A Project Report on

IOT BASED NOISE MONITORING SYSTEM WITH AWS INTEGRATION

Submitted in Partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

J. Bhaskar Anil	**21P31A0491**
T. Ramesh Babu	**21P31A04C3**
V. Karthik	**21P31A04C9**
P. Manohar	**21P31A04B4**

Under the esteemed guidance of

G. Sattibabu M. Tech, (Ph. D)

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY[A]

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada & Accredited by NBA & NAAC(A^+) with CGPA of 3.4 recognized by UGC under the sections 2(f) and 12(B) of the UGC act 1956) **Aditya Nagar, ADB Road,**

Surampalem, E.G.Dt., - 533 437

(2021-2025)

INSTITUTE VISION AND MISSION

VISION:

To induce higher planes of learning by imparting technical education with

- International standards
- Applied research
- Creative Ability
- Value based instruction and to emerge as a premiere institute

MISSION:

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- Innovative research and development
- Industry institute interaction
- Empowered manpower

[illegible]

PRINCIPAL

PRINCIPAL
Aditya College of
Engineering & Technology
SURAMPALEM- 533 437

DEPARTMENT VISION AND MISSION

VISION: To emerge as a center of excellence in education and research

MISSION:

- ❖ To establish skill and learning centric infrastructure in thrust areas
- ❖ To develop Robotics and IOT based infrastructure laboratories
- ❖ To organize events through industry institute collaborations and promote innovation
- ❖ To disseminate knowledge through quality teaching learning process.

[illegible]

Head of the Department

Head of the Department
Dept.of ECE
Aditya College of
Engineering & Technology
SURAMPALEM-533 437

PROGRAM SPECIFIC OUTCOMES (PSOs)

Program Name: Bachelor of Technology (B.Tech) in Electronics & Communication Engineering

PSO1: Industry ready in the arena of Electronics & Communication, VLSI, Robotics, Embedded Systems, IOT and allied fields.

PSO2: Acquire the required ability and knowledge to design, test, verify and develop innovative electronics projects through theoretical and laboratory practice.

[illegible]

Head of the Department

Head of the Department
Dept.of ECE
Aditya College of
Engineering & Technology
SURAMPALEM-533 437

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

Program Name: Bachelor of Technology (B.Tech) in Electronics and Communication Engineering.

PEO1: Graduates shall evolve into skilled professionals capable of handling interdisciplinary work atmosphere and excel in problem solving.

PEO2: Graduates shall inculcate the urge to progress in the chosen field of Electronics & Communication through higher education and research.

PEO3: Graduates shall ingrain professional values through an ethics-based teaching-learning process.

PEO4: Graduates shall exhibit leadership skills and advance towards entrepreneurship, innovation and lifelong learning.

[illegible]

Head of the Department

Head of the Department
Dept.of ECE
Aditya College of
Engineering & Technology
SURAMPALEM-533 437

PROGRAM OUTCOMES (POs)

PO1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems

PO2. **Problem Analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. **Design/Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. **Conduct Investigations of Complex Problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. **Modern Tool Usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6. **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. **Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. **Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, give and receive clear instructions.

PO11. **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. **Life-Long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
B.Tech 4/4, II-SEMESTER

Course Outcomes

Upon completion of the course, students will be able to:

CO#	Course Outcomes	Blooms Taxonomy level
CO1	Identify the problem by applying acquired knowledge.	Remember
CO2	Use literature to identify the objective, scope and the concept of the work.	Apply
CO3	Analyse and categorize executable project modules after considering risks.	Analyse
CO4	Choose efficient tools for designing project modules.	Evaluate
CO5	Integrate all the modules through effective team work after efficient testing.	Create
CO6	Explain the completed task and compile the project report.	Understand

CO-PO/PSO MATRIX:

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
CO1	3	2	-	-	-	-	-	3	3	-	-	-	-	3
CO2	3	3	-	2	-	-	-	-	3	-	-	-	-	3
CO3	2	3	2	-	1	-	-	2	2	-	2	2	-	2
CO4	3	2	3	-	3	-	-	-	2	2	2	-	-	-
CO5	2	2	3	-	2	3	-	-	3	2	3	-	-	-
CO6	2	-	-	-	-	-	-	3	-	3	2	2	2	2
Course	2.5	2.4	2.6	2	2	3	-	2.6	2.5	2.33	2	2	2	2.5

Signature of the Guide

CO-PO Justification:

CONo.	PO/PSO	CL	Justification
CO1	PO1	3	Highly mapped as we need basic concepts of LSTM.
	PO2	2	Moderately mapped as we need to identify, analyze and generating the text and audio output.
	PO4	3	Highly mapped as we need to design solutions for problems by choosing efficient tools.
	PO6	3	Highly mapped as we need to apply modern software to the society and environment development.
	PO7	2	Moderately mapped as we need to Understand the impact of the professional engineering
	PO8	2	Moderately mapped as we will be able to understand importance of in real time applications.
	PO9	3	Highly mapped as we will need required software.
	PO10	2	It is moderately mapped as the we will be able to communicate their work in reviews and paper presentations.
	PO11	3	Highly mapped as team need to work on project management.
	PO12	2	Moderately mapped as we will need to engage in independent and life-long learning in the context of technological change.
	PSO1	3	Highly mapped as we will be able to understand importance of in real time applications.
	PSO2	2	Moderately mapped as we will able to understand how the lstm works in sign language detection.
CO2	PO1	2	Moderately mapped as we need to apply lstm for the sign language detection.
	PO2	3	Highly mapped as we need to analyze the problems related to sign language detection.
	PO6	2	Moderately mapped as we will be able to apply the algorithm to the society and environment development.
	PO8	2	Moderately mapped as we need to have the basic idea about the rules to be followed.

	PO9	2	Moderately mapped as we need to work together to study the tools and technologies.
	PO10	3	Highly mapped as the we will be able to communicate their work in reviews and paper presentations.
	PSO1	2	Moderately mapped as we need to have to train ourselves industry-ready in the field of electronics and communication.
CO3	PO1	2	Moderately mapped as we should know about sign language detection.
	PO3	2	Moderately mapped as we need to select the reqired software for the sign language detection.
	PO4	2	Moderately mapped as we will be able to investigate the complex problems in the project.
	PO6	2	Moderately mapped as we will be able to apply the algorithm to the society and environment development.
	PO7	2	Moderately mapped as we need to Understand the impact and usage of the professional engineering.
	PO9	2	Moderately mapped as we need to work together to study the tools and technologies.
	PO10	3	Highly mapped as the we will be able to communicate their work in reviews and paper presentations.
	PO11	2	Higly mapped as team need to work on project management.
	PO12	2	Moderately mapped as we will be able to understand importance of in real time applications.
	PSO2	2	Moderately mapped as we will need to reqired software.
CO4	PO1	3	Mapped as we should know about sign language detection.
	PO2	2	Moderately mapped as we require analysis of efficient software required.
	PO3	3	Highly mapped as we will be able to find the solution for the problem identified in CNN network.
	PO4	2	Moderately mapped as we require research knowledge on designing.
	PO5	2	Highly mapped as we need to know tools used related to CNN network.
	PO8	2	Moderately mapped as we know the rules which are followed while designing the project.
	PO9	2	Moderately mapped because for the designing system, it requires teamwork as well as individual.
	PO11	2	Moderately mapped as designing requires team work as well as individual work.

	PO1	2	Mapped as we should know about sign language detection using LSTM.
CO5	PO3	2	Moderately mapped as we need to design system components for integration.
	PO4	2	Moderately mapped as we need to design solutions for CNN networks.
	PO6	2	Moderately mapped as we will be able to apply the machine learning to the society and environment development.
	PO8	2	Moderately mapped as we need to have the basic idea about the rules to be followed.
	PO9	3	Highly mapped as we need to identify the risks involved in the projectand its effecton society.
	PO10	2	Moderately mapped as we have better communication while integrating the components for effective results.
	PO11	3	It is Highly mapped as the students will be able to manage the financial constraints.
CO6	PO1	3	Highly mapped as we have full knowledge on the working and the project report.
	PO6	2	Highly mapped as we will be able to apply the deep learning to the society and environment development.
	PO7	2	Moderately mapped as we need to Understand the impact and usage of the professional engineering.
	PO9	2	Moderately mapped as we need to identify the risks involved in the project and its effecton society.
	PO10	3	Highly mapped as we need to have the basic idea about LSTM networks.
	PO11	2	Moderately mapped as team need to work on project management.
	PO12	2	Moderately mapped as we will need to engage in independent and life-long learning in the context of technological change.
	PSO1	2	Moderately mapped as we need to have to train ourselves industry ready in the feild of electronics and communication.
	PSO2	2	Mapped as we utilized laboratory infrastructure for practical applications

Signature of the Guide

ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY

An AUTONOMOUS Institution

Approved by AICTE, Permanently Affiliated to JNTUK, Accredited by NBA & NAAC with A+ Grade
Recognized by UGC under Sections 2(f) and 12(B) of UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Kakinada District - 533 437, A.P.
Ph: 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

Department of Electronics and Communication Engineering

CERTIFICATE

This is to certify that the project report entitled **"IOT BASED NOISE MONITORING SYSTEM WITH AWS INTEGRATION."** is being submitted by J. Bhaskar Anil (21P31A0491),T. Ramesh Babu (21P31A04C3), V. Karthik (21P31A04C9), P. Manohar (21P31A04B4),has been carried out in the partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering, Aditya College of Engineering & Technology, Surampalem, affiliated to JNTUK, Kakinada is a record of bonafide work carried out by them under my guidance and supervision during the academic period 2024-2025.

Project Guide	Head of the Department
Mr. G Sattibabu, M. Tech (Ph. D)	**Dr. R. V. V. Krishna, M. Tech, Ph.D.,**
Assistant Professor	Professor
ECE Department	ECE Department

EXTERNAL EXAMINER

DECLARATION

We here by declare that the project entitled "**IOT BASED NOISE MONITORING SYSTEM WITH AWS INTEGRATION**" has been undertaken by our batch, and the project report is submitted to Aditya college of engineering & technology, Surampalem in partial fulfillment of the requirements for the award of degree of Bachelor of technology in Electronics & communication engineering.

Yours Sincerely

J. Bhaskar Anil (21P31A0491)

T. Ramesh Babu (21P31A04C3)

V. Karthik (21P31A04C9)

P. Manohar (21P31A04B4)

ACKNOWLEDGEMENT

It gives us immense pleasure to express a deep sense of gratitude to my guide **Mr. G. Sattibabu M. Tech, (Ph. D)**, Department of ECE for wholehearted and invaluable guidance throughout the project work. Without his sustained and sincere effort, this project work would not have taken this shape. He encouraged and helped me to overcome various difficulties that we have faced at various stages of our project work.

we would like to sincerely thank our Head of the department **Dr.R.V.V. Vijaya Krishna M. Tech, Ph. D**, for providing all the necessary facilities that led to the successful completion of our project work.

we would like to take this opportunity to thank our beloved Principal **Dr. A. Ramesh**, **M. Tech, Ph.D**, for providing all the necessary facilities and a great support to us in completing the project work. We would like to thank all the faculty members and the non-teaching staff of the Department of Electronics and Communication Engineering for their direct or indirect support in helping us in the completion of this project work.

Finally, we would like to thank all our friends and family members for their continuous help and encouragement.

J. Bhaskar Anil (21P31A0491)
T. Ramesh Babu (21P31A04C3)
V. Karthik (21P31A04C9)
P. Manohar (21P31A04B4)

ABSTRACT

Maintaining a suitable noise-controlled environment is essential across various settings, including educational institutions, workplaces, hospitals, and public spaces. Excessive noise can disrupt concentration, reduce productivity, and negatively impact overall well-being. This project presents an IoT-based Noise Monitoring System integrated with AWS Cloud Services to address this issue effectively. The system leverages a sensor to capture ambient noise levels, processes the data using a microcontroller, and transmits it securely to AWS IoT Core via the MQTT protocol. The captured data is analyzed in real-time to detect noise levels exceeding predefined thresholds. Upon detecting excessive noise, the system triggers notifications to the concerned authorities via Amazon Simple Notification Service (SNS). AWS Lambda functions process incoming data, ensuring seamless integration between IoT devices and cloud-based analytics. This cost-effective and scalable solution enhances noise management in diverse environments by providing real-time alerts and actionable insights, demonstrating the potential of IoT and cloud computing in addressing noise pollution across multiple domains.

INDEX

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In modern environments, noise pollution has become a significant concern, affecting productivity, concentration, and overall well-being. Whether in educational institutions, office spaces, hospitals, or public areas, excessive noise levels can lead to stress, reduced efficiency, and even long-term health issues. Traditional methods of monitoring noise pollution rely on manual observation and periodic measurements, which are inefficient and fail to provide real-time data for immediate action. To address this challenge, this project introduces an automated noise monitoring system utilizing IoT technology and cloud-based data processing. By integrating advanced microcontroller technology with cloud services, this system enables continuous noise level monitoring and automated alerts whenever predefined thresholds are exceeded. This approach ensures better control over noise pollution in sensitive environments, making it easier for authorities to take timely action.

The core of this project is built around the ESP32 microcontroller, a powerful and cost-effective device with built-in Wi-Fi and Bluetooth capabilities. Equipped with a high-sensitivity microphone sensor, the ESP32 collects real-time sound data from the surrounding environment. This data is processed and transmitted securely to AWS IoT Core, where it is further analyzed for noise level fluctuations. When noise levels exceed a specified threshold, AWS Lambda processes the incoming data and triggers Amazon Simple Notification Service (SNS) to send alerts via email or SMS. This ensures that the appropriate personnel receive notifications immediately, allowing them to take corrective measures. The seamless integration of hardware and cloud computing makes this system both scalable and adaptable to various settings where noise control is essential.

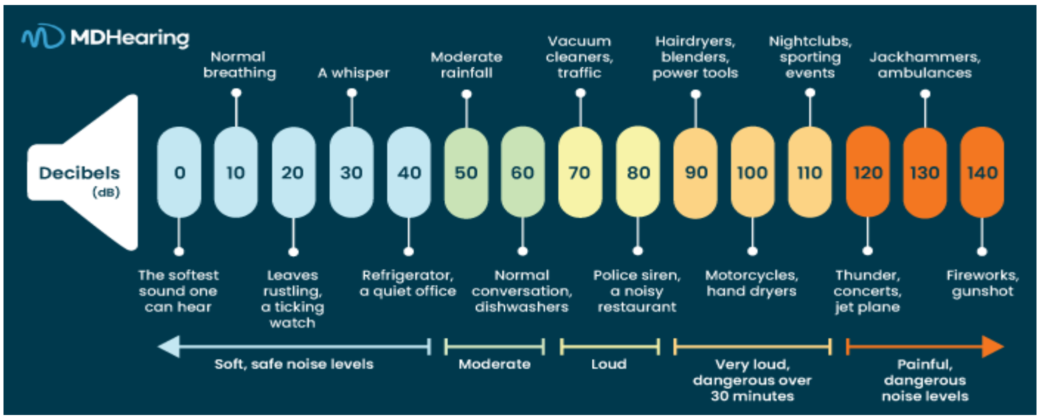


Fig 1.1 Desibel chart

One of the primary motivations behind developing this system is to enhance the learning experience in educational institutions. Classrooms and lecture halls are often affected by unwanted noise, which disrupts student concentration and reduces teaching effectiveness. By deploying this noise monitoring solution in educational settings, school administrators and faculty members can receive instant alerts when noise levels become disruptive, allowing them to take necessary actions to maintain a conducive learning environment. Similarly, libraries and study areas can benefit from such monitoring, ensuring that students can work in a quiet and focused atmosphere. The implementation of such a system reflects the increasing need for smart technology solutions in education, improving both student engagement and overall academic performance.

The Fig 1.1 illustrates noise levels in decibels (dB) and their impact on human hearing. It categorizes sound intensity from soft and safe levels, like normal breathing (10 dB), to dangerously loud noises, such as fireworks (140 dB). Understanding these levels is crucial for monitoring and managing noise pollution in various environments.Beyond educational settings, this noise monitoring system has broad applications in workplaces and office environments where excessive noise can hinder productivity. Open-plan offices, call centers, and collaborative workspaces often experience noise disturbances that affect employee efficiency and communication. Implementing an automated noise detection and alert system helps maintain a balanced acoustic environment, promoting a more productive and comfortable workspace. Moreover, industries and factories with strict occupational health and safety regulations can utilize this system to monitor noise exposure levels for workers. By continuously tracking noise levels, organizations can ensure compliance with safety standards and take preventive measures to protect employees from noise-induced health risks.Healthcare facilities, particularly hospitals and clinics, require a quiet environment to provide optimal patient care. High noise levels in hospitals have been linked to increased stress among patients and medical staff, negatively impacting recovery times and decision-making. A noise monitoring system can help hospital administrators identify areas with excessive noise and implement measures to minimize disturbances. For instance, in intensive care units (ICUs), maternity wards, and recovery rooms, maintaining low noise levels is essential for patient comfort and well-being. By leveraging cloud-based noise monitoring, hospitals can enforce noise control policies more effectively, ultimately enhancing the quality of healthcare services.

In urban settings, noise pollution is a growing concern due to traffic, construction activities, and public gatherings. Municipal authorities and environmental agencies can utilize this system to monitor noise levels in residential areas, commercial zones, and high-traffic intersections. Real-time noise data can be analyzed to enforce noise regulations, issue warnings, and implement necessary measures to reduce excessive noise pollution. Additionally, smart city initiatives can integrate this noise monitoring system into broader environmental monitoring networks, providing valuable insights into urban noise trends. Such implementations contribute to making cities more livable, promoting a healthier and more sustainable urban environment.The development of this noise monitoring system also highlights the increasing role of IoT and cloud computing in real-world problem-solving. The use of AWS IoT Core, Lambda functions, and SNS ensures reliable and secure data handling while minimizing infrastructure costs. By adopting a serverless architecture, the system eliminates the need for expensive on-premises hardware and maintenance, making it a cost-effective solution. Furthermore, the ability to scale the system based on demand allows organizations to deploy noise monitoring solutions across multiple locations without significant additional investment. The project serves as a practical example of how cloud-based IoT applications can transform traditional monitoring systems into intelligent, automated solutions.

This noise monitoring system is a comprehensive and innovative approach to addressing noise pollution in various environments. By combining the capabilities of IoT devices with cloud computing, it provides a highly effective solution for real-time noise detection and alerting. Whether in classrooms, workplaces, hospitals, or urban areas, this system offers a proactive method for maintaining optimal noise levels. The integration of automation, scalability, and cost efficiency makes it a valuable tool for organizations and authorities seeking to create quieter, more controlled environments. This project not only demonstrates the technical feasibility of IoT-driven noise monitoring but also underscores its potential impact on improving quality of life in noise-sensitive areas.

This project document is structured into several chapters, each providing a detailed explanation of different aspects of the IoT-based Noise Monitoring System with AWS Integration. Chapter 1 (Introduction) presents an overview of noise pollution and its impact, emphasizing the need for an automated noise monitoring system that integrates IoT and cloud computing for real-time detection and alerts. Chapter 2 (Literature Survey) reviews previous research studies on noise monitoring technologies, analyzing different methodologies, challenges, and solutions while identifying gaps that the proposed system aims to address. Chapter 3 (Key Hardware) describes the essential components used in the system, including the ESP32 microcontroller, microphone sensors, power supply, and charging module, explaining their role in noise detection, data processing, and transmission. Chapter 4 (Amazon Web Services - AWS) provides an introduction to AWS cloud services and their significance in the project, detailing key services such as AWS IoT Core, AWS Lambda, AWS SNS, and AWS CloudWatch, which facilitate secure data handling, automated notifications, and system scalability. Chapter 5 (Existing System) examines the current noise monitoring methodologies, their limitations, and challenges that affect their efficiency. Chapter 6 (Proposed System) presents the system's architecture, including the block diagram, working principles, and key features such as real-time noise monitoring, wireless connectivity, cloud integration, and automated alerts and Implementation Steps provides a step-by-step guide on configuring the ESP32 microcontroller, integrating AWS services, setting up MQTT communication, and deploying AWS Lambda functions for automated responses. Chapter 7 (Results) showcases the system's performance through Arduino Serial Monitor logs, AWS CloudWatch logs, and email notifications, demonstrating its real-time functionality and effectiveness in detecting and reporting noise levels. This structured approach ensures a comprehensive understanding of the system from conceptualization to implementation and performance evaluation, making it a scalable and efficient solution for noise pollution control.

CHAPTER 2
LITERATURE SURVEY

The literature survey provides a comprehensive analysis of previous research studies related to noise monitoring systems. Various studies have been conducted to develop effective noise monitoring solutions using IoT, machine learning, and cloud-based platforms. These systems aim to track, analyze, and control noise pollution in different environments, such as educational institutions, workplaces, and urban areas.

1. Siddhartha CV et al., “Classroom Activity Detection in Noisy Preschool Environments with Audio Analysis” (2023) This paper presents an automated system for detecting classroom activities using audio recordings in preschool environments. By leveraging machine learning techniques, the study categorizes noise levels to evaluate teacher-student engagement. The research emphasizes the importance of real-time noise monitoring to improve learning environments and suggests the potential application of IoT-based noise monitoring solutions. The system employs a multilabel classification approach to accurately identify classroom noise disturbances and optimize learning conditions. Additionally, the study highlights the challenges faced in differentiating between speech and background noise, ensuring the robustness of the classification model. The system architecture includes various sensors and audio processing units that work collaboratively to provide detailed acoustic data analysis. By implementing advanced deep learning techniques, the study improves noise classification accuracy, making it an effective solution for real-world applications. Furthermore, the study proposes recommendations for educators and policymakers to adopt noise monitoring systems for creating distraction-free classrooms. The proposed model can also be extended to other environments such as libraries, examination halls, and open workspaces where noise control is crucial. This research provides a comprehensive framework for using AI-driven noise classification in educational institutions, paving the way for future innovations in intelligent noise monitoring systems.
2. Aris Castillo de Valencia et al., “Development of a Noise Monitoring and Control Sensor Network System for the Enclosed Spaces within a University Environment” (2023) This study introduces an IoT-based noise control and monitoring system for university classrooms. The system utilizes networked sensors to collect real-time noise level data, allowing faculty and students to take corrective measures when

noise exceeds predefined thresholds. The research highlights how noise pollution affects student performance and proposes an effective technology-driven solution for maintaining an optimal learning atmosphere. The study also discusses implementation challenges and strategies for large-scale deployment. The proposed system incorporates cloud computing for efficient data storage and analysis, enabling seamless access to historical noise trends. The authors present a comparative analysis of different sensor placement methodologies to optimize noise detection accuracy. Furthermore, the study outlines an automated alert mechanism that notifies faculty and administrative personnel when excessive noise is detected. The system's scalability ensures that it can be deployed in multiple university settings, enhancing its applicability. The research also provides valuable insights into the psychological impact of noise on students and faculty members, emphasizing the need for effective noise management strategies. The proposed framework integrates machine learning algorithms for predictive noise analysis, allowing institutions to take preventive measures before noise levels become disruptive. The study's results demonstrate a significant improvement in student concentration and overall academic performance when noise levels are monitored and controlled effectively.

3. Gonçalo Marques et al., “A Real-Time Noise Monitoring System Based on Internet of Things for Enhanced Acoustic Comfort and Occupational Health” (2020) This paper discusses an IoT-enabled real-time noise monitoring system aimed at improving workplace and residential environments. By integrating mobile computing and wireless sensors, the system provides real-time data visualization and notifications when noise levels surpass safety limits. The research validates the role of IoT in ensuring acoustic comfort and suggests future advancements in noise regulation. Additionally, it demonstrates the impact of noise levels on employee productivity and health, emphasizing the need for continuous monitoring and intervention strategies. The study introduces a novel algorithm for adaptive noise control that dynamically adjusts noise thresholds based on time and location-specific factors. The authors present an in-depth evaluation of various noise mitigation strategies, including acoustic insulation and smart sound absorption techniques. The research also explores the effectiveness of noise zoning, where different areas within a workplace are categorized based on their acceptable noise limits. The integration of IoT-based noise control measures in smart office designs is also discussed. Furthermore, the study highlights the potential application of

wearable noise monitoring devices for personal noise exposure tracking, allowing employees to be aware of their daily noise intake. The research concludes that real-time noise monitoring systems can significantly reduce workplace stress and enhance cognitive efficiency by ensuring optimal acoustic environments.

4. M. B. Badruddin et al., "An IoT-Based Noise Monitoring System (NOMOS)" (2020) This study presents an IoT-powered noise monitoring system for university environments. Using cloud-based platforms, the system continuously records noise levels and provides insights into noise patterns affecting students' learning experiences. The authors emphasize the necessity of noise regulation in educational institutions and demonstrate the effectiveness of IoT solutions in enhancing academic environments. The study also evaluates the impact of noise disturbances on cognitive performance and proposes recommendations for noise mitigation policies. The proposed system integrates AI-driven analytics to differentiate between constructive and disruptive noise sources. The study highlights the role of real-time dashboards in visualizing noise trends, allowing administrators to implement timely corrective measures. The authors conduct a detailed case study within a university setting, showcasing the system's ability to reduce classroom disruptions. The research also explores the correlation between noise exposure and long-term health effects, emphasizing the need for regulatory policies to ensure minimal auditory stress. The proposed NOMOS system serves as a foundation for future innovations in educational noise control, providing a scalable solution adaptable to various academic institutions.
5. Aram Mohammad Abdulqadir & Mohammed Hussein Shukur, "Development of an IoT Noise Monitoring Network" (2017) This paper presents an IoT-based noise monitoring system designed to continuously measure and analyze noise levels in urban environments. The system is built using Raspberry Pi as the central processing unit. The study emphasizes the importance of continuous noise monitoring in densely populated areas to ensure compliance with noise pollution regulations. By leveraging IoT technology, the proposed system enables real-time data collection, remote access, and improved sensor accuracy, making it an effective solution for urban noise assessment. The research also highlights the need for enhanced sensor calibration, improved data visualization techniques, and machine learning integration to predict noise trends and improve decision-making.The study was conducted at the Department of Computer Science, Cihan University - Erbil, Iraq, and serves as a foundational framework for implementing scalable and efficient

noise monitoring solutions. The findings provide valuable insights into the challenges of urban noise control and suggest future enhancements, such as energy-efficient sensor networks and AI-driven noise classification for more accurate analysis.

Author(s), Year, Ref No.	**Title**	**Contribution**	**Challenge(s)**	**Key Difference with Proposed Work**
Siddhartha et al. (2023) [1]	Audio analysis for classroom noise detection	Improved noise classification in preschool environments	Handling high noise levels in dynamic settings	Focuses on classroom environments; the proposed work extends to broader real-time monitoring
Castillo de Valencia et al. (2023) [2]	Noise monitoring sensor network for university spaces	Developed a sensor-based network for indoor noise control	Deployment in large-scale university areas	Targets enclosed spaces, while the proposed work considers urban and educational environments
Marques & Pitarma (2020) [3]	IoT-based real-time noise monitoring system	Enhances acoustic comfort and occupational health	Handling real-time data fluctuations	Focuses on occupational health, while the proposed work aims at broader environmental monitoring
Badruddin et al. (2020) [4]	NOMOS: IoT-based noise monitoring system	Improves noise monitoring using IoT	Scalability and real-time processing issues	The proposed work incorporates AWS cloud-based noise analysis techniques for real-time adaptation
Abdulqadir & Shukur (2017) [5]	IoT noise monitoring network	Developed a wireless sensor network for noise tracking	Network stability and accuracy of measurements	Proposed work integrates AWS cloud-based data analysis techniques

Anachkova et al. (2023) [6]	Real-time IoT-based urban noise monitoring	Focuses on urban noise data collection	Handling data inconsistencies in large areas	The proposed work approach optimizes noise analysis with cloud computing and real-time data processing
Garcia et al. (2020) [7]	Wireless acoustic sensor network for urban noise	Applied spatial statistical analysis for noise mapping	Handling missing or erroneous data	Proposed work integrates AWS cloud-based data analysis techniques
Alsina-Pagès et al. (2017) [8]	HomeSound: Real-time audio event detection	Used HPC for real-time surveillance and behavior monitoring	Processing high-volume audio streams efficiently	The proposed work approach extends detection capabilities to diverse environments
Badruddin et al. (2020) [9]	IoT-based noise monitoring for educational institutions	Enhances acoustic comfort in schools and universities	Managing real-time large-scale noise data	Focuses on classroom environments; the proposed work extends to broader real-time monitoring
Jafari et al. (2019) [10]	Study on noise exposure effects on cognitive performance	Explored the cognitive impact of noise	Measuring and quantifying cognitive changes accurately	The proposed work research integrates noise impact assessment with real-time monitoring

Table 1 : Comparison of Related Research on Noise Monitoring Systems

The Table 1 provides a comparative analysis of existing research on noise monitoring systems, highlighting their methodologies, technologies used, and key findings. This comparison helps in identifying gaps and improvements that the proposed system addresses for enhanced efficiency and functionality.

CHAPTER 3

KEY HARDWARE

The Key Hardware section outlines the essential components used in the noise monitoring system, ensuring accurate noise detection and efficient data transmission. This chapter details the ESP32 microcontroller, microphone sensors, power modules, and communication interfaces, highlighting their roles in system functionality.

3.1 ESP32 Board

The ESP32 is a low-cost, power-efficient microcontroller with built-in Wi-Fi and Bluetooth capabilities, making it an ideal choice for IoT applications, including noise monitoring systems. Developed by espressif Systems, the ESP32 is equipped with a dual-core processor, ample GPIOs, and advanced features that support real-time data processing and wireless communication.One of the standout features of the ESP32 is its ability to handle multiple sensor inputs simultaneously while maintaining efficient power consumption. It includes an integrated Wi-Fi module, which allows seamless data transmission to cloud platforms like AWS IoT Core or Google Firebase for remote monitoring. Additionally, its Bluetooth functionality enables short-range wireless communication, which can be used for device pairing and data exchange with mobile applications

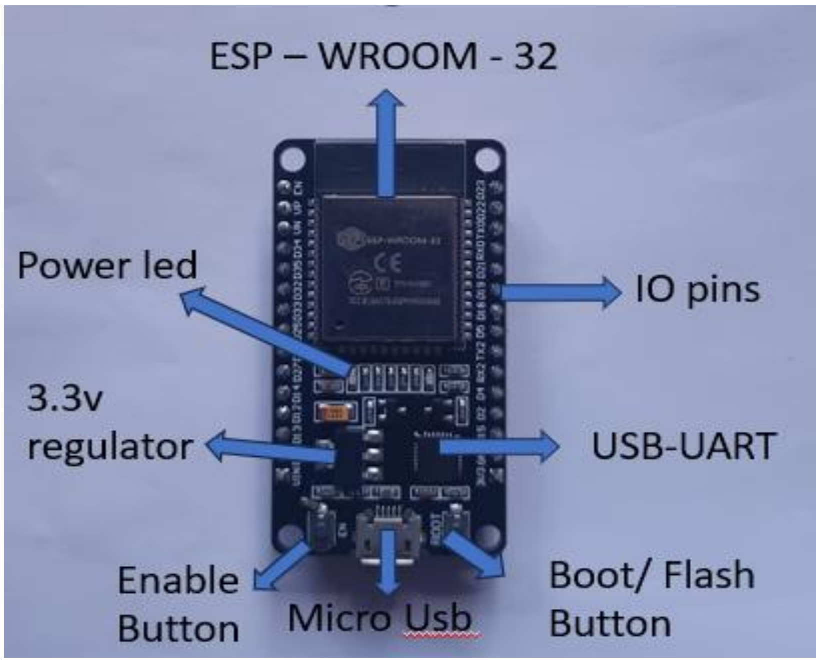


Fig 3.1 Components of ESP32 board

As shown in Fig 3.1 an ESP32 board have a USB-to-serial converter, Voltage regulator, ESP32-WROOM-32 module, GPIO pins, Boot and Reset buttons, Power LED indicator, Micro-USB port, PCB antenna, Capacitors and resistors, Crystal oscillator, Flash memory, UART interface, SPI interface, I2C interface, PWM output, ADC and DAC pins, Hall sensor, Capacitive touch inputs, Debugging ports, Power management circuitry.

3.1.1 Pin description of ESP32

The pin diagram of ESP32, as illustrated in Fig 3.2, provides a detailed overview of the available GPIOs and their functionalities. The ESP32 has a total of 38 GPIO pins, which support multiple functions such as ADC, DAC, UART, SPI, I2C, and PWM, making it highly versatile for various applications. Some pins are reserved for specific tasks, like power management and boot configuration, ensuring optimal performance and stability. The power supply pins (3.3V and GND) are used to provide stable voltage levels to the board and connected peripherals, preventing voltage fluctuations. Additionally, certain GPIOs support capacitive touch sensing, making them useful for touch-based applications such as interactive panels and smart switches. The ADC pins allow for precise analog input readings, whereas the DAC pins enable the generation of analog signals for audio and waveform applications. Several GPIOs are dedicated to serial communication, allowing seamless data transfer with external modules and sensors. Proper understanding and utilization of the pin diagram help developers maximize the ESP32's capabilities for IoT, automation, and real-time data processing applications.

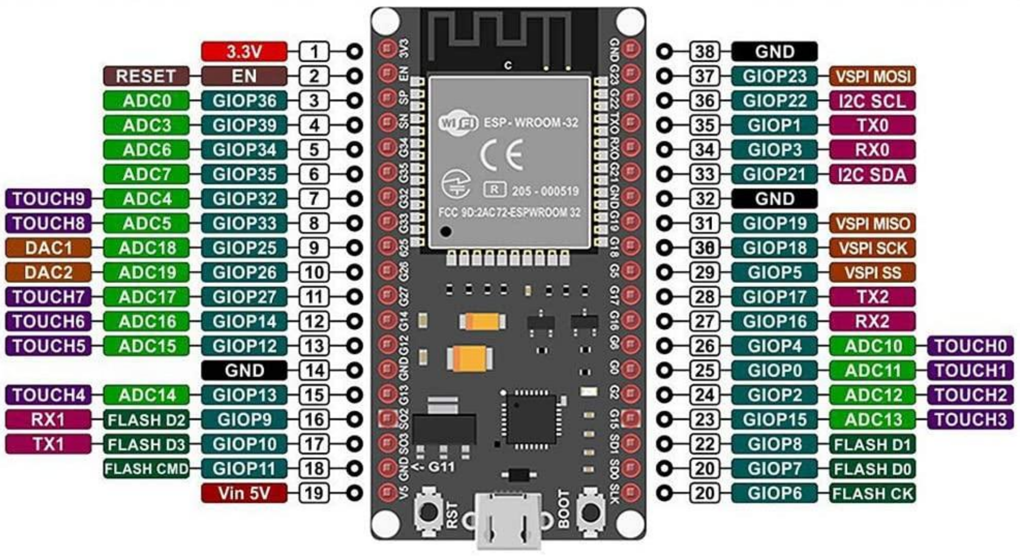


Fig 3.2 Pin diagram of ESP32 board

a) Power and Ground Pins

The ESP32 operates at 3.3V, and its power and ground pins play a crucial role in ensuring stable operation. The 3V3 (3.3V power supply) pin provides regulated power to the ESP32 and can also be used to power external components that require 3.3V. The VIN (Voltage Input) pin allows the ESP32 to be powered using a 5V external source, which is internally regulated to 3.3V for safe operation. Multiple GND (ground) pins are available to provide a common ground reference for all components in the circuit. These power and ground connections are essential for maintaining circuit stability and ensuring proper voltage levels for peripherals connected to the ESP32.

b) General Purpose Input/Output (GPIO) Pins

The ESP32 features 34 GPIO pins, which can be used for both input and output operations. These GPIOs allow the ESP32 to interact with sensors, actuators, and other external devices. Some GPIOs have special functionalities, such as supporting Pulse Width Modulation (PWM), Analog-to-Digital Conversion (ADC), and various communication protocols like SPI, I2C, and UART. However, some pins are restricted during boot-up or when Wi-Fi is active, so developers must carefully select GPIOs based on their project requirements. GPIOs can be programmed as digital inputs to read signals from buttons or sensors, or as digital outputs to control LEDs, relays, and motors.

c) Analog Input Pins (ADC - Analog-to-Digital Converter)

The ESP32 has a 12-bit ADC resolution, allowing it to convert analog signals into digital values with a range from 0 to 4095 levels. This feature is particularly useful for reading sensor data such as temperature, humidity, light intensity, and voltage levels. The ADC is divided into ADC1 and ADC2. While ADC1 (GPIOs 32 to 39) can be used freely for analog input, ADC2 (GPIOs 0, 2, 4, 12-15, 25-27) is shared with the Wi-Fi module, meaning that it cannot be used when Wi-Fi is enabled. Because of this limitation, it is recommended to use ADC1 pins for analog sensing applications in projects that require simultaneous Wi-Fi communication.

d) Digital-to-Analog Converter (DAC) Pins

The ESP32 features two DAC channels that allow the conversion of digital signals into analog voltages. These DAC pins, DAC1 (GPIO 25) and DAC2 (GPIO 26), are useful in applications that require the generation of analog signals, such as audio waveform generation, sound synthesis, and voltage-controlled devices. The 8-bit DAC resolution provides a smooth analog output, making it suitable for audio applications, signal modulation, and power control systems. Unlike PWM, which creates an analog-like signal using rapid on/off switching, the DAC provides a true analog voltage output, making it preferable for applications requiring precise voltage control.

e) Touch Sensor Pins

The ESP32 includes capacitive touch sensors, allowing it to detect touch-based inputs without the need for mechanical buttons. These touch-sensitive GPIOs can measure small changes in capacitance when touched by a finger or conductive object. The ESP32 supports up to 10 touch-sensitive GPIOs (Touch0 to Touch9, mapped to GPIOs 0, 2, 4, 12-15, 27, 32, and 33). This feature is commonly used in applications such as gesture-based control panels, interactive devices, and smart touch-switches. The built-in capacitive sensing feature helps reduce external hardware requirements, simplifying the design of user-friendly interfaces.

f) Pulse Width Modulation (PWM) Pins

Pulse Width Modulation (PWM) is a technique used to control the power supplied to electronic devices by varying the duty cycle of a signal. The ESP32 has a highly flexible PWM implementation, as any GPIO pin can be configured as a PWM output. This allows precise control over LED brightness, motor speed, and audio signal generation. The ESP32 supports high-frequency PWM with up to 16-bit resolution, making it useful for fine-tuned power adjustments in industrial automation, robotics, and home automation projects.

g) Serial Communication Pins

The ESP32 provides multiple options for serial communication, allowing it to interface with other microcontrollers, sensors, and external modules. It supports UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit) protocols. UART0 (GPIO 1 for TX and GPIO 3 for RX) is mainly used for debugging and programming, while UART1 and UART2 can be used for connecting serial devices such as GPS modules, GSM modems, and RFID readers. SPI communication (MOSI, MISO, SCLK, and CS on GPIOs 23, 19, 18, and 5) is used for high-speed data exchange with devices such as SD cards, displays, and memory modules. The I2C interface (SDA - GPIO 21 and SCL - GPIO 22) allows easy communication with multiple sensors and peripheral modules, making it widely used in IoT applications.

h) Interrupt Pins

Interrupts allow the ESP32 to respond immediately to external events, such as button presses or sensor triggers. Almost all GPIO pins support interrupts, making it easy to configure event-driven tasks in low-power applications. Interrupts can be set to trigger on rising, falling, or both edges of an input signal, enabling efficient power management and quick response to sensor inputs. In battery-powered applications, interrupts are often used to wake up the ESP32 from sleep mode only when necessary, reducing unnecessary energy consumption.

i) Boot and Flashing Pins

Some GPIOs have special functions related to booting and firmware flashing, and they must be handled carefully when designing hardware circuits. GPIO 0 is crucial for entering the bootloader mode, where the ESP32 can receive new firmware. It must be held LOW to enable flashing mode and HIGH for normal operation. GPIO 2 and GPIO 15 also influence boot settings and should be properly configured to avoid startup issues. Misusing these boot-related pins may result in boot failures or the inability to flash firmware.

j) Strapping Pins

Strapping pins determine boot configurations and operating modes of the ESP32 during startup. Pins such as GPIO 0, 2, 4, 5, 12, and 15 are used for selecting options like flash voltage selection, debugging modes, and Wi-Fi coexistence settings. Developers must ensure that these pins are correctly set to avoid unintended behavior, especially when designing PCBs or custom ESP32 modules. Understanding the function of strapping pins is important for troubleshooting startup and connectivity issues.

The ESP32 microcontroller offers a versatile set of features, including built-in Wi-Fi and Bluetooth, low power consumption, multiple GPIOs, and support for various communication protocols, making it ideal for IoT applications.

i. **Dual-Core Processor** – The ESP32 features a dual-core Xtensa LX6 processor, which allows it to handle multiple tasks simultaneously. This makes it ideal for real-time applications where multitasking is essential, such as IoT, automation, and signal processing.

ii. **Wi-Fi and Bluetooth Connectivity**- The ESP32 integrates both 2.4 GHz Wi-Fi and Bluetooth, supporting both Bluetooth Classic and Bluetooth Low Energy (BLE). This dual wireless capability makes it a preferred choice for IoT applications, where devices need seamless connectivity.

iii. **High-Speed Performance** The ESP32 operates at clock speeds of up to 240 MHz, allowing it to execute complex tasks quickly and efficiently. This high-speed processing capability makes it suitable for AI applications, robotics, and industrial automation where real-time performance is crucial. It supports advanced algorithms for signal processing, making it ideal for audio applications, motor control, and real-time analytics.

iv. **Low Power Consumption** The ESP32 is designed with multiple power-saving modes, making it highly energy-efficient. It supports deep sleep, light sleep, and modem sleep modes, which significantly reduce power consumption when the device is idle.

v. **Multiple GPIOs and Interfaces** – The ESP32 offers multiple GPIO (General Purpose Input/Output) pins, making it highly flexible for integrating sensors, actuators, and other peripherals. The wide range of communication options makes it suitable for diverse applications, from smart home automation to industrial process control. Developers can use these interfaces to build scalable and modular IoT projects with ease.Secure Boot and Encryption – Features like flash encryption and secure boot enhance device security, making it suitable for applications requiring data protection.

vi. **PSRAM and Flash Memory** – Some variants of the ESP32, such as the ESP32-WROVER, come equipped with additional PSRAM (Pseudo-Static RAM), enhancing the device's memory capabilities. PSRAM is useful for applications that require large data storage, such as image processing, AI-based applications, and real-time analytics.

vii. **Integrated Hall Sensor and Capacitive Touch Inputs** The ESP32 includes a built-in hall sensor, which detects magnetic fields, making it useful for applications such as door sensors, motor control, and proximity detection. It also features capacitive touch-sensitive GPIOs, enabling touch-based interactions without the need for additional hardware

viii. **Multiple Analog and Digital Interfaces** – The ESP32 provides high-resolution ADC (Analog-to-Digital Converter) and DAC (Digital-to-Analog Converter) capabilities. The ADC allows the device to read analog signals from sensors such as temperature, humidity, and light sensors. The DAC enables the ESP32 to generate analog signals, making it useful for applications such as audio output and waveform generation.

ix. **Support for RTOS and Multi-threading** – The ESP32 supports FreeRTOS (Real-Time Operating System), enabling multi-threaded processing and real-time task scheduling. The multi-threading capability of the ESP32 enhances performance in complex systems, allowing it to handle networking, sensor data acquisition, and cloud communication simultaneously.

3.1.2 Major Types of ESP32 Boards

The ESP32 microcontroller comes in various variants, each designed to cater to different application requirements. These variations differ in processing power, memory capacity, connectivity options, and power efficiency, making them suitable for diverse IoT implementations**.**

a) **ESP32 Dev Kit V1** – One of the most commonly used ESP32 development boards, the ESP32 Dev Kit V1 shown in Fig 3.3 is widely preferred for prototyping and IoT applications. It features a built-in USB-to-serial converter, making programming and debugging easier. The board provides multiple GPIOs, supports both Wi-Fi and Bluetooth, and is compatible with various sensors and peripherals. Its compact size and ease of use make it a popular choice for developers working on IoT and embedded systems projects. The Dev Kit V1 includes an onboard voltage regulator, allowing it to operate efficiently with different power sources. Its compatibility with Arduino IDE and MicroPython enables beginners to quickly develop and test applications. The board supports SPI, I2C, and UART interfaces, making it adaptable to various communication requirements. It is widely used in smart home automation, industrial monitoring, and educational projects due to its affordability and extensive documentation.

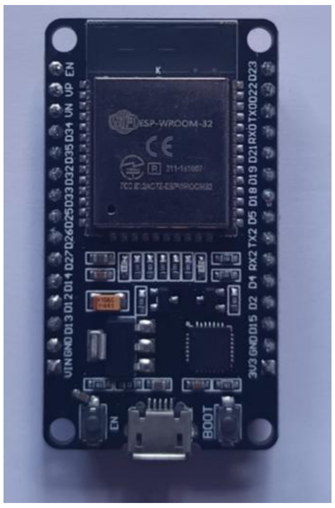

ESP-WROOM-32
EN
BOOT
VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23

Fig 3.3 ESP32 Dev kit v1 board

b) **ESP32-WROVER** – This version includes additional PSRAM (Pseudo Static RAM), making it ideal for applications requiring higher memory capacity, such as real-time audio processing and complex AI models. The additional PSRAM allows developers to handle large datasets, making it suitable for image processing, speech recognition, and deep learning applications. The ESP32-WROVER shown in Fig 3.4 offers better performance in applications where multiple processes run concurrently. Its high-speed SPI interface ensures fast data transfers, crucial for tasks like video streaming and real-time sensor data acquisition.

Fig 3.4 ESP32 wrover board

c) **ESP32-S2** –The ESP32-S2 shown in Fig 3.5 is a single-core variant with lower power consumption and improved security features. While it lacks Bluetooth, it is ideal for low-power, Wi-Fi-only applications.Developers benefit from its large number of GPIOs, which enable integration with multiple sensors and actuators. Its deep sleep mode allows devices to run on small batteries for extended periods, making it ideal for energy-efficient designs.

Fig 3.5 ESP32- S2 Board

d) **ESP32-C3** – The ESP32-C3 shown in Fig 3.6 is based on the RISC-V architecture, this variant offers a cost-effective alternative with enhanced security, suitable for lightweight IoT applications. Unlike other ESP32 variants, the ESP32-C3 features a single-core processor but retains Wi-Fi and Bluetooth LE (Low Energy) supportThis module is preferred for low-cost, mass-production IoT applications due to its affordability and efficiency.

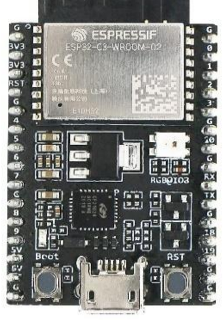


Fig 3.6 ESP32- C3 board

e) **ESP32-S3** – The ESP32-S3 shown in Fig 3.7 is an upgraded version of ESP32 with AI acceleration, making it ideal for machine learning and voice recognition applications. Developers can utilize esp-DSP libraries to optimize digital signal processing (DSP) tasks, improving the efficiency of audio and video applications.

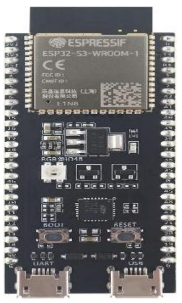


Fig 3.7 ESP32- S3 board

The selection of an ESP32 board depends on the specific requirements of the noise monitoring system, such as processing power, memory constraints, and power efficiency. The ESP32-Dev Kit is generally preferred for IoT-based noise monitoring due to its balance of performance, cost, and ease of integration with various sensors.

3.2 Microphone Sensors

Microphone sensors are essential components in noise monitoring systems as they capture sound waves and convert them into electrical signals for further processing. Different types of microphone sensors are used based on their sensitivity, amplification, and noise filtering capabilities. The commonly used microphone sensors in IoT-based noise monitoring applications include KY-037, KY-038, MAX4466, and MAX9814 shown in Fig 3.8.

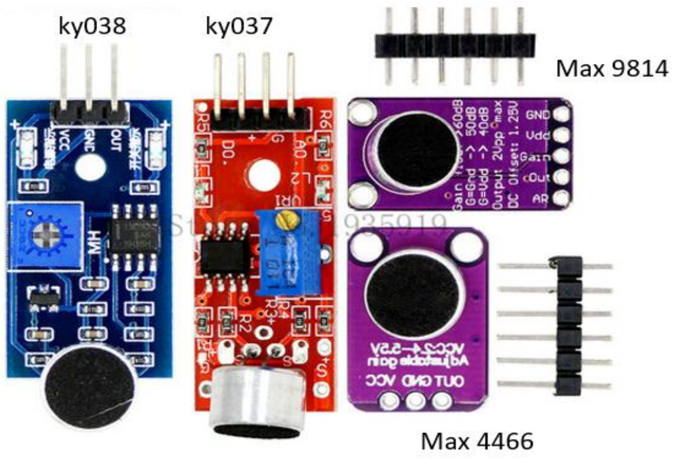


Fig 3.8 commonly used microphone sensors

a) **KY-037 Microphone Sensor** The KY-037 shown in Fig 3.8 is a highly sensitive microphone sensor module designed for sound detection applications. It provides both analog and digital outputs, allowing easy interfacing with microcontrollers like Arduino and ESP32. The module features an onboard LM393 comparator, enabling real-time noise detection.

   Advantages: High sensitivity, dual output (analog and digital), adjustable sensitivity via potentiometer, simple integration with microcontrollers, suitable for loud noise detection.

   Disadvantages: Prone to background noise interference, not ideal for capturing low-volume sounds, requires fine-tuning for optimal performance.

b) **KY-038 Microphone Sensor** The KY-038 microphone sensor is similar to the KY-037 but features a slightly smaller microphone component, making it less sensitive to distant soundsThis sensor is widely used in automatic sound detection applications such as home automation, voice control systems, and sound-triggered alarms.

   Advantages: Compact size, dual output (analog and digital), adjustable sensitivity, cost-effective, easy to integrate with IoT systems.

   Disadvantages: Lower sensitivity compared to KY-037, more susceptible to environmental noise, not suitable for precise audio processing.

c) **MAX4466 Microphone Sensor** The MAX4466 is a high-performance electret microphone amplifier module specifically designed for low-noise applicationsThe MAX4466 is commonly used in applications such as voice recognition, noise monitoring, and sound analysis projects where precise audio input is necessary.

   Advantages: Low-noise amplifier, adjustable gain control, excellent sound clarity, ideal for voice recognition and precise noise measurements.

   Disadvantages: Requires external power supply for optimal performance, limited dynamic range, more expensive than KY-series microphones.

d) **MAX9814 Microphone Sensor** The MAX9814 shown in Fig 3.8 is a highly advanced microphone sensor featuring an Automatic Gain Control (AGC) amplifier. The MAX9814 is widely used in professional audio equipment and IoT-based sound detection systems.

   Advantages: Automatic Gain Control (AGC) for balanced sound levels, high sensitivity, low distortion, suitable for varying noise environments.

Disadvantages: Complex calibration, requires stable power supply, higher power consumption compared to passive microphones.

3.3 Power Supply

The 3.7V lithium-ion (Li-Ion) battery is widely used in portable electronic devices due to its high energy density, long cycle life, and lightweight nature. It provides a stable voltage supply for IoT applications, including noise monitoring systems, ensuring efficient operation of microcontrollers and sensors. These batteries come in various capacities, typically ranging from 500mAh to several thousand mAh, allowing customization based on power requirements.

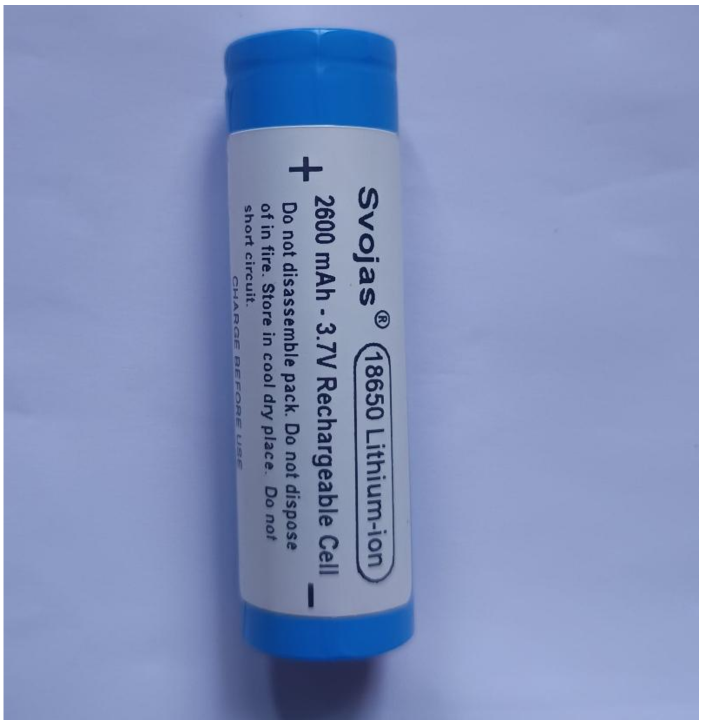

Svojas®
18650 Lithium-ion
2600 mAh - 3.7V Rechargeable Cell
Do not disassemble pack. Do not dispose of in fire. Store in cool dry place. Do not short circuit.
CHARGE BEFORE USE
+
-

Fig 3.9 3.7v li-ion battery

Features of 3.7V Li-Ion Battery

i. High Energy Density: Li-Ion batteries store more energy per unit weight compared to other rechargeable battery types.

ii. Low Self-Discharge Rate: They retain charge for extended periods when not in use.

iii. Rechargeable and Long Cycle Life: Can undergo multiple charge-discharge cycles, making them cost-effective.

iv. Stable Output Voltage: Ensures consistent power supply for connected components.

v. Compact and Lightweight: Ideal for portable and embedded systems.

vi. Overcharge and Overdischarge Protection: Many Li-Ion batteries include built-in safety circuits to prevent damage.

vii. Fast Charging Capability: Supports rapid energy replenishment, reducing downtime in battery-powered systems.

viii. Eco-Friendly Option: Compared to older battery technologies, Li-Ion batteries have lower environmental impact when disposed of properly.

Despite their advantages, Li-Ion batteries require proper handling to prevent overheating, swelling, or potential hazards. Charging should always be done using a dedicated module, such as the TP4056, to ensure safe and efficient energy replenishment.

3.4 Charging Module: TP4056

The TP4056 shown in Fig 3.10 is a widely used charging module for 3.7V Li-Ion batteries. It is a linear charger designed for single-cell lithium batteries and provides a safe, regulated charging process. The module includes built-in overcharge, overcurrent, and short-circuit protection, making it a reliable choice for battery-powered IoT systems.

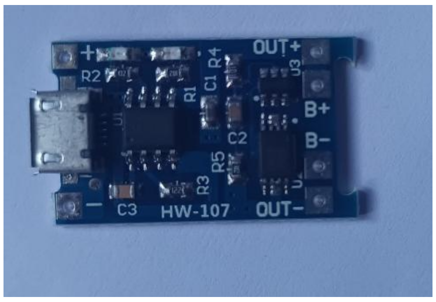


Fig 3.10 TP4056 Charging module

Features of TP4056 Charging Module

i. **Constant Current/Constant Voltage (CC/CV) Charging:** Ensures efficient battery charging with minimal stress.

ii. **Built-in Protection Circuits:** Prevents overcharging, overdischarging, and short circuits, extending battery life.

iii. **Micro-USB Input:** Supports convenient charging via standard USB power sources.

iv. **Charge Status Indicators:** LED indicators display charging progress and completion.

v. **Automatic Recharge:** The module detects when battery voltage drops and recharges accordingly.

vi. **Compact Design:** Small form factor allows easy integration into IoT projects.

vii. **Thermal Regulation:** Controls temperature during charging to prevent overheating and ensure safety.

viii. **Adjustable Charging Current:** The module allows for fine-tuning of charging current based on battery capacity.

ix. **Wide Compatibility:** Can be used with various Li-Ion battery sizes, making it a versatile solution for different applications.

x. **Reverse Polarity Protection:** Prevents damage in case of incorrect battery connection.

3.4.1 Working of TP4056 Charging Module

The TP4056 module works by supplying a controlled current to charge a connected Li-Ion battery. Initially, the battery is charged at a constant current (CC) phase until it reaches a predefined voltage threshold. Once this voltage is reached, the module switches to the constant voltage (CV) phase, gradually reducing the current flow while maintaining the battery at full charge.This functionality makes the module ideal for IoT applications where continuous operation is necessary.

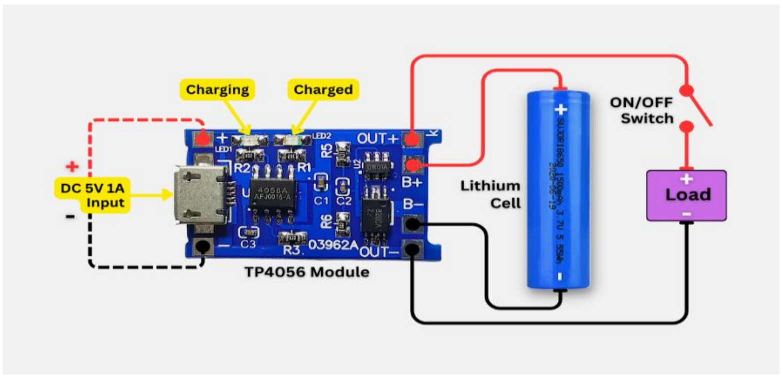


Fig 3.11 Circuit diagram of TP4056 Charging module

Fig 3.11 showcases a TP4056 lithium battery charging module, a compact circuit board designed for safely charging single-cell lithium-ion batteries. It features a micro USB input for power, LED indicators for charging status (red for charging, green for charged), and terminals for connecting the battery and a load. The accompanying diagram illustrates a basic circuit with a lithium cell connected to the module, an ON/OFF switch, and a load, demonstrating how the module facilitates charging and powering devices. The TP4056 chip at the core of the module ensures proper constant-current/constant-voltage (CC/CV) charging, protecting the battery and ensuring efficient operation.

The combination of a 3.7V Li-Ion battery and the TP4056 charging module provides an efficient, safe, and portable power solution for noise monitoring systems and other IoT applications. These components ensure stable power delivery, enhanced battery longevity, and safety features to prevent overcharging and overheating.

Applications of 3.7V Li-Ion Battery and TP4056

i. **IoT Devices:** Provides efficient power for smart home automation, sensors, and embedded systems.
ii. **Wearable Electronics:** Used in fitness trackers, smartwatches, and medical monitoring devices.
iii. **Portable Audio Systems:** Powers Bluetooth speakers, wireless microphones, and hearing aids.
iv. **Drones and RC Vehicles:** Provides lightweight and high-energy output for drones and remote-controlled devices.
v. **Solar-Powered Systems:** Used in energy storage solutions for solar-based projects.
vi. **Emergency Backup Power:** Ensures uninterrupted power supply for critical applications.

CHAPTER -4

AMAZON WEB SERVICES

The Amazon Web Services (AWS) section explains the cloud-based infrastructure used for processing noise data. This chapter covers key AWS services such as AWS IoT Core, AWS Lambda, AWS SNS, and AWS CloudWatch, which enable real-time monitoring, automated alerts, and secure data management.

4.1 Introduction to AWS

Amazon Web Services (AWS) is one of the world's leading cloud computing platforms, offering a broad range of services designed to provide flexible, scalable, and cost-effective solutions for various industries. AWS was launched by Amazon in 2006 and has since revolutionized the way businesses, governments, and individuals use cloud computing. The platform provides access to computing power, networking, analytics, machine learning, and IoT services, making it an essential tool for modern applications, including noise monitoring systems. AWS operates on a global infrastructure, with multiple data centers across various regions, ensuring high availability and reliability. One of its key advantages is the ability to scale resources up or down based on demand, eliminating the need for businesses to invest in expensive on-premises hardware. By leveraging AWS, organizations can focus on their core operations while relying on Amazon's robust and secure infrastructure to handle their computing needs.

AWS follows a pay-as-you-go pricing model, allowing users to pay only for the resources they consume. This cost-efficient approach makes AWS an attractive option for startups, enterprises, and government agencies looking to optimize their IT spending. The platform offers multiple service categories, including computing, storage, databases, artificial intelligence, and IoT, providing end-to-end solutions for various applications. One of the most significant benefits of AWS is its security and compliance capabilities. AWS adheres to strict security standards, offering encryption, identity access management, and multi-factor authentication to protect user data. The platform is compliant with various regulatory requirements, including GDPR, HIPAA, and ISO 27001, making it suitable for businesses operating in highly regulated industries.

4.2 Key AWS services used for noise monitoring systems

AWS provides a robust cloud infrastructure that enables efficient real-time noise monitoring and data processing. By leveraging AWS services, the system ensures seamless device connectivity, secure data transmission, and automated alert mechanisms. The following key AWS services play a crucial role in enhancing the functionality of the noise monitoring system.

a) AWS IoT Core

AWS IoT Core is a managed cloud service that allows connected devices to interact securely with cloud applications and other devices. It provides seamless connectivity for billions of IoT devices, supporting MQTT, HTTP, and WebSockets protocols for efficient data transmission. In noise monitoring systems, AWS IoT Core plays a crucial role by enabling real-time data collection from noise sensors and processing that data in the cloud for further analysis.One of the key features of AWS IoT Core is its device shadow functionality, which maintains a virtual representation of physical devices. This allows applications to interact with sensors even when they are offline, ensuring continuous monitoring without data loss. The service also supports rules engine functionality, enabling automated actions when certain noise thresholds are exceeded.

By using AWS IoT Core, organizations can deploy scalable and cost-effective noise monitoring solutions that process and analyse sound levels in real-time. It enables smart automation, reduces manual intervention, and ensures timely responses to noise threshold breaches.

b) AWS Simple Notification Service (SNS)

AWS Simple Notification Service (SNS) is a highly reliable, fully managed messaging service that enables the delivery of notifications through various communication channels, including SMS, email, and mobile push notifications. In noise monitoring systems, AWS SNS plays a critical role by providing instant alerts when noise levels exceed predefined limits. AWS SNS follows a publish-subscribe model, allowing multiple recipients to receive alerts simultaneously. This is particularly useful in industrial and urban noise monitoring applications, where multiple stakeholders—such as regulatory authorities, environmental agencies, and property managers—need to be informed immediately.

By leveraging AWS SNS, noise monitoring systems can enhance response times, streamline communication, and ensure prompt corrective measures are taken when noise levels exceed acceptable thresholds.

c) AWS Lambda

AWS Lambda is a serverless computing service that enables developers to run code without provisioning or managing servers. It automatically scales based on demand and executes code in response to predefined triggers. In noise monitoring systems, AWS Lambda plays a vital role by processing incoming data from IoT sensors and automating necessary actions when noise levels exceed predefined thresholds.

One of the primary use cases of AWS Lambda in noise monitoring is event-driven automation. When AWS IoT Core detects excessive noise, it triggers a Lambda function that processes the data, determines if an alert is necessary, and then sends notifications via AWS SNS. This eliminates the need for manual intervention and ensures a seamless, real-time response to noise violations.

AWS Lambda enhances the overall efficiency of noise monitoring systems by enabling automated responses, reducing operational overhead, and ensuring continuous monitoring without requiring dedicated infrastructure.

d) Amazon CloudWatch

Amazon CloudWatch is a comprehensive monitoring and observability service that provides real-time visibility into AWS resources and applications. In noise monitoring systems, CloudWatch plays a critical role in logging, analyzing, and visualizing noise data, allowing administrators to track trends and optimize system performance.One of the primary functions of CloudWatch in noise monitoring is real-time alerting. By collecting logs from AWS IoT Core and AWS Lambda, CloudWatch can generate alarms when noise levels exceed predefined thresholds. These alarms can then trigger automated actions, such as invoking an AWS Lambda function or sending alerts through AWS SNS.

Another major benefit of CloudWatch is anomaly detection, which uses machine learning models to identify unusual noise patterns. This feature enhances noise monitoring by proactively detecting potential disturbances before they become significant issues. Furthermore, CloudWatch Metrics allow organizations to track system health and performance, ensuring optimal operation of noise monitoring solutions.By integrating Amazon CloudWatch into a noise monitoring system, organizations can gain actionable insights, improve system reliability, and ensure compliance with noise regulations.

e) AWS Identity and Access Management (IAM)

AWS Identity and Access Management (IAM) is a critical security service that controls access to AWS resources. In noise monitoring systems, IAM ensures that only authorized personnel can access and modify system settings, preventing unauthorized changes and potential security threats.One of the key features of AWS IAM is role-based access control (RBAC). Administrators can define user roles and assign specific permissions, ensuring that users only have access to the resources necessary for their tasks. For example, sensor data analysts may have read-only access to CloudWatch logs, while system administrators have full control over AWS IoT Core configurations.IAM also supports multi-factor authentication (MFA), adding an extra layer of security to prevent unauthorized access. Additionally, IAM policies can be used to enforce fine-grained permissions, restricting access based on conditions such as IP address, device type, or time of access.

Another advantage of IAM is its integration with AWS CloudTrail, which logs all access requests and modifications, providing an audit trail for compliance and security analysis. This ensures accountability and helps organizations identify potential security risks.By leveraging AWS IAM, noise monitoring systems can enhance security, enforce strict access controls, and ensure that sensitive environmental data is protected from unauthorized access or modifications.

4.3 Advanced Features of AWS for Noise Monitoring Systems

AWS provides a comprehensive cloud-based infrastructure for real-time noise monitoring, offering numerous advantages in terms of scalability, security, automation, and cost efficiency. The following sections elaborate on how AWS enhances noise monitoring systems through scalability, real-time processing, security, automated alerts, cost efficiency, seamless integration, and remote accessibility.

a) Scalability

Scalability is one of the key benefits of using AWS for noise monitoring systems. AWS services dynamically allocate resources based on the volume of incoming noise data, ensuring smooth performance without manual intervention. This elasticity prevents performance bottlenecks and downtime, making the system highly responsive. AWS also allows users to scale storage and processing power as needed, ensuring that the system remains efficient even as the number of deployed noise sensors grows over time.

b) Real-Time Processing

AWS ensures that noise monitoring operates with minimal latency through real-time data processing. Furthermore, AWS CloudWatch can be used to analyze trends in noise levels, allowing for predictive analytics and proactive decision-making to prevent disturbances before they occur.

c) Security and Compliance

Security is a critical factor in cloud-based noise monitoring systems, especially when handling sensitive environmental data. Additionally, AWS enables encryption of data at rest and in transit, ensuring compliance with global security standards such as ISO 27001, GDPR, and HIPAA. With AWS security mechanisms in place, noise monitoring data is protected from cyber threats, unauthorized modifications, and data breaches, ensuring high reliability and trustworthiness.

d) Automated Alerts

AWS enables automated notifications through AWS Simple Notification Service (SNS), ensuring that stakeholders receive instant alerts when noise levels exceed predefined thresholds. These alerts can be configured to notify multiple recipients through email, SMS, or push notifications, ensuring that the right people are informed in real time. This feature is particularly useful in environments such as schools, hospitals, office spaces, and industrial areas, where excessive noise can be disruptive or hazardous. By leveraging automated alerts, users can take immediate corrective action, such as adjusting noise policies, alerting security personnel, or activating noise suppression systems.

e) Cost Efficiency

One of the most attractive benefits of AWS is its pay-as-you-go pricing model, which eliminates the need for expensive hardware infrastructure. Additionally, AWS reduces operational costs by eliminating the need for dedicated servers and IT maintenance, as all processing is managed within the AWS cloud. Organizations and governments implementing noise monitoring solutions can scale their systems without worrying about costly upgrades or maintenance overheads.

f) Seamless Integration

AWS services are designed to work together effortlessly, providing a fully automated and efficient noise monitoring system. AWS IoT Core, AWS Lambda, AWS SNS, AWS IAM, and AWS CloudWatch integrate smoothly to create a unified ecosystem. The flexibility of AWS allows developers to build customized workflows, automate noise reporting, and integrate noise control mechanisms such as automated soundproofing and noise mitigation strategies.

g) Remote Accessibility

AWS offers the advantage of cloud-based remote access, allowing users to monitor noise levels and manage alerts from anywhere in the world. This remote accessibility is particularly beneficial for government agencies, environmental organizations, and smart city projects, allowing them to monitor urban noise pollution without being physically present at the location. Additionally, AWS supports multi-user access control, ensuring that different teams can view and manage noise data based on their specific roles and permissions.

CHAPTER -5

EXISTING SYSTEM

The existing noise monitoring and control system within university environments utilizes an Internet of Things (IoT)-based approach to assess and mitigate excessive noise levels. This system is designed to enhance the learning experience by ensuring an optimal acoustic environment in enclosed spaces such as classrooms. Excessive noise has been identified as a major factor affecting students' concentration, comprehension, and overall academic performance.

5.1 Overview of the Existing System

Existing system consists of a network of sound sensors strategically placed within classrooms to capture real-time noise levels. These sensors communicate with a centralized database where data is analyzed, logged, and visualized through graphical representations. The system is programmed to issue notifications when noise levels exceed predetermined thresholds, prompting immediate corrective actions.

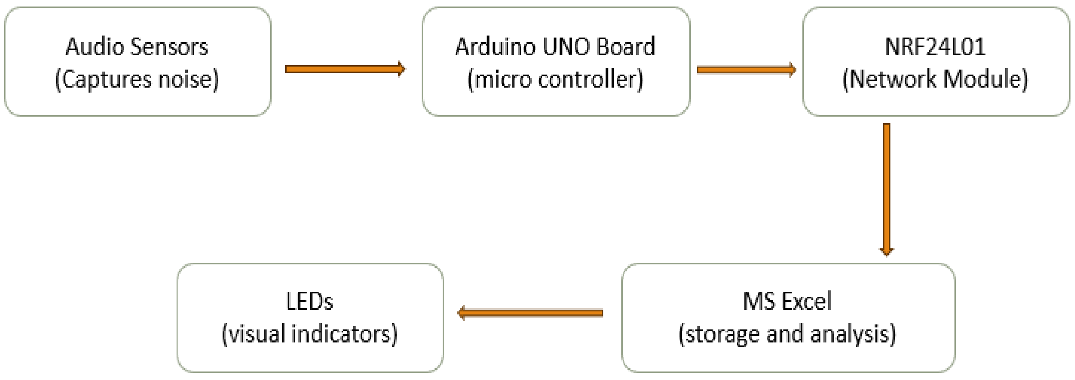


Fig 5.1 workflow of existing methodology

The workflow of the noise monitoring system, as represented in the Fig 5.1 involves multiple hardware and software components working together to capture, transmit, analyze, and display noise levels. Below is a step-by-step explanation of the system's operation.

a) Noise Capture Using Audio Sensors

The system starts with **audio sensors** (such as KY-037), which are strategically placed in enclosed university spaces to monitor ambient noise levels. These sensors detect sound intensity in the environment and convert the acoustic signals into electrical signals for further processing.

b) Signal Processing by Arduino UNO Board

The Arduino UNO Board (microcontroller) receives the electrical signals from the audio sensors. It processes the data to determine whether the noise level exceeds predefined thresholds. If the noise level is too high, the Arduino triggers a response in real time, such as activating visual indicators (LEDs).

c) Wireless Data Transmission via NRF24L01

After processing the noise levels, the Arduino board transmits the data wirelessly using the NRF24L01 network module. This module ensures efficient communication between the noise monitoring hardware and the data storage system. The NRF24L01 sends noise level data to a central computer or data storage unit for further analysis.

d) Data Storage and Analysis in MS Excel

The noise data received via the NRF24L01 module is stored in MS Excel for detailed analysis. The system logs noise levels over time, enabling the generation of graphical representations such as noise trends, peak noise periods, and statistical summaries. This helps university administrators and faculty assess classroom noise levels and take corrective measures.

e) Real-time Visual Indication Using LEDs

To provide immediate feedback on noise levels, LEDs (visual indicators) are directly connected to the Arduino board. These LEDs change color or blink to indicate noise intensity:

Green LED → Noise is within acceptable limits.

Yellow LED → Moderate noise, caution needed.

Red LED → Excessive noise, immediate action required.

5.2 Challenges Faced by the Existing Methodology

While the current noise monitoring system effectively captures, transmits, and analyzes noise data, several challenges limit its efficiency and accuracy. These challenges can be categorized into hardware limitations, data processing issues, and real-time monitoring constraints.

a) Limited Sensitivity of Audio Sensors

The audio sensors used in the system may not capture low-intensity noises accurately, leading to incomplete data collection.Background noise interference can sometimes result in false readings.

b) Limited Processing Power of Arduino UNO

The Arduino UNO board has limited computational capability, making it difficult to handle large-scale real-time processing or advanced noise filtering techniques.Complex noise analysis requires external processing tools, adding to the system's complexity.

c) Limited Wireless Transmission Range (NRF24L01 Module)

The NRF24L01 has a limited range of communication, making it unsuitable for large university campuses with multiple enclosed spaces.Obstacles such as walls and furniture may weaken the signal strength, leading to data loss.

d) Lack of Advanced Data Analysis Features in MS Excel

MS Excel, used for data storage and analysis, does not provide real-time noise pattern detection or predictive analytics.The system lacks integration with machine learning models, which could help in identifying trends and suggesting corrective actions.

e) Data Overload and Management Issues

As the system collects more data, managing and analyzing large datasets in Excel becomes challenging.The system does not have an automatic alert system (e.g., SMS or email notifications) to notify administrators about excessive noise levels.

f) Limited Real-Time Alerts

The LED indicators provide a visual alert but do not offer audible alarms or notifications to relevant stakeholders, such as university administrators.There is no mobile or web-based interface for remote monitoring.

g) No Automatic Noise Control Mechanism

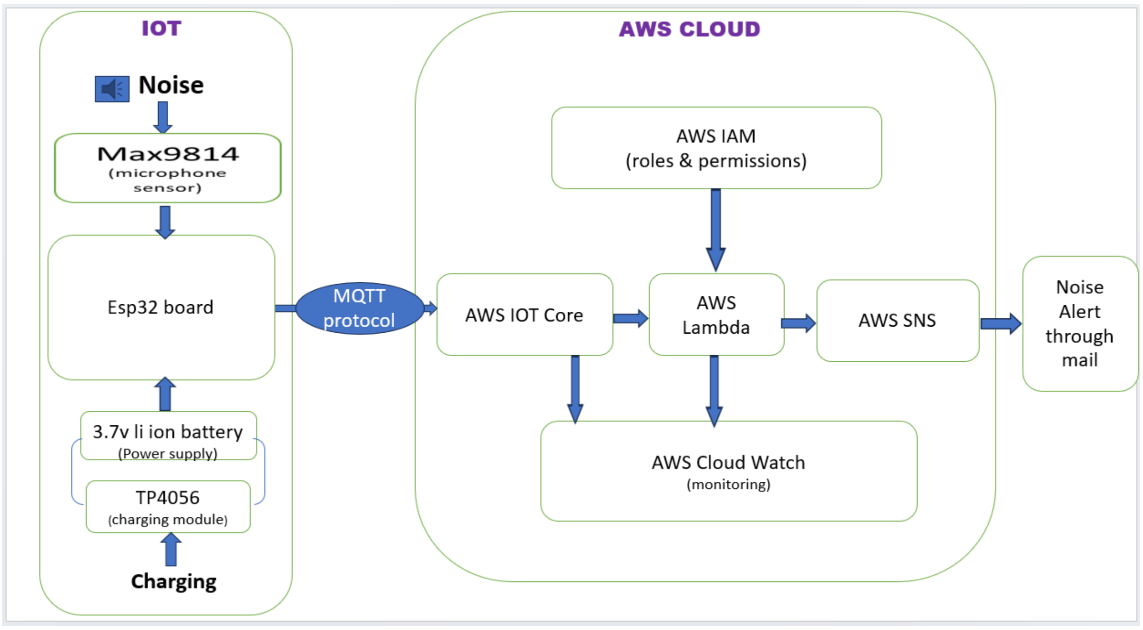
While the system detects excessive noise, it does not implement automatic corrective actions, such as activating soundproofing mechanisms or adjusting classroom settings.Manual intervention is still required to respond to high noise levels.

CHAPTER 6
PROPOSED SYSTEM

In the proposed noise monitoring system, an IoT-based approach is utilized to detect noise levels in an environment and send real-time alerts through cloud-based services. The system is designed to operate autonomously, continuously monitoring noise levels and triggering notifications when predefined thresholds are exceeded. The workflow involves sensor-based data collection, real-time processing, wireless data transmission, cloud-based decision-making, and notification services to alert concerned users.

6.1 Block Diagram

The primary objective is to provide an efficient and reliable system for noise monitoring that can be deployed in various environments, such as industrial zones, residential areas, and workplaces, where noise pollution needs to be controlled.



` **Fig 6.1 Block diagram**

As shown in Fig 6.1 , The process begins with the microphone sensor detecting the surrounding noise levels. The sensor continuously captures audio signals and converts them into electrical signals, which are then processed by the ESP32 microcontroller. The ESP32, acting as the central processing unit, reads the sensor data, applies any necessary filtering or processing, and determines whether the noise levels exceed the predefined threshold. If the noise level remains within an acceptable range, the system remains idle, consuming minimal power. However, if the noise level surpasses the threshold, the ESP32 immediately triggers an event and prepares the data for wireless transmission using the MQTT protocol.

To transmit the data to the cloud, the ESP32 board establishes a connection with AWS IoT Core using the MQTT protocol. AWS IoT Core acts as the central hub, receiving and processing incoming sensor data in real-time. The message broker within AWS IoT Core ensures secure and efficient data transmission between the IoT device and cloud-based services. Once the data is received, predefined rules are applied to determine the next course of action. If the noise level is found to be excessive, AWS IoT Core triggers an AWS Lambda function for further processing.

AWS Lambda serves as the serverless computing unit, allowing the system to process noise data without requiring a dedicated server. The Lambda function analyzes the incoming noise levels and makes a decision regarding notifications. If the noise level is deemed critical, Lambda forwards the data to AWS Simple Notification Service (SNS) to generate alerts. AWS SNS is responsible for distributing notifications to the relevant stakeholders.

A key advantage of this proposed system is its real-time functionality. The entire process, from noise detection to alert generation, occurs almost instantaneously, allowing for immediate action. Additionally, the system is designed to be low-power and cost-effective, making it suitable for continuous deployment in both urban and industrial environments.

Furthermore, the system is highly customizable, allowing users to modify noise thresholds, alert recipients, and notification preferences based on their specific needs. For example, in a factory setting, different noise thresholds can be set for daytime and nighttime operations. Similarly, in residential areas, noise alerts can be configured based on local regulations. The use of AWS services enhances security and reliability, ensuring that data transmission is encrypted and that only authorized users can access the noise monitoring system.

6.2 Features of proposed system

The proposed system is an advanced IoT-based noise monitoring solution that integrates ESP32 microcontroller and AWS cloud services to provide efficient, real-time noise tracking. It overcomes the limitations of traditional monitoring methods by enabling automated data analysis, instant notifications, and remote accessibility. Below are the key features that enhance its performance and usability

a) Portability

The system is designed to be compact and lightweight, allowing easy deployment in various locations without the need for complex installation procedures. It is ideal for indoor and outdoor noise monitoring, including industrial sites, classrooms, hospitals, and residential areas

b) Real-time Monitoring

The system continuously monitors and analyzes noise levels, providing real-time data to ensure immediate action when necessary. By leveraging the ESP32's high-speed processing capabilities, noise data is captured with minimal latency and sent to the cloud for further analysis.

c) Wireless Connectivity

The noise monitoring system utilizes Wi-Fi and MQTT protocol to transmit data wirelessly, eliminating the need for wired connections that can be restrictive.

d) Automated Alerts

The system is programmed to send instant email notifications when noise levels exceed predefined thresholds, ensuring timely intervention. Additionally, the system can be configured to trigger visual or audio alerts, such as alarms or LED indicators, for immediate on-site notifications.

e) Low Power Consumption

The system is optimized for energy efficiency, utilizing a 3.7V Li-ion battery as its primary power source. The ESP32's deep sleep mode ensures that the device conserves power when noise monitoring is not actively needed, significantly extending battery life. This feature makes it suitable for remote and long-term deployments, where regular battery replacements may not be feasible.

f) Cloud Integration

The noise monitoring system seamlessly integrates with AWS cloud services, including AWS IoT Core, AWS Lambda, and AWS SNS, to ensure scalable and secure data processing. AWS IoT Core manages device connectivity and real-time data transfer, while AWS Lambda processes incoming noise data to determine if an alert needs to be triggered. AWS SNS is responsible for sending notifications to users when thresholds are exceeded. This cloud-based architecture eliminates the need for local storage and processing, reducing hardware costs and improving accessibility. Users can access historical noise data, generate reports, and analyze trends via cloud dashboards, enabling data-driven decision-making.

g) Customizable Thresholds

The system allows users to set custom noise level thresholds, enabling personalized monitoring based on specific environmental conditions. Users can configure different noise limits for day and night, ensuring compliance with regulatory noise standards.

h) Scalability

The system is highly scalable, allowing multiple ESP32-based noise monitoring units to be deployed across different locations while maintaining a centralized data collection system. AWS IoT Core enables efficient management of multiple devices, ensuring seamless connectivity and data synchronization. Whether monitoring a single location or an entire city, the system can be expanded without compromising performance.

i) Cost-Effective

The system is designed using affordable yet high-performance hardware components, ensuring a balance between cost and functionality. By utilizing open-source platforms and cloud-based processing, the need for expensive local infrastructure is minimized.

Limitations of the Previous System	**Improvements in the Enhanced System**
The previous system relied on periodic noise level measurements rather than continuous tracking.	The enhanced system captures and processes noise data in real time using ESP32 and AWS IoT Core.
Noise data had to be manually entered into spreadsheets, making it inefficient and error-prone.	AWS Lambda automates data storage and analysis, ensuring accuracy and efficiency.
Users had to manually check noise levels without automated notifications.	AWS SNS sends automatic alerts via email and SMS when noise exceeds set limits.
The system required a continuous power source, limiting its deployment flexibility.	A rechargeable 3.7V Li-ion battery with TP4056 charging enables mobile and remote use.

Short-range connectivity restricted data transmission across large areas.	Wi-Fi and MQTT protocols ensure seamless data transfer to AWS IoT Core for remote monitoring.
The system was not designed for easy scalability across multiple locations.	AWS services allow multiple monitoring units to be connected and managed centrally.
The previous system did not retain long-term records for trend analysis.	AWS enables long-term storage and analytics for better decision-making.
Regular physical maintenance was required to ensure system functionality.	Cloud-based processing reduces manual intervention, and system updates can be done remotely.

Table 2 : comparision between existing and proposed systems

The above Table 2 highlights the limitations of the previous noise monitoring system and how the enhanced system with AWS integration overcomes them. The new system offers real-time monitoring, automated alerts, cloud-based data storage, and scalability, making it more efficient and user-friendly.

6.3 Hardware Components

The proposed system relies on a combination of efficient and reliable hardware components to ensure accurate noise monitoring and seamless data transmission. These components are carefully selected to provide real-time processing, wireless communication, and low power consumption. By integrating advanced sensors and microcontrollers, the system ensures precise noise detection and efficient cloud connectivity. Below is a detailed description of the essential hardware components used in the system.

a) ESP32 Board

The ESP32 shown in Fig 6.2 is a powerful microcontroller that integrates Wi-Fi and Bluetooth capabilities, making it ideal for IoT applications. It features a dual-core processor, multiple GPIO pins, and built-in security features for secure communication.

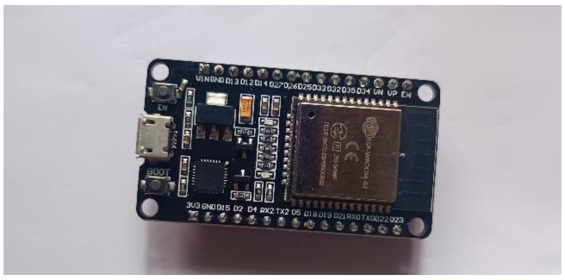


Fig 6.2 ESP32 board

Specifications:

i. Dual-core Tensilica LX6 processor running at up to 240MHz.
ii. Integrated Wi-Fi (802.11 b/g/n) and Bluetooth (4.2 and BLE).
iii. Multiple GPIOs for interfacing with sensors and modules.
iv. Supports SPI, I2C, UART, PWM, and ADC/DAC interfaces.
v. Power-efficient deep sleep mode for battery-operated applications.

Working in Noise Monitoring System:

The ESP32 board acts as the central processing unit for the system. It continuously reads data from the microphone sensor and processes it to detect excessive noise levels. When a threshold is exceeded, it transmits the data using the MQTT protocol to AWS IoT Core. The integration of cloud services allows remote monitoring and alerts.

Advantages:

i. Low power consumption with deep sleep mode.
ii. High processing power for real-time signal processing.
iii. Integrated Wi-Fi and Bluetooth reduce the need for additional modules.
iv. Flexible GPIO configuration enables easy sensor interfacing.
v. Cost-effective for large-scale IoT deployments.

b) Max9814 Sensor

The Max9814 shown in Fig 6.3 is a high-performance microphone sensor with an automatic gain control (AGC) feature, making it ideal for noise detection applications. It amplifies weak audio signals while reducing background noise for accurate sound capture. The sensor operates efficiently with low power consumption, making it suitable for battery-powered systems.

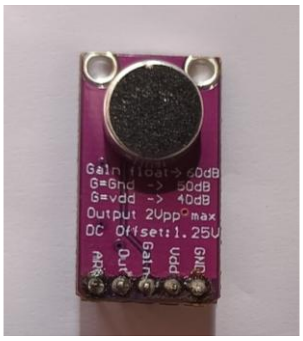


Fig 6.3 Max9814 Sensor

Specifications:

i. Integrated low-noise preamplifier with adjustable gain.

ii. Automatic Gain Control (AGC) for consistent output levels.

iii. Wide frequency response for accurate noise detection.

iv. Operates on a 2.7V to 5.5V power supply.

v. Compact design for easy integration into embedded systems.

Working in Noise Monitoring System:

The Max9814 microphone captures surrounding noise levels and converts them into an electrical signal. The signal is then processed by the ESP32 to analyze its amplitude. If the noise exceeds a predefined threshold, the system triggers an alert via AWS services.

Advantages:

i. High sensitivity ensures accurate noise detection.

ii. Built-in AGC minimizes signal distortion.

iii. Low power consumption makes it suitable for battery-powered systems.

iv. Compatible with multiple microcontrollers and processors.

c) 3. 3.7V Li-ion Battery

A 3.7V lithium-ion battery shown in Fig 6.4 is used to power the entire system, providing a reliable and portable energy source. These batteries are known for their high energy density, offering long operational hours without frequent recharging. They have a low self-discharge rate, ensuring extended standby time when the device is not in use. The battery efficiently powers the ESP32 board and connected sensors while maintaining a stable voltage output. It is lightweight and compact, making it an ideal choice for portable noise monitoring systems. These batteries come with built-in protection circuits to prevent overcharging and discharging. Their rechargeable nature makes them cost-effective and environmentally friendly compared to disposable batteries. They provide consistent performance even in varying temperature conditions. The battery can be charged using the TP4056 charging module, ensuring safe and efficient energy replenishment. Overall, a 3.7V Li-Ion battery is a crucial component for uninterrupted system operation

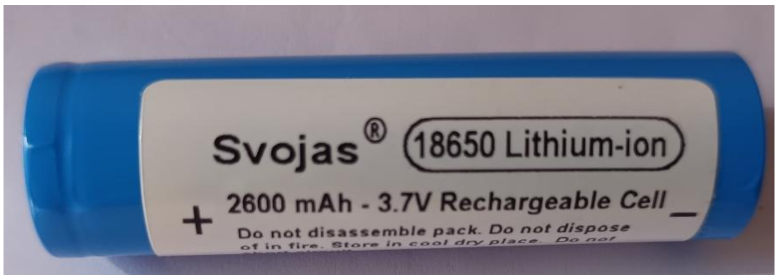

Svojas®
18650 Lithium-ion
2600 mAh - 3.7V Rechargeable Cell
+
−
Do not disassemble pack. Do not dispose
of in fire. Store in cool dry place.

Fig 6.4 3.7v Li ion Battery

.**Specifications:**

- **i.** Voltage: 3.7V nominal, 4.2V fully charged.
- **ii.** Capacity: Ranges from 500mAh to 5000mAh depending on requirements.
- **iii.** Rechargeable with long cycle life.
- **iv.** Lightweight and compact design.
- **v.** Overcharge and discharge protection circuits.

Working in Noise Monitoring System:

The battery powers the ESP32 and microphone sensor, ensuring uninterrupted operation. It provides stable voltage levels, maintaining system reliability. When depleted, it can be recharged using the TP4056 charging module.

Advantages:

- **i.** High energy density for prolonged use.
- **ii.** Rechargeable design reduces maintenance costs.
- **iii.** Lightweight for portable applications.
- **iv.** Provides stable voltage levels.

4. TP4056 Module

The TP4056 shown in Fig 6.5 is a widely used charging module designed for lithium-ion batteries, ensuring safe and efficient charging. It features overcharge, over-discharge, and short-circuit protection, preventing battery damage and extending lifespan. The module operates on a 5V input, making it compatible with USB charging sources. It has an automatic charge cut-off feature that stops charging once the battery reaches full capacity.

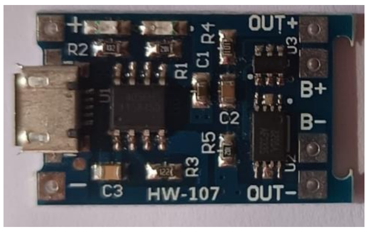


Fig 6.5 TP4056 Charging Module

Specifications:

- **i.** Input voltage: 5V (USB or solar panel input).
- **ii.** Output voltage: 4.2V (regulated for battery charging).
- **iii.** Charging current: Adjustable up to 1A.
- **iv.** Overcharge and short-circuit protection.
- **v.** Compact and easy to integrate with IoT devices.

Working in Noise Monitoring System:

The TP4056 module is responsible for charging the Li-ion battery using a 5V power source. It prevents overcharging and protects against excessive discharge, enhancing battery longevity. The module ensures a stable power supply to the ESP32 and microphone sensor.

Advantages:

i. Provides safe and efficient battery charging.
ii. Overcharge and short-circuit protection.
iii. Supports multiple power input sources (USB, solar, etc.).
iv. Compact size for easy integration into embedded systems.

6.4 Software Requirements

In this project, software components play a critical role in enabling the communication between hardware devices and AWS cloud services. The system uses Arduino IDE for writing and deploying firmware to the ESP32 board, C++ and Python for coding different parts of the application, and JSON for writing IAM policies. Additionally, various AWS services like AWS Lambda, AWS SNS, AWS CloudWatch, AWS IoT Core, and IAM are integrated for cloud-based data processing, alerting, and monitoring. The combination of these software components ensures efficient, scalable, and secure noise monitoring.

a) Arduino IDE

Fig 6.6 Aurdino ide

The Arduino IDE is used for writing, compiling, and uploading firmware to the ESP32 microcontroller. It provides a user-friendly interface with built-in libraries, allowing seamless integration with sensors and cloud services. The logo of Aurdino Ide shown in Fig 6.6 .The IDE supports C++, which is used to program the ESP32 board for collecting noise data and transmitting it via the MQTT protocol. Serial Monitor in Arduino IDE helps in debugging and monitoring sensor readings in real time. The code written in the Arduino IDE ensures efficient data acquisition, low-latency transmission, and real-time monitoring of noise levels.

b) C++ (Programming Language for ESP32)

Fig 6.7 C++

C++ is used in the Arduino IDE to program the ESP32 microcontroller. It allows efficient memory management, making it suitable for resource-constrained IoT devices. The code written in C++ handles sensor data acquisition, noise level processing, and MQTT communication with AWS IoT Core. The logo of c++ shown in Fig 6.7 .The language provides object-oriented programming (OOP) features, which help in writing modular and reusable code.

c) Python (Programming Language for AWS Lambda)

Fig 6.8 Python

Python is used to write AWS Lambda functions that process incoming noise level data. It is a high-level, interpreted language known for its simplicity and efficiency in handling cloud-based applications. The logo of python shown in Fig 6.8 .Python's built-in AWS SDK (Boto3) simplifies interactions with AWS services like IoT Core, SNS, and CloudWatch. It also allows serverless execution, reducing infrastructure costs and improving scalability.

d) AWS Lambda

Fig 6.9 AWS Lambda

AWS Lambda is a serverless computing service used to process noise level data in real time. It automatically executes Python scripts when new data arrives from AWS IoT Core. The logo of AWS Lambda shown in Fig 6.9 . The Lambda function analyzes the data, applies threshold conditions, and triggers an alert if necessary. Since Lambda is event-driven, it runs only when needed, reducing computing costs. It seamlessly integrates with SNS for notifications, CloudWatch for logging, and IAM for security controls. AWS Lambda eliminates the need to manage servers, making deployment and scaling effortless.

e) AWS SNS (Simple Notification Service)

Fig 6.10 AWS SNS

AWS SNS is a cloud-based notification service used to send alerts when noise levels exceed the threshold. AWS Lambda triggers SNS, which then sends notifications via email or SMS to predefined recipients. The logo of AWS SNS shown in Fig 6.10 . SNS ensures real-time communication, allowing users to take immediate action. It supports multiple messaging formats and can integrate with mobile push notifications, HTTP endpoints, and Lambda functions.

f) AWS CloudWatch

Fig 6.11 AWS cloud watch

AWS CloudWatch is used for monitoring and logging system performance. It collects real-time logs from AWS Lambda and IoT Core, helping in debugging and optimizing the system. The logo of AWS cloud watch shown in Fig 6.11 . CloudWatch stores error logs, execution times, and system performance metrics, enabling administrators to identify issues and improve efficiency. It also supports automated alarms, which can trigger notifications if the system encounters failures or unusual activity. CloudWatch plays a vital role in maintaining system reliability, performance tracking, and security auditing.

g) AWS IoT Core

Fig 6.12 AWS IOT Core

AWS IoT Core acts as the central hub for processing data from the ESP32 device. It receives noise level readings via the MQTT protocol, validates the data, and forwards it to AWS Lambda for processing. IoT Core provides secure device authentication using certificates, ensuring only trusted devices can send data. The logo of AWS IOT core shown in Fig 6.12 . It enables real-time communication between IoT devices and cloud applications, making it ideal for noise monitoring. IoT Core also supports device shadowing, allowing the system to store and retrieve the latest device state even when offline. With its scalability and reliability, AWS IoT Core ensures smooth operation of the noise monitoring system.

h) AWS IAM (Identity and Access Management)

Fig 6.13 AWS IAM

AWS IAM is responsible for managing security and access control in the system. It defines roles and permissions for AWS Lambda, IoT Core, SNS, and CloudWatch using JSON-based policies. The logo of AWS IAM shown in Fig 6.13 . IAM ensures that only authorized users and devices can access cloud resources, preventing unauthorized actions. It supports multi-factor authentication (MFA) and access keys, enhancing security. IAM plays a crucial role in maintaining data integrity, preventing security breaches, and managing user access efficiently.

6.5 Implementation Steps

The implementation of the noise monitoring system involves setting up both hardware and software components to ensure seamless functionality. The following steps outline the process of configuring, programming, and integrating the system for real-time noise monitoring and cloud-based data processing.

6.5.1 Setting Up the ESP32

The ESP32 serves as the central processing unit of the system, receiving noise level data from the microphone sensor and ensuring efficient power management. Proper connections with the microphone sensor and TP4056 charging module enable seamless operation and reliable power supply.

a) Connect Microphone Sensor to the ESP32

The microphone sensor is responsible for measuring sound levels and sending data to the ESP32 for processing. Follow these steps to set up the connections:

Connections:

i. Connect the VCC pin of the microphone sensor to the 3.3V pin on the ESP32.
ii. Connect the GND pin of the microphone sensor to the GND pin on the ESP32.
iii. Connect the OUT pin (signal output) of the microphone sensor to GPIO 34 on the ESP32 (an analog input pin).

b) Connect Charging Module (TP4056) and Battery

The TP4056 module ensures safe and efficient charging of the 3.7V Li-ion battery that powers the ESP32 and sensor.

Connections:

i. Connect the B+ and B- terminals of TP4056 to the 3.7V Li-ion battery.
ii. Connect the OUT+ and OUT- terminals of TP4056 to the ESP32 Vin and GND pins to provide power.
iii. Use a micro-USB cable to charge the battery through the TP4056 module.

6.5.2 Program the ESP32 Using Arduino IDE

Programming the ESP32 is essential for establishing communication with AWS and ensuring accurate data transmission. Configuring the necessary libraries and AWS IoT Core credentials allows the device to process and send real-time noise data.

a) Install the Arduino IDE

i. Install Arduino IDE: Download and install the latest version from the official Arduino website.
ii. Add ESP32 Board Support:

Open Arduino IDE and go to File > Preferences.
In Additional Board Manager URLs, add:
https://dl.espressif.com/dl/package_ESP32_index.json
Click OK.
Go to Tools > Board > Boards Manager, search for ESP32, and click Install.

b) Install Required Libraries

i. Install the PubSubClient library for MQTT communication.
ii. Install the WiFi library (pre-installed in most Arduino IDE versions).

c) Configure AWS IoT Core Credentials in Code

i. Replace YourWiFiSSID and YourWiFiPassword with your Wi-Fi credentials.
ii. Update your-aws-endpoint with your AWS IoT Core endpoint.
iii. Insert your Root CA certificate, device certificate, and private key (downloaded from AWS IoT Core).

6.5.3 Configuring AWS IoT Core

AWS IoT Core acts as the cloud-based hub for collecting and processing noise data from the ESP32. Creating a Thing and attaching the required security certificates ensure a secure and authenticated connection between the device and the cloud.

a) Create a Thing in AWS IoT Core

i. Log in to AWS Console and navigate to AWS IoT Core.
ii. Create a Thing: Go to Manage > Things and click Create single thing.
iii. Download Security Certificates: Generate and download the device certificate, private key, and Root CA certificate.
iv. Activate the Certificate in the AWS IoT Core console.

b) Attach an IoT Policy

i. Create an IoT Policy: Go to Secure > Policies and click Create policy.
ii. Add Permissions: Grant permissions for iot:Connect, iot:Publish, iot:Subscribe, and iot:Receive.
iii. Attach the Policy and Certificate: Link the created policy to the device certificate.

6.5.4 Configuring MQTT Test Client

MQTT is used to enable real-time data exchange between the ESP32 and AWS IoT Core. Subscribing to the appropriate topic ensures that noise level data is successfully published and received for further processing.

i. Open the MQTT Test Client in AWS IoT Core.
ii. Subscribe to Topic: Enter noise-monitoring/data and click Subscribe.
iii. Test the ESP32: Run the device and check if JSON messages are published.

6.5.5 Creating AWS IoT Rule to Trigger Lambda

AWS IoT Rules Engine enables automatic event processing based on incoming noise data. Setting up a rule to trigger a Lambda function ensures that noise threshold violations are immediately detected and acted upon.

i. Go to Message Routing > Rules in AWS IoT Core.

ii. Click Create Rule and set up an SQL Query:
SELECT * FROM 'classroom/noise'

iii. Add an Action: Choose Send a message to a Lambda function and link to an existing function or create a new one.

6.5.6 Writing and Deploying AWS Lambda Function

AWS Lambda processes incoming noise data and determines whether an alert needs to be sent. Deploying a well-configured Lambda function ensures efficient event handling and seamless integration with AWS SNS for notifications.

a) Create a Lambda Function

i. Open AWS Lambda and click Create function.

ii. Select Author from scratch, enter a function name, and choose Python 3.x as the runtime.

iii. Set up permissions and click Create function.

b) Modify the Lambda Function Code

i. Insert the Python script that reads noise level data and triggers an SNS notification.

ii. Update the SNS Topic ARN in the script.

iii. Click Deploy to save changes.

6.5.7 Setting Up AWS SNS for Notifications

AWS Simple Notification Service (SNS) is used to send instant alerts when noise levels exceed predefined thresholds. Configuring a topic and subscribing an email ensures timely notifications for prompt corrective actions.

a) Create an SNS Topic

i. Go to Amazon SNS and click Create topic.

ii. Select Standard Topic, name it NoiseMonitorTopic, and click Create.

b) Subscribe an Email Address

i. Click Create subscription, select Email, and enter your email address.

ii. Confirm the subscription via the link received in your email.

6.5.8 Testing the System

i. Publish a Test Message: Use AWS SNS to send a test email notification.

ii. Run the ESP32 Device: Monitor noise levels and verify if alerts are triggered correctly.

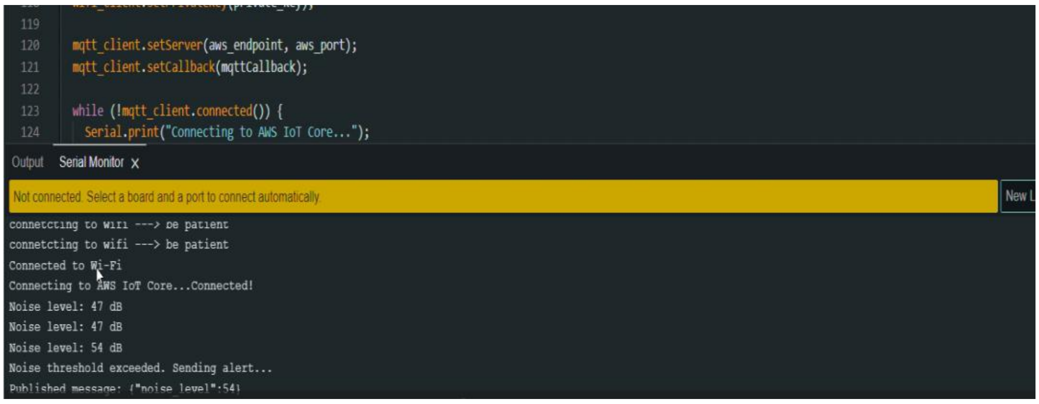
iii. Check Logs in AWS CloudWatch to ensure correct execution of Lambda and SNS functions.

CHAPTER - 7

RESULTS

The results section presents the **output logs and system responses** recorded during the implementation of the Noise Monitoring System. It includes **Arduino Serial Monitor logs, AWS CloudWatch logs, and email notifications**, demonstrating the system's ability to detect, process, and respond to excessive noise levels in real time

7.1 Arduino Serial Monitor Logs for Noise Monitoring System



```

  mqtt_client.setServer(aws_endpoint, aws_port);
  mqtt_client.setCallback(mqttCallback);

  while (!mqtt_client.connected()) {
    Serial.print("Connecting to AWS IoT Core...");
```

Output
Serial Monitor
Not connected. Select a board and a port to connect automatically.
New L

```
connetcting to wifi ---> be patient
connetcting to wifi ---> be patient
Connected to Wi-Fi
Connecting to AWS IoT Core...Connected!
Noise level: 47 dB
Noise level: 47 dB
Noise level: 54 dB
Noise threshold exceeded. Sending alert...
Published message: {"noise_level":54}
```

Fig. 7.1 serial monitor logs

The above Fig. 7.1 displays the serial monitor logs from the Arduino used in the Noise Monitoring System. The logs confirm the real-time noise level readings and system response.

Description of Serial Monitor Logs:

a) **System Initialization:**

The Arduino starts, initializing the microphone sensor and serial communication.

A message confirms that the ESP32 board conneted to wifi and aws iot core

b) **Real-time Noise Level Readings:**The Arduino continuously measures and prints the noise level in decibels (dB).

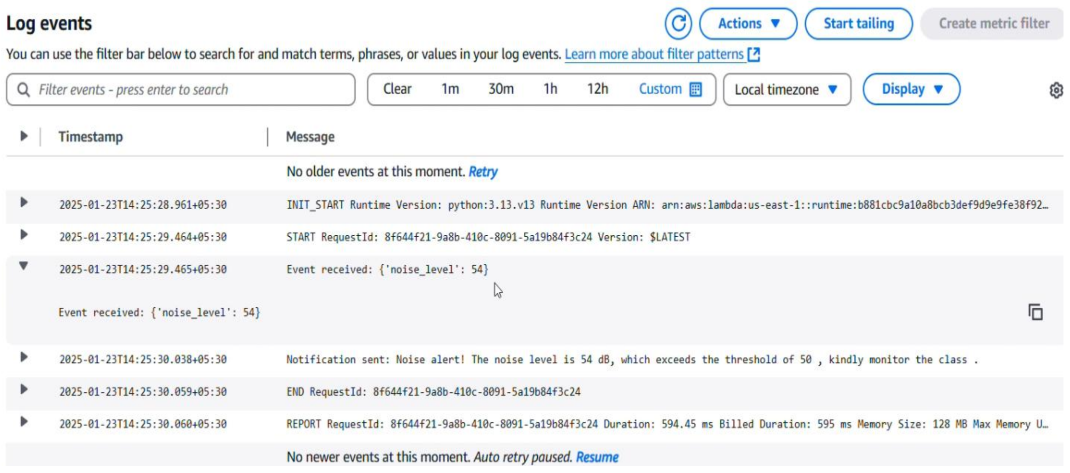
Example output:

Noise Level: 45 dB

Noise Level: 52 dB

c) **Threshold Breach Detection:**When the noise level crosses the 50 dB threshold, an alert message is printed: "Noise level exceeded: 54 dB Sending alert notification..."This confirms that the Arduino correctly detects high noise levels and triggers an alert mechanism.

7.2 AWS Cloud watch logs



Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                       Message
                                No older events at this moment. Retry
2025-01-23T14:25:28.961+05:30   INIT_START Runtime Version: python:3.13.v13 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:b881cbc9a10a8bcb3def9d9e9fe38f92…
2025-01-23T14:25:29.464+05:30   START RequestId: 8f644f21-9a8b-410c-8091-5a19b84f3c24 Version: $LATEST
2025-01-23T14:25:29.465+05:30   Event received: {'noise_level': 54}

Event received: {'noise_level': 54}

2025-01-23T14:25:30.038+05:30   Notification sent: Noise alert! The noise level is 54 dB, which exceeds the threshold of 50 , kindly monitor the class .
2025-01-23T14:25:30.059+05:30   END RequestId: 8f644f21-9a8b-410c-8091-5a19b84f3c24
2025-01-23T14:25:30.060+05:30   REPORT RequestId: 8f644f21-9a8b-410c-8091-5a19b84f3c24 Duration: 594.45 ms Billed Duration: 595 ms Memory Size: 128 MB Max Memory U…
                                No newer events at this moment. Auto retry paused. Resume
```

Fig. 7.2 Cloud watch logs

The above Fig. 7.2 represents the AWS CloudWatch logs for the Noise Monitoring System implemented using an AWS Lambda function. The log entries indicate the sequence of events that occur when the function executes upon receiving noise level data.

Description of Log Events:

a) Lambda Initialization:

The function initializes with the Python 3.13 runtime.The execution starts with a unique RequestId.

b) Noise Level Event Received:

The function processes an event containing a noise level of 54 dB.. The log entry confirms that the event was successfully received.

c) Noise Alert Triggered:

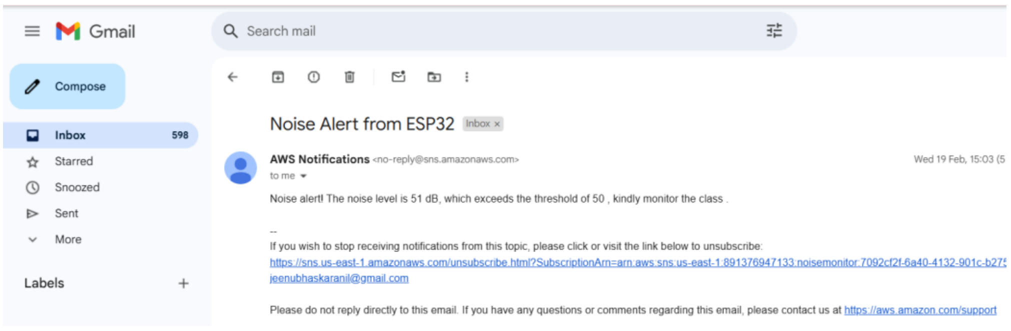
Since the noise level exceeds the predefined threshold of 50 dB, the function triggers a notification alert.The log entry states:

*"Noise alert! The noise level is 54 dB, which exceeds the threshold of 50, kindly monitor the class."*This indicates that the system successfully detected an excessive noise level and responded accordingly.

d) Lambda Execution Completion:

The execution concludes with the END RequestId, indicating the function completed its process.The billed duration for execution is 595 ms, and the allocated memory usage is 128 MB.

7.3: Email Notification for Noise Alert



Noise Alert from ESP32 Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> Wed 19 Feb, 15:03 (5

to me

Noise alert! The noise level is 51 dB, which exceeds the threshold of 50 , kindly monitor the class .

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:891376947133:noisemonitor:7092cf2f-6a40-4132-901c-b275
jeenubhaskaranil@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Fig. 7.3 Email Notification

The above Fig. 7.3 represents the email notification generated when a high noise level is detected. The system sends an automated email alert to the concerned personnel.

Description of Email Notification:

a) **Subject:**Noise Alert: High Noise Level Detected!

b) **Email Body:**

The email provides details of the noise event, including:

Alert: Noise level has exceeded the threshold!

Current Noise Level: 54 dB

Threshold: 50 dB

Location: Classroom 101

It advises the recipient to take immediate action to control the noise.

c) **Timestamp and Sender Details:**

The email includes the timestamp of the event and is sent from a configured AWS SES (Simple Email Service) or SMTP server.The email is automatically triggered by the system upon detecting high noise levels.

REFERENCES

[1] Siddhartha, C. V., Rao, P., & Velmurugan, R. (2023). Classroom Activity Detection in Noisy Preschool Environments with Audio Analysis. *Proceedings of the IEEE International Conference on Smart Systems for Applications in Electrical Sciences (ICSSES)*

[2] Castillo de Valencia, A., Caballero, L., Cilli, Á., González, J., Rojas, I., & Torres, B. (2023). Development of a Noise Monitoring and Control Sensor Network System for Enclosed Spaces within a University Environment. *IEEE Conference on Smart Environments*, 8, 311-318.

[3] Marques, G., & Pitarma, R. (2020). A Real-Time Noise Monitoring System Based on Internet of Things for Enhanced Acoustic Comfort and Occupational Health. *IEEE Access*, 8, 139743-139746.

[4] Badruddin, M. B., Hamid, S. Z. A., Rashid, R. A., & Hamsani, S. N. M. (2020). An IoT-Based Noise Monitoring System (NOMOS). *IOP Conference Series: Materials Science and Engineering*,

[5] Abdulqadir, A. M., & Shukur, M. H. (2017). Development of an IoT Noise Monitoring Network. *Cihan University-Erbil Scientific Journal*, Special Issue, 180-187.

[6] Anachkova, M., Domazetovska, S., Petreski, Z., & Gavriloski, V. (2023). A real-time IoT-based noise monitoring system for urban areas. *International Conference on Smart Systems and Applications in Electrical Sciences (ICSSES)*, 312-318.

[7] Garcia, J. S., Solano, J. P., Serrano, M. C., Camba, E. N., Castell, S. F., & Suay, F. M. (2020). Spatial statistical analysis of urban noise data from a wireless acoustic sensor network. *Applied Sciences, 6*(12), 380.

[8] Alsina-Pagès, R., Navarro, J., Alías, F., & Hervás, M. (2017). HomeSound: Real-time audio event detection based on high-performance computing for behavior and surveillance remote monitoring. *Sensors, 17*(4), 854.

[9] Badruddin, M. B., Hamid, S. Z. A., Rashid, R. A., & Hamsani, S. N. M. (2020). IoT-based noise monitoring system for acoustic comfort in educational institutions. *IOP Conference Series: Materials Science and Engineering, 884*(1), 012080.

[10] Jafari, M. J., Khosrowabadi, R., Khodakarim, S., & Mohammadian, F. (2019). The effect of noise exposure on cognitive performance and brain activity patterns. *Open Access Macedonian Journal of Medical Sciences*.

APPENDIX

Aurdino code:

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
// Wi-Fi credentials
const char* ssid = "………………..";
const char* password = "……………….";
// AWS IoT Core details
const char* aws_endpoint = "………………………………..";
const int aws_port = 8883;
const char* thing_name = "noisemonitor";
// AWS IoT Core certificates
const char* root_ca_cert = R"(
-----BEGIN CERTIFICATE-----
……………………………………………………………..
-----END CERTIFICATE-----
)";
const char* device_cert = R"(
-----BEGIN CERTIFICATE-----
………………………………………………………..
-----END CERTIFICATE-----
)";
const char* private_key = R"(
-----BEGIN RSA PRIVATE KEY-----
……………………………………………………………
-----END RSA PRIVATE KEY-----
)";
// Topics
const char* publish_topic = "classroom/noise";
const char* subscribe_topic = "classroom/noise/control";
// Pins and noise threshold
const int mic_pin = 34;  // Microphone connected to pin 34
const int noise_threshold = 50;  // Threshold in decibels (dB)
// WiFi and MQTT clients
WiFiClientSecure wifi_client;
PubSubClient mqtt_client(wifi_client);
// Function to connect to Wi-Fi
void connectWiFi() {
  Serial.print("Connecting to Wi-Fi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print("\nconnetcting to wifi ---> be patient");
  }
  Serial.println("\nConnected to Wi-Fi");
}
// Function to connect to AWS IoT Core
void connectAWS() {
  wifi_client.setCACert(root_ca_cert);
  wifi_client.setCertificate(device_cert);
  wifi_client.setPrivateKey(private_key);
  mqtt_client.setServer(aws_endpoint, aws_port);
  mqtt_client.setCallback(mqttCallback);
  while (!mqtt_client.connected()) {
    Serial.print("Connecting to AWS IoT Core...");
    if (mqtt_client.connect(thing_name)) {
      Serial.println("Connected!");
      mqtt_client.subscribe(subscribe_topic);  // Subscribe to a topic
    } else {
      Serial.print("Failed. Error state: ");
      Serial.println(mqtt_client.state());
      delay(2000);
    }
  }
}
// MQTT callback function for incoming messages
void mqttCallback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message received on topic: ");
  Serial.println(topic);
  Serial.print("Message: ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
// Function to read microphone data and calculate noise level in dB
int readNoiseLevel() {
  int raw_value = analogRead(mic_pin);  // Read raw microphone data
  float voltage = (raw_value / 4095.0) * 3.3;  // Convert to voltage (3.3V ADC)
  float dB = 20.0 * log10(voltage / 0.00631);  // Convert voltage to decibels
  return (int)dB;  // Return decibel value
}
// Function to publish noise level to AWS IoT Core
void publishNoiseLevel(int noise_level) {
  if (mqtt_client.connected()) {
    char message[100];
    snprintf(message, sizeof(message), "{\"noise_level\":%d}", noise_level);
    mqtt_client.publish(publish_topic, message);
    Serial.print("Published message: ");
    Serial.println(message);
  }
}
void setup() {
  Serial.begin(115200);
  pinMode(mic_pin, INPUT);  // Set microphone pin as input
  connectWiFi();
  connectAWS();
}
void loop() {
  mqtt_client.loop();  // Keep MQTT connection alive

  // Read noise level and check against threshold
  int noise_level = readNoiseLevel();
  Serial.print("Noise level: ");
  Serial.print(noise_level);
  Serial.println(" dB");
  if (noise_level > noise_threshold) {
    Serial.println("Noise threshold exceeded. Sending alert...");
    publishNoiseLevel(noise_level);
  }
  delay(2000);  // Delay before next reading
}
```

AWS Lambda code:

```
import json
import boto3
# Initialize SNS client
sns_client = boto3.client('sns')
# Your SNS Topic ARN
SNS_TOPIC_ARN = '……….'
# Threshold for noise level
def lambda_handler(event, context):
    try:
        # Log the incoming event for debugging
        print("Event received:", event)
        # Parse the noise level from the IoT Core message
        noise_level = event.get('noise_level', 0)
        # Check if the noise level exceeds the threshold
            # Send an alert via SNS
         message = f"Noise alert! The noise level is {noise_level} dB, which exceeds the
threshold of 50 , kindly monitor the class ."
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=message,
            Subject="Noise Alert from ESP32")
        print("Notification sent:", message)
        return {
            'statusCode': 200,
            'body': json.dumps('Processed successfully!')
        }
    except Exception as e:
        print("Error processing event:", str(e))
        return {
            'statusCode': 500,
            'body': json.dumps('Error processing event')
        }
```

AWS IOT – Core Policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": "*"
    }
  ]
}
```