

A Wide Area Synchronized Frequency Measurement System using Network Time Protocol

Arnav Bansal (2022EEB1160)

Arjun Rana (2022EEB1370)

Srijan Kar (2022EEB1349)

(Github Link to the project : https://github.com/arnavb2004/Frequency_estimation)

Objective:

- To develop and implement a low-cost, easy-to-deploy Wide Area Frequency Measurement System (WAFMeS) for power systems.
- To use low-cost hardware, specifically Frequency Measurement Device (FMD) with simple microcontroller setups (Arduino Mega) using Network Time Protocol (NTP) for time synchronization.
- To transmit the time-stamped frequency data over the internet via UDP communication to a central server, store it, and analyse power system disturbances post-facto.
- The system captures time-stamped frequency measurements from distributed client nodes and uploads the data over to the internet, where it is ingested into a MongoDB database for structured storage, retrieval, and real-time analysis.

Methodology:

System Architecture Overview

The overall structure includes:

- Frequency Measurement Devices (FMDs) at each remote site.
- Client computers synchronized using NTP.
- Central server for data collection and analysis.

1. **Hardware:** Frequency Measurement Device (FMD)

Components:

- Step-down Transformer: Reduces 230V AC to a 12V AC.
- RC Filter: Smooths the signal and removes noise in the output.
- Zero Crossing Detector (ZCD): Detects when the AC signal crosses zero voltage.
- Diode to remove the negative pulses in output.

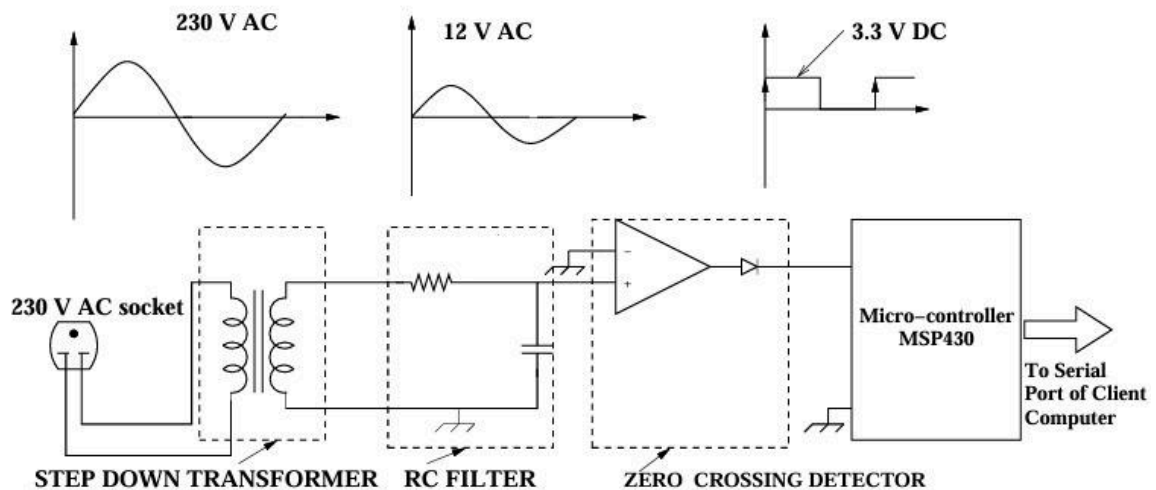


Fig. 2. Block diagram of Frequency Measurement Device

2. Working Principle of Microcontroller and its Code:

Microcontroller (Arduino Mega): Measures the time interval between consecutive zero crossings to calculate frequency.

```
const int signalPin = 2;
volatile unsigned long lastTime = 0;
volatile unsigned long period = 0;
unsigned long lastUpdate = 0;
int readingsIgnored = 0; // Counter to ignore first few unstable readings

void signalISR() {
    unsigned long currentTime = micros();
    period = currentTime - lastTime;
    lastTime = currentTime;
}

void setup() {
    Serial.begin(115200);
    pinMode(signalPin, INPUT);
    attachInterrupt(digitalPinToInterrupt(signalPin), signalISR, RISING);
}

void loop() {
    if (millis() - lastUpdate >= 20) {
        lastUpdate = millis();
        if (period > 0) {
            float frequency = 1000000.0 / period;
            unsigned long timestamp = millis();

            if(timestamp>10000){
                Serial.print(timestamp);
                Serial.print(" ms, Frequency: ");
                Serial.print(frequency, 2);
                Serial.println(" Hz");
            }
        }
    }
}
```

Code explanation :

- This Arduino code measures the frequency of a square wave signal at pin 2.
- An interrupt (signalISR) captures the time between rising edges (i.e., one period) using micros().
- In the loop(), every 20 milliseconds, it calculates the frequency (frequency = 1,000,000 / period) and prints it along with a timestamp (after 10 seconds of running) via serial at 115200 baud rate

3. **Software:**

Client Side

- Each client computer is responsible for:
 - Real-Time Data Acquisition: Reading serial data from Microcontroller Output immediately upon arrival.
 - Timestamping: Tagging the frequency value with a time synchronized via NTP.
 - Graphing the time stamped frequencies for better analysis.
- Network Time Protocol (NTP):
 - Synchronizes the system clocks of client computers.
 - Our operating systems utilize the Network Time Protocol (NTP) to synchronize with internet time servers.
- Store measurements in a database (MongoDB used for scalable storage and querying).

4. **Data Communication**

• **Protocol Used:**

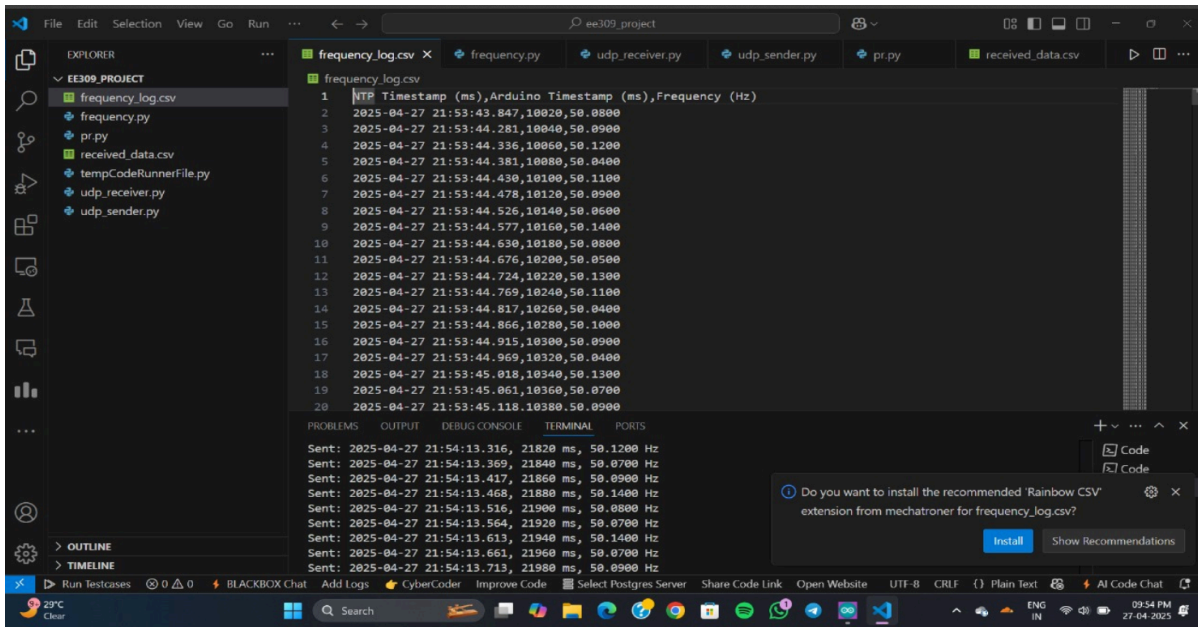
UDP (User Datagram Protocol)

- UDP is connectionless so it is faster and hence it is preferred over TCP.
 - UDP was chosen over TCP because occasional packet loss is acceptable given the high transmission frequency (every 20 ms), and the system prioritizes low latency over guaranteed delivery.
 - Prioritizes continuous, lightweight, and fast data flow.
- **Data Packet Structure:**
 - Each packet includes location ID, timestamp, and measured frequency.
 - Packet Size: ~68 bytes.
 - **Transmission Rate:**
 - Frequency measurements are sent every 20 ms.
 - Very low bandwidth requirement (~4 KB/sec per client).

5. Server-Side Setup

- **Server Responsibilities:**
 - Receive UDP packets from the client computer.
- **Data Handling:**
 - Organize data by timestamp and location.
 - Graphing the time stamped frequencies for better analysis.

Simulations and Results:



The screenshot displays a Visual Studio Code (VS Code) environment. The Explorer sidebar on the left shows a project named 'EE309_PROJECT' containing files: 'frequency_log.csv', 'frequency.py', 'pr.py', 'received_data.csv', 'tempCodeRunnerFile.py', 'udp_receiver.py', and 'udp_sender.py'. The main editor area has 'frequency_log.csv' open, showing a table with three columns: 'NTP Timestamp (ms)', 'Arduino Timestamp (ms)', and 'Frequency (Hz)'. The data consists of 20 rows of timestamp and frequency readings. Below the editor, the 'TERMINAL' panel is active, showing a series of 'Sent' messages with timestamps and frequencies, such as 'Sent: 2025-04-27 21:54:13.316, 21820 ms, 50.1200 Hz'. A notification bubble in the bottom right corner asks if the user wants to install the 'Rainbow CSV' extension.

NTP Timestamp (ms)	Arduino Timestamp (ms)	Frequency (Hz)
2025-04-27 21:53:43.847	10020	50.0800
2025-04-27 21:53:44.281	10040	50.0900
2025-04-27 21:53:44.336	10060	50.1200
2025-04-27 21:53:44.381	10080	50.0400
2025-04-27 21:53:44.430	10100	50.1100
2025-04-27 21:53:44.478	10120	50.0900
2025-04-27 21:53:44.526	10140	50.0600
2025-04-27 21:53:44.577	10160	50.1400
2025-04-27 21:53:44.630	10180	50.0800
2025-04-27 21:53:44.676	10200	50.0500
2025-04-27 21:53:44.724	10220	50.1300
2025-04-27 21:53:44.769	10240	50.1100
2025-04-27 21:53:44.817	10260	50.0400
2025-04-27 21:53:44.866	10280	50.1000
2025-04-27 21:53:44.915	10300	50.0900
2025-04-27 21:53:44.969	10320	50.0400
2025-04-27 21:53:45.018	10340	50.1300
2025-04-27 21:53:45.061	10360	50.0700
2025-04-27 21:53:45.118	10380	50.0900

```
Sent: 2025-04-27 21:54:13.316, 21820 ms, 50.1200 Hz
Sent: 2025-04-27 21:54:13.369, 21840 ms, 50.0700 Hz
Sent: 2025-04-27 21:54:13.417, 21860 ms, 50.0900 Hz
Sent: 2025-04-27 21:54:13.468, 21880 ms, 50.1400 Hz
Sent: 2025-04-27 21:54:13.516, 21900 ms, 50.0800 Hz
Sent: 2025-04-27 21:54:13.564, 21920 ms, 50.0700 Hz
Sent: 2025-04-27 21:54:13.613, 21940 ms, 50.1400 Hz
Sent: 2025-04-27 21:54:13.661, 21960 ms, 50.0700 Hz
Sent: 2025-04-27 21:54:13.713, 21980 ms, 50.0900 Hz
```

Fig: Client Side Communication

	A	B	C	D
1	NTP Times	Arduino Ti	Frequency (Hz)	
2	48:31.6	22100	50.1	
3	48:31.6	22120	50.1	
4	48:31.7	22140	50.09	
5	48:31.7	22160	50.1	
6	48:31.7	22180	50.1	
7	48:31.7	22200	50.09	
8	48:31.7	22220	50.08	
9	48:31.8	22240	50.11	
10	48:31.8	22260	50.1	
11	48:31.8	22280	50.1	
12	48:31.8	22300	50.09	
13	48:31.9	22320	50.1	
14	48:31.9	22340	50.09	
15	48:31.9	22360	50.1	
16	48:31.9	22380	50.09	
17	48:31.9	22400	50.11	
18	48:31.9	22420	50.09	
19	48:32.0	22440	50.1	
20	48:32.0	22460	50.1	
21	48:32.0	22480	50.11	
22	48:32.0	22500	50.11	
23	48:32.1	22520	50.1	
24	48:32.1	22540	50.1	
25	48:32.1	22560	50.1	
26	48:32.1	22580	50.1	
27	48:32.1	22600	50.09	

Fig: Server Side Communication

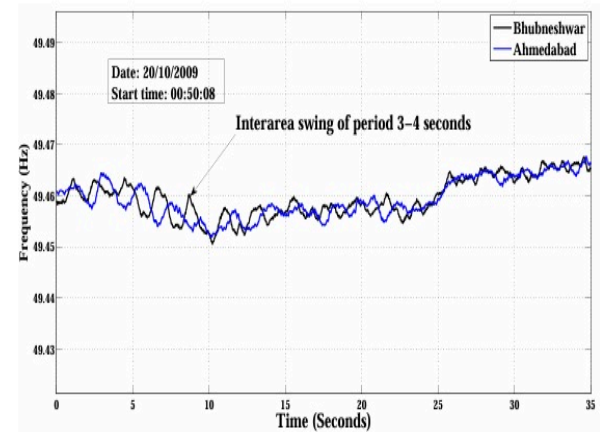
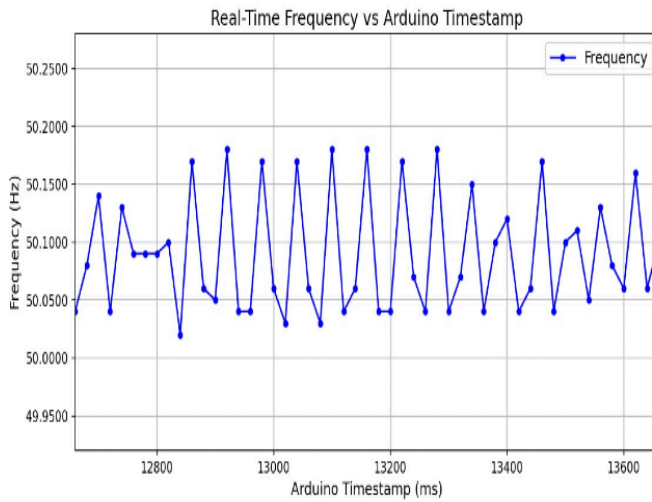


Fig. 12. Plot of frequencies showing inter-area oscillations

Fig: Server Side Graph VS Paper Graph

Discussion:

- **Power System Dynamics:** Power systems experience continuous variations due to changes in load demand, generation scheduling, and unexpected events such as faults and equipment tripping.
- **Need for Monitoring:** These disturbances cause electromechanical transients, including inter-machine oscillations and system frequency fluctuations, which can affect grid stability and performance.
- **Traditional Monitoring Methods:** Conventional monitoring systems use Phasor Measurement Units (PMUs) synchronized via GPS, but they are expensive and require specialized infrastructure.
- **Proposed Solution:** This study presents a low-cost, easily deployable, and internet-based method using the Network Time Protocol (NTP) for synchronized frequency measurement.

- **Understanding Frequency in a Power System:**

1. The frequency of a power system depends on the balance between generation and load. If generation exceeds load, frequency increases and if load exceeds generation, frequency decreases.
2. Large disturbances (like generator trips or major load changes) can cause major frequency deviations.
3. In a power system, electrical power output (P_e) is supplied by generators, which convert mechanical power (P_m) from turbines into electricity.
4. The total mechanical power supplied must match the electrical power consumed plus losses.
5. The generators' rotors have inertia, meaning they store kinetic energy.
6. When generation exceeds load, excess mechanical power increases the kinetic energy of the system, speeding up the rotors and increasing frequency.
7. When load exceeds generation, the system uses kinetic energy from spinning generators, slowing them down and reducing frequency.

Extra Contribution to Project:

As part of the original project requirements, the system was designed to capture frequency data from an Arduino and transmit it over UDP to a designated server. To enhance the functionality and robustness of the system, an additional feature was implemented: real-time cloud storage of the collected data using **MongoDB Atlas**.

In this extended version, every frequency measurement — along with precise timestamps from both the Arduino and the system clock — is now also stored in a MongoDB cloud database. This provides several advantages:

1. **Data Redundancy:** In case of transmission failures or system crashes, no measurements are lost as they are securely stored in the cloud.
2. **Remote Accessibility:** Authorized users can access the live or historical data from anywhere, enabling advanced monitoring and analysis.

3. **Scalability:** MongoDB Atlas supports large volumes of data, making the system future-proof for longer deployments or higher data rates.
4. **Ease of Integration:** Cloud-stored data can later be connected with data visualization dashboards, analytics platforms, or machine learning models for further insights.

Technically, after reading and parsing the serial data, each record is immediately inserted into a MongoDB collection along with structured fields such as system timestamp, Arduino timestamp, and frequency value. This integration ensures that data is simultaneously logged locally (CSV), transmitted (UDP), and preserved in the cloud (MongoDB), thereby significantly enhancing the overall reliability, functionality, and future potential of the project.

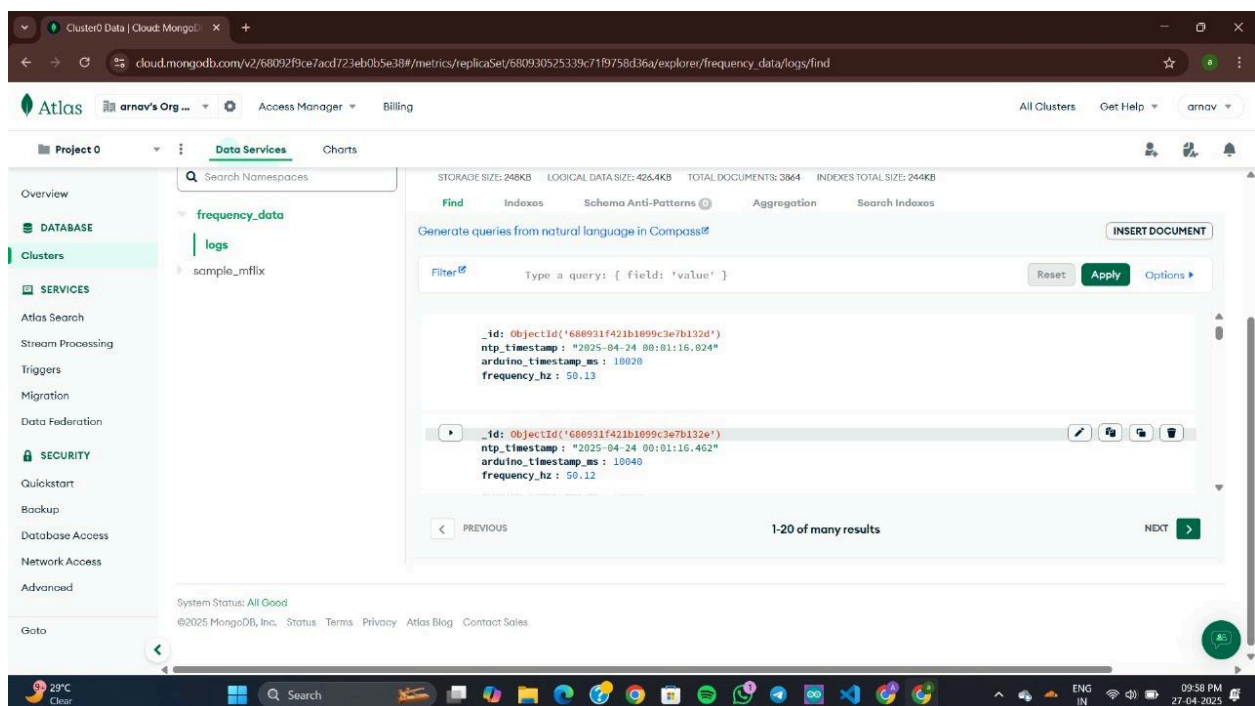


Fig: Mongo DB DataBase

Conclusion:

A low-cost, easily deployable Wide Area Frequency Measurement System (WAFMeS) was developed using Arduino-based Frequency Measurement Devices and NTP synchronization. Time-stamped frequency data was transmitted via UDP and stored in a MongoDB database for real-time analysis.

Despite lower accuracy than GPS, NTP proved sufficient for capturing slow electromechanical transients. The system is effective for wide-area monitoring and post-disturbance analysis of power systems, offering a scalable and practical solution.

References:

- “A Wide Area Synchronized Frequency Measurement System using Network Time Protocol” -Kunal A. Salunkhe and A. M. Kulkarni
- <https://www.geeksforgeeks.org/user-datagram-protocol-udp/>
- <https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/>
- <https://forum.arduino.cc/t/frequency-measurement-with-interrupts/23719>
- <https://www.onetransistor.eu/2018/09/how-to-count-frequency-with-arduino.html>