NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

MEEC/MIEEC

Electronics for Micro-Systems

Project
Voice Activity Detector

Authors:

Martim Duarte Agostinho (70392) md.agostinho@campus.fct.unl.pt
Francisco Simões Coelho Sá da Costa (70386) fsc.costa@campus.fct.unl.pt
Sofia Margarida Mafra Dias Inácio (58079) sm.inacio@campus.fct.unl.pt

2024/2025 – 1st Semester - DEEC

Contents

List of Figures

List of Tables

1 Introduction

The Voice Activity Detector (VAD) project focuses on the design, implementation, and testing of a mixed-mode embedded system capable of detecting and processing voice activity. The primary objective is to create an energy-efficient system that transitions from idle to active states upon detecting human voice frequencies. This project integrates both analog and digital signal processing techniques, along with IoT components, to provide a comprehensive solution for voice-activated control.

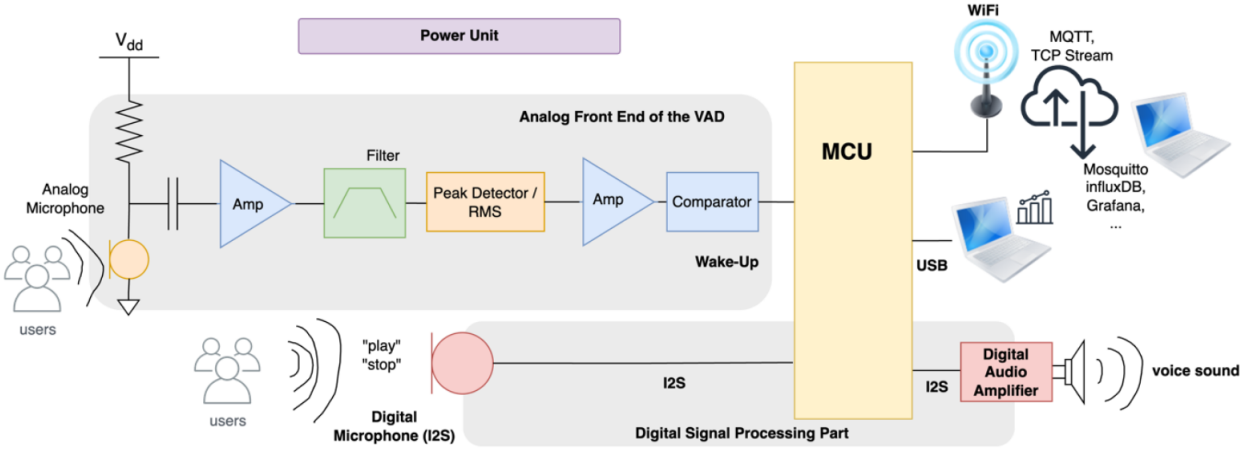


Figure 1: Voice Activity Detector Block Diagram [1] .

Figure 1 shows a simplified block diagram of the proposed system with two major components, the Analog Front End (AFE) and the digital component.

The AFE block is responsible for detecting audio peaks in the human voice frequency range in order to wake the MCU from deep sleep. This block will need an amplifier, a band-pass filter a peak detector and a comparator.

The digital component, as to wake up to the AFE wake up signal, detect key words, play music according to the keywords and handle the user interaction with the system. In addition to the real time interaction with the system there will be an IoT component, first with Grafana in order to visualize data and finally to enhance system capability integration with Home Assistant was developed.

2 Analog Front End

In this section, the complete Analog Front End circuit will be presented. This is composed by four different stages, the Pre-Amp stage, the Filter, a Peak Detector and finally a Comparator. The complete circuit can be seen in Figure 2.

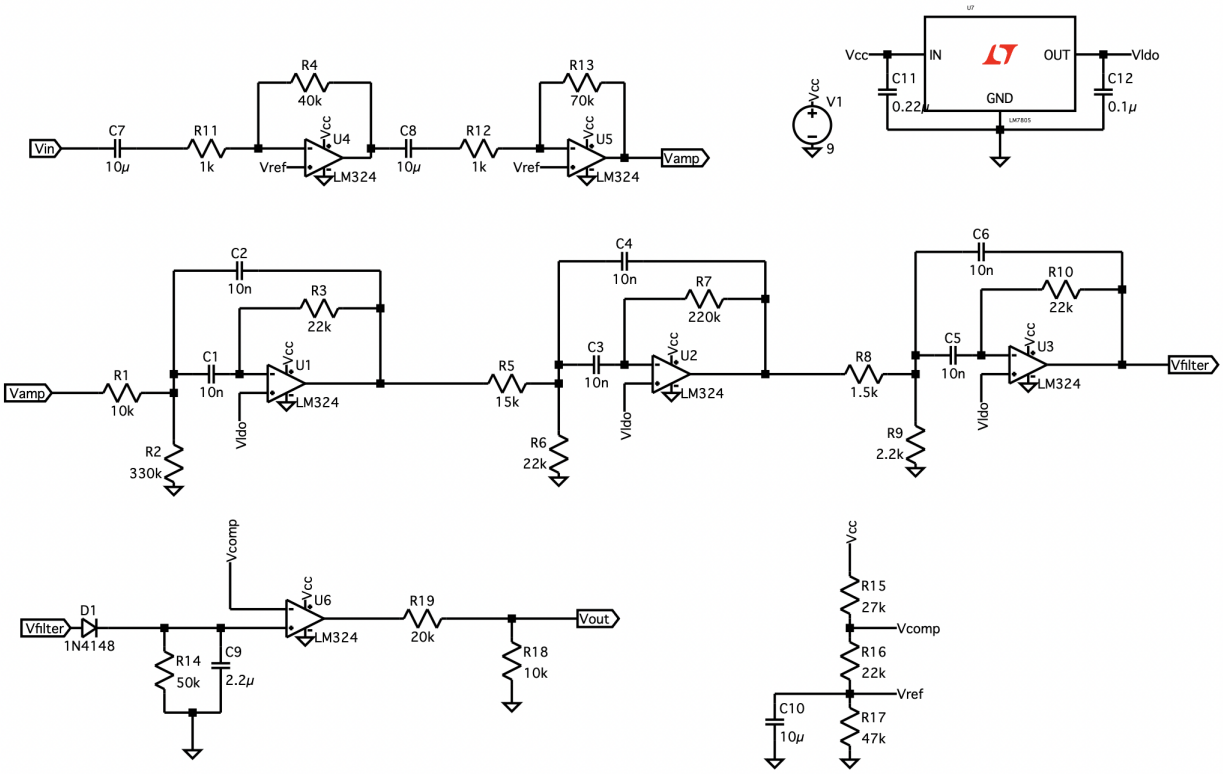


Figure 2: Analog Front End full circuit.

Throughout this project, the LM324 Operation Amplifier was used, because it can be powered by $9V$, unlike the MCP6001, and because of its low power consumption and low output noise [3]. The only downside of this OpAmp is the fact that it is not rail-to-rail, but this fact will not be a problem, since the signals that will be used will have low amplitudes. In addiction to that, the E12 series for resistors and capacitors was also used.

2.1 Pre-Amp

2.1.1 Design

To design the Pre-Amp stage, the variation of the output voltage of the Electret microphone used is critical to define the total gain of this circuit, so the sensibility of the microphone used is described by the following equation [1]:

$$Sens_{dBV} = 20 \cdot log_{10}(\frac{Sens_{mv/Pa}}{1000\,\mathrm{mV/Pa}}) \tag{1}$$

So the output voltage in function of the Sound Pressure Level in dB (SPLdB), can be obtained from equation 2 [4].

$$V_{in} = 10^{\frac{SPLdB-42}{20}} \cdot \frac{20e^{-6}}{\sqrt{2}} \cdot 1000[\mathrm{mV}] \tag{2}$$

Resulting in the graphic of Figure 3.

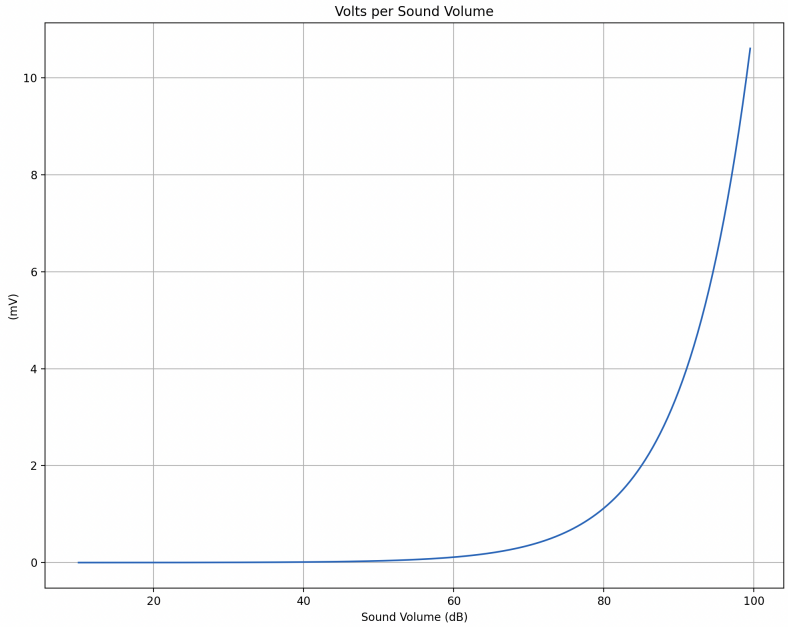


Figure 3: Electret microphone voltage in function of SPLdB.

For this project, the SPLdB obtained by the human voice is the most important, and the gain of the Pre-Amp will depend on this factor, if the gain is too low, the output voltage will also be to low for the desired range, but if this gain is too high, the output voltage for the human voice may reach the supply voltage and so, the signal generated by the voice would be lost after this process, resulting in a simple DC value equal to the supply voltage, and only sounds with a lower SPLdB would have an effect on the remaining of the circuit.

So, considering that the human voice SPLdB is contained in $[60, 70]dB$ [5], the output voltage of the Electret microphone will be $[0.112, 0.355]$mV, respectively.

With this information, the total gain of the Pre-Amp circuit needs to a high value, to avoid using resistors with very high, or very low resistance values, two Inverter circuits connected in series, as shown in figure 4.

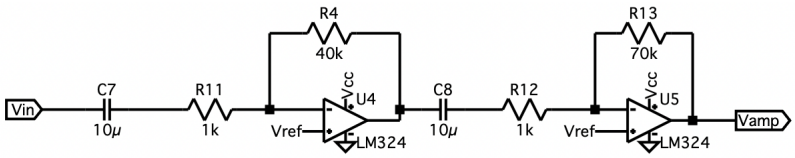


Figure 4: Pre-Amp circuit.

The resulting AC component of the output voltage is described by equation 3.

$$v_{amp} = G_1 \cdot G_2 \cdot v_{in} = \frac{R_{b1}}{R_{a1}} \cdot \frac{R_{b2}}{R_{a2}} \cdot v_{in}[\text{V}] \tag{3}$$

To obtain a maximum value of approximately $1V$, a gain of 2800 is taken into account for this circuit, so assuming $R_{a1} = R_{a2} = 1k\Omega$, the final values for the remaining two resistors will be, $R_{b1} = 40k\Omega$ and $R_{b2} = 70k\Omega$.

In order to guarantee a DC component of $V_{CC}/2$ in the amplified signal, in a single supply circuit, an DC voltage, generated by the circuit in Figure 5 is added to the positive input node of both OpAmps, and finally, by using two capacitors with high capacitive, the DC component after both stages is $V_{CC}/2$.

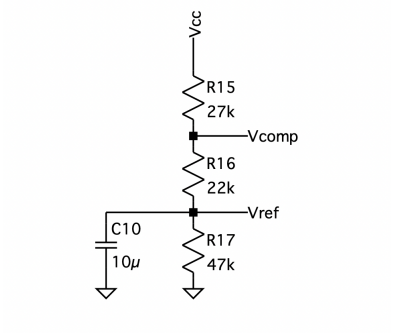
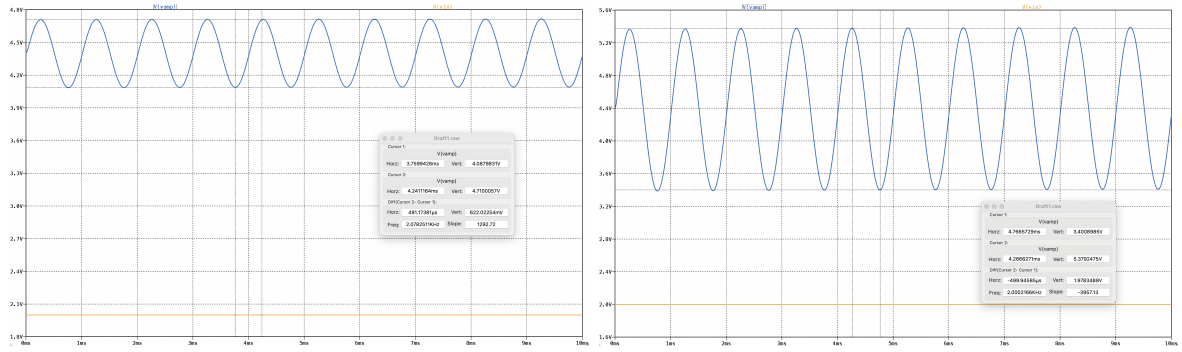


Figure 5: V_{ref} and V_{comp} circuit.

2.1.2 Simulation

Using the program LTSpice, the response of the Pre-Amp circuit for both cases mentioned above is obtained through a transient simulation.

The results for two Sine waves equal to $V_{in} = 2 + 0.112e^{-3} \cdot sin(2\pi 1k \cdot t)$ and $V_{in} = 2 + 0.355e^{-3} \cdot sin(2\pi 1k \cdot t)$, i.e., for a SPLdB of 60 and $70dB$, respectively, are shown in Figure 6.



(a) V_{amp} for $60dB$. **(b)** V_{amp} for $60dB$

Figure 6: V_{amp} for a SPLdB of 60 and $70dB$

Other way to visualize the response of this circuit is through the AC analysis, the resulting are shown in Figure 7.

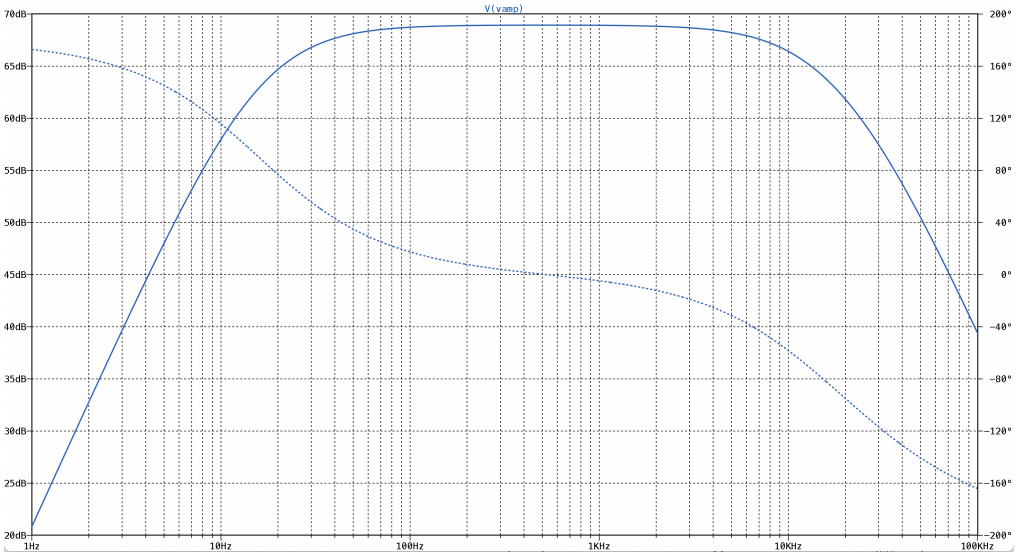


Figure 7: Bode diagram of the Pre-Amp circuit.

Despite showing a band-pass filter response, the Pre-Amp circuit will not have an effect in the signal generated by the human voice, given that the frequency of this signal will be contained in $[100, 4k]$Hz.

To evaluate the Pre-Amp sensibility to variations in the values of its components, the Monte-Carlo analysis should be used, therefore, the Monte-Carlo analysis of the Pre-Amp circuit can be observed in Figure 8, assuming a tolerance of 5% for the resistance as well as the capacitance values.

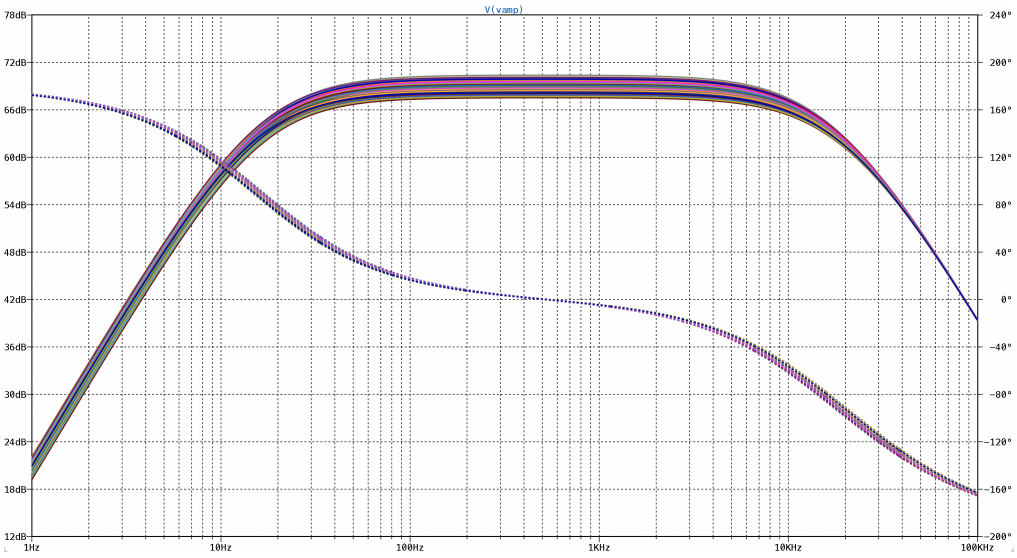


Figure 8: Monte-Carlo analysis of the Pre-Amp circuit.

Analyzing the graphic above, the maximum variation of the gain for the desired frequencies is approximately $3dB$, witch represents a variation that will not negatively affect the final result of this circuit, given that the input voltage is too low to reach the supply voltage value.

2.2 Analog Filter

2.2.1 Filter Type and Approximation

In filter design, the first step is to evaluate the filter type and the corresponding Passband and Stopband regions. For that, a Cutoff and a Stop frequency must be defined.

Since the signal generated by the human voice is contained in $[100, 4k]$Hz, the Band-Pass filter is the right choice for this project. For this type of filter, the Passband is contained between two different Cutoff frequencies, so, the filter designed must meet the following conditions:

- First Cutoff Frequency f_1 : 200Hz;
- Second Cutoff Frequency f_2 : 4000Hz;
- Central Frequency f_c : $\sqrt{f_1 f_2} = 895$Hz;
- Maximum Ripple in the Passband : $1dB$;
- Order : $6th$.

After choosing the filter speciations, a filter response approximation must be chosen as well. For this case, the only factor to consider for this choice is the frequency response of each approximation, the phase and group delay will not have any major influence, given that there is no need to maintain the signal integration for the remaining of the AFE circuit.

With these considerations in mind, in Figure 9, are on display the different frequency response for the four main approximations.

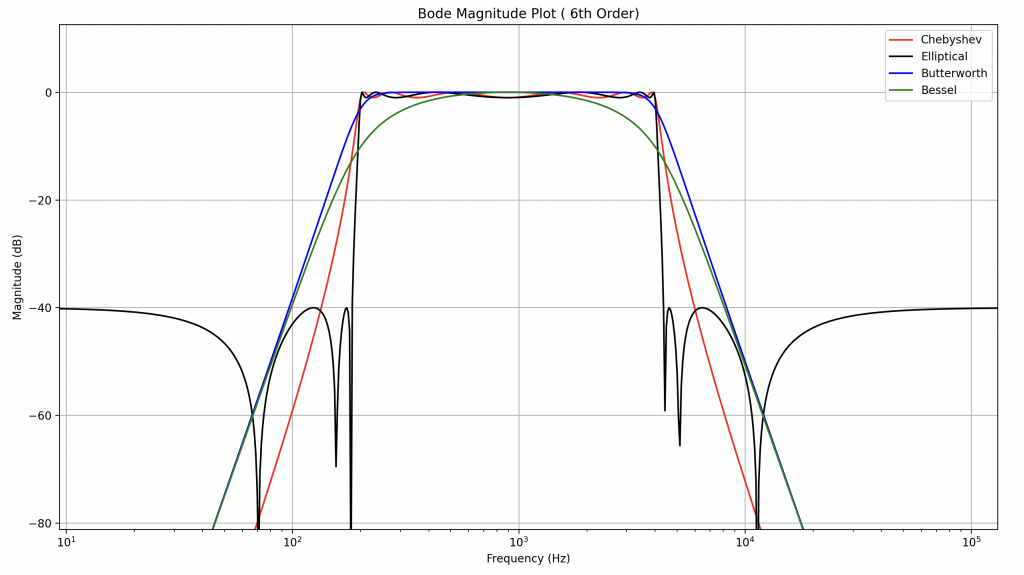


Figure 9: Frequency response comparison.

The desired approximation is the one that most resembles the ideal filter response, that is, the one with the sharpest drop in the Transition band, region where the magnitude goes from $0dB$ to $-40dB$. Following this factor, the best approximation will be the Elliptical

approximation, but, implementing the resulting equation in an Active Filter circuit, would require much greater complexity than then what is possible to implement using all the available components. Because of that, the second-best approximation, the Chebyshev approximation was chosen for this application.

To better improve the result of the filter, the Passband was shrunk by increasing the Fisrt Cutoff Frequency from 200Hz to 350Hz and reducing the Second Cutoff frequency from 4000Hz to 3500Hz, sacrificing a small portion of desired frequencies, but also eliminating frequencies that were not intended. The Bode diagram of the resulting filter approximation can be seen in Figure 10.

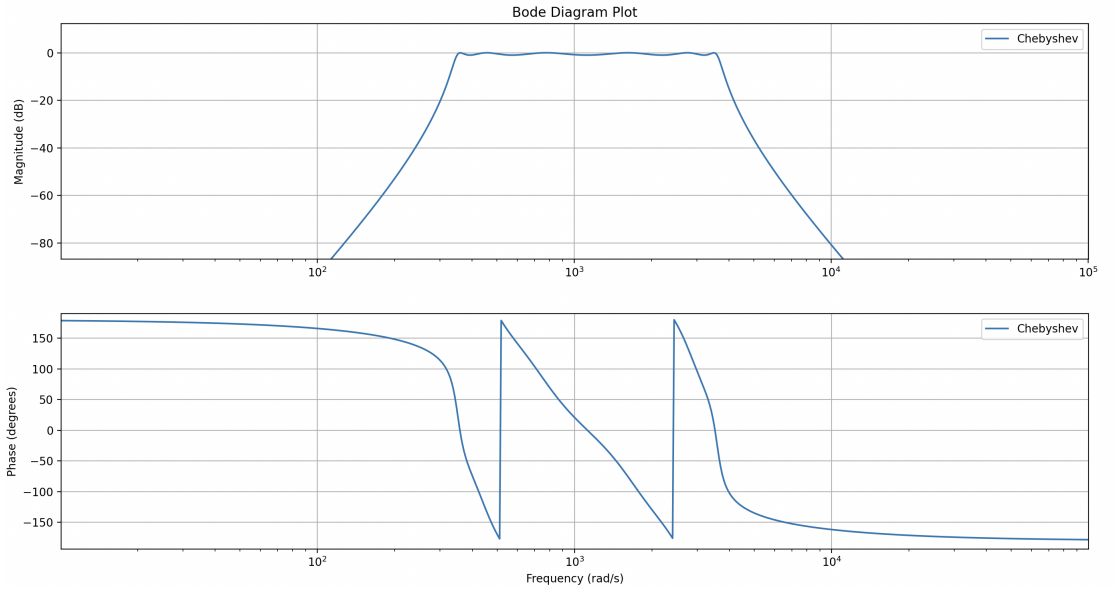


Figure 10: Final Filter Bode Diagram.

Which can be described by the following equation:

$$\frac{7.2 \cdot 10^{19} s^6}{1.0s^{12} + 3.0 \cdot 10^3 s^{11} + 2.8 \cdot 10^7 s^{10} + 6.0 \cdot 10^{10} s^9 + 2.3 \cdot 10^{14} s^8 + 3.2 \cdot 10^{17} s^7 + 5.8 \cdot 10^{20} s^6 + \\ + 4.0 \cdot 10^{23} s^5 + 3.7 \cdot 10^{26} s^4 + 1.2 \cdot 10^{29} s^3 + 7.0 \cdot 10^{31} s^2 + 9.6 \cdot 10^{33} s + 4.0 \cdot 10^{36}} \tag{4}$$

2.2.2 Filter Topology

The next step in the design of a filter is to choose the filter topology. There are two main options, the most common Sallen-Key topology, and the Multiple Feedback topology. The implementation of both topologies is quite similar and the decision between these two approaches for the final filter implemented will be decided after analyzing their behavior in the simulations.

2.2.3 Sallen-Key Filter Design and Simulation

First, the Sallen-Key topology was implemented, by using three Second Order Bandpass Filters, equal to the filter in Figure 11 [6].

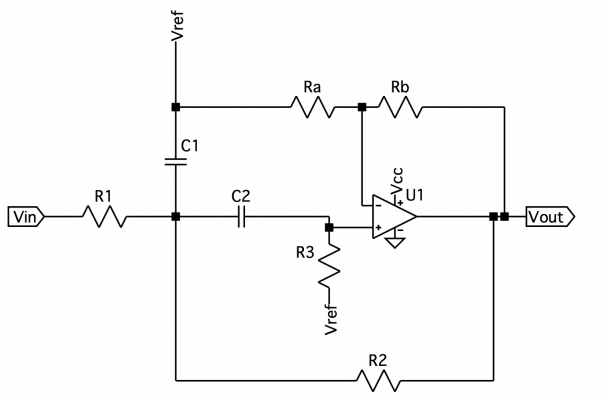


Figure 11: Sallen-Key Second Order Bandpass Filter

With this topology, the complete Sixth order filter is shown in Figure 12.

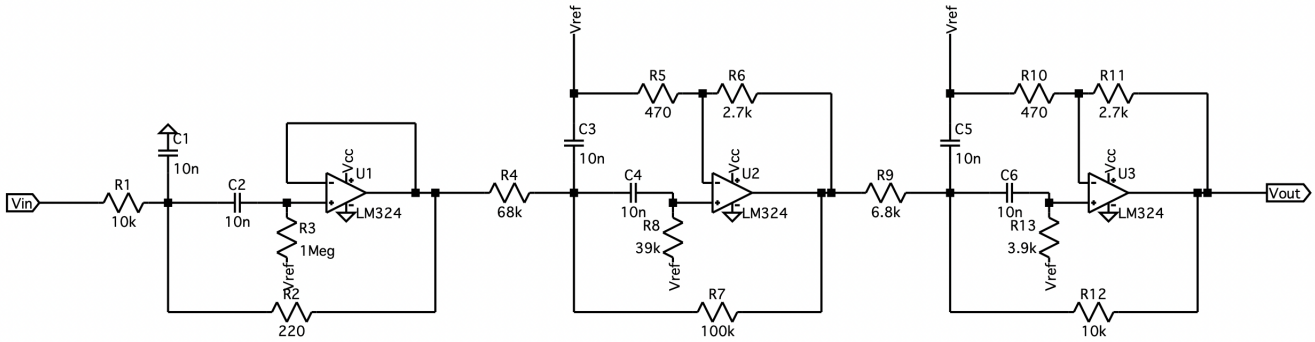


Figure 12: Complete Sallen-Key Sixth Order Bandpass Filter

Using the AC analysis from LTSpice, the Frequency Response graphic is obtained, and this can be observed in Figure 13.

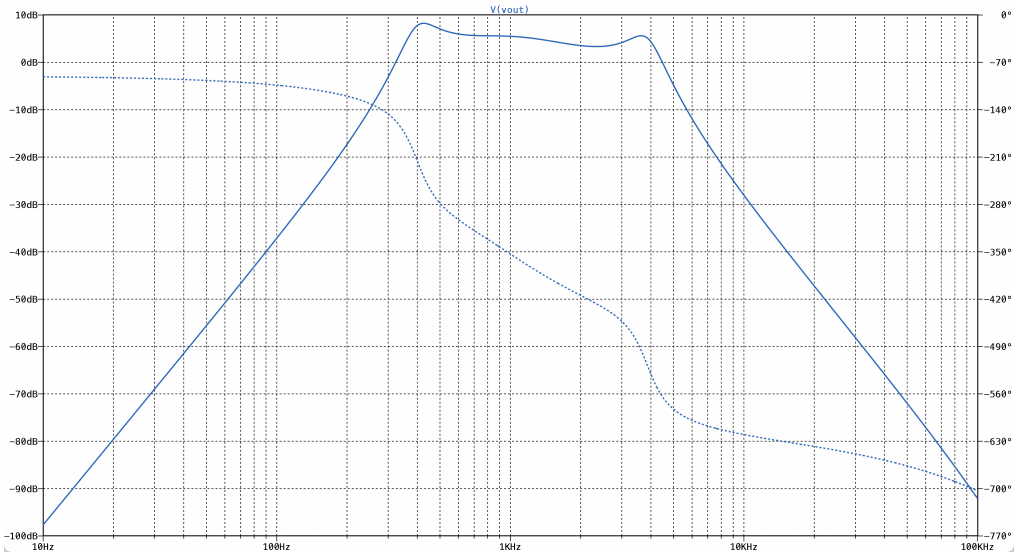


Figure 13: Bode Diagram of the Sallen-Key Filter

Analyzing the Bode Diagram above, it is possible to observe that exists a small gain of approximately $5dB$, and a ripple above $1dB$, this difference in the filter behavior due to the restriction on the values of the resistors capacitors, since only the series E12 are available. So it is possible to assume that the Sallen-Key topology is quite sensitive to changes in its components. On the other end, the size of the Passband and the location of both Cutoff Frequencies did not suffer major changes.

To confirm the question of the filter's sensibility, a Monte-Carlo analysis was made, with the results in Figure 14.

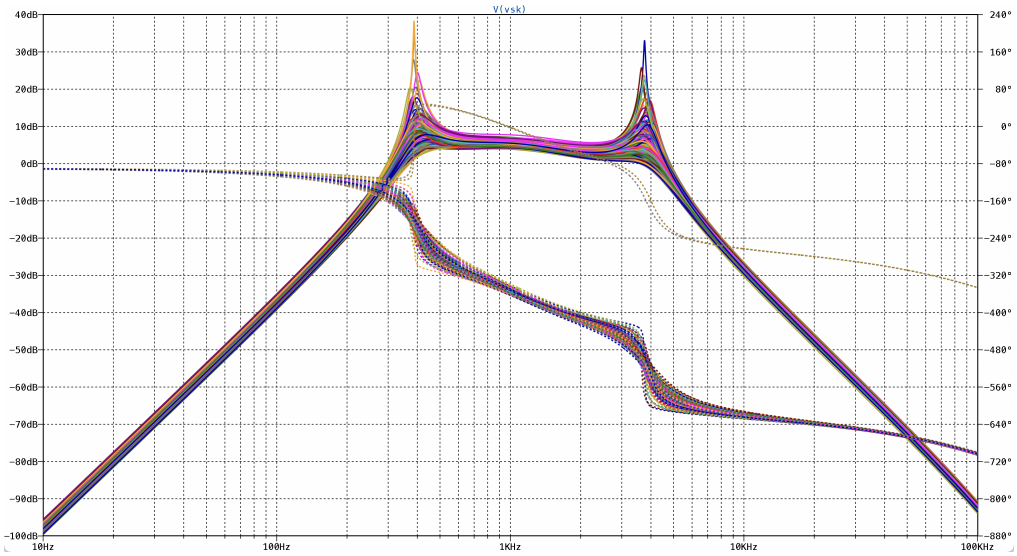
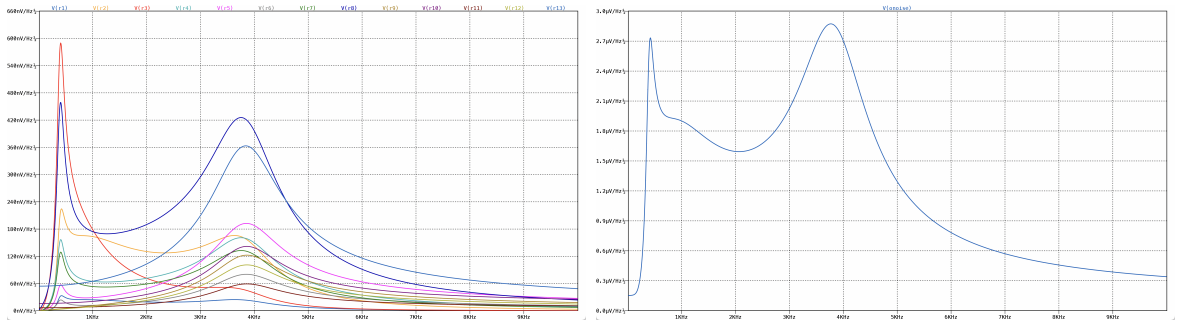


Figure 14: Monte-Carlo analysis of the Sallen-Key Filter

This graphic proves that the filter is very sensitive to changes, even having instable pole near the first Cutoff Frequency on two iterations and in other iterations, the overshoot in this area reach almost $40dB$, this is not a good factor when designing a filter for any application. Other

factor that influences the choice of the topology is the noise generated by every component in the circuit. Using the Noise analysis of LTSpice, the noise per resistor and total noise is shown in Figure15.



(a) Noise generated by each resistance. **(b)** Total Noise output

Figure 15: Noise generated by the Sallen-Key Filter

The total RMS noise is $146.78\mu V/\sqrt{\text{Hz}}$, this value is not favorable for this circuit, this value is the result of resistors with very high resistance values, like, e.g. $1M\Omega$.

2.2.4 Multiple Feedback Filter Design and Simulation

Moving on to the Multiple Feedback Filter Topology, the implementation of the complete filter is the same as in the previous case, three Second Order Bandpass Filters, equal to the one shown in figure 16 [6].

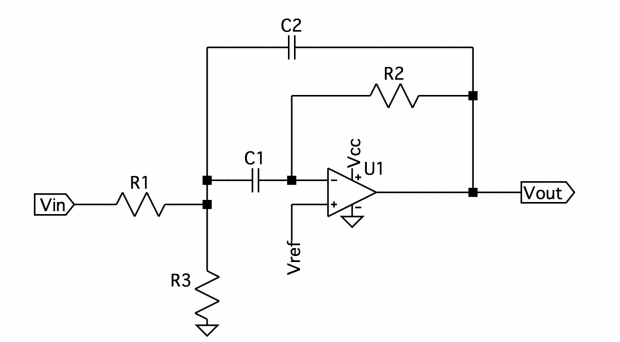


Figure 16: Multiple Feedback Second Order Bandpass Filter

So, the complete Sixth Order Bandpass Filter can is displayed in Figure 17.

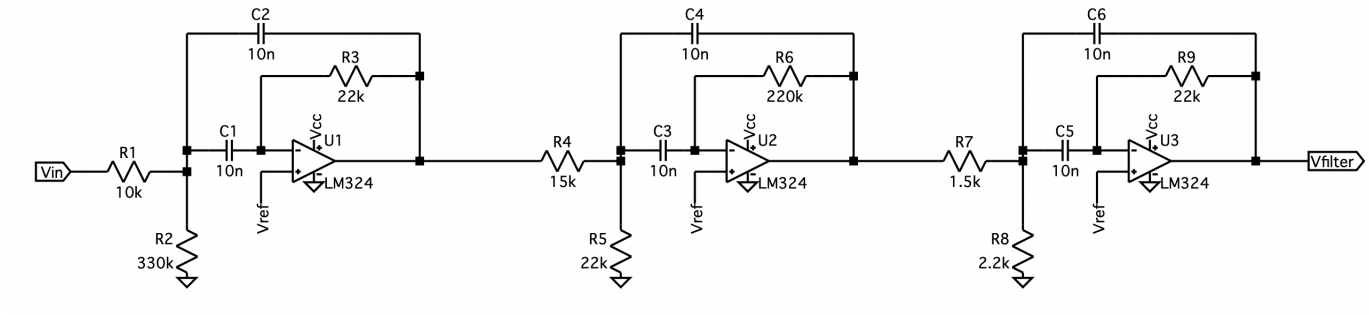


Figure 17: Complete Multiple Feedback Sixth Order Bandpass Filter

Repeating the same simulation process as before, first, the filter Frequency response is expressed in Figure 18.

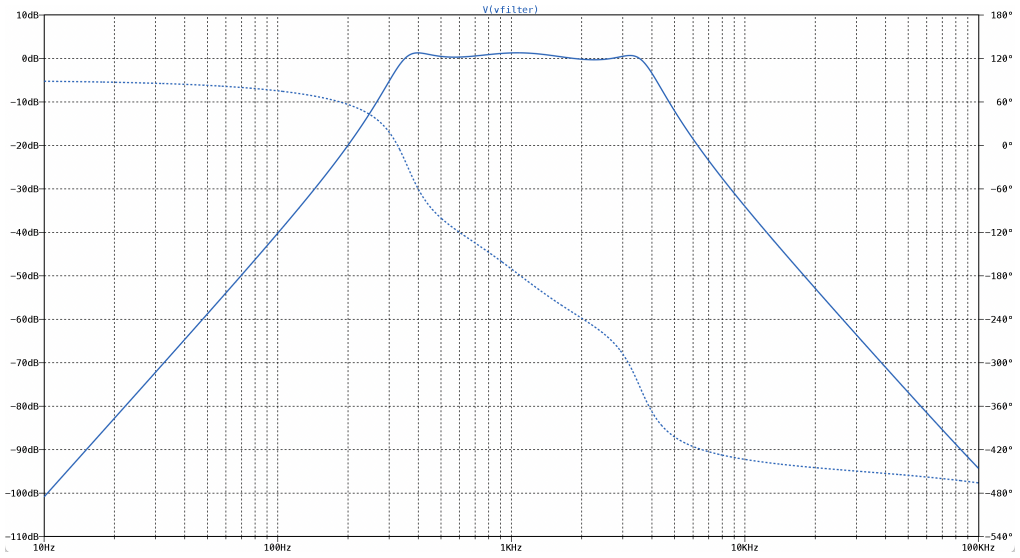


Figure 18: Bode Diagram of the Multiple Feedback Filter

Unlike the Sallen-Key filter, the Multiple Feedback filter presents the expected Frequency Response, with the correct unitary gain and ripple lower than $1dB$, even with the same restriction cause by the series E12 of the components. Added to this point comes the fact that the Passband is equal to the theoretically determined.

Even so, sensitivity analysis is crucial to verify any design, therefore, the Monte-Carlo analysis of the Multiple Feedback filter can be seen in Figure 19.

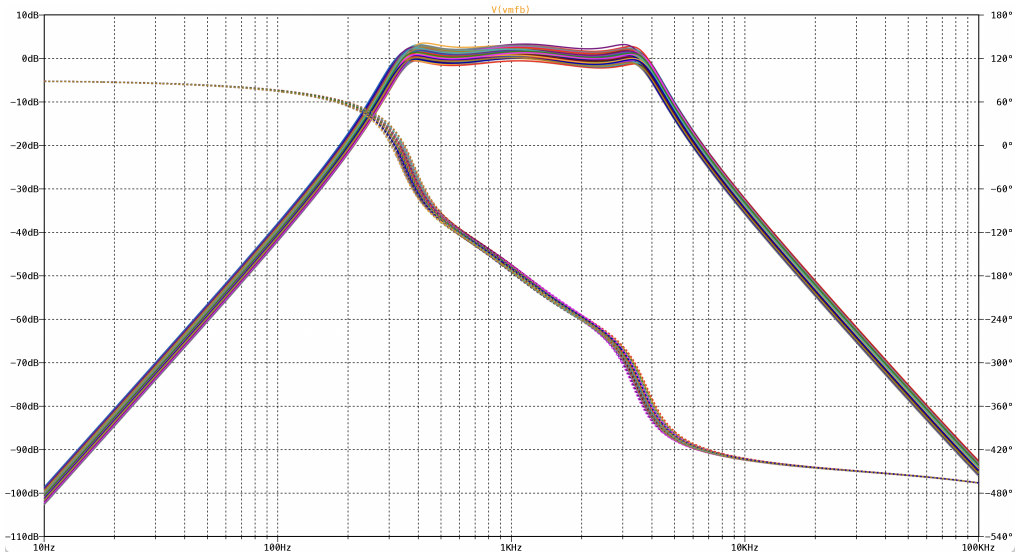
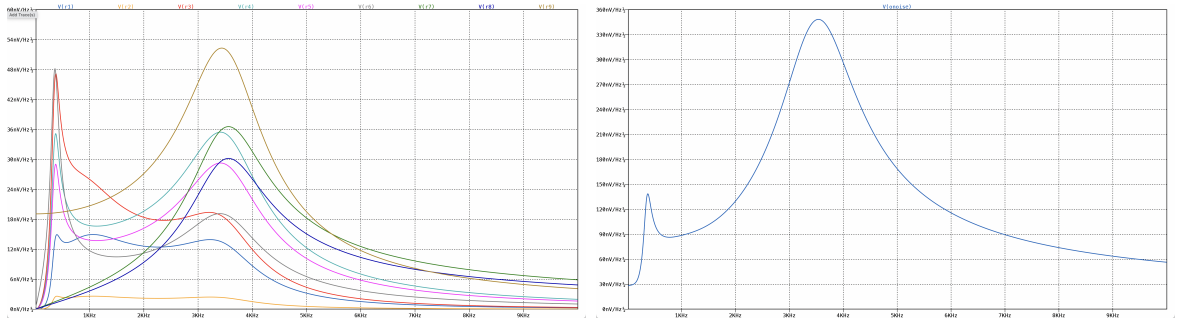


Figure 19: Monte-Carlo analysis of the Multiple Feedback Filter

Just like the previous case, this graphic proves the assumption presented, but, for this time, the Monte-Carlo analysis proves that any variation in any component in this circuit will have a minimum effect in the final response, given that the maximum variation with the same tolerance of 5% in every component is only of $3dB$, in the Passband gain, without any iteration with a different behavior like before. This is a very positive factor for the design of any circuit.

Lastly, using the Noise analysis the noise per resistor and total noise is shown in Figure20.



(a) Noise generated by each resistance. **(b)** Total Noise output

Figure 20: Noise generated by the Multiple Feedback Filter

The total RMS noise is $16.053\mu V\sqrt{Hz}$, although this value is lower than the generated by the Sallen-Key filter, it still is quite high, the main responsible component for this value, higher than expected, is the OpAmp chosen for this project.

In conclusion, the Multiple Feedback topology is better than the Sallen-Key topology in all the factor considered for this project, and because of that, this was the chosen topology.

2.3 Peak Detector and Comparator

The Peak Detector circuit consists of a simple Half Bridge Rectifier with a resistor and a capacitor, responsible for retaining the voltage peak of the input signal for a period of time,

thus transforming the signal generated by the human voice, in a DC value. This value is then compared, by a OpmAmp without feedback, with a reference value, generated in the circuit in Figure 5 as V_{comp}, thereby determining if the MCU should wake up or stay in deep sleep. The Peak Detector and Comparator circuit can be seen in Figure 21.

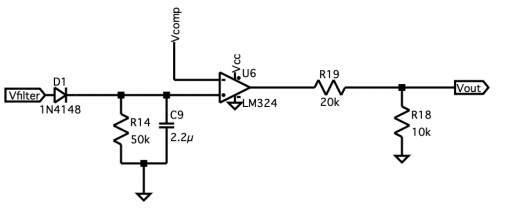


Figure 21: Peak Detector and Comparator circuit

The final output voltage value also goes through a voltage divider, to avoid surpassing the maximum voltage value for the GPIO pins of the MCU used, in the case of the ESP32, $3,3V$ [7]. The simulation of this circuit with a sine wave generated with a small delay is shown in Figure 22.

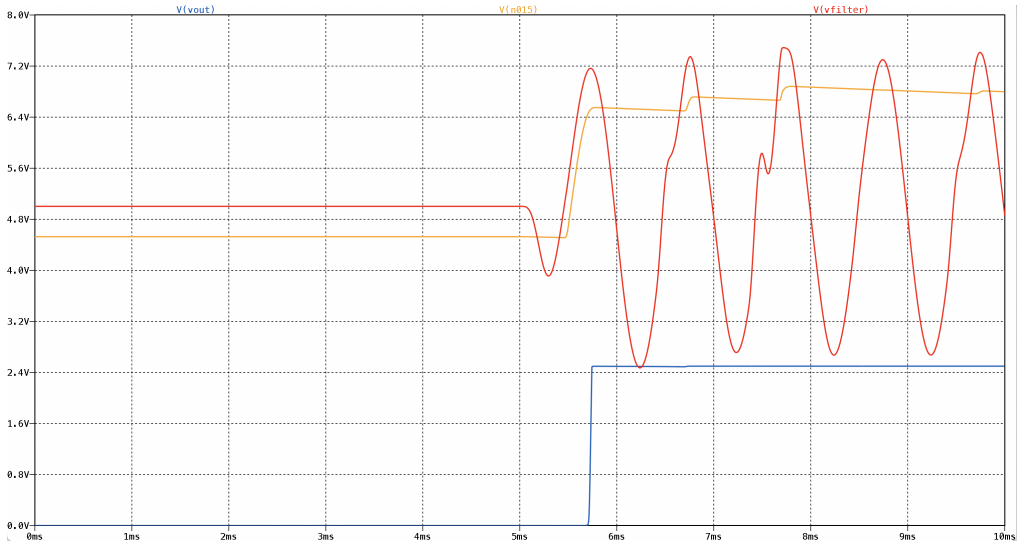


Figure 22: Final output with and without voice signal

3 Power Management Unit

The Power Management Unit (PMU), is responsible for supplying voltage to the entire AFE circuit and also to the MCU. The aim of this project is to have a battery as its main voltage source, so, since these batteries generate $9V$, this value most be reduced to a value supported by the ESP32 [7], for this purpose, the LM7805 Linear Regulator was implemented, reducing the voltage to $5V$ [?]. The complete PMU circuit is exposed in Figure 23.

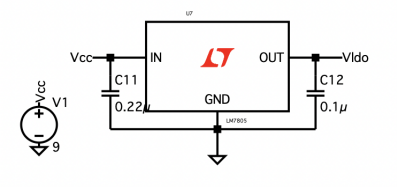


Figure 23: Power Management Unit circuit

Since the objective of this project is to remain on the entire time, the power consumed by the circuit is extremely important, where this value can not be very high. The total power consumed by the AFE circuit without any input signal is displayed in Figure 24.

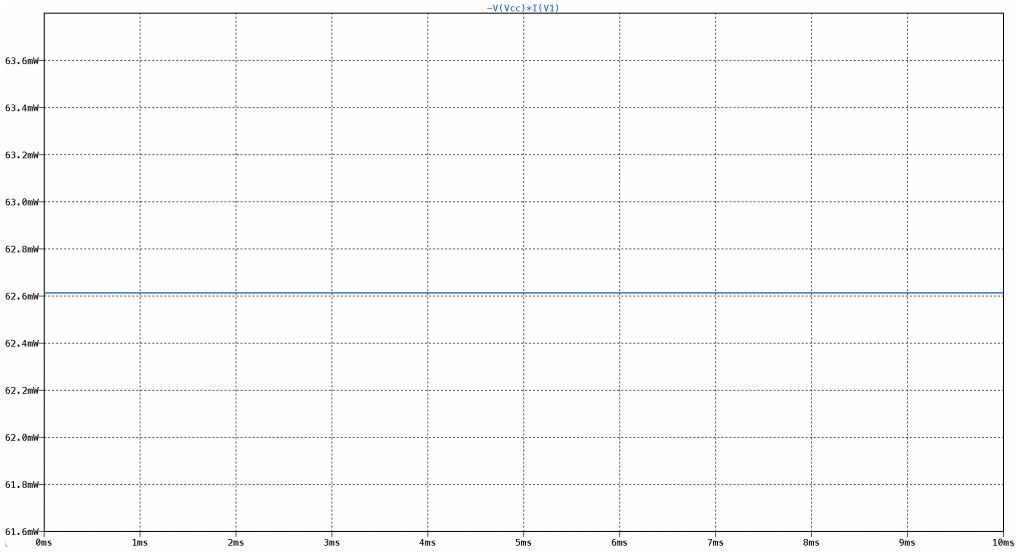


Figure 24: AFE total power consumed

Observing the graphic above, the value for the power consumed by the AFE circuit is slightly high, but the complexity and number of component of this circuit explains this fact.

4 Digital Signal Processing

In this section, it will be discussed how the digital audio signal was processed and recognized.

The digital signal processing is divided into three main parts: I2S input, audio processing, and I2S output. The I2S input is responsible for receiving the digital audio signal from the microphone. The signal processing is responsible for processing the digital audio signal, and the I2S output is responsible for sending the processed digital audio signal to the speaker. In figure 25 is shown the block diagram of the digital signal processing.

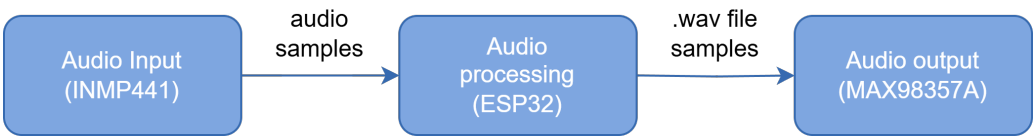


Figure 25: Block diagram of the digital signal processing.

4.1 I2S DMA Buffer

The DMA buffer (Direct Memory Access buffer) is a memory region used to transfer data between the I2S peripheral and the main memory without continuous CPU intervention. It is crucial for managing high-speed data streams, such as audio, ensuring efficient operation.

In reception (Audio Input), the DMA buffer acts as an intermediate storage area where data received by the microphone is temporarily stored before being processed by the application, a diagram of the reception process is shown in figure 26.

The process of the DMA buffer in reception is:

1. The I2S receives serialized audio data from the microphone.
2. The DMA writes this data into the buffer.
3. When the buffer is full, the audio processor (program) processes the data.

To have a continuous audio retrieval, it was implemented a circular buffer. When the buffer is full, the DMA continues to write data to the buffer, overwriting the oldest data. This way, the audio processor can continuously process the audio data without interruptions.

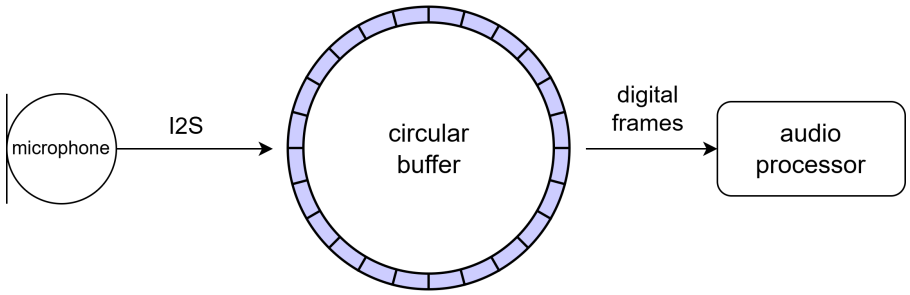


Figure 26: DMA buffer in reception.

In transmission, the DMA buffer is filled with data to be sent. The DMA reads the data from the buffer and automatically transfers it to the I2S amplifier, which transmits it to the speaker.

The process of the DMA buffer in transmission is:

1. The WAV reader (program) writes data to the buffer.
2. The DMA transfers data from the buffer to the I2S amplifier.
3. The I2S transmits the serialized data to the speaker.

A diagram of the DMA buffer transmission is depicted in Figure 27.

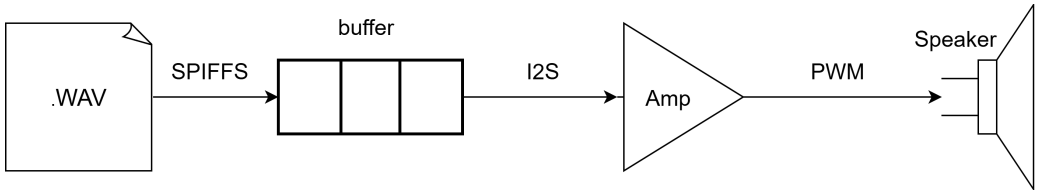


Figure 27: DMA buffer in transmission.

4.2 Audio Input

The audio input was done with a I2S MEMS (Micro-Electro-Mechanical System) breakout board (INMP441) [8] that sends the audio signal in I2S format. The I2S format is a serial protocol that is used to transmit audio data and has three lines: the bit clock (BCLK), the word select (WS), and the data line (DIN).

- BCLK: The bit clock is the serial clock that is used to synchronize the data transmission from a peripheral.
- WS: The word select line indicates if the data is sent or received from left or right channels.
- DIN: The data line carries the audio data.

A diagram of the I2S format is shown in figure 28.

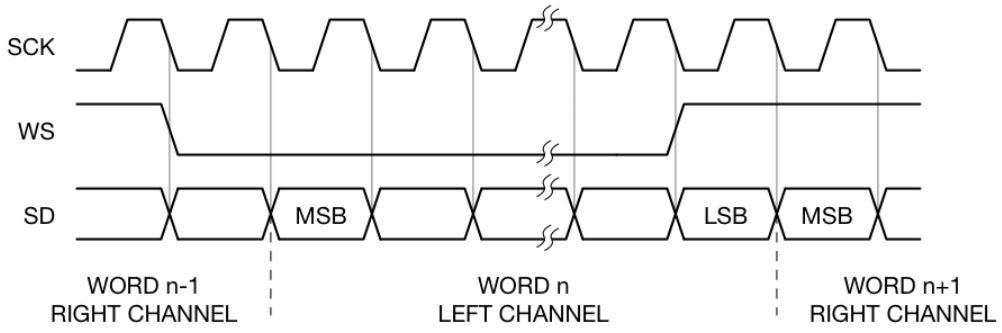


Figure 28: I2S format.

The I2S MEMS breakout board is a interesting for this application because it integrates an audio amplifier, ADC and the I2S interface. Thus, the output from the board can be fed directly into the ESP32 without using the internal ADC of the ESP32.

To configure the communication the I2S MEMS breakout board was set to record audio from right channel, 32 bits per sample and a sample rate of 16 kHz that is suitable for voice recording.

4.3 Audio Output

The audio output was obtained with a MAX98357A [9] I2S Class D amplifier that receives the audio signal in I2S format and outputs the audio signal to the speaker. The MAX98357A is a low-cost, low-power mono audio amplifier that is suitable for this application and has a built-in DAC and PWM modulator that converts the digital audio signal to an analog audio signal.

It was configured to receive the audio signal in both channels right and left, 16 bits per sample and a sample rate of 16 kHz that is suitable for voice reproduction, the two channels were mixed to output the audio signal to the speaker because in this application it was only used one speaker (mono).

The amount of power that the MAX98357A can deliver is limited by the speaker impedance it can deliver up to 1 W of power to a 8 Ω speaker with a power supply voltage of 5 V, that is what was implemented in this project.

To maximize the gain of the MAX98357, a pull down resistor of 100 kΩ was connected to the gain pin of the amplifier.

The class D amplifiers are also known as switching amplifiers because they operate by rapidly switching the output transistors on and off, they output a modulated signal that switches between the positive and negative power rails. The signal is passed to a speaker to recover the audio signal. This makes the amplifier very efficient as the transistors are only dissipating power when they are switching from high to low and low to high.

4.4 Audio Processor

The audio processing was done with the ESP32 microcontroller that receives the digital audio signal from the I2S MEMS breakout board, processes the audio signal and sends the processed to the prediction model presented below.

The audio processor has to recreate the same process that was used for the training data and then convert the audio signal to a spectrogram.

The process involved was:

- Find the minimum and maximum values of the samples.
- Normalize the samples.
- Copy the samples to the FFT input buffer.
- Apply a hamming window to the samples.
- Perform the FFT.
- Extract the energy in which frequency bin.
- Average pooling process similar to the one used in the training data.
- Take the logarithms of the values that give reasonable values to feed into the model.

The spectrogram to a mel spectrogram that is a spectrogram where the frequencies are converted to the mel scale that is a perceptual scale of pitches that is based on how humans perceive sound. The audio processor has to convert the mel spectrogram to a MFCC (Mel-frequency cepstral coefficients) that are coefficients that collectively make up an MFC. The MFCC is a representation of the short-term power spectrum of a sound that is based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

5 Digital Audio Recognition

To detect and recognize commands from the audio signal, a mechanism with to types of recognition was designed: recognition with a machine learning model implemented in the ESP32 using TensorFlow Lite and recognition outsourcing the process to the Wit.ai platform. A diagram of the voice recognition is shown in figure 29.

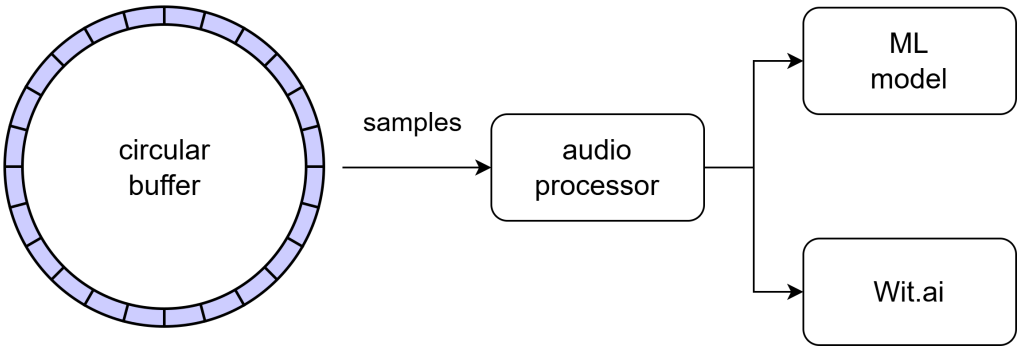


Figure 29: Voice recognition block diagram.

5.1 Model and TensorFlow Lite

The machine learning model was trained with the Google Speech Commands dataset that contains 65,000 one-second long utterances of 30 short words. The model was trained to detect the presence of a word in the audio signal.

The training was done with a convolutional neural network of an open sourced Jupyter notebook, that predicts the presence of a word in the audio signal by analyzing the spectrogram of it, in Figure 30 is an example of the spectrograms generated for the word selected. Then the trained model was converted to C code to be implemented in the ESP32.

Figure 30: Spectrogram of the word selected.

The TensorFlow Lite library was used to implement the voice activity detection algorithm of the machine learning model. The TensorFlow Lite library is a lightweight version of the TensorFlow library that is optimized for mobile and embedded devices.

This solution has the advantage of being able to detect the presence of a word in the audio signal without the need to send the audio signal to the cloud, ensuring the privacy of the user. In other hand, the processing power of the ESP32 is limited, and the model can be heavy to run in the ESP32, diminishing the accuracy of the model and the number of tasks that the ESP32 can perform.

5.2 Interact with Wit.ai

The Wit.ai [10] platform allows the development of voice recognition applications that can be used in the project. It has a REST API that allows the application to send audio data to the platform and receive the recognized text. The Wit.ai platform uses machine learning algorithms to recognize the speech and convert it to text.

To interact with the Wit.ai platform, the application has to send a POST request to the platform with the audio data. The platform will process the audio data and return the recognized text. To avoid having to buffer the entire audio sample in memory a chucked upload of the data was performed.

A connection to wit.ai is created and then chunks of data are uploaded until it collects sufficient audio to capture the user's command. After that the results are returned from the API and the important relevant information is extracted (intent, entity). The intents and entities trained in the wit.ai platform are shown in Figure 31.

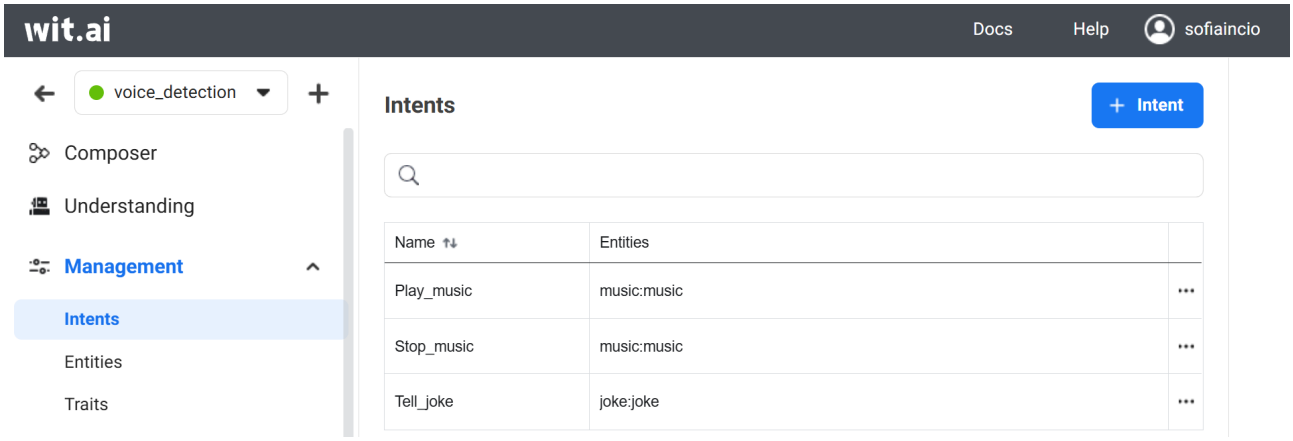


Figure 31: Intents and entities trained in the wit.ai platform.

6 User Interaction

In this section the interaction between the user and the voice activity detecting (VAD) system will be discussed. This component of the project aims to deliver information about the system and provide the user with an interface to change system options. Three primary tools were implemented to achieve this goal: a Graphical User Interface (GUI) for direct interaction, a Grafana dashboard for real-time data visualization, and an additional Home Assistant

dashboard designed to extend functionality and integrate with other Internet of Things (IoT) ecosystems. The information exchanged between the MCU and the user is displayed in Table 1 and Table 2.

Table 1: MCU data

Data Provided	Values
LED status indicator	True/False
Speaker Volume	$[0, 100]$
MCU State	Idle,Listening,Playing
Time until sleep	$[0, 120]$ s

Table 2: System Options

Option	Values
LED status indicator	True/False
Speaker Volume	$[0, 100]$
Start Sleep	N/A
Time until sleep	$[0, 120]$ s
MQTT	On/Off
No Sleep mode	True/False

6.1 Wireless Communication

Any wireless communication was handled by MQTT (Message Queuing Telemetry Transport) protocol. MQTT is a lightweight messaging protocol designed for publish/subscribe communication. It operates on a client-server architecture where devices (clients) communicate by publishing messages to specific topics on a central broker. Other clients subscribe to these topics to receive the messages.

This makes this method of communication suitable for this project since there is no need for low latency communication and makes the program relatively lightweight and ensures flexibility and scalability in IoT networks.

6.1.1 Grafana

Grafana is an open-source platform for monitoring and visualizing time-series data. Grafana was integrated as the visualization component of the user interaction system to provide users with a clear and intuitive interface for data analysis.

It is important to note that not all values are numeric, so in order to display this values to grafana the conversion to numeric values is shown in Table 3.

Table 3: MCU Data in Grafana

Data	Value	Grafana
LED status indicator	True/False	0,1
MCU status	Idle,Listening,Playing	0,1,2

Grafana by itself it is not able to receive and store the MQTT data, therefore there is a need to use Telegraf to translate MQTT messages to Flux and influxDB to store and send the data to Grafana, this architecture is shown in Figure 32.

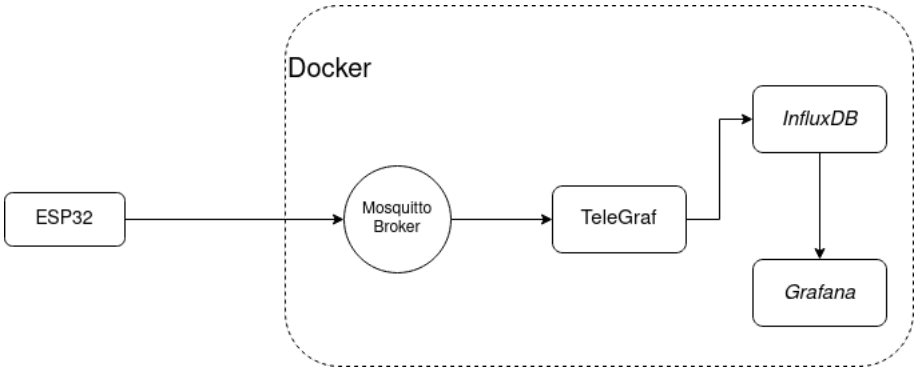


Figure 32: Grafana data flow architecture [2].

The final dashboard with real data sent by the Esp32 can be seen in Figure 33.

Figure 33: Grafana Dashboard.

6.1.2 Home Assistant

Home Assistant is an open-source home automation platform designed to provide centralized control and monitoring of smart devices within a connected environment. This tool allows advanced automations and integrations between different IoT systems and highly customizable

dashboards. This means that in the future with other IoT devices, that are not necessarily made to be used together, can be integrated.

Two important concepts for Home Assistant are sensors and automations. In Home Assistant, sensors are entities that provide data about the environment or the state of connected devices, this is the data acquisition aspect. Automations, in this context, enable actions to be triggered automatically based on specific conditions, events, schedules or sensor state.

Although not originally specified in the lab work requirements, a Home Assistant integration and a dashboard was developed as part of this project to enhance the system's functionality. This option was considered a better option since grafana is used to analyze and display data over time while Home Assistant allows for deeper automation and integration with other IoT systems, while still allowing visualization of data evolution over time.

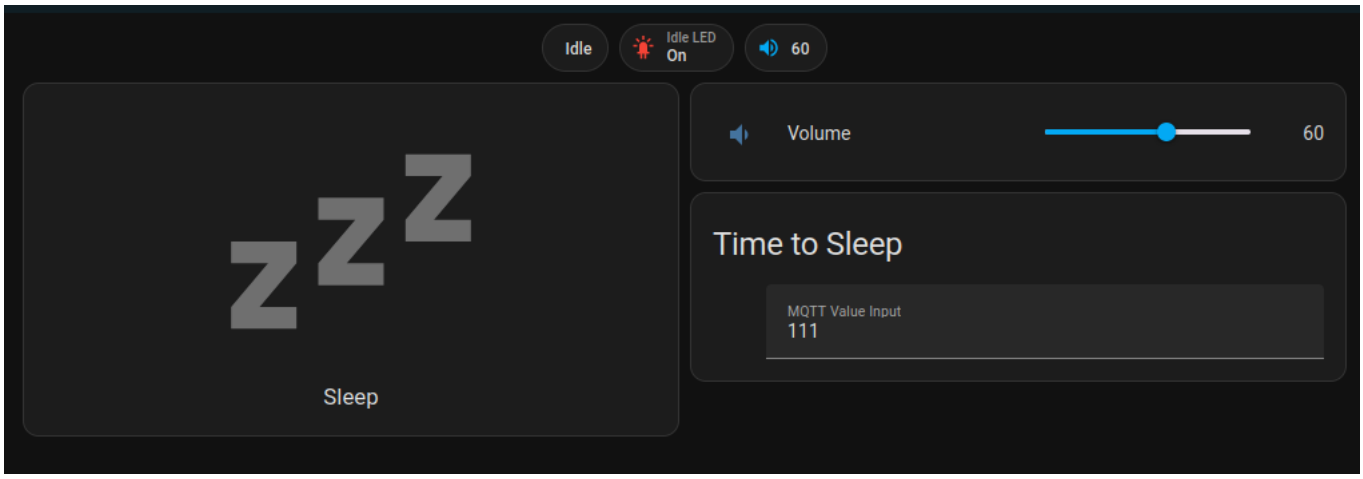


Figure 34: Home Assistant Dashboard.

The dashboard shown in Figure 34 is more intuitive and allows both system visualization and control. Another big advantage is the fact that Home Assistant has a smartphone application.

Since this was not part of the labwork requirements, the Home Assistant implementation will be explain in greater detail.

In the Home Assistant ecosystem most of the dashboard's cards can be build and customized through the graphical interface the more complex configurations is through the yaml configuration files.

There is only one yaml file that needs to be edited for this project, the configuration.yaml, but in order to make the code easier to read two other files were edited and or created.

```
automation: !include automations.yaml
mqtt: !include mqtt.yaml

input_number:
  esp_volume:
    icon: mdi:volume-medium
    name: Volume
    initial: 30
    min: 1
```

```
    max: 100
    step: 1

input_text:
  mqtt_value_input:
    name: MQTT Value Input
    initial: ""
    max: 255

input_boolean:
  led_mode:
    name: LED Mode
    initial: off
```

The previous code shows the configuration.yaml, it first includes the two other files that were edited in order to add the VAD functionality and then creates the volume slider, the text input card and a boolean for the LED.

```
    - sensor:
        - name: "Status"
          state_topic: "ems/g1/t1/status"
    - sensor:
        - name: "Volume"
          state_topic: "ems/t1/g1/volume/state"
    - sensor:
        - name: "Idle LED"
          state_topic: "ems/t1/g1/enableled/state"
```

The mqtt.yaml file creates three sensors, this way Home Assistant is able to receive data from the MCU and use it as sensor.

```
- alias: 'Send Volume'
  trigger:
    - platform: state
      entity_id: input_number.esp_volume
  action:
    - service: mqtt.publish
      data:
        topic: ems/t1/g1/volume/set
        payload: "{{ states('input_number.esp_volume') | int }}"
        qos: 0
        retain: false
      target: {}

- alias: 'Validate MQTT Input'
  trigger:
    - platform: state
```

```
      entity_id: input_text.mqtt_value_input
  action:
    - choose:
      - conditions:
        - condition: template
          value_template: >
              {{ states('input_text.mqtt_value_input').isdigit() and states('input_text.mqtt_v
        sequence:
          - service: mqtt.publish
            data:
              topic: ems/t1/g1/timesleep/set
              payload: "{{ states('input_text.mqtt_value_input') }}"
              qos: 0
              retain: false
            target: {}
      - conditions: []
        sequence:
          - service: notify.persistent_notification
            data:
              message: "Invalid input! Please enter a positive integer."

- alias: "LED On"
  trigger:
  - platform: state
    entity_id: input_boolean.led_mode
    to: "on"
  action:
  - service: mqtt.publish
    data:
      topic: ems/t1/g1/enableled/set
      payload: "1"
    target: {}
- alias: "LED Off"
  trigger:
  - platform: state
    entity_id: input_boolean.led_mode
    to: "off"
  action:
  - service: mqtt.publish
    data:
      topic: ems/t1/g1/enableled/set
      payload: "0"
    target: {}
```

The automation.yaml is the more complex file, the first alias detects changes in the volume bar and send a MQTT message to the MCU. The second detects if the input in the text box is a number if it is sends the MQTT message, if not sends a notification saying "Invalid input! Please enter a positive integer.". The rest of the file will ensure the correct value of the LED

boolean and according to that boolean will send a MQTT message to turn the activity LED on or off.

As a last example of how integrations can be made, the sleep button is a basic button created through the GUI, it's configurations can be viewed in Figure 35.

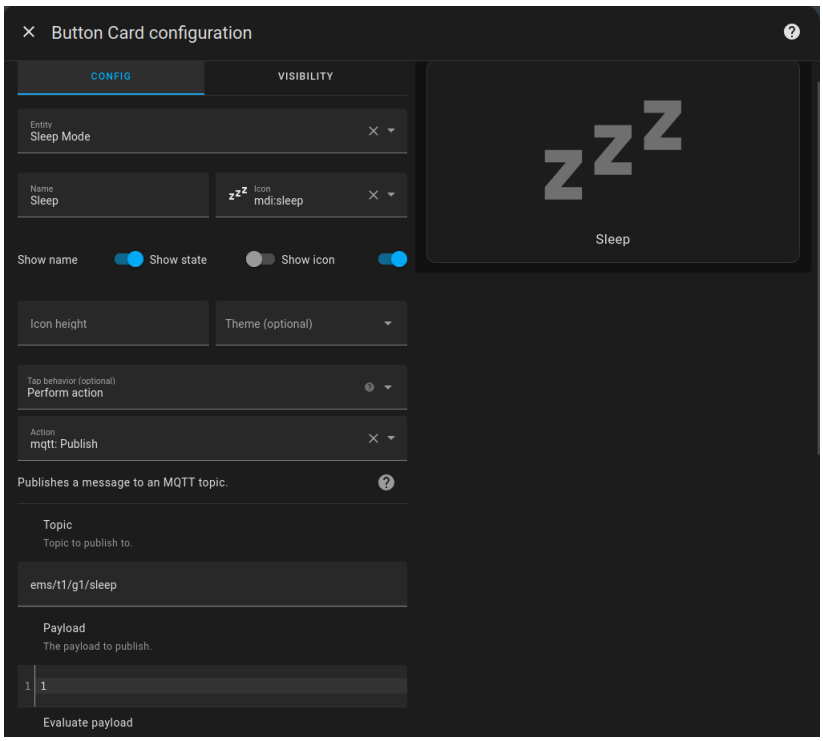


Figure 35: Sleep card configuration.

6.2 Wired Communication

The GUI was developed to serve as a user-friendly interface for interacting with the system over the wired connection. A wired connection is important to ensure a reliable with low latency and direct interaction to the VAD system. This way the system is still usable when there is no internet connection. The wired communication was all performed through serial communication.

6.2.1 Graphical User Interface (GUI)

This GUI provides a user-friendly and interactive platform for controlling and configuring the system. To ensure cross-platform compatibility and ease of development, the GUI was implemented using Python. The interface was designed with a focus on clarity, responsiveness, and functionality.

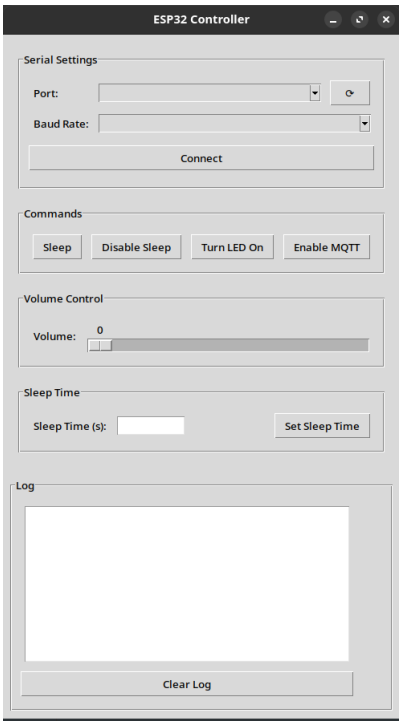


Figure 36: Graphical User Interface.

As seen in Figure 36, the GUI is divided in five sections.

Serial settings, here the user is able to select the communication port, this allows the use of multiple MCU's and to disconnect and reconnect in another port, which is useful for debugging.

Commands, this section contains four buttons that send a non numeric command, according to Table 2.

Volume Control, a simple volume slider that sends its value when changed.

Sleep Time, a text box that test if the input is valid before sending.

Log, is just a text box that shows information about the system, such as errors in communication.

7 Implementation

This section presents the implementation of the system, including the hardware and software, as well as the testing of the analog and digital interfaces. The system was developed in modules, with the hardware and software components developed independently and later integrated. An imaged of the systems implementation is depicted in Figure 37.

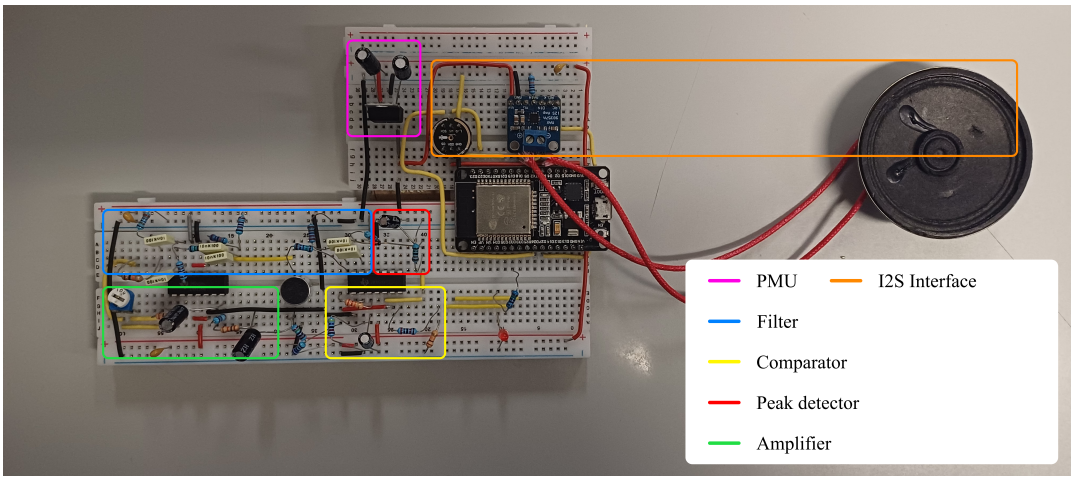


Figure 37: System Implementation.

7.1 Digital Audio Software

7.1.1 Implementation

To implement in software the digital audio interface, the ESP32 was used, as it has the necessary peripherals to handle the audio data and the WiFi communication. The ESP32 was programmed using the PlatformIO.

To perform all the tasks required by the system in the digital audio interface a main task called Listen was created. This task is continually running and alternating between to states: Detect wake up word (detectWord) and Recognize command (recognizeCmd). A flow chart of the Listen task can be seen in Figure 38.

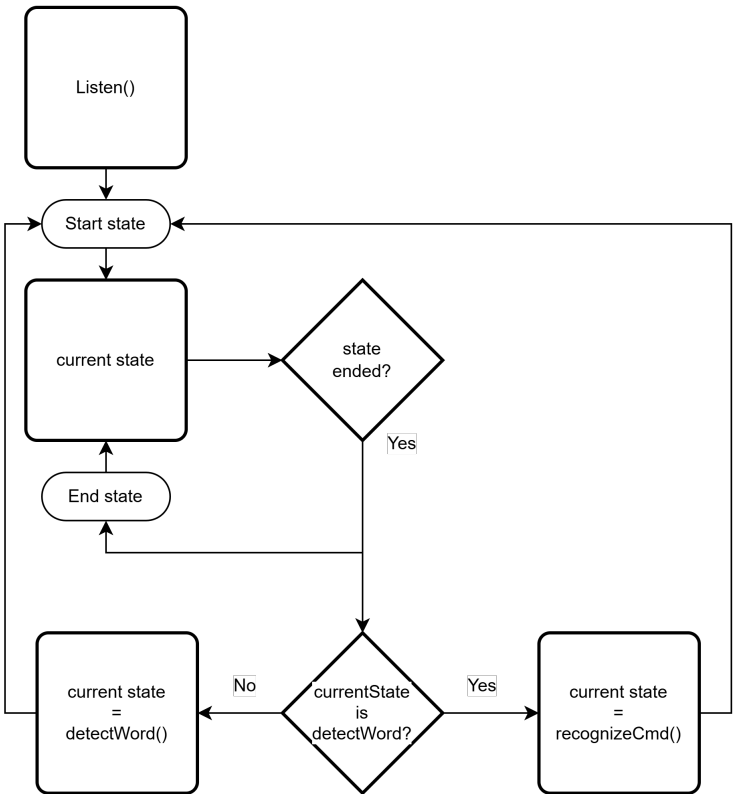


Figure 38: Listen Task Flowchart.

The code was divided into two main parts, the Wake up word detection (detectWord) and the command recognition (recognizeCmd).

The wake up word detection was implemented using a I2S input sampler to fetch the data and a Ml model to predict if the audio retrieved corresponds to the wake up word. A flowchart of the algorithm implemented is shown in Figure 39. This implementation used the word marvin as the wake up word to compare with the model trained. The result of the comparison with the model is a value between 0 and 1, where 0 means that the audio retrieved is not the wake up word and 1 means that the audio retrieved is the wake up word. If the value is above the 0.9 threshold, the system goes to the recognize command part.

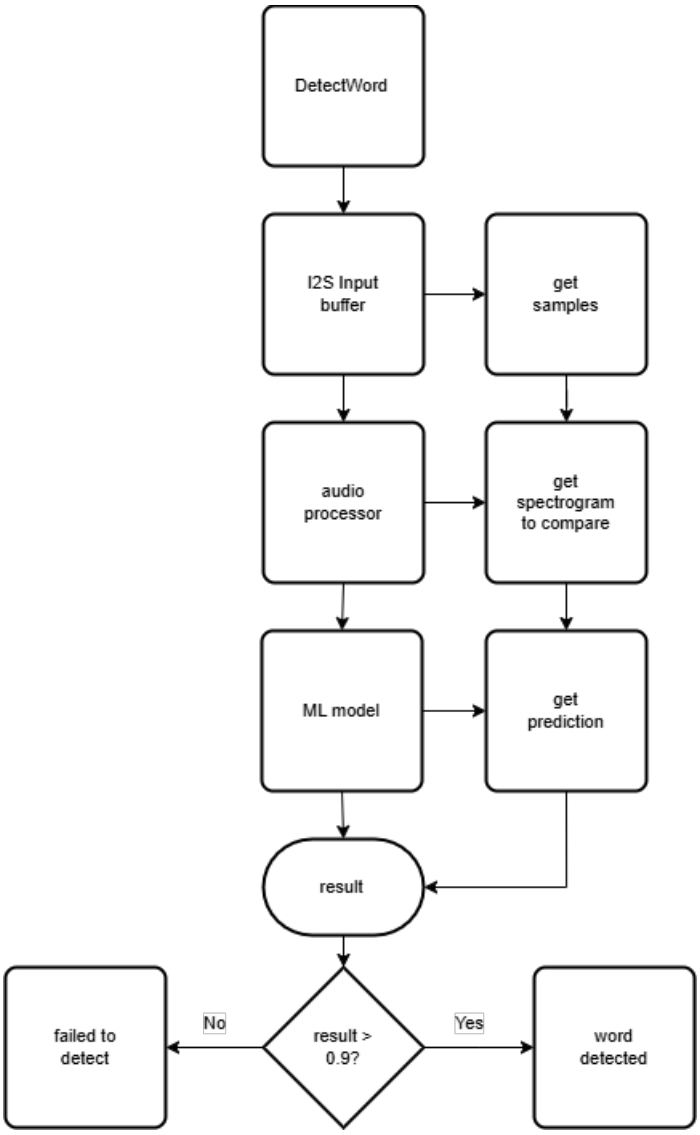


Figure 39: Detect Word Flowchart.

The detected command part, that is triggered after the wake up word is detected, was implemented using the same I2S input sampler to fetch the data and connection to the wit.ai API to recognize the command. In the wit.ai API, a model was trained to recognize the commands that the system can perform. A POST request is made to the wit.ai API with the audio data retrieved and the API returns with a the command recognized in a JSON format. The command recognized is then processed, by the intent processor, and the system performs the action associated with the command.

A flowchart of the algorithm implemented is shown in Figure 40.

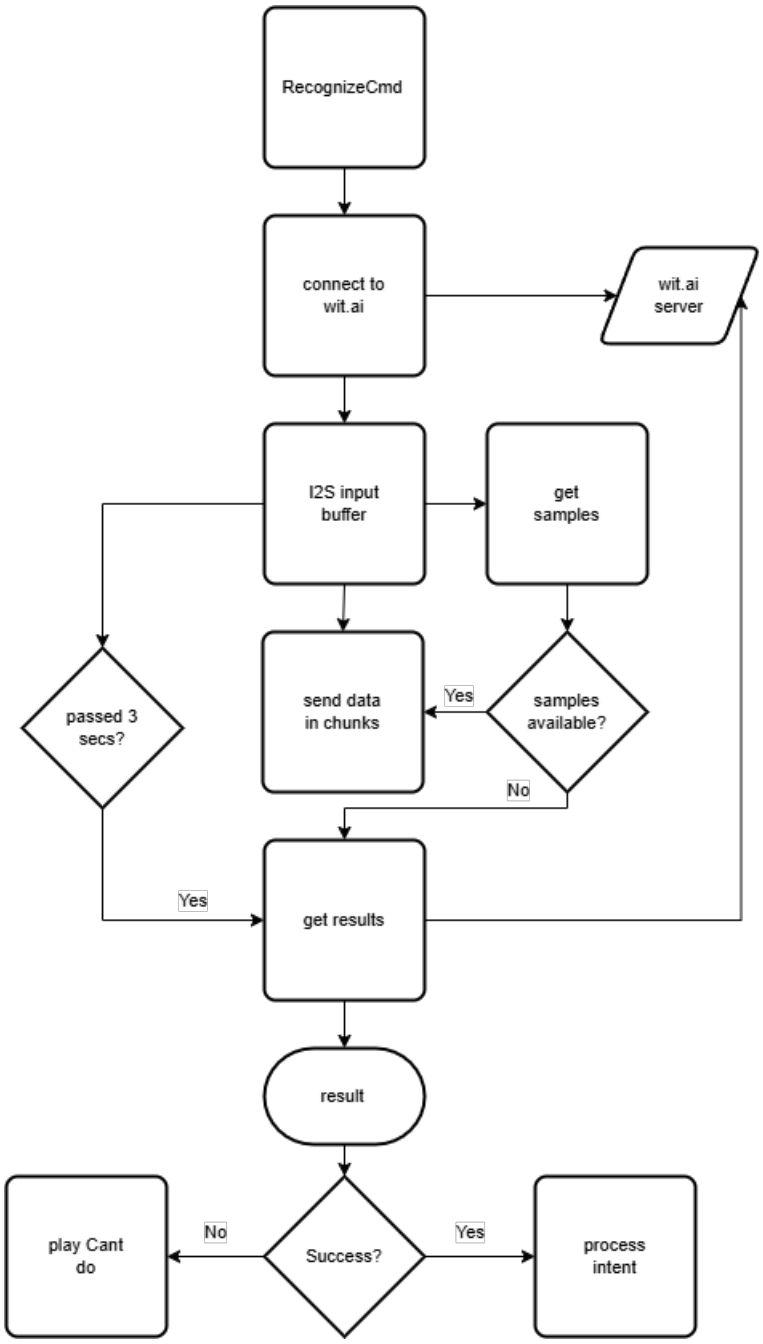


Figure 40: Recognize Command Flowchart.

The intent processor described in the Command Recognition flowchart is a function that receives the command recognized and performs the action associated with the command. The actions that the system can perform are: play music, stop music and tell a joke. A flowchart of the intent processor is shown in Figure 41. The DMA buffer for transmission block was described in the section Digital Signal Processing.

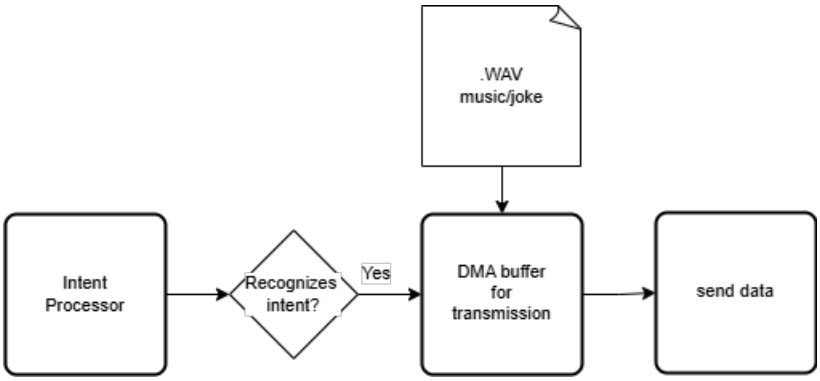


Figure 41: Intent Processor Flowchart.

7.1.2 Testing

The digital audio interface was tested using the ESP32 and the I2S microphone. The test consisted of saying the wake up word and give all the possible commands implemented. The test was successful, the system was able to recognize the wake up word and perform the action associated with the command. The test was performed with the wake up word marvin and the commands "play music", "stop music" and "tell a joke".

7.2 Software integration

Running on the ESP32 microcontroller, the firmware manages the voice activity detection, system control, and communication tasks, with external platforms for data visualization and interaction. A flowchart representing the code running on the MCU can be seen in Figure 42.

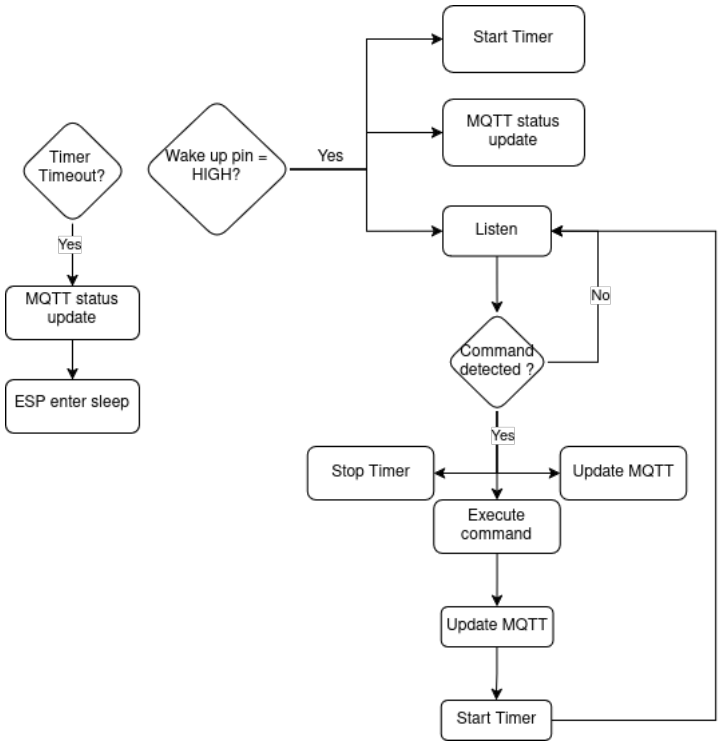


Figure 42: Firmware Flowchart.

After validating and successfully testing the user interaction and voice detection code independently, the two were merged into a single firmware following the flowchart shown in Figure 42. This presented unforeseen problems, since the voice detecting code is already pushing the limits of what the Esp32 can do, when merged with code that also uses WiFi, which is a process intensive task, the Esp32 was not able to handle everything together.

To address this, audio processing tasks were pinned to Core 1 while WiFi operations were confined to Core 0, leveraging the dual-core architecture of the ESP32. Additionally, the I2S peripheral, critical for audio data handling, was configured with a higher interrupt priority to ensure timely processing. Despite these optimizations, the system's performance remained inadequate, indicating that the ESP32's computational resources are insufficient for handling both tasks simultaneously at the required level of performance.

7.3 Analog Front End

To help with the calibration to reach the correct level of gain for the implementation, the second resistor in the Pre-Amp circuit, R_{b2} was changed to a potentiometer, thus, through testing, the correct value for the final gain of the Pre-Amp was reached, which guarantees of the human voice detection in any environment.

7.3.1 Test

To test and verify the implemented filter circuit, an analysis, equal to the AC analysis made during the simulations, was carried out, with the help of the ADALM 2000, the resulting frequency response diagram can be seen in Figure 43.

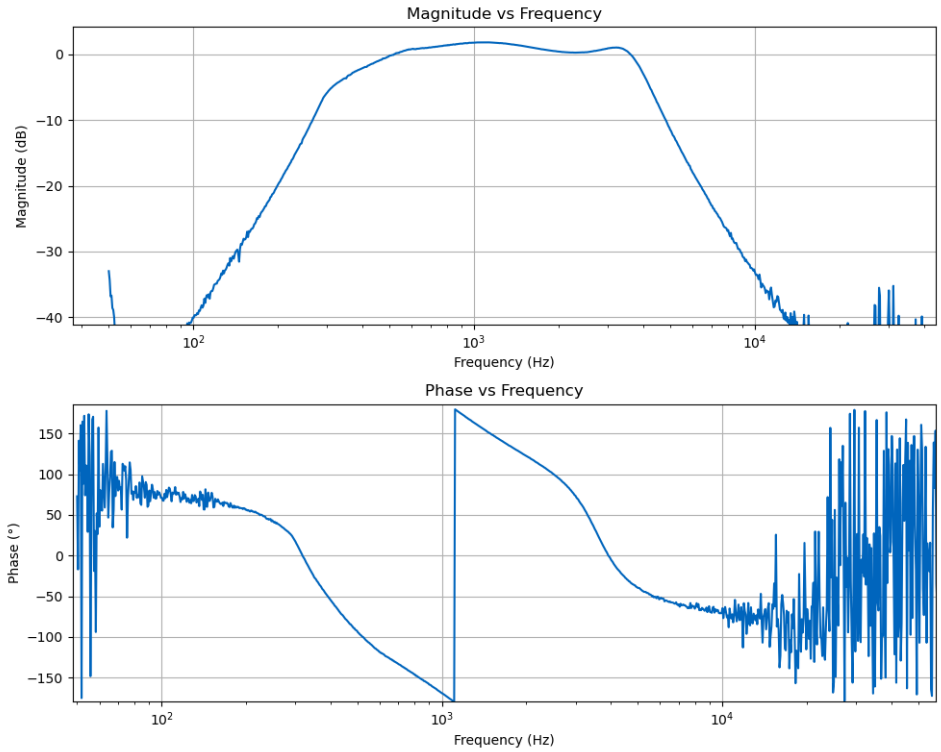


Figure 43: Implemented Filter Bode Diagram.

The response above presents the expected behavior, with just a small difference in the first Cutoff Frequency, where the overshoot how appeared before disappeared. This change will not cause any problems for the performance of the Bandpass Filter.

Finally, to test and verify the complete Analog Front End circuit, an oscilloscope was connected to the output of the filter and the comparator. The final results of this test are exposed in Figure 44.

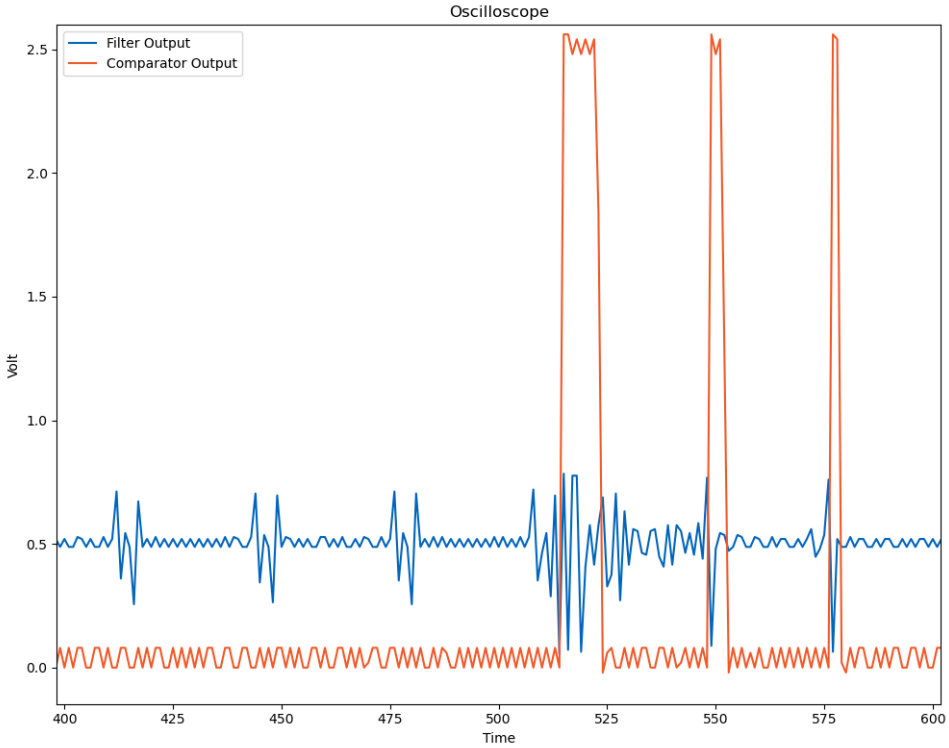


Figure 44: Analog Front End final test.

Both signals have different scales and reference points, because of this fact, the output of the filter does not appear with the correct values in the graphic above.

With this information in mind, form this test, it can be concluded that all stages of the Analog Front End worked as expected, from the Pre-Amp, which was able to amplify the signal generated by the human voice in moment 510 to 525 to a value high enough to ensure the correct functioning of the rest of the circuit, moving on to the Sixth Order Bandpass Filter, where only the signal expected passed to the final stage of the AFE, expected some disturbances, which did not cause any false wake up signal to be read by the ESP32, since these were not superior to the value of the Comparator circuit.

8 Conclusion

The Voice Activity Detector (VAD) project successfully reached the desired goals, detecting and responding to voice activity and send information with MQTT, even though it was not able to integrate all functions simultaneously due to lack of processing power.

With Home Assistant, Grafana and GUI this project delivered a versatile user interaction methods to deliver a comprehensive voice-activated control solution.

The Analog Front End proved effective in detecting audio peaks within the human voice frequency range, ensuring reliable wake-up signals for the microcontroller. An important addition to the AFE was a potentiometer that regulates the microphone gain, this allowed for quicker testing and made for more modular system. Another really important aspect of the AFE was the use of Multiple Feedback Topology in the filter, making the filter moree robust

to component variations.

9 Future Work

9.1 Microcontroller upgrade

As explain in section 7.2, the ESP32 is not powerful enough to handle this project requirements, hence, future work should explore a more powerful MCU capable of handling the increased computational and memory demands. Possible candidates could be the ESP32-S3,STM32F746 or Teensy 4.1 .

9.2 Digitally controlled noise threshold

In order to improve the performance of the system, making it more flexible to its environment, a digitally controlled noise threshold could be a useful feature to have.

One solution to implement this feature, is to dynamically change what the comparator reference voltage. This solution presents a few issues, since the MCU goes to sleep.

The first solution is to have a capacitor attached to the inverting input of a OpAmp, in order to hold a voltage set by the MCU Digital to Analog Converter (DAC), as seen in Figure 45. This might not work when the MCU enters sleep, the DAC goes to zero volt. Another shortcoming of this solution is that the capacitor will discharge over time.

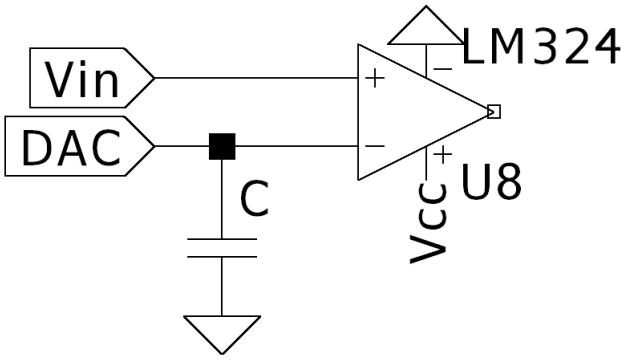


Figure 45: Comparator with capacitor.

The second solution is to have N pins of the MCU to act as a DAC, where all bit values are maintained with latches, this solution is more complex in terms of implementation, but it is more robust. The circuit that implements this solution is presented in Figure 46.

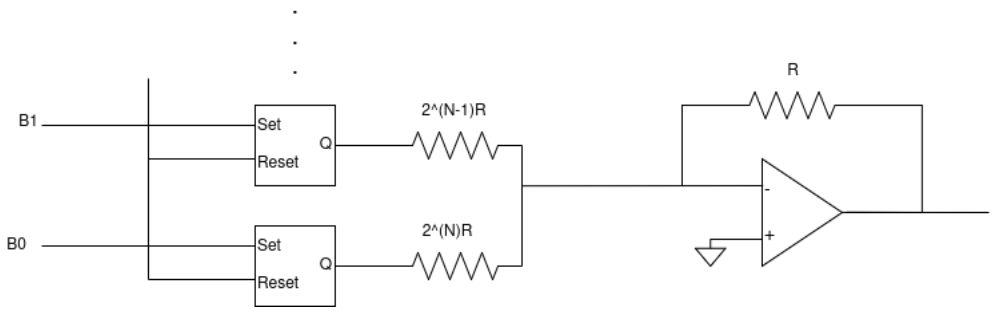


Figure 46: DAC with latched bits.

9.3 PCB implementation

The PCB implementation of this project is a important step to ensure the system's reliability and robustness. The PCB should be designed to minimize noise and interference, and to ensure that the system is as compact as possible.

References

[1] J. P. Oliveira, "Final project - voice activity detector," 2024.

[2] L. Filipe, "Ems final project," 2024, accessed: 2024-10. [Online]. Available: https://github.com/Coelhomatias/ems

[3] T. Instruments, "Lm324," https://www.ti.com/lit/ds/symlink/lm324.pdf?ts=1729155971844&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM324, 2024.

[4] C. Devices, "Cma-4544pf-w," https://moodle.fct.unl.pt/pluginfile.php/775891/mod_page/content/10/Microphone_cma-4544pf-w.pdf, 2020.

[5] J. Lewis, "Understanding microphone sensitivity," https://moodle.fct.unl.pt/pluginfile.php/775891/mod_page/content/10/10.1.1.309.4153.pdf?time=1730689009210, 2012.

[6] T. Kugelstadt, "Active filter design techniques," https://www2.seas.gwu.edu/~ece121/Spring-11/filterdesign.pdf, 2001.

[7] E. Systems, "Esp32 series datasheet," https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, 2024.

[8] T. InvenSense, "Inmp441 datasheet," https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf, 2024.

[9] A. Devices, "Max98357a datasheet," https://www.analog.com/media/en/technical-documentation/data-sheets/MAX98357A-MAX98357B.pdf, 2019.

[10] Wit.ai, "Wit api documentation," 2024. [Online]. Available: https://wit.ai/docs/http/20240304/