UCLouvain

LELEC2102 - Project in wireless embedded sensing systems

# R6 - Global system characterization and analysis


**GROUP E**

Baudoux Romain **31602100**
Bertrand Van Ouytsel Victor-Emmanuel **37442100**
Carballes Cordoba Victor **34472100**
Carbonnelle Gautier **52282000**
Chowdhury Kamrul Islam **19072400**
Loog Aurélien **80802000**


Academical year 2024-2025

# Contents

# 1 Introduction

The modern world is ever-changing, new technologies emerge every day. It is possible to communicate across the world in the blink of an eye, to fly over oceans, watch the movements of electrons and the birth of stars millions from light-years away. However, all the advancements of modern science should not distract us from the significance of the natural world we inhabit. While we have constructed cities of astonishing size, natural ecosystems still cover most of Earth's emerged surface. As we live on the brink of a new massive extinction event, it is crucial more than never before to protect nature and its various ecosystems. Among those ecosystems, forests represent a large portion of them. These particular habitats are especially important for nature's balance. They are reservoirs of biodiversity, a particular important role as every year, WWF[1] estimates that between 200-2000 species go extinct annually. Furthermore, forest help fight against climate change by storing and converting carbon dioxide to breathable oxygen. They also are an important component of the economy for many countries all around the world.

Unfortunately, forests are far from doing well. They suffer due to both environmental events and human activity. Wildfires are part of the natural cycle of forests, but due to climate change, small fires that could help renew the ecosystem tend to grow out of proportion, becoming huge uncontrollable infernos more and more frequently. They are responsible for massive ecological and economical losses as well as countless human tragedies. The fires aren't the only thing threatening nature's reservoir. Human activities, including poaching and illegal deforestation, pose significant threats to both flora and fauna.

All hope is not lost as technology can help us. By remotely monitoring abnormal events, it is possible to learn more about the different events occurring in the forest. This can lead to multiple applications in various scientific fields but more importantly it could be used to prevent catastrophes before they get out of hands. It is easier to put out a fire when it is only a few meters wide rather than hundreds of kilometers wide. It can also be used to react faster to illegal human activity. This is where, us engineers, can make an impact. The goal is to implement a remote wireless system to detect the audio signals generated in the forests, process them, monitor if any are out of the ordinary and in particular if they could be linked to a specific activity (birds, fire, chainsaws, etc) to finally decide to intervene or not.

The goal of this report is to deliver a complete snapshot of the global performances of the system. This will be the foundation for the work on the optimization of the system next semester. The snapshot is based on important quantitative information obtained from experimental measures. These measures where obtained using a rigorous protocol. This allows for good comparison of the experimental characterization with the theory both from the lectures and additional sources. The analysis of this multi-blocks complex system is especially important as it necessary for understanding the various trade-offs between different performance metrics.

The structure of the report is as follows: the System Overview section provides a recap of the entire system, detailing its operations and hardware components. The Characterization and Analysis section includes multiple subsections covering the accuracy of classification using real acoustic signals, communication distance, power consumption of the smart sensor, sensing duty cycle (the proportion of time audio signals are actively acquired), and FPGA resource usage. Finally, the report concludes with a summary that highlights the key aspects of system characterization, serving as a foundation for optimization efforts in the next semester.

# 2 System overview

The system architecture is comprised of key components such as a transmitter, receiver, and processing modules that work cohesively to monitor acoustic signals. At the transmitter end, a microphone interface captures analog sound signals, which undergo signal conditioning to enhance their quality. These signals are converted into digital form through sampling and quantization, enabling further processing. Critical features are extracted from the audio data using time-frequency analysis techniques, such as Mel-spectrogram transformations.

The receiver module facilitates wireless reception, noise filtering, and packet decoding to retrieve transmitted data. Classification algorithms are employed to identify sound types, such as natural or anthropogenic noises, using features derived from the transmitted signals.

[1] World Wide Fund for Nature

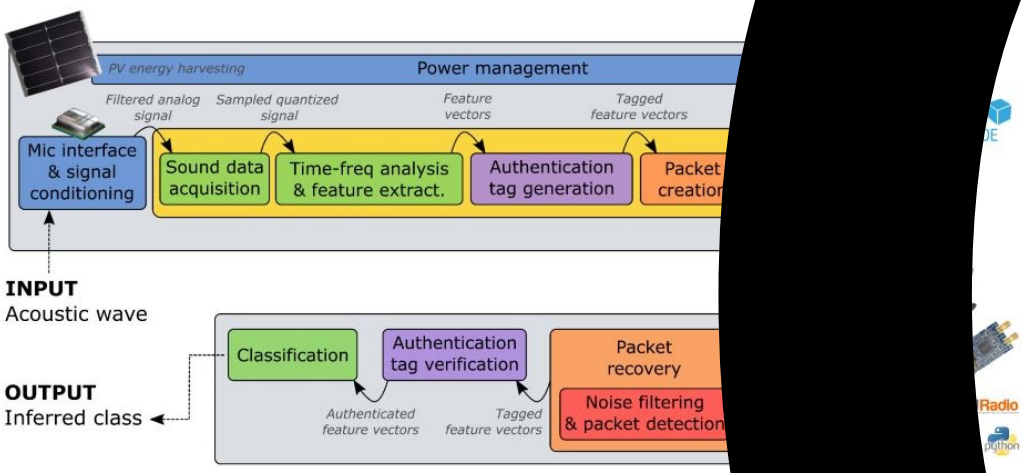



Figure 1: System Level Block Diagram

The system is effectively divided into two main parts as you can see in Figu

## 2.1 Transmitter (TX):

This component focuses on capturing, processing, and transmitting audio signal prise: a microphone, an antenna, a wireless sensor node itself comprised of Nucleo MCU b an audio sensing and PV power-management board as well as USB and audio jack cable project, PV cells will be used to power that part of the system. Its software blocks in the microphone interface, signal conditioning, digital conversion, feature extraction, authen creation and wireless transmission.

- **Mic Interface and Signal Conditioning:** The input of the global signal, which is also t transmitter, are acoustic waves that are measured by the microphone. Those waves are compose sound signals on which the signal conditioning unit applies filtering and amplification to enhance signa This ensures that the captured data is clean and representative of the environment.
- **Sound Data Acquisition:** The analog signals are transformed into digital through sampling and quantization. This transformation enables further digital processing and analysis of the audio data. This step is required as computers can't effectively work with analog signals.
- **Frequency Analysis and Feature Extraction:** The digitized audio signals are analyzed in detail to extract key time-frequency features vectors. Those are essential for identifying characteristics of the sound that are useful for further classification.
- **Authentication Tag Generation:** To ensure data integrity and security, we generate authentication tags for the extracted feature vectors. These tags prevent unauthorized modifications and maintain trust in the transmitted data. This is important as an exterior malicious agent could try to corrupt transmitted information.
- **Packet Creation:** The authenticated feature vectors are then packaged into data packets. We transmit the latter through wireless medium. The transmission through packet is more efficient (bandwidth), reliable, scalable and is quite quick. The speed is important as we want to be able to transmit loads of information if necessary.
- **Wireless Transmission:** At the final stage of the transmission section, these packets are wirelessly transmitted using S2LP programmable radio, ensuring efficient and reliable data transfer.

- **Power Management:** Throughout the whole transmission, the power needs to be managed, distributed and regulated to ensure maximum power efficiency and maximum power autonomy. This is vital for maintaining functionality and extending battery life. One example of power management policy is the fact that we don't record all the time and instead separate time in recording, data processing and classification to save power without disrupting the functionality too much. Furthermore, our system includes a photovoltaic energy harvesting mechanism that will be used next semester.

## 2.2 Receiver (RX):

This part deals with receiving, decoding, and classifying the transmitted signals. Its hardware parts comprise: an antenna, a LimeSDR FPGA/radio board as well as a USB cable. Its software blocks include wireless reception, noise filtering, packet recovery, and classification algorithms to identify sound types.

- **Wireless Reception:** The radio signals emitted from the transmitter are captured at the receiver thanks to the antenna and the LimeSDR FPGA. However those signals have been impacted by noise and other potential propagation effects.
- **Noise Filtering and Packet Detection:** To classify as accurately as possible later on, the noise needs to be filtered out. This is what this block is responsible for as well as detecting the packets which contain the useful information.
- **Packet Recovery:** At this stage we recover the packets from noise and other distortions, we then prepare the decoded packets for downstream processing.
- **Authentication Tag Verification:** To ensure that the data comes from a trusted source, and that none of the information inside has been corrupted or tampered, we use a tag at the end of the packet that authenticates the packet using a secret key and the content of the packet.
- **Classification:** Classification algorithms are used to analyze the feature vectors and identify the type of sound captured. This allows us to distinguish between various different types of sounds: birds, chainsaw, handsaw, fire or helicopter. We then have the information on what event is happening in the forest and can choose to intervene or not.

This structured design enables seamless integration and robust functionality, ensuring the system meets the diverse requirements of environmental audio monitoring.

# 3 Characterization and analysis

## 3.1 Classification

In this section, we evaluate the performance of our classification model across two setups. First, we assess its performance in a simulated environment with augmented data. Next, we test the model's performances in real-world conditions. These steps help us understand the model's strengths, weaknesses, and areas for improvement.

### 3.1.1 Classification in simulation

We have 40 audio files for each of the 5 classes in our dataset. To augment our dataset, we apply a time-shift transformation to all samples. This transformation is particularly important as it enables the model to identify sounds that do not necessarily begin at t=0, increasing its robustness to temporal variations.

After augmentation, we split the dataset into a training set and a test set using a 70/30 ratio and ensuring that all classes are evenly represented in both subsets. This stratified split guarantees that the model is trained uniformly across all classes. We normalize both subsets to improve numerical stability. Normalization is also useful in this context as the norm of each sample is influenced by microphone distance, which does not provide reliable information. By normalizing, we allow the model to focus on the spectral content of the audio, which is more relevant for classification.

We choose a decision tree as the classifier model due to its interpretability and low computational complexity. Additionally, decision trees are inherently capable of capturing non-linear relationships in the data, which is useful for our classification task.

To optimize the hyperparameters of the decision tree, we perform a grid search with 4-fold cross-validation. The hyperparameters we tune are listed in Table 1.

| Hyperparameter | Description |
|---|---|
| max depth | The maximum depth of the decision tree. |
| min samples split | The minimum number of samples required to split an internal node. |

Table 1: Hyperparameters tuned during learning phase

The best set of hyperparameters are { `max depth = 6, min samples split = 4` } . We used the macro F1 score and confusion matrix to evaluate the model. The macro F1 score provides a single measure of overall performance, favoring balanced results across all classes, while the confusion matrix offers detailed insights into class-specific errors, highlighting potential misclassification patterns. To visualize the results of the grid search, we can plot a heatmap of the macro F1 score.

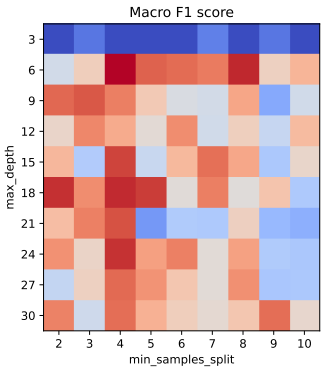
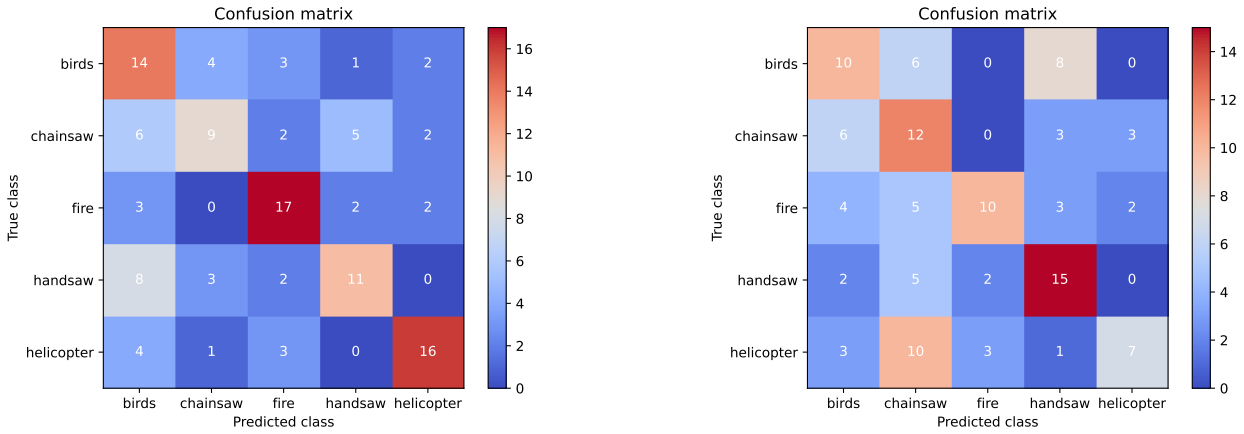



Figure 2: Heatmap of the macro F1 score

The first row of the heat map shows a very low macro F1 score, likely because a maximum depth of 3 is too low for the model to learn patterns from the data, leading to underfitting. A similar effect is observed when the minimum samples split reaches a high value, although the impact is less pronounced since maximum depth is the hyperparameter with the most significant influence on the results. Overfitting can be observed in the column where the minimum samples split is the lowest and the row where the maximum depth is the highest.

A maximum depth of 6 and a minimum samples split of 4 appear to provide the best trade-off between bias and variance. This combination balances learning capacity with regularization effectively.

The best model achieves a mean macro F1 score of 0.57 during cross-validation and 0.56 on the test set. It is important to note that the test set cannot be used to tune hyperparameters, as its purpose is specifically to evaluate the model's performance on new data. To gain a better understanding of the model's performance on the test set, we examine the confusion matrix shown in Fig. 3a.




(a) Confusion matrix, default test set

(b) Confusion matrix, transformed test set

Figure 3: Confusion matrices on the test set with/without data augmentation

The model appears to struggle with predicting the chainsaw and handsaw, while performing well with the birds, fire, and helicopter. The highest off-diagonal value in the confusion matrix is 8, where the model misclassifies the

handsaw as birds. To better understand this behavior, we can examine the upper part of the decision tree (see Fig. 22 in the Appendix). It becomes apparent that the first few internal nodes at the right side of the tree primarily focuses on the high-frequency content of a sample, as the frequency is determined by its index modulo twenty, given that the mel-spectrogram is a 20x20 matrix reshaped into 400 arrays. The decision tree tends to group chainsaw and birds together due to their shared high-frequency content, which might explain some of the misclassifications. However, this approach is not ideal, as chainsaw also contains notable low-frequency content. It would be better for the model to account for this as well. This behavior indicates that the model might be overfitting to specific patterns in the data or that its simplicity limits its effectiveness for this task. Despite these challenges, the overall performance, as reflected in the confusion matrix, remains acceptable.

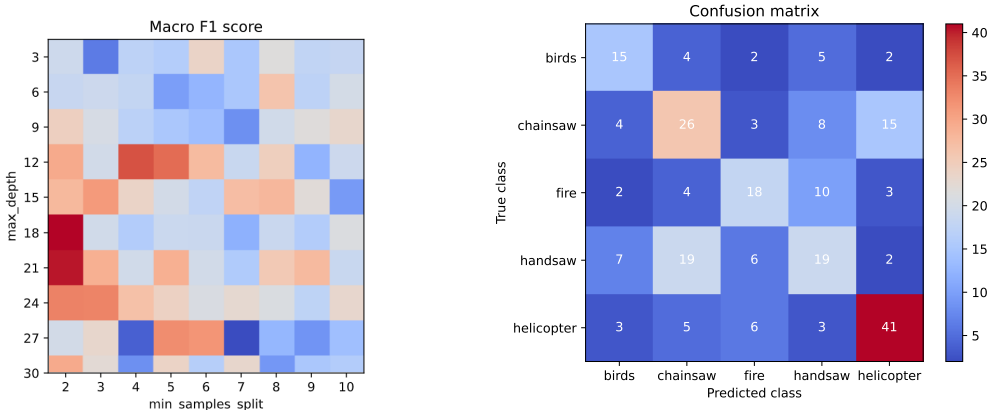
### 3.1.2 Robustness of the model in simulation

To evaluate the robustness of the model, we modified the test set by applying a transformation not included in the training data. Specifically, we introduced an additive white Gaussian noise (AWGN) transformation to the audio files. This transformation is particularly relevant, as AWGN is common and typically present in real-world conditions. The resulting confusion matrix is shown in Fig. 3b. We observe that the model's performance decreases when predicting classes with sparse mel-spectrograms, such as birds and fire. However, it performs better with classes that have denser mel-spectrograms, like chainsaw and handsaw, since AWGN adds power to every frequency, making the matrix less sparse. Helicopter is often misclassified, likely because its mel-spectrogram is similar to that of chainsaw, and the added noise causes some helicopter samples to resemble chainsaw samples more closely.

### 3.1.3 Classification using the whole chain

Now that we have evaluated the performance of our model in a simulation setup, we can move on to the next step: evaluating the model using the full chain of our system. We still have 40 audio files for each of the 5 classes, but since the MCU can only compute a spectrogram for one second of audio at a time, and each audio file lasts for 5 seconds, we end up with 1000 feature vectors. However, some of these feature vectors must be removed, as not every second of every audio file contains something meaningful to classify. For example, birds may stop singing for a second, and we don't want to give the model a feature vector with no sound and incorrectly label it as a bird. A simple way to address this is by removing feature vectors with a norm that is too low, as the norm represents the intensity of the sound. A low norm indicates either no sound or sound too faint to be classified properly.

We follow a similar approach to the one used in the simulation setup, but this time, data augmentation is not necessary since the feature vectors already include a time shift due to the audio being split into 5 spectrograms. We remove feature vectors with a norm lower than 47, a threshold determined by testing different values until we find one that removes approximately half of the bird samples. The birds class is the least consistent, as their audio files often contain periods of silence when the birds stop singing. After filtering, we split the dataset into a training set and a test set using a 70/30 ratio, ensuring that all classes are evenly represented in both subsets. We then normalize both subsets. Finally, we perform a grid search to tune the hyperparameters described in Tab. 1, and the resulting heat map is presented in Fig. 4a.




(a) Heat map of macro F1 score

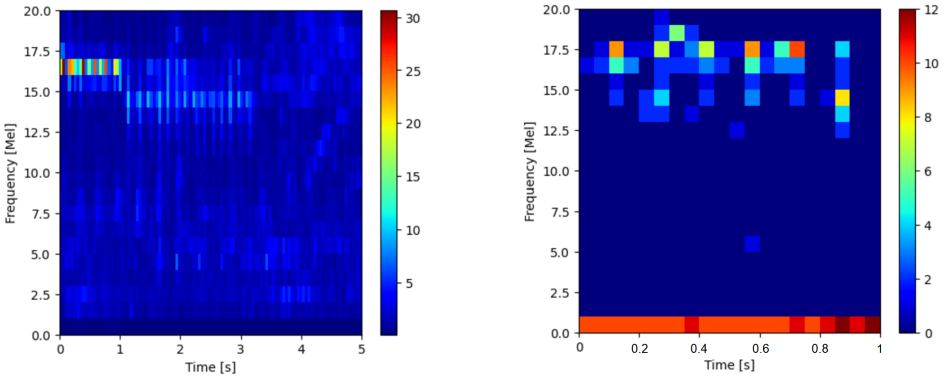
(b) Confusion matrix of the test set

Figure 4: Results of the models grid search and performance on the test set using the full chain

As we can observe, it seems that a lower value for min samples split and a higher value for max depth result in

better performance. These hyperparameter values are associated with increased variance for the model (increasing max depth leads to more variance, and similarly, decreasing min samples split has the same effect). This makes sense because the model has to fit 1-second spectrograms instead of 5-second ones. With shorter spectrograms, there is less temporal information available, leading the model to focus on more localized features, which often requires a higher model complexity to capture the finer variations in the data. A spectrogram of 1 second contains more rapid changes, which might make the model need to learn finer distinctions, and therefore, a more complex model can better capture those variations.

The best model achieves a mean macro F1 score of 0.54 during cross-validation and 0.50 on the test set. Performance using the full chain is lower than that observed in the pure simulation, primarily due to the non-idealities inherent in the full system. To understand the confusion matrix presented in Fig. 4b, it is interesting to consider the non-idealities that impact the acoustic signal acquisition. Several factors contribute to these distortions. First, the microphone's transfer function acts as a bandpass filter, with a maximum at 10kHz (see Fig. 23). Second, quantization errors are introduced by the 12-bit ADC. In addition, other factors, such as thermal noise from the components and environmental noise from the room where the sound was played, further contribute to the overall distortion. Moreover, the embedded execution faces its own set of non-idealities. Matrix computations are performed using fixed-point representation, which introduces a finite resolution. Furthermore, the use of SIMD (Single Instruction, Multiple Data) instructions, which break the 32-bit values into 4-byte segments to perform computations on these bytes, reduces overall precision. To visualize the overall impact of these non-idealities, we can compare a spectrogram from the simulation a spectrogram of the same sound acquired through the microphone (see Fig. 5).




(a) Spectrogram of birds sound (Simulation)

(b) Spectrogram of birds sound (Microphone)

Figure 5: Comparison of spectrogram from obtained from simulation and microphone

In the simulation (Fig. 5a), the signal is clean, with a narrow frequency band concentrated around 15-17.5 kHz, and minimal noise. By contrast, the microphone-acquired spectrogram (Fig. 5b) shows significant distortions: the frequency range is broader and noise is dominant at lower frequencies (<2.5 KHz). These effects arise from the non-idealites discussed earlier, degrading the signal quality and contributing to the lower performance of the classification.

Given these distortions, we can analyze the confusion matrix to interpret the classification results. Classes such as helicopter and chainsaw are easier to predict, as the distortions—namely, the sparsity of the spectrogram and the presence of low-frequency noise—align with typical characteristics of these classes. For example, helicopters produce consistent, low-frequency components, while chainsaws generate periodic bursts of sound energy with strong low-frequency elements, both of which are more robust against the introduced noise and distortions.

On the other hand, classes such as birds and fire are more challenging to predict. Bird sounds are characterized by high-frequency, less dense spectrograms, which are easily masked or degraded by noise and frequency smearing. Similarly, fire sounds have less distinct time-frequency structures, making them harder to differentiate from noisy environments. These factors lead to increased misclassification for these classes in the confusion matrix. It is also important to keep in mind that some samples, particularly from the birds and fire classes, were removed because their norms were not high enough. However, it is likely that some of these discarded samples contained valuable information that could have improved classification performance.

The performance gap between simulation and real-world setups can be attributed to various non-idealities in the system chain that distort the feature vectors. In contrast, simulations are noise-free and tend to yield better performance. By incorporating real-world distortions into the training process of the simulation setup, we can reduce this gap and improve this model's robustness.

In summary, the performance of the model using the full chain demonstrates the challenges introduced by real-world distortions. While classes with more distinct and robust acoustic signatures, such as helicopters and chainsaws, maintain relatively higher accuracy, subtle and sparse spectrograms like those of birds and fire are more vulnerable to these distortions, leading to increased misclassification. The removal of low-norm feature vectors further impacts these classes, potentially discarding valuable data. These findings underscore the importance of addressing the non-idealities in the signal acquisition chain and optimizing feature extraction to improve classification performance in practical scenarios.

### 3.1.4 Possible improvements

There are several ways to improve the performance of our classification system. First, we could explore using more complex models, such as random forests (which combine multiple decision trees) or support vector classifiers (SVCs), which is well-suited for high-dimensional data. Additionally, we could incorporate the memory effect, which involves using a sequence of slightly delayed mel spectrograms to predict class probabilities and then combining these probabilities into a single, more robust decision. Expanding the dataset size is another critical improvement, as 40 audio files per class are relatively limited and may hinder the model's ability to generalize. Finally, data augmentation could be further leveraged by experimenting with new, meaningful transformations to artificially enhance the diversity of the dataset.

## 3.2 Telecommunication

### 3.2.1 Telecommunication setup

We investigate the performance of the telecommunication component of our project, which includes everything from the data packetization of feature vectors on the transmission side to the retrieval of these packets on the reception side.

To evaluate the communication chain, we tested the S2LP radio module (integrated with the MCU for transmission) and the LimeSDR module (used for reception). Below are the fixed parameters used throughout all measurements:

Table 2: Parameters of radio evaluation

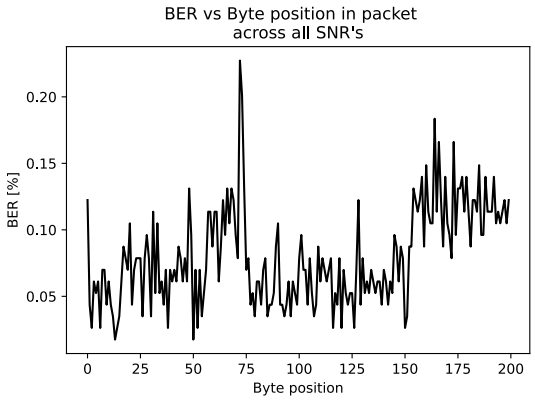
| **Parameter** | **Value** |
|---|---|
| Carrier Frequency | 868 [MHz] |
| Datarate (inside a packet) | 50 [kbit/s] |
| Preamble | 0xAAAAAAAA |
| Sync word | 0x3E2A54B7 |
| Payload length | 200 [bytes] / 1600 [bits] |
| Delay between packets | 20 [ms] |
| Packet rate | 40 [packets/s] |
| Number of packets sent | 200 |
| Noise power at reception | -62 [dBW] |
| RX gain | 70 [dB] |
| Canal attenuation | 70 [dB] |

The sync word is known by both the receiver and the transmitter and will be used to make a frame estimation at reception side.

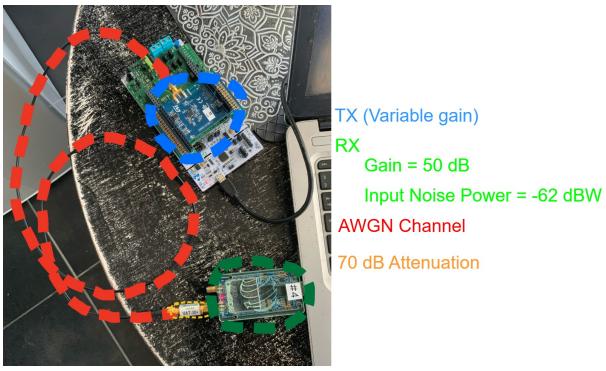
The preamble serves several purposes: it enables carrier frequency offset (CFO) estimation, symbol timing offset (STO) estimation, and, most importantly, packet detection at the receiver.

To ensure reliable communication, we limited the payload to 200 bytes per packet, equivalent to half a melspectrogram. This size was chosen to minimize the propagation of CFO estimation errors across symbols, which would otherwise increase the bit error rate (BER).

The BER as a function of byte position in the packet was used as an indicator of the ideal payload length for a given CFO estimation's quality. Ideally, the BER remains constant across the packet; however, a poor CFO estimation typically results in higher BER values toward the end of the packet. Figure 6a confirms that 200 bytes is a suitable payload size for the chosen parameters.




(a) BER does not increase wrt byte position. We can assume that the estimated parameters hold for whole packet




(b) Setup overview for the telecommunication

Figure 6: Comparison of BER and telecommunication setup.

The transmitter sends packets at various power levels, resulting in different input SNRs at the receiver. We assume an AWGN channel without multipath effects. Additionally, the receiver only demodulates signals surpassing a predefined threshold, ensuring noise does not trigger false detections.

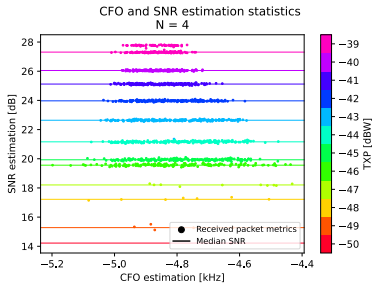
### 3.2.2 CFO Metrics

As previously noted, the carrier frequency offset (CFO) estimation is a critical parameter. Transmission and reception clocks are not perfectly synchronized, necessitating CFO correction. The CFO depends on temperature and power usage variations within the chip, requiring recalibration for each transmitted packet.
The Moose algorithm was employed for CFO estimation. Its parameter $N$ determines the number of samples used, and the following constraints must hold:

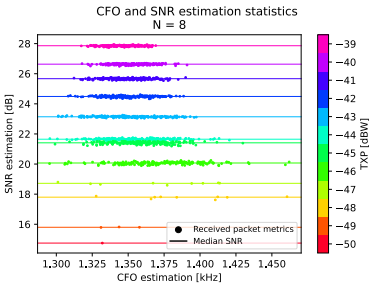
$$|\Delta f_c| \leq \frac{B}{2N} \tag{1}$$

$$N \leq \frac{\#\text{ bits in preamble}}{2} \tag{2}$$

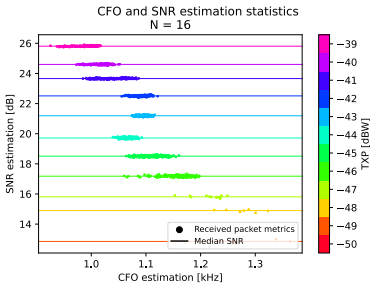
Figure 7 and Figure 8 show the impact of $N$ CFO estimation:




(a) BER ≈ 50% for all TXP




(b) Correct BER




(c) Correct BER

Figure 7: 3 measurements with similar setups, temperatures, etc...

The mean CFO evolves across measurements due to clock sensitivity. While $N = 8$ and $N = 16$ provide similar results, $N = 4$ leads to incorrect CFO values. A larger $N$ reduces variance without significantly increasing computational load. Thus, $N = 16$ was chosen for subsequent measurements, ensuring the range of $\pm 1.5kHz$ around the carrier frequency. Equation 1 is satisfied:

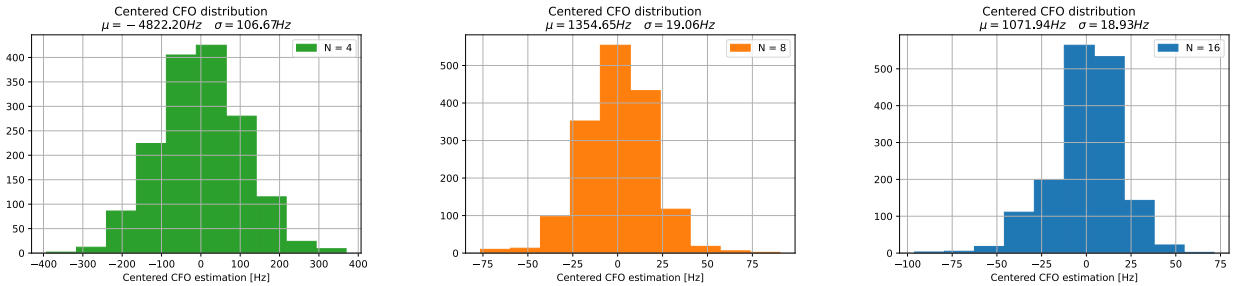
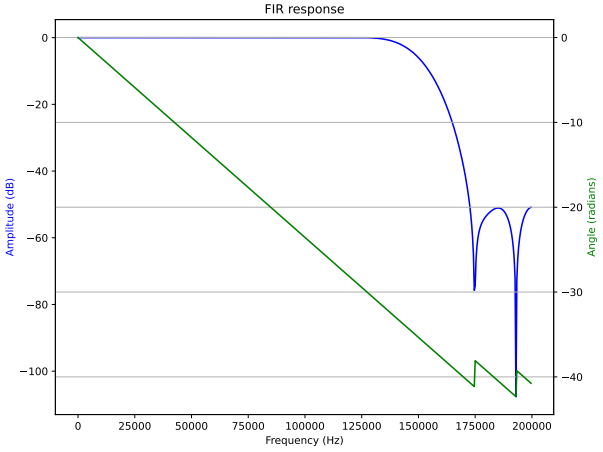



Figure 8: Distribution of the estimated CFOs
Each value was centered compared to the mean CFO estimation at the corresponding TXP (not the mean CFO across all powers that is called $\mu$)

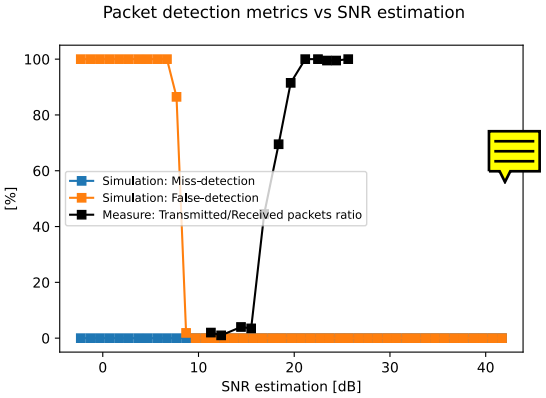
$$|\Delta f_c| \leq \frac{B}{2N} = 1.5625kHz \tag{3}$$

### 3.2.3 Simulation and measurements

In this section, we analyze the results of the measurements and simulations. First, we ensured that the simulation used a low-pass filter (LPF) matching the one at the input of our receiver with Figure 9a.




(a) FIR response used in simulation
$f_s = 400kHz$, $f_{cutoff} = 150kHz$, numtaps $= 31$




(b) Detection metrics. Note that the threshold of detection was set such that noise only can't be detected by the preamble detection block.

Figure 10 shows the evolution of the bit error rate (BER) and packet error rate (PER) as functions of the signal-to-noise ratio (SNR).

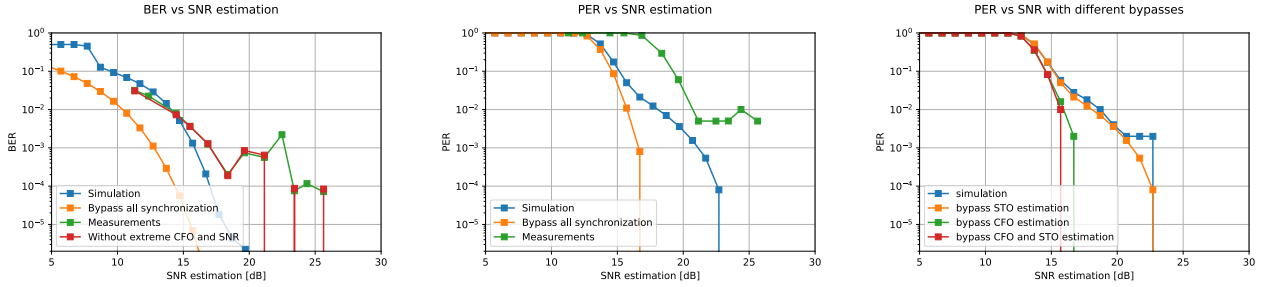
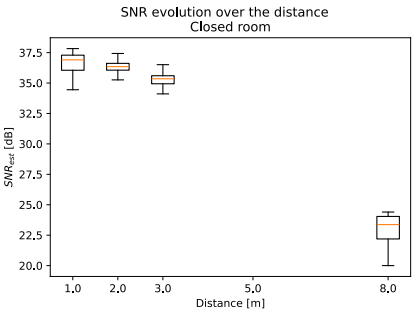



Figure 10: Evolution of BER and PER with respect to SNR

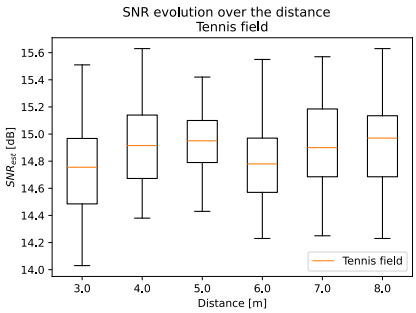
The curves in Figure 10 provide key insights.
The first two graphs compare two simulation models. The first curve represents a simulation that replicates the measurement setup as accurately as possible, incorporating all the synchronization and detection blocks described earlier. The second curve assumes ideal conditions: perfect synchronization between the transmitter and receiver, and the assumption that only transmitted packets are received and demodulated.

As illustrated in the third graph, the main bottleneck lies in the carrier frequency offset (CFO) estima As noted earlier, even minor inaccuracies in CFO estimation propagate throughout the packet, significantly deteriorating the PER.
The measurements align closely with the simulations, particularly for the BER curve, demonstrating the robustness of the simulation model.

### 3.2.4 Over the air transmission




(a) Measures performed at RX gain = $50dB$ and TX power = $0dBm$




(b) Measures performed at RX gain = $66dB$ and TX power = $0dBm$

For the first over-the-air measurement, we conducted an SNR test in a closed environment (the Marconi lab). While the results at a distance of 8 meters showed a decrease in SNR, the environment's reflections and multipath effects introduced significant variability, making it difficult to obtain precise information.

When we repeated the test in an open-air environment, specifically on a tennis court, the situation worsened. The barriers surrounding the court acted as a Faraday cage, reflecting signals and preventing accurate packet retrieval. Despite these challenges, the results confirmed that:

- The antennas function as expected.
- Packets are successfully modulated and demodulated during over-the-air transmission.

## 3.3 Power consumption

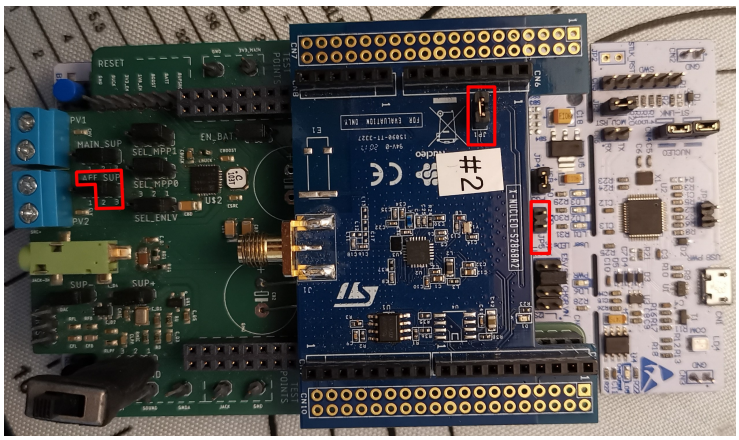
### 3.3.1 Measurement setup

Our power measurement setup consists of using a shunt resistor to evaluate the power flow of the circuit, following the diagram in Figure 12b. We chose 100Ω (100.54Ω to be exact), as it would allow us to have a big enough voltage drop, whilst still staying well bellow the maximum operating voltages. From the datasheet of the Nucleo-STM32L4A6ZG devboard, we can see that the minimum and maximum continuous operation voltages are respectively 1.71V to 3.6V. Since our system uses the typical 3.3 volts, the maximum power we can sense, will be $P_{measure,max} = \frac{(3.3[V])^2}{100[\Omega]} = 99[mW]$. Since we can expect a maximum power consumption of less than 40mW, the maximum voltage drop would then be $\Delta V_{shunt} = \frac{40[mW]*100[\Omega]}{3.3[V]} = 1.2[V]$. This voltage drop would lead to a voltage range of 2.1V to 3.3V, above the minimum voltage required to power the MCU. But, since the regulators have losses outside of their normal operating range, if we do drop to 2.1V, then, non linearity may play a bigger role, mostly at higher clock frequencies (more consumption). Practically, the highest power we have seen, is about 20 mW peak, so the voltage drop would be about 0.6V which is still reasonable.

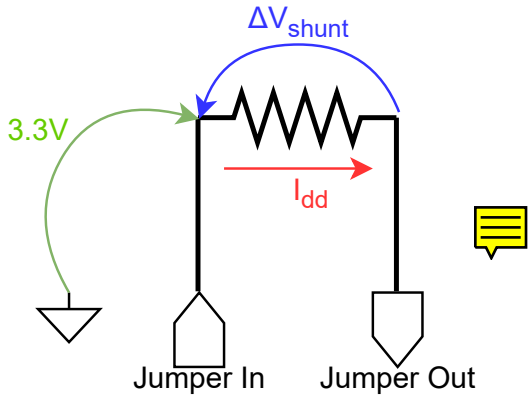
Since we want to measure the MCU's, the radio's and the AFE audio board's consumptions, we have to correctly place the shunt resistor to get the right values in the right places. Figure 12a shows all jumper connections to use for power measurements.

Since the oscilloscope[2] offers two different resolutions : 2.5 mV (1 V per division) and 0.25 mV (100 mV per division); we had to reduce the maximum voltage drop to ensure it remained within 10 divisions, making it measurable by the device. To achieve this, we used the setup shown in Figure 12b, which allowed us to preserve the maximum possible precision. The oscilloscope settings were not changed for any of the subsequent measurements; it was configured to 2.5 kS/s with a buffer depth of 10 kS. Additionally, some signals (duty cycles) were truncated to better highlight specific events and facilitate energy calculations.

[2]The oscilloscope used for these measurements is the DOS1102 from Hanmatek




(a) Jumper positions for the 3 hardware measurements.



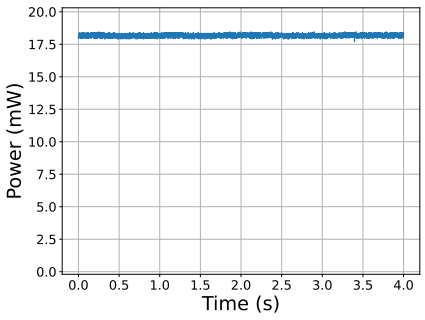

(b) Oscilloscope setup for power measurement. Both voltages are measured at the same time, as the supply could vary.

Figure 12: Hardware setup for power measurements.

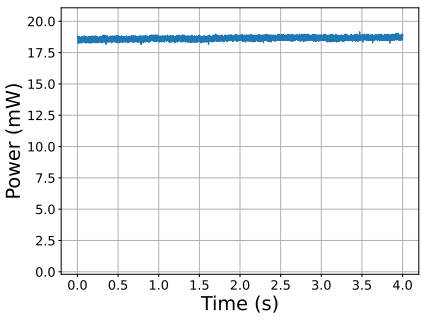
### 3.3.2 Power consumption MCU

Here, we performed numerous measurements under various conditions and calculated the energy consumption for cases that displayed duty cycles of the board. To achieve this, we shunted the JP5 jumper to measure power consumption. Our results are presented in three 6×3 graph clusters, organized this way for both formatting clarity and to reflect minor changes in measurement techniques over time.

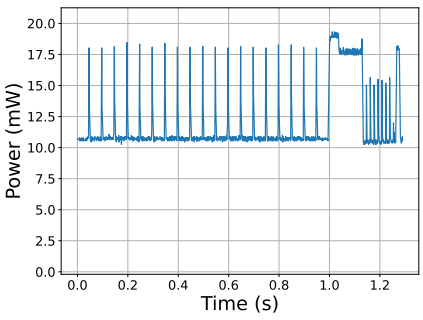
For Parts 1 and 2, the measurements were conducted with DEBUG enabled, UART disabled, all LED events turned off (except for the green power-on LED and the STLink RGB LED, which remained on for unknown reasons), and all cycle counts disabled. In Part 3, the setup remained the same, except DEBUG was disabled. For all parts, transmission power was set to 0 dB (when transmitting), and the STM32's core frequency was set to 48 MHz (This does drive the power consumption up slightly).



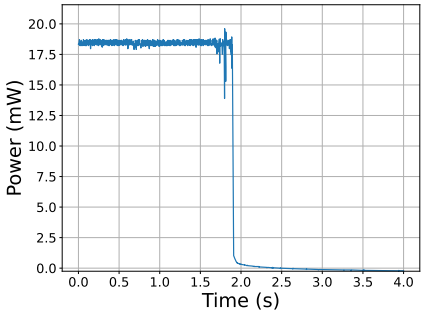

(a) Static power consumption through the 5V ST-Link supply



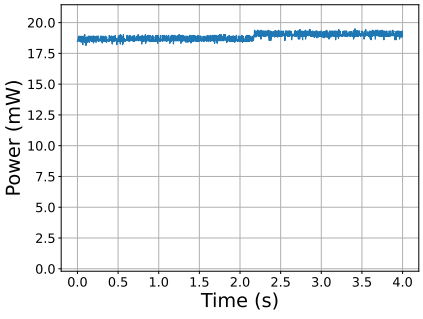

(b) Static power consumption through the external 3.3V supply pins



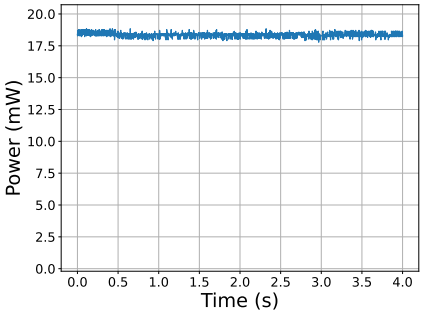

(c) Full acquisition and transmission of the main app (38.03 mJ)




(d) Integrated capacitor discharge (about 0.28 mJ of energy stored)

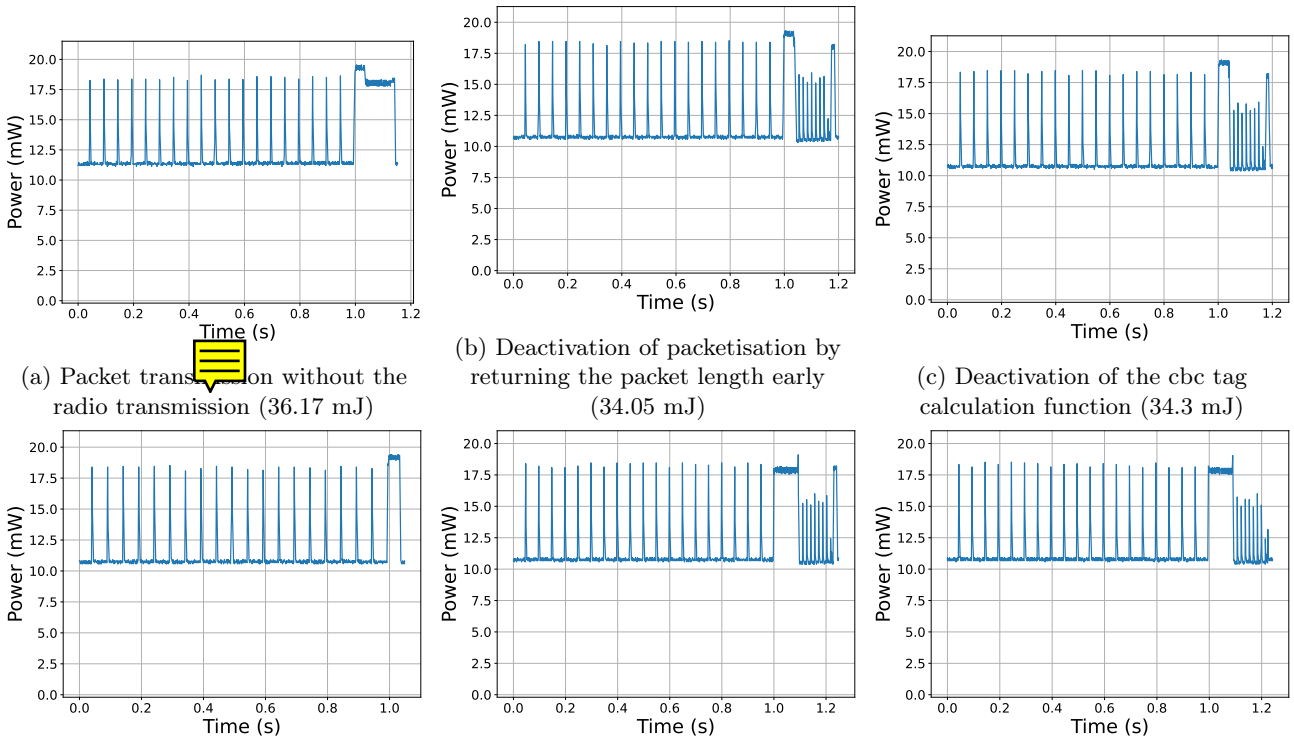



(e) Removal of the Radio (JP1) jumper at 2.2s




(f) Removal of the AFE_SUP 3-2 jumper at 1.6s

Figure 13: Power measurement graphs (Part 1)




(a) Packet transmission without the radio transmission (36.17 mJ)

(b) Deactivation of packetisation by returning the packet length early (34.05 mJ)

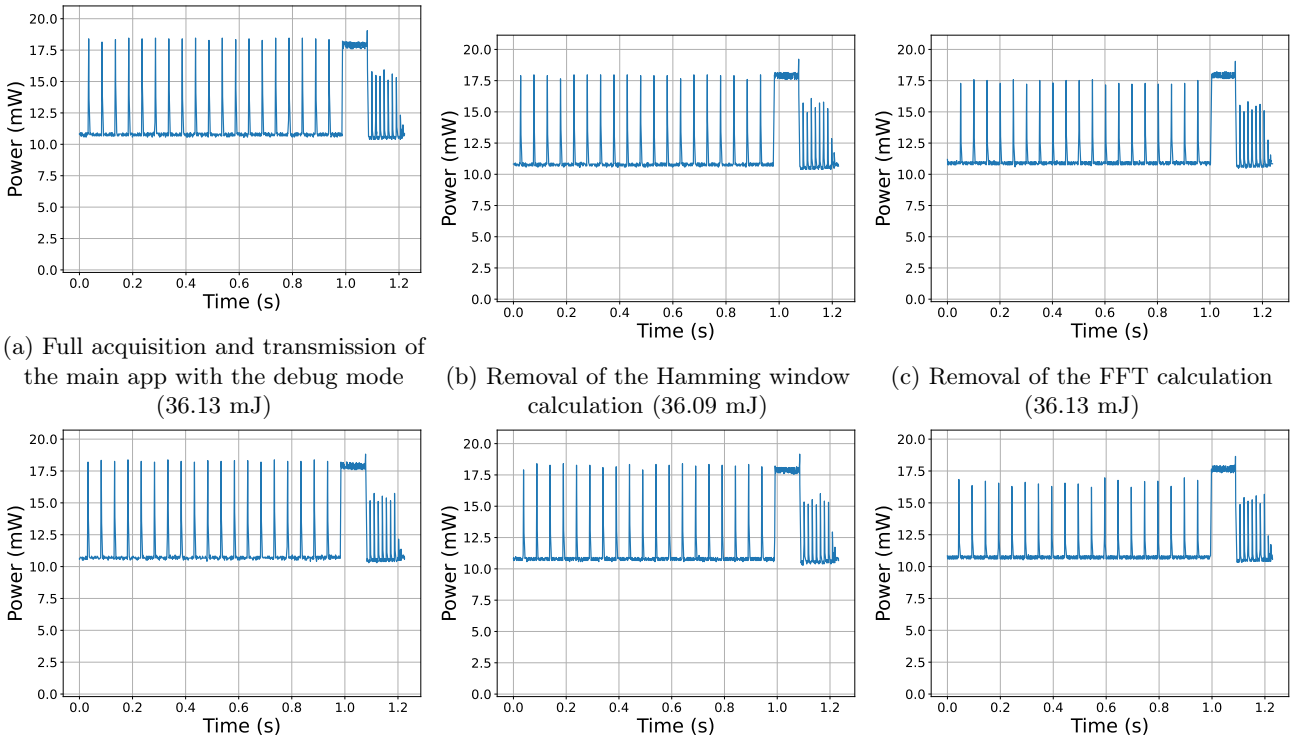
(c) Deactivation of the cbc tag calculation function (34.3 mJ)

(d) In debug mode, without uart, only the print_spectrogram() (29.87 mJ)

(e) Only the send_packet() function in debug mode (36.59 mJ)

(f) In debug mode, without uart, removal of all debug prints (35.7 mJ)

Figure 14: Power measurement graphs (Part 2)




(a) Full acquisition and transmission of the main app with the debug mode (36.13 mJ)

(b) Removal of the Hamming window calculation (36.09 mJ)

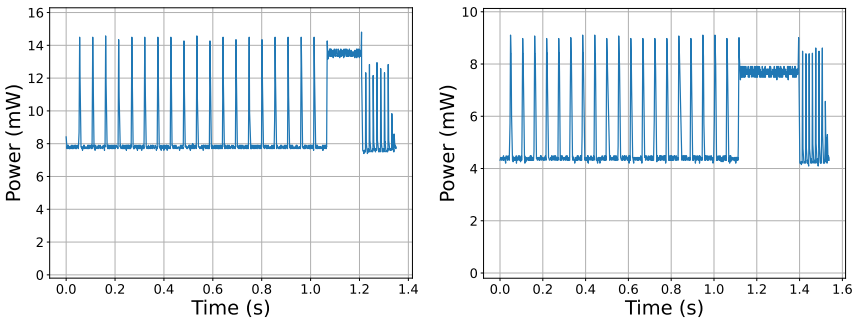
(c) Removal of the FFT calculation (36.13 mJ)

(d) Removal of the complex magnitude (35.67 mJ)

(e) Removal of the matrix multiplication (35.97 mJ)

(f) Removal of the matrix multiplication and FFT(35.19 mJ)

Figure 15: Power measurement graphs (Part 3)




(a) Change from 48MHz to 32MHz (29.51 mJ)

(b) Change from 48MHz to 16MHz (20.62 mJ)

Figure 16: Power measurement graphs for different frequencies

The first notable thing we can see from all these graphs, is the power idle power consumption from running the processor at 48 MHz constantly, which reaches the 10.5 mW of power passively. This means that by dynamically reducing the clock frequency we could potentially already reduce our power consumption substantially, by tens of mJ.

Another notable thing to see, is the power consumption of the debug prints. They consume much more than the computation of the CBC mac (in power), whilst doing nothing since UART is turned off. This means that even though we have turned off the UART, something else in or out of the MCU is turning on and consuming a whole lot of power.

Analyzing the packet acquisitions, we can sometimes see a peak of power just after the transmission of the packet. This peak is not always present, and probably corresponds to the first mel-spectrogram calculation. Which would align quite well with the fact that the data had been sent to the S2LP, then resumed normal execution, and had enough time to complete a full DMA acquisition, within the clock cycle. This is then a possible involuntary speedup.

Before analyzing the data acquisition part using the DMA, it is important to note that because of our 2.5 kS/s sampling rate of the oscilloscope, the peaks are slightly too fast (<10.2 kHz), and so a oscillating pattern appears in the power peaks. Through Figure 15f, we can see that after removing both the matrix multiplication and FFT, the energy consumption peaks have lowered slightly. This does show that these operations are heavy, but since the processor still has to run at full power in those moments, the peaks are still very visible, as logic inside the processor turns on.

The data in Figure 16b demonstrates that reducing the frequency significantly lowers the power consumption of the MCU. We further attempted to decrease the frequency to 2 MHz and 100 kHz, but the power consumption remained constant. Interestingly, during 4 seconds of recording at these frequencies, the power consumption stayed at 2.6 mW, resulting in a total energy consumption of 26.2 mJ.

As for the audio sampling frequencies (or mel samples), since the relation is linear with the energy, we did not plot them.

### 3.3.3 Power consumption radio (S2LP)

From Figure 13e, we can determine that the amount of power consumed by the S2LP on standby, is about 2 mW or so. Here, as we transmitted at 0 dBm, we are sure to not consume much less than at least 10mW.

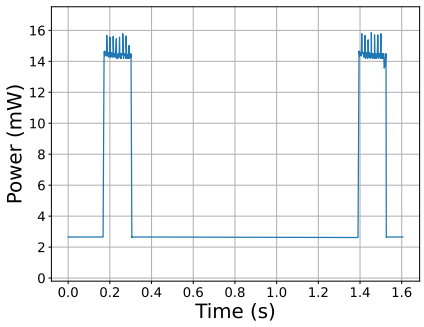



Figure 17: End and start of a transmission cycle, with passive consumption and active consumption

We can see in Figure 17 that the passive power consumption is about 3 mW a bit higher than our first estimates. And in accordance wotj our first transmission estimates, when the transceiver is turned on, we can expect a power draw of about 14.3 mW at 0 dBm, and about 15.9 mW for transmission bursts. Here, the smaller peaks are probably due to memory limitations on the S2LP module itself. The passive energy consumed during inactivity between transmissions was 7.13 mJ, while the energy used for the transmission, was 4.87 mJ. This means that the majority of the energy we are consuming is coming from the idle power of the transceiver.

### 3.3.4 Power consumption AFE

To measure the AFE supply, we had to jump the AFE_SUP pins 3 and 2 with the shunt resistor (Technically, we could of have used the SUP+ jumper, but since in the future, we will have to change the AFE_SUP to pins 1 and 2, we decided to go with it). As only the microphone, voltage regulator, filters and amplifier are powered from here, we can expect a very constant power supply, as they don't really consume as much, and don't spike in energy consumption.

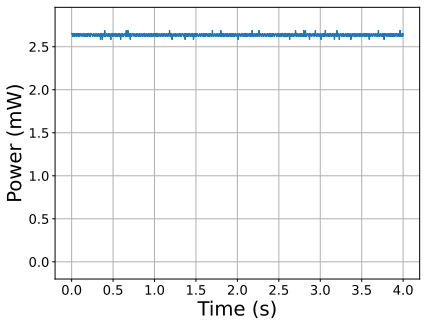



Figure 18: Power used by the AFE during 4 seconds

The energy consumed during a 1 second acquisition was 6.6 mJ, and its mean power was 2.6 mW, which is quite high. We can also see quite a bit of noise, this may be due to the amplification of the signal, sometimes peaking, and needing more power. This test was done in a room with only one person walking on a wooden floor, and only 1 fan running in the background.

### 3.3.5 Energy consumption

From looking at the e-peas AEM10941 datasheet, the board shematic and the components on the board, we can see that there should be 2 electrolytic capacitors serving as batteries for the AEM, 1mF and 2.2mF each. But only the 1 mF is soldered on, with a voltage rating of 6.3 V. From the energy formula for capacitors ($E = \frac{1}{2}CV_{BAT}^2$), and the maximum rating of the battery controller of the AEM being 4.5 V. We can calculate that the 1mF capacitor would store 10.13 mJ of energy, and the 2.2 mF, 22.27 mJ. With both charged at full capacity, we could power a full acquisition at our energy consumption without too much trouble. We also have a big advantage here, as the AEM allows for capacitors to go down to 0 V. Something impossible with lithium or sodium batteries.

Something weird about our setup, is observable from Figure 13d, as the energy stored is much smaller than our calculations done above. This means, that even if we did enable the use of the battery (EN_BATT jumper), something is not routing it to the capacitor properly or the AEM is not being used (most probable explanation, as

the MAIN_SUP header is set to 3-2: 3V3_USB and not 1-2: 3V3_EH).

From what we could improve for energy preservation and consumption, are the following :

- Turn off things that are not needed from the S2LP and MCU
- Dynamically reduce the frequency of the processor (Figure 16b), and trigger DMA acquisition after the copy of the mel-vector into the packet, since the tag is independent of the buffer of the mel-vector buffer (To reduce delay between acquisitions). This means that as long as we are computing the tag within a window of 1s, we could get the data automatically in the background. But interrupts are something to be worried about.
- Try to disconnect the LED's of the ST-Link and PWD indicator (they consume quite a bit in the end).

More optimizations will be discussed in the next sections.

## 3.4 Sensing duty cycle

### 3.4.1 Cortex-M4-L4A6ZG Characterization

The board we are using is the Nucleo-STM32L4A6ZG, which features the STM32L4A6ZG microprocessor. Its key specifications for our current and future use cases are listed below:

- **80 MHz Arm 32-bit Cortex-M4 PCU** with integrated FPU and real-time adaptive accelerator, based on the ARMv7-M instruction set.[3]
- **16 timers** of various types.
- **Real-Time Clock (RTC)** with hardware calendar, alarms, and calibration.
- **Dedicated Chrom-ART Accelerator** for graphic content creation (DMA2D).[4]
  - We are aware that this component probably consumes a lot of energy, and is very limited in its use. As it can only draw, animate and process basic primitives. This component could have been a lot more useful to speed up processes if we could run compute shaders on it as a real GPU, and not a primitive graphics accelerator.
- **AES-128/256 and SHA-256** encryption hardware accelerator.
- **Memory:**
  - 1 MB Flash with 2 read-while-write banks and proprietary readout protection.
  - 320 KB SRAM, with 64 KB hardware parity check.
- **Clocks:**
  - 4 to 48 MHz variable integrated crystal oscillator.
  - Internal 48 MHz clock with clock recovery.
  - 32 kHz clock for the RTC.
  - 3 PLLs for clock, USB, audio, and ADC.
- **14-channel DMA controller** on the Multi-AHB bus matrix.
- **True Random Number Generator (TRNG).**
- **CRC calculation unit** with 96-bit unique ID.
- **Ultra-Low Power Mode:**
  - Supply voltage rating (V_DD): 1.71 V to 3.6 V.
  - Backup operating voltage rating (V_BAT): 1.55 V to 3.6 V.
  - 108 µA/MHz for Run (100 kHz -> 80 MHz, 0.06 mA -> 9.44 mA @ 25°C)

[3] Arm Developer: Cortex-M4 Instructions
[4] DMA2D Presentation for the STM32F7 by STMicroelectronics

- 129 µA/MHz for LPRun (100 kHz -> 2 MHz, 63 µA -> 274 µA @ 25°C)
- 32 µA/MHz for Sleep (100 kHz -> 80 MHz, 0.06 mA -> 2.57 mA @ 25°C)
- 51 µA/MHz for LPSleep (100 kHz -> 2 MHz, 53.7 µA -> 103 µA @ 25°C)
- 428 nA Standby mode with RTC (for DMA).
- 108 nA Standby mode (5 wakeup pins).
- 37 µA/MHz run mode (at 3.3 V in SMPS[5] mode).
- 91 µA/MHz run mode (LDO[6] mode).
- Batch Acquisition Mode (BAM).

Our system operates with a 48 MHz main clock and an audio sampling frequency of 10.2 kHz, managed using an internal counter. To achieve a target audio acquisition time of 1 second, we process and transmit packets containing 20 mel-spectrograms, represented as a single mel-vector. This is facilitated by two DMA controllers, ensuring continuous data acquisition and transmission. As our continuous acquisition relies on turning on the ADC acquisition BAM, when processing interrupts, like packetisation and sending the data to the S2LP, we will have a computational delay. In practice, from Figure 15a, we can see that the post-processing time is about 0.21 seconds. This means that our true duty cycle here is actually 1.21 seconds, and not the 1 second we actually wanted. Which could cause additional problems with classification if not accounted for.

Each DMA buffer handles 512 samples, computes their respective spectrograms, and transfers them to the mel-vector. Once 20 spectrograms are acquired, the complete packet is prepared for transmission and sent via the S2LP module[7]. Prior to transmission, the data is encapsulated into a packet and a CBC MAC tag is appended for identity and integrity verification. The majority of the energy consumption occurs during this phase, whereas the computation of individual mel-vectors is negligible, only showing up in spikes, with possibility of reducing peak consumption with optimization techniques, and dynamic clock adjustments. From Figures 14, we can see that the operation that actually consumes the most energy in the end, is the computation of the CBC MAC tag. This means that if we offload it to the dedicated accelerator, we could gain a substantial amount of energy back.

### 3.4.2 Memory utilization

| Region | Start addre | End addres | Size | Free | Used | Usage (%) |
|---|---|---|---|---|---|---|
| RAM | 0x20000... | 0x2004ffff | 320 KB | 299 KB | 21 KB | 6.56% |
| FLASH | 0x08000... | 0x080fffff | 1024 KB | 883.39 KB | 140.61 KB | 13.73% |

Figure 19: Build Analyzer output

Through the screenshot of the build analyzer in Figure 19, we can see that we are very far from fully utilizing the memory of the MCU. This means that we could easily expand freely our program without memory struggles, and if need be, we could increase the number of samples by 10 times, this means that we could have mel-vectors that represent 10 entire seconds without a hitch. The only problem there would be the amount of processing (and energy) needed. This will be discussed in more detail in the next section.

This means that the best compiler optimization level would range from `-O1` to `-O3`, depending on the future codebase. With these settings, we would be optimizing for speed at the cost of memory footprint, which is not an issue in this context.

[5] Switched-Mode Power Supply
[6] Low Drop Out power regulator, instead of the SMPS
[7] Nucleo shield for a 868 MHz RF transceiver

### 3.4.3 Processing and energy

Table 3: Computational complexity of the system (512 samples)

| Element | Mean | Std Dev | Min | Max |
|---|---|---|---|---|
| Print FV | 13,609,812.44 | 53,012.57 | 13,558,655 | 13,718,503 |
| Encode Packet | 4,371,878.75 | 15,200.17 | 4,356,264 | 4,387,543 |
| Send Packet | 6,582,632.62 | 1,351.53 | 6,579,560 | 6,583,886 |
| Spectrogram Shift (init) | 3,692.29 | 24.17 | 3,677 | 3,731 |
| Remove DC Component (init) | 14,900.43 | 19.83 | 14,892 | 14,949 |
| Spectrogram Multiplication | 3,951.86 | 20.48 | 3,943 | 4,002 |
| FFT Init | 86.00 | 0.00 | 86 | 86 |
| FFT | 23,969.00 | 30.05 | 23,941 | 24,005 |
| Absmax Spec | 16,829.86 | 1,011.58 | 15,216 | 17,722 |
| Complex Magnitude | 6,072.50 | 2,243.36 | 4,590 | 11,460 |
| De-normalization | 6,460.62 | 24.76 | 6,446 | 6,504 |
| Fast Matrix Multiplication | 20,917.29 | 27.76 | 20,883 | 20,945 |
| Packet Memset | 237.00 | 0.00 | 237 | 237 |
| CBC MAC Calculation | 4,356,681.25 | 282.15 | 4,356,264 | 4,357,060 |

In accordance to the power measurement part, we can see that debug prints and UART in general consumes a immense amount of energy. We can also now explain why the packetization was consuming so much energy. We can now compare it with the results given by the static stack analyzer (that we haven't shown because of space constraints on the document). From the table above, and the static stack analyzer, we can see that the heaviest function calls, are :

- **print_encoded_packet()** : Many loops with many writes to the I/O
- **send_spectrogram()** : Has to call all the heavy functions to communicate with the S2LP (and possibly print the mel_vector)
- **AES128_encrypt()** : Heavy algorithm in general, unload-able to the accelerator
- **ComputeFreqDeviation()** : Computationally heavy
- **Spectrogram_Compute()** : Many data points to compute, and all heavy operations to make the mel-spectrogram
- **tag_cbc_mac()** : Many times the AES128_encrypt() function

These functions, being the reason for our problems, will require if possible, significant speedups. The main speedup that is quite obvious, is just the deactivation of the DEBUG prints, as they consume a ungodly amount of cycles. Another thing we could/should change, is offloading the encryption to its accelerator. Another good thing we could do, is more smartly process the FFT and the MEL-vectors together. By this i mean that certain frequencies of the mel vector do not need the entire FFT calculation. But this could lead to a extremely more complex FFT-MEL sort of algorithm.

We will now consider 3 different situations where the cycle counts increase :

- Feature Vector Calculation becomes slower
  - Nothing much withing the duty cycle time window will change, apart that the amount of energy consumed will increase substantially.
- Packetization becomes slower
  - Since the CBC MAC Calculation is the slowest operation of this module, the only outcome is that the duty cycle slows down, from 1.21 s to many many more seconds depending on the slowdown. This will also substantially increase the energy consumption, as encryption is heavy in calculations, and consumes a lot of cycles.

- Radio transmission becomes slower
  - In the same way as the packetization situation, the duty cycle will be impacted, but this time, the energy will not increase as much. What could cause such a slow down, would probably be a slower I/O between the S2LP and the MCU.

### 3.4.4 Reflections on possible optimizations

Through the previous sections and the datasheets, we can observe that the STM32L lineup is highly capable and includes numerous hardware accelerators. Below is a list of potential optimizations we have identified, providing insights into possible directions to explore:

- Offloading the AES-128/256 encryption algorithm for the CBC MAC tag to the dedicated hardware accelerator.
- Dynamically adjust the clock frequency of the MCU to minimize power usage, and maximize efficiency of computations.
- Disabling all unnecessary modules to minimize energy usage.
- Offloading DSP tasks to the dedicated FPU of the Cortex-M4 by utilizing the appropriate instructions. Since the hardware FFT algorithms operate efficiently on powers of 2, the optimal number of samples per mel-spectrogram will likely range between 512 and 4096 samples (1 to 8 seconds). Additionally, the ADCs can be directly coupled to the FPU via the interconnection matrix for maximum processing speed. Though, since floating point operations are very heavy, this module will consume a lot of energy, and can't handle 16 bit floats. Meaning, we will probably have or transfer bottlenecks or a increased complexity on the processor.
- Alternating between Low-Power Sleep and Low-Power Run modes depending on computation requirements. Since only low power modes allow us to reach 1 Mhz clock speeds, it will be required for us to adjust the parameters as so. We also don't really care on wake-up times, as our system work at a maximum speed of a few ms. We also have to take into account the different features that are available in each mode, like the DMA's and ADC's.
- Modifying the mathematical approach of certain calculations. For example, the Mel matrix could be computed using a cumulative sum of two triangular elements on the signal, either buffered or calculated at runtime, to reduce unnecessary "0 × N" calculations. We could also take advantage of pre-computations as we have so much free memory.
- Possibly take advantage of the low power modes and high memory capacity to make much bigger mel-vectors, optimizing the use of the DMA BAM, to get as many values as possible. Then, we can maybe compress them using maybe Arithmetic Coding [8], but it may be futile, as the entropy of the data may be higher than a text file. But then again, when representing a mel-vector, we can see a sort of continuity to sounds, certain frequency rise, then lower. This could then possibly be less entropic, and allow for higher compression.
- Swapping the Mel-vectors to a machine learning compressor, and use a machine learning signal processor at the receiving end. Theoretically, this approach could get even better results if donne smartly. But, its true that this would add a black box to our system, which would make it harder to develop, and could also lead to a increased computational cost depending on the implementation.
- Since two of the heaviest steps are packetisation and transmission, we could just check while doing the Mel-spectrogram calculations, if we have at least one value higher than a threshold (maybe from a fitted function) before sending. This would allow us to reduce the energy consumption significantly in case of no event.

## 3.5 FPGA resource utilization

The FPGA currently accelerates two key functions: the low-pass filter, implemented as an FIR, and packet presence detection. This acceleration is highly beneficial because it improves energy efficiency—specialized hardware consumes less power—and boosts processing speed since these functions are at the start of the receiver chain.

At this stage of the project, the focus is limited to the lms-dsp submodules, which covers operations from the low-pass filtering to adding a flag indicating the detection of a packet. Overall resource usage is shown in Tab 4. Resource repartion of the lms-dsp, FIR and PPD are represented on Fig. 20 to Fig. 21.

[8] Arithmetic Wikipedia Article

| Resource | Usage |
|---|---|
| Total logic elements | 13594/15840 (86%) |
| Total registers | 10738 |
| Total pins | 117/130 (90%) |
| Total memory bits | 244624/562176 (44%) |
| Embedded Multiplier 9-bit elements | 3/90 (3%) |
| Total PLLs | 1/1 (100%) |

Table 4: Summary of ressource usage

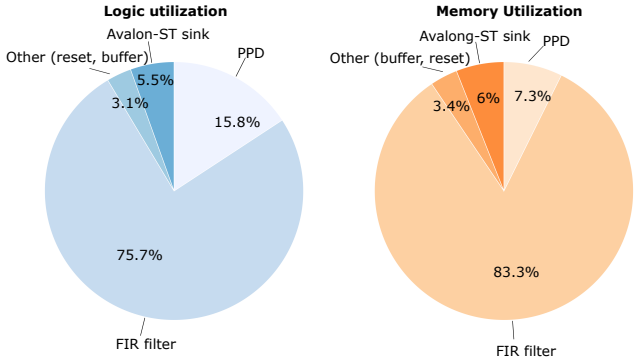



Figure 20: Resource repartition of the lms-dsp submodule (Logic Cells: 3256 ; Dedicated logic registers : 2938)

### 3.5.1 FIR

The FIR sub-block of the LMS-DSP module consumes the most logic elements, which is expected. This is because it requires two FIR filters (real and imaginary), each with 31 taps, and a 12-bit multiplier for each tap. The theoretical calculation,

$$31 \times 2 \times 12 \times 12 \approx 9,000$$

logic elements, reflects this. However, in practice, the FIR sub-block utilizes only 2,464 logic elements. This significant reduction is due to the use of embedded multipliers within the FPGA, which help minimize the reliance on logic cells. Interestingly, these embedded multipliers do not appear in the DSP column of Quartus. This discrepancy may stem from the FPGA's architecture, which seems to be somewhat unknown to Quartus.

### 3.5.2 PPD

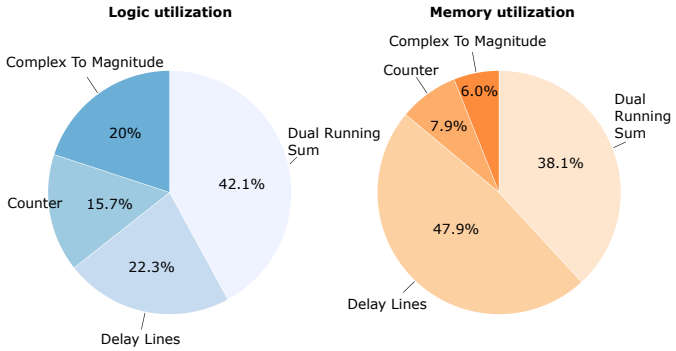



Figure 21: Resource repartition of the PPD (Logic Cells: 466; Dedicated logic registers: 215)

M9K blocks are used by the dual running sum to store previous samples, allowing them to be subtracted from the current sum during each cycle. The long shift is expected to require $256 \cdot 16 = 4096$ and the short shift $32 \cdot 16 = 512$ memory bits. However, in practice, the long shift uses 4064 bits, and the short shift uses 480 bits which is 32 bits less than the calculated values in both cases. This difference may come from the first and last samples being stored

outside the M9K blocks, possibly in registers or other memory structures. This approach optimizes memory usage and ensures efficient access during processing.

### 3.5.3 Resource Optimization and Timing Analysis

If resources need to be reduced, the length of the short shift and long shift could be adjusted, though this would result in a less reliable estimation of noise and current power. Alternatively, the number of taps in the FIR filter could be reduced, but this would increase the cut-off frequency, allowing more noise to pass through the processing chain. Both options involve trade-offs between resource usage and performance quality.

Finally, we can have a look at the timing analysis summary from quartus given in Tab. 5.

| Clock | Setup | Hold | Recovery | Removal | Minimum pulse |
|---|---|---|---|---|---|
| Worst case slack | -0.041 | 0.006 | 0.377 | 0.253 | 0.000 |

Table 5: Worst case slack

A very small setup time violation has been observed, calculated under worst-case conditions, including reduced supply voltage, transistor performance variations, and temperature fluctuations. Given its very low value and the likelihood of these conditions occurring, this setup time violation is considered acceptable as it is unlikely to cause any issues in practice.

# 4 Conclusion

To conclude on this report, we have seen how the systems works as a whole and the goals it entails. The characterization and analysis of important aspects have highlighted important limitations, and by effect, possible future optimization. The different **limitations** can be summarized:

1. Classification accuracy: while acceptable performances have been achieved, some errors tends to happen frequently (confusing one classes for another) and this leads to not optimum classification. Due to real-world distortions and ambient noise, some classes (birds and fire) tend to perform more poorly as perturbations increases. This is detrimental to optimal use in a possibly noisy forest.
2. Telecommunication: high noise channel, and multi path needing to be handled, high quantities of data that are heavy to transmit.
3. Power consumption: some operations are still responsible for high-energy consumption such as the CBC MAC tag calculation and the static consumption due to the high frequency of the processor.

The **optimizations** linked to those limitations are:

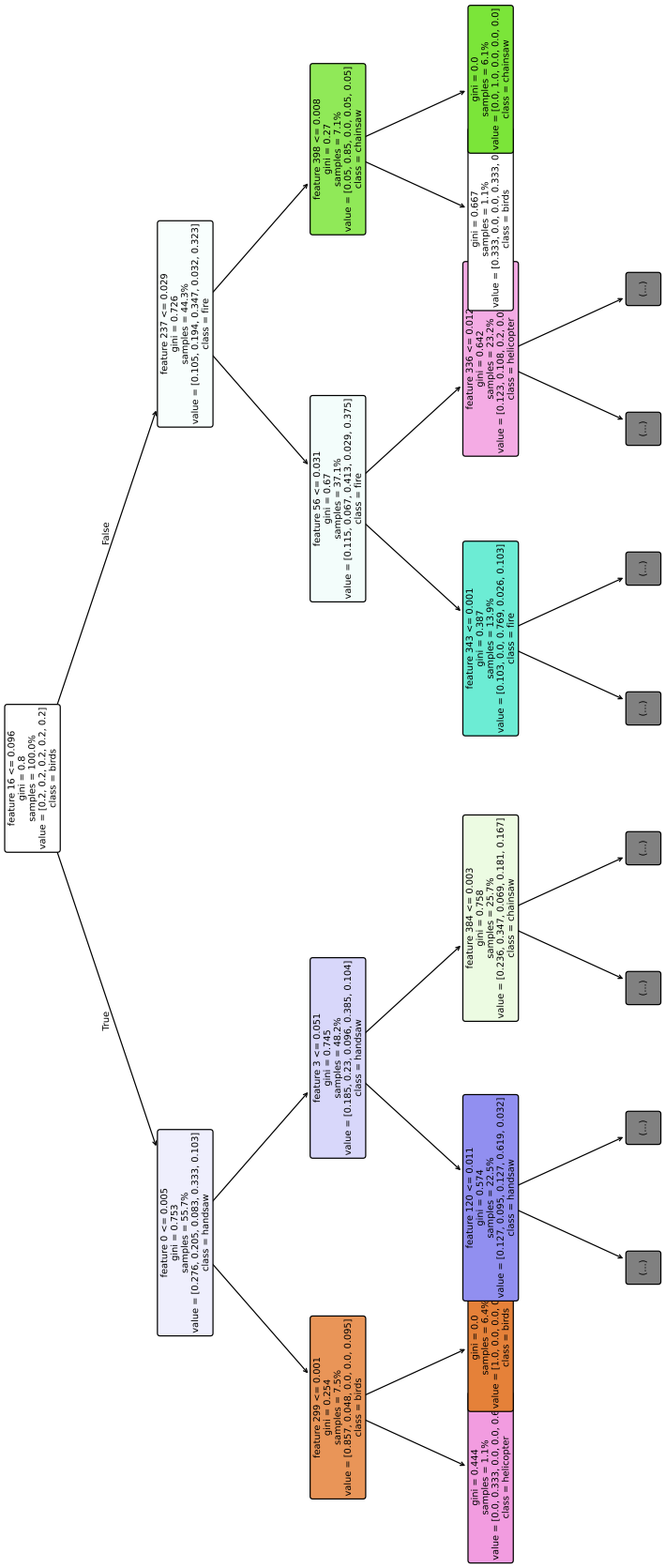
1. Classification accuracy: more complex models (e.g. random forests) or support vector classifiers. The use of memory effect or expanding the data set size could also improve the model efficiency. Data augmentation could be further used.
2. Telecommunication: improve packetisation and modulation to be able to preform well at much lower power levels and lower SNR. By using techniques such as chirp spread spectrum[9] and error correcting codes. Or adapting our system to pre-existing systems such as LoRa[10][11], increasing complexity substantially.
3. Power consumption: Reducing the usage of high-energy consumption tasks such as CBC MAC tag, dynamically reducing the operating frequency and dynamically or statically turning off modules that aren't required for the current steps or for the entire system.
4. Memory utilization: currently, we don't make full use of memory. We could use more of it to improve processing and precision by using more buffering and pr-computations.
5. Sensing duty cycle: a lengthily list of those optimization is written at section 3.4.4.

In conclusion, a lot of work has been done this semester. We have learnt new practical skills and how to use new tools but also new theory and concepts. We have also been able to find some limitations of our system and possible solutions for optimization in the future. This is a complex project that is only getting started.

[9] Wikipedia article on Chirp Spread Spectrum
[10] Wikipedia article on LoRa
[11] Github repository of a reverse engineering effort for MCU ready, bit banged, LoRa transmitter

# 5 Appendices

## 5.1 Images



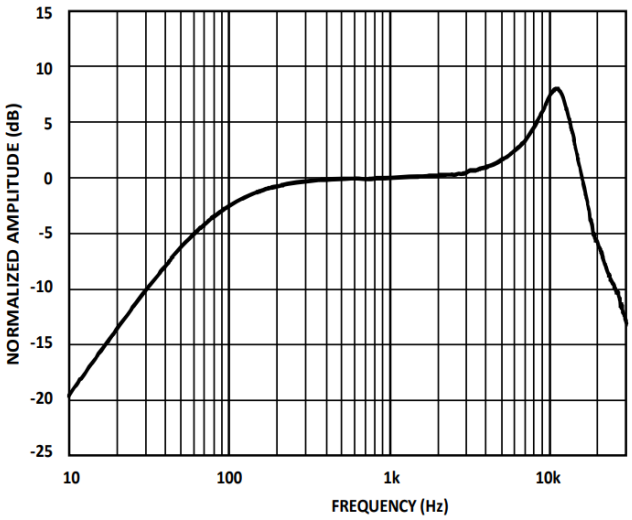



Figure 23: Measured transfer function of the microphone from the datasheet

| Function | Local cost | Type | Location | Info |
|---|---|---|---|---|
| print_encoded_packet | 1672 | STATIC | adc_dblbuf.c:55 | |
| send_spectrogram | 832 | STATIC | adc_dblbuf.c:81 | |
| AES128_encrypt | 576 | STATIC | aes_ref.c:312 | |
| HAL_ADC_ConfigChannel | 224 | STATIC... | stm32l4xx_hal_adc.c:2... | Local cost uncertain d... |
| SearchDatarateME | 216 | STATIC | s2lp.c:302 | |
| HAL_ADC_MspInit | 184 | STATIC | adc.c:93 | |
| HAL_UART_MspInit | 184 | STATIC | usart.c:60 | |
| SearchFreqDevME | 168 | STATIC | s2lp.c:349 | |
| S2LP_PLLConf | 152 | STATIC | s2lp.c:221 | |
| HAL_ADCEx_MultiModeCo... | 144 | STATIC | stm32l4xx_hal_adc_ex.... | |
| arm_absmax_q15 | 120 | STATIC... | arm_absmax_q15.c:106 | Local cost uncertain d... |
| UART_CheckIdleState | 104 | STATIC... | stm32l4xx_hal_uart.c:3... | Local cost uncertain d... |
| SystemClock_Config | 96 | STATIC | main.c:206 | |
| ComputeDatarate | 96 | STATIC | s2lp.c:283 | |
| UART_EndRxTransfer | 88 | STATIC... | stm32l4xx_hal_uart.c:3... | Local cost uncertain d... |
| ComputeFreqDeviation | 80 | STATIC | s2lp.c:338 | |
| Spectrogram_Compute | 80 | STATIC | spectrogram.c:50 | |
| UART_SetConfig | 72 | STATIC | stm32l4xx_hal_uart.c:3... | |
| MixColumns | 64 | STATIC | aes_ref.c:182 | |
| KeyExpansion | 64 | STATIC | aes_ref.c:209 | |
| MX_GPIO_Init | 64 | STATIC | gpio.c:48 | |
| tag_cbc_mac | 64 | STATIC | packet.c:17 | |
| S2LP_WriteTxFIFO | 64 | STATIC... | s2lp.c:70 | Local cost uncertain d... |
| MX_ADC1_Init | 48 | STATIC | adc.c:31 | |
| HAL_SPI_MspInit | 48 | STATIC | spi.c:64 | |
| HAL_ADC_Init | 48 | STATIC | stm32l4xx_hal_adc.c:3... | |
| HAL_SPI_TransmitReceive | 48 | STATIC | stm32l4xx_hal_spi.c:1... | |
| SPI_WaitFifoStateUntilTime... | 48 | STATIC | stm32l4xx_hal_spi.c:4... | |
| HAL_UART_Transmit | 48 | STATIC | stm32l4xx_hal_uart.c:1... | |
| HAL_UART_Receive | 48 | STATIC | stm32l4xx_hal_uart.c:1... | |
| S2LP_ReadReg | 40 | STATIC... | s2lp.c:36 | Local cost uncertain d... |
| MX_TIM3_Init | 40 | STATIC | tim.c:30 | |
| ADC_ConversionStop | 40 | STATIC | stm32l4xx_hal_adc.c:3... | |
| NVIC_EncodePriority | 40 | STATIC | core_cm4.h:1856 | |
| HAL_RCC_OscConfig | 40 | STATIC | stm32l4xx_hal_rcc.c:405 | |
| HAL_RCC_GetSysClockFreq | 40 | STATIC | stm32l4xx_hal_rcc.c:1... | |
| SPI_WaitFlagStateUntilTim... | 40 | STATIC | stm32l4xx_hal_spi.c:3... | |
| ShiftRows | 32 | STATIC | aes_ref.c:160 | |
| make_packet | 32 | STATIC | packet.c:48 | |
| _write | 32 | STATIC | retarget.c:40 | |
| _read | 32 | STATIC | retarget.c:71 | |
| S2LP_Command | 32 | STATIC... | s2lp.c:21 | Local cost uncertain d... |
| S2LP_WriteReg | 32 | STATIC... | s2lp.c:54 | Local cost uncertain d... |
| S2LP_Send | 32 | STATIC... | s2lp.c:89 | Local cost uncertain d... |
| S2LP_SetModulation | 32 | STATIC | s2lp.c:388 | |
| S2LP_Init | 32 | STATIC... | s2lp.c:456 | Local cost uncertain d... |
| _sbrk | 32 | STATIC | sysmem.c:54 | |
| hex_encode | 32 | STATIC | utils.c:45 | |
| LL_ADC_SetOffset | 32 | STATIC | stm32l4xx_ll_adc.h:36... | |
| LL_ADC_SetOffsetState | 32 | STATIC | stm32l4xx_ll_adc.h:38... | |
| LL_ADC_REG_SetSequencer... | 32 | STATIC | stm32l4xx_ll_adc.h:42... | |
| LL_ADC_SetChannelSampli... | 32 | STATIC | stm32l4xx_ll_adc.h:54... | |

Figure 24: Static stack analyzer trace

## 5.2 Contributions

During this semester, we also explored the limitations of the tools provided by our teachers and worked to enhance them. Developing the uart_reader GUI and submitting pull requests to contribute to the course's repository was a lengthy but rewarding task that taught us valuable lessons. We have planned several improvements and are committed to enhancing the project experience for everyone while actively contributing our own solutions.