TEXT TO SPEECH SYNTHESIS USING ESP32

A MINI PROJECT REPORT

Submitted by

PUSHPA RANI S 9517202109041

VITHYA SRI R 9517202109057

in

19AD752 – INTELLIGENT SYSTEMS FOR IOT LABORATORY

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE
(AUTONOMOUS)
SIVAKASI
OCTOBER 2024

MEPCO SCHLENK ENGINEERING COLLEGE,

SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **“Pushpa Rani S 9517202109041, Vithya Sri R 9517202109057”** for the Mini project titled **“Text to Speech Synthesis using ESP32 ”** in 19AD752–Intelligent Systems for IOT Laboratory during the seventh semester June 2024–November 2024 under my supervision.

SIGNATURE

Dr.A.Shenbagarajan ,
Associate Professor,
AI&DS Department,
Mepco Schlenk Engg.College(Autonomous)
Sivakasi

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Professor & Head,
AI&DS Department,
Mepco SchlenkEngg.College(Autonomous)
Sivakasi

ABSTRACT

This project aims to implement a text-to-speech (TTS) system using an ESP32 microcontroller to interact with Google's Gen-AI Gemini model. The ESP32 is connected to a speaker via an audio amplifier, allowing it to convert text-based responses generated by the Gemini AI into speech. The system utilizes the ESP32's Wi-Fi capabilities to query the Gen-AI Gemini API for AI-generated answers to user-provided questions. The responses are processed, cleaned, and synthesized into audible speech, providing real-time interaction with the AI. The implementation involves integrating a relay-controlled speaker system for audio output and utilizing a TTS library compatible with the ESP32, such as ESP8266Audio or other suitable alternatives. Additionally, the ESP32 I2S or DAC (Digital-to-Analog Converter) interface is used to handle audio signal generation, amplified by an external module such as the MAX98357A or PAM8403. This system demonstrates how the ESP32 can be utilized as a low-cost, efficient platform to interact with advanced AI models like GenAI-Gemini and convert their text outputs into speech, offering potential applications in smart assistants, automated customer service, and voice-controlled devices

ACKNOWLEDGEMENT

First and foremost we **praise and thank "The Almighty",** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa Sujana M.E.,Ph.D.,** Professor & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We extremely thank our project coordinator **Dr.A.Shenbagarajan M.E.,Ph.D,** Associate Professor and **Dr.E.Emerson Nithiyaraj M.E., Ph.D,** Assistant Professor, Department of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

TABLE OF CONTENT

1. INTRODUCTION **Page No**

1.1 Introduction………………………………………………………. 8

1.2 Objectives……………………………………………………… 8

1.3 Scope of the project…………………………………………………. 9

2. PROPOSED SOLUTION

2.1 Block Diagram…………………………………………………… 11

2.2 System Behavior……..……………………………………………….. 11

2.2.1 API Integration………………………………………. 11

2.2.2 Text to Speech Synthesis……………………………………… 12

2.2.3 Audio Amplification…………………………………………… 13

2.3 Circuit Diagram…………………………………………………… 14

2.4 Hardware Requirements…………………………………………….. 14

2.6 Software Requirement……………………………………………. 15

2.6 E S P 3 2 ………………………………………………….. 15

2.6.1 ESP 32 Diagram………………………………………. 15

2.7 Audio Amplifier ……………………………………………... 17

2.7.1 Audio Amplifier - I2S………………………………………… 18

2.8 Bot………..…………………………………………. 19

2.8.1 GEMINI Bot…………..…………………….............. 20

2.9 Arduino Software…………………………………………........ 22

3. IMPLEMENTATION

3.1 Source Code…………………………………………………………….... 24

3.2 Sample Output……………………………………………….............. 29

4. CONCLUSION

4.1 Conclusion……………………………………………………………. 30

5. REFERENCE

5.1 Reference………………………………………….......................... 31

LIST OF FIGURES

S.NO	NAME	PAGE NO.
2.1.1	Block diagram	10
2.3.1	Circuit Diagram	12
2.6.1	ESP 32 Board	13
2.7.1	I2S Diagram	15
3.2.1	Execution Circuit Image	23
3.2.2	Circuit SetUp	23
3.2.3	Chat Execution - I	23

1. INTRODUCTION

1.1 INTRODUCTION

Text-to-speech (TTS) technology plays a pivotal role in this transformation by enabling machines to convert text into spoken language, making interactions more accessible, especially for people with disabilities or in situations where hands-free communication is necessary. This project explores the integration of the ESP32 microcontroller with Google's GenAI-Gemini model to create an innovative and low-cost TTS system capable of producing natural-sounding speech from AI-generated text responses. The ESP32 microcontroller is an ideal platform for this project due to its built-in Wi-Fi capabilities, processing power, and support for audio output through digital (I2S) or analog (DAC) interfaces. Its small form factor, low power consumption, and affordability make it a popular choice for IoT projects and embedded systems. By leveraging the ESP32's ability to connect to the internet, the system communicates with Google's GenAI-Gemini model through an API, allowing it to generate context-aware, conversational responses based on user queries.

1.2 OBJECTIVES

The aim of this project is the integration of the ESP32 microcontroller with G GenAI model to create an innovative and low-cost TTS system . To achieve this objective, we consider:

1. **To** develop a cost-effective text-to-speech (TTS) system using the ESP32 microcontroller integrated with the GenAI-Gemini model for AI-generated responses.
2. **To** enable real-time, natural-sounding speech output from text generated by the GenAI-Gemini model through a connected speaker and amplifier system.
3. **To** implement Wi-Fi connectivity on the ESP32 for seamless interaction with the GenAI-Gemini API, ensuring dynamic and context-aware responses to user queries.
4. **To** control speaker power efficiently using a relay module, reducing energy consumption by managing the power supply during playback.
5. **To** amplify the audio output of the ESP32, ensuring clear and audible speech using external audio amplifiers such as the MAX98357A (I2S) or PAM8403 (analog).
6. **To** enhance user interaction through hands-free, voice-based communication by providing natural, real-time feedback from AI-driven systems.

1.3 SCOPE OF THE PROJECT

The Text-to-Speech System using ESP32 with GenAI-Gemini Integration focuses on the development of a low-cost, microcontroller-based solution that converts AI-generated text into real-time spoken responses. The scope of the project includes the following:

Hardware Design

1. **Integration of the ESP32 microcontroller** for Wi-Fi connectivity and communication with the GenAI-Gemini API to receive AI-generated responses.
2. **Connection of a relay module** to control the power to the speaker for efficient energy usage during text-to-speech output.
3. **Use of external audio amplifiers** (such as MAX98357A or PAM8403) to enhance sound quality and volume during speech playback.

Software Development

1. **Developing a program** that enables the ESP32 to send user input queries to the GenAI-Gemini API and retrieve text-based responses.
2. **Implementing text-to-speech (TTS) libraries** to convert the received text into audible speech and output it through the connected speaker.
3. **Integrating real-time control algorithms** to manage the system's response, ensuring that the speaker is activated only when audio output is required.

Testing and Validation

1. **Testing the system** in different environments to evaluate the clarity and audibility of the speech output under various conditions, including distance from the speaker and ambient noise levels.
2. **Ensuring system reliability** in generating contextually accurate and clear speech for different queries and responses provided by the GenAI-Gemini model.

Real-World Applications

1. **Application in smart home systems**, allowing users to interact with AI voice assistants using a low-cost, DIY TTS solution.
2. **Adaptation for kiosks, public services, and customer support systems** where real-time voice-based communication is necessary.

2. PROPOSED SOLUTION

2.1 BLOCK DIAGRAM

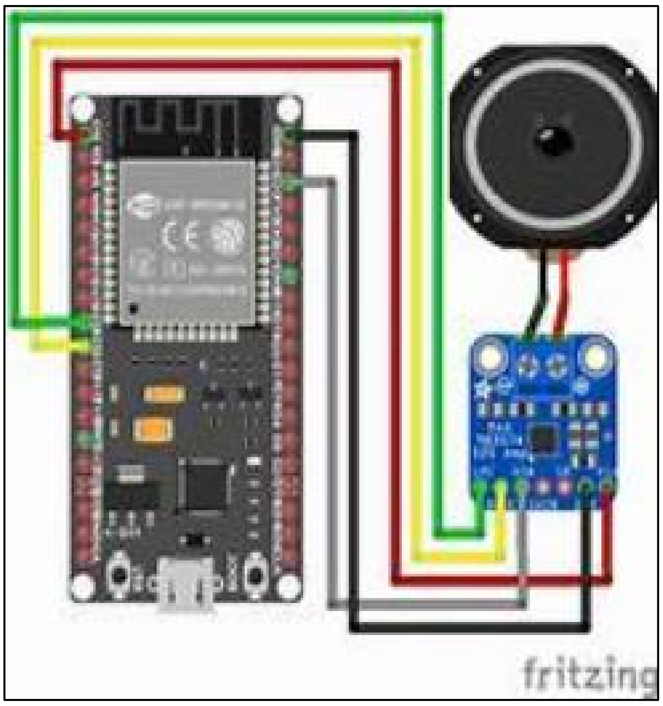


Figure 2.1.1 Block Diagram

2.2 SYSTEM BEHAVIOR

This section describes the detailed operation of the system, which starts with retrieving data from an API and culminates in audio output. It focuses on how the different components interact, emphasizing data flow and the transformation of the data into a usable output (speech). It outlines how the system reacts to external requests, processes data, and delivers real-time responses through a sequence of defined operations.

2.2.1 API Integration

In this part, the report focuses on how the **Gemini API** is integrated into the system. The ESP32 microcontroller acts as the core processor that sends requests to the API over a network (usually using Wi-Fi). Upon sending a request, the API returns a text-based response. This response could be based on user input, sensor data, or a predefined command.

The system behavior starts by making HTTP requests to the API. The API processes the input query and returns the required textual data. For example, if the system queries for a status update or a specific response from the Gemini API, the API will send back a text string that the system processes further. The API integration also handles potential errors, such as network delays or failed responses, ensuring that the system can either retry or provide alternative feedback in case of communication failure.

2.2.2 Text to Speech Synthesis

Once the text is retrieved from the Gemini API, the system needs to convert this textual data into an audible form. This is done through **Text-to-Speech (TTS) synthesis**. The ESP32 is capable of running TTS software or modules, which take the input text and transform it into digital speech.

In this process:

- The retrieved text data is parsed and processed by the TTS engine, which converts the text into phonetic components.
- The phonetic data is then synthesized into a waveform, representing the speech that will be outputted.
- The system may also allow for voice customization (e.g., tone, speed, pitch) based on the TTS capabilities.

The synthesized speech is output as a digital signal, but this signal alone is not enough for driving a speaker effectively, which is why the next step, audio amplification, is necessary.

2.2.3 Audio Amplification

After the text-to-speech synthesis converts the text into an audio signal, the system uses an audio amplifier to ensure the sound is loud and clear enough for the user to hear. The ESP32 typically sends the digital audio signal to a MAX98357A DAC (Digital-to-Analog Converter), which converts the digital signal into an analog audio signal suitable for amplification.

The analog signal is then fed into an audio amplifier circuit. The purpose of this amplifier is to boost the audio signal so that it can drive a speaker at a suitable volume. Without amplification, the raw signal from the TTS system may be too weak to produce audible sound.

The behavior of the system here includes:

- Ensuring that the signal is amplified without distortion, to maintain clear sound quality.
- Handling power requirements, as amplifiers typically require higher power to drive speakers.
- Ensuring the system dynamically adjusts the output volume, depending on user preferences or environmental conditions.

2.3 CIRCUIT DIAGRAM

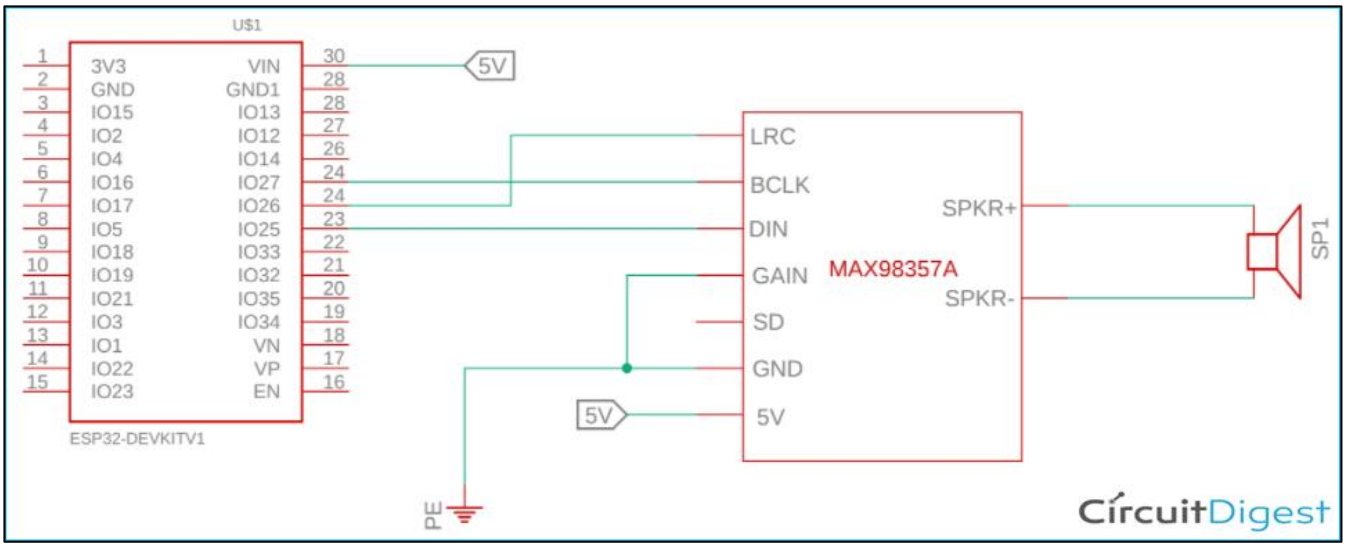


Figure 2.3.1 Circuit Diagram

2.4 HARDWARE REQUIREMENTS

- ESP32 Micro controller
- Jumper wires
- USB Cable
- LED
- I2S - MAX987532
- Breadboard

2.5 SOFTWARE REQUIREMENT

- Arduino IDE Software

2.6 ESP 32

ESP32 is a series of low-cost, low-power system-on-chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series employs either a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations, an Xtensa LX7 dual-core microprocessor, or a single-core RISC-V microprocessor and includes built-in antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management modules. Commonly found either on device specific PCBs or on a range of development boards with GPIO pins and various connectors depending on the model and manufacturer of the board.

2.6.1 ESP32 DEVKIT

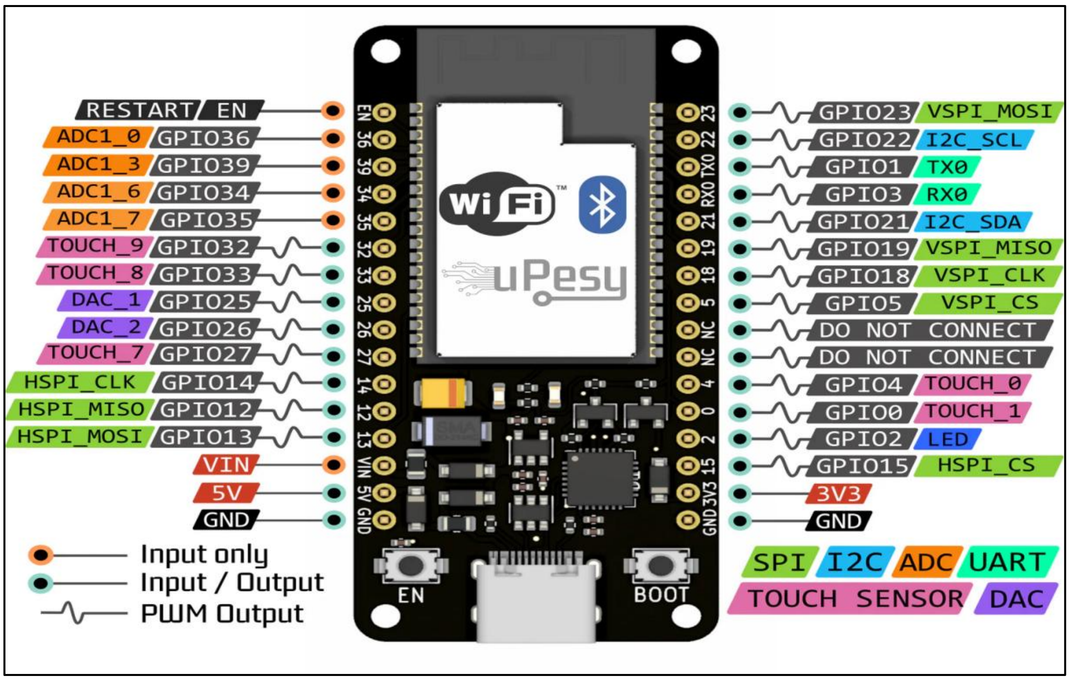


Figure 2.6.1 Esp32 Pin Diagram

1. **Power Pins**

   - **VIN:** Accepts input voltage (3.3V or 5V).
   - **3.3V:** Provides a 3.3V output for external components.
   - **GND:** Ground pins for completing the circuit.

2. **GPIO (General Purpose Input/Output) Pins**

   - Configurable as digital input/output, PWM, or special function pins (GPIO0 to GPIO39).

3. **Analog Pins (ADC)**

   - **ADC1 (GPIO32 to GPIO39)** and **ADC2** support analog input (0–3.3V).

4. **DAC (Digital to Analog Converter) Pins**

   - **GPIO25 (DAC1)** and **GPIO26 (DAC2)** provide analog output signals.

5. **Touch Sensor Pins**

   - **GPIO0, GPIO2, GPIO4, GPIO12 to GPIO15, GPIO27, GPIO32, GPIO33** are touch-sensitive and detect capacitance changes.

6. **SPI (Serial Peripheral Interface) Pins**

   - **GPIO12 (MISO), GPIO13 (MOSI), GPIO14 (SCLK), GPIO15 (CS):** Default SPI pins for high-speed communication.

7. **I2C (Inter-Integrated Circuit) Pins**

   - **GPIO21 (SDA)** and **GPIO22 (SCL)** used for I2C communication with devices like sensors.

8. **UART (Universal Asynchronous Receiver/Transmitter) Pins**

   - Supports serial communication: **UART0 (GPIO1, GPIO3)**, **UART1 (GPIO9, GPIO10)**, and **UART2 (GPIO16, GPIO17).**

9. **I2S (Inter-IC Sound) Pins**

   - Used for audio data communication with devices like DACs or microphones (e.g., **GPIO25, GPIO26, GPIO27** for I2S Data).

10. **EN (Enable) and BOOT Pins**

    - **EN (Enable Pin):** Resets or activates the ESP32.
    - **BOOT Pin:** Used to put the ESP32 into flash mode for code uploading.

2.7 AUDIO AMPLIFIER - I2S

An audio amplifier is a key component that increases the power of an audio signal to a level that can drive a speaker and produce sound. The ESP32 generates a low-power signal when converting text to speech using its DAC (Digital-to-Analog Converter) or I2S (Inter-IC Sound) interface. However, this signal is usually not strong enough to drive a speaker directly, which is why an audio amplifier is used.

- The purpose of the amplifier is to ensure that the generated audio signal is audible with sufficient clarity and volume when played through speakers.

- The amplifier takes the weak output from the ESP32 and boosts it, preventing signal distortion and ensuring the sound is clear.
- In this system, an I2S-based audio amplifier (which supports digital audio formats) is used to improve the audio quality.

2.7.1 Audio Amplifier I2S MAX

The I2S (Inter-IC Sound) protocol is a digital audio interface that is used for sending audio data between digital audio devices. The MAX98357A is a commonly used I2S DAC and amplifier module in ESP32-based audio systems.

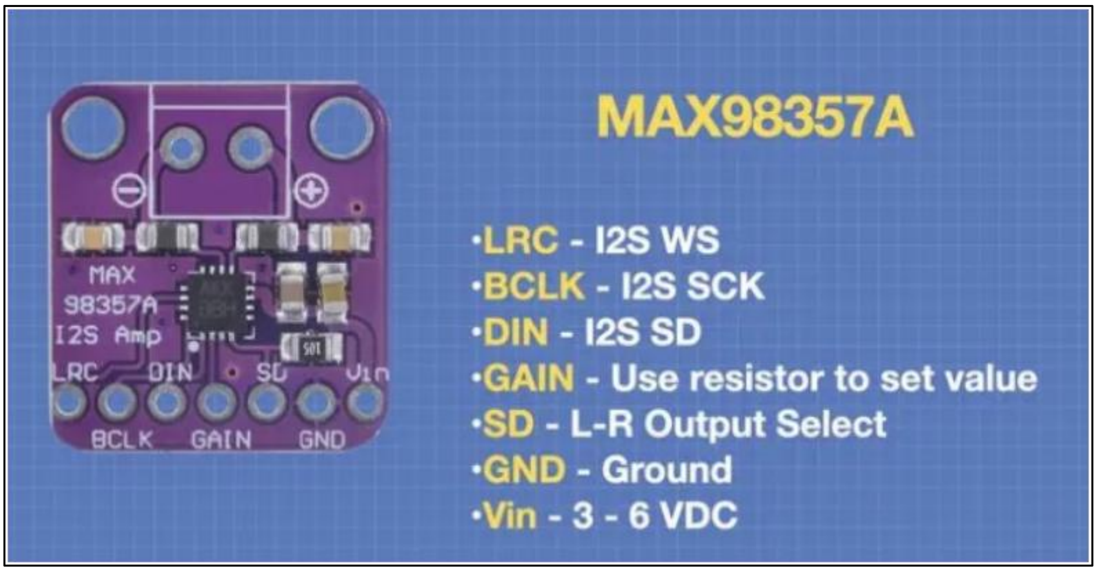


Figure 2.7.1 I2S Pin Diagram

- MAX98357A is an efficient mono amplifier that can take the digital audio data (I2S protocol) directly from the ESP32 and convert it to an analog signal while simultaneously amplifying it to drive a speaker.
- The I2S interface on the ESP32 sends digital audio data (bitstream), which the MAX98357A processes into an analog audio signal.
- This configuration allows for high-quality sound output because the digital signal is less prone to noise and interference before amplification, unlike traditional analog signals.
- The module also has a built-in amplifier, eliminating the need for an external analog amplifier. It can output up to 3.2W, which is enough for small to medium-sized speakers.

2.8 Bot

The Bot refers to the entire automated system being built, typically a chatbot or an interactive system that responds to user commands or queries. In this case, the Bot is a device that processes user input, retrieves responses from the Gemini API, and delivers spoken responses using the text-to-speech and audio amplification components.

- The Bot integrates multiple hardware and software components such as the ESP32, text-to-speech system, and the I2S amplifier, working together to provide real-time responses in the form of audible speech.
- The bot takes input either from sensors, voice commands, or other user interfaces, processes the data, and communicates back using voice output.
- The purpose of the Bot is to provide an intelligent, conversational interface for users to interact with, driven by the data from the Gemini API.

2.8.1 GEMINI Bot

The GEMINI Bot is the specific implementation of the bot, designed to interact with the Gemini API. It receives data from the API, processes it, and uses text-to-speech synthesis to communicate with the user.

- Gemini API Integration: The GEMINI Bot communicates with the Gemini API, which could provide various types of information like weather updates, news, or other requested data. The bot sends queries and retrieves text-based responses.
- Text-to-Speech Synthesis: After receiving a response from the API, the GEMINI Bot converts the text into speech using an integrated text-to-speech engine (TTS). This allows the bot to "speak" the data back to the user.
- Real-time Interaction: The GEMINI Bot is designed to provide real-time feedback. It receives user inputs (e.g., spoken commands or text queries), processes them through the API, and then responds in a natural, conversational manner using the audio amplification system for clear sound output.

2.8 ARDUINO SOFTWARE

1. Main Components

- Menu Bar: Located at the top, this includes options for File (new, open, save), Edit (cut, copy, paste), Sketch (verify, upload), Tools (board selection, port selection), and Help (documentation, examples).

- Code Editor: The central area where users write their Arduino sketches (programs). It supports syntax highlighting, which helps differentiate between keywords, variables, functions, and comments for better readability.

2. Code Editor Features

- Line Numbers: Displayed on the left side of the editor to help keep track of code lines.
- Auto-Completion: Offers suggestions as you type, helping speed up coding and reducing errors.

3. Output and Serial Monitor

- Output Window: Located at the bottom, this shows messages related to the compilation process, including errors and warnings. It provides feedback after attempting to upload the code to the board.
- Serial Monitor: Accessible via the toolbar, this allows you to view and send data between the Arduino and your computer. It is useful for debugging and monitoring real-time data.

4. Sketch Area

- Setup Function: This function runs once when the program starts. It's used for initializing variables, pin modes, etc.
- Loop Function: This function runs continuously after the setup function. It contains the main code that controls the Arduino's behavior.

5. Library Management

- Library Manager: Accessible through the Sketch menu, it allows users to include and manage libraries, which are collections of code that simplify complex tasks and expand the Arduino's capabilities.

6. Board and Port Selection

- Board Selector: Located in the Tools menu, this lets you select the specific Arduino board you are using, ensuring that the code is compiled for the correct hardware.
- Port Selector: This allows you to select the correct port for uploading the code to your Arduino board, which is essential for communication between the computer and the board.

3. IMPLEMENTATION

3.1 SOURCE CODE

```
#include <WiFi.h>

#include <HTTPClient.h>

#include <WebServer.h>

#include <ArduinoJson.h>


// WiFi Credentials

const char* ssid = "PORSCHE";

const char* password = "kinnsporsche";


// Gemini API Token and Settings

const char* Gemini_Token = "AIzaSyC_OMNC2wV4zjMx0KRJoUmDOc1N9B3QApg";

const char* Gemini_Max_Tokens = "100";


// Create a web server object that listens for HTTP requests on port 80

WebServer server(80);


// To store question and answer

String question = "";

String answer = "";


// LED and Speaker Pins

#define LED_PIN 2

#define SPEAKER_PIN 32


// HTML page with improved styling

String getStyledHTMLPage(String question, String answer) {
  String page = "<!DOCTYPE html><html><head><title>Question Answer App</title>";

  page += "<style>";

  page += "body { font-family: Arial, sans-serif; margin: 0; padding: 0; background-color: #f4f4f9; color: #333; }";

  page += "header { background-color: #4CAF50; padding: 20px; text-align: center; color: white; }";

  page += "main { display: flex; justify-content: center; align-items: center; height: 100vh; }";

  page += "form { background-color: white; padding: 40px; border-radius: 8px; box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1); }";

  page += "h1 { font-size: 24px; }";

  page += "label, input[type=text], input[type=submit] { display: block; width: 100%; margin: 10px 0; font-size: 16px; }";

  page += "input[type=text] { padding: 10px; border: 1px solid #ccc; border-radius: 4px; }";

  page += "input[type=submit] { background-color: #4CAF50; color: white; border: none; padding: 12px 20px; cursor: pointer; border-radius: 4px; }";

  page += "input[type=submit]:hover { background-color: #45a049; }";

  page += "p { font-size: 18px; margin-top: 20px; }";

  page += "</style></head><body>";

  page += "<header><h1>TEXT TO SPEECH USING ESP32</h1></header>";

  page += "<main>";

  page += "<form action=\"/ask\" method=\"POST\">";

  page += "<label for=\"question\">Ask Your Question:</label>";

  page += "<input type=\"text\" id=\"question\" name=\"question\" placeholder=\"Type your question here...\" value=\"" + question + "\">";

  page += "<input type=\"submit\" value=\"Submit\">";



  if (answer != "") {

    page += "<p><strong>Answer:</strong> " + answer + "</p>";

  }
  page += "</form>";
  page += "</main></body></html>";


  return page;
}


// Function to handle the root URL ("/") and serve the input form
void handleRoot() {
  server.send(200, "text/html", getStyledHTMLPage("", ""));  // Serve the
form initially
}


// Function to handle the question submission
void handleAsk() {
  // Get the question from the POST request
  if (server.hasArg("question")) {
    question = server.arg("question");
    question.trim();


    // Check for "LED ON" or "LED OFF" command in the question
    if (question.equalsIgnoreCase("LED ON")) {
      digitalWrite(LED_PIN, HIGH);
      answer = "LED is turned ON";
      server.send(200, "text/html", getStyledHTMLPage(question, answer));
      return;
    } else if (question.equalsIgnoreCase("LED OFF")) {
      digitalWrite(LED_PIN, LOW);
      answer = "LED is turned OFF";
      server.send(200, "text/html", getStyledHTMLPage(question, answer));
      return;
    }

    // Fetch the answer from Gemini API if no LED command is found
    if (fetchAnswerFromAPI()) {
      // Check if answer contains "LED ON" or "LED OFF"
      if (answer.indexOf("LED ON") >= 0) {
        digitalWrite(LED_PIN, HIGH);  // Turn LED on if "LED ON" is in the
answer
      } else if (answer.indexOf("LED OFF") >= 0) {
        digitalWrite(LED_PIN, LOW);   // Turn LED off if "LED OFF" is in
the answer
      }

      // Play answer as tones
      playAnswerAsTone(answer);
      server.send(200, "text/html", getStyledHTMLPage(question, answer));
    } else {
      server.send(500, "text/html", "<h2>Error fetching answer!</h2>");
    }
  } else {
    server.send(400, "text/html", "<h2>No question received!</h2>");
  }
}

// Function to make the request to the Gemini API and get the response
bool fetchAnswerFromAPI() {
  HTTPClient https;
  // Prepare the Gemini API request

  if
(https.begin("https://generativelanguage.googleapis.com/v1beta/models/gemi
ni-1.5-flash:generateContent?key=" + (String)Gemini_Token)) {

    https.addHeader("Content-Type", "application/json");



    // Create JSON payload with the question

    String payload = String("{\"contents\": [{\"parts\":[{\"text\":\"") +
question  +  "\"}]}],\"generationConfig\":  {\"maxOutputTokens\":  "  +
(String)Gemini_Max_Tokens + "}}";



    int httpCode = https.POST(payload);  // Send the POST request



    // Check if the request was successful

    if (httpCode == HTTP_CODE_OK) {

      String response = https.getString();

      DynamicJsonDocument doc(2048);

      deserializeJson(doc, response);



      // Get the answer from the JSON response

      answer                                                             =
doc["candidates"][0]["content"]["parts"][0]["text"].as<String>();

      answer.trim();  // Trim whitespace and newlines



      https.end();

      return true;  // Successfully got the answer

    } else {

      Serial.printf("[HTTPS]     POST     failed,     error:     %s\n",
https.errorToString(httpCode).c_str());

      https.end();

      return false;  // Failed to get answer
    }
  } else {
    Serial.println("Failed to connect to Gemini API");
    return false;
  }
}

// Function to play answer as tones using speaker
void playAnswerAsTone(String text) {
  int duration = 200;  // Duration of each tone in milliseconds

  for (size_t i = 0; i < text.length(); i++) {
    char c = text[i];
    if (isalnum(c)) {
      int freq = 440 + ((c - 'A') * 10);  // Generate a frequency based on
the character
      tone(SPEAKER_PIN, freq, duration);  // Play the tone on the speaker
      delay(duration);  // Delay for the tone duration
    } else if (isspace(c)) {
      delay(duration);  // Add a pause for spaces
    }
  }
  noTone(SPEAKER_PIN);  // Stop the tone after playing
}

void setup() {
  Serial.begin(115200);

  // Set LED and speaker pins
  pinMode(LED_PIN, OUTPUT);

  pinMode(SPEAKER_PIN, OUTPUT);

  digitalWrite(LED_PIN, LOW);


  // Connect to WiFi

  WiFi.mode(WIFI_STA);

  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi...");

  while (WiFi.status() != WL_CONNECTED) {

    delay(1000);

    Serial.print(".");

  }

  Serial.println("Connected!");

  Serial.print("IP address: ");

  Serial.println(WiFi.localIP());


  // Setup server routes

  server.on("/", handleRoot);          // Serve the input form on the root
URL

  server.on("/ask", HTTP_POST, handleAsk);  // Handle form submission via
POST


  // Start the server

  server.begin();

  Serial.println("HTTP server started.");

}

void loop() {

  server.handleClient();  // Handle incoming client requests

}
```

3.2 RESULTS

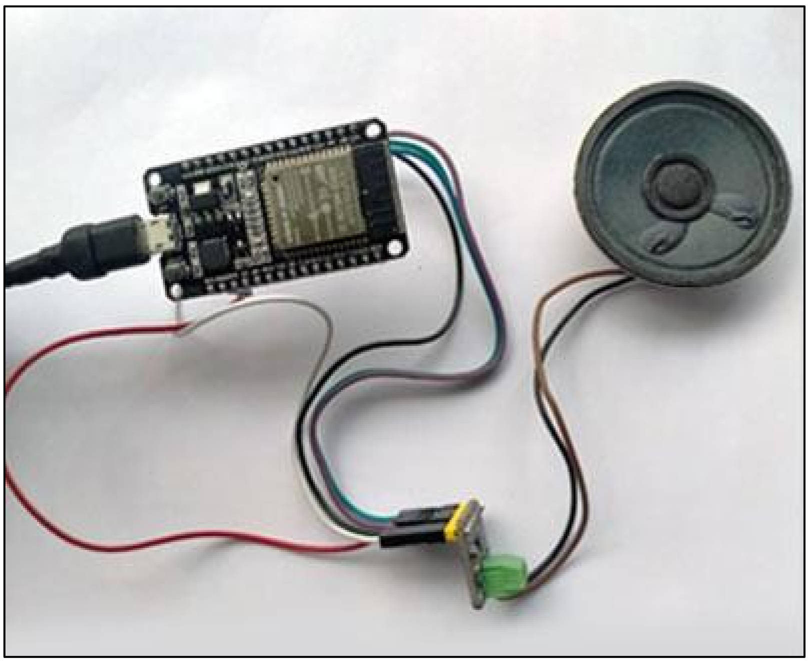
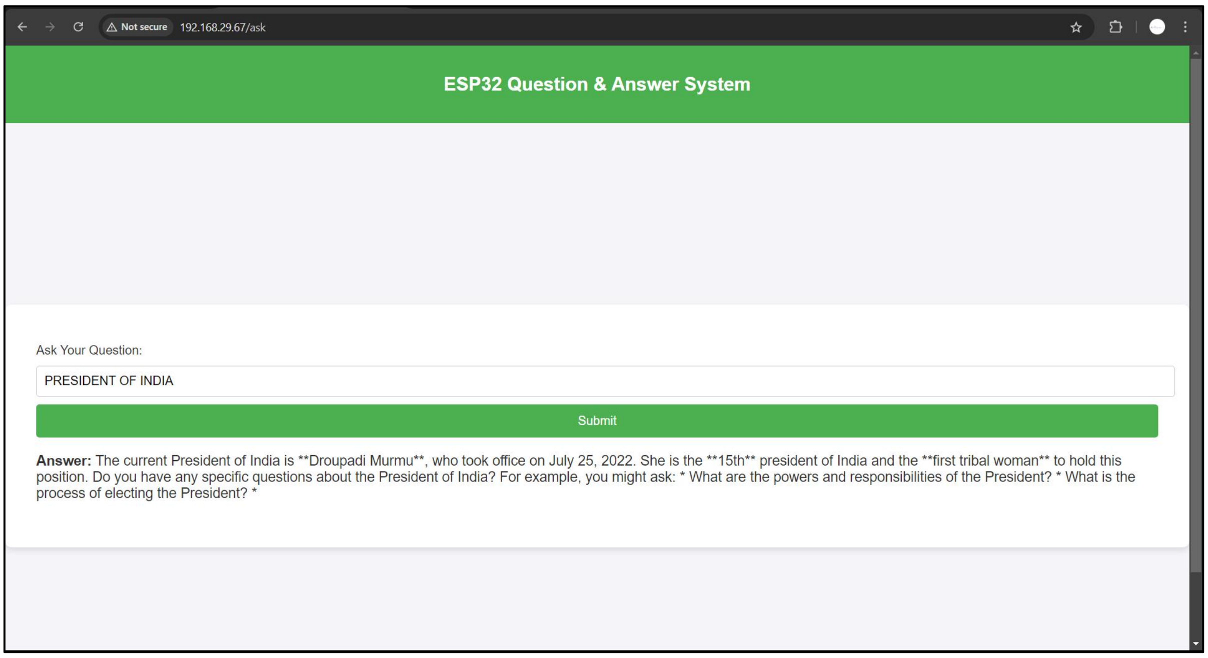


Figure 3.2.1 Circuit Setup



⚠ Not secure 192.168.29.67/ask

ESP32 Question & Answer System

Ask Your Question:

PRESIDENT OF INDIA

Submit

Answer: The current President of India is **Droupadi Murmu**, who took office on July 25, 2022. She is the **15th** president of India and the **first tribal woman** to hold this position. Do you have any specific questions about the President of India? For example, you might ask: * What are the powers and responsibilities of the President? * What is the process of electing the President? *

```````

Figure 3.2.2 Chat Execution-I
```````

4. CONCLUSION

4.1 CONCLUSION

In Conclusion ,this project demonstrates a comprehensive and efficient system that integrates multiple technologies, resulting in an interactive bot capable of real-time communication through voice. The ESP32, with its versatile GPIO pins, low power consumption, and ability to handle both Wi-Fi and Bluetooth, serves as the core of the system, enabling seamless API integration, text-to-speech conversion, and audio signal processing.The use of the I2S protocol and the MAX98357A audio amplifier ensures that the digital-to-analog conversion of audio signals is precise and that the sound output is amplified without distortion, providing a high-quality audio experience. This combination allows for a compact, energy-efficient design while maintaining robust performance, especially in terms of clear, amplified audio.

Through the GEMINI Bot, which interfaces with the Gemini API, the system can retrieve real-time data and convert it into natural, human-like speech. The bot not only interacts with users by answering queries or performing specific tasks but also provides a seamless user experience by converting text-based information into spoken language. This makes the system highly versatile for a range of applications, from home automation and smart assistants to customer service bots. The success of this system lies in its ability to merge hardware and software components into a cohesive, fully functional voice assistant. This project highlights the potential of using IoT devices like the ESP32 for smart applications, showcasing their capacity for handling both complex tasks (like API interaction) and real-world interfacing (like sound output through speakers). Furthermore, the scalability and adaptability of this setup make it suitable for expanding into future projects involving additional sensors, machine learning integration, or enhanced voice recognition systems.

5. REFERENCE

5.1 REFERENCE

- https://dronebotworkshop.com/esp32-i2s/
- https://diyi0t.com/i2s-sound-tutorial-for-esp32/
- https://iotdesignpro.com/projects/esp32-based-text-to-speech-converter-webserver
- https://dev.to/panlee123/lets-talk-espressif-esp32-s3-voice-text-to-speech-tts-55h
- https://youtu.be/43B6RtXiTkQ
- https://www.autodesignmagazine.com
- https://www.arduino.cc/en/Main/ArduinoBoardUno
- https://www.sae.org