IoT Farm Energy Management (Solar) System: Enabling Real-time Monitoring of Solar Charge Controller via IoT Devices

Bumjun Oh
Psychology
Chung-Ang University
Seoul, Republic of Korea
dhsys199@gmail.com

Hanbyeol Lee
Software Engineerng
Chung-Ang Universiy
Seoul, Republic of Korea
yhb18340@gmail.com

Hyeongyu Lee
Russian Language and Literature
Chung-Ang University
Seoul, Republic of Korea
snicket@cau.ac.kr

Jiacheng Wang
Computer and Information Technology
Purdue University
West Lafayette, IN, USA
wang4499@purdue.edu

Sangje Jeong
Software Engineering
Chung-Ang University
Seoul, Republic of Korea
jun915350@gmail.com

Minji Lee
Computer and Information Technology
Purdue University
West Lafayette, IN, USA
lee3450@purdue.edu

Eric Matson
Computer and Information Technology
Purdue University
West Lafayette, IN, USA
ematson@purdue.edu

***Abstract*— Amongst renewable energy sources, solar power energy is gaining popularity thanks to low carbon dioxide emission and increasing efficiency. Populations who are taking part in agricultural activities tend to rely on electricity generated from a long distance and solar power energy is not an exception. Photovoltaic(PV) solar panels are one of the ways to harness solar power into renewable energy with only a major initial cost but low maintenance. However, due to the tendency of solar energy systems being far away from the electricity users, energy health monitoring in real-time and maintaining recovery from faults is the key goal. The emerging Internet-of-Things(IoT) This paper describes implementing the integration of IoT devices with manufactured solar charge controllers that already have functions to monitor the health of a solar power system. The potential plan is to build a LoRa module that plays multiple roles like data collecting from solar charge controllers and data transmitting across long-distance. Uploaded data drawn from the controller will be visualized using web platforms with the assistance of Node-Red and MongoDB. Hence, the owner of the solar power system will be able to have access to real-time information wherever the Internet is available.**

***Index Terms*— AWS, Internet of Things, Modbus protocol, MongoDB, Node-Red, RS485, database, server, solar charge controller**

I. Introduction

The purpose of this project is to build a module that monitors the real-time status of an off-grid solar energy system via communication with a solar charge controller and LoRa device to send data through a long distance. The main use of the application is in rural areas with difficult access to solar energy due to distance.

II. Literature review

Author *Ambuj Gupta*, *Rishabh Jain*, *Rakshita Joshi*, and *Ravi Saxena* [1], introduces an idea about IoT-based solar energy monitoring systems. The main aim of this system is to monitor and analyze solar setup, the range is from the smallest PV arrays to the biggest solar farms operating in the world. Based on modern world technology, everything could be accessible via the internet. Therefore, the system aims to collect data from the IoT platform which could help the user monitoring of the data in real-time. The whole process becomes real-time monitoring. The whole process used Raspberry Pi to receive data from Analog to Digital converter on the Lab View platform, and a DC transducer to converts an available DC input into the appropriate DC signal that could be received.

Suprita M. Patil, *M. Vijayalashmi*, and *Rakesh Tapaskar* [2], introduces a solar energy remote management system, the system could monitor the power and energy usage, and the main objective of this system is to use the Arduino to monitor the current and voltage value. The data stored in the cloud could be analyzed by using MatLab. The Arduino is used for reading the sensor values, Raspberry Pi used as a server, by connected Arduino and Raspberry Pi, the data information will display on the web through Raspberry Pi, for these data information that received will be upload into the cloud.

Mirsad Hyder Shah and *Nasser Hassan Abosaq* [3], explained an experimental work that using for real-time and low-cost monitoring systems for solar energy management. By using the Node-Red and NodeMCU, the system can monitor the current and voltage of the solar panel. Moreover, the paper also explained that the typical solar panel wastes about 15-20% of its max power potential due to magnetic declination. To solve this problem, the paper explained by correcting the tilt angle to improve the solar panel efficiency.

III. Environment Settings

A. Raspberry Pi Setting

The application is currently using Raspberry Pi 3 Model B [4] running the 32-bit Raspberry Pi OS. It has Python 3 and Python 3 IDLE for Python script editing and running. It is required to have various python modules and libraries to execute communication with the solar charge controller.

B. Photovoltaic Solar Panel

A photovoltaic solar panel (shortly PV panel) used in this application generates 30W(watts) with 18V output. It generates electricity with the solar cells attached to it and sends electricity through direct current(DC) electric wires.

Figure 1. Solar Panel

C. MPPT solar charge controller

The equipment that is used in this project is MPPT solar charge controller manufactured by a company named EPSOLAR [5], a model named XTRA3120N [6]. The controller receives electricity generated from the solar panels. Simultaneously, the controller either charges connected batteries or give output load for immediate use of electricity. The controller has a microcomputing unit(MCU) inside that monitors and saves real-time data that has been generated by the flow of electric current. The data are stored in 16-bit registers and single-bit coils. This equipment can have communications with other devices via RS485 serial communication with Modbus protocol. It uses a physical port that is widely used for internet connection, such as the RJ45 port for COMS.

D. 12V Lead-acid Battery

Solar charge controller needs to save power in a battery that has the compatible voltage input and ampere according to the controller. The controller can charge batteries with an input voltage of 38v. However, since the solar panel's optimal output voltage is 18v, a battery with a lower input volt rate is needed.

E. Improvised USB to RS485 cable(COMS cable)

The EPSOLAR company produces and sells communication cables that can allow users to connect between PCs and controllers. However, the project uses an improvised and economical way that can function identically to the official product. Here are the materials to make up the USB to RS485 cable:

1) CAT.5e: Category 5e ethernet cable should be peeled off on one end to make a connection with the USB to RS485 module which is described after.

2) USB to RS485 module: the module has several components that allow data flow from RS485 Modbus protocol to USB. The improvising process requires connecting CAT.5e wires number 4(RS-485-B) and number 6(RS-485-A)(green and blue respectively), to the screw terminal with 'A' and 'B' accordingly.

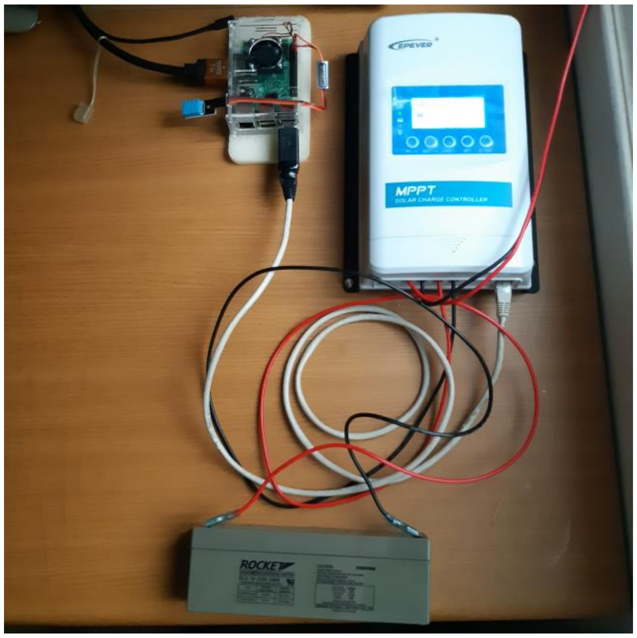
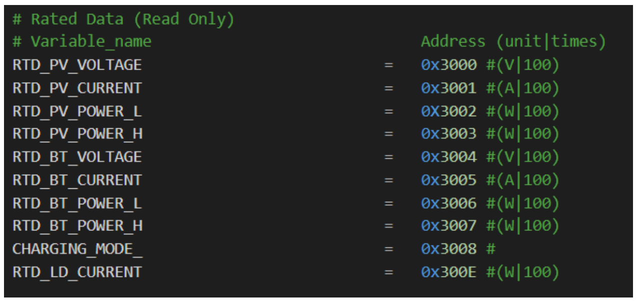


Figure 2. Raspberry Pi, Solar Charge Controller, Battery, and COMS cable

F. Python scripts for communication between the controller and Raspberry Pi

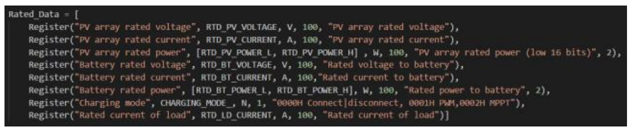
1) Register addresses: The solar charge controller uses 16-bit addresses and single-bit coils as a Modbus protocol. There are read input register, read and write holding registers, read and write coils and read discrete inputs. The datasheet of these addresses was saved in the script as constants for easier use.



```
# Rated Data (Read Only)
# Variable_name                               Address (unit|times)
RTD_PV_VOLTAGE                           =    0x3000 #(V|100)
RTD_PV_CURRENT                           =    0x3001 #(A|100)
RTD_PV_POWER_L                           =    0X3002 #(W|100)
RTD_PV_POWER_H                           =    0x3003 #(W|100)
RTD_BT_VOLTAGE                           =    0x3004 #(V|100)
RTD_BT_CURRENT                           =    0x3005 #(A|100)
RTD_BT_POWER_L                           =    0x3006 #(W|100)
RTD_BT_POWER_H                           =    0x3007 #(W|100)
CHARGING_MODE_                           =    0x3008 #
RTD_LD_CURRENT                           =    0x300E #(W|100)
```

Figure 3. Register addresses assignments to variables

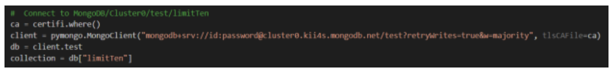
2) Register object arrays: Using the previous addresses, all related data such as variable name, unit, descriptions are saved to an instance of a class named 'Register'. Collection of those instances form arrays by their categories.



```
Rated_Data = [
    Register("PV array rated voltage", RTD_PV_VOLTAGE, V, 100, "PV array rated voltage"),
    Register("PV array rated current", RTD_PV_CURRENT, A, 100, "PV array rated current"),
    Register("PV array rated power", [RTD_PV_POWER_L, RTD_PV_POWER_H] , W, 100, "PV array rated power (low 16 bits)", 2),
    Register("Battery rated voltage", RTD_BT_VOLTAGE, V, 100, "Rated voltage to battery"),
    Register("Battery rated current", RTD_BT_CURRENT, A, 100,"Rated current to battery"),
    Register("Battery rated power", [RTD_BT_POWER_L, RTD_BT_POWER_H], W, 100, "Rated power to battery", 2),
    Register("Charging mode", CHARGING_MODE_, N, 1, "0000H Connect|disconnect, 0001H PWM,0002H MPPT"),
    Register("Rated current of load", RTD_LD_CURRENT, A, 100, "Rated current of load")]
```

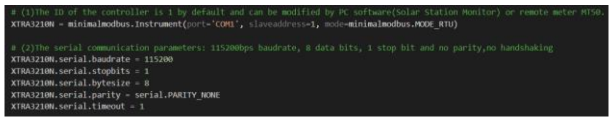
Figure 4. Arrays of Registers

3) Main: The main python script firstly checks the connection with the solar charge controller and if a connection is made, then makes another connection to the database server (MongoDB) to upload generated data in real-time. When the connection with the database server is finally done, the running script is ready to receive data from the controller and save individual data from each register to a dictionary. Iteration in the register arrays allows referring to different data representing different things like the voltages and currents of PV panels and batteries. Once the dictionary is filled with data that were requested, the dictionary is finally sent to the database with respective time stamps and object ids.



```
#  Connect to MongoDB/Cluster0/test/limitTen
ca = certifi.where()
client = pymongo.MongoClient("mongodb+srv://id:password@cluster0.kii4s.mongodb.net/test?retryWrites=true&w=majority", tlsCAFile=ca)
db = client.test
collection = db["limitTen"]
```

Figure 5. Connection to database



```
# (1)The ID of the controller is 1 by default and can be modified by PC software(Solar Station Monitor) or remote meter MT50.
XTRA3210N = minimalmodbus.Instrument(port='COM1', slaveaddress=1, mode=minimalmodbus.MODE_RTU)

# (2)The serial communication parameters: 115200bps baudrate, 8 data bits, 1 stop bit and no parity,no handshaking
XTRA3210N.serial.baudrate = 115200
XTRA3210N.serial.stopbits = 1
XTRA3210N.serial.bytesize = 8
XTRA3210N.serial.parity = serial.PARITY_NONE
XTRA3210N.serial.timeout = 1
```

Figure 6. Connection with solar charge controller via Modbus protocol

G. *AWS EC2 Instance for Deployment*

AWS EC2 Ubuntu Instance was used as a remote server. Amazon EC2 Instance is a virtual server in Amazon's Elastic Compute Cloud for running applications on the AWS infrastructure [7].

AWS is a comprehensive, evolving cloud computing platform. EC2 is a service that enables business subscribers to run application programs in the computing environment.

H. *Node-Red Application for Data Visualization*

Electricity, temperature, humidity, and other information had to be shown in detail by web application so that users can easily see the status of the solar power system.

Node-Red [8] is for data visualization which shows the data extracted from the database in real-time.

Node-Red is a flow-based development tool [9] for data visualization. It provides a web browser-based flow editor.

In Node-Red, the data can be uploaded through the Inject node like *Timestamp*. And the *Timestamp* controls to delay the time per 30 seconds in this project. Then the *function node*'s code and the *mongodb3 in* node are collecting all real-time data. Next function nodes and text nodes are collecting each of the data that fits the purpose.

Figure7. Amazon Web Service [10]

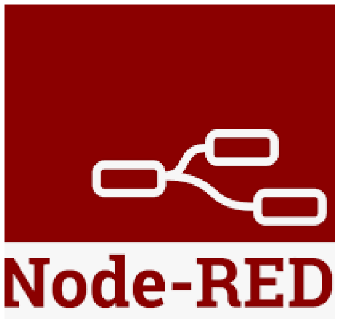


Figure 8. Node-RED

I. *MongoDB Database for Database*

Figure 9. MongoDB: Non-Sequel Database Service [11]

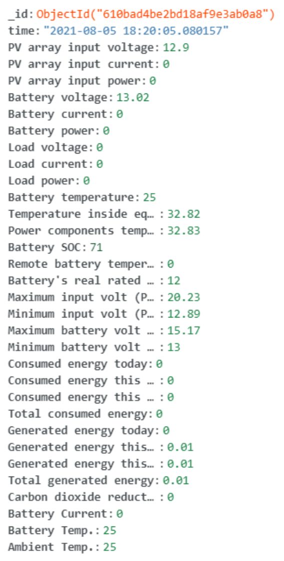
MongoDB [12] is an open-source, non-relational database management system that uses flexible documents instead of tables and rows to process and store various forms of data [13]. Solar Panel info coming from the solar controller is saved to MongoDB through Raspberry Pi.

Database saved in MongoDB is displayed by Node-Red Framework

IV. Data and Results

A. *Data from the solar charge controller*

Real-time data with the Real-time status of the controllers are requested and sent to Raspberry Pi and again sent to the database.



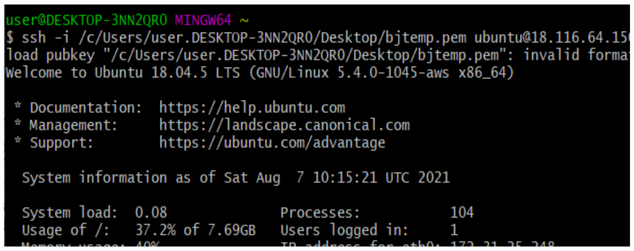
```
_id: ObjectId("610bad4be2bd18af9e3ab0a8")
time: "2021-08-05 18:20:05.080157"
PV array input voltage: 12.9
PV array input current: 0
PV array input power: 0
Battery voltage: 13.02
Battery current: 0
Battery power: 0
Load voltage: 0
Load current: 0
Load power: 0
Battery temperature: 25
Temperature inside eq… : 32.82
Power components temp… : 32.83
Battery SOC: 71
Remote battery temper… : 0
Battery's real rated … : 12
Maximum input volt (P… : 20.23
Minimum input volt (P… : 12.89
Maximum battery volt … : 15.17
Minimum battery volt … : 13
Consumed energy today: 0
Consumed energy this … : 0
Consumed energy this … : 0
Total consumed energy: 0
Generated energy today: 0
Generated energy this… : 0.01
Generated energy this… : 0.01
Total generated energy: 0.01
Carbon dioxide reduct… : 0
Battery Current: 0
Battery Temp.: 25
Ambient Temp.: 25
```

Figure 10. Data collection uploaded to MongoDB

B. *AWS EC2 Server Running*

The virtual server in Amazon's Elastic Compute Cloud runs applications on the AWS infrastructure.

http://18.116.64.150:1880 is the link to the Solar Energy Management System.



```
user@DESKTOP-3NN2QRO MINGW64 ~
$ ssh -i /c/Users/user.DESKTOP-3NN2QRO/Desktop/bjtemp.pem ubuntu@18.116.64.15
load pubkey "/c/Users/user.DESKTOP-3NN2QRO/Desktop/bjtemp.pem": invalid forma
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1045-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Aug  7 10:15:21 UTC 2021

  System load:  0.08              Processes:           104
  Usage of /:   37.2% of 7.69GB   Users logged in:     1
  Memory usage: 40%               IP address for eth0: 172.21.25.248
```

Figure 11. AWS EC2 Server Running

C. *Node-RED*

Interacting with MongoDB, Node-red shows the controller's Real-time data using AWS EC2 Instance as a Server.

The Node-RED pages consist of the main tracer and gauge. The main tracer shows the real-time data from the MPPT controller. Data mainly consist of Solar, Battery, DC Load, Controller, Energy Information. Real-time Voltage data of PV, Battery, DC Load are shown as Chart also so that users can compare them easily. The data from the controller are sent to MongoDB which is stored in this system.

On the other hand, *Gauge* shows external humidity and temperature data. Users can also see the real-time change of data with the chart below the page.

Displayed data is updated per 30 seconds, according to the timestamp. The most recent data is shown every 30 seconds.

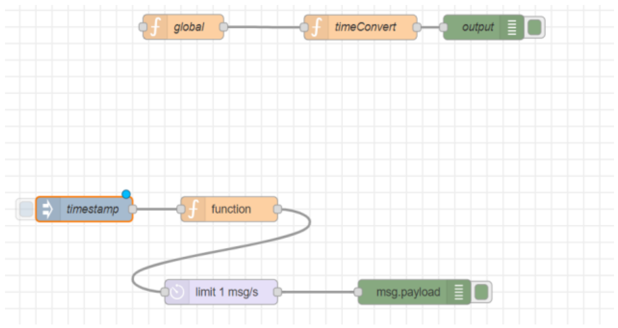
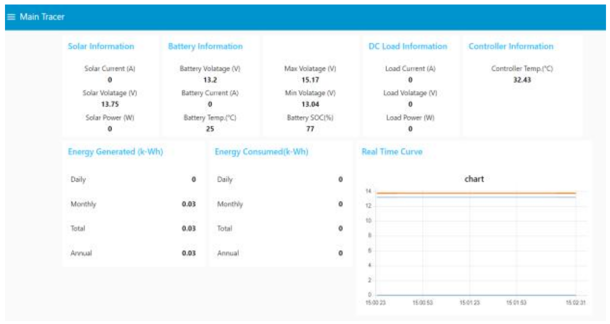


Figure 12. Node-RED coding example



≡ Main Tracer

Solar Information
Solar Current (A)
0
Solar Volatage (V)
13.75
Solar Power (W)
0

Battery Information
Battery Volatage (V)
13.2
Battery Current (A)
0
Battery Temp.(°C)
25

Max Volatage (V)
15.17
Min Volatage (V)
13.04
Battery SOC(%)
77

DC Load Information
Load Current (A)
0
Load Volatage (V)
0
Load Power (W)
0

Controller Information
Controller Temp.(°C)
32.43

Energy Generated (k-Wh)	
Daily	0
Monthly	0.03
Total	0.03
Annual	0.03

Energy Consumed(k-Wh)	
Daily	0
Monthly	0
Total	0
Annual	0

Real Time Curve

Figure 13. Real-time Solar Charge Controller Tracing

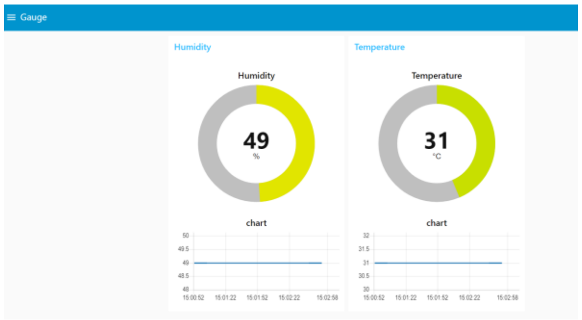


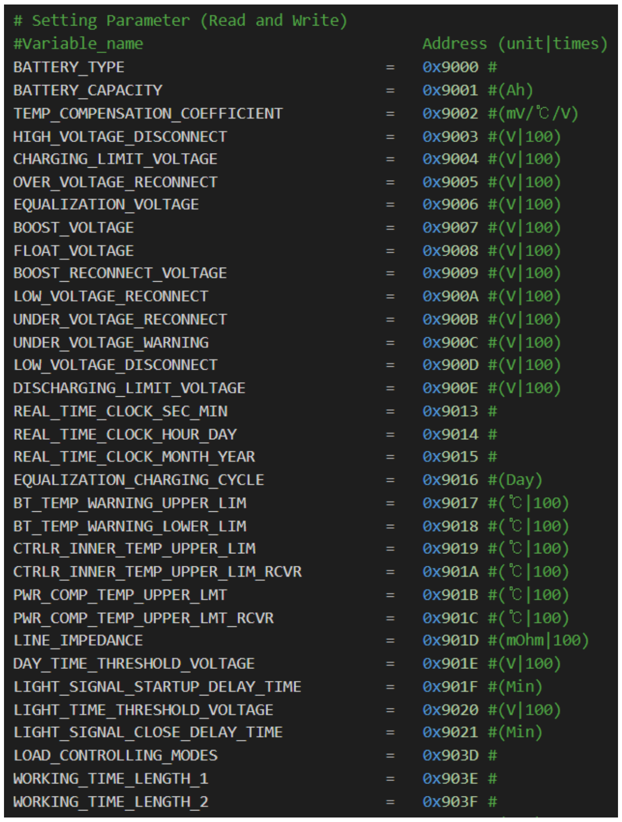
Figure 14. Real-time Temperature/Humidity tracing

V. DISCUSSION

Enabling communication between Raspberry Pi and EPEVER solar charge controller was a major challenge during the progress of this project. Thanks to the handful of resources on how to make an improvised communication cable, the stable data flow from the solar charge controller to the Raspberry Pi is possible. Despite the success of communication between two devices, some points could be considered as drawbacks and limitations.

First, although the applied Modbus protocol between the two devices can be applied to other solar charge controllers manufactured by the same company(EPSOLAR), there is no guarantee that this method can also be applied to other solar charge controllers from other companies.

Second, data can be not only read from the controller but also written to it. The current project only shows data results that are read from registers in the solar charge controller. However, according to the Modbus protocol [14] embedded in the controller, there are 16-bit registers called "Setting Parameters". These registers are not only readable, but users can set values to them for customization. For the better user experience and true management of solar power systems, further software development should be implemented in the future.



```
# Setting Parameter (Read and Write)
#Variable_name                                  Address (unit|times)
BATTERY_TYPE                               =    0x9000 #
BATTERY_CAPACITY                           =    0x9001 #(Ah)
TEMP_COMPENSATION_COEFFICIENT              =    0x9002 #(mV/℃/V)
HIGH_VOLTAGE_DISCONNECT                    =    0x9003 #(V|100)
CHARGING_LIMIT_VOLTAGE                     =    0x9004 #(V|100)
OVER_VOLTAGE_RECONNECT                     =    0x9005 #(V|100)
EQUALIZATION_VOLTAGE                       =    0x9006 #(V|100)
BOOST_VOLTAGE                              =    0x9007 #(V|100)
FLOAT_VOLTAGE                              =    0x9008 #(V|100)
BOOST_RECONNECT_VOLTAGE                    =    0x9009 #(V|100)
LOW_VOLTAGE_RECONNECT                      =    0x900A #(V|100)
UNDER_VOLTAGE_RECONNECT                    =    0x900B #(V|100)
UNDER_VOLTAGE_WARNING                      =    0x900C #(V|100)
LOW_VOLTAGE_DISCONNECT                     =    0x900D #(V|100)
DISCHARGING_LIMIT_VOLTAGE                  =    0x900E #(V|100)
REAL_TIME_CLOCK_SEC_MIN                    =    0x9013 #
REAL_TIME_CLOCK_HOUR_DAY                   =    0x9014 #
REAL_TIME_CLOCK_MONTH_YEAR                 =    0x9015 #
EQUALIZATION_CHARGING_CYCLE                =    0x9016 #(Day)
BT_TEMP_WARNING_UPPER_LIM                  =    0x9017 #(℃|100)
BT_TEMP_WARNING_LOWER_LIM                  =    0x9018 #(℃|100)
CTRLR_INNER_TEMP_UPPER_LIM                 =    0x9019 #(℃|100)
CTRLR_INNER_TEMP_UPPER_LIM_RCVR            =    0x901A #(℃|100)
PWR_COMP_TEMP_UPPER_LMT                    =    0x901B #(℃|100)
PWR_COMP_TEMP_UPPER_LMT_RCVR               =    0x901C #(℃|100)
LINE_IMPEDANCE                             =    0x901D #(mOhm|100)
DAY_TIME_THRESHOLD_VOLTAGE                 =    0x901E #(V|100)
LIGHT_SIGNAL_STARTUP_DELAY_TIME            =    0x901F #(Min)
LIGHT_TIME_THRESHOLD_VOLTAGE               =    0x9020 #(V|100)
LIGHT_SIGNAL_CLOSE_DELAY_TIME              =    0x9021 #(Min)
LOAD_CONTROLLING_MODES                     =    0x903D #
WORKING_TIME_LENGTH_1                      =    0x903E #
WORKING_TIME_LENGTH_2                      =    0x903F #
```

Figure 15. Unused Setting Parameter Register Addresses

Another challenge was enabling communication between MongoDB and Node-Red. Due to the useful online resources on how to connect MongoDB and Node-Red, the connection was successfully done. Despite the success, there was a drawback.

First, MongoDB which was installed in AWS EC2 Instance was the Database that had to be connected to MongoDB. After putting in the MongoDB URL into Node-Red, the connection itself was successful, since Node-Red showed a success message in terms of connection. But the main problem was that the actual database did not reflect the changes that occurred by Node-Red. For instance, one sample data was inserted into MongoDB, but there was no change in MongoDB even if Node-Red sent the success message after inserting data.

Second, as a solution, a personal MongoDB URL was used for the project, and it worked. A connection was successful and change in Node-Red was properly reflected in MongoDB. For better service, future software development should use MongoDB in the AWS EC2 Instance. Using a personal MongoDB database is vulnerable to several problems such as hacking or losing connection. Using the MongoDB installed in AWS EC2 Instance will guarantee a more stable, safe connection to the database.

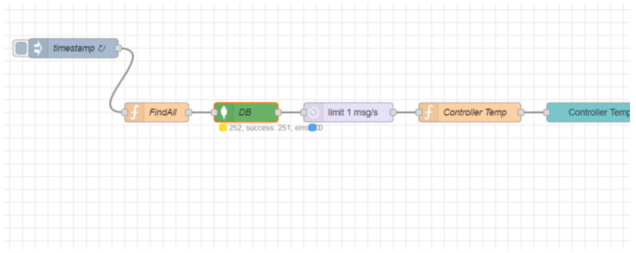


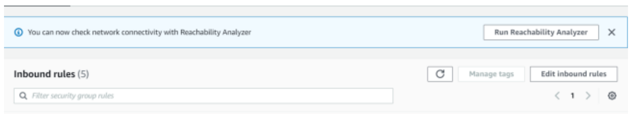
Figure 16. MongoDB with Node-RED

The last challenge was preparing AWS EC2 Instance with Ubuntu as a remote server. There were mainly 2 hardships along the process.

First was setting up Node-Red in AWS EC2 Instance [15] with Ubuntu. The proper installation process should be done step by step so that no error occurs. Thanks to helpful content online and official documentation in Node-Red Homepage, setting up Node-Red was very successful.

The second was allowing access of anyone using the application to the public IP of AWS EC2 Instance. Official documentation only mentioned 1880 port is used for Node-Red Application. But didn't mention how to properly allow the 1880 port to be used.

There were mainly 2 things that need to be done. The first was opening 1880 port and the other was allowing access from anywhere according to IP by editing the Inbound rule of Instance.



You can now check network connectivity with Reachability Analyzer

Figure 17. AWS EC2 Instance

VI. Conclusion

As the Internet of Things is widely used in our daily life in the form of a smart home, this paper discusses how to use the Internet of Things for remote monitoring of renewable energy(Solar). Based on the character of LoRa, the monitor system could be working in a low-power and wide-area network. The goal of this project is through real-time monitoring values (voltage and current), the raspberry pi connect with the sensor helps the users be able to monitor in real-time, once the raspberry pi catches the data, these data will upload to the database server(MongoDB) in real-time. Through a series of processes, the data could be monitored on the Node-RED monitor page. The effective utilization of renewable energy technology can be further strengthened.

Acknowledgment

We would like to thank Purdue University for taking us as participants of the IoT project this summer. We would like to thank Minji Lee for her dedication to being the supervisor throughout the whole program, giving us detailed information for documentation and practical tools for the progress of the project. Especially, would like to give thanks for helping with the Node-Red Framework. Her advice was very helpful with getting familiar with the Node-Red Framework. We would also like to show gratitude to the professors at Purdue University. Professor Eric Matson's guidance clarified the right way of our project and Professor Anthony Smith provided useful detailed information to help develop our project and make a breakthrough during hard moments. We all were able to make progress and improve our skills during the program thanks to the contribution of the professors and supervisors mentioned above.

References

[1] S. M. Patil, M. Vijayalashmi and R. Tapaskar, "IoT based solar energy monitoring system," 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017, pp. 1574-1579, doi: 10.1109/ICECDS.2017.8389711.

[2] A. Gupta, R. Jain, R. Joshi and R. Saxena, "Real time remote solar monitoring system," 2017 3rd International Conference on Advances in Computing, Communication & Automation (ICACCA) (Fall), 2017, pp. 1-5, doi: 10.1109/ICACCAF.2017.8344723.

[3] Mirsad, H. S., & Nasser, H. A. (2020). IoT based efficient solar panel monitoring. 3C Tecnologia, 9(4), 87-93. Retrieved from https://www.proquest.com/scholarly-journals/iot-based-efficient-solar-panel-monitoring/docview/2477688312/se-2?accountid=13360

[4] Buy a Raspberry Pi 3 Model B – Raspberry Pi

https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[5] Solar Charge Controllers | Inverters & Inverter Chargers | MPPT solar charge controller | Battery Charger | PWM Solar Charge Controller (epsolarpv.com)

[6] EPEVER-Datasheet-XTRA-N (epsolarpv.com)

[7] Contributor, T. T., &amp; Wigmore, I. (2021, July 16). What is an Amazon EC2 instance? SearchAWS. https://searchaws.techtarget.com/definition/Amazon-EC2-instances

[8] https://nodered.org

[9] Flow based programming tool | Node-Red

https://nodered.org/about/

[10] Cloud Services - Amazon Web Services (AWS)

https://aws.amazon.com/?nc1=h_ls

[11] The most popular database for modern apps | MongoDB

https ://www.mongodb.com

[12] Managed MongoDB Hosting | Database-as-a-Service | MongoDB

https://www.mongodb.com/cloud/atlas

[13] IBM Cloud Education. (n.d.). mongodb. IBM. https://www.ibm.com/cloud/learn/mongodb.

[14] Modbus application protocol specification v1.1b3, April 26, 2012

http://www.modbus.org

[15] Official Documentation About Running Node-Red FrameWork on AWS EC2 Instance With Ubuntu.

https://nodered.org/docs/getting-started/aws