Multiple interface Automation

A Major Project Semester Report

Submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

by

I K Vibhav (13EE118)

Alex Biswas (13EE153)

Manjit Kalgutkar (13EE154)

Saba Hussain *(*13EE254*)*

Under the guidance of

Dr Sheron Figarado

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE 575025

April, 2017

DECLARATION

We hereby declare that the Project work entitled: Multiple interface Automation, taken up as a Major Project to the Department of Electrical and Electronics Engineering is a record of original work done by our team under the guidance of: Dr Sheron Figarado, Assistant Professor Department of Electrical and Electronics Engineering, NITK Surathkal.

All the material and information included in the report and analysis have been duly acknowledged.

The results embodied in this study have not been submitted to any other University or Institute for the award of any degree.

Date – 28th April, 2017

Name of Team Members:

I K Vibhav (13EE118)
Alex Biswas (13EE153)
Manjit Kalgutkar (13EE254)
Saba Hussain (13EE255)

CERTIFICATE

This is to certify that the Project Report titled “Multiple Interface Automation” is a bonafide work carried out by:-

I K Vibhav (13EE118)
Alex Biswas (13EE153)
Manjit Kalgutkar (13EE254)
Saba Hussain (13EE255)

(Group ID 13EEG10)

The Students of 8th semester B.Tech Electrical & Electronics Engineering, National Institute of Technology Karnataka-Surathkal. This project has been completed under the guidance of Dr Sheron Figarado, Assistant Professor Department of Electrical and Electronics Engineering, NITK Surathkal.during the academic year 2016-17 as a Major Project (EE499).

ACKNOWLEDGEMENT

We would like to take this opportunity to express our profound gratitude and deep regards to: Dr. Sheron Figarado - Assistant Professor, Department of Electrical and Electronics Engineering for giving us this opportunity and mentoring us throughout the course of the project. Without their continuous support, reference material and knowledge on the subject this project would not have been possible and seen the progress we have achieved today.

We would also like to thank our friends and family for their constant support and appreciation for our work. This project would not have been a success if it had not been for any of these mentioned people.

IK Vibhav, 13EE118
Alex Biswas, 13EE153
Manjit Kalgutkar, 13EE154
Saba Hussain, 13EE254

ABSTRACT

As part of Major Project done last, we hosted a website locally that could control appliances connected to switchboard with 4 sockets. Also we designed a circuit that could indicate the ON-OFF status of the appliance. Further we used the status obtained to help employ a staircase switching arrangement in this system which allowed both manual and online control of the appliances. We used Raspberry Pi as the controller here and used a software library called Cherry Py to host the website that could be accessed on mobile devices connected to the common router.

This semester for Major Project 2, the plan was to achieve two main goals, implement a system that can control multiple switchboards at a time with a common control dashboard hosted on the server, design a system to display approximate power usage by each appliance connected to the system on the dashboard.

The network of multiple switchboards controlled using wireless or serial communication between Raspberry Pi and Arduino using Master-Slave architecture is implemented. It involved reading up on the various methods of communication procedures between Master and Slave modules. The slave modules being switchboards enabled with multiple interface options for the control of loads and providing approximate power measurement for the port.

While the master module has still been fixated on R Pi, which would control these slave modules, Arduino was chosen as the slave module because of its ability to read up Analog values and a ADC converter for converting these Analog values sensed by the sensor employed to digital values. Thus making power measurement and transfer of these values to the master easier. The converted values are then communicated to the master module (Rpi).

Integration of the features of the project from previous semester was done into the project worked on for this semester resulting in a platform which enables the user the control and indicates of the status and power consumption values of the loads connected to various slaves.

CONTENTS

Table of Figures and tables

CHAPTER 1

INTRODUCTION

1.1 PYTHON ENVIRONMENT

Python is a widely used high-level which facilitates dynamic programming language for general purposes. Its syntax allows programmers to write code with can express the algorithms in fewer lines than in languages such as C++ or Java. The language thrives to enhance the readability. The language provides constructs for easily decipherable and writable long and short codes.
Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python can be compiled and interpreted on interpreters available for various operating systems allowing it to be used and shared among different platforms. Using third-party tools, such as Py2exe or Py Installer, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, so Python-based software can be distributed to, and used on, those environments with no need to install a Python interpreter.

In order to get on with working on Raspberry Pi, we had to make sure we were comfortable writing basic codes using python. Because to be able to control sensors and using the web framework using python later it was necessary to be comfortable with Python syntax. Since, this familiarity with the language was achieved in last semester, we had to struggle a lot less in editing the Cherry Py codes and necessary changes were made to the website which were left to be desired last semester.

1.2 RASPBERRY PI FEATURES

The Raspberry Pi 3 is the third generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. Compared to the Raspberry Pi 2 it has:

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Taking few of the features from its older Pi 2, it also has:

- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot (now push-pull rather than push-push)
- Video Core IV 3D graphics core

The Raspberry Pi 3 has an identical form factor to the previous Pi 2 (and Pi 1 Model B+) and has complete compatibility with Raspberry Pi 1 and 2. It is an open source hardware technology combined with a programming language and an Integrated Development Environment (IDE). The Raspberry Pi platform allows the user to create custom hardware and applications to control it via programming languages.
The Raspberry Pi has four distinct power modes:

- Run – The central processing unit (CPU) and all functionality of the ARM11 core are available and powered up.
- Standby – The main core clocks are shut down (the parts of the CPU that process instructions are no longer running) although the power circuits on the core are still active. In this mode, known as ‘‘Wait for Interrupt’’ (WFI) mode, the core can be quickly woken up by a process generating a special call to the CPU called an interrupt. This interrupt will stop any current processing and do what the calling process has asked for.
- Shutdown – The power is shut down.
- Dormant – The core is powered down but all the cache are left powered on.

1.3 RASPBERRY PI ACCESS

A new Raspberry pi unit out of the box is set to generate dynamic IP addresses at the Ethernet and Wi-Fi adapter. This is done because as a unit it’s meant to act as a CPU with a screen and I/O devices connected to it directly. But here we had to use it as a separate system remotely accessed by our laptops. So we needed a static IP address for the Wi-Fi adapter and the Ethernet adapter to access the system via Remote desktop access provided on Microsoft Windows 10 environment.
First we set the IP addresses of our laptop’s Wi-Fi and Ethernet adaptors static by using the Network and sharing settings. Next we edit the OS files of Raspberry pi related to networks and type in the desired constant IP addresses. Now we can directly enter the IP address of the raspberry pi’s adapter on the remote desktop access box on our computer and this shares the screen of raspberry pi.

1.4 CHERRYPY

CherryPy is a python based, object-oriented web framework. CherryPy allows developers to build web applications in much the same way they would build any other object-oriented Python program. This results in smaller source code developed in less time. CherryPy is now more than ten years old and it is has proven to be fast and reliable. It is being used in production by many sites, from the simplest to the most demanding.

The following features of Cherry Py are considered as its strengths −

- User Friendly: The main objective of CherryPy is to make it easier for the users to be able to develop a web framework using the usual syntax and indentations of python. CherryPy supports modular style of programming. Using correct logic concepts primary components can be managed and parent classes are expandable to child classes.
- Power: CherryPy leverages all the power of Python. It also provides tools and plugins, which are powerful extension points needed to develop world-class applications.
- Open-source: CherryPy is an open-source Python Web Framework (licensed under the open-source BSD license), which means this framework can be used commercially at ZERO cost. Thus benefitting the younger generation who can’t afford other web developing frameworks.

- Community Help: It has a devoted community which provides complete support with various types of questions and answers. The community tries to give complete assistance to the developers starting from the beginner level to the advanced level.
- Deployment: There are cost effective ways to deploy the application. CherryPy includes its own production-ready HTTP server to host your application. CherryPy can also be deployed on any WSGI-compliant gateway.

1.5 MASTER/SLAVE MODEL

Master/slave is a model of communication and control where one module has unidirectional control over the flow and content of communication between the module and other slave modules. In Electronic hardware space this method of control and communication is used where a module acts as a controller controlling and deciding the work done by the controlled modules. In short, the one controlling is Master and the ones being controlled are slaves. The most common example of this is the master/slave configuration of IDE disk drives attached on the same cable, where the master is the primary drive and the slave is the secondary drive.

The master/slave model is commonly used in all the major technology industries, be it electronic or mechanical. In electronic technology, it is often used to simplify communication like, instead of having a separate interface to communicate with each disk drive, we can connect most of them via one interface and cable and the computer has to give instructions/ orders to only one drive acting as the master and the orders are further propagated down to corresponding slave drives based on the work assigned to them, thus setting a hierarchy of instruction flow.

In mechanical technology, the term can refer to the configuration of motors such as two motors connected to different drives that are acting on the same load; one drive is defined as the master, doing the speed and control of the load, whereas the slave is there to help increase the torque. In pneumatic and hydraulic systems, we also have master cylinders, which feed the pressure and control the slave cylinders. Thus the work is divided among the slaves, while the master controls and communicates necessary details to the slaves.

The Master module in our project is Raspberry Pi. Arduino UNO's have been assigned the work of slave modules in the project, the flow of data and the importance and application of the data flowing has been described in later sections, the following part mentions the different ways in which communication can be carried out between Raspberry PI and Arduino UNO.

Raspberry Pi is great tool for embedded engineers but it lacks the lability to take in analog input from the external world and the necessary ADC to convert these analog values to corresponding digital values. Another drawback is all its IO's are 3.3V level. On the other side Arduino is good at sensing the analog values measured from the physical world using sensors. To be able to use current sensors in our particular project it was necessary to use a microcontroller with an ability to take in analog values. Thus Arduino Uno's were chosen to perform this job of converting the sensed Analog values and changing them to corresponding Digital values and sending these over to the Raspberry Pi.

1.6 INTERFACING MICROCONTROLLERS

We can divide this interfacing problem into two steps. There are many interface options available in both boards like UART, IIC and SPI. The choice of protocol depends on the requirement of the application and roles of the Arduino and Raspberry Pi.

1.6.1 USB

This way of communication between Raspberry Pi and Arduino employs the use of Serial Communication i.e. the data to be sent is made into a serial data and is sent over either way using the USB cable connected between the Modules. The flow of data is controlled by the Baud rates. The advantage of this was of communication is that the connections are very sturdy and very reliable, at suitable baud rates there is no data loss in the communication between the two microcontrollers. The downside is the USB port of RPi is used up for powering and serial communication with the Arduino.

1.6.2 UART

Raspberry Pi can be connected to Arduino using Tx-Rx, the Rx pin of Arduino is connected to Tx pin. The Rx pin of Raspberry Pi to Tx pin of Raspberry Pi.

This is method is similar to the above connection, but this time we are not using the USB port. The data is serially communicated between the devices making use of UART present in both the modules. The connections enable each module's ability to act as a transceiver. The downside of this method is that an extra hardware is required to run this safely – a logic level converter because of the running voltage level difference between the two devices.

Another method is to use GPIO pins to communicate data, this is a very trivial and unreliable method of communicating data between microcontrollers.

1.6.3 WIRELESS

Bluetooth modules can be connected to each of the microcontrollers, these are connected to Rx or TX pin based on the role they're going to be playing in the circuit. The data is serially transferred to receive or transmit buffer of the Bluetooth module by the Rx or TX pin and is transmitted over to the device to which the module is paired with. The advantage of this method is that it can be used to communicate over to mobiles with apps for using this data sent or sending data over Bluetooth for controlling applications attached to this Bluetooth module.

1.7 SERIAL COMMUNICATION

Since all these above mentioned methods include serial communication between two different modules, it is important to understand how this mode of communication works. Embedded electronics is all about interlinking circuits (processors or other integrated circuits) to create a symbiotic system. In order for those individual circuits to swap their information, they must share a common communication protocol. Hundreds of communication protocols have been defined to achieve this data exchange, and, in general, each can be separated into one of two categories: parallel or serial.

1.7.1 PARALLEL VS SERIAL

Parallel interfaces transfer multiple bits at the same time. They usually require **buses** of data - transmitting across eight, sixteen, or more wires. Data is transferred in huge, crashing waves of 1's and 0's.
Serial interfaces stream their data, one single bit at a time. These interfaces can operate on as little as one wire, usually never more than four.

Think of the two interfaces as a stream of cars: a parallel interface would be the 8+ lane mega highway, while a serial interface is more like a two-lane rural country road. Over a set amount of time, the mega-highway potentially gets more people to their destinations, but that rural two-lane road serves its purpose and costs a fraction of the funds to build

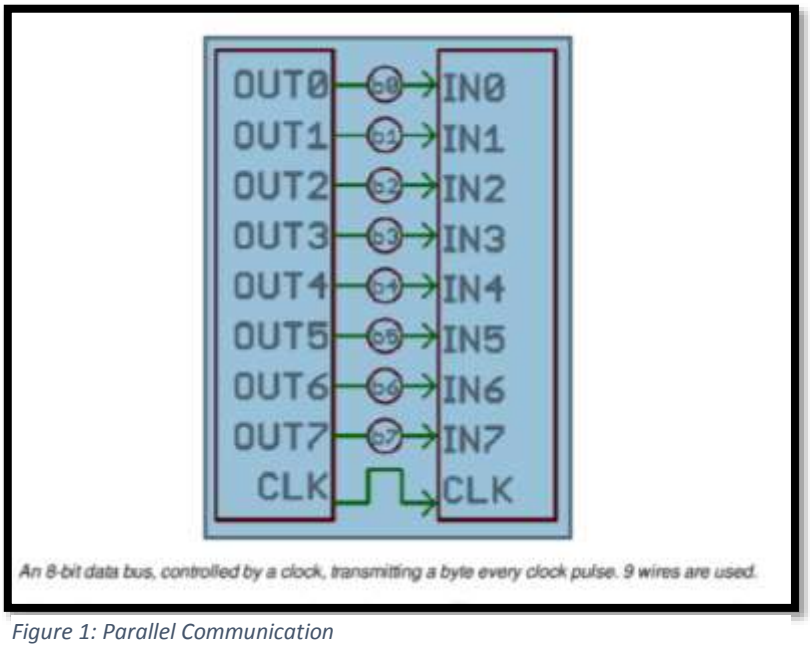


Figure 1: Parallel Communication

Parallel communication certainly has its benefits. It's fast, straightforward, and relatively easy to implement. But it requires many more input/output (I/O) lines but since the I/O lines on a microprocessor can be precious and few. So, we often opt for serial communication, sacrificing potential speed for pin real estate.
Over the years, dozens of serial protocols have been crafted to meet particular needs of embedded systems. USB (universal serial bus), and Ethernet, are a couple of the more well-known computing serial interfaces. Other very common serial interfaces include SPI, I^2C, and the serial standard. Each of these serial interfaces can be sorted into one of the two groups: Synchronous or Asynchronous.

A synchronous serial interface always pairs its data line(s) with a clock signal, so all devices on a synchronous serial bus share a common clock. This makes for a more straightforward, often faster serial transfer, but it also requires at least one extra wire between communicating devices. Examples of synchronous interfaces include SPI, and I^2C.

Asynchronous means that data is transferred without support from an external clock signal. This transmission method is perfect for minimizing the required wires and I/O pins, but it does mean we need to put some extra effort into reliably transferring and receiving data. The serial protocol is the most common form of asynchronous transfers.

1.7.2 RULES OF SERIAL

The asynchronous serial protocol has a number of built-in rules - mechanisms that help ensure robust and error-free data transfers. These mechanisms, which we get for eschewing the external clock signal, are:

- Data bits
- Synchronization bits
- Parity bits
- Baud rate

Through the variety of these signalling mechanisms, we find that there's no one way to send data serially. The protocol is highly configurable. The critical part is making sure that **both devices on a serial bus are configured to use the exact same protocols.**

Data bits are the bits which transfer the actual data, synchronization bits are bits which are added to the original data to facilitate serial transfer. These bits include Start and Stop bits, which help in detection and direction of data. Parity bits are used for checking error in the data.

***Baud Rate*: -** The baud rate specifies **how fast** data is sent over a serial line. It's usually expressed in units of bits-per-second (bps). If you invert the baud rate, you can find out just how long it takes to transmit a single bit. This value determines how long the transmitter holds a serial line high/low or at what period the receiving device samples its line. Baud rates can be just about any value within reason. The only requirement is that both devices operate at the same rate. One of the more common baud rates, especially for simple stuff where speed isn't critical, is **9600 bps**. Other "standard" baud are 1200, 2400, 4800, 19200, 38400, 57600, and 115200.

The higher a baud rate goes, the faster data is sent/received, but there are limits to how fast data can be transferred. Speeds exceeding 115200 is usually not seen - that's fast for most microcontrollers. Arduino's can operate at 2000000 with the COM port's serial monitor. But serial communication between different microcontrollers sometimes suffers from data loss.

1.8 STAIRCASE WIRING

Staircase wiring or multiway switching or two way light switch wiring is a commonly used connection to control a load from two different positions. In staircase wiring two SPDT (single pole double throw) switches i.e, two way switches are used, and its wiring is shown in the circuit below(Traveller system or common system).

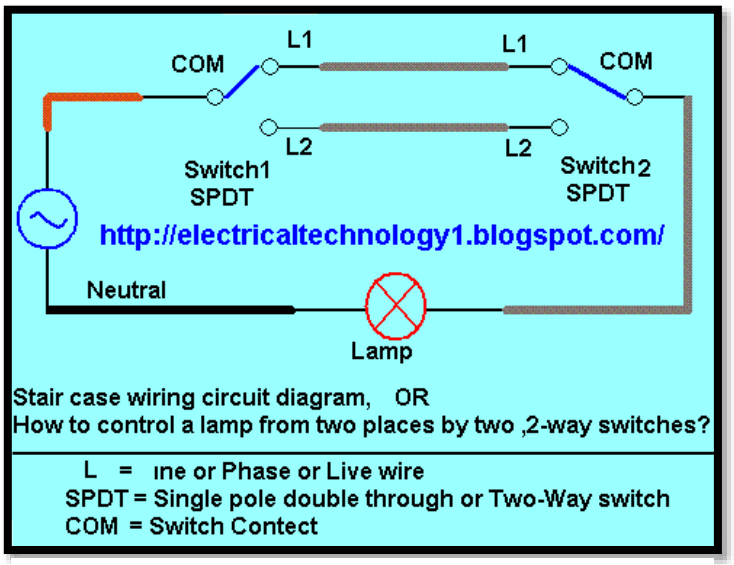


Figure 2: Staircase wiring

Staircase wiring makes it feasible for the user to turn ON and OFF the load from two different positions, its circuit arrangement is in such a way, from common pole we can switch to both 1 & 2 poles, the 1st pole is connected to 1st pole of next and 2nd pole to the 2nd pole of the next, i.e. both poles to corresponding pole of the other. The phase of supply is connected to the common pole of one switch and phase supply to the load is taken from the common pole of the next switch. So in such an arrangement to close the circuit both the switches should be in the same position in order to make the two common poles in contact to achieve a closed circuit. Changing ON & OFF condition of a single switch can determine whether the circuit is closed or open. Thus, in staircase wiring we can control from both positions. For the above Traveller system circuit the truth table of the connection will be similar to an XNOR gate, as the light ONs when both the switches are in the same position.

1.9 CURRENT SENSORS

- Current sensors measure AC and/or DC current.
- The sensors in this selection guide measure current and provide some sort of output.
- The most important distinction to make when selecting a current sensor is whether AC and/or DC current has to be measured.
- An important aspect to consider is if the sensor needs to be in-line with the circuit or if it has to work by being clamped around the wire to be measured.

1.9.1 CURRENT SENSOR TECHNOLOGY OPTIONS

The technology used by the current sensor is important because different sensors can have different characteristics for a variety of applications. Most sensors work on the principle that a current-carrying wire produces a magnetic field.

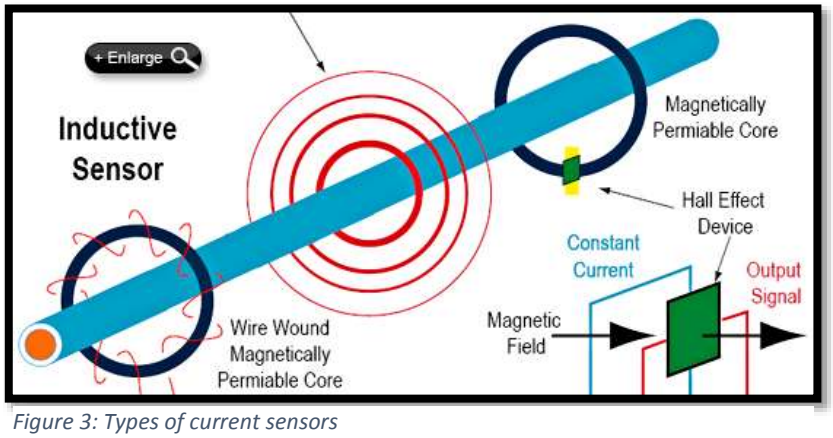


Figure 3: Types of current sensors

Different types of current sensors include: -

Hall Effect - Hall Effect sensors have of a core Hall Effect device and signal conditioning circuitry. It works when the current conductor passes through the core that concentrates the conductor's magnetic field. The Hall Effect device, mounted within the core, is at 90 degrees to the concentrated magnetic field and a constant current excites the Hall Effect device. The energized device is then exposed to the magnetic field from the core and gives a potential difference that can be measured and amplified into signals such as 4-20mA.

Inductive- These use a coil through which a current carrying wire passes. This causes a power to flow in the coil that is proportional to the current. This is due to the magnetic field produced by flowing current. These sensors used for AC currents. It has a wire-wound core and a signal conditioner. As the current conductor passes through, it becomes magnified by the conductor's magnetic field. Since AC current changes potential from negative to positive, it therefore creates an expanding and collapsing magnetic field. So, a current is induced in the windings. Secondary current is converted to a voltage and conditioned to output process; signals such as 4-20mA or contact closures.

Magnetoresistive- Magnetoresistive effect is the property of certain materials to change the value of its resistance as function of a magnetic field applied to it. If no magnetic flux is applied, current flows straight through the plate. If magnetic flux is applied, a Lorentz force proportional to the magnetic flux density deflects the current path. As current path is deflected, current flows through the plate for longer distance, causing resistance to be increased.

1.9.2 CURRENT SENSOR SPECIFICATIONS

Usually Current sensors receive current inputs and provide voltage outputs:

Performance specifications describe how a current sensor will operate in the desired environment. Following are some performance specification parameters.

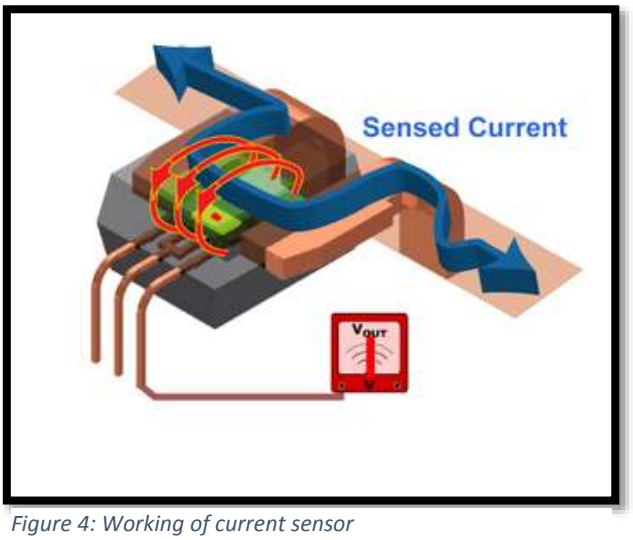


Figure 4: Working of current sensor

- **Measuring range** is the maximum current the current sensor is capable of measuring.
- **Input voltage** is the required voltage to operate the device.
- **Frequency range** specifies the range of values of the input frequency that the sensor can operate in.
- **Response time** is the interval between the application of an input excitation and the occurrence of the corresponding output signal.
- **Isolation voltage** describes the maximum voltage the sensor can handle to protect devices connected to it. Inaccurate measurements and damage occurs for Voltages above this specification.
- **Accuracy** measures the degree of closeness of the measured value to the actual value. Accurate measurements is ensured by regular calibrations on the Current sensors.
- **Operating temperature** describes the temperature range the sensor is designed to operate in.

The current sensor used in the following project till now is ACS712 20A. The following graphs

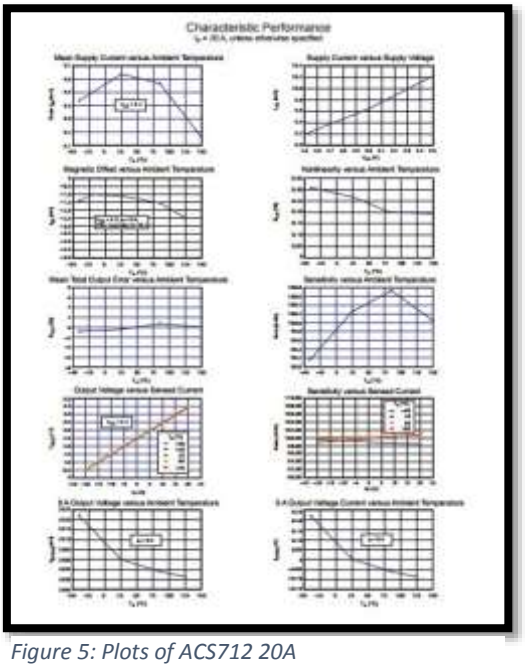


Figure 5: Plots of ACS712 20A

Electrical Appliance Typical Energy Consumption Table

Appliance	**Consumption (Watts)**
Air conditioner - Room	1000
Ceiling fan	40
Table fan	20
Blow dryer	1000
Laptop	20-60
PC	80-150
Lights:100 watt incandescent	100
Lights: Compact fluorescent 40 watt Incandescent equivalent	11
Lights: Compact fluorescent 60 watt Incandescent equivalent	16

Figure 6: Power consumption of various appliances

CHAPTER 2

SOFTWARE WORK

2.1 HTML WEBPAGE

The website we are currently hosting uses HTML5 and CSS. The Cascading style sheet has been used to make the website more presentable over the last version due to the following reasons:-

Responsive page
The site is responsive, meaning it can automatically adjust the spacing and position of divisions on the page based on the size of the screen it's being displayed on, or resizing of the browser window.
One way to make the page responsive is by using W3.CSS which is a modern CSS framework with built-in responsiveness.

Web framework
We've used the CherryPy framework which is a light and fast python web framework. Following were the options we considered and the reasons why CherryPy was selected.

1. Apache web server
Apache is an HTTP server that is the most commonly used web server software today. It allows the user to simply host multiple pages by dropping them in a specified location within the software.
Although it's quite easy to use, it's more suited to applications where web development and server management are areas of focus. We needed a framework that was easy to connect to the Raspberry Pi and help in the control of appliances using a single code block.

2. CherryPy
CherryPy framework is python based, is light and easy to work with and is a commonly used framework for applications like home automation. The entire page can be hosted and integrated with the control code in a single code block. Also the installation of the directory was quick and no extra IDE was needed.

Elements on the page

Button: We've used the form HTML tag to interconnect the user's command to the action. The button transfers control to functions "On" or "Off" which are part of the python code. To make the buttons more attractive and to ensure a better user interface, we have made use of CSS under the "style" tag.

Displaying power figures: The current sensor data coming from the Arduino is saved in a variable s0 and is displayed on the web page along with the appliance status. We've made use of only 2 current sensors in this project and have placed one on each of the boards. The data is displayed on the page by converting it into a string as it wasn't possible to directly transfer a value from python domain to HTML domain.

Graphs: We also intended to display a live graph of the power figures on the HTML page.

Following are the two frameworks or directories which can be used to do the same.

1. CanvasJS:
CanvasJS allows you to create dynamic charts that update at a given interval. Dynamic charts are useful in displaying data that changes with time like stock price, temperature, etc. Dynamic updates are supported by all chart types including line, area, column, bar, pie, etc. Below are some examples of HTML5 & JavaScript based dynamic charts. Dynamic charts are also referred to as live charts or real-time charts.

2. D3.JS
D3.js is a JavaScript library for manipulating documents based on data.D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

Although it was possible to use the above tools to achieve the task we faced certain limitations with respect to our hardware which we were unable to fix. Due to the new network administration setup by the Central Computer Centre, NITK or due to some problem with the Wifi adapter of our device, the Raspberry-Pi is unable to access internet and hence we couldn't install the directories via terminal. We also tried to find an alternative installation procedure but that did not work either.

2.2 SERIAL COMMUNICATION BETWEEN RPI AND ARDUINO

Current sensor data processed on the Arduino had to be sent to the master module, in this case the RaspberryPi. The method used was Serial communication over USB due to the following advantages:-

1) Minimal hardware and wiring connections required.

2) Serial communication packages was already readily installed in each device.

3) The RPi could provide power to the Arduino through the usb port by this method, thereby removing the need to use an extra power source.

To establish serial communication betwwen the devices, 2 separate codes have to be written in each RPi and Arduino. Each should have a matching baud rate to make the transmission perfect. We define the main commands exclusively used for this purpose in each microcontroller separately:-

RPI:-

```
import serial

ser = serial.Serial('/dev/ttyACM0', 9600)
```

In the above statement, we first import the package used to control the whole process of serial communication. We then move on to tell the browser from where to get serialvalues from. The argument – /dev/ttyACM0 is the name for the USB interface used. 9600 is the baud rate that is being chosen.

```
ser.readline()
```

The above statement is used to read values sent by the Arduino on the RPi.

ser.write('3')

The above statement is used to write values serially onto the Arduino. In the above case, 3 is being written.

Arduino: -

Serial.begin(9600);

The above statement is used to setup the speed of data transfer from the Arduino through the serial communication wire. It is placed in the void setup() function.

Serial.println(sample2);

The above statement is used to write of send values from the Arduino.

Serial.available()

The above statement is used to check if values are being sent from another device to the Arduino. The statement returns a TRUE if values are sent else it returns a FALSE. This is extremely important in order to avoid junk values being READ by the Arduino.

Serial.read()

The above statement is used to read values being sent to the Arduino.

In the project the following had to transferred

From Arduino to RPi – Computed RMS values of current based on the algorithm written in the Arduino. These RMS values have been calculated based on the values obtained from the Current Sensor on the ADC.

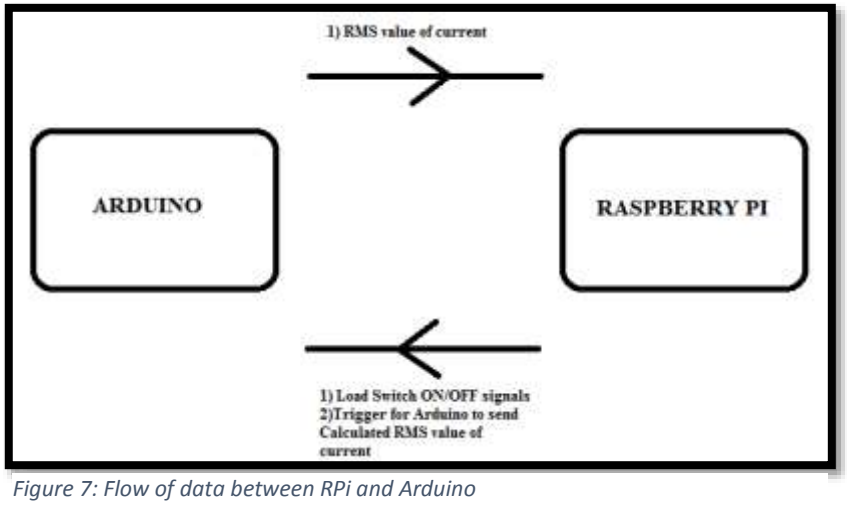


Figure 7: Flow of data between RPi and Arduino

From RPi to Arduino- Commands to Switch ON/OFF the Loads attached to the Arduino, Trigger Signal for the Arduino to send computed RMS current values.These values are sent at a baud rate of 9600. Upon checking in the COM port it was found that 160 samples were being taken in 20ms. The current value is sent by the Arduino to the RPI only when it gets a signal from the RPi to do so. Continuous sending of data was done before, but it was found that the entire system would crash after making it run for more than 3 minutes. 115200 was also tried as a baud rate value but the system would run very slow when this was done.

2.3 ARDUINO ALGORITHMS

The hall-effect current sensor was chosen as the current measurement device, as the conversion of the sensed voltage value from the current sensor by the Arduino has been discussed before. This section will be describing the various algorithms which were used to do the same and limitations faced and the scope for improvement in the same

First stage

The Aim initially was to take the samples from the current sensor and convert the samples into rms current through the algorithms and send it over to Raspberry Pi while also receiving the necessary switching details from Raspberry Pi. This was the ideal case of operation which allowed uninterrupted measurement and communication. The algorithms with the codes in which serial communication part was in the "Void loop ()" were tested using COM port Serial Monitor.

Baud rates were varied up-to 20, 00,000 bits/sec, and Giving 140 samples/cycle. The baud rate determines the efficiency of taking in data, since if the instructions involving serial data communication take up more time then the sampling rate reduces and chances of replicating the curve being sampled diminish. The problem first appeared when the serial communication of continuous data transfer at 115200 bits/sec was carried out between Raspberry PI and Arduino was carried out. The Arduino's transmission started hanging, these issues have been broadly explained in the previous section.

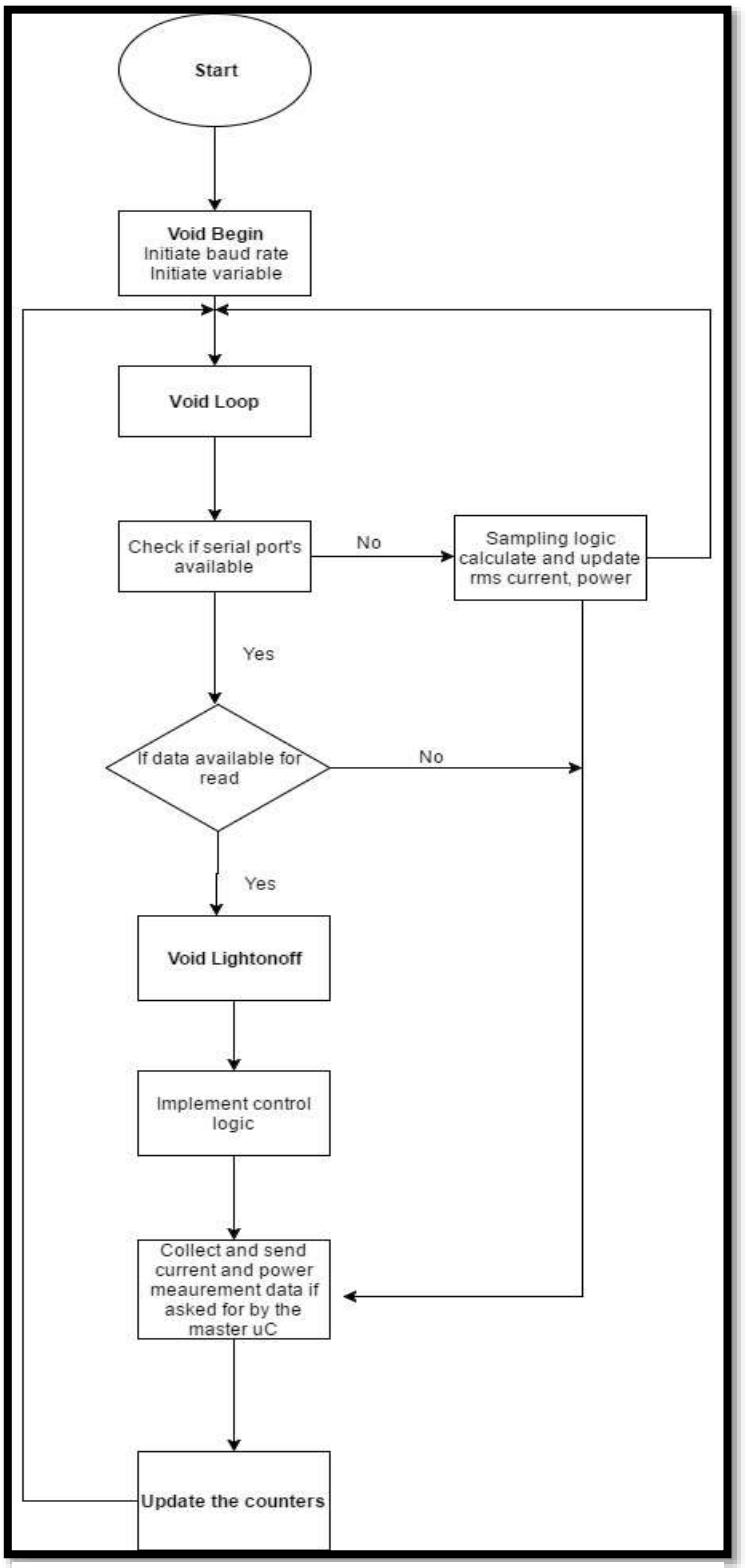


Figure 8: Flow chart of overall Arduino code

Second Stage

The code written for this stage has two separate functions

I. Void Loop

II. Void lightonoff

Void loop was made to accommodate the code written for sampling and determination of the current measured by the current sensors. The values of RMS currents measured were stored in variables based on the algorithm used for the code.

Void lightonoff is a function which deals with all the serial communication exchange requirements of the Arduino. It is called from Void loop based on the availability of the serial read buffer and updating of data in it from the Raspberry PI side.

This loop also takes care of the counters initialized for measurement purposes in the Void loop, since these variables are globally declared. The shortcoming this method is that there is loss of data in measurement for a cycle or more in between when the user wants to check the status or change the status of a switch, but on a long run depending on whether the code displays instantaneous RMS values or Average of RMS values calculated over every cycle, these minor inconveniences in the code become negligible. Many measures were taken to make sure the calculations accommodated all these peculiarities we faced due to the limitations in the rate of data transfer between the Master and Slave modules in the project. The following is the flow chart of the overall code, i.e the flow of data and control between the before mentioned loops. The Algorithms used have been explained after the flowchart.

Algorithm A

The first algorithm was made assuming the data transfer between the Arduino and Raspberry Pi could be carried out at baud rate of half a million or more, thus not effecting the rate of sampling of the data measured by the current sensor which would've meant that the chances of recreating the measured signal were positive.

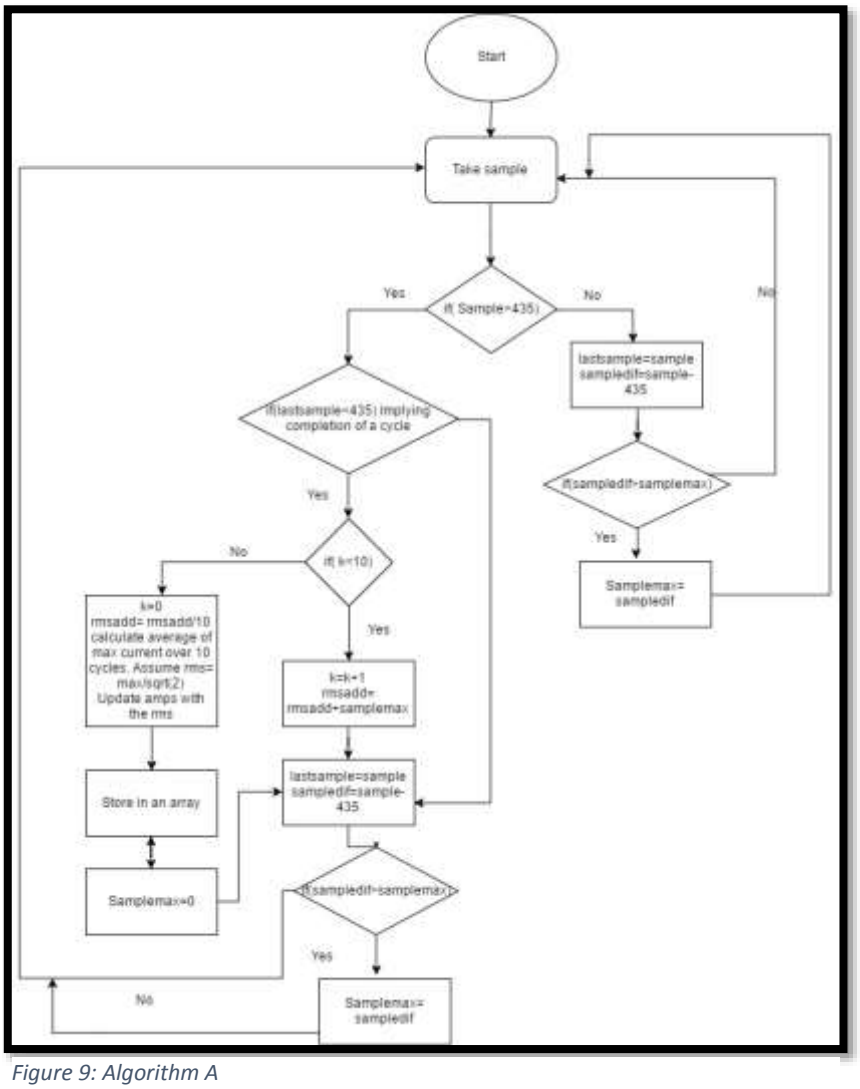


Figure 9: Algorithm A

The algorithm was designed to calculate the RMS value of the current's cycle in a trivial way and was later realized that it was not very reliable.

The idea was to calculate the maximum sample in a given cycle. The first step towards achieving this was to manipulate the algorithm to be able to differentiate between cycles. It was required to measure the value which the ADC produced for 0A current. I.e. find the cross over point of cycles. It was found out to be 435. Thus based on the values crossing this value the cycles were differentiated.

The maximum sample was taken as a variable which was continuously updated during a cycle by adding the difference between sample measured and 435 to 435, thus obtaining an unambiguous value of sample which represented the maximum deviation from zero even in the negative half of the cycle.

The value obtained was converted into amperes based on the algorithm for conversion mentioned earlier. This value divided by square root of 2 gave an approximate value of RMS current for the cycle. It was decided since the instantaneous value of RMS isn't very reliable since the load may vary, the connections might have irregularities, so the average of RMS values of 10 consecutive cycles was sent over to RPI instantaneously.

Algorithm B

The Second algorithm is more reliable given the fact it takes into account the limitations faced by the serial communication carried out between the modules. Thus instead of sending out instantaneous values over to RPI, an interrupt service routine was made (Void Lightonoff).

Thus RMS values of current were calculated over each cycle and were averaged. This averaging included data from the beginning of the time of operation of circuit. Thus we needed to develop various counters to reduce the storage and classify it in order to be able to calculate average with lesser data points.

Let us first look into the calculation of RMS, RMS current was calculated as the square root of the sum of squares of the value of samples converted into current divided by the number of samples.

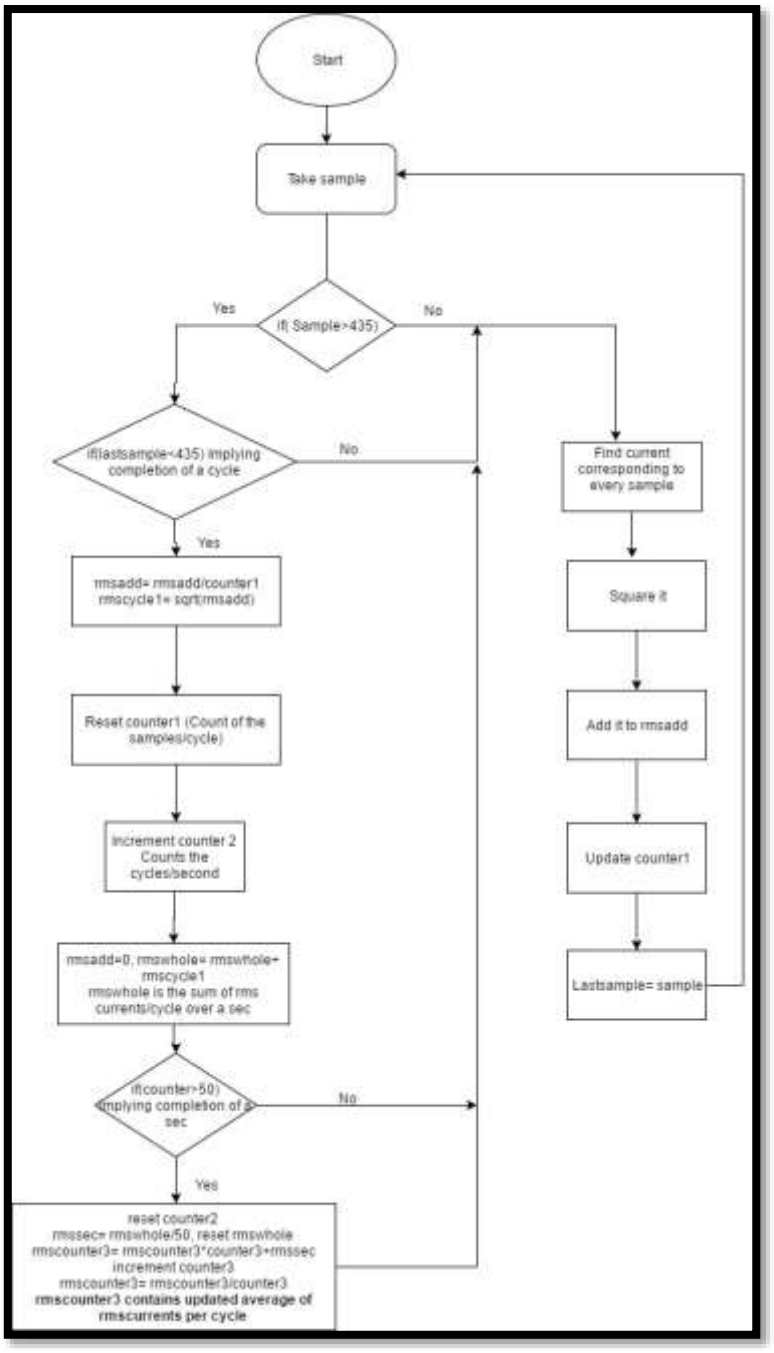


Figure 10: Algorithm B

Counter1: It is used to calculate the number of samples taken by the ADC from the current sample in a cycle of 50Hz(I.e. the frequency of current.

Counter2: It is used to keep a count on the number of cycles, it was reset to 0 as and when it reached the count 50, i.e implying completion of 50 cycles. 50 cycles imply the completion of 1sec of time since the frequency of current measured is 50Hz. Average of the rms current measured over the 50 cycles was taken and stored in the variable rmscycle.

Counter3: It was basically used to keep the count of secs the system has been running

Rmscounter = (Rmscounter x counter3(Older value) + rmscycle)/ counter3(new value);

The flowcharts of the mentioned Algorithms have been presented below

Chapter 3

CONCLUSIONS

3.1 IMPLEMENTATION

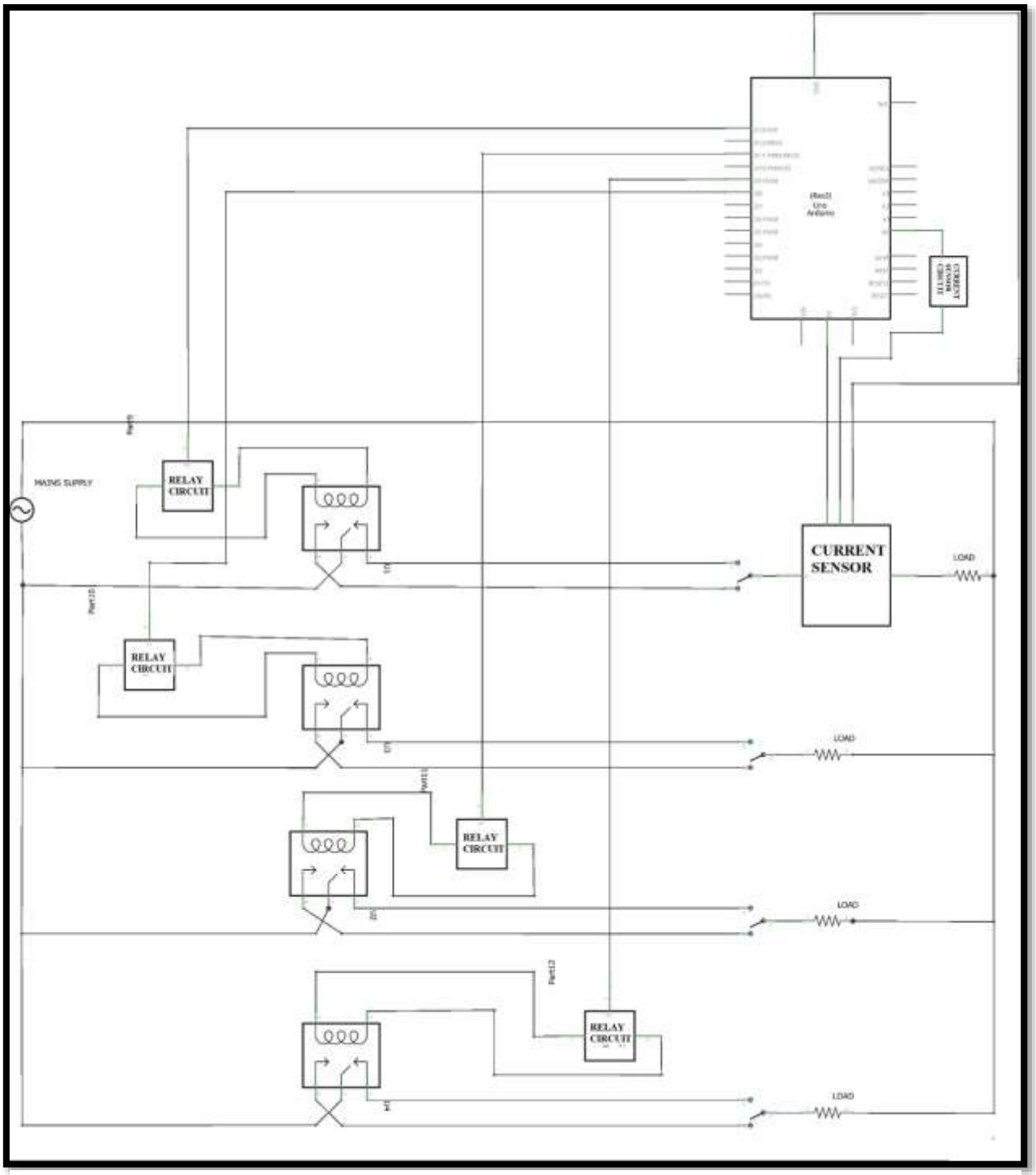


Figure 11: Switchboard 1 circuit diagram

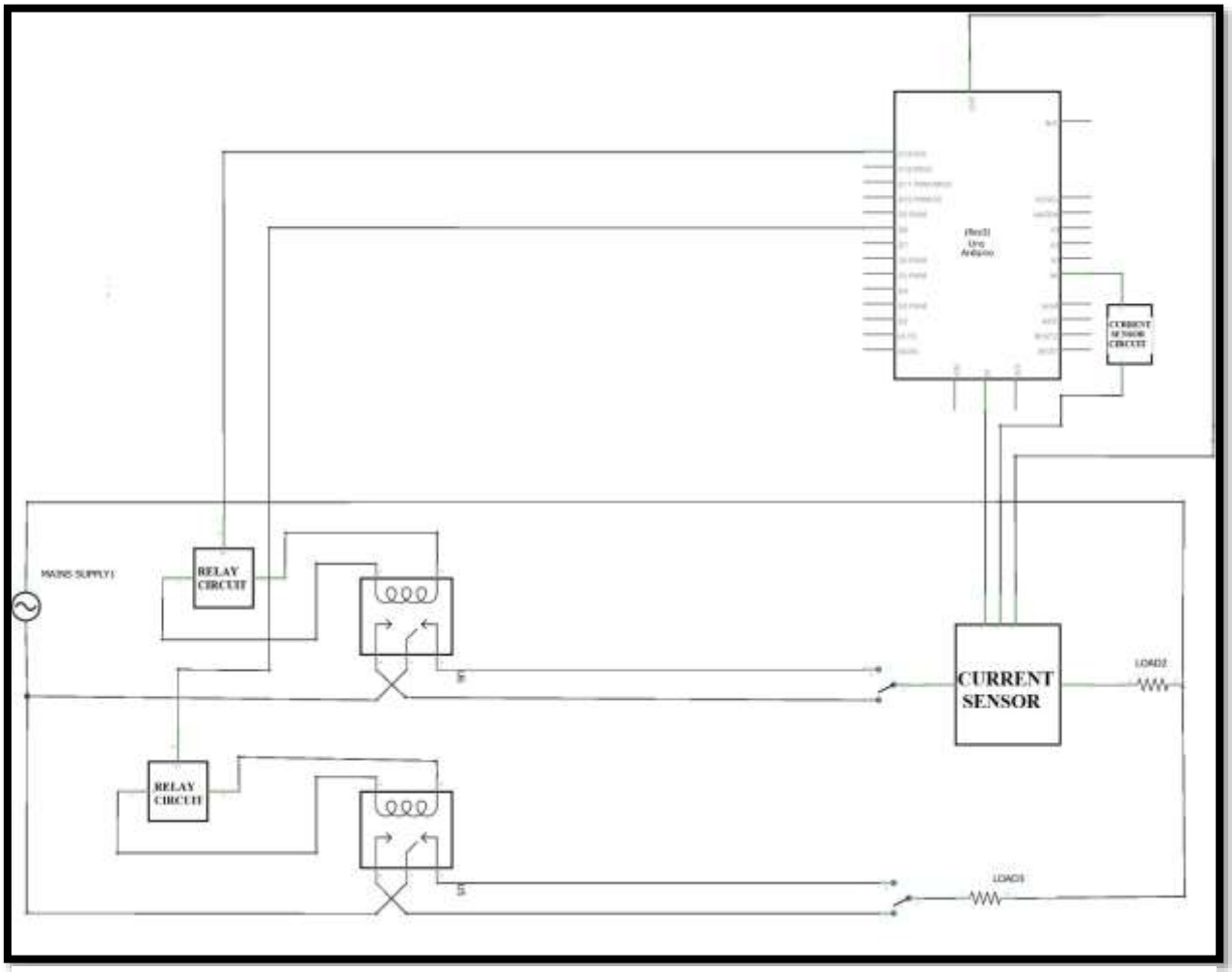


Figure 12: Switchboard 2 circuit diagram

The system comprises of two switchboards, one driving 4 loads. The other driving 2. Both the switchboards have one switch each with the capability of staircase switching as well as current measurement. The other switches have only staircase switching. The system was made with an objective to provide the user with multiple interface options and also an idea of the current and hence power consumption of the loads driven by the user. Since the project aims to highlight the use of IoT in day to day use, one of the interfaces is web based. After pondering over the possible options we came to a conclusion that the physical aspect of the switching already existing in houses cannot be completely done away with because not everyone in the household would be comfortable using Android applications or websites.

Thus a system which imbibes both the dimensions of usage i.e. Physical switch and online control is available on the website.

This is where the concept of staircase switching comes. Staircase switching as has been mentioned in the introduction before allows the user to control the load using both the switches.

Staircase wiring makes it feasible for the user to turn ON and OFF the load from two different positions, its circuit arrangement is in such a way, from common pole we can switch to both 1 & 2 poles, the 1st pole is connected to 1st pole of next and 2nd pole to the 2nd pole of the next, i.e. both poles to corresponding pole of the other. The phase of supply is connected to the common pole of one switch and phase supply to the load is taken from the common pole of the next switch. So in such an arrangement to close the circuit both the switches should be in the

same position in order to make the two common poles in contact to achieve a closed circuit. Changing ON & OFF condition of a single switch can determine whether the circuit is closed or open.

Thus, in staircase wiring we can control from both positions. For the above traveler system circuit the truth table of the connection will be similar to an XNOR gate, as the light ONs when both the switches are in the same position.

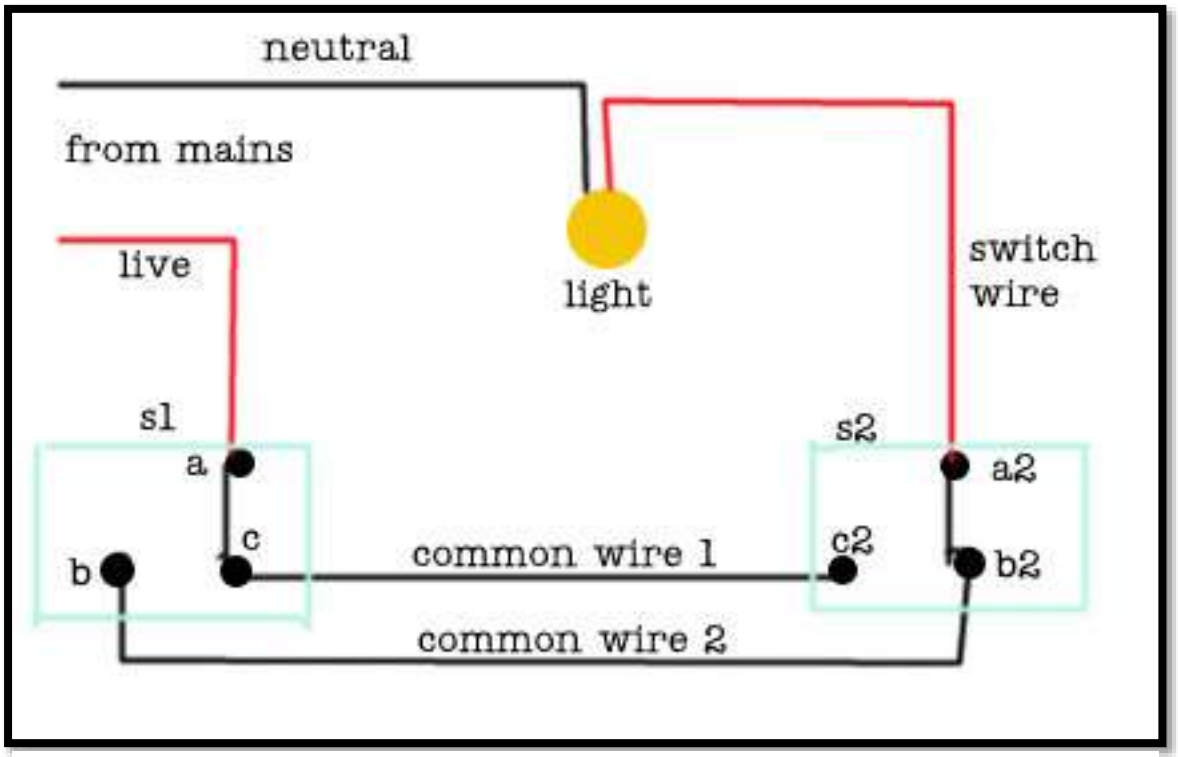


Figure 13: Staircase wiring in the switchboards

Coming to the elaboration on the exact working of the switches and the kind of switches used in our system.

In reference with the diagram above,

Switch S1 is a Relay. Relay is a single pole double throw switch. Relay is controlled by a trigger. This trigger is supplied by the GPIO Pins of the Arduino. The input to this GPIO pin is given through the python code written to take inputs from the online platform, these inputs are serially communicated to the Arduino. Therefore based on the input provided by the user the pole switches between two throws C and B.

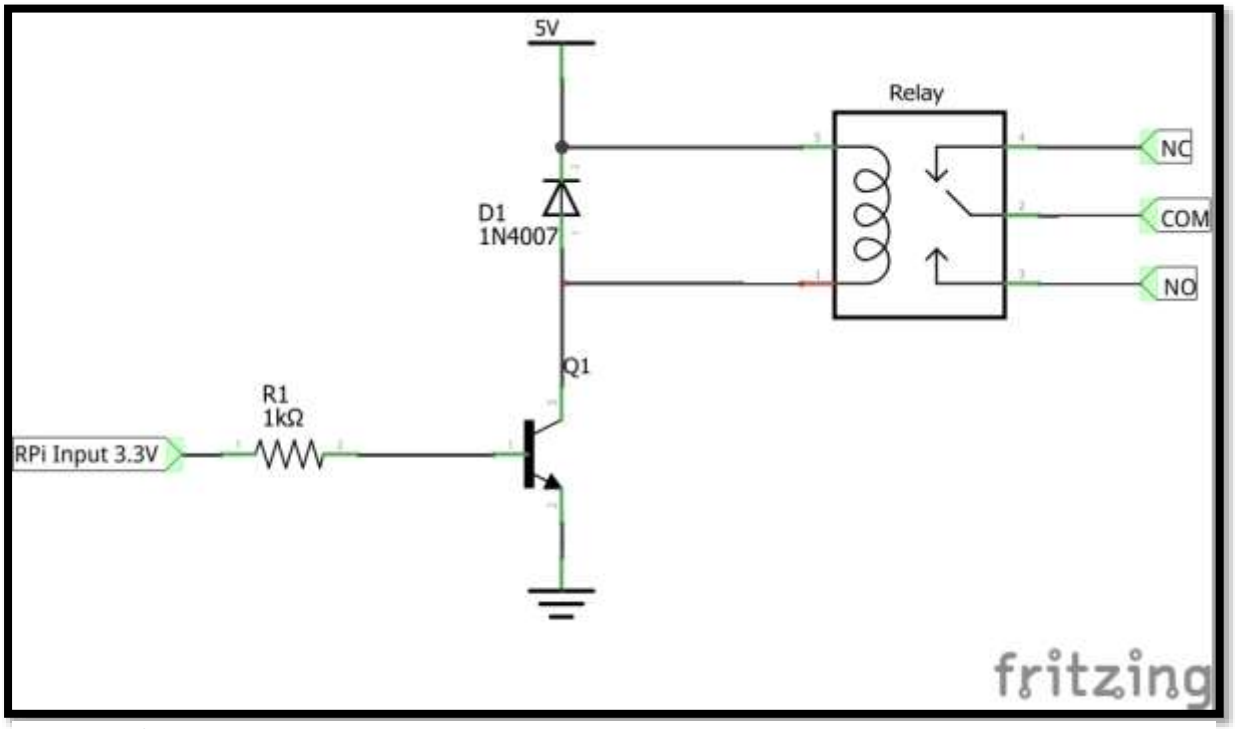


Figure 14: Relay testing circuit

This relay circuit was designed and tested with inputs from basic codes to test LEDs supplied from a 5V source before moving to the mains supply.

The circuit shown above was successfully soldered on a GPB (General purpose board) and tested first with LED as the load with a 5V source and corresponding resistances and then an upgrade was made in order to test its applicability to the mains supply which was our primary goal. Thus the 5V source in the circuit was replaced by 220V source.

Elaborating a bit on the transfer of the flow of the command placed online and the working of the relay. The commands are obtained from an online HTML page and input/output control is done using CherryPy. The page has submit button which when clicked, causes an action to be performed. This action is set in the CherryPy code itself. In our case when the button is pressed, the GPIO pin is made high or low depending on the previous state of the switch.

Coming to the second switch, the second switch used is a basic single pole double through switch with an additional state which allows it to stay disconnected from either throw. The throws of this switches are named as C2and B2 as shown in the figure above (Figure 4.1). This was again tested with a GPB soldered circuit to understand the working better. This switch is used to work as the traditional switch.

Thus these two switches were used to implement the staircase switching concept.

Online Switch State	Mechanical Switch state	Load Status
C	B2	off
C	C2	on
B	B2	on
B	C2	off

Figure 15: Truth table of staircase wiring for project

The basic need which this staircase switching system fulfils is the ability to control the load remotely. Suppose the load is currently on when the user is not at home, all he needs to do is to toggle the switch online. What toggling the switch online does is, it changes the status of the

relay from C to B or B to C, thus making the circuit open. Similarly, assuming the user doesn't want to open the web interface to control the load, so the person can just change the status of the switch to connect to the other throw, thus the status of the circuit is opened as it was closed before.

As discussed before since the platform allows the user to control the load even when the load is not in his sight i.e. the user doesn't need to visually determine the status of the load. In these conditions, the toggle functionality of the switches can be ambiguous. Therefore the need for a system that determines the status of the load and updates it online was felt.

Detection and measurement Circuit

For the 7th Sem project the detection circuit was based on LDR detection of light. A LED was placed parallel to the mains and thus the ability of the LDR detection circuit was used to determine the state of the switch. i.e if the LED glows, the Load is connected . If it doesn't, the circuit is open.

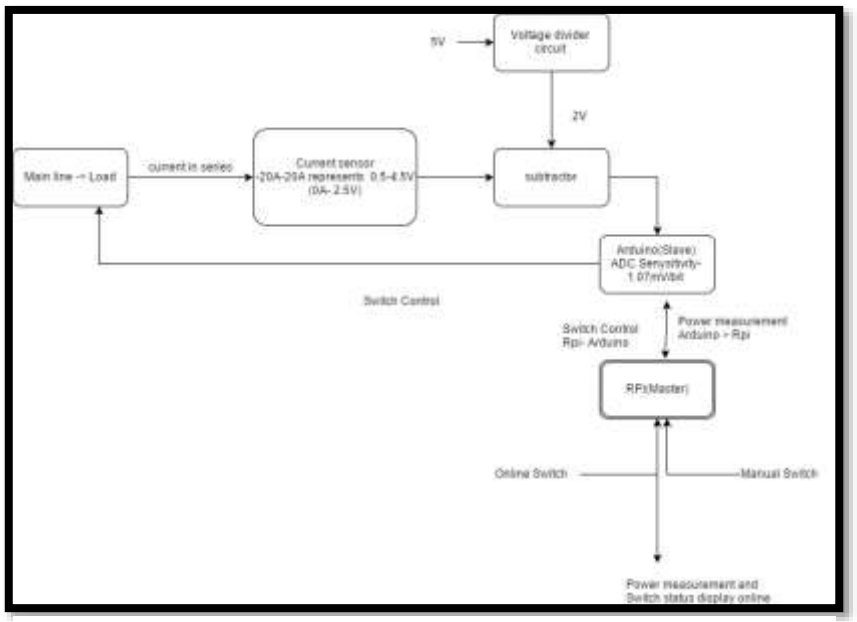


Figure 16: Detection and measurement circuit

Since this doesn't take care of the power measurement needs of the project. After pondering over various options we came to a conclusion of going ahead with current sensor. ACS 712 20A hall effect current sensor to be specific.

The specifications of the current sensor and related graphs have been explained under introduction. Going back to the important features, the mentioned current sensor gives a range of voltage 0.5-4.5V for current values -20A-20A. The zero crossing giving a voltage of 2.5V, so in order to be able to work with this range of voltage for the range provided by Arduino a few changes were made.

The reference voltage of the inbuilt ADC of the Arduino was changed to 1.1V, thus the 10bit ADC was providing us with the precision of 1.07mv/bit. Thus the overall voltage provided by the ADC had to be altered. So a subtractor circuit was put in place between Arduino and Raspberry Pi. 2V was subtracted from the voltage given by the current sensor to fit the range within the range of ADC. Following is the flow chart of the flow of measured values and the conversion of data from measured ADC value to current.

If the current value measured is less than 0.1A the system takes that given load is OFF, else it considers it as ON. The corresponding values of current measured is displayed on the web page and the associated power consumed is computed by multiplying by 220V.

The following figure is of the subtractor circuit shown in the above block diagram.The values shown in the picture are not exactly what we used in the actual circuit.

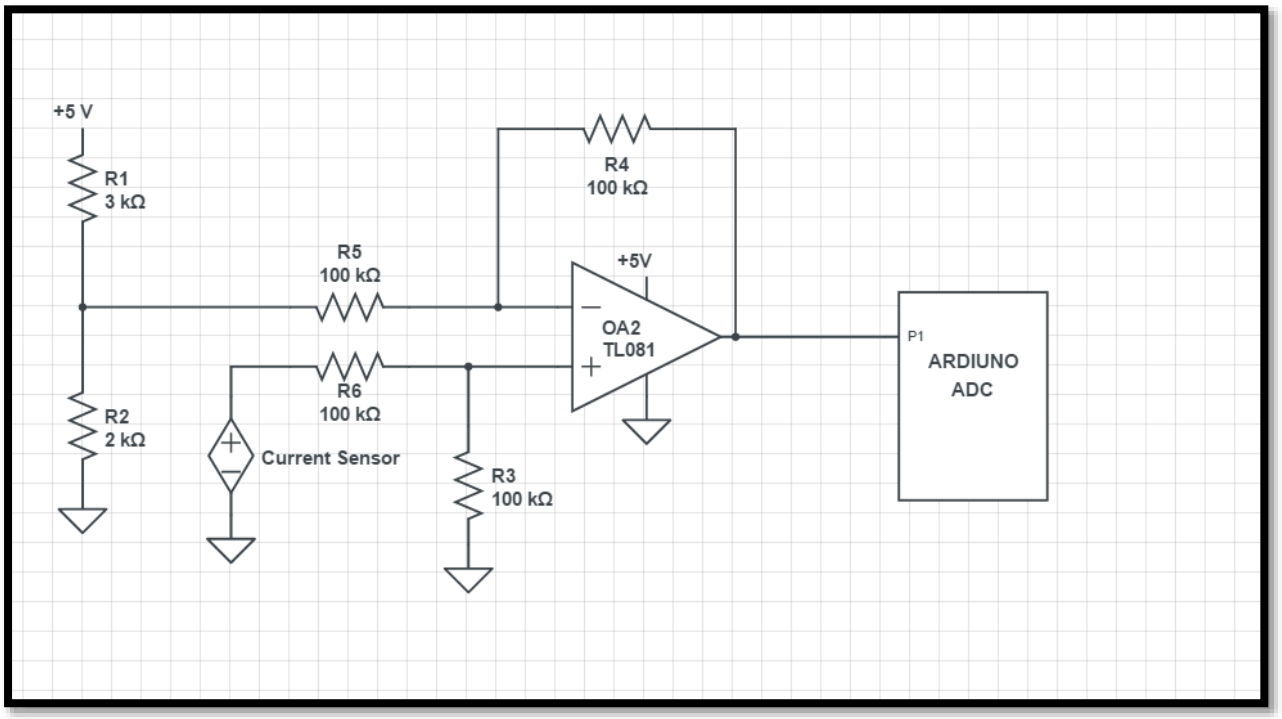


Figure 17: Subtractor circuit

CHASSIS

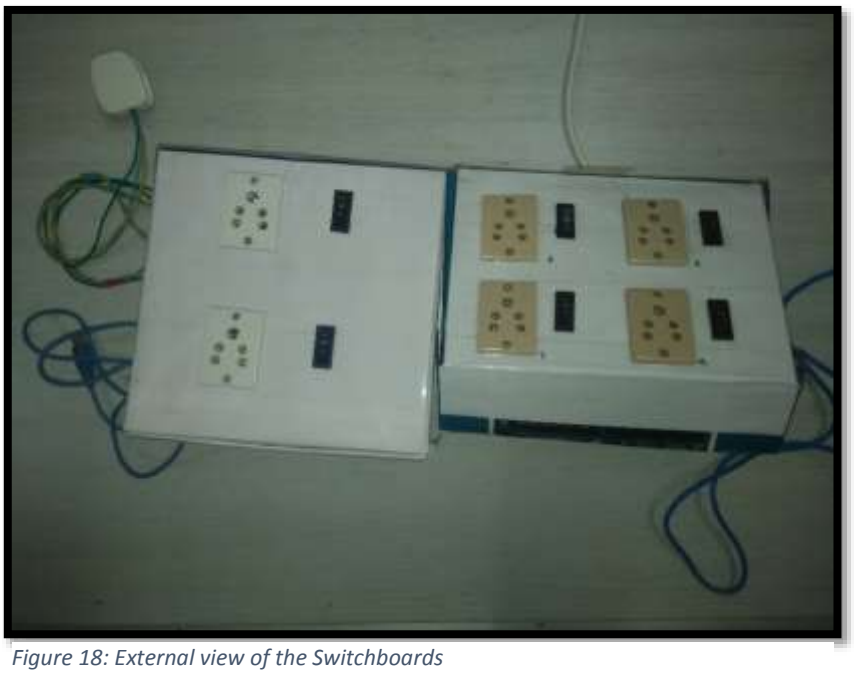


Figure 18: External view of the Switchboards

The systems are built on 17cm *24cm*6cm cardboard boxes. Box 1 has 4 Ceramic plug points placed on it, The 4 plug points are connected to SPDT switches. Box 2 again has a similar arrangement but with just 2 plug points. The plug points marked "1" on either of the boxes have been connected to current sensors, and all data displayed on the web page will be related to these. Arduinos that control the relays are placed in the respective boxes keeping in mind that they do not interfere with the mains.

The relay circuit set has been made on a single board and placed at the base of each box. As we are using a cardboard box body with 220 V AC inputs flowing in it, special care had to be taken in order to insulate all connections and manage wirings as neatly as possible.

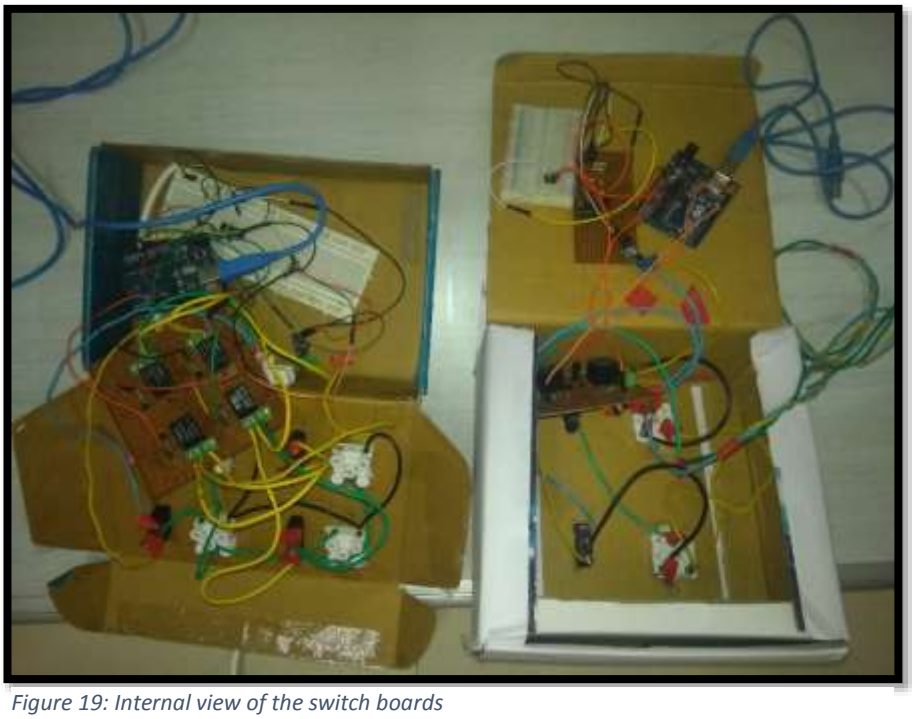


Figure 19: Internal view of the switch boards

The raspberry pi will be placed outside the box in this case as it would allow easy connection to the computer system, USB devices and Ethernet ports.

3.2 FINAL CODES

Raspberry Pi code: -

```
#ACM0 corresponds to the switchboard with 4 plugpoints
#ACM1 corresponds to the switchboard with 2 plugpoints
import cherrypy
import serial # Import Serial Library
import numpy  # Import numpy

global s0
global s1
global cs0
global cs1
s0=0
s1=0
cs0=0
cs1=0
CurrentSensor0= []
CurrentSensor1= []
ser0 = serial.Serial('/dev/ttyACM0', 9600, timeout=None,parity='N',stopbits=1,rtscts=0) #Creating our serial object name
ser1 = serial.Serial('/dev/ttyACM1', 9600, timeout=None,parity='N',stopbits=1,rtscts=0) #Creating our serial object name

ser0.flushInput()
ser0.flushOutput()
ser1.flushInput()
ser1.flushOutput()

class Binary(object):

    @cherrypy.expose
    def index(self):
        return self.loadpage()

    @cherrypy.expose
    def toggle0(self):
        global s0
        global cs0
        if cs0==0:
            ser0.write('a')
            cs0=1
        else:
            ser0.write('b')
            cs0=0
        return self.loadpage()
    @cherrypy.expose
    def status0(self):
        global s0
        ser0.write('c')
        while (ser0.inWaiting()==0): #Wait here until there is data
            pass #do nothing
        s0=ser0.readline()
        s0=float(s0[0:4])
        print s0
        CurrentSensor0.append(s0)
        return self.loadpage()

    @cherrypy.expose
    def on0_2(self):
        ser0.write('d')
        return self.loadpage()

    @cherrypy.expose
    def off0_2(self):
        ser0.write('e')
        return self.loadpage()

    @cherrypy.expose
    def on0_3(self):
        ser0.write('f')
        return self.loadpage()

    @cherrypy.expose
    def off0_3(self):
        ser0.write('g')
        return self.loadpage()

    @cherrypy.expose
    def on0_4(self):
        ser0.write('h')
        return self.loadpage()

    @cherrypy.expose
    def off0_4(self):
        ser0.write('i')
        return self.loadpage()
    @cherrypy.expose
    def toggle1(self):
        global s1
        global cs1
        if cs1==0:
            ser1.write('a')
            cs1=1
        else:
            ser1.write('b')
            cs1=0
        return self.loadpage()

    @cherrypy.expose
    def status1(self):
        global s1
        ser1.write('c')
        while (ser1.inWaiting()==0): #Wait here until there is data
            pass #do nothing
        s1=ser1.readline()
        s1=float(s1[0:4])
        print s1
        CurrentSensor1.append(s1)
        return self.loadpage()

    @cherrypy.expose
    def on1_2(self):
        ser1.write('d')
        return self.loadpage()

    @cherrypy.expose
    def off1_2(self):
        ser1.write('e')
        return self.loadpage()

    @cherrypy.expose
    def loadpage(self):
        global s0
        global s1
        text= """ <!DOCTYPE html>
                  <html lang="en-US">
                  <head>
```

```
            .button {
                background-color: #4CAF50;
                border: none;
                color: white;
                padding: 20px 37px;
                text-align: center;
                text-decoration: none;
                display: inline-block;
                font-size: 16px;
                margin: 4px 2px;
                cursor: pointer;
            }

            header, footer {
                padding: 1em;
                color: white;
                background-color: black;
                clear: left;
                text-align: center;
            }
            .city {
               float: left;
               border-radius: 25px;
               border: 2px solid #73AD21;

               margin: 10px;
               padding: 10px;
               width: 600px;
               height: 350px;
               border: 1px solid white;
            }
            </style>
            </head>
            <body background="IoT-background.jpg">

            <header>
               <h1>Multiple Interface Home Automation</h1>
            </header>
            <div class="city">
              <h2>Room-1</h2>
                 <br>
                 <form method="get" action="toggle0"><button class='button' type="submit">TOGGLE</button></form>
                 <br>
                 <form method="get" action="status0"><button class='button' type="submit">STATUS</button></form>
                 <br>
                 <br>'''

    print s0
    if s0<0.1:
        text=text+'''<p>LOAD 1 is OFF</p>'''+'''<p>Current:-</p>'''+str(s0)+'''<p>Power:-</p>'''+str(s0*220)
    else:
        text=text+'''<p>LOAD 1 is ON</p>'''+'''<p>Current:-</p>'''+str(s0)+'''<p>Power:-</p>'''+str(s0*220)

    text=text+'''   <form method="get" action="on0_1"><button class='button' type="submit">ON_1</button></form>
                    <br>
                  <form method="get" action="off0_1"><button class='button' type="submit">OFF_1</button></form>
                  <br><br>
                  <form method="get" action="on0_2"><button class='button' type="submit">ON_2</button></form><br>
                  <form method="get" action="off0_2"><button class='button' type="submit">OFF_2</button></form>
                   <br><br>
                  <form method="get" action="on0_4"><button class='button' type="submit">ON_4</button></form><br>
                  <form method="get" action="off0_4"><button class='button' type="submit">OFF_4</button></form>
                <br>
                <br>
                <br>
                <br>
              </div>

              <div class="city">
                <h2>Room-2</h2>
                <br>
                    <form method="get" action="toggle1"><button class='button' type="submit">TOGGLE</button></form><br>
                    <form method="get" action="status1"><button class='button' type="submit">STATUS</button></form><br>

    print s1
    if s1<0.1:
        text=text+'''<p>LOAD 1 is OFF</p>'''+'''<p>Current:-</p>'''+str(s1)+'''<p>Power:-</p>'''+str(s1*220)
    else:
        text=text+'''<p>LOAD 1 is ON</p>'''+'''<p>Current:-</p>'''+str(s1)+'''<p>Power:-</p>'''+str(s1*220)

    text=text+''' <form method="get" action="on1_1"><button class='button' type="submit">ON_1</button></form><br>
                  <form method="get" action="off1_1"><button class='button' type="submit">OFF_1</button></form><br>
                  <br>
              </div>


              <footer></footer>

              </body>
              </html>
'''

    return text

if __name__ == '__main__':
    cherrypy.quickstart(Blinky())
```

Figure 20: Cherry Py code

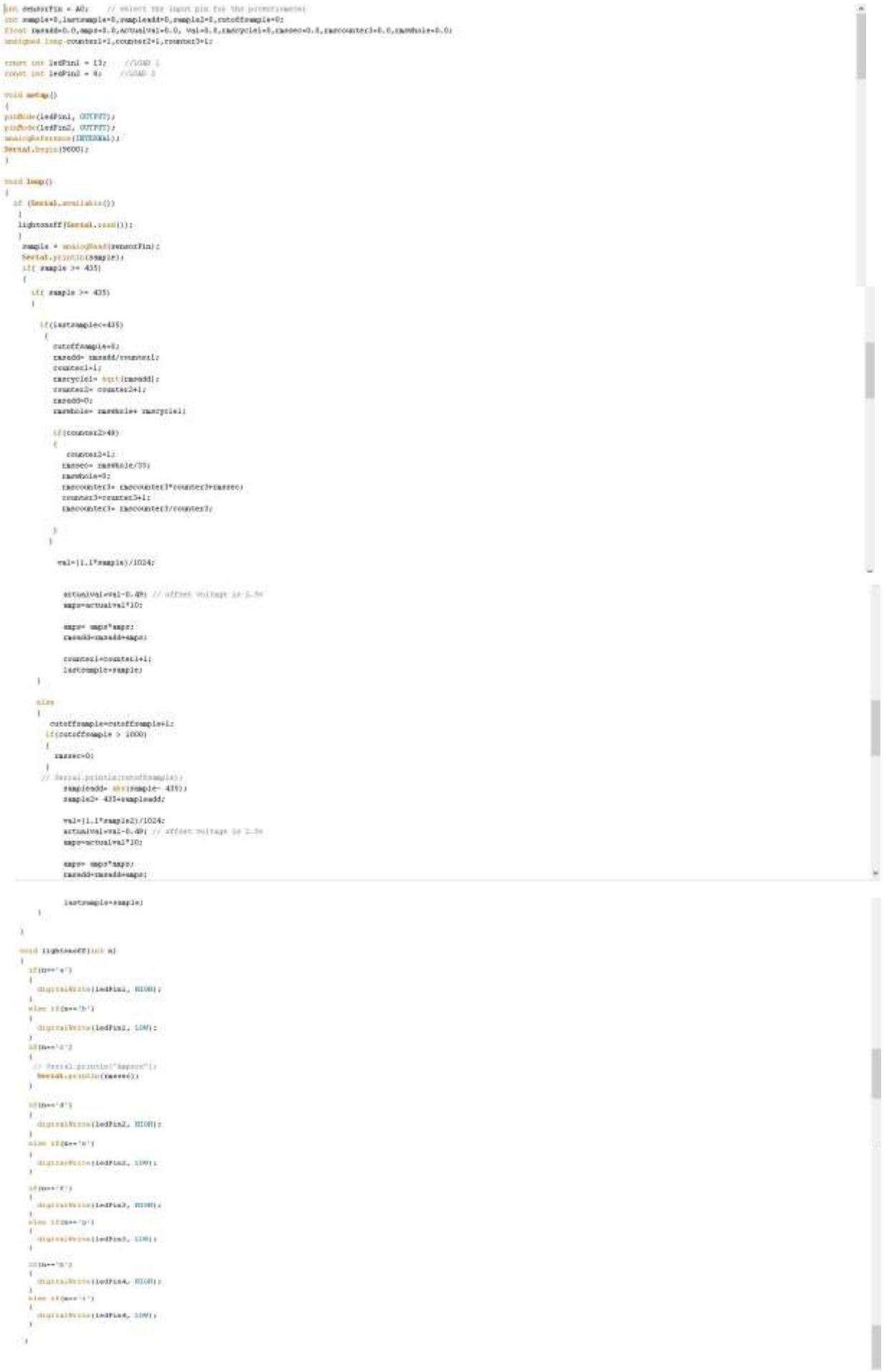


Figure 21: Arduino code using Algorithm B for switchboard 1

3.2.1 CODE EXPLANATION

Cherry Py

The characters sent over to Arduino for checking status and controlling the appliances.

Pair of characters	Load	Work
a,b	GPIO13	Toggle
c	GPIO13	RMScurrent value
d,e	GPIO8	Toggle
f,g	GPIO9	Toggle
h,i	GPIO11	Toggle

- Toggle Button

Toggle button toggles the state of the relay based on the previous data sent over to the Arduino. This was a necessary change inducted in the code to make last semester's site more user friendly

Previous state	After toggle
a	b
b	a
d	e
e	d
f	g
g	f
h	i
i	h

- Status Button

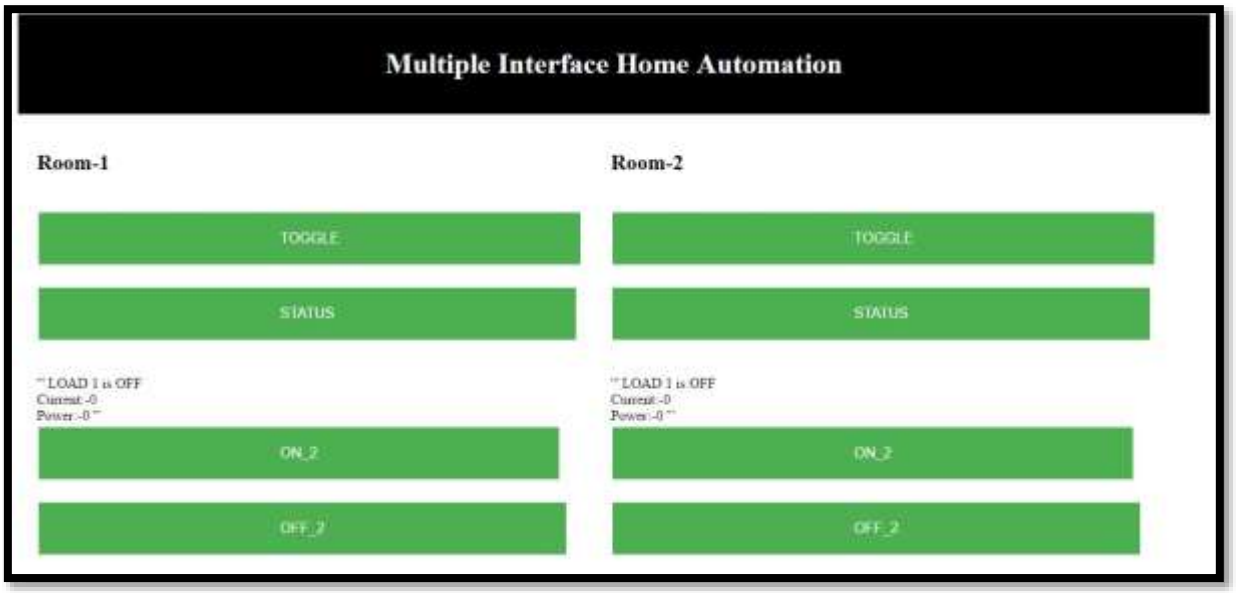
Instruction 'c' was used to ask the Arduino for status updates of GPIO13 of Arduino. The status button hence displayed "on" or "off" status of the load based on the rmscurrent value sent over by Arduino determined the status and also displayed the current and power values on the site.

ArduinoUno

The algorithm followed for Arduino was Algorithm B [2.3]. The code presented is for the switchboard which accommodates four switches. The calculation part and the serial communication part has been kept separate in the loop void lightonoff. The following table describes the specific letters and their jobs.

Instruction	GPIO	Status of relay
a	13	NC
b	13	NO
c	The calculated RMS current is sent over serially to R.Pi	
d	8	NC
e	8	NO
f	9	NC
g	9	NO
h	11	NC
i	11	NO

3.3 WEB PAGE LAYOUT



The web page made is shown above. It consists of a total of 12 buttons controlling 6 plug points. Room1 consists of 4 switches and Room 2 has 2 switches. Along with this, the ON/OFF status and value of current and power is displayed below the switch. This has been done for a total of 2 switches to which the current sensors have been attached. For the remaining switches which only has online control, we used 2 switches as ON and OFF only.

3.4 Estimated Cost

Name	Quantity	Rate	Availability	Amount
Raspberry Pi 3 kit+SD	1	2649+450	available	3099
relay	6	30	available	180
Arduino Uno	2	525	available	1050
LM324	2	10	available	20
ACS712(20A)	2	250	available	500
PCB	2	30	available	60
Breadboard	1	80	available	80
wires	5m	15	available	75
Berg strip	1 set	10	available	10
connector	6	5	available	30
Resistors	20	1	available	20
2N2222	6	10	available	60
Jumper wires	20	5	available	100
switch	6	20	available	120
diodes	6	1	available	6
sockets	6	10	available	60

Total Cost of Hardware components: - Rs 2371

Total Cost of Project including R Pi:- Rs 5470

3.5 FUTURE SCOPE

1) Keeping the project in mind, the Arduino if continued as the slave module can be used for the following features, the respective data of each of the application would them be sent to Master(R pi) through serial communication or RF modules as was in previous cases.

   a) Smart Night Lamp for Kids: Lights up when dark and changes color automatically, would make use of light sensors

   b) Motion using PIR/Ultrasonic sensors: Detecting the presence and number of people in a room.

   c) Motion Following Camera Base: Upgrade your home security camera, webcam, or any other type with a motorized stand that will detect, track, and follow any motion in the room

   d) A simple PID fan controller with LCD display

2) Images and video will be made available via a remotely controlled webcam. These peripherals can be made totally wireless, communicating with the central Raspberry Pi hub via low cost and low power wireless communication. These peripherals will be controlled via a web-based interface served on the Raspberry Pi and accessible from inside or outside the home.
   a) RF modules can be used for this purpose, we tried using an NRF transceiver this semester for the project but due to lack of beforehand experience and lack of information on online forums we couldn't debug the technical difficulties faced by it while interfacing it with Arduino. An effort can be made by any group working on this project in future to look into the technical aspects of NRF communication in detail and making the wireless aspect of it work.
   b) Arduino Ethernet shield can be used for making Arduino access internet, in cases like that the control can be made completely online and the physical aspect of the communication is done away with.

3) In order to predict more accurate values of power consumed and current, voltage sensors can be used to record real time voltage values. A precise power factor calculation method must also be used.

3.6 REFERENCES

[1] https://docs.python.org/

[2] http://developer.android.com/training/index.html

[3] http://elinux.org/RPi_Hub

[4] http://www.raspberrypi.org

[5] http://cherrypy.org

[6] http://electronics.howstuffworks.com

[7] N. Sriskanthan and Tan Karand. “Bluetooth Based Home Automation System”. Journal of Microprocessors and Microsystems, Vol. 26, pp.281 289, 2002.

[8] Muhammad Izhar Ramli, Mohd Helmy Abd Wahab, Nabihah, “TOWARDS SMART HOME: CONTROL ELECTRICAL DEVICES ONLINE”,Nornabihah Ahmad International Conference on Science and Technology: Application in Industry and Education, 2006

[9] E. Yavuz, B. Hasan, I. Serkan and K. Duygu. “Safe and Secure PIC Based Remote Control Application for Intelligent Home”. International Journal of Computer Science and Network Security, Vol. 7, No. 5, May 2007