IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/47ICT/2026/4725426/
Vol. 2026 (13), Article ID 4725426

Hierarchical ESP32 Multi-Node Inertial Data Acquisition System with a Custom Binary IMU Protocol*

Michał KIMBAR, Fryderyk GAJDZIK and Pawel KROWICKI

Wroclaw University of Science and Technology, Faculty of Mechanical Engineering, Wroclaw, Poland,

Correspondence should be addressed to: Fryderyk GAJDZIK, fryderykgajdzik@gmail.com

*Presented at the 47th IBIMA International Conference, 29-30 June 2026, Madrid, Spain

Abstract

In multi-node measurement systems, direct storage of samples in CSV files is insufficient for organizing data acquisition from multiple sources, buffering, wireless transmission, and stream demultiplexing. This paper presents the design and functional verification of the software layer of an inertial data acquisition system based on ESP32 microcontrollers and LSM9DS1 MEMS sensors. The prototype uses a hierarchical architecture: slave nodes perform local IMU readout and buffering, a master node polls them through ESP-NOW, aggregates binary frames, and forwards the stream to a computer through Bluetooth Serial/SPP, while the computer decodes the stream in Python and stores the data in CSV files. The central element of the proposed solution is a binary poll-response protocol based on POLL_REQUEST, DATA_FRAME, and STATUS_FRAME messages. A data frame consists of a 16-byte header, a 216-byte IMU data payload, and a CRC-16 checksum. Functional verification confirmed correct node polling, frame-type recognition, CRC checking, 16-bit value decoding, source demultiplexing, CSV storage, and the use of decoded data for illustrative Pitch and Roll estimation.

Keywords: ESP32; ESP-NOW; Bluetooth Serial/SPP; MEMS; IMU; LSM9DS1; binary protocol; CRC-16; Python; data acquisition.

Introduction

Inertial data acquisition systems based on microcontrollers are often initially developed around a single configured sensor and direct sample logging to a file. In a multi-node system, such a model becomes restrictive. Data generated by several modules must be identified, buffered, transmitted wirelessly, aggregated at a single receiving point, and separated according to the originating source.

The problem is primarily software- and system-oriented. On the microcontroller side, cyclic sensor readout, buffer management, and non-blocking communication must be coordinated. On the receiver side, frames must be acquired, verified for integrity, classified by message type, demultiplexed by source, and stored in a format suitable for further analysis.

Cite this Article as: Michał KIMBAR, Fryderyk GAJDZIK and Pawel KROWICKI Vol. 2026 (13) "Hierarchical ESP32 Multi-Node Inertial Data Acquisition System with a Custom Binary IMU Protocol " Communications of International Proceedings, Vol. 2026 (13), Article ID 4725426, https://doi.org/10.5171/2026.4725426

A previous publication presented the general prototype of a modular device for limb motion analysis (Kimbar and Grysiński, 2025). The present paper narrows the scope to the software layer: firmware organization, transmission structure, binary frame protocol, buffering, data reception, and downstream processing.

Technological Background and Related Work

MEMS inertial sensors are widely used in portable motion measurement systems due to their small size, low power consumption, and ability to record multi-axis kinematic data (Digo et al., 2022; García-de-Villa et al., 2023). Accelerometer and gyroscope data can be used both for raw signal recording and for estimating the orientation of body segments (Nazarahari and Rouhani, 2021; Caruso et al., 2021; Laidig and Seel, 2023). In multi-sensor systems, however, the organization of the data stream itself is also a critical design issue.

The prototype uses the LSM9DS1 MEMS sensor, which integrates a three-axis accelerometer, a three-axis gyroscope, and a three-axis magnetometer (STMicroelectronics, 2015). A single IMU sample set contains nine components: linear acceleration, angular velocity, and the magnetic field measured along three axes. In the communication protocol, each component is encoded as an int16 value read from pairs of sensor registers. This representation defines the transmission and decoding format; it should not be interpreted as a separate assessment of the effective metrological resolution of the sensor.

Communication between ESP32 modules was implemented using ESP-NOW, a protocol that enables direct data exchange between ESP devices without establishing a conventional Wi-Fi connection (Pasic et al., 2021; Urazayev et al., 2023). This organization is consistent with approaches used in wireless sensor networks, where a local aggregation node structures data before forwarding it to the next processing layer (Landaluce et al., 2020; Ojha and Gupta, 2025). The second communication channel is Bluetooth Serial/SPP, through which the master module forwards frames to the computer. On the computer side, a Python decoder based on pySerial and the struct module was used (pySerial Developers, 2025; Python Software Foundation, 2026). Frame integrity is checked using CRC-16 (Koopman and Chakravarty, 2004).

Aim and Contribution

The aim of this work was to develop and functionally verify a software layer for organizing the acquisition, transmission, decoding, and storage of inertial data originating from multiple ESP32 modules.

The scope of the work includes ESP32 firmware, LSM9DS1 configuration through I²C, cyclic IMU readout, buffering, polling of slave nodes, ESP-NOW transmission, Bluetooth Serial/SPP forwarding, Python-based frame reception and decoding, CSV storage, and illustrative Pitch/Roll processing.

The main contributions of the paper are as follows:

1. design of a hierarchical data acquisition architecture consisting of ESP32 slave nodes, an ESP32 master node, and
   a computer receiver;
2. replacement of text-based polling with a binary poll-response protocol;
3. formalization of POLL_REQUEST, DATA_FRAME, and STATUS_FRAME messages;
4. introduction of a frame header containing protocol version, message type, source and destination identifiers, sequence number, local timestamp, payload length, and CRC-16;
5. implementation of a Python decoder responsible for stream resynchronization, CRC validation, source_id-based demultiplexing, CSV storage, and downstream processing.

Proposed Architecture

The architecture is divided into three layers. Slave nodes are responsible for local IMU readout and preparation of data blocks. The master node acts as an aggregation gateway: it performs its own measurement, polls the slave nodes, receives frames through ESP-NOW, queues data, and forwards it to the computer through Bluetooth Serial/SPP.

The computer is responsible for decoding, storage, and downstream processing. The resulting layered architecture of the IMU data acquisition system is shown in Fig 1.

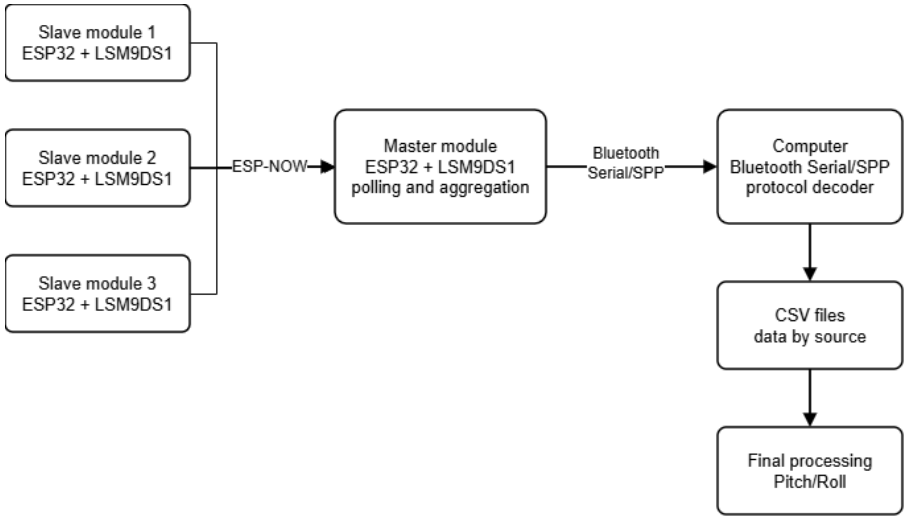

Fig 1. Layered architecture of the IMU data acquisition system.

In the prototype configuration, three ESP32 + LSM9DS1 slave nodes and one master node equipped with its own LSM9DS1 sensor were considered. This enables parallel acquisition from four IMU nodes: three nodes polled through ESP-NOW and one local IMU node integrated with the master module. Each node generates sample sets containing acceleration, angular velocity, and magnetic field components, while source identification is handled at the protocol level through the source_id field.

This division separates acquisition from analysis. Microcontrollers perform hardware-adjacent tasks such as sensor readout, buffering, and transmission. The computer executes less time-critical operations, including decoding, file storage, unit conversion, angle estimation, and visualization.

Design Rationale

The adopted architecture was motivated by the need to combine continuous IMU acquisition with the constraints of wireless communication and microcontroller resources. The master node reduces the number of connections handled by the computer because it aggregates data from slave nodes and forwards them as a single ordered binary stream. Separation of the individual data sources is performed at the protocol decoder level using the source_id field. Communication between ESP32 modules was based on ESP-NOW, while communication with the computer used Bluetooth Serial/SPP. ESP-NOW enables direct frame exchange between microcontrollers without a conventional Wi-Fi connection, whereas Bluetooth Serial/SPP allows the master node to be treated as a single serial data source on the computer side.

The poll-response model was selected instead of a push-based transmission scheme because the master node retains control over the polling rhythm of the slave nodes. In a multi-node setup, this reduces uncontrolled transmission from multiple sources and simplifies queue organization. A slave node responds only after receiving a POLL_REQUEST frame, returning either a DATA_FRAME or a STATUS_FRAME.

The introduction of a frame header, sequence number, local timestamp, explicit payload length, and CRC-16 improves protocol clarity. The sequence_id field enables subsequent analysis of stream discontinuities, timestamp_us describes the local data generation time, payload_len separates header interpretation from payload length, and CRC-16 allows corrupted frames to be rejected.

Binary Frame Protocol and Acquisition Flow

The system uses a binary poll-response protocol. The master node sends a POLL_REQUEST frame. A slave node responds with a DATA_FRAME if a measurement buffer is ready, or with a STATUS_FRAME if it reports buffer state or diagnostic information.

Each frame consists of a 16-byte header, an optional data payload, and a 2-byte CRC-16 checksum. The header contains the frame start marker, protocol version, message type, source identifier, destination identifier, sequence number, local timestamp, and payload length. Multi-byte values are encoded in little-endian order. The structure of the DATA_FRAME used for IMU data transmission is presented in Fig. 2. The frame types used in the communication protocol are summarized in Table 1.

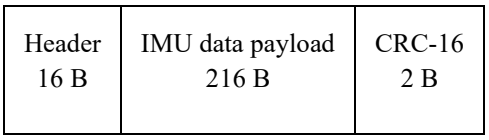

Fig 2. Format of the DATA_FRAME

Tab 1: Frame types used in the communication protocol.

Frame type	Size	Role	Main fields
POLL_REQUEST	18 B	polling of a slave node	header, CRC-16
DATA_FRAME	234 B	transmission of an IMU block	header, 216 B IMU data payload, CRC-16
STATUS_FRAME	32 B	slave-node diagnostics	status code, buffer state, diagnostic counters

A DATA_FRAME carries twelve consecutive IMU sample sets. Each set contains nine 16-bit values: Ax, Ay, Az, Mx, My, Mz, Gx, Gy, and Gz. The number of twelve sample sets represents a compromise between header and CRC overhead, transmission frequency, and buffer fill time. This format reduces the relative proportion of metadata in each transmission while avoiding excessive waiting time before a measurement block becomes available.

A DATA_FRAME of 234 B fits within the 250 B ESP-NOW v1.0 data limit (Espressif Systems, 2026b).With a 15 ms readout interval, filling one frame requires approximately 180 ms, which corresponds to about 5.6 frames/s per node.
In a four-node configuration, this gives approximately 22 frames/s and about 5.2 kB/s of DATA_FRAME data, excluding communication-layer overhead, POLL_REQUEST control frames, and status frames.

A STATUS_FRAME contains a 14-byte diagnostic data field; therefore, its total size is 16 B of header, 14 B of status data, and 2 B of CRC-16. After receiving a valid POLL_REQUEST, a slave node checks buffer readiness and responds with either a data frame or a status frame. The master node validates the response and forwards data frames to the computer.

Prototype Implementation

The firmware was prepared for the ESP32 platform using the Arduino/ESP32 environment (Espressif Systems, 2026a). In the master node, the implementation includes a polling task, a queue for frames received through ESP-NOW, a queue for frames to be transmitted through Bluetooth, and a task responsible for forwarding data to the computer. In the slave node, the implementation includes IMU payload buffering, POLL_REQUEST handling, and generation of DATA_FRAME and STATUS_FRAME messages. The message sequence used in the binary poll-response protocol is shown in Fig 3.

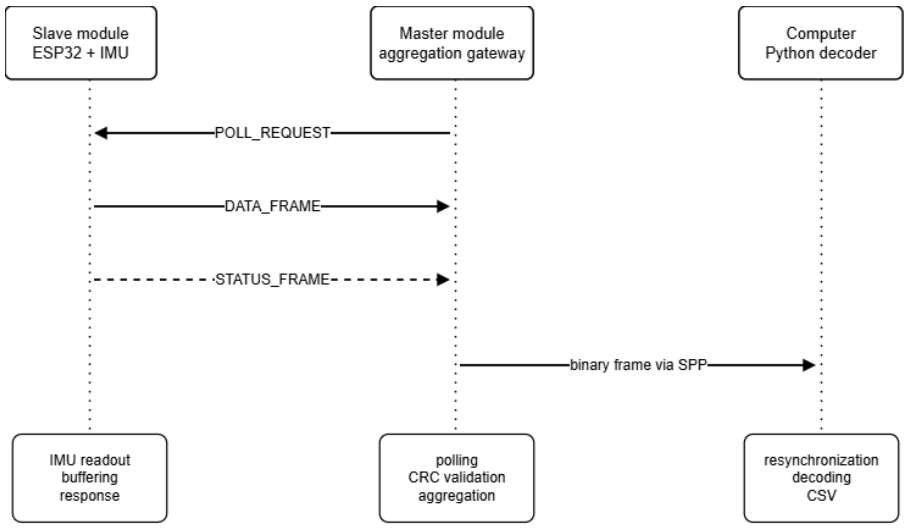

Fig 3. Message sequence in the binary poll-response protocol. Source: author's own work.

In a slave node, periodic sensor readout was performed with a 15 ms interval. After each readout, accelerometer, magnetometer, and gyroscope values are appended to a buffer. After twelve sample sets have been collected, a 216-byte IMU data payload is formed and can be transmitted as part of a DATA_FRAME.

On the computer side, the Python decoder searches for the frame start marker, reads the header, checks the protocol version, payload length, and CRC. For a DATA_FRAME, it decodes 216 B as 108 int16 values, divides the data into twelve IMU sample sets, and stores them in CSV format together with frame metadata: source_id, sequence_id, timestamp_us, and the sample-set index.

The data stored in CSV files can be used for downstream processing. In the prototype, Pitch and Roll angles were estimated using a complementary filter (AHRS Documentation, 2025):

$$\theta_k = \alpha\left(\theta_{k-1} + \omega_y \Delta t\right) + (1-\alpha)\theta_{\mathrm{acc},k} \tag{1}$$

$$\phi_k = \alpha(\phi_{k-1} + \omega_x \Delta t) + (1-\alpha)\phi_{\mathrm{acc},k} \tag{2}$$

where θ denotes Pitch, φ denotes Roll, and α = 0.95.

Functional Verification

The verification focused on the correctness of the data path in a bench-top configuration. The complete flow was checked: from IMU readout and buffering to transmission, decoding, CSV storage, and downstream processing.

On the communication side, the verification covered generation of POLL_REQUEST frames, slave-node responses using either DATA_FRAME or STATUS_FRAME, message-type recognition, payload-length consistency,
and CRC-16 correctness. On the receiver side, the verification covered resynchronization using the frame start marker, header decoding, interpretation of source_id, sequence_id, and timestamp_us, IMU payload decoding, and CSV storage.

In the bench-top configuration, all checked stages of the data path were executed correctly. Data could be acquired by measurement nodes, transmitted to the master node, forwarded to the computer, verified at the header and CRC level, separated according to the source identifier, stored in CSV files, and used by a script computing Pitch and Roll angles. These results should be interpreted as functional verification of the data path. Quantitative assessment of packet loss, sampling jitter, transmission latency, and orientation accuracy requires a separate experimental procedure.

Discussion and Limitations

The proposed protocol structures the transmission of IMU blocks by introducing explicit frame types, source and destination identifiers, sequence numbering, a local timestamp, a length field, and CRC-16. As a result, a frame carries not only measurement values but also the minimal communication context required for demultiplexing, integrity checking, and subsequent transmission diagnostics.

Another relevant aspect is the separation of firmware from downstream processing. Microcontrollers are responsible for readout, buffering, and transmission, whereas the computer performs decoding, storage, and analysis. This division reduces the computational load on the ESP32 nodes and allows processing algorithms to be developed on the Python side.

The current protocol does not address all problems associated with multi-node transmission. The timestamp_us field is a local timestamp generated by a given microcontroller and therefore does not provide full clock synchronization between nodes. CRC-16 detects frame-content errors but does not provide correction or retransmission. Further work should include time synchronization between modules, packet-loss analysis, jitter assessment, and evaluation of the impact of shared ESP32 radio resources used by ESP-NOW and Bluetooth Serial/SPP (Urazayev et al., 2023; Magzym et al., 2023).

Conclusions

This paper presented the software layer of a multi-node inertial data acquisition system based on ESP32 and LSM9DS1 MEMS sensors. The developed data path includes local IMU readout, buffering, binary polling over ESP-NOW, aggregation in a master node, Bluetooth Serial/SPP transmission, Python-based frame decoding, CSV storage, and illustrative Pitch/Roll processing.

The main result of the work is the replacement of simple block transmission with an explicit binary poll-response frame protocol. The header fields enable source and destination identification, message-type recognition, frame sequencing, local timestamping, payload-length checking, and integrity validation using CRC-16.

Functional verification confirmed the correctness of the main software-based data acquisition path. Further development should focus on longer transmission tests, time synchronization between nodes, and assessment of orientation accuracy against a reference system.

References

- AHRS Documentation (2025), Complementary Filter. [Online]. AHRS Documentation. Available at: https://ahrs.readthedocs.io/en/latest/filters/complementary.html.
- Caruso, M., Sabatini, A. M., Laidig, D., Seel, T., Knaflitz, M., Della Croce, U. and Cereatti, A. (2021), 'Analysis of the Accuracy of Ten Algorithms for Orientation Estimation Using Inertial and Magnetic Sensing under Optimal Conditions: One Size Does Not Fit All', Sensors, vol. 21, no. 7, pp. 2543, doi: 10.3390/s21072543.
- Digo, E., Pastorelli, S. and Gastaldi, L. (2022), 'A Narrative Review on Wearable Inertial Sensors for Human Motion Tracking in Industrial Scenarios', Robotics, vol. 11, no. 6, pp. 138, doi: 10.3390/robotics11060138.
- Espressif Systems (2026a), Arduino Core for ESP32 Documentation. [Online]. Espressif Systems. Available at: https://docs.espressif.com/projects/arduino-esp32/en/latest/.
- Espressif Systems (2026b), ESP-NOW API Reference. [Online]. Espressif Systems. Available at: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_now.html.
- García-de-Villa, S., Casillas-Pérez, D., Jiménez-Martín, A. and García-Domínguez, J. J. (2023), 'Inertial Sensors for Human Motion Analysis: A Comprehensive Review', IEEE Transactions on Instrumentation and Measurement, vol. 72, pp. 1-39, doi: 10.1109/TIM.2023.3276528.
- Kimbar, M. and Grysiński, T. (2025), 'Projekt i realizacja modułowego urządzenia umożliwiającego analizę ruchu kończyn z funkcją wieloosiowego pomiaru przyspieszenia', In: Mydlikowski, R. (ed.), Stowarzyszenie Elektryków Polskich na Politechnice Wrocławskiej 2025: trendy i rozwiązania technologiczne w elektrotechnice, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, pp. 239-247, doi: 10.37190/SEP_Trendy_i_rozwiazania_2025.

- Koopman, P. and Chakravarty, T. (2004), 'Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks', International Conference on Dependable Systems and Networks, pp. 145-154, doi: 10.1109/DSN.2004.1311885.
- Laidig, D. and Seel, T. (2023), 'VQF: Highly Accurate IMU Orientation Estimation with Bias Estimation and Magnetic Disturbance Rejection', Information Fusion, vol. 91, pp. 187-204, doi: 10.1016/j.inffus.2022.10.014.
- Landaluce, H., Arjona, L., Perallos, A., Falcone, F., Angulo, I. and Muralter, F. (2020), 'A Review of IoT Sensing Applications and Challenges Using RFID and Wireless Sensor Networks', Sensors, vol. 20, no. 9, pp. 2495, doi: 10.3390/s20092495.
- Magzym, Y., Eduard, A., Urazayev, D., Fafoutis, X. and Zorbas, D. (2023), 'Synchronized ESP-NOW for Improved Energy Efficiency', 2023 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), pp. 57-62, doi: 10.1109/BlackSeaCom58138.2023.10299761.
- Nazarahari, M. and Rouhani, H. (2021), '40 Years of Sensor Fusion for Orientation Tracking via Magnetic and Inertial Measurement Units: Methods, Lessons Learned, and Future Challenges', Information Fusion, vol. 68, pp. 67-84, doi: 10.1016/j.inffus.2020.10.018.
- Ojha, A. and Gupta, B. (2025), 'Evolving Landscape of Wireless Sensor Networks: A Survey of Trends, Timelines, and Future Perspectives', Discover Applied Sciences, vol. 7, pp. 825, doi: 10.1007/s42452-025-07070-6.
- Pasic, R., Kuzmanov, I. and Atanasovski, K. (2021), 'ESP-NOW Communication Protocol with ESP32', Izzivi prihodnosti / Challenges of the Future, vol. 6, no. 1, pp. 53-60, doi: 10.37886/ip.2021.019.
- pySerial Developers (2025), pySerial Documentation. [Online]. pySerial. Available at: https://pyserial.readthedocs.io/.
- Python Software Foundation (2026), struct - Interpret Bytes as Packed Binary Data. [Online]. Python Software Foundation. Available at: https://docs.python.org/3/library/struct.html.
- STMicroelectronics (2015), LSM9DS1: iNEMO Inertial Module: 3D Accelerometer, 3D Gyroscope, 3D Magnetometer. [Online]. STMicroelectronics. Available at: https://www.st.com/resource/en/datasheet/lsm9ds1.pdf.
- Urazayev, D., Eduard, A., Ahsan, M. and Zorbas, D. (2023), 'Indoor Performance Evaluation of ESP-NOW', 2023 IEEE International Conference on Smart Information Systems and Technologies (SIST), IEEE, doi: 10.1109/SIST58284.2023.10223585.