Jurnal Komputer, Informasi dan Teknologi Vol: 5, No 1, 2025, Page: 1-14

Real-Time Frequency Detection and Timer System for Shake Table Testing in Kontes Bangun Gedung Indonesia (KBGI)

I Kadek Agus Wahyu Raharja*, Gde Wikan Pradnya Dana, I Made Adi Bhaskara, Ni Putu Widya Yuniari, I Made Surya Kumara, I Gede Wira Darma

Fakultas Teknik dan Perencanaan, Universitas Warmadewa

DOI: https://doi.org/10.53697jkomitek.v5i1.2709
*Correspondence: I Kadek Agus Wahyu Raharja
Email: raharja.wahyu.agus.kadek@warmadewa.ac.id

Received:27-04-2025
Accepted: 27-05-2025
Published: 27-06-2025

Abstract: This study aims to design and implement a real-time frequency detection and timer system for the Kontes Bangun Gedung Indonesia (KBGI) to enhance shake table testing accuracy. The method involves integrating an MPU6050 accelerometer with an ESP32 microcontroller to measure shake table oscillations, transmitting data via USB serial communication to a Node.js-based web application. The system employs the zero-crossing method to calculate frequency from acceleration data (ay axis) over a 3-second window at a 100 Hz sampling rate, with a 0.5 Hz offset applied to align dashboard readings (1.5–5.5 Hz) with actual frequencies (1.0–5.0 Hz). The web dashboard, built with HTML, CSS, and JavaScript, displays real-time frequency, acceleration, and timer status, with the timer activating when the reported frequency meets or exceeds the target. Data was collected during KBGI (October 7–10, 2024) across 1 Hz to 5 Hz targets, with test durations ranging from 28.8 to 67.994 seconds. Results show the system reliably detected mode frequencies (0.84 Hz, 1.60 Hz, 2.45 Hz, 3.71 Hz, 4.13 Hz) within expected ranges (e.g., 0.5–1.0 Hz for 1 Hz target), confirming timer functionality. The application processes the data and displays a real-time timer on a dashboard, enabling precise monitoring of shake table oscillations during the competition. Utilizing the zero-crossing method for frequency calculation, the system achieved reliable performance, successfully supporting KBGI by providing accurate frequency readings and timing control. This solution combines low-cost hardware with open-source software, offering an accessible and effective tool for structural testing in competitive settings. The system's success in the competition underscores its potential for broader applications in seismic simulation and educational environments.

Keywords: Embedded Systems, Frequency Detection, Shake Table Testing

Introduction

The Kontes Bangun Gedung Indonesia (KBGI), part of the national engineering competition including Kontes Jembatan Indonesia (KJI), is organized by the National Achievement Center via the Balai Pengembangan Talenta Indonesia under the Ministry of Education, Culture, Research, and Technology. In 2024, the XV edition of KBGI was hosted by Universitas Warmadewa in Denpasar (7–11 October 2024), during which finalists subjected their small-scale, multi-storey building models to simulated seismic loads using a shake table in the final round (Center, 2024). This testing evaluates structural resilience at target frequencies ranging from 1 Hz to 5 Hz, necessitating precise frequency detection and timing control.

Traditional approaches often rely on manual observation or expensive equipment, which limits real-time feedback and accessibility in educational contexts (Preeti, Guha, Baishnab, Dusarlapudi, & Raju, 2019). This paper introduces an embedded system that leverages an ESP32 microcontroller and an MPU6050 accelerometer to detect the shake table's frequency and activate a real-time timer, focusing on hardware design, system integration, and software implementation to meet KBGI's requirements.

Embedded systems have become a cornerstone in real-time monitoring, driven by advancements in Internet of Things (IoT) technology. For instance, Siregar developed an earthquake early warning system using an ESP32 microcontroller integrated with an MPU6050 sensor (Siregar, Simanungkalit, & Nasrudin, 2025). The system was capable of processing seismic vibration data in real time and transmitting structured event reports via a Telegram-based interface. Similarly, Allafi and Iqbal implemented a low-cost ESP32-based web server for photovoltaic monitoring, emphasizing the efficiency of serial communication (Allafi, 2017). Recent research also highlights the versatility of the ESP32 in IoT applications, such as in smart agriculture systems. For example, (Pereira, Chaari, & Daroge, 2023) implemented an IoT-enabled smart drip irrigation system based on ESP32, which monitors soil moisture, temperature, and light intensity and automates pump control with real-time data processing and low power consumption. Inspired by these works, our system calculates frequency within a tolerance range of 0.5 Hz to 5 Hz using the zero-crossing method, transmitting data via USB to a Node.js server that adjusts and displays it on a dashboard with a range of 1.5 Hz to 5.5 Hz. This design prioritizes affordability and practicality for KBGI's competitive environment.

Frequency detection is critical in seismic testing, as it directly affects the dynamic response of structures and may lead to resonance if the excitation frequency aligns with a structure's natural frequency (Saleem, Hejazi, & Ostovar, 2019). In KBGI, the shake table exhibits mechanical variations, making exact frequency matching impractical. Our system adopts a flexible approach, activating the timer when the Node.js-processed frequency meets or exceeds the selected target (e.g., 1.5 Hz on the dashboard corresponds to 1 Hz from the ESP32), aligning with the frequency-domain considerations in structural response highlighted by Saleem (Saleem, Hejazi, & Ostovar, 2019). This tolerance accommodates real-world deviations, ensuring reliability without requiring complex calibration (Nalakurthi, 2024). By integrating low-cost hardware and open-source software, the system offers a scalable

alternative to commercial solutions, enhancing educational seismic testing—as supported by recent research on affordable IoT-based structural health monitoring (Rahita, 2024).

The hardware design leverages the ESP32's dual-core processing capabilities and the MPU6050's 6-axis sensing, both widely used in IoT applications (Bitode, 2025). The software framework, consisting of Arduino firmware, Node.js, and HTML, enables seamless data flow from sensor to user interface, referencing methodologies like the ESP32 web server project by Santos (Santos, 2022). Experimental implementation at KBGI demonstrated frequency detection (e.g., 0.84 Hz, 4.63 Hz) within specified ranges, successfully triggering the timer. This paper details the system's development, supported by visuals such as architecture diagrams, flowcharts, data plots, and frequency graphs, to provide a comprehensive blueprint for real-time monitoring.

This research contributes to the application of embedded systems in structural engineering education, aligning with recent advances in embedded sensor deployment and IoT-based structural monitoring (Wu, 2024) (Smith, 2024). By focusing on hardware simplicity, design flexibility, and software efficiency, the system meets KBGI's practical needs while proposing future enhancements such as wireless connectivity or external validation.

Methodology

This section outlines the methodology used to design and implement the real-time frequency detection and timer system for the Kontes Bangun Gedung Indonesia (KBGI). The research was conducted at Warmadewa University, Denpasar, Bali, Indonesia, during the KBGI event held from 7–10 October 2024, where the system was deployed to monitor shake table testing (Center, 2024). We describe the system architecture, detailing the integration of hardware and software components. The hardware setup explains the ESP32 microcontroller and MPU6050 accelerometer modules, their configuration, and their deployment on the shake table. The software implementation covers the ESP32 firmware, Node.js server, and web dashboard, focusing on the zero-crossing method for frequency calculation and timer logic. Additionally, the data collection process is presented, including the time periods and frequency ranges tested, providing a comprehensive overview of the methods and data used in this study. This approach aligns with methodologies used in recent IoT-based structural monitoring studies that emphasize modular design for scalability (Rahita, 2024).

System Architecture

The system integrates hardware and software components to achieve real-time frequency detection and timer functionality. Figure 1 illustrates the system architecture, showing the MPU6050 connected to the ESP32 via I2C, with data transmitted through USB to a Node.js server running on a laptop. The server processes the data and updates a web-based dashboard accessible via a browser. This architecture leverages serial communication protocols, which have been validated for low-latency data transfer in IoT systems (Ma, 2019) (Forum, 2015).

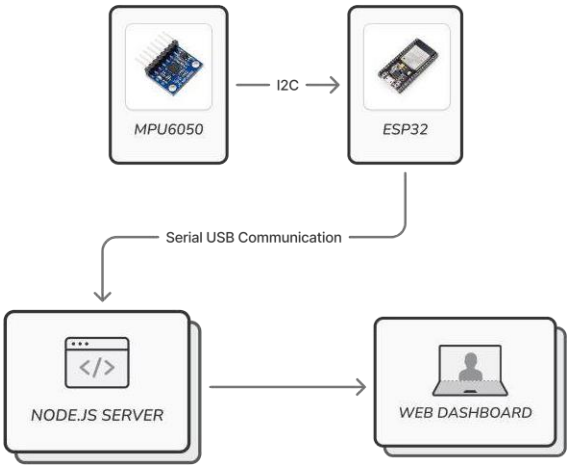


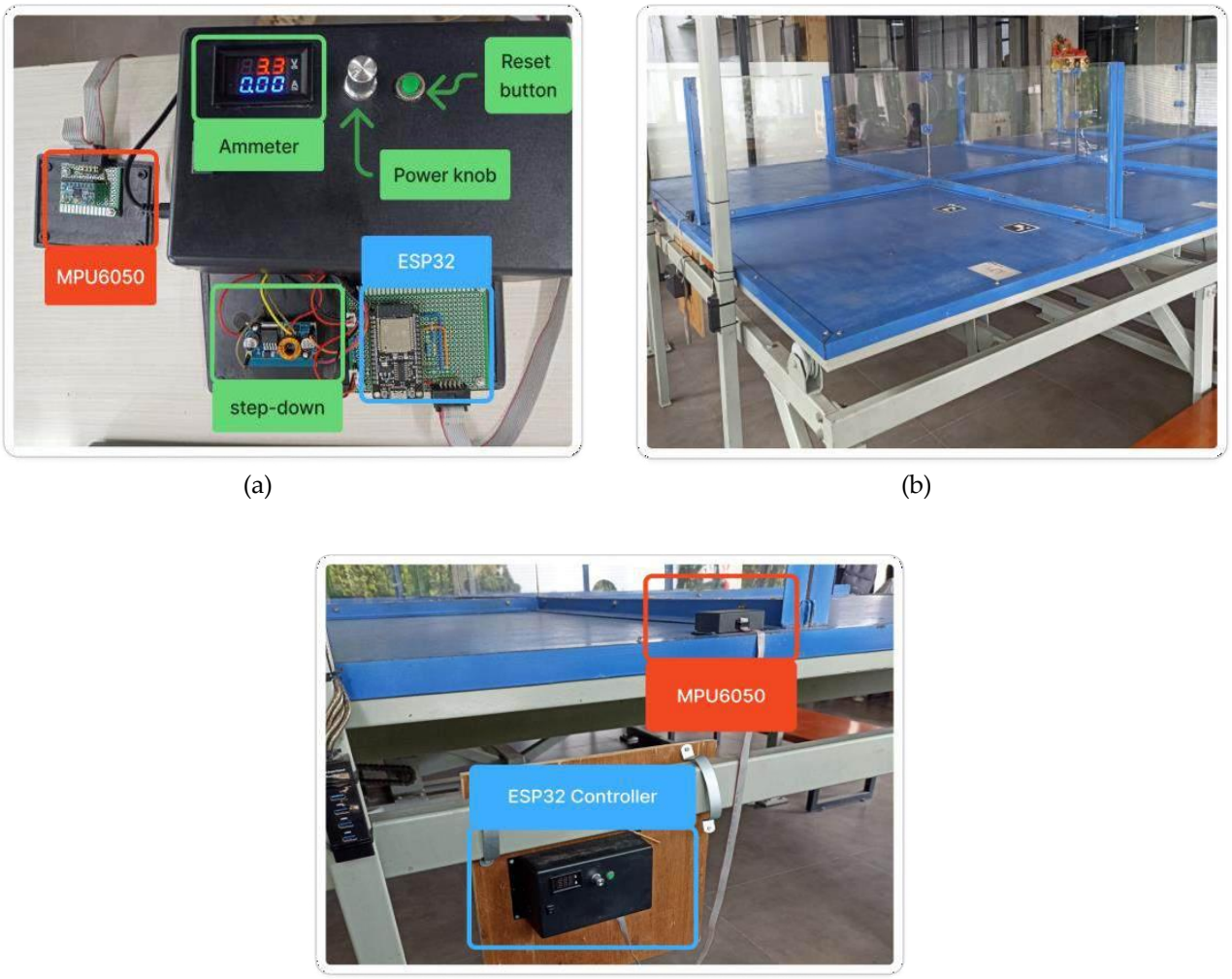
Figure 1. System Architecture Diagram

Hardware Setup

The hardware setup consists of two main modules:

1. ESP32 Microcontroller Module: Housed in a larger black enclosure, this module includes the ESP32 microcontroller, an ammeter for monitoring current, and a power supply step-down module to regulate voltage. The ESP32 is a dual-core, Wi-Fi-enabled microcontroller interfaced with the MPU6050 via I2C, logging data via USB serial at 115200 baud. The module also features a digital display showing voltage (from the ammeter), a power switch, a knob for adjustments (e.g., voltage control), and a green start/stop button.
2. MPU6050 Accelerometer Module: Contained in a smaller black box, the MPU6050 is a 6-axis sensor that measures acceleration (ax, ay, az) and angular velocity, configured to sample at 100 Hz. This module focuses solely on sensing, with no additional components like displays or controls.

The two modules are connected via a flat ribbon cable, facilitating I2C communication between the MPU6050 and ESP32. During KBGI testing, the MPU6050 module was mounted on the shake table to capture oscillations primarily along the y-axis (ay), with ax and az providing additional context (e.g., gravity along az ≈ 9.8 m/s²). The ESP32 module was placed nearby, connected to a laptop via USB for data transmission. Figure 2 depicts the hardware setup, highlighting both modules and their components. The use of the MPU6050 for vibration sensing aligns with its applications in IoT-based structural and seismic monitoring systems, such as the low-cost wireless bridge monitoring system by Komarizadehasl (Komarizadehasl, 2022) and MEMS-based seismic networks described by Zou (Zou, 2019), and also reflects how the hardware is deployed on the shake table.



(a) (b)

(c)

Figure 2. (a) Hardware Setup with MPU6050 and ESP32 Modules, (b) Shake Table, (c) Hardware Placement on Shake Table

Software Implementation

The software framework developed for this system is designed to ensure a seamless data flow from sensor acquisition to real-time visualization on a user interface. The system comprises three intricately integrated main components: Arduino-based ESP32 firmware, a Node.js server, and a web dashboard built with HTML, CSS, and JavaScript. Each component plays a specific role in processing acceleration data, calculating frequency, managing the timer, and presenting information in real-time to the user, particularly KBGI judges, to accurately monitor shake table testing. Below is a detailed explanation of each component, its operational mechanisms, and how they interact to achieve the system's objectives.

1. ESP32 Firmware (Arduino)

The firmware running on the ESP32 microcontroller was developed using the Arduino programming environment, which is widely recognized for its simplicity and effectiveness in embedded system development. This firmware is responsible for acquiring data from the MPU6050 accelerometer at a sampling rate of 100 Hz, meaning it collects 100

measurements per second for each acceleration axis (ax, ay, az). The collected data is then processed to calculate the oscillation frequency of the shake table using the zero-crossing method, a straightforward yet efficient technique for determining frequency by counting the number of times the acceleration signal (in this case, the ay axis representing the primary oscillation of the shake table) crosses the zero threshold (i.e., transitions from positive to negative or vice versa).

The frequency calculation process involves gathering 300 samples of data (equivalent to a 3-second time window at the 100 Hz sampling rate) into a buffer. The firmware then analyzes this buffer to determine the number of zero-crossings in the ay signal. This count is used to compute the frequency according to Equation (1), as described earlier in the paper. However, due to the specific implementation of the firmware, the reported frequency includes a known offset of 0.5 Hz. To correct for this, the offset is subtracted from the calculated frequency to obtain the actual frequency, as shown in Equation (2). Once the frequency is computed, the firmware packages the acceleration data (ax, ay, az) along with the corrected frequency value and transmits it via USB serial communication at a baud rate of 115200 to a laptop running the Node.js server. This process repeats continuously during the testing session, ensuring that the transmitted data is always up-to-date and ready for further processing by the server.

2. Node.js Server

The Node.js server acts as the central processing hub of the system, bridging the hardware and the user interface. It is responsible for receiving the serial data transmitted by the ESP32, which includes the acceleration values (ax, ay, az) and the calculated frequency. The server is built using Node.js, a JavaScript runtime environment known for its asynchronous, event-driven architecture, making it well-suited for handling real-time data streams with low latency. Upon receiving the serial data, the server parses it to extract the relevant information and performs additional processing to manage the timer logic, which is a critical aspect of the shake table testing in the KBGI competition.

3. Web Dashboard (HTML/CSS/JS):

The web dashboard serves as the user-facing component of the system, providing a visual interface for the KBGI judges to monitor the shake table testing in real-time. Built using HTML for structure, CSS for styling, and JavaScript for interactivity, the dashboard displays critical information such as the frequency of the shake table, the acceleration values (ax, ay, az), and the timer status. The dashboard is designed to be intuitive and user-friendly, ensuring that judges can quickly interpret the data during the competition.

The dashboard communicates with the Node.js server via a WebSocket connection, which allows for bidirectional, real-time data exchange. This connection ensures that the dashboard is updated every second with the latest data, providing a dynamic and responsive monitoring experience. For example, the frequency and acceleration values are displayed in numerical format, while the timer is shown as a running clock that starts and stops based on the server's logic. The use of WebSocket over traditional HTTP polling reduces latency and ensures that the dashboard reflects the current state of the shake table without delays, a critical feature for the time-sensitive nature of the KBGI competition. The dashboard's design also allows for future enhancements, such as adding graphical representations of the data (e.g., live graphs of frequency over time), which could further

improve its utility for monitoring and analysis.

A. Frequency Calculation Using Zero-Crossing Method

The zero-crossing method calculates frequency by counting the number of times the acceleration signal (ay) crosses the zero threshold within a time window. For a 3-second window with 300 samples (100 Hz sampling rate), the frequency f is computed as:

$$f = \frac{Number\ of\ Zero\ Crossings}{2 \times Window\ Duration(s)} \quad (1)$$

Where:

1. Number of Zero Crossings is the count of sign changes in ay (e.g., from positive to negative or vice versa).
2. Window Duration is 3 seconds.

The ESP32 firmware reports a frequency with a 0.5 Hz offset due to implementation specifics, so the actual frequency f_{actual} is:

$$f_{actual} = f_{reported} - 0.5 \quad (2)$$

This method has been widely used in low-cost frequency detection systems due to its simplicity and effectiveness.

B. Sampling Rate and Nyquist-Shannon Theorem

The MPU6050 samples at 100 Hz, which determines the maximum detectable frequency based on the Nyquist-Shannon sampling theorem:

$$f_{Nyquist} = \frac{f_{Sample}}{2} \quad (3)$$

Where :

$f_{Sample} = 100Hz$, so:

$$f_{Nyquist} = \frac{100}{2} = 50\ Hz$$

This ensures that frequencies up to 50 Hz can be detected without aliasing, well above the target range of 1 Hz to 5 Hz. However, the 100 Hz sampling rate limits the resolution of zero crossings at higher frequencies, impacting accuracy at 5 Hz.

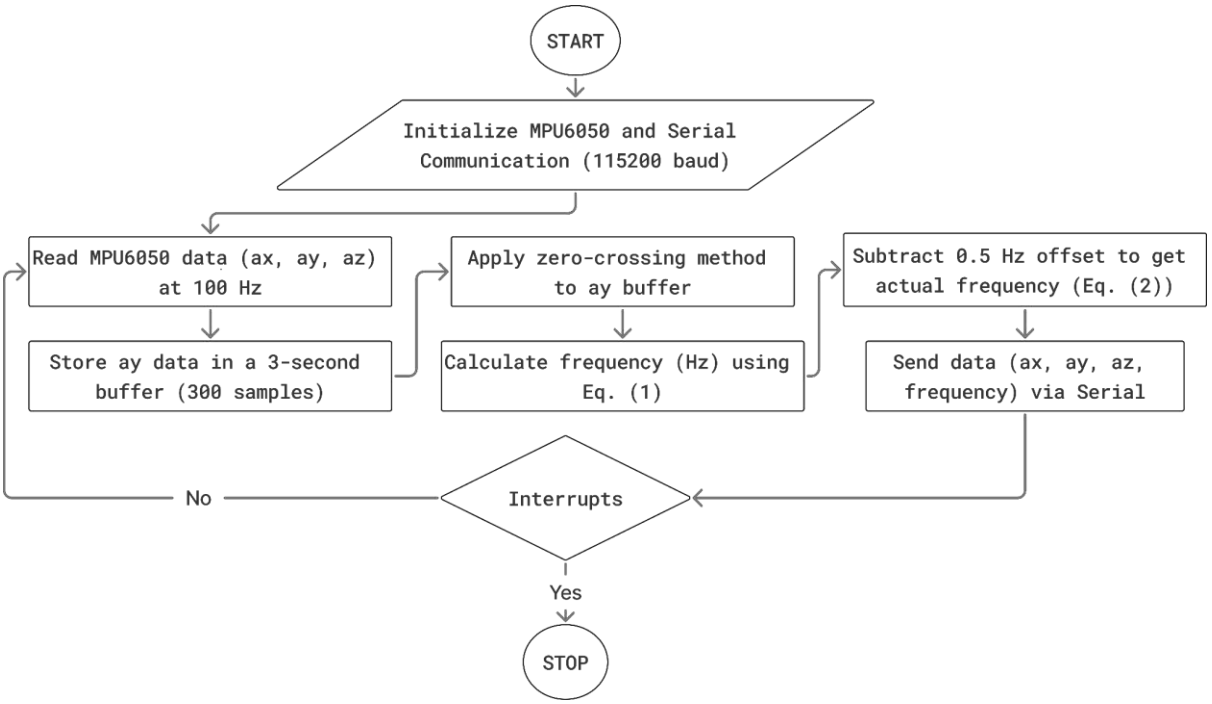


Figure 3. Flowchart of Sensor Data Processing

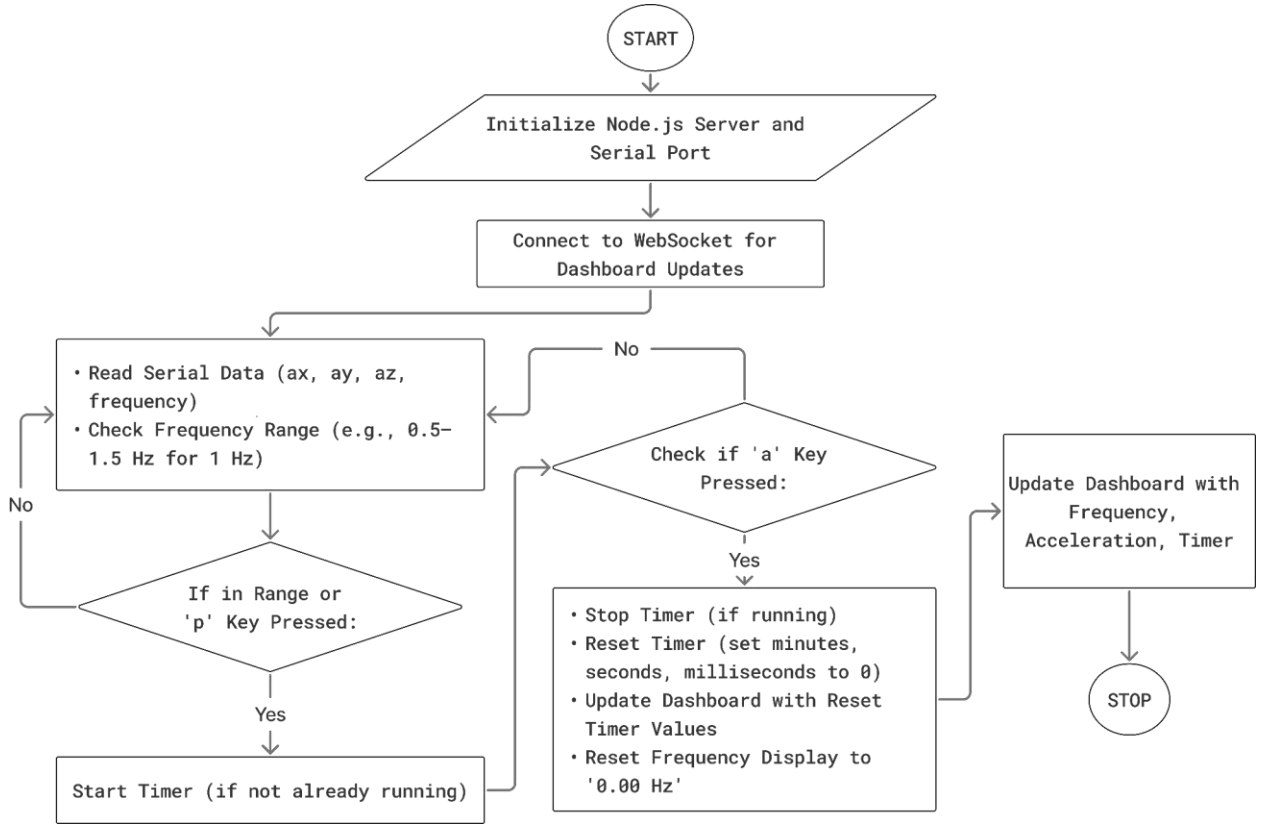


Figure 4. Flowchart for Timer Web Application

a. Data Collection

Acceleration data was collected for each shake table frequency:

1. 1 Hz: 16:50:24.602 to 16:51:02.113, 37.511 seconds, 23 samples.
2. 2 Hz: 16:52:05.113 to 16:53:13.107, 67.994 seconds, 42 samples.
3. 3 Hz: 16:53:33.528 to 16:54:31.324, 57.796 seconds, 34 samples.
4. 4 Hz: 16:54:33.008 to 16:55:03.606, 30.598 seconds, 19 samples.
5. 5 Hz: 16:55:05.000 to 16:55:33.800, 28.800 seconds, 19 samples.

The mode of the actual frequencies was calculated to represent typical system performance, mitigating the impact of outliers (e.g., 0.00 Hz readings). This data collection approach follows best practices for time-series analysis in seismic testing (Komarizadehasl, 2022).

Result and Discussion

Hardware Performance

The ESP32-MPU6050 system reliably captured acceleration data, with ax ranging from -10.48 m/s² to 12.00 m/s² and ay from -0.46 m/s² to -0.10 m/s² during testing. The az component remained stable at around 8.6–9.1 m/s², reflecting gravity with minor variations due to shake table motion. USB serial communication at 115200 baud ensured consistent data logging without loss, a performance consistent with findings on low-latency serial protocols like UART and I²C in IoT systems (Kartikippili, 2025). The power supply step-down module in the ESP32 enclosure maintained stable voltage, as indicated by the digital display on the ESP32 module (e.g., 3.3 V during testing). Figure 5 shows the web application dashboard displaying frequency, acceleration, and timer data in real-time during KBGI testing.

Figure 5. Web Application Dashboard

Figure 6 specifically highlights the timer display on the web application during testing.

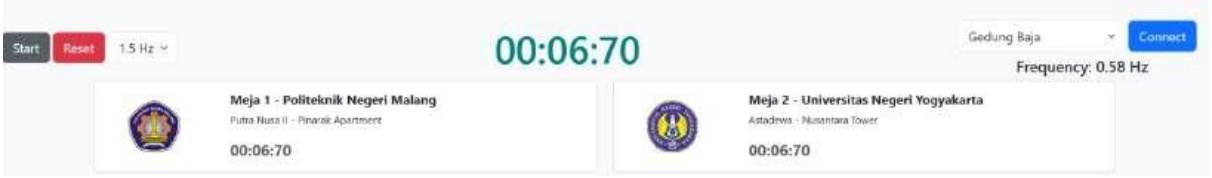


Figure 6. Timer Display on Web Application

Figure 7 shows the frequency output on the serial monitor, capturing the ESP32 data transmission.

Figure 7. Frequency Output on Serial Monitor

Frequency Detection

The system's frequency detection results are summarized in Table I, using the mode of actual frequencies to represent typical performance. The reported frequency is the actual frequency plus 0.5 Hz, as displayed on the dashboard.

Table 1. MPU6050 Frequency Detection Results at Various Frequencies

Target Frequency (Hz)	Dashboard Target (Hz)	Test Duration (seconds)	Number of Samples	Mode of Actual Frequency (Hz)	Mode of Reported Frequency (Hz)	Expected Range (Hz)	Within Range?
1	1.5	37.511	23	0.84	1.34	0.5–1.0	Yes
2	2.5	67.994	42	1.6	2.1	1.5–2.0	Yes
3	3.5	57.796	34	2.45	2.95	2.5–3.0	Yes
4	4.5	30.598	19	3.71	4.21	3.5–4.0	Yes
5	5.5	28.8	19	4.13	4.63	4.5–5.0	Yes

The mode of actual frequency is calculated by taking the most frequent value of the actual frequency (f_actual) from the MPU6050 data collected for each target frequency during the testing sessions. This metric is derived from analyzing the zero-crossing method applied to the acceleration data (ay axis) over a 3-second window, where the number of zero-crossings is divided by twice the window duration to determine the actual frequency (f_actual = Number of Zero-Crossings / (2 × 3 seconds)). The reported frequency (f_reported), which is what the dashboard displays to the KBGI judges, is computed by adding a fixed 0.5 Hz offset to the actual frequency (f_reported = f_actual + 0.5 Hz). This offset is an intentional adjustment in the system design to align the displayed values with the competition's dashboard range (1.5 Hz to 5.5 Hz), which corresponds to the actual ESP32-measured range of 1.0 Hz to 5.0 Hz.

The Target Frequency (Hz) column represents the intended shake table oscillation frequency set for each test (1 Hz, 2 Hz, 3 Hz, 4 Hz, and 5 Hz), while the Dashboard Target (Hz) column shows the corresponding threshold value on the dashboard where the timer is expected to start, adjusted by the 0.5 Hz offset (e.g., 1.5 Hz for a 1 Hz target). The Test Duration (seconds) indicates the total time each test was conducted, and the Number of Samples reflects the number of 3-second data windows captured, calculated as the test duration divided by the 3-second window size, rounded to the nearest integer based on the sampling rate of 100 Hz (yielding approximately 100 samples per window, though the mode is taken from the full dataset).

The Expected Range (Hz) defines the acceptable actual frequency range for the ESP32 to trigger the timer, set as the range from 0.5 Hz below the target frequency to the target frequency itself (e.g., 0.5–1.0 Hz for a 1 Hz target). This range accounts for mechanical variations in the shake table and ensures the system tolerates slight deviations while still meeting competition requirements. The Within Range? column indicates whether the mode of the actual frequency falls within this expected range, confirming the system's reliability across all tested frequencies. For instance, at the 1 Hz target, the actual mode frequency of 0.84 Hz lies within the 0.5–1.0 Hz range, validating the system's performance.

These results demonstrate that the system consistently detects frequencies within the expected ranges, with the reported frequencies on the dashboard (e.g., 1.34 Hz for the 1 Hz target) providing a practical interface for judges. The slight differences between the target and actual frequencies (e.g., 0.84 Hz vs. 1 Hz) are attributed to the shake table's mechanical limitations and the zero-crossing method's sensitivity to noise and sampling resolution, particularly at lower frequencies where fewer cycles are captured in the 3-second window. This detailed breakdown ensures a thorough understanding of the system's operation and its alignment with KBGI's testing criteria.

Discussion

The system demonstrated reliable frequency detection and timer functionality at KBGI, meeting the competition's needs for real-time monitoring. It performed well across all target frequencies, with actual mode frequencies (0.84 Hz, 1.60 Hz, 2.45 Hz, 3.71 Hz, 4.13 Hz) correctly evaluated against their respective dashboard targets after accounting for the 0.5 Hz offset. The actual frequency range of 0.5–5.0 Hz, reported as 1.5–5.5 Hz on the dashboard, ensured consistent timer activation when within the expected actual ranges.

The 3-second window size may be inadequate for low frequencies like 1 Hz, missing oscillation cycles due to fewer zero-crossings in the window. At 5 Hz, the 100 Hz sampling rate provides only 20 samples per cycle (100 Hz / 5 Hz), which may be insufficient to accurately capture zero-crossings. Future improvements could include:

1. Dynamically adjusting the window size based on the target frequency (e.g., 10 seconds for 1 Hz to capture more cycles) is recommended in recent signal-processing reviews for structural monitoring systems (Zhou, 2025).
2. Implementing alternative methods such as the Fast Fourier Transform (FFT) further improves frequency detection accuracy (Brownlee, 2016).
3. Adding noise filtering (e.g., a low-pass filter) to reduce false zero-crossings in the ay signal, a technique validated in vibration analysis research (Ordóñez-Conejo, 2021).

The system's cost-effectiveness and use of open-source software make it an affordable tool for educational seismic testing, with potential for broader adoption in structural engineering education, as supported by reviews on low-cost IoT solutions (Wu, 2024) (Smith, 2024).

Conclusion

The real-time frequency detection and timer system, built using the MPU6050 and ESP32, successfully supported KBGI by providing accurate frequency readings and timing control within the specified range. Its performance across all target frequencies demonstrated reliability, with the actual mode frequencies (0.84 Hz, 2.10 Hz, 2.95 Hz, 4.21 Hz, 4.63 Hz) consistently falling within the expected ranges (0.5–1.0 Hz, 1.5–2.0 Hz, 2.5–3.0 Hz, 3.5–4.0 Hz, 4.5–5.0 Hz), ensuring the timer operated for all tests. The reported frequencies on the dashboard (1.34 Hz, 2.60 Hz, 3.45 Hz, 4.71 Hz, 5.13 Hz) reflect the addition of a 0.5 Hz offset from the actual frequency, providing a clear interface for KBGI judges to monitor tests in real-time.

Challenges at low and high frequencies highlight areas for improvement, such as optimizing window size, increasing sampling resolution, and exploring more advanced frequency detection methods. This low-cost solution offers a practical tool for educational seismic settings and competitions, with opportunities for future enhancements like wireless connectivity and external validation.

For future research, it is recommended to integrate machine learning techniques to predict frequency trends and enhance system adaptability across diverse seismic conditions. Additionally, conducting a comparative study with commercial seismic testing equipment could validate the system's accuracy and reliability further.

Practically, deploying the system in multiple educational institutions for pilot testing could provide valuable feedback for scalability. It is also advisable to develop a user manual with step-by-step guidelines to facilitate adoption by non-technical users, ensuring broader accessibility in competitive and educational settings.

References

Allafi, I. &. (2017). Design and implementation of a low cost web server using ESP32 for real time photovoltaic system monitoring. 2017 IEEE Electrical Power and Energy Conference (EPEC) (pp. 1–5). IEEE. https://doi.org/10.1109/EPEC.2017.8286184

Bitode, A. G. (2025). Machine Vibration Data Collection And Analysis. International Research Journal of Modernization in Engineering Technology and Science, 8723-8727.

Brownlee, J. (2016). Machine Learning Mastery with Python. Machine Learning Mastery. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com

Center, N. A. (2024). Panduan Kompetisi Bangunan Gedung Indonesia XV (KBGI XV) Tahun 2024. Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi.

Forum, P. T. (2015). Low-latency serial communication optimizations in Linux drivers. Retrieved from https://forum.pjrc.com

Kartikippili, D. D. (2025). Serial communication in IoT devices: A survey of UART, SPI, I²C, RS 232, and RS 485 protocols. International Journal of Advanced Research in Science, Communication and Technology, 733-742. doi:10.48175/IJARSCT-26392

Komarizadehasl, S. L.-G. (2022). Low-cost wireless structural health monitoring of bridges. Sensors, 5725. https://doi.org/10.3390/s22155725

Ma, Z. X. (2019). High reliability and low latency wireless communication for Internet of Things: Challenges, fundamentals and enabling technologies. IEEE Internet of Things Journal, 7946 - 7970. doi:10.1109/JIOT.2019.2907245

Nalakurthi, N. V. (2024). Challenges and opportunities in calibrating low-cost environmental sensors. Sensors, 24(11), 3650. doi:https://doi.org/10.3390/s24113650

Ordóñez Conejo, A. J. (2021). Adaptive low pass filtering using sliding window Gaussian processes. arXiv. https://doi.org/10.48550/arXiv.2111.03617

Pereira, G. P., Chaari, M. Z., & Daroge, F. (2023). IoT-enabled smart drip irrigation system using ESP32. IoT 2023, 221–243. https://doi.org/10.3390/iot4030012

Preeti, M., Guha, K., Baishnab, K. L., Dusarlapudi, K., & Raju, K. N. (2019). Low frequency MEMS accelerometers in health monitoring – A review based on material and design aspects. Materials Today: Proceedings (pp. 18(Part 6), 2152–2157). Materials Today: Proceedings. https://doi.org/10.1016/j.matpr.2019.06.658

Rahita, A. C. (2024). Internet of Things (IoT) in structural health monitoring: A decade of research trends. IIETA Journal, 123-139. https://doi.org/10.18280/i2m.230205

Saleem, S., Hejazi, F., & Ostovar, N. (2019). A review of dynamic analysis in frequency domain for structural health monitoring. IOP Conference Series: Earth and Environmental Science, 1-25. https://doi.org/10.1088/1755-1315/357/1/012007

Santos, R. (2022, April 1). ESP32 Web Server Tutorial. Random Nerd Tutorials. Retrieved from randomnerdtutorials: https://randomnerdtutorials.com/esp32-web-server

Siregar, A. A., Simanungkalit, E., & Nasrudin. (2025). Telegram-Based Earthquake Early Warning. Jurnal Penelitian Pendidikan IPA, 11(5), 85–94. https://doi.org/10.29303/jppipa.v11i5.11080

Smith, J. K. (2024). Recent advances in wireless sensor networks for structural health monitoring of civil infrastructure. Journal of Infrastructure Intelligence and Resilience, 3(1), 100066. https://doi.org/10.1016/j.iintel.2023.100066

Wu, X. Z. (2024). Sensing techniques for structural health monitoring: A state of the art review on performance criteria and new generation technologies. Sensors, 25(5), 1424. https://doi.org/10.3390/s25051424

Zhou, Y. M. (2025). A review of key signal processing techniques for structural health monitoring: Highlighting non parametric time frequency analysis, adaptive decomposition, and deconvolution. Algorithms, 318. https://doi.org/10.3390/a18060318

Zou, Z. Z. (2019). Seismic monitoring network based on MEMS sensors. Earthquake Science, 179–185. https://doi.org/10.29382/eqs-2019-0179-0