

# IOT SMART METER FOR ELECTRICITY CONSUMERS

By

Md. Wali-ur- Rahman<sup>1</sup>

15121031

Shahrukh Sattar Chowdhury<sup>2</sup>

15101103

Syed Istiak Ahmed<sup>1</sup>

15121043

Simon Siddique<sup>1</sup>

12221053

A thesis submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree of

<sup>1</sup>Bachelor of Science in Electrical and Electronic Engineering

<sup>2</sup>Bachelor of Science in Computer Science and Engineering

<sup>1</sup>Department of Electrical and Electronic Engineering

BRAC University

April 2019

<sup>2</sup>Department of Computer Science and Engineering

BRAC University

April 2019

## **Declaration**

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at BRAC University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**

---

**Md. Wali-ur- Rahman**

15121031

---

**Shahrukh Sattar Chowdhury**

15101103

---

**Syed Istiak Ahmed**

15121043

---

**Simon Siddique**

12221053

## **Approval**

The thesis/project titled “IoT Smart Meter for Electricity Consumers” submitted by

1. Md. Wali-ur-Rahman (15121031)
2. Shahrukh Sattar Chowdhury (15101103)
3. Syed Istiak Ahmed (15121043)
4. Simon Siddique (12221053)

of Spring, 2019 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of <sup>1</sup>Department of Electrical and Electronic Engineering and <sup>2</sup>Department of Computer Science and Engineering on 25-4-2019.

### **Examining Committee:**

Supervisor:

(Member)

---

Prof. Dr. S. M. Lutful Kabir

Professor, IICT

BUET

Departmental Head:

(Chair)

---

Prof. Dr. Shahidul Islam Khan

Professor, EEE

BRAC University

## **Abstract/ Executive Summary**

The purpose of this project is to develop an IoT electric meter, which would collect all the meter data. The IoT meter will send the necessary meter data filtering out irrelevant ones to a central server via a wireless communication mean. The data of all the meters will be processed in the central server, and decisions should be taken using the central server. If any sort of tampering is attempted, the server should give an immediate automated response and turn off the power supply. Also, if the bills are due, the server would warn the user, and then if necessary would cut off the line. Last time, when we demonstrated our work to our respected supervisor, the Energy measurements were not properly calibrated and not giving expected readings. And without his approval to our solution to this problem, we proceeded with our thesis defense.

**Keywords:** IoT; Electricity; Meter; ADE7758; ESP8266.

## **Acknowledgement**

We would like to thank Prof. Dr. S. M. Lutful Kabir, Professor, Institute of Information and Communication Technology, BUET; for his supportive guidance and feedback which led us to the completion of this thesis. Without his assistance and perseverance it would be quite hard to accomplish this task. We would also like to thank Engr. Rashedul Hasan, for his continuous co-operation and perseverance throughout the research.

## **Table of Contents**

<b>Declaration .....</b>	<b>ii</b>
<b>Approval.....</b>	<b>iii</b>
<b>Abstract/ Executive Summary .....</b>	<b>iv</b>
<b>Acknowledgement .....</b>	<b>v</b>
<b>Table of Contents .....</b>	<b>vi</b>
<b>List of Tables .....</b>	<b>xiii</b>
<b>List of Figures.....</b>	<b>xv</b>
<b>List of Acronyms .....</b>	<b>xviii</b>
<b>Chapter 1 Introduction .....</b>	<b>1</b>
<b>    1.1 Introduction to Energy Metering.....</b>	<b>1</b>
<b>        1.1.1 Analog Meters.....</b>	<b>2</b>
<b>        1.1.2 Digital Electronic Meters .....</b>	<b>3</b>
<b>    1.2 Introduction to IoT.....</b>	<b>4</b>
<b>    1.3 Few Examples of IoT Applications .....</b>	<b>6</b>
<b>        1.3.1 IoT Smart Home System .....</b>	<b>6</b>
<b>        1.3.2 IoT Smart Grid System .....</b>	<b>7</b>
<b>    1.4 Motivation.....</b>	<b>10</b>
<b>        1.4.1 Importance of IoT Smart Energy Meter in Bangladesh .....</b>	<b>11</b>

<b>1.5 Objective.....</b>	<b>12</b>
<b>1.6 Scopes in Our Country.....</b>	<b>13</b>
<b>Chapter 2 Proposed Design of the system .....</b>	<b>14</b>
<b>Chapter 3 Meter.....</b>	<b>16</b>
<b>    3.1 Introduction to Meter Integrated Circuits (ICs).....</b>	<b>16</b>
<b>    3.2 Meter ICs Studied .....</b>	<b>17</b>
<b>    3.3 Comparison of Meter ICs.....</b>	<b>21</b>
<b>    3.4 Meter IC Selected .....</b>	<b>25</b>
<b>    3.5 ADE7758: Poly Phase Multifunction Energy Metering IC with Per Phase Information .....</b>	<b>25</b>
<b>        3.5.1 ADE7758 Test Circuit and Testing Procedure .....</b>	<b>27</b>
<b>        3.5.2 Registers Used for Measurement .....</b>	<b>28</b>
<b>        3.5.3 Calibration.....</b>	<b>29</b>
<b>Chapter 4 Serial Peripheral Interface (SPI) with ADE7758 .....</b>	<b>31</b>
<b>    4.1 Data Transmission.....</b>	<b>31</b>
<b>        4.1.1 Clock Signal.....</b>	<b>32</b>
<b>        4.1.2 Slave Select .....</b>	<b>32</b>
<b>        4.1.3 Multiple Slaves .....</b>	<b>33</b>
<b>        4.1.4 Steps of Data Transferring.....</b>	<b>33</b>
<b>        4.1.5 MOSI and MISO .....</b>	<b>35</b>

<b>4.1.6 Advantages.....</b>	<b>35</b>
<b>4.1.7 Disadvantages .....</b>	<b>35</b>
<b>4.2 SPI Registers.....</b>	<b>35</b>
<b>4.3 SPI Connection between ADE7758 and ATmega32.....</b>	<b>36</b>
<b>4.3.1 Circuit Diagram.....</b>	<b>36</b>
<b>4.3.2 ADE7758 Timing Diagram.....</b>	<b>37</b>
<b>4.4 SPI Initial Code Analysis .....</b>	<b>38</b>
<b>4.4.1 Writing Data.....</b>	<b>40</b>
<b>4.4.2 Reading Data .....</b>	<b>41</b>
<b>Chapter 5 Communication with the Server using ESP-12E.....</b>	<b>43</b>
<b>5.1 Introduction to ESP8266.....</b>	<b>43</b>
<b>5.1.1 Pin Description .....</b>	<b>44</b>
<b>5.1.2 ESP-12E Description .....</b>	<b>45</b>
<b>5.1.3 Pin Description for ESP-12E.....</b>	<b>46</b>
<b>5.2 USART Communication between ESP-12E and ATmega32 .....</b>	<b>47</b>
<b>5.2.1 Circuit Diagram for ESP-12E and ATmega32.....</b>	<b>48</b>
<b>5.3 USART Communication Overview .....</b>	<b>49</b>
<b>5.4 USART Settings Code Breakdown .....</b>	<b>51</b>
<b>5.5 AT Commands Used.....</b>	<b>52</b>
<b>5.5.1 AT Commands used in Wifi Methods.....</b>	<b>52</b>

<b>5.6 AVR-WIFI Library Code Breakdown .....</b>	<b>54</b>
<b>    5.6.1 Wifi Send Command Functions .....</b>	<b>54</b>
<b>    5.6.2 Wifi Initialization .....</b>	<b>55</b>
<b>    5.6.3 Wifi Connect.....</b>	<b>55</b>
<b>    5.6.4 Wifi Link Open.....</b>	<b>56</b>
<b>    5.6.5 Wifi Send Function.....</b>	<b>56</b>
<b>    5.6.6 Wifi Listen (ESP Response Code).....</b>	<b>57</b>
<b>Chapter 6 Importance of RTC.....</b>	<b>59</b>
<b>    6.1 Brief Discussion on TWI Communication Protocol for RTC .....</b>	<b>59</b>
<b>    6.2 TWI Registers.....</b>	<b>60</b>
<b>    6.3 RTC Code Breakdown .....</b>	<b>61</b>
<b>Chapter 7 Server .....</b>	<b>63</b>
<b>    7.1 Introduction to Server System .....</b>	<b>63</b>
<b>    7.2 Insightful Diagrams to the project.....</b>	<b>65</b>
<b>        7.2.1 Sequence Diagram .....</b>	<b>66</b>
<b>        7.2.2 Entity Relationship Diagram .....</b>	<b>67</b>
<b>    7.3 Tier Based Cost .....</b>	<b>70</b>
<b>    7.4 Relay Enabler Diagram.....</b>	<b>71</b>
<b>    7.5 Server Part Code Breakdown .....</b>	<b>72</b>
<b>        7.5.1 Controller Part .....</b>	<b>72</b>

<b>7.5.2 Model .....</b>	<b>75</b>
<b>7.5.3 View .....</b>	<b>76</b>
<b>7.6 Database.....</b>	<b>76</b>
<b>Chapter 8 Website.....</b>	<b>82</b>
<b>8.1 Introduction.....</b>	<b>82</b>
<b>8.2 User-end.....</b>	<b>82</b>
<b>    8.2.1 Dashboard.....</b>	<b>82</b>
<b>    8.2.2 Account .....</b>	<b>84</b>
<b>    8.2.3 Consumptions .....</b>	<b>85</b>
<b>    8.2.4 Billing.....</b>	<b>87</b>
<b>8.3 Admin-End .....</b>	<b>88</b>
<b>Chapter 9 Final Program and Circuit Integration .....</b>	<b>91</b>
<b>9.1 Description of the Established System.....</b>	<b>91</b>
<b>9.2 Main Code of the System .....</b>	<b>93</b>
<b>    9.2.1 Main Program Code Events.....</b>	<b>94</b>
<b>    9.2 Server Side of the System.....</b>	<b>97</b>
<b>        9.2.1 Server Side Code Events .....</b>	<b>97</b>
<b>9.3 Results.....</b>	<b>100</b>
<b>Chapter 10 Conclusion.....</b>	<b>102</b>
<b>10.1 Shortcomings .....</b>	<b>102</b>

<b>10.2 Solutions.....</b>	<b>103</b>
<b>10.3 Summary.....</b>	<b>104</b>
<b>References.....</b>	<b>106</b>
<b>Appendix A.....</b>	<b>111</b>
<b>Appendix B.....</b>	<b>112</b>
<b>Appendix C.....</b>	<b>113</b>
<b>Appendix D.....</b>	<b>116</b>
<b>Appendix E.....</b>	<b>117</b>
<b>Appendix F .....</b>	<b>117</b>
<b>Appendix G .....</b>	<b>118</b>
<b>Appendix H .....</b>	<b>119</b>
<b>Appendix I.....</b>	<b>120</b>
<b>Appendix J.....</b>	<b>121</b>
<b>Appendix K .....</b>	<b>121</b>
<b>Appendix L.....</b>	<b>122</b>
<b>Appendix M.....</b>	<b>122</b>
<b>Appendix N.....</b>	<b>123</b>
<b>Appendix O .....</b>	<b>124</b>
<b>Appendix P .....</b>	<b>124</b>
<b>Appendix Q .....</b>	<b>125</b>

<b>Appendix R.....</b>	<b>126</b>
<b>Appendix S .....</b>	<b>126</b>
<b>Appendix T .....</b>	<b>127</b>
<b>Appendix U.....</b>	<b>127</b>
<b>Appendix V.....</b>	<b>128</b>
<b>Appendix W.....</b>	<b>129</b>
<b>Appendix X.....</b>	<b>130</b>
<b>Appendix Y.....</b>	<b>131</b>
<b>Appendix Z.....</b>	<b>131</b>
<b>Appendix AA.....</b>	<b>132</b>
<b>Appendix BB .....</b>	<b>132</b>
<b>Appendix CC .....</b>	<b>133</b>
<b>Appendix DD .....</b>	<b>134</b>
<b>Appendix EE .....</b>	<b>135</b>
<b>Appendix FF .....</b>	<b>136</b>
<b>Appendix GG .....</b>	<b>140</b>
<b>Appendix HH .....</b>	<b>140</b>
<b>Appendix II .....</b>	<b>141</b>
<b>Appendix JJ.....</b>	<b>142</b>
<b>Appendix KK .....</b>	<b>143</b>

<b>Appendix LL .....</b>	<b>144</b>
<b>Appendix MM .....</b>	<b>146</b>
<b>Appendix NN:.....</b>	<b>148</b>
<b>Appendix OO .....</b>	<b>149</b>

## **List of Tables**

Table 3. 1: Meter IC Comparison [17], [18], [19], [20] .....	24
Table 5. 1: ESP-12E Pin Description.....	47
Table 5. 2: UBRR values for 16 MHz Oscillator Frequency [24] .....	50
Table 5. 3: AT Commands in Wifi Methods .....	51
Table 6. 1: TWI Terminology [24] .....	59
Table 7. 1: Consumptions table .....	79
Table 7. 2: Invoices table .....	79
Table 7. 3: Meter Table .....	80
Table 7. 4: User Table .....	80
Table 8. 1: Usage Table.....	91

Table 9. 1: VRMS Data from the AVRMS Register and the Calibrated Actual VRMS Values for 200W Load .....	100
Table 9. 2: VRMS Data from the AVRMS Register and the Calibrated Actual VRMS Values for 400W Load .....	100
Table 9. 3: VRMS Data from the AVRMS Register and the Calibrated Actual VRMS Values for 500W Load .....	101
Table 9. 4: IRMS Data from the AIRMS Register and the Calibrated Actual IRMS Values for 200W Load .....	101
Table 9. 5: IRMS Data from the AIRMS Register and the Calibrated Actual IRMS Values for 400W Load .....	101
Table 9. 6: IRMS Data from the AIRMS Register and the Calibrated Actual IRMS Values for 500W Load .....	101

## List of Figures

Figure 1. 1: Electromechanical Induction Meter [14] .....	2
Figure 1. 2: Digital Electronic Meter [15].....	3
Figure 1. 3: IoT Graphic [13] .....	4
Figure 1. 4: Smart Home Graphic [10] .....	6
Figure 1. 5: Smart Grid Graphic [11].....	8
Figure 2. 1: Block diagram representing the proposed design of the system.....	15
Figure 3. 1: Functional Block Diagram of ADE7751 [17].....	18
Figure 3. 2: Functional Block Diagram of ADE7753 [18].....	19
Figure 3. 3: Functional Block Diagram of ADE7754 [19].....	20
Figure 3. 4: Functional Block Diagram of ADE7758 [20].....	21
Figure 3. 5: ADE 7758 Pin Configuration [20] .....	26
Figure 3. 6: Test Circuit for ADE7758 IC [20] .....	27
Figure 3. 7: The ADE7758 Meter IC circuit implemented on a breadboard connected with the MCU and a load-board via the CT.....	30
Figure 4. 1: SPI Communication Overview [3].....	31
Figure 4. 2: Single Master Multiplane Slave SPI [1] .....	33
Figure 4. 3: Single Master Multiple Slave SPI [1].....	33
Figure 4. 4: Master Sets SS low [1] .....	34
Figure 4. 5: MOSI sends Data Bits [1].....	34

Figure 4. 6: MISO receives Data Bits from Slave [1] .....	34
Figure 4. 7: SPI Pin Connection .....	36
Figure 4. 8: Serial Write Diagram for ADE7758 [25] .....	37
Figure 4. 9: Serial Read Diagram for ADE7758 [25] .....	37
Figure 4. 10: MOSI and SCLK in the Oscilloscope .....	39
Figure 4. 11: MOSI and SCLK in the Oscilloscope .....	39
Figure 4. 12: ADE7758 Write data Code Flowchart .....	40
Figure 4. 13: ADE7758 Read Data Code Flowchart .....	42
Figure 5. 1: Esp-01 Module [5] .....	43
Figure 5. 2: Generic ESP Module [6].....	44
Figure 5. 3: ESP-12E [7] .....	45
Figure 5. 4: ESP-12E Pin Description [7] .....	46
Figure 5. 5: AMS1117 5V Module [35].....	48
Figure 5. 6: Connection between ESP-12E and ATmega32 .....	49
Figure 5. 7: Wifi Connect Flowchart .....	56
Figure 5. 8: Data Sending Flowchart .....	57
Figure 6. 1: DS3231 RTC Module [36] .....	59
Figure 6. 2: Data Validity for TWI [24].....	60
Figure 7. 1: Sequence Diagram.....	66
Figure 7. 2: Entity Relationship Diagram .....	67

Figure 7. 3: Tier Based Cost Flowchart .....	70
Figure 7. 4: Relay Enabler Diagram .....	71
Figure 7. 5: Consumption controller Flow-Chart .....	73
Figure 7. 6: Models .....	75
Figure 7. 7: Database Management System [33] .....	77
Figure 7. 8: Data Tables .....	78
Figure 8. 1: Dashboard upper part .....	82
Figure 8. 2: Graphical representation of the usage .....	83
Figure 8. 3: Account Information .....	84
Figure 8. 4: Password Reset Section .....	85
Figure 8. 5: Manage meter section of end-user .....	85
Figure 8. 6: Customized Graph with accordance to timeline .....	86
Figure 8. 7: Billing section .....	87
Figure 8. 8: Admin Dashboard .....	88
Figure 8. 9: Manage Users Section .....	88
Figure 8. 10: View Page .....	89
Figure 10. 11: Alphanumeric Keyboard [34] .....	103
Figure 9. 1: Full Circuit PCB Design.....	92
Figure 9. 2: Full Circuit PCB.....	93
Figure 9. 3: Main MCU program flow chart .....	96
Figure 9. 4: Flow Chart of the main server side program .....	99

## List of Acronyms

IoT	Internet of Things
ADC	Analog to Digital Converter
RTC	Real Time Clock
LCD	Liquid Crystal Display
RFID	Radio Frequency Identification
ITU	International Telecommunication Union
WSIS	World Summit on the Information Society
Wi-Fi	Wireless Fidelity
2G	2 <sup>nd</sup> Generation
3G	3 <sup>rd</sup> Generation
4G	4 <sup>th</sup> Generation
IC	Integrated Circuit
ADE	Analog Devices Energy

DSP	Digital Signal Processing
SPI	Serial Peripheral Interface
DIP	Dual In-line Package
SSOP	Shrink Small-Outline Package
RMS	Root Mean Square
APCF	Active Power Calibration Frequency
VARCF	VAR Calibration Frequency
IRQ	Interrupt Request
PGA	Pin Grid Array
CMOS	Complementary Metal-Oxide-Semiconductor
MCU	Microcontroller Unit
SOIC	Small Outline Integrated Circuit
LSB	Least Significant Bit
CLKIN	Clock in
CT	Current Transformer
SCLK	Clock signal line
MOSI	Master Output Slave Input
MISO	Master Input Slave Output

SS	Slave Select
UART	Universal Asynchronous Receiver Transmitter
CPOL	Clock Polarity
CPHA	Clock Phase
MSB	Most Significant Bit
GPIO	General Purpose Input Output
ESP	Espressif
PWM	Pulse Width Modulation
SOC	System on Chip
AT	Attention
USART	Universal Synchronous Asynchronous Receiver Transmitter
TWI	Two Wire Interface
LAN	Local Area Network
WAN	Wide Area Network
HTTP	HyperText Transfer Protocol
SMTP	Simple Mail Transfer Protocol
LED	Light Emitting Diode
RAM	Random Access Diode

SSD	Solid State Drive
HTML	Hypertext Markup Language
ASP	Active Server Pages
PHP	Hypertext Preprocessor
TCP	Transmission Control Protocol
IP	Internet Protocol
FTP	File Transfer Protocol
MVC	Model View Controller
SQL	Structured Query Language

# **Chapter 1**

## **Introduction**

### **1.1 Introduction to Energy Metering**

Ever since 1882, when Edison Electric Illuminating Company of New York first supplying electricity to homes of Manhattan, it has played the most important part in the modernization our lives. For a developing country like Bangladesh ensuring availability of electricity in every corner of our country is a huge challenge. Approximately 77.9% of the people in the country have access to electricity [14]. Now the new challenge is to make the best use of the electricity available. Even at recent times, supply of electricity was administered fully by a traditional analog electric meter in our country. However, these meters are slowly declining in popularity with the advancement of technology. The main problem regarding these meters are that they can only give consumers monthly consumption readings. Therefore the consumers are unaware of their daily usage of electricity. So, consumers may unwillingly at times misuse and waste a large amount of electricity which is a large drawback for the energy sector of this developing nation. However, smart meters overcome the limitations of the analog meters by providing energy data on a daily basis. It can make people aware of their usage and ensure the efficient use of electricity. Also, as consumers start using electricity in a smarter more efficient manner, less electricity will be consumed and bills will lower as well. Wastage of electricity would be also be reduced sharply and as a result lot more people would get access to electricity.

### 1.1.1 Analog Meters



*Figure 1. 1: Electromechanical Induction Meter [14]*

When the distribution of electricity first started, people used a traditional analog meter to measure the amount of electricity used. A traditional analog meter is actually the electromechanical induction type watt-hour meter. And it's the most common type of watt-hour meter and has been in use for a long time now [2]. It has a rotating aluminum disc mounted on a spindle between two electromagnets, speed of rotation of the disc is proportional to the power and this power is integrated by the use of counter mechanism and gear trains. It has two silicon steel laminated electromagnets called series and shunt magnets. Firstly, the series magnet has a coil which has few turns of thick wire connected in series with line whether shunt magnet carries coil with many turns of thin wire connected across the supply. Breaking magnet is a type of permanent magnet which moves the disc to a balanced position by applying a force opposite to normal disc rotation and also use that to stop the disc while power is off. Again, series magnet produces the flux that is proportional to the current flowing whether shunt magnet produces the flux proportional to the voltage. Both of the fluxes lag only by 90 degrees due to their inductive nature. From the interaction of these two fields generates eddy current in the disk, exerting a force that is proportional to the product of instantaneous voltage, current, and phase angle between them. The

shaft of the aluminum disc is connected in such an arrangement that it records a number that is proportional to the number of revolutions of the disc [4]. Then this gear arrangement sets the number in a series of dials and indicates the energy consumed over time. This type of meter is simple to construct and less accurate due to creeping and other external fields. A major problem of these types of meters is they can be tampered very easily, so an external monitoring system is required for security. As these types of meters are very commonly used in domestic and industrial applications [4]. Figure 1.1 [14] shows an Electromechanical Induction Meter.

### 1.1.2 Digital Electronic Meters



Figure 1. 2: Digital Electronic Meter [15]

These meters use high performing microprocessors and digital signal processing. The voltage and current transducers are connected to high resolution analog to digital converter (ADC). The voltage and current data are multiplied and integrated by digital circuitry to measure the energy information after they are converted to digital samples from analog signal. These meters also can measure the reactive power using the phase angle between current and voltage. The microprocessor does this. These meters take into account the tariff, power factor, peak hours etc.

and is programmed to bill accordingly. Real time clock (RTC), liquid crystal display (LCD) and communication devices are usually also included with these meters [4]. Figure 1.2 [15] shows a digital electronic meter widely used in Bangladesh.

## 1.2 Introduction to IoT

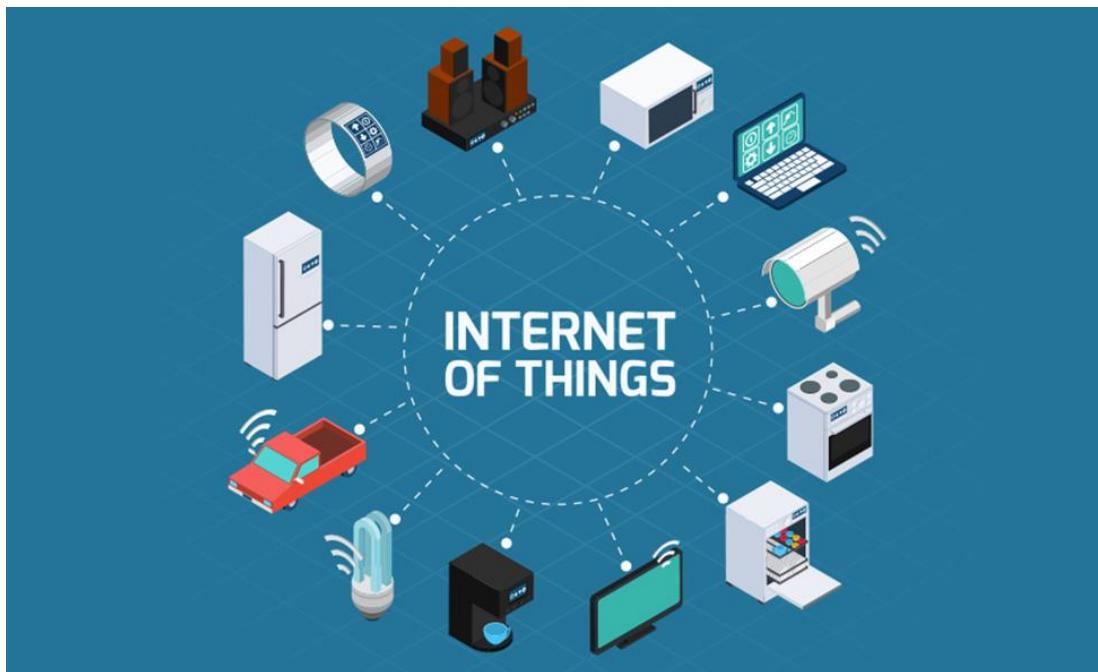


Figure 1. 3: IoT Graphic [13]

IoT is the abbreviation of the Internet of Things. IoT is the means of connecting various physical devices and objects throughout our surroundings to the internet. So, anyone can use connected devices by accessing the internet with their unique identification. The earliest examples of smart devices come when Carnegie Mellon University modified a soda vending machine to connect with the internet and report whether sodas were cold or hot. In 1999, the Auto-ID lab in Massachusetts Institute of Technology (MIT) first came up with the name of “Internet of Things” [12]. Kevin Ashton, the one who proposed liked the idea that every device can be controlled by RFID (Radio-

frequency Identification) which will be run by computer. Then, a research article defining the Internet of Things was submitted in the Nordic Researchers conference which was held in Norway in June 2002 [12]. After that, ITU (International Telecommunication Union) of Tunisia issued a report on the World Summit on the Information Society (WSIS) on November 17, 2005, which also further improved in the idea of IoT. Cisco Systems defined the Internet of things as "simply the point in time when more things or objects were connected to the Internet than people" and estimated that the IoT was born between 2008 and 2009 [12].

Internet of Things is primarily the networking of physical objects that contain electronics embedded within the design architecture that enables them to communicate and sense interactions amongst each other and human user. IoT connection needs four fundamental objects to form. They are - data, hardware, software, and connectivity. The hardware connects digital devices with physical devices. Data carries information that shapes and forms IoT. Software part is used to analyze the data part. Like – Facebook, Google are good examples of software. They analyze data and create meaning out of the communication. This creates meaning out of the data from the hardware and completes the task that is set by the user. Connectivity connects the hardware and software with each other. It facilitates the transfer of information between the devices. Ethernet was the first connecting way then Wi-Fi, 2G, 3G, and 4G followed. As technology is upgrading it is growing cheaper and faster. IoT is represented by using a graphic in Figure 1.3 [13].

## 1.3 Few Examples of IoT Applications

### 1.3.1 IoT Smart Home System

Among all IoT based systems smart home system is the most popular, important and efficient application that stands out the most. Smart Home technology is the future of residential-related technology designed to deliver and distribute a number of services inside and outside the house through networked devices that integrate and interconnect all the various applications and intelligence behind them [37]. Figure 1.4 [10] displays a smart home graphic.



Figure 1. 4: Smart Home Graphic [10]

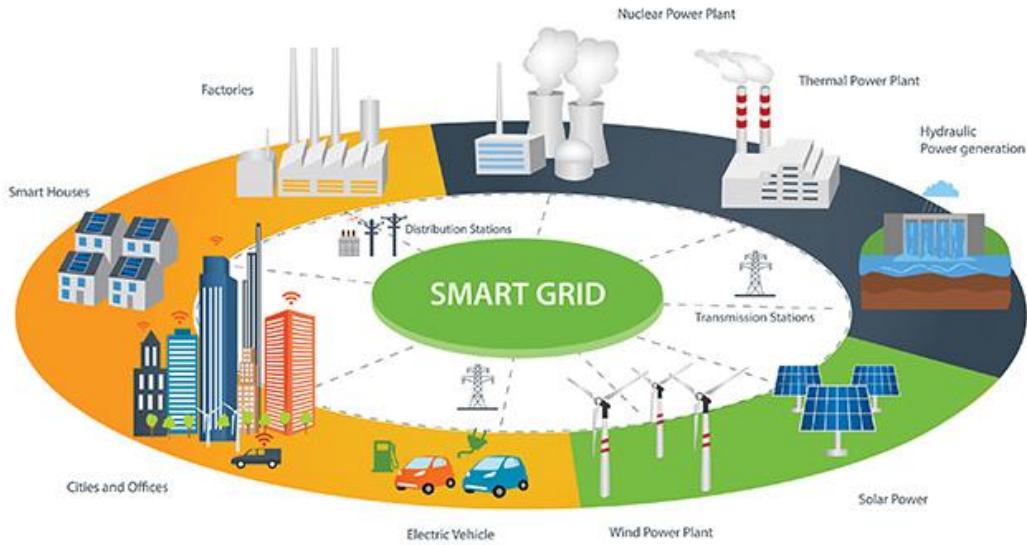
Common accessories of smart home features:

- Smart lighting systems, like Hue from Philips Lighting Holding B.V., can detect when peoples present in the room and adjust lighting accordingly. Smart light bulbs can also sense daylight and regulate as per needed.
- Smart TVs connect to the internet to see channels, YouTube video, and music. Some smart TVs also detect voice or gesture recognition.

- Smart locks and openers can detect people and grant or deny access. Then use face or figure detectors to check.
- Smart thermostats, such as Nest from Nest Labs Inc., have integrated Wi-Fi, allows a user to schedule monitor and remotely control the home temperature.
- Smart security cameras enable residents can monitor their homes when they are on work or on vacation.
- Smart motion sensors can identify the difference between residents, visitors, pets and burglars, and can call the police if something happens.
- Smart gardening equipment automatically mows the lawn, waters trees, etc.
- Smart refrigerators that keep track of item expiration dates, detect depleted item or make shopping lists; Smart toaster toast at a fixed time, smart microwave ovens detect if the food is heated or cooked, then it stops.
- Household system monitors may, for example, sense high and low voltage and turn off all electronics or sense water failures or freezing pipes and turn off the water so there isn't a flood in your basement.

### **1.3.2 IoT Smart Grid System**

The smart grid is an electrical grid which includes various operation and energy measures. It includes smart meters, smart appliances, renewable energy resources, and energy efficient resources with its network. IoT can be used for various smart grid applications such as monitoring power plants, predicting power generation and consumption, power consumption monitoring, energy storage monitoring, power demand side management and various energy production areas [38]. Smart grid is graphically represented in Figure 1.5 [11].



*Figure 1. 5: Smart Grid Graphic [11]*

After the invention of the internet, it was possible to communicate clearly from any places from the world. As IoT is advancing smart controlled devices increased. So, after developing Smart Phone and Smart Homes, people now are trying to make use of IoT on a grander scale. That is when smart grid theory kicked in. It was first proposed in the USA as “Energy Independence and Security Act of 2007” [11]. The proposition they gave are as follows:

"It is the policy of the United States to support the modernization of the Nation's electricity transmission and distribution system to maintain a reliable and secure electricity infrastructure that can meet future demand growth and to achieve each of the following, which together characterize a Smart Grid:

- (1) Increased use of digital information and controls technology to improve reliability, security, and efficiency of the electric grid.
- (2) Dynamic optimization of grid operations and resources, with full cyber-security.
- (3) Deployment and integration of distributed resources and generation, including renewable resources.
- (4) Development and incorporation of demand response, demand-side resources, and energy-efficiency resources.
- (5) Deployment of 'smart' technologies (real-time, automated, interactive technologies that optimize the physical operation of appliances and consumer devices) for metering, communications concerning grid operations and status, and distribution automation.
- (6) Integration of 'smart' appliances and consumer devices.
- (7) Deployment and integration of advanced electricity storage and peak-shaving technologies, including plug-in electric and hybrid electric vehicles, and thermal storage air conditioning.
- (8) Provision to consumers of timely information and control options.
- (9) Development of standards for communication and interoperability of appliances and equipment connected to the electric grid, including the infrastructure serving the grid.
- (10) Identification and lowering of unreasonable or unnecessary barriers to adoption of smart grid technologies, practices, and services." [11]

After that, The European Union Commission Task Force also gave a proposition for the smart grid [11]. Their definition of the smart grid is:

"A Smart Grid is an electricity network that can cost efficiently integrate the behavior and actions of all users connected to it – generators, consumers and those that do both – in order to ensure economically efficient, sustainable power system with low losses and high levels of quality and security of supply and safety. A smart grid employs innovative products and services together with intelligent monitoring, control, communication, and self-healing technologies in order to:

- Better facilitate the connection and operation of generators of all sizes and technologies.
- Allow consumers to play a part in optimizing the operation of the system.
- Provide consumers with greater information and options for how they use their supply.
- Significantly reduce the environmental impact of the whole electricity supply system.
- Maintain or even improve the existing high levels of system reliability, quality and security of supply.
- Maintain and improve the existing services efficiently." [11]

## **1.4 Motivation**

For a developing country, making the best use of the resources available is one of the keys to advancement. Reducing the electricity wastage and expenditures by making people aware of their usage by a fast and secured medium that can be accessed from everywhere was the motivation of our team. In the case of using the traditional meters, there is wastage of energy and manpower. As conventional meters can only produce the electricity consumption of the household on a monthly basis, there is a scope in this technology to explore a new development for the benefit of the

consumers as well as themselves. Also, for conventional digital meters, you have to go to the meter and look at its reading time and time again to keep tabs on the expenditure. And for analog meters, consumers cannot know how much each unit of current costs and at the end of the month when the sum total of the cost billed, this is a huge drawback for them and more complaints stack up against the authorities. So the only way to keep track of the energy consumption from the user end was through manual data collection which is never a feasible option for any consumer. Therefore, our motivation to start the project is to eliminate these problems and make life easier for everyone.

#### **1.4.1 Importance of IoT Smart Energy Meter in Bangladesh**

Analog electromechanical induction type watt-hour meter is normally used in our country some years ago. Now because of the steps taken by our government, prepaid meters are taking the place of the analog meters. These meters require prepaid bills to be paid to the relevant authorities to start and run. It has a keypad on it. Prepaid meter is directly controlled by the government control station. The consumer has to input various codes in the keypads to recharge balance, check balance, and check electricity consumption. It eliminated most of the accounts received and human involvement of the government. It also has reduced corruption and electric theft in our country as it is very closely monitored by the government.

The type of system we worked on in our thesis is a type of upgrade on the existing system. Our system is run by IoT technology, meaning it is connected to the internet and communicates wirelessly. Aforementioned IoT based smart meters will be connected to the internet server. The consumers will have their unique username and passwords to access the server and see their consumption history and cost at lowest time intervals. As users can access it via the internet, they can check from anywhere. The server will be run by government officials who will oversee any kind of inconvenience in the electric meters installed. It will also increase the security of the meter.

If any problem arises consumers can file a complaint to the electricity provider much sooner by using their server. Unlike existing digital meters users don't have to go to nearest electricity providers' offices to get the prepaid codes, in this meter, they can pay their charges using credit cards or mobile payment systems such as bKash which will be faster and more convenient.

## **1.5 Objective**

The main purpose of this thesis project is to make people aware of their energy consumption in their homes or workplaces to ensure efficient and low-cost use of electricity. We are taking the electricity consumption data from the meter and then upload it to an internet server. Consumers will use their own unique username and passwords to access the server where they can see their daily consumption rate with costing data. Here we want to achieve the following goals:

- Get accurate data readings within the shortest time.
- Keeping a structured database of the consumptions for the user and the electricity provider.
- Creating an opportunity for the administration to constantly monitor meter consumptions and keeping a record of it to reduce corruption in the energy sector.
- Better meter data security
- Smart payment and recharging system for higher convenience.

## **1.6 Scopes in Our Country**

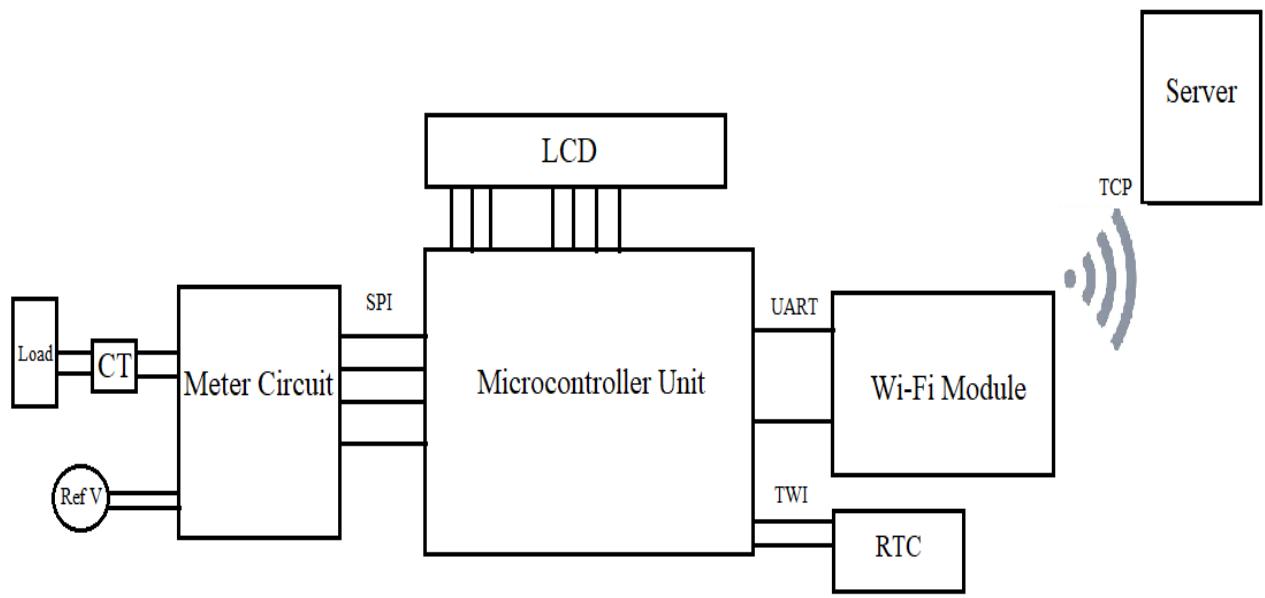
Analog electromechanical induction type watt-hour meter have been normally used in our country for some years. Now because of the steps taken by our government, new smart prepaid meters are taking the place of the analog meters. These meters require prepaid bills to be paid to the relevant authorities to start and run. It has card punching system on it, which is basically RFID system. A Registered Customer has to punch his card containing information on the prepaid balance and validity, to recharge balance, check balance, and check electricity consumption. It eliminated most of the accounts received and human involvement of the government. It also has reduced the corruption and electric theft in our country as it is very closely monitored by the government.

The type of system we worked on in our thesis is a type of upgrade on the existing system. Our system is run by IOT technology, meaning it is connected to internet and communicates wirelessly. Aforementioned IOT based power meters will be connected to a single internet server. The customers will have their unique username and passwords to access the server and see their consumption history and cost in lowest time intervals. As customers can access it by internet, they can check from anywhere. The server will be run by government officials who will oversee any kind of inconvenience in the electric meters installed. It will also increase the security of the meter. If any problem arises customers can file complaint to government much sooner by using their server. Unlike existing smart meters customers don't have to go to nearest electric authority's office to get the prepaid codes, in this meter they can pay their charges using credit cards or systems like BKASH which will be faster and hassle free.

## **Chapter 2**

### **Proposed Design of the system**

The proposed design consists of the Microcontroller Unit (MCU), the Meter Circuit, the Wi-Fi Module, the Server, LCD, RTC and a Current Transformer (CT). The components are connected as per Figure 2.1. Load is connected to the Meter Circuit through a CT. The CT scales the large current values to small standardized values which are proportional to the primary values but easier to measure. A Reference Voltage (220V) is also connected to the Meter Circuit for measuring and calibration purposes. The Meter Circuit is connected to the MCU, and data from the Meter Circuit after measurement is sent by using SPI. This communication is both way, i.e. data can be sent from the meter to the MCU and vice versa. The Wi-Fi Module is connected to the MCU through the TX and RX connections. Data is sent and received both way to and from the MCU to the Wi-Fi Module respectively using UART. The Wi-Fi Module is used to send and receive data to and from the Server respectively by accessing the internet. For this process TCP protocol is used. The Server uses the measurement data to calculate energy readings and present the information online to the user or electricity provider. A RTC is connected to the MCU to keep track of the real time information, it uses TWI to send and receive data to and from the MCU respectively. An LCD is also connected to the MCU for visuals signaling confirmation, connection etc.



*Figure 2. 1: Block diagram representing the proposed design of the system*

The five major segments: Meter Circuit, SPI communication between Meter Circuit and MCU, UART communication between MCU and Wi-Fi Module, TWI communication between RTC and MCU and the server along with the website, of the proposed system has been elaborately discussed in the upcoming chapters.

# **Chapter 3**

## **Meter**

### **3.1 Introduction to Meter Integrated Circuits (ICs)**

Information at this day and age is of immense value. The importance of data collection cannot be underestimated in any way since it is the driving force behind study, further analysis and research. Electricity providers or companies understand the benefits of data and therefore their demand for additional information from energy meters is always on the rise. Specifically, energy meters used in residential premises are often restricted to kilowatt-hours. Reactive energy measurement, multi tariff billing, data security and power quality measurement are a few of many new and popular properties that can be added as an extension along with the energy data with more information from the meter. These features will strongly revamp the production of electricity, its distribution, troubleshooting and electricity billing. Furthermore, not to mention, more information leads to more accurate energy data. But these high requirements cannot be achieved through the use of electromechanical meters. Thus the meter manufacturing industry has come up with fully electronic solutions in the form of energy measurement integrated circuits (ICs). These ICs possess a much higher degree of accuracy, reliability and power along with the attractive additional features [16].

The most prominent developer of energy meter ICs is Analog Devices. They put forth their expertise in designing high performance analogue to digital converter (ADC) to evolve electronic energy meters. The ADE (Analog Devices Energy) measurement ICs integrate highly accurate ADCs with fixed function digital signal processing (DSP) [16]. They directly interface with current and voltage sensors and execute all the necessary calculations for metering purposes. A Serial Port

Interface (SPI) is also available on some of the ADE ICs [16]. This allows communication with a microprocessor to administer the functionality, execute digital calibration, and interpret the measurement results. This is huge positive because meter designers will now no more need to spend hours developing algorithms in microprocessors to do the calculations and calibrations. Moreover, solutions with high functionality can also be implemented in much shorter time.

### **3.2 Meter ICs Studied**

There are a number of ADE ICs available. But all the different types consist of distinct additional features along with their basic metering capacity. Single phase metering ICs; polyphase metering ICs; metering ICs with active, reactive and apparent energy measurements; metering ICs with only active energy measurements; low power metering ICs; ICs with harmonic monitoring property; Metering ICs with and without SPI compatibility and so on. The list continues as each IC suits a particular metering purpose. For our project, we studied four different ADE ICs. These are the ADE7751, the ADE7753, the ADE7754 and the ADE7758.

The ADE7751 is a high-accuracy, fault-tolerant electrical energy measurement IC that is intended for use with 2-wire distribution systems. The accuracy of this IC surpasses the IEC1036 standard. The reference circuit and the ADCs are only circuitry analog in this IC. All the other signal processing is done digitally. Thus superior stability and accuracy can be achieved even at extreme environmental conditions and over time. It includes a fault detector which warns of fault conditions and bills accurately during any fault occurrences. The IC provides average real power data and also has a power supply monitoring circuit. The ADE7751 is available in 24-lead DIP and SSOP packages [17]. The functional block diagram is given in Figure 3.1 [17].

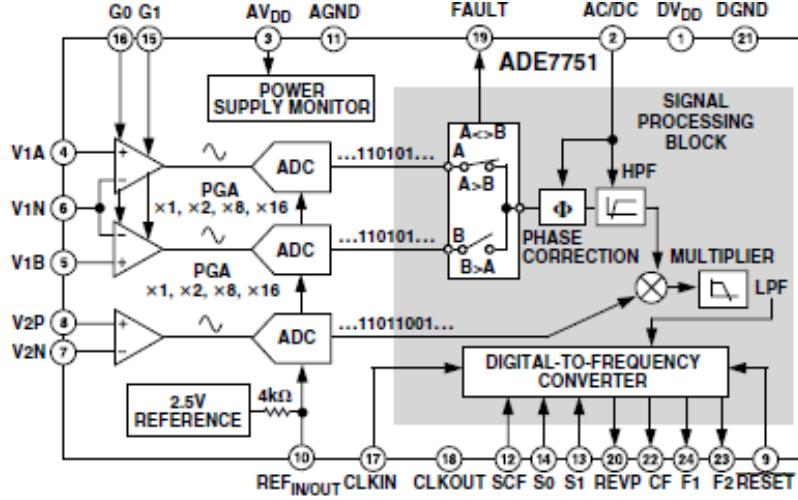


Figure 3. 1: Functional Block Diagram of ADE7751 [17]

The ADE7753 incorporates proprietary ADCs and digital signal processing for high accuracy in varying environmental states and time. It features two second-order 16-bit  $\Sigma\Delta$  ADCs, a selectable on-chip digital integrator, reference circuitry, temperature sensor and all the signal processing required to carry out active, reactive and apparent energy measurements, line-voltage period measurement and RMS calculation on the voltage and current. The chip includes a serial interface to read data and a pulse output frequency. It also has different system calibration features. The positive only accumulation mode provides the option of accumulating energy only when positive power has been detected. There is also an interrupt status register to point out the nature of the interrupt. The ADE7753 is available in a 20-lead SSOP package [18]. The functional block diagram is given in Figure 3.2 [18].

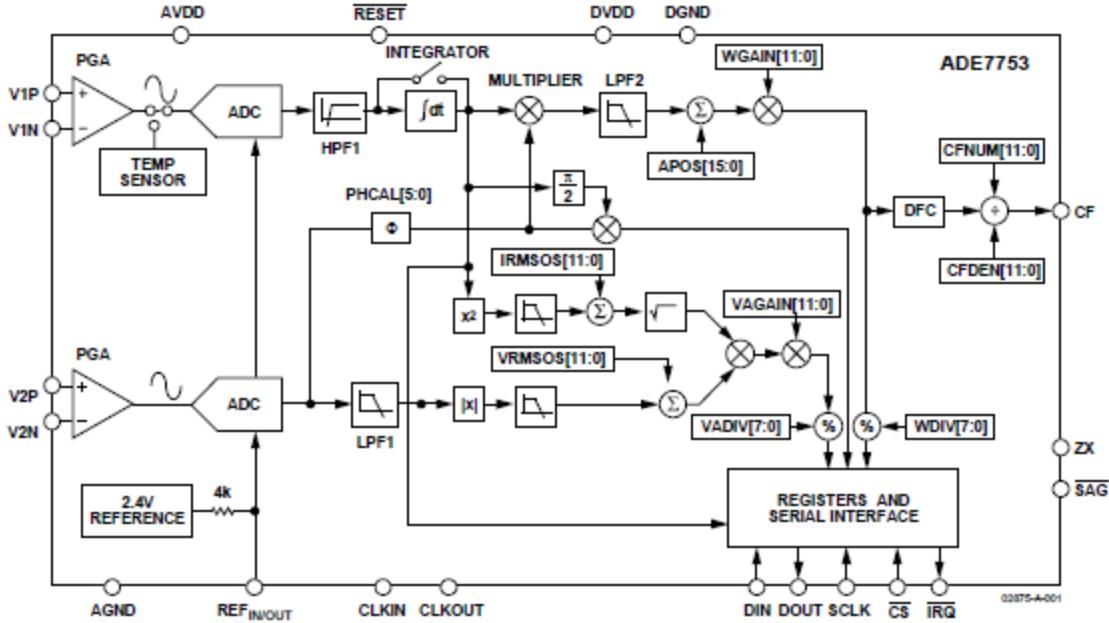


Figure 3. 2: Functional Block Diagram of ADE7753 [18]

The ADE7754 is a highly accurate polyphase electrical energy metering IC with serial interface and a pulse output. It features second-order  $\Sigma-\Delta$  ADCs, temperature sensor, reference circuitry, and all the signal processing required to carry out active, reactive and apparent energy measurements, and RMS calculations. The ADE7754 provides distinct solutions for measuring active and apparent energy from the six analog inputs, therefore enabling the use of the IC in various power meter services such as 3-phase/4-wire, 3-phase/3-wire, and 4-wire delta. It also provides system calibration features for each phase. The ADE7754 incorporates a waveform sample register that enables access to ADC outputs. It also contains a detection circuit for short duration low or high voltage variations. The SPI serial interface is used to read data from the ADE7754. The IC is available in a 24-lead SOIC package [19]. The functional block diagram is given in Figure 3.3 [19].

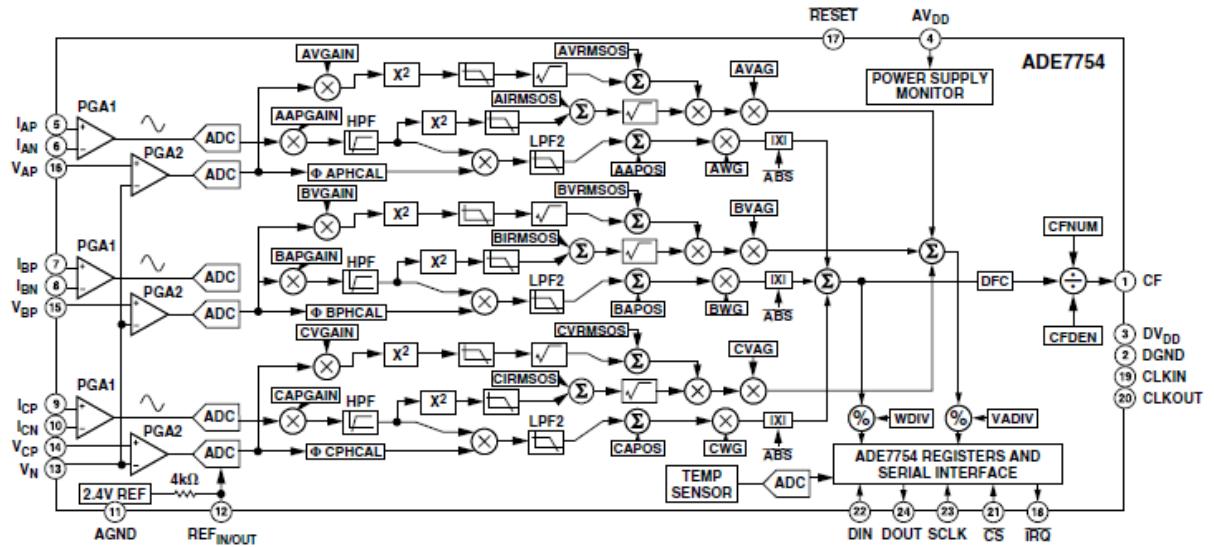


Figure 3. 3: Functional Block Diagram of ADE7754 [19]

The ADE7758 is highly accurate three phase energy measurement IC. It has a serial interface and two pulse outputs. It features second-order  $\Sigma\Delta$  ADCs, a digital integrator, a temperature sensor, reference circuitry, and all the signal processing required to carry out active, reactive and apparent energy measurements, and RMS calculations. The ADE7758 is appropriate for measuring active, reactive, and apparent energy in various 3-phase configurations, such as WYE or DELTA services, with both three and four wires. The ADE7758 incorporates system calibration features for each phase, that is, RMS offset correction, phase calibration, and power calibration. The APCF logic output provides active power data while the VARCF logic output gives instantaneous reactive or apparent power information. The ADE7758 comes in a 24-lead SOIC package [20]. The functional block diagram is given in Figure 3.4 [20].

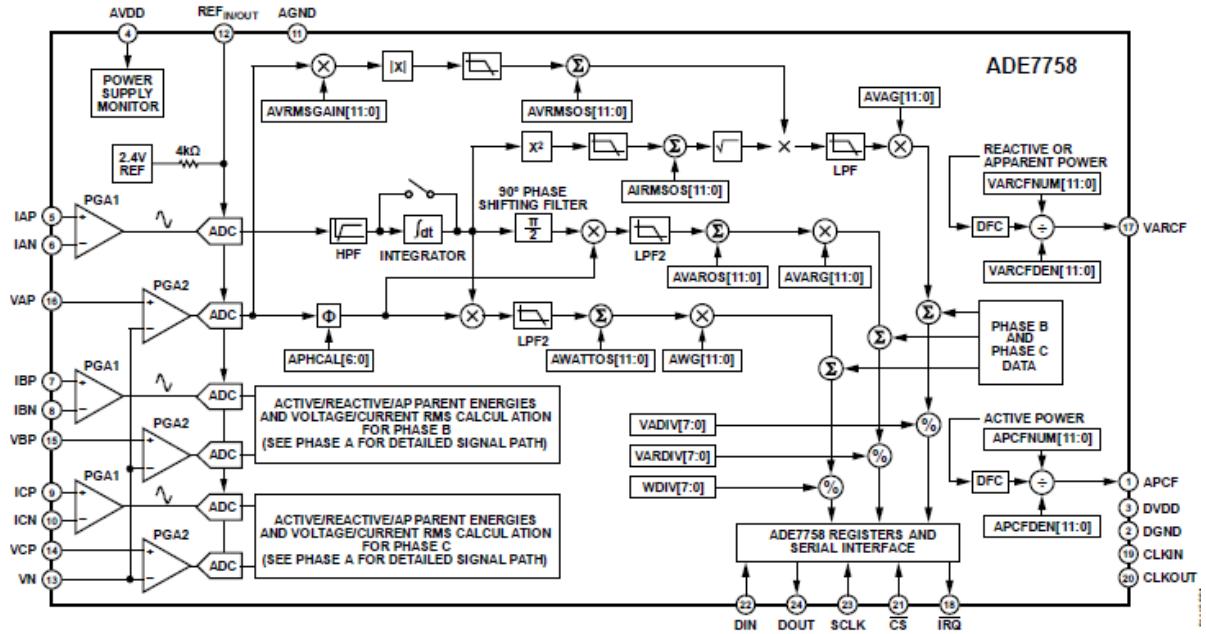


Figure 3. 4: Functional Block Diagram of ADE7758 [20]

### 3.3 Comparison of Meter ICs

ADE7754	ADE7753	ADE7751	ADE7758
Poly-phase Multi-Function Energy Metering IC with Serial Port	Single-Phase Multifunction Metering IC with di/dt Sensor Interface	Energy Metering IC with On-Chip Fault Detection	Poly Phase Multifunction Energy Metering IC with Per Phase Information
#High Accuracy, supports IEC 687/61036. Compatible with 3-phase/3-wire, 3-phase/4-wire and any type of 3-phase services.	#High accuracy; supports IEC 60687/61036/61268 and IEC 62053-21/62053-22/62053-23.	#High Accuracy, Surpasses 50 Hz/60 Hz IEC 687/1036	#Highly accurate; supports IEC 60687, IEC 61036, IEC 61268, IEC 62053-21, IEC 62053-

			22, and IEC 62053-23 #Compatible with 3-phase/3-wire, 3-phase/4-wire, and other 3-phase services
#Less than 0.1% error in Active Power Measurement over a dynamic range of 1000 to 1.	#Less than 0.1% error in active energy measurement over a dynamic range of 1000 to 1 at 25°C.	#Less than 0.1% Error over a Dynamic Range of 500 to 1	#Less than 0.1% active energy error over a dynamic range of 1000 to 1 at 25°C
#The ADE7754 supplies Active Energy, Apparent Energy, Voltage rms, Current rms and Sampled Waveform Data.	#Active, reactive, and apparent energy; sampled waveform; current and voltage rms.	#Supplies Average Real Power on the Frequency Outputs F1 and F2	#Supplies active /reactive /apparent energy, voltage RMS, current RMS, and sampled waveform data
#Digital Power, Phase & Input Offset Calibration.	#Digital calibration for power, phase, and input offset.		#Digital power, phase, and RMS offset calibration
#A SPI compatible Serial Interface with Interrupt Request line (IRQ).	#SPI compatible serial interface #Interrupt request pin (IRQ) and status register		#An SPI-compatible serial interface with IRQ
#A pulse output with programmable frequency	#Pulse output with programmable frequency	#Two Logic Outputs (FAULT and REVP) Can Be Used to	#Two pulse outputs, one for active power and the

		Indicate a Potential Mis-wiring or Fault Condition	other selectable between reactive and apparent power with programmable frequency
	#A PGA in the current channel allows direct interface to shunts and current transformers.	#A PGA in the Current Channel Allows the Use of Small Values of Shunt and Burden Resistance	#A PGA in the current channel allows direct interface to current transformers
#Proprietary ADCs and DSP provide high accuracy over large variations in environmental conditions and time.		#Proprietary ADCs and DSP Provide High Accuracy over Large Variations in Environmental Conditions and Time	#Proprietary ADCs and DSP provide high accuracy over large variations in environmental conditions and time
#An On-Chip temperature sensor ( $\pm 3^\circ\text{C}$ typ. After calibration), On-Chip user Programmable thresholds for line voltage, SAG and overdrive detections.	#On-chip digital integrator enables direct interface to current sensors with $\text{di}/\text{dt}$ output.  #On-chip user programmable threshold for line voltage surge and SAG and PSU supervisory.  #On-chip temperature sensor ( $\pm 3^\circ\text{C}$ typical)	#On-Chip Power Supply Monitoring  #On-Chip Creep Protection (No Load Threshold)	#On-chip, user-programmable thresholds for line voltage SAG and overvoltage detections  #An on-chip, digital integrator enables direct interface-to-current sensors with $\text{di}/\text{dt}$ output

	#Positive-only energy accumulation mode available		
		<p>#High-Frequency Output CF Is Intended for Calibration and Supplies Instantaneous Real Power</p> <p>#Continuous Monitoring of the Phase and Neutral Current Allows Fault Detection in 2-Wire Distribution Systems</p> <p>#ADE7751 Uses the Larger of the Two Currents (Phase or Neutral) to Bill—Even During a Fault Condition</p> <p>#Direct Drive for Electromechanical Counters and 2-Phase Stepper Motors (F1 and F2)</p>	
#Reference 2.4V±8% (Drift 30 ppm/°C typical) with external overdrive capability. #Single 5V Supply, Low power (80mW typical).	#Reference 2.4 V with external overdrive capability #Single 5 V supply, low power (25 mW typical)	#Reference 2.5 V +/- 8% (30 ppm/°C Typical) with External Overdrive Capability #Single 5 V Supply, Low Power (15 mW Typical)	#Reference 2.4 V (drift 30 ppm/°C typical) with external overdrive capability #Single 5 V supply, low power (70 mW typical)
		#Low-Cost CMOS Process	

Table 3. 1: Meter IC Comparison [17], [18], [19], [20]

### **3.4 Meter IC Selected**

All the four meter ICs studied are extremely efficient and accurate at measuring energy and additionally providing information specific to their characteristic properties. However, the ADE7753 and the ADE7754 were not available in Bangladesh to purchase. It meant that the options were now limited to the ADE7751 and the ADE7758. Both the meter ICs are highly accurate but the ADE7758's accuracy is over a longer range than that of the ADE7751. Moreover, the ADE7758 had an SPI compatible serial interface with IRQ [20], which was not available with the ADE7751 IC. The SPI compatible serial interface is vital for integrating the chip with a microcontroller unit (MCU) for operating the functions, executing digital calibration, and interpreting the measurement results. Therefore, the ADE7758 measurement IC was chosen for our project, IoT energy meter.

### **3.5 ADE7758: Poly Phase Multifunction Energy Metering IC with Per Phase**

#### **Information**

The ADE7758 consists of a waveform sample register which leads the accessing of the ADC outputs. A detection circuit for short duration low or high voltage variations is also available on the IC [20]. The voltage threshold levels and the duration (number of half-line cycles) of the variation are user programmable. A zero-crossing detection is synchronized with the zero-crossing point of the line voltage of any of the three phases. Measurement of the period of any one of the three voltage inputs is possible by using this data. The zero-crossing detection is in the ADE7758 for the line cycle energy accumulation mode, it allows faster and more accurate calibration by synchronizing the energy accumulation with an integer number of line cycles [20]. The SPI serial

interface is used to read data from the chip. The interrupt request output (IRQ) is an open-drain, active low logic output. The IRQ output goes active low when one or more interrupt events have occurred in the IC. A status register indicates the nature of the interrupt. The specifications of the IC is presented in detail in the ADE7758 datasheet (See Appendix A). The ADE7758 has 24 pins which are shown in Figure 3.5 [20] and the pin functions have been stated and briefly described in the appendix section (See Appendix B).

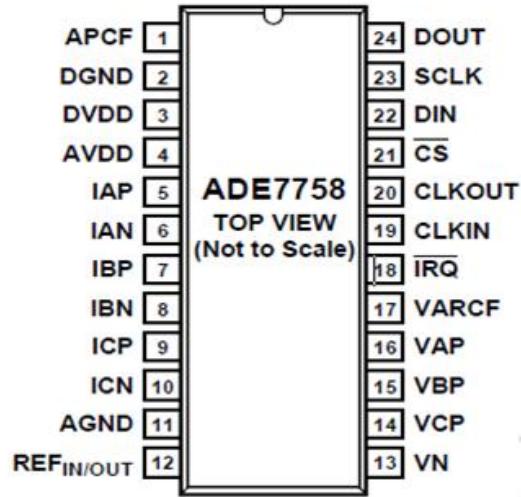


Figure 3.5: ADE 7758 Pin Configuration [20]

### 3.5.1 ADE7758 Test Circuit and Testing Procedure

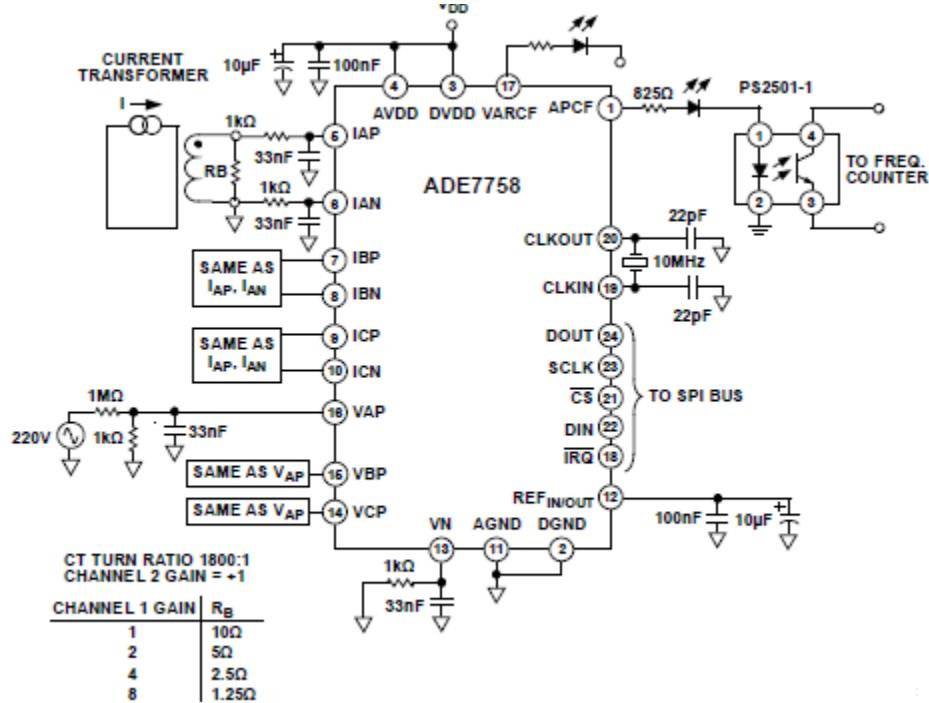


Figure 3.6: Test Circuit for ADE7758 IC [20]

The circuit shown in Figure 3.6 [20] has to be implemented to test the ADE7758 IC. The load is connected to the Meter IC via a Current Transformer (CT). The CT scales the large current values to small standardized values which are proportional to the primary values but easier to measure. The CT is connected to the Meter IC through a shunt, which is a small resistor ( $2.2\Omega$  used). The shunt is connected across IAP and IAN pin of the Meter IC through low-pass filters. The IC performs measurements on this current by using the reference voltage supplied in the VAP pin of the IC. The 220V is connected to the pin through a voltage divider. 5V is supplied to the AVDD and DVDD pins for IC operation. The AGND and DGND pins are grounded. Tests can be performed by measuring the frequency and the temperature provided by the on-chip temperature sensor. The measurement results can be read and administered via a MCU connected to the SPI serial interface of the IC. These data are received from the ADE7758 by reading and writing to

specific Registers of the IC, the Register chart is given in the appendix section (See Appendix C). The frequency data can be achieved in a relatively straightforward manner by reading the 0x10 Register of the IC. But the temperature reading needs some calibration. The temperature measurement is taken each 4/CLKIN seconds [20]. The output from the temperature sensing circuit is connected to an ADC for digitizing. The resultant code is processed and placed in the temperature register (TEMP[7:0]). This register can be read by the user and has an address of 0x11. The contents from the temperature register are signed and has a resolution of 3°C/LSB. The offset of the 0x11 register has a chance of varying significantly. Therefore, for calibration, the nominal value should be measured, and the equation should be adjusted accordingly [20].

$$\text{Temp } (\text{°C}) = [(\text{TEMP}[7:0] - \text{Offset}) \times 3\text{°C/LSB}] + \text{Ambient}(\text{°C}) \quad [20]$$

### **3.5.2 Registers Used for Measurement**

The IoT smart energy meter developed is primarily for residential sector application. And since majority of residential electricity connection is in single phase, the meter has been developed as a single phase IoT smart energy meter. Therefore during development and also implementation phase A samples of the IC is used only. This has been done by using the Measurement Mode Register (0x14) and the Waveform Mode Register (0x15) (See Appendix D and E). As mentioned earlier, temperature of the IC from the temperature sensor is read from the TEMP register (0x11) while, frequency is read from the FREQ register (0x10) (See Appendix C). Both of these are for testing purposes. To measure the energy information, we use the VRMS and IRMS data and calculate the energy data on the server end. The AVRMS register (0x0D) which is the phase A voltage channel register is read to obtain the VRMS data (See Appendix C). Similarly the AIRMS register (0x0A) which is the phase A current channel RMS register is read to obtain the IRMS data (See Appendix C).

### **3.5.3 Calibration**

The VRMS and IRMS data obtained from the ADE7758 talked about in 2.5.9 is not the actual RMS Voltage and RMS Current values respectively. To obtain the actual information these data must be calibrated. A load-board with known variable loads is connected to the IC via the Current Transformer (CT). By varying the known loads several VRMS and IRMS data are obtained from the ADE7758. Concurrently using a multimeter the AC voltage is recorded for the different known loads. The AC current can be figured out for these different loads as the loads are of known value and also the AC voltage is measured and now known. Therefore, by following the relation between these two sets of data a VRMS constant and an IRMS constant is developed. These constants multiplied with the respective VRMS and IRMS data from the ADE7758 IC gives the actual VRMS and IRMS values, which are in turn used to calculate the energy information. The whole calculation process is performed in the server end of the IoT smart energy meter instead of the MCU and thus achieves results rapidly. Figure 3.7 displays The ADE7758 Meter IC circuit implemented on a breadboard connected with the MCU and a load-board via the CT.



*Figure 3. 7: The ADE7758 Meter IC circuit implemented on a breadboard connected with the MCU and a load-board via the CT*

## Chapter 4

### Serial Peripheral Interface (SPI) with ADE7758

Serial Peripheral Interface (SPI) is a serial communication interface which is used to send data between multiple microcontrollers and small peripherals such as shift registers, sensors, and SD cards. This type of interface was developed by Motorola in the mid-1980s. Since then it has been very widely used as it is fast, easy to use and simple.

In SPI communication, devices have master-slave relationship between each-other. The controlling device is called master and slave is usually a sensor, display or memory chip which takes commands from the master device. SPI bus can also operate with a single master device and multiple slave devices.

#### 4.1 Data Transmission

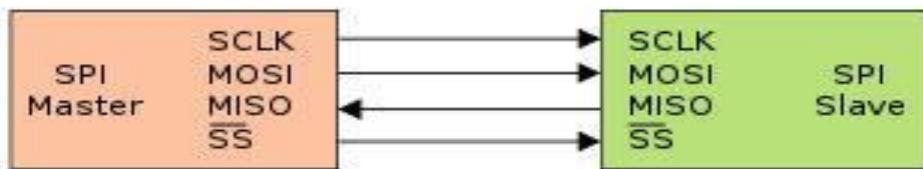


Figure 4. 1: SPI Communication Overview [3]

The Figure 4.1 above describes the connections for a basic SPI communication between a master and a slave device,

Where,

SCLK – Clock signal line

MOSI – Master output Slave input

MISO – Slave output Master input

SS – Slave Select

#### **4.1.1 Clock Signal**

To begin communication, master configures the clock, by SCLK signal using a frequency supported by the slave device, typically up to a few MHz. Each Clock cycle takes 1 bit data. [21]

Typically master configures and generates clock signal to start SPI communication. SPI is a synchronous communication protocol. That means here data will go bit by bit one after another.

Asynchronous methods don't use clock signal like UART.

Also, in SPI clock polarity and clock phase must be set. It sets the bit transmission timing and sampling whether it's rising or falling edge.

#### **4.1.2 Slave Select**

To select the exact slave device which we will transmit data from the master device we have to set that slave's SS line to 0. In the idle, non-transmitting state, the slave select line is kept at a high voltage level [21]. Also, Multiple CS/SS pins may be available on the master, which allows for multiple slaves to be wired in parallel.

#### 4.1.3 Multiple Slaves

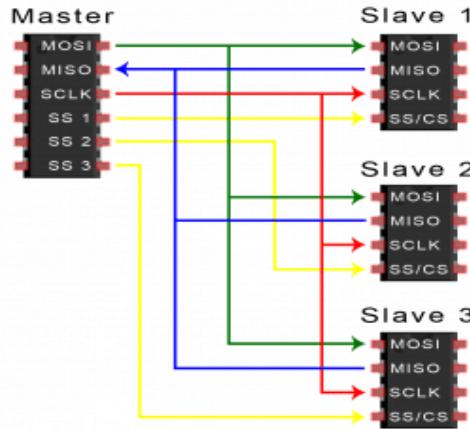


Figure 4. 2: Single Master Multiplane Slave SPI [1]

The figure 4.2 above shows us the connection diagram if a single master device is connected to multiple slave devices.

SPI can be set up to two modes of operations, one is a single master and a single slave and other is single master with multiple slaves controlled by the master device. Like the figure, if the master has multiple slave select pins, the slaves can be wired in parallel.

#### 4.1.4 Steps of Data Transferring

1. The master sends clock signal from its SCLK pin to Slave's SCLK pin as shown in the figure below.

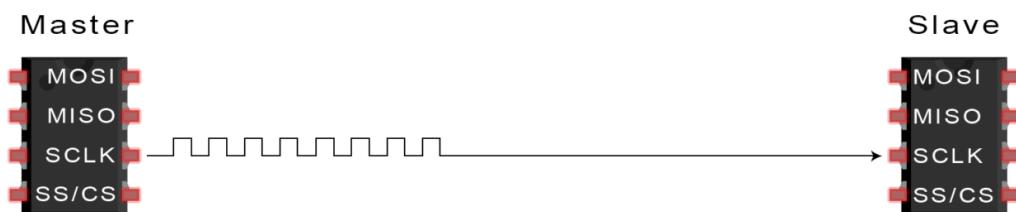
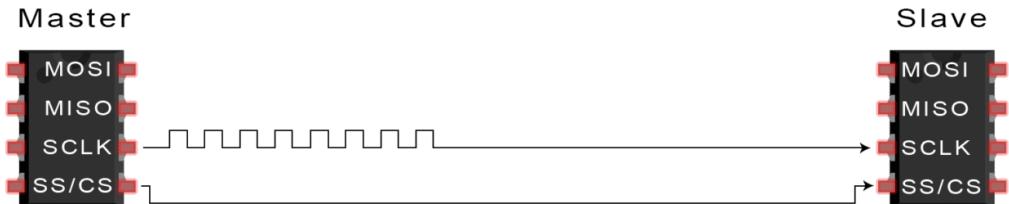


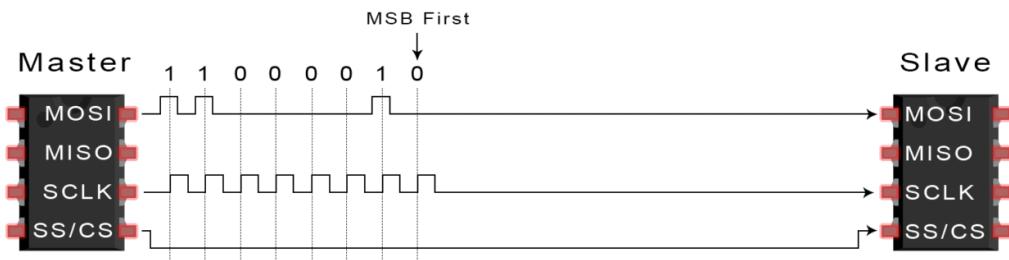
Figure 4. 3: Single Master Multiple Slave SPI [1]

2. Then master activates the slave by switching SS/CS pin to a low voltage state. This step is shown in the figure below.



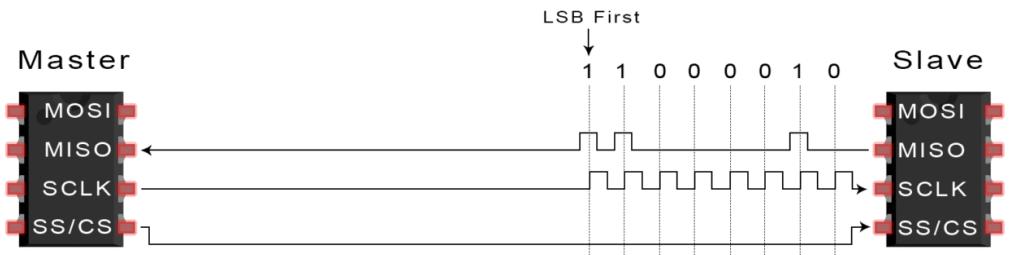
*Figure 4. 4: Master Sets SS low [1]*

3. Master then sends data one bit at a time to the slave by using the MOSI line. Slave reads data bits in the order received. The step is shown in the figure below.



*Figure 4. 5: MOSI sends Data Bits [1]*

4. For giving response slave returns data one bit at a time to the master using the MISO line. Master reads data bits in the order receive. The step is shown in the following figure.



*Figure 4. 6: MISO receives Data Bits from Slave [1]*

#### **4.1.5 MOSI and MISO**

MOSI line passes data bits from master to slave devices in synchronous mode. The slave device receives by MOSI pin. Slave can also send back the data to master by MISO line. The order of the data bits is determined by the DORD bit.

#### **4.1.6 Advantages**

- No start and stop bits, so the data can be streamed continuously without interruption
- No complicated slave addressing system like I2C
- Higher data transfer rate than I2C (almost twice as fast)
- Separate MISO and MOSI lines, so data can be sent and received at the same time.

#### **4.1.7 Disadvantages**

- Uses four wires (I2C and UARTs use two).
- No acknowledgement that the data has been successfully received (I2C has this)
- No form of error checking like the parity bit in UART
- Only allows for a single master.

### **4.2 SPI Registers**

Most of the AVR<sub>s</sub>, including the Atmel microcontrollers, use SPI protocol for communicating with an IC. In our case, ADE7758 supports SPI protocol for communication. So we connect it with an Atmega32. For SPI communication, there are three registers associated. They are SPSR (Serial Peripheral Status Register), SPCR (Serial Peripheral Control Register) and SPDR (Serial Peripheral Data Register) [22].

For our initial attempt to establish an SPI Communication between the meter IC and the microcontroller, the necessary SPI register values were set in the following order.

In the SPSR Register (See Appendix EE), the SPIF register was monitored after each data transmission and reception.

In the SPCR Register (See Appendix FF), the SPE register was set to 1, to enable the SPI Communication. The MSTR bit was also set to 1, indicating the ATmega32 functioned as the master device. The DORD remained 0, so that the MSB of any data was transmitted first during any data transmissions. The CPOL and CPHA was set to 0 and 1 respectively, to enable SPI Mode 1 for communication. The SPR0 was set to 1, for 1MHz of clock frequency.

Necessary data bits were loaded on SPDR Register (See Appendix GG), and the received data was also extracted from this register.

## 4.3 SPI Connection between ADE7758 and ATmega32

### 4.3.1 Circuit Diagram

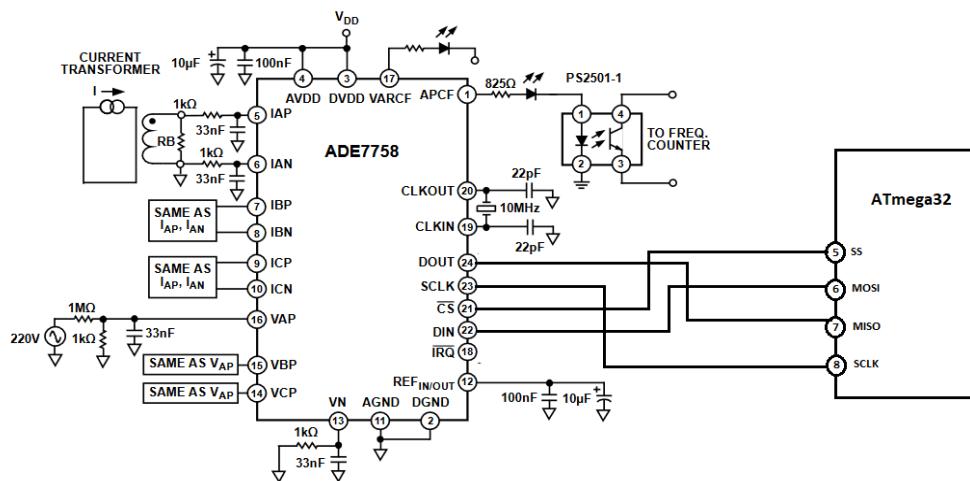


Figure 4. 7: SPI Pin Connection

To establish an SPI Connection, these pins should be connected properly like the figure 4.7 above:

- PIN 21 of ADE7758 to PIN 5 of ATmega32 (Slave Select Connection)
- PIN 22 of ADE7758 to PIN 6 of ATmega32 (D<sub>in</sub> to MOSI)
- PIN 24 of ADE7758 to PIN 7 of ATmega32 (D<sub>out</sub> to MISO)
- PIN 23 of ADE7758 to PIN 8 of ATmega32 (Clock)

#### 4.3.2 ADE7758 Timing Diagram

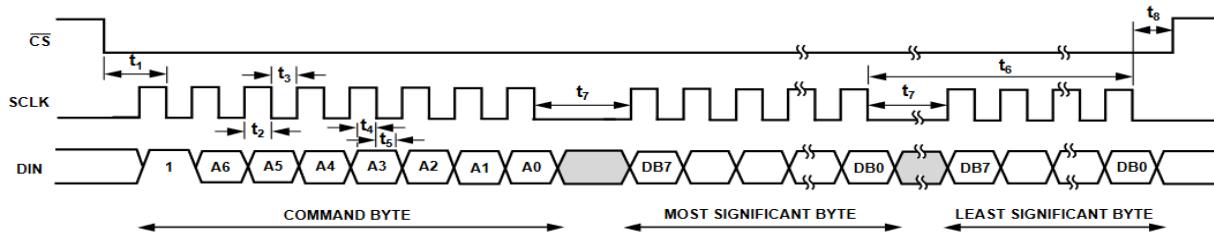


Figure 4. 8: Serial Write Diagram for ADE7758 [25]

From the Serial Write Program [25], we can see that the ADE7758 operates at SPI Mode 1, meaning it reads data from falling edges and the data changes during the rising edges of the clock. We can also see that to write data to a particular register of ADE7758, we need to write the address of that particular register, before loading the data on the SPDR. And when writing the register address in the SPDR, the MSB of that address should be set to 1 to indicate write operation.

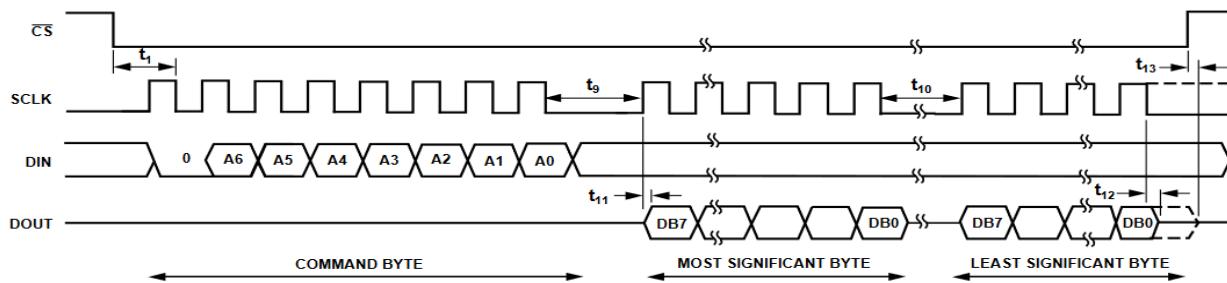


Figure 4. 9: Serial Read Diagram for ADE7758 [25]

Similarly, to read data from a particular register, we have to write the address of that particular register. This time, the MSB of the address bit should be set to 0, to indicate read operation. The data then should be read by the MISO of the ATmega32. The data is read after the byte of the address is transmitted, so during this time we write 0x00, so that we can read the data from the addressed register in exchange. Figures 4.8 and Figures 4.9 shows us the timing diagram for Data Writing and Data Reading respectively.

#### 4.4 SPI Initial Code Analysis

PB 4, 5 and 7 are the Slave Select, MOSI and SCLK bits respectively. We first initialize them as outputs. Then we Enable the SPI, configure the device as the Master Device and set the clock frequency to  $f_{osc}/16$ . And make the Slave Select high as default.

This method loads the data to be transmitted in the SPDR. Then waits for the Interrupt Flag to be set as 1, indicating the end of data transmission. By then the data coming from the Slave device will be loaded in the SPDR. Extracting the data from the SPDR will give the response from the Slave (See Appendix F).

In the main function, PB 4 or the slave select pin was laid low during the transmission. During this time, the main function called spi\_tranceiver function with the parameter 0x11, which is the address for the temperature address and then 0x00. However the results were unexpected. When the clock was low, MISO still seemed to be transmitting the data, as can be seen in the Oscilloscope in the figure 4.10. The output on the other hand was too low to take a reading of and can be seen in figure 4.11.

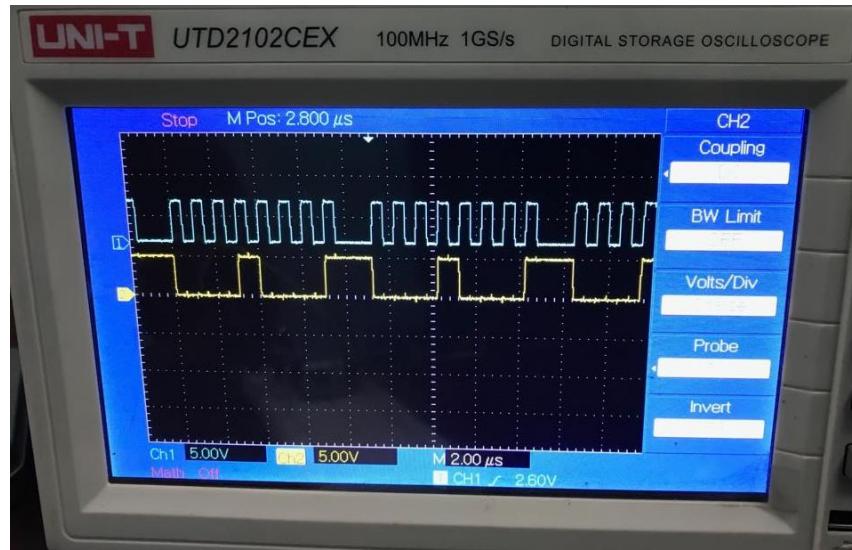


Figure 4. 10: MOSI and SCLK in the Oscilloscope

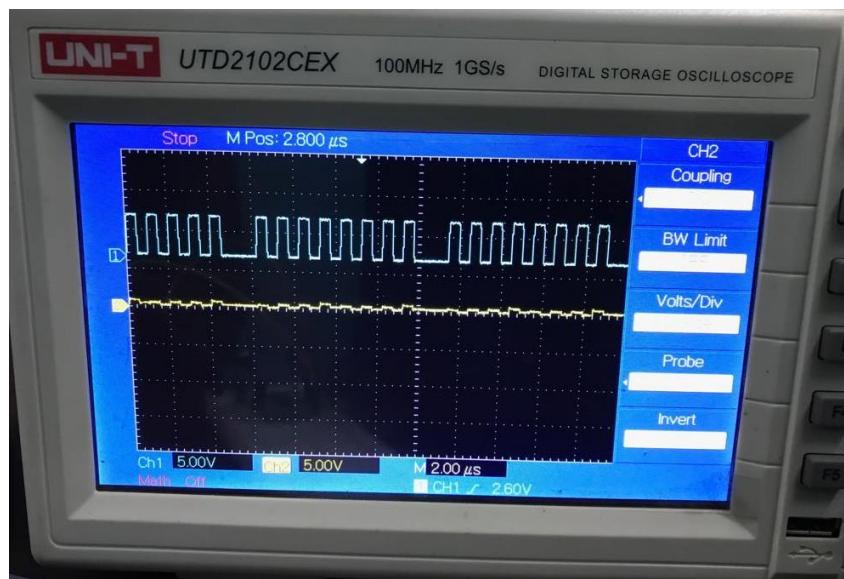


Figure 4. 11: MOSI and SCLK in the Oscilloscope

As the ADE7758 was not responding, the SPI Communication was made in a different approach.

Instead of setting high or low all the bits of SPI registers, the SCLK, SS, and MOSI pins were all configured as GPIO pins.

#### 4.4.1 Writing Data

As the previous SPI Communication code did not give the expected result, the write and read functions were changed with a new approach. This method takes the address hex value and the data hex value as parameters and performs the write function. Figure 4.12 is a flowchart of the write function.

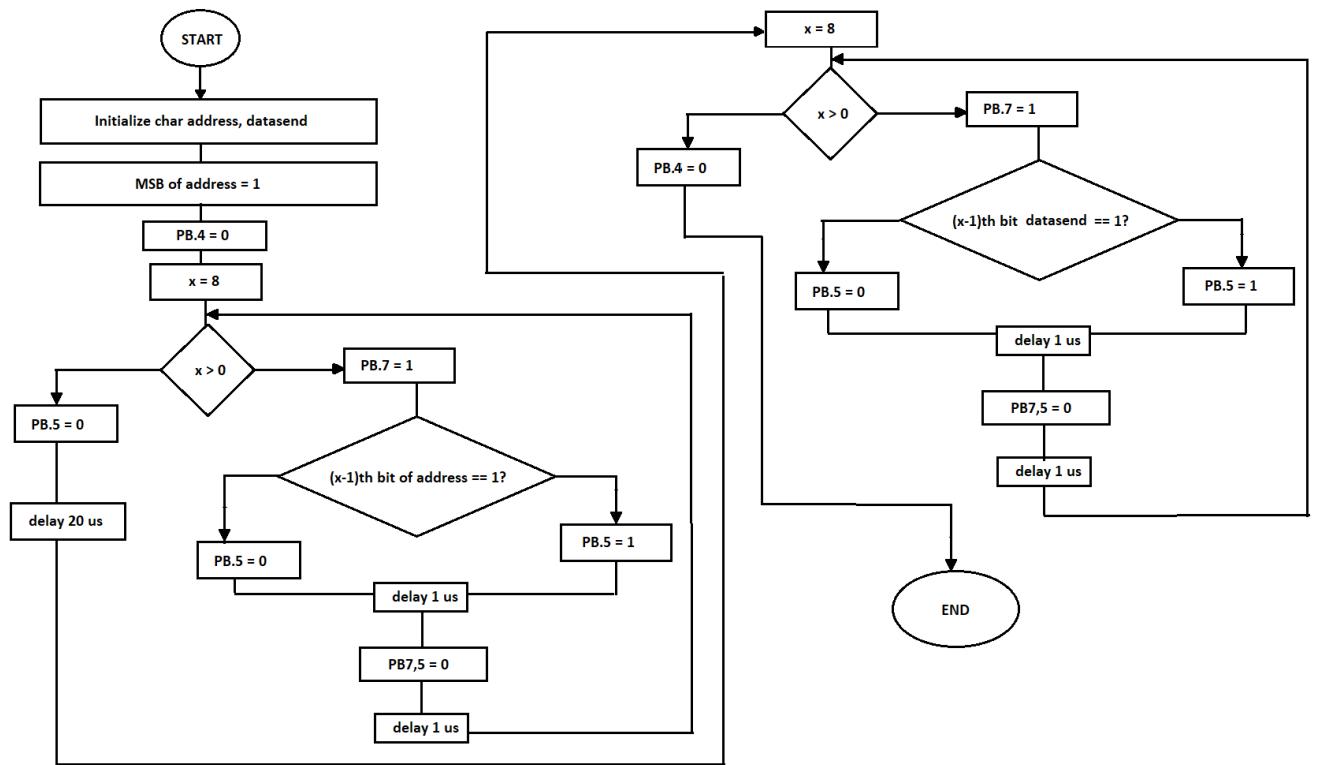


Figure 4. 12: ADE7758 Write data Code Flowchart

The 7<sup>th</sup> bit of the address is set to 1 to indicate data write. The slave select pin then gives 0 as output as it was done previously before data transmission. For the 8 bits, the SCLK pin was set to 1 for one microsecond. Then all of the bits of the address were checked whether or not they were 0 or 1. If they were 0, the MOSI pin remained low. If they were 1, the MOSI pins were high. The same way all of the bits were checked and MOSI pins transferred data to D<sub>in</sub> pin of ADE7758 accordingly. After each bit is transmitted, the SCLK pin output became low for one microsecond.

This entire process continues for 8 bits and after that there is a delay for 20 microseconds before the next byte can be sent. In this process, the address and the data to be written both are transmitted (See Appendix G).

#### **4.4.2 Reading Data**

As it was mentioned earlier, the reading of data from ADE7758 is done in two steps. First the address bits are written and then the data is read from MISO pin of ATmega32 which is connected with  $D_{out}$  (Data out) of ADE7758.

The writing of the address is same as the way it was for the writing code block. Only the MSB of the address is set to 0 to indicate the reading mode.

After the 8 bit address is written there is a 20 microsecond delay. The clock is toggled again for 8 cycles every 1 microsecond. The MISO pin in ATmega32 was configured as Input pin initially. The status of the input pin is checked during the falling edges. If the received bit is 0, the variable named “dataReceive”’s corresponding bit is stored as 0. If PB6 pin receives high signal then the corresponding bit of “dataReceive” is stored as 1. After 8 clock cycles, we get the full byte of data (See Appendix H). There were separate methods for receiving 16 bits of data, and 24 bits of data.

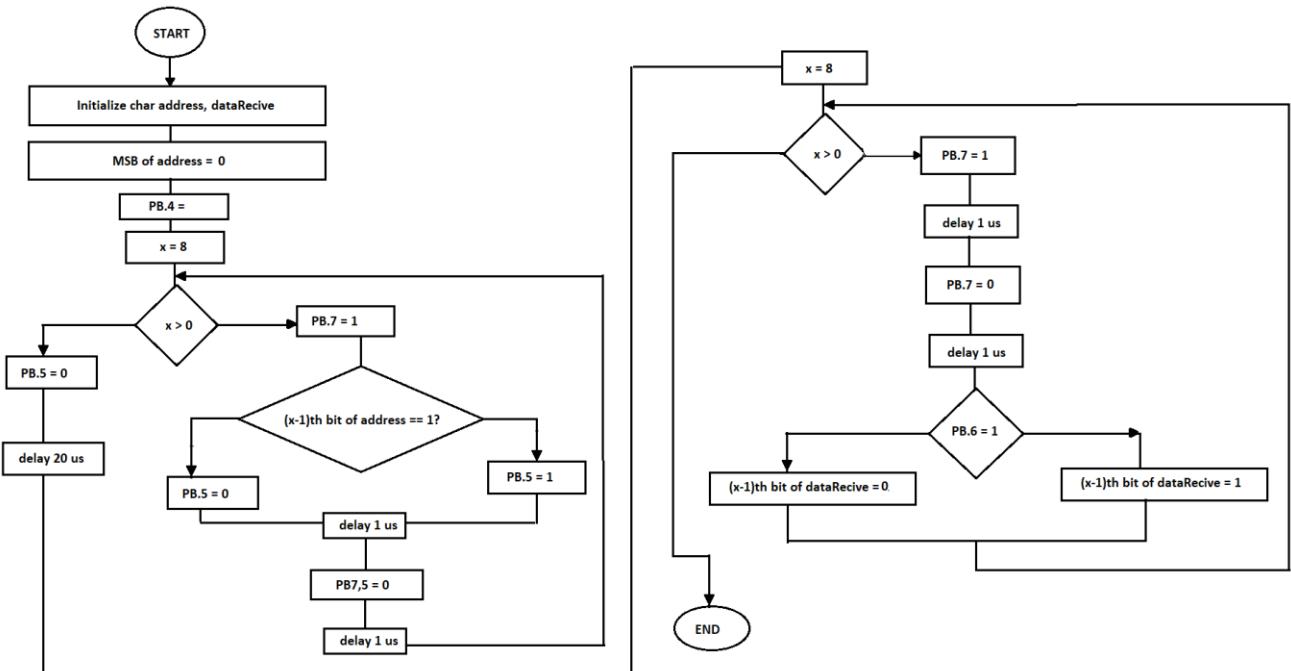


Figure 4. 13: ADE7758 Read Data Code Flowchart

Similarly we can receive data of 16 bits and 24 bits. But after every 8 clock cycles, there should be a delay of 20 microseconds. Figure 4.13 represents the flow chart for read function of 8 bits.

## Chapter 5

### Communication with the Server using ESP-12E

#### 5.1 Introduction to ESP8266

ESP8266 is a Wi-Fi enabled system on chip (SoC) module developed by Espressif system. It is generally used for IoT (Internet of Things) embedded applications. Its purpose is to connect with the internet via Wi-Fi. Microcontrollers are used to control this chip. Figure 5.1 is the first of its series of ESP8266.

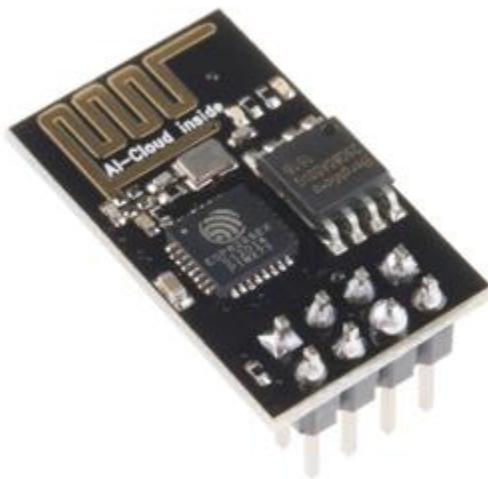


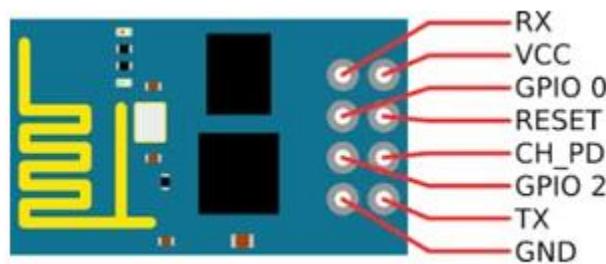
Figure 5. 1: Esp-01 Module [5]

ESP8266 specifications are given below:

- 2.4 GHz Wi-Fi (802.11 b/g/n, supporting WPA/WPA2),
- general-purpose input/output (16 GPIO),
- Inter-Integrated Circuit (I<sup>2</sup>C) serial communication protocol,
- analog-to-digital conversion (10-bit ADC)

- Serial Peripheral Interface (SPI) serial communication protocol,
- I<sup>2</sup>S (Inter-IC Sound) interfaces with DMA(Direct Memory Access) (sharing pins with GPIO),
- UART (on dedicated pins, plus a transmit-only UART can be enabled on GPIO2), and
- Pulse-width modulation (PWM).

### 5.1.1 Pin Description



*Figure 5. 2: Generic ESP Module [6]*

ESP8266 Wi-Fi module has 8 types of pins as shown in Figure 5.2. They are given below:

**VCC:** - 3.3 V Power; can handle up to 3.6 V.

**GND:** - Ground Pin (0V).

**RST:** - Reset Pin.

**CH\_PD:** - Enable Pin; must be 1 to enable.

**TX:** - Serial Transmit Pin of UART.

**RX:** - Serial Receive Pin of UART.

**GPIO-0:** - General-purpose input/output No. 0.

**GPIO-2:** - General-purpose input/output No. 2.

### 5.1.2 ESP-12E Description

ESP-12E is another version of ESP modules that comes with more pins and features. This specific version of ESP is going to be used for our project. It is a miniature Wi-Fi module found in the market and is used for establishing a wireless network connection for microcontroller or processor. The core of ESP-12E which is shown in Figure 5.3, is ESP8266, which is a high integration wireless SoC (System on Chip). It can do all the things ESP8266 can do and also it has unique feature to embed Wi-Fi capabilities to systems or to function as a standalone application. It costs cheaper than most other Wi-Fi modules.



Figure 5. 3: ESP-12E [7]

### 5.1.3 Pin Description for ESP-12E

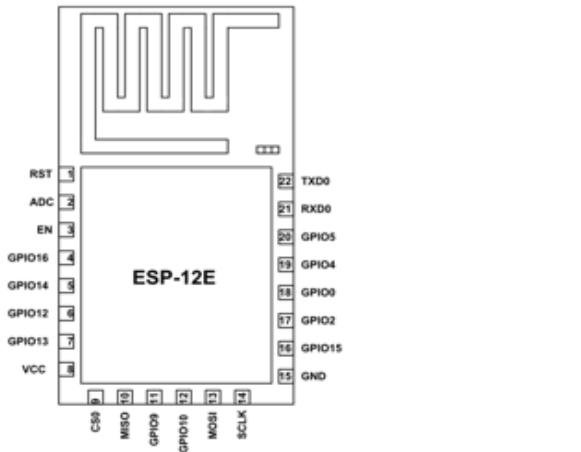


Figure 5. 4: ESP-12E Pin Description [7]

Figure 5.4 is the pin description for ESP-12E. Table 5.1 contains descriptions of the pins.

Pin	Name	Description
1	RST	Reset Pin of the module
2	ADC	Analog Input Pin for 10-bit ADC (0V to1V)
3	EN	Module Enable Pin (Active HIGH)
4	GPIO16	General Purpose Input Output Pin 16
5	GPIO14	General Purpose Input Output Pin 14
6	GPIO12	General Purpose Input Output Pin 12
7	GPIO13	General Purpose Input Output Pin 13
8	VDD	+3.3V Power Input
9	CS0	Chip selection Pin of SPI interface
10	MISO	MISO Pin of SPI interface
11	GPIO9	General Purpose Input Output Pin 9

12	GPIO10	General Purpose Input Output Pin 10
13	MOSI	MOSI Pin of SPI interface
14	SCLK	Clock Pin of SPI interface
15	GND	Ground Pin
16	GPIO15	General Purpose Input Output Pin 15
17	GPIO2	General Purpose Input Output Pin 2
18	GPIO0	General Purpose Input Output Pin 0
19	GPIO4	General Purpose Input Output Pin 4
20	GPIO5	General Purpose Input Output Pin 5
21	RXD0	UART0 RXD Pin
22	TXD0	UART0 TXD Pin

Table 5. 1: ESP-12E Pin Description

## 5.2 USART Communication between ESP-12E and ATmega32

To connect ESP-12E with the ATmega32, we would need a voltage converter. Because the ESP takes 3.3V input voltage, while the operating voltage for ATmega32 is 5V. For this purpose we have used AMS1117 5V Module, which converts 5 volt to 3.3 volt. The device is shown in Figure 5.5.

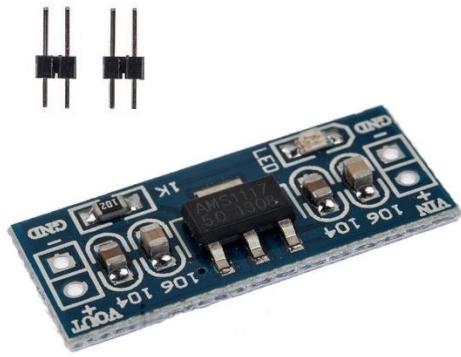
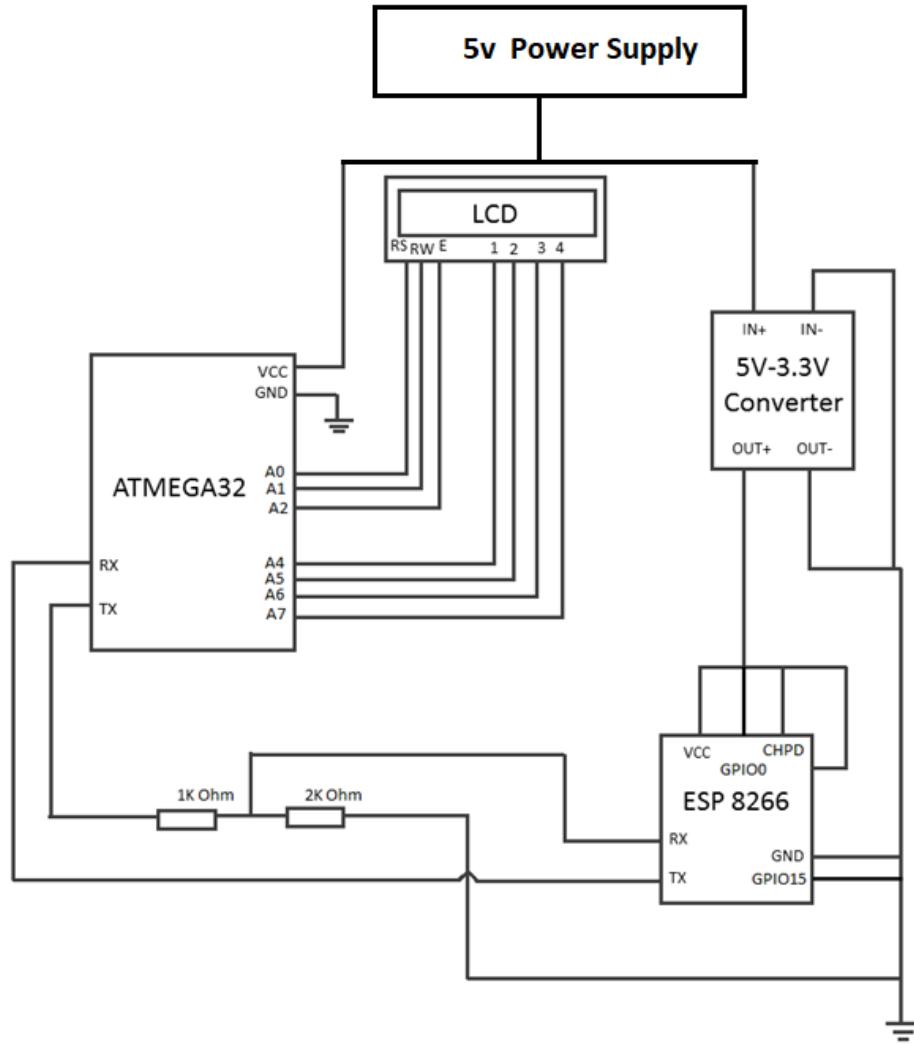


Figure 5. 5: AMS1117 5V Module [35]

The  $V_{cc}$  pin of ATmega32, which is Pin 10 is connected to  $V_{in}$  of AMS1117 and  $V_{out}$  is connected with 3.3V pin of ESP-12E. The Rx pin of ATmega32 is directly connected with th Tx pin of ESP, while the Tx pin of ATmega32 goes through a voltage divider comprised of a  $1K\Omega$  and  $2K\Omega$ , so that the ESP-12E doesn't get more than 3.3 Volts. The LCD is connected in Port A, as Port B is used for SPI Communication, Port C for connecting RTC and Port D for the USART communication with the ESP.

### 5.2.1 Circuit Diagram for ESP-12E and ATmega32

As the connections with the AMS1117 and ESP-12E are formed, we set the ESP-12E to operate in Working Mode. And to do that, the CHPD and RST are connected with the  $V_{cc}$  and GPIO0 is pulled high. In the Programming Mode it was made low previously. The GPIO15 is also made low. The ESP-12E is set to commence its normal operations using AT commands delivered by the ATmega32 using USART Communication Protocol. The Circuit diagram for connection between ESP-12E and ATmega32 is shown in Figure 5.6.



*Figure 5. 6: Connection between ESP-12E and ATmega32*

### 5.3 USART Communication Overview

Universal Asynchronous Receiver Transmitter (UART) is a type of communication protocol which is used to transfer data between multiple devices. It's unlike communication protocols like SPI and I2C; it is a physical circuit in a microcontroller, or a stand-alone IC. The main function of UART is serial data communication. By UART, the communication between two devices can be done in two ways as serial data communication and parallel data communication. UART can only do asynchronous type transmission. There is another device named Universal Asynchronous Receiver

and Transmitter (USART) which can do synchronous transmission also. But as we have other synchronous devices like SPI and I2C we don't use USART's synchronous properties much.

The Registers used for Serial Communication [24]:

The USART Receive Data Buffer Register and USART Transmit Data Buffer Register both share the same address, and is called UDR (See Appendix HH). The data written on UDR Register makes its way to Transmit Data Buffer Register while the data received through UDR Register comes via Receive Data Buffer Register. If the data is less than 8 bits, then the upper bits are made 0 by default.

The UBRRH (See Appendix II) share the same I/O location as of the UCSRC Register, and to shift between the two Registers, URSEL is toggled. When URSEL is 0, UBRRH is selected, and when it is 1, UCSRC is selected for operation. URSEL was set to 1.

Baud Rate (bps)	$f_{osc} = 16.0000MHz$			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%

Table 5. 2: UBRR values for 16 MHz Oscillator Frequency [24]

The UBRR Register values indicate baud rates depending on Oscillator Frequencies. As  $f_{osc}$  for the crystal used with ATmega32 is 16 MHz, we notice from the table 5.2 above, that to get 9600 baud rate, the UBRR value is 103 in decimal, which is 0x67 in hexadecimal.

In the UCSRA Register (See Appendix JJ), only the RXC and TXC registers were monitored, to verify whether the Receive and Transmission of data are completed or not.

In the UCSRB Register (See Appendix KK), the RXEN and TXEN to enable the overall receive and transfer function.

As for the UCSRC Register (See Appendix LL), the UMSEL was 0, for asynchronous communication, the UCSZ1 and UCSZ0 were both set to 1, for 8 bit data transmission, and UPM bits were set to 0, for the parity checker to be disabled.

## 5.4 USART Settings Code Breakdown

The uart library of the code is used only for the communication between the ATmega32 and ESP8266 [26]. The ESP8266 needs AT Commands to function and this commands are passed through with the help of uart library [6].

From the code fragment of `uart_init()`, we can say that the desired baud rate and the oscillator frequency was given as parameters to determine the appropriate UBRR value (See Appendix I). And the necessary baud rate value is then stored in UBRRH and UBRRL register. We can also see that RXCIE, RXEN and TXEN bits are set as 1 for Receive and Transfer functions. The UCSZ0 is set as 3, meaning that UCSZ2, UCSZ1, UCSZ0 are all set as 1 respectively.

## 5.5 AT Commands Used

As mentioned earlier, the ESP8266 works mainly on the Basis of their AT Commands. Now that it has been discussed how the AT Commands will be transferred using USART communication protocol, estimations should be made which AT Commands need to be sent for this project. There are a lot of AT Commands, each with different functions and different parameters [27]. But the ones that we would be using are:

- ATE (See Appendix J)
- AT+RST (See Appendix K)
- AT+CWMODE\_DEF (See Appendix L)
- AT+CIPMUX (See Appendix M)
- AT+CWJAP\_DEF (See Appendix N)
- AT+CIPSTART (See Appendix O)
- AT+CIPCLOSE (See Appendix P)
- AT+CIPSEND (See Appendix Q)

### 5.5.1 AT Commands used in Wifi Methods

The avr-wifi library in the code has a number of methods. But the ones used in the main function, use these AT Commands the most in different cases. The AT commands used in the most frequent wifi methods and their role are listed below in table 5.3.

Function	AT Commands	General Use	Use in Code
static inline void wifi_connect()	AT+CWMODE_DEF	Sets the Default Wi-Fi mode;	Here, CWMODE is set to “1” or station

		Configuration saved in the flash	mode. Station (STA) mode is used to get ESP module connected to a Wi-Fi network established by an access point.
	AT+CIPMUX	Enable or Disable Multiple Connections	Here, CIPMUX is set to “0” or Single Connections.
	AT+CWJAP	Connects to an AP; Configuration saved in the flash	Here, CWJAP takes user ID and user Password as input and then connects to the target AP if ID and Password is correct. This command requires Station mode to be enable.
static inline void wifi_link_open()	AT+CIPSTART	Establishes TCP Connections, UDP Transmission or SSL Connection	Here, CIPSTART takes protocol type, url address and port number. Then it establishes the connection to that server.
static inline void wifi_send_no_block()	AT+CIPSEND	Sends Data	Here, as this is a single connection (CIPMUX=0) only data length is inputted. When data

			length is met, transmission of data starts.
static inline void wifi_link_close()	AT+CIPCLOSE	Closes the TCP/UDP/SSL Connection	Here, this command closes the connection made by CIPSTART and stops data transmission.

Table 5. 3: AT Commands in Wifi Methods

## 5.6 AVR-WIFI Library Code Breakdown

### 5.6.1 Wifi Send Command Functions

These functions are the most essential in terms of the ESP code. Although these functions are not called directly from the main function, these functions are called inside every wifi method in the code. As stated earlier, the ESP initiates its regular operations through AT Commands, and without AT Commands the ESP cannot transfer or receive data between the server and the microcontroller, which in this case is ATmega32. The *wifi\_send\_command\_no\_block()* takes AT Command fragments as parameters, where the “AT” in every command is absent. Such as “+RST” or “E0” or “+GMR”. Upon calling the function, the function immediately transmits “AT” through its transfer buffer register, and afterwards, the remainder portion of the full AT Commands, received as parameters. Upon receiving the commands from the ATmega32, the ESP might or might not send back a response. By calling the function *wifi\_wait\_for\_status()*, the ATmega32 receives back responses from ESP using *uart\_getc()* functions. Here *wifi\_send\_command()* simply waits for responses after every commands transmitted, while *wifi\_send\_command\_no\_block()* does not (See Appendix R).

## **5.6.2 Wifi Initialization**

Wifi is initialized after disabling echo and the reset function. Both these functions are called inside *wifi\_init()*.

Before every ESP Communication with the server, the wifi is initialized by disabling echo throughout the channel and a fresh reset (See Appendix S).

## **5.6.3 Wifi Connect**

The most crucial wifi function (See Appendix T). Without connecting to an AP, the microcontroller will not be able to transfer the data to the server, and there will be data loss.

“AT+CWMODE = 1” sets the ESP to function as Station Mode, ‘0’ sets it as AP Mode, which will enable the ESP itself to operate as a Wifi Access Point.

“AT+CIPMUX = 0” sets the Single Connection Mode. The other mode is Multi Connection.

“AT+CWJAP\_DEF=“%s”,“%s””. This AT Command makes the ESP jump / connect to a specific Access Point, where the SSID and Password of the Access Point are given as Parameters.

ATmega32 then waits for the response from the ESP.

## CONNECTING TO AN ACCESS POINT

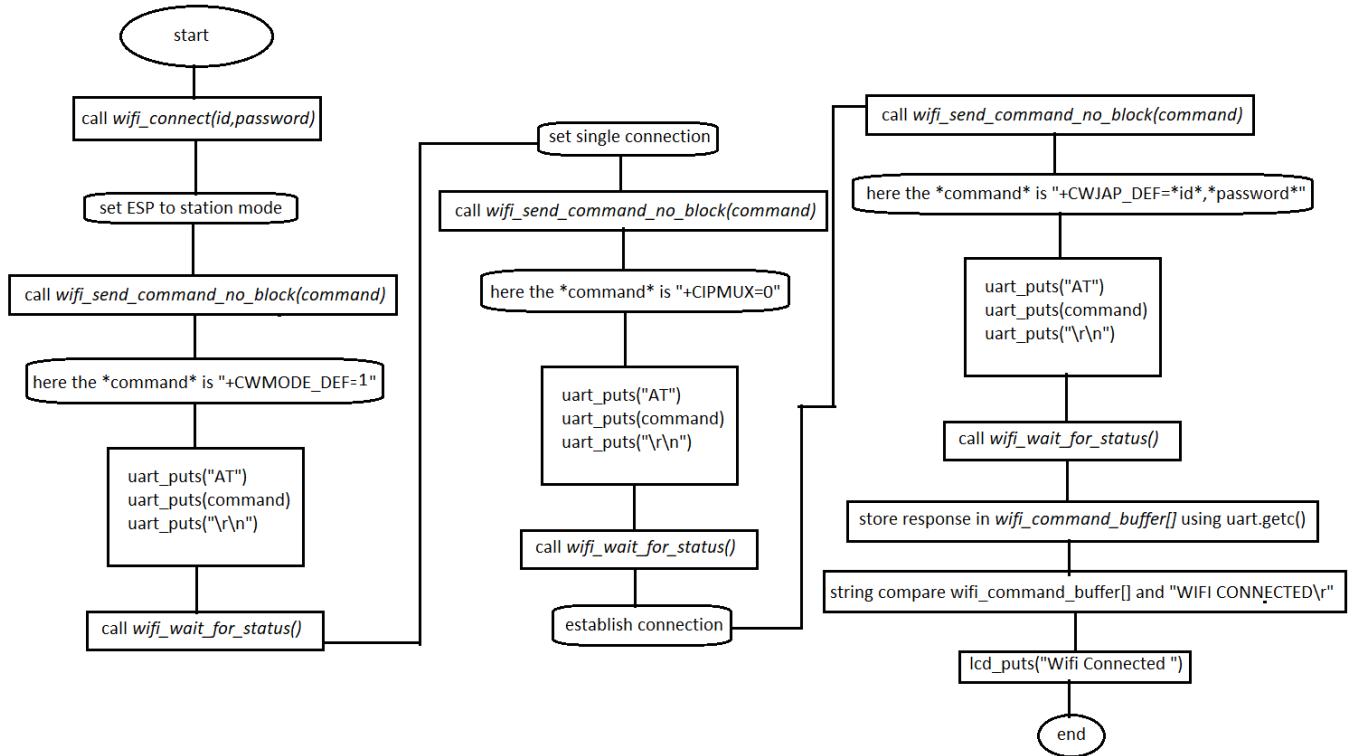


Figure 5. 7: Wifi Connect Flowchart

### 5.6.4 Wifi Link Open

After establishing a stable connection with an Access Point, the ESP then goes on to open a link for TCP Communication with the Server (See Appendix U). The “AT+CIPSTART” command takes the name of protocol, the url and port number as parameters to open a link between the server and the ESP.

### 5.6.5 Wifi Send Function

The above two methods are used for sending data to the server. For this project,  $V_{rms}$ ,  $I_{rms}$ , RTC Time and Device ID are all sent to the server. The function takes the strings and finds out the length of the strings. The contents are pushed by the `uart_puts()` function. This is also one of the

most crucial function without which this main function is not going to work (See Appendix V). A detailed flowchart will give better explanation of the code.

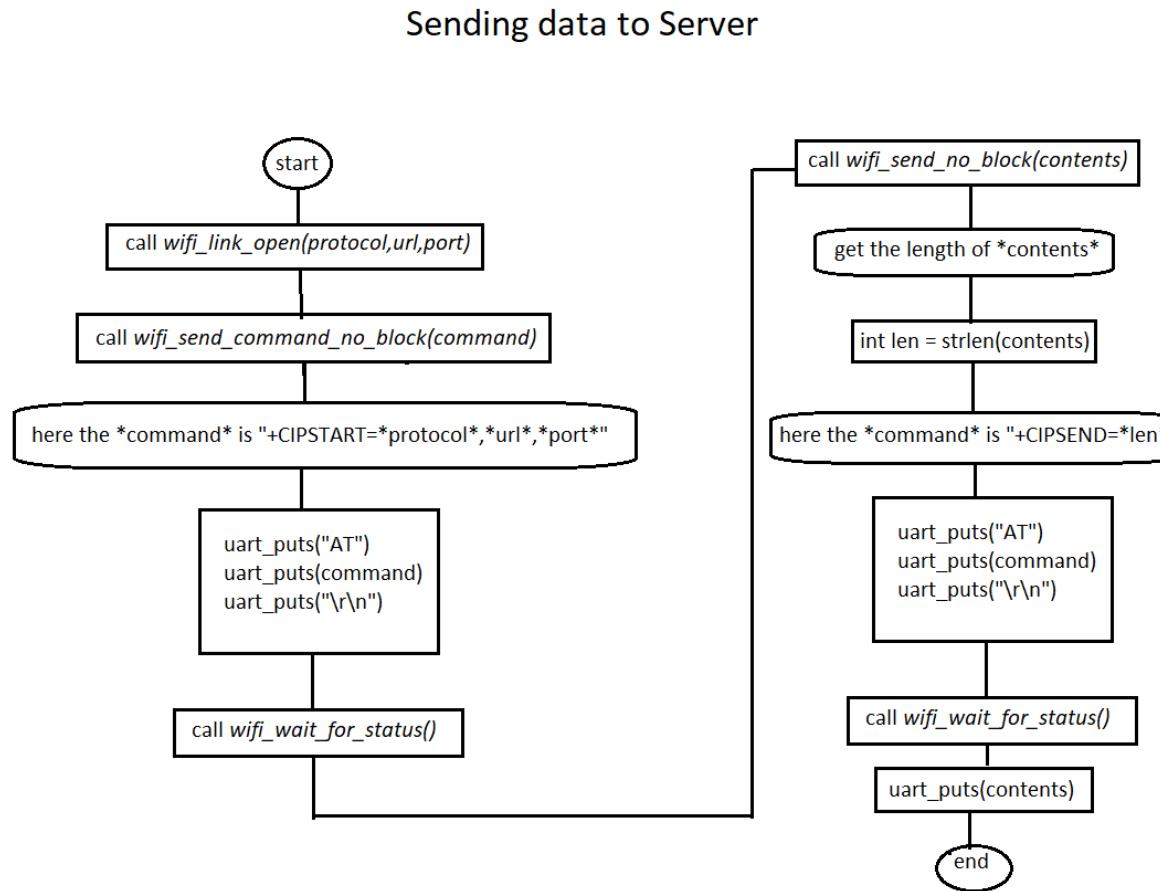


Figure 5. 8: Data Sending Flowchart

### 5.6.6 Wifi Listen (ESP Response Code)

Of all the wifi functions that we have talked about so far, almost all of them either waited for status or waited for data, from the ESP. The wifi methods, termed as `wifi_wait_for_status()` and `wifi_wait_for_data()` all called a single method, from their code blocks termed as `wifi_listen()` (See Appendix W).

This function initializes a char array termed as `wifi_command_buffer` which gathers the string responses coming from the ESP. The response strings are collected through `uart_getc()`. After

continuously storing ESP response strings or data returned from the server in the char variable, the function *wifi\_event\_handler()* is called within, whose main purpose is to compare the strings between the *wifi\_command\_buffer* char array and some predicted response strings from the ESP.

By cross-checking all the strings, we can ensure responses made for distinct AT Commands made by the ESP. As long as the TCP connection is maintained, the server can also send back some data back to ESP and through USART Communication back to the microcontroller. These wifi events are handled in the main function by switching cases, making sure the correct wifi event is being handled.

In cases, where data is being transferred through ESP to the server to compute the Real Power, Balance is continuously being deducted in the server. When the balance reduces to a certain amount, server sends the string “disable” to the ESP and ATmega32. Upon receiving this string, the code disconnects the overall connection of the building through a Relay switch. If the server passes “ok” the connection is maintained.

# Chapter 6

## Importance of RTC

This project required an RTC (Real Time Clock) for operation. Although this real time was not anywhere shown in the database in the server, the importance of passing the real time information is vital for the whole operation. By analyzing the time gap, between the real time when the rms voltages and currents have been recorded, and the time when these information are posted in the server, the administrators can identify any problem or suspicious activity around the meter. For the real time calculation, an RTC was used, model DS3231. It uses TWI (Two wire Interface) Protocol for communication with the microcontroller.

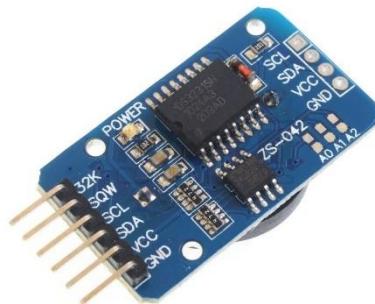


Figure 6. 1: DS3231 RTC Module [36]

### 6.1 Brief Discussion on TWI Communication Protocol for RTC

The TWI Protocol is best suited for typical communication operations. This protocol allows the system designer to interconnect with 128 different devices [4], with only just Two Bidirectional Wires, One known as SCL which is used for clock (PC0 in the ATmega32), the other is SDA, which is used as the data line (PC1 in the Atmega32).

Term	Description
Master	The device that initiates and terminates a transmission. The master also generates the SCL clock.
Slave	The device addressed by a master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

Table 6. 1: TWI Terminology [24]

Each data byte transferred to the slave device is accompanied by a clock pulse. Data line must remain stable (either high or low) during SCL remains high. The figure 6.2 tells us the necessary conditions for data transmission in TWI. The only exceptions to this rule are, Start Bits and Stop Bits.

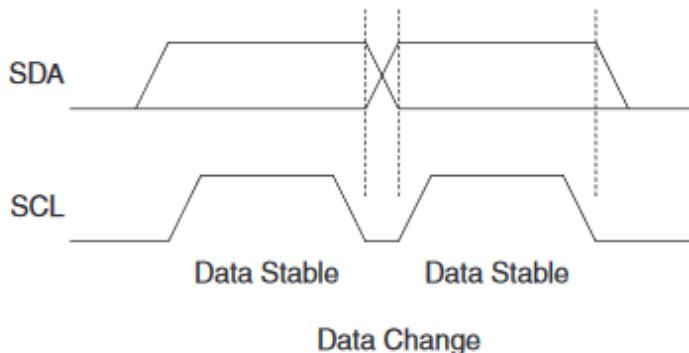


Figure 6. 2: Data Validity for TWI [24]

## 6.2 TWI Registers

There are five Registers in Total for Two Wire Interface [28] (See Appendix MM).

When the TWI is operating in master mode, the TWBR register determines the bit rate and controls the period of SCL. TWBR was set to 0x0F for setting the clock frequency to 347 kHz for 16 MHz Crystal Frequency used.

TWCR is the Control Register that contains an Interrupt bit TWINT which is set mainly when the TWI is done with its current operation. In our code, the TWINT bit was monitored for all operations, such as start condition and stop condition Initiation, data transmission and receive. TWEA bit enables Acknowledgement which has not been used here. TWSTA and TWSTO are respectively the start condition bit and the stop condition bit, setting this bit to 1 will generate a start / stop condition respectively. TWEN enables TWI communication which is set to 1. TWIE enables Two Wire Interface interrupt TWINT. So this bit must be set to 1.

TWS [7:3] bits in TWSR, reflect the status of the TWI Logic and Two Wire Serial bus. While TWPS1 and TWPS0 are Prescaler bits. In the code, the TWSR was set to 0x00.

The two registers, TWDR and TWAR contain the data and address bits respectively for communication.

### 6.3 RTC Code Breakdown

The DS3231 library [9] used in this project has been slightly modified according to our needs. The library was originally made for KS0108 lcd display. This library also uses the twi core library (See Appendix Y).

As we can see that during the *twi\_start()* and *twi\_stop()* TWSTA and TWSTO were set to 1 respectively for Start and Stop Conditions. Upon any TWI events, TWINT interrupt flag was being monitored as can be seen in the code fragment.

This library has been used in the DS3231 which enabled us to get the real time from the RTC for server's use.

The ds3231 library in question has three major functions. And these functions also called the functions from twi core library.

The *ds3231\_init()* function initializes the RTC. First it calls *twi\_init()* to initialize the TWI Communication. Then it initiates a start condition for twi communication using *twi\_start()*. After that it sends 0xD0 using the function *twi\_send()*. This is the address for ds3231 to write the next byte. Then it sends 0x0E similarly, which is the location of the control register of RTC. At the end of this function 0x00 is sent, which is the necessary data bit to make the SQW of the RTC module to 1Hz, to synchronize this module with ATmega32 for every second. The *twi\_stop()* function is called to indicate the stoppage of data transmission.

The *ds3231\_set()* and *ds3231\_get()* works similarly, calling different core functions of TWI library. (See Appendix NN). To write (or set) a time data, we have to send 0xD0 first using *twi\_send()* to indicate write operation of the RTC. Similarly, to read (or get) time data, we have to send 0xD1 to indicate read operation. The different types of time data is retrieved using character pointers. Both of these functions begin with a start condition, and ends with a stop condition.

# **Chapter 7**

## **Server**

### **7.1 Introduction to Server System**

A server is basically a computer which provides data to other computers while it can provide data to itself too. It may serve data to other systems connected on a local area network (LAN) or a wide area network (Wan) over the internet. There are many types of server such as web servers, mail servers, and file server. Each type of server runs on a software which is specific to the purpose. For example, a web server may run Apache HTTP Server, Hiawatha or lighttpd, where all of them provide access to websites over the Internet. A mail server can run programs like postfix (a mail transfer agent) or Microsoft SMTP server, which provides SMTP (simple mail transfer protocol) services for sending and receiving mail. A file server can use Samba or the operating system's built-in file sharing services to share files over a network. While server software is specific to server type, hardware doesn't matter as much. Indeed, by adding the appropriate software, a regular desktop computer can be turned into a server. For example, it is possible to designate a computer connected to a home network as a file server, print server, or both. Most big companies use rack-mountable devices designed specifically for server functions while any computer can be configured as a server. These systems take up minimal space, often 1U in size, and often have useful features such as LED status lights and hot-swappable hard drive bays. It is possible to place multiple rack-mountable servers in one rack and often share the same monitor and input devices. Using remote access software, most servers are accessed remotely, so input devices are often not even needed. Although servers can run on various types of computers, it is necessary that the hardware supports server requirements. For example, a web server running many real-time web

scripts should have a fast processor and enough RAM to handle the load without slowing down. One or more fast hard drives or SSDs should be installed on a file server to read and write data fast. Regardless of the server type, it is critical to have a quick network connection. The project uses a web server and it is quite essential for the long run. A web server is server software, or hardware for running said software, which can satisfy requests from the World Wide Web client. In general, a web server may have one or more websites. Incoming network requests are processed by a web server via HTTP and several other related protocols. The main function of a web server is to store, process and deliver web pages to clients. The communication is established using the Hypertext Transfer Protocol (HTTP) between client and server. The delivered pages are mostly HTML documents which are concluded by images, style sheets and scripts in addition to text content. Using HTTP, a user agent, usually a web browser or web crawler, initiates communication by making a request for a specific resource and the server responds with the resource's content or error message if it cannot. Typically, the resource is a real file on the secondary storage of the server, but this is not necessarily the case and depends on how the web server is implemented. While the primary function is to serve content, it also includes ways to receive content from clients through full implementation of HTTP. This functionality is used to submit web forms, including file uploading. Using Active Server Pages (ASP), PHP (Hypertext Preprocessor), or other scripting languages, many generic web servers also support server-side scripting. This means that the web server's behavior can be scripted in separate files, while the actual server software remains the same. Out of the two, PHP is much faster and more easily cached but cannot deliver dynamic content. Web servers can frequently be found embedded in devices and serving only a local network. The web server can then be used to monitor or administer

the device as part of a system. This usually means that there is no need to install additional software on the client computer, as only a web browser is required. [30]

In our project, the thing we did is send continuous data (volt, ampere, meter id) through a certain a port number. It was done using a transmission control protocol. The server receives data and then trims it into 3 parts as it is taking the meter id, volts and ampere (See Appendix Z).

Servers are of many kinds such as the file server, print server, communications server, application server, database server etc. The type of server which we are using in our project is the database server. By using the method of socket-based programming (See Appendix OO), we are creating an endpoint with which the device can communicate with the server. It enables two host to establish a connection. TCP is one of the main protocols of network. A port number identifies each service offered by a computer uniquely. Each inter packet contains the host and the destination. In our project, we are also establishing connection through a port where the server, in our case the laptop is accepting requests from the atmega32. This programming is done by the help of PHP. Basically, the laptop is containing all the data which are being given by the microcontroller.

## **7.2 Insightful Diagrams to the project**

To get a clear overview of the project we had to create some diagrams which helped us to build the model in an efficient manner. One of them is the sequence diagram.

### 7.2.1 Sequence Diagram

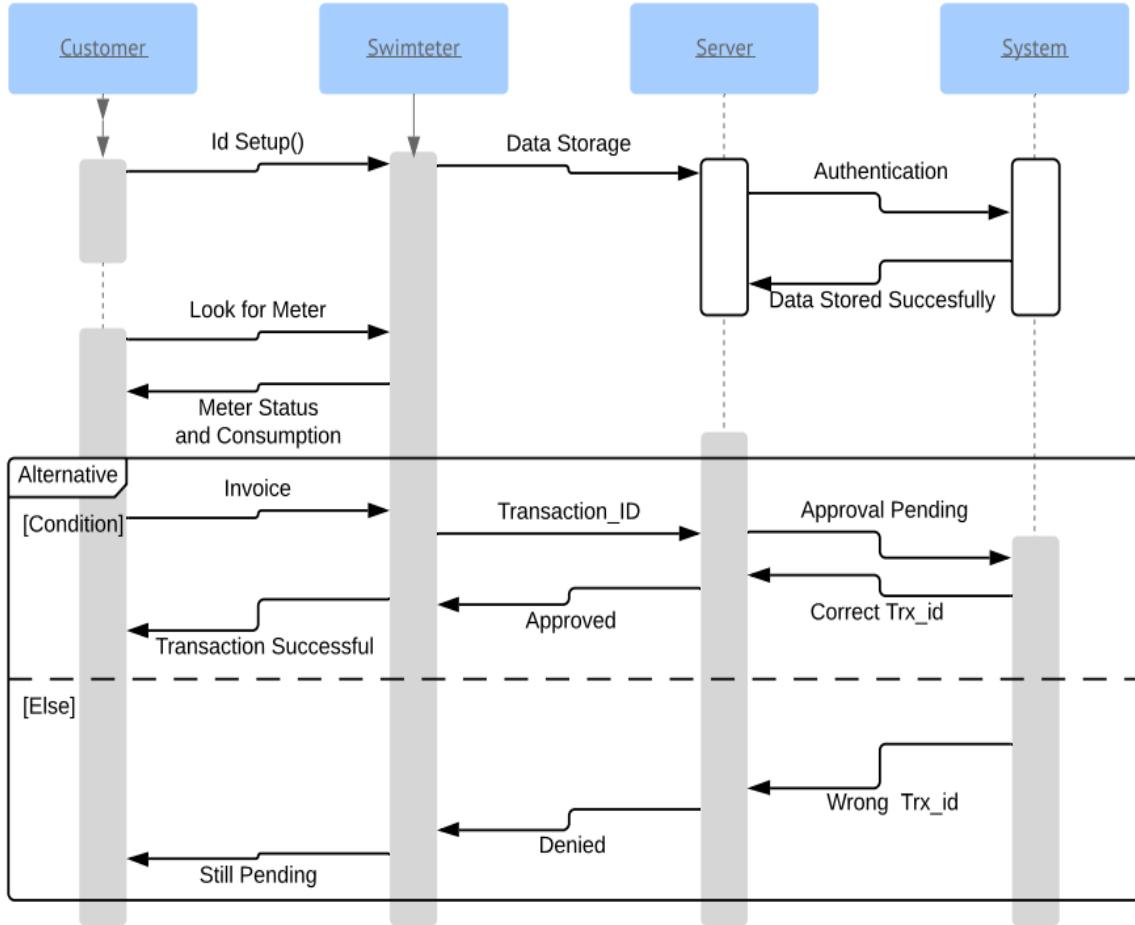


Figure 7. 1: Sequence Diagram

From the Figure 7.1, we can see that the customer is going to the page situated on our local server is setting an Id which includes a lot of things. After that, the data are being sent to the system which is basically an algorithm run on the backend server to check for authenticity and omission of duplicate data. After the checking of data, it is stored in the web server successfully. On the second part, the customer is searching for the meters engaged with his/her account and examine the status and the consumption of a certain meter. If the balance is running low or in a negative figure, one needs to recharge the wallet balance. It can be done through any mobile banking application where the user will enter the transaction id to send the money. After that the server will

check if the transaction id matches with the system. If it matches, then it will automatically add the amount to the user's account and give a message saying that the transaction has been successful. After that, it will normally consume the usage and show the amount of money being cut prior to the usage. If it does not match then the money won't be added and as a result the money will still be pending. This is the main concept of our project from the customer and server's part. Customer can easily register one or many meters in their account to maintain if he/she owns more than one meter.

### 7.2.2 Entity Relationship Diagram

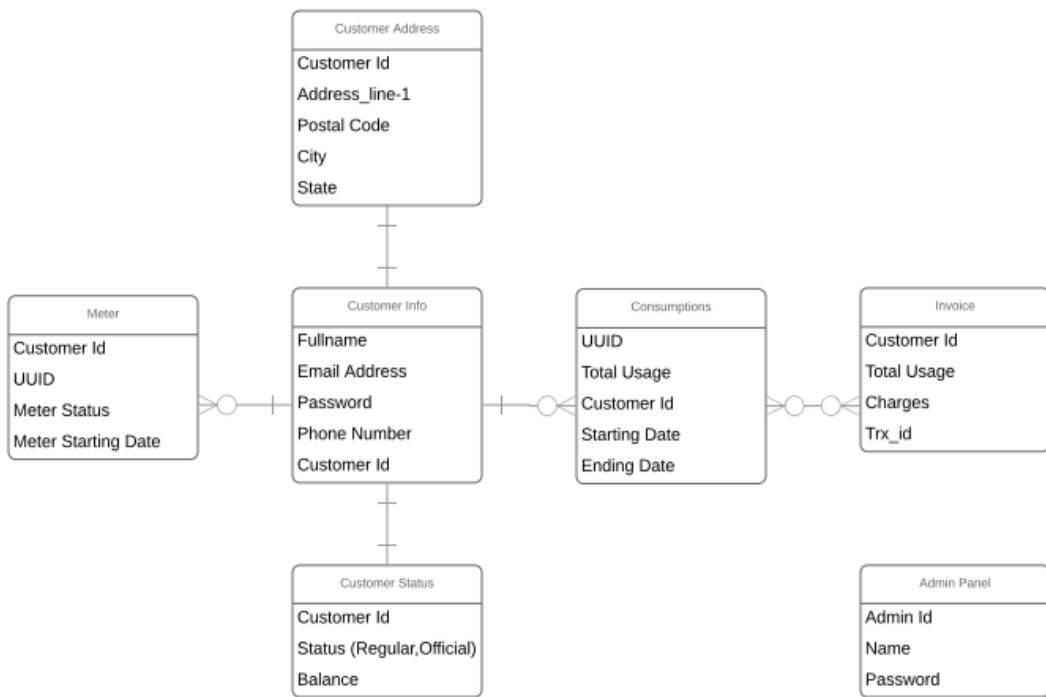


Figure 7. 2: Entity Relationship Diagram

From this entity relationship diagram in the Figure 7.2, we can clearly see the relationships within the fields which are necessary to build this project. In the meter field, there are four attributes which are customer id, UUID, meter status and the meter starting data. This data are essential because a meter must be identified with a customer id and an UUID. The meter status is also necessary as it tells us if the meter is still in use and gives us the status regarding it. The status contain success/ok, disable, invalid (in the server), and device invalid (in the server). This meter field is connected with the customer info part which is crucial for determining the meter of a person. A person can have many meters as they want. It's only normal that he/she can have many house/flats and according to that he/she might have two to three more meters whereas a meter can only be registered to one person. The customer info field is filled with attributes like full name, email address, password, phone number and customer id. This is the field where actually we can know some details of a person. It gives us the basic idea of a person. From that customer info field we can go to the customer address field which is basically a one-to-one relationship. It is supposed to be a one-to-one relationship because it directs you to a page where the location of the customer id can be seen. This field consists of the customer Id, address\_line-1, postal code, city and state. These attributes are quite necessary to locate the person if there are any problems regarding the meter. The customer status field also comes out form the customer info field. They are also related with a one-to-one relationship. It consists of two things named as status and balance. In this part, the balance can be seen. It can be credited by different ways. The balance can also show a negative number when it runs out of money. After that, we can enter to the vital part which is the consumptions field. The consumption field is filled with attributes such as the UUID, total usage, customer id, starting date and the ending date. The UUID is exclusive to the meter and it also passes on to the consumption field when the user wants to see the details of a meter. Total usage

can be seen in the consumption field with the starting date and ending date of the usage till that day. Customer info has a one-to-many relationship with the consumption field because a customer can have one or many meters and he/she can check the details of the particular meter registered to that account. From the consumption we can go to the invoice field which has a many-to-many relationship with it. In invoice, the user can see the total usage for a tentative period. It is quite helpful as it gives an idea of how much usage are happening for a day, week, month or a year. The invoice also consists of a transaction Id which actually verifies the authentication of the money prior to the system. The cost is something which the user cannot see from their interface. It is calculated based on the system of our country. The balance actually is debited simultaneously after the usage which is continuous. The admin part has an admin id, name and a password. They can look for a particular user or a meter to make changes on it. Admins have the full control to make changes on anything except the transaction Id. The authentication is within the system and the balance can only be credited if the transaction id is correct. (See Appendix AA)

### 7.3 Tier Based Cost

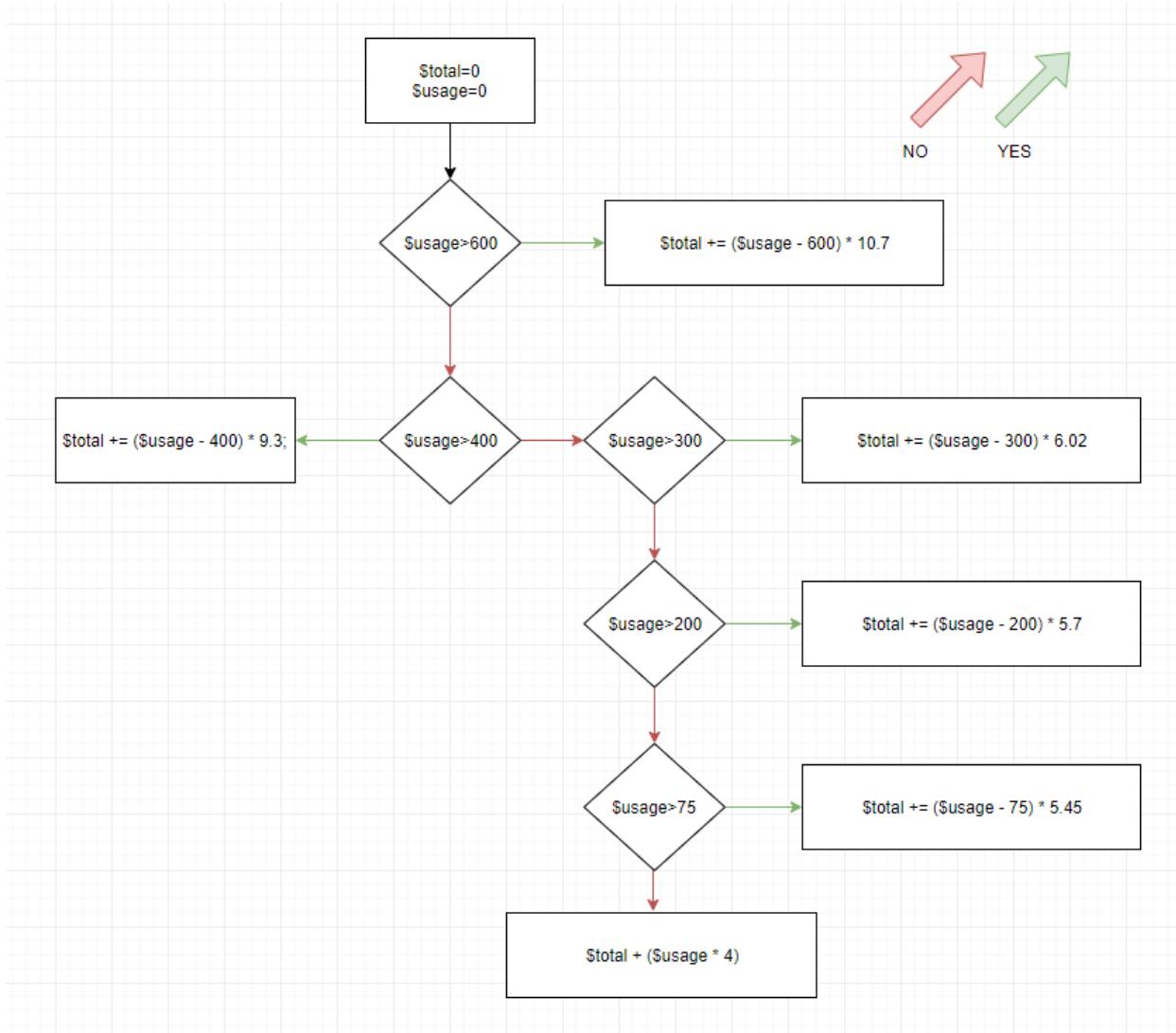


Figure 7.3: Tier Based Cost Flowchart

From the above flowchart in the Figure 7.3, we can see that we have set a structure for the usage flow. This calculation is important for the project as it shows the accurate calculation of the watt per hour. We broke down the calibration of the voltage and ampere on the main code which was discussed in the main-code function. So, at first we can see that the usage is zero as well as the total is also zero. After that it comes to the first threshold, if the usage is above 600, the usage is

multiplied by 10.7. If it is below 600 but above 400, then the units will be calculated by 9.3. In the third threshold, usage is multiplied by 6.02. The fourth and fifth one is followed by 5.7 and 5.45. The base tier is multiplied by 4. We developed a system where it will calculate the usage till now from the last usage. (See Appendix BB)

#### 7.4 Relay Enabler Diagram

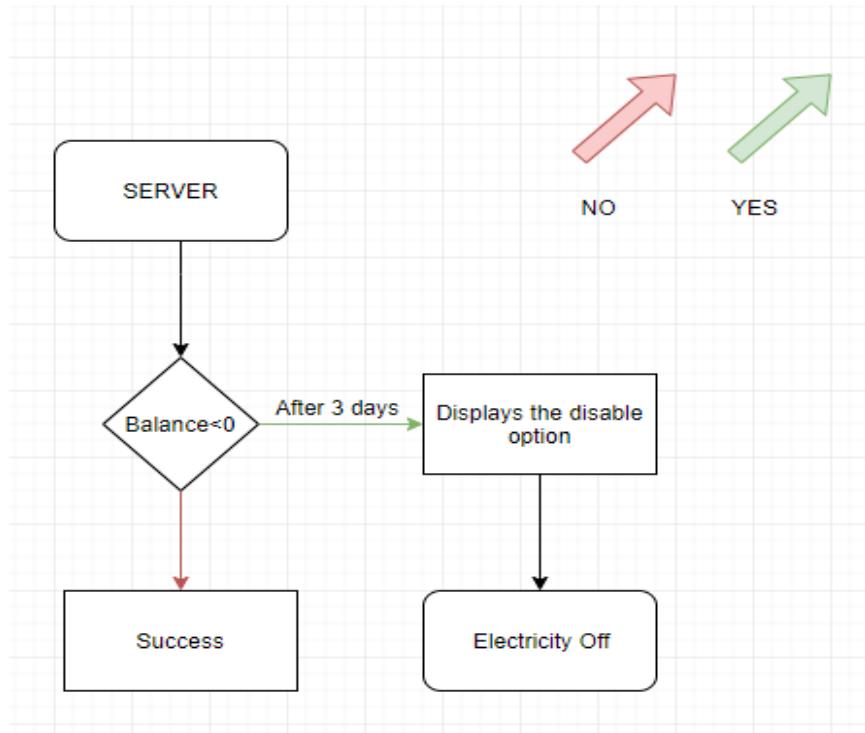


Figure 7.4: Relay Enabler Diagram

Relay enabler is a short side of the server program where the server sends a message saying ‘disable’ to the microcontroller. It is done if the balance of the user is below 0 and it has been zero for the past 3 days like that. After that the meter will be programmed to be off from the hardware part. If the balance is not zero then a continuous message of ‘success/ok’ is being sent to the display. The flowchart of the relay enabler diagram is shown in the Figure 7.4.

## **7.5 Server Part Code Breakdown**

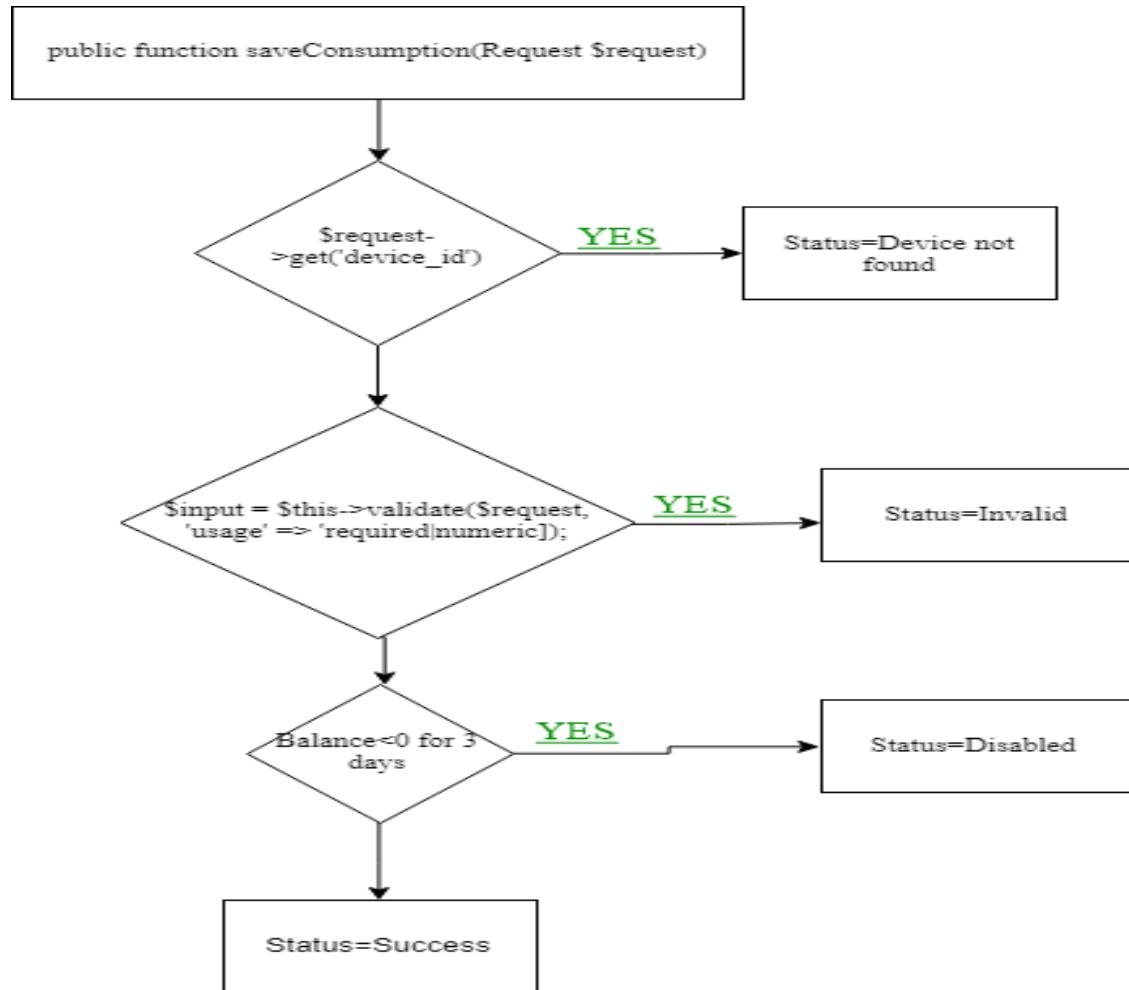
The server is the most pivotal part of the whole project. Most of the events and the calculation lie on this part of the project. If we have to describe the server part in details then we have to break it down to a MVC framework. MVC usually stands for model, view and controller.

### **7.5.1 Controller Part**

We can break the controller part into many small parts such as the consumption controller, meter controller, forgot password controller, login controller, register controller, reset password controller, verification controller, account controller, billing controller, controller, dashboard controller and meter controller. The admin part also consists of some controllers such as the dashboard controller, invoice controller, meter controller and user controller.

## **Consumption Controller:**

In this part, we can see some status of the meter according to the following events



*Figure 7. 5: Consumption controller Flow-Chart*

From the above flowchart in Figure 7.5, we can see that a request is being caught on the `saveConsumption` function. The request then searches for a device id. If device id is not found the server will have an error and tell that the device was not found. But, if the device id is valid, it will go in the next step and check if the usage is being caught on the server. If the server cannot catch any usage it will show an invalid status. If both steps are a success, it will go and check if the balance is below zero for three days. In that case, the server will send a signal to disable the device.

Finally, if all the steps are successful, the status will show as a success/ok and store it in the web server and the database.

The consumption controller also resets the usage after a month so that the user can see and analyze the cost and usage of a month. (See Appendix CC)

### **Login Controller:**

Login controller consists of two functions which are showLoginForm and login. showLoginForm shows a form with the username and password field that needs to be filled by the user.

### **Reset Password Controller:**

Reset Password Controller is used to change the password. It is normal for a user to forget their password and for this reason we have included the option to change it. Initially, a form is shown to the user where he/she needs to put the email address. Once the form is submitted, the system checks if the email address exists in the database or not. If not, an error message is shown, otherwise an email containing a link to set a new password is sent to that email address.

### **Account Controller:**

In this controller a function called showAccountInfoPage is being used. Basically here we can create an account or update it. The account usually needs a name, email, phone, password, address line 1, address line 2, postal code, locality, administrative area and country. (See Appendix DD)

### **Dashboard Controller:**

Dashboard controller basically shows the consumptions in a graph, total energy usage and available balance.

### **Meter Controller:**

Meter controller shows a list of all the available meters of the user and when clicked, shows all the information related to a meter and its monthly usage is also shown.

### **Invoice Controller:**

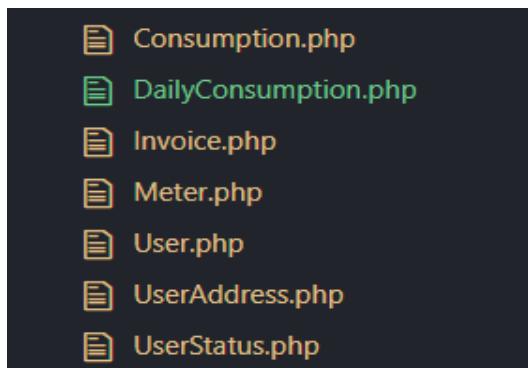
It lists all the invoices for the admin and can manage individual invoices which includes approving or disapproving the invoice. After approving, an amount is being transferred to the user's account.

### **Billing Controller:**

In this controller, the user can view his/her invoices, check the status or submit new BKASH transaction ids to top up their balance.

## **7.5.2 Model**

The Model component is consistent with all the logic related to the data with which the user works. This can either represent the data being transferred between the components of the View and Controller or any other data related to business logic. A customer object, for example, will retrieve customer information from the database, manipulate it and update it back to the database or use it to render data.



*Figure 7. 6: Models*

In our project, the models are being cut down to seven parts which are consumption, daily consumption, invoice, Meter, user, address and user status. In the consumption we can see a protected array consisting of usage, cost, created at, updated at and the time stamp of the details are set. The invoice is filled with amount, charges, transaction id and status. It also consists of the public function user to determine the user. In the meter part, there are status and UUID where the UUID is a protected function. The user function is also in it. The user part consists of some public functions such as the status, meters, consumptions and invoices. Lastly, the user status part keeps the status type and the balance. The models we used in our project are given in the Figure 7.6.

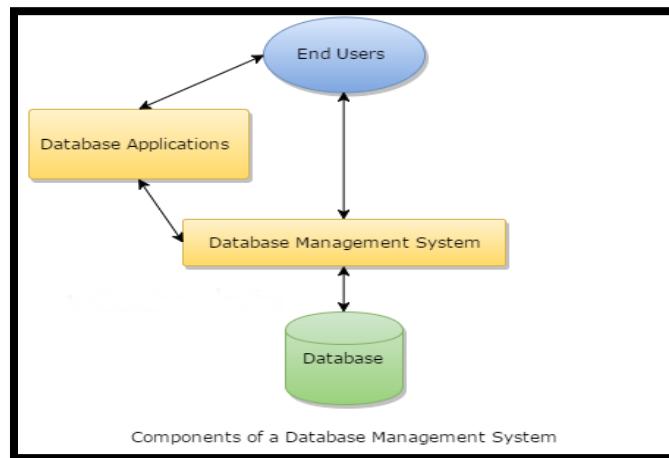
### 7.5.3 View

The view is where the user is presented with the data provided by the model. A view supervises the elements of the visual (or other) interface. It selects, filters, and arranges the model's information. Different views can present information in different ways, managing the display of information on a mobile or desktop browser would be an obvious implementation of multiple views. Keep in mind that there are many different ways to implement an MVC architecture, and the above is a high-level sketch of the fundamental principles and does not reflect any particular implementation.

## 7.6 Database

A database is a collection of an organized data, generally stored from a computer system and accessed electronically. They are often developed using formal design and modeling techniques where databases are more complex. The database management system (DBMS) is the software that interacts to capture and analyze the data with end users, applications, and the database itself.

Additionally, the DBMS software includes the core facilities for managing the database. The total sum of the database, DBMS and related applications can be called a "database system." The term "database" is also often used to refer loosely to any of the database-related DBMS, database system, or application. Computer scientists can classify database management systems based on the models they support in the database. In the 1980s, relationship databases became dominant. In a series of tables, these model data as rows and columns and the vast majority use SQL to write and query data. Non - relational databases became popular in the 2000s, called NoSQL. Our project uses the HeidiSQL database for storing and managing the databases. HeidiSQL is an open and free management tool for MySQL and its forks, as well as for Microsoft SQL Server and PostgreSQL. The Figure 7.7 shows a database management system. [32]



*Figure 7. 7: Database Management System [33]*

Our project also uses a database which is handled by a database management system like the HeidiSQL and the end-users and admins can engage with the data as well as in the website.

 consumptions	80.0 KiB
 daily_consumptions	32.0 KiB
 invoices	32.0 KiB
 meters	48.0 KiB
 migrations	16.0 KiB
 password_resets	16.0 KiB
 users	48.0 KiB
 user_addresses	32.0 KiB
 user_statuses	32.0 KiB

Figure 7. 8: Data Tables

These are the tables of our database which consists of lots of data. The Figure 7.8 shows us the tables we have used in the database structure. The consumptions table consists of auto-increment id, meter\_id, usage, cost, created\_at, updated\_at. The usage are calculated from the server part where the server gets the data from the calibration of voltage and current. The cost is being calculated according to the tier-based cost calculation in the server. The time is also saved in the database to get a clear usage of the short period time interval. The other tables are invoices, meters, users, user\_addresses and user\_statuses.

## Consumption:

The screenshot shows the MySQL Workbench interface. On the left, the database schema is displayed with the following structure:

```

localhost
  information_schema 176.0 KiB
  mysql
  performance_schema
  swimeter 336.0 KiB
    consumptions 80.0 KiB
      daily_consumptions 32.0 KiB
      invoices 32.0 KiB
      meters 48.0 KiB
      migrations 16.0 KiB
      password_resets 16.0 KiB
      users 48.0 KiB
      user_addresses 32.0 KiB
      user_statuses 32.0 KiB
  test

```

The `consumptions` table is selected and its data is shown in the main pane:

id	meter_id	usage	cost	created_at	updated_at
850	1	0.00010222	0.00040889	2019-04-07 18:07:46	2019-04-07 18:07:46
849	1	0.00010217	0.00040868	2019-04-07 18:07:27	2019-04-07 18:07:27
848	1	0.00010219	0.00040877	2019-04-07 18:07:08	2019-04-07 18:07:08
847	1	0.00010195	0.00040781	2019-04-07 18:06:49	2019-04-07 18:06:49
846	1	0.00009683	0.00038731	2019-04-07 18:06:30	2019-04-07 18:06:30
845	1	0.00010219	0.00040874	2019-04-07 18:06:12	2019-04-07 18:06:12
844	1	0.00010764	0.00043055	2019-04-07 18:05:53	2019-04-07 18:05:53
843	1	0.00010222	0.00040887	2019-04-07 18:05:33	2019-04-07 18:05:33
842	1	0.00010209	0.00040836	2019-04-07 18:05:14	2019-04-07 18:05:14
841	1	0.00010223	0.00040891	2019-04-07 18:04:55	2019-04-07 18:04:55
840	1	0.00010195	0.00040780	2019-04-07 18:04:36	2019-04-07 18:04:36
839	1	0.00010225	0.00040901	2019-04-07 18:04:17	2019-04-07 18:04:17
838	1	0.00010197	0.00040787	2019-04-07 18:03:58	2019-04-07 18:03:58
837	1	0.00010226	0.00040904	2019-04-07 18:03:39	2019-04-07 18:03:39
836	1	0.00010225	0.00040898	2019-04-07 18:03:20	2019-04-07 18:03:20
835	1	0.00009679	0.00038716	2019-04-07 18:03:01	2019-04-07 18:03:01
834	1	0.00010216	0.00040862	2019-04-07 18:02:43	2019-04-07 18:02:43
833	1	0.00009689	0.00038755	2019-04-07 18:02:24	2019-04-07 18:02:24
832	1	0.00010211	0.00040843	2019-04-07 18:02:06	2019-04-07 18:02:06
831	1	0.00009677	0.00038707	2019-04-07 18:01:47	2019-04-07 18:01:47
830	1	0.00010213	0.00040850	2019-04-07 18:01:29	2019-04-07 18:01:29
829	1	0.00010229	0.00040818	2019-04-07 18:01:10	2019-04-07 18:01:10

Table 7. 1: Consumptions table

From the Table 7.1, the consumptions table consists of the id, meter\_id, usage, cost, created\_at and updated\_at. The id is an auto-incremental field which chronologically adds number. The meter\_id is also a chronological number but upon clicking that number you will get a unique meter\_id for each and every one of them. The second meter\_id will consist a totally different number than the first one. The meter ids are case-sensitive. The usage shows the usage in units whereas all the cost fields are occupied with a calculation with accordance to the usage. Finally, the created\_at and updated\_at are given because of the RTC.

## Invoice:

The screenshot shows the MySQL Workbench interface. On the left, the database schema is displayed with the following structure:

```

localhost
  information_schema
  mysql
  performance_schema
  swimeter
    invoices 2 rows total (approximately)
  test

```

The `invoices` table is selected and its data is shown in the main pane:

id	user_id	status	amount	txn_id	charges	created_at	updated_at
1	2	approved	995.00	676ruytuyuy	(NULL)	2019-02-12 15:56:12	2019-02-12 16:11:01
2	2	pending	5,500.00	5id981nf2p	(NULL)	2019-03-05 00:10:28	2019-03-05 00:10:28

Table 7. 2: Invoices table

From the Table 7.2, it shows that the Invoice table has id which works same as the field in consumptions table. The status just shows approved or declined. If the Bkash transaction is successful, the system will show approved or else it will be declined. The amount field shows how much money is being recharged after the transaction. Txn\_id shows the transaction id and charges is a field we used for future implementations. We will use this field, if there is a grace period.

### Meter:

swimeter.meters: 3 rows total (approximately)							Next	Show all	Sorting	Columns (7/7)	Filter
id	user_id	status	uuid	created_at	updated_at	grace_period_started_at					
1	2	active	5076eb63-ec36-4354-a663-5a7f3dcf7986	2019-01-11 20:45:31	2019-04-01 23:20:38	(NULL)					
2	2	active	69c552fa-3134-4146-9fc5-fbafe63f1579	2019-03-05 00:27:24	2019-03-05 00:27:24	(NULL)					
3	4	active	9663d48b-56ee-476d-9f83-3fe51d478898	2019-03-05 00:28:55	2019-03-05 00:28:55	(NULL)					

Table 7. 3: Meter Table

It shows the user\_id, uuid, status (whether is it active, disable or device).

The meter table consists of user\_id which was explained in the consumptions table. The uuid consists of the unique meter id and finally it introduces us to a status field where two things can be seen. They are disabled and active. Active is shown when the meter is active while its shows disabled when the meter is not connected. The table 7.3 shows us the descriptions.

### Users Table:

swimeter.users: 6 rows total (approximately)							Next	Show all	Sorting	Columns (10/10)	Filter
id	is_admin	name	email	phone	email_verified_at	password					
1	1	Shahrukh Sattar Chowdhury	srrsc4780@gmail.com	+880 1759-006190	(NULL)	\$2y\$10\$8Pp4R9iyufZqDOLK0gV.N.RSRpfBbGb7ogZVRH...	low60Mb70tWdByqmbQvUJL1ScB2UMKe1afDX...				
2	0	Sharukh Sattar Chudury	srrsc4780+sattar@gmail.com	+880 1734-721318	(NULL)	\$2y\$10\$NL6CegPwEV2fgf3w1.0OL2jtkKLwzdbzICur/t...	xzTRkLatnYBhrdmYbPldKwe7KOsJxPD5fM1lcb1N...				
3	1	Saif Mahmud	sikhiana@gmail.com	+880 1757-111189	(NULL)	\$2y\$10\$0tVBDAzSg1OI7b1fC9.yeohyogL/mo6tQFgPNg...	(NULL)				
4	0	Walur Rawnak Rahman	nomoregloryhole@gmail.com	+880 1757-111181	(NULL)	\$2y\$10\$877ed5GK2nPETcyU0E5Z.2vJi1VLzC5g5CxCCvp...	(NULL)				
5	0	Sapla Sattar Chowdhury	verdascos@yahoo.com	+880 1701-046520	(NULL)	\$2y\$10\$kjekVAvnNsSPYvSRtqIzoO1xuQF01346iofOH99...	(NULL)				
10	0	Seruk Sattar Chowdhury	shahrukh.sattars@gmail.com	+880 1715-331854	(NULL)	\$2y\$10\$SmvTCjeLmT.mn7PyN0/.TSNKdMajuHyXmJZY...	(NULL)				

Table 7. 4: User Table

User table also consists of the admins which can be understood if there is a 1 in the is\_admin row. The passwords are encrypted. In the table 7.4 we can see an unusual field named as the remember\_token. It is only assigned with some value when the user changes his/her password. The other fields are quite normal such as the email, phone etc.

# Chapter 8

## Website

### 8.1 Introduction

Website is an essential part for the end-users. End-users are basically the customers for whom the project is based upon. Our website is a simple one just for the users. It shows the details of the usage of a month as well as the cost of the usage in a window. It is built with the help of PHP.

### 8.2 User-end

As the whole concept completely focuses on the end-users we created some tabs for the users to easily engage with the meters online.

#### 8.2.1 Dashboard

It is a page where the user can see the usage of a current month and the available balance, which is shown with a wallet icon.

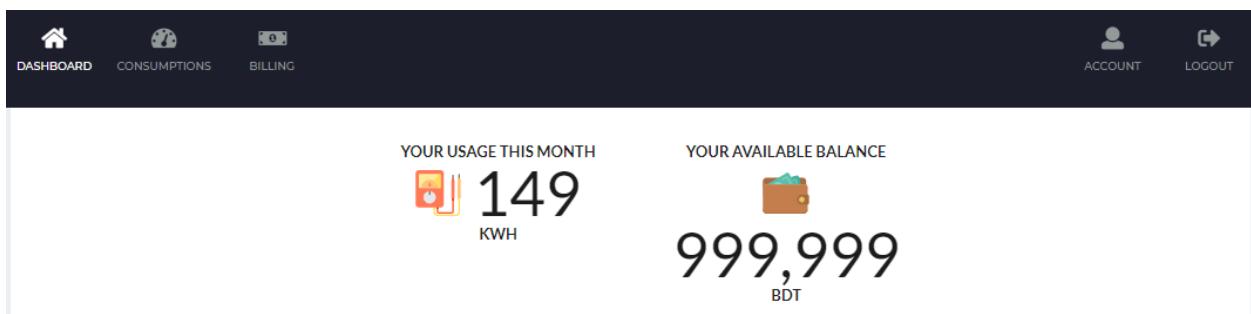
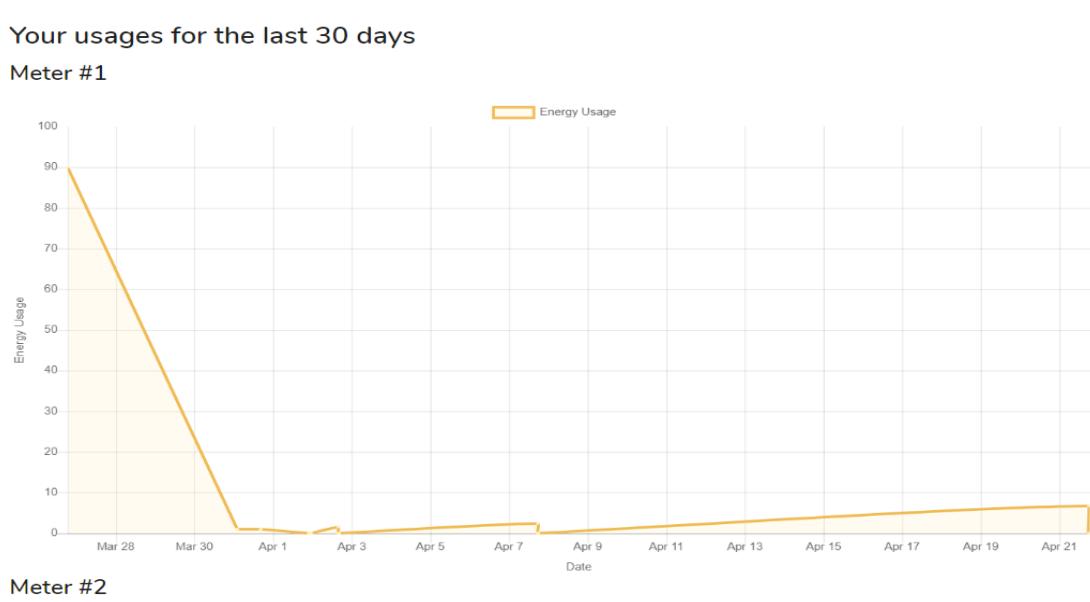


Figure 8. 1: Dashboard upper part

So basically from the Figure 8.1 we can see that the users can see their transactions happening. If the usage goes up, the balance is cut. After the balance becomes zero, it will go to a negative balance given that the connection is still on. This event is set to a maximum of 3 days. After a negative balance for 3 days straight, the meter status is set to ‘disable’ from the server. No transactions will happen unless the user recharges their wallet. This uses the idea of a pre-paid meter system. The dashboard also shows the user a graph from which he/she can see the usage for the past 30 days. If the user has more than one meter registered in their account, the dashboard will show all the meter’s graph registered to that account. All the meter’s usage will be cut through that user’s wallet. A graphical presentation of the usage was made for the user to have a concise idea of the happening events. A graph of a user’s meter usage is shown below:



*Figure 8. 2: Graphical representation of the usage*

From the Figure 8.2, we can clearly see that the user can see how much usage is there for the past 30 days. At the end of March, the user had a huge amount of usage whereas it drastically decreased from the month of April. From the graph, the user can have a clear idea of the events happening and make their monthly budget on consumptions. Meter #2 has no graph because it has no usage.

### 8.2.2 Account

The account keeps all the information of the user which includes full name, phone number, customer type (future implementation), billing address, city, zip code, and state. It also keeps an email address to send out mail to a user attaching the monthly usage table shown in the Figure 8.3.

The screenshot shows a web-based account management interface. At the top, there is a dark header bar with icons for Dashboard, Consumptions, Billing, Account, and Logout. Below the header, the main content area is titled "Account Information". It contains four input fields: "FULL NAME" with the value "Sharukh Sattar Chudury", "EMAIL ADDRESS" with the value "srsc4780+sattar@gmail.com", "PHONE NUMBER" with the value "+880 1734-721318", and "CUSTOMER TYPE" with the value "regular".

Figure 8. 3: Account Information

Another thing which is associated with the account is the password reset system. In case, a user forgets his/her password, a mail is sent to recover it. The verification is done on the mail. A number is generated and told to paste as a token for authenticity. The dashboard is shown in the Figure 8.4

**Change Password**

PASSWORD

Keep this empty if you don't want to change the password.

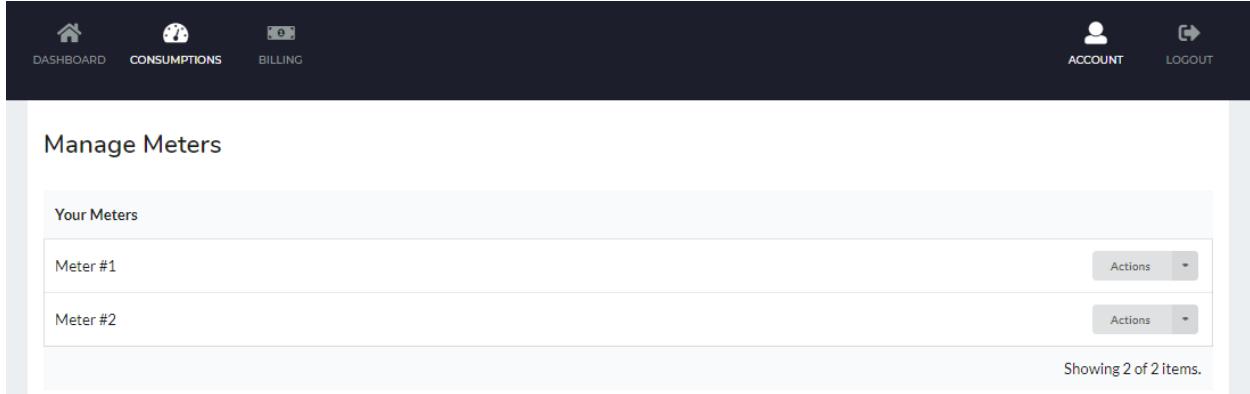
CONFIRM PASSWORD

 **SAVE**

Figure 8. 4: Password Reset Section

### 8.2.3 Consumptions

In the ‘Consumptions’ tab, the user can choose the meter to see its in-depth details



The screenshot shows a dark-themed user interface with a navigation bar at the top. The navigation bar includes icons for Home, Consumptions, Billing, Account, and Logout. The 'CONSUMPTIONS' tab is currently selected. Below the navigation bar, the title 'Manage Meters' is displayed. A section titled 'Your Meters' lists two items: 'Meter #1' and 'Meter #2'. Each item has an 'Actions' dropdown button to its right. At the bottom of the list, it says 'Showing 2 of 2 items.'

Figure 8. 5: Manage meter section of end-user

From the Figure 8.5, we can see the drop down button beside the meters, the user can choose view and see all the meter details. That page will show the meter and all the data related with it. The user can choose a time period from the calendar and see the usage of a particular timeline. The

graph of the timeline is shown in the Figure 8.6. The meter's UUID is also given in the page which can be seen from the user easily. The total usage for the recent month can also be seen from the consumptions page. Another thing which is related with this page is the most essential and important part. It is the meter status. Meter status is very important as it shows if the meter is active or disabled.

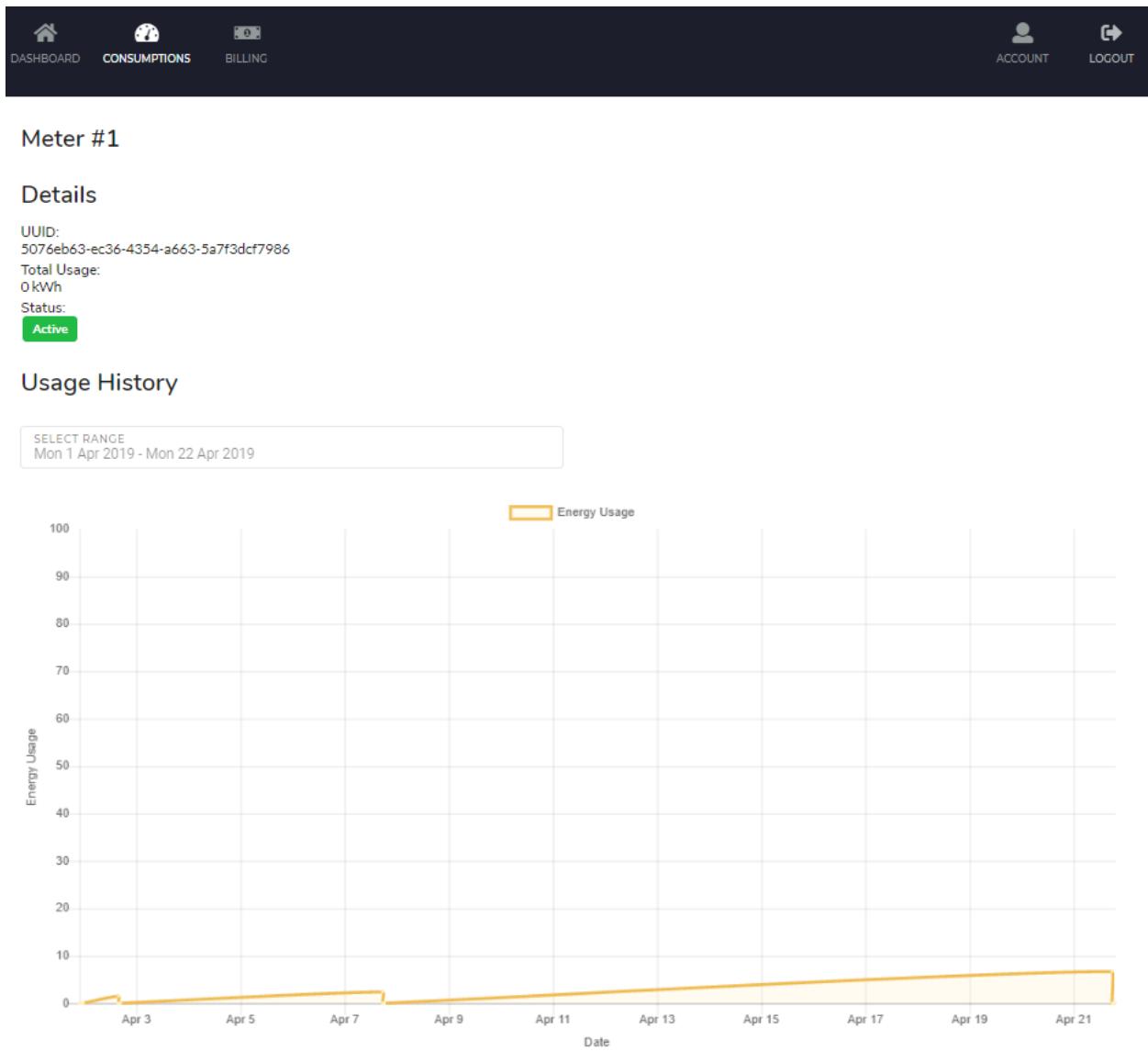


Figure 8. 6: Customized Graph with accordance to timeline

### 8.2.4 Billing

Billing is a tab where the customers can recharge their wallet. It is done with help of BKASH. In future, we are going to set many more companies for this kind of transaction. If we put a transaction id number in the ‘validate new transaction’ box and press validate, the system checks the authenticity of it. If it matches, it will show a message showing status approved and the amount of money which has been credited to the balance. Besides this billing system, the user can go to a bank and give the money. This online system was done to save time for the common people. The page of the billing section is given in the Figure 8.7 below.

The screenshot shows the 'Billing' section of a web application. At the top, there is a dark header bar with icons for Dashboard, Consumptions, Billing, Account, and Logout. Below the header, the main content area has a title 'Your Bills'. A 'Validate New Transaction' button is visible above a text input field labeled 'BKASH TRANSACTION ID'. A green 'VALIDATE' button with a white arrow icon is positioned below the input field. To the right of the validate button, the text 'Showing 2 of 2 items.' is displayed. Below the validation area, a section titled 'Your Invoices' contains a single item: 'Invoice #1' with the status 'Status: Approved, Amount: BDT 995'.

Figure 8. 7: Billing section

### 8.3 Admin-End

The Admin-End is a very basic part of the website where the admin can change and see all the stored data. It has three tabs. They are dashboard, invoices and users. The dashboard in the Figure 8.8 shows the number of users and meters connected with the system.

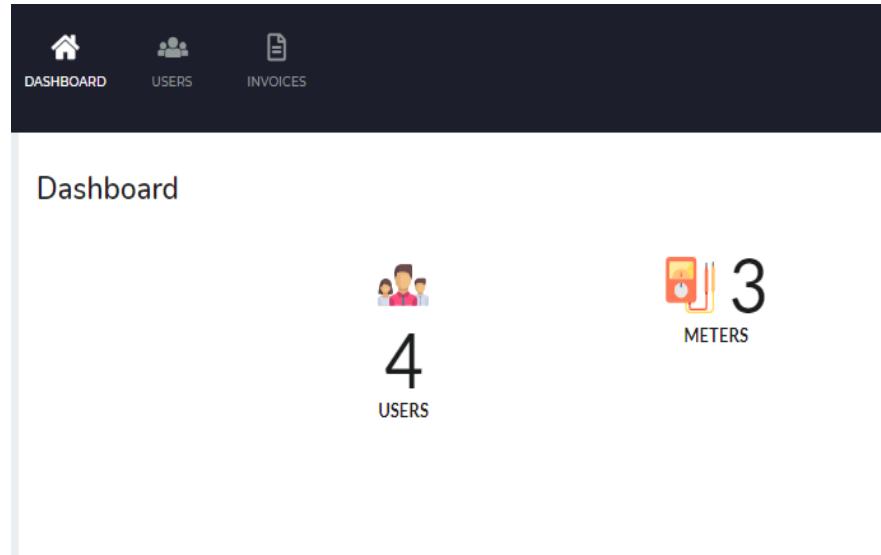


Figure 8. 8: Admin Dashboard

Besides the dashboard, there is the ‘users’ tab where the admin can get a full detail of a user’s account. The admin can also go to a particular user or add/delete a user. A snippet of the page is given in the Figure 8.9

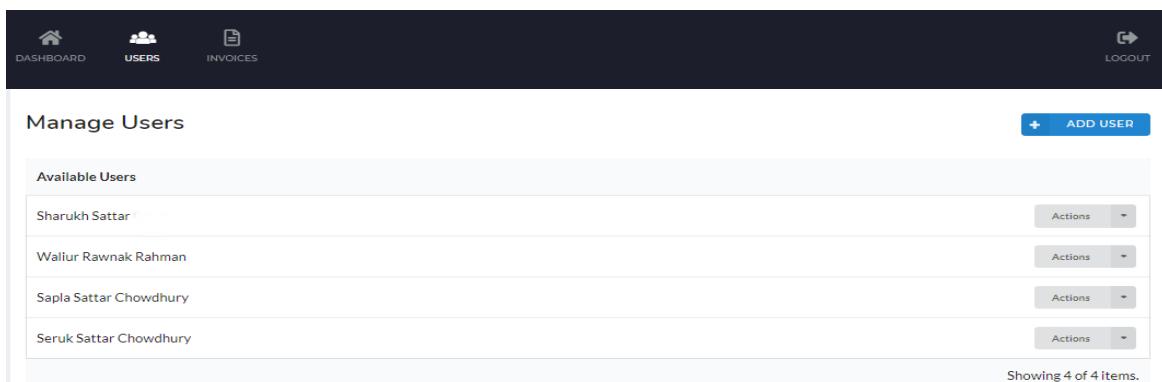


Figure 8. 9: Manage Users Section

From the drop down button of a particular user the admin can find ‘view’ and see the details of the meters related with that account.

The screenshot shows the 'View' page for a user named Sharukh Sattar. The top navigation bar includes links for Dashboard, Users, Invoices, and Logout. Below the header, the user's name 'Sharukh Sattar' is displayed, along with 'EDIT USER' and 'ADD METER' buttons. The page is divided into sections: 'Personal Information' (containing contact details like email and phone), 'Billing Address' (listing the address as 113, Bara Maghbazar, Dhaka-1217, Dhaka-1217, Bangladesh), 'Owned Meters' (listing two meters, Meter #1 and Meter #2, each with an 'Actions' dropdown), and 'User Invoices' (listing one invoice, Invoice #1, with status: Approved, Amount: BDT 995, Transaction ID: 676ruytuyuy, and an 'Actions' dropdown). A message at the bottom right indicates 'Showing 2 of 2 items.'

*Figure 8. 10: View Page*

This is a vital page from the admin part. It is shown in the Figure 8.10. Here, the admin can edit the user information as well as add a meter in that account. The personal information and billing address can also be seen in this page. User invoices list are also shown here. From the available

meters, the admin can go to a particular meter and see a usage table of that particular meter. The usage table contains all the information of the meter.

Consumption History		
Date	Total Usage	Cost
February 28, 2019 23:56:34	798.00000000	BDT667.00
February 16, 2019 00:00:00	60.00000000	BDT72.00
February 15, 2019 00:00:00	42.00000000	BDT8.00
February 14, 2019 00:00:00	40.00000000	BDT32.00
February 13, 2019 00:00:00	32.00000000	BDT4.00
February 12, 2019 00:00:00	31.00000000	BDT24.00
February 11, 2019 00:00:00	25.00000000	BDT20.00
February 10, 2019 00:00:00	20.00000000	BDT8.00
February 9, 2019 00:00:00	18.00000000	BDT4.00
February 8, 2019 00:00:00	17.00000000	BDT4.00
February 7, 2019 00:00:00	16.00000000	BDT12.00
February 6, 2019 00:00:00	13.00000000	BDT12.00
February 5, 2019 00:00:00	10.00000000	BDT8.00
February 4, 2019 00:00:00	8.00000000	BDT8.00
February 3, 2019 00:00:00	6.00000000	BDT8.00
February 2, 2019 00:00:00	4.00000000	BDT8.00
February 1, 2019 00:00:00	2.00000000	BDT8.00

Table 8. 1: Usage Table

From the above Table 8.1, we can see the consumption history. It gives the admin an overall idea of the usage of the meter on a particular date. The last tab is ‘invoices’ where the admin can see all the transactions done by the user.

## **Chapter 9**

### **Final Program and Circuit Integration**

#### **9.1 Description of the Established System**

Based on the discussions of the earlier chapters, the proposed system is developed. The ADE7758 sends  $V_{rms}$  and  $I_{rms}$  register values to ATmega32 via SPI communication. The DS3231 RTC provides real time data and sends it to ATmega32 using TWI communication. The ATmega32 passes on these information to the ESP-12E by UART communication system. Meanwhile the ESP-12E is connected with the nearby Wi-Fi Access Point using its SSID and Password. The ESP-12E establishes a TCP communication protocol with the Server. The ATmega32 sends  $V_{rms}$ ,  $I_{rms}$  register values, Time data, Device\_id of the meter to the Server with the help of ESP. The Device\_id is the unique primary key, by which all of the meters in the server are distinguished. The server keeps data in the necessary tables of the respective meters. With increasing consumptions, the balance of an account keeps decreasing. After each reduction of the balance, if the remaining balance is greater than 0, the server sends a string “ok” back to the ESP, and through that to ATmega32. Upon receiving this string, the ATmega32 does not perform any function. However after the whole balance is reduced to 0, the server sends the string “disable” to ATmega32. Upon receiving this, the ATmega32 cuts off the connection to the load. The connection is established after the balance is replenished again. The whole both way communication continues until the TCP connection is maintained.

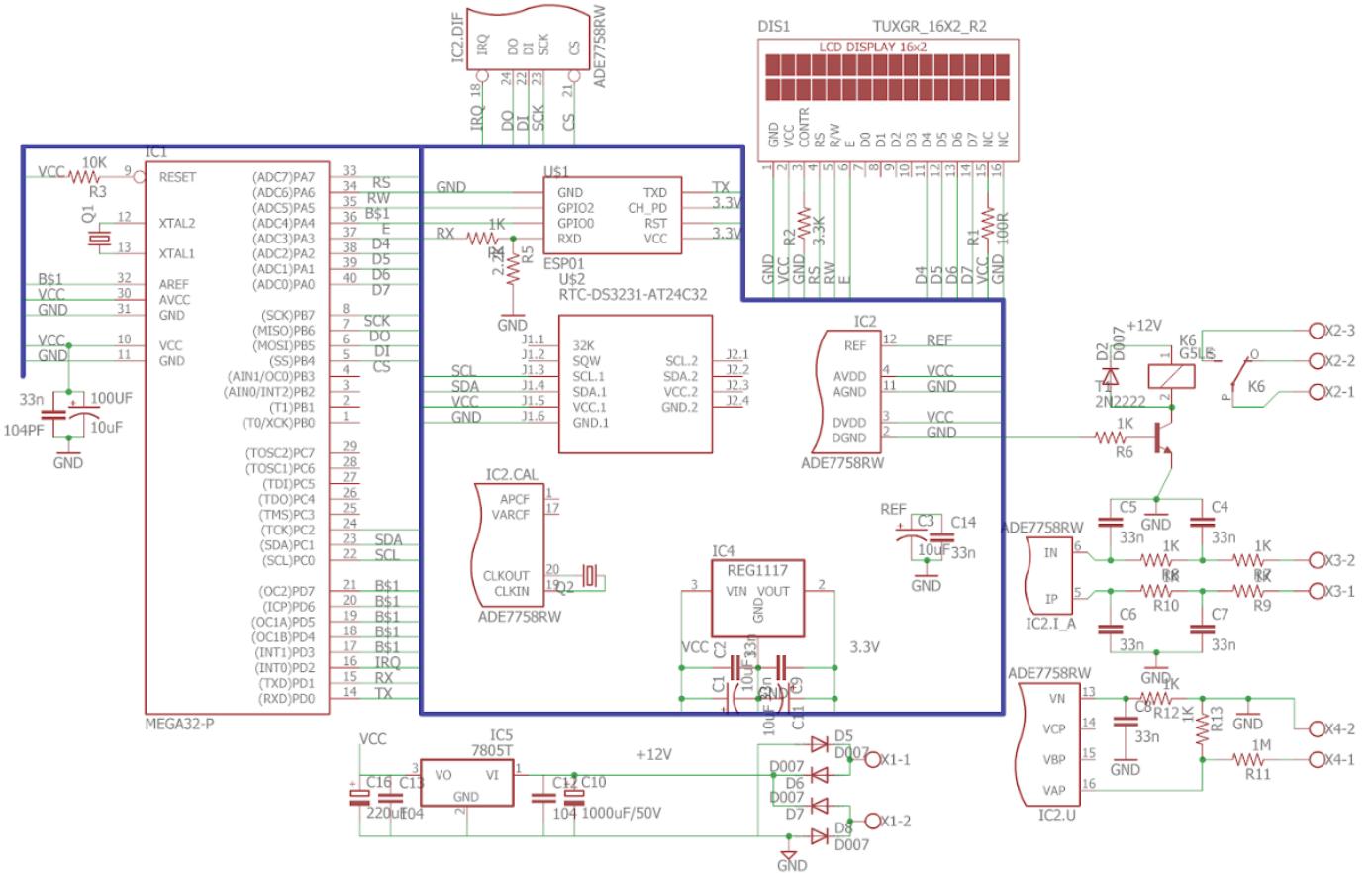


Figure 9. 1: Full Circuit PCB Design

Integrating all of the components, discussed before and connecting all of them gives us the final design. In the final design, the load is connected to the ADE7758 IC in pins IAP (5) and IAN (6) via the Current Transformer (CT). The IC through its SPI serial interface is connected to the MCU which is the ATmega32. The CS (21), DIN (22), DOUT (24) and SCLK (23) pins of the ADE7758 are connected to the SS (B4), MOSI (B5), MISO (B6) and SCK (B7) respectively. The Wi-Fi Module's TX and RX pins are connected to the MCU's RX (D0) and TX (D1) pins respectively. The Wi-Fi module is also connected to a voltage converter which converts the VCC and supplies the Wi-Fi module with 3.3V. A Real Time Clock (RTC) is also connected to the C1

and C0 pins of the MCU. A Relay switch is also connected with the load and the relay channel is connected to D6 pin of the MCU. An LCD is also connected with the MCU.

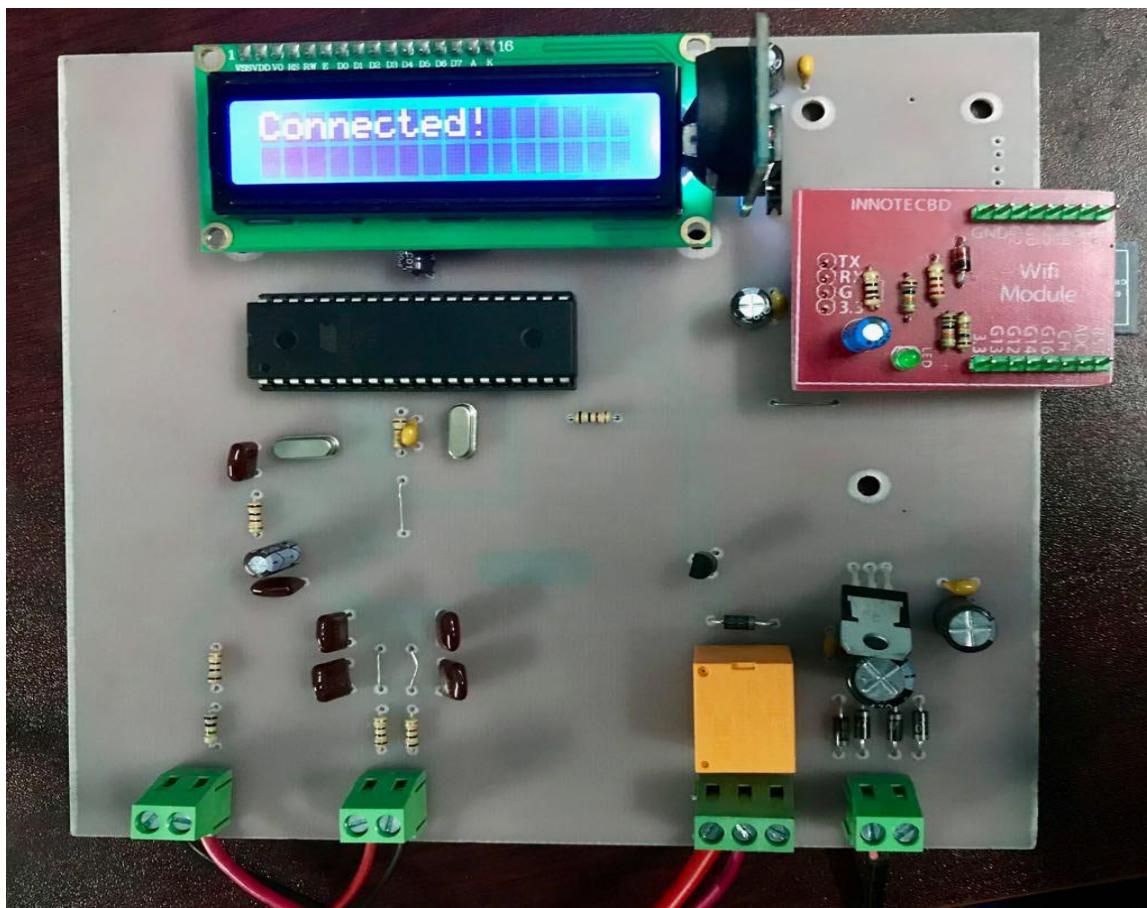


Figure 9. 2: Full Circuit PCB

## 9.2 Main Code of the System

The main function of the program ensures that the  $V_{rms}$  and  $I_{rms}$  data, the device id and the time obtained from RTC, reaches the server, after some short time intervals. The data obtained from  $V_{rms}$  and  $I_{rms}$  registers are not the exact data but the register numbers that would be multiplied by a calibration constant in the server to get the real rms voltages and current. The RTC time is sent to compare between the time data was sent and the time data was posted in the server. The device id

acts as the primary key for the meter in the database in the server. The device id is the key factor by which the data would be categorized to different meters individually.

### **9.2.1 Main Program Code Events**

1. The Microcontroller takes Access Point SSID, Access Point Password and the device id as Input. So that the microcontroller can connect to a specific wifi connection from the access point listed. The device id is obtained for meter identification.
2. The USART is initialized through calling `uart_init()`. The microcontroller communicates with the ESP-12E through USART communication and it needs to be initialized. The baud rate and the clock frequency is taken as parameters.
3. For the SPI communication, PB4, 5 and 7 are all initialized as output as they are Slave Select, MOSI and SCLK respectively. PB6 will be initialized as Input as it is the MISO pin. ATmega32 acts as the Master Device.
4. PB4 is set as high as default while PB5 and PB7 are low as default. When there are no data to be transmitted, SS needs to stay high and there are no clock pulses, so SCLK should remain low as default.
5. Calls `wifi_init()`. Initializes the ESP by sending AT command “AT+RST”. Also echo is disabled using “ATE0”.
6. Calls `ds3231_init()` method. Initializes the RTC’s Two Wire Interface. Sets the clock frequency in TWBR.
7. Calls `wifi_connect()`. This method helps ATmega32 connect to a specific Access Point via the ESP. Here the user given Access Point SSID and Password are passed as parameters for the method.

8. If the ESP fails to connect to a specific Access Point, the Microcontroller gets a response from the ESP as “WIFI DISCONNECT\r”. Until a proper connection is established, or the SSID and the Password are changed, the main function will keep repeating step 7.
9. If connection is established, ESP will give a response as “WIFI CONNECTED\r”. After ensuring the connection, the main function will proceed to next steps.
10. Calls ADE7758\_Write(0x16,0x00) which is setting the computation mode for multiplying rms voltage and current if the WATTHr register is called.
11. Calls ADE7758\_Write(0x15,0x00) which is setting the phase A for all kinds of computation. The main function is designed to work on only 1 phase for now and phase B and C are ignored.
12. Obtain the IP Address, port number and Transmission Protocol.
13. Calls wifi\_link\_open(). This method takes obtained IP Address, Port Numbers and the Transmission protocol as parameters.
14. Calls ADE7758\_READ\_24\_bit(0x0D) and ADE7758\_READ\_24\_bit(0x0A). 0x0D is the register address for rms voltage and 0x0A is the address for rms current. This methods return the values from the registers, which will be later multiplied by some calibration constants in the server.
15. Calls ds3231\_get(), which returns the time calculated in the RTC and forms it to string.
16. Calls wifi\_send(). This method sends all the data to the server. The data transmitted are the RTC time,  $I_{rms}$  and  $V_{rms}$  values and device id.
17. If the balance money in the server is greater than 0, the server sends the string “ok”. Ensuring the string is “ok”, the main function maintains the circuit connection. Else, if the money in the server is null, the server sends the string “disable”. Ensuring the string is

“disable”, the main function disconnects the circuit connection using latch relay. The relay is activated using an output pin.

#### 18. Calls wifi\_close().

The flow chart of the main program is given in Figure 9.3:

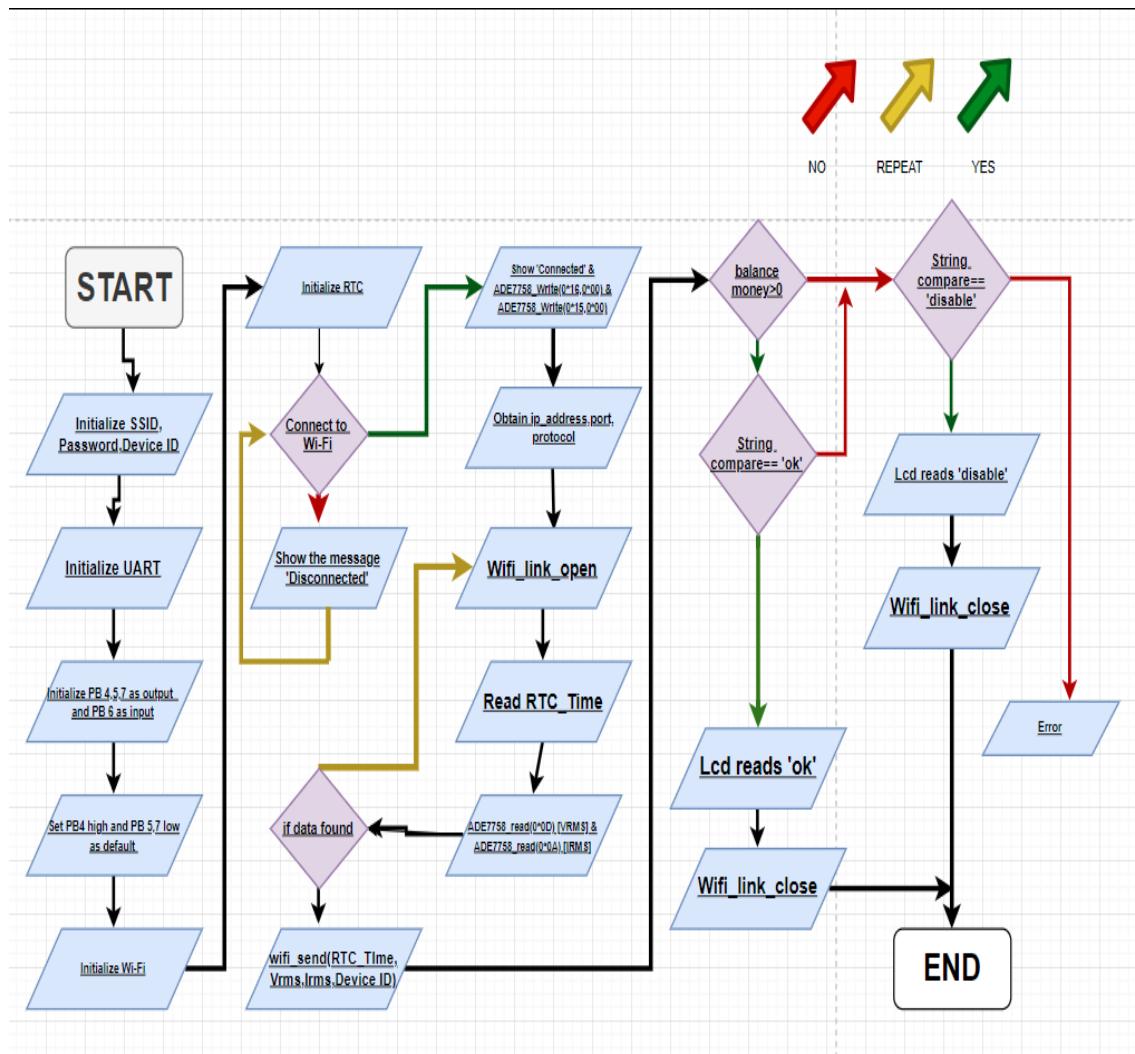


Figure 9. 3: Main MCU program flow chart

## 9.2 Server Side of the System

The server side is creating a TCP endpoint which is helping the device to communicate with the server. It serves a TCP socket to accept data from meters.

### 9.2.1 Server Side Code Events

1. Creates a socket which is a new TCP server and sets the connection. This action is done for the connection to the microcontroller. It gets the host and port number.
2. After getting data, it is trimmed into 3 parts. If the count does not match the number 3, then the data will be invalid.
3. The device id is checked at first. If the device id is not met, it will show the message ‘device’ which basically means that the device was not found.
4. The volts and ampere which are received in the server are in hexadecimal format. Those volts and ampere are converted to decimal.
5. The duration of the time window is being calculated. Basically, we subtract the last data arrival form the one we received now to get the time duration.
6. Checks if a new month has started. The usage turns to zero when a new month arrives and per unit cost also starts from the base.
7. The calculation of the real power consumption which is known as the usage is given below:

$$Usage = \frac{((Volts * Vrms\ constant) * (Current * Irms\ Constant) * time\ interval * Power\ Factor)}{(60 * 60 * 1000)}$$

The voltage and current are multiplied by a constant and the time interval followed by a

power factor equaling the value of 0.7. After that we divide the part with  $(60*60*1000)$  to get in unit.

8. If the balance is less than cost the server will set the meter status as disable. It is done because the system won't be able to cut the balance form the user's available balance.
9. If the transaction is successful the meter will check if the status is disabled. If the status is disable then it will change the status to active and send the message 'ok' to the meter. Else it will also be sending the message 'ok' after each successful transaction.
10. The cost will be cut form the user balance which is situated in the user's dashboard. Users can easily see the deduction of the money in the dashboard.
11. The connection is closed.
12. After this, the loop will start again for a new set of data waiting to be received. Ends if no loop.

Figure 9.4 shows the flow chart of code events in the server side.

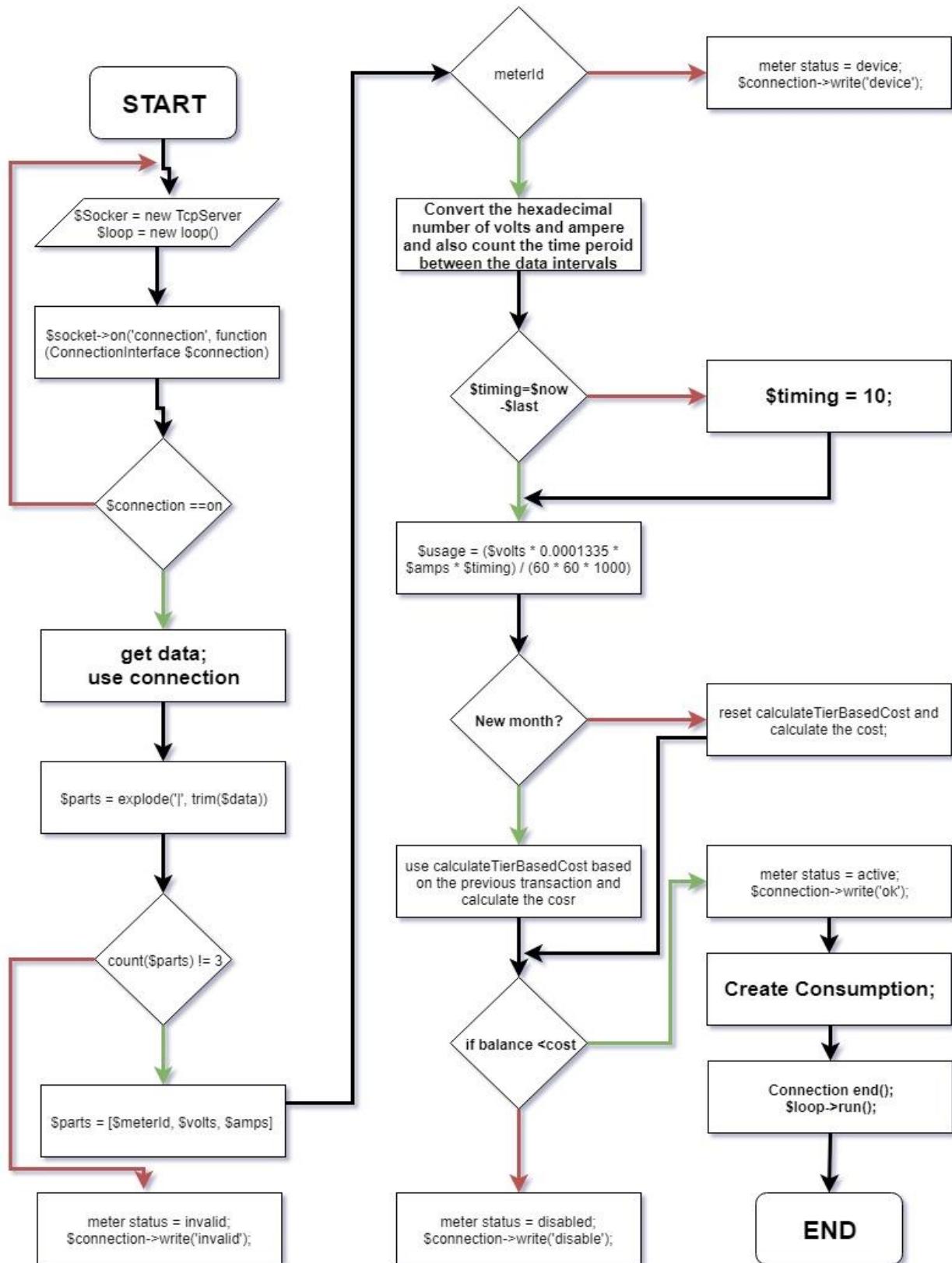


Figure 9. 4: Flow Chart of the main server side program

### 9.3 Results

The retrieved  $V_{rms}$  and  $I_{rms}$  register values and calibrated actual values are all presented in the following tables 9.1, 9.2, 9.3, 9.4, 9.5 and 9.6.

These are the stable data collected after the PCB was built. During the demonstration of the project, in front of the defense panel, these data were presented. However, these data were not verified by the respected supervisor before the demonstration.

<b>AVRMS data in hex</b>	<b>AVRMS data in decimal</b>	<b>Calibrated VRMS in decimal (V)</b>
1015bc	1054140	223.79
10f92a	1112362	236.15
10124d	1053261	223.61
102258	1057368	224.48
101849	1054793	223.93

Table 9. 1: VRMS Data from the AVRMS Register and the Calibrated Actual VRMS Values for 200W Load

<b>AVRMS data in hex</b>	<b>AVRMS data in decimal</b>	<b>Calibrated VRMS in decimal (V)</b>
1013c5	1053637	223.69
10f207	1110535	235.77
1019b6	1055158	224.01
102051	1056849	224.37
101fea	1056746	224.35

Table 9. 2: VRMS Data from the AVRMS Register and the Calibrated Actual VRMS Values for 400W Load

<b>AVRMS data in hex</b>	<b>AVRMS data in decimal</b>	<b>Calibrated VRMS in decimal (V)</b>
1020ad	1056941	224.39

101283	1053315	223.62
101d67	1056103	224.21
1019e1	1055201	224.02
101da7	1056167	224.22

Table 9. 3: VRMS Data from the AVRMS Register and the Calibrated Actual VRMS Values for 500W Load

AIRMS data in hex	AIRMS data in decimal	Calibrated IRMS in decimal (A)
2a94c	174412	0.8895
2a841	174145	0.8881
2a884	174212	0.8884
2a83b	174139	0.8881
2a8c2	174274	0.8887

Table 9. 4: IRMS Data from the AIRMS Register and the Calibrated Actual IRMS Values for 200W Load

AIRMS data in hex	AIRMS data in decimal	Calibrated IRMS in decimal (A)
553fd	349181	1.7808
55334	348980	1.7798
55499	349337	1.7816
55550	349520	1.7826
555ea	349674	1.7833

Table 9. 5: IRMS Data from the AIRMS Register and the Calibrated Actual IRMS Values for 400W Load

AIRMS data in hex	AIRMS data in decimal	Calibrated IRMS in decimal (A)
6b5f8	439800	2.2430
6b375	439157	2.2397
6b4ee	439534	2.2416
6b44c	439372	2.2408
6b596	439702	2.2425

Table 9. 6: IRMS Data from the AIRMS Register and the Calibrated Actual IRMS Values for 500W Load

# **Chapter 10**

## **Conclusion**

### **10.1 Shortcomings**

To find out the real power consumption, we have used  $V_{rms}$  and  $I_{rms}$  Register values (0x0D and 0x0A) for real power calculation, which does not provide us the most accurate real power consumption. For more accurate measurements, AWATTHR register (0x01) should have been more suitable for calculation. However, the calibration process for Watt-hour registers are more complex than that of rms registers, and due to insufficient time and complexity, we chose not to follow this path, although, calibration in this procedure would give more accurate results.

Also there have been no options for customization added to this project, as the Access Point SSID or password may change and there we might need some customizations. No such user friendly components have been added to this final design that might help the user customize, other than reprogramming the whole microcontroller chip.

Also, we should be aware of the fact that, there might be some circumstances when the meter might not be able to connect to a wifi access point, the internet service could be temporarily unavailable for some reason. In those cases, the microcontroller will be continuously reading data from the meter, but will not be able to post those information in the server. As a result, some consumption data might not be calculated.

Another issue with the system is that, after the balance is reduced to zero, the system cuts off the main power to the load, without any warning.

## 10.2 Solutions

To address the customization problem, a keypad should be added to the design. Using the keypad, the user should be able to connect to different access points, or give input the changed SSID or password to the microcontroller to establish connection. An Alphanumeric Keypad could be used for this purpose.



Figure 10. 1: Alphanumeric Keyboard [34]

About the second issue, we could use EEPROM (Electrically Erasable Programmable Read Only Memory). ATmega32 has an internal EEPROM which has a size of total 1024 bytes. It is organized as a separate data space where single bytes could be read and written. The EEPROM in ATmega32 has an endurance of 100000 write-erase cycles. It would be better to use an external EEPROM rather than the internal one in ATmega32, as when the write-erase cycles will come to an end, the

entire microcontroller might need to be changed. In case of the External EEPROM, we'd need to change only that after some time intervals.

When the ATmega32 might fail to connect with the wifi access point for some reason, it would get response “WIFI DISCONNECT\r” from the ESP. Receiving this the ATmega32 would transfer the data to the EEPROM by some communication protocol. After establishing connection again, the data written in the EEPROM would be transferred to the Server via ESP. The RTC data would provide the information when this data was recorded.

About the issue with the sudden power disconnection, the system could warn the user of a possible null balance, after the balance is reduced to a certain limit. When the balance has fully depleted to zero, the system could provide a grace period, for the users to refill their balance, for a limited amount of time, decided by the administrators.

### **10.3 Summary**

Throughout this paper, there were discussions on the implementations of IoT in recent times in various areas. Some areas where this implementation might come in handy, just like in the case of Electricity Meters. There had been thorough discussions on some metering IC's used nowadays and why we chose ADE7758 for our project. We studied its features, we learned about its communication protocol. We have utilized its communication properties to store valuable data from it to a microcontroller, which is in our case ATmega32. And towards the end of the paper, we discussed how we would send those information wirelessly to the server. We had an overview of the database prepared for our cause and studied its structures and data hierarchy. And also, we showed how those information can be viewed with solid authorization to maintain data

transparency between the users and the administrators. We also pointed out some lacking and flaws of this design and we also offered some relevant solutions. Last time, when we demonstrated our work to our respected supervisor, the Energy measurements were not properly calibrated and not giving expected readings. And without his approval to our solution to this problem, we proceeded with our thesis defense. If the project along with all these discussions, can reduce corruption and contribute to making our daily lives faster by bringing necessary electricity consumption data directly to our phones, then this project will have served its true purpose.

## References

- [1] "Basics of the SPI Communication Protocol" by Circuit Basics. Retrieved from:  
<http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/>
- [2] F. Castelli, "The power-frequency converter, by the induction watthour meter, with application to bimodal tariff," Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference [Cat. No. 00CH37066], Baltimore, MD, USA, 2000, pp. 164-168 vol.1.
- [3] Understanding about the SPI Communication Protocol in Embedded. Retrieved from:  
<https://www.elprocus.com/serial-peripheral-interface-spi-communication-protocol/>
- [4] Explain about 3 Basic Types of Energy Meters? Retrieved from:  
<https://www.watelectrical.com/introduction-on-energy-meter-different-types-of-energy-meters/>
- [5] ESP8266 Wi-Fi Module. Retrieved from  
<https://www.electronicwings.com/sensors-modules/esp8266-wifi-module>
- [6] ESP8266. Retrieved from: <https://en.wikipedia.org/wiki/ESP8266>
- [7] ESP-12E- Wi-Fi Module. (2018, October 17). Retrieved from:  
<https://components101.com/wireless/esp12e-pinout-datasheet>
- [8] Circuit Basics. BASICS OF THE I2C COMMUNICATION. Retrieved from:  
<http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>

- [9] Circuit Basics. **BASICS OF UART COMMUNICATION**. Retrieved from:  
<http://www.circuitbasics.com/basics-uart-communication/>
- [10] Margaret Rouse. (2018, July). Retrieved from:  
<https://internetofthingsagenda.techtarget.com/definition/smart-home-or-building>
- [11] Smart grid. Retrieved from: [https://en.wikipedia.org/wiki/Smart\\_grid](https://en.wikipedia.org/wiki/Smart_grid)
- [12] Internet of things. Retrieved from:  
[https://en.wikipedia.org/wiki/Internet\\_of\\_things](https://en.wikipedia.org/wiki/Internet_of_things)
- [13] Internet of Things. (2018, November 11). Retrieved from:  
<https://www.miltonpark.co.uk/blog/the-internet-of-things>
- [14] Electricity meter. Retrieved from: [https://en.wikipedia.org/wiki/Electricity\\_meter](https://en.wikipedia.org/wiki/Electricity_meter)
- [15] Super Star Energy Meter. Retrieved from: <https://www.ssgbd.com/energy-meter/>
- [16] Smart Energy International. Energy measurement ICs simplify meter design (2003, March 31). Retrieved from:  
<https://www.smart-energy.com/industry-sectors/components/energy-measurement-ics-simplify-meter-design/>.
- [17] Analog Devices. ADE7751 Datasheet: Energy Metering IC with On-Chip Fault Detection. Retrieved from:  
<https://www.analog.com/media/en/technical-documentation/data-sheets/ade7751.pdf>.

- [18] Analog Devices. ADE7753 Datasheet: Single-Phase Multifunction Metering IC with di/dt Sensor Interface. Retrieved from:  
<https://www.analog.com/media/en/technical-documentation/data-sheets/ADE7753.pdf>.
- [19] Analog Devices. ADE7754 Datasheet: Polyphase Multifunction Energy Metering IC with Serial Port. Retrieved from:  
<https://www.analog.com/media/en/technical-documentation/data-sheets/ade7754.pdf>.
- [20] Analog Devices. ADE7758 Datasheet: Poly Phase Multifunction Energy Metering IC with Per Phase Information. Retrieved from:  
<https://www.analog.com/media/en/technical-documentation/data-sheets/ADE7758.pdf>.
- [21] “Basics of the SPI Communication Protocol” by Circuit Basics. Retrieved from:  
<http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/>
- [22] Y. Tambi and M. Prasad, “The SPI of the AVR” on *Atmel AVR, Microcontrollers*, November 2013. Retrieved from:  
<http://maxembedded.com/2013/11/the-spi-of-the-avr/>
- [23] Nugent, Stephen. “Precision SPI Switch Configuration Increases Channel Density.” *Analog Dialogue*, May 2017.  
 Usach, Miguel. AN-1248 Application Note: *SPI Interface*. Analog Devices, Inc., September 2015. Retrieved from:  
<https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html>
- [24] Microchip Technology Inc. ATmega32 datasheet: Retrieved from:  
<http://ww1.microchip.com/downloads/en/devicedoc/doc2503.pdf>
- [25] Analog Devices. ADE7758 datasheet. Retrieved from:  
<https://www.analog.com/media/en/technical-documentation/data-sheets/ADE7758.pdf>

- [26] Based on original library by Peter Fluery, Tim Sharpe, Nicholas Zambetti. Retrieved from:  
<http://homepage.hispeed.ch/peterfleury/avr-software.html>
- [27] Espressif Systems. “ESP8266 AT Instructions Set”: Retrieved from:  
[https://www.espressif.com/sites/default/files/documentation/4a-esp8266\\_at\\_instruction\\_set\\_en.pdf#%5B%7B%22num%22%3A452%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C70.86614%2C717%2C0%5D](https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf#%5B%7B%22num%22%3A452%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C70.86614%2C717%2C0%5D)
- [28] Oregon State University Lecture on Two Wire Interface. Retrieved from:  
<http://web.engr.oregonstate.edu/~traylor/ece473/lectures/twi.pdf>
- [29] Clock-Calender Using DS3231+GLCD+Atmega32 by By SHARANYADAS in *Microcontrollers*: Retrieved from:  
<https://www.instructables.com/id/Clock-Calender-Using-DS3231GLCDAtmega32/>
- [30] Christensson, P. (2014, April 16). Server Definition. Retrieved 2019, Apr 24, from  
<https://techterms.com>
- [31] Buyya, R. Object Oriented Programming with Java: Essentials and Applications: Chapter 13 Socket Based Programming. Retrieved from:  
<http://www.buyya.com/java/Chapter13.pdf>
- [32] Wikipedia. Database. Retrieved from <https://en.wikipedia.org/wiki/Database>
- [33] Bartleby Writing. Essay on Databases. Retrieved from:  
<https://www.bartleby.com/essay/Databases-P3JFVYWZTJ>
- [34] Vidnis. Alphanumeric Keypad. Retrieved from:

<http://www.vidnis.com/2013/03/4x4-keypad-alphanumeric-mobile-cell.html>

- [35] Odd Wires. Retrieved from:  
<https://www.oddwires.com/ams1117-5v-module-adjustable-and-fixed-voltage-regulator/>
- [36] Hobby Components. Retrieved from:  
<https://hobbycomponents.com/rtc/699-high-accuracy-ds3231-rtc-eeprom>
- [37] L. Salman et al., "Energy efficient IoT-based smart home," 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, 2016, pp. 526-529.
- [38] M. H. Yaghmaee and H. Hejazi, "Design and Implementation of an Internet of Things Based Smart Energy Metering," 2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE), Oshawa, ON, 2018, pp. 191-194.

## Appendix A

### Specifications of ADE7758 [20]

AVDD=DVDD=5V $\pm$ 5%, AGND=DGND=0V, on-chip reference, CLKIN=10 MHz XTAL, TMIN to TMAX=−40°C to +85°C [20].

Parameter <sup>1,2</sup>	Specification	Unit	Test Conditions/Comments
<b>ACCURACY</b>			
Active Energy Measurement Error (per Phase)	0.1	% typ	Over a dynamic range of 1000 to 1
Phase Error Between Channels			
PF = 0.8 Capacitive	$\pm 0.05$	$^{\circ}\text{max}$	Line frequency = 45 Hz to 65 Hz, HPF on
PF = 0.5 Inductive	$\pm 0.05$	$^{\circ}\text{max}$	Phase lead 37°
AC Power Supply Rejection			Phase lag 60°
Output Frequency Variation	0.01	% typ	AVDD = DVDD = 5 V + 175 mV rms/120 Hz
DC Power Supply Rejection			V1P = V2P = V3P = 100 mV rms
Output Frequency Variation	0.01	% typ	AVDD = DVDD = 5 V $\pm$ 250 mV dc
Active Energy Measurement Bandwidth	14	kHz	V1P = V2P = V3P = 100 mV rms
IRMS Measurement Error	0.5	% typ	Over a dynamic range of 500:1
IRMS Measurement Bandwidth	14	kHz	
VRMS Measurement Error	0.5	% typ	Over a dynamic range of 20:1
VRMS Measurement Bandwidth	260	Hz	
<b>ANALOG INPUTS</b>			
Maximum Signal Levels	$\pm 500$	mV max	See the Analog Inputs section
Input Impedance (DC)	380	k $\Omega$ min	Differential input
ADC Offset Error <sup>3</sup>	$\pm 30$	mV max	Uncalibrated error, see the Terminology section
Gain Error <sup>3</sup>	$\pm 6$	% typ	External 2.5 V reference
<b>WAVEFORM SAMPLING</b>			
Current Channels			Sampling CLKIN/128, 10 MHz/128 = 78.1 kSPS
Signal-to-Noise Plus Distortion Bandwidth (−3 dB)	62 14	dB typ kHz	See the Current Channel ADC section
Voltage Channels			
Signal-to-Noise Plus Distortion Bandwidth (−3 dB)	62 260	dB typ Hz	See the Voltage Channel ADC section
<b>REFERENCE INPUT</b>			
REF <sub>IN/OUT</sub> Input Voltage Range	2.6 2.2 10	V max V min pF max	2.4 V + 8% 2.4 V − 8%
Input Capacitance			
<b>ON-CHIP REFERENCE</b>			
Reference Error	$\pm 200$	mV max	Nominal 2.4 V at REF <sub>IN/OUT</sub> pin
Current Source	6	$\mu\text{A}$ max	
Output Impedance	4	k $\Omega$ min	
Temperature Coefficient	30	ppm/ $^{\circ}\text{C}$ typ	
<b>CLKIN</b>			
Input Clock Frequency	15 5	MHz max MHz min	All specifications CLKIN of 10 MHz
<b>LOGIC INPUTS</b>			
DIN, SCLK, CLKIN, and $\overline{\text{CS}}$			
Input High Voltage, V <sub>INH</sub>	2.4	V min	DVDD = 5 V $\pm$ 5%
Input Low Voltage, V <sub>INL</sub>	0.8	V max	DVDD = 5 V $\pm$ 5%
Input Current, I <sub>IN</sub>	$\pm 3$	$\mu\text{A}$ max	Typical 10 nA, V <sub>IN</sub> = 0 V to DVDD
Input Capacitance, C <sub>IN</sub>	10	pF max	

Parameter <sup>1,2</sup>	Specification	Unit	Test Conditions/Comments
<b>LOGIC OUTPUTS</b>			
IRQ, DOUT, and CLKOUT			DVDD = 5 V $\pm$ 5%
Output High Voltage, V <sub>OH</sub>	4	V min	I <sub>RO</sub> is open-drain, 10 k $\Omega$ pull-up resistor
Output Low Voltage, V <sub>OL</sub>	0.4	V max	I <sub>SOURCE</sub> = 5 mA I <sub>SINK</sub> = 1 mA
APCF and VARCF			
Output High Voltage, V <sub>OH</sub>	4	V min	I <sub>SOURCE</sub> = 8 mA
Output Low Voltage, V <sub>OL</sub>	1	V max	I <sub>SINK</sub> = 5 mA
<b>POWER SUPPLY</b>			
AVDD	4.75 5.25	V min V max	For specified performance 5 V − 5% 5 V + 5%
DVDD	4.75 5.25	V min V max	5 V − 5% 5 V + 5%
A <sub>LOO</sub>	8	mA max	Typically 5 mA
D <sub>LOO</sub>	13	mA max	Typically 9 mA

## Appendix B

### ADE7758 Pin Function Descriptions [20]

Pin No.	Mnemonic	Description
1	APCF	Active Power Calibration Frequency (APCF) Logic Output. It provides active power information. This output is used for operational and calibration purposes. The full-scale output frequency can be scaled by writing to the APCFNUM and APCFDEN registers (see the Active Power Frequency Output section).
2	DGND	This provides the ground reference for the digital circuitry in the ADE7758, that is, the multiplier, filters, and digital-to-frequency converter. Because the digital return currents in the ADE7758 are small, it is acceptable to connect this pin to the analog ground plane of the whole system. However, high bus capacitance on the DOUT pin can result in noisy digital current that could affect performance.
3	DVDD	Digital Power Supply. This pin provides the supply voltage for the digital circuitry in the ADE7758. The supply voltage should be maintained at $5\text{ V} \pm 5\%$ for specified operation. This pin should be decoupled to DGND with a $10\text{ }\mu\text{F}$ capacitor in parallel with a ceramic $100\text{ nF}$ capacitor.
4	AVDD	Analog Power Supply. This pin provides the supply voltage for the analog circuitry in the ADE7758. The supply should be maintained at $5\text{ V} \pm 5\%$ for specified operation. Every effort should be made to minimize power supply ripple and noise at this pin by the use of proper decoupling. The Typical Performance Characteristics show the power supply rejection performance. This pin should be decoupled to AGND with a $10\text{ }\mu\text{F}$ capacitor in parallel with a ceramic $100\text{ nF}$ capacitor.
5, 6, 7, 8, 9, 10	IAP, IAN, IBP, IBN, ICP, ICN	Analog Inputs for Current Channel. This channel is used with the current transducer and is referenced in this document as the current channel. These inputs are fully differential voltage inputs with maximum differential input signal levels of $\pm 0.5\text{ V}$ , $\pm 0.25\text{ V}$ , and $\pm 0.125\text{ V}$ , depending on the gain selections of the internal PGA (see the Analog Inputs section). All inputs have internal ESD protection circuitry. In addition, an overvoltage of $\pm 6\text{ V}$ can be sustained on these inputs without risk of permanent damage.
11	AGND	This pin provides the ground reference for the analog circuitry in the ADE7758, that is, ADCs, temperature sensor, and reference. This pin should be tied to the analog ground plane or the quietest ground reference in the system. This quiet ground reference should be used for all analog circuitry, for example, antialiasing filters, current, and voltage transducers. To keep ground noise around the ADE7758 to a minimum, the quiet ground plane should be connected to the digital ground plane at only one point. It is acceptable to place the entire device on the analog ground plane.
12	REF <sup>IN/OUT</sup>	This pin provides access to the on-chip voltage reference. The on-chip reference has a nominal value of $2.4\text{ V} \pm 8\%$ and a typical temperature coefficient of $30\text{ ppm/}^{\circ}\text{C}$ . An external reference source can also be connected at this pin. In either case, this pin should be decoupled to AGND with a $1\text{ }\mu\text{F}$ ceramic capacitor.
13, 14, 15, 16	VN, VCP, VBP, VAP	Analog Inputs for the Voltage Channel. This channel is used with the voltage transducer and is referenced as the voltage channels in this document. These inputs are single-ended voltage inputs with the maximum signal level of $\pm 0.5\text{ V}$ with respect to VN for specified operation. These inputs are voltage inputs with maximum input signal levels of $\pm 0.5\text{ V}$ , $\pm 0.25\text{ V}$ , and $\pm 0.125\text{ V}$ , depending on the gain selections of the internal PGA (see the Analog Inputs section). All inputs have internal ESD protection circuitry, and in addition, an overvoltage of $\pm 6\text{ V}$ can be sustained on these inputs without risk of permanent damage.
17	VARCF	Reactive Power Calibration Frequency Logic Output. It gives reactive power or apparent power information depending on the setting of the VACF bit of the WAVMODE register. This output is used for operational and calibration purposes. The full-scale output frequency can be scaled by writing to the VARCFNUM and VARCFDEN registers (see the Reactive Power Frequency Output section).
18	IRQ	Interrupt Request Output. This is an active low open-drain logic output. Maskable interrupts include: an active energy register at half level, an apparent energy register at half level, and waveform sampling up to $26\text{ kSPS}$ (see the Interrupts section).
19	CLKIN	Master Clock for ADCs and Digital Signal Processing. An external clock can be provided at this logic input. Alternatively, a parallel resonant AT crystal can be connected across CLKIN and CLKOUT to provide a clock source for the ADE7758. The clock frequency for specified operation is $10\text{ MHz}$ . Ceramic load capacitors of a few tens of picofarad should be used with the gate oscillator circuit. Refer to the crystal manufacturer's data sheet for the load capacitance requirements
20	CLKOUT	A crystal can be connected across this pin and CLKIN as previously described to provide a clock source for the ADE7758. The CLKOUT pin can drive one CMOS load when either an external clock is supplied at CLKIN or a crystal is being used.
21	CS	Chip Select. Part of the 4-wire serial interface. This active low logic input allows the ADE7758 to share the serial bus with several other devices (see the Serial Interface section).
22	DIN	Data Input for the Serial Interface. Data is shifted in at this pin on the falling edge of SCLK (see the Serial Interface section).
23	SCLK	Serial Clock Input for the Synchronous Serial Interface. All serial data transfers are synchronized to this clock (see the Serial Interface section). The SCLK has a Schmidt-trigger input for use with a clock source that has a slow edge transition time, for example, opto-isolator outputs.
24	DOUT	Data Output for the Serial Interface. Data is shifted out at this pin on the rising edge of SCLK. This logic output is normally in a high impedance state, unless it is driving data onto the serial data bus (see the Serial Interface section).

## Appendix C

### ADE7758 Register Chart [20]

Address [A6:A0]	Name	R/W <sup>1</sup>	Length	Type <sup>2</sup>	Default Value	Description
0x00	Reserved	-				Reserved.
0x01	AWATTHR	R	16	S	0	Watt-Hour Accumulation Register for Phase A. Active power is accumulated over time in this read-only register. The AWATTHR register can hold a maximum of 0.52 seconds of active energy information with full-scale analog inputs before it overflows (see the Active Energy Calculation section). Bit 0 and Bit 1 of the COMPMODE register determine how the active energy is processed from the six analog inputs.
0x02	BWATTHR	R	16	S	0	Watt-Hour Accumulation Register for Phase B.
0x03	CWATTHR	R	16	S	0	Watt-Hour Accumulation Register for Phase C.
0x04	AVARHR	R	16	S	0	VAR-Hour Accumulation Register for Phase A. Reactive power is accumulated over time in this read-only register. The AVARHR register can hold a maximum of 0.52 seconds of reactive energy information with full-scale analog inputs before it overflows (see the Reactive Energy Calculation section). Bit 0 and Bit 1 of the COMPMODE register determine how the reactive energy is processed from the six analog inputs.
0x05	BVARHR	R	16	S	0	VAR-Hour Accumulation Register for Phase B.
0x06	CVARHR	R	16	S	0	VAR-Hour Accumulation Register for Phase C.
0x07	AVAHRS	R	16	S	0	VA-Hour Accumulation Register for Phase A. Apparent power is accumulated over time in this read-only register. The AVAHRS register can hold a maximum of 1.15 seconds of apparent energy information with full-scale analog inputs before it overflows (see the Apparent Energy Calculation section). Bit 0 and Bit 1 of the COMPMODE register determine how the apparent energy is processed from the six analog inputs.
0x08	BVAHRS	R	16	S	0	VA-Hour Accumulation Register for Phase B.
0x09	CVAHRS	R	16	S	0	VA-Hour Accumulation Register for Phase C.
0x0A	AIRMS	R	24	S	0	Phase A Current Channel RMS Register. The register contains the rms component of the Phase A input of the current channel. The source is selected by data bits in the mode register.
0x0B	BIRMS	R	24	S	0	Phase B Current Channel RMS Register.
0x0C	CIRMS	R	24	S	0	Phase C Current Channel RMS Register.
0x0D	AVRMS	R	24	S	0	Phase A Voltage Channel RMS Register.

Address [A6:A0]	Name	R/W <sup>1</sup>	Length	Type <sup>2</sup>	Default Value	Description
0x0E	BVRMS	R	24	S	0	Phase B Voltage Channel RMS Register.
0x0F	CVRMS	R	24	S	0	Phase C Voltage Channel RMS Register.
0x10	FREQ	R	12	U	0	Frequency of the Line Input Estimated by the Zero-Crossing Processing. It can also display the period of the line input. Bit 7 of the LCYCMODE register determines if the reading is frequency or period. Default is frequency. Data Bit 0 and Bit 1 of the MMODE register determine the voltage channel used for the frequency or period calculation.
0x11	TEMP	R	8	S	0	Temperature Register. This register contains the result of the latest temperature conversion. Refer to the Temperature Measurement section for details on how to interpret the content of this register.
0x12	WFORM	R	24	S	0	Waveform Register. This register contains the digitized waveform of one of the six analog inputs or the digitized power waveform. The source is selected by Data Bit 0 to Bit 4 in the WAVMODE register.
0x13	OPMODE	R/W	8	U	4	Operational Mode Register. This register defines the general configuration of the ADE7758.
0x14	MMODE	R/W	8	U	0xFC	Measurement Mode Register. This register defines the channel used for period and peak detection measurements.
0x15	WAVMODE	R/W	8	U	0	Waveform Mode Register. This register defines the channel and sampling frequency used in the waveform sampling mode.
0x16	COMPMode	R/W	8	U	0x1C	Computation Mode Register. This register configures the formula applied for the energy and line active energy measurements.
0x17	LCYCMODE	R/W	8	U	0x78	Line Cycle Mode Register. This register configures the line cycle accumulation mode for WATT-HR, VAR-HR, and VA-Hr.
0x18	Mask	R/W	24	U	0	IRQ Mask Register. It determines if an interrupt event generates an active-low output at the IRQ pin.
0x19	Status	R	24	U	0	IRQ Status Register. This register contains information regarding the source of the ADE7758 interrupts.
0x1A	RSTATUS	R	24	U	0	IRQ Reset Status Register. Same as the STATUS register, except that its contents are reset to 0 (all flags cleared) after a read operation.
0x1B	ZXTOUT	R/W	16	U	0xFFFF	Zero-Cross Timeout Register. If no zero crossing is detected within the time period specified by this register, the interrupt request line (IRQ) goes active low for the corresponding line voltage. The maximum timeout period is 2.3 seconds.
0x1C	LINECYC	R/W	16	U	0xFFFF	Line Cycle Register. The content of this register sets the number of half-line cycles that the active, reactive, and apparent energies are accumulated for in the line accumulation mode.
0x1D	SAGCYC	R/W	8	U	0xFF	SAG Line Cycle Register. This register specifies the number of consecutive half-line cycles where voltage channel input may fall below a threshold level. This register is common to the three line voltage SAG detection. The detection threshold is specified by the SAGLVL register.
0x1E	SAGLVL	R/W	8	U	0	SAG Voltage Level. This register specifies the detection threshold for the SAG event. This register is common to all three phases' line voltage SAG detections. See the description of the SAGCYC register for details.
0x1F	VPINTLVL	R/W	8	U	0xFF	Voltage Peak Level Interrupt Threshold Register. This register sets the level of the voltage peak detection. Bit 5 to Bit 7 of the MMODE register determine which phases are to be monitored. If the selected voltage phase exceeds this level, the PKV flag in the IRQ status register is set.
0x20	IPINTLVL	R/W	8	U	0xFF	Current Peak Level Interrupt Threshold Register. This register sets the level of the current peak detection. Bit 5 to Bit 7 of the MMODE register determine which phases are to be monitored. If the selected current phase exceeds this level, the PKI flag in the IRQ status register is set.
0x21	VPEAK	R	8	U	0	Voltage Peak Register. This register contains the value of the peak voltage waveform that has occurred within a fixed number of half-line cycles. The number of half-line cycles is set by the LINECYC register.

Address [A6:A0]	Name	R/W <sup>1</sup>	Length	Type <sup>2</sup>	Default Value	Description
0x22	IPEAK	R	8	U	0	Current Peak Register. This register holds the value of the peak current waveform that has occurred within a fixed number of half-line cycles. The number of half-line cycles is set by the LINECYC register.
0x23	Gain	R/W	8	U	0	PGA Gain Register. This register is used to adjust the gain selection for the PGA in the current and voltage channels.
0x24	AVRMSGAIN	R/W	12	S	0	Phase A VRMS Gain Register. The range of the voltage rms calculation can be adjusted by writing to this register. It has an adjustment range of ±50% with a resolution of 0.0244%/LSB.
0x25	BVRMSGAIN	R/W	12	S	0	Phase B VRMS Gain Register.
0x26	CVRMSGAIN	R/W	12	S	0	Phase C VRMS Gain Register.
0x27	AIGAIN	R/W	12	S	0	Phase A Current Gain Register. This register is not recommended to be used and it should be kept at 0, its default value.
0x28	BIGAIN	R/W	12	S	0	Phase B Current Gain Register. This register is not recommended to be used and it should be kept at 0, its default value.
0x29	CIGAIN	R/W	12	S	0	Phase C Current Gain Register. This register is not recommended to be used and it should be kept at 0, its default value.
0x2A	AWG	R/W	12	S	0	Phase A Watt Gain Register. The range of the watt calculation can be adjusted by writing to this register. It has an adjustment range of ±50% with a resolution of 0.0244%/LSB.
0x2B	BWG	R/W	12	S	0	Phase B Watt Gain Register.
0x2C	CWG	R/W	12	S	0	Phase C Watt Gain Register.
0x2D	AVARG	R/W	12	S	0	Phase A VAR Gain Register. The range of the VAR calculation can be adjusted by writing to this register. It has an adjustment range of ±50% with a resolution of 0.0244%/LSB.
0x2E	BVARG	R/W	12	S	0	Phase B VAR Gain Register.
0x2F	CVARG	R/W	12	S	0	Phase C VAR Gain Register.
0x30	AVAG	R/W	12	S	0	Phase A VA Gain Register. The range of the VA calculation can be adjusted by writing to this register. It has an adjustment range of ±50% with a resolution of 0.0244%/LSB.
0x31	BVAG	R/W	12	S	0	Phase B VA Gain Register.
0x32	CVAG	R/W	12	S	0	Phase C VA Gain Register.
0x33	AVRMSOS	R/W	12	S	0	Phase A Voltage RMS Offset Correction Register.
0x34	BVRMSOS	R/W	12	S	0	Phase B Voltage RMS Offset Correction Register.
0x35	CVRMSOS	R/W	12	S	0	Phase C Voltage RMS Offset Correction Register.
0x36	AIRMSOS	R/W	12	S	0	Phase A Current RMS Offset Correction Register.
0x37	BIRMSOS	R/W	12	S	0	Phase B Current RMS Offset Correction Register.
0x38	CIRMSOS	R/W	12	S	0	Phase C Current RMS Offset Correction Register.
0x39	AWATTOS	R/W	12	S	0	Phase A Watt Offset Calibration Register.
0x3A	BWATTOS	R/W	12	S	0	Phase B Watt Offset Calibration Register.
0x3B	CWATTOS	R/W	12	S	0	Phase C Watt Offset Calibration Register.
0x3C	AVAROS	R/W	12	S	0	Phase A VAR Offset Calibration Register.
0x3D	BVAROS	R/W	12	S	0	Phase B VAR Offset Calibration Register.
0x3E	CVAROS	R/W	12	S	0	Phase C VAR Offset Calibration Register.
0x3F	APHCAL	R/W	7	S	0	Phase A Phase Calibration Register. The phase relationship between the current and voltage channel can be adjusted by writing to this signed 7-bit register (see the Phase Compensation section).
0x40	BPHCAL	R/W	7	S	0	Phase B Phase Calibration Register.
0x41	CPHCAL	R/W	7	S	0	Phase C Phase Calibration Register.
0x42	WDIV	R/W	8	U	0	Active Energy Register Divider.
0x43	VARDIV	R/W	8	U	0	Reactive Energy Register Divider.
0x44	VADIV	R/W	8	U	0	Apparent Energy Register Divider.

Address [A6:A0]	Name	R/W <sup>1</sup>	Length	Type <sup>2</sup>	Default Value	Description
0x45	APCFNUM	R/W	16	U	0	Active Power CF Scaling Numerator Register. The content of this register is used in the numerator of the APCF output scaling calculation. Bits [15:13] indicate reverse polarity active power measurement for Phase A, Phase B, and Phase C in order; that is, Bit 15 is Phase A, Bit 14 is Phase B, and so on.
0x46	APCFDEN	R/W	12	U	0x3F	Active Power CF Scaling Denominator Register. The content of this register is used in the denominator of the APCF output scaling.
0x47	VARCFNUM	R/W	16	U	0	Reactive Power CF Scaling Numerator Register. The content of this register is used in the numerator of the VARCF output scaling. Bits [15:13] indicate reverse polarity reactive power measurement for Phase A, Phase B, and Phase C in order; that is, Bit 15 is Phase A, Bit 14 is Phase B, and so on.
0x48	VARCFDEN	R/W	12	U	0x3F	Reactive Power CF Scaling Denominator Register. The content of this register is used in the denominator of the VARCF output scaling.
0x49 to 0x7D	Reserved	-	-	-	-	Reserved.
0x7E	CHKSUM	R	8	U	-	Checksum Register. The content of this register represents the sum of all the ones in the last register read from the SPI port.
0x7F	Version	R	8	U	-	Version of the Die.

## Appendix D

### Measurement Mode Register (0x14) Chart [20]

By writing to the MMODE register the configuration of the period and peak value measurements are done by the ADE7758 [20].

Bit Location	Bit Mnemonic	Default Value	Description																	
0 to 1	FREQSEL	0	These bits are used to select the source of the measurement of the voltage line frequency.																	
			<table border="1"> <thead> <tr> <th>FREQSEL1</th> <th>FREQSEL0</th> <th>Source</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Phase A</td> </tr> <tr> <td>0</td> <td>1</td> <td>Phase B</td> </tr> <tr> <td>1</td> <td>0</td> <td>Phase C</td> </tr> <tr> <td>1</td> <td>1</td> <td>Reserved</td> </tr> </tbody> </table>			FREQSEL1	FREQSEL0	Source	0	0	Phase A	0	1	Phase B	1	0	Phase C	1	1	Reserved
FREQSEL1	FREQSEL0	Source																		
0	0	Phase A																		
0	1	Phase B																		
1	0	Phase C																		
1	1	Reserved																		
2 to 4	PEAKSEL	7	These bits select the phases used for the voltage and current peak registers. Setting Bit 2 switches the IPEAK and VPEAK registers to hold the absolute values of the largest current and voltage waveform (over a fixed number of half-line cycles) from Phase A. The number of half-line cycles is determined by the content of the LINECYC register. At the end of the LINECYC number of half-line cycles, the content of the registers is replaced with the new peak values. Similarly, setting Bit 3 turns on the peak detection for Phase B, and Bit 4 for Phase C. Note that if more than one bit is set, the VPEAK and IPEAK registers can hold values from two different phases, that is, the voltage and current peak are independently processed																	
5 to 7	PKIRQSEL	7	These bits select the phases used for the peak interrupt detection. Setting Bit 5 switches on the monitoring of the absolute current and voltage waveform to Phase A. Similarly, setting Bit 6 turns on the waveform detection for Phase B, and Bit 7 for Phase C. Note that more than one bit can be set for detection on multiple phases. If the absolute values of the voltage or current waveform samples in the selected phases exceeds the preset level specified in the VPINTLVL or IPINTLVL registers the corresponding bit(s) in the STATUS registers are set																	

## Appendix E

### Waveform Mode Register (0x15) Chart [20]

Waveform sampling mode of the ADE7758 is defined by writing to the WAVMODE register [20].

Bit Location	Bit Mnemonic	Default Value	Description			
0 to 1	PHSEL	0	These bits are used to select the phase of the waveform sample.			
			<b>PHSEL[1:0]</b>		<b>Source</b>	
			0 0	0	Phase A	
			0 1	1	Phase B	
			1 0	0	Phase C	
			1 1	1	Reserved	
2 to 4	WAVSEL	0	These bits are used to select the type of waveform.			
			<b>WAVSEL[2:0]</b>		<b>Source</b>	
			0 0 0	0	Current	
			0 0 1	1	Voltage	
			0 1 0	0	Active Power Multiplier Output	
			0 1 1	1	Reactive Power Multiplier Output	
			1 0 0	0	VA Multiplier Output	
			Others-			Reserved
5 to 6	DTRT	0	These bits are used to select the data rate.			
			<b>DTRT[1:0]</b>		<b>Update Rate</b>	
			0 0	0	26.04 kSPS (CLKIN/3/128)	
			0 1	1	13.02 kSPS (CLKIN/3/256)	
			1 0	0	6.51 kSPS (CLKIN/3/512)	
			1 1	1	3.25 kSPS (CLKIN/3/1024)	
7	VACF	0	Setting this bit to Logic 1 switches the VARCF output pin to an output frequency that is proportional to the total apparent power (VA). In the default state, Logic 0, the VARCF pin outputs a frequency proportional to the total reactive power (VAR).			

## Appendix F

### SPI Initial Codes

```

//Initialization
void spi_init_master (void)
{
    DDRB = (1<<5)|(1<<7) | (1<<4);      //Set MOSI, SCK and SS as Output
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0); //Enable SPI, Set as Master, Prescaler: f/16
    PORTB |= (1<<4);
}

```

```

    //Function to send and receive data
    unsigned char spi_tranceiver (unsigned char data)
    {
        SPDR = data;                      //Load data into buffer
        while(!(SPSR)&(1<<SPIF));
        SPSR |= (1<<7);                //Make SPIF high again
        return(SPDR);
    }
}

```

## Appendix G

### ADE7758\_WRITE method

```

void ADE7758_WRITE(unsigned char address,unsigned char datasend)
{
    address |= (1 << (7)); //write mode
    PORTB &= ~(1 << 4); //cs low
    for(int x = 8;x>0; x--)
    {
        PORTB |= (1 << 7);
        if((address >> (x-1) & 0x01)){
            PORTB |= (1 << 5); //SI high
        }
        _delay_us(1);
        PORTB &= ~(1 << 7);
        PORTB &= ~(1 << 5); //SI low
        _delay_us(1);
    }
    PORTB &= ~(1 << 5); //SI low
    _delay_us(20);
    for(int x = 8;x>0; x--)
    {
        PORTB |= (1 << 7);
        if((datasend >> (x-1) & 0x01)){
            PORTB |= (1 << 5); //SI high
        }
        _delay_us(1);
        PORTB &= ~(1 << 7);
        PORTB &= ~(1 << 5); //SI low
        _delay_us(1);
    }
    PORTB |= (1 << 4); //cs high
}

```

## Appendix H

### ADE7758\_READ\_8\_bit

```
char ADE7758_READ_8_bit(unsigned char address){
    unsigned char dataReceive = 0;
    //unsigned char _MSB =0,_LSB =0,_KBS =0;

    PORTB &= ~(1 << 4); //cs low
    for(int x = 8;x>0; x--)
    {
        PORTB |= (1 << 7);
        if((address >> (x-1) & 0x01))
        {
            PORTB |= (1 << 5); //SI high
        }
        else{
            PORTB &= ~(1 << 5);
        }
        _delay_us(1);
        PORTB &= ~(1 << 7);
        PORTB &= ~(1 << 5);
        _delay_us(1);
    }
    PORTB &= ~(1 << 5); //SI low
    _delay_us(20);
    for(int x = 8;x>0; x--)
    {
        PORTB |= (1 << 7);
        _delay_us(1);
        PORTB &= ~(1 << 7);
        _delay_us(1);
        if(PINB & (1<<6)){dataReceive |= (1 << (x-1)); }
        else{dataReceive &= ~(1 << (x-1));}
    }
    PORTB |= (1 << 4); //cs high
    return dataReceive;
}
```

# Appendix I

## UART Settings Code

```
#if defined(AT90_UART)
    /* set baud rate */
    UBRR = (uint8_t) baudrate;

    /* enable UART receiver and transmitter */
    UART0_CONTROL = (1<<RXEN)|(1<<TXEN);

#elif defined (ATMEGA_USART)
    /* Set baud rate */
    if (baudrate & 0x8000) {
        UART0_STATUS = (1<<U2X); //Enable 2x speed
        baudrate &= ~0x8000;
    }
    UBRRH = (uint8_t) (baudrate>>8);
    UBRL = (uint8_t) baudrate;

    /* Enable USART receiver and transmitter */
    UART0_CONTROL = (1<<RXEN)|(1<<TXEN);

    /* Set frame format: asynchronous, 8data, no parity, 1stop bit */
#endif URSEL
    UCSRC = (1<<URSEL)|(3<<UCSZ0);
#else
    UCSRC = (3<<UCSZ0);
#endif
```

## Appendix J

### ATE- AT Commands Echoing [27]

Execute Command	ATE
Response	OK
Parameters	<ul style="list-style-type: none"><li>• ATE0: Switches echo off.</li><li>• ATE1: Switches echo on.</li></ul>
Note	This command ATE is used to trigger command echo. It means that entered commands can be echoed back to the sender when ATE command is used. Two parameters are possible. The command returns OK in normal cases and ERROR when a parameter other than 0 or 1 was specified.

## Appendix K

### AT+RST- Restarts the Module [27]

Execute Command	AT+RST
Response	OK
Parameters	-

## Appendix L

### AT+CWMODE\_DEF – Sets the Default Wi-Fi Mode [27]

Commands	Test Command: AT+CWMODE_DEF=?	Query Command: AT+CWMODE_DEF? Function: to query the current Wi-Fi mode of ESP8266.	Set Command: AT+CWMODE_DEF=<mode> Function: to set the current Wi-Fi mode of ESP8266.
Response	+CWMODE_DEF:<mode> OK	+CWMODE_DEF:<mode> OK	OK
Parameters	<mode>: ▶ 1: Station mode ▶ 2: SoftAP mode ▶ 3: SoftAP+Station mode		
Note	The configuration changes will be saved in the system parameter area in the flash.		
Example	AT+CWMODE_DEF=3		

## Appendix M

### AT+CIPMUX – Enable or Disable Multiple Connections [27]

Commands	Query Command: AT+CIPMUX?	Set Command: AT+CIPMUX=<mode> Function: to set the connection type.
Response	+CIPMUX:<mode> OK	OK
Parameters	<mode>: ▶ 0: single connection ▶ 1: multiple connections	
Notes	<ul style="list-style-type: none"> <li>The default mode is single connection mode.</li> <li>Multiple connections can only be set when transparent transmission is disabled (AT+CIPMODE=0).</li> <li>This mode can only be changed after all connections are disconnected.</li> <li>If the TCP server is running, it must be deleted (AT+CIPSERVER=0) before the single connection mode is activated.</li> </ul>	
Example	AT+CIPMUX=1	

## Appendix N

### AT+CWJAP\_DEF – Connects to an Access Point [27]

Commands	<p>Query Command: AT+CWJAP_DEF?</p> <p>Function: to query the AP to which the ESP8266 Station is already connected.</p>	<p>Set Command: AT+CWJAP_DEF=&lt;ssid&gt;,&lt;pwd&gt;,[&lt;bssid&gt;] [,&lt;pci_en&gt;]</p> <p>Function: to set the AP to which the ESP8266 Station needs to be connected.</p>
Response	<p>+CWJAP_DEF:&lt;ssid&gt;,&lt;bssid&gt;,&lt;channel&gt;,&lt;rss&gt;</p> <p>OK</p>	<p>OK</p> <p>or</p> <p>+CWJAP_DEF:&lt;error code&gt;</p> <p>FAIL</p>
Parameters	<p>&lt;ssid&gt;: a string parameter showing the SSID of the target AP.</p>	<ul style="list-style-type: none"> <li>• &lt;ssid&gt;: the SSID of the target AP, MAX: 32 bytes.</li> <li>• &lt;pwd&gt;: password, MAX: 64-byte ASCII.</li> <li>• [&lt;bssid&gt;]: optional parameter, the target AP's MAC address, used when multiple APs have the same SSID.</li> <li>• [&lt;pci_en&gt;]: optional parameter, disable the connection to WEP or OPEN AP, and can be used for PCI authentication.</li> <li>• &lt;error code&gt;: (for reference only) <ul style="list-style-type: none"> <li>▶ 1: connection timeout.</li> <li>▶ 2: wrong password.</li> <li>▶ 3: cannot find the target AP.</li> <li>▶ 4: connection failed.</li> </ul> </li> </ul> <p>This command requires Station mode to be enabled. Escape character syntax is needed if SSID or password contains any special characters, such as , or " or \.</p>
Note	The configuration changes will be saved in the system parameter area in the flash.	
Examples	<p>AT+CWJAP_DEF="abc","0123456789"</p> <p>For example, if the target AP's SSID is "ab\,c" and the password is "0123456789"\\", the command is as follows:</p> <p>AT+CWJAP_DEF="ab\\\",c","0123456789\"\\\"</p> <p>If multiple APs have the same SSID as "abc", the target AP can be found by BSSID:</p> <p>AT+CWJAP_DEF="abc","0123456789","ca:d7:19:d8:a6:44"</p>	

## Appendix O

### AT+CIPSTART – Establishing TCP Connection [27]

Set Command	Single TCP connection (AT+CIPMUX=0): AT+CIPSTART=<type>,<remote IP>,<remote port>[,<TCP keep alive>]	Multiple TCP Connections (AT+CIPMUX=1): AT+CIPSTART=<link ID>,<type>,<remote IP>,<remote port>[,<TCP keep alive>]
Response	OK or ERROR  If the TCP connection is already established, the response is: ALREADY CONNECTED	
Parameters	<ul style="list-style-type: none"> <li>• &lt;link ID&gt;: ID of network connection (0~4), used for multiple connections.</li> <li>• &lt;type&gt;: string parameter indicating the connection type: "TCP", "UDP" or "SSL".</li> <li>• &lt;remote IP&gt;: string parameter indicating the remote IP address.</li> <li>• &lt;remote port&gt;: the remote port number.</li> <li>• [&lt;TCP keep alive&gt;]: detection time interval when TCP is kept alive; this function is disabled by default. <ul style="list-style-type: none"> <li>▶ 0: disable TCP keep-alive.</li> <li>▶ 1 ~ 7200: detection time interval; unit: second (s).</li> </ul> </li> </ul>	
Examples	AT+CIPSTART="TCP","iot.espressif.cn",8000 AT+CIPSTART="TCP","192.168.101.110",1000	

## Appendix P

### AT+CIPCLOSE – Closes the TCP Connection [27]

Commands	Set Command (used in multiple connections): AT+CIPCLOSE=<link ID> Function: close the TCP/UDP Connection.	Execute Command (used in multiple connections): AT+CIPCLOSE
Response	OK	
Parameters	<link ID>: ID of the connection to be closed. When ID is 5, all connections will be closed. (In server mode, the ID 5 has no effect.)	-

## Appendix Q

### AT+CIPSEND – Sends Data [27]

Commands	<p>Set Command:</p> <ol style="list-style-type: none"> <li>1. Single connection: (+CIPMUX=0) <code>AT+CIPSEND=&lt;length&gt;</code></li> <li>2. Multiple connections: (+CIPMUX=1) <code>AT+CIPSEND=&lt;link ID&gt;,&lt;length&gt;</code></li> <li>3. Remote IP and ports can be set in UDP transmission: <code>AT+CIPSEND=[&lt;link ID&gt;,&lt;length&gt; [,&lt;remote IP&gt;,&lt;remote port&gt;]]</code></li> </ol> <p>Function: to configure the data length in normal transmission mode.</p>	<p>Execute Command: <code>AT+CIPSEND</code></p> <p>Function: to start sending data in transparent transmission mode.</p>
Response	<p>Send data of designated length. Wrap return &gt; after the Set Command. Begin receiving serial data. When data length defined by &lt;length&gt; is met, the transmission of data starts.</p> <p>If the connection cannot be established or gets disrupted during data transmission, the system returns: <code>ERROR</code></p> <p>If data is transmitted successfully, the system returns: <code>SEND OK</code></p> <p>If it failed, the system returns: <code>SEND FAIL</code></p>	<p>Wrap return &gt; after executing this command. Enter transparent transmission, with a 20-ms interval between each packet, and a maximum of 2048 bytes per packet.</p> <p>When a single packet containing +++ is received, ESP8266 returns to normal command mode. Please wait for at least one second before sending the next AT command.</p> <p>This command can only be used in transparent transmission mode which requires single connection.</p> <p>For UDP transparent transmission, the value of &lt;UDP mode&gt; has to be 0 when using AT+CIPSTART.</p>
Parameters	<ul style="list-style-type: none"> <li>• &lt;link ID&gt;: ID of the connection (0~4), for multiple connections.</li> <li>• &lt;length&gt;: data length, MAX: 2048 bytes.</li> <li>• [&lt;remote IP&gt;]: remote IP can be set in UDP transmission.</li> <li>• [&lt;remote port&gt;]: remote port can be set in UDP transmission.</li> </ul>	-

## Appendix R

### Wi-fi Send AT Command Function Codes

```
[-] static inline void wifi_send_command_no_block(const char* command) {
    uart_puts("AT");
    uart_puts(command);
    uart_puts("\r\n");
    _delay_ms(10);
}

[-] static inline void wifi_send_command(const char* command) {
    wifi_send_command(command);
    wifi_wait_for_status();
}
```

## Appendix S

### Wi-fi Echo Disabling and Reset Functions

```
[-] static inline void wifi_disable_echo(void) {
    wifi_send_command_no_block("E0");
    wifi_wait_for_status();
}

[-] static inline void wifi_reset() {
    wifi_send_command_no_block("+RST=1");
    //wifi_wait_for_status();
}
```

## Appendix T

### Wi-fi Connect to an AP Function

```
static inline void wifi_connect(const char* id, const char* passwd) {  
    wifi_send_command_no_block("+CWMODE_DEF=1");  
    wifi_wait_for_status();  
  
    wifi_send_command_no_block("+CIPMUX=0");  
    wifi_wait_for_status();  
  
    wifi_clear_command_buffer();  
    sprintf(wifi_command_buffer, "+CWJAP_DEF=\"%s\", \"%s\"", id, passwd);  
    wifi_send_command_no_block(wifi_command_buffer);  
    wifi_wait_for_status();  
}
```

## Appendix U

### Wi-fi Link Open Code

```
static inline void wifi_link_open(const char* protocol, const char* url, const int port) {  
    //wifi_clear_command_buffer();  
    sprintf(wifi_command_buffer, "+CIPSTART=\"%s\", \"%s\", %d", protocol, url, port);  
    wifi_send_command_no_block(wifi_command_buffer);  
    //wifi_send_command_no_block("+CIPSTART=\"TCP\", \"api.openweathermap.org\", 80");  
    wifi_wait_for_status();  
}
```

## Appendix V

### Wi-fi Send Data Code

```
static inline void wifi_send_no_block(const char* contents) {
    const int len = strlen(contents);
    wifi_clear_command_buffer();
    sprintf(wifi_command_buffer, "+CIPSEND=%d", len);
    wifi_send_command_no_block(wifi_command_buffer);
    wifi_wait_for_status();
    _delay_ms(500);
    uart_puts(contents);
}

static inline void wifi_send(const char* contents) {
    wifi_send_no_block(contents);
    wifi_wait_for_data();
}
```

## Appendix W

### Esp Response String Compare Code

```
wifi_event_handler(WIFI_EVENT_ANY, wifi_command_buffer, command_buffer_iter);

if(command_buffer_iter <= 1) {

} else if(strcmp(wifi_command_buffer, "OK\r") == 0) {
    wifi_event_handler(WIFI_EVENT_OK, wifi_command_buffer, command_buffer_iter);
    return 1;
} else if(strcmp(wifi_command_buffer, "ERROR\r") == 0) {
    wifi_event_handler(WIFI_EVENT_ERROR, wifi_command_buffer, command_buffer_iter);
    return 0;
} else if(strcmp(wifi_command_buffer, "> \r") == 0) {

} else if(strcmp(wifi_command_buffer, "\r") == 0) {

} else if(strcmp(wifi_command_buffer, "WIFI GOT IP\r") == 0) {
    wifi_event_handler(WIFI_EVENT_WIFI_GOT_IP, wifi_command_buffer, command_buffer_iter);
} else if(strcmp(wifi_command_buffer, "CONNECT\r") == 0) {
    wifi_event_handler(WIFI_EVENT_CONNECTED, wifi_command_buffer, command_buffer_iter);
} else if(strcmp(wifi_command_buffer, "SEND OK\r") == 0) {
    wifi_event_handler(WIFI_EVENT_SEND_OK, wifi_command_buffer, command_buffer_iter);
} else if(strcmp(wifi_command_buffer, "CLOSED\r") == 0) {
    wifi_event_handler(WIFI_EVENT_CLOSED, wifi_command_buffer, command_buffer_iter);
    if(wait_mode == WAIT_FOR_DATA) return 0;
} else if(strcmp(wifi_command_buffer, "WIFI CONNECTED\r") == 0) {
    wifi_event_handler(WIFI_EVENT_WIFI_CONNECTED, wifi_command_buffer, command_buffer_iter);
} else if(strcmp(wifi_command_buffer, "WIFI DISCONNECT\r") == 0) {
    wifi_event_handler(WIFI_EVENT_WIFI_DISCONNECTED, wifi_command_buffer, command_buffer_iter);
} else if(strcmp(wifi_command_buffer, "busy p... \r") == 0) {
    _delay_ms(500);
} else if(strncmp(wifi_command_buffer, "Recv ", 5) == 0) {
```

## Appendix X

### Wi-fi Event Handler Code

```
void wifi_event_handler(int event, const char* input, const int len) {
    switch (event) {
        case WIFI_EVENT_ANY: {
            return;
        } break;

        case WIFI_EVENT_DATA: {
            lcd_clrscr();
            lcd_puts(input);

            if (strcmp(input, "ok", 2) == 0) {
                PORTC |= (0<<7);
            } else if (strcmp(input, "disable", 7) == 0) {
                PORTC |= (1<<7);
            }

            _delay_ms(10000);
        } break;

        case WIFI_EVENT_WIFI_CONNECTED: {
            lcd_clrscr();
            lcd_puts("Connected!");

            _delay_ms(200);
        } break;

        case WIFI_EVENT_WIFI_DISCONNECTED: {
            lcd_clrscr();
            lcd_puts("Disconnected!");

            _delay_ms(200);
        } break;
    }
}
```

## Appendix Y

### Twi Core Library

```
#include "twi.h"

void twi_init(void)
{
    TWCR=0x00;
    TWBR=0x0F;//see how to configure clock freq of twi.here the freq for 16mhz xtal is nearabout 347khz
    TWCR =(1<<TWEN); //enable TWEN bit
}

void twi_start(void)
{
    TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN); //TWINT=0 by writting one to it,TWSTA bit=1,TWEN bit=1 to transmit start condition when the bus is free.
    while((TWCR & (1<<TWINT))==0); //stay here and poll until TWINT becomes 1 at the end of transmit.
}

void twi_stop(void)
{
    TWCR=(1<<TWINT)|(1<<TWSTO)|(1<<TWEN); //TWINT=0 by writting one to it,TWSTO bit=1,TWEN bit=1 to transmit stOP condition.
}

void twi_send(uint8_t data)
{
    TWDR=data;//send the data to TWDR resistor
    TWCR=(1<<TWINT)|(1<<TWEN); //TWINT=0 by writting one to it,TWEN bit=1 to enable twi module
    while((TWCR & (1<<TWINT))==0); //stay here and poll until TWINT becomes 1 at the end of transmit.
}

uint8_t twi_receive(uint8_t ack_val)
{
    TWCR=(1<<TWINT)|(ack_val<<TWEA); //if we want to receive more than one byte,we will send 1 as ack_val to send an acknowledge.At the time of last byte,we will send 0 as ack_val to send NACK.
    while((TWCR & (1<<TWINT))==0); //stay here and poll until TWINT becomes 1 at the end of transmit.
    return TWDR;
}
```

## Appendix Z

### Connection Establishing Code

```
$loop = Factory::create();
$socket = new TcpServer($this->uri(), $loop);

$socket->on('connection', function (ConnectionInterface $connection) {
    $this->line("<info>New connection:</info> <{$connection->getRemoteAddress()}>");

    $connection->on('data', function ($data) use ($connection) {
        $parts = explode('|', trim($data));
```

## Appendix AA

### Dummy Transaction Ids

```
private $dummyTxnIds = [
    '5id981nf2p' => 5500,
    '5ii0969t92' => 4200,
    '5kb1muhv8f' => 2000,
    '5l98v65s9i' => 3000,
    '5kgh7ydy25' => 1200,
];
```

These are some dummy transactions which are restored in the system. You need to generate or type manually to add a transaction, consisting of a value to the system

## Appendix BB

### Cost Calculation Code

```
protected function calculateTierBasedCost($usage, $tillNow = null)
{
    if (! is_null($tillNow)) {
        return $this->calculateTierBasedCost($usage + $tillNow) - $this->calculateTierBasedCost($tillNow);
    }

    $total = 0;

    if ($usage > 600) {
        $total += ($usage - 600) * 10.7;
        $usage = 600;
    }
}
```

At first, it gets the usage and checks it in the tier-based cost. If the cost is above 600, per unit cost will be multiplied with 10.7. After that it goes to the other tiers for checking.

## Appendix CC

### Code of the Proper Usage of the Month

```
class ConsumptionController extends Controller
{
    public function index(Request $request, Meter $meter)
    {
        if ($meter->user_id != $this->visitor->id) {
            abort(403);
        }

        if ($request->has('start', 'end')) {
            $consumptions = $meter->consumptions()->fromStartToEnd(
                $request->get('start'), $request->get('end')
            )->get();
        } else {
            $consumptions = $meter->consumptions()->forTheLastThirtyDays()->get();
        }

        return ConsumptionResourceCollection::make($consumptions);
    }
}
```

## Appendix DD

### Account Controller

```
public function showAccountInfoPage()
{
    return view('public.account.info');
}

public function updateAccountInfo(AccountUpdateRequest $request)
{
    $input = $this->getInputForSave($request);

    $this->visitor->update($input['user']);

    if ($address = $this->visitor->address) {
        $address->update($input['address']);
    } else {
        $this->visitor->address()->create($input['address']);
    }

    return $this->updated($this->visitor, 'Successfully updated your account information.');
}

protected function getInputForSave(Request $request)
{
    $input = $request->only([
        'name', 'email', 'phone',
        'password', 'address_line_1', 'address_line_2',
        'postal_code', 'locality', 'administrative_area',
        'country',
    ]);

    if (empty($input['password'])) {
        unset ($input['password']);
    } else {
        $input['password'] = Hash::make($input['password']);
    }

    $input['phone'] = phone($input['phone'], $input['country'])->formatInternational();
}
```

## Appendix EE

### SPSR Register

Bit	7	6	5	4	3	2	1	0	SPSR
SPIF	WCOL	-	-	-	-	-	-	SPI2X	
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure: SPSR Register [24]

#### SPIF:

When the device is configured as a Master Device, this bit is set in two different circumstances.

When SS pin is configured as an input pin and has been driven low by an external device. Or for our case, when a data transfer is complete.

#### WCOL:

This bit is set when we load data on SPDR before the Data Transfer is even completed.

#### SPI2X:

Setting this bit to one, doubles the data transfer speed. Works in Master mode only.

## Appendix FF

### SPCR Register

Bit	7	6	5	4	3	2	1	0	SPCR
Read/Write	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	
Initial Value	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0 0 0 0 0 0 0 0 0

Figure: SPCR Register [24]

#### SPIE:

If this bit is set and Global Interrupt is enabled, then when SPIF in the SPSR register is set, an Interrupt Service Routine will initiate. SPIE bit was not used in our case.

#### SPE:

Setting this bit simply enables the SPI Protocol for communication.

#### DORD:

Data Order bit decides the order of data bits to be sent or received. Setting this bit to 1, will make the LSB of the data bit to be transmitted / received first. Setting this bit to 0, will make the MSB of the data to be transmitted / received first. By default, it remains 0, and for our case this bit is 0 as well to ensure MSB is transmitted first.

#### MSTR:

Setting this bit to 1, configures the device as the Master Device, and 0 as the Slave device. For our case, ATmega32 is the Master Device and Ade7758 functions as the Slave device.

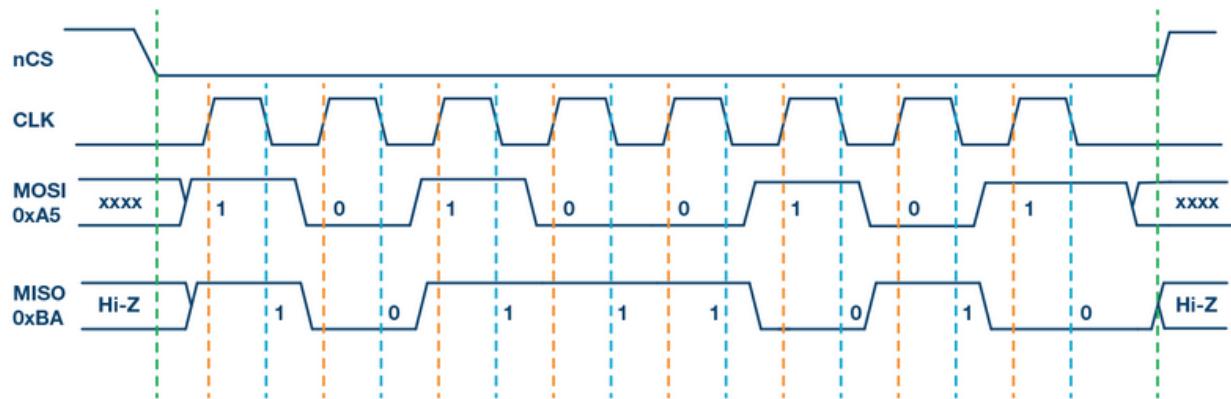
### CPOL and CPHA:

For the SPI Communication to work properly, both the master and the slave device must agree on CPOL (Clock Polarity) and CPHA (Clock Phase). The CPOL and CPHA determines the SPI Modes for communication.

<b>CPOL</b>	<b>CPHA</b>	<b>Data Read and Change Time</b>	<b>SPI Mode</b>
0	0	Read on rising edge, changed on a falling edge	0
0	1	Read on falling edge, changed on a rising edge	1
1	0	Read on falling edge, changed on a rising edge	2
1	1	Read on rising edge, changed on a falling edge	3

*Table: SPI Communication Modes [24]*

These figures explain clearly the SPI Communication modes:



*Figure: SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge. [23]*

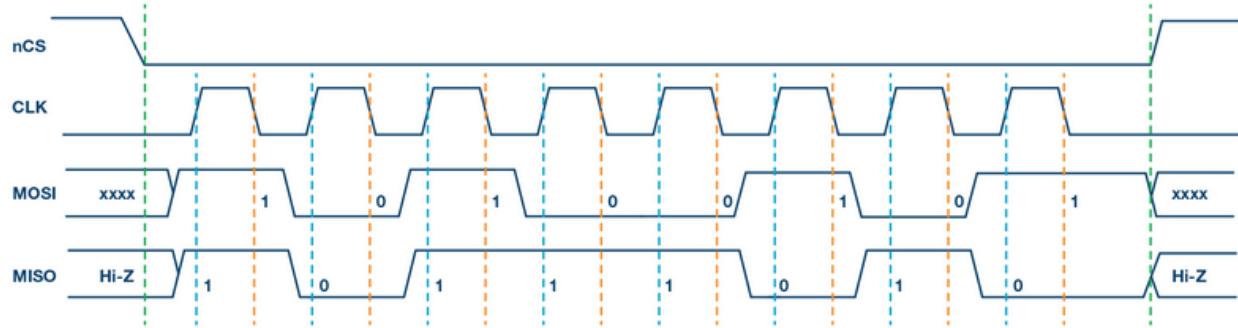


Figure: SPI Mode 1,  $CPOL = 0$ ,  $CPHA = 1$ :  $CLK$  idle state = low, data sampled on the falling edge and shifted on the rising edge. [23]

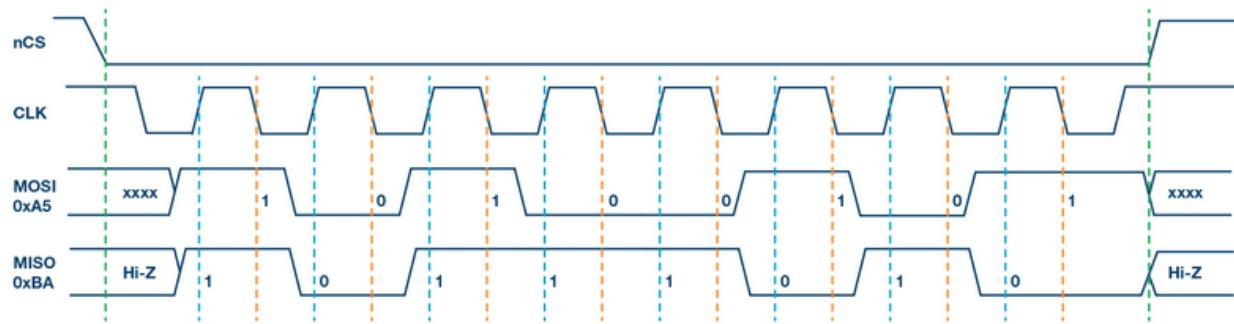


Figure: SPI Mode 2,  $CPOL = 1$ ,  $CPHA = 1$ :  $CLK$  idle state = high, data sampled on the falling edge and shifted on the rising edge. [23]

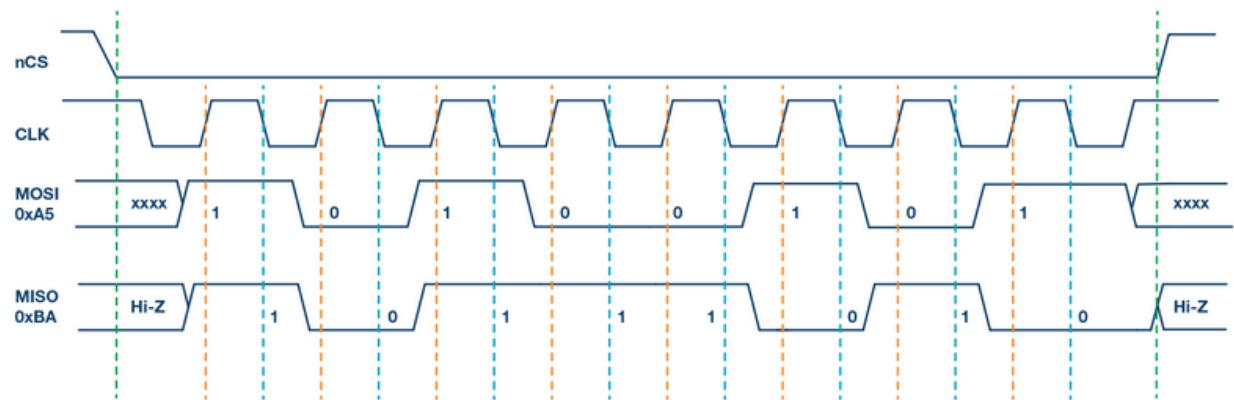


Figure: SPI Mode 3,  $CPOL = 1$ ,  $CPHA = 0$ :  $CLK$  idle state = high, data sampled on the rising edge and shifted on the falling edge. [23]

The Timing Diagram of ADE7758 suggests that the SPI Communication mode is 1 for that particular IC. We'll explain the ADE7758 Timing diagram in the later Chapters.

### **SPR0 and SPR1:**

These bits along with the SPI2X in the SPSR register sets the clock rate. They are used to choose the oscillator frequency divider where  $f_{osc}$  is the internal clock frequency. In our case, we are using an external oscillator crystal of 16MHz connected with the ATmega32. So  $f_{osc}$  is 16MHz.

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

*Table: Frequency Divider [24]*

For our use, we set SPR0 to 1, so that the Clock Frequency becomes  $f_{osc} / 16 = 1$  MHz.

## Appendix GG

### SPDR Register

Bit	7	6	5	4	3	2	1	0	SPDR
Read/Write	R/W								
Initial Value	X	X	X	X	X	X	X	X	Undefined

*Figure: SPDR Register [24]*

The SPI Data Register is a Read / Write Register. To write data to Shift Register, data must be written in the SPDR Register. As for reading data from the Shift Register, data must be read from SPDR Register as well. Writing before the previous data being sent, or reading before the previous data being received will result to data collision.

## Appendix HH

### UDR Register

Bit	7	6	5	4	3	2	1	0	UDR (Read)
Read/Write	R/W	UDR (Write)							
Initial Value	0	0	0	0	0	0	0	0	

*Figure: UDR Register [24]*

## Appendix II

### UBRR Register

Bit	15	14	13	12	11	10	9	8		
	<b>URSEL</b>	-	-	-	<b>UBRR[11:8]</b>					
	<b>UBRR[7:0]</b>								<b>UBRRH</b>	
	7	6	5	4	3	2	1	0	<b>UBRRL</b>	
ReadWrite	R/W	R	R	R	R/W	R/W	R/W	R/W		
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
Initial Value	0	0	0	0	0	0	0	0		
	0	0	0	0	0	0	0	0		

Figure: UBRR Register [24]

## Appendix JJ

### UCSRA Register

Bit	7	6	5	4	3	2	1	0	UCSRA
Read/Write	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	
Initial Value	0	0	1	0	0	0	0	0	

Figure: UCSRA Register [24]

**RXC:** This bit is set when there is a Received data in the UDR, waiting to be read, and is cleared when the Receive Buffer becomes empty once again.

**TXC:** This bit is set when the entire data in UDR is transmitted out and there are no longer any data in the transfer buffer register.

**UDRE:** Bit is set when UDR is Empty.

**FE:** Bit is set when there is a Frame Error, that is when the first stop bit of the next character in the receive buffer is 0.

**DOR:** Data Overrun. This bit is set when the Receive buffer is full and yet a new Start bit is detected.

**PE:** This bit is set when a Parity Error has occurred when receiving data bits, and this bit is activated by setting 1 in the UPM.

**U2X:** This bit needs to be set, if the data transmission speed needs to be doubled (2x).

**MPCM:** Is high when Multi Processor Communication mode needs to be enabled.

## Appendix KK

### UCSRB Register

Bit	7	6	5	4	3	2	1	0	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure: UCSR B Register [24]

**RXCIE:** Receive Complete Interrupt Enable. Setting this bit to 1 enables the Interrupt Flag on RXC. This bit was set to 1 for the program.

**TXCIE:** Transfer Complete Interrupt Enable. Setting this bit to 1 enables the Interrupt Flag on TXC.

**UDRIE:** Setting this bit to 1 enables the interrupt flag when the UDR is empty, in other words when UDRE is 1.

**RXEN and TXEN:** These bits enables receive and transfer function and this bits need to be set as 1, for any UART Communication.

**UCSZ2:** Details about this bit will be discussed in the next section.

**RXB8 and TXB8:** These bits are the 9<sup>th</sup> data bits to be received / transmitted when working in 9 data bit mode.

## Appendix LL

### UCSRC Register

Bit	7	6	5	4	3	2	1	0	UCSRC
URSEL	R/W	UCSRC							
UMSEL	1	0	0	0	0	1	1	0	
UPM1									
UPM0									
USBS									
UCSZ1									
UCSZ0									
UCPOL									

Figure: UCSRC Register [24]

**URSEL:** Setting this bit to 1, will select this Register to operate as UCSRC.

**UMSEL:** When this bit is 0, the communication mode is Asynchronous, 1 for Synchronous Communication.

**UPM:** UPM1 and UPM0 bits sets the Parity Error Checker.

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Table: UPM Settings [24]

For our case, the Parity Checker was Disabled.

**USBS:** This bit selects the number of stop bits for the transmission data. This bit is ‘0’ when stop bits are 1 and ‘1’ when there are 2 stop bits.

**UCSZ:** UCSZ1 and UCSZ0 along with UCSZ2 from the UCSRB sets the number of data bits.

<b>UCSZ2</b>	<b>UCSZ1</b>	<b>UCSZ0</b>	<b>Character Size</b>
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

*Table: UCSZ Settings [24]*

**UCPOL:** This bit decides the clock polarity.

<b>UCPOL</b>	<b>Transmitted Data Changed (Output of TxD Pin)</b>	<b>Received Data Sampled (Input on RxD Pin)</b>
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

*Table: UCPOL Settings [24]*

## Appendix MM

### TWI Registers:

#### TWBR:

Bit	7	6	5	4	3	2	1	0	TWBR
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Figure: TWBR Register [24]

#### TWCR:

Bit	7	6	5	4	3	2	1	0	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure: TWCR Register [24]

#### TWSR:

Bit	7	6	5	4	3	2	1	0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Figure: TWSR Register [24]

### **TWDR and TWAR:**

Bit	7	6	5	4	3	2	1	0	
	<b>TWD7</b>	<b>TWD6</b>	<b>TWD5</b>	<b>TWD4</b>	<b>TWD3</b>	<b>TWD2</b>	<b>TWD1</b>	<b>TWD0</b>	<b>TWDR</b>
Read/Write	R/W								
Initial Value	1	1	1	1	1	1	1	1	

*Figure: TWDR register [24]*

Bit	7	6	5	4	3	2	1	0	
	<b>TWA6</b>	<b>TWA5</b>	<b>TWA4</b>	<b>TWA3</b>	<b>TWA2</b>	<b>TWA1</b>	<b>TWA0</b>	<b>TWGCE</b>	<b>TWAR</b>
Read/Write	R/W								
Initial Value	1	1	1	1	1	1	1	0	

*Figure: TWAR register [24]*

## Appendix NN:

```
void ds3231_set(uint8_t hr,uint8_t min,uint8_t sec,uint8_t ampm,uint8_t yr,uint8_t mnth,uint8_t dt,uint8_t day)
{
    hr &=0b00011111;//clear first three bits.
    hr |=0b01000000;//make D7=0,D6=1 for am/pm view,D5=0 for AM/1 for PM.
    hr |=(ampm<<5);

    twi_start();
    twi_send(0xD0);//initial address of ds3231+0 to write next byte
    twi_send(0x00);//select location 0 to update
    twi_send(sec);//update second
    twi_send(min);//write location of control resistor
    twi_send(hr);//write 0 to the content of the resistor
    twi_send(day);//update day of the week
    twi_send(dt);//update date
    twi_send(mnth);//update month
    twi_send(yr);//update year
    twi_stop();
}

void ds3231_get(uint8_t *h,uint8_t *m,uint8_t *s,uint8_t *yr,uint8_t *mnth,uint8_t *dt,uint8_t *day)
{
    twi_start();
    twi_send(0xD0);
    twi_send(0x00);

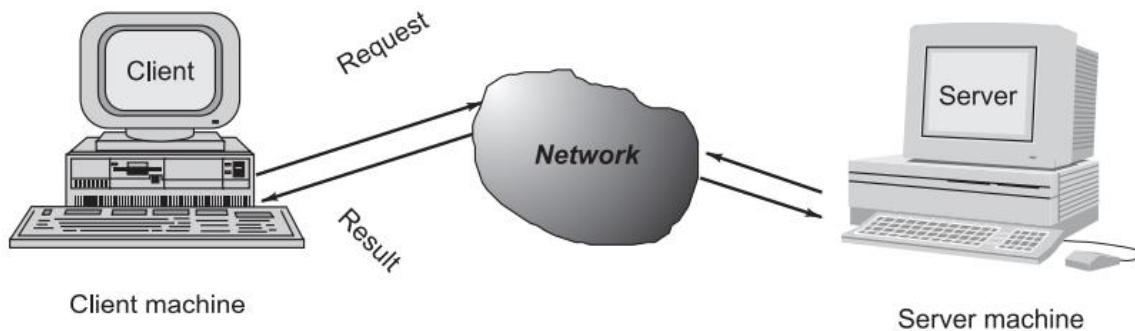
    twi_start();
    twi_send(0xD1);//ds3231 address+1 to read
    *s=twi_receive(1);//1 to send ACK
    *m=twi_receive(1);//1 to send ACK
    *h=twi_receive(1);//0 to send NACK
    *day=twi_receive(1);//1 to send ACK,read day
    *dt=twi_receive(1);//1 to send ACK,read date
    *mnth=twi_receive(1);//1 to send ACK,read month
    *yr=twi_receive(0);//0 to send NACK,read year

    twi_stop();
}
```

## Appendix OO

### Connection using TCP Client Socket

TCP (Transmission Control Protocol) is one of the main protocols in TCP/IP networks. Using TCP, we are creating an endpoint with which the device communicate with the server. It enables two hosts to establish a connection and exchange streams of raw data shown in the fig 3.2. The delivery of data is guaranteed by TCP and the packets will also be delivered in the same order in which they were sent. The server on which our project is built is a HTTP web server and it is on top of TCP, meaning that every HTTP request uses TCP as its transfer protocol making it a subset of TCP. It occurs on port 80 of a web server.



Businesses are continuously searching for new and innovative ways and means to offer their services via the Internet to take advantage of the opportunities presented by the Internet. This created enormous demand for software designers and engineers with the ability to create new internet-enabled applications or transfer existing/legacy applications to the internet platform. The key elements for the development of Internet-enabled applications are a good understanding of the

issues involved in implementing distributed applications and sound knowledge of the basic models of network programming. The project also uses a socket programming which helps us to send and receive data during the transmission. Network-based systems consist of a server, client and a media for communication. A running computer which makes a request for services is called a client machine whereas a running computer which accepts requested services from one or more clients is called a server machine. The communication can be done through a network which can be wired or wireless. In our case, we are using a wireless communication. Generally, programs running on client machines make requests to a server running machine. This whole process involves networking services which is provided by the transport layer, being a part of the Internet software stack, often called TCP/IP. Socket programming are widely used for these protocols. TCP is a connection-oriented protocol that provides a reliable flow of data between two computers as mentioned before. HTTP, FTP, Telnet are some services which use this TCP. [31]

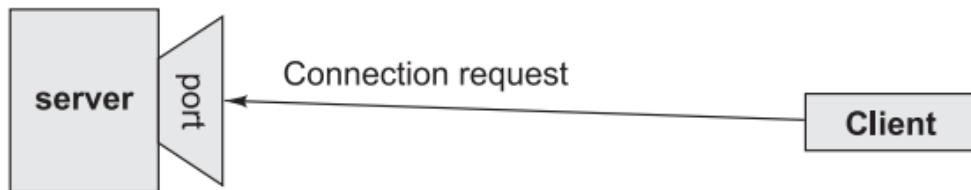
## **Host Identification and Service Ports**

The computer that are connected to the internet are identified by a unique 4-byte IP address. It is written in dotted quad format where each byte is an unsigned value between 0 and 255. Domain names are registered and names are converted into IP addresses. Generally, there is only one Internet address for each computer. Computers, however, often need to communicate and provide more than one type of service or speak at a time to multiple hosts/computers. For example, all running simultaneously can be multiple ftp sessions, web connections, and chat programs. A port concept, a logical access point, represented by a 16-bit integer number, is used to distinguish these services. That means that a port number identifies each service offered by a computer uniquely. Each internet packet contains both the host address of the destination and the port number on the host to which the message/request must be sent. The host computer displays the packets it receives

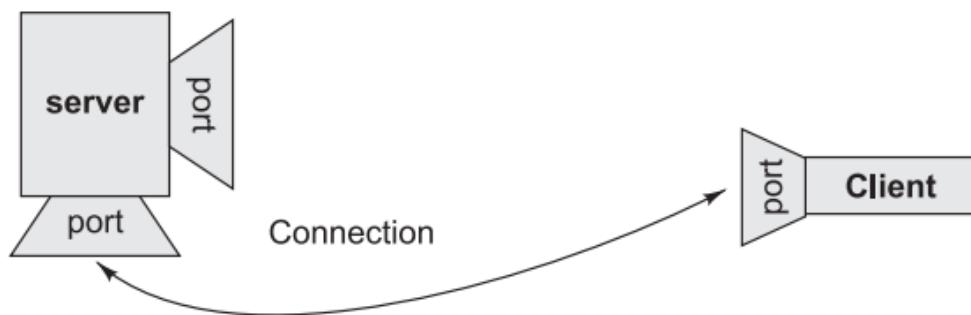
within the packets by looking at the specified port numbers within the packets. That is, when a letter is sent via post/snail mail and port number as the name of a specified person to whom the letter must be delivered afterwards, IP address can be considered as a house address. In our project, we also establishing a connection with host through a local server where the server is accepting requests from the ATmega32 and sending an acknowledgement to it afterwards. At first, we are starting the server from the command window and also starting another TCP server on 8001 port to establish connection.

## **Socket and Socket-based Communication**

Sockets provide an interface on the transport layer for programming networks. Sockets network communication is very similar to I/O file communication. In fact, the handle of sockets is treated as the handle of files. Also applicable to socket-based I/O is the streams used in file I/O operation. Communication based on sockets is independent of the programming language used to implement it. That means a C language written socket program can communicate to a non-C socket program written program. A server runs on a particular computer and has a socket linked to a particular port. The server is listening to a client's socket to request a connection (Fig. 3.2.2 a). The server accepts the connection if all goes well (Fig. 3.2.2 b). The server receives a new socket bound to a different port upon acceptance. In order to continue listening to the original socket for connection requests while serving the connected client, it needs a new socket (i.e. a different port number).



[a]: a client making a connection request to the server



[b]: session established with temporary ports used for two way communication.

*Figure: Establishment of path for two-way communication between a client and a server [31]*