Engineering Journal issn: 0125-8281

Article

Cost-Effective IIoT Gateway Development Using ESP32 for Industrial Applications

Paradon Boonmeeruk[a], Pichet Palrat[b], and Kiattisak Wongsopanakul[c,*]

Department of Electrical & Biomedical Engineering, Faculty of Engineering, Prince of Songkla University, Songkhla, Thailand
E-mail: [a]6510120011@email.psu.ac.th, [b]pichet6276@gmail.com, [c,*]kiattisak.w@psu.ac.th (Corresponding author)

Abstract. The Industrial Internet of Things (IIoT) connects industrial devices, such as measuring equipment and production line machines, to the Cloud system via the internet, creating a database for equipment data storage and performance analysis. Implementing an IIoT system requires an IIoT Gateway to interface with industrial controllers using protocols like Modbus TCP/IP or OPC UA, enabling data transmission to the Cloud. These Gateways, often produced by PLC manufacturers, are typically expensive. This research investigates using an ESP32 microcontroller as a cost-effective alternative to the Simatic IOT2050 IIoT Gateway. The study focuses on connecting the Siemens Simatic S7-1200 12144C AC/DC/RLY PLC via Modbus TCP/IP and facilitating data transmission between cloud systems using MQTT and REST API protocols. Results show that the IIoT Gateway's response time for writing 16-bit payload data to the PLC via Modbus TCP/IP averages 0.0591 seconds. Additionally, the device supports data scaling from 16-bit Integer to 32-bit Float for Modbus TCP/IP communication and converting 32-bit Float data to Message data for transmission via MQTT to ThingSpeak Cloud and REST APIs to Blynk Cloud. This approach offers a viable, cost-effective solution for IIoT implementations.

Keywords: ESP32 microcontroller, IIoT gateway, Modbus TCP/IP, MQTT and REST API protocols.

ENGINEERING JOURNAL Volume 28 Issue 10
Received 20 May 2024
Accepted 20 October 2024
Published 31 October 2024
Online at https://engj.org/
DOI:10.4186/ej.2024.28.10.93

DOI:10.4186/ej.2024.28.10.93

1. Introduction

The Industrial Internet, also known as the Industrial Internet of Things (IIoT), is a concept designed to connect industrial devices, such as measuring equipment or machines operating in production lines, to the Cloud system via the internet network. This integration facilitates the creation of a database to store data from equipment and analyze the operational efficiency of machines [1]. The IIoT concept has been applied across various domains [2]. For instance, in the agricultural sector, Y. Liu et al. [3] proposed the use of IIoT technology, referred to in their work as AIoT (Internet of Agricultural Things), to address food safety and quality issues in a large country like China. This technology enables the transfer of data from farmers' farms to the Cloud system. An overview of the system presented in their study is illustrated in (Fig. 1).

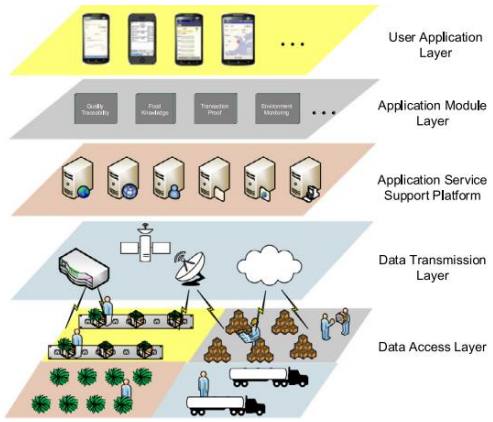

Fig. 1. An Overview of the Internet of Agricultural Things (AIoT) System.

The application of the AIoT system in the agricultural sector enables users to access high-quality information on farmers' agricultural products, facilitating informed purchasing and consumption decisions. Additionally, for environmental control in plantation areas, J. Muangprathub et al. [4] proposed applying IIoT technology to integrate data from measuring devices, such as temperature sensors, humidity sensors, and soil moisture sensors, into a web-based application system for predicting and controlling environmental conditions. This system ensures optimal conditions for plant growth, including appropriate soil moisture levels. Furthermore, it allows operators to monitor and manage the plantation environment remotely. The implementation of this system has demonstrated effective soil moisture control conducive to plant growth, cost reduction in cultivation, and increased productivity for farmers. An overview of the system used for environmental prediction and control is illustrated in (Fig. 2).

In the field of smart grid systems, W. Li et al. [5] proposed the application of IIoT technology combined with machine learning and statistical models to develop a Smart Energy Theft System (SETS) for detecting electricity theft. The system operates by collecting real-time data from power meters and inputting it into machine learning and statistical models to identify anomalies indicative of energy theft by users. Simulation results showed that the system achieved an accuracy of 99.96% in detecting electricity theft.

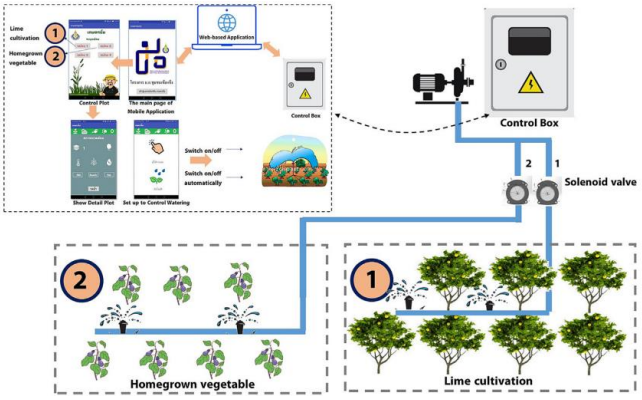

Fig. 2. Overview of Environmental Prediction and Control Systems.

O. Monnier [6] presents the application of IIoT technology to provide electricity users with access to information within the smart grid system. This enables users to monitor electricity consumption in buildings or homes and modify their usage behavior to achieve efficient energy use.

In the medical field, M. S. Hossain et al. [7] presented the use of IIoT technology in creating the HealthIIoT-Enabled Monitoring system. This system collects health data from users or patients, measured by an ECG sensor and sensors within a smartphone, and sends it to the Cloud system. An overview of the system is shown in (Fig. 3). This system allows doctors or health professionals to access patient health information stored on the Cloud system to analyze and track the patient's condition in a timely manner.

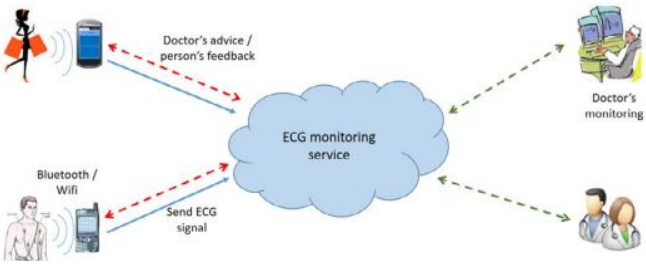

Fig. 3. Overview of HealthIIoT-Enabled Monitoring System.

Data from field-level devices is sent to the Cloud system according to the concept of the Industrial Internet of Things (IIoT) system, which requires an IIoT Gateway to connect to industrial controllers, such as programmable controllers and Programmable Logic Controllers (PLCs), through communication protocols like Modbus TCP/IP or OPC UA [8]. This connection allows measuring devices, actuators, and industrial controllers (PLCs) to exchange

data with Cloud systems, aligning with the IIoT system concept. As a result, various PLC manufacturers recognize the need for IIoT Gateways to connect industrial control equipment and support communication with external systems. For example, Mitsubishi has produced the PLC iQ-R IoT Gateway, which supports industrial communication protocols such as OPC UA and Modbus TCP/IP, and the basic communication protocol MQTT for sending data to the Cloud. Allen Bradley has developed Open Automation Software (OAS), allowing personal computers or small single-board computers (e.g., Raspberry Pi) to access data within Allen Bradley PLCs and send data to the Cloud via the internet. Siemens has produced the IIoT Gateway Simatic IOT2050, which supports industrial communication protocols such as OPC UA, Modbus TCP/IP, RTU, and the Siemens-specific S7 Protocol, as well as basic communication protocols like MQTT and REST API for Cloud data transmission. However, the high cost of this industrial equipment limits access for small industries.

This research article focuses on applying a microcontroller (ESP32) as a replacement for the Siemens Simatic IOT2050 IIoT gateway for connecting to industrial control devices, such as the Siemens Simatic S7-1200 12144C AC/DC/RLY PLC. The ESP32 utilizes the Modbus TCP/IP protocol for communication and enables simultaneous data transmission to the cloud system using multiple protocols, including MQTT and REST API.

2. Review of Previous Studies

Driven by cost constraints, previous research has explored using microcontrollers or small single-board computers (SBCs) to develop IIoT gateways for data exchange between field-level devices and cloud systems. For example, C.-H. Chen et al. [9] presented an IIoT gateway design utilizing a multi-MCU (multi-unit microcontroller) or FPGA (Field-Programmable Gate Array) for data transmission. Their work demonstrates the use of Modbus RS-485 and Modbus 2.4 GHz RF protocols for field device communication, along with TCP/IP for communication between the gateway and the cloud system. However, this approach can introduce latency between devices. Their machine-to-machine response time of 10.8 ms, while lower than the typical 30 ms for IIoT gateways, comes at the cost of a lower maximum data transmission speed (82.41 Mb/s) and higher power consumption (1.5 W).

C. Liu et al. [8] present a service-oriented plug-and-play (PnP) IIoT Gateway designed to transmit data from field-level manufacturing equipment to a Cloud-based time-series database (TSDB). This system utilizes the central processor of a Raspberry Pi 3 Model B in conjunction with RESTful APIs and Hypertext Transfer Protocol (HTTP) for data transmission between field-level devices. For transmitting data to the Cloud, the system employs the Line Protocol Format, a text-based communication protocol. The PnP IIoT Gateway demonstrates application flexibility by supporting industry-standard communication protocols and open-source APIs or control software, enabling seamless access to data from field-level devices. This approach has been successfully applied to monitor the operation of 3D printers and CNC machines.

R. M. Salem et al. [10] introduce a system designed for monitoring parameters in wastewater treatment processes. This system monitors temperature and pH values in wastewater to regulate valves, thereby preventing the discharge of untreated wastewater into the process and issuing alerts to operators in case of anomalies. The authors employ the NodeMcu ESP8266 as the central processor for sensor data acquisition, utilizing a DS18B20 temperature sensor and an ADS1115 acidity sensor. Additionally, the NodeMcu ESP8266 serves as a gateway for data transmission between the Cloud system and the sensors, facilitated by a wireless internet network and the RESTful APIs communication protocol. The system efficiently retrieves data from the wastewater treatment process and transmits it to a Web Server for display, with an upload time of approximately 5 seconds, enabling real-time process monitoring. Furthermore, the system allows operators remote control over valve operations for introducing wastewater into the process.

P. Nguyen-Hoang et al. [11] describe the implementation of a small single-board computer, Raspberry Pi, for the development of an Open-Source Industrial IoT Gateway. The device operates as an IIoT Gateway, leveraging the compact form factor of a single-board computer to establish connections with cloud systems using the REST API communication protocol for data transmission to AWS Cloud Services and the MQTT communication protocol for data transmission to IBM Watson Cloud Services. Additionally, the IIoT Gateway device facilitates connections with industrial devices through open communication protocols such as Modbus TCP/IP and Modbus RTU, as well as closed communication protocols like Siemens' proprietary S7 Protocol. Test results of the IIoT Gateway demonstrate an average transmission time of approximately 13.44 ms for data exchange with the PLC via the S7 Protocol communication call. Furthermore, data transmission using the Modbus communication protocol, employed for data reception from the Inverter, averages around 54.4 ms. Sending data to Cloud systems via REST APIs and MQTT communication protocols takes approximately 21.38 ms and 11.95 ms, respectively.

S. Nuratch et al. [12] detail the utilization of a 16-bit microcontroller in conjunction with a Wi-Fi Module to develop an IIoT Gateway, alternatively referred to as a Devices-to-Cloud Gateway in this article. The device operates with a 16-bit microcontroller, specifically the PIC24FJ48GA002 model, to interface with sensors and actuators deployed in industrial settings. Data transmission occurs via standard current electrical signals (4 – 20 mA) and standard voltage (0 – 12 V, 0 – 24 V). Furthermore, the device employs UART-to-RS485 communication to interface with the power meter, utilizing the Modbus-RTU communication protocol for data retrieval.

For transmitting data to the Cloud system, the device utilizes a Wi-Fi Module connected to the microcontroller via the UART protocol, enabling connectivity to a local wireless internet network for data transmission from the IIoT Gateway device to the cloud system.

Y. Zhang et al. [13] present an IIoT Gateway designed to manage data from industrial devices utilizing various communication protocols and convert them into the format compatible with MQTT communication, a standard protocol for cloud-based data upload. The IIoT Gateway operates through Concurrent Tasks, enhancing device reliability and enabling real-time observation. Additionally, it ensures data exchange security with the Cloud system through a Three-layer method. The central processing unit selected for this application is the Raspberry Pi 3B, connected to three Expansion Boards comprising the NB-IoT ME3136 expansion port, Clock Holding Unit module, and Power supply module. Performance testing involves the use of a Process Control Experimental kit for simulating detector operations. The results highlight the IIoT Gateway's capability to access data from various Filed Devices with different communication protocols and transmit them securely to the Cloud system in real-time, employing the three-layer method to ensure encrypted data transmission, thereby enhancing data exchange security.

A. Nugur et al. [14] describe the development of an IoT gateway utilizing Network Address Translation (NAT) devices and software for seamless data transmission to a cloud-based Building Energy Management (BEM) system, aimed at optimizing energy monitoring and management within buildings. The gateway integrates BACnet, Modbus, and HTTP RESTful communication protocols for data exchange among various devices, including power meters, plug loads, and lighting loads. The system was tested in a laboratory environment, where it was demonstrated that the response time between the 40 field devices and the cloud BEM system was consistently less than 12 ms, a response time deemed acceptable for energy monitoring and control applications in building automation systems.

B.-Y. Ooi et al. [15] introduce a Collaborative IoT gateway designed to manage IoT solution systems, including Sensor Node connections, Cloud system connections, Gateway connections, and cost management for data transmission between Cloud systems. The article outlines the use of a Raspberry Pi3 with an external Wi-Fi network interface for Gateway-to-Gateway commu-nication and a cellular 4G modem for Cloud system con-nectivity. The presented Gateway demonstrates a 75% reduction in data usage costs or cellular network expenses associated with data exchange between Cloud systems, with an error rate in data transmission of less than 1.5%.

Drawing insights from existing literature, case studies, and empirical investigations, it is apparent that in certain instances, experiments have utilized microcontroller-based devices to construct IIoT Gateways for data exchange between cloud systems, simultaneously facilitating data acquisition from measuring devices and operational control. However, such applications may not be suitable for microcontrollers with limited multi-instruction processing capabilities. Alternatively, some studies have employed small single-board computer setups for IIoT Gateway implementation. Nonetheless, this approach is constrained by equipment costs, akin to IIoT Gateways offered by PLC manufacturers, which often outperform small single-board computer devices. Hence, this research article leverages a low-cost microcontroller device, namely the ESP32, to mitigate the challenge of processing multiple instructions. The ESP32 serves as an intermediary for data exchange between the Cloud system and the PLC, functioning as a conduit for both data retrieval from sensors and operational control, thus underscoring the aim of this study.

3. Principles of Communication Protocols

3.1. Modbus TCP Protocol

Modbus TCP is an industrial communication protocol that operates on TCP/IP and utilizes a Master-Slave architecture. The protocol's framework comprises the Modbus Application Protocol (MBAP Header), a 7-byte header, and the Protocol Data Unit (PDU) for data transmission [16]. The data format of the PDU is depicted in (Fig. 4).

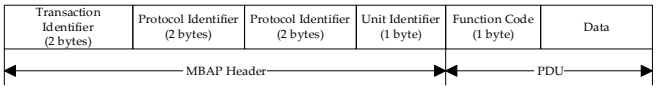

Fig. 4. Data Format of Modbus TCP/IP Communication Protocol.

As depicted in (Fig. 4), the Modbus TCP communication protocol follows a defined structure. It consists of two primary components: the Modbus Application Protocol (MBAP) Header and the Protocol Data Unit (PDU). The MBAP Header is further divided into four subfields. The second component, the PDU, is further divided into two subfields. According to the Modbus TCP/IP protocol, data access addresses can be specified within a range of 0 to 65535 [17, 18].

3.2. SPI Protocol (Serial Peripheral- Interface)

The SPI operates as a synchronous communication protocol, where data transmission between the Master and Slave is regulated by the Master's clock signal. Utilizing 8-bit Serial data, as depicted in (Fig. 5), the protocol's structure illustrates its functioning. When the Master intends to write data to the Slave, the Master's data position shifts (Shift Register), informing the Slave through an 8-Cycle Clock Pulses signal sent from the MOSI port. Conversely, if the Slave intends to relay data back to the Master, the process mirrors that of the Master, with data transmitted via the MISO port. Illustrated in (Fig. 6), the SPI communication protocol's connection ports include Slave Select (SS) for Slave selection, Clock

(SCLK) for generating clock signals, MOSI for writing data from the Master to the Slave, and MISO for the Slave to relay data back to the Master [19].

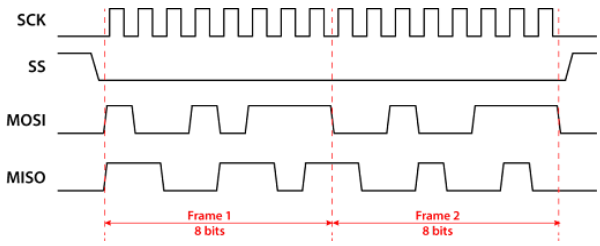

Fig. 5. Data Format of SPI Interface Communication Protocol.

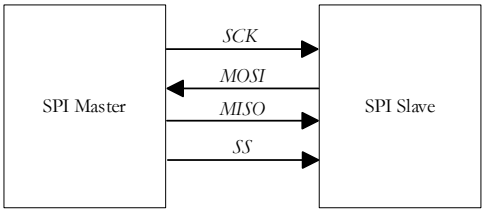

Fig. 6. Data Transmission between Master and Slave in SPI Interface Communication Protocol.

3.3. MQTT Protocol (Message Queue Telemetry Transport)

MQTT functions as a communication protocol facilitating data transmission between Server and Client through Publish and Subscribe messages, based on the TCP/IP communication protocol. Comprising three key components-Publisher or Producer, Broker, and Subscriber or Consumer-the protocol's structure is depicted in (Fig. 7). Operating on a Publish-Subscribe model, the exchange format relies on Topics to access data locations, with Subscribers referencing data locations on devices or servers and Publishers pinpointing data locations within devices or servers. The Broker or MQTT Server acts as an intermediary, managing Topics between Publishers and Subscribers. Operating as a bi-directional communication protocol, data transmission size comprises a 2-byte header and a payload size of up to 256 MB [20].

3.4. REST API (Representation State Transfer)

REST APIs function as a communication format that enables clients to access and manage server data using the HTTP (Hypertext Transfer Protocol). The architecture supports four primary operations: GET for retrieving data, POST for creating new data sets on the server, PUT for updating existing data sets, and DELETE for removing data. Clients access specific data by referencing corresponding URLs. The communication format and operational structure of REST APIs are depicted in Fig. 8 [21].

3.5. IEEE 802.11n Standard

IEEE 802.11n stands out as a standard in WiFi Wireless Technology, facilitating data transmission via frequencies of 2.4 GHz and 5.0 GHz, achieving a maximum data transmission speed of 150 Mbit/s, and boasting a device response time of 400 ns. This response time surpasses that of older standards such as IEEE 802.11a and g, which typically have a response time of around 800 ns. Moreover, in terms of bandwidth, data transmission adhering to the IEEE 802.11n standard utilizes a bandwidth of 40 MHz to support Multi-input Multi-output (MIMO) capabilities. This represents a bandwidth enhancement from the 20 MHz bandwidth employed in older standards to 40 MHz as per the IEEE 802.11n standard, thereby facilitating data exchange rates at the physical layer level of up to 600 Mbps [22, 23].

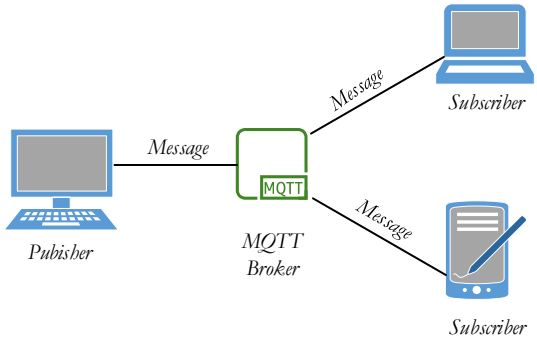

Fig. 7. Structural Format of the MQTT Communication Protocol.

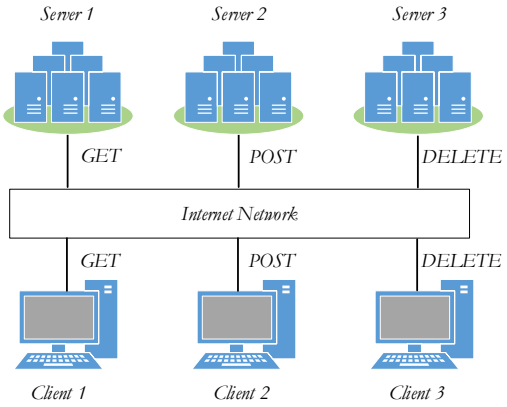

Fig. 8. Data Exchange between Server and Client Utilizing REST API Communication.

4. Design and Fabrication of IIoT Gateway

4.1. Designing Programs for IIoT Gateway

This section outlines the operation of the program downloaded to the ESP32 microcontroller, which functions as an IIoT Gateway. It facilitates data exchange between the Siemens Simatic S7-1200 12144C

AC/DC/RLY PLC via the Modbus TCP/IP communication protocol and the Cloud system using MQTT and REST APIs. The program begins by defining the necessary variables for data storage and specifying data locations between the Server (IIoT Gateway) and Client (PLC) to ensure proper data read/write operations, adhering to the Modbus TCP/IP protocol standards. Once the values and data locations are defined, the program instructs the ESP32 microcontroller to establish a connection to the wireless Internet using the IEEE 802.11n communication standard. This connection enables the device to access the Internet for uploading and downloading data between the PLC and the Cloud system. Upon successful connection to the wireless network, the program directs the ESP32 to establish communication with the PLC and retrieve data from the pre-defined data locations. The retrieved data is then stored in the specified variables within the ESP32 microcontroller. Since data transmission between the PLC and the ESP32 uses the Modbus TCP/IP communication protocol, the transmitted data follows a structured format, as illustrated in (Fig. 9) This structure includes a 16-bit Integer data payload. The process variable data sent to the PLC from the field devices is scaled using the equation Eq. (1). The scaled data is then transferred to a 16-bit Integer variable using the PLC's Move function. This variable serves as the designated location for accessing and exchanging data between the devices using the Modbus TCP/IP protocol.

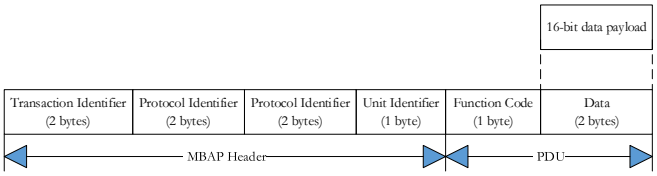

Fig. 9. Data structure for reading and writing data between devices (IIoT Gateway and PLC) using the Modbus TCP/IP communication protocol.

$$Datapayload = PV \times 100 \tag{1}$$

From the equation, the Process Variable (PV) represents the data received by the PLC from the field device. The Data Payload, a 16-bit integer variable, is then transmitted to the IIoT Gateway via the Modbus TCP/IP communication protocol.

To convert data sent to the IIoT Gateway into 32-bit Float type with two decimal places, the process begins by receiving 16-bit Integer data from the PLC. The next step involves converting this data into 32-bit Float using a program method that scales the read data, as outlined in Eq. (2). Once the conversion is complete, the resulting data is stored in a predefined 32-bit Float variable. To transmit this data to the Cloud system via the MQTT communication protocol and REST APIs, it must first be converted from 32-bit Float to text format using the "Convert Floating-point number to character array" function. The conversion process involves splitting the floating-point number into two parts: the Integer part and the Fractional part. For the Integer part, the data is divided by the base number to determine the corresponding ASCII value, following the example in Eq. (3), which is used for IIoT Gateway data in decimal form. The same method is applied to the Fractional part, but with an additional step: multiplying the fractional data by the base number raised to the power of the number of decimal places, as described in Eq. (7), to convert it into an integer. The values are then recalculated using Eq. (3). After converting both the Integer and Fractional parts into ASCII characters, the final step involves combining these parts to create a new text-formatted data set. This process is summarized in the illustration (Fig. 10), which provides an overview of the "Convert Floating-point number to character array" function.

$$32bit(Floating-point\ Number) = \frac{Datapayload}{100} \tag{2}$$

The equation 32-bit (Floating-point Number) signifies 32-bit Float data. This data is obtained by taking the Data Payload received from the PLC and applying the necessary scaling process.

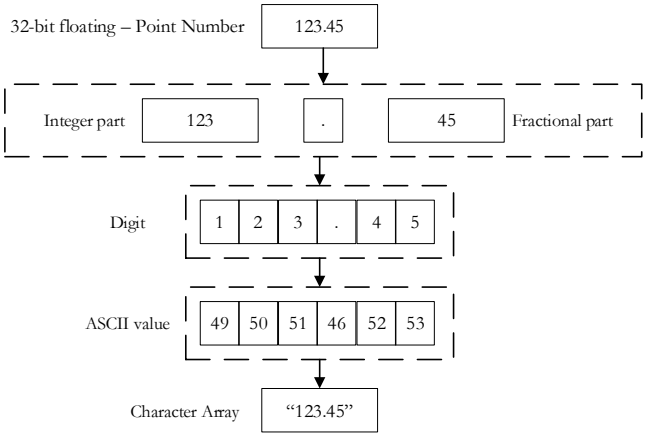

Fig. 10. The diagram illustrates the process of converting a 32-bit Float data type into a text data type.

$$Digits = \frac{Integer}{BaseNumber} \tag{3}$$

$$12.3 = \frac{123}{10}; Digit3 = 3 \tag{4}$$

$$1.2 = \frac{12}{10}; Digit2 = 2 \tag{5}$$

$$0.1 = \frac{1}{10}; Digit1 = 1 \tag{6}$$

As shown in Eq. (4)–(6), the process of finding the digits for each position is performed by dividing the value by the base number. The calculation continues until the result of the division equals zero, at which point the process is considered complete.

$$Integer = Fractional \times BaseNumber^{\Pr ecision} \quad (7)$$

$$45 = 0.45 \times 10^{2} \quad (8)$$

Equation Eq. (8) provides an example of the calculation process used to convert a fractional number to an integer, which is necessary for determining the position of the number.

After completing the data type conversion process, the next step involves uploading the process variable data, now in text format, to the ThingSpeak Cloud system via the MQTT communication protocol. The message format used for data upload is illustrated in (Fig. 11). This message structure consists of two key components: the Topic, which identifies the data location on the Cloud system, and the Data Payload, which contains the data to be uploaded. Once the data is successfully uploaded to ThingSpeak Cloud, the ESP32 microcontroller is instructed to interact with the Blynk Cloud system via REST APIs for both data upload and download operations. For data upload to the Blynk Cloud system, the message format is depicted in (Fig. 12). The message structure is divided into three main sections: the first specifies the Cloud system's access point, the second determines the location for uploading data, and the third is the data payload itself. Upon successful upload, the program initiates the data download process from the Cloud system. The download method follows similar principles to the upload, but with a different message structure. As shown in (Fig. 13), the message used for downloading consists of two parts: the first specifies the Cloud access location, and the second identifies the location of the data to be retrieved.

Once the ESP32 Microcontroller receives data from the Blynk Cloud system, the final step in the program involves converting the data from text format to a 32-bit float and scaling it to a 16-bit integer format before sending it back to the PLC via the Modbus TCP/IP communication protocol. In this step, the data payload downloaded from the Blynk Cloud system is split into two parts: Integer and Fractional, which are separated into digits in character array format. After this separation, the digits are converted to decimal numbers. The Integer part is processed using Eq. (9), which calculates each digit position and replaces the ASCII value to convert it into decimal data. The digits are then combined using Eq. (10). For the Fractional part, Eq. (15) calculates the digits and Eq. (16) combines them. Once the Integer and Fractional parts are converted, the next step is to merge them into a 32-bit float. This process is illustrated in Fig. G. After the data has been successfully converted into a 32-bit float, the final step involves scaling it to a 16-bit integer using Eq. (11). This allows the data to be sent back to the PLC via the Modbus TCP/IP communication protocol. The complete operation of the IIoT Gateway program is depicted in the diagram shown in (Fig. 16).

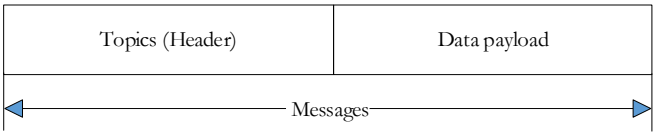

Fig. 11. The figure illustrates the message format used for uploading data to the ThingSpeak Cloud via the MQTT communication protocol.

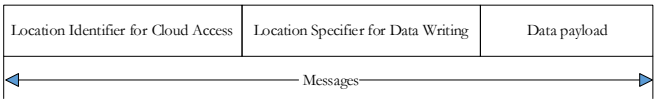

Fig. 12. The figure illustrates the message format used for uploading data to the Blynk Cloud via the REST APIs communication protocol.

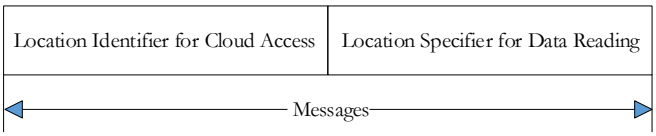

Fig. 13. The figure illustrates the message format used for downloading data from the Blynk Cloud system via the REST APIs communication protocol.

$$Integer = Digit \times BaseNumber^{N} \quad (9)$$

$$Integerpart = Integer_1 + Integer_2 + Integer_3 + Integer_n \quad (10)$$

$$900 = 9 \times 10^{2} \quad (11)$$

$$80 = 8 \times 10^{1} \quad (12)$$

$$7 = 7 \times 10^{0} \quad (13)$$

$$Integerpart = 900 + 80 + 7 = 987 \quad (14)$$

As outlined in Eq. (9), each position of the Integer is calculated as the product of the digit and the base number raised to the power of the digit's position. Therefore, the method for calculating each digit's position, as well as the sum of the digits in the Integer part, follows the process described in Eq. (11)-(14).

$$Fraction = \frac{Digit}{BaseNumber^{N}} \quad (15)$$

$$Fractionpart = Fraction_1 + Fraction_2 + Fraction_n \quad (16)$$

$$0.6 = \frac{6}{10^{1}} \quad (17)$$

$$0.05 = \frac{5}{10^{2}} \quad (18)$$

$$Fractionpart = 0.6 + 0.05 = 0.65 \quad (19)$$

As shown in Eq. (15), the method for calculating the fraction at each position follows the equation where the fraction at any position is equal to the digit divided by the base number raised to the power of N, with N starting

from 1. Therefore, the example calculation, including the fraction part, is presented in Eq. (17)-(19).

$$Serverpayload(Integer) = Serverpayload(float) \times 100 \quad (20)$$

As shown in Eq. (20), the calculation method scales data from a 32-bit floating-point format to a 16-bit integer format for transmission back to the PLC via the Modbus TCP/IP communication protocol. This is done by setting the Serverpayload (16-bit Integer) equal to the Serverpayload (32-bit Float) multiplied by 100, ensuring that the payload sent back to the PLC represents a real number with two decimal places.

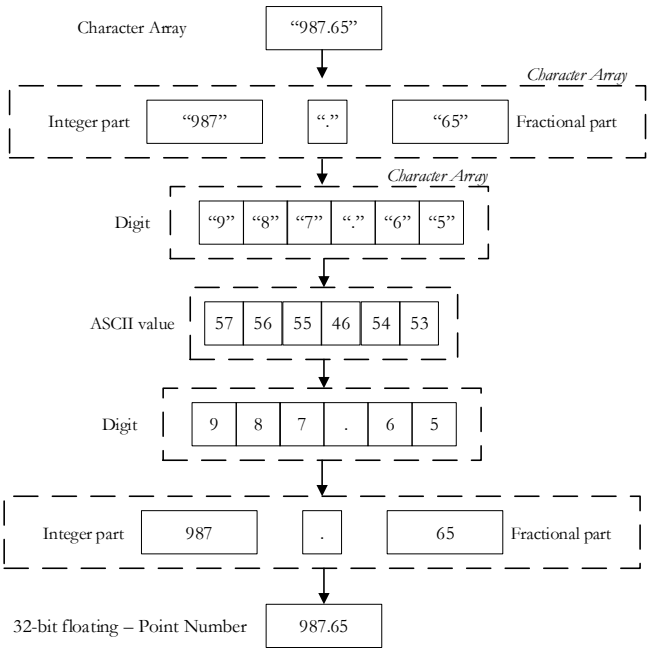

Fig. 14. The diagram illustrates the process of converting data from text format to 32-bit float format.

4.2. Design and Assembly Procedures for Constructing an IIoT Gateway

The illustration (Fig. 17) displays the device connection for constructing an IIoT Gateway, comprising equipment such as the LM2596S DC-DC Step-Down module, utilized to reduce the DC voltage from 24 VDC to 5 VDC for powering various devices. For establishing connectivity between the IIoT Gateway and PLC via the Modbus TCP/IP communication protocol, the Internet Module ENC28J60 facilitates the connection to the PLC, incorporating functionalities for interfacing with a microcontroller (ESP32) to enable data exchange between the ESP32 (Server) and PLC (Client) through the SPI Interface communication protocol. The IIoT Gateway's central processing unit relies on the ESP32 microcontroller, serving as the central processing unit and facilitating connection to a local wireless internet network compliant with the IEEE 802.11n standard, enabling data transmission and reception between Cloud systems via MQTT and REST API communication protocols upon integrating all modules, depicted in the illustration (Fig. 17). Consequently, the IIoT Gateway depicted in this research article is illustrated in (Fig. 18), showcasing the connectivity of each module within the IIoT Gateway, while (Fig. 19) presents the external structure of the IIoT Gateway outlined in this study.

5. Methodology of Testing

5.1. Testing Methodology for Response Time Evaluation of IIoT Gateway

In the testing process for evaluating the Response Time of the IIoT Gateway (Server), the overall test setup is shown in the illustration (Fig. 20), which includes three components. First, the IIoT Gateway (Server); second, the Siemens Simatic S7-1200 12144C AC/DC/RLY PLC, acting as a client to retrieve data and wait for the IIoT Gateway to write data back. The data format used for testing, depicted in (Fig. 9), follows the Modbus TCP communication protocol with a 16-bit (Word) data payload. The third component is a computer with the TIA Portal IDE software installed, used to adjust the data retrieval frequency of the client to assess the impact of different request frequencies from the server. Three frequencies were tested: 10Hz, 2Hz, and 1Hz. This setup also evaluates the time taken by the IIoT Gateway to send data back to the client. Initially, the client sends a data request as shown in (Fig. 9) to the IIoT Gateway. Upon receiving the request, the IIoT Gateway returns the 16-bit data payload to the client, allowing for Response Time evaluation. The TIA Portal software plots a graph comparing the client's data retrieval time with the time it takes for the IIoT Gateway to return the data. The Response Time is calculated by determining the difference between these two time points, as described in Eq. (21)-(22). This process is repeated five times at each frequency. The testing diagram for evaluating the Response Time of the IIoT Gateway is shown in (Fig. 15).

$$responseTime = \Delta T \quad (21)$$

$$\Delta T = t_{receive} - t_{request} \quad (22)$$

As per Eq. (21) – (22), the Response Time is calculated as the difference in time ΔT, where $t_{request}$ represents the time when the Client initiates data retrieval from the

Table 1. The Data Addresses on Server and Client Devices.

Data Name	Data Type	Data Address (Server)	Mode	Data Address (Client)
Water Level	Int 16-bit	0	Read/Write	40001
Flow Rate	Int 16-bit	1	Read/Write	40002
Volume	Int 16-bit	2	Read/Write	40003
Setpoint	Int 16-bit	3	Read/Write	40004
Drain Valve	Int 16-bit	4	Read/Write	40005

Server, and $t_{receive}$ represents the time when the Server completes writing data back to the Client. The calculation begins from the moment the Client starts retrieving data, and the Response Time is determined by subtracting the data retrieval initiation time from the data writing completion time.

5.2. Testing Data Flow between Cloud Systems via MQTT Communication Protocol and REST APIs

This experimental procedure delineates a method for testing data uploading to the ThingSpeak Cloud system using the MQTT communication protocol, along with testing data transmission between controllers using the Blynk Cloud system as a medium for data exchange via the REST APIs communication protocol. An overview of the experiment is depicted in the illustration (Fig. 21 and Fig. 22). The experiment methodology relies on the Factory IO simulation program to simulate the operation of Field Devices, such as water level sensors or flow control valves. These devices connect to the PLC, set as the Client, and to the IIoT Gateway, set as the Server, via Ethernet Port for reading and writing data between devices using the Modbus TCP/IP protocol. The server and client devices utilized in the experiment are outlined in (Table 1). For control, the Node-Red program is employed to create three sections within the control segment. The first section displays the water level height and is utilized to set the Setpoint to control the height of the water level inside the tank. The second section controls and displays the flow rate of water out of the tank. Lastly, a section shows the amount of water inside the tank, as illustrated in (Fig. 23). Additionally, all three control sections facilitate data transmission between Field Devices using the REST APIs communication protocol, leveraging the Blynk Cloud system for data exchange.

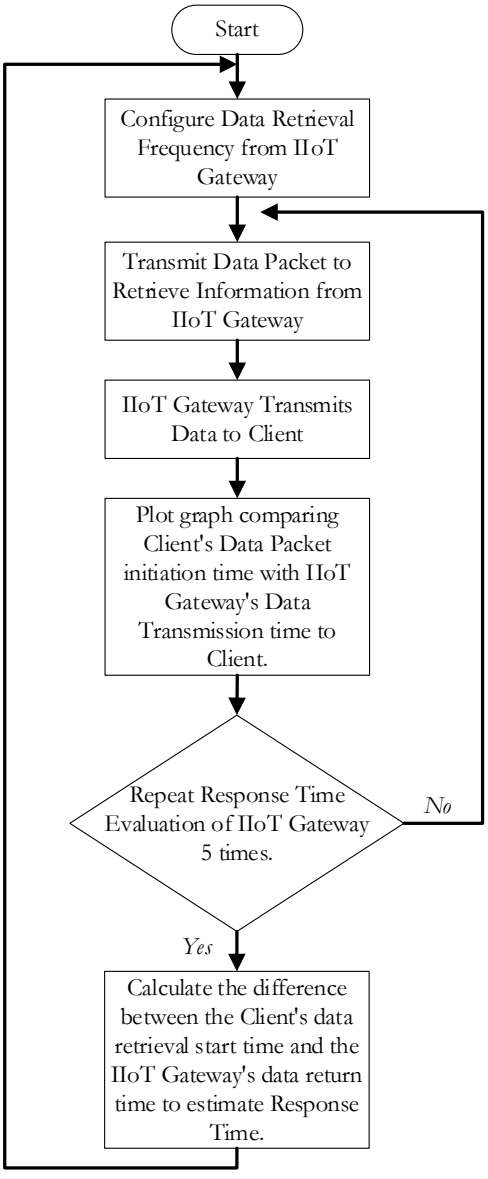

Fig. 15. The diagram illustrates the procedure for testing and evaluating the Response Time of the IIoT Gateway.

DOI:10.4186/ej.2024.28.10.93

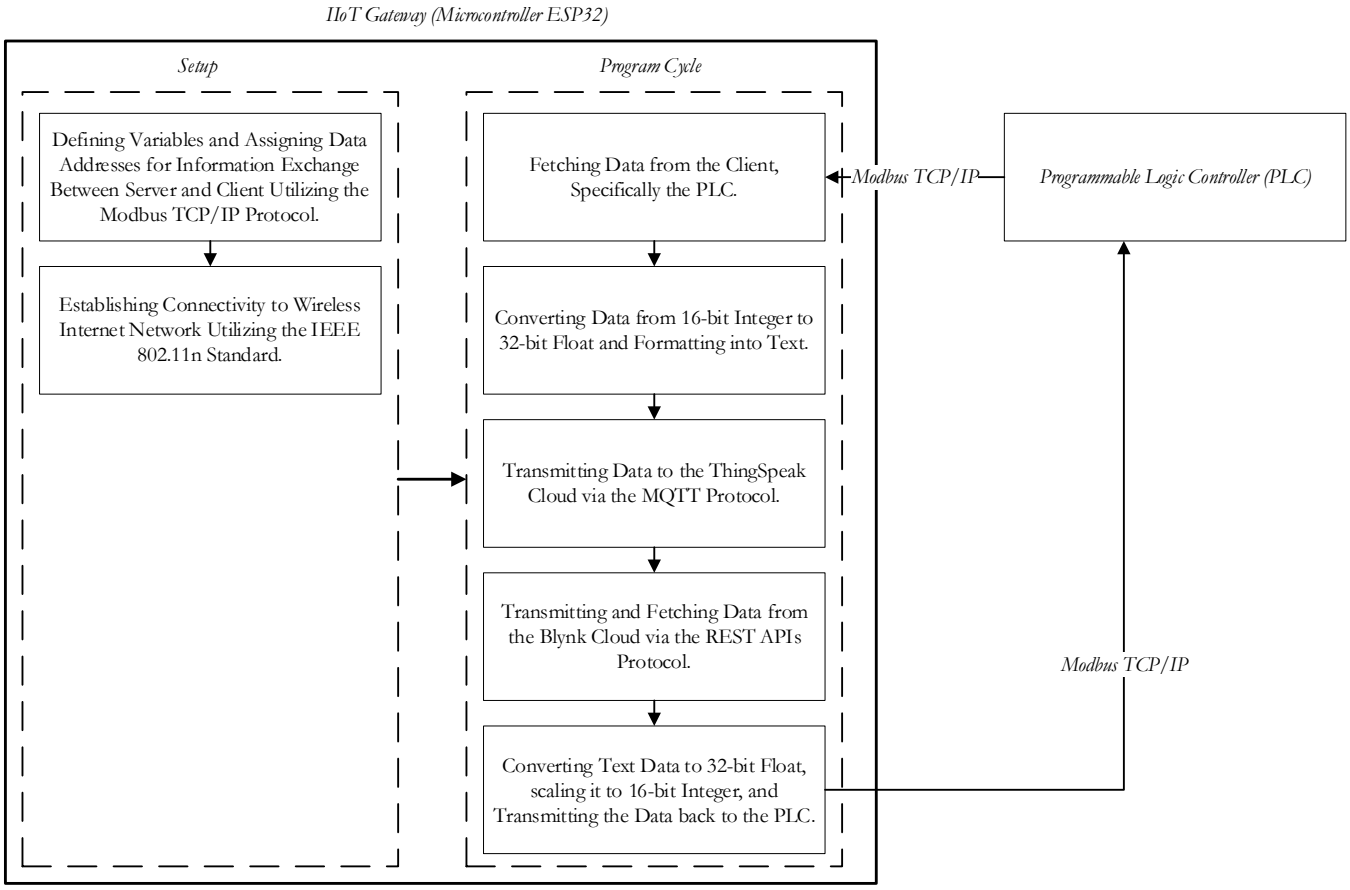

Fig. 16. Diagram Illustrating the Operation of the IoT Gateway Program.

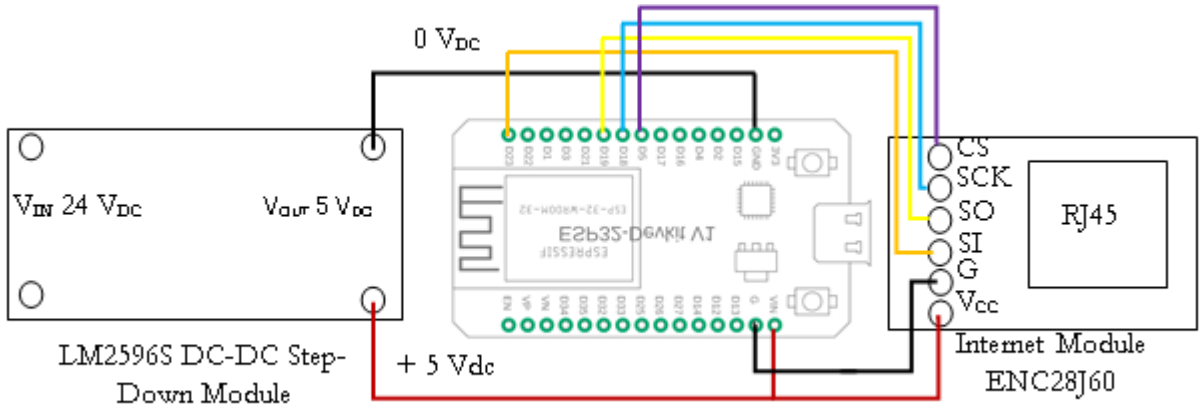

Fig. 17. The diagram depicts the interconnected modules within the IIoT Gateway.

Fig. 18. The Components of the IIoT Gateway.

Fig. 19. The external architecture of the IIoT Gateway.

Table 2. IIoT Gateway Response Time at Different Request Frequencies.

Request Frequency	Response Time (sec.)					Average Response Time (sec.)
	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	
10 Hz	0.0773	0.0363	0.0545	0.0818	0.0591	0.0618
2 Hz	0.0626	0.0582	0.0447	0.0402	0.0447	0.0501
1 Hz	0.0629	0.0584	0.0629	0.0674	0.0761	0.0655

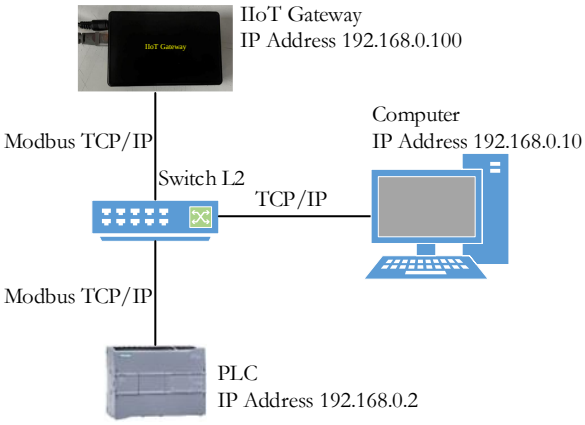

Fig. 20. Diagram Illustrating Response Time Testing of the IoT Gateway.

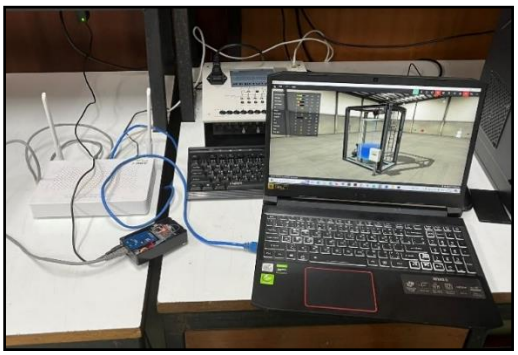

Fig. 21. Overview of Test Preparation for the IIoT Gateway.

6. Results of the Tests

6.1. Response Time Evaluation Results of the IIoT Gateway

As shown in (Fig. 24 - 26), the graph illustrates the data retrieval process between the Client (PLC) and the Server (IIoT Gateway). The blue line represents the data sent back to the Client, while the red line indicates the data request initiated by the Client. In the graph, the X-axis represents time, and the Y-axis represents the data transmitted from the Server to the Client. The test data

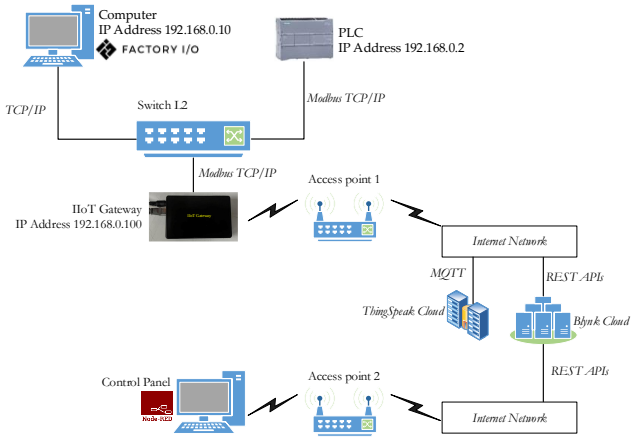

Fig. 22. Overview Diagram of Testing Traffic between the IIoT Gateway and Cloud via MQTT Communication Protocol and REST APIs.

follows the standard Modbus TCP/IP communication protocol with a 16-bit payload, as depicted in (Fig. 9). The test results are summarized in (Table 2). For a data request frequency of 10 Hz, the IIoT Gateway took an average of approximately 0.0618 seconds to respond. For frequencies of 2 Hz and 1 Hz, the average response times were approximately 0.0501 seconds and 0.0655 seconds, respectively. It is observed that changing the data request frequency has a minimal effect on the IIoT Gateway's response time across different frequencies. Thus, the IIoT Gateway developed using the ESP32 microcontroller demonstrates an average response time of approximately 0.0591 seconds when communicating via the Modbus TCP/IP protocol with a 16-bit payload.

DOI:10.4186/ej.2024.28.10.93

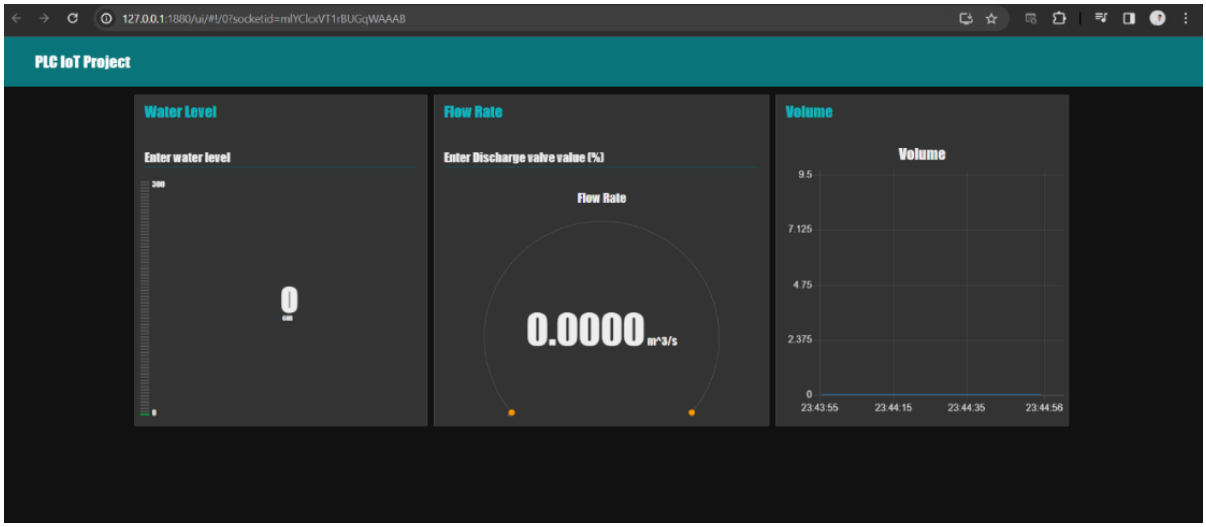

Fig. 23. Control Panel Interface in the Node-Red Program.

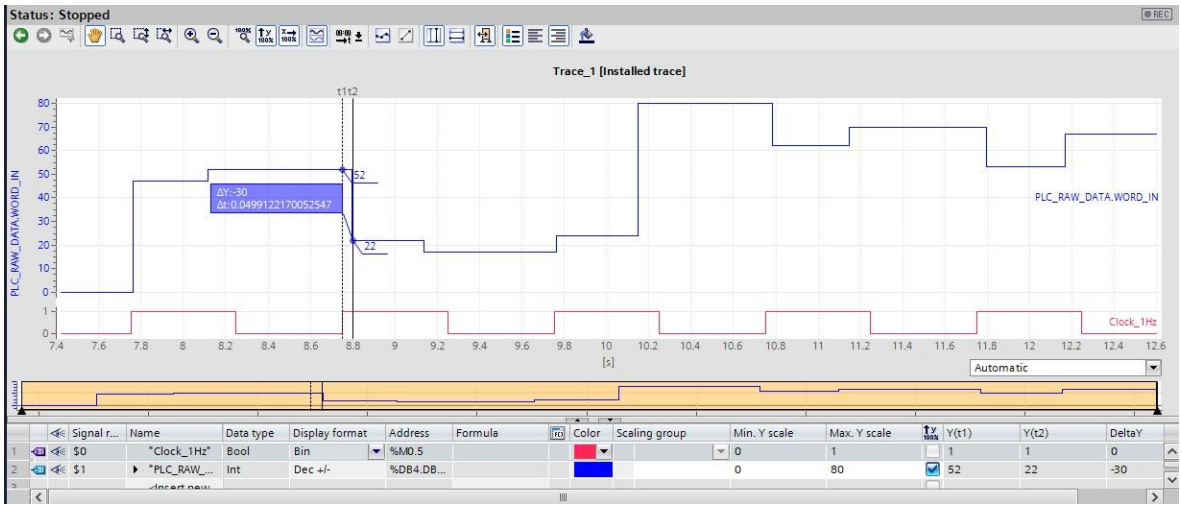

Fig. 24. Graph illustrating IIoT Gateway Response Time at 1 Hz Request Frequency.

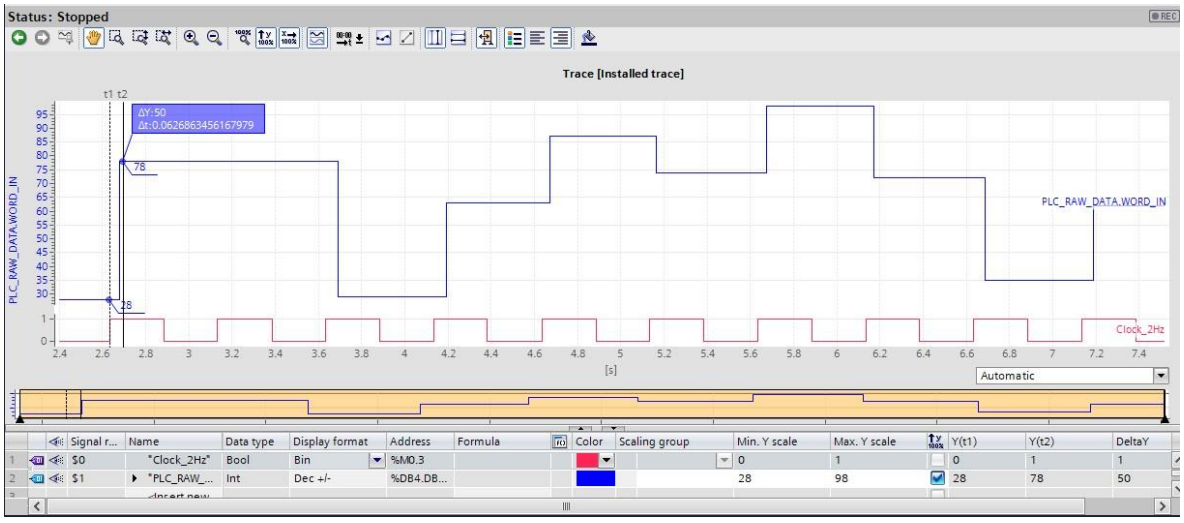

Fig. 25. Graph illustrating IIoT Gateway Response Time at 2 Hz Request Frequency.

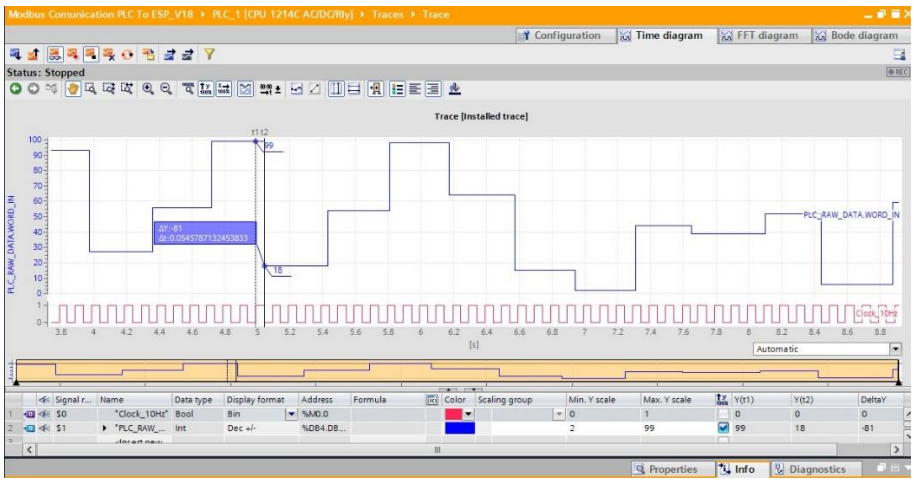

Fig. 26. Graph illustrating IIoT Gateway Response Time at 10 Hz Request Frequency.

6.2. Data Transmission Test Results Between Cloud Systems Using MQTT Protocol and REST APIs

The test results are segmented into two components: one encompasses the data upload to ThingSpeak Cloud, while the other pertains to data transmission between the Control Panel and the IIoT Gateway, employing Blynk Cloud as an intermediary for data exchange. The test data is categorized into two groups, each serving distinct purposes on the control panel. One group mirrors the dataset sent to ThingSpeak Cloud, encompassing parameters such as water level, flow rate, and water volume. These data, of 32-bit Float type, are derived by scaling the 16-bit Integer data obtained from the PLC during the transmission test between the Control Panel and the IIoT Gateway, utilizing Blynk Cloud alongside the REST API communication protocol. The second group of data comprises Setpoint and Drain Value, which are utilized for process control. The illustration (Fig. 27) depicts the upload of PLC data to the designated storage area on ThingSpeak Cloud through the MQTT communication protocol. Concurrently, the test results for data transmission between the control panel and IIoT Gateway, employing the REST API communication protocol with Blynk Cloud, are presented in the illustrations (Fig. 28 - 29). Regarding the data transmitted back to the PLC for operational control subsequent to the IIoT Gateway's data reception, it assumes the format of a sequence of messages adhering to the REST API protocol standards. The device undertakes the task of formatting this data, converting it into a 32-bit Float format, and subsequently scaling it as16-bit Integers before dispatching it back to the PLC via the Modbus TCP/IP communication protocol. Upon receipt, the PLC further scales the data to 32-bit REAL format to facilitate additional process control based on the received data from the IIoT Gateway. This process, including the data scaling procedure, is elucidated in the illustration (Fig. 30).

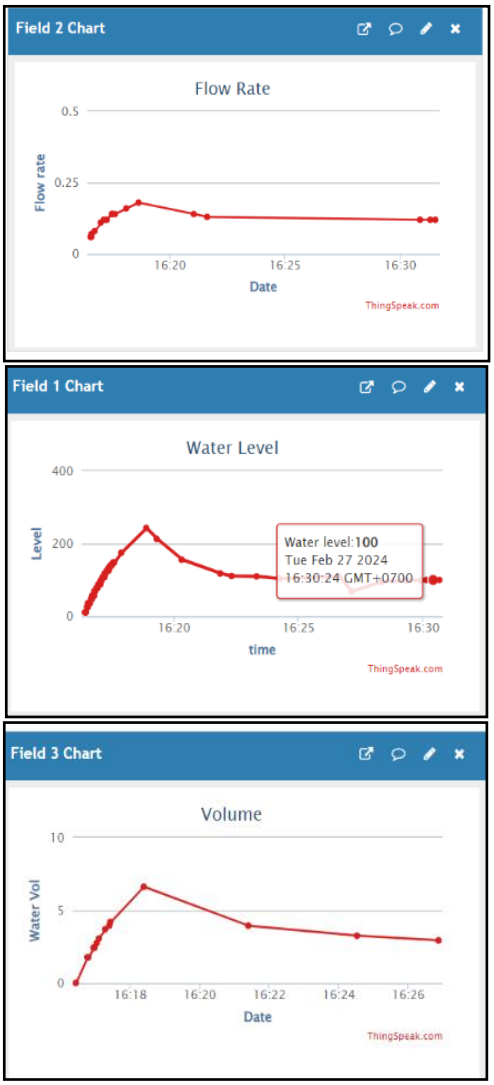

Fig. 27. Test Results Illustrating the Transmission of 32-Bit Float Data to the ThingSpeak Cloud System via the MQTT Protocol.

DOI:10.4186/ej.2024.28.10.93

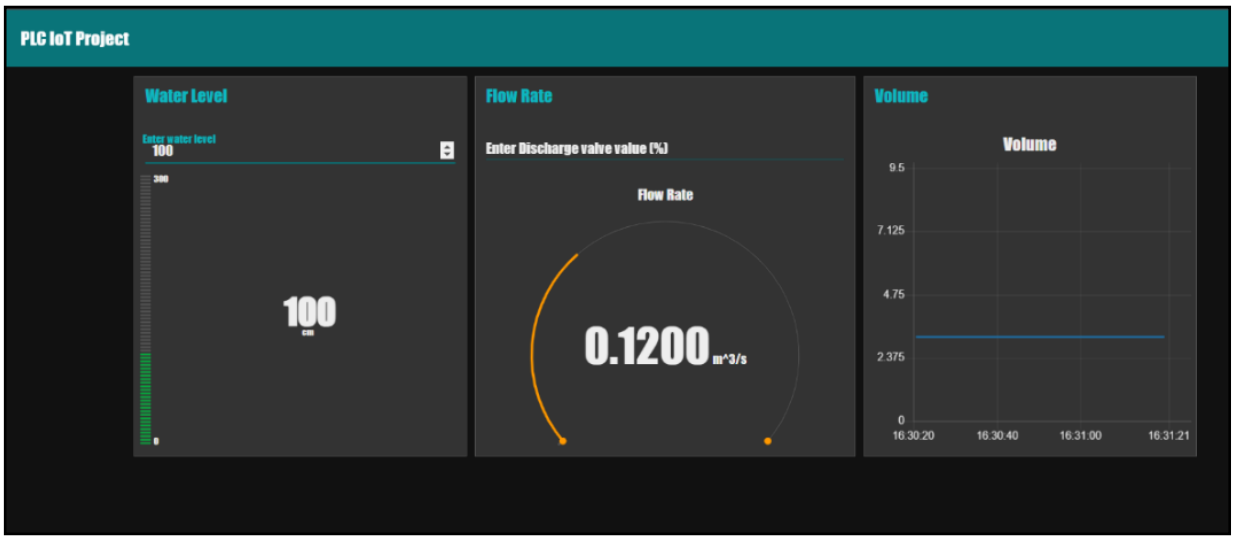

Fig. 28. Visualization of Field Devices data, encompassing measurement and process control data, within the controls on the Node-Red program.

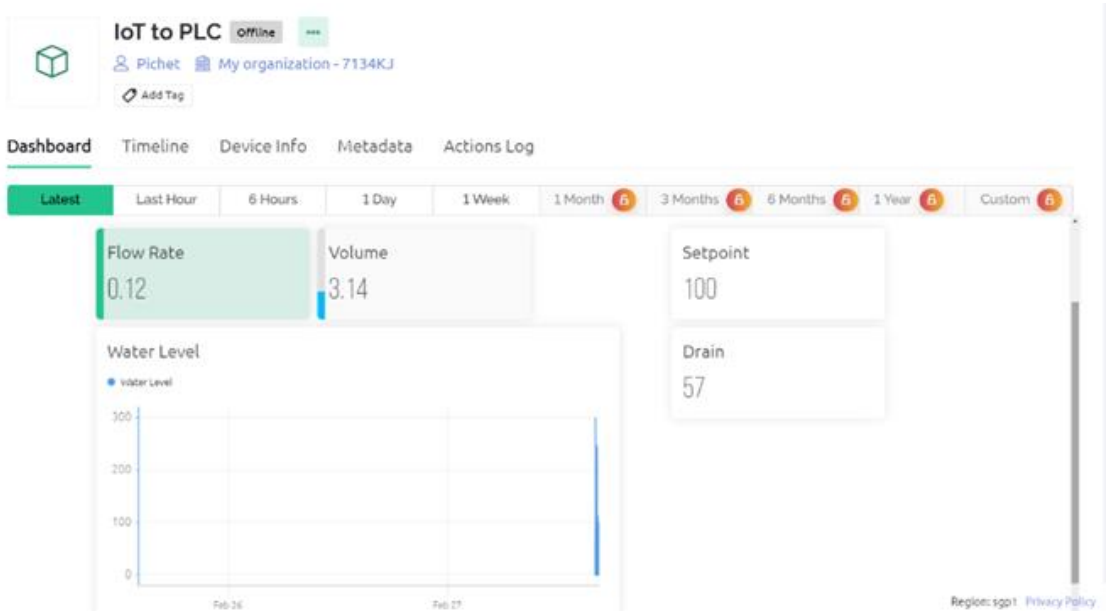

Fig. 29. Transmission of data to the Blynk Cloud system utilizing the REST API communication protocol.

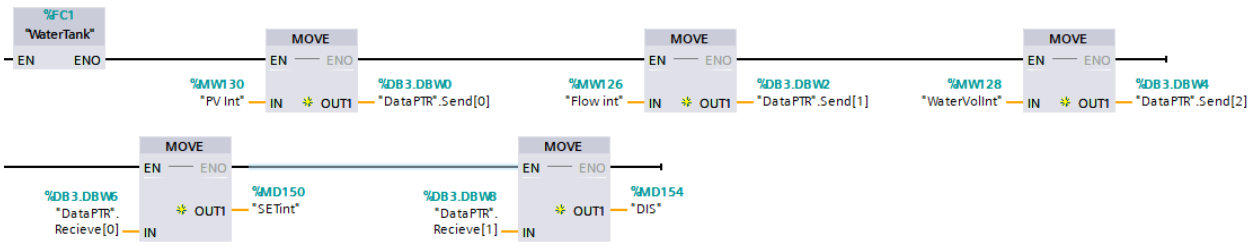

Fig. 30. Implementation of PLC program: Converting 16-bit Integer data from IIoT Gateway to 32-bit REAL format via Modbus TCP/IP protocol.

7. Conclusion

According to the objective of this research, the aim was to develop an Industrial IIoT Gateway based on the ESP32 microcontroller as a low-cost alternative to the high-cost Industrial IIoT Gateway (SIMATIC IOT2050). This article presents a method of designing a C language-based program to create an ESP32 microcontroller control system that functions as an IIoT Gateway, facilitating data exchange between field devices (PLC Siemens Simatic S7-1200 12144C AC/DC/RLY) via the Modbus TCP/IP protocol and cloud systems using MQTT and REST APIs. The article also describes the development of a prototype IIoT Gateway and evaluates its performance in terms of response time for data exchange between the IIoT Gateway and the PLC, as well as its capability for simultaneous data exchange with the cloud system via MQTT and REST APIs. The response time tests indicated that changes in client data retrieval frequency had minimal impact on the IIoT Gateway's response time. The device demonstrated an average response time of approximately 0.0591 seconds for exchanging 16-bit payload data via the Modbus TCP/IP protocol. The IIoT Gateway successfully supported data type conversions, such as converting 16-bit integer data to 32-bit float and then to text format for uploading to ThingSpeak and Blynk cloud systems via MQTT and REST APIs. Furthermore, the device could download data from the Blynk Cloud system, convert text data back to 32-bit float, and scale it to 16-bit integer format for communication with the PLC via Modbus TCP/IP. In terms of cost, the ESP32-based IIoT Gateway provided a cost reduction of approximately 66.69 times compared to the SIMATIC IOT2050 (based on 2024 online prices). However, the ESP32-based solution has limitations in the number of communication protocols it supports, offering only Modbus TCP/IP for field devices and MQTT/REST APIs for cloud systems. Additionally, the ESP32 microcontroller, with its Xtensa® dual-core 32-bit LX6 processor at 240 MHz, lacks the processing power of the ARM TI AM6528 GP used in the SIMATIC IOT2050, which operates at 1.1 GHz. To improve the system's efficiency, future developments should focus on expanding protocol support, such as incorporating OPC UA for enhanced scalability and real-time data access. Additionally, exploring microcontrollers with capabilities closer to the SIMATIC IOT2050 but with lower costs could further enhance the system's industrial applicability.

References

[1] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (IIoT): An analysis framework," *Computers in Industry*, vol. 101, pp. 1-12, 2018.

[2] P. K. Malik et al., "Industrial Internet of Things and its applications in industry 4.0: State of the art," *Computer Communications*, vol. 166, pp. 125-139, 2021.

[3] Y. Liu, W. Han, Y. Zhang, L. Li, J. Wang, and L. Zheng, "An Internet-of-Things solution for food safety and quality control: A pilot project in China," *Journal of Industrial Information Integration*, vol. 3, pp. 1-7, 2016.

[4] J. Muangprathub, N. Boonnam, S. Kajornkasirat, N. Lekbangpong, A. Wanichsombat, and P. Nillaor, "IoT and agriculture data analysis for smart farm," *Computers and Electronics in Agriculture*, vol. 156, pp. 467-474, 2019.

[5] W. Li, T. Logenthiran, V.-T. Phan, and W. L. Woo, "A novel smart energy theft system (SETS) for IoT-based smart home," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5531-5539, 2019.

[6] O. Monnier, "A smarter grid with the Internet of Things," Texas Instruments, pp. 1-11, 2013.

[7] M. S. Hossain and G. Muhammad, "Cloud-assisted Industrial Internet of Things (IIoT)–Enabled framework for health monitoring," *Computer Networks*, vol. 101, pp. 192-202, 2016.

[8] C. Liu, Z. Su, X. Xu, and Y. Lu, "Service-oriented industrial internet of things gateway for cloud manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102217, 2022.

[9] C.-H. Chen, M.-Y. Lin, and C.-C. Liu, "Edge computing gateway of the industrial internet of things using multiple collaborative microcontrollers," *IEEE Network*, vol. 32, no. 1, pp. 24-32, 2018.

[10] R. M. Salem, M. S. Saraya, and A. M. Ali-Eldin, "An industrial cloud-based IoT System for real-time monitoring and controlling of wastewater," IEEE Access, vol. 10, pp. 6528-6540, 2022.

[11] P. Nguyen-Hoang and P. Vo-Tan, "Development an open-source industrial IoT gateway," in *19th International Symposium on Communications and Information Technologies (ISCIT), 2019*, IEEE, pp. 201-204.

[12] S. Nuratch, "The IIoT devices to cloud gateway design and implementation based on microcontroller for real-time monitoring and control in automation systems," in *12th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2017*, IEEE, pp. 919-923.

[13] Y. Zhang, W. Sun, and Y. Shi, "Architecture and Implementation of Industrial Internet of Things (IIoT) Gateway," in *2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT, 2020), IEEE*, pp. 114-120.

[14] A. Nugur, M. Pipattanasomporn, M. Kuzlu, and S. Rahman, "Design and development of an IoT gateway for smart building applications," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9020-9029, 2019.

[15] B.-Y. Ooi, Z.-W. Kong, W.-K. Lee, S.-Y. Liew, and S. Shirmohammadi, "A collaborative IoT-gateway architecture for reliable and cost effective measurements," *IEEE Instrumentation & Measurement Magazine*, vol. 22, no. 6, pp. 11-17, 2019.

[16] L. Xuan and L. Yongzhong, "Research and implementation of Modbus TCP security enhancement protocol," *Journal of Physics: Conference Series*, vol. 1213, no. 5, p. 052058, 2019.
[17] M. Faisal, A. A. Cardenas, and A. Wool, "Modeling Modbus TCP for intrusion detection," in *IEEE Conference on Communications and Network Security (CNS), 2016*, pp. 386-390.
[18] V. G. Găitan and I. Zagan, "Modbus protocol performance analysis in a variable configuration of the physical Fieldbus architecture," *IEEE Access*, vol. 10, pp. 123942-123955, 2022.
[19] N. Anand, G. Joseph, S. S. Oommen, and R. Dhanabal, "Design and implementation of a high speed serial peripheral interface," in *International Conference on Advances in Electrical Engineering (ICAEE), 2014*, IEEE, pp. 1-3.
[20] B. Mishra and A. Kertesz, "The use of MQTT in M2M and IoT systems: A survey," *IEEE Access*, vol. 8, pp. 201071-201086, 2020.
[21] L. Li, W. Chou, W. Zhou, and M. Luo, "Design patterns and extensibility of REST API for networking applications," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 154-167, 2016.
[22] F. Tramarin, S. Vitturi, M. Luvisotto, and A. Zanella, "On the use of IEEE 802.11 n for industrial communications," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1877-1886, 2015.
[23] C.-Y. Wang and H.-Y. Wei, "IEEE 802.11 n MAC enhancement and performance evaluation," *Mobile Networks and Applications*, vol. 14, pp. 760-771, 2009.

Paradon Boonmeeruk He received a B.S. degree in Electronics Engineering Technology Instrumentation and Control from King Mongkut's University of Technology North Bangkok in 2020. He is currently pursuing a master's degree in Electrical & Biomedical Engineering at the Faculty of Engineering, Prince of Songkla University, Songkhla, Thailand.

Pichet Palrat He received a B.S. degree in Electrical & Biomedical Engineering from the Faculty of Engineering, Prince of Songkla University, Songkhla, Thailand, in 2023.

Kiattisak Wongsopanakul He received a B.S. degree in Electrical Engineering from the Faculty of Engineering, Prince of Songkla University, Songkhla, Thailand, an M.S. degree in Electrical Engineering from the New York Institute of Technology, USA, and an M.S. degree in Business Administration from the Faculty of Management Sciences, Prince of Songkla University, Songkhla, Thailand. He earned his Ph.D. degree in Electrical Engineering from Wayne State University, USA.At present, Dr. Kiattisak Wongsopanakul serves as a lecturer in Electrical & Biomedical Engineering at Prince of Songkla University, Songkhla, Thailand.