



IOT POWER MONITORING USING EASTRON SDM230 AND ARDUINO

**A final project report
presented to
the Faculty of Engineering**

**By
Timothy Toby
002201400020**

**in partial fulfillment
of the requirements of the degree
Bachelor of Science in Electrical Engineering**

**President University
January 2020**



IOT POWER MONITORING USING EASTRON SDM230 AND ARDUINO

**A final project report
presented to
the Faculty of Engineering**

By

Timothy Toby

002201400020

**in partial fulfillment
of the requirements of the degree
Bachelor of Science in Electrical Engineering**

President University

January 2020

DECLARATION OF ORIGINALITY

I declare that this final project report, titled “IoT Power Monitoring Using Eastron SDM230 and Arduino” is my own original piece of work and, to the best of my knowledge and belief, has not been submitted, either in whole or in part, to another university to obtain a degree. All sources that are quoted or referred to are truly declared.

Cikarang, Indonesia, January 2020

A handwritten signature in black ink, appearing to read 'Timothy Toby', with a stylized, cursive style.

Timothy Toby

APPROVAL PAGE

IOT POWER MONITORING USING EASTRON SDM230 AND ARDUINO

By


Timothy Toby

002201400020

Approved by



Irwan Purnama Ph.D
Final Project Advisor



Antonius Suhartomo Ph.D
Final Project Supervisor &
Head of Study Program
Electrical Engineering



Dr. Ing. Erwin P. Sitompul, ST., M.Sc

Dean of Faculty of Engineering

ACKNOWLEDGEMENT

I would like first to thank GOD in all of his wisdom and kindness that I have managed to be able to finish this thesis. I would also like to thank my thesis advisor Mr. Antonius Suhartomo, M.Eng.Sc., M.M., Ph.D. to have always supported me in this long journey to this point, for the reminders and encouragements all the while I was completing the project. I would also thank Mr. Irwan Purnama, M.Sc.Eng., Ph.D. as my second thesis advisor for the guidance and advices concerning the troubles and hiccups of the project.

I would also like to express my gratitude to my parents whose material and immaterial support have been invaluable to the completion of this project. I would also like to thank my classmates and my friends who have given me unfailing support during the time span I was working on this project. This accomplishment would have not been possible without them. Thank you.

Cikarang, January 2020

Timothy Toby

APPROVAL FOR SCIENTIFIC PUBLICATION

I hereby, for the purpose of development of science and technology, certify and approve to give President University a non-exclusive royalty-free right upon my final project report with the title:

IOT POWER MONITORING USING EASTRON SDM230 AND ARDUINO

Along with the related software or hardware prototype (if needed). With this non-exclusive royalty-free right, President University is entitled to conserve, to convert, to manage in a database, to maintain, and to publish my final project report. These are to be done with the obligation from President University to mention my name as the copyright owner of my final project report.

Cikarang, January 2020



Timothy Toby

002201400020

ABSTRACT

Internet of Things are more widely being implemented all around the world. One of the things that are being integrated to the system is power monitoring. Currently available power monitoring systems have not implemented the Internet of Things extensively. This project designs and constructs a working power monitoring system that is integrated to the internet, using an SDM 230 power meter and an Arduino UNO. The experiment is conducted using a laptop and an Ethernet shield for the Arduino for internet connectivity, and a third party website ThingSpeak for data display and IoT functionality. Using an electric water heater as a load and a series data recorded in the time span of 20 minutes, the analysis is made of the accuracy of the data that is obtained from the power meter and the display on the website, with the result of 0.68% accuracy.

Keywords: Internet of Things, Arduino, Power monitoring, IoT, ThingSpeak

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY	ii
APPROVAL PAGE	iii
ACKNOWLEDGEMENT	iv
APPROVAL FOR SCIENTIFIC PUBLICATION	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	ix
LIST OF TABLES	x
CHAPTER 1 INTRODUCTION	1
1.1. Final Project Background.....	1
1.2. Problem Statement	2
1.3. Final Project Objectives	2
1.4. Final Project Scopes and Limitations.....	2
1.4.1. Final project scopes	2
1.4.2. Final project limitations.....	2
CHAPTER 2 BACKGROUND	3
2.1. Internet of Things	3
2.2. Power Monitoring	4
2.3 Arduino.....	7
2.4 ThingSpeak.....	9
2.5 Literature Study.....	11
CHAPTER 3 DESIGN IMPLEMENTATIONS	12
3.1. Prototype Design Overview	12
3.2 Components.....	14
3.2.1. Eastron SDM 230 Modbus	14
3.2.2. Arduino UNO	15
3.2.3. MAX RS485	16
3.2.4 Arduino Ethernet shield.....	16
3.3 Program Code.....	16
CHAPTER 4 RESULTS AND ANALYSIS	19
CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS	24
5.1 Conclusions	24

5.2 Future Recommendations.....	24
References.....	25
APPENDIX	A
Program code.....	A

LIST OF FIGURES

Figure 2.1 Internet of Things	3
Figure 2.2 Power dissipation of the load in practical application.....	5
Figure 2.3 Waveform graph of instantaneous power $p(t)$, voltage $v(t)$, and current $i(t)$	6
Figure 2.5 Arduino IDE Software with sketch	8
Figure 2.4 Arduino IDE Software UI.....	8
Figure 2.6 Arduino IDE Software Library Manager.....	8
Figure 2.7 ThingSpeak homepage (https://ThingSpeak.com/)	9
Figure 2.9 ThingSpeak write and read keys.....	10
Figure 2.8 ThingSpeak user channels page.....	10
Figure 3.1 Project Design Flow Chart	13
Figure 3.2 Project Design Schematics	14
Figure 3.3 SDM230 Modbus	14
Figure 3.4 Arduino UNO	15
Figure 3.5 Arduino UNO schematics diagram.....	15
Figure 3.6 MAX RS485	16
Figure 3.7 Ethernet Shield	16
Figure 3.8 Setting up the union variables	17
Figure 3.9 Combining the data retrieved	17
Figure 3.10 Float to String conversion.....	17
Figure 3.11 Sending data to ThingSpeak and status check.....	18
Figure 3.12 List of return codes to check progress of data transfer.....	18
Figure 4.1 Final Project Prototype	19
Figure 4.2 Experiment Results.....	20
Figure 4.2 Voltage values	20
Figure 4.3 Current values.....	21
Figure 4.4 Power values.....	21
Figure 4.5 Faulty value comparison.....	22
Figure 4.6 Power field stays consistent.....	22

LIST OF TABLES

Table 4.1 Accuracy test results	23
--	----

CHAPTER 1

INTRODUCTION

1.1. Final Project Background

As long as humans have existed on earth, we always make advancements on our lives to make it easier. Nowadays there are a lot of technological advancements that are an essential part of our lives. Some have even become a necessity on people's lives. But, as humans, we want to have things be able to be easily accessed or used. This makes a new problem of accessibility in the current existing technologies.

Over the span of the last decade, many accessibility improvements are applied to existing technologies. Bank accounts now can be accessed through online means instead of having to go to the bank's physical building itself. Cashless payments are now readily available everywhere more than ever, and are probably going to continue to spread, and even cardless transactions can be made on ATMs and similar machines. This shows that our need of ease of access is real.

This project will address to the problem of ease of access in one sector, specifically power monitoring. In all the technology used today, almost everything are now equipped with some sort of electronics, either a small, microchip, or even a large transformer. This technology all relies on electricity. The electricity used are of course not readily available in nature in usable form, it needs to be converted and delivered to each of the users. This have cost and one way to cover all the cost are billing the usage of the electricity. In order to be able to quantify the usage, there needs to be a monitoring system that are implemented on the usage.

This project will address to the need of easier means of accessing such monitoring system using wireless systems. The project will be focused on making a power monitoring system that can be accessed even when the user does not have direct access to the physical monitoring device itself.

1.2. Problem Statement

Based on the final project background, the problem statements to reach the result of this final project are:

- How to monitor the power usage on a load?
- How to get the readings from the monitoring device to other storage means?

1.3. Final Project Objectives

The final project objectives which have to be fulfilled are as follows:

- To design and construct a fully working power monitoring system.
- To make sure the monitoring system can send the data analyzed to other devices and eventually to cloud services.

1.4. Final Project Scopes and Limitations

1.4.1. Final project scopes

This final project will be conducted under the following scopes:

- The monitoring system will implement an existing power monitoring device, specifically the Eastron SDM 230.
- The data from the monitoring device will be analyzed and processed using an Arduino UNO Project Development Board.

1.4.2. Final project limitations

This final project will be conducted under the following limitations:

- The monitoring system will be tested on a single load of an electric water heater through the monitoring device.

CHAPTER 2

BACKGROUND

2.1. Internet of Things

As internet is becoming more and more pivotal to our daily lives, more things are getting integrated to its networks. From computers and now mobile devices, a lot of them have the capability to connect to the internet. Nowadays, even home appliances can have internet connectivity [3]. This connection of objects that can share information between each other is what is commonly called the *Internet of Things* [4].

This opens up new possibilities in the current industry. Common household appliances are now starting to be integrated into the Internet of Things and are starting to be able to be accessed remotely. Further applications of this systems include smart homes that are becoming more and more common, having the components of the said house to be able to be accessed from the internet, starting from garage doors, door locks, and even coffee machines. As this trend continues to go on forward, the need for more things to be able to have that integration to the internet is crucial.

Figure 2.1 shows the simplified chart of internet of things. First the home appliance will be equipped with the ability to connect to the internet, then through the internet connection, it will be able to send and receive data from a service, usually through a cloud that will manage the data that the appliance sent and also data that the appliance will receive. The user then will be able to access or sent data to the appliance using an application or a web service. This system can then be implemented on a larger scale that can cover an entire house (Smart Homes) or even an entire city (Smart Cities) [4].

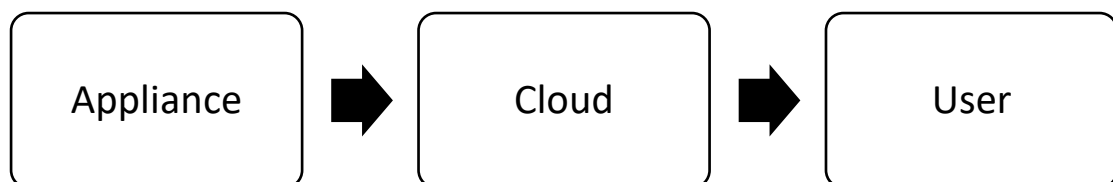


Figure 2.1 Internet of Things

2.2. Power Monitoring

Power monitoring is an act of monitoring this usage for different purposes. One of the main uses for power monitoring systems are billing. Since electricity used by electronics and electrical appliances are not readily available in nature, there are costs needed to produce them. Thus, a billing system is needed. To monitor the usage, such systems are implemented, both for the biller and the user. For the biller, this kind of system is useful to know the quantity of energy used and the amount to bill according to the quantity. For the user, this system is useful to know how much energy they use and based on that, improve the usage to minimize the bills [5].

Electricity that are used by users for the current existing technology is measured in units of power. Electrical power dissipated by a load (P_L) are measured by taking the voltage (V_L) and multiplying it by the amount of current (I_L), shown by the expression:

$$P_L = V_L \times I_L \quad (2.1)$$

where P_L is the amount of power in Watts which is the result of the voltage in Volts multiplied by the current in Ampere. This is what is used in DC circuits power measurement. All the values are measured by inspecting the load that is present on the circuit. In AC circuits, different formulas are used. In AC circuits, electric power is defined as voltage drop across the load times the current flowing through it [1], thus the function becomes:

$$p(t) = v(t) \times i(t) \quad (2.2)$$

this formula is used to measure the instantaneous power of the circuit. The value that will be used in power monitoring is the mean value of the instantaneous power in a time period (T) [1], the formula for the active power is:

$$P = \frac{1}{T} \int_0^T p(t) dt \quad (2.3)$$

In AC circuits, the circuit will be supplied by a sinusoidal power supply and the voltage ($v(t)$) and current ($i(t)$) can be in a different phase. When the two values is in the same phase, the $p(t)$ is given by:

$$p(t) = VI[1 - \cos(2\omega t)] \quad (2.4)$$

where the V and I in the equation is the Root Mean Square (rms) value of $v(t)$ and $i(t)$, and the ω is the angular frequency of the power supply. But when the load is a purely reactive load, the $v(t)$ and $i(t)$ will be out of phase by 90° , then the formula will become:

$$p(t) = VI \cos(2\omega t) \quad (2.5)$$

Because of the phase introduced in this reactive load, the active power dissipated by such load will be zero. As can be seen in Figure 2.2, in practical application, the load will have both types of load and the load will be represented by a quantity called Impedance (Z) which expresses the sum of the electrical power dissipated in the resistive load (R) and reactive load (X).

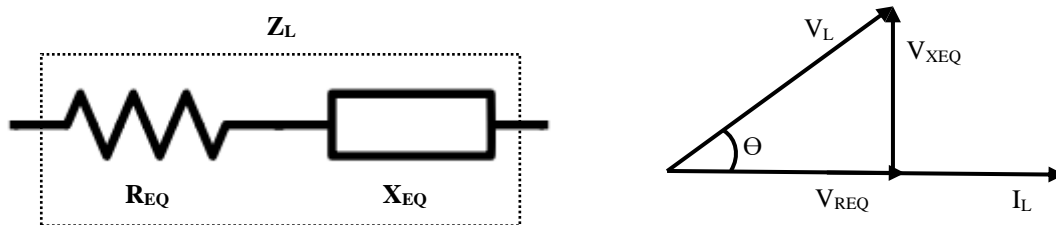


Figure 2.2 Power dissipation of the load in practical application

In light of this fact, the formula for power calculation becomes:

$$P = V_{REQ}I_L = V_L I_L \cos\theta \quad (2.6)$$

This formula is formed because as seen in Figure 2.2, the part of V_{XEQ} does not really contribute to the actual amount of power used by the load as it is 90° out of phase with the current flowing into the load. As only a portion of the voltage actually contributes to the active power usage, thus the $\cos\theta$ part of the equation is needed, this part is commonly referred to as a power factor. Figure 2.3 shows the effect of power factor in correlation to the waveforms of instantaneous power, voltage, and current. It shows in the DC component of $p(t)$ that varies from a null wave towards the VI value [1].

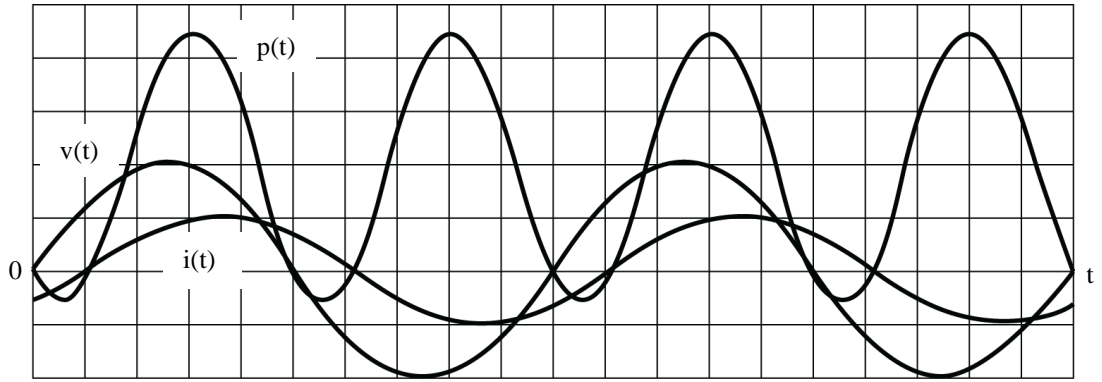


Figure 2.3 Waveform graph of instantaneous power $p(t)$, voltage $v(t)$, and current $i(t)$

In relation to Figure 2.2, in calculating the power dissipations, when considering both the active and reactive power, the formula used will be simply:

$$S = V_L I_L \quad (2.7)$$

when using this formula, we will get the apparent power, which considers all the components of the load. It is measured in units of Volt-Amps (VA). When only considering the actual active power used, the formula:

$$P = V_{REQ} I_L = V_L I_L \cos\theta \quad (2.8)$$

will be used. This is what is called active power and are most commonly measured in kilo Watt (kW) or mega Watt (MW). To get the values of the reactive power which does not actually contributes to the power dissipated in the load itself, we can use the formula:

$$Q = V_{XEQ} I_L = V_L I_L \sin\theta \quad (2.9)$$

with this formula we can get the value of power dissipated in inductive and capacitive loads, which counts as the reactive power measured in units of Volt-Amps Reactive (VAR). The three types of power have correlations that can be concluded from the Figure 2.2 as such:

$$S = \sqrt{P^2 + Q^2} \quad (2.10)$$

That is the basics of power monitoring and the formulas that are usually used in the process. The formulas are integrated on the system of the power monitoring devices that are available today.

2.3 Arduino

Arduino is a project development board based on a microcontroller chip, combined with its own modules coming from a company of the same name. This board is used in a lot of projects due to its simplicity and versatility in usage. Coming with its own software and hardware modules, the usage of this project development board is very efficient. The main micro controller unit modules can be programmed using a software that is developed to suit the chip and its additional modules. This enables projects wider range of control that otherwise would not be available for such projects.

The programmable microcontroller chip on the Arduino enables automation projects and other projects of such nature to be created with components that are not programmable by design. One such example is this IoT Power Monitoring project. The SDM 230 Modbus power monitor is not programmable by nature and can only analyze the data and display it in a built-in display, this does not allow for a remote viewing and monitoring, but the Arduino can be programmed to extract the data from the Modbus to then further send it through an Ethernet connection to a cloud.

Arduino have its own development software to program the microcontroller to control other module and devices. This software is developed by the same company that developed the project board and are compatible with all the other Arduino project boards that are developed by other companies. This compatibility also includes the other additional modules that can be used with the main microcontroller unit Arduinos. This combination of features enables automation projects to be done as the Arduino microcontroller units can run its program automatically after it is first programmed by the user. It will just need power and the sufficient modules and peripherals to execute the program. Figure 2.4 have an example of the user interface of the software and Figure 2.5 is an example of the software containing a sketch or a program that can be uploaded onto an Arduino microcontroller unit.

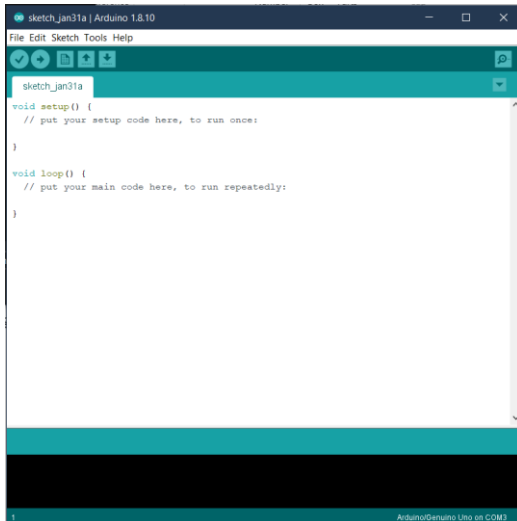


Figure 2.4 Arduino IDE Software UI

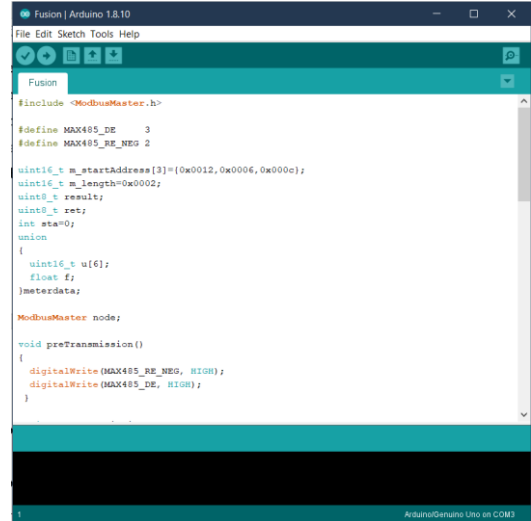


Figure 2.5 Arduino IDE Software with sketch

The software comes with libraries that can enable the usage of the additional modules for the Arduino without additional configurations. There are additional libraries to download. The libraries can be developed by other parties as the software is open-sourced. The distribution of the libraries can be done using the software itself by using the built in library manager as can be seen in Figure 2.6, or by third party website downloads. The libraries contains a set of commands that will execute a specific function that have been specified in the libraries files.

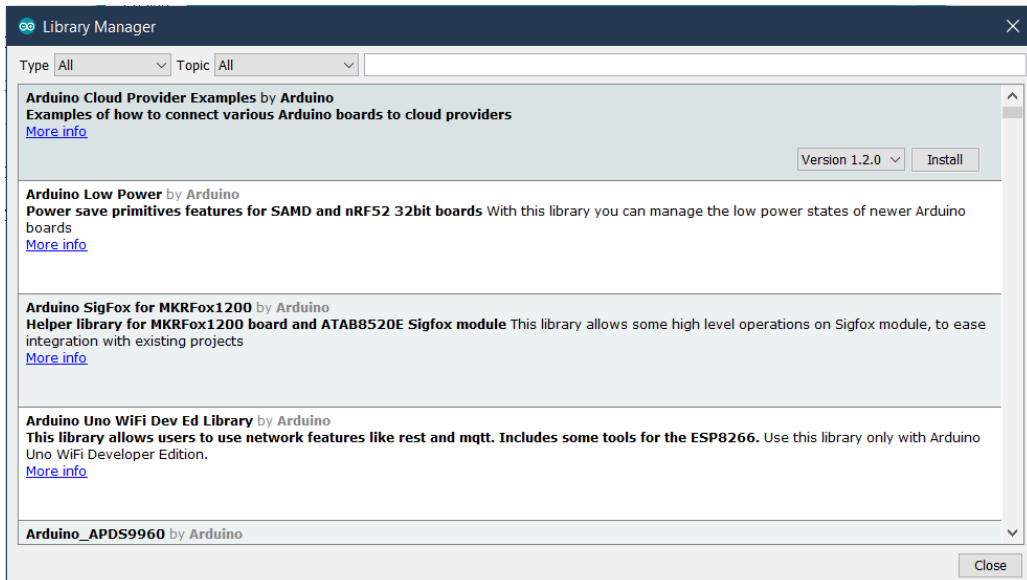


Figure 2.6 Arduino IDE Software Library Manager

2.4 ThingSpeak

ThingSpeak is an IoT analytics service website that provides free services for IoT projects and within their website is also provided visualization tools to make graphs and such of the data sent to their servers. It is one of the services that Mathworks provides. Mathworks itself is a computer software developer mainly focused on engineering software. ThingSpeak is used in the project in lieu of their ease of access and their already well-known service. It is simple to register an account and their services can be used immediately after setting said account. They also have their own libraries to be used on the Arduino IDE software so the compatibility with Arduino is already confirmed. We can see the main page of the ThingSpeak web page in Figure 2.7, after setting up an account and accessing their own account homepage, the user can see their 'channels' (Figure 2.8) that contain visualization of their project data that they sent to the ThingSpeak servers.

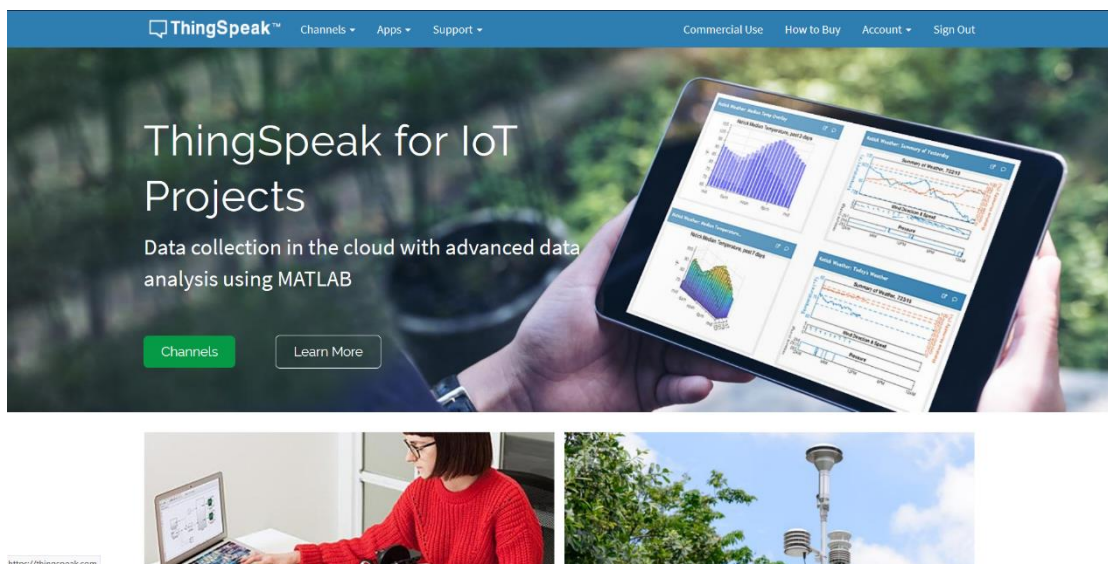


Figure 2.7 ThingSpeak homepage (<https://ThingSpeak.com/>)

In this project, the Arduino will be programmed to send data received from the SDM 230 Modbus power monitor to this cloud service. When the Arduino is connected to the internet through the Ethernet Shield, the Arduino will be able to connect to the service and using a unique write key (Figure 2.9) to access the channel on Thingpeak, it can modify or add values to it. The key is provided by ThingSpeak and is needed whenever the user wants access to write the data on to the channel, to prevent unauthenticated access. And if the channel is set to private view, a read key will be required to read the data also.

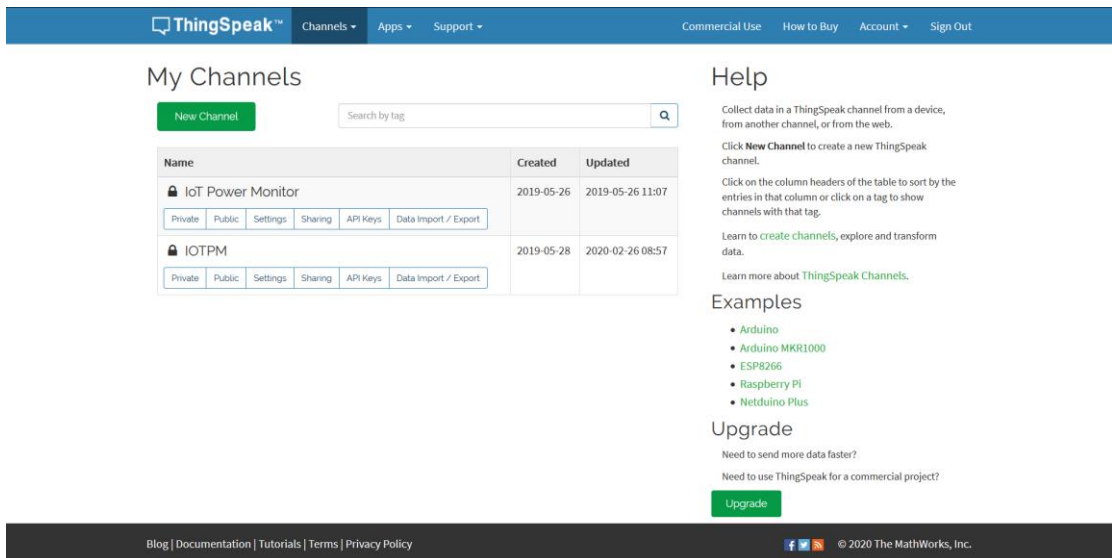


Figure 2.8 ThingSpeak user channels page

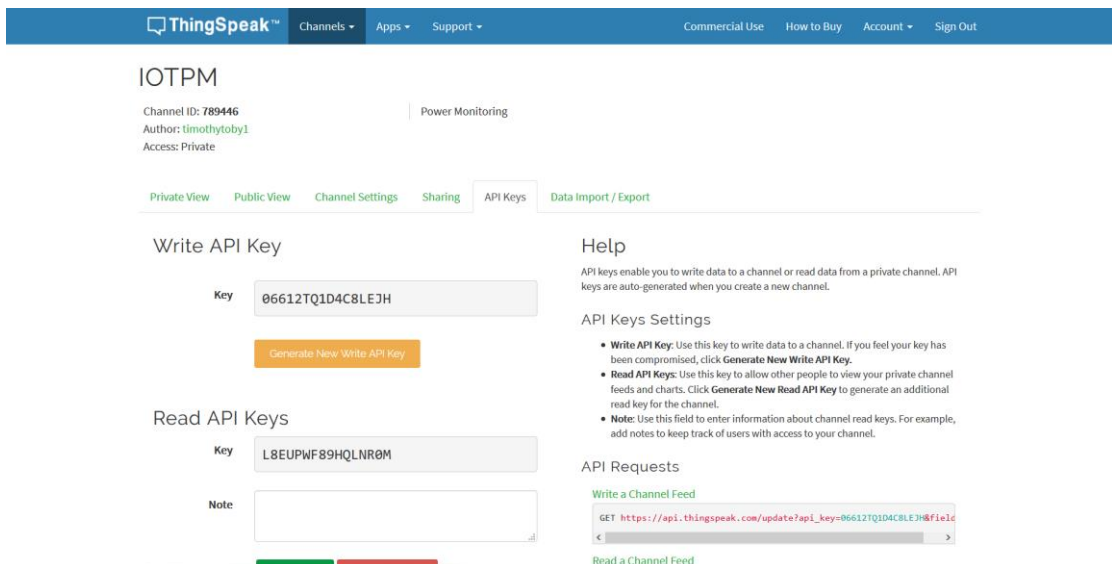


Figure 2.9 ThingSpeak write and read keys

Note that based on Figure 2.8 and Figure 2.9, the user can have multiple channels and each of the channels will have its own unique keys to access the channels. This is what makes the ThingSpeak one of the preferable service to be used in this project. The simplicity of usage, integration and organization of the service fits the project.

2.5 Literature Study

A study of similar basis was conducted by a Rahayu Dwi Lestari from Telkom University under a report titled “Sistem Pemantauan Daya Berbasis Internet of Things”. In that study, a different model of an Eastron power meter is used, the SDM 120, the model used in this project is the SDM 230 which is a model that hosts more features. The signal converter used are also different, which uses a RS485 DFRobot, while this project uses the MAX RS485, and the studied project uses a XAMPP localhost server, while this project uses a third party website ThingSpeak for more accessibility over the internet.

CHAPTER 3

DESIGN IMPLEMENTATIONS

3.1. Prototype Design Overview

The SDM230 Modbus Power Meter will be the device that will enable us to read the data needed from the physical source. This device will then display the data on its built in LCD display. As the SDM 230 Modbus does not have a built-in capability to be programmed freely, the Arduino UNO will be used in order to have more flexibility in handling the data extracted from the power meter. The Arduino UNO will enable the project to be modified further. As the project goal is to transfer the data to some other device, in this case, a ThingSpeak server through an internet connection.

This is where the ethernet shield module for the Arduino will be used. One of the advantages of using an Arduino is the variety of modules that are compatible with the project board as it is made to be used with it. This will ensure the compatibility and will eliminate any problem that might be caused by using other ethernet modules with the Arduino. And the code to be able to run the system will be supported by Arduino development software also.

Since the SDM 230 Modbus outputs the data signal through a RS485 port, the signal needs to be converted first for the Arduino to be able to read, since the Arduino uses the TTL communication protocol. Because the RS485 protocol uses a differential to decide the values of the data and the TTL just uses the state of the data, for example, in RS485 protocol, there will be 2 data, A and B, these 2 data will have different voltage and the voltage difference will determine the value. But in TTL protocol, there will be only one data and the voltage of that data will determine that value. This makes the Arduino and the SDM 230 cannot communicate, so we need to 'translate' the signal for the Arduino. To do this, we used a Max RS485 to TTL converter, this is an additional module that converts signals from RS485 communication protocol to a TTL protocol. This module will be the bridge that connects the SDM 230 Modbus to the Arduino. After the signal is converted, we then need to program the Arduino to retrieve the data and then send it over the Ethernet to ThingSpeak.

As can be seen in figure 3.1, the SDM 230 module will be connected to the AC power source, this will enable the module to monitor the incoming voltage that will be used in the system, and then a load will be connected to the SDM 230, this will complete the basic functionality of the SDM 230 power monitor. The SDM 230 will then start to measure the amount of current and voltage used by the load and then display it on its built-in display. The Arduino will then be connected to a communication port on the SDM 230, this is the port that are assigned on the SDM 230 to export analyzed data to other devices. As explained before, there will be a signal converter in between the two modules to enable communication. When connected, the Arduino can be programmed to request data which in turn will be sent by the SDM 230 back to the Arduino to be further bridged to ThingSpeak.

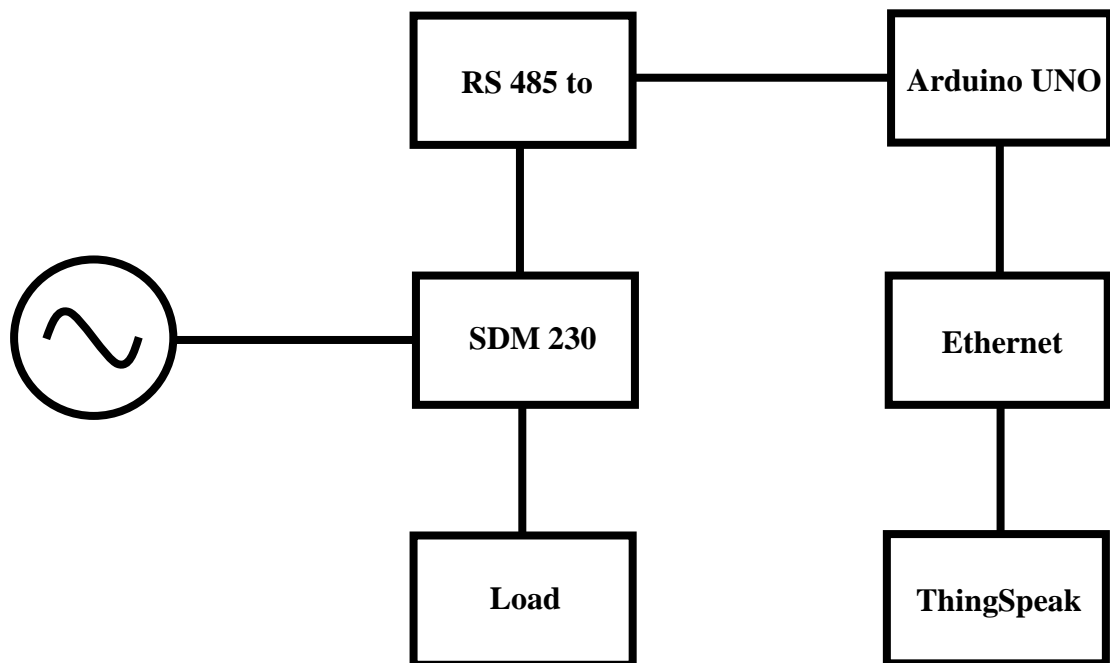


Figure 3.1 Project Design Flow Chart

Figure 3.2 shows the schematics of this project, the Arduino UNO part will be under the Ethernet Shield and all the pins that they have will overlap and connect to each other, with the MAX RS485 pins of RO, RE, DE, and DI will connect to the Ethernet Shield's pin RX, 2, 3, and TX respectively. The MAX RS485 vcc pin will go to the 3,3v Ethernet Shield pin, GND to GND pin, and the A and B pin will go to the SDM 230 Power meters RS 485 export port. The Arduino and Ethernet shield will be connected to a laptop during the experiment for internet and programming.

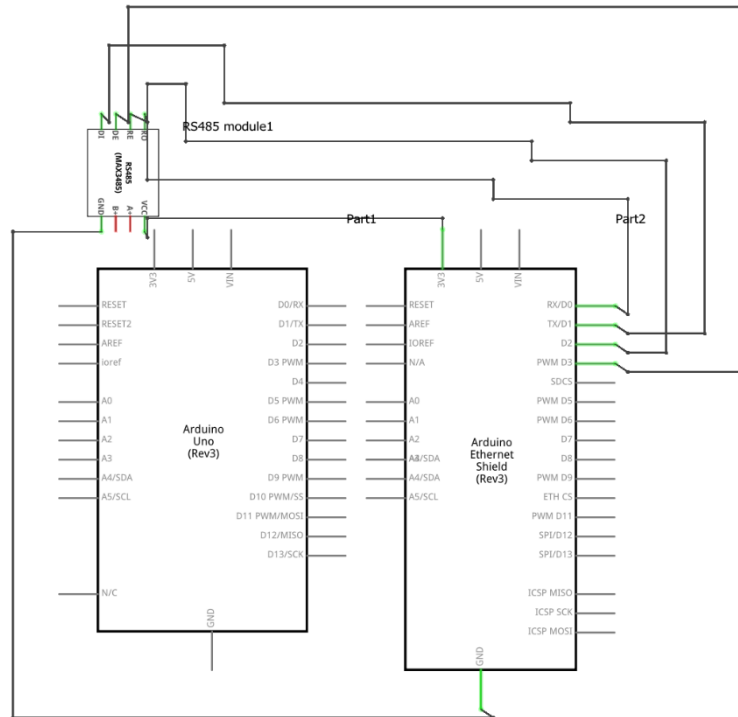


Figure 3.2 Project Design Schematics

3.2 Components

3.2.1. Eastron SDM 230 Modbus

This is the power monitoring device that will be used in this project. This device can monitor the voltage, current, energy usage. This device is one of the models available from the Eastron manufacturer that produces a lot variety of power monitoring devices. This SDM 230 Modbus model have this specifications:

- Voltage range : 179 ~ 276V AC
- Base Current : 10A
- Max. Current : 100A
- Min. Current : 0.5A
- Frequency : 50/60Hz
- Max. Reading : 999999.9 KWh



Figure 3.3 SDM230 Modbus

3.2.2. Arduino UNO

This is the device that is going to be used to analyze and process the data received from the power meter, as the power meter itself cannot be programmed by the user, this module will be used to control what data will be extracted from the Modbus power meter and where will the data go. This will let the user choose where to transfer the data to with more flexibility. This is the specifications for the Arduino UNO:

- Microcontroller : ATmega328P
- Operating Voltage : 5V
- Input Voltage : 6-20V
- Flash Memory : 32KB
- SRAM : 2KB
- EEPROM : 1KB
- Clock Speed : 16 MHz
- Analog Input Pins : 6
- Digital I/O pins : 14 (6 provides PWM output)



Figure 3.4 Arduino UNO

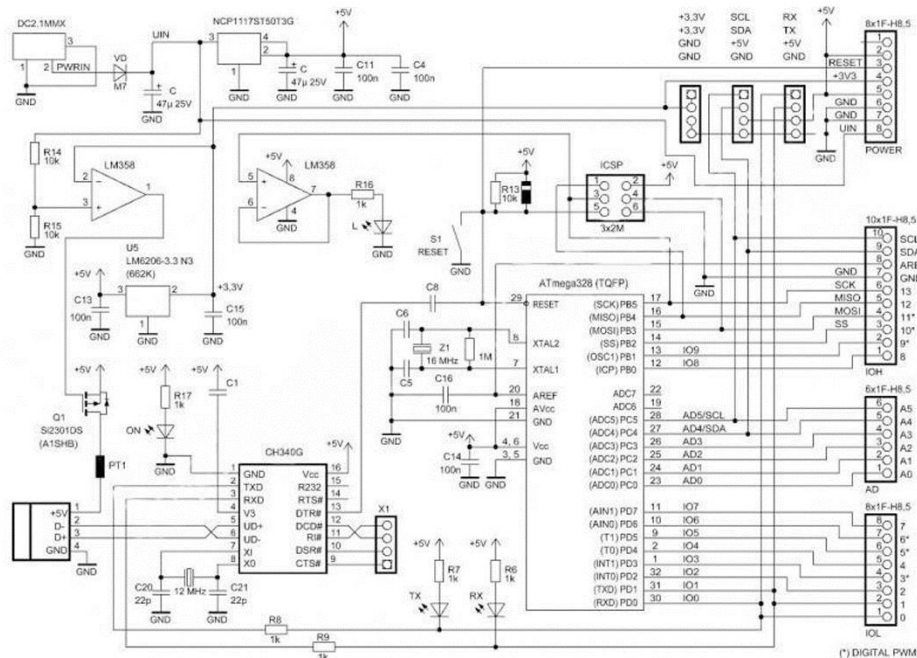


Figure 3.5 Arduino UNO schematics

3.2.3. MAX RS485

Max RS485 is a converter from an RS485 communication port to TTL type port. This is needed to be able to send data acquired by the Eastron SDM 230 Modbus to be analyzed by the Arduino UNO to be transmitted over through a wireless network. Because the RS 485 communication protocol used by the SDM 230 Modbus uses 2 lines and compare the voltage between the two lines to determine the data value, while the TTL protocol used by the Arduino only uses one line and the voltage value will be the data, if connected directly, the Arduino will not be able to read the data properly, hence the need of this converter.

3.2.4 Arduino Ethernet shield

This is one of the modules that are available as a complement to the Arduino boards. This module is what will enable the Arduino UNO to be able to connect to an ethernet connection, in order to transfer the data to the web server.

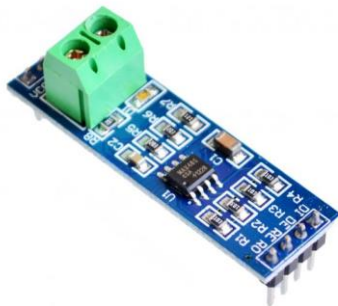


Figure 3.6 MAX RS485

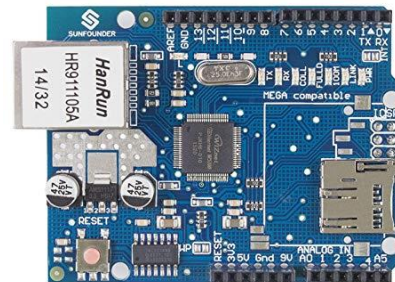


Figure 3.7 Ethernet Shield

3.3 Program Code

In order for the Arduino to be able to get the data needed, it needs to be programmed first to call out the data stored in the SDM 230 Modbus. The SDM 230 reads the values of the system in a 16 bit float type data value, but stores it in 2 parts of 8 bit binary data type, thus there must be some conversion of the data called from the storage in SDM 230 in order for the Arduino to receive the correct reading of the values.

In order to do that, the values will need to be combined using the union function of the code. As can be seen in Figure 3.8, the code will contain the union variables defined first before the function can be used later. After that, the code that request the data from the SDM 230 can be modified so that it will read the data on the two cells and combine it to form the actual float value using the union function prepared before (Figure 3.9).

```
uint8_t voltageRead, currentRead, powerRead;
union {
    uint16_t vu[2];
    float vf;
} voltage;
union {
    uint16_t cu[2];
    float cf;
} current;
union {
    uint16_t pu[2];
    float pf;
} power;
```

Figure 3.8 Setting up the union variables

```
voltageRead = node.readInputRegisters (0x30000, 2);
if (voltageRead == node.ku8MBSuccess) {
    voltage.vu[1] = node.getResponseBuffer (0);
    voltage.vu[0] = node.getResponseBuffer (1);
}

currentRead = node.readInputRegisters (0x30006, 2);
if (voltageRead == node.ku8MBSuccess) {
    current.cu[1] = node.getResponseBuffer (0);
    current.cu[0] = node.getResponseBuffer (1);
}

powerRead = node.readInputRegisters (0x30012, 2);
if (powerRead == node.ku8MBSuccess) {
    power.pu[1] = node.getResponseBuffer (0);
    power.pu[0] = node.getResponseBuffer (1);
}
```

Figure 3.9 Combining the data retrieved

The float values are the actual readings of the device and are what we need. After the data is successfully converted, we then can use that data and send it to ThingSpeak. But since ThingSpeak expects strings as data type that will be received, we need to convert the float type values to the string type that ThingSpeak expected. This will require the bit of the code seen on Figure 3.10.

```
char v_buffer[10];
char c_buffer[10];
char p_buffer[10];
float V = voltage.vf;
float C = current.cf;
float P = power.pf;
String strVoltage = dtostrf(V, 0, 5, v_buffer);
String strCurrent = dtostrf(C, 0, 5, c_buffer);
String strPower = dtostrf(P, 0, 5, p_buffer);
```

Figure 3.10 Float to String conversion

After the data is converted to string type, we can use the serial.Print function code to check if the values are converted successfully. The serial.Print function will print the data on the serial monitor of the Arduino IDE. After the data has been successfully converted, it is then just needed to be sent to the ThingSpeak service. Figure 4.5 is the code that will command the Arduino to send the data and check if the data has been successfully sent. If there are a failure, the serial monitor will print out the failure code and it can be checked with the code database obtained from the GitHub page of the library [6] to check what went wrong (Figure 4.6).

```
//-----Setting Thingspeak Fields-----//
ThingSpeak.setField(1, strVoltage);
ThingSpeak.setField(2, strCurrent);
ThingSpeak.setField(3, strPower);

//-----Send Updates to Thingspeak Fields-----//
int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if (x == 200) {
    Serial.println("Channel update successful.");
}
else {
    Serial.println("Problem updating channel. HTTP error code " + String(x));
}
```

Figure 3.11 Sending data to ThingSpeak and status check

Value	Meaning
200	OK / Success
404	Incorrect API key (or invalid ThingSpeak server address)
-101	Value is out of range or string is too long (> 255 characters)
-201	Invalid field number specified
-210	setField() was not called before writeFields()
-301	Failed to connect to ThingSpeak
-302	Unexpected failure during write to ThingSpeak
-303	Unable to parse response
-304	Timeout waiting for server to respond
-401	Point was not inserted (most probable cause is the rate limit of once every 15 seconds)
0	Other error

Figure 3.12 List of return codes to check progress of data transfer

CHAPTER 4

RESULTS AND ANALYSIS

The final project prototype uses an electric water heater rated at 350W to test the system, the amount of voltage, current, and power drawn by the heater will be analyzed by the SDM 230 Modbus Power Meter and then be displayed in a simple website for remote monitoring. The prototype configuration can be seen in Figure 4.1.

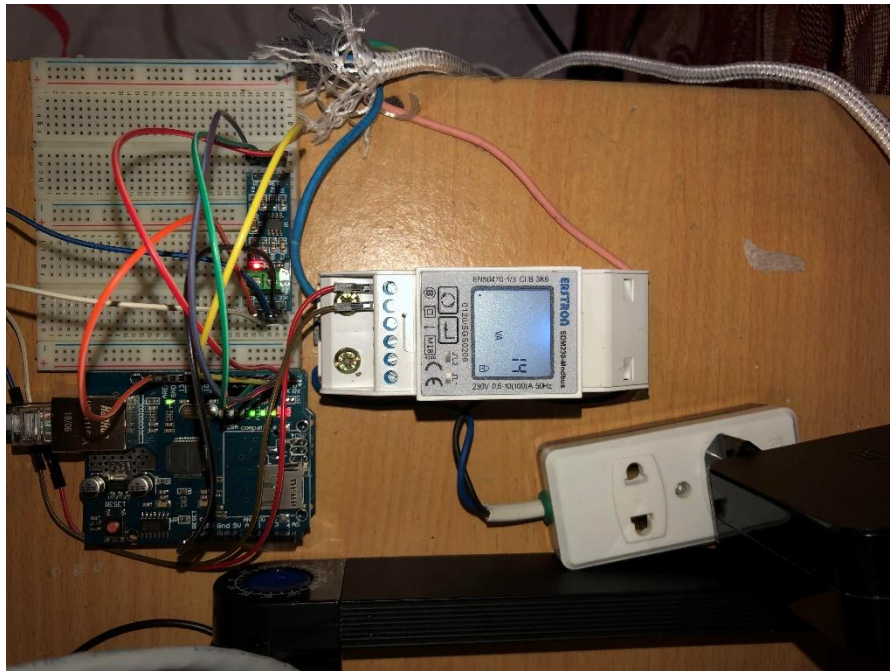


Figure 4.1 Final Project Prototype

As discussed in Chapter 3, the Arduino will be mounted by its Ethernet shield module which both are connected to a laptop that will provide the program to the Arduino and internet connection to the Ethernet shield. After the program is uploaded to the Arduino though, the connection to the Arduino itself only serves as power delivery for both chips, as such, the connection to the Arduino can be simply replaced with a normal dc adapter that the Arduino is compatible with. The data will then be able to be accessed through the ThingSpeak channel when the program is successfully executed.

The data displayed on the ThingSpeak of the experiment result can be seen in Figure 4.2. In the experiment, a total of 3 data values were observed, Voltage, Current and Active Power.

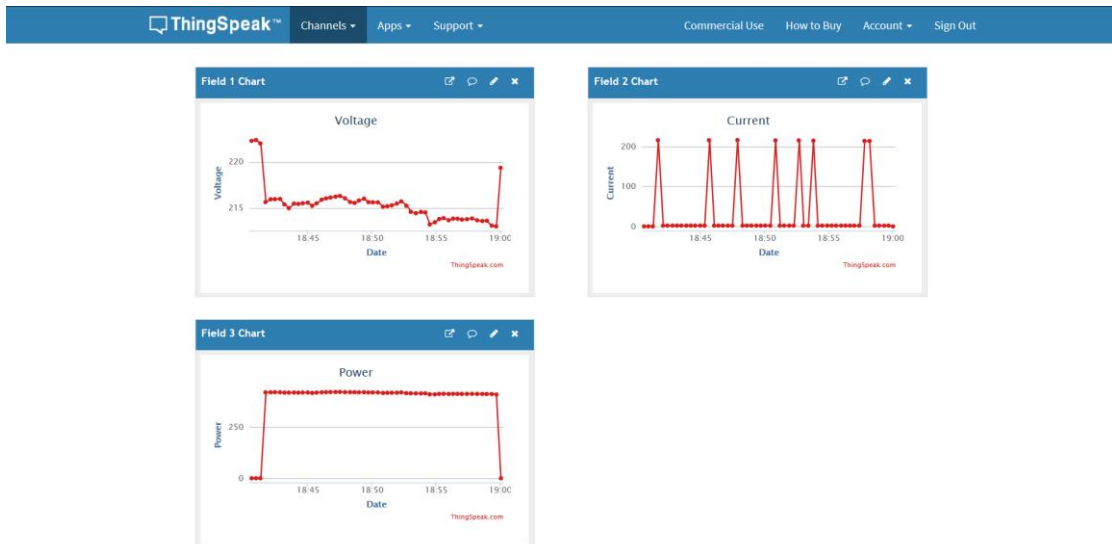


Figure 4.2 Experiment Results

The first field is the Voltage field, this records the voltage drawn by system from the AC power source. The AC power source is rated at 220 V AC and the graph on Figure 4.3 shows that there are some voltage drop while the electric water heater is drawing power from the source. Meanwhile, while the system is idle, the voltage is at around 220V AC.

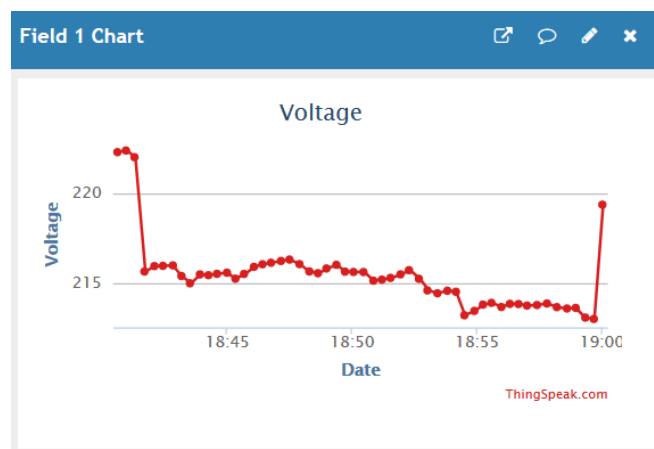


Figure 4.2 Voltage values

The second field contains the values for the current drawn by the system, in the graph, we can see some fluctuations in the value. During normal operation, the heater only takes about 2A of current, while the spike in value is valued at about 220, this could probably be happening because of some technical issues in the data transfer in the MAX RS485, as the Arduino is just receiving the data then sending the data to ThingSpeak and ThingSpeak just displays the received data from the Arduino.

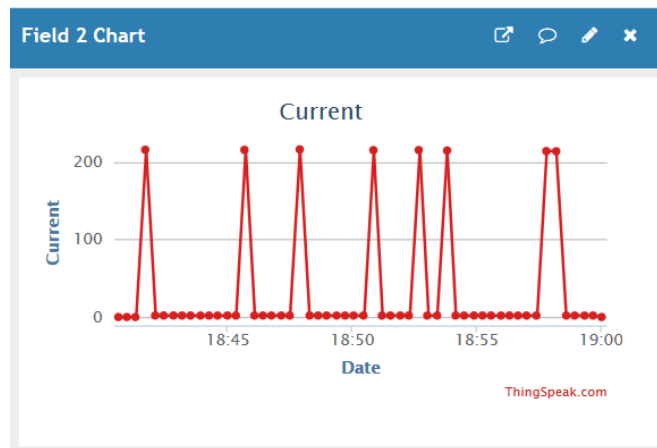


Figure 4.3 Current values

The last field displays the values for the power measurement. The amount of power drawn by the heater during use is shown as the spike from the 0 value up to about 420W, despite the power rating of the water heater of 350W. As we can see from Figure 4.4, the power drawn by the system goes up sharply as soon as the heater is turned on, and as the heater is turned off, the power draw goes back down to 0 instantly.

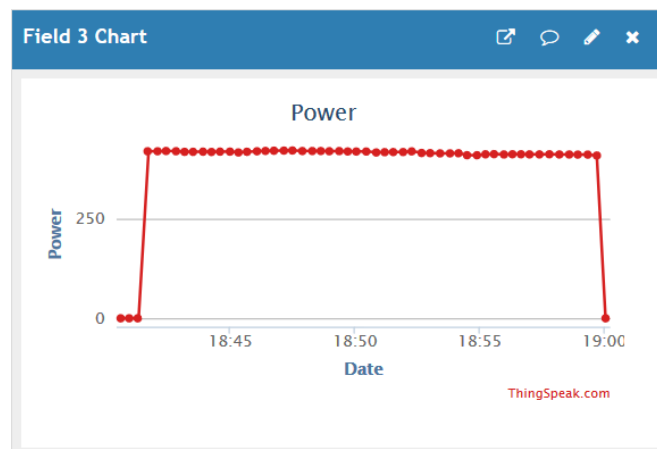


Figure 4.4 Power values

Another experiment is conducted to see the spikes in the current value. The result shows that the spikes in the current value are the same as the values from the voltage field, as shown in Figure 4.5. As stated before, because ThingSpeak is just displaying received data, the fault lies elsewhere. While a probability exists in the error while converting the data types from the SDM 230, the similarity of value from the fault to the Voltage graph makes the most probable issue is the data transfer from the SDM 230 to the Arduino through the MAX RS485, as during the conversion, the Arduino refers to the data received, and if the conversions are faulty, the numbers will not probably be exactly the same as another field's value.

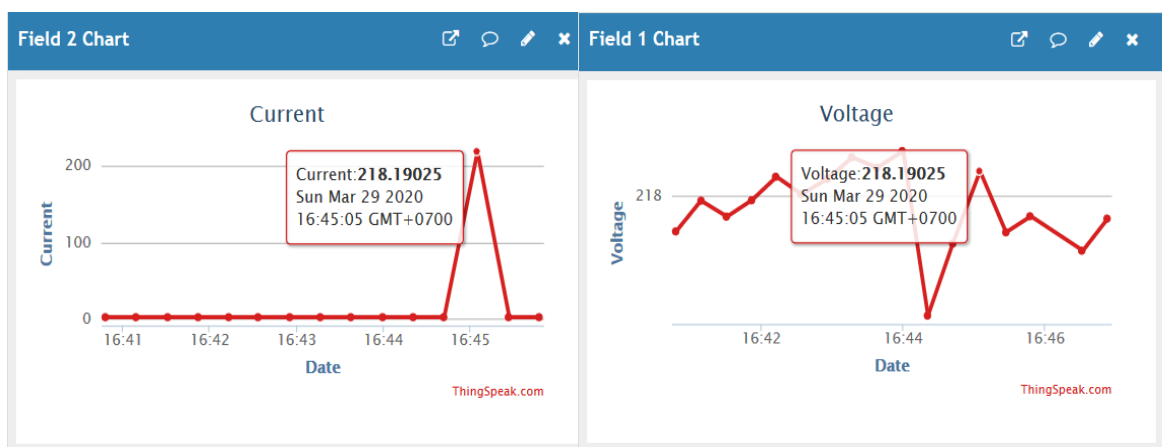


Figure 4.5 Faulty value comparison

Despite the faulty values in the Current field, the Power field are not affected by this error, and still consistently show values around 420W (Figure 4.6). This affirms that the fault is most probably lies during the data transfer from the SDM 230 to the Arduino.

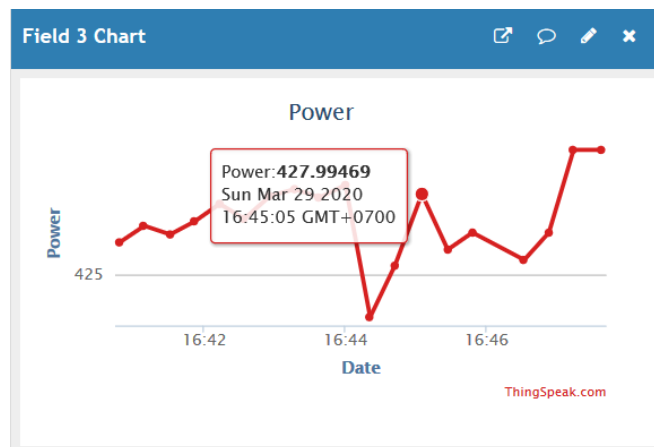


Figure 4.6 Power field stays consistent

Seeing as there are some faults in the current value, another test is conducted to see the accuracy of data delivered from the SDM 230 to the ThingSpeak channel. The test is to see the values displayed on the SDM 230 built-in display and compare it to the one on ThingSpeak. The test will be done using the Power values of the system. As the Arduino is programmed to send data to ThingSpeak every 20 seconds, the test will be done by using a stopwatch to time the data collection from the display of the SDM 230. The test result can be seen in Table 4.1.

SDM 230	Thingspeak	Difference(%)
428	426.19199	0,422432243
428	426.8089	0,278294393
429	426.48605	0,586002331
430	426.97849	0,702676744
432	427.6275	1,012152778
433	427.06918	1,369704388
430	427.92682	0,482134884
429	428.17645	0,191969697
428	427.86505	0,031530374
429	428.33234	0,155631702
428	423.40375	1,073890187
430	425.32275	1,087732558
432	427.99469	0,927155093
428	425.92065	0,485829439
433	426.55707	1,487974596

Table 4.1 Accuracy test results

From the data obtained in the test, the result shows that the accuracy of the data displayed on ThingSpeak in accordance to the data displayed on the SDM 230 is averaging at about 0.68% difference.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

Comparing the Final Project Objectives from Chapter 1 and the findings on Chapter 4, we can conclude as such:

- The goal of constructing a power monitoring system are achieved, as the power monitor itself can monitor the power consumption and display it on a built-in LCD.
- The goal of connecting the power monitoring system to another device and to cloud services are also successfully achieved in this project, using Arduino UNO connected to an Ethernet shield and a laptop for data management and internet connection, and ThingSpeak for data display for remote viewing, with 0.68% accuracy.

5.2 Future Recommendations

There are some possible improvements that can be done in the future for this project to achieve a more accurate and efficient way of power monitoring. Here are three recommendations for future projects:

1. Seeing as the Current readings in the ThingSpeak channel are still sometimes taking readings from the Voltage readings, some hardware changes can be made to avoid such problems.
2. As the project right now relies on the ThingSpeak service for visualization of the data, further improvements can apply a private server or cloud storage and the ability to manually log the data as right now, the ThingSpeak only displays and does not log the data received.

References

- [1] J. G. Webster, "Power Measurement," in *Measurement, Instrumentation, and Sensors Handbook*. IEEE Press 1999, pp. 1223-1227
- [2] R. D. Lestari, "Sistem Pemantauan Daya Berbasis Internet of Things," thesis S.T., Fakultas Teknik Elektro, Universitas Telkom, 2018.
- [3] P. Keyur, P. Sunil,"*Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & future Challenges*," International Journal of engineering Science and Computing, 6(5), May 2016, pp. 6112-6131
- [4] Dr. Ovidiu Vermesan SINTEF, Norway, Dr. Peter FriessEU, Belgium, "*Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*", river publishers' series in communications, 2013.
- [5] B. Jon, (2007), "*The Basics of Power Monitoring Systems*", [online]. Available: <https://www.ecmweb.com/maintenance-repair-operations/metering-monitoring/article/20896291/the-basics-of-power-monitoring-systems>
- [6] "Return Codes" in *ThingSpeak Communication Library for Arduino, ESP8266 and ESP32*, [online]. Available: <https://github.com/mathworks/ThingSpeak-arduino>

APPENDIX

Program code

```
#include <SPI.h>
#include <Ethernet.h>
#include <ThingSpeak.h>
#include <ModbusMaster.h>

#define MAX485_DE 3
#define MAX485_RE_NEG 2

//-----Local Network Settings-----//
byte mac[] = {0x90, 0xA2, 0xDA, 0x10, 0x40, 0x4F};

//-----Variable Setup-----//
uint8_t voltageRead, currentRead, powerRead;
union {
    uint16_t vu[2];
    float vf;
} voltage;
union {
    uint16_t cu[2];
    float cf;
} current;
union {
    uint16_t pu[2];
    float pf;
} power;
unsigned long myChannelNumber = 789446;
unsigned long UpdateInterval = 0;
const char * myWriteAPIKey = "06612TQ1D4C8LEJH";
```

```

//-----Initialization-----//
IPAddress ip(192, 168, 137, 177);
EthernetClient client;
ModbusMaster node;

void preTransmission()
{ digitalWrite(MAX485_RE_NEG, HIGH);
  digitalWrite(MAX485_DE, HIGH);
}

void postTransmission()
{ digitalWrite(MAX485_RE_NEG, LOW);
  digitalWrite(MAX485_DE, LOW);
}

void setup() {
  Ethernet.begin(mac, ip);
  ThingSpeak.begin(client);
  pinMode(MAX485_RE_NEG, OUTPUT);
  pinMode(MAX485_DE, OUTPUT);
  digitalWrite(MAX485_RE_NEG, 0);
  digitalWrite(MAX485_DE, 0);
  Serial.begin(1200);
  node.begin(1, Serial);
  node.preTransmission(preTransmission);
}

```

```

node.postTransmission(postTransmission);
}

void loop() {

//-----Defining Float Unions-----//
voltageRead = node.readInputRegisters (0x30000, 2);
if (voltageRead == node.ku8MBSuccess) {
    voltage.vu[1] = node.getResponseBuffer(0);
    voltage.vu[0] = node.getResponseBuffer(1);
}

currentRead = node.readInputRegisters (0x30006, 2);
if (voltageRead == node.ku8MBSuccess) {
    current.cu[1] = node.getResponseBuffer(0);
    current.cu[0] = node.getResponseBuffer(1);
}

powerRead = node.readInputRegisters (0x30012, 2);
if (powerRead == node.ku8MBSuccess) {
    power.pu[1] = node.getResponseBuffer(0);
    power.pu[0] = node.getResponseBuffer(1);
}

//-----Setting Up String Conversion-----//
char v_buffer[10];
char c_buffer[10];
char p_buffer[10];
float V = voltage.vf;
float C = current.cf;

```

```
float P = power.pf;
String strVoltage = dtostrf(V, 0, 5, v_buffer);
String strCurrent = dtostrf(C, 0, 5, c_buffer);
String strPower = dtostrf(P, 0, 5, p_buffer);

//-----Setting ThingSpeak Fields-----//
ThingSpeak.setField(1, strVoltage);
ThingSpeak.setField(2, strCurrent);
ThingSpeak.setField(3, strPower);

//-----Send Updates to ThingSpeak Fields-----//
int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if (x == 200) {
    Serial.println("Channel update successful.");
}
else {
    Serial.println("Problem updating channel. HTTP error code " + String(x));
}

delay(20000);

}
```