## Remote Monitoring Interface For Critical Parameters In Atomic Clock

A Project Report Submitted in Partial Fulfillment of the Requirements for the Degree of

## B. Tech

**(Computer Science and Engineering)**

*By*

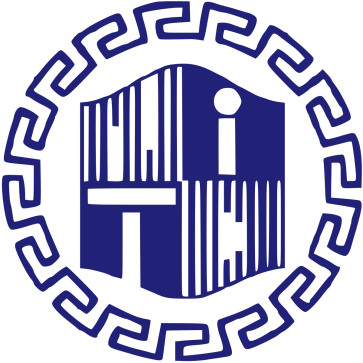
## MOKSH YADAV (221210071)

*Under the supervision of*

## DR. POONAM ARORA

## SENIOR PRINCIPAL SCIENTIST

**HEAD OF TIME AND FREQUENCY METROLOGY CSIR-NPL INDIA**



## NATIONAL INSTITUTE OF TECHNOLOGY DELHI

**DELHI-110036**

**June – July 2024**

# NIT DELHI

**DECLARATION**

I declare that the project entitled Interfacing of Sensors, coding and data logging of critical operational parameters. The atomic clock is an authentic record of my work at CSIR – National Physical Laboratory, for the degree of B. Tech Computer Science Engineering award. From National Institute of Technology Delhi, under the guidance of DR. Poonam Arora (SR. Principal Scientist, Indian Standard Time Division, National Physical Laboratory) from June 3rd, 2024, to July 31st, 2024.

DATE:

Moksh Yadav

221210071

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Dr. Poonam Arora Guide

(Sr. Principal Scientist, CSIR-NPL) New Delhi-110012

**CERTIFICATE**

This is to certify that the work incorporated in this report entitled “**Remote Monitoring Interface For Critical Parameters For Atomic Clock”** is submitted by **Mr. Moksh Yadav**, a 3rd-year B.tech Computer Science Engineering student at the National Institute of Technology, Delhi. The work was carried out from 3rd June 2024 to July 31st, 2024 under

**Dr. Poonam Arora, Senior Principal Scientist, in CSIR -National Physical Laboratory, New Delhi.**

**Dr .Poonam Arora**

Senior Principal Scientist

CSIR-NPL

New Delhi-110012

**ACKNOWLEDGEMENTS**

I am expressing my sincere gratitude and appreciation to everyone who contributed to the successful completion of this project.

First and foremost, I extend my heartfelt thanks to **Dr. Poonam Arora Ma'am**, Chief Scientist at NPL, for her invaluable guidance and encouragement throughout this project. Her expertise and insights were instrumental in shaping the direction and outcome of this work. Dr. Arora's dedication and support have been a constant source of inspiration.

Special thanks to **Miss. Anjali Bisht Mam, Mr Kishor Kumar, and Mr Sanjeev Kumar** for their constant mentorship and thorough explanations. Their patience and willingness to share their knowledge played a crucial role in my understanding of the project's intricacies. Miss. Bisht's hands-on approach and practical advice were invaluable.

I would also like to thank Dr. Rikmantra Basu, Professor at the National Institute of Technology Delhi, for his support and advice. His academic guidance and constructive feedback were essential in refining my work. Dr. Basu's encouragement and belief in my abilities have been profoundly motivating. His efforts in coordinating and supporting my involvement in this project were vital. Sir's commitment to providing valuable opportunities for students is commendable.

Lastly, I am deeply grateful to my family and friends for their unwavering support. Their encouragement and understanding provided me with the strength and motivation to persevere. The love and support from my family and friends have been the foundation for this project.

To everyone mentioned and to those whose names may not appear here but who have been a part of this journey, I extend my deepest gratitude.

**Moksh Yadav**

# ABSTRACT

# This report details developing and implementing a data logging system that leverages dual ADS1115 analog-to-digital converters (ADCs) and an ESP8266 microcontroller, with PHP integration for robust data management. The system is designed to monitor laser power via photodiodes, converting laser intensity into millivolt signals, which are then precisely measured and analyzed by the ADS1115 ADCs.

# The project emphasizes software implementation, where I played a pivotal role in developing the Arduino code for data acquisition and control, PHP code for server-side data management, and Python scripts for data fetching, deletion, and visualization. The Arduino Integrated Development Environment (IDE) programmed the ESP8266 microcontroller, ensuring efficient data transmission and integration with the XAMPP server-managed system. The PHP scripts, incorporating SQL for database management, enabled seamless real-time data logging and visualization. Meanwhile, the Python scripts, also utilizing SQL, provided comprehensive data analysis through interactive Dash applications.

# This system showcases a cost-effective and efficient approach to laser power monitoring, utilizing readily available components and open-source software for easy implementation and adaptability to various research and industrial applications. The successful integration of microcontrollers with web technologies highlights the potential for developing robust data acquisition systems, addressing design considerations and challenges encountered during the development process. Additionally, the project involved measuring laser powers while cooling cesium atoms, contributing to advancements in atomic clock technology.

# Overall, the internship at CSIR NPL has significantly contributed to my academic and professional journey, equipping me with practical skills, knowledge, and a broader perspective in computer science engineering.

# CONTENTS

| **Pg. No.**  **DECLARATION i**  **CERTIFICATE ii**  **ACKNOWLEDGEMENT iii**  **ABSTRACT iv**  **CONTENTS v**  **LIST OF FIGURES vi** | |
| --- | --- |
| **Chapter 1 – NATIONAL PHYSICAL LABORATORY 1-2** | |
| 1.1 | About NPL |
| 1.2 | Indian Standard Time |
| 1.3 | S.I Units |
| **Chapter 2 - MOTIVATION 3** | |
| **Chapter 3 - HARDWARE COMPONENTS USED 4** | |
| 3.1 Abstract | |
| 3.2 Principle | |
| **Chapter 4 –METHODOLOGY 5-11** | |
| 4.1 Arduino Code for Data Acquisition and Transmission | |
| 4.2 PHP Server for Data Reception and Storage | |
| 4.3 Python Script for Data Processing | |
| 4.4 Python Dash Application for Data Visualization | |
| **Chapter 5 - FUTURE SCOPE 12-13** | |
| **Chapter 6 - CONCLUSION 14** | |
| **Chapter 7 - REFERENCES 15** | |
| **Chapter 8 - ANNEXURE 16-23** | |

**LIST OF FIGURES**

| **Fig. No.** | **Figure Description** |
| --- | --- |
| 3.1 | Demonstration of Complete Process |
| 4.1 | Algorithm ( Arduino Code ) |
| 4.2 | Arduino Serial Monitor Output |
| 4.3 | Output table on https://localhost/phpmyadmin**/** |
| 4.4 | Algorithm for Python code (DB Updation) |
| 4.5 | Daily and Monthly CSV Files formed with MJD calculation |
| 4.6 | Dash Server presenting Data according to starting and ending date |

**CHAPTER 1**

# NATIONAL PHYSICAL LABORATORY

* 1. **About NPL**

CSIR- National Physical Laboratory (NPL-India) is mandated to be India’s “National Measurement Institute” (NMI) by an act of Parliament and its associated rules for legal metrology. CSIR-NPL is the custodian of “National Standards” and is responsible for realizing, establishing, upgrading, maintaining, and disseminating standards at par to the international level through R and D and the latest technology. CSIR-NPL's National Metrology has not only championed its primary mandate as the custodian of Measurement Standards for the nation but also served the Indian industry, academia, and strategic sectors to excel in their endeavors.

The mission of the CSIR-NPL is:

* + 1. Developing India's measurement standards that are internationally accepted and disseminating the measurement capabilities to industry, government, strategy and academia that underpin India's prosperity and quality of life.
    2. Conducting multidisciplinary R&D with a mission to establish futuristic quantum standards and upcoming technologies so that India remains on par with international measurement laboratories.
    3. Developing sophisticated analytical equipment (i.e. import substitutes) under the "Make in India" program to cater to the ever-increasing demands of emerging India.
    4. Training of young scientists and industry personnel in the measurements under the "Skill India" program.

To fulfill the fast-growing demand for accurate time within the nation and to establish a legal time in the entire country, CSIR-National Physical Laboratory, New Delhi, has developed a backup time scale in addition to the existing one. The National Physical Laboratory provides accurate time to institutions like Indian Railways, ISRO, National Knowledge Network, and UIDAI. Strengthening of Primary

Time Scale and development of a new backup time scale can disseminate IST more effectively to various industrial time and frequency users and millions of Indians synchronizing their computer's clock through the internet and web time display portal. The Indian Space Research Organization (ISRO) has developed the Indian Regional Navigation Satellite System (IRNSS), which has an operational name, Navigation with Indian Constellation (NAVIC), which can provide accurate real-time positioning and timing services. So, the purpose of linking national time, IST, with the NAVIC is to disseminate IST to the whole subcontinent within an uncertainty of several microseconds. The NAVIC is now synchronized with the Indian Standard Time and traceable to Coordinated Universal Time (UTC) through UTC(NPLI). For time dissemination over telephone networks (FonoClock), a new hardware incorporating communication delay compensation has been developed to provide a time synchronization accuracy of ± 10 ms. The National Time Scale and National Time dissemination services are aimed at strengthening the national timing infrastructure and providing the legalization of Indian Standard Time.

## Indian Standard Time

IST is based on longitude 82.5°, which passes through Mirzapur, near Allahabad in Uttar Pradesh. It is 5 hours 30 minutes ahead of Greenwich Mean Time (GMT), now called the Universal Coordinated Time (UTC). Keeper of the time in India is the CSIR-National Physical Laboratory (NPL), New Delhi, which records time using five cesium atomic clocks.

## S.I Units

The international system of units or SI units are the primary measurement units used for many years. There are seven SI base units. Six of these are maintained at NPL India: meter, kelvin, second, kilogram, ampere, and candela.

Time and frequency division maintains the Indian Standard Time (IST). It is entrusted with preserving and synchronizing international standards of time and dissemination.

**CHAPTER 2**

# MOTIVATION

# The primary motivation behind this project stems from the critical need for real-time and precise monitoring of analog signals in various scientific and industrial applications. The ability to efficiently acquire, log, and visualize data from multiple sensors is essential at the National Physical Laboratory (CSIR-NPL), where maintaining and disseminating accurate time information is paramount. This project addresses several key challenges and offers significant improvements in data acquisition and visualization:

# 1. Enhanced Data Accuracy and Integrity: Utilizing ADS1115 ADCs and the ESP8266 microcontroller ensures high-resolution, accurate data collection from eight channels, which is crucial for precise monitoring.

# 2. Real-Time Data Acquisition: The Arduino code captures max values over 20 seconds and transmits data to a PHP server within the next 10 seconds, enabling near real-time analysis.

# 3. Efficient Data Management: Integration of PHP and MySQL allows efficient storage and management of large data volumes with unique identifiers for daily and monthly data, reducing the risk of loss or corruption.

# 4. Automated Data Processing: A Python script using `sql.connector` automates fetching and deleting data from the PHP server, storing it in Excel files, minimizing manual errors, and improving efficiency.

# 5. Comprehensive Data Visualization: The Python Dash app visualizes collected data, showing the latest readings from all ADC channels, allowing date range selection and data download, enhancing usability and analysis.

# 6. Scalability and Flexibility: The system's design supports easy scalability, accommodating additional sensors or different data intervals, making it adaptable to various research and industrial needs.

# 7. Support for Critical Research: This project supports CSIR-NPL's research, such as cooling cesium atoms for atomic clocks, by providing reliable monitoring and recording of voltage values.

# In conclusion, this project advances real-time monitoring systems by combining advanced hardware, robust data management, and intuitive visualization, supporting CSIR-NPL's mission and broader scientific applications.

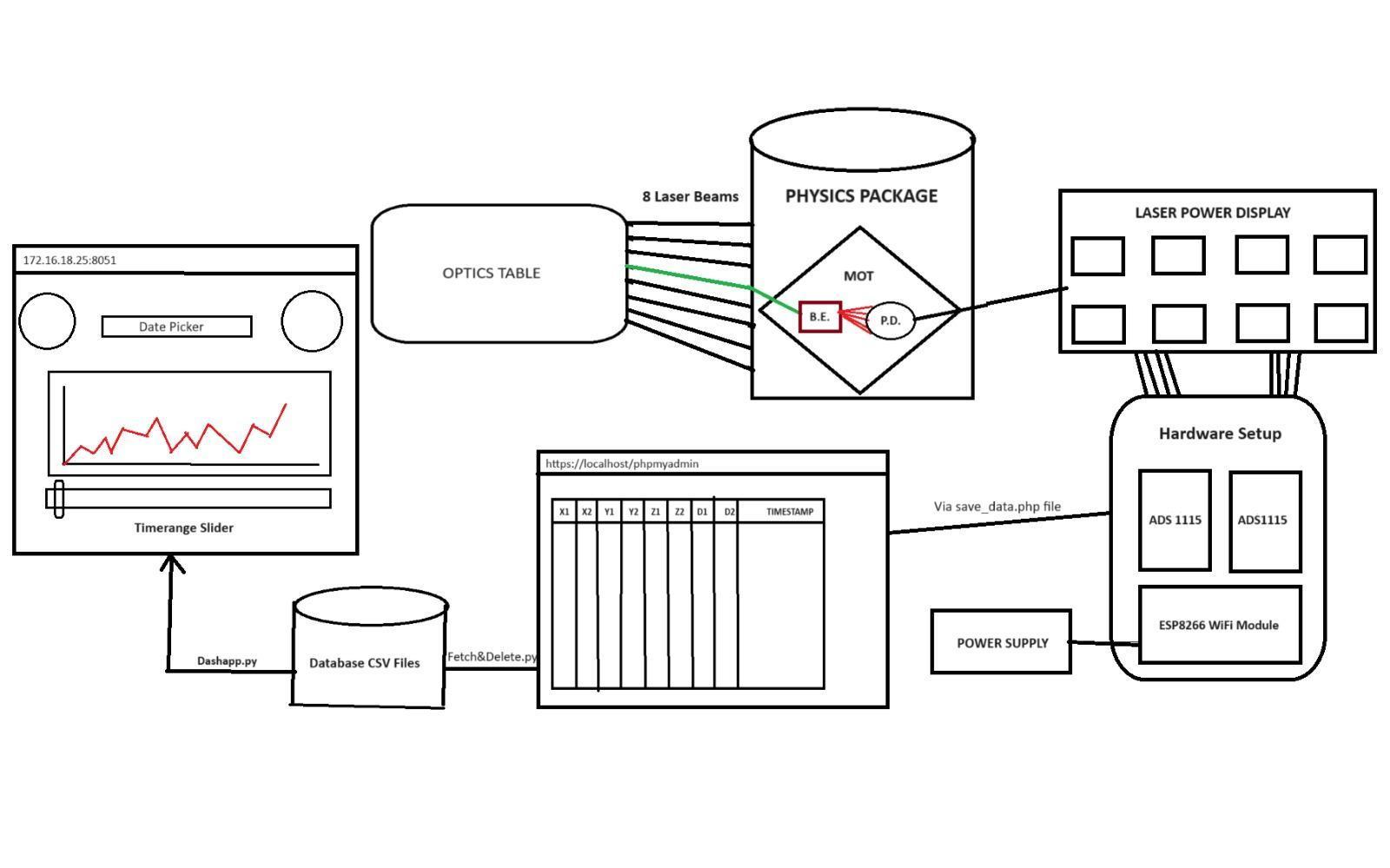
**CHAPTER 3**

# HARDWARE COMPONENTS USED

This project uses two ADC1115 analog-to-digital converters (ADCs) and an ESP8266 microcontroller to monitor continuously and record values from eight analog channels. The ADC1115, known for its high precision and low noise, provides 16-bit resolution, ensuring highly accurate analog signal capture. The ESP8266 reads these values, computes the maximum among them, and transmits this data. This hardware setup is crucial for continuously monitoring laser powers to ensure the reliable operation of the NPLI-CsF1.

### PRINCIPLE:

* + - The ADC1115 is a 16-bit precision ADC with four single-ended or two differential analog input channels. It communicates with the microcontroller via an I2C interface, allowing easy integration and low pin count.
    - The ADC1115's internal reference voltage and programmable gain amplifier (PGA) ensure that even low-level signals can be accurately digitized.
    - The ESP8266 is a low-cost Wi-Fi microcontroller capable of handling various tasks, including processing and wireless communication.
    - It features a 32-bit Tensilica processor, sufficient RAM and flash memory, and built-in Wi-Fi, making it an ideal application choice.

****

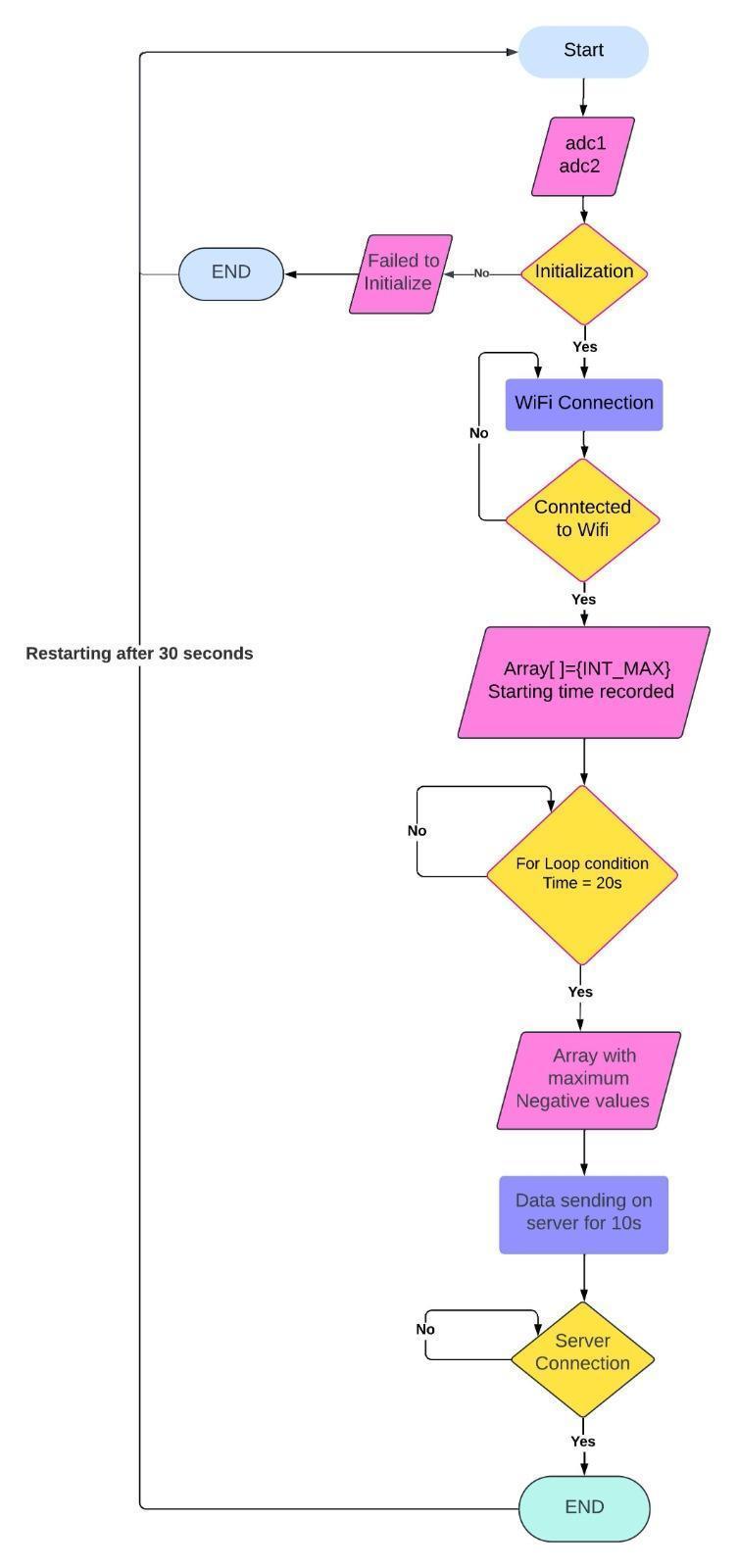
**( Fig. 3.1: Demonstration of Complete Process )**

**CHAPTER 4**

# METHODOLOGY

# The project's software implementation involves several critical stages to ensure seamless data acquisition, storage, and visualization. Here is a detailed breakdown of each stage:

### 4.1. Arduino Code for Data Acquisition and Transmission

**4.1.1. Setup:**

* **Initialize Components:** Configure I2C for ADS1115 with ESP8266 and initialize the WiFi connection. Perform any necessary calibration of the ADS1115.
* **WiFi Connection:** Connect to WiFi and handle connection errors with feedback.

**4.1.2. Data Reading:**

* **Read Values:** Continuously read from ADS1115 channels over a set interval (e.g., 20 seconds). Store the minimum value from each channel during this period.

**4.1.3. Data Transmission:**

* **Prepare and Send Data:** Format the stored values into a JSON or URL-encoded string. Send the data to the server via HTTP POST every specified interval (e.g., 30 seconds). Handle connection errors and validate server responses.

**4.1.4. Timing and Control:**

* **Manage Timing:** Use millis() to control measurement and transmission intervals. Switch between measurement and transmission modes efficiently.

**4.1.5. Optimization:**

* **Enhance Efficiency:** Minimize delays and manage memory to ensure stable and continuous operation.

**Fig 4.1 Algorithm (Arduino Code)**

# 

# 

# Fig 4.2:Output on Arduino Screen

# Fig 4.2 Arduino Serial Monitor Output

# 4.2. PHP Server for Data Reception and Storage

# 4.2.1. Database Connection:

# - Define Database Credentials: Set the variables for the server name, username, password, and database name.

# - Create Connection: Using the provided credentials, use the mysqli class to initiate a connection to the MySQL server.

# - Check Connection: Verify if the connection is successful. If there is a connection error, terminate the script and display an appropriate error message.

# 4.2.2. Data Reception and Validation:

# - Check for Data: Verify that all required POST variables (X1, X2, Y1, Y2, D1, D2, Z1, Z2) are set and available.

# - Store Data in Variables: Assign the values of the POST variables to local PHP variables if they are present.

# 4.2.3. Data Reception and Validation:

# - Prepare SQL Statement: Construct an 'SQL INSERT' statement with placeholders for the data values.

# - Bind Parameters: Use 'bind\_param( )' to securely bind the data values to the placeholders in the SQL statement.

# - Execute Statement: Execute the prepared statement to insert the data into the database.

# - Handle Execution Result: Check the execution result to determine if the data was inserted successfully. Provide feedback based on the result.

# - Close Statement: Close the prepared statement to free resources.

# 4.2.4. Error Handling and Closing Connection

# - Check for Data Presence: If any expected POST variables are missing, output a message indicating no data was received.

# - Close Connection: Use the close() method to end the connection with the MySQL server.

# 

# 

# 

# Fig4.3: Output table on https://localhost/phpmyadmin/

# 

# 

# 4.3. Python Script for Data Processing

# 4.3.1. Database Connection:

# - Setup Connection

# 1. Library Import: Use mysql.connector to handle the connection to the MySQL database.

# 2. Configure Connection Parameters: Provide the hostname, username, password, and database name to connect to the arduino\_data database.

# - Fetch Data

# 1. SQL Query: Use a SELECT statement to retrieve data from the sensor\_data table for the current date.

# 2. Execute Query: Execute the query using the cursor object and fetch the results.

# 4.3.2. Data Storage:

# - CSV Writing

# 1. File Naming: Create filenames for daily and monthly CSV files based on the current date and month.

# 2. Read Existing Data: Check if the CSV file exists and read it to determine the last used ID.

# 3. Assign Unique IDs: Generate unique IDs based on the last used ID for each entry.

# 4. Append Data: Write the new data entries to the CSV files, appending to existing data if necessary.

**- Data Integrity**

# 1. File Handling: Ensure files are opened and written correctly, handling any potential file I/O errors gracefully.

# 4.3.3. Deletion:

# - Deletion Query

# 1. Construct SQL Statement: Use a DELETE query to remove the data for the current date from the sensor\_data table.

# 2. Execute Query: Execute the delete query using the cursor object after ensuring the data has been successfully written to the CSV files.

# - Transaction Management

# 1. Ensure Atomicity: Use database transactions to ensure that data is only deleted if successfully saved to the CSV files.

# 2. Commit Changes: Use 'conn.commit( )' to apply the changes to the database.

# 

# 

# Fig 4.4 Algorithm for Python code (DB Updation)

# 

# Fig 4.5 Daily and Monthly CSV Files formed with MJD calculation

# 

# 4.4. Python Dash Application for Data Visualization

# 4.4.1. Data Loading:

# - Read CSV Files

# 1. Use the pandas library to read data from daily CSV files in the current directory. The filenames should start with 'live\_data\_' and end with '.csv.'

# 2. Load the data into a combined DataFrame and parse the timestamp column for accurate time-based operations.

**- Data Filtering**

# 1. Implement functionality to filter data based on user-selected date ranges from the 'DatePickerRange' component.

# 2. If 'start\_date' and 'end\_date' are provided, filter the combined DataFrame to include only the data within this range.

**- Data Refresh**

# Set up a periodic update mechanism using the 'Interval' component to refresh the data at regular intervals (e.g., every 30 seconds) and keep the visualizations current.

# 4.4.2. Dashboard Interface:

# - Table Display: Use Dash's `dash\_table` component to display the latest readings from all eight ADC channels. Ensure the table is dynamically updated as new data comes in.

# - Date Selection: Implement date picker components to allow users to specify date ranges for viewing historical data.

# 4.4.3. User Interaction:

# - Download Data Button: Add a button component that triggers the selected data download. Use callback functions to handle the download logic, ensuring the chosen date range data is accurately packaged and sent to the user.

# - Callbacks: Implement Dash callbacks to handle user interactions, such as selecting date ranges and triggering data downloads.

# By following these detailed steps, the software components of the project are effectively integrated. This comprehensive methodology ensures the system's accuracy, efficiency, and user-friendliness, making it a valuable tool for scientific and industrial applications.

All my resources and codes can be viewed here: [NPLWork](https://drive.google.com/drive/folders/1oBgctEx_YkNTxfclmy0PKqiLyf1OiSqt?usp=sharing)



**Fig 4.6 Dash Server presenting Data according to starting and ending date**

**CHAPTER 5**

# FUTURE SCOPE

# The project outlined provides a robust framework for real-time data acquisition, management, and visualization. However, there is significant potential for further enhancements and applications. One promising area for future development is integrating an SMS alert system using a GSM module. This system could provide real-time notifications to scientists when critical parameters, such as laser power, fall below specified thresholds. Here is a detailed exploration of the future scope and potential impact of this enhancement:

# SMS Alert System for Critical Thresholds

# Motivation:

# In scientific experiments, especially those involving precise measurements and controlled environments, it is crucial to maintain certain parameters within specified ranges. For instance, the power of a laser in an experiment may need to stay above a certain threshold to ensure accurate results. If the power drops below this threshold, immediate action might be required to prevent data corruption or equipment damage. An SMS alert system can provide real-time notifications to the responsible personnel, allowing them to take prompt corrective actions.

# Implementation

# 1. Hardware Integration

# - GSM Module: Integrate a GSM module with the existing Arduino setup. Modules such as SIM900 or SIM800 can be used to send SMS messages.

# - Connections: Connect the GSM module to the Arduino, ensuring proper communication via the serial interface (TX and RX pins). Provide a suitable power supply to the GSM module.

# 2. Arduino Code Modification

# - Threshold Definition: Define thresholds for critical parameters such as laser power within the Arduino code. For instance, if the laser power drops below a certain value, an alert should be triggered.

# - Reading Values: Continuously monitor the sensor readings from the ADS1115 modules. Compare the current values against the predefined thresholds.

# - SMS Sending Logic: When a threshold breach is detected, use the GSM module to send an SMS alert to a predefined list of recipients (e.g., scientists and technicians).

# Benefits

# - Real-Time Alerts: The SMS alert system ensures that critical threshold breaches are communicated to the relevant personnel in real-time, enabling quick response and mitigation.

# - Enhanced Monitoring: Adding an alert system enhances the monitoring capabilities of the existing setup, making it more robust and reliable.

# - Reduced Downtime: By providing immediate notifications, the system helps reduce downtime and potential damage to equipment, ensuring the continuity of experiments and data collection.

# - Scalability: The SMS alert system can be scaled to monitor multiple parameters and send alerts to different groups of recipients based on the severity of the threshold breach.

# Potential Applications

# - Laboratory Experiments: In laboratories where precise control of environmental conditions is essential, the SMS alert system can help maintain these conditions by providing timely notifications of deviations.

# - Industrial Processes: In industrial settings, the system can monitor critical parameters of machinery and processes, ensuring operational efficiency and safety.

# - Environmental Monitoring: For environmental monitoring stations, the system can alert scientists to changes in environmental conditions that may require immediate attention.

# Future Enhancements

# - Email Notifications: In addition to SMS alerts, email notifications can be integrated to provide more comprehensive alerting options.

# - Mobile App Integration: Developing a mobile app to receive and manage alerts can provide a more user-friendly interface for scientists and technicians.

# - Advanced Analytics: Incorporating advanced analytics and machine learning algorithms can help predict potential threshold breaches before they occur, providing proactive alerts.

# In conclusion, integrating an SMS alert system using a GSM module significantly enhances the functionality and utility of the current project. It provides a critical layer of monitoring and alerting that ensures timely responses to threshold breaches, ultimately improving the reliability and effectiveness of scientific experiments and industrial processes. This future enhancement aligns with the project's goal of providing accurate, real-time data acquisition and management, thereby supporting the broader mission of the National Physical Laboratory and similar institutions.

**CHAPTER 6**

# CONCLUSION

The contributions made at the Council of Scientific and Industrial Research - National Physical Laboratory (CSIR-NPL) in India have significantly advanced the time and frequency measurements field. As India's premier institute for maintaining national measurement standards, CSIR-NPL is crucial in ensuring accurate timekeeping across the country. During my internship at NPL, I worked on a project focused on real-time data acquisition and visualization of laser power voltages across all channels of a cesium clock. This project utilized ADC1115 analog-to-digital converters (ADCs), an ESP8266 micro-controller, and technologies including PHP, MySQL, Python, and C++.

A key achievement of this project was developing a highly efficient system for monitoring and recording the laser power voltages from all channels in the cesium clock. The ADC1115's high precision and low noise, combined with the ESP8266's robust data processing and Wi-Fi capabilities, ensured accurate and reliable data capture and transmission. This setup allowed for precise measurement of the voltage levels, which are critical for the performance and calibration of the cesium clock.

Integrating Python Dash into the system provided real-time visualization of these voltage readings through interactive graphs, significantly enhancing data analysis and monitoring capabilities. PHP and MySQL were utilized for adequate data reception and storage, offering a scalable and flexible solution for managing large volumes of data. Additionally, C++ programming enabled efficient microcontroller operation and real-time data processing, ensuring the system's robustness and responsiveness.

Overall, this project represents a significant advancement in real-time monitoring systems, particularly for measuring laser power voltages in the cesium clock at NPL. By enhancing NPL's technical capabilities and contributing to the broader field of time and frequency measurements, this work demonstrates the considerable potential for future innovations in timekeeping systems. The successful integration of high-precision hardware with advanced software underscores the project's value in advancing the accuracy and efficiency of timekeeping systems, reinforcing CSIR-NPL's mission to maintain and disseminate precise time information.

**CHAPTER 7**

# REFERENCES

In developing this project, various resources were utilized to ensure accurate data acquisition, storage, and visualization. The following references provide comprehensive information on the components and technologies used:

**1. Arduino Official Documentation**

- Provides comprehensive guides and references for using Arduino boards, including the ESP8266.

- [Arduino Documentation](https://www.arduino.cc/reference/en/)

**2. ADS1115 Analog-to-Digital Converter**

- Detailed datasheet and usage guidelines for the ADS1115 ADC.

- [ADS1115 Datasheet](https://www.ti.com/lit/ds/symlink/ads1115.pdf)

**3. ESP8266 WiFi Module**

- Documentation and resources for programming and using the ESP8266 microcontroller.

- [ESP8266 Community Forum](https://www.esp8266.com/)

**4. PHP and MySQL Integration**

- Guides on setting up PHP servers and using MySQL for database management.

- [PHP Manual](https://www.php.net/manual/en/)

- [MySQL Documentation](https://dev.mysql.com/doc/)

**5. Python `mysql.connector` Library**

- Official documentation for the `mysql.connector` library used to connect Python applications to MySQL databases.

- [MySQL Connector/Python Developer Guide](https://dev.mysql.com/doc/connector-python/en/)

**6. Python `csv` Module**

- Documentation for the `csv` module used for reading from and writing to CSV files in Python.

- [Python `csv` Module Documentation](https://docs.python.org/3/library/csv.html)

1. **Plotly Dash**

- Official documentation for Plotly Dash, used to create interactive web-based dashboards in Python.

- [Plotly Dash Documentation](https://dash.plotly.com/)

These references successfully implemented the project, providing essential guidance and technical details for each component and technology used.

**CHAPTER 8**

# ANNEXURE

This annexure provides a comprehensive collection of all the source codes used in this project. These codes include Arduino sketches for data acquisition, PHP scripts for data handling, Python scripts for data processing, and the Dash application for data visualization.

#### Arduino Code for Data Acquisition

#include <ESP8266WiFi.h>

#include <Adafruit\_ADS1X15.h>

Adafruit\_ADS1115 ads1; // Address 0x48

Adafruit\_ADS1115 ads2; // Address 0x49

const char\* ssid = "KRC-IST128";

const char\* password = "128istrc@";

const char\* server = "172.16.18.25"; // XAMPP server's IP address

int serverPort = 80;

WiFiClient client;

unsigned long measureStartTime;

bool measuring = false;

void setup() {

Serial.begin(9600);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

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

}

Serial.println("Connected to WiFi");

if (!ads1.begin(0x48)) {

Serial.println("Failed to initialize ADS1.");

while (1);

}

if (!ads2.begin(0x49)) {

Serial.println("Failed to initialize ADS2.");

while (1);

}

measureStartTime = millis();

measuring = true;

}

void loop() {

static int16\_t minAdc[8] = {INT16\_MAX, INT16\_MAX, INT16\_MAX, INT16\_MAX, INT16\_MAX, INT16\_MAX, INT16\_MAX, INT16\_MAX};

unsigned long currentTime = millis();

if (measuring) {

// Measure ADC values for 50 seconds

if (currentTime - measureStartTime < 50000) {

for (int channel = 0; channel < 8; channel++) {

int16\_t adcValue;

if (channel < 4) {

adcValue = ads1.readADC\_SingleEnded(channel);

} else {

adcValue = ads2.readADC\_SingleEnded(channel - 4);

}

if (adcValue < minAdc[channel]) {

minAdc[channel] = adcValue;

}

}

} else {

measuring = false; // Stop measuring after 50 seconds

measureStartTime = currentTime; // Reset timer for sending data

}

} else {

// Pause for 10 seconds to send data

if (currentTime - measureStartTime < 10000) {

// Do nothing, just wait

} else {

float voltage[8];

for (int i = 0; i < 8; i++) {

voltage[i] = (minAdc[i] == INT16\_MAX) ? 0.0 : abs(minAdc[i]) \* 0.1875;

}

float X1 = voltage[0], X2 = voltage[1], Y1 = voltage[2], Y2 = voltage[3];

float D1 = voltage[4], D2 = voltage[5], Z1 = voltage[6], Z2 = voltage[7];

Serial.println("Attempting to connect to server...");

if (client.connect(server, serverPort)) {

Serial.println("Connected to server");

String url = "/phpfiles/save\_data.php";

String postData = "X1=" + String(X1) + "&X2=" + String(X2) +

"&Y1=" + String(Y1) + "&Y2=" + String(Y2) +

"&D1=" + String(D1) + "&D2=" + String(D2) +

"&Z1=" + String(Z1) + "&Z2=" + String(Z2);

client.print("POST ");

client.print(url);

client.println(" HTTP/1.1");

client.print("Host: ");

client.println(server);

client.println("Content-Type: application/x-www-form-urlencoded");

client.print("Content-Length: ");

client.println(postData.length());

client.println();

client.println(postData);

while (client.connected() || client.available()) {

if (client.available()) {

String line = client.readStringUntil('\n');

Serial.println(line);

}

}

client.stop();

Serial.println("Data sent");

// Reset minAdc array for the next measurement cycle

for (int i = 0; i < 8; i++) {

minAdc[i] = INT16\_MAX;

}

// After sending data, start measuring again

measuring = true;

measureStartTime = currentTime;

} else {

Serial.println("Connection to server failed, retrying in 1 second...");

client.stop();

delay(1000);

}

}

}

1. **Save\_data.php Code :**

<?php

$servername = "localhost";

$username = "ESP8266NPL";

$password = "VCI\_ioCJa1M574\_p";

$dbname = "arduino\_data";

// Create connection

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

// Check if data is received

if (isset($\_POST['X1']) && isset($\_POST['X2']) && isset($\_POST['Y1']) && isset($\_POST['Y2']) &&

isset($\_POST['D1']) && isset($\_POST['D2']) && isset($\_POST['Z1']) && isset($\_POST['Z2'])) {

$X1 = $\_POST['X1'];

$X2 = $\_POST['X2'];

$Y1 = $\_POST['Y1'];

$Y2 = $\_POST['Y2'];

$D1 = $\_POST['D1'];

$D2 = $\_POST['D2'];

$Z1 = $\_POST['Z1'];

$Z2 = $\_POST['Z2'];

// Insert data into database

$sql = $conn->prepare("INSERT INTO sensordata (X1, X2, Y1, Y2, D1, D2, Z1, Z2)

VALUES (?, ?, ?, ?, ?, ?, ?, ?)");

$sql->bind\_param('dddddddd', $X1, $X2, $Y1, $Y2, $D1, $D2, $Z1, $Z2);

if ($sql->execute()) {

echo "New record created successfully";

} else {

echo "Error: " . $sql->error;

}

$sql->close();

} else {

echo "No data received";

}

$conn->close();

?>

1. **Fetch&Delete Python Script :**

import mysql.connector

import pandas as pd

from datetime import datetime, date

import time

import os

# Function to calculate Modified Julian Date (MJD) with second precision

def datetime\_to\_mjd(dt):

# MJD starts from 17 November 1858

mjd\_start = datetime(1858, 11, 17)

delta\_days = (dt - mjd\_start).days + (dt - mjd\_start).seconds / 86400

return delta\_days

# Database connection

conn = mysql.connector.connect(

host="localhost",

user="ESP8266NPL",

password="VCI\_ioCJa1M574\_p",

database="arduino\_data"

)

cursor = conn.cursor()

# Function to fetch data from the database

def fetch\_data(date):

query = "SELECT timestamp, X1, X2, Y1, Y2, D1, D2, Z1, Z2 FROM sensordata WHERE DATE(timestamp) = %s"

cursor.execute(query, (date,))

return cursor.fetchall()

# Function to get the last ID from the CSV

def get\_last\_id(file):

if os.path.exists(file):

df = pd.read\_csv(file)

return df['id'].max()

return 0

# Initialize current\_date and file names

current\_date = date.today()

current\_month = datetime.now().strftime("%B\_%Y")

csv\_file\_daily = f'live\_data\_{current\_date}.csv'

csv\_file\_monthly = f'month\_{current\_month}.csv'

# Continuous fetching loop

while True:

# Check if the date has changed

if date.today() != current\_date:

# Update the date and file names

current\_date = date.today()

current\_month = datetime.now().strftime("%B\_%Y")

csv\_file\_daily = f'live\_data\_{current\_date}.csv'

csv\_file\_monthly = f'month\_{current\_month}.csv'

# Fetch data from the database

records = fetch\_data(current\_date)

if records:

last\_id\_daily = get\_last\_id(csv\_file\_daily) # Get the last used ID for daily file

last\_id\_monthly = get\_last\_id(csv\_file\_monthly) # Get the last used ID for monthly file

# Create DataFrame

df = pd.DataFrame(records, columns=['timestamp', 'X1', 'X2', 'Y1', 'Y2', 'D1', 'D2', 'Z1', 'Z2'])

# Convert timestamp column to datetime

df['timestamp'] = pd.to\_datetime(df['timestamp'])

# Calculate MJD for each entry with second precision

df['mjd'] = df['timestamp'].apply(lambda dt: datetime\_to\_mjd(dt))

# Assign unique IDs

df['id'] = range(last\_id\_daily + 1, last\_id\_daily + len(df) + 1)

df['unique\_id'] = range(last\_id\_monthly + 1, last\_id\_monthly + len(df) + 1)

# Save to daily CSV

df.to\_csv(csv\_file\_daily, mode='a', index=False, header=not os.path.exists(csv\_file\_daily))

# Save to monthly CSV

df.to\_csv(csv\_file\_monthly, mode='a', index=False, header=not os.path.exists(csv\_file\_monthly))

# Log the new entries

print(f"Inserted data for {current\_date}: {len(records)} new records")

# Delete the records from the database after fetching

delete\_query = "DELETE FROM sensordata WHERE DATE(timestamp) = %s"

cursor.execute(delete\_query, (current\_date,))

conn.commit()

# Sleep for 120 seconds before the next fetch

time.sleep(240)

# Clean up

cursor.close()

conn.close()

1. **Dash Server Python Code :**

import dash

from dash import dcc, html, Input, Output, State

import pandas as pd

import plotly.graph\_objs as go

import os

from datetime import datetime

# Function to load data within a specified date and time range

def load\_data(start\_datetime=None, end\_datetime=None):

all\_data = []

for filename in os.listdir('.'):

if filename.startswith('live\_data\_') and filename.endswith('.csv'):

df = pd.read\_csv(filename, parse\_dates=['timestamp'])

all\_data.append(df)

if all\_data:

combined\_df = pd.concat(all\_data, ignore\_index=True)

else:

combined\_df = pd.DataFrame()

if start\_datetime and end\_datetime:

combined\_df = combined\_df[(combined\_df['timestamp'] >= start\_datetime) & (combined\_df['timestamp'] <= end\_datetime)]

return combined\_df

# Function to get the latest values from the data

def get\_latest\_values():

df = load\_data()

if not df.empty:

latest\_row = df.iloc[-1]

else:

latest\_row = pd.Series()

return latest\_row

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

server = app.server # Required for deploying with Flask server

# Available ADC channels

adc\_channels = ['X1', 'X2', 'Y1', 'Y2', 'D1', 'D2', 'Z1', 'Z2']

app.layout = html.Div(style={'textAlign': 'center', 'padding': '20px'}, children=[

html.Div(style={'display': 'flex', 'alignItems': 'center', 'justifyContent': 'space-between'}, children=[

html.Img(src='assets/logo.png', className='custom-header', style={'height': '180px', 'margin-right': '60px'}),

html.H1("ADC Channel Data Visualization", className='custom-header', style={'color': '#007bff', 'flex': '1'}),

html.Img(src='assets/logo2.png', className='custom-header', style={'height': '160px', 'margin-left': '60px'})

]),

dcc.DatePickerRange(

id='date-range',

display\_format='YYYY-MM-DD',

className='date-picker-range'

),

html.Div(style={'display': 'flex', 'justify-content': 'center', 'margin-top': '10px'}, children=[

html.Div([

html.Label('Start Time:'),

dcc.Input(id='start-time', type='text', placeholder='HH:MM:SS', className='time-input')

], style={'margin-right': '10px'}),

html.Div([

html.Label('End Time:'),

dcc.Input(id='end-time', type='text', placeholder='HH:MM:SS', className='time-input')

])

]),

html.Div(style={'display': 'flex', 'justify-content': 'center', 'margin-top': '20px'}, children=[

html.Div(style={'flex': '1'}, children=[

dcc.Dropdown(

id='adc-dropdown',

options=[{'label': channel, 'value': channel} for channel in adc\_channels],

value=adc\_channels,

multi=True,

className='dropdown-style'

)

]),

html.Div(style={'flex': '1'}, children=[

html.Div(id='latest-values-table', className='table-container', style={'margin-left': '20px'})

])

]),

dcc.Graph(id='adc-graph', className='custom-graph'),

dcc.RadioItems(

id='time-format',

options=[

{'label': 'Normal Timestamp', 'value': 'timestamp'},

{'label': 'Modified Julian Date (MJD)', 'value': 'mjd'}

],

value='timestamp',

className='custom-container'

),

dcc.Interval(

id='update-interval',

interval=30 \* 1000, # 30 seconds

n\_intervals=0

),

html.Button("Download Data", id="download-button", className="custom-button"),

dcc.Download(id="download-data"),

html.Div(id='click-data', className='click-data-style')

])

@app.callback(

Output('adc-graph', 'figure'),

[

Input('date-range', 'start\_date'),

Input('date-range', 'end\_date'),

Input('start-time', 'value'),

Input('end-time', 'value'),

Input('adc-dropdown', 'value'),

Input('update-interval', 'n\_intervals'),

Input('time-format', 'value'),

]

)

def update\_graph(start\_date, end\_date, start\_time, end\_time, selected\_channels, n\_intervals, time\_format):

if not start\_date or not end\_date or not start\_time or not end\_time:

return go.Figure()

start\_datetime = f"{start\_date} {start\_time}"

end\_datetime = f"{end\_date} {end\_time}"

df = load\_data(start\_datetime, end\_datetime)

if df.empty:

return go.Figure()

figure = go.Figure()

for column in selected\_channels:

figure.add\_trace(go.Scatter(

x=df[time\_format],

y=df[column],

mode='lines+markers',

name=column,

hoverinfo='text',

hovertemplate=f"{column}: %{{y}}<br>{time\_format}: %{{x}}<extra></extra>",

text=df['timestamp'].dt.strftime('%Y-%m-%d %H:%M:%S')

))

figure.update\_layout(

title="ADC Channel Data",

xaxis\_title="Timestamp",

yaxis\_title="Value",

hovermode='closest',

xaxis\_rangeslider\_visible=True,

)

return figure

@app.callback(

Output('latest-values-table', 'children'),

Input('update-interval', 'n\_intervals')

)

def update\_table(n\_intervals):

latest\_row = get\_latest\_values()

if latest\_row.empty:

return html.Table([])

table\_header = [

html.Thead(html.Tr([html.Th('ADC Channel'), html.Th('Latest Value')]))

]

table\_body = [

html.Tbody([

html.Tr([html.Td(channel), html.Td(latest\_row.get(channel, 'N/A'))]) for channel in adc\_channels

])

]

table = table\_header + table\_body

return html.Table(table)

@app.callback(

Output("download-data", "data"),

Input("download-button", "n\_clicks"),

[State('date-range', 'start\_date'),

State('date-range', 'end\_date'),

State('start-time', 'value'),

State('end-time', 'value')],

prevent\_initial\_call=True

)

def generate\_csv(n\_clicks, start\_date, end\_date, start\_time, end\_time):

if not n\_clicks:

return dash.no\_update

if not start\_date or not end\_date or not start\_time or not end\_time:

return None

start\_datetime = f"{start\_date} {start\_time}"

end\_datetime = f"{end\_date} {end\_time}"

df = load\_data(start\_datetime, end\_datetime)

return dcc.send\_data\_frame(df.to\_csv, f"data\_{start\_date}\_{start\_time}\_to\_{end\_date}\_{end\_time}.csv")

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True, host='0.0.0.0', port=8051)