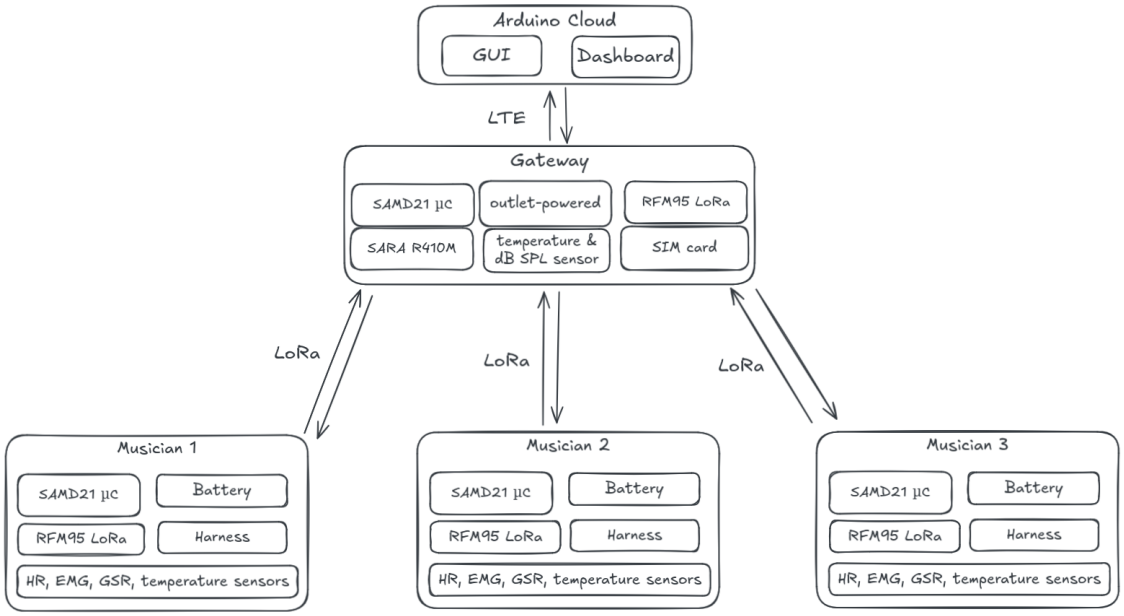


Figure 2.1: Initial system architecture

2.2.2 Initiation

During this phase all the nodes that will be used (can change depending on the size of the orchestra) will sent a message notifying their presence to the gateway. The gateway will the provide each node with a precise timing for the sensor data communication. We decided to work with a half-synchronous solution with one master (the gateway), representing the ground truth timewise. Following each communication of one node, the gateway will respond with a 'next timestamp' as ACK. By doing so, we are sure that there won't be collisions later on.

2.2.3 Measurements

Every boards measures the sensor data every 30 seconds and agglomerates the data in order to only send once every 10 minutes. This is done through LoRa.

2.2.4 Gateway

The gateway stores the necessary information inside an SD card and sends the data to the GUI. Due to some challenges faced by team gateway, the initial system architecture displayed in 2.1 is not accurate to the final product. The exact challenges and solutions are discussed in detail in chapter 4.2. Below, the updated and final system architecture is displayed.

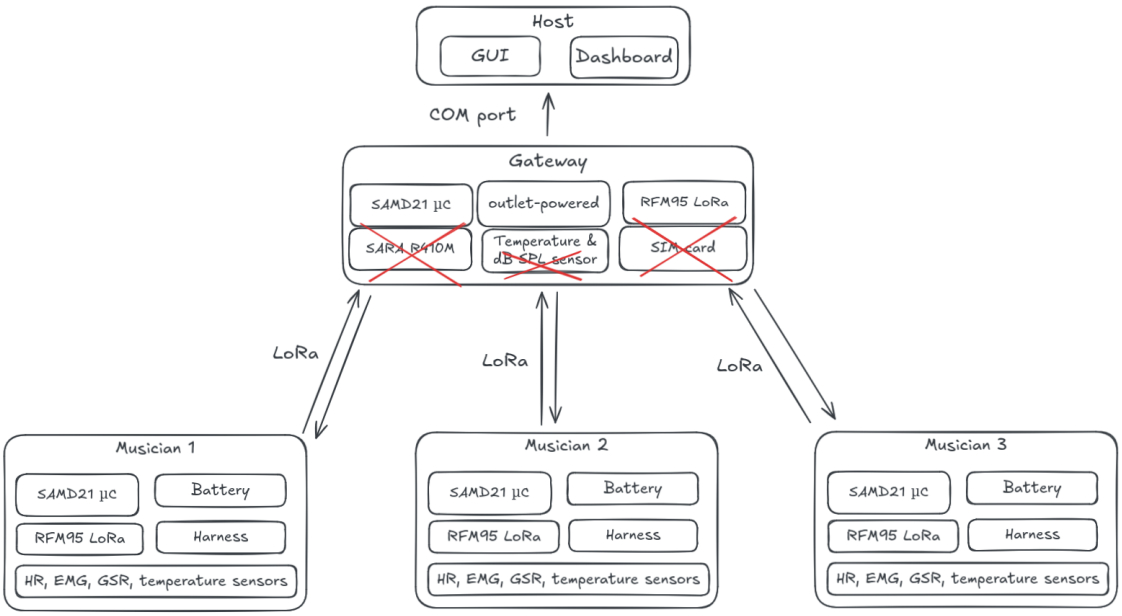


Figure 2.2: Final system architecture

2.3 Groups

Three groups were created, to split the class evenly. The responsibilities of each group can be found below.

2.3.1 Team Stress:

Musicians tend to experience a lot of stress during repetitions and performances. Due to pressure to perform and a highly competitive environment, musicians tend to avoid talking about their stress levels. Therefore an objective measurement is needed, in order to achieve this, a heart-rate sensor, a temperature sensor and a galvanic skin response sensor were used. This is discussed in depth in chapter 3.1.

The team members and respective task distribution can be found here:

- **Bert Pyck:** Responsible for the overall design of the PCBs for the sensor nodes used by team-stress and team-posture, soldering, burning the bootloader, and conducting hardware and component testing.
- **Klaas Meersman:** Heart rate and temperature sensor testing, code for measuring the sensors, code for LoRa communication with team gateway, integration of all sensor code and data aggregation.
- **Hugo Charmant:** GSR sensor testing, bridging the gap between hardware and software, harness creation, design of the multiple access protocol.

2.3.2 Team Posture:

- **Ybe Vandamme:** Responsible for testing the sensor and making software to use the sensor in a correct way with data analysis.
- **Florian Vandenbogaerde:** Responsible for designing the PCB used for posture

monitoring (in collaboration with team stress), soldering components, and serving as a test subject for evaluating the posture sensor's performance.

2.3.3 Team Gateway:

- **Joeri Sevens:** Responsible for designing the gateway hardware and deploying firmware on the modules.
- **Arthur Valentin:** Responsible for developing the code for the LoRa receiver and adapting it for visualization in the GUI.
- **Wouter Fouquet:** Responsible for ensuring communication between the gateway and the cloud, as well as visualizing the data.

3.1 Sensor Choice

Initially, we were tasked to measure the stress level based on the heart-rate measurements, an increase would mean the musician is stressed. Although this is definitely an important physioligical effect of stress, it's not enough to identify it. We therefore conducted a literature study on stress monitoring. This left us with a selection of sensors that, when combined, allows us to accurately distinguish stress from other situations (like physical activity).
The sensors and their importance is discussed in detail in this section.

3.1.1 Heart-rate Sensor

The MAX30105 sensor (Figure 3.1) measures heart rate and oxygen saturation using the principle of photoplethysmography (PPG), which involves detecting changes in light absorption and reflection caused by blood flow. The MAX30105 features two LEDs: green, red, and infrared (IR), and are used for heart rate and oxygen saturation measurements. These LEDs emit light through the skin, typically at the fingertip or earlobe, where the skin is thin enough for the light to penetrate. When the LEDs shine light through the skin, some of it is absorbed by the underlying tissues and blood, while the rest is reflected back. The amount of reflected light varies with each heartbeat due to changes in blood volume and oxygenation levels. The photodetector captures these variations in reflected light at both red and IR wavelengths. The sensor returns relative intensities of the reflected light at both wavelengths via an I2C protocol. The algorithm and in the can be and our own additions to it then interprets these intensities to extract the heart rate.

Shortcomings

The MAX30105 sensor has shortcomings though. The sensor cannot inherently distinguish between signals caused by muscle contractions, general movements, or actual heartbeats. As a result, during physical activities or even minor movements, the readings can fluctuate dramatically, causing the algorithm to misinterpret these variations as heartbeats. To improve the reliability of heart rate measurements from the MAX30105, incorporating a movement sensor (such as an accelerometer) can provide contextual information. An algorithm could be developed to filter out heart rate data collected during high-movement periods. If the accelerometer detects rapid motion or specific patterns indicative of physical activity (e.g., running or jumping), the system could disregard heart rate data collected during these times. When minimal movement is detected, the algorithm could prioritize those readings for more accurate heart rate calculations.

Figure 3.1: MAX30105 heartrate and temperature sensor

3.1.2 Temperature Sensor

The temperature sensor allows us to distinguish between stress and physical activity. It is also included in the MAX30105 sensor (Figure 3.1). Studies have consistently shown that skin temperature changes in response to stress, making it a viable physiological marker. For instance, acute stress triggers peripheral vasoconstriction, which leads to a rapid, short-term drop in skin temperature. This phenomenon has been observed in both humans and animals. It is thus a parameter, in combination with other parameters, commonly used when performing an analysis on stress levels [1].

3.1.3 Galvanic Skin Reponse Sensor

Why GSR?

The purpose of a GSR sensor is to measure the skin resistance. Sweat is a natural response to stress. The produced sweat is very conductive due to all the minerals present in it. The resistance of the skin therefore drops immensely when sweating.
Obviously, stress is not the only cause of sweating, but by combining this data with other sensors, we can single out stressful situations.

Which sensor?

The Grove GSR sensor was the only existing GSR sensor that allowed accurate measurements whilst being affordable. There were no relevant other options on the market.

3.2 Communication Choice: LoRa

For the communication between the nodes and gateway, we opted for LoRa (Long Range) communication technology. LoRa is designed for low power consumption, long communication range, and robust interference immunity.

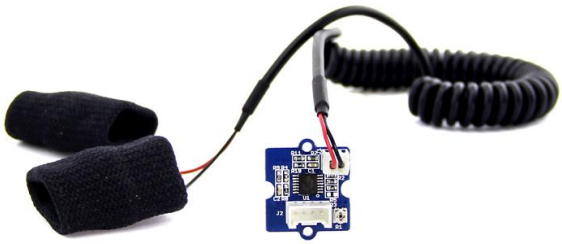


Figure 3.2: Grove GSR sensor [2]

Why LoRa?

In a stress monitoring setup, the main constraints are wearable comfort, battery longevity, and the need for reliable, low-throughput data transmission. Traditional short-range wireless technologies (e.g. BLE or ZigBee) either consume too much power at the distances required or require frequent gateway installations. Cellular technologies (e.g., LTE-M or NB-IoT) can provide long range and throughput but are more power-hungry, more complex to implement, and require SIM cards or subscriptions. Additionaly, there was no need for such high datarates in our application.

The sensor node, equipped with heart-rate, GSR, and temperature sensors, needs to periodically send a relatively small payload (255 bytes) at low frequency (every 10 minutes). LoRa's typical data rate and payload capacity are more than sufficient for this application, ensuring that the battery-powered device can operate for extended periods.

Transmission Parameters and Settings

We used a center frequency of 868 MHz, a standard European LoRa channel. The transmission power was set at a high level to maximize link budget, noting that European regulations typically allow up to 14 dBm Equivalent Isotropic Radiated Power (EIRP) for the 868 MHz ISM band. As an improvement, we could have reduced this EIRP to extende battery life.

Data Rate and Spreading Factor (SF):

Higher SF values (e.g., SF12) increase range and sensitivity but reduce data rate and increase transmission time, thus consuming more energy. Lower SF values (e.g., SF7) provide higher data rates at shorter range. For this application, we selected SF7 as we do not require great ranges in our application. At SF7 and a typical bandwidth of 125 kHz, we can reliably transmit our 255-byte payloads over the required distance, likely on the order of hundreds of meters indoors, while keeping the airtime and energy consumption low.

Duty Cycle and Transmission Interval:

European ISM band regulations impose a duty cycle limit (e.g., 1% per hour in certain sub-bands). Our application transmits data every 10 minutes, resulting in a very low duty cycle, well below regulatory limits. This compliance ensures that we do not cause undue interference and that the network remains scalable if multiple devices are deployed simultaneously.

3.3 Hardware Design

3.3.1 Requirements

The hardware design is based on the ATSAMD21G18 microcontroller, a highly energy-efficient 32-bit ARM Cortex-M0+ processor running at 48 MHz, ideal for low-power applications. It features a variety of peripheral interfaces, making it suitable for sensor-based systems. The system supports both Analog & Digital I/O, including two analog inputs for precise data acquisition from GSR (Galvanic Skin Response) and EMG (Electromyography) sensors. Additionally, the microcontroller integrates six serial communication interfaces (SERCOM), which can be configured for I^2C, SPI, and UART protocols, ensuring seamless communication with external devices and sensors such as the heartrate and temperature sensor.

To enable audio processing capabilities, the hardware includes I^2S functionality, which facilitates efficient digital audio input. For Autonomy & Energy Efficiency, the system is powered by a LiPo or Li-ion battery, complemented by a robust charging circuit and protection mechanisms to prevent overcharging and ensure long battery life. Power management is optimized to extend operational autonomy, making it suitable for portable or remote applications.

Wireless communication is achieved through the integrated LoRa module (RFM95), which offers long-range, low-power connectivity. The module is compatible with external antennas, enabling reliable data transmission over extended distances, making it ideal for IoT applications or environments with limited infrastructure. Combined, these features provide a versatile, energy-efficient, and robust hardware platform capable of supporting sensor data acquisition, processing, and wireless communication.

During this project, we did not focus extensively on improving or adjusting the performance parameters of the LoRa module. However, after the presentation, it became clear to us that these parameters have a significant impact on the node's lifespan. Performance improvements could be achieved by putting the LoRa module into different sleep modes when it is not transmitting. Additionally, increasing the spreading factor is another possibility. This results in slower chirps, leading to a lower data transmission rate, which would be perfect because it has a minimal impact on this application.

We were also made aware of the possibility of a non-transmission option. Each node could store data locally on an SD card, and the data could be collected after an event. This would be a good alternative as it allows for the registration of more data, providing clearer insights into stress-moments. Unfortunately, this was not feasible within the scope of the IoT-Systems course, as wireless communication was an expected requirement.

3.3.2 Block Diagram

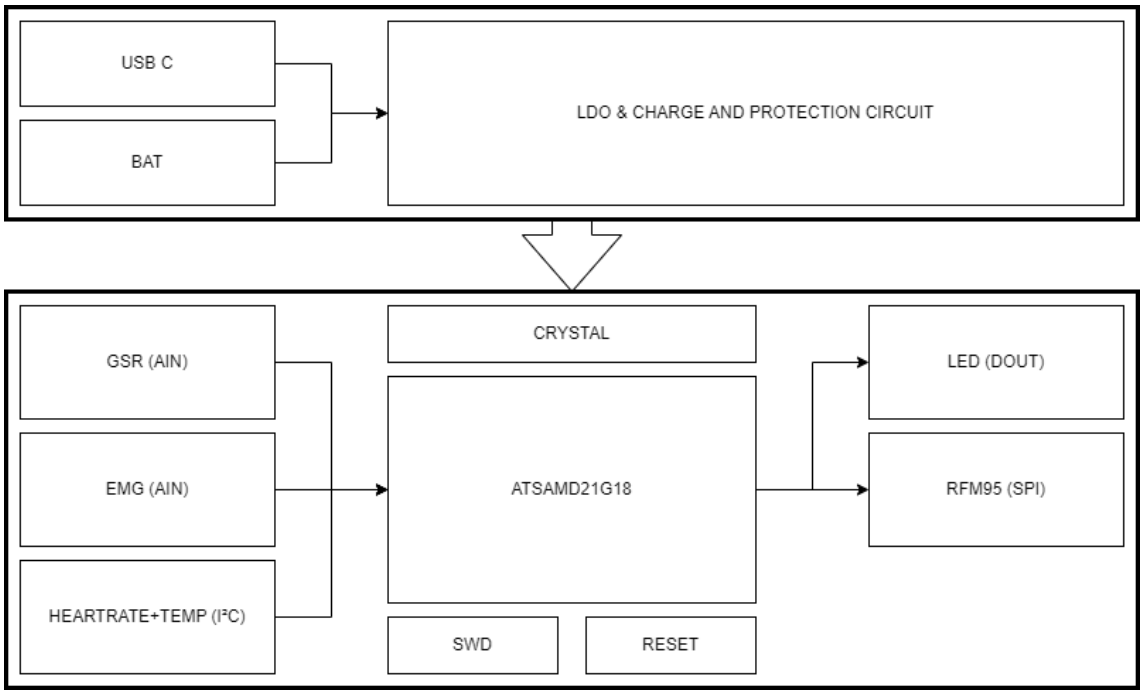


Figure 3.3: Block Diagram hardware design

The complete hardware design is divided into two main sections (Figure 3.3). The first section, as shown on the schematic in figure 3.4 , consists of the power supply part of the node. This section has two power inputs. The first input is a USB-C connector, which provides power, charging functionality for the battery, data communication, and programming capabilities. The second power input is a single-cell LiPo/Li-ion battery. The power section includes a switching, charging, and protection circuit, which ensures that the battery charges or discharges depending on the connected input. Additionally, a Low Dropout Regulator (LDO) is included to ensure that the rest of the node operates at the required standard voltage.

The second section of the hardware (Figure 3.5) is where the processing takes place. As previously mentioned, all computations are handled by the ATSAMD21G18 microcontroller. The node receives input from three sensors that measure physiological parameters from the human body: Galvanic Skin Response (GSR), Electromyography (EMG), heart rate, and temperature. The GSR and EMG signals are read as analog values. These sensors require dedicated electronics to convert the raw signals into accurate analog signals, and this circuitry is integrated directly onto the PCB.

The communication with the heart rate and temperature sensors is achieved through a SERCOM I^2C connection. The microcontroller is also equipped with a crystal oscillator, which enables the use of various sleep modes through the Real-Time Clock (RTC). Furthermore, a Serial Wire Debug (SWD) connector is included, which allows the bootloader to be burned onto the microcontroller, making it programmable via the USB interface.

A physical reset button is present to restart the microcontroller when needed. For

output, an indication LED can be turned on or off to provide status feedback, such as indicating potential issues or other states of operation. This LED is primarily used as a sign of life-indicator. To transmit all measured data to an uplink gateway, a LoRa connection is established via the RFM95 module, which communicates with the ATSAMD21G18 microcontroller through an SPI connection (Figure 3.6).

3.3.3 Schematic Explanation

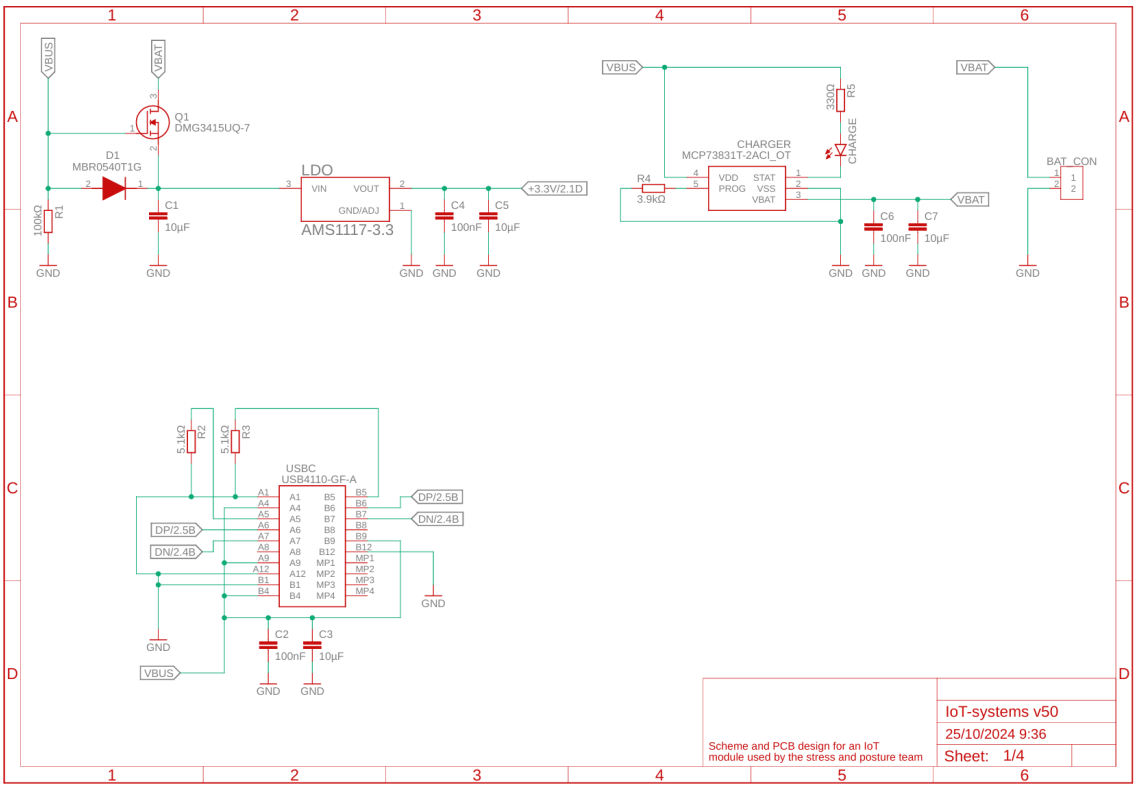


Figure 3.4: Schematic node

- **USB-C**
  - Power/host negotiation
  - "Consuming device" (5.1 kΩ) to GND
- **Switching Circuit**
  - USB to Vbat
  - LDO
- **Charging Circuit**
  - MCP73831
  - Indication LED

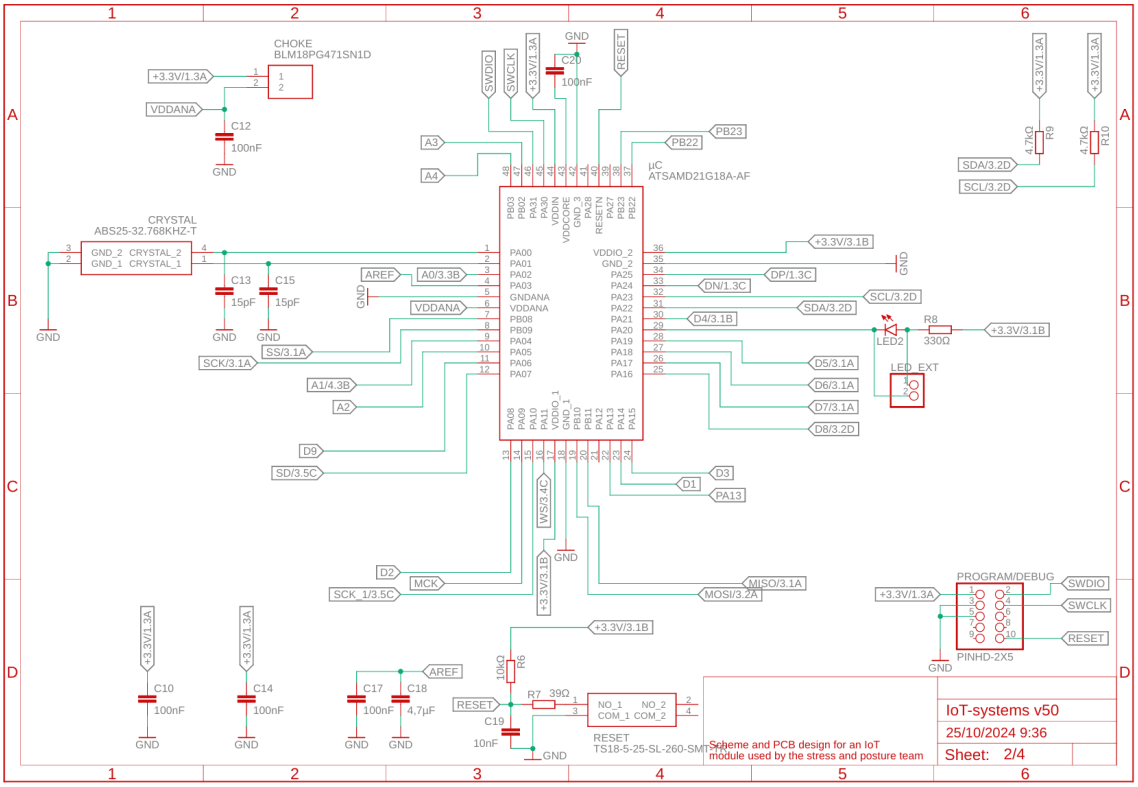


Figure 3.5: Schematic node

- **Analog Choke**
- **Crystal**
  - RTC
- **Serial Wire Debug (SWD)**
  - Programming and debug options

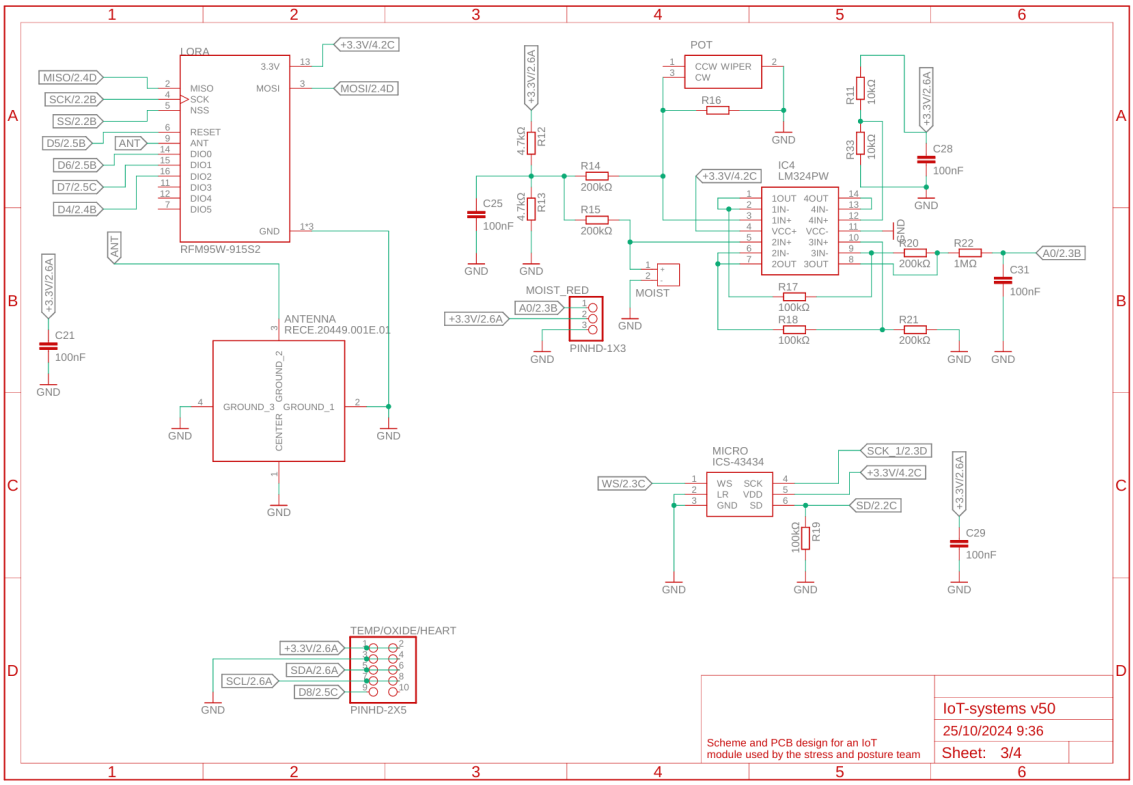


Figure 3.6: Schematic node

- **RFM95**
  - Interrupt options
  - External antenna
- **I²C**
- **I²S**

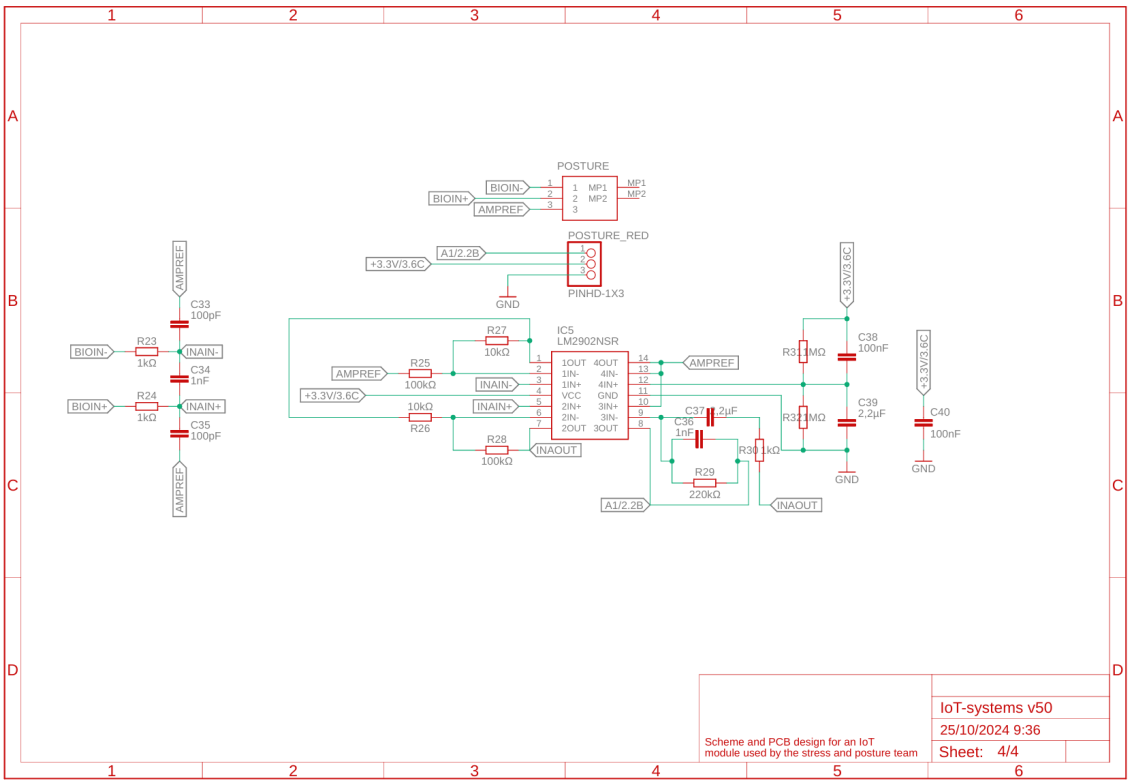


Figure 3.7: Schematic node

3.3.4 PCB Explanation

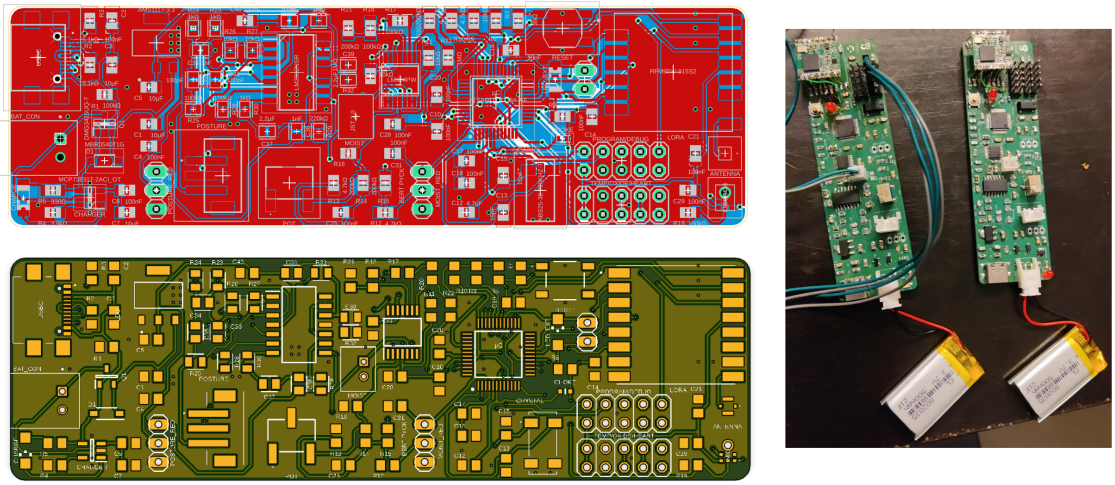


Figure 3.8: PCB node

After validating the schematic design, I proceeded to create the layout of the PCB using Fusion (Figure 3.8). Several important considerations had to be taken into account during this process. First, the differential pairs of the USB traces had to be routed to ensure they were of equal length and parallel to each other. Additionally, the correct resistors for the USB-C interface had to be soldered to ensure that the node would be recognized as a "consumer device" by the host.

At the university, we had 0805 SMD resistors available, so it was necessary to select the appropriate packages for all passive components. Furthermore, the crystal and its accompanying capacitors needed to be placed as close as possible to the microcontroller to minimize noise and ensure stable operation.

Due to the significant presence of analog electronics for the measurement circuitry, it was essential to separate the analog and digital electronics on the PCB to avoid interference. Since we did not have the option to use a solder mask, all components had to be soldered manually. This increased the risk of poor connections. However, despite these challenges, we successfully got the nodes working in the end.

3.3.5 Power Consumption Measurement

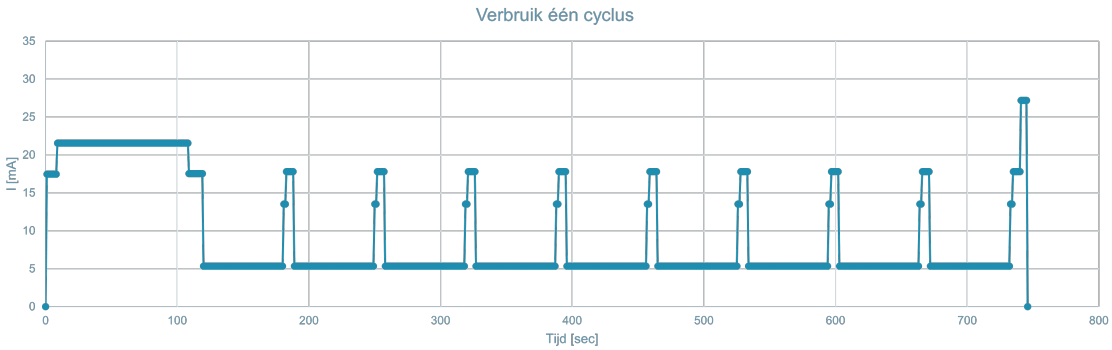


Figure 3.9: Power Consumption

In figure 3.9 we see the measured power consumption for one complete cycle is shown. One cycle lasts approximately 10 minutes. The measured signal represents a worst-case scenario. The cycle begins with the initialization phase with the gateway, which can take a maximum of 2 minutes. During this initial phase, a current of approximately 22 mA is drawn due to the indication LED being turned on until a connection with the gateway is established. If no connection is made, the initialization will automatically stop after 2 minutes.

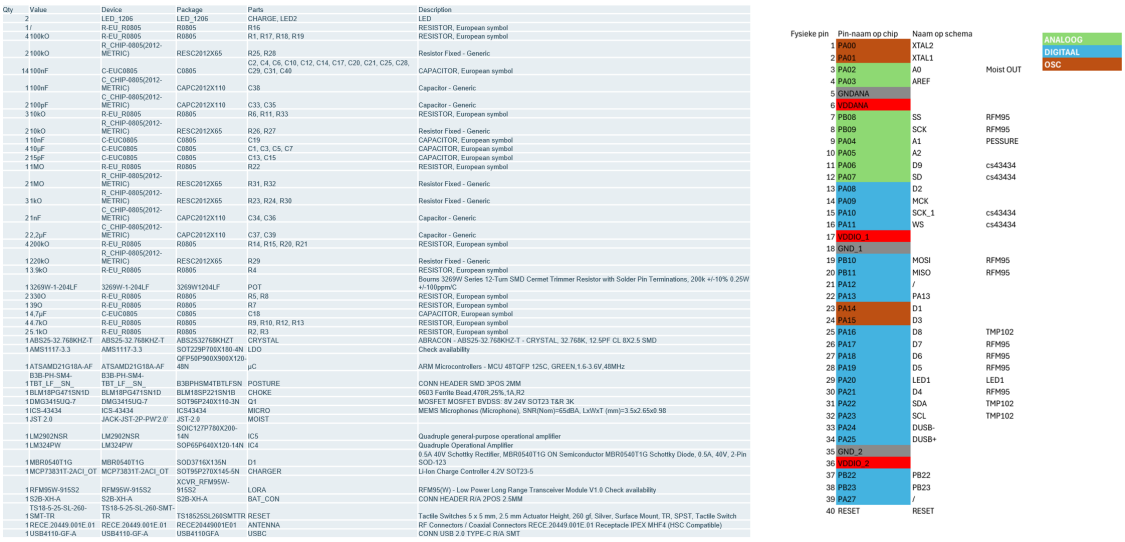
Following the initialization, a repetitive process occurs. Every minute, a current spike will appear, which corresponds to the measurements and calculations performed by all the sensors. This is also accompanied by the blinking of the indication LED. During this period, over a few seconds, the current drawn ranges from 13.5 mA to 17.5 mA.

At the end of the cycle, the LoRa RFM95 module transmits a data packet. This occurs at the maximum power of 24 dBm, resulting in a high current spike of 27 mA.

On average, the current consumption is calculated to be 9.02 mA for one cycle (10 minutes), which amounts to 45.5 mAh. Therefore, for a battery capacity of 500 mAh, the system can operate for approximately 11 hours.

3.3.6 External Documentation

In addition to the details provided earlier, two supplementary documents are available to offer further insights into the node's design and implementation (Figure 3.10):



Qty	Value	Device	Package	Parts	Description
2		LED_1206	LED_1206	CHARGE, LED2	LED
1	1	R-EU_R0805	R0805	R16	RESISTOR, European symbol
4	100kΩ	R-EU_R0805	R0805	R1, R17, R18, R19	RESISTOR, European symbol
2	100kΩ	R_CHIP-0805(2012-METRIC)	RESC2012X65	R25, R28	Resistor Fixed - Generic
14	100nF	C-EUC0805	C0805	C2, C4, C6, C10, C12, C14, C17, C20, C21, C25, C28, C29, C31, C40	CAPACITOR, European symbol
1	100nF	C_CHIP-0805(2012-METRIC)	CAPC2012X110	C38	Capacitor - Generic
2	10pF	C_CHIP-0805(2012-METRIC)	CAPC2012X110	C33, C35	Capacitor - Generic
3	10kΩ	R-EU_R0805	R0805	R6, R11, R33	RESISTOR, European symbol
2	10kΩ	R_CHIP-0805(2012-METRIC)	RESC2012X65	R26, R27	Resistor Fixed - Generic
1	10nF	C-EUC0805	C0805	C19	CAPACITOR, European symbol
4	10µF	C-EUC0805	C0805	C1, C3, C5, C7	CAPACITOR, European symbol
2	15pF	C-EUC0805	C0805	C13, C15	CAPACITOR, European symbol
1	1MΩ	R-EU_R0805	R0805	R22	RESISTOR, European symbol
2	1MΩ	R_CHIP-0805(2012-METRIC)	RESC2012X65	R31, R32	Resistor Fixed - Generic
3	1kΩ	R_CHIP-0805(2012-METRIC)	RESC2012X65	R23, R24, R30	Resistor Fixed - Generic
2	1nF	C_CHIP-0805(2012-METRIC)	CAPC2012X110	C34, C36	Capacitor - Generic
2	2,2µF	C_CHIP-0805(2012-METRIC)	CAPC2012X110	C37, C39	Capacitor - Generic
4	200kΩ	R-EU_R0805	R0805	R14, R15, R20, R21	RESISTOR, European symbol
1	220kΩ	R_CHIP-0805(2012-METRIC)	RESC2012X65	R29	Resistor Fixed - Generic
1	3,9kΩ	R-EU_R0805	R0805	R4	RESISTOR, European symbol
1	3269W-1-204LF	3269W-1-204LF	3269W1204LF	POT	Bourns 3269W Series 12-Turn SMD Cermet Trimmer Resistor with Solder Pin Terminations, 200k +/-10% 0.25W +/-100ppm/C
2	330Ω	R-EU_R0805	R0805	R5, R8	RESISTOR, European symbol
1	9kΩ	R-EU_R0805	R0805	R7	RESISTOR, European symbol
1	4,7µF	C-EUC0805	C0805	C18	CAPACITOR, European symbol
4	4,7kΩ	R-EU_R0805	R0805	R9, R10, R12, R13	RESISTOR, European symbol
2	5,1kΩ	R-EU_R0805	R0805	R2, R3	RESISTOR, European symbol
1	ABS25-32.768KHZ-T	ABS25-32.768KHZ-T	ABS25327.68KHZT	CRYSTAL	ABRACON - ABS25-32.768KHZ-T - CRYSTAL, 32.768K, 12.5PF CL 8X2.5 SMD
1	AMS1117-3.3	AMS1117-3.3	SOT229P700X180-4N	LDO	Check availability
1	ATSAMD21G18A-AF	ATSAMD21G18A-AF	QFP50P900X900X120-48N	µC	ARM Microcontrollers - MCU 48TQFP 125C, GREEN,1.6-3.6V,48MHz
1	B3B-PH-SM4-TBT_LF__SN_	B3B-PH-SM4-TBT_LF__SN_	B3BPHSM4TBTLFSN	POSTURE	CONN HEADER SMD 3POS 2MM
1	BLM18PG471SN1D	BLM18PG471SN1D	BLM18SP221SN1B	CHOKE	0603 Ferrite Bead,470R,25%,1A,R2
1	DMG3415UQ-7	DMG3415UQ-7	SOT96P240X110-3N	Q1	MOSFET MOSFET BVDSS: 9V 24V SOT23 T&R 3K
1	ICS-43434	ICS-43434	ICS43434	MICRO	MEMS Microphones (Microphone), SNR(Nom)=65dBA, LxWxT (mm)=3.5x2.65x0.98
1	JST 2.0	JACK-JST-2P-PW2.0'	JST-2.0	MOIST	
1	LM2902NSR	LM2902NSR	SOIC127P780X200-14N	IC5	Quadruple general-purpose operational amplifier
1	LM324PW	LM324PW	SOP65P640X120-14N	IC4	Quadruple Operational Amplifier
1	MBR0540T1G	MBR0540T1G	SOD3716X136N	D1	0.5A 40V Schottky Rectifier, MBR0540T1G ON Semiconductor MBR0540T1G Schottky Diode, 0.5A, 40V, 2-Pin SOD-123
1	MCP73831T-2ACI_OT	MCP73831T-2ACI_OT	SOT95P270X145-5N	CHARGER	Li-Ion Charge Controller 4.2V SOT23-5
1	RFM95W-915S2	RFM95W-915S2	XCVR_RFM95W-915S2	LORA	RFM95(W) - Low Power Long Range Transceiver Module V1.0 Check availability
1	S2B-XH-A	S2B-XH-A	S2B-XH-A	BAT_CON	CONN HEADER R/A 2POS 2.5MM
1	SMT-TR	TS18-5-25-SL-260-SMT-TR	TS18525SL268SMTTR	RESET	Tactile Switches 5 x 5 mm, 2.5 mm Actuator Height, 260 gf, Silver, Surface Mount, TR, SPST, Tactile Switch
1	RECE.20449.001E.01	RECE.20449.001E.01	RECE2044900 1E01	ANTENNA	RF Connectors / Coaxial Connectors RECE.20449.001E.01 Receptacle IPEX MHF4 (HSC Compatible)
1	USB4110-GF-A	USB4110-GF-A	USB4110GFA	USBC	CONN USB 2.0 TYPE-C R/A SMT

Fysieke pin	Pin-naam op chip	Naam op schema	
1	PA00	XTAL2	
2	PA01	XTAL1	
3	PA02	A0	Moist OUT
4	PA03	AREF	
5	GNDANA		
6	VDDANA		
7	PB08	SS	RFM95
8	PB09	SCK	RFM95
9	PA04	A1	PESSURE
10	PA05	A2	
11	PA06	D9	cs43434
12	PA07	SD	cs43434
13	PA08	D2	
14	PA09	MCK	
15	PA10	SCK_1	cs43434
16	PA11	WS	cs43434
17	VDDIO_1		
18	GND_1		
19	PB10	MOSI	RFM95
20	PB11	MISO	RFM95
21	PA12	/	
22	PA13	PA13	
23	PA14	D1	
24	PA15	D3	
25	PA16	D8	TMP102
26	PA17	D7	RFM95
27	PA18	D6	RFM95
28	PA19	D5	RFM95
29	PA20	LED1	LED1
30	PA21	D4	RFM95
31	PA22	SDA	TMP102
32	PA23	SCL	TMP102
33	PA24	DUSB-	
34	PA25	DUSB+	
35	GND_2		
36	VDDIO_2		
37	PB22	PB22	
38	PB23	PB23	
39	PA27	/	
40	RESET	RESET	

ANALOOG
DIGITAAL
OSC

Figure 3.10: External documentation

- Bill of Materials (BOM): A comprehensive list of all components required to assemble a single node can be found. This ensures clarity and completeness regarding the hardware components used in the design.
- Microcontroller Pinout: The pinout diagram of the microcontroller provides an overview of all input/output connections, facilitating the understanding of the circuit layout and signal flow.

Both documents, along with additional resources and information, are accessible on the project's GitHub repository. This repository serves as a central hub for all technical documentation, source files, and further explanations, ensuring transparency and ease of access for future reference or modifications.

3.3.7 Changing the SPI SERCOM protocol in Software and Hardware

```
//#define LORA_DEFAULT_SPI SPI
#define LORA_DEFAULT_SPI_FREQUENCY 200000
#define LORA_DEFAULT_SS_PIN A1
#define LORA_DEFAULT_RESET_PIN 12
#define LORA_DEFAULT_DIO0_PIN 10
#endif

#define PA_OUTPUT_RFO_PIN 0
#define PA_OUTPUT_PA_BOOST_PIN 1
```

Listing 3.1: Header parameters LoRa.h

```
//SPIClass mySPI (&sercom4, A2 , SCK, MOSI, SPI_PAD_2_SCK_3, SERCOM_RX_PAD_1);
SPIClass mySPI (&sercom4, MISO , SCK, MOSI, SPI_PAD_2_SCK_3, SERCOM_RX_PAD_0);

LoRaClass::LoRaClass() :
  _spiSettings(LORA_DEFAULT_SPI_FREQUENCY, MSBFIRST, SPI_MODE0),
```

```
  _spi(&mySPI),
  _ss(LORA_DEFAULT_SS_PIN), _reset(LORA_DEFAULT_RESET_PIN), _dio0(
      LORA_DEFAULT_DIO0_PIN),
  _frequency(0),
  _packetIndex(0),
  _implicitHeaderMode(0),
  _onReceive(NULL),
  _onTxDone(NULL)
{
  // overide Stream timeout value
  setTimeout(0);
}
```

Listing 3.2: Adding SPI object

In the codes above, the default SPI object used by the LoRa library is disabled. Instead, a new SPI object is created with the correct parameters just above the constructor of the LoRa library to enable the use of SERCOM4. This step is necessary to establish a proper SPI connection with the LoRa module. The parameters provided for the SPI object specify the pins used for communication. The last two parameters represent the combination of physical pins to which the communication lines are connected.

3.3.8 Hardware Conclusion

Throughout the project, significant knowledge and practical skills were gained in designing and implementing a low-energy circuit, specifically for the ATSAMD21G18 microcontroller. Key challenges and accomplishments included the successful implementation of the SERCOM interface, encompassing both software and hardware aspects, as well as programming the bootloader onto the microcontroller. The integration of a USB-C interface was also realized, allowing for both power delivery and communication. Additionally, a reliable battery circuit was implemented, ensuring proper power management and charging functionality.

While the project resulted in a fully functional node, areas for improvement were identified. The introduction of sufficient test pads would facilitate easier debugging and testing during future iterations. Furthermore, better dimensioning of the crystal and its associated components would improve the overall reliability and accuracy of the microcontroller's timing functionality.

In summary, this project provided invaluable experience in embedded hardware design, low-energy circuit development, and microcontroller integration, while also highlighting areas for future enhancements to optimize the design.

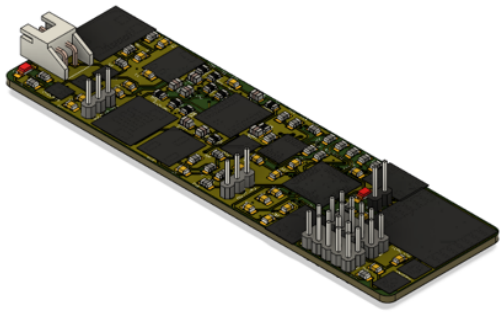


Figure 3.11: 3D view of the PCB

3.4 Multiple Access Design

3.4.1 Synchronous vs Asynchronous Design

In order to avoid collisions when sending information with multiple nodes, a thorough analysis of the multiple access technologies was performed. The multiple access in the article 'An Energy-Efficient LoRa Multi-Hop Protocol through Preamble Sampling for Remote Sensing' was our initial method [3]. We quickly discovered that the proposed Channel Activity Detection worked well in their use case, but we could make a much simpler solution. Additionally, constantly listening for transmission would increase our energy consumption drastically. An initial assessment of the type of multiple access algorithm led to the choice between a synchronous and asynchronous algorithm.
Since our network is completely static (once the measurements have begun), we understood that we could use the gateway as the system master. This way, the nodes don't have to constantly exchange timing information.
Our preferred solution is one of Time Division Multiplexing, with the gateway initializing the nodes and sending each node the delta for the first transmission time and the interval between transmissions. This is discussed in great detail in section 3.5.3.

3.4.2 Possible improvements

At room temperature, the frequency of the oscillator can vary from 32668 kHz to 32868 kHz according to the datasheet[4]. This implies a higher clock drift (about $\pm$ 263 seconds per day) than initially accounted for. To avoid collisions with a 100% certainty in an 8 hour concert when sending every 10 minutes, we would only be able to allow 3 nodes in the system. There are however a few solutions to this problem:

- We could simply allow more nodes without accounting for the drifts. Collisions would be rare since the nodes transmit at different times and assuming a transmission time of 35 ms for 255 bytes. This approach is suboptimal.
- We could calibrate the clock for each node to significantly reduce the drift. This would, however, require fine-tuned adjustments for each individual node.
- Our preferred solution involves the gateway initializing the nodes and sending each node its timing for the next transmissions, with resynchronization occurring every 10 minutes. This approach ensures that the maximum delta between two nodes sending is only 3.652 seconds. As a result, each node needs a 3.687-second (3.652+0.035)

time window, allowing up to 162 nodes to be deployed simultaneously.

3.5 Software Design

In this section we give a detailed explanation of the functionality of the node code in the following section. All code can be found on `https://github.com/BrtPck/IoT-Systems/tree/main/Team%20Stress/Software`.

3.5.1 Changeable Parameters

There are several predefined parameters that can be easily adjusted, and these changes will propagate to other dependent parameters, ensuring the system remains consistent and functional. '`deltaMeasurementUnitsInSec`' defines the interval in seconds between each set of measurements. It is set to 60 seconds by default. This influences the sleep time between measurements and the overall timing of the system. '`amountOfHeartRateMeasurementsPerUnit`', '`amountOfSkinConductanceMeasurementsPerUnit`', '`amountOfSkinTempMeasurementsPerUnit`' and '`amountOfMuscleTensionMeasurementsPerUnit`' specify the number of measurements to be taken for each sensor type within a measurement unit. Adjusting these values will change the accuracy and granularity of the measurements but also impact the time spent on each measurement unit. We observed that setting this to 1 was generally fine, except for measuring the heart rate. '`deltaSendToGateInMilis`' is set by the '`initWithGate()`' function based on the response from the gateway. '`bufferSize`' is then calculated as `measurementUnitsBeforeSend * amountOfSensors`. Our node will measure every minute and send every 10 minutes, the latter is set by the gateway.

3.5.2 Sensor Measurements

Heart Rate Measurement

The '`measurementUnitHeartRateSensor`' function measures the heart rate by detecting beat using the the MAX30105 sensor. It averages the heart over 10 beats and handles the case where no 4 consecutive measurements of the heartrate can be made. If the measurement of 4 heartbeats fails 3 times in a row, the measurement unit will stop measuring and the algorithm will proceed. The time spent measuring the heart rate is timed to adjust the sleep time accordingly. We do not want the synchronization system to go out of sync. This is important because of the variability in the duration of the heart rate measurement. We also wrote debug code that subtracts the moving average from the current value. Plotting this code clearly shows the variation in vessel diameter when e.g. used on the finger.

Skin Conductance Measurement

The '`measurementUnitSkinConductanceSensor`' measures skin conductance. The measurement can be done multiple times to avoid noise and can then be averaged by adjusting the `size` parameter. This seemed unnecessary as we did not notice noise on the measurements, and we thus set this parameter to 1.

Skin Temperature Measurement

The '`measurementUnitSkinTemperature`' function measures skin temperature using the MAX30105 sensor and averages the readings. As mentioned before, one measurement provided sufficient accuracy.

Muscle Tension Measurement

The '`measurementUnitMuscleTension`' function measures muscle tension using the EMG input. It filters the signal, calculates the envelope, and checks for bad posture based on the filtered signal. The function returns the percentage of bad posture detected. A more detailed explanation of the code internals can be found in section 4.2.

Possible improvements

If the measurements are noisy, it is more accurate to use the median rather than the average. A single extremely noisy measurement would not influence the median. Additionally, occasional zero readings were observed in the temperature measurements, which we attribute to skin contact with the I2C pins, or a malfunctioning temperature sensor on the board. To address this issue, we can implement a strategy of taking multiple measurements and applying a filtering process to identify and exclude erroneous readings.

3.5.3 Implementing the multiple access

In this section, we describe the operation of our LoRa protocol on the node, including the data aggregation and transmission processes. Figure 3.12 shows a schematic representation of the code that implements this functionality.

LoRa Protocol Operation

The node signs up to the gateway by sending its ID using '`signUpToGate()`'. This function sends the node's ID as a hexadecimal string over LoRa. The node waits for a response from the gateway using '`waitForGateAndGetInterval()`. This function parses the received message to extract the interval at which the node should send data and any setup time left before the gateway is ready. The interval received from the gateway is stored in '`deltaSendToGateInMilis`, which determines how often the node should send data to the gateway. All parameters are recalculated in '`calculateParameters()`' to ensure correct timing of measuring and sending.

Data Aggregation

The '`performMeasurementsWithSleepInBetween()`' function is responsible for collecting measurements from various sensors. The measurements are stored in a float array '`bufferToSend`'. The array is structured to store multiple measurements of each type before sending them. Figure 3.13 shows the layout of the data that is send every 10 minutes to the gate.

3.5.4 Data Transmission

The aggregated data in `bufferToSend` is converted into a string format in '`sendMeasurementsString()`'. This string (Figure 3.13) includes the node's ID and the

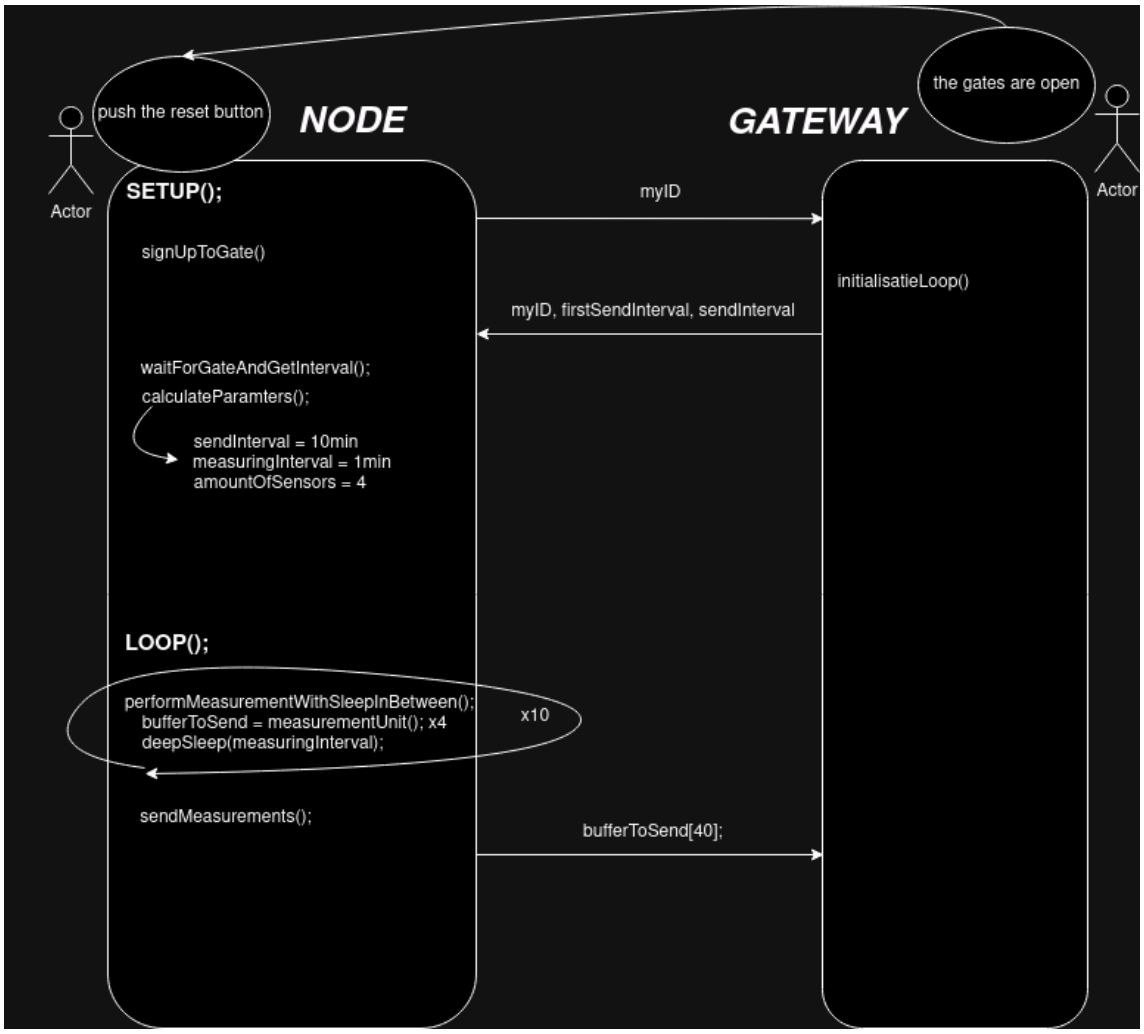


Figure 3.12: Node code schematic

Figure 3.13: id and `bufferToSend[]` float array. s0,s1... are sensor 0 and sensor 1. t0, t1... are time sample 0 and 1.

measured values separated by commas. The formatted string is sent to the gateway. During data transmission, LED indicators are used to signal the status.

3.5.5 Low Power

After each set of measurements, the node goes into deep sleep using '`LowPower.deepSleep(deltaMeasurementUnitsInMilis - waistedTimeMeasuringHR)`' to conserve energy.

3.5.6 Possible improvements

- During current measurements of our node PCB we noticed a higher current draw than expected. We suspect this is caused by the LoRa module not being turned off separately. We could solve this by making an RFM95 object, and putting it to sleep with '`RFM95object.sleep()`'.
- We transmit data as strings, but this approach limits the amount of information we can send since all variables must be converted to ASCII characters. Each ASCII character takes up two bytes, allowing us to send only about 40 measurements at a time. Instead, we can optimize our data representation: the heart rate can be represented with 8 bits, skin conductance with 10 bits, skin temperature with 6 bits, and bad posture percentage with 4 bits. By using this method, we could transmit sensor measurements for 72 timestamps in one go. This means we would only need to send data every 72 minutes instead of every 10 minutes, significantly improving our efficiency.

Musicians often experience fatigue during long performances, which can negatively impact their playing posture. To address this issue, Team Posture has developed an IoT system utilizing the muscle BioAmp Candy EMG sensor from Upside Down Labs. This sensor was chosen for its suitability in effectively monitoring musicians' posture. The selection process, operational details, and criteria for choosing this EMG sensor are further explained in this chapter.

4.1 Hardware

4.1.1 Sensor

The muscle BioAmp Candy sensor from Upside Down Labs, as shown in Figure 4.1, was selected for this project. This PCB prominently features an amplifier surrounded by various supporting components. Three electrodes are connected to the human body via this PCB.

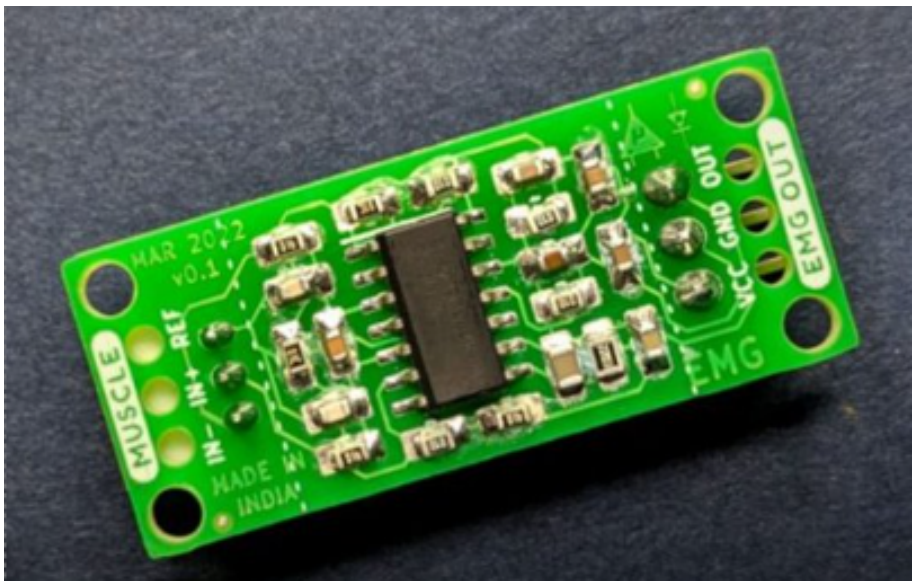


Figure 4.1: Posture sensor by BioAmp Candy.

This sensor was chosen based on the following criteria:

- Availability of hardware documentation
- Availability of software documentation
- Cost-effectiveness

- Product availability

An alternative considered was the Grove EMG detector by Seeed Studio. However, this was ultimately not selected due to less comprehensive documentation for both hardware and software. The chosen BioAmp Candy sensor had clear documentation, all organized in a GitHub repository. Additionally, the BioAmp Candy sensor was more affordable. Most important, the alternative Grove EMG detector was no longer in stock.

4.1.2 Operation

Figure 4.2 illustrates the recommended placement of the posture sensor on an arm. However, for posture monitoring, the electrodes (pads) should ideally be attached to the back of the musician to track posture effectively. More specifically on the trapezius transversalis muscle.

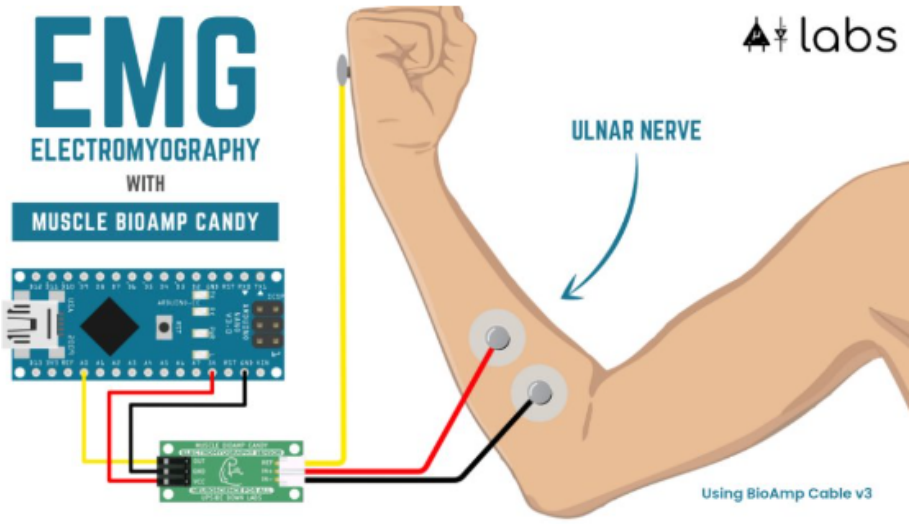


Figure 4.2: Operational setup of the posture sensor.

It is crucial to place the red and black wires on the same muscle, while the yellow wire should be positioned on a different muscle.

The functionality of an EMG sensor is relatively straightforward. When a muscle is contracted, tiny electrical impulses travel through the nervous system. By attaching electrodes (pads) to the muscles, these signals can be detected. Since these electrical signals are extremely small—only a few microvolts—the sensor amplifies them. The amplification is x2420. This amplification ensures the signals are easily measurable and interpretable.

4.1.3 Schematic

On GitHub there you can see the schematic. The amplifier is at the center of the design, surrounded by various supporting components. The inputs BIOIN-, BIOIN+, and AMPREF are connected to a muscle. Additional components provide not only signal amplification but also filtering. A band-pass filter is implemented between 72Hz and 720Hz. The sensor operates within a voltage range of 3V to 26V.

4.2 Software

The code for this sensor specifically can be found on `https://github.com/BrtPck/IoT-Systems/tree/main/Team%20Posture`

The sensor we chose already came with an initial code that was very easy to understand. In this code they worked with an envelop of the values they read. There we could see that when a muscle is contracted the envelope value would rise to a higher value. This was usefull for our implementation so we could see when the musician would gave a bad posture.

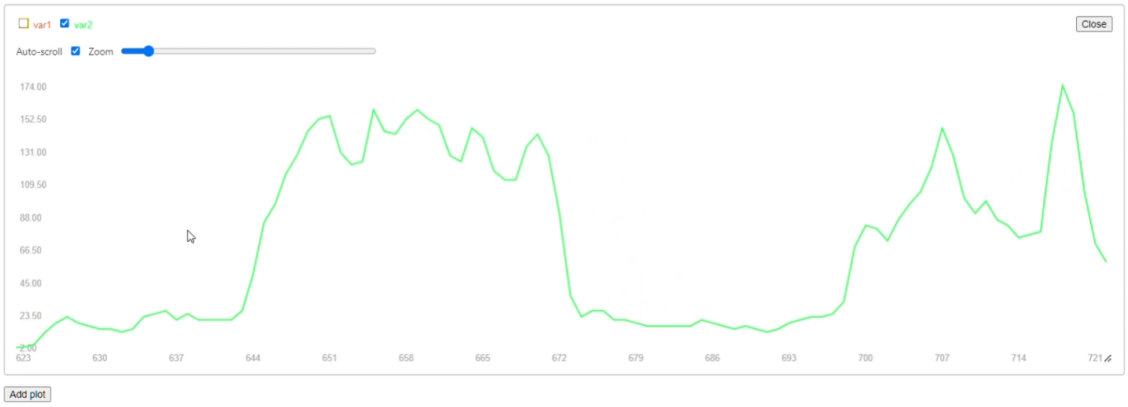


Figure 4.3: Output EMG sensor

We then just created a few new functions wich includes a calibratesensor function and a check posture.

The calibrate sensor was made to check the minimum and maximum values we could read when the musician was sitting. We then used these values to create a value range and with that value range we could make a threshold value. Its that threshold value that will indicate if a musician has a bad posture or not. This is then done in the check posture function.

4.3 Problems

For this project there are also a few problems. The problem we had most trouble with was noise. In the lab classrooms there was a lot of noise that interfered with our sensor. This led to misreadings and a lot of frustrations. This was than fixed by sitting somewhere else our just unplugged the power of the laptop.

Another problem was that when we anlysed the data we get we came to the conclusion that we couldn't measure a bad posture. What we could then measure was the change in position. We only could see when a person was using his muscle to change the posititon he was in. But when the person just sat in a bad position the change wasn't big enough to be detected.

We also think that when a musician is playing, he will need to use the muscle we monitor. This would indicate he is sitting bad, but actually he is just playing his instrument. This

we discussed and we combinded our solutions in the next section.

4.4 Alternatives

Monitoring posture with a single EMG sensor is not ideal, as it can only capture data from one muscle at a time. Our solution would be that we use minimum two EMG sensors, one on each side of the back. This does lead to the disadvantage that the musician may be more bothered. Another alternative to the selected sensor would be pressure sensors.

These could be mounted on the musician's chair, as shown in Figure 4.4.

Figure 4.4: Alternative setup using pressure sensors.

This alternative has the advantage of not requiring multiple EMG sensors to be attached to the musicians, thereby increasing comfort during performances.

5.1 Global Introduction

As mentioned in previous chapters, the gateway is responsible for receiving data from the nodes and forwarding it to the cloud.

In the requirements set by the client, real-time feedback was requested. To achieve this, we opted to work with a cloud solution. The data can be immediately visualized on a cloud-based dashboard.

Several factors were considered when implementing this system. Firstly, the chosen technology for cloud connectivity needed sufficient coverage to function reliably across Belgium, including in challenging locations such as underground concert halls. Additionally, the communication between the nodes and the gateway must span a reasonable distance. For example, if the gateway is placed at the far end of a concert hall, the system should still operate effectively, even in larger venues.

Since the gateway is not battery-powered, the primary focus was on minimizing the power consumption of the nodes. This prioritization influenced the choice of communication methods between the nodes and the gateway.

5.2 Communication

We chose NB-IoT in the LTE spectrum and LTE-M due to their extensive coverage, which surpasses that of LoRaWAN and similar technologies, ensuring reliable operation across larger areas. Additionally, we recognize that concert halls are often well-insulated structures that block electromagnetic signals. However, these venues frequently include signal boosters for mobile reception. By selecting the LTE spectrum, which benefits from these boosters, we can ensure robust and consistent communication.
By choosing Arduino Cloud, we leveraged its user-friendly interface and seamless integration with Arduino Boards, specifically the MKR NB 1500 in our project. The platform's optimization for Arduino devices ensures a smooth development process, minimizing setup complexities and maximizing efficiency. Additionally, Arduino Cloud offers an intuitive and visually appealing dashboard that allows us to effectively monitor and visualize the data from our sensors in real time.

The combination of NB-IoT/LTE-M for reliable connectivity and Arduino Cloud for streamlined data management and visualization forms a robust solution tailored to our project's requirements.

5.3 Hardware

For the hardware, we began with the Arduino MKR NB1500 platform because it was readily available in our lab, allowing us to start testing while waiting for our PCB to arrive. This platform supports NB-IoT and LTE-M communication through the SARA-R410M-02B module and features the SAMD21G processor, which we chose for its proven performance in previous university projects. The SAMD21G is ideal for applications requiring support for multiple communication protocols, low power consumption, cost-efficiency, compact size, and sufficient processing power. Using this platform also offered the advantage of starting with an existing schematic, streamlining the development process (see the schematic in Appendix 7.2).

Next, we selected the RFM95W-868S2 LoRa module, which operates on the 868 MHz band. This module was chosen for its ease of integration, low power requirements, and robust +20 dBm signal strength. Our familiarity with the module, which we had used in the Dramco Uno project, further reinforced our decision. The RFM95W communicates via SPI, though we did not use the standard SPI pins initially. For future iterations, we plan to adopt the standard configuration to simplify programming and improve compatibility.

For temperature measurement, we selected the TMP102 sensor, known for its compact size, affordability, and accuracy. The TMP102 uses the I^2C protocol. Additionally, to monitor sound levels, we integrated the ICS 43434 microphone, which covers a wide frequency range (60 Hz–20 kHz) and offers low power consumption. This makes it suitable for environments like concerts, where sound levels can reach up to 120 dB.

To simplify installation, we chose a micro USB port over USB-C or mini USB, as it is easier to install on the board and more commonly used than mini USB, while still providing good compatibility with a range of devices. We also added PESD diodes on the communication lines to protect sensitive components from voltage spikes, including on the lines between the SIM card and the SARA-R410M module.

The images of the gateway PCB design and 3D view are shown in Figures 5.1 and 5.2, respectively, while a photo of the completed gateway is shown in Figure 5.3.

A complete Bill of Materials (BOM) for the project can be found on the GitHub repository.

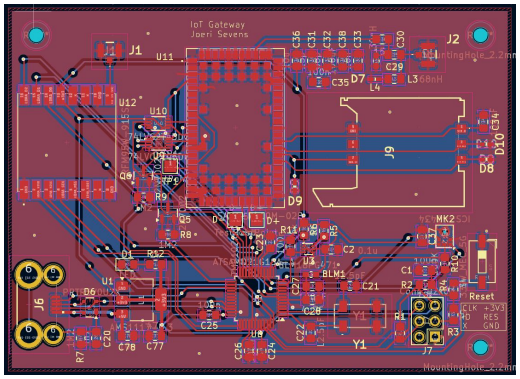


Figure 5.1: Gateway PCB

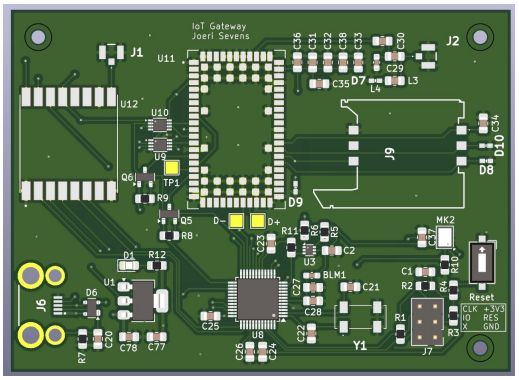


Figure 5.2: 3D view Gateway

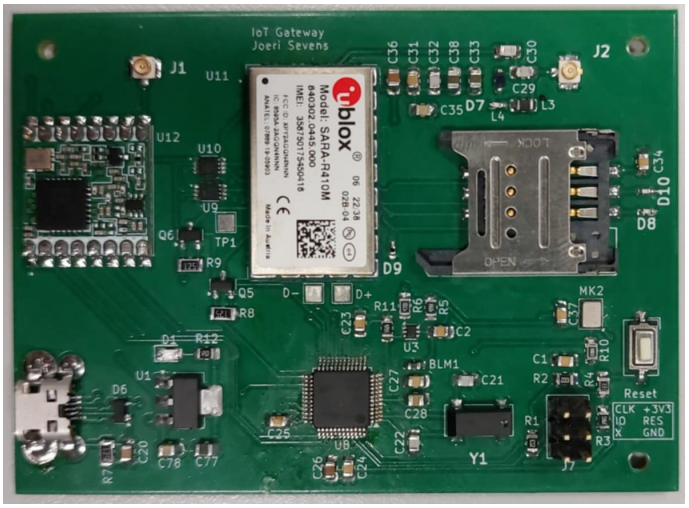


Figure 5.3: Photo gateway

5.4 Code

The protocol chosen for communication between the nodes and the gateway was designed to maximize the battery life of the nodes. The implementation can be divided into two main parts.

The first part is the initialization process, during which a registration period is defined (ideally configured via the cloud). This allows researchers to decide how much time they need to set up and activate all nodes with the musicians. For example, if only one musician needs to be monitored, a 1-minute initialization period might suffice. However, if there are 20 nodes, they might allocate 25 minutes for initialization.

Nodes can register by sending a valid ID. A valid ID is 2 bytes long, starts with a letter (e.g., 'A'), and is followed by a number (0 to F in ASCII). Invalid IDs are ignored and not registered. If a node with an already registered ID attempts to register again (e.g., due to a restart or power failure of the node during the initialization process), the system does not register it again but instead resends an acknowledgment.

The gateway sends an acknowledgment (ACK) for each successful registration. This ACK includes the node's ID, the time when it is allowed to start transmitting (once the initialization time is over), and the interval for subsequent transmissions. This mechanism ensures that nodes never transmit simultaneously, preventing interference.

The initialization process is illustrated in the flowchart below, see Figure 5.5.

Once the initialization loop is complete, meaning the set initialization time has elapsed, the code transitions to the data loop, see figure 5.6. In this phase, the code continuously listens for incoming data from the nodes. This process could be further optimized, as we know the total number of nodes and their transmission schedules. For example, the system could enter deep sleep between transmissions from two nodes to save power.

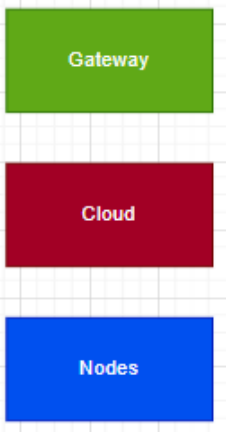


Figure 5.4: Legend flowchart

When data is received, it is first verified whether it originates from a valid node (an ID registered during the initialization process). Once validated, the data stream is read and forwarded to the cloud.

5.5 Introduction - GUI

The data collected from various sensors must be aggregated and presented in a graphical format to facilitate analysis and interpretation. To achieve this, a graphical user interface (GUI) is essential. This chapter discusses the initial design considerations, challenges encountered, and the final solution implemented for data visualization. Additionally, it highlights the steps taken to ensure reliability and efficiency in handling sensor data.

5.6 Initial Approach: Cloud-Based Solution

Originally, a cloud-based solution was proposed using Arduino Cloud. The concept involved transmitting sensor data via a LoRa module to a gateway, which then used an NB-IoT connection, facilitated by a SARA module and a SIM card, to send the LoRa-received data to the cloud. Researchers could access and visualize the data through the Arduino Cloud platform. A visualization of this principle can be seen in image 5.7.

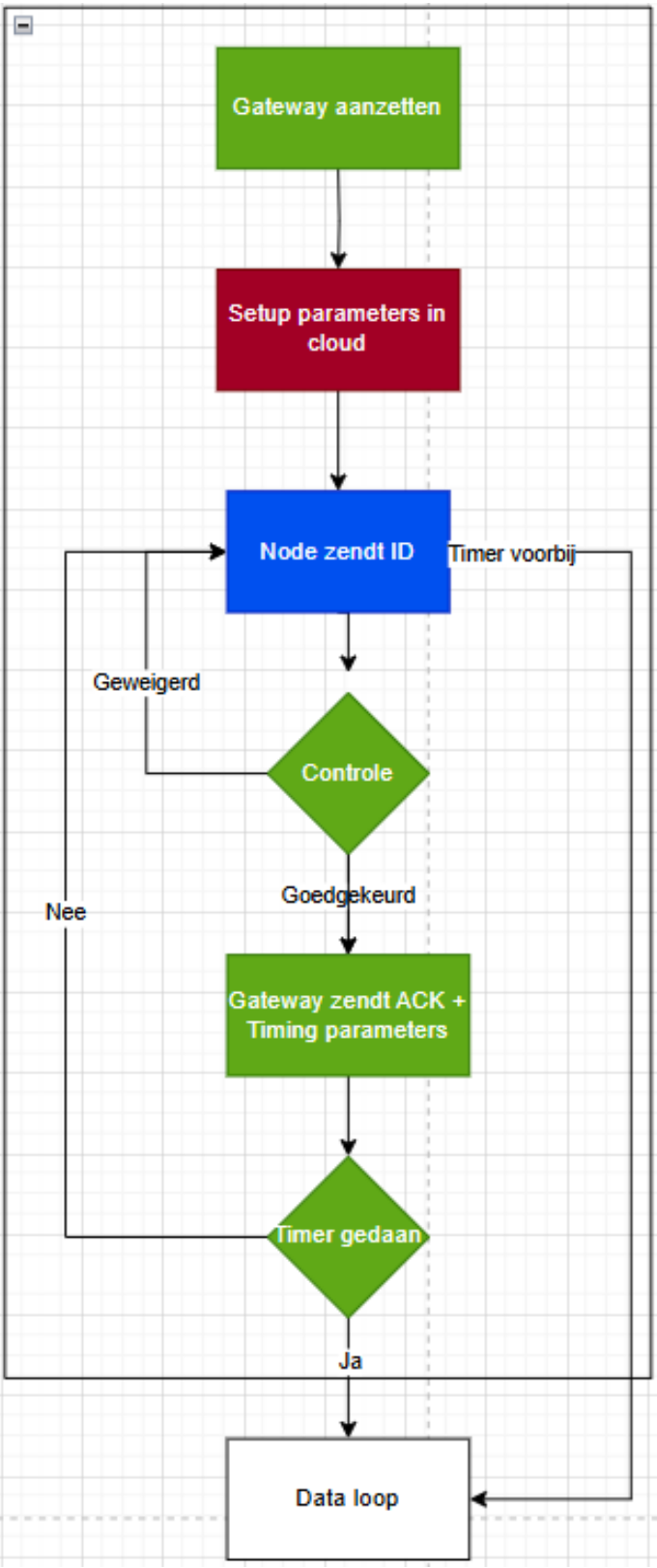


Figure 5.5: Initialization code flowchart

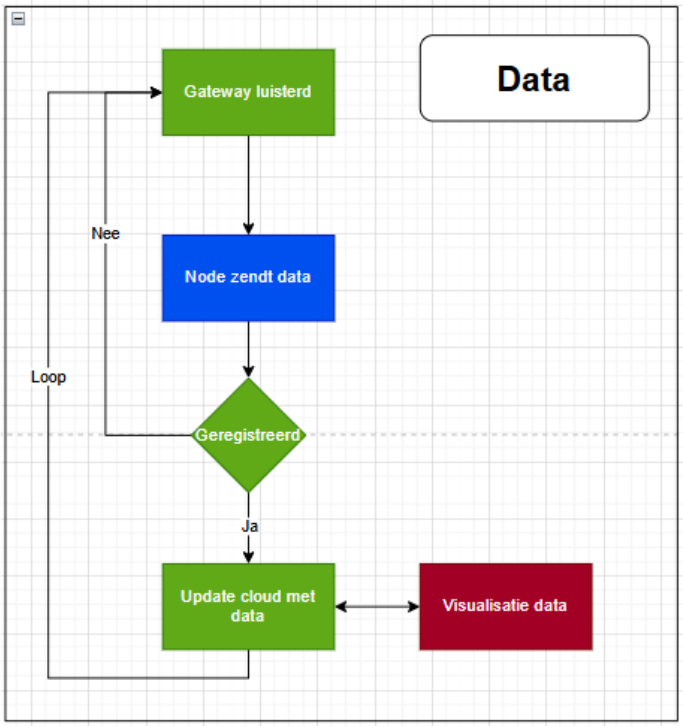


Figure 5.6: Data code flowchart

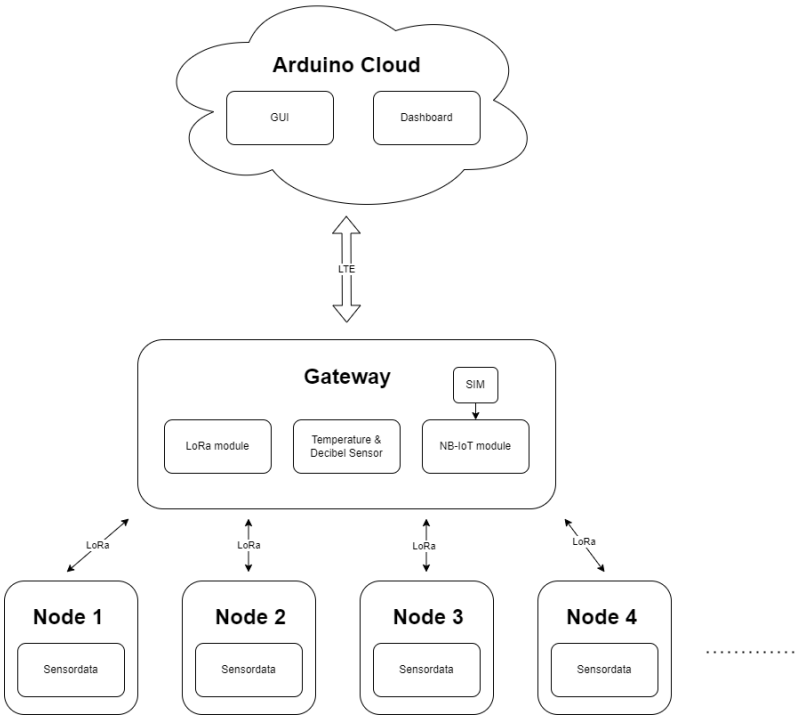


Figure 5.7: Block diagram Arduino Cloud

However, this approach presented significant challenges. The NB-IoT connection to the cloud proved to be unreliable, leading to periodic disconnection from the cloud. Additionally, the cloud service required a paid subscription, which we want to avoid. Lastly, the chip lacked essential firmware capabilities to fully support the desired operations. Due to these issues, the cloud-based solution was deemed unsuitable, prompting the development of a more reliable local monitoring approach.

5.7 Solution: Local Data Monitoring

The implemented solution relies on a local setup for data collection and visualization. Sensor data is still transmitted through LoRa to a gateway device. The received data are now printed on the serial monitor on the gateway. A computer connected to this serial monitor makes it possible to read and process these data locally using Python scripts. After consultation with the researchers, the temperature and decibel sensor were also removed from the gateway as this data is not required. We can find the updated block diagram on figure 5.8.

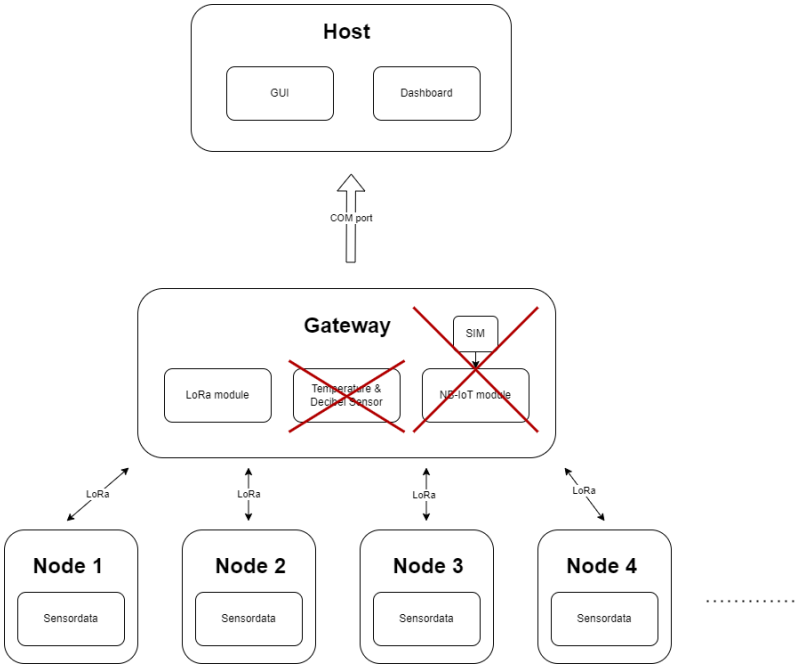


Figure 5.8: Block diagram local host

This approach eliminates the reliance on cloud connectivity and subscription fees, but introduces the requirement that a computer be able to handle real-time monitoring. To support this setup, two Python programs were developed:

1. **Serial Monitor Listener:** This program reads incoming data from the serial monitor and saves it in a structured CSV file format. The CSV file serves as a persistent storage medium, enabling offline analysis and archiving.
2. **Data Visualization Tool:** This program processes the CSV file, allowing users to select a desired node or sensor and generates dynamic, clear graphs.

5.8 Overview

This section provides a step-by-step guide on how to effectively use the scripts to measure and visualize the sensor data from the nodes. Both GUI's are visible in figures 5.9 and 5.10.

5.8.1 Program 1: `LoRa_to_CSV.py`

1. **Registered Nodes Box:** Displays all successfully registered nodes.
2. **Debug Information Box:** Provides critical information regarding node registration and received data for debugging purposes.
3. **Error Messages Box:** Displays error messages for duplicate or invalid IDs, corrupt, incorrect, or invalid data.
4. **Node Data Box:** Shows real-time data transmitted from nodes.

5.8.2 Program 2: `CSV_to_graph.py`

5. **CSV File Input:** Users can load the desired CSV file. Files located in the same directory appear in a dropdown menu, while others can be selected via the file browser.
6. **Node Selection:** After loading a CSV file, users can load all nodes by clicking the "Load Nodes" button and select a specific node from a dropdown menu.
7. **Sensor Data Selection:** Allows users to choose desired sensor data by checking corresponding boxes.
8. **Graph Generation:** By clicking the "Show Graphs" button, a graph is generated based on the selected parameters for further analysis of sensor data.

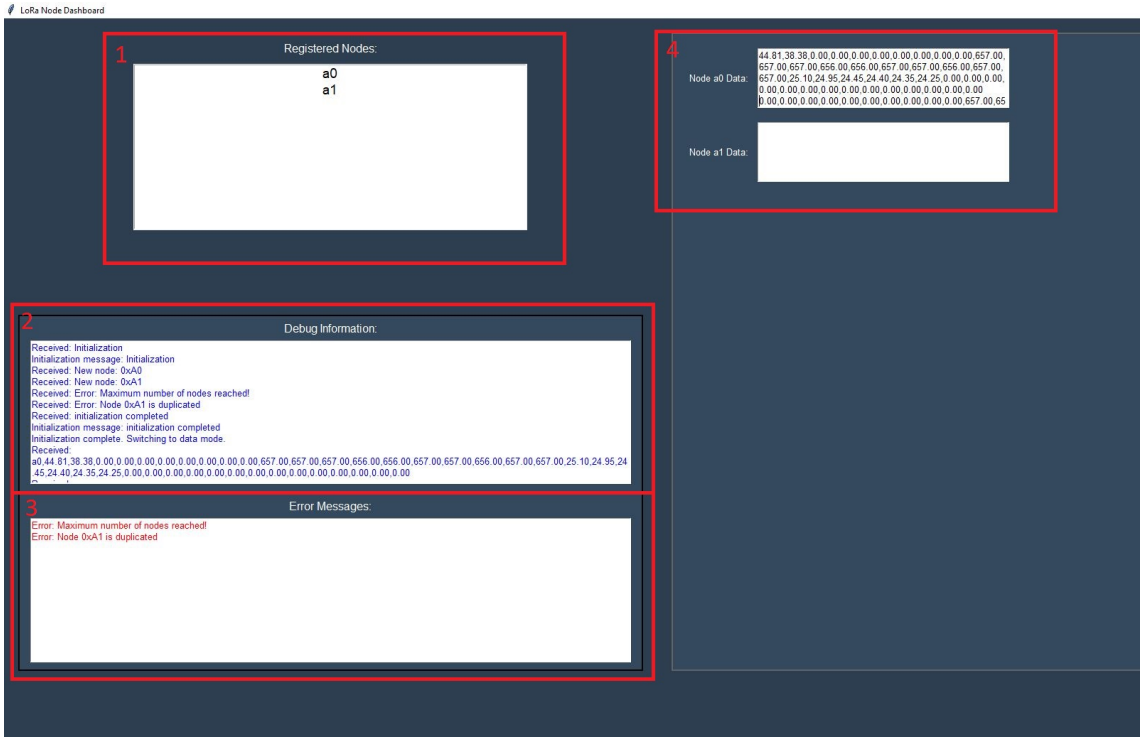


Figure 5.9: GUI: LoRa_to_CSV

5.9 Step-by-Step Instructions

1. **Connecting the Gateway:** Connect the gateway to the computer using a micro-USB cable. This establishes a connection via a COM-port.
2. **Starting the `LoRa_to_CSV.py` Script:** Run the `LoRa_to_CSV.py` script to launch the GUI. The program automatically detects the COM port to which the gateway is connected. An initialization timer starts immediately, and all incoming data is displayed in the **Debug Information Box (2)** for real-time debugging.
3. **Node Registration:** Nodes can now connect to the gateway by registering with their IDs:
   - Successfully registered IDs are displayed in the **Registered Nodes Box (1)**.

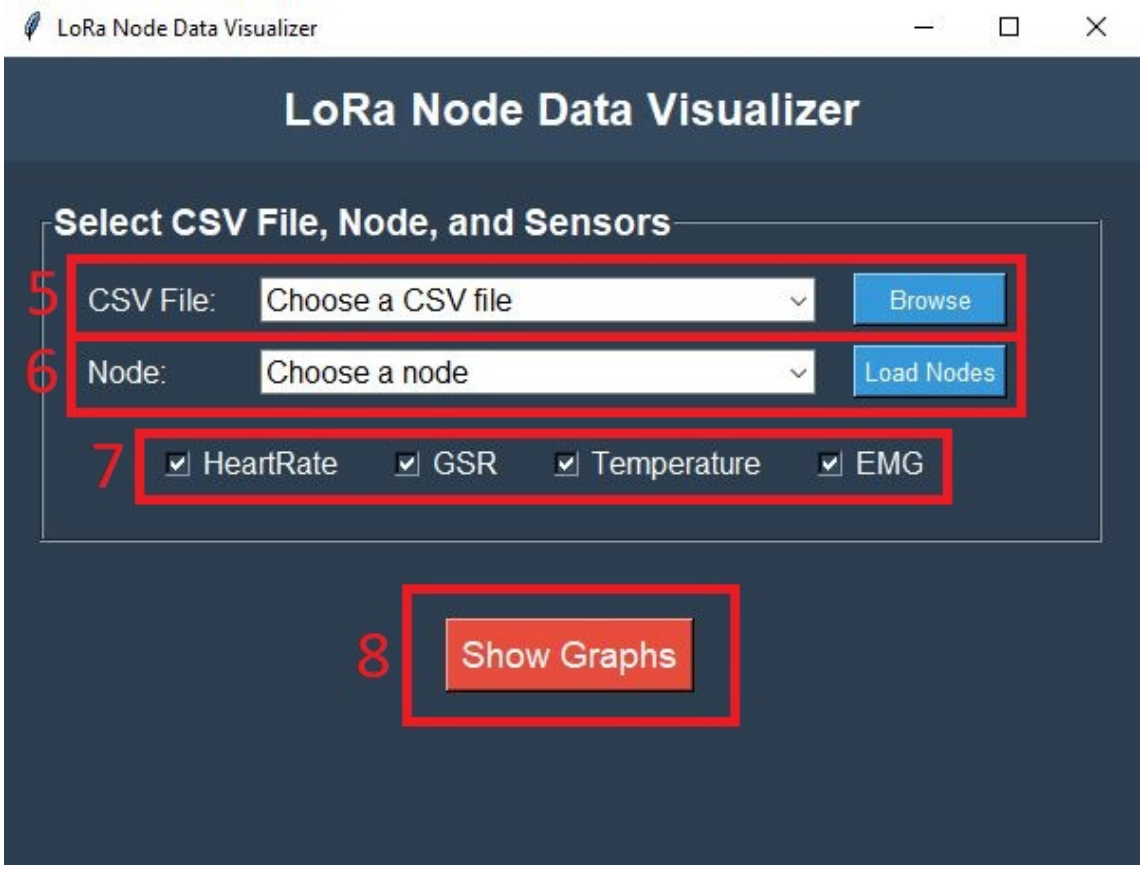


Figure 5.10: GUI: CSV_to_graph

- Duplicate or invalid IDs are rejected, and corresponding error messages appear in the **Error Messages Box (3)**.
- The gateway responds to registered nodes with an acknowledgment (ACK) message, including information on when they may transmit data.

4. **Node Data Display:** Each registered node automatically receives a dedicated **Node Data Box (4)** on the left-hand side of the interface, where its real-time data is displayed.
5. **Node Data Reception:** Once the initialization timer completes, a message appears in the **Debug Information Box**, indicating that the gateway is now ready to receive node data. Incoming data for each node is:
   - Displayed in its respective **Node Data Box (4)** for real-time confirmation.
   - Automatically saved to a CSV file named `lora_data_1.csv`.
6. **Starting the `CSV_to_graph.py` Script:** Once data collection is complete, run the `CSV_to_graph.py` script to open the graphing GUI.
7. **Selecting a CSV File:** Choose the desired CSV file using the dropdown menu or the **Browse** button **(5)**.
8. **Loading and Selecting Nodes:** Click the **Load Nodes** button to load all nodes from the selected CSV file, then select the desired node using the dropdown menu

(6).

9. **Selecting Sensor Data:** Check the boxes corresponding to the desired sensor data **(7)**.
10. **Generating Graphs:** Click the **Show Graphs** button **(8)** to generate and display the sensor data graphs for analysis.

Output: The sensor data will be visualized as shown in Figure 6.2.

The visualized data exhibits unexpected behavior. This will be further discussed in section 6.1.

5.10 Advantages and Limitations

5.10.1 Advantages

The local monitoring approach offers several key benefits:

- **Reliable Data Reception:** The LoRa gateway ensures consistent and robust data communication.
- **Cost-Effectiveness:** The solution avoids recurring cloud subscription fees.
- **Enhanced Data Control:** Local data processing provides greater flexibility and security, as sensitive information remains within the project's infrastructure.
- **Customizability:** The Python scripts allow customized data visualization, accommodating diverse analysis requirements.

5.10.2 Limitations

Despite its advantages, the solution has certain limitations:

- **Dependence on a computer:** A dedicated computer is required for real-time monitoring and data processing.
- **Limited collaboration:** A local GUI makes real-time collaboration challenging, as it cannot be easily accessed by multiple users remotely. This limits the ability to work simultaneously on the same project unless additional tools are implemented.

5.11 Costs

The total cost of the gateway can be divided into hardware and subscription fees. The hardware costs amount to €120.23 and include various components such as the PCB (€6.42), the SAMD21G microcontroller (€4.13), the SARA R410M module (€54.18), the RFM95 module (€21.00), the TMP102 temperature sensor (€0.87), the ICS 43434 microphone (€2.00), antennas (€20.00), an LDO (Low-Dropout Regulator) (€1.63), and additional components (€10.00).

In addition to the hardware, subscription fees are required to maintain operation. These include a SIM card subscription costing €4 per month and access to the Arduino Cloud for €6 per month, resulting in a total subscription cost of €10 per month.

This breakdown highlights both the upfront investment in hardware and the ongoing expenses necessary for the gateway's continuous operation and connectivity.

For the final testing of our prototype, a pianist was monitored during a jazz repetition. All the sensors were attached and ultimately, the node communicated with the gateway as expected. Due to some sensors not being attached correctly the data is not always correct.

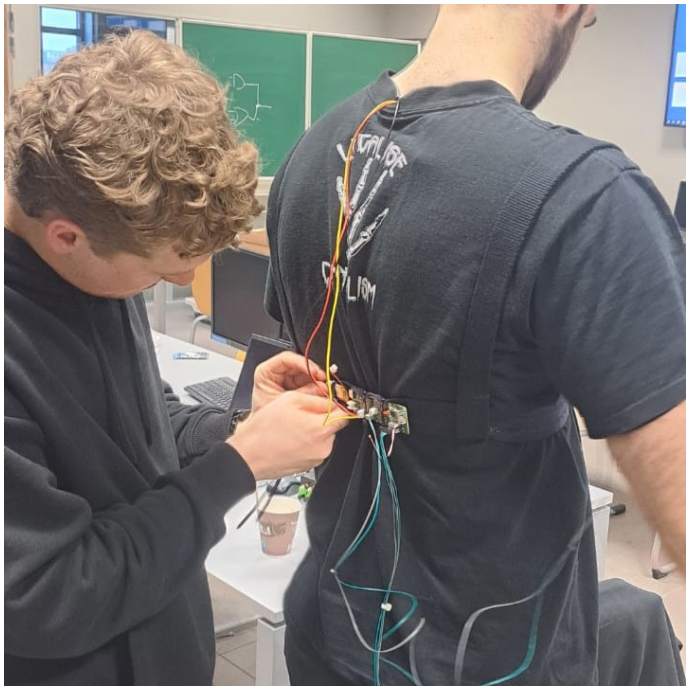


Figure 6.1: Proof of concept node

6.1 Test results

Figure 6.2 displays our measurements of our sensors on the pianist during the rehearsal. It is clear that the heart rate sensor was not working as expected. We identified the cause of these measurements to be the placement. The sensor was placed in the sock of the pianist and was not secured in place. During the first seconds of the play it got dislodged and stopped measuring. **Takeaway 1: Secure the sensors**. Secondly it appears that the GSR sensor was not varying as much as during the pre-rehearsal testing. This has either to do with our makeshift solution (the original sensors were lost) or with the positioning. We may need to identify a part of the body that is more prone to sweating. Thirdly, the temperature sensors measured very inconsistently. Since this sensor was integrated in the heart rate sensor, it is clear that the securing of the sensor is the issue here. Finally the

EMG sensor did not provide any useful data, but at least, there was some exchanged data.
Takeaway 2: Choose a different EMG sensor.

6.2 Remarks orchestra

The pianist told us that the harness was noticeable but did not interrupt his playing, stating "It felt like wearing pants with suspenders."

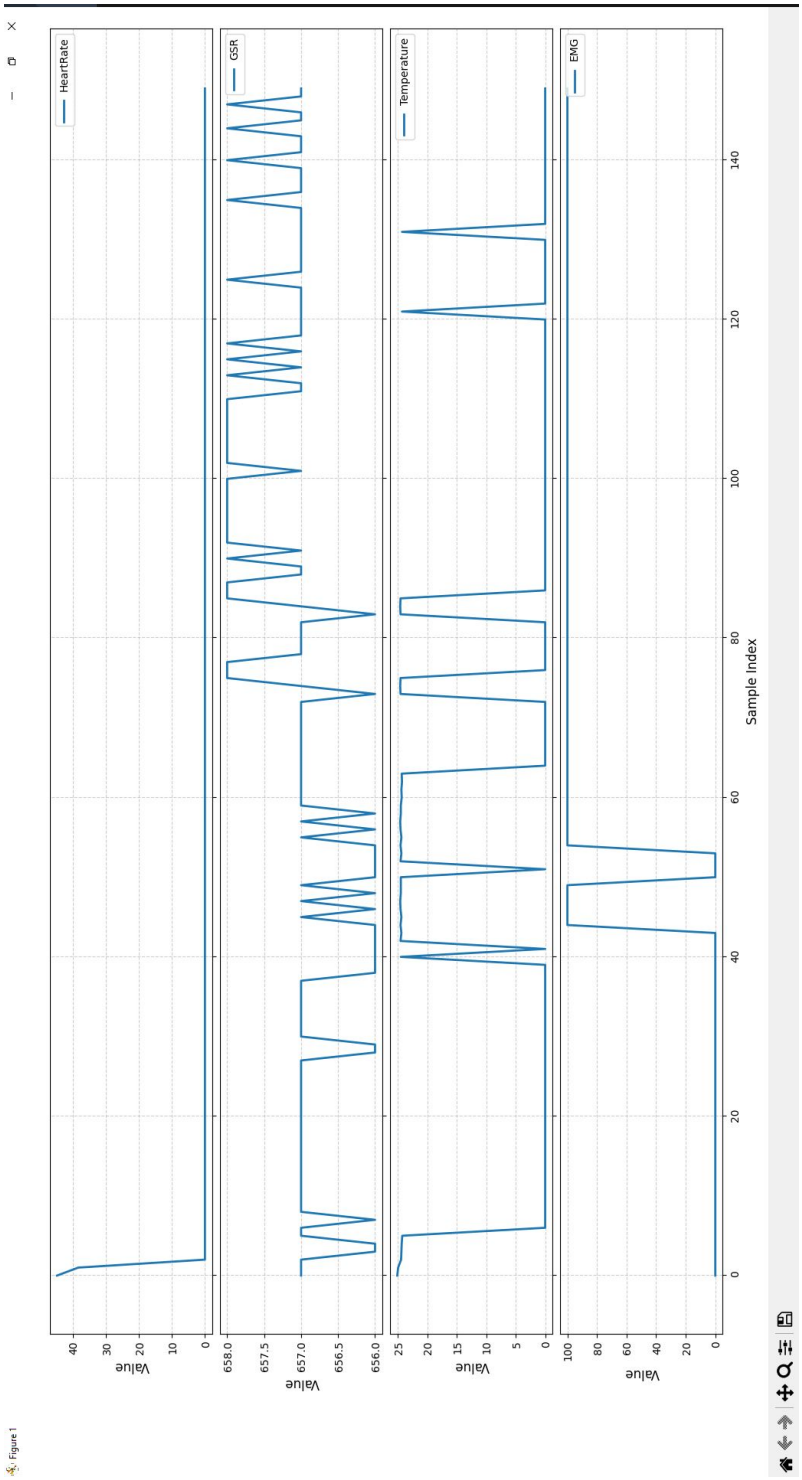


Figure 6.2: Output sensor data

We conclude that our module was overall successful, fulfilling all requirements and demonstrating energy efficiency. However, there are several important considerations and areas for improvement that we need to address.

7.1 Possible improvements

- LoRa communication can be more energy efficient. Lowering power, sending less data (raw bytes instead of strings), lowering the spreadfactor resulting in slower chips so less power consumption.
- Change the EMG sensor altogether.
- Secure the sensors more tightly
- Redesign the gateway PCB to ensure NB-IoT connectivity.
- Find a suitable cloud solution
- We could have achieved way greater measurement resolution by avoiding the sending of data altogether. By storing the data locally on each node, we could have achieved way greater battery life, with more frequent measurements.
- Better dimensioning of the crystal and its associated components.
- The introduction of sufficient test pads would facilitate easier debugging and testing during future iterations.

7.2 Future directions

The research can be further developed based on the documentation we provide in our github repository:`https://github.com/BrtPck/IoT-Systems/tree/main`.

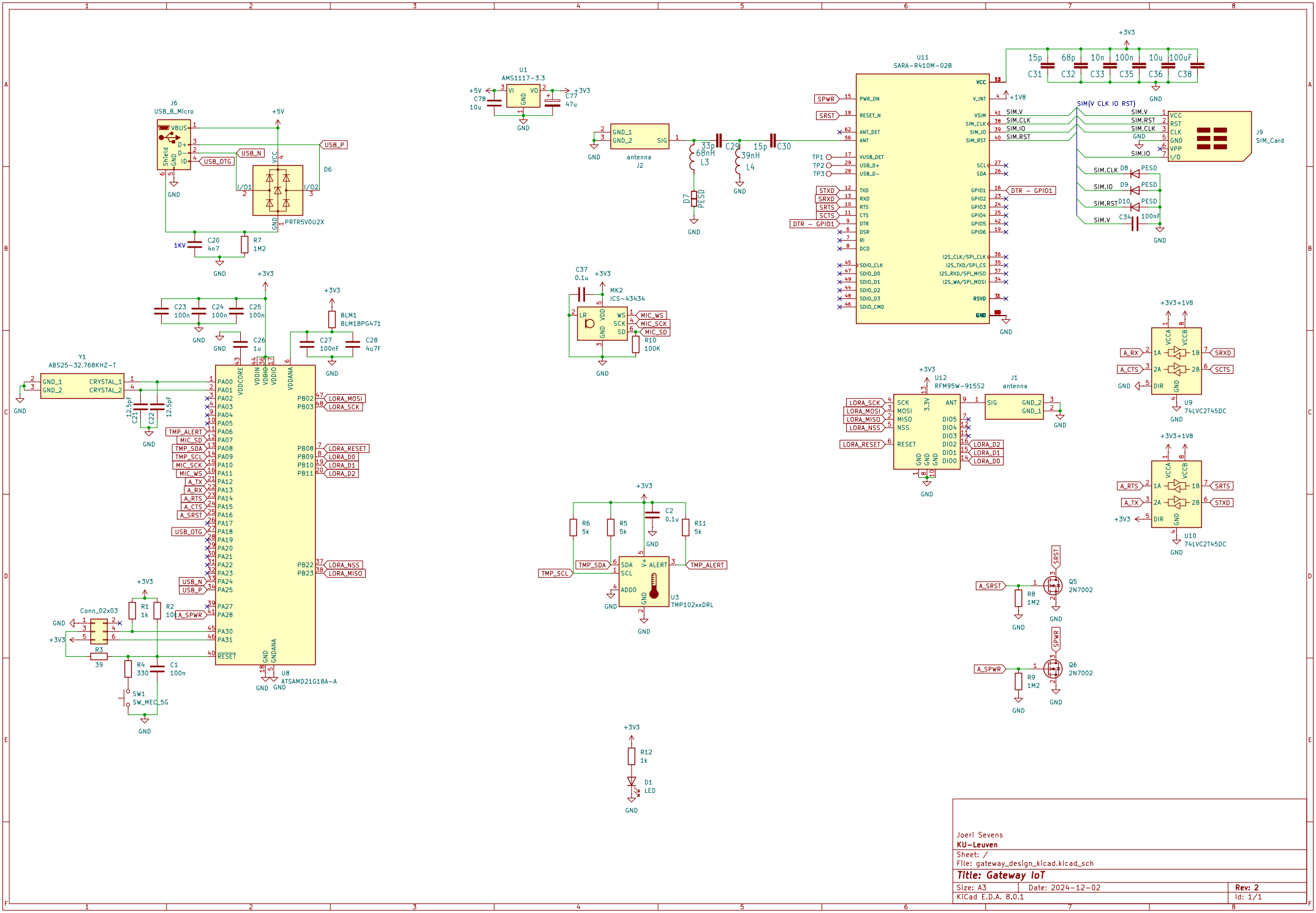
Bibliography

[1] Abdullah A. Al-Atawi et al. “Stress Monitoring Using Machine Learning, IoT and Wearable Sensors”. In: *Sensors* 23.21 (2023). ISSN: 1424-8220. DOI: 10.3390/s23218875. URL: https://www.mdpi.com/1424-8220/23/21/8875.

[2] Seeed Studio. *Grove - GSR Sensor*. Accessed: 2024-12-17. 2024. URL: https://wiki.seeedstudio.com/Grove-GSR_Sensor/.

[3] Guus Leenders et al. “An Energy-Efficient LoRa Multi-Hop Protocol through Preamble Sampling for Remote Sensing”. In: *Sensors* 23.11 (2023). ISSN: 1424-8220. DOI: 10.3390/s23114994. URL: https://www.mdpi.com/1424-8220/23/11/4994.

[4] Abracon. *ABS25 Series - 2.5mm x 2.0mm Crystal*. Accessed: 2024-12-17. 2024. URL: https://abracon.com/Resonators/abs25.pdf.


J6
USB_B_Micro
VBUS
D+
D-
ID
GND
Shield
USB_N
USB_P
USB_OTG
+5V
VCC
I/O1
I/O2
GND
PRTR5V0U2X
D6
1KV
C20
4n7
R7
1M2
GND
U1
AMS1117-3.3
VI
VO
GND
+5V
+3V3
C78
10u
C77
47u
GND_1
GND_2
SIG
antenna
J2
33p
68nH
L3
C29
15p
39nH
L4
C30
D7
PESD
GND
U11
SARA-R410M-02B
SPWR
SRST
PWR_ON
RESET_N
ANT_DET
ANT
VUSB_DET
USB_D+
USB_D-
TP1
TP2
TP3
TXD
RXD
RTS
CTS
DTR
DSR
RI
DCD
STXD
SRXD
SRTS
SCTS
DTR - GPIO1
SDIO_CLK
SDIO_D0
SDIO_D1
SDIO_D2
SDIO_D3
SDIO_CMD
VCC
V_INT
+1V8
VSIM
SIM_CLK
SIM_IO
SIM_RST
SCL
SDA
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
I2S_CLK/SPI_CLK
I2S_TXD/SPI_CS
I2S_RXD/SPI_MISO
I2S_WA/SPI_MOSI
RSVD
GND
15p
C31
68p
C32
10n
C33
100n
C35
10u
C36
100uF
C38
SIM{V CLK IO RST}
SIM.V
SIM.CLK
SIM.IO
SIM.RST
J9
SIM_Card
VCC
RST
CLK
GND
VPP
I/O
D8
D9
D10
PESD
C34
100nF
C23
100n
C24
100n
C25
100n
C26
1u
BLM1
BLM18PG471
C27
100nF
C28
4u7F
Y1
ABS25-32.768KHZ-T
GND_1
GND_2
CRYSTAL_1
CRYSTAL_2
12.5pF
C21
C22
12.5pF
C37
0.1u
MK2
ICS-43434
LR
GND
VDD
WS
SCK
SD
MIC_WS
MIC_SCK
MIC_SD
R10
100K
U12
RFM95W-915S2
SCK
MOSI
MISO
NSS
RESET
3.3V
ANT
DIO5
DIO4
DIO3
DIO2
DIO1
DIO0
LORA_SCK
LORA_MOSI
LORA_MISO
LORA_NSS
LORA_RESET
LORA_D2
LORA_D1
LORA_D0
J1
antenna
U9
74LVC2T45DC
VCCA
VCCB
DIR
A_RX
A_CTS
SRXD
SCTS
U10
74LVC2T45DC
A_RTS
A_TX
SRTS
STXD
PA00
PA01
PA02
PA03
PA04
PA05
PA06
PA07
PA08
PA09
PA10
PA11
PA12
PA13
PA14
PA15
PA16
PA17
PA18
PA19
PA20
PA21
PA22
PA23
PA24
PA25
PA27
PA28
PA30
PA31
RESET
VDDCORE
VDDIN
VDDIO
VDDANA
GNDANA
PB02
PB03
PB08
PB09
PB10
PB11
PB22
PB23
TMP_ALERT
MIC_SD
TMP_SDA
TMP_SCL
MIC_SCK
MIC_WS
A_TX
A_RX
A_RTS
A_CTS
A_SRST
USB_OTG
USB_N
USB_P
A_SPWR
LORA_MOSI
LORA_SCK
LORA_RESET
LORA_D0
LORA_D1
LORA_D2
LORA_NSS
LORA_MISO
U8
ATSAMD21G18A-A
Conn_02x03
R1
1k
R2
10
R3
39
R4
330
C1
100n
SW1
SW_MEC_5G
R6
5k
R5
5k
C2
0.1u
R11
5k
TMP_SCL
TMP_SDA
TMP_ALERT
SDA
SCL
V+
ALERT
ADD0
U3
TMP102xxDRL
R12
1k
D1
LED
A_SRST
R8
1M2
Q5
2N7002
A_SPWR
R9
1M2
Q6
2N7002
Joeri Sevens
KU-Leuven
Sheet: /
File: gateway_design_kicad.kicad_sch
Title: Gateway IoT
Size: A3
Date: 2024-12-02
Rev: 2
KiCad E.D.A. 8.0.1
Id: 1/1