electronics

MDPI

Article

Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing

Lawrence Oriaghe Aghenta * and Mohammad Tariq Iqbal *

Department of Electrical and Computer Engineering, Faculty of Engineering and Applied Science, Memorial University of Newfoundland (MUN), St. John's, NL A1B 3X5, Canada

* Correspondence: loaghenta@mun.ca (L.O.A.); tariq@mun.ca (M.T.I.)

Received: 30 June 2019; Accepted: 20 July 2019; Published: 24 July 2019

Abstract: Supervisory Control and Data Acquisition (SCADA) is a technology for monitoring and controlling distributed processes. SCADA provides real-time data exchange between a control/monitoring centre and field devices connected to the distributed processes. A SCADA system performs these functions using its four basic elements: Field Instrumentation Devices (FIDs) such as sensors and actuators which are connected to the distributed process plants being managed, Remote Terminal Units (RTUs) such as single board computers for receiving, processing and sending the remote data from the field instrumentation devices, Master Terminal Units (MTUs) for handling data processing and human machine interactions, and lastly SCADA Communication Channels for connecting the RTUs to the MTUs, and for parsing the acquired data. Generally, there are two classes of SCADA hardware and software; Proprietary (Commercial) and Open Source. In this paper, we present the design and implementation of a low-cost, Open Source SCADA system by using Thinger.IO local server IoT platform as the MTU and ESP32 Thing micro-controller as the RTU. SCADA architectures have evolved over the years from monolithic (stand-alone) through distributed and networked architectures to the latest Internet of Things (IoT) architecture. The SCADA system proposed in this work is based on the Internet of Things SCADA architecture which incorporates web services with the conventional (traditional) SCADA for a more robust supervisory control and monitoring. It comprises of analog Current and Voltage Sensors, the low-power ESP32 Thing micro-controller, a Raspberry Pi micro-controller, and a local Wi-Fi Router. In its implementation, the current and voltage sensors acquire the desired data from the process plant, the ESP32 micro-controller receives, processes and sends the acquired sensor data via a Wi-Fi network to the Thinger.IO local server IoT platform for data storage, real-time monitoring and remote control. The Thinger.IO server is locally hosted by the Raspberry Pi micro-controller, while the Wi-Fi network which forms the SCADA communication channel is created using the Wi-Fi Router. In order to test the proposed SCADA system solution, the designed hardware was set up to remotely monitor the Photovoltaic (PV) voltage, current, and power, as well as the storage battery voltage of a 260 W, 12 V Solar PV System. Some of the created Human Machine Interfaces (HMIs) on Thinger.IO Server where an operator can remotely monitor the data in the cloud, as well as initiate supervisory control activities if the acquired data are not in the expected range, using both a computer connected to the network, and Thinger.IO Mobile Apps are presented in the paper.

Keywords: open source; SCADA; Thinger.IO; Internet of Things; ESP32 Thing; Raspberry Pi; automation; instrumentation and control

1. Introduction

The acronym, SCADA, stands for Supervisory Control And Data Acquisition. A SCADA system is a collection of both software and hardware components that allows for the supervision and control of

plants and industrial processes both locally and remotely. The system involves the examination, collection, and processing of data in real time, as well as data logging for historical purposes. The architectural design of a standard SCADA system starts with Remote Terminal Units (RTUs), and/or Programmable Logic Controllers (PLCs). These RTUs and PLCs are micro-controllers or microprocessors that communicate and interact with Field Instrumentation Devices (FIDs) such as sensors, actuators, valves, pumps and transmitters. These communication data are routed from the controllers or processors via a SCADA Communication Channel to the SCADA computers known as Master Terminal Units (MTUs) where the data are interpreted and displayed on a Human Machine Interface (HMI) allowing operators and important decision makers to analyze and react to process events [1–5].

SCADA technology has evolved over the past 30 years as a method of monitoring and controlling distributed processes [5]. Before the emergence of SCADA, plant personnel had to monitor and control industrial processes via selector switches, push buttons, and dials (for analog signals), which meant that plants needed to maintain a good number of personnel on site during production to be able to carry out these manual monitoring and control tasks. As industrial processes grew and sites became more remote in nature, relays and timers were used to assist in the supervision and control of processes, which meant that fewer number of personnel were required on site to oversee their operations. While relays and timers provided a decent level of automation, they required more resources to manage their operations. The need to automate more processes, which coupled with the difficulties associated with the previous monitoring and control solutions gave birth to the first generation SCADA systems in the 1970s called Monolithic SCADA, and they were stand-alone units. With the advent of Local Area Network (LAN) and HMI softwares in the 1980s and 1990s, and with computer systems getting smaller and smarter, it became possible to connect SCADA systems to related systems which gave rise to the second generation SCADA called Distributed SCADA. Unfortunately then, the communications were typically proprietary which meant that the connections outside of the vendors of a particular SCADA system were not possible [2–4].

Later in the 1990s and 2000s, SCADA began to implement open system architectures with communication protocols that were no longer vendor specific, leading to more connection capabilities in the form of a wide area network. This resulted in a more improved SCADA system called Networked SCADA system (3rd generation). With computer technologies growing rapidly, this SCADA was quickly out of date as other technologies were becoming more efficient and more in tune with the latest Information Technology (IT) developments. SCADA manufacturers had to rise to the challenge to come up with a SCADA architecture with great advantages over older SCADA systems. This is the Internet of Things SCADA architecture (4th generation) which incorporates cloud services with the conventional SCADA system for more robust monitoring and control [6]. This SCADA allows for real-time plant information to be accessed from anywhere around the world using various operating systems and platforms [2,3,7]. The Internet of Things (IoT) concept has to do with the connection of physical objects with embedded electronics, software, sensors and connectivity to enable data interchange between these devices and an operator over a common platform or the web [6–10].

Presently, as in the past, automation companies such as Emerson, Siemens, Schneider Electric, and Allen Bradley develop various forms of SCADA hardware and software which they sell as turnkey solutions to end users. Examples of such solutions include Ovation SCADA communication server (Emerson), Simatic WinCC (Siemens), Clear SCADA server (Schneider Electric), and Micro SCADA (Allen Bradley) [3,11]. Currently, these systems come with various IoT-based MTUs for data visualization and process management such as the Totally Integrated Automation (TIA) portal in Siemens' Simatic WinCC [11]. In addition to the huge initial capital costs of buying these SCADA systems, which are in thousands of dollars, the end user is made to pay for annual maintenance and support fees to use the SCADA system solutions. For very large companies, like most companies in the Oil and Gas sectors, the costs of owning these commercial SCADA systems might be justifiable or affordable. However, for smaller companies with no enormous financial resources, but with the

need to deploy SCADA solutions to monitor and control their distributed facilities, such as companies in the power sector or renewable generation systems, these commercial SCADA solutions do not represent a profitable choice. With the commercial systems, there is also the problem of interoperability with the existing infrastructures, for example power electronic converters in a power system [12]. Seamless communication among devices in modern power systems is the key to successful SCADA implementation [13,14]. Therefore, a low-cost, open source SCADA solution represents the most viable solution [4,7,12].

In this work, we propose a low-cost, open source SCADA system based on the most recent SCADA architecture, which is the Internet of Things [6,10,15]. Our proposed SCADA system uses reliable and commonly available components to achieve the four basic functions of a SCADA system which the available commercial SCADA systems also perform: Data acquisition, networked data communication, data presentation, and remote monitoring and supervisory control [1,9].

The remainder of this paper is organized as follows. In Section 2, we present the related works, problem statements, and the proposed SCADA system as a solution to the identified problems. In Section 3, we present system descriptions, including the proposed SCADA system design configurations. In Section 4, we present the components of the proposed SCADA system and the detailed description of each of the components. Section 5 presents the implementation methodology, Section 6 presents the prototype design, Section 7 presents the experimental setup of the proposed SCADA system, and Section 8 presents the testing carried out and the results. In Section 9, the key features of the designed SCADA system are discussed, including the system cost and power consumption analysis. The paper is concluded in Section 10, and future work presented in Section 11.

2. Literature Review

The research communities all over the world have tried to solve the problems associated with commercial SCADA systems (high costs and compatibility issues) by developing various open source SCADA solutions, each with varying costs and functionalities. Rajkumar et al. [16] have proposed a low-cost, open source SCADA system using Arduino Uno micro-controller as the sensor gateway, and ZigBee Radio Modules for data transmission from their field instrumentation devices such as flow sensors, temperature sensors, level sensors, control valves, and pumps to a data processing software, which they created with Java, where reports on the state of the process facility are generated and visualized. In [17], a form of low-cost web-based SCADA solution is proposed. In this solution, Radio Frequency (RF) Receivers connected to a controller circuitry collect plant data, and the data are made available to the controller which, in turn parses the data to a driver circuitry for set point verification before the data get transmitted via the internet to a web-based SCADA server for visualization. The proposed solution here is complex as it involves the use of a mathematical model for data verification in the controller and driver circuitries before transmission to the web-based SCADA server.

Elsewhere, Merchan et al. [18] implemented an open source SCADA system by using the general purpose programming platform, Python. Their proposed system is made up of three layers: a Device Layer, which is the process plant being managed, a Controller Layer, which comprises of two PLCs, and lastly, a Supervision Layer comprising of HMI SCADA client, and Python Open Platform Communications Unified Architecture (OPC-UA) Server with Structured Query Language (MySQL) Database, both installed in a Raspberry Pi3. Here, communication between the control devices is via Ethernet. This solution uses a lot of components which means more power consumption and less reliability of the resultant SCADA system. The authors in [19] have implemented a form of open source SCADA system for the monitoring of the level and flow rate of water in a container by using the open source SCADA platform called OpenSCADA where HMIs are created for data visualization. The major issues here are that the OpenSCADA software is not entirely free as it requires user subscriptions, and the proposed solution involves a large amount of logical programming for accurate data acquisition and visualization in the OpenSCADA platform.

Similar to [19], Avhad et al. [20] have proposed a form of open source automation system using Vijeo Citect SCADA software as the MTU, AtMEGA 2560 micro-controller as the RTU for sensor data collection, and ModBus protocol as the communication channel between the MTU and the RTU. In another development, Mononen et al. [21] proposed a low-cost open source SCADA system comprising of sensor network for data collection, and a microprocessor for data processing and transmission via UDP Ethernet frame to a cloud environment. The cloud environment is composed of virtual machines running user-defined algorithms in the cloud for accurate data processing and handling. Just like [21], the authors in [7] have developed a low-cost SCADA system based on IoT technology where the RTU is connected to the field devices through TCP/IP and supported on a NoSQL database. The functionalities of their developed system were evaluated using a traffic management process. The major problem with these systems is that they are complex systems which will be difficult for an ordinary end-user to use. Furthermore, the solution in [21] is prone to errors as it requires a great deal of user-defined algorithms in the cloud.

In this paper, we present the detailed design of a SCADA system using very few low-cost, low-power, and completely open source components. In our proposed SCADA solution, a locally installed Thinger.IO Server IoT Platform serves as the Master Terminal Unit where HMIs are created for data visualization and processing, and the versatile Sparkfun ESP32 Thing micro-controller serves as the Remote Terminal Unit for receiving and sending the sensor data from the Field Instrumentation Devices (sensors), while a local Wi-Fi Network provides the Communication Channel between the MTU and the RTU, resulting in a form of industrial network as implemented by the commercial SCADA vendors all over the world [3,9,11].

The authors in [22–24] have used Thinger.IO to implement their respective remote monitoring and automation systems. In their systems, the acquired sensor data are sent to Thinger.IO cloud platform where HMI dashboards are created for data visualization. In [22], the visualized data are used for smart home energy decisions, while the acquired data in [23] are used for RESTful motion detection where the operator is notified of process changes via the Thinger.IO platform. The implemented system in [24] is a smart emergency response system using the powerful IoT properties of the Thinger.IO platform. Unlike our work where a locally installed Thinger.IO IoT server is used, these authors have used the web-based Thinger.IO platform requiring the public internet for access which means that the resulting monitoring solutions are vulnerable to internet attacks. Security in a SCADA system is a serious issue as attacks on a SCADA system can compromise the critical process facilities being managed [4,9,25,26]. To ensure the security of a SCADA system, several techniques can be used. Some of these techniques include a security technique focused on the communication channel or network as discussed in [27–29], a technique focused on protecting the hardware components as in [30], and a data-driven technique focused on protecting the cloud server as discussed in [6,25,31,32], or a combination of two or more of these techniques [27,33]. The SCADA system proposed in this work considers a combination of some of these security techniques, including the private network management and the data-driven private cloud server management techniques. Here, because the main cloud server is locally hosted on Memorial University (MUN) network, the operator has full control of the security of the system and can take several measures to protect the data in the cloud, such as ensuring access control, whitelists, authentication, authorization, firewalls, regular risk assessment, continuous monitoring and log analysis, updating and patching regularly, etc. [6,15,25,28]. This is why we propose a locally installed Thinger.IO IoT server in this work, and to the best of our knowledge from reviewed literatures, and related works, we have not found a SCADA system solution where a locally installed Thinger.IO IoT server has been used.

It should be noted that the SCADA system solution presented in this paper is similar in functionalities to our previous open source IoT-based SCADA system [9], where we used a locally installed EMONCMS IoT server as the Master Terminal Unit, and Arduino Uno and Raspberry Pi as the RTUs to receive sensor data, with Node-RED flows, which used Ethernet for communication, serving as the Communication Channel between the MTU and the RTUs. However, as the components

and the design and implementation methodologies used in both systems are totally different, the work presented here is not a continuation of our previous work but rather a different open source SCADA system solution meant to tackle the shortcomings in the available commercial SCADA systems, and the reviewed open source SCADA system solutions. Also, in our previous open source SCADA system solution, more components were used which meant more difficulty in implementation and usage, more power consumption, less reliability, and more cost. Furthermore, unlike our previous SCADA system solution where a single configuration was considered, two configurations are tested and presented in this current work. The first configuration is one in which data on the Thinger.IO IoT platform are accessible over the internet as long as the user is within the Memorial University network and has the authorizations to connect to the network. In this case, the Raspberry Pi micro-controller hosting the Thinger.IO IoT Server is connected to MUN Network using an RJ45 Ethernet cable. The second configuration is such that only the users connected to the locally created Wi-Fi Network (this connection is either wireless or via network cables to the LAN ports of the Wi-Fi Router) can connect to the Thinger.IO IoT platform for data visualization, remote monitoring and supervisory control. This second configuration creates a form of industrial network which is only accessible to authorized users on the network. In either configurations, the proposed SCADA system solution here will help to tackle the mentioned setbacks in the systems presented in literatures, high cost and compatibility issues in the available commercial SCADA systems, and the minor setbacks in our previous SCADA system solution by using fewer components which are known to consume less power, and are less expensive, while ensuring the same robust SCADA functionalities as in both our previous work and the available commercial SCADA system solutions.

The rest of this paper is dedicated to the system and components descriptions, implementation methodology, prototype design, experimental setup and testing of the proposed low-cost, open source IoT-based SCADA system solution.

3. System Description

The proposed low-cost, open source SCADA system is based on the Internet of Things (IoT) SCADA architecture which is the fourth and most recent SCADA architecture [4,9,10]. In connecting the hardware components together, two configurations are considered. In the first configuration, the Raspberry Pi is connected to MUN Network via an Ethernet cable so that users on the network with the right authorizations can visualize the acquired data on the Thinger.IO IoT server platform. Also, in this configuration, by opening the Thinger.IO port on MUN network, users can access the stored data over the internet using their private office or home network. In the second configuration, the Raspberry Pi is connected to one of the LAN ports of the local Wi-Fi Router such that only the machines connected to the Wi-Fi network, either wirelessly or via Ethernet cables connected to the LAN ports of the Router, can access the data on Thinger.IO local server IoT platform by using the IP address of the Raspberry Pi. Although the first configuration is more flexible, the second configuration is more secure, as this configuration ensures a kind of industrial network is created, with external internet users, even on MUN network, shut out. In either of the configurations, the ESP32 Thing can connect to the Thinger.IO server over Wi-Fi by using the IP address of the Raspberry Pi. The two configurations are shown below in Figures 1 and 2 respectively:

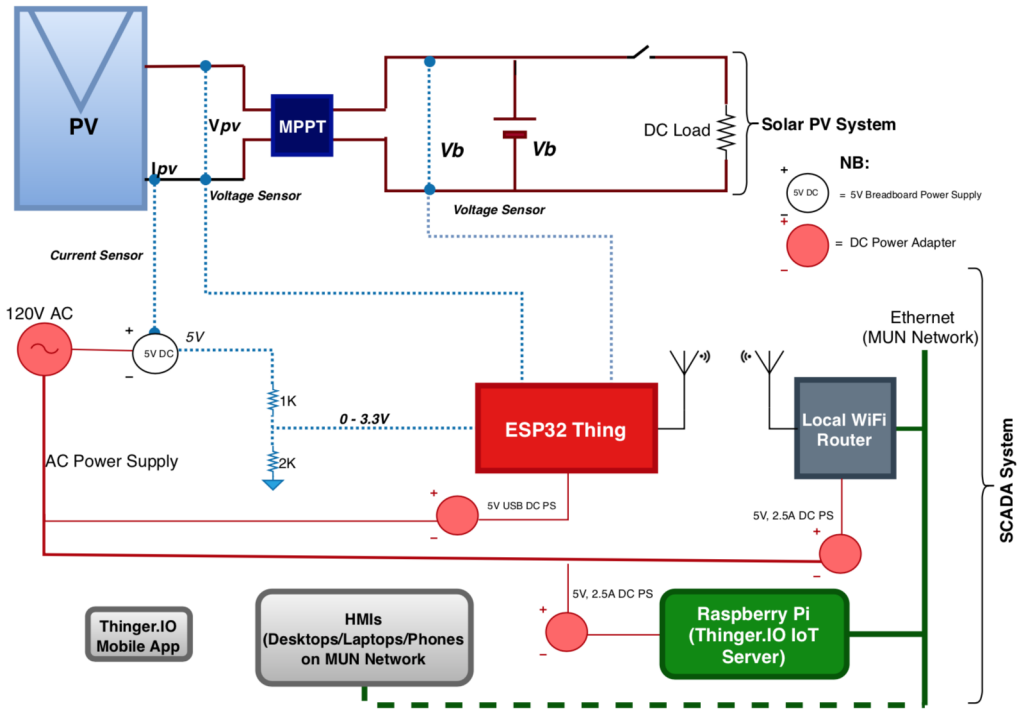

Figure 1. The first configuration (A) of the proposed Supervisory Control and Data Acquisition (SCADA) system.

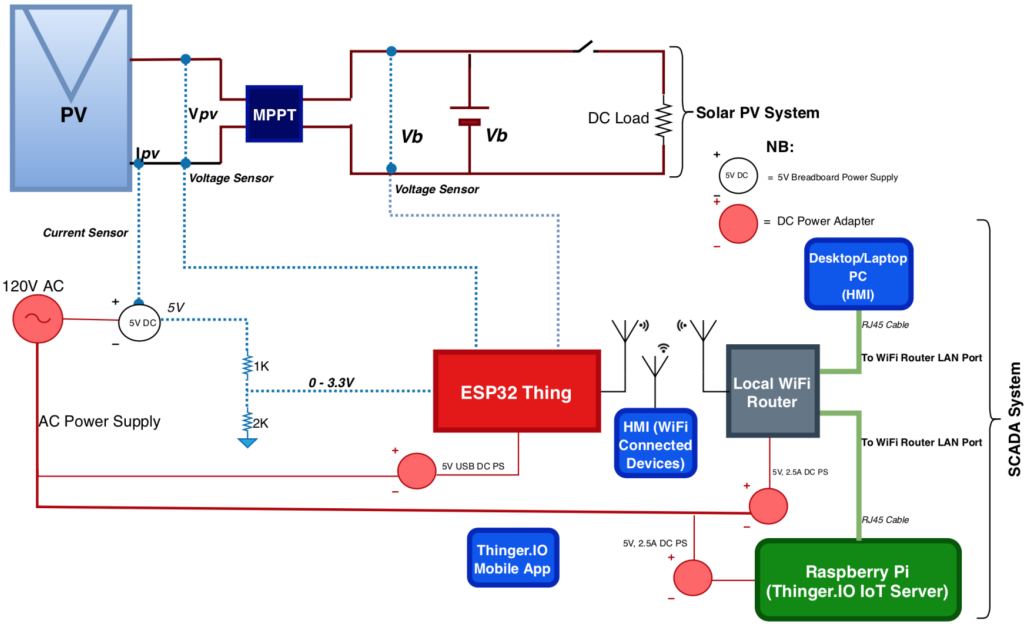

Figure 2. The second configuration (B) of the proposed SCADA system.

4. Components of the Proposed SCADA System

The SCADA system proposed in this work is made up of analog voltage and current sensors for data acquisition, SparkFun ESP32 Thing micro-controller for receiving, processing and parsing the sensor data, Wi-Fi Router for local Wi-Fi network creation (communication channel), and Thinger.IO local IoT server with graphical user interface (dashboards) created for remote sensor data monitoring and supervisory control. Each of these components is described in details below:

4.1. Sensors

Sensors are the Field Instrumentation Devices in the proposed SCADA system as they are connected directly to the PV system being managed to acquire the desired data [1,34]. Three analog sensors are used in our setup; one ACS 712 Hall Effect Current Sensor, and two MH Electronic Voltage Sensor modules. The operating signal voltage (VCC) of the current sensor is 5 V single supply, and that of voltage sensor is between 3.3 V to 5 V, while that of the ESP32 Thing micro-controller is 0 V to 3.3 V. This means that the current sensor is not suitable for direct connection to the analog-to-digital converter (ADC) Pins of the ESP32 Thing. Therefore, in order to ensure that the sensor matches the 3.3 V signal voltage of the ESP32 Thing, some level shifting is carried out by using a pull-down or step-down resistor arrangement (Figure 3). This will ensure the accuracy of the measured sensor value. The properties of these sensors and their usage in this work are described below:

4.1.1. ACS 712 Hall Effect Current Sensor

The ACS 712 Hall Effect Current Sensor, manufactured and supplied by Allegro MicroSystems, LLC. is a low-cost, fully integrated, Hall Effect-based linear current sensor IC with 2.1 kVRMS Isolation and a low-resistance current conductor. It has a low-noise analog signal path, 5 V single supply operation, 66 to 185 mV/A output sensitivity, and nearly zero magnetic hysteresis. In its operation, the applied current flowing through the copper conduction path generates a magnetic field which the Hall IC converts into a proportional output voltage [35]. The output voltage is proportional to the AC or DC currents being measured [35]. In this project, the 30 A module is used to measure the DC current from the solar PV system. This module is able to measure current values from 0 A to 30 A. Also, with this module, 0 A corresponds to 2.5 V, while 30 A corresponds to 5 V. These parameters, as well as the parameters of the ESP32 Thing ADC Pins are taken into consideration in writing the ESP32-Arduino code to measure the PV current with the sensor. With respect to the physical connections, the pull-down or step-down resistors arrangement described in "Sensors" above is used to match the 5V signal requirement of the Current Sensor to the 3.3 V signal capability of the ESP32 Thing ADC Pins. The step-down resistors arrangement showing the connection of the sensor to the ESP32 Thing is shown in Figure 3, and the voltage divider equation is shown in Equation (1). As can be seen, it's VCC pin is powered with the 5 V supply from the Breadboard, the OUT pin is connected via the pull-down resistors to the Analog pin 32 on the ESP32 Thing micro-controller, and its ground (GND) pin is connected to the GND pin on the Breadboard while the two Input pins are connected in series to the PV system to measure the DC current flowing through the system.

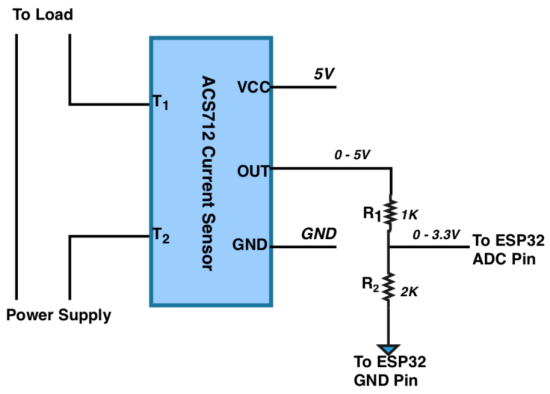

Figure 3. ACS712 step-down resistors connection.

$$V_{out} = \frac{R_2}{(R_1 + R_2)} \times V_{in} \quad (1)$$

where V_{out} = ESP32 ADC Voltage (3.3 V), and V_{in} = Sensor Input Voltage from the Breadboard (5 V), while and R_1 and R_2 are calculated to match these voltage values using the voltage divider equation above.

4.1.2. MH Electronic Voltage Sensor Modules

This is a low-cost analog voltage sensor capable of detecting supply voltages in the range of 0.025 V to 25 V. The sensor uses the concept of voltage divider to measure voltage. This voltage divider is a series connection of a 30 K resistor and a 7.5 K resistor. Its operating voltage range is 3.3 V to 5.0 V, and the voltage analog resolution is 0.000806 V for a 12-bit ADC [36]. In our setup, two voltage sensors are used; one of the voltage sensors is connected in parallel to the PV system to measure the voltage across it while the second one is connected in parallel across the lead acid battery system to measure the storage battery voltage. For the first voltage sensor, PIN S is connected to Analog PIN 34 on the ESP32, PIN—is connected to a GND pin on the ESP32 while its GND and VCC pins are connected in parallel across the PV panel output to measure the voltage across the PV system (PIN + on the sensor is not used). For the second voltage sensor, PIN S is connected to Analog PIN 35 on the ESP32, PIN—is connected to a GND pin on the ESP32 while its GND and VCC pins are connected (after the MPPT module) in parallel across the battery to measure the battery voltage (PIN + on the sensor is not used).

4.2. ESP32 Thing Micro-Controller (RTU)

The ESP32 Thing, manufactured and supplied by SparkFun Electronics, is a comprehensive development platform. It is a Wi-Fi compatible micro-controller, it supports Bluetooth Low-Energy (i.e., BLE, BT4.0, Bluetooth Smart), and it has nearly 30 Input/Output (I/O) pins [37]. According to the manufacturer, it is called the "Thing" because it is the perfect foundation for Internet of Things projects [37]. It is one of the most unique low-cost (about $20 CAD), low-power (about 0.5 W) micro-controllers available on the market today. The board can be powered with either a 5 V USB power supply or with a single-cell lithium-polymer (LiPo) battery, and its operating signal voltage range is 2.2 V to 3.6 V. The I/O pins of the ESP32 Thing board are only 3.3 V tolerant, hence the need for the level shifting of the connected 5 V current sensor explained earlier. Figure 4 shows a picture of the SparkFun ESP32 board while Figure 5 shows a summary of the hardware specifications of the board [37].

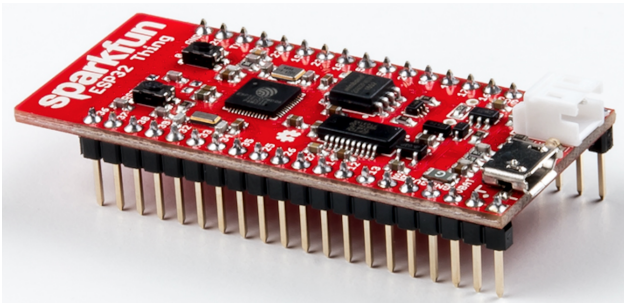

Figure 4. Image of the SparkFun ESP32 Thing board.

The ESP32 Thing micro-controller is programmed with an Arduino software integrated development environment (IDE). The Arduino IDE allows one to write the desired programs and upload them to the board via a USB cable. Arduino programs are called Sketches, and its language is merely a set of C/C++ functions [37]. In this project, the ESP32 Thing is hooked to a Breadboard, and the current and voltage sensors are connected to it as described in the "Sensors" section above. First, an Arduino sketch to measure the voltage and DC current from the PV system, and calculate the PV power output from the voltage and current values, as well as to separately measure the storage battery voltage via the sensors is written in the Arduino IDE and uploaded to the board. The measured and calculated values are displayed on the Serial Monitor of the Arduino IDE using the specified Baud Rate.

Secondly, having uploaded the program (sketch) into the ESP32 Thing board, the board is powered with a 5 V USB power cable.

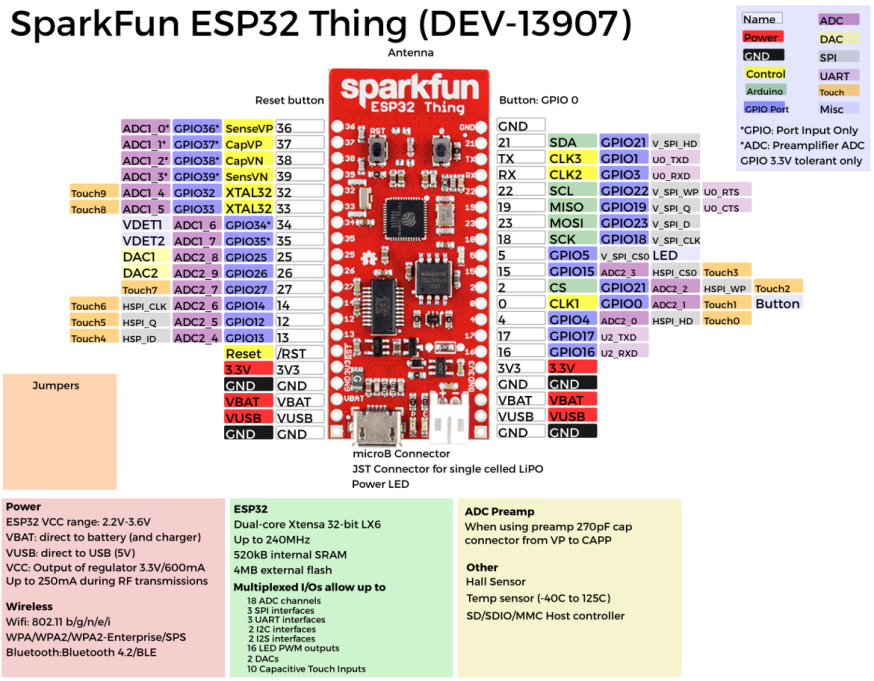

Figure 5. Hardware and peripherals specification summary of the ESP32 Thing.

4.3. Raspberry Pi Micro-Controller

The Raspberry Pi 2 model B used in this project is a 85 × 56 mm single board computer (micro-computer) device with BCM2836 quad core (4 processors in one chip) ARMv7 processor. It is a low-cost chip and it has the following properties which make it robust and suitable for the proposed SCADA system design [38,39]:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM
- One hundred Base Ethernet
- Four USB ports
- Forty GPIO pins
- Full HDMI port
- Combined 3.5 mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core.

In this work, the Thinger.IO IoT Server is installed and configured on the Raspberry Pi and the Raspberry Pi is connected to the other components in either of the two configurations described earlier.

Wi-Fi Router (Communication Channel)

In the proposed SCADA system, Wi-Fi serves as the communication channel between the ESP32 Thing (RTU), and the Thinger.IO IoT Server (MTU). This local Wi-Fi network is created using a D-Link Router (DI-524 Airplus G). It is a high speed router with 54 Mbps data transfer rate, 802.11b/g wireless protocol, and it is IEEE 802.11 standards compliant. The router has one WAN port and four LAN ports, and its operating frequency band is 2.4 GHz. Since the ESP32 Thing can implement TCP/IP, full 802.11b/g/e/i WLAN MAC protocol and Wi-Fi direct, the router is configured to setup the

needed local Wi-Fi network for transmitting the sensor data from the ESP32 Thing to the Thinger.IO IoT Server by using the server IP address to identify the platform. For the Wi-Fi access control and security purposes, the ESP32 Thing connects to the router using the Wi-Fi network name (SSID) and the assigned password.

4.4. Thinger.IO Local Server IoT Platform

Thinger.IO is a powerful open source platform for the Internet of Things (IoT). It supports Representational State Transfer (REST) Application Programming Interface (API) which enables controlling and reading of smart devices [40]. Representational State Transfer (REST) is an architecture based on web standards which uses HTTP protocol for communication. This protocol enables interoperability among the different machines on the internet by treating each and every component as a resource, such that each resource can be accessed by a common interface utilizing the general HTTP methods like OPTIONS, GET, PUT, POST, and DELETE, etc., [22–24,40]. The specific response of each resource is gotten in JSON/XML format [40]. Some of the important features of the protocol which make it possible to connect and manage IoT devices are automatic discovery of API, and bandwidth savings [24]. Thinger.IO is wholly supported by GitHub (the popular open source development platform) [40].

Thinger.IO IoT platform supports the integration of Arduino compatible hardware (any board that can be programmed with Arduino IDE, such as Arduino + Ethernet, Arduino + Wi-Fi, ESP32/8266/13, NodeMCU, TC CC320, etc.), Linux-powered devices like the Raspberry Pi, Intel Edison or any other Linux computer running Ubuntu or MacOS, and the ARM Mbed platform and compatible devices. The IoT platform has a Cloud Console with a beautifully designed front-end where a user can manage the connected devices and visualize the device information in the cloud [40]. To start an IoT project in Thinger.IO, the first step is to create devices by adding the device parameters which will grant access to connect the devices to the Thinger.IO account. Any device on the platform must be registered to have access to the cloud, and each device is identified by its unique identifier and credentials, such that an infinite number of devices can be added to the cloud platform without one device interfering with the other (Figure 7). Once a user creates an account on the Thinger.IO cloud, access token is obtained from the user's Username and Password, and this access token grants authorization access to the account resources of the connected devices. On the Cloud Console is a Statistics section where a user can see some basic information about the account such as the number of connected devices, endpoints, data buckets, statistics about device consumption in terms of sent and received data, as well as a Google Map of the approximate current locations of the connected devices (Figure 12). In the Cloud Console, an operator can add or remove devices, create real-time dashboards, access the device API, and perform other device and data management operations [22–24,40]. Also, the left side of the Statistics screen has a main menu with all the platform features needed to build IoT projects (Figure 6) [40].

One unique feature of the Thinger.IO IoT platform is that it allows an operator to discover the resources defined in the connected devices. A resource, in this case, can be a sensor reading like current, voltage, temperature, humidity, pressure, or any actionable element like a light, a relay, a motor, etc. Once a device is connected to an account, an operator can access its resources and explore the API REST endpoints using the API Explorer which is accessible over the Device Dashboard by clicking on a small blue button called Device API (Figure 6). Essentially, any device resource is like a callback function that can be called (on demand) through a REST API. On the Thinger.IO platform, four different types of resources can be defined, one for input (sending data to the device), one for output (sending data or information from the device), one for input/output (sending and receiving information in one call), and lastly, a callback resource which can be executed without sending or receiving information [40]. From the API perspective, the input and output data can be any JavaScript Object Notation (JSON) document [40].

Thinger.IO has a Local Server option where a user can purchase (one-off) the hardware or hardware installation ISO image and install it on a standalone or networked machine for proper management. It also has a web-based server option which can be accessed using its URL just like every other web-application [40]. For example, the authors in [22–24] used the web-based server option. Both the web-based and local server options have IoT capability as the data stored in them can be accessed remotely with an internet enabled computer or with an internet enabled phone via Thinger.IO mobile app or phone browser. However, the locally-installed server option is more secure as the user has a better control of the server and stored data for security purposes [40]. In this paper, the local server option is used. The low-cost Thinger.IO Raspberry Pi ISO Image (about $15.00 CAD) has been purchased, installed and configured on a Raspberry Pi machine.

Thinger.IO IoT platform allows a user to create all kinds of real-time visualization dashboards and charts for remote monitoring, as well as supports monitoring and control via emails, HTTP requests, etc., using endpoints. The platform also has free Android and iOS Apps that can be downloaded from Google Play Store and Apple Store respectively. Any of these Apps can be connected to a registered device in the cloud by scanning the QR barcode of the device. For example, the Thinger.IO Mobile App on my iPhone is connected to the ESP32 device on the local server by scanning the QR barcode of the device in the cloud as shown in Figure 6. This means that in addition to the cloud console monitoring and control features provided by the platform, an operator can also check the status of the connected devices, visualize and update output resources, edit and post input resources, as well as run resources anywhere in the world by using the Mobile Apps [40].

In this work, the Thinger.IO local server receives the PV voltage, PV current, measured PV power, and storage battery voltage from the connected ESP32 device at the Cloud Console, and automatically logs the data to the created Dashboards and Data Buckets at the specified data transfer rates. This means that an operator can either remotely visualize (monitor) the received data at the Cloud Console by clicking on the Device API, or by checking the Dashboards and Data Buckets. An operator can also initiate Supervisory Control actions by setting up and using the endpoint features of the platform. Here, an endpoint is defined as a target destination that can be called by the connected devices to perform any action, such as sending an email, sending SMS, calling a REST API, interacting with IFTTT (If This Then That), calling a device from a different account, or calling any other HTTP endpoint [24,40]. Figures 6 and 7 show a visual of the Thinger.IO Cloud Console and Console Dashboards respectively.

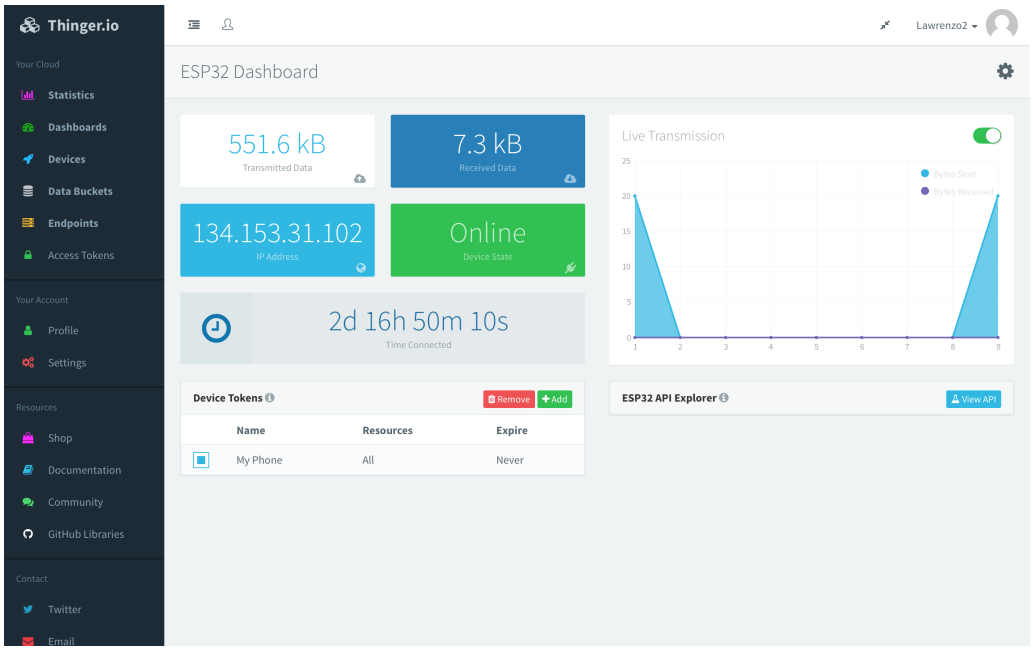

Figure 6. Thinger.IO Cloud Console.

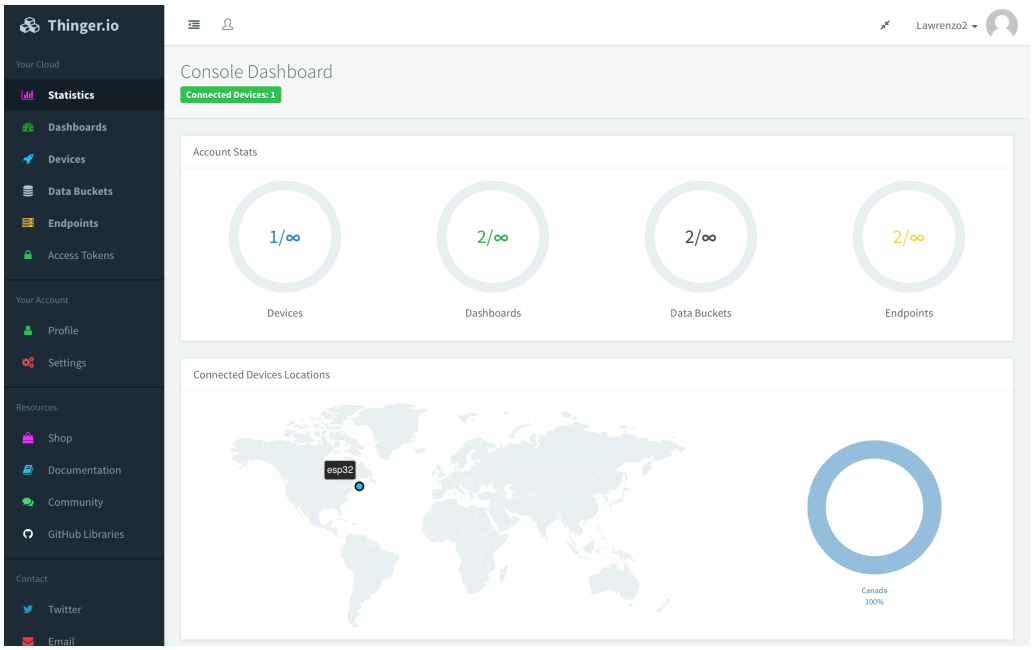

Figure 7. Thinger.IO Console Dashboard.

4.5. MUN ECE Laboratory PV System Overview

The Memorial University Electrical and Computer Engineering Laboratory photovoltaic system is made up of 12 Solar Panels covering a total area of 14 square meter and producing about 130 W and 7.6 Amps each. Two modules are connected in parallel such that it contains six sets of 260 W, and 14 A each. It has Maximum Power Point Tracking (MPPT) system to ensure that maximum power is captured from the solar panels under all operating conditions, and lead acid electrical battery system is connected to the MPPT to store the energy from the solar panels for use during prolonged extreme weather conditions.

In this project, the SCADA system is set up to acquire the PV voltage, PV current, the measured PV power from the voltage and current values, as well as the storage battery voltage, for remote monitoring and supervisory control. Here, only one set of the modules (about 260 W, and 14 A output) is used for testing purposes.

5. Implementation Methodology

In implementing the proposed low-cost, open source SCADA solution, the analog sensors are connected to the Solar PV System to collect the required data from the system, and the Sparkfun ESP32 Thing micro-controller is programmed with Arduino IDE to receive these sensor data, display them on the Arduino IDE Serial Monitor, and then send them via the locally configured Wi-Fi network to the locally installed Thinger.IO IoT server platform for remote monitoring and supervisory control. On the Thinger.IO Cloud Console, HMI Dashboards (Graphical User Interface, GUI), are created by an operator for data visualization, remote monitoring and supervisory control. The pseudocode for the implementation methodology is shown in Algorithm 1 below, and the appearance of the received data on the Thinger.IO server which is the JSON format described earlier (Name: Value) is shown in Figure 8.

Algorithm 1: Data Logging Algorithm:

Initialization;
1. Read Sensor Values on Analog Pins 32, 34 and 35, and Calculate Values for Pins 32 × 34;
2. Display the above Values on Arduino IDE Serial Monitor;
3. Connect to Local Wi-Fi Network with Wi-Fi Name and Password;
4. Connect to Thinger.IO Local Server IP Address;
5. Identify the specified Thinger.IO Account Name, Device ID and Credentials;
6. Post Sensor Data to the specified Thinger.IO Device;

while *Thinger.IO Server Acknowledges Data Receipt* **do**
    7. Display Sensor Data on Thinger.IO Cloud Console, and;
    8. Display "Ok" on Arduino IDE Serial Monitor;
    if *No Data Receipt Acknowledgement from Thinger.IO Server* **then**
        9. Display Debug/Error Message on Arduino IDE Serial Monitor;
    else
        10. Go to Step 1;
    end
end

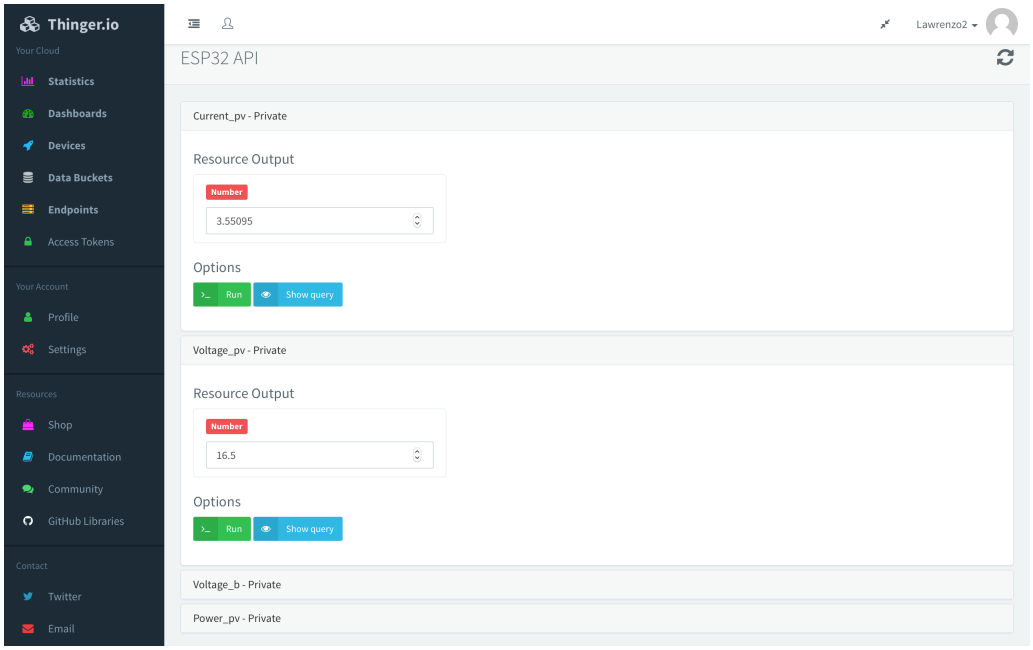

Figure 8. Thinger.IO received data page.

6. Prototype Design

Using the described hardware components and the operational principles of each of the components, the proposed SCADA system is designed and implemented as shown in Figure 9. As shown in the figure, the analog sensors, the pull-down resistors arrangement, and the ESP32 Thing micro-controller are connected together on a Breadboard. The 3.3 V voltage signal needed for the current sensor to match the 3.3 V requirement of the ESP32 ADC pins is acquired from the Breadboard with the pull-down resistors arrangement shown while the 5 V power supply for the ESP32 Thing board is provided using a 5 V USB power supply. The Wi-Fi Router and the Raspberry Pi-hosted Thinger.IO local server IoT platform are both placed in the building, and integrated into the system in two different configurations. For the first configuration, the Raspberry Pi is placed at a different office in the building and connected to MUN network via an RJ45 Ethernet cable. However, for the second configuration, the Raspberry Pi is placed close to the setup and connected to one of the LAN ports of the Wi-Fi Router.

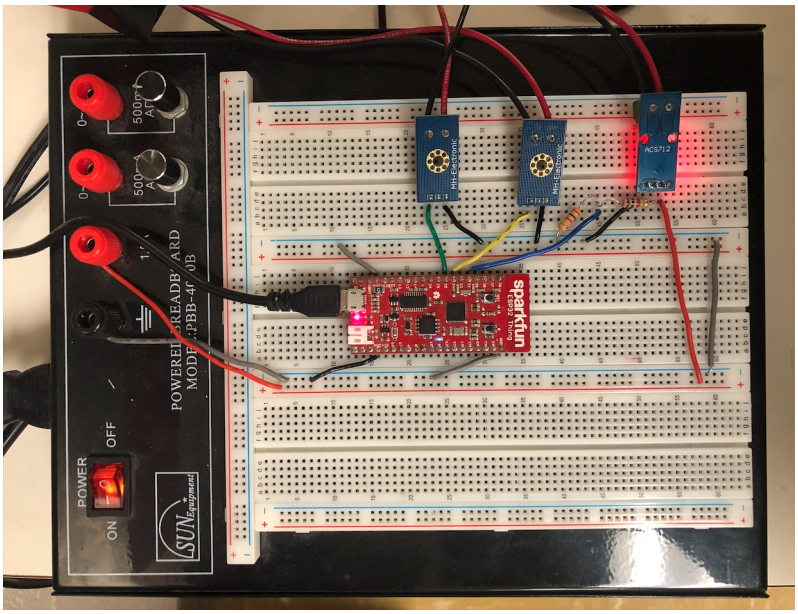

Figure 9. Hardware implementation of the proposed SCADA system.

7. Experimental Setup of the Proposed SCADA System

In order to test the proposed SCADA system solution, the implemented hardware for each of the two configurations is setup in MUN ECE Laboratory as shown in Figure 10. The inputs of the current and voltage sensors are connected to the PV and battery systems to acquire the PV data using electrical cables, and the sensor outputs are connected via cables to the ESP32 Thing to capture and parse the acquired data to the Thinger.IO IoT Server.

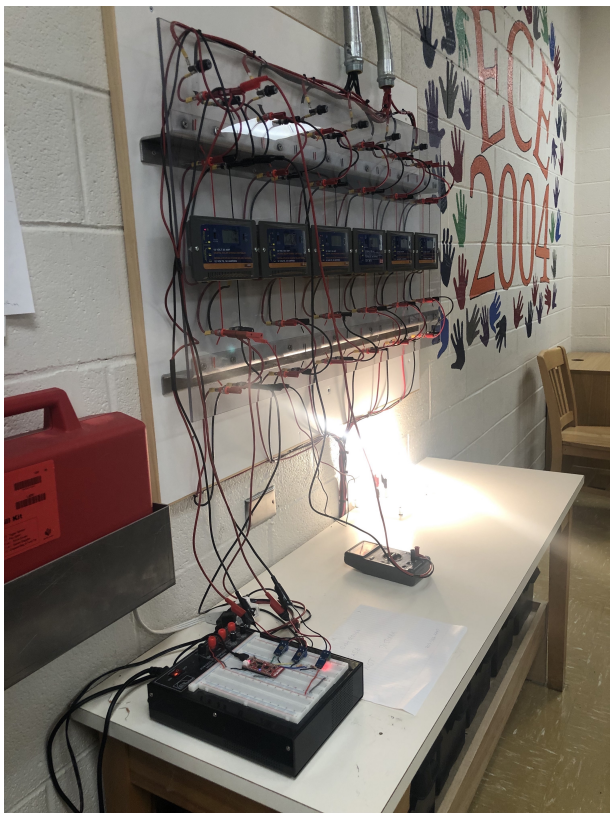

Figure 10. Experimental setup of the proposed SCADA system.

8. Testing and Results

As described above, each of the two configurations of the proposed low-cost, open source SCADA system solution was setup in the MUN ECE Laboratory to acquire the PV system data, and to parse the acquired data to Thinger.Io local server IoT platform for remote monitoring and supervisory control. A flow chart of the data acquisition, processing, visualization and supervisory control process from the sensors to the Thinger.IO server platform is shown in Figure 11.

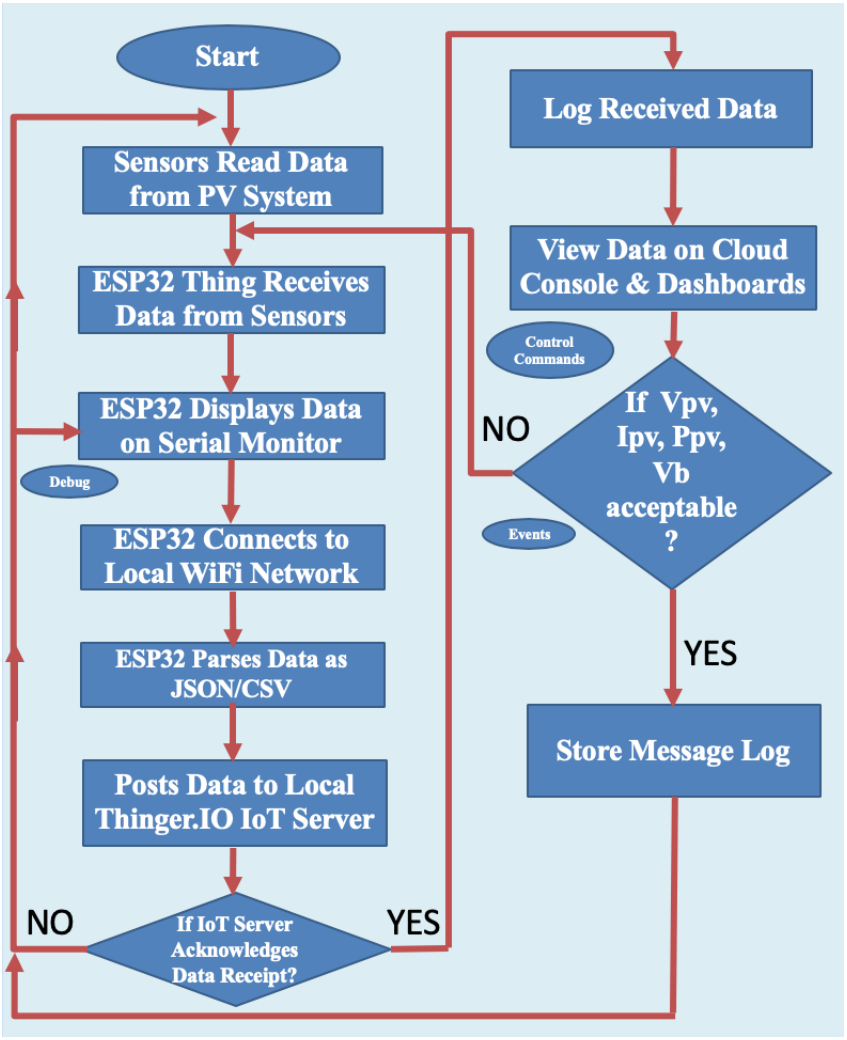

Figure 11. Flow chart of the SCADA system solution.

Results

The Thinger.IO local server IoT platform was configured, and using both hardware configurations A and B described earlier and the developed ESP32-Thinger.IO program (Pseudocode shown in Algorithm 1), the acquired sensor data were posted to the Thinger.IO server platform, and received at the Cloud Console in JSON format (Name: Value), and through the "View API" menu, an operator could view the received data directly on the Cloud Console. Having received the PV data at the cloud console, dashboards and data buckets were created and configured such that the received data were automatically logged to both the dashboards and data buckets for remote monitoring and supervisory control via computers and mobile devices.

In the first configuration (A), authorized users on MUN Network can view the data on Thinger.IO platform using desktops, laptops and mobile browsers pointed to Thinger.IO local server IP Address. Authorized internet users outside MUN Network can also view the data on the platform if the

Thinger.IO port is left open on MUN network. In the second configuration (B), only the authorized users connected to the Local Wi-Fi Network can view the data on the Thinger.IO platform using Wi-Fi enabled machines and the IP address of the server. Also, in both configurations, authorized personnel can view the stored data on the platform using the Thinger.IO Mobile App by scanning the QR barcode of the device on the platform. However, only the users connected to either of the networks can view the real-time data on the Mobile App. While the system was being tested, a digital multimeter was connected to each of the points of interest to measure the desired values. In both configurations, the acquired sensor data matched the values measured locally using the digital multimeter, with minor measurement errors. The system was tested for about a month, disconnected and reconnected on different days during this testing period to test its robustness as shown on the Data History window in Figure 12. Also, during testing, a load (an electric bulb) was connected to the battery to discharge it so that a significant amount of current can flow from the PV system across the MPPT to recharge the battery. Various dashboards were created on the Thinger.IO Server IoT platform to log the real-time PV data for remote monitoring and supervisory control, and from the dashboards, variations in the acquired sensor values were seen depending on the prevalent weather conditions affecting the PV system at the time of testing. As shown in Figures 13 and 14, the vibrations in the real-time values were due to the frequent changes in the weather conditions in St. John's during the testing. Furthermore, by clicking on the GPS at the bottom of Figures 13 and 14 (dashboards) on the cloud platform, the exact location of the connected devices can be seen. Aside from these two dashboards presented, the Thinger.IO Mobile App installed on an iPhone was connected to the local server by scanning the QR Barcode of the device for real-time monitoring.

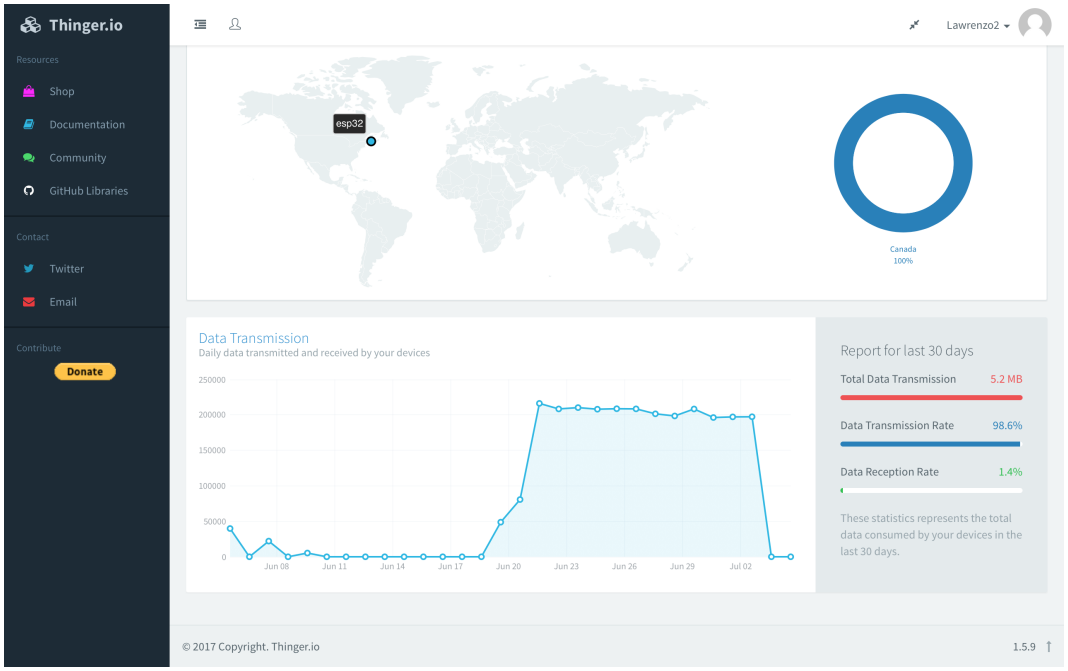

Figure 12. Logged data history on Thinger.IO Server.

In each of the two configurations, the received data are essentially the same, and only affected by the prevalent environmental conditions affecting the PV system (process plant). The major difference between the two configurations is the manner in which the received data can be accessed. In the first configuration, the real-time and stored data can be accessed with the server IP address as long as the user is connected to MUN network or if the Thinger.IO port is open for access via the user's private office or home internet connection. In the second configuration, only the users connected to the local Wi-Fi network, either wirelessly or via network cables connected to the LAN ports of the Wi-Fi Router, can view the real-time and stored data. Although the first configuration is more flexible, the second

configuration is more secure, and the decision to implement either configurations rests solely with the user. In addition, even though the Wi-Fi coverage distance was limited to the building where the tests were carried out in our system, the Wi-Fi range can be extended in other applications requiring wider coverage distance by using a Wi-Fi repeater.

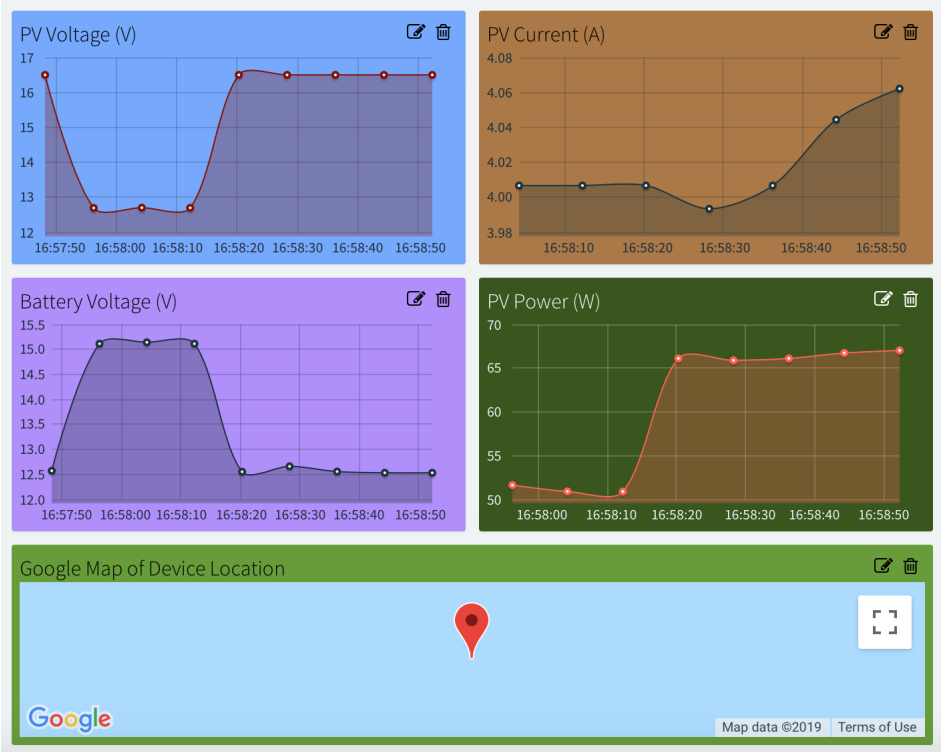

Figure 13. Created Dashboard (A) showing real-time data.

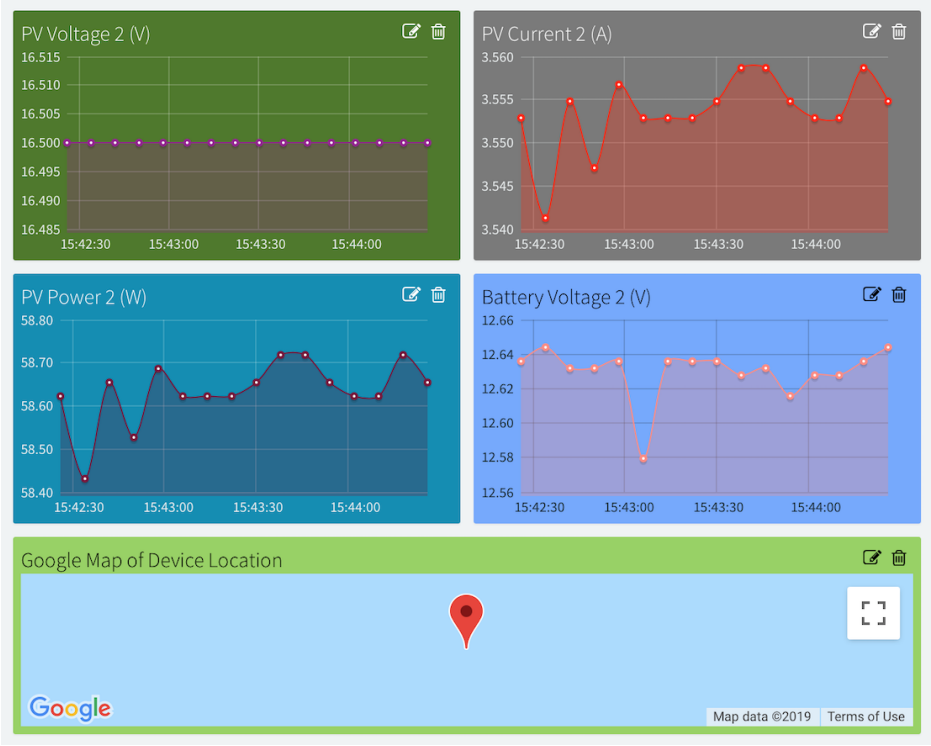

Figure 14. Created Dashboard (B) showing real-time data.

9. Discussion

Some of the key features of the designed low-cost, open source SCADA system are enumerated below:

- Internet of Things-based SCADA System: It is based on the Internet of Things SCADA architecture (the fourth, and most recent SCADA architecture), and it has the four basic elements of a SCADA system listed earlier in this paper. The PV System is the Process Facility (Plant) being managed, the Current and Voltage Sensors are the Field Instrumentation Devices as they acquire the desired data from the PV system, the ESP32 Thing micro-controller acts as the Remote Terminal Unit (RTU) as it helps to receive and parse the acquired sensor data, and the Thinger.IO Local Server IoT Platform serves as the Master Terminal Unit (MTU) as it provides means of handling data processing and human machine interactions. The SCADA Communication Channel between the RTU and the MTU is Wi-Fi which is created using a Wireless Router.
- Low-Cost and Open Source: All the components of the proposed SCADA system are manufactured and supplied by different manufacturers (mix and match), they are readily available and are cheap. The components are also compatible with related process facilities and components from various vendors. Therefore, the consumer is not beholden to a single manufacturer/supplier which is one of the key features of an open source system [4,12,13]. Table 1 below shows the cost of each of the components and the overall cost of the designed SCADA system. As can be seen, the overall cost is just under $300 CAD. It is indeed a low-cost SCADA system solution compared to the available commercial SCADA system solutions which are in thousands of dollars [3,11].

Table 1. Bill of Materials.

S/N	COMPONENT	QTY	PRICE (CAD)
1	Thinger.IO RPi ISO Image	1	15.62
2	Raspberry Pi 2 B	1	45.95
3	ESP32 Thing	1	31.90
4	Current Sensor	1	5.25
5	Voltage Sensor	2	11.98
6	D-Link D1-524 Wireless Router	1	98.51
7	8GB SD Card	1	12.66
8	Miscellaneous (Breadboard, Resistors, Wires, Boxes, etc.)	1	70.00
	Grand Total:		**$291.87 CAD**

- Data Acquisition and Historic Storage: The SCADA system stores the received data and maintains data history [41] (Figure 12).
- Low-Power: The system uses low-power components. For example, the two major components of the system (Raspberry Pi (Thinger.IO Local Server) and ESP32 Thing) consume a combined total of 2.2 W while in operation, which is low. It should be noted that the power consumption values for all the components shown in Table 2 were measured simultaneously while the system was running. As shown in Table 2, the overall power consumption of the designed system while in operation is under 10 W. This power can further be reduced by eliminating the D-Link Wi-Fi Router and configuring the Raspberry Pi as a Wireless Access Point, and using a less power demanding breadboard as the breadboard only provides 5 V power supply to the VCC pin of the current sensor in this current setup.

Table 2. Power consumption of hardware components.

S/N	HARDWARE	POWER (W)
1	Raspberry Pi 2 B	1.7
2	ESP32 Thing (alone)	0.5
3	Breadboard (with Sensors, ESP32, Resistors, etc. connected)	3.3
4	D-Link D1-524 Wireless Router	4.4
	Total Power Consumption (less ESP32 alone):	**9.4 W**

- Monitoring: The system provides Dashboards for events and data monitoring via the Thinger.IO Cloud Console on web browsers and Thinger.IO Mobile Apps [42].
- Supervisory Control: The system enables an administrator to issue supervisory control commands to a local operator for critical actions in the events where the received data do not correspond to a predetermined value or expected range. The administrator also has the option of creating Endpoints on the platform for more supervisory control options.
- Reporting: The system presents reports to the administrator and key decision makers in the form of charts, data logs, and alarms (local, email, web, and mobile app).
- Security: Because the proposed solution here involves the use of a locally hosted cloud server on MUN network, data integrity and security measures such as access control, whitelists, authentication, authorization, firewalls, regular risk assessment, continuous monitoring and log analysis, updating and patching regularly, etc., can easily be taken by the administrator. Also, for more critical infrastructure monitoring and control such as traffic light system or oil and gas facilities monitoring applications, additional security measures such as hardware protection and data encryption on the SCADA communication channel might be necessary [29,30,43].
- Reliability and Availability: Although system reliability calculation is beyond the scope of this work, research has shown that SCADA system reliability and availability are affected by delayed or wrong operator decisions [31]. In this project, because the components are completely open source and readily available, and the fact that the main cloud server is locally installed and self-managed, it is easy for an operator or administrator to manage the system to ensure its continuous reliability and availability. However, this might not be the case in a proprietary SCADA system where the customer is beholden to a single vendor, and as such there could be time-lags in responding to customer's complaints since having a trained operator on stand-by 24/7 on the customer's site might not be feasible.
- Ease of Use: The designed SCADA system solution is easy to use as the Thinger.IO IoT Platform is very user-friendly.

10. Conclusions

In most industries all over the world, like the energy industry, critical assets are distributed over large geographical areas, sometimes in harsh environments such as deep offshore and swamps. While it may be necessary to have local means of managing the operations of these critical assets, it is equally important to have a reliable, flexible, cost-effective and sophisticated coordinated monitoring and control. Although SCADA systems have revolutionized the way these critical, complex and geographically distributed industrial systems are monitored and controlled, SCADA system designs and implementations have largely remained proprietary. However in many applications, for example in power systems, the energy storage systems such as inverters and batteries, power electronic converters, and other devices connected to the grid are from various manufacturers or vendors, which often result in compatibility issues between these existing infrastructures and the proprietary SCADA system solutions. Therefore, an open source SCADA system solution represents the most flexible SCADA solution as it allows a user to "mix and match" components and choose the most appropriate from several manufacturers and suppliers. Furthermore, proprietary SCADA system solutions are largely expensive, and while it may be affordable for big companies like most oil and gas producing companies, it is pricey for smaller companies with no enormous financial resources like most small

power companies, especially power companies into renewable generation systems. Therefore, an open source SCADA solution represents the most cost effective solution as the user is not beholden to a single vendor.

In this paper, a low-cost, open source SCADA system solution based on the Internet of Things (IoT) SCADA architecture, which is the most recent SCADA architecture, has been presented. The hardware design of the proposed SCADA system solution has been carried out, and the implemented system has the four basic elements needed in a SCADA system, including Field Instrumentation Devices (Current and Voltage Sensors), Remote Terminal Units (ESP32 Thing micro-controller), Master Terminal Units (Thinger.IO IoT Server Platform), and SCADA Communication Channel (Local Wi-Fi Network). In order to validate the functionalities of the designed SCADA system, it was setup and tested at the Memorial University Electrical and Computer Engineering Laboratory to acquire and remotely monitor a 260 W, 12 V Solar PV System data, as well as to initiate supervisory control activities whenever necessary. From the tests, the system was found to be capable of carrying out the desired functions of a SCADA system which include Data Acquisition, Networked Data Communication, Data Presentation, and Remote Monitoring and Supervisory Control. The power consumption of each of the components was measured while the system was in operation and the entire system was found to require low power for operation, less than 10 W. The overall cost of the system was also found to be below $300 CAD which is extremely low for such a critical solution. Furthermore, the acquired real-time PV data visualized at the Thinger.IO IoT Server Cloud Console and Dashboards were found to be within the same range as the data locally measured using the conventional digital multimeter. From information security and system reliability points of view, because the main cloud server is privately hosted and self-managed on own network, data security measures such as access control, authorization, authentication, whitelisting, firewalls, log analysis, etc., are easily implemented, and since the system is self-managed, it means that the operator or administrator is readily available to carry out these tasks, thereby ensuring the security, reliability and availability of the proposed system.

Although a PV system has been used as the process plant for testing purposes in this work, the designed SCADA system can also be applied in other industries and fields to remotely monitor and control critical infrastructures such as electric power generation, transmission and distribution systems, buildings, facilities and environments, oil and gas production facilities, mass transit systems, water and sewage systems, and traffic signal systems. However, it should be noted that the designed system for this particular application is not a plug and play turnkey solution or a one-size-fits-all system for the above possible applications as further measures will need to be taken to customize the system for any chosen application in order to guarantee its functionalities, reliability and security, which might increase the overall cost and power consumption presented in this application. For example, for outdoor deployment of the proposed SCADA system, options for boxes and shields will have to be considered for the electronic components to prevent damage due to moisture and wild animals. Also, to increase the security of the proposed SCADA system for very critical applications like oil and gas facilities monitoring and traffic signal control, additional security measures such as data encryption algorithms can be implemented on the ESP32 micro-controller side to encrypt the acquired sensor data before sending them to the Thinger.IO server, and at the server side, data decryption and further security measures can be carried out. These could relatively increase the overall cost and power consumption of the system.

11. Future Work

In the future, endpoints can be created on Thinger.IO platform to carry out more supervisory control events like sending an email, sending an SMS, calling a REST API, interacting with IFTTT (If This Then That), calling a device from a different account, or calling any other HTTP endpoint directly from the server to complement the current supervisory control strategy where an administrator has to notify a local operator of any abnormal variations in the received data for necessary actions. Also, to further reduce the overall power consumption which is already low, the D-Link Wi-Fi

Router can be eliminated, and the Raspberry Pi configured as a Wireless Access Point to provide the Wi-Fi connection needed for communication between the ESP32 Thing and the Thinger.IO Server. Furthermore, to increase the security of the proposed SCADA system, especially when the system is deployed in very critical applications like oil and gas facilities monitoring and traffic signal control, data encryption algorithms can be implemented on the ESP32 micro-controller side to encrypt the acquired sensor data before sending them to the Thinger.IO server, and on the server side, data decryption and further security measures can be carried out.

Author Contributions: L.O.A. carried out most of the research work, and he was supervised by M.T.I. during the research. M.T.I. also reviewed and corrected the Manuscript, provided the required components and contributed research ideas throughout the research.

Funding: This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Energy Storage Technology Network (NESTNet).

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 providing the necessary funds and the conducive environment to carry out this research. The authors would also like to acknowledge the technical and emotional supports of friends and families during the difficult period of carrying out this research work.

Conflicts of Interest: The authors declare that there is no conflict of interest regarding the publication of this paper. All the components and software used in the actualization of this work were selected on a professional basis.

Abbreviations

The following major abbreviations are used in this manuscript:

SCADA	Supervisory Control And Data Acquisition
IoT	Internet of Things
HMI	Human Machine Interface
PLC	Programmable Logic Controller
MTU	Master Terminal Unit
RTU	Remote Terminal Unit
FID	Field Instrumentation Device
REST	Representative State Transfer
MUN	Memorial University of Newfoundland
API	Application Programming Interface
GUI	Graphical User Interface
ADC	Analog-to-Digital Converter
I/O	Input/Output
HTTP	HyperText Transfer Protocol
QR	Quick Response
PV	Photovoltaic

References

1. Lu, X. Supervisory Control and Data Acquisition System Design for CO_2 Enhanced Oil Recovery. Technical Report No. UCB/EECS-2014-123. Master of Engineering Thesis, EECS Department, University of California, Berkeley, CA, USA, 21 May 2014.
2. Anderson, M. Supervisory Control and Data Acquisition. Available online: https://realpars.com/scada/ (accessed on 3 June 2019).
3. Industrial Automation and Control. Available online: https://www.schneider-electric.com/en/work/products/industrial-automation-control/ (accessed on 10 June 2019).
4. Abbey, L. Telemetry/SCADA Open Systems vs Proprietary Systems. Available online: https://www.abbey.co.nz/telemetry--scada-open-vs-proprietary-systems-2003.html (accessed on 21 June 2019).

5. Boyer, S.A. *SCADA: Supervisory Control and Data Acquisition*, 4th ed.; International Society of Automation: Research Triangle Park, NC, USA, 2009. Available online: https://automation.isa.org/files/chapters/SCADA-Supervisory-Control-and-Data-Acquisition-Fourth-Edition-Chapter-10.pdf (accessed on 9 July 2019).
6. Sheng, Z.; Mahapatra, C.; Zhu, C.; Leung, V.C.M. Recent Advances in Industrial Wireless Sensor Networks Toward Efficient Management in IoT. *IEEE Access* **2015**, *3*, 622–637. [CrossRef]
7. Medrano, K.; Altuve, D.; Belloso, K.; Bran, C. Development of SCADA using a RTU based on IoT controller. In Proceedings of the 2018 IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA), Concepcion, Chile, 17–19 October 2018; pp. 1–6. [CrossRef]
8. Al-Kuwari, M.; Ramadan, A.; Ismael, Y.; Al-Sughair, L.; Gastli, A.; Benammar, M. Smart-home automation using IoT-based sensing and monitoring platform. In Proceedings of the 2018 IEEE 12th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG 2018), Doha, Qatar, 10–12 April 2018; pp. 1–6. [CrossRef]
9. Aghenta, L.O.; Iqbal, M.T. Development of an IoT Based Open Source SCADA System for PV System Monitoring. In Proceedings of the 32nd IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2019), Edmonton, AB, Canada, 5–8 May 2019.
10. Fortino, G.; Savaglio, C.; Zhou, M. Toward opportunistic services for the industrial Internet of Things. In Proceedings of the 2017 13th IEEE Conference on Automation Science and Engineering (CASE), Xi'an, China, 20–23 Auguest 2017; pp. 825–830. [CrossRef]
11. Unique Automation Portfolio. Available online: https://new.siemens.com/ca/en/products/automation.html (accessed on 10 June 2019).
12. Almas, M.S.; Vanfretti, L.; Løvlund, S.; Gjerde, J.O. Open source SCADA implementation and PMU integration for power system monitoring and control applications. In Proceedings of the 2014 IEEE PES General Meeting/ Conference and Exposition, National Harbor, MD, USA, 27–31 July 2014; pp. 1–5. [CrossRef]
13. Thomas, M.S.; McDonald, J.D. *Power System SCADA and Smart Grids*; CRC Press: Boca Raton, FL, USA, 19 December 2017. [On-line]. Available online: https://books.google.ca/books?id=bAhEDwAAQBAJ (accessed on 23 June 2019).
14. Shabani, H.; Ahmed, M.M.; Khan, S.; Hameed, S.A.; Habaebi, M.H.; Zyoud, A. Novel IEEE802.15.4 Protocol for Modern SCADA communication systems. In Proceedings of the 2014 IEEE 8th International Power Engineering and Optimization Conference (PEOCO2014), Langkawi, Malaysia, 24–25 March 2014; pp. 597–601. [CrossRef]
15. Sajid, A.; Abbas, H.; Saleem, K. Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges. *IEEE Access* **2016**, *4*, 1375–1384. [CrossRef]
16. Rajkumar, R.I.; Alexander, T.J.; Devi, P. ZigBee based design of low cost SCADA system for industrial process applications. In Proceedings of the 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Chennai, India, 15–17 December 2016; pp. 1–4. [CrossRef]
17. Unde, M.D.; Kurhe, P.S. Web based control and data acquisition system for industrial application monitoring. In Proceedings of the 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, India, 1–2 August 2017; pp. 246–249. [CrossRef]
18. Merchán, D.F.; Peralta, J.A.; Vazquez-Rodas, A.; Minchala, L.I.; Astudillo-Salinas, D. Open Source SCADA System for Advanced Monitoring of Industrial Processes. In Proceedings of the 2017 International Conference on Information Systems and Computer Science (INCISCOS), Quito, Ecuador, 23–25 November 2017; pp. 160–165. [CrossRef]
19. Prokhorov, A.S.; Chudinov, M.A.; Bondarev, S.E. Control systems software implementation using open source SCADA-system OpenSCADA. In Proceedings of the 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Moscow, Russia, 29 January–1 February 2018; pp. 220–222. [CrossRef]
20. Avhad, M.; Divekar, V.; Golatkar, H.; Joshi, S. Microcontroller based automation system using industry standard SCADA. In Proceedings of the 2013 Annual IEEE India Conference (INDICON), Mumbai, India, 13–15 December 2013; pp. 1–6. [CrossRef]

21. Mononen, T.; Mattila, J. A low-cost cloud-extended sensor network for supervisory control. In Proceedings of the 2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM), Ningbo, China, 19–21 November 2017; pp. 502–507. [CrossRef]
22. Kodali, R.K.; Yerroju, S. Energy Efficient Home Automation Using IoT. In Proceedings of the 2018 International Conference on Communication, Computing and Internet of Things (IC3IoT), Chennai, India, 15–17 February 2018; pp. 151–154. [CrossRef]
23. Kodali, R.K.; Gorantla, V.S.K. RESTful Motion Detection and Notification using IoT. In Proceedings of the 2018 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 4–6 January 2018; pp. 1–5. [CrossRef]
24. Kodali, R.K.; Mahesh, K.S. Smart emergency response system. In Proceedings of the TENCON 2017—2017 IEEE Region 10 Conference, Penang, Malaysia, 5–8 November 2017; pp. 712–717. [CrossRef]
25. Piggin, R.S.H. Securing SCADA in the cloud: Managing the risks to avoid the perfect storm. In Proceedings of the IET and ISA 60th International Instrumentation Symposium 2014, London, UK, 24–26 June 2014; pp. 1–6. [CrossRef]
26. Mo, Y.; Chabukswar, R.; Sinopoli, B. Detecting Integrity Attacks on SCADA Systems. *IEEE Trans. Control Syst. Technol.* **2014**, *22*, 1396–1407. [CrossRef]
27. Yang, Y.; Xu, H.; Gao, L.; Yuan, Y.; McLaughlin, K.; Sezer, S. Multidimensional Intrusion Detection System for IEC 61850-Based SCADA Networks. *IEEE Trans. Power Deliv.* **2017**, *32*, 1068–1078. [CrossRef]
28. Rosa, L.; Freitas, M.; Mazo, S.; Monteiro, E.; Cruz, T.; Simões, P. A Comprehensive Security Analysis of a SCADA Protocol: From OSINT to Mitigation. *IEEE Access* **2019**, *7*, 42156–42168. [CrossRef]
29. Iqbal, A.; Iqbal, M.T. Low-Cost and Secure Communication System for SCADA System of Remote Microgrids. *J. Electr. Comput. Eng.* **2019**, *2019*, 1986325. [CrossRef]
30. Alves, T.; Das, R.; Morris, T. Embedding Encryption and Machine Learning Intrusion Prevention Systems on Programmable Logic Controllers. *IEEE Embed. Syst. Lett.* **2018**, *10*, 99–102. [CrossRef]
31. Nasr, P.M.; Yazdian-Varjani, A. Toward Operator Access Management in SCADA System: Deontological Threat Mitigation. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3314–3324. [CrossRef]
32. Falco, G.; Caldera, C.; Shrobe, H. IIoT Cybersecurity Risk Modelling for SCADA Systems. *IEEE Internet Things J.* **2018**, *5*, 4486–4495. [CrossRef]
33. Yang, Y.; McLaughlin, K.; Sezer, S.; Littler, T.; Im, E.G.; Pranggono, B.; Wang, H.F. Multiattribute SCADA-Specific Intrusion Detection System for Power Networks. *IEEE Trans. Power Deliv.* **2014**, *29*, 1092–1102. [CrossRef]
34. Endi, M.; Elhalwagy, Y.Z.; Hashad, A. Three-layer PLC/SCADA system Architecture in process automation and data monitoring. In Proceedings of the 2010 the 2nd International Conference on Computer and Automation Engineering (ICCAE), Singapore, 26–28 February 2010; pp. 774–779. [CrossRef]
35. Current Sensor ICs. Available online: https://www.allegromicro.com/en/Products/Current-Sensor-ICs/Zero-To-Fifty-Amp-Integrated-Conductor-Sensor-ICs.aspx (accessed on 24 June 2019).
36. 25V Voltage Sensor Module. Available online: https://hobbycomponents.com/sensors/389-25v-voltage-sensor-module (accessed on 24 June 2019).
37. Jimblom. ESP32 Thing Hookup Guide. Available online: https://learn.sparkfun.com/tutorials/esp32-thing-hookup-guide (accessed on 31 May 2019).
38. Shete, R.; Agrawal, S. IoT based urban climate monitoring using Raspberry Pi. In Proceedings of the 2016 International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, India, 6–8 April 2016; pp. 2008–2012. [CrossRef]
39. Raspberry Pi 2 Model B. Available online: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (accessed on 3 June 2019).
40. Thinger.io—Documentation. Available online: http://docs.thinger.io (accessed on 27 June 2019).
41. Bagri, A.; Netto, R.; Jhaveri, D. Supervisory Control ad Data Acquisition. *Int. J. Comput. Appl.* **2014**, *102*, 1–5.
42. Tautz-Weinert, J.; Watson, S.J. Using SCADA data for wind turbine condition monitoring—A review. *IET Renew. Power Gener.* **2017**, *11*, 382–394. [CrossRef]

43. Grilo, A.M.; Chen, J.; Díaz, M.; Garrido, D.; Casaca, A. An Integrated WSAN and SCADA System for Monitoring a Critical Infrastructure. *IEEE Trans. Ind. Inform.* **2014**, *10*, 1755–1764. [CrossRef]

Sample Availability: All the materials, data and software used in this project, including the ESP32-Arduino Code used in the data logging phase, are available from L.O.A.