Final Report

ECM3101

Title: ONLINE ENERGY MONITORING SYSTEM

Date of submission: 02/05/2013

Student Name: Pablo Trigo-López
Programme: ERASMUS
Student number: 620034340
Candidate number: 035376

Supervisor: Dr. Abel Nyamapfene

List of figures

List of tables

Abstract

Nowadays reducing the energy consumption is fundamental either for environmental (climate change) or economic reasons. Monitoring energy is the first step to achieve this objective. With all the technology that we have at our reach, it is really disappointing that there is not a product at every house to monitor the energy we use.

A fairly recent study by CenterPoint Energy Inc. and the Department of Energy [1] found that 71% of customers reported changing their energy consumption as a result of having access to energy data in their homes. Another energy-saving campaign [2] conducted in Sabadell, Spain, reported savings of the 14.3% in the bill. Thus, it can be deduced that it is something that clearly helps the user.

The main object of this project is designing a system capable of measuring the power consumed by a device and send the data over the internet, plotting it in a friendly graph that everyone can understand. Uploading it to the Internet is an easy way to catalogue the energy consumption since the first moment you start measuring it. At the same time information can be accessed from any fix or mobile browser instead of depending on a specific one.

The main hardware used for this project was the Arduino Ethernet, responsible for collecting the power data. As for the software, *MySQL* database was used for data logging the energy consumption, *PHP* language was needed to send the data collected from the Arduino board to the local web server that was implemented with *Apache* software and *JavaScript* was used to design the friendly graphs.

The methodology followed in this project could be divided in three steps. First, design a sensing system and program the code to capture the current and voltage with the Arduino micro-processor. Second, implementing a method to send the data collected to an online database. And finally and only if this was completely achieved, go further and design a script to graph this data. Everything would be independent for third-party websites. Managing your own website gives more freedom as the project does not depend in other applications that may be outdated in a future.

The principal aim of the project of plotting a graph of the energy consumption over the internet was successfully achieved. Besides, the sensing circuits for the current and the voltage were theoretically designed and simulated.

Keywords: Energy monitoring – Arduino – Hall Effect sensor – Online Database – *PHP* – *JavaScript* graphs

Table of contents

1. Introduction

With the recent tendency for green energy, more and more people are concerned about how much energy they consume at their homes. To reduce your consume you have to be able to monitor your energy consumption.

In this project a system capable of measuring the power consumed by a device was designed. This system sends the data over the internet, plotting it in a friendly graph that everyone can understand.

It is specially focused on the online feature implemented with a non-third-party application what makes the system completely independent. The main aim is displaying useful information about the energy consumed to the home-user.

The project consists of a micro-controller device (Arduino) connected to the local area network (LAN) that is able to measure the amount of power being used and graph the data in real-time in a personal website using a script developed with *JavaScript*. For the full development of this system it knowledge of *PHP* as well as *JavaScript* were required.

The system for measuring the current and voltage was designed theoretically and simulated, but for the experimental demonstration of this project a signal-generator simulating current and voltage waveforms were used.

In the development of this project, instead of using an external web server a local web server was used. Therefore, the website with the graphs could only be accessed by the devices connected to the LAN.

This document will clarify how this system was implemented. First, a brief background on the subject will be provided, followed by the methodology used in the project and the main design of the system. Finally, in the last chapters, the main results and a brief conclusion will be available to summarise the project.

2. Health and safety risk assessment

This project is related with the energy. Measuring the energy consumption requires direct contact with the mains. Although they are other risks, they are less dangerous or its likelihood is lower. Other risks could be producing a short circuit, a leakage current or inhaling gases during possible soldering.

A risk assessment table was elaborated to take into account all the possible risks present in the elaboration of this project. The following table 1 would be considered to evaluate each of the actual risks, and table 2 shows the risk management.

Table1. *Hazard and risk assessment criteria*

Risk Rating (RR)

Severity (S)				
	3	3	6	9
	2	2	4	6
	1	1	2	3
		1	2	3
		Likelihood (L)		

9	Unacceptable risk
4-6	Only if no other method viable and with high level controls
2-3	Acceptable if suitable controls
1	Acceptable, no further action required

Key: S = Severity rating
L= Likelihood of occurrence
RR = Risk Rating

Table2. *Risk Management*

Activity	Hazards/Risks	Pre-control risk rating			Control measures	Residual risk rating		
		S	L	RR		S	L	RR
Measuring power	Electric shock	3	2	6	Use a low powered circuit that simulates the main lines	1	1	1
Signal conditioning for the micro-controller	Short circuit. Component damage	2	2	4	Draw schematic circuit before assembling the circuit.	2	1	2
Welding	Inhaling noxious gases	1	2	2	Avoid bringing your face close to the welding	1	1	1

3. Background

The idea of this project was inspired by several previous projects whose main aims were to monitor the energy consumption within specified premises.

Cliff Jao and Xi Guo from Cornell University have already managed to monitor the energy consumed by a specific device with their project "PowerBox" [3]. This "box" allows the user to measure the amount of AC power being used by a device connected to the wall socket. Data is graphed by a C# application. The math used by these students was taken into account to develop this project.

In addition, another two students from Cornell University, Ken Bongort and Adam McCann developed and interesting project called "XBee RF Smart Energy Compliant Power Meter" [4]. This project completed Cliff Jao and Xi Guo work by adding RF communication to the system, endowing the project with flexibility. This idea was initially taken into account for this project, but due to lack of time it was not implemented in this project.

"PowerGoogle meter" [5] was also a source of inspiration. Regrettably this project was retired on 2011. As they asserted in their website, "PowerMeter included key features like visualizations of your energy usage, the ability share information with others, and personalized recommendations to save energy". The feature of sharing information with others was considered for this project prompting into the online feature of this project.

And finally "openenergymonitor project" [6] was the main source of background information. It is a project to develop open-source energy monitoring tools to help the user relate to his use of energy. It uses the Arduino platform [7] that will be later explained. The final data is uploaded into a third-party websited named as Cosm [8]. This project served as a source of information about sensors and tips about programming code for the Arduino board. Besides, Cosm website served as an intermediate step for the developing of this project: before designing a personal website, data was uploaded to this site using specific libraries provided by this website.

There are also available in the market products like "Kill A Watt" [9] whose main function is displaying the power consumed in a LCD screen.

Considering all these previous works, this project tries to unite all of them, specially focusing on the online feature but with a non third-party website that makes you completely independent. Projects like [4-6] are extremely dependent on third-party applications. They have a due date. If these third-party applications expire (like happened with [5] and therefore with [4]) then the project becomes obsolete. Using your own server and developing your own script to graph the data solves this problem.

4. Methodology

To achieve the main objectives of this project an elaborated methodology had to be followed. A largely used methodology used in engineering consists in dividing a problem into sub-problems. This methodology has great advantages:

- Solving little problems is easier than solving large ones.
- Each problem can constitute a specific task of the whole problem. This is an effective way of organizing the work.
- Independent modules can be faced in different times or by parallelism.

Taking into account this methodology a modular design approach is to be adopted for this project. It could be divided in three main modules or tasks:

- **Measuring Power:** Designing and programming a system to capture the current and voltage level of a device or net. This implies researching for the best methods to sense these electric magnitudes as well as investigating how to program an effective, modular and reusable code to calculate the real power consumed with the previously captured data.
- **Receiving and storing data:** Investigating and elaborating an accurate procedure to send this data over the internet to be stored in a database.
- **Graphing data:** Developing successful scripts to graph the data stored in the earlier mentioned database.

Each of these three major steps required a deep investigation that would be meticulously explained in the following pages. The steps followed to reach the final result in each of the tasks mentioned above would be clearly explained. As in any project, during its elaboration some difficulties may have appeared. If that is the case, these problems will be cited and it will be described how they were fruitfully solved.

This chapter is an outline of the design process, which led to the development of the final system. However, specific details of the design will be further discussed in the Design chapter (chapter 5).

4.1 Measuring Power

There are plenty of techniques to measure the power of a system. As it is known, in general terms the power consumed by a system could be expressed like the product of the voltage and the current flowing in a circuit in a specific lapse of time.

$$P(t) = v(t) \cdot i(t) \tag{1}$$

In the case of an AC circuit, the magnitudes of this product are sinusoidal waves.

$$v(t) = |V| \cdot \cos(\omega \cdot t) \tag{2}$$

$$i(t) = |I| \cdot \cos(\omega \cdot t - \theta) \tag{3}$$

Where $'|V|'$ and $'|I|'$ represent the module of the voltage and current respectively, $'\omega'$ represents the frequency, $'\theta'$ indicates a time delay between the two waves or a phase gap and *'t'* is the time.

Therefore, to measure the power of a system the voltage and the current must be sensed. The following sections would explain the methods for doing it.

4.1.1 Current sensing. Hall Effect sensor

The three most common techniques to measure current are [10]:

- Low resistance current shunt
- The current transformer (CT)
- The Hall effect sensor

There is also a method used in industrial applications: The Rogowski coil method. The basic operating principle of a Rogowski coil is to measure the primary current through mutual inductance. Because Rogowski coil relies on measuring magnetic field, it makes this type of current sensor susceptible to external magnetic field interference comparing with the CT. Due to the conditions of the laboratory, where a breadboard will be used; there would be a lot of interference that would make the Rogowski coil less effective.

The low resistance current shunt requires inserting a small resistor through the main power line, as figure 1 shows:

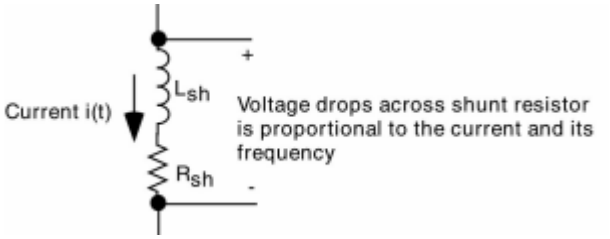


Fig.1 *Low resistance current shunt [10]*

Resistive sensing is very widely used, low-cost, and easily understood. It offers a good accuracy and it is really simple and economic (the parasitic inductance is only

considered for high precision current measurement, not in this case). However, the shortcomings are its insertion loss (heating and wasted power) and lack of isolation [11]. It is an "intrusive" technique since you have to manipulate the main power wire. Due to its high risk (direct contact with main lines) this solution is completely ignored. For this project it is desirable a non-contact sensor that protects the user.

The Current Transformer or CT is a transformer that converts the primary current into a smaller secondary current. It is very common in house energy monitoring. The main problem is that it can get saturated at high current. As Paul Emerald asserts, "Current Transformers close out the last low-cost technology, and (as the term transformer should imply) are only useful with alternating currents. Most low-cost current transformers are designed for narrow frequency ranges, are more expensive than resistive or Hall-effect" [11].

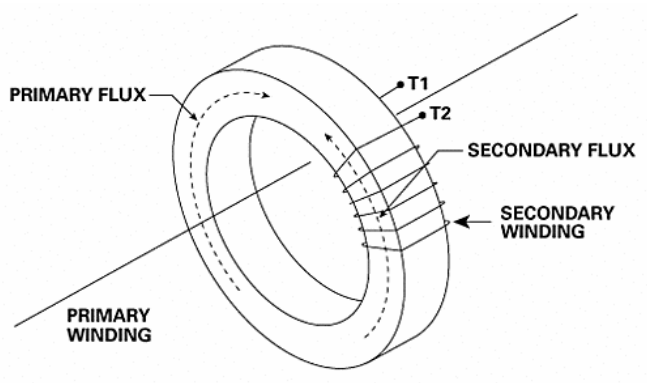


Fig.2 *Current Transformer [12]*

Thus, although they seem a good solution, they are limited to AC current and for a good frequency response they are more expensive than Hall Effect sensors. For making this project as much useful as it can be, DC measurement may also be considered. Hence, the Hall Effect sensor will be studied before making the last decision.

Finally the Hall Effect sensor is a transducer that varies its output voltage in response to changes in magnetic field. The Hall element is constructed from a thin sheet of conductive material with output connections perpendicular to the direction of current flow. When a magnetic field is applied, a voltage proportional to this field appears in the output. *Honeywell* explains that "The voltage output is tiny (μV) and requires additional electronics to achieve useful voltage levels. When the Hall element is combined with the associated electronics, it forms a Hall Effect sensor" [13].

It can be used to sense current considering Faraday-Lenz law, as an electric current generates a magnetic field around a conductor.

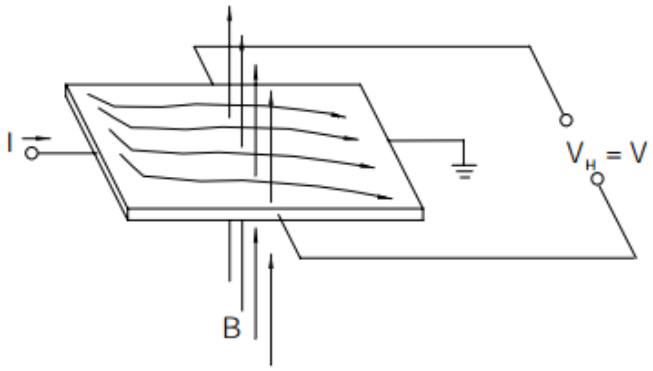


Fig.3 *Hall Effect principle [13]*

Figure 3 illustrates the basic principle of the Hall Effect. There is a thin sheet of semiconducting material through which a current is passed. When a perpendicular magnetic field is present, a Lorentz force is applied on the current. This force perturbs the current distribution, resulting in a potential difference across the output. This voltage is the Hall voltage (VH). The relation between the magnetic field and the current is shown in equation (4)

$$V_H \propto B \times I \tag{4}$$

Integrated Hall Effect transducers

Making a Hall transducer out of silicon, using standard integrated circuit processing techniques, allows one to build complete sensor systems in a chip. The addition of electronics to the bare transducer allows sensor manufacturers to provide a very high degree of functionality and value to the end user, for a modest price.

For sensing current *Linear Hall-Effect Sensor ICs* are the most appropriate. These devices deliver an output signal which is a linear function of the magnetic flux density passing perpendicularly through its Hall plate. An example of a transfer curve for a ratiometric linear is illustrated in figure 4. It can be noted that at each extreme of its range, the output saturates.

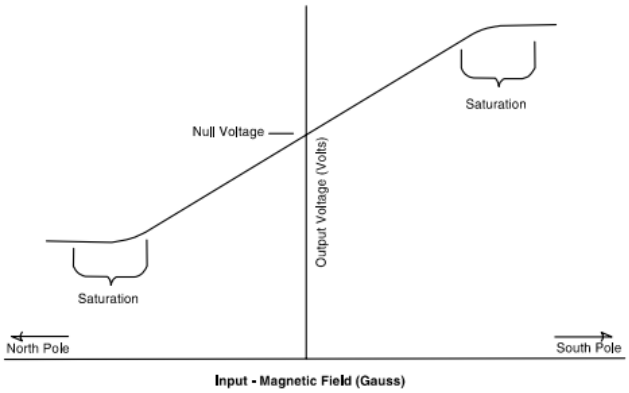


Fig.4 *Typical Transfer curve for a radiometric linear Hall Effect IC*

Between the Hall Effect IC, two different types can be distinguished: closed loop and open loop

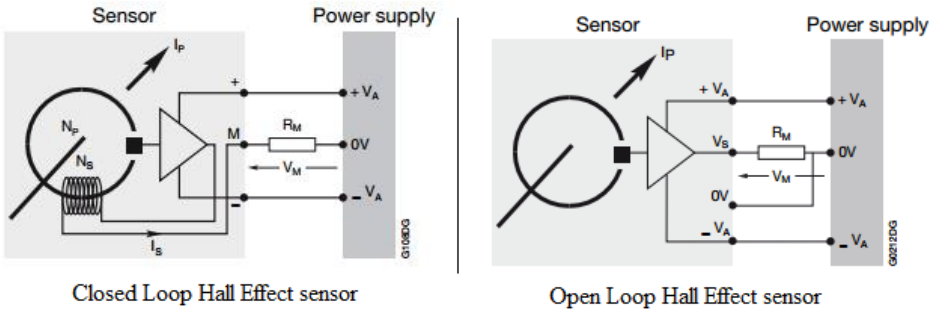


Fig.5 *Closed and Open loop Hall Effect sensors [14]*

Hence, after this brief description of the main methods used nowadays to sense current, the following comparison table 3 is displayed to review all the information.

Table3. *Strength and weakness of the main current sensing technologies*

Current sensing technology	Low resistance current shunt	Current Transformer	Hall effect Closed loop	Hall Effect Open Loop
Relative cost	Low	High	Medium	High
Insertion Loss	Yes	Yes (AC)	No	No
Circuit isolation	None	Yes	Yes	Yes
External Power	None	None	Yes	Yes
Offset	None	None	None	Yes
Accuracy (Est.)	>99%	>95%	>95%	>95%

The reasons for using a particular sensor vary according to the application. In this project, cost and performance are essential. From table 3 it can be deduced that the CT has insertion loss and its relative cost is higher that the Hall Effect sensor. The Hall effect sensor has the disadvantage that it does require an external power, but considering an Arduino microcontroller is going to be used in this project (it includes a power supply) this problem is solved. Therefore, a Hall Effect sensor would be considered for this project.

Once a Hall Effect integrated circuit is selected, additional signal conditioning is needed. This would be extensively explained in the consequent Design chapter 5.

4.1.2 Voltage sensing

There are several manners for sensing the voltage: a simple voltage divider, voltage detectors, etc. Nonetheless, a few of them provide the isolation needed between the high and low AC voltage of this project.

After a thorough research the best solution would be implementing an electrostatic voltmeter. It is a device that does not require direct contact with the main lines. Trek provides some useful documentation [15] to implement it. However, developing this voltage sensor could be considered another full project due its complexion. Therefore, the most optimal solution found after the electrostatic voltmeter was utilizing an AC/AC converter that lowers the voltage level to the adequate level of the micro-controller device (Arduino). The transformer in the adapter provides the isolation desired. It is not as good as the first solution considered, but it is a good one.

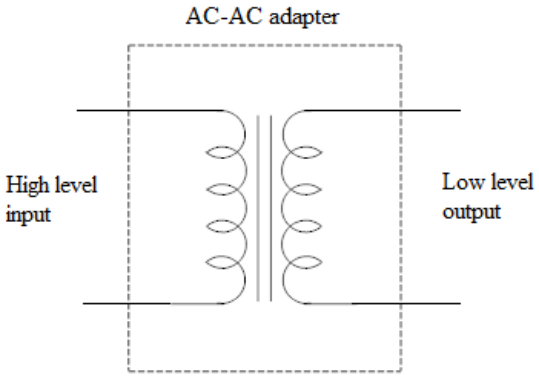


Fig.6 *AC/AC adapter schematic*

Once an AC adaptor is selected, the following step is conditioning the output of the AC power adapter so that it meets the input requirements of the micro-controller device analog inputs. This would require additional electronics that would be explained in the design chapter 5 of this document.

4.1.3 Sampling current and voltage: Arduino Ethernet board without PoE

After sensing the current and voltage levels with the previous sensors, the outputs of these sensors have to be sampled by the micro-controller, converting them from analog to digital signals in order to conduct the required operations with a micro-controller program to obtain the real power consumed.

There are several micro-controllers available in the market. One of the most economic and simplest is the Arduino. In fact, it is very common in house-energy monitoring systems. Its core is an *ATmega328* chip. In order to interface with the environment this chip is integrated in a board called ‘Arduino Ethernet Board without PoE’. PoE refers to ‘Power over Ethernet’. It is a technology that incorporates electrical power to a standard LAN infrastructure. This allows a single cable to provide both data

connection and electrical power. This board lacks of it. There is plenty of documentation about this board in the official page of Arduino project [16].

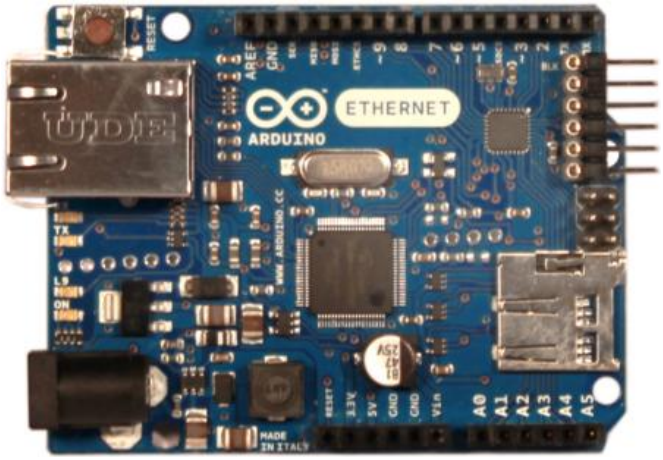


Fig.7 *Arduino Ethernet without PoE [16]*

This board was chosen taking into account its popularity and simplicity. Moreover, it contains everything this project need. The main features of this board that should be point out are:

- It incorporates 6 analog inputs pins where the sensors output can be connected to be sampled. These inputs are connected to an ADC that converts the analog signals to digital.

- It is equipped with a RJ45 connection or Power over Ethernet ready Magnetic Jack so it can be connected to a LAN.

- The chip in charge of managing the Ethernet connection is the W5100 TCP/IP Embedded Ethernet Controller. For more information about this chip refer to [17].

To interface with this board the programmer can use the software provided by Arduino official website[1].

Arduino IDE software

The Arduino Integrated Development Environment (*Arduino IDE*) software provides comprehensive facilities to computer programmers for software development. It consists of a source code editor, build automation tool and a debugger.

1. http://www.arduino.cc

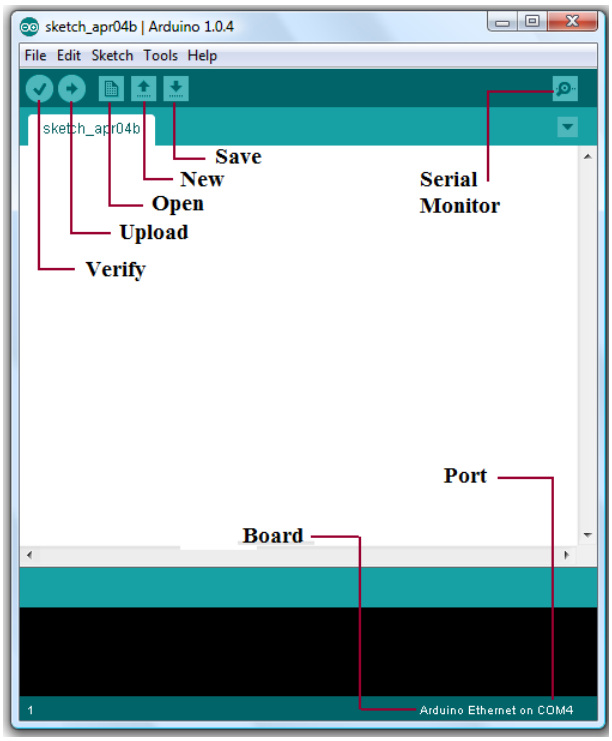


Fig.8 *Arduino IDE software*

The software includes a text editor for writing code, a text console, a toolbar with buttons for common functions, and some menus. It connects to the Arduino hardware (the Ethernet Board in this case) to upload programs and communicate with them.

Intended for easing the programmer job, this software includes built-in functions called 'libraries'. Common code that is subject to be reused is incorporated in these libraries. Additional libraries can be found over the internet. Consequently, some libraries may be used along this project: "If the wheel is already invented, there is no point in inventing it again". It can be improved, but the basics will remain.

4.1.4 Power calculations

Once the voltage and current signals are sampled by the micro-controller device, it is time to calculate the instantaneous power that is being consumed by the device under test.

The power the company supply is known at complex power *'S'*. Complex power can be expressed by the vectorial sum of the real power or active power *'P'*, plus the imaginary power or reactive power *'Q'*

$$S = P + jQ \tag{5}$$

Real power is defined as the power used by a device to produce useful work. It is the one the company bills home users for. Mathematically it is the definite integral of voltage, *'v(t)'*, times current, *'i(t)'*, as follows:

$$\mathrm{P} = \frac{1}{\mathrm{T}} \int \mathrm{v(t)} \cdot \mathrm{i(t)}\, \mathrm{dt} \tag{6}$$

Which can be expressed in terms of the root-mean-square (RMS) values of the current and the voltage, where $'\theta'$ is the gap phase between *'v(t)'* and *'i(t)'* signals.

$$P = V_{RMS} \cdot I_{RMS} \cdot \cos(\theta) \tag{7}$$

Equally, the reactive power *'Q'* that is the energy that flows back and forth in an inductive or capacitive load can be written in a similar way:

$$Q = V_{RMS} \cdot I_{RMS} \cdot \sin(\theta) \tag{8}$$

As the complex power *'S'* is the vectorial sum of *'P'* and *'Q'*:

$$S = V_{RMS} \cdot I_{RMS} \tag{9}$$

Finally, the power factor *'PF'* is a measure of efficiency. It is defined as:

$$PF = \cos(\theta) = \frac{\mathrm{P}}{|\mathrm{S}|} \tag{10}$$

For the common user, the valuable magnitudes are the real power *'P'* and the power factor *'PF'*. The voltage and current is extra information that can also be useful.

Hence, for calculating the power factor, considering (10), complex power *'S'* and real power *'P'* have to be determined. The complex power can be calculated attending to (9). It should be said that the RMS value is defined as the square root of the mean value of the squares of the instantaneous values of a periodically varying quantity, averaged over one complete cycle like (11) illustrates:

$$\mathrm{V_{RMS}} = \sqrt{\frac{1}{T} \int v(t)^{\,2}\, dt} \tag{11}$$

Moreover, the micro-controller device that sense the current and the voltage will be working in the discrete time domain, instead of the continue time domain, as an ADC

(analog to digital converter) is the responsible of sensing these magnitudes. Therefore, the equivalent discrete time equation for (11) would be (12)

$$V_{RMS} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} v(n)^2} \tag{12}$$

Where *'N'* is the number of samples and *'n'* the current sample. The same procedure can be applied for the current.

With (12) and the equivalent equation for the current, the complex power *'S'* can be calculated using (9). To resolve the Power Factor *'PF'* with the equation (10) the real power *'P'* must be estimated. This formula could be used:

$$P = \frac{1}{N}\sum_{n=0}^{N-1} v(n) \cdot i(n) \tag{13}$$

This concise theory should be enough to be able to develop a program for the micro-controller device that calculates the real power *'P'* and the power factor *'PF'*.

4.3 Receiving and storing the data

Once the desired magnitudes (Real Power, Power Factor, Current and Voltage) are computed with the Arduino micro-controller they have to be graphed.

The first thought for doing this was directly drawing on a third-party application such as *Cosm*[1]. This simplifies the process as the data is not required to be stored before graphing it. It is directly sent to this website which manages the data storage by itself. Figure 9 shows an example of the aspect of *Cosm* graphs.

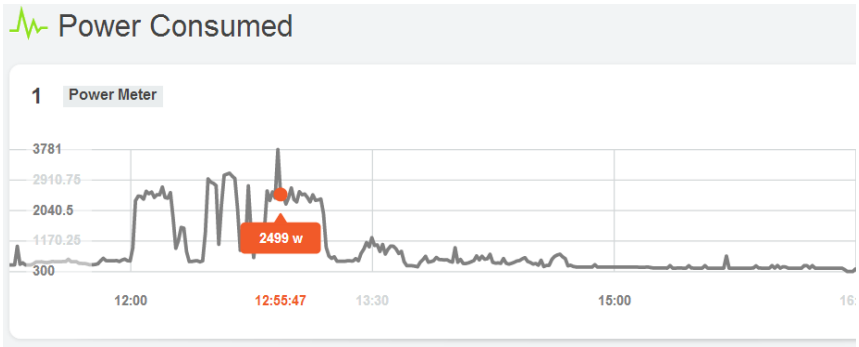


Fig.9 *Cosm feed example [18]*

1. http://www.cosm.com

However this option was finally abandoned as it creates unnecessary ties with *Cosm* website. The project could become obsolete if *Cosm* changed its API or disappeared, like happened with Ken Bongort and Adam McCann project [4] after GooglePower meter project [5] was abandoned. In fact, *Cosm* was initially called *Pachube* and after this major change a lot of applications needed to be changed to make them useful again.

Therefore, it is definitely better storing the desired data computed with Arduino micro-controller in a database to graph it afterwards. This way the system is completely independent. A *PHP* script will be the intermediate between the Arduino code and the database. This script will be in command of receiving the data from the Arduino and push it to the database.

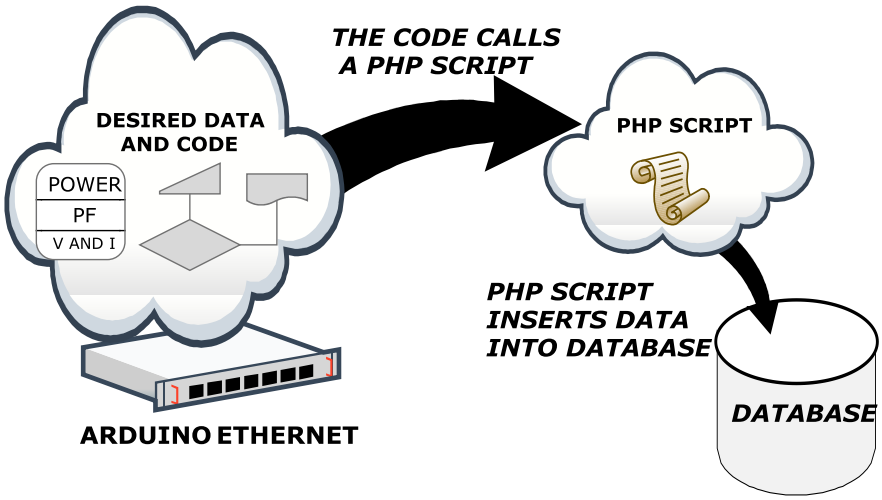


Fig.10 *Receiving and storing data diagram*

A server able to run a *PHP* script is required for developing this structure. Similarly, a database with database administrator software is needed. The following sections of this document would try to briefly introduce the reader to these concepts. It is not the intention of this document exposing a deep explanation of each of them but providing a general idea to understand this project. For more information access the official homepage of each concept provided in the reference section.

4.3.1 *Apache* local web-server

A server is a node that belongs to a net and is able to provide services to other nodes called clients. In most cases, the server is a physical computer (a computer hardware system) that serves other computers connected to the same net. There are two main ways of owing a server: buying a hosting to a dedicated company or use your own computer as a server. This last option was selected for the development of this project. It is an opportunity to learn about server managing and it is also more economic.

In order to set up a web-server (a computer that helps to deliver web content that can be accessed through the Internet) a web-server software program is needed. There is

plenty of software designed for this purpose. The following figure 11 shows the market share for top servers across all domains in the last 17 years [19]

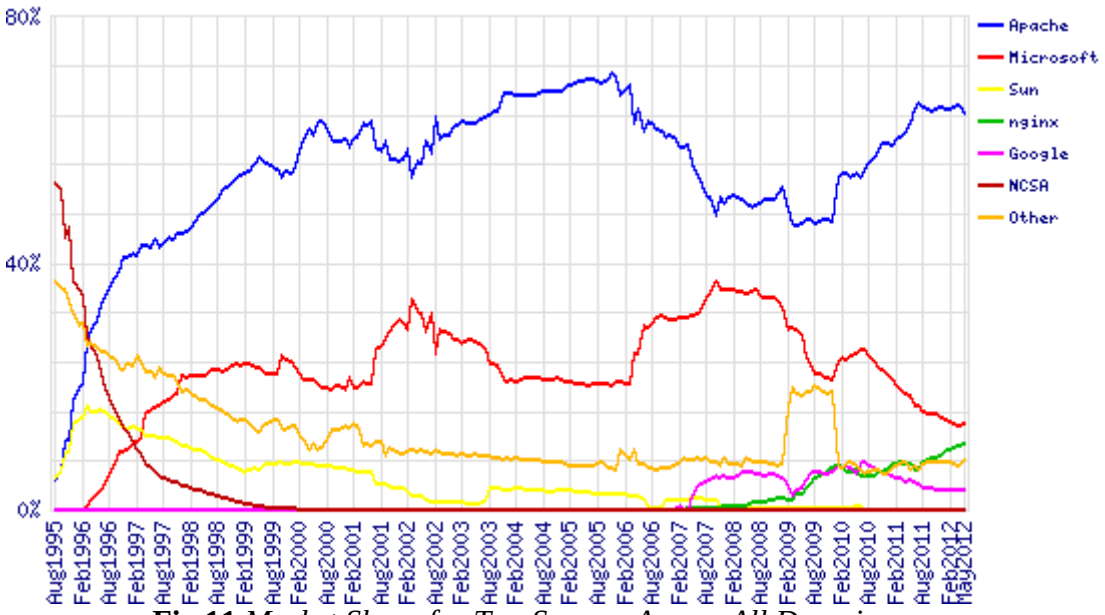


Fig.11 *Market Share for Top Servers Across All Domains. August 1995 - May 2012 [19]*

The most common one is '*Apache HTTP server'*. According to The Apache Software Foundation "The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards" [20]

To run a local server (use your own computer as the server) the network domain should be 'localhost' or the IP of your computer in the LAN as shown in figure 12.

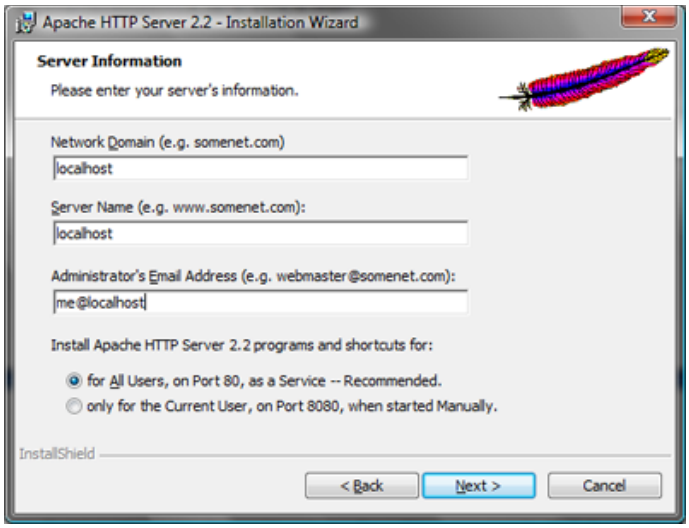


Fig.12 *Apache 2.2 Installation Wizard*

This server supports *PHP* that is needed for the progress of the project as well as providing HTTP services. HTTP functions as a request-response protocol in the client-server model [21]. Arduino micro-controller, for example, may be the client and the *PHP* script running on the computer hosting may be the server. The client (Arduino) submits an HTTP request message to the server (*PHP* script) and afterward the server response to the client. With this it would be possible to send data from the Arduino to the server.

4.3.2 *PHP*: Hypertext Preprocessor

According to the official webpage of *PHP*, "*PHP* is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML" [22]. It is one of the most common programming languages for web pages. It is a server-side scripting (that is why the server must have a *PHP* interpreter). *PHP* code is interpreted by the web-server that generates the resulting webpage. Its ancestor is the *C* programming language.

PHP has hundreds of functions and predefined variables. These functions are well documented on the *PHP* site. Nonetheless, it would be extremely suitable explaining one of these predefined variables which is essential for the development of this project.

This predefined variable is the $_GET variable. It consists of an associative array of variables passed to the current *PHP* script via the URL parameters.

Example #1 *$_GET* example

```
<?php
echo 'Hello ' . htmlspecialchars($_GET["name"]) . '!';
?>
```

Assuming the user entered http://example.com/?name=Hannes

The above example will output something similar to:

```
Hello Hannes!
```

Fig.13 *$_GET predefined variable example [22]*

Figure 13 shows how this variable works. It is capable of capturing the data written as an URL parameter ('name' in the example). Thanks to this variable the data could be received by the *PHP* script. The code of the client (Arduino micro-controller) would be in charge of making the HTTP request including as URL parameters the variables that contain the real power, power factor, current and voltage values.

4.3.3 *MySQL* database *PHPMyAdmin*

Once *PHP* has collected the data from the Arduino, this data has to be stored in a database. A database is just an organized collection of data. As occurred with the web-server software, there are several database management systems (software system designed to allow the definition, creation, querying, update, and administration of databases) such as *MySQL*, *PostgreSQL*, *Microsoft SQL Server*, *Microsoft Access*, *SQLite*, etc. Nevertheless, the popular choice for web applications is *MySQL [23]*. In addition, it is under a GNU General Public License, a free software license. Therefore, *MySQL* will be used in this project.

Fig.14 *MySQL database [23]*

Managing this database from a browser requires extra software. Once more a full variety of tools is available in the market [24]. The chosen one for this project was *PHPMyAdmin*. It is a free and open source tool written in *PHP*. It is capable of handling the administration of *MySQL* using a web browser. This fact simplifies the managing of the database. This tool can perform several tasks. For instance, creating and deleting databases, tables, fields or rows. It can execute *SQL* statements as well as managing users and permissions. All from a web interface as figure 15 illustrates.

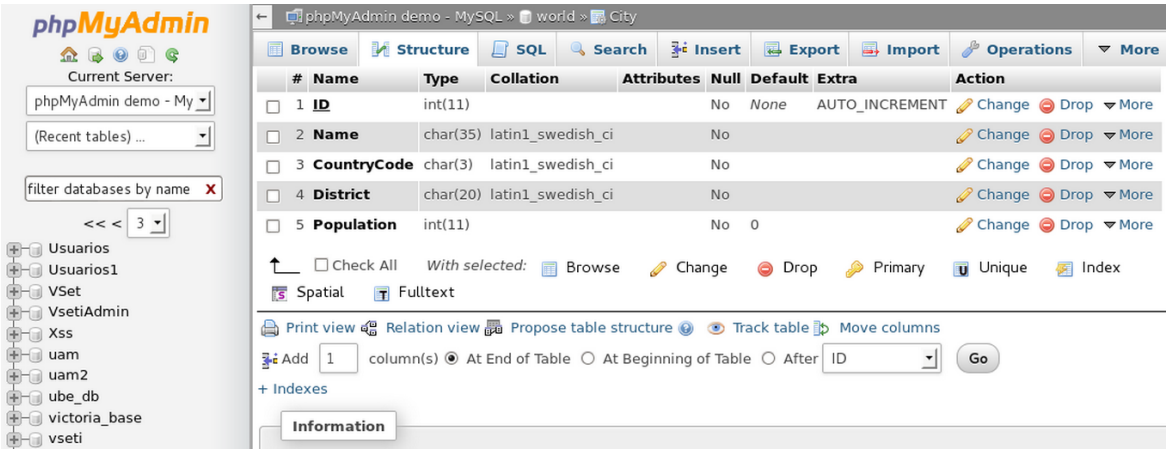


Fig.15 *PHPMyAdmin table structure example*

The procedure to insert data into this database is appealing to *SQL* statements. *SQL* is a special programming language designed for managing data held in a database system. The *PHP* script will conduct this task subsequent to collecting the data from the Arduino micro-controller.

4.4 Graphing the data

At this point the data required (real power, power factor, current and voltage mean values) would be already stored in the *MySQL* database. The following step is graphing it so common users can understand it better. Here is when *JavaScript* enters into scene. This section will introduce the reader to this programming language. Nonetheless, it will be a plain and brief introduction. More information is accessible in the reference section.

Figure 16 shows the procedure for the accomplishment of the task described before:

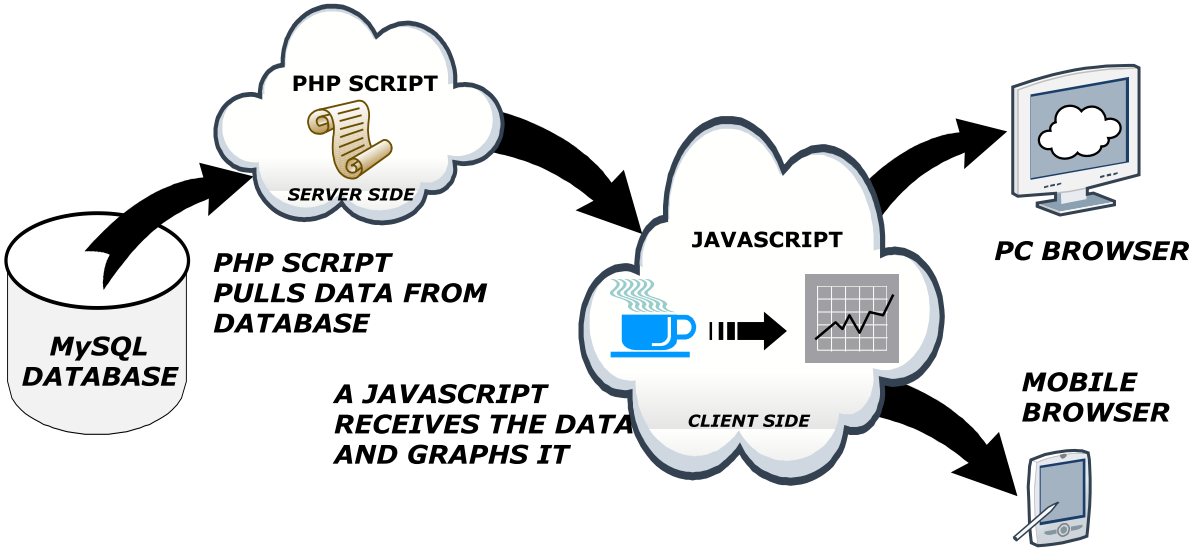


Fig.16 *Graphing data diagram*

Once more a *PHP* script will be the one who pulls the data from the database. This *PHP* script has the function of preparing the data in a specific style so the *JavaScript* can understand it. The graph will be implemented with *JavaScript* libraries (*jQuery* and *Highcharts*) that simplify the programming to the developer. The concept of library was already introduced, but to remind it to the reader, it is a cluster of functions for common tasks that has a well-defined interface making the programmer job easier. In the following sections of this chapter all this will be clarify.

It could be interesting for the reader to pinpoint that the *PHP* script runs in the local web-server previously installed, but the *JavaScript* is able to run in the client-side: *JavaScript* support is built right into all the major web browser, so as long as the user has a updated web browser, it will run without problems.

4.4.1 *JavaScript*

It is not the intention of this document to deepen in the *JavaScript* language. Therefore, only a basic and essential introduction will be provided for the reader.

JavaScript is an interpreted programming language used to make web pages interactive. Its syntax was influenced by the language *C*. It runs on the client's computer so these scripts are able to interact with the user and alter the document content that is being displayed. It is a scripting language; this is a lightweight programming language. *JavaScript* is also an interpreted language, so no particular program is required to create code. Any plain text editor is sufficient.

As it was mentioned before, all the major web browsers available in the market support *JavaScript*. The advantage of using *JavaScript* is that supports client-side scripting. It is able to run after the webpage has already loaded. This feature is essential for this project. The graphs will be in a webpage that will only be loaded once. After that, *JavaScript* will be in charge of updating the series of the graphs without the need of reloading the website.

4.4.4 *jQuery* and *Highcharts*

In the introduction of the '3.4 Graphing data' section it was mentioned that the graphs will be implemented making use of *JavaScript* libraries. These free libraries are *jQuery* and *Highcharts*.

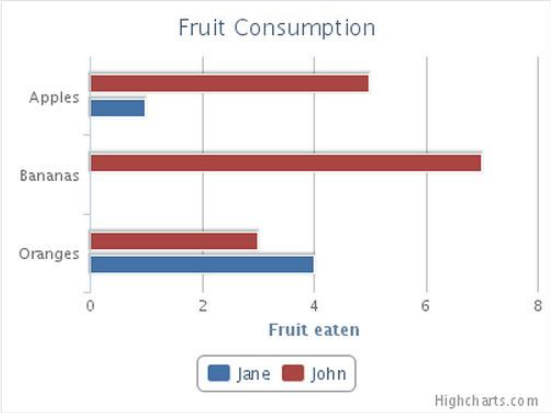
The best manner to explain *jQuery* is resorting to the official website who defines it like "a fast, small, and feature-rich *JavaScript* library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, *jQuery* has changed the way that millions of people write *JavaScript*" [25]. In other words, *jQuery* provides a cluster of functions or routines to perform ordinary tasks. Programmers used *jQuery* to avoid developing by themselves basic tasks since they are already included and tested in this library. Basically, it makes the programmer job simpler.

jQuery is just a means to use *Highcharts*. *Highcharts* is the real useful library for this project but its foundation is *jQuery*. That is why it is previously explained in this document.

Highcharts, as its developers affirm, "is a charting library written in pure *JavaScript*, offering intuitive, interactive charts to your web site or web application" [26]. This library is ideal to create web applications in any language (*PHP* for instance) and then integrate graphs with *jQuery JavaScript* framework. *Highcharts* consists in a script encapsulated in just one file called 'highcharts.js'. There are other existing libraries that fulfill this purpose such as jqPlot[1], FlotCharts[2], JpGraph[3], GoogleCharts[4], etc. But none of them are so versatile and well-designed. *Highcharts* has the friendliest interface for the user.

1. http://www.jqplot.com/
2. http://www.flotcharts.org/
3. http://jpgraph.net/
4. https://developers.google.com/chart/?hl=en

The official *Highcharts* website provides a complete API documentation to elaborate your own graphs or charts. Just with a simple code detailed graphs can be displayed in the website. The next figure 17 shows an example, although not all the code is displayed.




```
$(function () {
    $('#container').highcharts({
        chart: {
            type: 'bar'
        },
        title: {
            text: 'Fruit Consumption'
        },
        xAxis: {
            categories: ['Apples', 'Bananas', 'Oranges']
        },
        yAxis: {
            title: {
                text: 'Fruit eaten'
            }
        },
```

Fig.17 *Highcharts example [26]*

The script at the right of the image is written in a separate file with the .js extension. Then this file is called from the html website.

With these tools multiple graphs for the data required (real power, power factor, current and voltage mean values) could be elaborated.

5. Design

This chapter will describe the design evolution of the project. The very last methodology chapter 4 smoothed the way to understand the design of each module of the project. Now this chapter will be divided in the three main tasks that have to be designed to reach the final aim of the project: measuring the power, receiving and storing the data and finally graphing it. Finally all these tasks will be united in a unique program to reach the final aim of the project.

Some tasks of this project require of programming code. The code will not be exposed in this chapter but in the Appendix A and B. By the contrary, each code will be explained by a flow diagram. However, some code snippets may be added for better understanding of the design.

5.1 Measuring Power design

The first task to determine the power is designing how to sense the current and voltage. This was studied in the methodology chapter 4. The next step is conditioning the output signals of the sensors used. The following section will describe all the steps conducted in this area.

5.1.1 Current measurement design

The final sensor selected is the linear Hall Effect sensor ACS712 x05B [27].

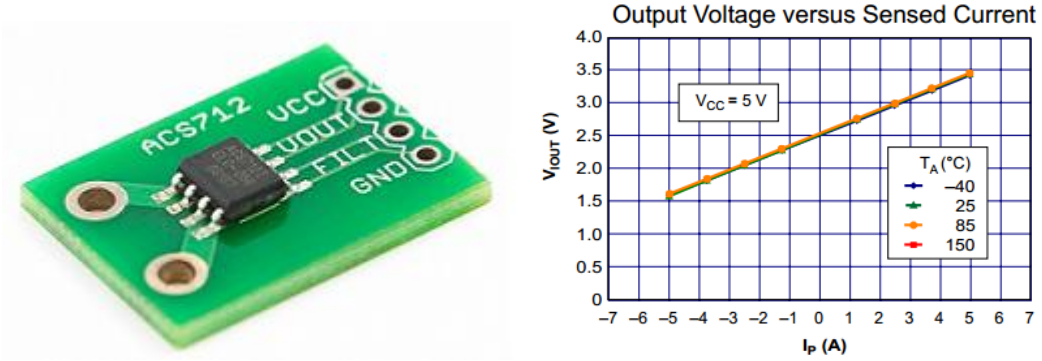


Fig.18 *ACS712x05B breakout board and transfer function from its datasheet [27]*

It has a range of ±5A and a typical sensitivity of 185mV/A. Figure 18 displays that the ACS712 outputs an analog voltage output signal that varies linearly with sensed current. The output voltage presents an offset of 2.5V. Therefore, when no current is sensed, the sensor outputs 2.5V. This occurs when the supply is 5VDC. The Arduino board can supply that required voltage.

The Arduino analog inputs must meet an important requirement: the input voltage has to be a positive voltage between 0V and the ADC reference voltage (5V in this case). The output of this sensor meets this requirement (it is always positive and below 5V), but

considering its sensitivity (185mV/A), the output variation is very weak. It requires amplification. So the step required for the signal conditioning is scaling up the output of the sensor thus a major range of the ADC is exploited. This can be easily achieved with operational amplifiers (op-amp)

As the output contains a DC component, a common non-inverting configuration of an op-amp cannot be used, as it would amplify this DC offset. The best solution would be a differential amplifier that only amplifies the difference between its input terminals. However, this solution would not meet the input requirements of the Arduino, as it would swing between positive and negative values. Retaking the idea of the non-inverting amplifier, it could be designed so it only amplifies AC components adding some capacitors. This way the only signal amplified would be the AC signal. Again, this implies negative values of the signal. A DC bias should be added to make this signal move around this bias point. If an op-amp is said to be biased, this means that, for no incoming signal or no sensor excitation, the output voltage will rest at the biased voltage.

So the steps required for the signal conditioning are:

1) **Amplifying the AC signal of the sensor output**. An AC-coupled non-inverting amplifier would be in charge of this. An AC-coupled op-amp only amplifies signals which change with time, so the 2.5V offset of the sensor output will not be amplified.

2) **Biasing the op-amp output.** This must be done applying a DC offset at the input of the op-amp. Since the op-amp output reflects the voltages present at the input, bias voltage can be applied at the input, and allow the negative feedback around the amplifier to bring the output to the voltage desired. This can be done replacing any ground in the circuit with the bias voltage (except if it the ground is only connected to a capacitor or if the ground is not connected in some way to the input of the op-amp)

3) **Designing the biasing circuit.** This can be easily done with a simple voltage divider. However, for reducing power consumption and give even more accurate gain and offset values, a voltage follower biasing will be used (with an op-amp). This way the impedance the biasing op-amp will present to the rest of the circuit will be infinitesimal compared to any other method.

The final biased AC-coupled non-inverting amplifier is displayed in the following figure:

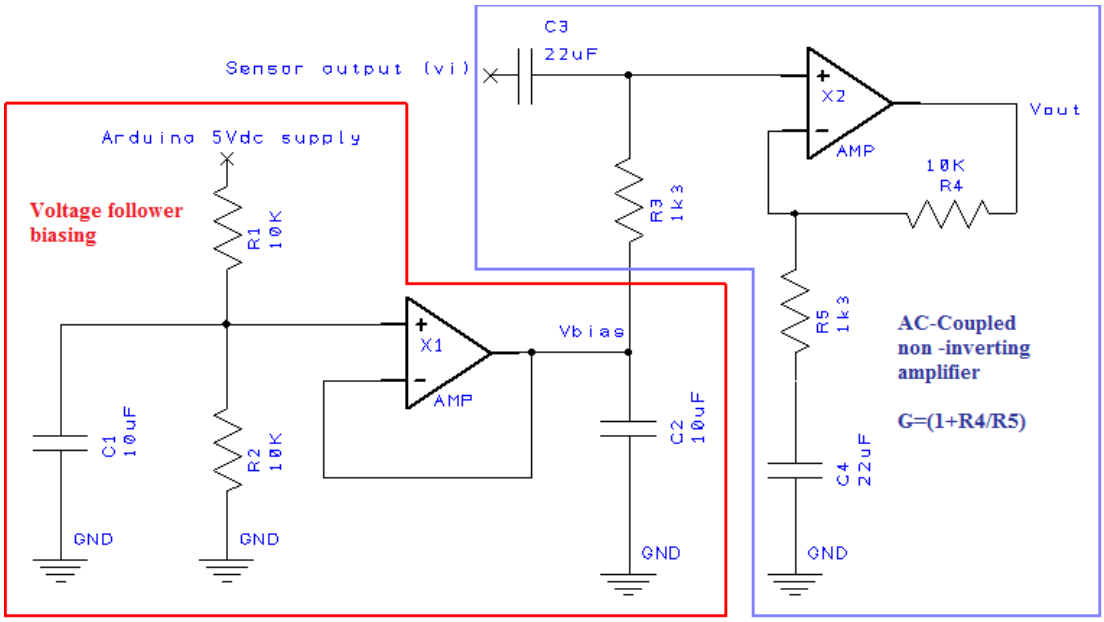


Fig.19 *ACS712x05B signal conditioning circuit*

Figure 19 shows the complete signal conditioning circuit. This circuit consists in two sub circuits. The first one is the voltage follower biasing circuit. This circuit has the aim of generating a DC voltage bias. It is an op-amp configured like a follower, so the output impedance is nearly zero, avoiding interfering with the next stage. The follower is fed with a voltage divider and a capacitor. These components together generate a low pass filter with a cutoff frequency of around 3Hz, like equation (14) shows. Capacitor *C2* presents low impedance to high frequencies as well.

$$f_{1-cut-off} = \frac{1}{2 \cdot \pi \cdot \frac{R1 \cdot R2}{R1 + R2} \cdot C1} = \frac{1}{2 \cdot \pi \cdot \frac{10K \cdot 10K}{10K + 10K} \cdot 10uF} = 3.18Hz \tag{14}$$

The second stage is the AC-coupled non-inverting amplifier whose gain is defined by equation (15):

$$G = \left(1 + \frac{R4}{R5}\right) = \left(1 + \frac{10K}{1k3}\right) = \left(1 + \frac{10000}{1300}\right) \approx 8.7 \tag{15}$$

The capacitors C4 and C2 are the ones who provide AC coupling. The cut-frequency of the filter that forms with the resistors follows the expression (16):

$$f_{2-cut-off} = \frac{1}{2 \cdot \pi \cdot R3 \cdot C3} = \frac{1}{2 \cdot \pi \cdot R5 \cdot C4} \tag{16}$$

The value of these components was chosen hence the cut-off frequency were around 10Hz. A certain value for the resistors was fixed so the capacitors value could be found out.

$$10Hz = \frac{1}{2 \cdot \pi \cdot 1300 \cdot C3} \rightarrow C3 = \frac{1}{2 \cdot \pi \cdot 1300 \cdot 10} = 12.25uF \tag{17}$$

The standard value of 22uF was chosen. This superior value ensures even a smaller cut-off frequency (5.56Hz).

With this circuit the output signal of the Hall Effect sensor is correctly conditioned to meet the Arduino input requirements. For instance, supposing a current with a peak of 1A wanted to be measured (for larger currents the gain of the non-inverting op-amp stage should be reduced to meet the Arduino input requirements) the maximum value the sensor would output with a sensitivity of 185mV/A would be 185mV plus the 2.5V of the offset. The signal conditioning will produce an AC signal whose peak would worth 1.61V and would swing around the Vbias added (2.5V in this case). Figure 20 shows the initial Hall Effect sensor output signal (yellow), the amplified signal if the offset voltage was not added (blue), and the final conditioned signal with the voltage bias added (red) so the Arduino input requirements are met.

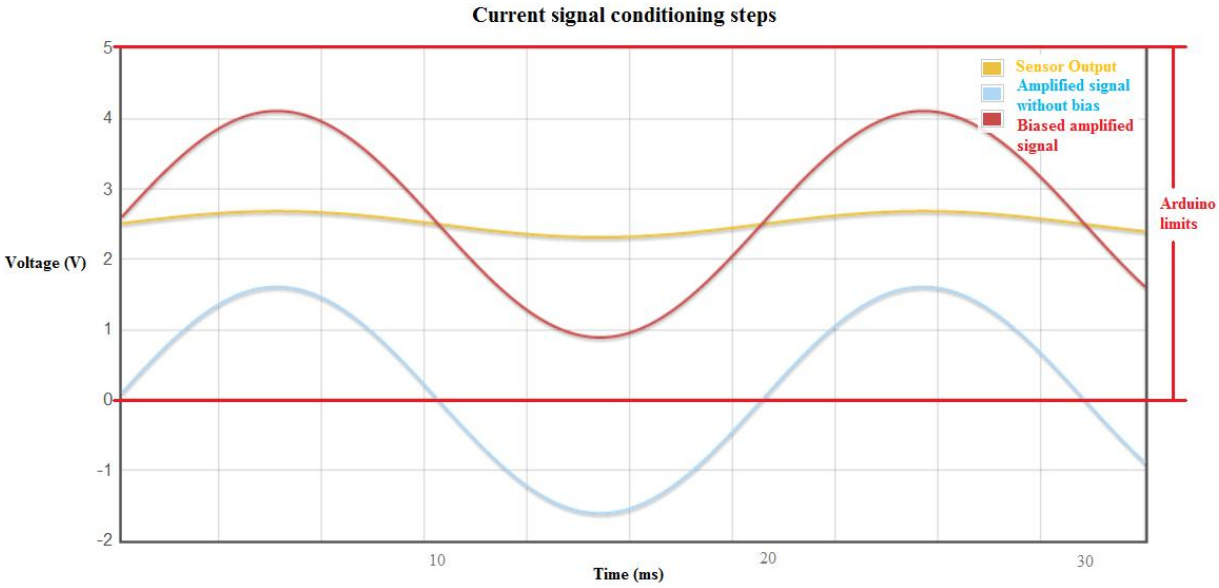


Fig.20 *Current signal conditioning graphs (1A peak supposed)*

The resolution of the system for 1A peak current range would be approximately:

$$Resolution = \frac{Range}{ADC\ Resolution} = \frac{2A}{2^{10}} = 1.95mA \tag{18}$$

Current calibration constant

The current is measured using a Hall Effect sensor that converts the current into a voltage. This voltage is then conditioned and measured by the analog input of the processor, who outputs a count *'C'* between 0 and 1024. Therefore, a calibration constant has to multiply this count to display the real value of the current sensed in the graph. The input voltage to the processor has an offset added to it, but this will be removed by a software filter that will be explained in following sections, so it can be ignored when calculating the calibration constant.

The output $'V_1'$ of the Hall Effect sensor will be:

$$V_1 = Sensitivity \cdot Current_{peak\ -value} = 0.185\,{}^{V}/_{A} \cdot I_{pk} \tag{19}$$

This voltage $'V_1'$ is then amplified 8.7 times. See equation (15)

$$V_2 = 8.7 \cdot V_1 = 8.7 \cdot 0.185\,{}^{V}/_{A} \cdot I_{pk} \tag{20}$$

Then the ADC transforms this $'V_2'$ value into a count *'C'*.

$$C = \frac{1024}{5} V_2 = \frac{1024}{5} \cdot 8.7 \cdot 0.185\,{}^{V}/_{A} \cdot I_{pk} = 329.62 \cdot I_{pk} \tag{21}$$

Working with RMS values:

$$C = 329.62 \cdot I_{RMS} \cdot \sqrt{2} = 466.15 \cdot I_{RMS} \tag{22}$$

Hence, the calibration constant $'I_{cal}'$ needed is:

$$I_{RMS} = \frac{1}{466.15} \cdot C \rightarrow I_{cal} = \frac{1}{466.15} \tag{23}$$

5.1.2 Voltage measurement design

This section will cover briefly the electronics required to interface an AC power adaptor with the Arduino micro-controller. The methodology chapter 4 explained the reason for selecting this solution.

The AC power adapter chosen was a $9V_{RMS}$ adaptor. The output signal from the AC voltage adapter is a near-sinusoidal waveform. So the signal peak should value ±12.7V.

The Arduino analog inputs must be between 0V and 5VDC as it was explained before in the current measurement design section. The signal conditioning has to convert the output of the adapter to a waveform that has a positive peak that is less than 5V and a negative peak that is more than 0V. So the steps required are:

- **Scaling down the waveform.** This can be done using a simple voltage divider at the output terminals of the adaptor.

- **Adding an offset to avoid negative values.** The offset or DC bias can be added using an external voltage source. In fact, the Arduino has a 5V voltage supply.

Here is the designed circuit:

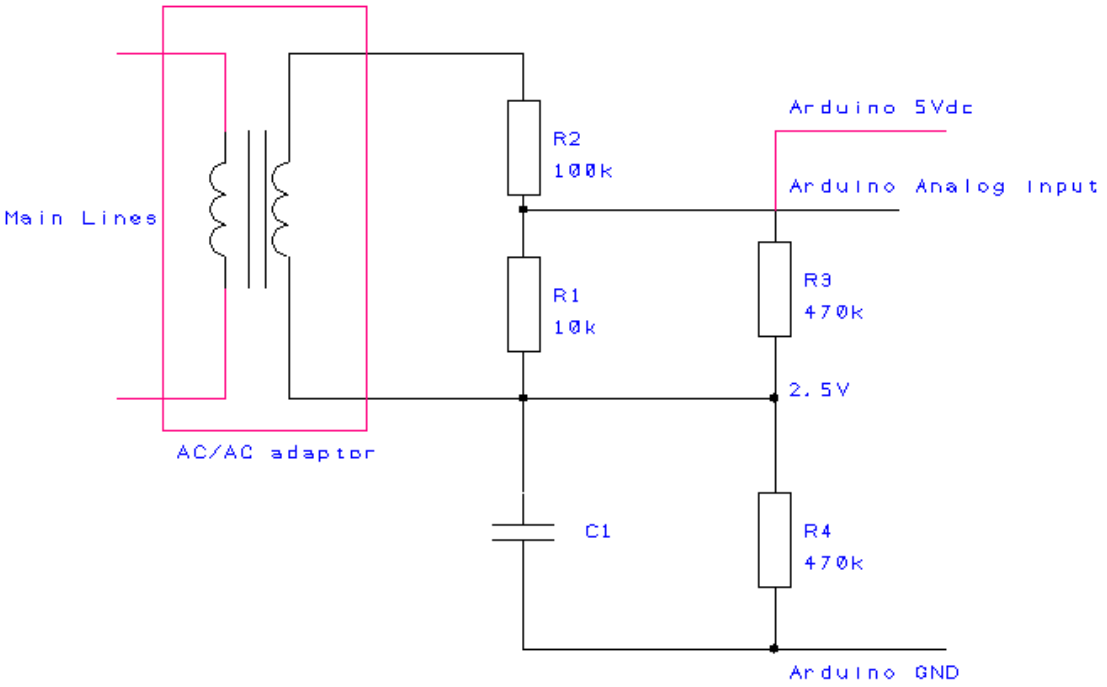


Fig.21 *Voltage signal conditioning*

Resistors *'R2'* and *'R1'* form the voltage divider and resistors *'R3'* and *'R4'* provide the voltage offset (2.5V). Capacitor *'C1'* derives high frequency components of the AC signal to the ground. It is a low-pass filter. *'R1'* and *'R2'* were chosen so the output peak value would be within the limits of the Arduino analog input. For an AC adapter with a $9V_{RMS}$ output the voltage divider provides a peak value of:

$$v_{out}^{peak} = \frac{R1}{R1+R2} \cdot v_{in}^{peak} = \frac{10k}{10k+100k} \cdot 12.7 = \frac{1}{11} \cdot 12.7 = 1.15V \quad (24)$$

'R3' and *'R4'* form another voltage divider. As they have identical values, the middle point between them has a voltage of 2.5V respect *'Arduino GND'*. Higher resistors were used to reduce energy consumption.

The resultant waveform of the circuit has a peak of 2.5V ± 1.15V. This satisfies the Arduino analog input voltage requirements and leaves a security room to prevent over or under voltage.

The next figure shows a schematic of the voltage signal conditioning:

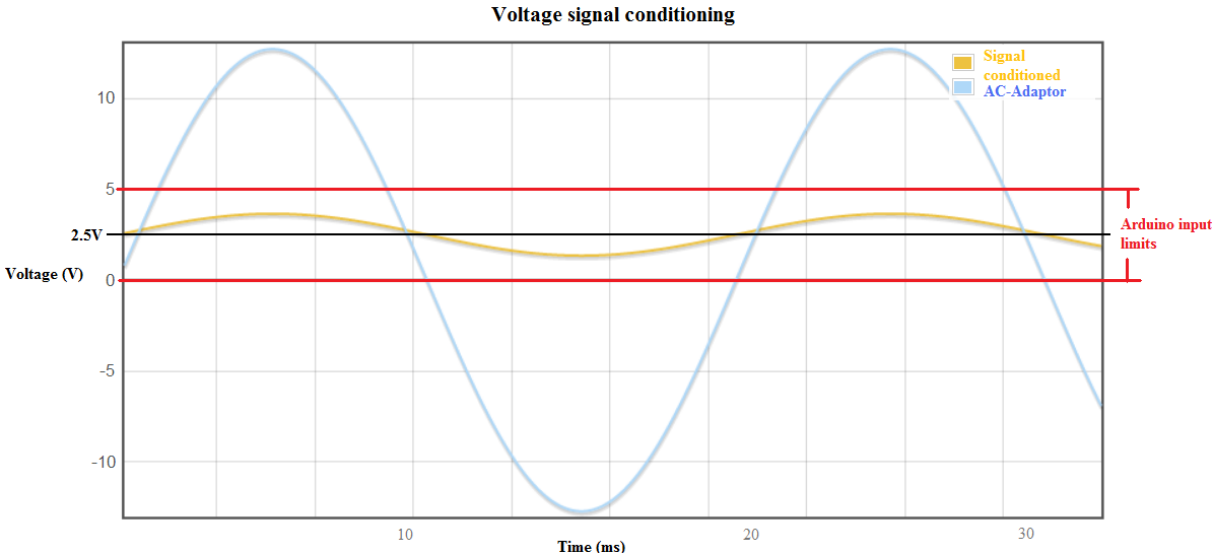


Fig.22 *Voltage signal conditioning graphs*

The resolution of the system for $230V_{RMS}$ voltage would be around:

$$Resolution = \frac{Range}{ADC\ Resolution} = \frac{230 \cdot \sqrt{2} \cdot 2}{2^{10}} = 635mV \tag{25}$$

Voltage calibration constant

The internal ADC converter of the Arduino micro-controller outputs a count between 0 and 1024 (2^{10}). So to display the real RMS value of the voltage measured, this count has to be multiplied by a voltage calibration constant *'V_{cal}'*.

The output *'V_1'* of the $9V_{RMS}$ AC adapter used connected to a $230V_{RMS}$ device will be:

$$V_1 = AdapterRatio \cdot InputVoltage_{RMS} = \frac{9V_{RMS}}{230V_{RMS}} \cdot V \tag{26}$$

This voltage *'V_1'* is then scaled down by a ratio of $\frac{1}{11}$. See equation (23)

$$V_2 = \frac{1}{11} \cdot V_1 = \frac{1}{11} \cdot \frac{9}{230} \cdot V \tag{27}$$

Then the ADC transforms this *'V_2'* value into a count *'C'*

$$C = \frac{1024}{5} V_2 = \frac{1024}{5} \cdot \frac{1}{11} \cdot \frac{9}{230} \cdot V = 0.728 \cdot V \tag{28}$$

Hence, the calibration constant *'V_{cal}'* needed is:

$$V = 1.37 \cdot C \rightarrow V_{cal} = 1.37 \tag{29}$$

5.1.3 Power consumed. Software design

Now that the current and the voltage signals are ready to be sampled by the Arduino micro-controller it is time to estimate the power consumed by the under test device. A library from "openenergymonitor" project [6] (mentioned in the background chapter) was used to achieve this duty. It was slightly modified to satisfy the needs of this specific project. The code of this library is attached in the Appendix A, but some snippets will be explained in this section to understand its function.

The main idea is implementing the mathematical formulas explained in the 'power calculations' section of the methodology chapter 4 in an Arduino code.

The procedure to estimate the power consumed *'P'* is simple: it consists in applying the equation (10)

$$\mathrm{P} = \frac{1}{N} \sum_{0}^{\mathrm{N}-1} v(n) \cdot \mathrm{i(n)} \tag{10}$$

First, a certain amount of samples *'N'* of a certain number of wavelengths of the signals of the analog channels are captured. These samples would be *'v(n)'* and *'i(n)'* in equation (10). For each sample *'n'*, the instantaneous power is estimated. Then all the instantaneous powers are summed and finally this value is divided by the number of samples *'N'* taken.

The main flow of the code that implements this method is displayed below in figure 23. Again remark that this code comes from a library of "openenergymonitor" project [6]. It is accessible in the net for everybody as it is open source code.

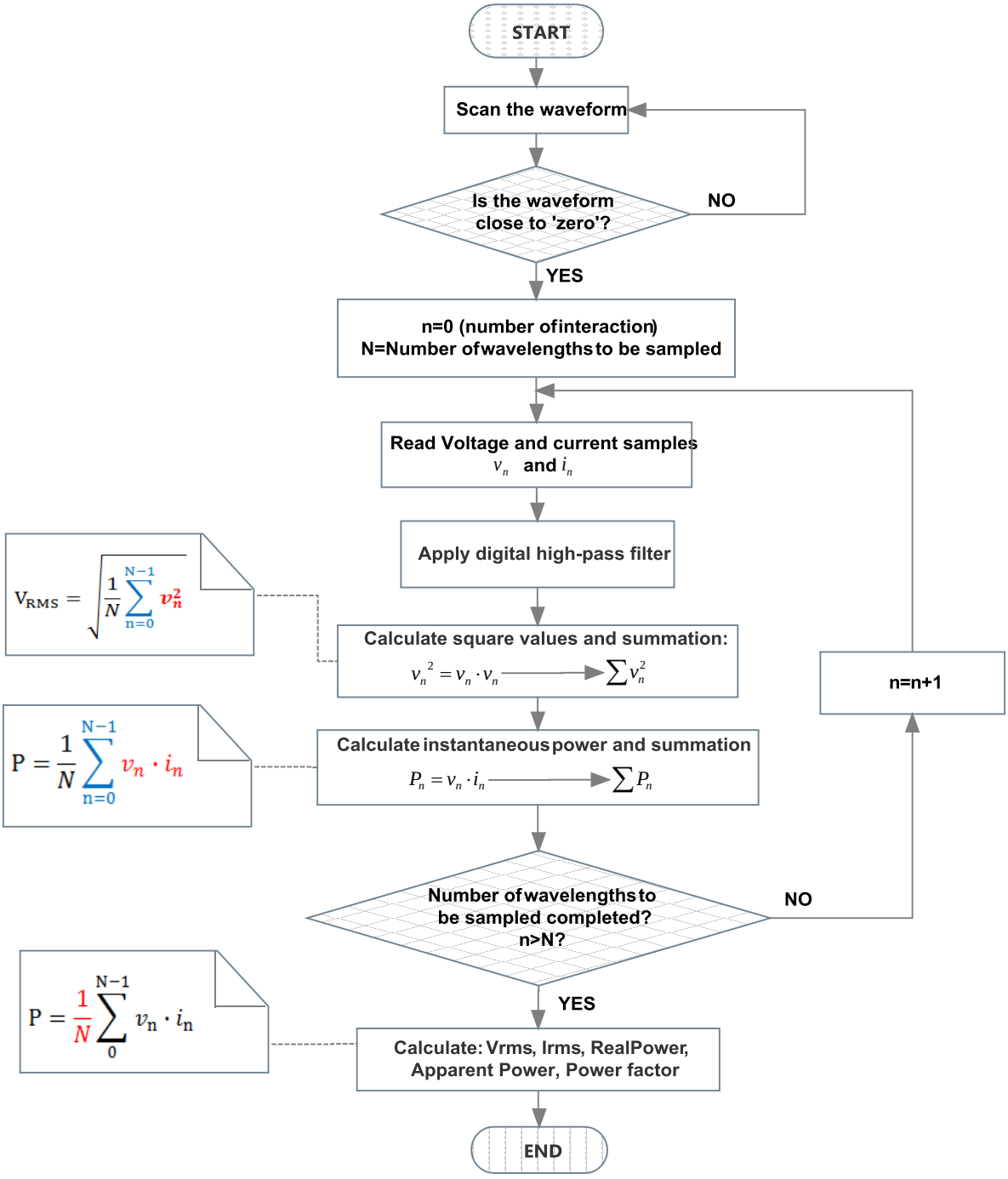


Fig.23 *Power calculation subroutine flow diagram*

Some snippets will be explained in the following lines to ease the comprehension of this flow diagram, but the main code is displayed in Appendix A (page 50)

First of all, the program starts scanning the voltage waveform. The program is stuck a while loop until one of the samples captured is next to zero crossing value. When this takes place the program carries on. Figure 24 illustrates this procedure.

```
while(loop==true)                    //the while loop...
{
    startV = analogRead(inPinV);     //using the voltage waveform
    if ((startV < 550) && (startV > 440)) loop = false;//check its
    within range
    if ((millis()-start)>timeout) loop = false;
}
```

Fig.24 *Checking if the waveform is close to zero point.*

The method to detect the zero cross point consists in verifying that the analog value of the sample is around the value 512. This is because the Arduino micro-controller has an internal 10-bit ADC, therefore the analog values are encoded between 0 and 1023. The input signal applied to the analog channel swings between 0 and 5V, so the ADC will assign 0 to 0V and 1023 to 5V. The mid value is 2.5V that correspond approximately with 512.

As the input signals have a DC bias of 2.5V (see previous 'voltage measurement design' section) the waveform oscillates around this value. Therefore, the zero point corresponds with 2.5V. This DC value has to be removed. Normally, DC components are filtered before sampling the signal, but in this case a software high-pass filter was applied. Figure 25 shows this.

```
//---------------------------------------------------------------
// B) Apply digital high pass filters to remove 2.5V DC offset
(centered on 0V).
//---------------------------------------------------------------
   filteredV = 0.996*(lastFilteredV+sampleV-lastSampleV);
   filteredI = 0.996*(lastFilteredI+sampleI-lastSampleI);
```

Fig.25 *Discrete high-pass filter design*

The theory behind this snippet is available in the net [28]. The value *'0.996'* is a constant. The higher this constant is the narrower the stop band of the filter will be.

The calculations of the root-mean-square values (needed to estimate the real power) are conducted in two steps. First the square values and the summation are calculated. In the following snippet the corresponding lines in charge of this task have been highlighted in colors to establish the relationship with the equation (14)

```
//---------------------------------------------------------------
// C) Root-mean-square method voltage
//---------------------------------------------------------------
    sqV= filteredV * filteredV;      //1) square voltage values
    sumV += sqV;                     //2) sum
```

Fig.26 *Square values and summation*

$$V_{RMS} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} v(n)^2} \tag{28}$$

Finally to calculate the root-mean-square values of the voltage and the current the calibration constant explained in previous sections has to be applied. This calibration constant has to be defined by the user in function of the sensor used. Again some parts of the code has been highlighted in colours to establish the relationship with the theoretical equation (15)

```
double V_RATIO = VCAL *((SUPPLYVOLTAGE/1000.0) / 1023.0); //
Vrms = V_RATIO * sqrt(sumV / numberOfSamples);
```

Fig.27 *Determination of real Vrms value*

$$V_{RMS} = V_{ratio} \cdot \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} v(n)^2} \tag{29}$$

In the preceding snippet it can be seen a *'SUPPLYVOLTAGE'* variable. This is because the real supply voltage of the ADC converter is measured for precise calculation [29].

The power calculation follows a similar procedure. Consequently, there is no point in showing more snippets as the code is available in the Appendix A.

5.2 Receiving and storing data design

Once the data is captured by the Arduino code, it has to be graphed. The first attempt was to directly graph it. Nevertheless, the final design first store the data, and the graph it. The advantages of this design will be cited in the following lines.

5.2.1 First attempt: *Cosm* website

As it was mentioned in the methodology chapter 4, the first try was to directly send the data stored in Arduino code variables into *Cosm* website, removing the intermediate step of storing the data in a database. The initial step to achieve this is creating a *Cosm* account.

The main code was elaborated using *Cosm* website examples. The following figure shows the diagram flow of the Arduino program. The full code is available in the Appendix A (page 43)

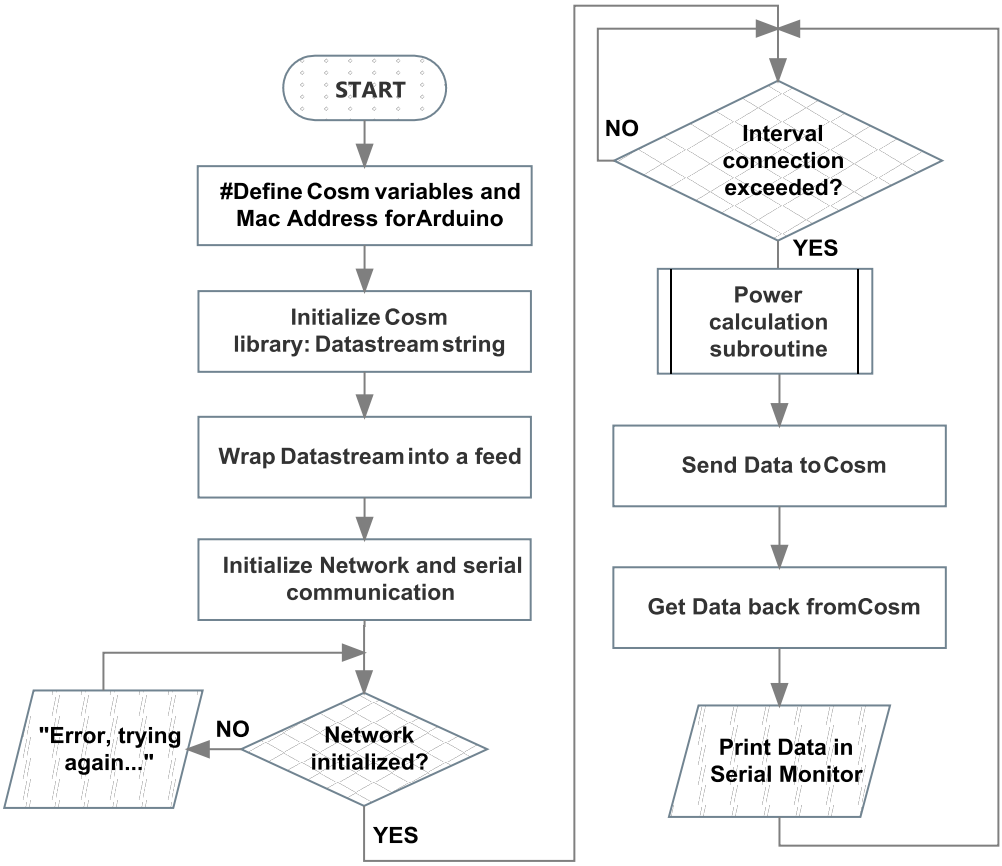


Fig.28 *Cosm code flow diagram*

In this diagram it can be perceived that this code calls the power calculation subroutine explained in figure 23. It can also be observed the lack of code to graph the data. This is because once the data is received by the *Cosm* feed, the own website graphs it by itself.

5.2.2 Final decision: Storing data with *PHP* script

However, as it was explained in the methodology chapter 4, this is not the best solution. Seeking independency is one of the aims of this project. Consequently, the option of storing the data in a local web-server and graph it later was finally developed.

There will be two methods of visualizing the data: the first one is in real time and the second one is the last day data. Therefore, two scripts are needed to store the data. One is in charge of storing the real time data, while the other only stores the data at the end of the day. It gathers all the data from the last day and stores it in other table.

Next diagram shows the flow of the *PHP* script that store the data in real time:

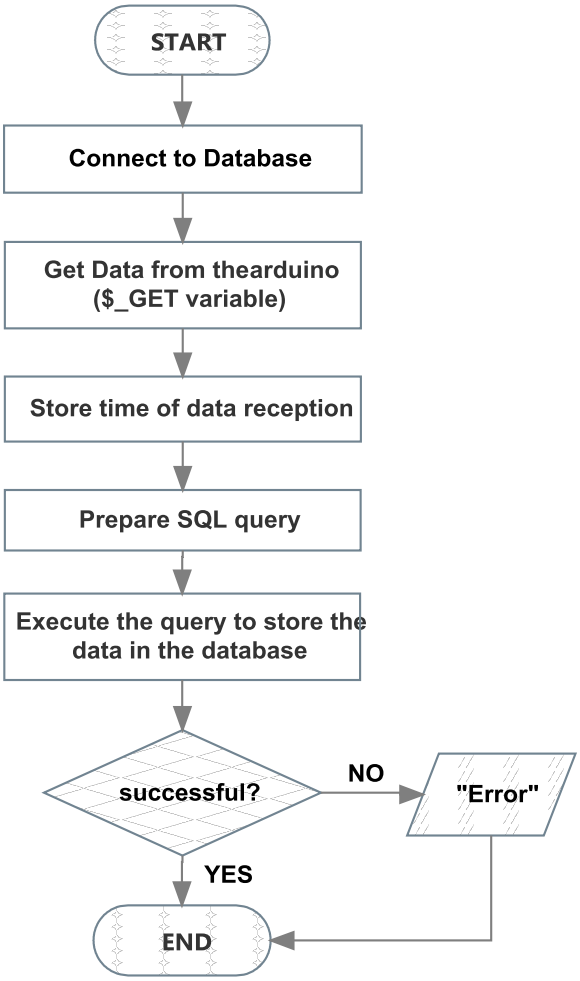


Fig.29 *Receive and store real time data in database flow diagram*

The *PHP* script for this diagram flow is available in Appendix B (page 55)

The script that stores the data from the last day is executed just once a day. This is done by a programmed task in the local server that is in charge of running this *PHP* script every day at midnight. The data is stored in the database 7 days. Then it is deleted to leave free space. Another programmed task is in charge of this. The *PHP* script code that is executed by this programmed task is in Appendix B (page 65), as well as the .bat files in charge of this duty.

Next diagram shows the flow of the *PHP* script in charge of storing the data from the last day. Code is available in Appendix B (page 56)

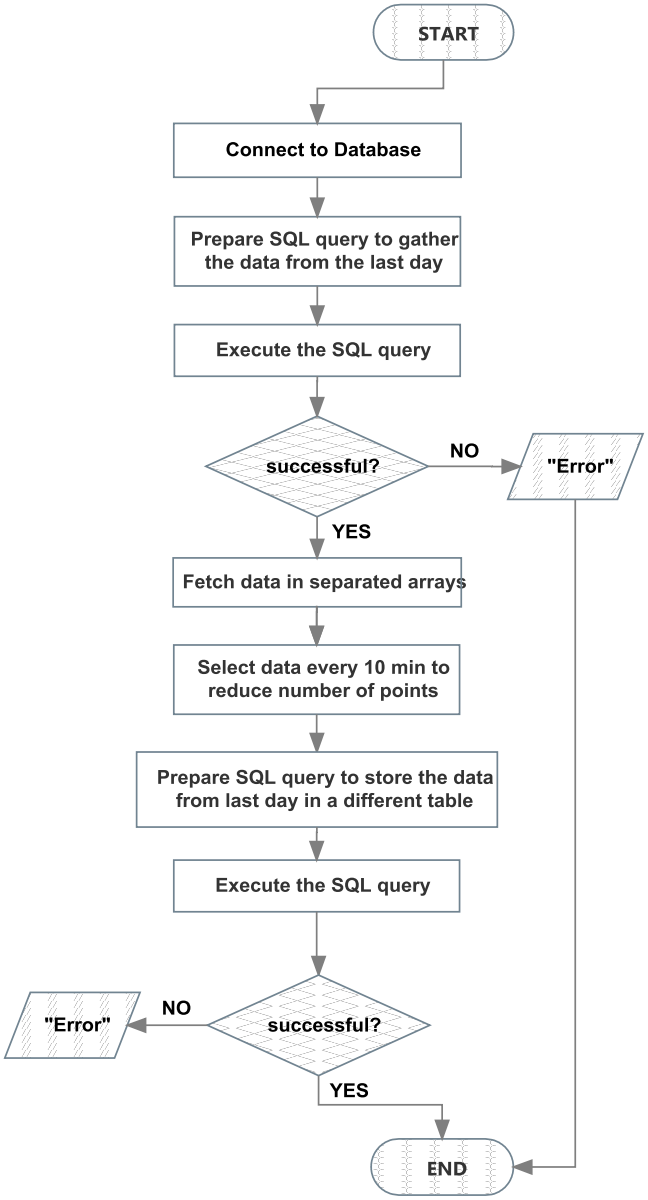


Fig.30 *Store data from last day flow diagram*

5.3 Graphing the data design

5.3.1 Data collecting and formatting

Once the data is in the database, a *PHP* script is in charge of pulling it from the database and gives it the correct format to be understandable later by the *JavaScript* that graphs it. As there will be two kinds of graphs (real time and last day graphs) each of them requires different formatting and different data. Therefore, two *PHP* scripts are needed.

Real time data collecting

Figure 31 shows the flow of this script:

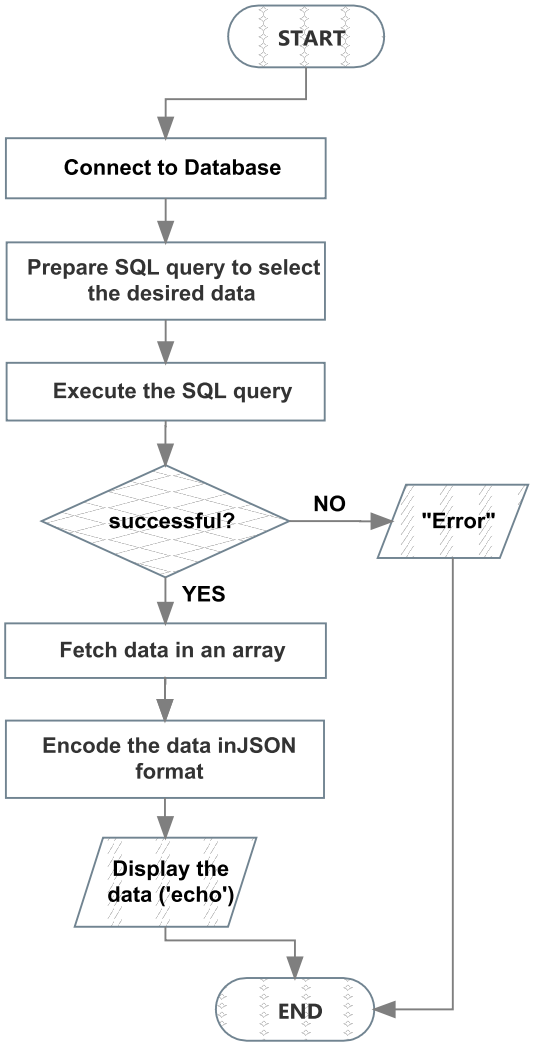


Fig.31 *Fetch and display real time data flow diagram*

This flow diagram shows that the data is encoded in *JSON*[1] format. *JSON* is a lightweight data-interchange format [30]. It displays the data in a format that is easy to read by humans and also easy to parse and generate to machines.

The data was formatted in this way thinking in the following step of the project: making the graphs. This would make easier the adding of points in the graphs using *Highcharts*. The code is available in Appendix B (page 57)

1. *JavaScript Object Notation*

Last day data

In this case the data of the last day is displayed in a graph. The data stored by the *PHP* script explained in figure 30 is formatted and displayed by this PHP script. Figure 32 shows the flow of the script. The code is available in Appendix B (page 58)

Figure 32 shows the diagram flow of the script that selects the data and format it so it can be later understood by *jQuery*.

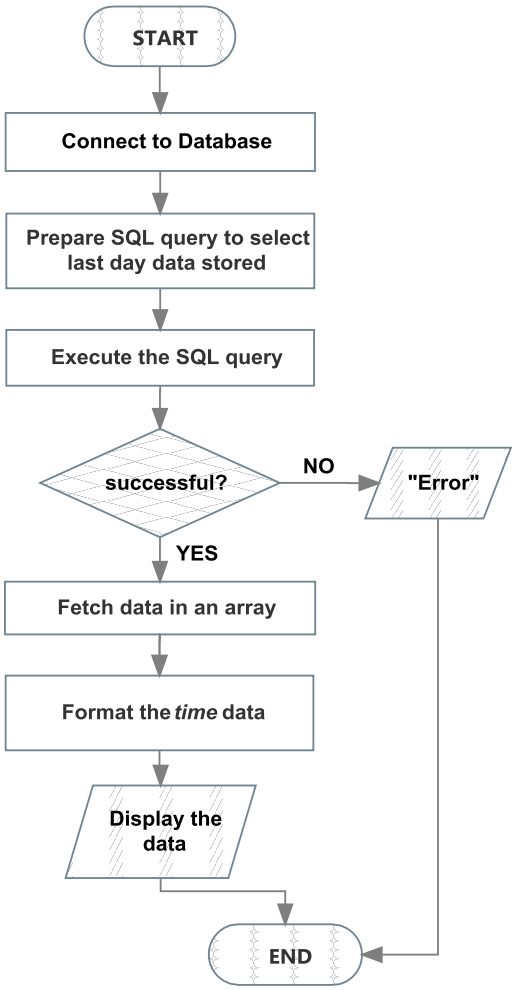


Fig.32 *Display data from last day flow diagram*

The code is available in Appendix B (page 58)

5.3.2 Graphing the data

Two types of modes for visualizing the data are available: real time and last day graphs.

Real time graph:

Once the data is encoded it is ready to be added to the graph. Here is the diagram flow of the *JavaScript* in charge of this duty. The code is in Appendix B (page 58)

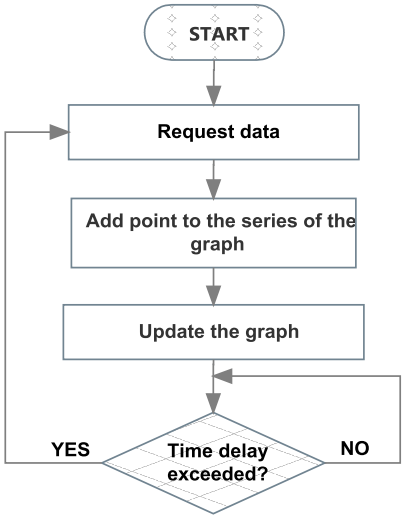


Fig.33 *Graph real time data flow diagram*

For requesting the data a function is programmed. This function loads the *PHP* script that displays the data to capture it and add a new point in the graph. Every certain period of time the script "reloads", requesting new data to update the graph in real time. This update period should match the sampling time of the Arduino micro-controller main program.

Last day graph:

The flow of the program is really similar to the last one explained. The main difference is that it is only executed when the user want to visualize these kind of graphs.

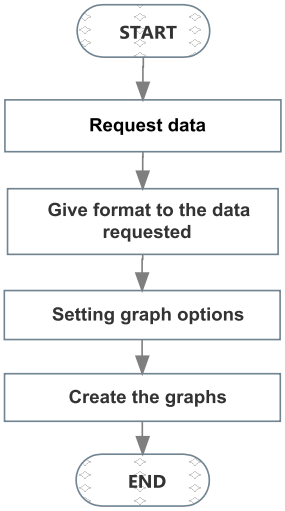


Fig.34 *Graph data from last day flow diagram*

The code for this flow diagram is in Appendix B (page 62)

5.4 Unifying all. The main program of the Arduino microcontroller

Finally all comes together as shown in the next diagram. The code is available in Appendix A (page 46)

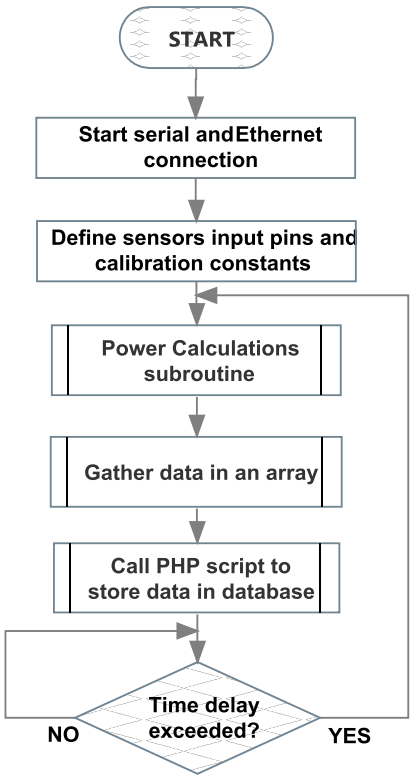


Fig.35 *Main Arduino program flow diagram*

6. Results and discussion

Results are shown in a dedicated website. The main features offered by this website are:

- **Simplicity**: the user is not saturated by a lot of text or options to choose. There are just the graphs wanted and one option to choose between the modes of monitoring available.
- **User friendly interface:** graphs that combine perfectly utility with a modern style.
- **Clarity of data representation:** clear graphs that allow selecting points to know its exact value and time.

The website offers two main modes of monitoring the consumption: real-time consumption or last day consumption. There are four graphs available: real power, voltage, current and power factor.

6.1 Interface

Data is updated automatically without the need of refreshing the website thanks to *JavaScript*.

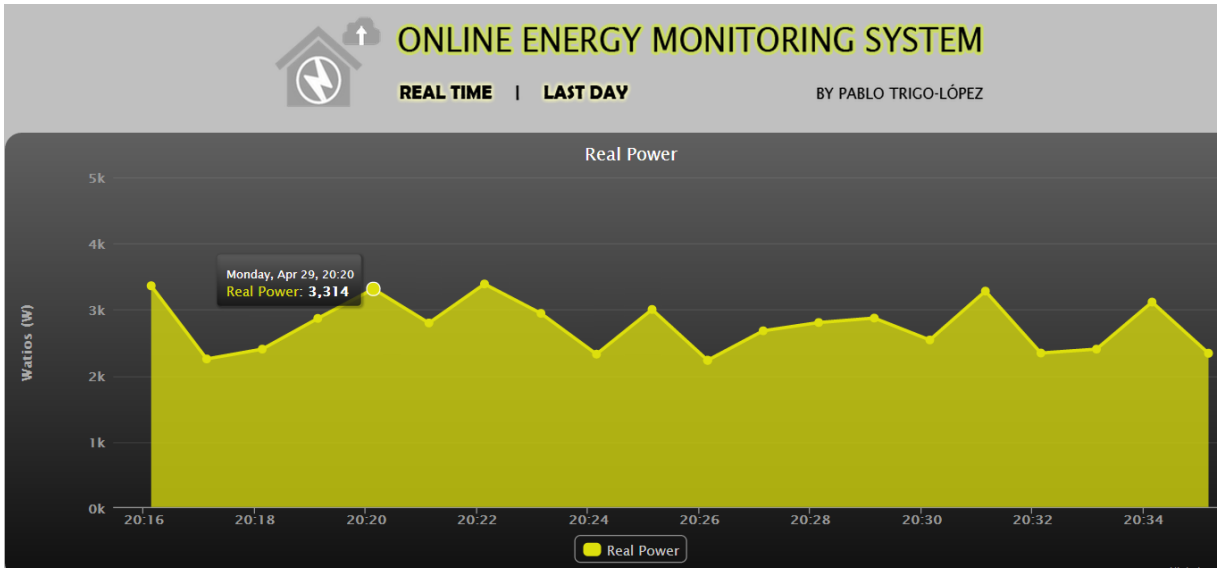


Fig.36 *Real Power graph (real time)*

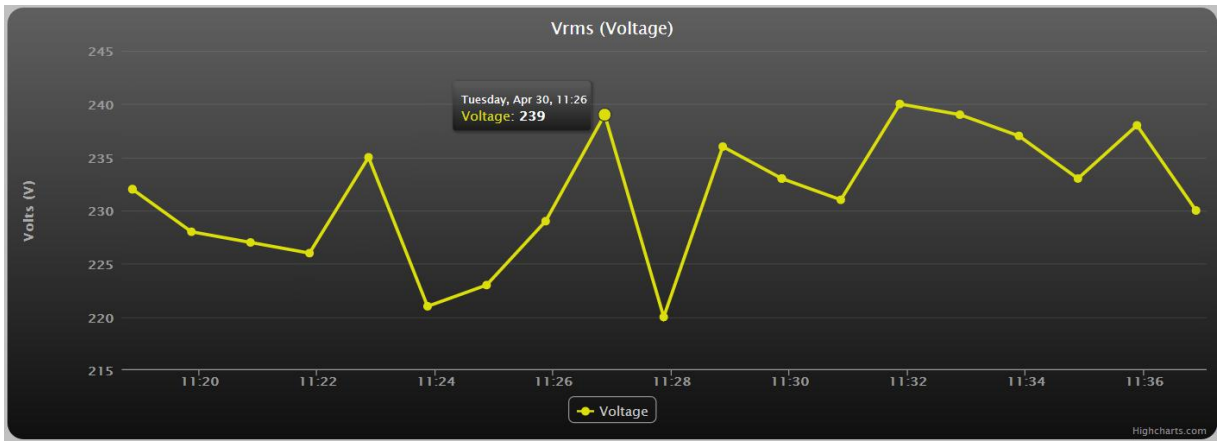


Fig.37 *Voltage graph (real time)*

The current and power factor graphs have exactly the same format as well as the last day graphs. The only difference of these last ones is that they show data from the last day and they do not refresh by themselves. The user is the one who refresh the website.

7. Conclusions

In general, it can be said that the main aim of this project was reached. A complete system for online monitoring of the energy consumption was finally completed. Along all the researching and designing process several difficulties have been encountered but they have been overcome with perseverance and thoroughness.

A good solution for sensing the power was designed. The power consumption was captured correctly connecting a Hall Effect sensor and step-down transformer to the Arduino micro-controller. Afterwards, the data from the sensors was processed by the micro-controller to calculate the desired variables. Next the data was stored in a database using *PHP* scripts to be later given a correct format by another script so it can be properly graphed using *JavaScript* (*jQuery*). The final result is a simple website that displays real time graphs as well as last day consumption graphs.

7.1 Further work

Nonetheless, as any project it is always open for new improvements. A further work could be designing a system to measure the current and voltage with non-contact sensors. The actual risk of manipulating high voltages should be lessened. Measuring the current with non-contact sensors was, in fact, achieved. However, for sensing the voltage a converter was needed. This requires direct contact with the electric wires. Although it provides isolation due to its magnetic nature, better solutions should be found to enhance the whole system. For instance, the electrostatic voltmeter mentioned in chapter 4 (4.1.2 voltage sensing) could be the most optimum solution.

Besides, the perfect solution would be using wireless connection to send the data from the micro-controller to the server database, instead of using an Ethernet connection. This manner, the server is not required to be next to the sensors. This idea was considered at the beginning, but it was not implemented due to lack of time considering its complexity.

In addition, another further work could be designing printed circuit boards (PCBs) for the signal conditioning circuits. The next step would be integrating all the pieces needed in this project in an electromagnetically shielded housing.

References

[1] US. Department of Energy [Online], Available: http://apps1.eere.energy.gov/news/progress_alerts.cfm/pa_id=580, last accessed 1/04/2013.

[2] *Manage energy* (2010) "Smart-metering domestic energy saving" [Online], Available: http://www.managenergy.net/resources/1416, last accessed 1/02/2013.

[3] *Cliff Jao and Xi Juo* (2008) "PowerBox: The safe AC Power Meter" [Online], Available: https://courses.cit.cornell.edu/ee476/FinalProjects/s2008/cj72_xg37/cj72_xg37/index.html, last accessed 1/04/2013.

[4] *Ken Bongort and Adam McCann* (2011) "XBee RF Smart Energy Compliant Power Meter", [Online], Available: http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/kjb79_ajm232/pmeter/index.html, last accessed 1/04/2013.

[5] Google (2011) "Google Power Meter: A Google.org Project" [Online], Available: http://www.google.com/powermeter/about/, last accessed 1/04/2013.

[6] Open Energy Monitor [Online], Available: http://openenergymonitor.org/emon/, last accessed 1/04/2013.

[7] *Arduino homepage* [Online], Available: http://arduino.cc/, last accessed 1/04/2013.

[8] *Cosm homepage*[Online], Available: http://cosm.com/, last accessed 1/04/2013.

[9] P3 International, "Kill A Watt" [Online], Available: http://www.p3international.com/products/special/p4400/p4400-ce.html, last accessed 1/04/2013.

[10] *William Koon* from Analog Devices, Inc "Current sensing for energy metering", [Online], Available: http://www.analog.com/static/imported-files/tech_articles/16174506155607IIC_Paper.pdf, last accessed 2/04/2013.

[11] *Paul Emerald, "*Non-Intrusive Hall-Effect Current-Sensing Techniques Provide Safe, Reliable Detection and Protection for Power Electronics", [Online PDF], Available: http://www.allegromicro.com/~/media/Files/Technical-Documents/STP98-1-Non-Intrusive-Hall-Effect-Current-Sensing-Techniques.ashx, last accessed: 2/04/2013.

[12] "CT Selection guide" [Online], Available: http://www.littelfuse.com/products/relays-controls-and-systems/protection-relays/protection-relay-pages/ct-selection-guide.aspx, last accessed 2/04/2013.

[13] *Honeywell*, "Hall Effect sensing and application" [Online PDF], Available: http://sensing.honeywell.com/index.*PHP*?ci_id=47847, last accessed 2/04/2013.

[14] *ABB*, "Current sensors Voltage Sensors" [Online PDF], Available: http://www05.abb.com/global/scot/scot209.nsf/veritydisplay/178528477d112c9bc12578c70042fa4e/$file/1sbc140152c0203%20-%20cat%20capteurs-br.pdf, last accessed 2/04/2013.

[15] *Dr, Maciej A. Noras*, "Non-contact surface charge/voltage measurements Fieldmeter and voltmeter methods" [Online], Available: http://www.trekinc.com/pdf/3002-field-voltmeter.pdf, last accessed 14/04/2013.

[16] *Arduino Project*, "ArduinoBoardEthernet" [Online], Available: http://arduino.cc/en/Main/ArduinoBoardEthernet, last accessed 3/04/2013.

[17] *WIZnet*, "I W5100", [Online], Available: http://www.wiznet.co.kr/Sub_Modules/en/product/Product_Detail.asp?cate1=5&cate2=7&cate3=26&pid=1011, last accessed 3/04/2013.
[18] *Cosm*, "Power Consumed Feed" [Online], Available: https://cosm.com/feeds/13743, last accessed 10/04/2013.
[19] *Netcraft*, "May 2012 Web Server Survey" [Online], Available: http://news.netcraft.com/archives/2012/05/02/may-2012-web-server-survey.html, last accessed 14/04/2013.
[20] *The Apache Software Foundation*, "Apache HTTP Server Project" [Online], Available: http://httpd.apache.org/, last accessed 3/04/2013.
[21] *Wikipedia*, "Client-server model", [Online], Available: http://en.wikipedia.org/wiki/Client-server, last accessed 4/04/2013.
[22] *PHP Homepage* [Online], Available: http://*PHP*.net/, last accessed 14/04/2013.
[23] *MySQL Homepage* [Online], Available: http://www.mysql.com/ last accessed 4/04/2013.
[24] *Wikipedia*, "Comparison of database tools" [Online], Available: http://en.wikipedia.org/wiki/Comparison_of_database_tools, last accessed 4/04/2013
[25] *jQuery* [Online], Available: http://jquery.com/, last accessed 4/04/2013
[26] *Highcharts* [Online], Available: http://www.highcharts.com/, last accessed 4/04/2013
[27] Allegro MicroSystems, Inc. [Online], Available: http://datasheet.elcodis.com/pdf2/84/73/847398/acs712.pdf , last accessed 24/04/2013
[28] *Wikipedia*, "High-pass filter" [Online], Available: http://en.wikipedia.org/wiki/High-pass_filter#Discrete-time_realization, last accessed 7/04/2013
[29] "Making accurate ADC readings on the Arduino" [Online], Available: http://hacking.majenko.co.uk/node/57, last accessed 7/04/2013
[30] *JSON* [Online], Available: http://www.json.org, last accessed 5/04/2013

Appendix A – Arduino code

***Cosm* program (flow diagram in figure 28)**

This is the main code for Arduino micro-controller using *Cosm.com* to graph the data

```
#include <SPI.h>
#include <Ethernet.h>
#include <HttpClient.h>
#include <EnergyLib.h>   // Include Energy library. It includes functions to simplify the power calculations
#include <Cosm.h>
PowerMonitor PowerBox;   // Create an instance of the class PowerMonitor (EnergyLib)

#define API_KEY "b2T9CblnPKbIAr_a8lahylZ4eI6SAKw5OUplS1duVVFyST0g" //  Cosm API key. Needed to connect to Cosm.com
#define FEED_ID 104737 //  Cosm feed ID. Change it if you change your feed.

// MAC address for the Ethernet board
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xB6, 0x8E };

unsigned long lastConnectionTime = 0;              // last time connected to Cosm
const unsigned long connectionInterval = 15000;      // delay between connecting to Cosm in milliseconds (15secs)

// Initialize the Cosm library. Define the string for our datastream ID
char data0[] = "Energy";
char data1[] = "Voltage";
char data2[] = "Current";
char data3[] = "PowerFactor";

CosmDatastream datastreams[] = {
  CosmDatastream(data0, strlen(data0), DATASTREAM_FLOAT),
  CosmDatastream(data1, strlen(data1), DATASTREAM_FLOAT),
  CosmDatastream(data2, strlen(data2), DATASTREAM_FLOAT),
  CosmDatastream(data3, strlen(data3), DATASTREAM_FLOAT),
};

// Wrap the datastream into a feed
CosmFeed feed(FEED_ID, datastreams, 4 /* number of datastreams */);

//Initialize an instance of Cosmclient and ethernetClient classes.
EthernetClient client;
CosmClient cosmclient(client);

//Initialize network, start serial communication, define input sensors pins and calibration
```

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Cosm Sensor Client Example");
  Serial.println("==========================");

  Serial.println("Initializing network");
  while (Ethernet.begin(mac) != 1) {
    Serial.println("Error getting IP address via DHCP, trying again...");
    delay(15000);
  }

  Serial.println("Network initialized");
  Serial.println();
  PowerBox.Vcalibrate(A0, 155.5, 1.7);  // Voltage: input pin, calibration, phase_shift
  PowerBox.Icalibrate(A1, 3.53);       // Current: input pin, calibration.
}

//Main program. Calculate power and upload the data to Cosm.com
void loop()
{
  if (millis() - lastConnectionTime > connectionInterval) {

  PowerBox.calcVI(20,2000);            // Calculate all. No.of half wavelengths (crossings),
time-out
  //PowerBox.serialprint();            // Print out all variables
  PowerBox.DataToUpload();
  sendData(PowerBox.dataToBeUploaded);  // send data to Cosm
  getData();                          // read the datastream back from Cosm
  lastConnectionTime = millis();       // update connection time so we wait before connecting
again
  }
}

// send the supplied value to Cosm, printing some debug information as we go
void sendData(double theData[]) {

  datastreams[0].setFloat(theData[0]);
  datastreams[1].setFloat(theData[1]);
  datastreams[2].setFloat(theData[2]);
  datastreams[3].setFloat(theData[3]);

  Serial.print("Power  ");
  Serial.println(datastreams[0].getFloat());
  Serial.print("Vrms  ");
  Serial.println(datastreams[1].getFloat());
```

```
  Serial.print("Irms  ");
  Serial.println(datastreams[2].getFloat());
  Serial.print("PowerFactor  ");
  Serial.println(datastreams[3].getFloat());

  Serial.println("Uploading to Cosm");
  int ret = cosmclient.put(feed, API_KEY);
  Serial.print("PUT return code: ");
  Serial.println(ret);

  Serial.println();
}

// get the value of the datastream from Cosm, printing out the value we received
void getData() {
  Serial.println("Reading data from Cosm");

  int ret = cosmclient.get(feed, API_KEY);
  Serial.print("GET return code: ");
  Serial.println(ret);

  if (ret > 0) {
    Serial.print("Datastream0 is: ");
    Serial.println(feed[0]);
    Serial.print("Power value is: ");
    Serial.println(feed[0].getFloat());

    Serial.print("Datastream1 is: ");
    Serial.println(feed[1]);
    Serial.print("Vrms value is: ");
    Serial.println(feed[1].getFloat());

    Serial.print("Datastream2 is: ");
    Serial.println(feed[2]);
    Serial.print("Irms value is: ");
    Serial.println(feed[2].getFloat());

    Serial.print("Datastream3 is: ");
    Serial.println(feed[3]);

    Serial.print("PowerFactor value is: ");
    Serial.println(feed[3].getFloat());
  }

  Serial.println();
}
```

Main program (flow diagram in figure 35)

This is the final version of the main code of the Arduino micro-controller. It sends the data to a database instead of using Cosm.com

```
//* Pablo Trigo-López
//
// Sketch that measures the power consumption of a device
// and store the data in a server. Later this data is plotted in a graph
//
//*

 #include <SPI.h>
 #include <Ethernet.h>
 #include "EnergyLib.h" // Include Energy library. It includes functions to simplify the
program. It is an open-source library.

 PowerMonitor PowerBox;  // Create an instance of the class PowerMonitor of Energylib.h

 //Ethernet configurations
 byte mac[] = {  0x90, 0xA2, 0xDA, 0x0D, 0xB6, 0x8E }; // Ethernet board MAC
 byte ip[] = {144,173,31,2 };     // Local ip direction
 byte server[] = {144,173,31,97}; // Server ip direction
 EthernetClient client;          // Create an instance of the class EthernetClient

 //Pines definition. For more information see the ethernet board datasheet
 const int inV = A0; //Voltage input sensor pin.
 const int inA = A1; //Current input sensor pin

 //Constants to control the frequency of data storage.
 unsigned long lastConnectionTime = 0;           // Last time connected to the server
 const unsigned long connectionInterval = 55000;  // Delay between connecting to the server
in milliseconds (55 secs)

 //General Configuration. Serial and ethernet connection. Calibration.
 void setup()
 {
   Serial.begin(9600); //Start Serial connection
   Serial.println("PowerBox Client Demostration");
   Serial.println("============================");
   Ethernet.begin(mac, ip); //Start the Ethernet connection
   delay(1000); //Give the Ethernet shield a second to initialize
   PowerBox.Vcalibrate(inV, 1.84, 1.7);  // Voltage: input pin, calibration, phase_shift
   PowerBox.Icalibrate(inA, 0.3);      // Current: input pin, calibration.
 }
```

```
//Main program. Required data is calculated and sent to the server database
void loop()
{
 if (millis() - lastConnectionTime > connectionInterval)
 {
    PowerBox.calcVI(20,2000); // Calculate all. No.of half wavelengths (crossings), time-out
    PowerBox.serialprint();   // Print out all variables
    PowerBox.DataToUpload();  // Load data in arrays
    sendData(PowerBox.dataToBeUploaded); //Store the data in the database
    lastConnectionTime = millis(); // Update connection time so we wait before connecting
again
  }
}

void sendData(double theData[]) {
  Serial.println("Connecting...");
  // If you get a connection, report back via serial:
  if (client.connect(server,80)) {
    // Make a HTTP request:
    client.print("GET /arduino/getData.PHP?rp="); // Send data using GET method
    client.print(theData[0]); //RealPower
    client.print("&vrms=");
    client.print(theData[1]); //Vrms
    client.print("&irms=");
    client.print(theData[2]); //Irms
    client.print("&pf=");
    client.print(theData[3]); //PowerFactor
    client.println(" HTTP/1.0");
    client.println("User-Agent: Arduino 1.0");
    client.println();
    Serial.println("Sucessful connection. Data stored");
  }
  else
  {
    Serial.println("Connection failed");
  }
  // if the server's disconnected, stop the client:
  if (client.connected()) {}
  else
  {
    Serial.println("Disconnected");
  }
  client.stop();
  client.flush();
}
```

EnergyLib.h (Class PowerMonitor)

This is the library used to calculate the data (real power, current, voltage, etc). It was obtained by slightly modifying EmonLib.h. This library is available at: https://github.com/openenergymonitor/EmonLib. **This code was not written by the author of this project. It was just slightly modified from the EmonLib**

Libraries are formed by two files (.h and .cpp). The .h extension file is the header that contains definitions for the library. Basically is a list of everything the library contains. The .cpp extension file is the source code of the library, the real code.

```
/*
  EnergyLib.h - Library created by Pablo Trigo modifying EmonLib.h
  (Created by Trystan Lea, April 27 2010) GNU GPL
*/

#ifndef EnergyLib_h
#define EnergyLib_h
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

class PowerMonitor
{
  public:

    //--------------------------------------------------------------------
    // Declare sensors pins and calibration
    //--------------------------------------------------------------------

    void Vcalibrate(int _inPinV, double _VCAL, double _PHASECAL);
    void Icalibrate(int _inPinI, double _ICAL);

    //--------------------------------------------------------------------
    // Calculate power, print results, prepare the data to be uploaded
    //and store data in the database
    //--------------------------------------------------------------------

    void calcVI(int crossings, int timeout);
    void serialprint();
    void DataToUpload();

    //--------------------------------------------------------------------
    // Function to read real ADC convertor high level
    //--------------------------------------------------------------------
```

```
    long readVcc();

    //--------------------------------------------------------------------
    // Useful value variables
    //--------------------------------------------------------------------

    double realPower,
      apparentPower,
      powerFactor,
      Vrms,
      Irms;

    //--------------------------------------------------------------------
    // Variable declaration for Collect data and prepare it to be uploaded
    //--------------------------------------------------------------------

    double dataToBeUploaded[4];

  private:

    //Set Voltage and current input pins
    int inPinV;
    int inPinI;
    //Calibration coeficients
    //These need to be set in order to obtain accurate results
    double VCAL;
    double ICAL;
    double PHASECAL;

    //--------------------------------------------------------------------
    // Variable declaration for Power calculation procedure
    //--------------------------------------------------------------------
    int lastSampleV,sampleV;   //sample_ holds the raw analog read value, lastSample_ holds
the last sample
    int lastSampleI,sampleI;
    double lastFilteredV,filteredV;        //Filtered_ is the raw analog value minus the DC offset
    double lastFilteredI, filteredI;
    double phaseShiftedV;                  //Holds the calibrated phase shifted voltage.
    double sqV,sumV,sqI,sumI,instP,sumP; //sq = squared, sum = Sum, inst = instantaneous
    int startV;                           //Instantaneous voltage at start of sample window.
    boolean lastVCross, checkVCross;  //Used to measure number of times threshold is crossed.
    int crossCount;

};#endif
```

EnergyLib.cpp (Includes PowerCalculations subroutine. Flow diagram in figure 23)

The .cpp extension file is the source code of the library, the real code. Here it is the PowerCalculation subroutine.

```
/*
  EnergyLib.cpp - Library modified by Pablo Trigo using EmonLib.cpp
  (Created by Trystan Lea, April 27 2010)
  GNU GPL
*/

//#include "WProgram.h" un-comment for use on older versions of Arduino IDE
#include "EnergyLib.h"
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

//--------------------------------------------------------------------------
// Sets the pins to be used for voltage and current sensors. Introduce
// the adecuaquate calibration constant.
//--------------------------------------------------------------------------
void PowerMonitor::Vcalibrate(int _inPinV, double _VCAL, double _PHASECAL)
{
  inPinV = _inPinV;
  VCAL = _VCAL;
  PHASECAL = _PHASECAL;
}

void PowerMonitor::Icalibrate(int _inPinI, double _ICAL)
{
  inPinI = _inPinI;
  ICAL = _ICAL;
}

//--------------------------------------------------------------------------
// POWER CALCULATIONS
// Calculates realPower,apparentPower,powerFactor,Vrms,Irms,kwh increment
// From a sample window of the mains AC voltage and current.
// The Sample window length is defined by the number of half wavelengths or crossings we
choose to measure.
//--------------------------------------------------------------------------
void PowerMonitor::calcVI(int crossings, int timeout)
{
 int SUPPLYVOLTAGE = readVcc();
 int crossCount = 0;        //Used to measure number of times threshold is crossed.
```

```
  int numberOfSamples = 0;    //This is now incremented

  //-------------------------------------------------------------------
  // 1) Waits for the waveform to be close to 'zero' (500 adc) part in sin curve.
  //-------------------------------------------------------------------
  boolean loop=true;            //an indicator to exit the while loop

  unsigned long start = millis();//millis()-start makes sure it doesnt get stuck in the loop if there
is an error.

  while(loop==true)             //the while loop...
  {
    startV = analogRead(inPinV);//using the voltage waveform
    if ((startV < 550) && (startV > 440)) loop = false;  //check its within range
    if ((millis()-start)>timeout) loop = false;
  }

  //-------------------------------------------------------------------
  // 2) Main measurment loop
  //-------------------------------------------------------------------
  start = millis();

  while ((crossCount < crossings) && ((millis()-start)<timeout))
  {
    numberOfSamples++;            //Count number of times looped.
    lastSampleV=sampleV;          //Used for digital high pass filter
    lastSampleI=sampleI;          //Used for digital high pass filter
    lastFilteredV = filteredV;    //Used for offset removal
    lastFilteredI = filteredI;    //Used for offset removal

    //-----------------------------------------------------------------
    // A) Read in raw voltage and current samples
    //-----------------------------------------------------------------
    sampleV = analogRead(inPinV);     //Read in raw voltage signal
    sampleI = analogRead(inPinI);     //Read in raw current signal

    //-----------------------------------------------------------------
    // B) Apply digital high pass filters to remove 2.5V DC offset (centered on 0V).
    //-----------------------------------------------------------------
    filteredV = 0.996*(lastFilteredV+sampleV-lastSampleV);
    filteredI = 0.996*(lastFilteredI+sampleI-lastSampleI);

    //-----------------------------------------------------------------
    // C) Root-mean-square method voltage
    //-----------------------------------------------------------------
    sqV= filteredV * filteredV;           //1) square voltage values
```

```
  sumV += sqV;                    //2) sum

  //-----------------------------------------------------------------
  // D) Root-mean-square method current
  //-----------------------------------------------------------------
  sqI = filteredI * filteredI;          //1) square current values
  sumI += sqI;                    //2) sum

  //-----------------------------------------------------------------
  // E) Phase calibration
  //-----------------------------------------------------------------
  phaseShiftedV = lastFilteredV + PHASECAL * (filteredV - lastFilteredV);

  //-----------------------------------------------------------------
  // F) Instantaneous power calc
  //-----------------------------------------------------------------
  instP = phaseShiftedV * filteredI;          //Instantaneous Power
  sumP +=instP;                   //Sum

  //-----------------------------------------------------------------
  // G) Find the number of times the voltage has crossed the initial voltage
  //    - every 2 crosses we will have sampled 1 wavelength
  //    - so this method allows us to sample an integer number of half wavelengths which
increases accuracy
  //-----------------------------------------------------------------
  lastVCross = checkVCross;
  if (sampleV > startV) checkVCross = true;
                else checkVCross = false;
  if (numberOfSamples==1) lastVCross = checkVCross;

  if (lastVCross != checkVCross) crossCount++;
 }

 //-------------------------------------------------------------------------
 // 3) Post loop calculations
 //-------------------------------------------------------------------------
 //Calculation of the root of the mean of the voltage and current squared (rms)
 //Calibration coeficients applied.

 double V_RATIO = VCAL *((SUPPLYVOLTAGE/1000.0) / 1023.0);
 Vrms = V_RATIO * sqrt(sumV / numberOfSamples);
 double I_RATIO = ICAL *((SUPPLYVOLTAGE/1000.0) / 1023.0);
 Irms = I_RATIO * sqrt(sumI / numberOfSamples);

 //Calculation power values
 realPower = V_RATIO * I_RATIO * sumP / numberOfSamples;
```

```
  apparentPower = Vrms * Irms;
  powerFactor=realPower / apparentPower;

  //Reset accumulators
  sumV = 0;
  sumI = 0;
  sumP = 0;
//--------------------------------------------------------------------
}

void PowerMonitor::serialprint()
{
   Serial.print("  RealPower: ");
   Serial.println(realPower);
   Serial.print("  Vrms: ");
   Serial.println(Vrms);
   Serial.print("  Irms: ");
   Serial.println(Irms);
   Serial.print("  PowerFactor: ");
   Serial.println(powerFactor);
   Serial.print("  FilteredV: ");
   Serial.println(filteredV);
   delay(100);
}

void PowerMonitor::DataToUpload()
{
   dataToBeUploaded[0]=realPower;
   dataToBeUploaded[1]=Vrms;
   dataToBeUploaded[2]=Irms;
   dataToBeUploaded[3]=powerFactor;
}

//Measuring the real Vcc suply of the ADC converter for precise calculations
long PowerMonitor::readVcc() {
  long result;

  #if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328__) || defined
(__AVR_ATmega328P__)
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  #elif defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) ||
defined(__AVR_ATmega2560__)
  ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  #elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) ||
defined(__AVR_ATtiny84__)
  ADMUX = _BV(MUX5) | _BV(MUX0);
```

```
  #elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) ||
defined(__AVR_ATtiny85__)
  ADMUX = _BV(MUX3) | _BV(MUX2);
  #endif

  delay(2);                // Wait for Vref to settle
  ADCSRA |= _BV(ADSC);             // Convert
  while (bit_is_set(ADCSRA,ADSC));
  result = ADCL;
  result |= ADCH<<8;
  result = 1126400L / result;          //1100mV*1024 ADC steps
http://openenergymonitor.org/emon/node/1186
  return result;
}
```

Appendix B – *PHP* and *JavaScript* code

Config.php script (Connecting to the database)

This *PHP* script is the essential code to connect to the database. It would be included in every *PHP* script.

```
<?PHP
//Database data
$host   = "localhost";
$db = "test";
$user  = "root";
$pass = "*****";
//Establish a connection
$conn = new PDO("mysql:host=$host;dbname=$db",$user,$pass);
?>
```

GetData.php script (Receiving the data. Flow diagram in figure 29)

This *PHP* script captures the real time data from the Arduino and store it in the database.

```
<?PHP
include("config.php");
//Getting the data from the arduino
$RealPower = $_GET["rp"];
$Vrms = $_GET["vrms"];
$Irms = $_GET["irms"];
$PowerFactor = $_GET["pf"];
//Preparing the sql query
$sql = "INSERT INTO energy(RealPower, Vrms, Irms, PowerFactor) VALUES (:realpower,
:vrms, :irms, :powerfactor)";
// Prepares and stores the SQL statement in query
$query = $conn->prepare($sql);
//Safe method to pass parameters and reject SQL injections.
$query->bindParam(':realpower', $RealPower);
$query->bindParam(':vrms', $Vrms);
$query->bindParam(':irms', $Irms );
$query->bindParam(':powerfactor', $PowerFactor);
//Executing the query
 if($query->execute()){
  //good
}else{
  echo "Error executing the query";
}
?>
```

getData1day.php script (flow diagram in figure 30)

This script selects all the data from the previous day and stores it in other table reducing the number of points.

```
<?php
/*This script is executed by a programmed task of the local server. It is executed every day at
midnight
to store the data from one day */

//Include the file to connect to the database
include("config.php");
//Preparing the SQL query. Selecting all the data from the previous day (-1 DAY)
$sql = "SELECT Time,RealPower,Vrms,Irms,PowerFactor FROM energy WHERE
(DATE(Time) = DATE_ADD(CURDATE(),INTERVAL -1 DAY))";
//Storing the SQL statement in $query
$query = $conn->prepare($sql);
//Executing the query
if($query->execute()){
  //fetch the data in an array
   $results = $query->fetchALL(PDO::FETCH_ASSOC);
   //Organize the data in differente arrays
   foreach($results as $result){
       $time[] = $result['Time'];
       $realPower[] = $result['RealPower'];
       $vrms[] = $result['Vrms'];
       $irms[] = $result['Irms'];
       $powerFactor[] = $result['PowerFactor'];
   }
}else{
   echo "Error executing the query 1";
}
//Storing points every 10 minutes instead of every minute.
$timeMean = get10min($time,10);
$realpowerMean = get10min($realPower, 10);
$vrmsMean = get10min($vrms, 10);
$irmsMean = get10min($irms, 10);
$powerfactorMean = get10min($powerFactor, 10);
$completeMean=array($timeMean,$realpowerMean,$vrmsMean,$irmsMean,$powerfactorM
ean);
//Inserting the last day values in the table energy1day
 for($i=0;$i<=count($completeMean)-1;$i=$i+1){
   $sql = "INSERT INTO energy1day(Time,RealPower,Vrms,Irms,PowerFactor) VALUES
(:time,:realpower,:vrms,:irms,:powerfactor)";
   // Prepares and stores the SQL statement in query
   $query = $conn->prepare($sql);
```

```
    //Safe method to pass parameters and reject SQL injections.
    $query->bindParam(':time', $completeMean[0][$i]);
    $query->bindParam(':realpower', $completeMean[1][$i]);
    $query->bindParam(':vrms', $completeMean[2][$i]);
    $query->bindParam(':irms', $completeMean[3][$i] );
    $query->bindParam(':powerfactor', $completeMean[4][$i]);
    //Executing the query
    if($query->execute()){
    }else{
      echo "Error executing the query";
    }
}
//Function to get points every 10 minutes instead of 1 minute.
function get10min($data, $minutes){
    for($cont=0; $cont< count($data); $cont=$cont+$minutes){
        $mean[] = $data[$cont];
    }
    return $mean;
}
?>
```

DisplayDataRT.php script (Data collecting and formatting. Flow diagram in figure 31)

This *PHP* script pulls back the data from the database and prepare it in a specific format to be graphed later.

```
<?php
header('Content-Type: application/json');
include("config.php");

//Preparing the sql query  for selecting the data. It only selects the last row stored.
$sql = "SELECT UNIX_TIMESTAMP(Time),RealPower,Vrms,Irms,PowerFactor FROM
energy ORDER BY id DESC LIMIT 1";
// Prepares and stores the SQL statement in query
$query = $conn->prepare($sql);
//Executing the query
if($query->execute()){
    //fetch the data in an array
    $data=$query->fetch(PDO::FETCH_NUM);
    //Preparing the data in separate arrays
    $powerArray=array($data[0]*1000,$data[1]);
    $vrmsArray=array($data[0]*1000,$data[2]);
    $irmsArray=array($data[0]*1000,$data[3]);
    $powerfactorArray=array($data[0]*1000,$data[4]);
 $completeArray=array($powerArray,$vrmsArray,$irmsArray,$powerfactorArray);
```

```
  //Encoding into json format. This is needed to use the data with highcharts script
   $jsonResult=json_encode($completeArray);
   //Deleting the double quotes from the json strings and printing them.
   //This is necessary due the method used to graph the data with highcharts.
   echo str_replace("", '', $jsonResult);
   //echo $jsonResult;
}else{
    echo "Error executing the query";
}
?>
```

DisplayData1day.php script (Data collecting and formatting. Flow diagram in figure 32)

```
<?php

//Making the code suitable for all the graphs
$type = $_GET["type"];
$validParams = array('RealPower', 'Vrms', 'Irms', 'PowerFactor');
if(!in_array($type, $validParams)){
   die('Error!');
}
include("config.php");
//Selecting just the last day data
$sql = "SELECT * FROM energy1day WHERE (DATE(Time) =
DATE_ADD(CURDATE(),INTERVAL -1 DAY))";
// Prepares and stores the SQL statement in query
$query = $conn->prepare($sql);
//Executes the query
if($query->execute()){
   //Fetch the data in an array
   $results = $query->fetchAll(PDO::FETCH_ASSOC);
   //Show the data in a correct format
   foreach($results as $result){
      $uts=strtotime($result['Time']);//convert Time to Unix Timestamp
      $date=date("l, F j, Y H:i:s",$uts);
      echo $date. "\t" . $result[$type]. "\n";
   }
}
?>
```

GraphRT.js *JavaScript* (Graphing the data. Flow diagram in figure 33)

This *JavaScript* is in charge of requesting the data from the displayData *PHP* script and graph it using Highcharts.

```
$(document).ready(function() {
/**
 * Request data from the server, add it to the graph and set a timeout to request again
 */
 function requestData() {
   $.ajax({
      url: '/arduino/displayDataRT.php',
      success: function(point) {
      var series1 = chart1.series[0];
      //point contains the data of displayData.php
      var shift1 = series1.data.length > 20; // shift if the series is longer than 20
       var series2 = chart2.series[0],
           shift2 = series2.data.length > 20; // shift if the series is longer than 20
       var series3 = chart3.series[0],
            shift3 = series3.data.length > 20; // shift if the series is longer than 20
         var series4 = chart4.series[0],
            shift4 = series4.data.length > 20; // shift if the series is longer than 20
         // add the point in each graph
         chart1.series[0].addPoint(point[0], true, shift1); //point[0] contains time and
RealPower
         chart2.series[0].addPoint(point[1], true, shift2); //point[1] contains time and Vrms
         chart3.series[0].addPoint(point[2], true, shift3); //point[2] contains time and Irms
         chart4.series[0].addPoint(point[3], true, shift4); //point[3] contains time and
PowerFactor
         // call it again after 55 secs
         setTimeout(requestData, 55*1000);
      },
      cache: false
   });
 }
 requestData();

 //Default settings for all the graphs.
 var defaultOptions={
   chart: {
      defaultSeriesType: 'line',
   },
   xAxis: {
      type: 'datetime',
      tickPixelInterval: 100,
       minRange: 1200*1000 //20 minutes
   },
   yAxis: {
      minPadding: 0.2,
      maxPadding: 0.2,
      title:{
```

```
        margin:50
      }
    }
};

//Specific settings for the RealPower graph
var chartPowerOptions={
   chart: {
      renderTo: 'RealPower',
      defaultSeriesType: 'area'
   },
   title: {
      text: 'Real Power'
   },

   yAxis: {
       title: {
          text: 'Watios (W)'
       }
   },
   series: [{
         name: 'Real Power',
         data: []
        }]
};
//Specific settings for the Voltage graph
var chartVrmsOptions={
   chart: {
      renderTo: 'Voltage'
   },
   title: {
      text: 'Vrms (Voltage)'
   },
   yAxis: {
      title: {
        text: 'Volts (V)'
      }
   },
   series: [{
          name: 'Voltage',
          data: []
         }]
};
//Specific settings for the Current graph
var chartIrmsOptions={
   chart: {
```

```
        renderTo: 'Current'
    },
    title: {
        text: 'Irms (Current) '
    },
    yAxis: {
         title: {
             text: 'Amperes (A)'
         }
    },
    series: [{
             name: 'Current',
             data: []
           }]
};
 //Specific settings for the PowerFactor graph
 var chartPFactorOptions={
    chart: {
        renderTo: 'PowerFactor'
    },
    title: {
        text: 'PowerFactor'
    },
    yAxis: {
         title: {
             text: 'PowerFactor',
         }
    },
    series: [{
             name: 'PowerFactor',
             data: []
           }]
};

//Extending the specific options: Adding the defaultoptions to every specific options
 chartPowerOptions = jQuery.extend(true, {}, defaultOptions, chartPowerOptions);
 chartVrmsOptions = jQuery.extend(true, {}, defaultOptions, chartVrmsOptions);
 chartIrmsOptions = jQuery.extend(true, {}, defaultOptions, chartIrmsOptions);
 chartPFactorOptions = jQuery.extend(true, {}, defaultOptions, chartPFactorOptions);

//Creating the graphs
 var chart1 = new Highcharts.Chart(chartPowerOptions);   //RealPower graph
 var chart2 = new Highcharts.Chart(chartVrmsOptions);    //Voltage graph
 var chart3 = new Highcharts.Chart(chartIrmsOptions);    //Current graph
 var chart4 = new Highcharts.Chart(chartPFactorOptions); //PowerFactor graph
});
```

Graph1day.js *JavaScript* (Graphing the data. Flow diagram in figure 34)

This *JavaScript* is in charge of requesting the data from the displayData1day *PHP* script and graph it using Highcharts to obtain the last day graphs.

```
 var chart;
$(document).ready(function() {
      //Options for the graph style. Highcharts code
      var options = {
         chart: {
            defaultSeriesType: 'line'
         },
         title: {
         },
         xAxis: {
            type: 'datetime',
            minRange: 3600*24*1000, //24 hours
         },
         yAxis: {
            minPadding: 0.2,
            maxPadding: 0.2,
            title: {
               margin:50
            },
         },
         tooltip: {
            formatter: function() {
               return Highcharts.dateFormat('%a, %b %e, %H:%M:%S', this.x)+'<b><br/>'+
this.series.name +': </b>'+'<span style="color:yellow">'+Highcharts.numberFormat(this.y,
2)+'</span>';
            }
         },
         series: [{
            name: ''
         }]
      }
   //Calling the functiong "graph" for each variable.
   graph('RealPower');
   graph('Vrms');
   graph('Irms');
   graph('PowerFactor');

   //Function Graph
   // Load data asynchronously using jQuery. On success, add the data
   // to the options and initiate the chart.
   // This data is obtained by exporting a GA custom report to TSV.
   // http://api.jquery.com/jQuery.get/
```

```
function graph(type){
    jQuery.get('displayData1day.php?type=' + type, null, function(tsv) {
        var lines = [];
        traffic = [];
        try {
            // split the data return into lines and parse them
            tsv = tsv.split(/\n/g);
            jQuery.each(tsv, function(i, line) {
                line = line.split(/\t/);
                date = Date.parse(line[0] +' UTC');
                traffic.push([
                    date,
                    parseInt(line[1].replace(',', ''), 10)
                ]);
            });
        } catch (e) {  }
        //Adding the data to the graph, setting some graph options and creating the graph
        options.series[0].data = traffic;
        options.chart.renderTo  = type;
        options.chart.defaultSeriesType = getChartType(type); //Selecting the chart type
        options.title.text = getTitle(type); //Selecting the title of the graph
        options.yAxis.title.text= getTitleYAxis(type); //Selecting the title of the Y Axis
        options.series[0].name = type; //Selecting the name of the series
        chart = new Highcharts.Chart(options);  //Creating the graph
    });
}

//Function to determine the title of each graph
function getTitle(type){
    switch (type){
        case "RealPower":
            return "Real Power";
        case "Vrms":
            return "Vrms (Voltage)";
        case "Irms":
            return "Irms (Current)";
        case "PowerFactor":
            return "Power Factor";
        default:
            return "";
    }
}
//Function to determine the title of the Y Axis of each graph
function getTitleYAxis(type){
    switch (type){
        case "RealPower":
```

```
            return "Watios (W)";
        case "Vrms":
            return "Volts (V)";
        case "Irms":
            return "Amperes (A)";
        case "PowerFactor":
            return "PowerFactor";
        default:
            return " ";
    }
  }
  //Function to determine the type of the chart of each graph
  function getChartType(type){
    switch (type){
        case "RealPower":
            return "area";
        case "Vrms":
            return "line";
        case "Irms":
            return "line";
        case "PowerFactor":
            return "line";
        default:
            return " ";
    }
  }
});
```

IndexRT.php file (Website)

This code is the main website that shows all the real time graphs.

```
<html>
<head>
<title>Highcharts example Real Time</title>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
<script src="http://code.highcharts.com/highcharts.js"></script>
<script type="text/javascript" src="js/themes/gray.js"></script>
<script type="text/javascript" src="/arduino/graphRT.js"></script>

</head>
<body bgcolor="#C0C0C0">
<center><img src="/images/logo.png" alt="Energy" width="656" height="98"
usemap="#planetmap"><center></p>
```

```
<map name="planetmap">
<area shape="realtime" coords="120,70,205,83"  href="indexRT.php" alt="Real Time">
<area shape="lastday" coords="240,70,323,83" href="/arduino/index1day.php" alt="Last
day"></map>
<div id="RealPower" style="min-width: 400px; height: 400px; margin: 0 auto"></div><br>
<div id="Voltage" style="min-width: 400px; height: 400px; margin: 0 auto"></div><br>
<div id="Current" style="min-width: 400px; height: 400px; margin: 0 auto"></div><br>
<div id="PowerFactor" style="min-width: 400px; height: 400px; margin: 0 auto"></div>
</body>
</html>
```

Index1day.php file (Website)

This code is the main website that shows all the last day graphs.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Highcharts Last day example</title>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
<script src="http://code.highcharts.com/highcharts.js"></script>
<script type="text/javascript" src="../js/themes/gray.js"></script>
<script type="text/javascript" src="/arduino/graph1day.js"></script>

</head>
<body bgcolor="#C0C0C0">
<center><img src="/images/logo.png" alt="Energy" width="656" height="98"
usemap="#planetmap"><center></p>
<map name="planetmap">
<area shape="realtime" coords="120,70,205,83" href="../indexRT.php" alt="Real Time">
<area shape="lastday" coords="240,70,323,83" href="/arduino/index1day.php" alt="Last
day"></map>
<div id="RealPower" style="min-width: 400px; height: 400px; margin: 0 auto"></div><br>
<div id="Vrms" style="min-width: 400px; height: 400px; margin: 0 auto"></div><br>
<div id="Irms" style="min-width: 400px; height: 400px; margin: 0 auto"></div><br>
<div id="PowerFactor" style="min-width: 400px; height: 400px; margin: 0 auto"></div>
</body>
</html>
```

delete7days.php script

This *PHP* script is executed by a programmed task every 7 days to delete all the data from the last 7 days. This is done to leave free space in the database after a while.

```php
<?php
/* This script is executed by a programmed task of the local server every 7 days.
It deletes all the data stored in the last 7 days to make space*/
//Include the file to connect to the database
include("config.php");
//Preparing the first SQL query.
$sql = "DELETE * FROM energy";
//Storing the SQL statement in $query
$query = $conn->prepare($sql);
//Executing the query
if($query->execute()){
  console.log('Energy table data deleted')
   }
}else{
   echo "Error executing the 1st query";
}
//Preparing the second SQL query.
$sql = "DELETE * FROM energy1day";
//Storing the SQL statement in $query
$query = $conn->prepare($sql);
//Executing the query
if($query->execute()){
  console.log('Energy1day table data deleted')
   }
}else{
   echo "Error executing the 2nd query";
}
?>
```

Task1day.bat script (for the every day programmed task)

This file is need to program the task that runs every day at midnight. It executes the 'getData1day' *PHP* script that stores the data at the end of the day in other table (script from the flow diagram of figure 30)

```
@ECHO OFF
C:\Server\PHP\php.exe -f "C:\Server\WEB\arduino\getData1day.php"
```

Task7days.bat script (for the 7 day programmed task)

This file is needed to program the task that runs every 7 days. It executes the 'delete7days' *PHP* script that deletes the data of the database.

```
@ECHO OFF
C:\Server\PHP\php.exe -f "C:\Server\WEB\arduino\delete7days.php"
```