A Low-Cost, Open Source IoT-Based SCADA System Design, and Implementation for Photovoltaics

Lawrence O. Aghenta, and M. Tariq Iqbal
ECE, Faculty of Engineering, Memorial University of Newfoundland
St. John's, NL, Canada.
Emails: loaghenta@mun.ca, and tariq@mun.ca

***Abstract*—Supervisory Control and Data Acquisition (SCADA) has largely been proprietary, pricey, and therefore uneconomical for smaller applications. With proprietary SCADA systems, there is also the problem of interoperability with the existing system components, and communication systems. In this paper, we present the design and implementation of a low-cost, open source SCADA system based on the Internet of Things SCADA architecture. The proposed SCADA system consists of current and voltage sensors for data collection, an ESP32 micro-controller (Remote Terminal Unit) for receiving and processing the sensor data, and ThingsBoard IoT Server (Master Terminal Unit) for historic data storage and human machine interactions. The ThingsBoard IoT Server is locally installed on a Raspberry Pi single-board computer. Message Queuing Telemetry Transport protocol is implemented for data transfer over a local Wi-Fi connection. The system design procedures, testing and the results are presented in the paper.**

***Index Terms*—Low Cost, Open Source, SCADA, ThingsBoard, Internet of Things, ESP32 with OLED, Raspberry Pi, MQTT, Automation, Instrumentation and Control.**

I. INTRODUCTION

Energy shortage and Global Warming are some of the major challenges facing the world today, especially with the recent rapid industrial development across the globe. As energy experts continue to capture clean and renewable energy sources for the benefit of mankind, these sources are continuously being injected into today's power systems. These clean and renewable sources are incorporated with the conventional energy generation systems to form Hybrid Power Systems (HPS). These hybrid power systems together with energy storage systems, power electronic converters, as well as other power system devices such as communication systems needed for their successful operations are usually spread over large geographical areas, sometimes in harsh environments. As a result of this distributed nature, the interconnection of these systems to generate and supply energy presents numerous challenges, such as power quality issues, voltage tolerances, frequency control, grid synchronization and metering, data exchange and communications between components, as well as the safety and security of both assets and personnel. In order to overcome these challenges and to ensure seamless power system operations, diverse sensors, micro-controllers, micro-processors, actuators, valves, pumps, etc. are usually connected to various points of interest in the entire HPS to acquire important data such as current, voltage, power, and so on. and for real-time data monitoring, remote and co-ordinated controls. Supervisory Control and Data Acquisition (SCADA) is the perfect solution for these tasks [1].

SCADA refers to the combination of telemetry and data acquisition. It encompasses the collection of information (data) from distributed process facilities, the transfer of these data to a central location, analysis of these data to know the current states of the distributed process facilities, supervisory control of the process facilities, displaying these data on a number of operator screens, and conveying the necessary control actions back to these distributed process facilities for the local operator's actions [2]. The major functions of SCADA include [2]: Data acquisition; Data presentation; Supervisory Control; Networked data communication; Alarm processing; Historic data storage, data trending and reporting; and Remote monitoring. The architectural design of a SCADA system is made up of four basic elements: field instrumentation devices such as sensors which collect data from the distributed process facilities being managed, Remote Terminal Units (RTUs) such as single-board computers (PLCs, micro-controllers, etc.) for acquiring, processing and parsing these sensor data, Master Terminal Units (RTUs) such as IoT servers and platforms for data processing and human machine interactions, and finally SCADA communication channels for connecting the RTUs to the MTUs, and for data transfer [3]. SCADA architectures have evolved over the years, starting from the very first generation SCADA systems in the 70s called Monolithic SCADA, through the second generation SCADA systems called distributed SCADA (80s and 90s), and the third generation SCADA systems called Networked SCADA systems in the 90s and early 2000s, to the most recent SCADA architecture called the Internet of Things (IoT) SCADA architecture (4th generation) [4]. The SCADA system proposed in this work is based on the Internet of Things SCADA architecture. The Internet of Things concept refers to the interconnection of physical objects, embedded electronics, software and sensors, and so on, to enable real-time data exchange and communication between these devices and an operator over a common network or the web [5]. The IoT-based SCADA system incorporates web or cloud services with the conventional SCADA system for a more robust remote monitoring and control [4]. In general, there are two classes of SCADA systems, and they include Proprietary (Commercial), and Open Source SCADA systems. However, proprietary SCADA solutions are largely expensive and mostly economically unjustifiable for smaller power system applications. In addition, there is the problem of interoperability with the existing power system infrastructures. For these reasons, an open source SCADA system represents the most flexible and the most cost-effective SCADA solution [6].

II. RELATED WORKS

Numerous attempts have been made to reduce the over-dependence on proprietary SCADA systems. For power system applications for example, J. Lee et al. [1] have proposed an IoT-based open source SCADA system for the remote monitoring, power control, and distributed data processing of a standalone offshore wave-wind hybrid power generation system based on the IEC61850 standard. Elsewhere, K. Kao et al. [7] developed an IoT-based SCADA system for inverter monitoring and remote control. In another development, authors in [8] proposed a cloud-based SCADA system by integrating JustIoT framework with the conventional open source SCADA architecture. Although there are some studies on the design of SCADA systems in general for standalone or grid connected hybrid power systems, most IoT-based remote monitoring and control systems, especially using the lightweight data transfer protocol for the Internet of Things called

MQTT, have focused on applications such as smart healthcare [9], [10], home automation [11], [12], infrastructure and transport [13], and so on.

The major issue with most of the reviewed IoT-based open source SCADA systems above is that the solutions are rather cumbersome as they involve a lot of technologies, tools and programming. In addition to the complex nature of the reviewed solutions, most of the authors either used a web-based IoT platform for data visualization, storage, and other human machine interactions or designed a web platform for data visualization and human machine interactions using multiple web technologies. The major problem with using web-based platforms for data management in a critical SCADA system is that the stored data are highly susceptible to internet attacks since the web-based platforms require the public internet for data access just like every other website out there. However, the importance of data security in a SCADA system cannot be overemphasized. This is because attacks on a SCADA system can compromise the critical infrastructures being managed, which could result in devastating economic and operational setbacks. Data integrity in a SCADA system can be ensured by using several techniques, including securing the data communication channel or network such as data encryption, securing the hardware components, or securing the cloud server where the data are stored [2], [4].

In this paper, we implement a combination of private network management and private cloud server management strategies to ensure the security of the proposed IoT-based SCADA system. To achieve this, the ThingsBoard IoT Server for data management, storage and human machine interaction is locally hosted on a Raspberry Pi machine, and a private Wi-Fi network is created with a Wi-Fi router while data transfer is made possible using the MQTT data transfer protocol over the Wi-Fi network, such that only the users with the right authorizations can have access to the stored data. On this private network (MUN and Wi-Fi Network), the operator can take multiple measures such as access control, authentication, authorization, firewalls, etc. to protect the data in the cloud. In addition to taking these measures to ensure the security of the proposed SCADA system, we implement the lightweight ISO standard data transfer protocol for the Internet of Things, the Message Queueing Telemetry Transport (MQTT) protocol, for the sensor data transfer from the MQTT client (ESP32 device) to the Raspberry Pi-installed ThingsBoard IoT Server which serves as the MQTT broker. To the best of our knowledge, we have not found a single literature where a locally installed ThingsBoard IoT Server has been used as the MTU in an IoT-based SCADA system design. Furthermore, with the organic light-emitting diode (OLED) display of the ESP32 device (RTU) used in our design, we ensure that a local operator is able to visualize the current state of the process plant being managed by seeing the data values on the OLED screen, in addition to receiving updates from the remote SCADA operator at the server side. This is also an additional SCADA feature considered in this work.

III. Proposed SCADA System Architecture

Two configurations are considered in our proposed solution. In configuration A (Fig. 1), the Raspberry Pi 2 micro-controller hosting the ThingsBoard IoT server (MQTT Broker) where the received PV data are processed and stored is connected through an Ethernet cable to MUN network. Here, authorized users on MUN network can access the stored PV data and visualize the created dashboards on the ThingsBoard Server. Although this configuration presents a great deal of flexibility, it poses security risks to the stored data since external internet users can access the stored data remotely. In configuration B (Fig. 2), the public internet is not used, and the Wi-Fi Router is used to create a form of industrial network such that only the authorized users nearby can access the stored PV data on the ThingsBoard IoT server platform.

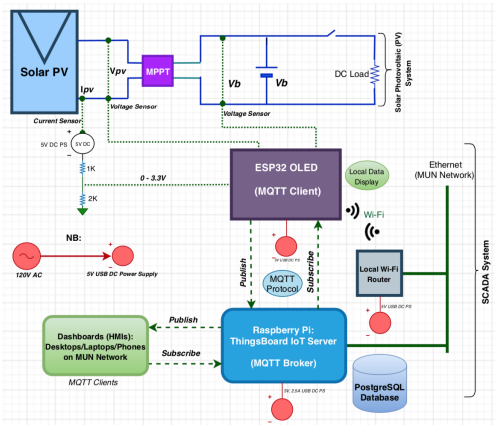

Fig. 1. The proposed SCADA system configuration A.

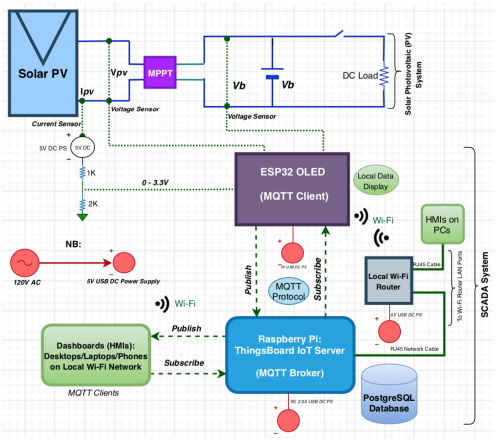

Fig. 2. The proposed SCADA system configuration B.

IV. Proposed SCADA System Components

The components of the proposed open source SCADA system include the Hall Effect Current and Voltage Sensors which serve as the field instrumentation devices to acquire the PV system data, the versatile ESP32 micro-controller, with OLED display, which is the remote terminal unit, and is configured as the MQTT Client to process and publish the sensor data using MQTT protocol, a Raspberry Pi2 single-board computer upon which the ThingsBoard IoT server, which is the master terminal unit, and is configured as the MQTT Broker, is built for human machine interactions, data storage, dashboards, alarms, data publishing and subscription, and finally, a Wi-Fi Router for creating the TCP/IP Wi-Fi connection for the MQTT protocol implementation.

A. Sensors (field instrumentation devices)

Three low-cost analog sensors are used, including two MH Electronic Voltage Sensor modules, and one ACS 712 Hall Effect Current Sensor.

1) MH Electronic voltage sensors: This low-cost analog voltage sensor uses the concept of voltage divider to measure voltage with its in-built series connection of a 7.5 K resistor and a 30 K resistor. Its operating voltage range is 3.3 V to 5.0 V, and it is capable of detecting supply voltages in the range of 0.025 V to 25 V using a 12-bit ADC.

2) ACS 712 Hall Effect current sensor: This low-cost sensor is based on the principle of Hall Effect. It has a low-noise resistance current conductor, low-noise analog signal path, and close to zero magnetic hysteresis. The 30 A DC module used in this work has 66 to 185 mV/A output sensitivity, and it operates on a 5 V single supply voltage. Because the signal voltage of the current sensor is 5 V, it is not suitable for direct connection to the ADC pins of the ESP32 micro-controller as the ADC pins operate between 0 V to 3.3 V. Therefore, a pull-down or step-down resistors arrangement is used to match the 5 V signal requirement of the current sensor to the 3.3 V signal capability of the ESP32 ADC pins.

B. TTGO ESP32 LoRa32 OLED micro-controller (RTU)

The most important specifications of the board include the following: IEEE 802.11 b/g/n Standards HT40 Wi-Fi Transceiver; Dual Core Processor, clocked at 240 MHZ; 4 MB on-board Flash, and Wi-Fi and Bluetooth Antenna; 18 ADC pins and over 30 GPIO pins (I/O Pins); 0.96 inch white OLED display screen; 5 V single power supply, and support for a single-cell lithium-polymer (LiPo) battery; and Operating ADC signal voltage range of 1.8 V - 3.7 V.

C. Raspberry Pi single-board computer

The Raspberry Pi 2 model B device is a portable credit card-sized single-board computer featuring the BCM2836 quad core (4 processors in one chip) ARMv7 processor, and it is completely open source. In this work, the ThingsBoard IoT server is installed on the Raspberry Pi computer together with the PostgreSQL Database.

1) Wi-Fi Router (TCP/IP Wi-Fi connection): The D-Link Router (DI-524 Airplus G) is used to create the TCP/IP Wireless network connectivity over which the MQTT protocol is implemented for data transfer from the MQTT Client (ESP32) to the MQTT Broker (ThingsBoard IoT server).

D. ThingsBoard local server IoT platform

ThingsBoard is an open-source IoT platform for data collection, processing, visualization, and device management [15]. It provides an out-of-the-box IoT cloud or on-premises solution to enable server-side infrastructure for various IoT applications [15]. Built on Java 8 platform, ThingsBoard provides 100 percent support for standard IoT protocols for device connectivity, including MQTT, CoAP, and HTTP(S), and it presently supports three different database options: SQL, NoSQL, and Hybrid databases. The ThingsBoard platform uses these databases to store *entities* (such as devices, assets, dashboards, users, alarms, customers, etc.), and *telemetry data* (attributes, time-series sensor readings, statistics, events, etc.). *Telemetries* are time-series of key-value pairs of data associated with a specific device, and ThingsBoard stores its received data as *telemetries*. In this work, the PostgreSQL database is installed on the ThingsBoard server for *entities* and *telemetry* storage. ThingsBoard has two different editions, the Community Edition, which is free and wholly open source, and the Professional Edition, which has more advanced features. The Community Edition is used in this project. The key features and functions of the major components of the ThingsBoard architecture are presented as follows [15];

- **Transport components:** These include MQTT, HTTP, and CoAP-based APIs for device applications and firmware. Being a part of the ThingsBoard "Transport Layer", each of the components here helps to push data to the *Rule Engine*, and could also use *Core Services* to issue requests to the database to validate device credentials. The MQTT-based device API supported by the MQTT communication protocol is implemented in this work. MQTT is favoured ahead of the HTTP and CoAP because of its unique features like support for constrained resources such as low bandwidth, and it can be implemented over various TCP/IP connectivities. In addition, ThingsBoard server nodes act as an MQTT Broker that supports QoS levels 0 (at most once) and 1 (at least once), and a set of predefined topics which means that an external device (ESP32 in this case) can be configured as an MQTT Client to publish data to the server nodes.
- **Rule Engine components:** The ThingsBoard *Rule Engine*, which comprises of *Rule Node* and *Rule Chain*, helps in processing the incoming messages with user-defined logic and flow.
- **Core services:** These are responsible for handling REST API calls, monitoring device connectivity states, and WebSocket Subscriptions on entity, telemetry and attribute changes.
- **External systems:** Using the Rule Engine, communications can be established between ThingsBoard and external systems. This involves pushing data to external systems, processing the data, and reporting the results of the processed data back to ThingsBoard server for visualization.

ThingsBoard server can either be utilized directly on the *Live Demo* platform, installed on a private machine *On-Premise*, or hosted on a *Cloud Server* such as Amazon Web Services (AWS). The Live Demo platform requires the public internet for data access just like every other web application out there, which could leave the stored data vulnerable to internet attacks. On the other hand, hosting the ThingsBoard server on a Cloud platform such as AWS, requires not just the public internet for data access, it also requires subscriptions, which means more cost and the possibility of internet attacks [15]. However, security in a SCADA system is a critical issue as attacks on the SCADA could compromise the important company data stored in the cloud. Therefore, in our proposed solution, the on-premise, self-hosted ThingsBoard server option is implemented. By installing the ThingsBoard server on the Raspberry Pi machine connected to a private network, it can be operated with and without the public internet, depending on what configuration is chosen based on the desired security and flexibility of the operation. This represents a major contribution of this paper as no related works have been found where such measures were considered.

The interface of the installed ThingsBoard server on the Raspberry Pi machine showing the IP address, and the numerous menus such as *Rule Chains, Customers, Assets, Devices*, and so on, for various actions is shown in Fig. 3, and Fig. 4 shows the sensor data being posted to the ThingsBoard server nodes.

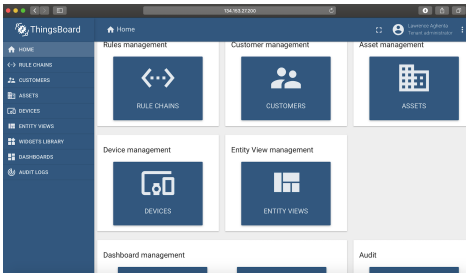

Fig. 3. Raspberry Pi-installed ThingsBoard server interface

E. MUN ECE Laboratory PV System Overview

In order to test the functionalities of the designed open source SCADA system, it is setup to acquire the solar photovoltaic (PV) data of the PV system at Memorial University (MUN) Electrical and Computer Engineering Department Laboratory. This PV system is made up of 12 Solar Panels covering a total area of 14 square meter

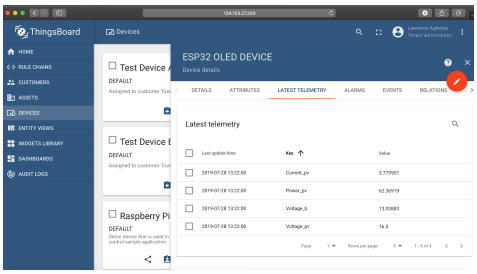

Fig. 4. Sensor data posting

and producing about 130 W and 7.6 A each. The proposed SCADA system is connected to just one set of the modules (about 260 W, and 14 A output) for testing purposes.

V. IMPLEMENTATION METHODOLOGY

The pseudocode describing the data (information) flow process is shown in Algorithm 1 below.

Algorithm 1: Data acquisition and logging algorithm:

Initialization;
1. Analog sensors measure and collect PV system data;
2. ESP32 reads sensor values on analog Pins 32, 34 and 35, and calculates values for Pins 32×34;
3. ESP32 displays the above values on Arduino IDE Serial Monitor and ESP32 OLED Screen;
4. ESP32 connects to local TCP/IP Wi-Fi Network with Wi-Fi Name and Password;
5. ESP32 MQTT Client identifies the local ThingsBoard IoT Server (MQTT Broker) via the Server IP Address;
6. ESP32 MQTT Client publishes sensor data to MQTT Broker over the TCP/IP Wi-Fi connectivity;
7. ThingsBoard Server displays data as Telemetry Messages on the specified Device using the Device Name and Access Token;
8. ThingsBoard Server Node logs the Telemetry Messages to Dashboards for data visualization;

while *ThingsBoard Server acknowledges data receipt* **do**
- 9. Display sensor data on ThingsBoard Server Node, Dashboards and ESP32 OLED Screen, and;
- 10. Display "DONE" on Arduino IDE Serial Monitor;
- **if** *No data receipt acknowledgement from ThingsBoard Server Node* **then**
  - 11. Display "FAILED.....retrying in 5 seconds" on Arduino IDE Serial Monitor;
- **else**
  - 12. Go to step 1;
- **end**

end

VI. PROTOTYPE DESIGN

As shown in Fig. 5, the Analog Current and Voltage Sensors are connected to the TTGO ESP32 LoRa32 OLED device on a Breadboard using electrical wires. The inputs of the sensors are connected to the points of interest on the PV panel and storage battery system (PV System) as shown.

VII. EXPERIMENTAL SETUP OF THE PROPOSED SCADA SYSTEM

As described in Section VI above, the hardware components were programmed, configured and setup for operation. The setup was then

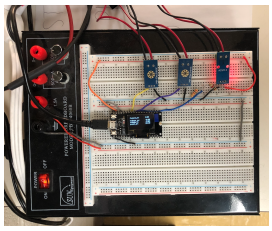

Fig. 5. Hardware implementation of the proposed SCADA system

hooked up to the solar PV System in MUN ECE Laboratory. Fig. 6 shows the analog sensors and ESP32 OLED device connected together and to the PV System, as well as some of the Dashboards created on the ThingsBoard IoT server platform (shown on the Laptop) for real-time data monitoring and supervisory control actions.

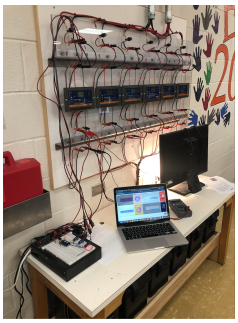

Fig. 6. Experimental setup of the proposed SCADA system

VIII. TESTING AND RESULTS

Having tested the proposed IoT-based open source SCADA system solution extensively, we present the results and some of the created HMIs (Dashboards) in this section.

A. Results

Each of the two hardware configurations, A and B (Figs. 1 and 2 respectively) was set up and connected to the standalone solar PV system. At the ThingsBoard IoT server platform, dashboards were created for the remote monitoring of the received sensor data, and for easy data trends visualizations. Fig. 7 shows multiple dashboards for the various PV system variables being acquired; the storage battery Voltage, and the PV Current, Voltage and Power. As can be seen, the vibrations of the values of each of the variables were due to the weather conditions in St. John's at the various times of testing as expected since PV system outputs are affected by environmental conditions such as solar irradiance and temperature. At the time of logging these data, a digital multimeter was also used to locally measure each of the PV system variables so as to validate the accuracy of the acquired data seen on both the OLED display screen and at the ThingsBoard server platform. The acquired sensor values were found to be the same as those measured locally with the multimeter. Fig. 8 shows a dashboard created to specifically test configuration A, while Fig. 9 shows another dashboard specifically created to test configuration B. As seen in the figures, the sudden increase and decrease in the values (especially in Fig. 8) happened at various times when the storage battery was being discharged with an electric load (a light bulb) connected across it. As expected, similar data values were recorded using both configurations A and B, with the values only affected by the prevalent environmental conditions at the time of testing. The major difference between the two configurations is the manner in which the recorded and stored data on the ThingsBoard server platform can be accessed as described earlier.

The proposed SCADA was tested extensively at various times of the day, and left connected to the PV system to continuously log the PV data for about a month so as to confirm the robustness of the designed system, and the results were found to be consistent with the locally measured values using a digital multimeter, showing that the SCADA system performed optimally and accurately regardless of the environmental conditions and duration of testing. Also, as shown in Figs. 5 and 6, the most recent data values were available for viewing on the ESP32 OLED display screen, thereby providing a local data monitoring interface whenever necessary.

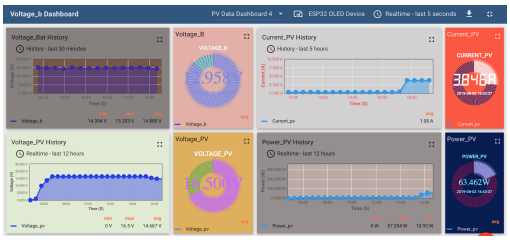

Fig. 7. Created dashboards showing real-time data

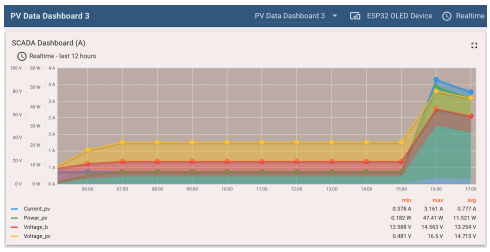

Fig. 8. Created dashboard (A) showing real-time data

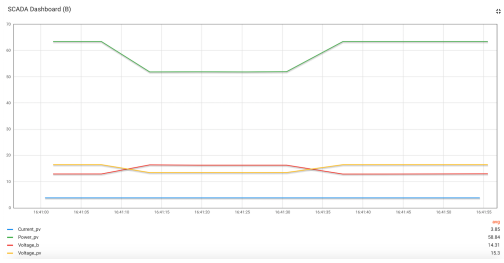

Fig. 9. Created dashboard (B) showing real-time data

IX. CONCLUSIONS

In this paper, we proposed a low-cost open source SCADA system based on the most recent SCADA architecture, the Internet of Things (IoT). We also demonstrated the hardware implementation of our proposed SCADA system solution using very few low-cost, low-power, open source and readily available components as the essential elements of the SCADA system. In designing our proposed SCADA solution, data security, data integrity, and system reliability were taken into consideration since security in a SCADA system is a critical issue. These considerations were implemented by locally installing the main data server, the ThingsBoard IoT server, on a Raspberry Pi single-board machine. Thus, the data server was locally hosted and self-managed on MUN Network such that data security and data integrity measures such as authentication, authorization, access control, log analysis, and firewalls are self-managed by the system administrator to ensure data security, data integrity, and system availability, and thus making sure that the system is reliable. We also demonstrated the use of the lightweight IoT application protocol, MQTT protocol, for data transmission in such applications. The overall SCADA system cost was found to be extremely low, about $280 CAD, and the overall power consumption while in operation was found to be minimal, about 9.3 W. We also demonstrated the performance of our proposed open source SCADA solution by testing it with a standalone solar photovoltaic (PV) system. From our testings and results, we showed that the proposed open source SCADA system operates properly and accurately. With the OLED display screen of the ESP32 micro-controller board used, a local real-time data monitoring interface was also incorporated into the proposed SCADA system solution. As a future work, we will look at incorporating various alarm types into the system to increase the functionalities of the system.

ACKNOWLEDGMENTS

The authors would like to thank the School of Graduate Studies, Faculty of Engineering and Applied Science, Memorial University and the Natural Sciences and Engineering Research Council of Canada (NSERC) Energy Storage Technology Network (NESTNet) for funding this research.

REFERENCES

[1] J. Lee, S. Lee, H. Cho, K. S. Ham and J. Hong, "Supervisory Control and Data Acquisition for Standalone Hybrid Power Generation Systems," Sustainable Computing: Informatics and Systems, Volume 20, 2018, Pages 141-154, ISSN 2210-5379, https://doi.org/10.1016/j.suscom.2017.11.003. (http://www.sciencedirect.com/science/article/pii/S2210537917303062).

[2] K. Stouffer, J. Falco and K. Kent, "Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security—Recommendations of the National Institute of Standards and Technology," Special Publication 800-82, Initial Public Draft, Sept. 2006.

[3] Xie Lu. "Supervisory Control and Data Acquisition System Design for CO2 Enhanced Oil Recovery." Master of Engineering Thesis, Technical Report No. UCB/EECS-2014-123. EECS Department, University of California at Berkeley, May 21, 2014.

[4] A. Sajid, H. Abbas and K. Saleem, "Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges,". IEEE Access, vol. 4, pp. 1375-1384, 2016. doi: 10.1109/ACCESS.2016.2549047.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,". IEEE Communications Surveys and Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015. doi: 10.1109/COMST.2015.2444095.

[6] M. Nicola, C. Nicola, M. Duță and D. Sacerdoțianu, "SCADA Systems Architecture Based on OPC and Web Servers and Integration of Applications for Industrial Process Control," International Journal of Control Science and Engineering, Vol. 8 No. 1, 2018, pp. 13-21. doi: 10.5923/j.control.20180801.02.

[7] K. Kao, W. Chieng and S. Jeng, "Design and development of an IoT-based web application for an intelligent remote SCADA system," 2018 IOP Conference Series: Materials Science and Engineering, vol. 323, pp. 012025. doi: 10.1088/1757-899X/323/1/012025.

[8] W. Li, J. Wang, C. Yen, Y. Lin and S. Tung, "Cloud supervisory control system based on JustIoT," 2018 IEEE International Conference on Smart Manufacturing, Industrial and Logistics Engineering (SMILE), Hsinchu, 2018, pp. 17-20. doi: 10.1109/SMILE.2018.8353974

[9] B. S. Sarierao and A. Prakasarao, "Smart Healthcare Monitoring System Using MQTT Protocol," 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, 2018, pp. 1-5. doi: 10.1109/I2CT.2018.8529764

[10] F. Wu, T. Wu, and M. Yuce, "An Internet-of-Things (IoT) Network System for Connected Safety and Health Monitoring Applications," Sensors, vol. 19, no. 1, p. 21, Dec. 2018. https://doi.org/10.3390/s19010021

[11] R. K. Kodali and S. Soratkal, "MQTT based home automation system using ESP8266," 2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), Agra, 2016, pp. 1-5. doi: 10.1109/R10-HTC.2016.7906845

[12] M. Bassoli, V. Bianchi, and I. Munari, "A Plug and Play IoT Wi-Fi Smart Home System for Human Monitoring," Electronics, vol. 7, no. 9, p. 200, Sep. 2018. https://doi.org/10.3390/electronics7090200

[13] B. Mishra, "TMCAS: An MQTT based Collision Avoidance System for Railway networks," 2018 18th International Conference on Computational Science and Applications (ICCSA), Melbourne, VIC, 2018, pp. 1-6. doi: 10.1109/ICCSA.2018.8439562

[14] Y. Lee, W. Hsiao, C. Huang and S. T. Chou, "An integrated cloud-based smart home management system with community hierarchy," in IEEE Transactions on Consumer Electronics, vol. 62, no. 1, pp. 1-9, February 2016. doi: 10.1109/TCE.2016.7448556

[15] ThingsBoard Documentation. Available online: https://thingsboard.io/docs/. (accessed on 20 September 2019)