2012

Student: Cornel Verster - 15673251

Study Leader: Thinus Booysen

[ONLINE INTERFACE FOR A PREPAID METER]

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch

Acknowledgements

To my Lord and Saviour Jesus Christ, for His love, mercy and acceptance throughout my life and to for the ability to complete this project.

To my parents for their love and support.

To Thinus Booysen for his motivation and support.

We would like to thank MTN and Trinity Telecommunications, who enabled and supported this work with their technical and financial assistance.

Declaration

I, the undersigned, hereby declare that the work contained in this report is my own original work unless indicated otherwise.

Signature .....

Date.....

Summaries

The remote monitoring and control of a prepaid electricity meter will prove beneficial to users and distributors alike. Effective gathering and analyses of data concerning electricity usage will help users to make better informed decisions concerning use of appliances and devices that consume electricity. This will allows users to save electricity and to monitor meters that are far away

Being able to remotely control prepaid meters will also be beneficial for users that are far from the meter. Instructions can be sent to the meter remotely via the Internet. These instructions can upload credit to meters or perform other functions such as tripping a meter's disconnecting device.

Originally, serial communication with the meter was attempted as a possible solution. This did not work as the specification provided for the meter for this project was inaccurate. Another approach had to be followed. An LED sensor, which was tested and found to work accurately, was used to gather usage data from the meter. A relay matrix was used to simulate key presses on the meter keypad, and allowed a user to remotely send instructions to the meter.

Die afstand moniteering en beheer van ‘n voorafbetaal elektrisiteitsmeter sal voordele inhou vir beide verbruikers sowel as verspruiders. Effektiewe insameling en analise van data rakende elektrisiteitsverbruik sal verbruikers help om beter ingeligte besluite te maak rakende gebruik van toerusting en toestelle wat elektrisiteit verbruik. Dit sal verbruikers toelaat om elektrisiteit te spaar asook hulle meters te monitor wat vêr weg is.

Om in staat te wees om voorafbetaal meters oor ‘n afstand te kan beheer sal ook voordelig wees vir verbruikers wat ver weg is van die meter. Instruksies kan oor afstand na die meter gestuur word via die Internet. Hierdie instruksies kan krediet op meters laai of ander funksies uitvoer soos om ‘n meter se ontkoppelingmeganisme te laat afskakel.

Aanvanklik is serie kommunikasie met die meter oorweeg as ‘n moontlike oplossing. Dit het nie gewerk nie omdat die spesifikasie soos vir die meter voorsien, onakkuraat was. ‘n Ander benadering moes gevolg word. ‘n LED sensor, wat getoets was en bevind om akkuraat te werk, is gebruik om gebruiksdata van die meter te versamel. ‘n Herlei matriks is gerbuik om sleuteldrukke op die meter se sleutelbord na te boots en het die verbruiker toegelaat om instruksies oor afstand na die meter te stuur.

Contents

Table of Symbols

I/O	Input / Output
M2M	Machine to Machine
TCP/IP	Transmission Control Protocol / Internet Protocol
STS	Standard Transfer Specification
UART	Universally Asynchronous Receiver / Transmitter
LDR	Light Detecting Resistor
LCD	Liquid Crystal Display
LED	Light Emitting Diode
SRAM	Static Random Access Memory
EEPROM	Electronically Erasable Programmable Read Only Memory
TTL	Transistor – Transistor Logic
RS-232	Recommended Standard 232
USB	Universal Serial Bus
PC	Personal Computer
SIM	Subscriber Identity Module

List of Figures

1. Introduction

1.1 Project Background

The current electricity situation in our country is of some concern [1]. This is due to a high demand for electricity in the country and limited investment in generation infrastructure. Effective analysis and management of electricity in South Africa thus becomes an essential aspect in supplying the nation with enough electricity to meet its demand. One aspect of data that needs to thus be managed is that of electricity usage by the public.

Prepaid electricity is a popular solution that is widely used in South Africa. The effective gathering and analysis of data surrounding prepaid electricity usage will provide distributors to more effectively manage distribution. A system that allows for such data gathering and analysis would need to be installed on individual prepaid units to gather data from them. The data gathered would then need to be presented in user friendly way to facilitate analysis. The data would also need to be remotely accessible.

Remote control of prepaid meters will also be beneficial to electricity distributors and consumers. Prepaid electricity meters can receive tokens entered via a keypad to carry out certain instructions in the meter. Currently, if a prepaid electricity user wants to load credit onto a prepaid meter, it is required that the user purchases a token from a distribution point or the internet, and then physically enters the token into the meter via the meter's keypad. Being able to enter these tokens remotely will be beneficial to users and will be a more convenient solution to this problem.

1.2 Literature Study

Research has been done into the remote control of prepaid meters. It was suggested that mobile communication be used to load credit to prepaid meters remotely. The meter would contain a prepaid card analogous to a mobile phone SIM card. The meter would then behave like a prepaid mobile phone, with credit being uploaded remotely and the electricity supply being cut once credit had run out. [2]

There are also products available on the market that allows clients to read information concerning electricity consumption from their meters. These product offer real time monitoring of electricity usage for users and also stores information concerning electricity usage that can be analysed by the user. These products help users to make more informed decisions concerning electricity usage. One such specific product is available from SEM. This product is installed onto a meter and uses a secure internet connection and allows a user to monitor a meter remotely. [3]

There are also products available that monitor a meter's electricity usage in real time and display it to a mobile device. One of these products is known as OWL, and it clips around the load or supply side cable running into or out of the electricity meter and monitors how much electricity is being used. It then transmits this data to a wireless, handheld device [4].

It is currently possible to purchase prepaid electricity vouchers via mobile phone. Once the voucher is purchased though, the credit token must still be entered to the meter physically at the site where the meter is located. Thus the purchasing of credit tokens can be done remotely, but the entering of the tokens must be done manually [5].

1.3 Aim

The project aims to gather usage data from a prepaid meter and to send instructions too it remotely. The instructions to be sent will be in the form of STS tokens [23], a software based identity provider. These are written to the meter to load credit onto it or to carry out other commands, for example, to trip the disconnecting device of the meter. The data to be gathered from the meter will be its electricity consumption. This data then needs to be displayed in an easy to understand and monitor way on the Internet.

1.3.1 Requirements

Trintel gave some requirements that the project had to adhere to. The first was that their online SMART platform was to be used to display the data read in from the meter. This platform is an online, interactive platform that will be discussed in detail later in the report (Chapter 2.1.2).

Trintel also supplied the meter and modem to be used. The meter model is the Itron ACE9000 Taurus ISP prepaid meter and the modem model is the Sierra Wireless Airlink GL6100. These components will also be discussed in more detail in a later chapter (Chapters 2.1.1 and 2.1.3 respectively).

1.3.2 Objectives

The following objectives are to be met in this project.

- Communicate with the meter.
- Communicate with the Trintel SMART platform via a modem.
- Continuously send data gathered from the meter to the Trintel SMART platform via the modem.
- Send instructions to the meter from the Trintel SMART platform via the modem.
- Be able to enter STS tokens into the meter from the Trintel SMART platform via the modem.

1.4 Approach

Originally, the desired approach to be followed was to communicate with the meter using a serial connection, which is used during assembly. This was not possible, as the specification that was provided for the meter was inaccurate. The work done until this was discovered and the reason why this route could not be followed will be discussed in a later chapter (Chapter 4).

A microprocessor is used to control the flow of data between the meter and the modem. A LED sensor is used to gather data concerning electricity usage from the meter. This form of electricity data gathering from a meter is not often used. The microprocessor also writes tokens to the meter by emulating key presses on the meter's keypad using a relay matrix. The relay matrix is necessary as the meter's ground floats at 230VAC. Thus, if it were connected

directly to a normally grounded device, the electronics would encounter problems. Finally, the microprocessor communicates with the modem through an RS-232 converter to give messages the correct voltage levels. These hardware components will be explained in greater detail in the following chapters.

Data is sent from the modem to the online Trintel SMART platform, which enables the effective displaying of data sent to the modem. The tokens that are to be written to the meter is sent from the online SMART platform to the meter via the modem and microprocessor.

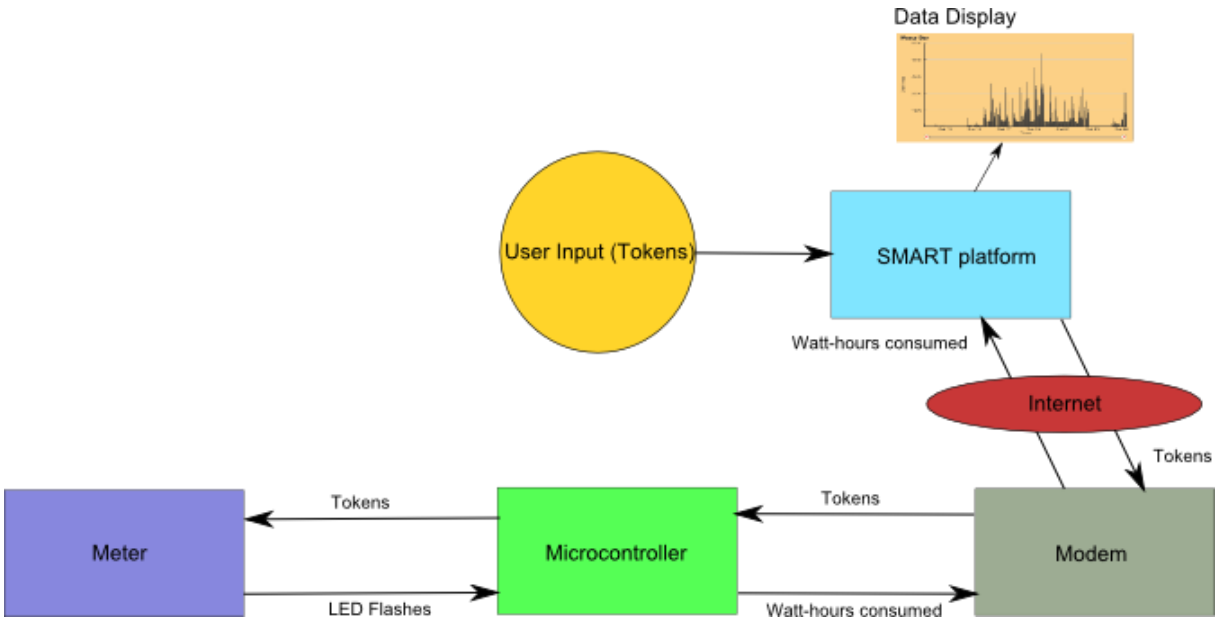


Figure 1 - System Diagram

A demonstration video for the project can be found at http://youtu.be/uH-bfwv4Dxo

2. System Layout

2.1 Key Components

There are a few components that are keys to this project and deserve further explanation here. These components are both hardware and software components and are listed below:

- Itron ACE9000 Taurus ISP Prepaid Meter
- Trintel Online SMART Platform
- Sierra Wireless Airlink GL6100 Modem

2.1.1 Itron ACE9000 Taurus ISP Electricity meter

The Itron ACE9000 Taurus is a single phase, one-way prepayment meter developed by Itron Incorporated [6]. It monitors the electricity use of a given house / electricity using point and is loaded with credit to allow control of the amount of electricity used by the point. Credit is loaded via tokens that can be purchased easily by the public.

Tokens are loaded onto the meter via a keypad, located on the meter. The keypad has buttons for digits 0 – 9, a Backspace button and an Enter button. There are also Reset and Test buttons that cannot be pressed unless the meter cover is removed. Tokens to be entered are of the STS standard and consist of 20-digits each [23]. The 20-digit token is entered via the keypad, and Enter is pressed to confirm token entry. The meter then either accepts or rejects the entry.

An STS-Token can carry more information than just the uploading of credit to the meter. Other instructions can also be given to the meter through the use of STS tokens. Some of these are tabulated below. These tokens are not meter-specific and can be used to perform the indicated commands on any meter.

Description	Token
Display Electricity Totals	00000000000201328896
Display Key Revision Number	18446744073843772416
Display Tariff Index	36893488147553322496
Display Power Limit	00000000001207974400
Display Tamper Status	00000000002281728512
Display Power Consumption	00000000004429208064
Display Version	00000000008724195840
Trip Disconnecting Device	00000000000150997584
Test Display	00000000000167774880
Initiate Dispenser Test	56493153725450313471
Display Phase Unbalance	00000000017314105857

Table 1 – STS token examples [24]

The meter has a LCD screen where data is displayed. The total credit available is displayed by default, and is decremented as credit is used. On the right hand side of the LCD screen there is a LED that flashes for every watt-hour consumed by the meter i.e. it flashes 10 times for the number on the LCD to be decremented by 0.01 (0.01 kWh is equal to 10 Wh). A picture of the meter is included below.

Figure 2 - Itron ACE9000 Taurus ISP Prepaid Electricity Meter

The meter connects an electricity source (from ESKOM) and a load (typically a house). It also contains a switching mechanism which allows for the connection of the source and the load or disallows it. It was also found by observation (measurement with a multi-meter) that the meter's logic's ground (including the external serial port) floats at 230VAC. This means that any logic signal in the meter has a 230VAC offset.

The meter was supplied by Trintel for the project.

2.1.2 Trintel Online Platform

Trintel offers a data processing and analysis solution online, known as the SMART platform [7]. This platform works hand-in-hand with the Sierra Wireless AirVantage tool [8], a tool that comes standard with Sierra Wireless Airlink modems (the modem provided for use in this project by Trintel), and allows a user to manipulate and present data variables, which have been created using the AirVantage tool, in various ways. This data will typically be data gathered from sensors monitoring various machines.

The platform as a whole consists of three separate platforms:

- The AirVantage Configuration Tool
- The AirVantage Operating Portal
- The SMART platform

(i) AirVantage configuration tool

The AirVantage configuration tool comes standard with a Sierra Wireless Airlink modem and allows the user to create data variables, events, commands, alarms and operating states that the modem will then recognize and be able to change and send up to the online platform when prompted. This is done by linking the AirVantage tool to the online AirVantage operating portal and updating the portal using the AirVantage tool. Through creating these data variables, an asset model is created, which can be thought of as an object with its own data variables, commands, events, alarms and operating states. Thus, data and commands are grouped into assets that can be linked to a specific device (in our case a modem) [8]. A diagrammatical representation of how this works is included below to help with understanding.

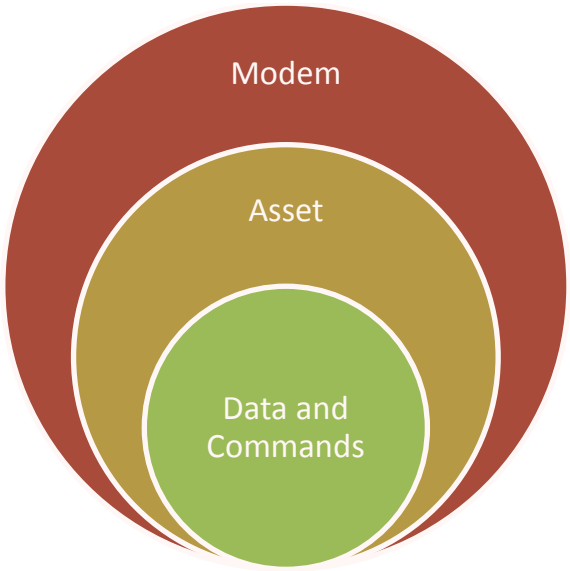


Figure 3 - Modem Hierarchy Diagram

It is important to note that a modem can have multiple assets linked to it.

(ii) AirVantage Online portal

The AirVantage online operating portal is an online tool that is used to monitor and manage devices (in the form of modems) and the assets (created by the user) that are linked to these devices. The portal monitors data variables contained in the modem and updates them as they change. It also allows a user to send data and commands from the platform to the modem. The data will then also be sent to devices that are serially connected to the modem. Below is a screenshot of the portal where devices being monitored are listed.

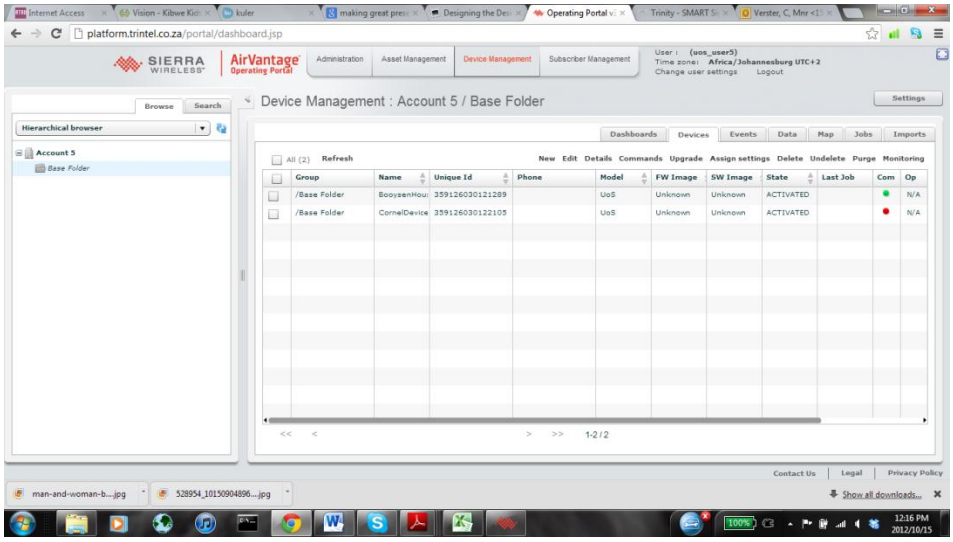


Figure 4 - AirVantage Portal device listing

Basic information such as which Asset model the devices use, what state the devices are in etc. are listed. Device data can also be viewed from the portal.

Every modem has certain default variables that are permanently monitored. The screenshot below shows these being monitored.

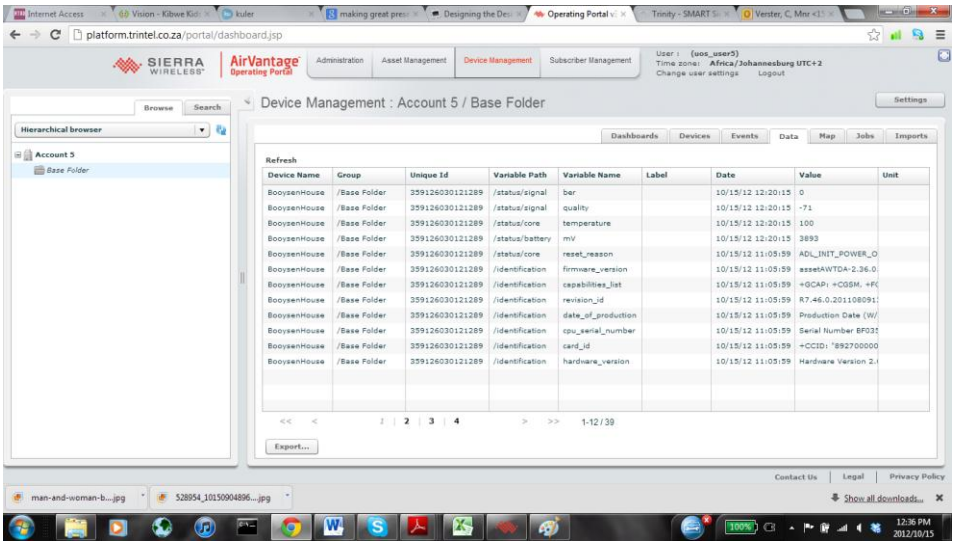


Figure 5 - AirVantage device data

As mentioned before, assets are created as objects containing data and commands for devices. These assets are also monitored and managed via the AirVantage portal.

Once again, basic information about these assets is included in this listing. By selecting an asset and clicking on the “Commands…” button, users can write commands to the device the asset is linked to, or can change the value of data variables contained in the asset. Commands are used to send data in the form of variables to devices. This data can either be a String, Integer, Boolean, Binary or Double variable. Commands sent to the modem via the AirVantage portal are in the form of unsolicited AT command messages. These will be explained in more detail later (Chapter 2.2.3). Data variables created for an asset can also be viewed on the portal and a screenshot of this can be found below.

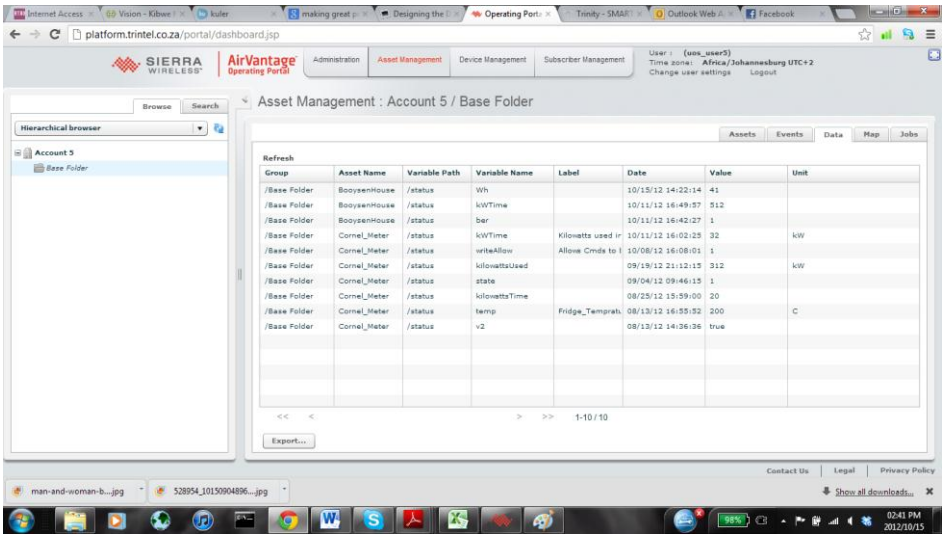


Figure 6 – AirVantage Asset data

(iii) The SMART platform

The SMART platform is an online tool by Trintel that allows a user to manipulate and present data gathered on the AirVantage online portal effectively. The SMART platform has a user-friendly interface that makes it easy for users to work with their data and to display it conveniently. The platform gathers data from the AirVantage portal and makes it ready to be used in various available transform and display techniques.

Each asset has its own dashboard, which is a space provided to display all data. Before data can be displayed, it must go through three stages of preparation on the platform and these are explained graphically below.

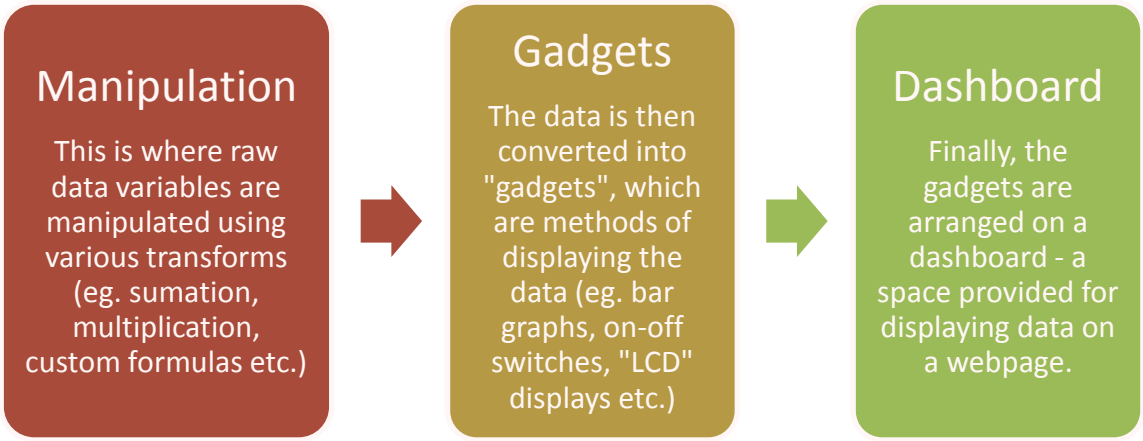


Figure 7 - SMART platform data cycle

A screenshot example of a dashboard is included below:

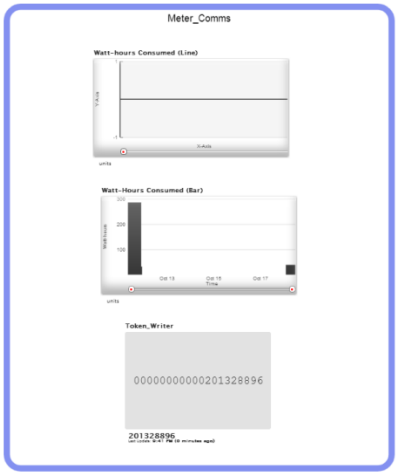


Figure 8 - SMART dashboard

The SMART platform also includes a tool called Spyglass, which allows users to monitor activity on devices in real-time.

2.1.3 Sierra Wireless Airlink GL6100 Modem

The Sierra Wireless Airlink series are data modems that bring cellular connectivity to any device. They are designed specifically for M2M customers and come with an embedded AT-command driven TCP/IP stack. Communication to the modem is done on RS-232 standard levels (12V).

The modems also come packaged with the AirVantage Management Service, a solution which allows for remote diagnosis and software upgrades to the modem. This tool also allows the user to create data variables and commands that the modem will recognise and that can then be used to send data to online platforms and to receive instructions from them. [9]

The modem only needs to be sent ASCII data in the form of AT-commands. These commands are then interpreted by the modem and sent as data to the online platform. Data processing and display is then done on the platform.

The modem is powered via a wall-socket plug provided with the modem.

(i) The AT-Commands

Embedded cellular networks area mainly controlled by the use of AT commands. The AT stands for Attention, referring to calling the modem to attention as an instruction is being sent. The commands usually start with "at". AT commands are used to get data from, and send data to a modem. In our case, we used only a few AT commands to perform our desired actions [10]. These commands will be explained later in the report (Chapter 2.2.3 (iii)).

The modem was supplied by Trintel to be used in the project.

2.2 Solution Components

To reach the objectives of the project, the following had to be done:

- Continuously gather data about electricity consumption from the meter.
- Display the gathered data on the SMART platform.
- Be able to write tokens to the meter.

The approaches followed to achieve these objectives are briefly described in the Approach section (Chapter 1.3). The details of the hardware, software and functionality used will now be further described.

2.2.1 The Microcontroller Board – Arduino Mega 2560

Arduino is an open-source electronics prototyping platform, founded by Massimo Banzi and David Cuartielles. The project was initiated in 2005 in Ivrea, Italy and was aimed at making a cheaper way for students to procure prototyping systems. [11][12] Arduino now has a larger range of products available from a variety of manufacturers.

The platform is built around the Wiring project, created as a master's thesis project by Hernando Barragan [13]. It provides users with an easy-to-code environment that allows for more user-friendly development and prototyping.

Arduino also has a large online-community that provides good support and help for new developers.

The Arduino Mega 2560 microcontroller board was used as the microcontroller for this project. Originally, the Arduino Uno microcontroller board was to be used, but this was decided against as the Uno only has one external serial UART. This means that we would not be able to communicate with the modem and the meter simultaneously.

The Arduino Mega 2560 was chosen in place of the Uno, mainly because it has 3 external serial ports, each with its own UART. The Mega uses the ATmega2560 microcontroller and has 54 digital I/O pins and 16 analog input pins [14]. The digital I/O pins can either be used as inputs or outputs. When used as an output, the pin can either output 0V (logical LOW), or 5V (logical HIGH). The analog input pins are used to monitor a voltage applied to them that can vary between 0V and 5V.

The Arduino Mega 2560 can either be powered via a USB cable from a computer (through which it can also be programmed), or from an external 12V source. The board also has pins that output 5V and 3.3V for used outside the board, and 3 ground pins. The board also contains on it 256KB of flash memory, 8KB of SRAM memory and 4KB of EEPROM memory. [14] A picture of the Arduino Mega 2560 is included below:

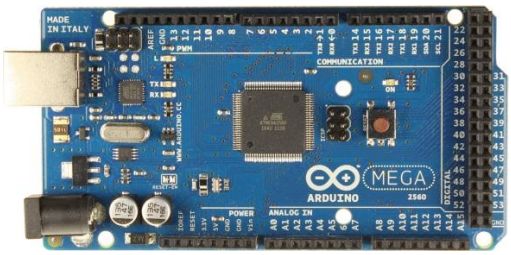


Figure 9 – Arduino Mega 2560

(i) Programming the Arduino

Arduino has its own programming environment that works with the C programming language. It also comes with a number of standard libraries to help user to easily program their Arduino boards. This environment was used in this project to program the Arduino Mega 2560 board. Connecting the Arudino board to the PC was done via a USB cable that plugs into a USB socket on the Arduino board. The board can then be programed directly from the Arduino programming environment after the correct drivers for the Arduino board have been installed on the PC.

(ii) Powering the Arduino

As previously mentioned, the Arduino board can be powered in one of two ways. It can either be powered via a USB cable, which is also used for programming the board, or via applying 7V-12V over the Vin pin and the ground pin on the board. During the testing phase of the project, the Arduino was powered using a USB cable. When the unit is deployed into the

field, the Vin port will be made use of. A voltage transformer that scales down voltage from 230V AC to 12V DC is used to power the Arduino board from a wall plug.

2.2.2 Gathering data from the meter – The LED Sensor

(i) Hardware

In this project, an LED sensor is used to extract data concerning electricity usage from the meter. As mentioned earlier in the report, the meter has a LED located to the right of the LCD screen, on the front of the meter. This LED flashes for every watt-hour consumed by the meter. The LCD screen displays available credit remaining on the meter by default. It displays the credit in kilowatt-hours available, to the second decimal. Thus, for every ten flashes of the LED (10 watt-hours consumed), the number displayed on the LCD screen is decremented by 0.01.

To monitor the watt-hours consumed by the meter, a sensor is placed over the LED and monitored. The sensor consists of a LDR (light detecting resistor) connected in a voltage divider circuit, the output of which is measured by an analog pin on the Arduino Mega 2560 board. A voltage divider uses two resistors to give a desired (equal or lower) output voltage from and input voltage. The output voltage is determined by the ratio of the two resistors in the voltage divider. This concept is shown graphically below.

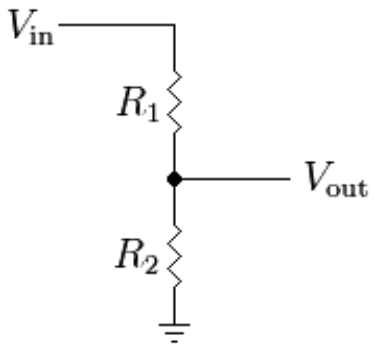


Figure 10 - Resistive Divider [15]

In our case, R1 is replaced by the LDR. When the LDR is exposed to light, its resistive value decreases. The smaller R1's resistance is, the closer the ratio of V_{out} to V_{in} gets to one, thus V_{out} becomes larger. In the project V_{out} is connected to the A0 analog input pin on the Arduino Mega 2560 device. This pin monitors a voltage that is applied to it. The voltage applied to it can vary between 0V and 5V. The Arduino then converts the voltage read on pin A0 to an integer. The higher the voltage on the pin is, the higher the value of the integer will be. When 0V is applied to A0, an integer value of 0 will be read on the pin. When 5V is applied to A0, an integer value of 1023 will be read on the pin. The integer value will increase by 4.9 for every mV that the applied voltage rises [16]. This creates a sensor by which it is possible to know when the LED is on or off.

In our case, the following specification was used for the sensor (referring to figure 11):

Symbol	Value
R1	LDR
R2	1kΩ
V_{in}	4.9V (Supplied from Arduino board)
V_{out}	100mV when LDR exposed to light* 0V when LDR not exposed to light*

Table 2 – Resistive divider specification

*Found by experimentation

A voltage profile showing the output of the sensor when the LED is switched on can be found below:

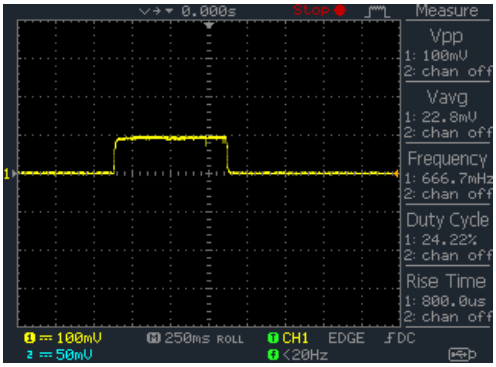


Figure 11 - LED Sensor Voltage Profile

By experimentation it was found that the value that was given by A0 is 0 when the LED is off, and about an average of 190 (varies around 190) when the LED is on. Different resistors were tested on R2 before a 1kΩ was chosen. The table below shows what integers were read on analog pin A0 with different resistor values for R2.

R2 Value	A0 Value (LED off)	A0 Value (LED on) (Average)
1kΩ	0	190
4.7kΩ	0	531
10kΩ	0	710
100kΩ	4	983

Table 3 - LDR values and results

A 1kΩ resistor was chosen for R2 because it gives a large enough range to measure whether the LED is on or off accurately. It also has the lowest value when the LED is on and thus is the simplest solution to use.

(ii) Software

The value read on A0 is interpreted to sense whether the LED is on or off. When the value read on A0 becomes larger than 80, the LED is registered as on. Only when the value on A0 then again drops below 80 does the sensor register the LED as off. This means that when the value rises above 30 and then again drops below 80, one LED flash is registered. This value is specific

to the meter provided for the project and will vary with different meters. This is because the LED strength on a different meter may be stronger or weaker than the LED on the meter provided. The software surrounding the sensor is explained diagrammatically below.

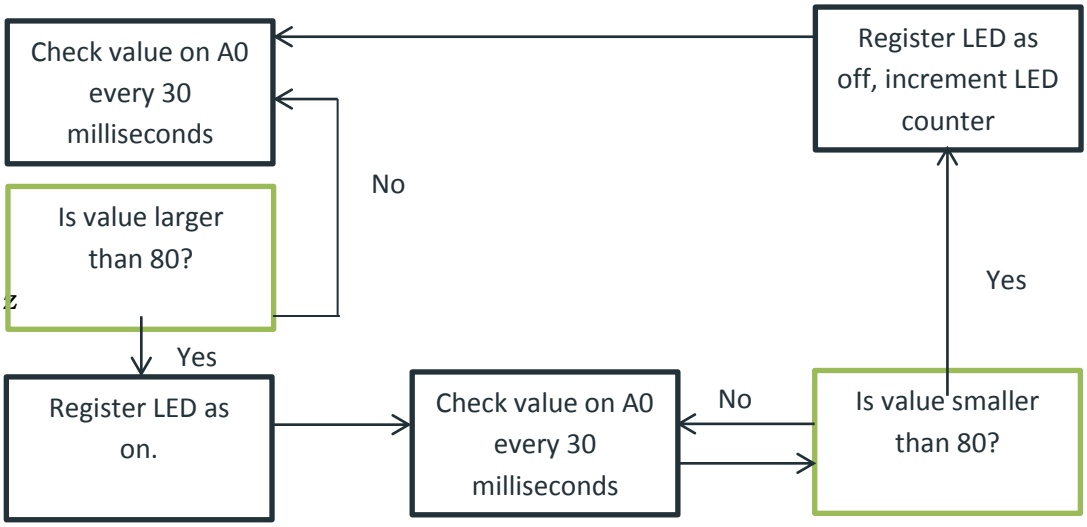


Figure 12 - LED Sensor check

(iii) Possible Alternatives:

Some alternatives were considered for the LED sensor other than using a LDR in a voltage divider. One possible alternative is the use of another LED to detect whether the LED on the meter is on or off. The second LED (used as sensor) is then connected in the same position as the LDR in our setup. The setup then works in a similar way to ours, except the value of R2 will differ. When the LED being monitored is switched on, a voltage is induced on the sensor LED and this voltage can then be monitored to see if the LED being watched is on or off. This route was not taken as it is more complex than a single LDR sensor and I was more comfortable with using a LDR in a voltage divider.

Another possible alternative was to measure the pulses that are sent by the meter to the LED by measuring the line to the LED on the meter's interface board. After inspecting the meter, it was concluded that I myself would not be able to solder a wire onto the LED line as it required some soldering expertise that exceeded my own. Also, using the line from the LED would require isolation between the meter and the Arduino, which the LDR approach avoided.

2.2.3 Communication with the SMART platform

(i) Serial Communication

Communication between the Arduino board and the modem happens serially. The Arduino has a UART, which is used to receive and send data. The UART transmits one bit at a time at a specified data, or baud, rate. In our case this baud rate is 9600, and how the Arduino and modem was set

up to transmit at this rate will be explained later (Chapter 2.2.3 (iii)). The Arduino's form of communication is referred to as TTL communication. In TTL communication, the voltage levels used in communicating will always remain between 0V and Vcc, which in our case is 5V. 0V represents a logic low (0) and 5V represents a logic high (1).

The modem uses the RS-232 standard to send and receive data. Similarly to the TTL standard, it transmits one bit at a time at a specified baud rate. Parity (a check to see if data was correctly received) and the amount of stop bits to be used can also be set on both standards. The only difference between the standards is a hardware difference. The RS-232 standard uses different voltage levels than the TTL standard. A logical high (1) is represented by a negative voltage anywhere between -3V to -25V. A logical low (0) is represented by a positive voltage anywhere from 3V to 25V. [17]

(ii) Hardware

The Sierra Wireless Airlink GI6100 communicates using RS-232 levels [18]. The Arduino Mega 2560, however, communicates using TTL voltage levels [14]. This means that for the Mega to communicate with the modem, the TTL levels sent by the Mega must be converted to RS-232 levels and vice-versa.

This was done by the use of a MAX3322E [26] chip. This chip converts TTL levels to RS-232 levels and vice-versa. The chip's datasheet clearly shows a user how to use the chip and correctly connect a device to it [19]. This guide was followed in the project to convert the signals to correct levels for use in communication. A circuit diagram of the chip and the way it is connected can be found below.

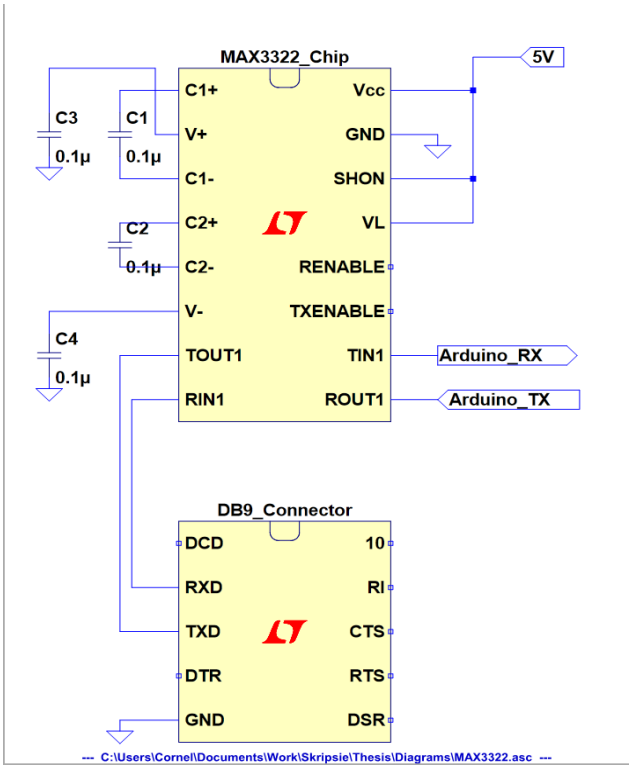


Figure 13 - MAX2233 chip implementation

(iii) Software

The software side of communicating with the SMART platform consists of a couple of stages. A detailed explanation of how AT commands apply to the modem will first be given.

a. AT Commands

As mentioned earlier in the report, AT commands are commands that are used to control modems. The Sierra Wireless Airlink Gl6100 modem also responds to AT commands. These commands prompt the modem to carry out certain functions, such as to give information about itself or to change settings on the modem. These commands are sent to the modem serially and as ASCII characters. All commands are ended with a carriage return character [8].

The modem will respond to commands sent to it. It can also serially send messages to devices that are connected to it periodically or when commanded to do so from the online portal.

A list of important commands that were used in the project and messages that the modem responds with or sends as commanded from the online portal follows.

AT Command	Description
AT+IPR=<rate>	Sets the baud rate of the modem. <rate> is replaced with the desired baud rate, such as 9600 or 115200. When '0' is entered as the rate, autobauding is enabled.
AT+IFC=<DCM_by_DTM>,<DTM_by_DCE>	Enables or disables flow control. Flow control is disabled by setting both parameters (<DCE_by_DTM> and <DTM_by_DCE>) to zero.
AT+AWTDA=d,"<asset_name>",<nr. Variables>,"<variable_name>,<variable_type>,<value>"	This is a command that sends data through the modem to the online portal. **<asset_name>** is the name of the asset the device is linked to on the online portal. **<nr. Variables>** is the number of variables you will be sending data to

	<variable_name> Is the name of the variable you would like to update with the data **<variable_type>** is the type of the variable you are updating. This will be one of the following: *BOOL* – Boolean *INT32* – A 32-bit integer **<value>** this is the value you are giving the variable. *Note: more than one variable can be updated in a single command. Each variable will then have its own set of name, type and value in inverted commas, separated by a comma.
AT+AWTDA=c,"<asset_name>","<command_name">	This command activates a listener on the modem that listens for commands coming down from the online portal. When a command is received, it will then be written out serially by the modem in the form of an unsolicited message.
AT+AWTDA=a,"<asset_name>",<ticket_ID>	This command acknowledges that a certain command or data sent from the modem has been received. The ticket ID is a unique number linked with that job (command) and is used to enable the portal to know which job has been completed.
AT	This is a command that tests the connection to the modem. The modem returns "OK".
AT&W	This command writes the active modem configuration into a non-volatile memory on the modem (EEPROM).

Table 4 - AT Commands given to modem [8], [20]

Messages sent by modem:

Message	Description
OK	Sent from the modem when a command is correctly received.
ERROR	Sent from modem when a command causes an error.
+AWTDA: UP	Sent periodically from the modem (default every 60 seconds) to show that the modem is up and running and the connection is working properly.
+AWTDA: DN	Sent from the modem when the modem is down, meaning it is not functioning properly.
+AWTDA: BOOT	Sent from the modem when the modem is booting up. This message is only sent once every time the modem boots to show that the modem is booting.
+AWTDA: TIMEOUT	Sent from the modem when the connection to the modem has timed out. This means that there is something wrong with the connection to the modem.
+AWTDA: c,<ticket_ID>,"<asset_name>",<command_name>",<variable_type>,<value>,CRC02	This message is sent from the modem that is a command sent from the online portal. Commands are used to send data variables from the online platform to devices connected to the modem. **<ticket_ID>** is a unique number assigned for the job being performed **<asset_name>** is the name of the asset the device is linked to on the online portal. **<command_name>** is the name of the command being performed **<variable_type>** is the type of the variable being sent down **<value>** is the value of the variable being sent

Table 5 - Modem Response Messages [8], [20]

(iv) Setting up communications:

To set up communication between the Arduino and the modem, both devices must be set to the same baud rate and flow control must be off on the modem. The settings set on the modem must then be saved as permanent. In the project, the modem was connected to a laptop with a RS-232 to USB connector and Terraterm (a terminal program) was used to communicate with the modem initially.

The following commands are sent to the modem to set it up properly:

Sent from Terminal	Modem Returns	Description
AT	OK	Used to test connection
AT+IPR=0	OK	Enable autobauding
AT+IFC=0,0	OK	Disable flow control
AT&W	OK	Save active modem configuration

Table 6 - Modem Setup Dialogue

The following commands are used in the Arduino programming environment to set up the serial ports to the correct baud rate.

Serial.begin(9600);
Serial1.begin(9600);

These two commands set up both the serial port that is used for communication between the Arduino and a computer (when programming and debugging), and the serial port used to communicate with the modem (serial port 1) to a baud rate of 9600. After setting up both the Arduino and the modem for serial communication, instruction and data can be sent directly from the Arduino to the modem and vice-versa.

(v) Timers

For this project, a timer library was used to allow for the use of interrupts at certain time intervals in the programming. The library is called Timer and is an external library programmed for Arduino as the normal Arduino interface does not have a library that allows for the easy use of timers. The library allows for the following:

- Use of periodic timers (interrupt every X seconds) through a function called every();
- Use of a delayed run of a function (run function after X seconds) through a function called after();

(vi) Sending data to the online portal

For the project, and Asset named “meter” was created on the online portal. This asset represents the prepaid electricity meter. A data variable (integer) named Wh (Watt-hours) was created on this asset to monitor the amount of electricity used by the meter per hour. This data variable is then then updated with the value of watt-hours consumed by the meter every hour, thus, a data writing command is sent from the Arduino (monitoring the LED sensor) to the meter once every hour. The dialogue between the Arduino and the modem when this variable is written is shown below:

Sent from Arduino: *AT+AWTDA:*
d,”meter.status”,1,”Wh,INT32,<value>”<CR>

Here, <CR> is the carriage return character. <value> is the amount of flashes detected by the LED sensor is the last hour.

Response from Modem: OK

If the modem return OK, the data writing command has been sent successfully.

ERROR

If the modem return ERROR, the data has not been successfully written and the Arduino will attempt to write the data again after a few seconds (2 seconds to be exact). This will continue until the data is successfully written.

This data writing cycles is described diagrammatically below.

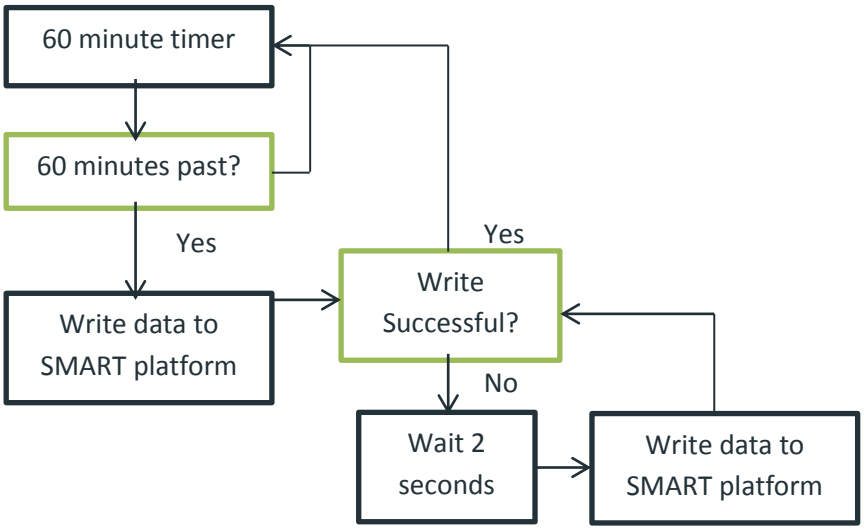


Figure 14 - Data Writing Cycle

A check is also done after writing data to ensure that a response is received from the modem. If no response is received after 5 seconds, the modem has probably timed out, and another attempt will be made to write the data to the modem.

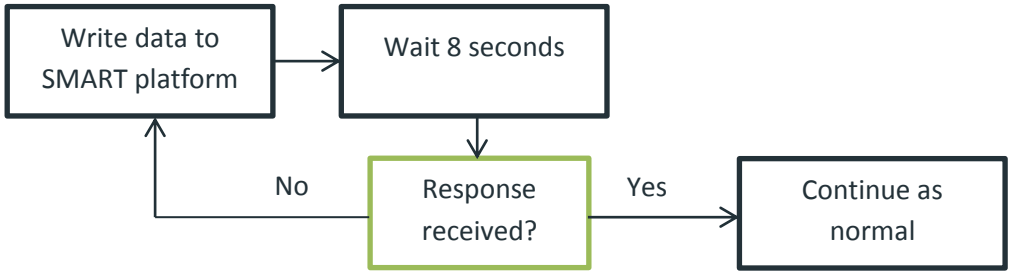


Figure 15 - Timeout Check Cycle

The data that is written to the SMART platform (watt-hours consumed per hour) is then displayed on a dashboard set up for the asset that represents the meter. The data is displayed on the dashboard in the form of a line

graph. Another graph indicated how much is paid for the electricity consumed. A screenshot of these graphs can be seen below.

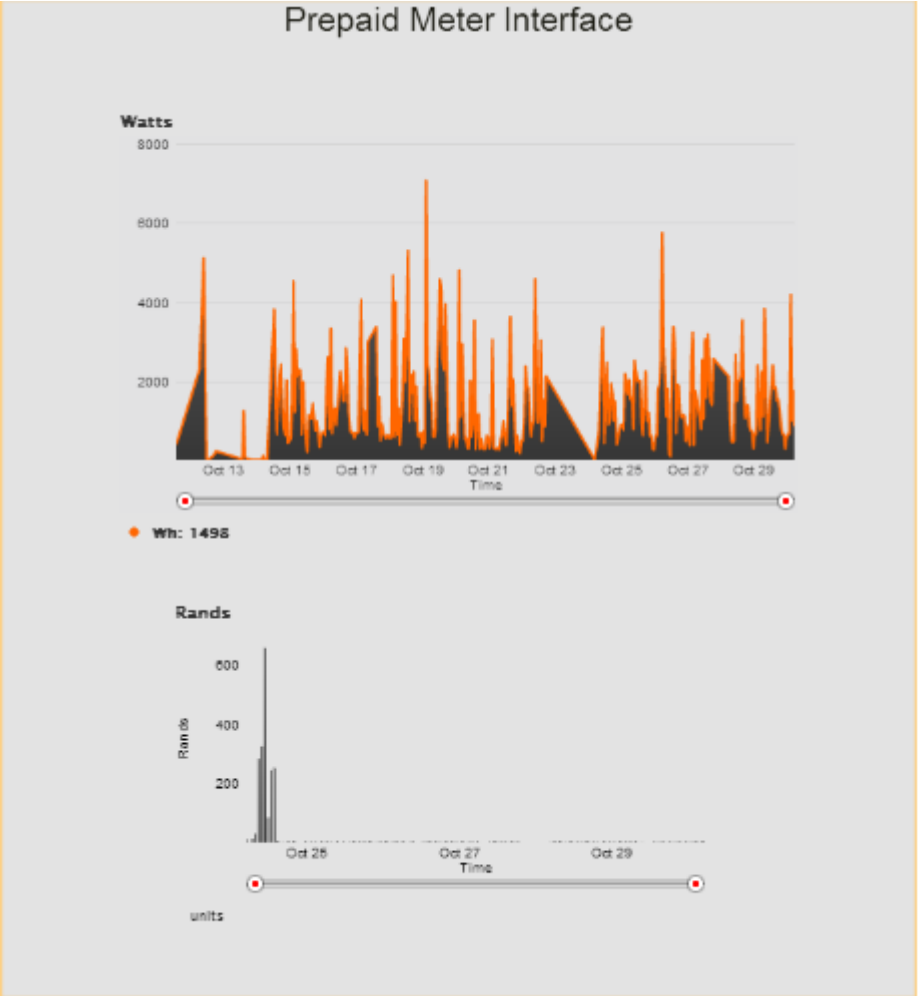


Figure 16 - Dashboard graphs

(vii) Sending tokens from the online SMART platform to the Arduino

A text box can be found on the asset dashboard on the SMART platform (see Chapter 2.2.4 (iii)). This text box is used to send tokens that the user wishes to enter into the meter. As mentioned in Chapter 2.1.1, tokens are 20 digit numbers that are entered into the meter via a keypad on the meter. To send a token from the SMART platform to the Arduino board, the following procedure must be followed.

Firstly, a command is created for the meter asset that allows the user to write a 20-digit string variable (the token) to the modem. The command is aptly named “Write_Token”. A listener must be activated on the modem to listen for this command when it is sent from the online platform. This listener must be activated every time the modem boots. A timer was used in the programming to activate this listener every 19 seconds. This amount of time may seem random, but it was chosen to ensure that this command is not sent at the same time as other commands that are sent periodically. The

following dialogue takes place between the Arduino and the modem when the listener is activated.

Sent from Arduino: ***AT+AWTDA: c,"meter","Write_Token"***

This command activates a listener for the "Write_Token" command on the modem

Response from Modem: OK

If OK is returned by modem, listener has successfully been activated.

ERROR

If ERROR is returned by modem, something has gone wrong and listener has not been successfully activated. The Arduino will wait 2 seconds and then attempt to activate the listener again.

NOTE: A timeout check is again applied for this command.

To send a token from the online SMART platform to the Arduino, the token to be sent is entered by the user into the text box on the asset dashboard on the SMART platform. When "Submit Changes" is clicked by the user, the Write_Token command containing the token entered by the user will be sent to the modem as soon as the platform is sure that the modem is online and working. The way the platform knows this, is by receiving a message from the modem. This creates a problem, as the modem then needs to send a message to the portal to receive a command from the portal.

To deal with this, a variable was created on the asset to be periodically written every 11 seconds, to ensure that a command will be sent to the modem from the portal a maximum of 11 seconds after the user issues it. This variable is an integer and is named "writeAllow". A timer is used to write a value of 1 to this variable every 11 seconds, plainly for the purpose of opening the path for any commands from the portal to come through to the modem. The dialogue between the Arduino and the modem looks as follows when this variable is written:

Sent from Arduino: ***AT+AWTDA: d,"meter.status",1,"writeAllow,INT32,1"***

This allows an open path for commands to be sent from the online portal to the modem.

Response from Modem: OK

The modem correctly received the instruction and writeAllow was written with a value of 1.

ERROR

The writeAllow variable was not correctly written. As this command is periodically written every 11 seconds, nothing is done when an ERROR is received from the modem.

When an open path has been created for a token to be written from the online portal to the modem, the token will be written down contained within a message. The token is a 20-digit string and is contained in a message from the online portal that looks as follows:

+awtda: c,<ticket_ID>,"meter","Write_Token",STR,<token>,<CRC_nr>

<ticket_ID> is a unique number assigned to each job.

<token> is the 20-digit token string

<CRC_nr> is a unique number

As can be seen from the message, asset and command names (meter and Write_Token respectively) as well as what type of variable is being sent are also indicated. The 20-digit token as well as the ticket ID is then extracted from this message and saved in variables on the Arduino. The 20-digit token is then immediately written to the meter through the relay matrix. This process will be explained in detail later in this chapter (Chapter 2.2.4).

After the token has been successfully written to the meter, an acknowledge message is sent to the modem from the Arduino to tell the online portal that the job has been done. The acknowledge message looks as follows:

AT+AWTDA=a,<asset_name>,<ticket_ID>

The ticket ID is used to identify which job has been completed. When this message has been successfully sent the online portal, it will be indicated on the portal that the job has been completed. As before, if the acknowledge message receives an ERROR message back from the modem, the Arduino will attempt to write the message again to the modem after 2 seconds.

This process is continually run on the Arduino board. Through experimentation it was discovered that the modem reboots quite often when it idles. The following statistics were found when the modem was left to idle for 3 hour 15 minutes:

UP counts	213
DN counts	20
Boot counts	10

Figure 17 - Modem Boot Statistics

If an attempt is made by the Arduino to write data to the modem while the modem is booting, the modem will return an ERROR message. As explained, code was written to work around this problem. When the modem boots, it was found by observation that the following messages are sent from the modem to the Arduino:

+AWTDA: DN

+AWTDA: DN

+AWTDA: BOOT

A check is put into the code to reactivate the command listener on the Arduino (that listens for commands from the online portal) 2 seconds after the *+AWTDA: BOOT* message is received. A writeAllow does not immediately

have to be sent to the modem as the modem will still write the command once the next writeAllow variable, or data variable is sent up to it.

When a token is sent from the online portal to the modem, the modem sends the token to the Arduino in a message as described previously. If the modem boots while this message is being sent and it does not go through, it cannot be recovered. The command message is lost and never reaches the Arduino. It is not possible to program around this as the Arduino cannot inform the online platform it has successfully received a command message because it does not know when one is coming.

2.2.4 Writing Tokens to the Meter - The Relay Matrix

Instructions can be sent to the Itron ACE9000 Taurus prepaid meter via STS tokens. As mentioned earlier, the STS token system is a system that allows users to send instructions to prepaid meters by entering 20-digit tokens via their keypads. A token is a 20-digit number. STS tokens can be divided into three categories:

- Credit Tokens – For dispensing credit to an electricity meter.
- Meter specific engineering tokens – Tokens used to set power limit, clear credit, change meter key, set tariff rate, clear tamper status, set phase power limit.
- Non-meter specific engineering tokens – Tokens used to trip disconnecting device, test display, display supply group code, display electricity totals, display key revision number, display tariff index, display power limit, display tamper status, display power consumption, initiate dispenser test, display phase unbalance.

A list of non-meter specific engineering tokens can be found in Table 1.

To enter these tokens remotely into the prepaid meter, a relay matrix was used to simulate key presses on the meter keypad. The relay matrix is then controlled by the Arduino.

(i) Meter Keypad

The meter has on it a keypad with press-button keys. The keypad consists of 14 keys, 12 of which are press able by the user. The press able keys include keys for digits 0 – 9, a backspace key and a enter key. The non-press able keys are a test key and a reset key. The keypad consists of 8 lines, four which are continually at 5V and four which are at 0.5V. These high and low lines are connected via a matrix of switches to let the meter know that a key on the keypad has been pressed. The diagram below explains the workings of the keypad:

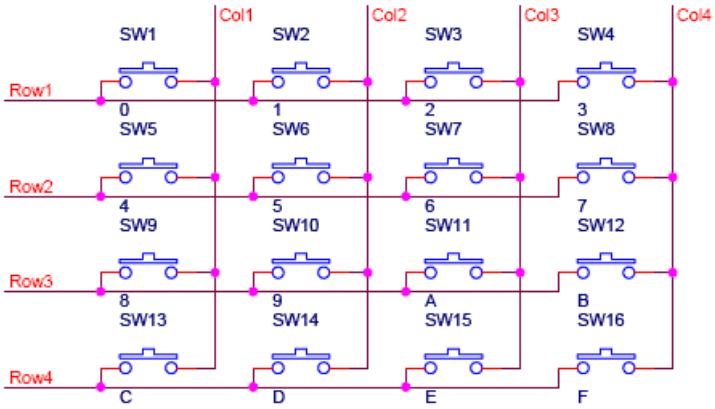


Figure 18 - Keypad Matrix

NOTE: For the meter, only 14 switches are used on the meter keypad as there are only 14 keys.

Through observation it was seen that the signals on the high lines (5V) spike down to 0V every 20ms (that is on 50Hz). Similarly, the low lines spike up to 5V every 20ms (also on 50Hz). When these two signals are compared using an oscilloscope, the spikes overlap almost precisely. This can be seen in the oscilloscope measurement below:

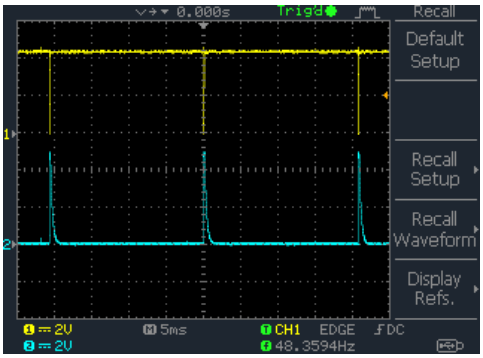


Figure 19 - Oscilloscope Measurement of Keypad Signals

When a button is pressed on the keypad, a high line and a low line are connected and both are pulled to about 1.3V. This triggers a key press.

(ii) Hardware

A relay matrix is necessary for two reasons:

1. There needs to be isolation between then meter and the Arduino as the meter ground floats at 230V and the Arduino ground is at 0V.
2. Relays act as switches that are used emulate key presses on the meter's keypad.

The relay matrix consists of 12 HFD41 relays, with their coils being triggered by digital output pins on the Arduino board. Each relay is used to switch two meter keypad lines, one high line (5V) and one low line (0.5V). Therefore,

each relay connects two different lines on the keypad. The diagram below shows how each relay is connected.

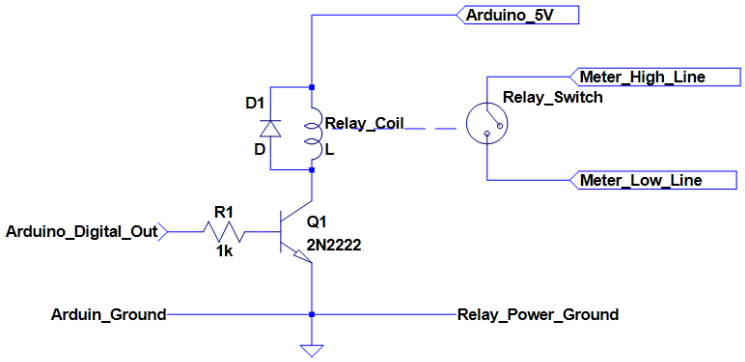


Figure 20 - Relay Implementation

For a full diagram of the relay matrix, please see the full circuit diagram in Appendix E.
NOTE: Each relay element in the full system diagram is in Figure 21.

Each relay is used to control one key. The relay matrix functions properly in emulating key presses when the meter is powered from a 5V source or from a 230V source. Before a relay matrix was used as the solution to programming tokens to the prepaid meter, an opto-coupler matrix was used. This method was abandoned after being built and tested. The reason for this was that when the meter was powered from a 230V source, the opto-coupler matrix caused multiple key presses when trigger to press a key only once. This occurred because the voltage which the high and low lines were pulled to when connected through the opto-coupler rose and fell between 1V and 5V. Every time it passed a certain threshold, the meter would register a key press, thus making it very unlikely to write a 20-digit token to the meter successfully. After this method failed, relays were implanted and found to work correctly. A diagram of how the opto-coupler matrix was implemented can be found below.

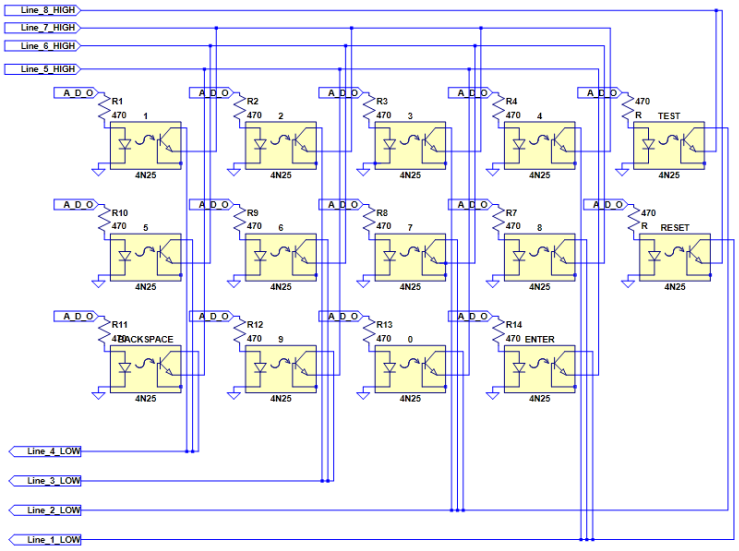


Figure 21 - Opto-Coupler Matrix

Referring to Figure 21, each relay is driven by an Arduino digital output pin using a 2n2222 transistor. This is done to act as a simple and effective switch to switch the relay on and off. When the digital output is driven high, the transistor is switched on and the relay is switched, connecting a high line on the meter keypad to a low line, and emulating a key press. A diode is connected across the relay coil to prevent back current from flowing into the Arduino. The current that is fed the 2n2222 is:

$$V = I * R$$
$$I = \frac{V}{R}$$
$$= \frac{5}{1000}$$
$$= 5mA$$

The pin-to-key relationships for which Arduino pin is used to emulate which key press is tabulated below:

Ardunio Pin	Meter Key
53	1
51	2
49	3
47	4
45	5
43	6
41	7
39	8
37	BACKSPACE
35	9
33	0
31	Enter

Table 7 - Pin-to-Key Relationships

One possible concern with using relays is that they break after a certain number of switches. The HFD41 relays used in this project can switch a maximum of $1x10^7$ times [21]. Let us assume that a prepaid electricity user enters two credit tokens into his / her meter per months. Let us also assume that every key on the meter is pressed twice during a token entry. That means that every key is pressed four times a month. For a relay to switch a key $1x10^7$ times, that would then take 2500000 months or 208333 years. Thus, it can be concluded that relay switching maximums will not be a problem in this project.

(iii) Software

As mentioned in section 2.2.3, tokens can be sent from the online SMART platform to the modem by using a textbox located on the SMART platform dashboard. A screenshot of the textbox can be found below.

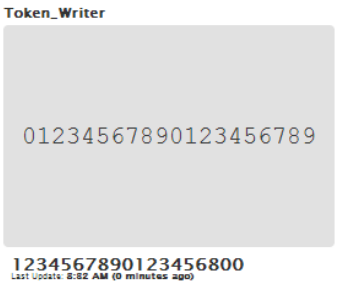


Figure 22 - Token Write Gadget

A 20-digit token is entered into this gadget and the "Submit changes" button on the dashboard is then clicked. Trintel made this gadget for specific use in this project.

When a token is sent from the online platform to the modem, it is done via a message of the following format:

+awtda: c,<ticket_ID>,"meter","Write_Token",STR,<token>,<CRC_nr>

<ticket_ID> is a unique number assigned to each job.

<token> is the 20-digit token string

<CRC_nr> is a unique number

When this message is received and sent from the modem to the Arduino, the token and ticket ID are extracted from the message and saved in two string variables. The string variable that the token is saved in is then used to write the token to the meter. This is done by driving the pin connected to the desired key to be pressed, high. Pin-key relationships can be seen in Table 7.

2.2.5 Full System Diagram

A full system diagram is included in Appendix E.

3. Test Unit and Results

A test unit was created to measure, over a period of two weeks, how well the LED sensor and data uploading to the SMART platform works. The data gathered by the test unit was uploaded to the SMART platform and displayed in the form of graphs.

(i) Hardware

The test unit consists of a LED sensor and a Sierra Wireless Airlink GL6100 modem connected to an Arduino Uno development board. Data was gathered from the LED sensor by the Arduino Uno and then sent in the form of AT commands to the modem, which in turn uploaded it to the online platform.

The LED sensor is similar to the LED sensor previously discussed in this report in Chapter 2.2.2. Serial communication with the modem was also performed in the same way as previously discussed in Chapter 2.2.3 of this report. One significant difference in the software of the LED sensor on the Arduino, is that the value at which the LED was registered as “on”, had to be lowered to a reading of 5 on the analog pin A0. The reason why the reading on the pin is so low is unclear, but the LED on the test meter (Conlog type) is probably of less intensity than the LED on the meter provided by Trintel. When the LED was tested to establish the value read on the analog pin, the Rate pin on the meter interface port (see Figure 27, Table 9) was grounded to switch the LED on. This could switch the LED on at a higher intensity than usual.

As mentioned, the Arduino Uno board was used for the test unit. This board is smaller than the Arduino Mega 2560 and has less pins and memory. It also has only one serial port, which was used to communicate with the Sierra Wireless modem. The board has six analog input pins, of which one was used to measure the LED sensor reading. The board was powered via its USB port.

Serial communication between the Arduino and the modem took place in much the same fashion as it did between the Arduino Mega 2560 and the modem. The hardware used to do the conversion from TTL levels to RS-232 levels was, however, different. The Digilent Pmod RS-232 chip was used to do this conversion. The chip is shown below:

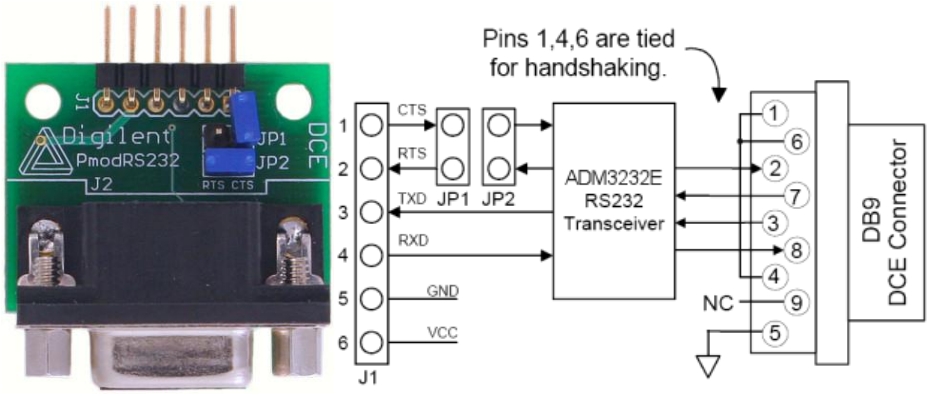


Figure 23 - Digilent Pmod RS-232

This chip converts TTL levels to RS-232 levels for serial communication. A full circuit diagram of the test unit is included below:

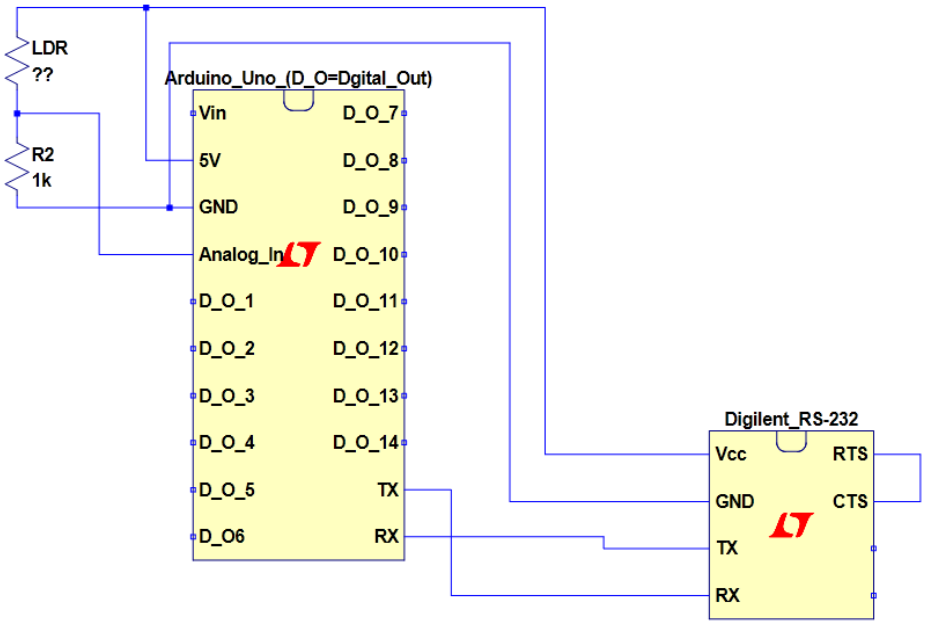


Figure 24 - LED Sensor Test Unit

(ii) Software

The test unit was setup to monitor the test prepaid electricity meter for a period accumulating to about one and a half weeks. The unit sent data to the SMART platform once every hour indicating how many watt-hours had been consumed by the meter during the previous hour. The data was then displayed on the SMART platform in a line graph. After a few days, it was noted that the unit missed about one reading every day (did not sent it to the SMART platform). This was probably due to there being no error-checking code on the unit for checking whether data was successfully written to the SMART platform. The code on the unit was changed to include error-checking and installed on the unit.

3.1 Results

Some results from the test are given below:

Device	Date and Time	Reading	Delta (kWh)
Meter LCD	21 Oct – 20:00	385.89kW	
Meter LCD	21 Oct – 22:00	384.36kW	1.53

From this table it can be seen that a total of 1.53kW was used between 20:00 and 22:00 on 21 October.

Device	Date and Time	Reading	Delta (kWh)
SMART platform	21 Oct – 20:59:17	1015Wh	1.015
SMART platform	21 Oct – 21:59:15	517Wh	1.532

From this table it can be seen that the LED sensor recorded a total of 1532Wh (1.53kW) being used between 20:00 and 22:00 on 21 October. This matches up with what was recorded by the prepaid meter.

Below are some more results:

Device	Date and Time	Reading	Delta (kWh)
Meter LCD	22 Oct – 7:00	380.35	5.54
Meter LCD	22 Oct – 8:00	378.56	7.33
Meter LCD	22 Oct – 9:00	377.99	7.90

Device	Date and Time	Reading	Delta (kWh)
SMART platform	21 Oct – 22:59:11	219	1.751
SMART platform	21 Oct – 23:59:07	305	2.056
SMART platform	22 Oct – 00:59:04	246	2.302
SMART platform	22 Oct – 01:59:01	168	2.47
SMART platform	22 Oct – 02:58:57	240	2.71
SMART platform	22 Oct – 03:58:53	240	2.95
SMART platform	22 Oct – 04:58:51	492	3.442
SMART platform	22 Oct – 05:58:47	1186	4.628
SMART platform	22 Oct – 06:58:44	919	5.547
SMART platform	22 Oct - 07:58:41	1830	7.377
SMART platform	22 Oct – 08:58:37	525	7.902

Table 8 - Test Unit Results

From these results tables it can be seen that the LED sensor is tracking the amount of watt-hours consumed very accurately.

The graph for the test-time is shown below. These graphs display watt-hours consumed over time. NOTE: The test unit was not correctly set-up until 15 October.

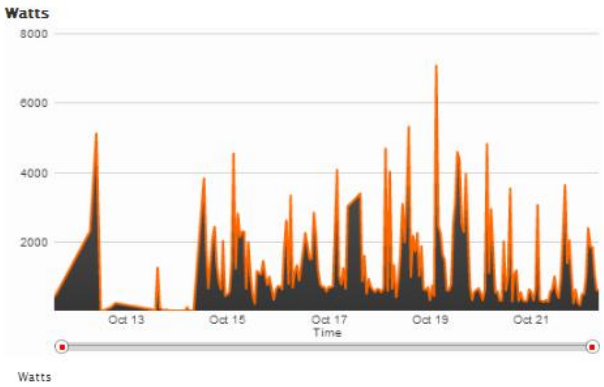


Figure 25 - SMART Test Unit Graph

4. Dead Ends

4.1 Hardware for serial communication with the meter

During the course of the project, routes to communicating other than the solution before-mentioned were attempted. Originally, it was planned to communicate serially with the meter via the interface port found on the back of the meter. A diagram of the interface port with its pinout description follows:

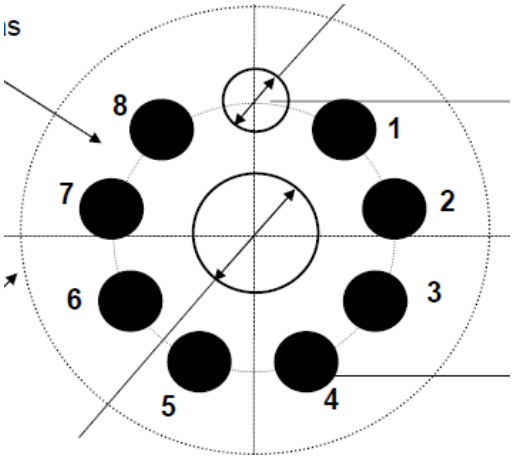


Figure 26 - Meter Interface Port

Pin number	Pin Function	Default state on reader
1	Manufacturer specific use (See A.3: Transmission Speed Option)	No connection in reader
2	Data to reader (Transmit)	Data 1
3	Data from reader (Receive)	Data 1
4	Prevent CPU reset / power down (Active low on probe.)	0 V
5	Common	0 V
6*	+5 V ± 0,2 V 60 mA max. consumption.	+5 V
7*	+15 V ± 0,5 V 60 mA max. consumption.	+15 V
8	Proprietary	No connection in reader

Table 9 - Meter Interface Port Pinout

It can be seen from the description above that the interface port has 8 pins. It was planned that the meter would be communicated with serially via the Transmit and Receive port (port 2 and 3 respectively). The Arduino's serial port 2 would be used for this communication with the meter. The communication would have to, however, go through an isolation stage as the meter's ground floats at 230VAC and the Arduino's ground is at 0V. If the two devices were to be directly connected, the Arudino would be damaged.

It was decided that a 4n25 opto-coupler would be used to isolate the meter from the Arduino. The data sent from the Arduino to the meter and from the meter to the Arduino will thus be isolated. A diagram of the isolation stage is included below:

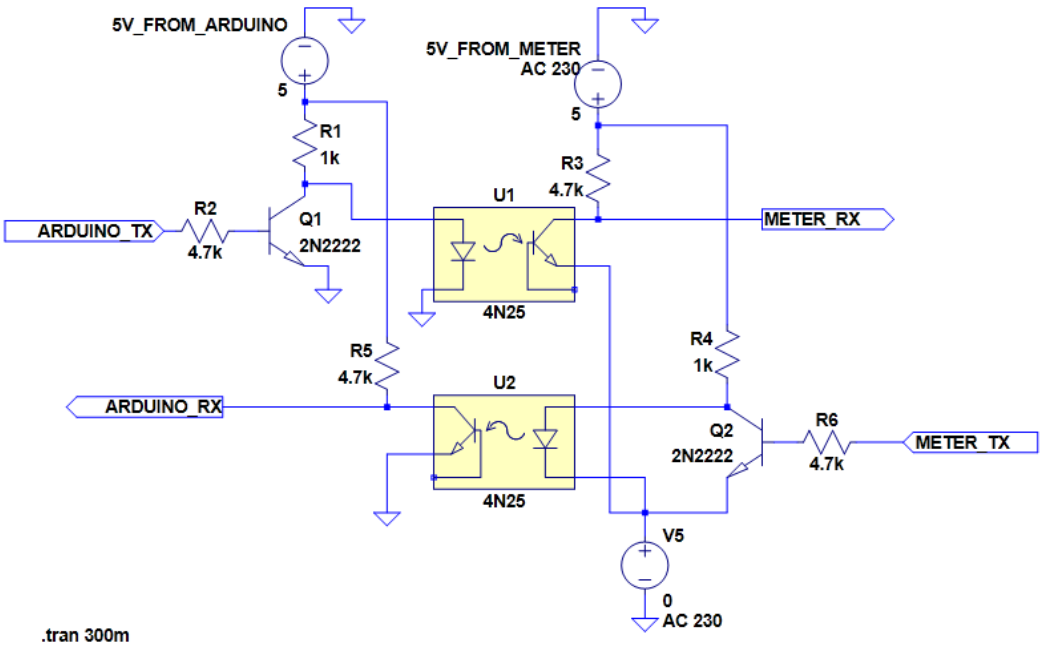


Figure 27 - Opto-Coupler Isolation

As can be seen from the diagram, 2n2222 transistors are included as a pre-stage for the opto-couplers. This is because the opto-coupler inverts any signals that are sent through it. The output stage of the opto-couplers are pulled up to 5V. On the Arduino side, this 5V is obtained from the 5V port on the Arduino board. On the meter side, the 5V is obtained from the 5V port on the interface port on the reverse side of the meter. When the LED of the opto-coupler is switched on, the opto-coupler transistor is switched on, and the opto-coupler output is pulled down to ground. This means that the signal will come out inverted on the output side. The 2n2222 transistor thus inverts the signal to be sent through the opto-coupler before it gets to the opto-coupler. Thus it comes out the right way around at the output of the opto-coupler.

The opto-coupler datasheet indicates that and ideal amount of current to send through the opto-couplers is 10mA [22]. Even though the specification supplied for the meter indicated that the 5V and 15V ports on the interface port should be able to supply 60mA [23], by experimentation it was found that when more than 10mA was drawn from the meter interface port, the meter tripped and switched off the load. The meter could not even give more than 10mA cumulatively over the 5V and 15V pins of the interface port. Therefore, less current had to be drawn from it. By experimentation using a bread board it was found that the 4n25 opto-coupler functions properly when given 5mA through the diode. Thus, the isolation stage was designed for 5mA to be given the opto-coupler diode to switch it on. The 2n2222 transistor does not need much current to switch it on, thus a 4.7kΩ resistor is used at its base, which is also connected to a source that can deliver 5V. The current through the 2n2222 is thus:

$$V = I * R$$

$$I = \frac{V}{R}$$
$$= \frac{5V}{4700\Omega}$$
$$= 1.06mA$$

A 1kΩ resistor is used to supply the opto-coupler diode with 5mA.

$$V = I * R$$
$$I = \frac{V}{R}$$
$$= \frac{5V}{1000\Omega}$$
$$= 5mA$$

The output of the opto-coupler is then pulled up to 5V using a 4.7kΩ resistor.

4.1.1 Testing the opto-couplers

To test whether the opto-couplers worked, a null-modem test was firstly applied to the opto-coupler isolation stage. For this test, an input square-wave signal with amplitude of 5V was applied to the input of one of the opto-couplers and the output of the same opto-coupler was attached to the input of the second opto-coupler. The output of the second opto-coupler was then measured. A diagram of how this circuit looked is included below:

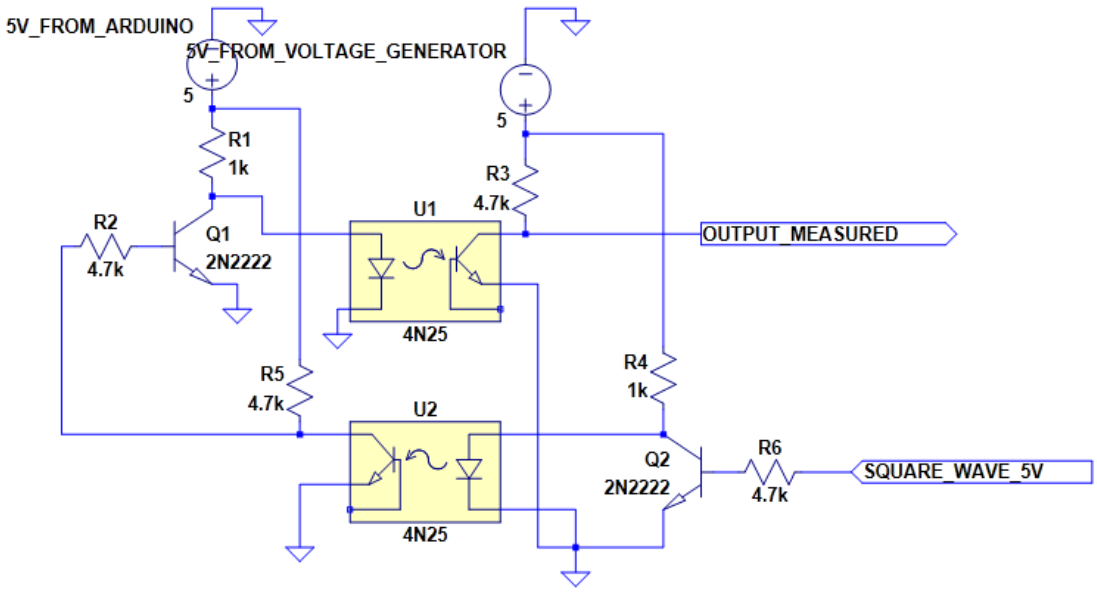


Figure 28 - Opto-Coupler Isolation Null-Modem

It was found that the output measured was similar to the input given with a slightly longer rise time. A oscilloscope screenshot of this test can be found below:

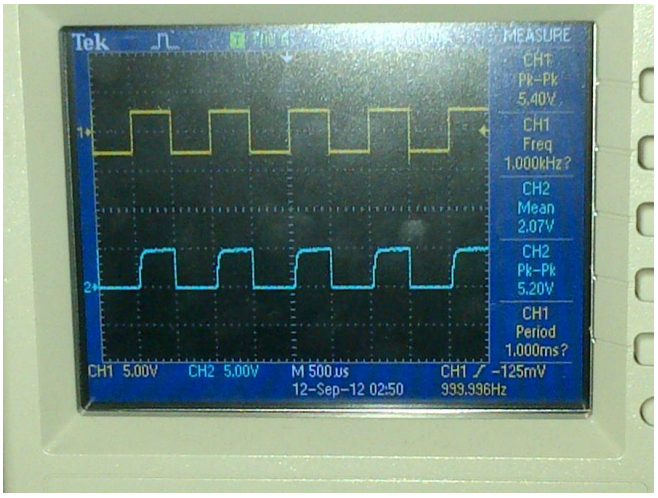


Figure 29 - Screenshot of Null Modem Test

The isolation between the meter and the Arduino was being tested when it was discovered that this approach would not work. At the time, it was found that isolation for logic signals from the Arduino to the meter worked, but the isolation from the meter to the Arduino did not work as opto-couplers kept on breaking when tested in this direction. This probably happened as a result of a voltage spike when the meter is turned on. This means that the say the 5V of the meter got to 235V (230VAC floating ground) before the meter ground got to 230V.

4.2 Serial communication with the meter

According to the specification supplied for the meter by Trintel, the meter can be communicated with serially to write data and commands to it and to read data from it. The transmission specification for the meter is contained in the following table:

Baud rate	2400 Bd
Start bits	1
Data bits	7
Parity	Even
Stop bits	1
Transmission Type	Asynchronous serial transmission – half duplex

Table 10 - Meter Serial Communication Specification [23]

The Receive and Transmit pins on the meter interface port is used to serially communicate with the meter. To serially communicate with the meter, the meter does not need to be plugged into a 230V source. 5V can be supplied to the 5V pin on the interface port and the GND pin on the interface port can be connected to ground. By doing this, the interface of the meter is switched on without turning on the source or the load. This allows for serial communication testing without having to isolate between the meter and the Arduino, and this is how testing was done in the project.

The Arduino's serial port 2 was used to communicate serially with the meter. The serial port had to be set up to the meter's transmission specification to be able to communicate with the meter (see Table 10). The Arduino

programming interface does not have a default library to modify the serial port to these standards; therefore registers in the ATmega2560 had to be modified. The register to be modified is the Control and Status Register for serial port 2, *UCSR2C*. This is an 8-bit register and a description of the register bits are tabulated below.

Bit / s	Name / s	Description
7 and 6	UMSEL21 and UMPSEL20	Select the mode of operation of the USART (Synchronous / Asynchronous)
5 and 4	UPM21 and UPM20	These bits enable and set the type of parity generation and check.
3	USBS2	This bit selects the number of stop bits.
2 and 1	UCSZ21 and UCSZ20	These bits set the number of data bits.
0	UCPOL2	Used to set clock polarity (in synchronous mode only, set to 0 in asynchronous mode)

Table 11 - Control and Status Register Description

To setup the serial port up to the meter's transmission specification, the register has to look like the following:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	1	0	0	1	0	0

Figure 30 - Desired Control and Status Register Setup

The following code is used in the Arduino programming program to set the register.

```
UCSR2C = UCSR2C | B00100100;
UCSR2C = UCSR2C & B00100101;
```

When the test setup had been done (5V to the 5V pin, Ground to the GND pin, Arduino TX (Serial port 2) to the meter Receive pin and Arduino RX (Serial port 2) to the meter Transmit pin), data was sent to the meter to test whether communication works. The meter specification indicates that the following message can serially be sent to the meter to test serial communication:

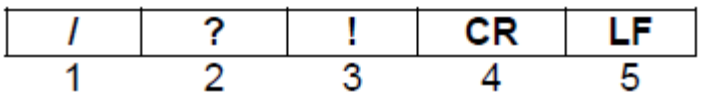


Figure 31 - Identification Message [23]

The message is an identification request, requesting for the meter to "identify" itself. A description of the characters contained in this message and their functions are tabulated below (see Appendix D for full ASCII table):

Character	Description	Function
/	ASCII '/'	Start character
?	ASCII '?'	Identification request command
!	ASCII '!'	End character
CR	ASCII carriage return	Completion character
LF	ASCII line feed	Completion character

Table 12 - Identification Message Character Description [23]

The meter should then respond with a message in the following format:

/	M	X	X	V	V	V	V	CR	LF
1	22	6	6	7	7	7	7	4	5

Figure 32 - Identification Response [23]

A description of the characters contained in this message and their functions are tabulated below:

Character	Description	Function
/	ASCII '/'	Start character
M	ASCII 'M'	Indicated that the manufacturer details follows in the next 6 characters
Xs	2-digit hexadecimal code	Manufacturer code
Vs	4-decimal hexadecimal code	Software version number
CR	ASCII carriage return	Completion character
LF	ASCII line feed	Completion character

Table 13 - Identification Message Character Description [23]

In the project, the identification message was sent to the meter. The following dialogue between the meter and the Arduino took place:

Sent from Arduino:
/?!<CR><LF>
Response form meter:
/M011101

This shows that serial communication between the meter and the Arduino is indeed possible. The next step in communication was to read and write data from and to the meter serially. The meter specification indicates that to read data from the meter, a message of the following form must be sent to it:

SOH	R	STX	ADDR	DL	ETX	BCC
8	9	10	11	12	13	14

Figure 33 - Data Read Message [23]

To write data to the meter, a message of the following format should be sent.

SOH	W	STX	ADDR	(	D	)	ETX	BCC
8	15	10	11	16	17	18	13	14

Figure 34 - Data Write Message [23]

The characters contained in these messages and their functions are tabulated below:

Character	Description	Function
SOH	ASCII start of header	Header character
R	ASCII 'R'	Read command
W	ASCII 'W'	Write command
STX	ASCCI start of text	Frame start character
ADDR	4-digit hexadecimal	Address field indicating from which address data must be read from or written to.
DL	1-digit hexadecimal	Data length. Indicated number of bytes to be read from address specified. Maximum data length is 10 bytes.
ETX	ASCII end of text	Frame end character
BCC	Block check character	Logical XOR of the characters starting with the first character following the first SOH or STX and up to and including the ETX character that terminates the message frame.
(	ASCII left parenthesis '('	Open data block
)	ASCII right parenthesis ')'	Close data block
D	Data	Data field representing data to be written or read. The maximum data length is 10 bytes (20 digits in hexadecimal format)

Table 14 - Data Read Message Character Description [23]

This message should allow a user to read from a specified address in the virtual memory of the meter. The specification supplied by Trintel includes a description of what the virtual memory of the meter should look like. This layout can be found in the specification document [23] and is also included in Appendix F. According to this layout, data containing the Total Cumulative Units consumed is contained in the memory at virtual address 4019h (hexadecimal).

The specification document gives the following flow diagram to indicate how communication flow should take place:

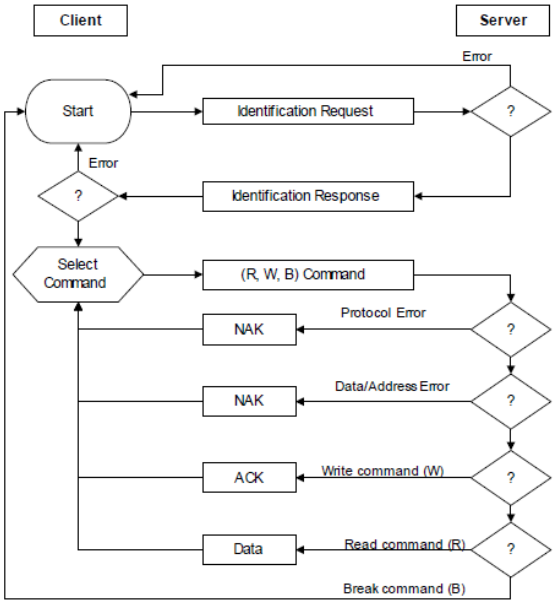


Figure 35 - Meter Communication Protocol Flow [23]

According to this diagram, in the project it had already been tested and found that the identification request and response worked. The next step was to send a Read or Write command to the meter. The following write command was sent to the meter:

Bit 1	**Bit 2**	**Bit 3**	**Bit 4**	**Bit 5**	**Bit 6**	**Bit 7**	**Bit 8**	**Bit 9**	**Bit 10**
SOH	W	STX	ADDR	ADDR	(	D	D	D	D
Bit 11	**Bit 12**	**Bit 13**	**Bit 14**	**Bit 15**	**Bit 16**	**Bit 17**	**Bit 18**	**Bit 19**	
D	D	D	D	D	D	)	ETX	BCC	

Where the D bits are data bits.

Hex:

Bit 1	**Bit 2**	**Bit 3**	**Bit 4**	**Bit 5**	**Bit 6**	**Bit 7**	**Bit 8**	**Bit 9**	**Bit 10**
0x01	0x57	0x02	0xff	0xff	0x28	0x01	0x01	0x01	0x01
Bit 11	**Bit 12**	**Bit 13**	**Bit 14**	**Bit 15**	**Bit 16**	**Bit 17**	**Bit 18**	**Bit 19**	
0x01	0x01	0x01	0x01	0x01	0x01	0x29	0x03	0x029	

(NOTE: The address is set to 0xFFFF when writing a token to the meter [23]). When this command was sent to the meter, no response came back from the meter. According to Figure 36, the meter should return a NAK character, even if the message sent to the meter is incorrect. After quite a few attempts

to send a write message to the meter without response, it was concluded that the specification provided is incorrect in accordance with this command.

As the specification is incorrect, serial communication can be established with the meter as tested, but the desired application of reading data from the meter and writing data to it would not be possible serially. Thus, another approach had to be found and followed, which is the point where the project methodology changed to using the LED sensor and relay matrix to gain the desired information from the meter and to write tokens to the meter.

5. Conclusion and Recommendation

This project aimed to monitor and control a prepaid meter. This was to be done by gathering data from the prepaid meter and displaying it in an effective manner that is accurate and easy to analyse, as well as controlling the meter by being able to send it various instructions to perform.

The original approach was to do this by serially communicating with the meter and doing most of the data gathering and sending digitally. This approach could not be followed as the specification provided for the meter was not accurate and after attempting to communicate with the meter, it was found it would not be possible.

Alternative approaches were considered and it was chosen to use an LED sensor to gather data concerning electricity usage of the meter from the meter. This approach was implemented and tested and found to work accurately and consistently.

A relay matrix was used to write STS tokens to the meter. These tokens allow for control of the meter and the use of a relay matrix in partnership with a microcontroller allows for remote control of the meter from the Internet.

This project benefits users in that it makes it possible to remotely monitor and control a prepaid meter. This adds to the convenience in the use of prepaid meters as it is not necessary to physically be at the position of the meter to upload credit to the meter or to monitor credit available or electricity usage of the meter. Monitoring of electricity usage can also be beneficial in providing users and distributors with information that could lead to electricity savings.

As a recommendation for future work, a proper specification for these prepaid meters can be obtained and the meter can be communicated with serially to gather data concerning electricity usage and the meter can also be controlled serially.

Also, according to Vincent van der Clis, employee of Appframeworks, STS tokens can be developed to allow for new instructions to be written to meters through them. The only prerequisite is that meter firmware must be updated to support this. This can be investigated as an option to exercise control over a meter [25].

A demonstration video for the project can be found at: http://youtu.be/uH-bfwv4Dxo

References

[1] NERSA, “NERSA Consultation Paper: Power Conservation Programme (PCP) Rules”, Dec. 2008.

[2] A. Jain, M. Bagree. “A prepaid meter using mobile communication” *International Journal of Engineering, Science and Technology*, vol. 3, pp. 160-166, April 2011.

[3] SEM, “SEM’s Automatic Meter Reading (AMR) System”, Internet: http://www.semsolutions.co.za/automatic-metering.html, Oct. 2012.

[4] OWL, “OWL Wireless Energy Monitor”, Internet: http://www.theowl.co.za/CM119_userguide.pdf, Oct. 2012.

[5] L. Kolver. “Prepaid electricity available instantly through sms”, Internet: http://www.engineeringnews.co.za/article/prepaid-electricity-available-instantly-via-sms-2009-07-10. Jul. 17, 2012.

[6] Itron, “ACE9000 Taurus ISP”, Internet: https://www.itron.com/mxca/en/PublishedContent/F29988-4090-ACE9000-ISP-SA_v5_LOW.pdf 2009.

[7] Trinity. “SMART Sense Telemetry Management Solution”, Internet: http://www.trintel.co.za/telemetry_management.html. Oct. 2012.

[8] T. Pawley, M. Weiss, W. Hayes. “Trintel Training Session”, 3 October 2012.

[9] Sierra Wireless Inc. “Sierra Wireless AirLink™ Programmable Gateways Fastrack Xtend and GL Series”, Available from: http://www.sierrawireless.com/productsandservices/AirLink/Programmable_Modems/GL6100_GL6110.aspx. 2010

[10] M. Ueland. “What Are AT Commands?”, Internet: http://mobiledevdesign.com/tutorials/what-are-at-commands-070110/, Jul. 1, 2010.

[11] P. Torrone (May 12, 2011). “Why Google Choosing Arduino Matters and Is This the End of “Made for iPod” ™?”. Makezine.com. Retrieved Jan. 1, 2012.

[12] “Arduino Introduction page”, Internet: http://www.arduino.cc/, Oct. 12, 2012.

[13] Shiffman, Daniel (September 23, 2009). "Interview with Casey Reas and Ben Fry". Rhizome.org.

[14] “Arduino Mega 2560”, Internet: http://arduino.cc/en/Main/ArduinoBoardMega2560, Oct. 12, 2012.

[15] “Resistive_divider.png”, Internet: http://upload.wikimedia.org/wikipedia/commons/d/db/Resistive_divider.png . March. 9, 2008.

[16] “Arduino Analog Input Tutorial”, Internet: http://arduino.cc/en/Tutorial/AnalogInput, Oct. 13, 2012.

[17] Jimbo. “RS-232 vs. TTL Serial Communication”, Internet: http://www.sparkfun.com/tutorials/215. Nov. 23, 2010.

[18] Sierra Wireless. “GL61x0 Product Technical Specification and User Guide”, Nov. 30, 2010. pp. 33.

[19] Maxim Integrated Products. “Typical Operating Circuit”, in MAX3322E-MAX3323E, Rev. 1, Jan. 2003.

[20] Sierra Wireless. “AT Commands Interface Guide for Firmware 7.46”, Aug. 2, 2011.

[21] Hongfa Relay, “HFD41/D41A”, Rev. 1. 2008.

[22] Isocom Components. “Optically Coupled Isolator Phototransistor Output”, Jul. 30, 1997.

[23] J. O’Kennedy. “Particular Requirements for Prepayment Meters”. Rev. 2. Aug. 2008.

[24] R. De Kock. “Prepaid meter Skripsie”, Personal e-mail (12 July 2012).

[25] V. van der Vlis. “RE: Tokens”, Personal e-mail (11 October 2012).

[26] Maxim Integrated Products. “MAX3322E-MAX3323E, Rev. 1, Jan. 2003.

Appendix A: Planning Schedule

Week	Plan	Work Done
1 (Week of 19 July 2012)	Read and understand meter specification	Study of meter specification
2 (Starting Monday 23 July 2012)	Read and understand meter specification and plan how to communicate	Study of meter specification, and tested to find meter does not work.
3	Get meter up and running and understand meter interface. Purchase microcontroller. Trintel training.	New meter obtained that worked. Purchased microcontroller. Attending Trintel training.
4	Test Week	Test Week
5	Understand microcontroller and C programming.	Understood and did modem communication and basic SMART platform operations.
6	Do isolation between meter and microcontroller. Understand modem and protocol	Decided on opto-couplers for isolation. Obtained opto-couplers.
7	Get microcontroller to communicate with meter and modem.	Tested opto-couplers. Obtained MAX2323 chip for communication between modem and microcontroller. Read and understood MAX232 datasheet.
8	Capability to write information to meter and read data from meter.	Isolation problems encountered. Continual opto-coupler testing.
9	Capability to write information to meter and read data from meter.	Found out meter specification is wrong. Planned new, more mechanical approach to gather data from the meter and control the meter.
10	Program the software system for the microcontroller to effectively gather and analyse data, and control the meter.	Built LED meter for gathering data and programmed software.
11	Understand and setup Trintel platform	Built opto-coupler matrix for entering tokens via meter keypad.
12	Field Testing	Tested opto-coupler matrix, found faulty.
13	Tie up loose ends	Obtained relays and build relay matrix to control meter keypad.

14	Tie up loose ends	Tested relay matrix.
15	Tie up loose ends	Wrote Report and field testing.
16	Write Report	Wrote report
17	Write Report	Exams

Appendix B: Project Specification

The specification for this project is to develop a web-based interface for the monitoring and control of a prepaid electricity meter.

The specification required that the meter must be communicated with. Data concerning electricity usage must be gathered from the meter and a user must be able to send instructions to the meter. The data gathered must then be displayed on Trinity's online SMART platform in a user friendly manner. A user must also be able to send instructions to the meter from this same SMART platform.

A modem was supplied by Trintel for this project. This modem must be used to facilitate communication between the meter and the online SMART platform. The meter to be in the project was also supplied by trintel.

An Arduino microcontroller was used to control the system. Data is gathered from the meter using an LED sensor to record how many watt-hours the meter consumed. This sensor is monitored by the microcontroller, which records how many watt-hours are consumed by the meter. Instructions are sent to the meter in the form of STS tokens, which are usually entered into the meter via the meter keypad. These tokens are written to the meter using the microcontroller and a relay matrix. The relay matrix is used to emulate key presses on the meter keypad.

A centralised point in created online where data gathered from the meter can be viewed and tokens can be written to the meter.

Appendix C: Outcomes Compliance

Problem Solving:

- Changing the solution approach when it was found meter specification was inaccurate *(Chapter 4, Chapter 2.2.2 and 2.2.4)*
- The approach to gathering data from the meter *(Chapter 2.2.2)*
- The approach to writing tokens to the meter *(Chapter 2.2.4)*
- Design of error checking software that ensures data is correctly sent to the modem *(Chapter 2.2.3)*

Application of scientific and engineering knowledge:

- The LED sensor. Design of an effective sensor. *(Chapter 2.2.2)*
- The relay matrix. Designing a working relay matrix to emulate key presses on a keypad *(Chapter 2.2.3 (ii))*
- Risk of relays breaking because of maximum switching limit *(Chapter 2.2.4)*
- Interpretation of test results *(Chapter 3)*

Engineering design:

- Solution design *(Chapter 1.4)*
- LED voltage profile *(Chapter 2.2.2. (i))*
- Choosing between opto-couplers and relays for writing tokens to the meter *(Chapter 2.2.4).*

Investigations, experiments and data analysis:

- Choice of microcontroller to use *(Chapter 2.2.1)*
- LED sensor field testing and analyses *(Chapter 3)*
- Literature study *(Chapter 1.2)*
- Conclusions about project *(Chapter 5)*

Engineering methods, skills and tools, including Information Technology:

- C programming for the solution system. This includes data handling, string manipulation, error checking and a state machine *(Chapter 2.2.3)*
- Setting up of the online SMART platform for use of the project *(Chapter 2.1.2)*

Professional and technical communication:

- This report is a partial fulfilment of this requirement. It is a communicating the project in a written form. An oral on the project will also be presented.

Independent learning ability:

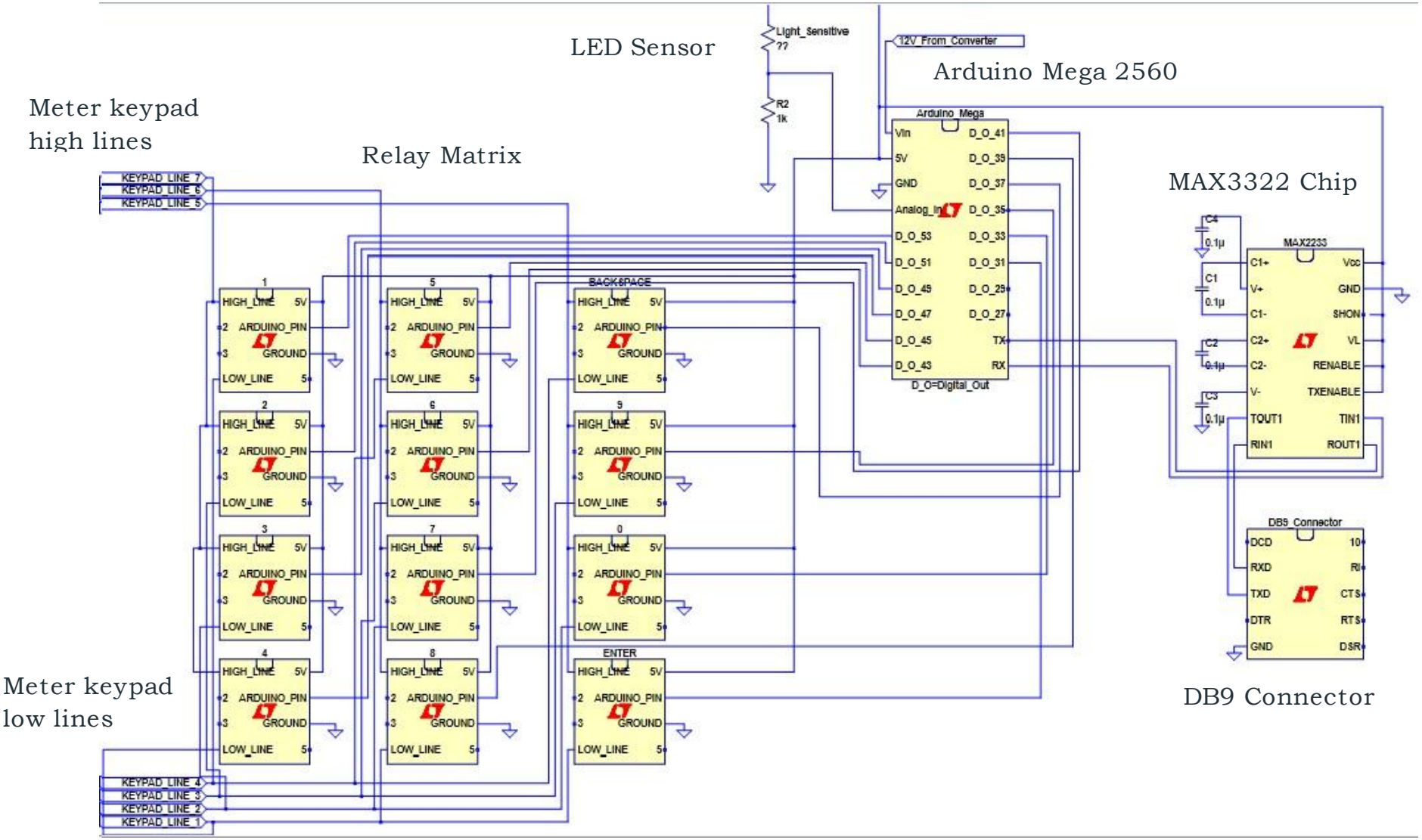
- Understanding and studying the prepaid meter provided. Also, drawing conclusions from this study *(Chapter 2.1.1, Chapter 4)*
- Understanding and effectively using the modem provided *(Chapter 2.1.3, Chapter 2.2.3)*
- C programming for the system *(Chapter 2.2.3)*
- Understanding of various components and using them in the solution components of the system *(Chapter 2.2.2, Chapter 2.2.3, Chapter 2.2.4, Chapter 4)*
- Understanding and effectively using the online SMART platform *(Chapter 2.1.2, Chapter 2.2.3)*

Appendix D: ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Appendix E: Full System Diagram



Appendix F: Meter Virtual Memory Configuration

Address	Virtual address	Context	Bytes	Ref
0000h - 3FFFh	*Manufacturer-specific*	Reserved for manufacturer-specific addresses.	X	A.4.4
4000h	VTC Version	Version of the VTC standard;	1	A.4.5
4001h	Virtual Address Map Version	Version of this standard virtual address map;	1	A.4.6
4002h	Meter Number	As defined in as defined in Annex A of NRS 009-6-7; Format: 14-digit hexadecimal, left pad with zero Data returns in standard sequential format with most significant byte first.	7	
4003h	Software Version	The 4 digit code as defined in field number 7 of the Message Content. Format: 4-digit hexadecimal (manufacturer specific) Data returns in standard sequential format with most significant byte first.	2	A.3.2
4004h	Primary Token Carrier Type	As defined by Token Technology in NRS 009-4-1. The main user token carrier interface normally used for credit token transfers. (type 07 defined as Virtual Token Carrier Type) Format: 8-bit binary	1	
4005h	Encryption Algorithm	As defined in NRS 009-6-7; Format: 8-bit binary	1	
4006h	Read VTC remaining lockout period	The remaining time in seconds that the token carrier interface will still remain in lockout status; Format: 16-bit binary Data returns in standard sequential format with most significant byte first.	2	6.5.6
4007h - 4010h	*STS Reserved*	Reserved for future STS defined use	X	
4011h	Tariff Index	As defined in NRS 009-6-7; Format: 8-bit binary	1	
4012h	Key Revision and Key Type	As defined in NRS 009-6-7; Format: 2 digit hexadecimal; Key Revision Number (MSN); Key Type (LSN)	1	6.5.1, 6.5.4
4013h	Key Expiry Number	As defined in NRS 009-6-7; Format: 8-bit binary	1	
4014h	Maximum Power Limit	As defined in NRS 009-6-7; Format: 16-bit binary; 0 = not enabled Data returns in standard sequential format with most significant byte first.	2	6.4.1
4015h	Maximum Phase Power Unbalance Limit	As defined in NRS 009-6-7; optional Format: 16-bit binary; 0 = not supported Data returns in standard sequential format with most significant byte first.	2	

4016h	Tariff Rate	As reserved in NRS 009-6-7; only currency meters	X	
4017h	Water Meter Factor	As defined in NRS 009-6-7; only water meters Format: 16-bit binary; 0 = not supported Data returns in standard sequential format with most significant byte first.	2	
4018h	Available Credit	Available credit as reflected by the accounting function in the meter; Format: 32-bit signed integer Data returns in standard sequential format with most significant byte first.	4	6.3.4.3
4019h	Total Cumulative Units consumed	The value of the register that records the cumulative kWh delivered by the meter; Format: 32-bit signed integer Data returns in standard sequential format with most significant byte first.	4	
401Ah	Last Credit Token	The value of the last credit token that was accepted by the meter (not the token with most recent token ID); Format: 20-digit hexadecimal Data returns in standard sequential format with most significant byte first.	10	
401Bh	Last Credit Token ID	The Token ID of the last credit token that was accepted by the meter; Format: 24-bit binary Data returns in standard sequential format with most significant byte first.	3	
401Ch	Tamper Status	Optional - if supported by the meter; Format: 16-bit binary 0 = not in tamper state Bit 1 = in tamper state (set by trigger signal, cleared by STS token) Bit 2 = bypass detected (set and cleared automatically by meter) Bit 3 = consumption irregularities detected (set and cleared automatically by meter) Bit 4 – 8 reserved Bit 9 – 16 manufacturer specific use	2	6.1.7
401Dh - 8FFCh	STS *Reserved*	Reserved for future STS defined use ***Start of Writeable / Scratchpad area***	X	
8FFDh	Latitude (Writeable address)	Optional - if supported by the meter; Format: 10-digit hexadecimal in format XDDDmmss.ss (decimal point implied) X = 0 for "+" (North); X = 9 for "-" (South) DDD = Degrees, mm = minutes, ss.ss = Seconds Data in standard sequential format with most significant byte first.	5	A.4.3
8FFEh	Longitude (Writeable address)	Optional - if supported by the meter; Format: 10-digit hexadecimal in format XDDDmmss.ss (decimal point implied) X = 0 for "+" (East); X = 9 for "-" (West) DDD = Degrees, mm = minutes, ss.ss = Seconds Data in standard sequential format with most significant byte first.	5	A.4.3
8FFFh	Supply Group Code (Writeable address)	Not directly linked to the meter key. Simply a label that can be updated via the VTC; The meter shall automatically overwrite this label with 000000h when a key change is performed; Format: 6-digit hexadecimal Data in standard sequential format with most significant byte first.	3	A.4.3
9000h – FFFDh	*Manufacturer specific*	Reserved for manufacturer-specific use ***End of Writeable / Scratchpad area***	X	A.4.4
FFFEh	Read Token Status	The server interprets the receipt of this value in the address field as indicating that the client wishes to "read" the value of the Token Status, being the result from the meter after executing the entered token. Format: 8-bit binary 0= reserved 1 = token is rejected; 2 = token is accepted; 3 = token has already been used (duplicate); 4 = token is old (expired); 5 = Meter key has expired; 6 = Token Lockout is active; 7 to 255 reserved	1	A.4.2
FFFFh	Write STS Token (Writeable address)	The server interprets the receipt of this value in the address field as indicating that the included data is a 20-digit STS token that is being "entered" into the meter. Format:20-digit hexadecimal with the most significant digit positioned adjacent to the ADDR field	10	A.4.1