AUDIO CLASSIFIER

*

Sourav Pramod
3rd year, Computer Science and Engineering
Amrita School of Computing, Amrita Vishwa Vidyapeetham
Ettimadai, Coimbatore-641112

S Anandakrishnan
3rd year, Computer Science and Engineering
Amrita School of Computing, Amrita Vishwa Vidyapeetham
Ettimadai, Coimbatore-641112

Vivek S
3rd year, Computer Science and Engineering
Amrita School of Computing, Amrita Vishwa Vidyapeetham
Ettimadai, Coimbatore-641112

***Abstract*—In the realm of human-computer interaction, audio classification has emerged as a pivotal technology, enabling seamless communication between users and devices. This project focuses on the development of an audio classifier using the Raspberry Pi Pico RP2040 microcontroller, an ADMP401 microphone, and a simple LED indicator. The objective is to create a user-friendly system that responds to vocal commands, specifically activating an LED when the user utters ”yes” and deactivating it upon hearing ”no.”**

I. Introduction

In recent years, the intersection of embedded systems and artificial intelligence has given rise to innovative applications that enhance user experience and interaction with technology. Audio classification, a subset of signal processing, plays a crucial role in enabling devices to comprehend and respond to human speech. This project delves into the realm of audio classification using accessible components, specifically the Raspberry Pi Pico RP2040 microcontroller and the ADMP401 microphone, to create a hands-free user interface.

The Raspberry Pi Pico RP2040, known for its versatility and performance, serves as the brain of the system. Its programmability and GPIO capabilities make it an ideal choice for integrating various sensors and actuators seamlessly. The ADMP401 microphone, chosen for its sensitivity and wide frequency response, becomes the ears of our system, capturing audio signals with precision.

The primary goal of this project is to design a practical application that showcases the potential of audio classification. The LED indicator, responding to affirmative and negative vocal cues, serves as a tangible output, demonstrating thereal-time responsiveness of the system. This interactive setup has implications in diverse fields, including home automation, accessibility solutions, and voice-controlled applications.

As we explore the intricacies of audio signal processing, we aim to provide a foundational understanding of how these technologies can be harnessed for user-friendly interfaces. By combining hardware and software components, this project exemplifies the synergy between embedded systems and artificial intelligence, offering a glimpse into the future of intelligent and responsive devices.

II. PRIMARY OBJECTIVES

- Audio Signal Processing
- Vocal Command Recognition
- Hardware Integration
- User Interaction
- Expandability and Modularity
- System Reliability
- Educational Outreach
- Demonstration of Practical Applications

III. Hardware requirements

-
- ADMP401 Microphone
- LED Indicator
- Wires and Breadboard
- USB Cable
- Computer

IV. SOFTWARE REQUIREMENTS

- Thonny Python
- Micro Python

V. METHODOLOGY OF AUDIO CLASSIFIER

A. Hardware Connection

Connect the Raspberry Pi Pico to your computer using a USB cable for power and programming.

Connect the ADMP401 microphone to the Raspberry Pi Pico using jumper wires.

Connect the AUD pin of the ADMP401 microphone to pin 32 on the Raspberry Pi Pico.

Connect the VCC pin of the ADMP401 microphone to pin 36 on the Raspberry Pi Pico.

Connect an LED to the Raspberry Pi Pico using a current-limiting resistor.

Connect the anode (longer lead) of the LED to pin 25 on the Raspberry Pi Pico.

Connect the cathode (shorter lead) of the LED to the ground (GND) pin on the Raspberry Pi Pico.

Ensure the Raspberry Pi Pico is powered either through the USB connection to your computer or an external power supply.

Component	Raspberry Pi Pico Pin	Connection
Raspberry Pi Pico	N/A	Connected to USB for power and programming
ADMP401 Microphone	AUD Pin (Microphone)	Connected to Pin 32
ADMP401 Microphone	VCC Pin (Microphone)	Connected to Pin 36
LED Indicator	Anode (Longer lead)	Connected to Pin 25
LED Indicator	Cathode (Shorter lead)	Connected to GND

B. Communication Protocol

The communication in this project is mainly internal to the Raspberry Pi Pico. The components communicate through GPIO pins, and the logic is controlled by the programmed code running on the microcontroller.

C. Explanation

Connect the Raspberry Pi Pico to your computer using a USB cable for both power and programming. Connect the microphone AUD (analog audio) pin to GPIO Pin 32 on the Raspberry Pi Pico. Connect the microphone VCC (power) pin to GPIO Pin 36 on the Raspberry Pi Pico. Connect the microphone GND (ground) pin to the Ground (GND) pin on the Raspberry Pi Pico. Connect the anode (longer lead) of the LED to GPIO Pin 25 on the Raspberry Pi Pico. Connect the cathode (shorter lead) of the LED to the Ground (GND) pin on the Raspberry Pi Pico. Ensure that the GPIO pin configurations in the code match the physical connections. Double-check the polarity of the LED and connect it with the correct orientation (anode to GPIO 25, cathode to GND).

D. Code

The threshold value (500) is set to determine when the sound level detected by the microphone is considered a ”yes” command. The is yessound() function checks if the ADC (Analog to Digital Converter) reading from the microphone is above the threshold. The getadc() function is a PIO (Programmable Input/Output) state machine assembly code. It is used to efficiently read the analog signal from the microphone. The ADC pins are configured to read the analog signal. The blinkled() function toggles the LED state to create a blinking effect. It is called when the ”yes” command is detected. The main loop continuously monitors for the ”yes” command using the isyessound() function. When the ”yes” command is detected, the LED starts blinking. The program waits for the ”no” command to stop the LED blinking. The is yessound() function is used inversely in this case. When the ”no” command is detected, the LED stops blinking. The print() statements provide feedback on the current state of the program, indicating when to say ”yes” or ”no.” A delay of 2 seconds (utime.sleep(2)) is added to prevent rapid toggling of the LED and provide a brief pause after the ”no” command is detected.

Command	Functionality
`from machine import Pin, ADC`	Import necessary modules for hardware interaction.
`import utime`	Import the microsecond-level time module.
`import rp2`	Import the RP2 module for PIO state machine.
`led_pin = Pin(25, Pin.OUT)`	Configure the Pico’s onboard LED.
`microphone_pin = 32`	Configure the GPIO pin for the microphone AUD pin.
`adc_pin = 36`	Configure the GPIO pin for the microphone VCC pin.
`threshold = 500`	Define the threshold for sound detection.
`def is_yes_sound():`	Function to check if the sound is a ”yes” command.
`microphone_pin = Pin(26)`	Create a Pin object for the microphone pin.
`adc = ADC(Pin(microphone`	Create an ADC object using ADC0.
`def get_adc():`	Assembly function to get ADC readings efficiently.
`def blink_led():`	Function to blink the LED.
`while True:`	Main loop that continuously monitors for commands.
`while not is_yes_sound():`	Wait for the ”yes” command.
`blink_led()`	Blink the LED when ”yes” is detected.

`led_pin.off()`
`utime.sleep(2)`
	Stop blinking the LED when ”no” is detected.
	Add a delay to prevent rapid toggling.

E. Pseudo Code

```
# Import necessary modules
from machine import Pin, ADC
import utime
import rp2
# Configure hardware pins
led_pin = Pin(25, Pin.OUT)
microphone_pin = 32
adc_pin = 36
# Set sound detection threshold
threshold = 500
# Function to check if the sound is a ”yes”
def is_yes_sound():
return adc.read_u16() ¿ threshold
# Initialize ADC for the microphone
@rp2.asm_pio(set_init=rp2.PIO.OUT_LOW)
def get_adc():
pull()
```

```
mov(x, osr)
mov(y, isr)
label("loop")
jmp(x_dec, "loop")
jmp(y_dec, "loop")
# Create a Pin object for the microphone_pin
microphone_pin = Pin(26)
# Create an ADC object using ADC0
adc = ADC(Pin(microphone_pin))
# Function to blink the LED
def blink_led():
for _ in range(5): # Blink the LED 5 times
led_pin.toggle()
utime.sleep_ms(200)
# Main loop
while True:
# Wait for the "yes" command
print("Say 'yes' to start blinking the LED...")
while not is_yes_sound():
pass
# Start blinking the LED
print("Yes detected! Blinking LED...")
blink_led()
# Wait for the "no" command to stop blinking
print("Say 'no' to stop blinking the LED...")
while is_yes_sound():
pass
# Stop blinking the LED
print("No detected! LED blinking stopped.")
led_pin.off()
utime.sleep(2) # Add a delay to prevent rapid toggling
```

VI. THONNY PYTHON

Thonny is an Integrated Development Environment (IDE) specifically designed for Python. It provides a user-friendly environment for writing, running, and debugging Python code. Thonny is suitable for beginners as it simplifies the process of setting up and managing Python environments.its Key features is mentioned below:

- Built-in Python interpreter.
- Simplified package management.
- Integrated debugger.
- Code completion and syntax highlighting.
- Supports virtual environments.
- Designed for educational use and ease of learning Python.

VII. MICROPYTHON

MicroPython is a lean implementation of the Python 3 programming language that is optimized for microcontrollers and embedded systems. It brings the simplicity and readability of Python to resource-constrained environments. MicroPython is designed to run on microcontrollers with limited resources, making it suitable for Internet of Things (IoT) devices.The key features are mentioned below:

- Optimized for microcontrollers and embedded systems.
- Subset of the Python 3 language.
- Small memory footprint.
- Includes a REPL (Read-Eval-Print Loop) for interactive development.
- Supports GPIO (General Purpose Input/Output) for hardware interaction.
- Enables Python programming on devices with limited resources.

VIII. CONCLUSION

In conclusion, the audio classifier project successfully demonstrated the integration of hardware and software components to create a simple yet effective system. Leveraging the Raspberry Pi Pico RP2040, the ADMP401 microphone, and an LED indicator, the project showcased the implementation of a sound-responsive device capable of interpreting user vocal commands.

The use of MicroPython on the Raspberry Pi Pico allowed for the seamless integration of Python programming into the embedded system. This choice not only facilitated rapid prototyping but also showcased the adaptability of Python even in resource-constrained environments.

The core functionality of the project revolved around sound detection, where a predefined threshold determined whether a user's vocalization constituted a "yes" or "no" command. The LED indicator provided visual feedback, blinking in response to a "yes" command and stopping when a "no" command was detected.

Through the course of the project, hardware connections were established, code was developed, and the system's behavior was tested. The implementation successfully illustrated the principles of hardware interaction, analog-to-digital conversion, and real-time decision-making in response to external stimuli.

While this project serves as a basic example, it lays the groundwork for further exploration and expansion. Future iterations could involve more sophisticated machine learning models for improved command recognition, additional sensors for enhanced input capabilities, or integration with other IoT devices for a broader range of applications.

In essence, the audio classifier project provides a hands-on experience in combining hardware and software elements, showcasing the potential of Python and MicroPython in the realm of embedded systems. As technology continues to advance, projects like these contribute to the exploration and understanding of the possibilities within the intersection of hardware and software development.

ACKNOWLEDGMENT

We would like to express our sincere appreciation to all those who contributed to the successful development and demonstration of the audio classification system on an ARM Cortex-M4 microcontroller, as showcased in Project.

Foremost, we extend our gratitude to Vijay Kumar Sir, our advisor, for their invaluable guidance, support, and expertise throughout the project. Their insights played a pivotal role in shaping the direction and success of this endeavor.

Lastly, we express our gratitude to our friends, family, and the wider scientific and research community for their unwavering support and contributions to the success of this project.

This project exemplifies the collaborative efforts of all involved, and we are thankful for the collective contributions that made it possible.

References

[1] Project 2: Demo of audio classification on any ARM-cortex m4 microcontroller.

https://blog.tensorflow.org/2021/09/TinyML-Audio-for-everyone.html

https://www.hackster.io/hlsw/pico-wake-word-1e2372

TinyML Keyword Detection (Audio) for Controlling RGB Lights.