Development of a laboratory model to demonstrate load management in a smart grid

A Thesis submitted to the

Dept. of Electrical & Electronic Engineering, BRAC University in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering

by

SYEDA MAYESHA AZIM

SALWA SHAHIDI

REHENUMA TARANNUM

SHARABAN TAHORA

Dept. of Electrical and electronic Engineering, BRAC University

April 2014

Declaration

We do hereby declare that the thesis titled "**Development of a laboratory model to demonstrate load management in a smart grid**" is submitted to the Department of Electrical and Electronics Engineering of BRAC University in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronics Engineering. This is our original work and has not submitted elsewhere for the award of any other degree or diploma.

SYEDA MAYESHA AZIM

Student ID: 10115002

SALWA SHAHIDI

Student ID: 10221068

REHENUMA TARANNUM

Student ID: 10121014

SHARABAN TAHORA

Student ID: 10121093

Countersigned:

Prof. Dr. S. Shahnawaz Ahmed

Department of Electrical and Electronic Engineering

Bangladesh University of Engineering and Technology

(Thesis Supervisor)

Acknowledgements

We are extremely grateful to our Supervisor, Prof. Dr. S. Shahnawaz Ahmed for his ideas, advice, guidance and assistance. Without his continuous support the thesis work would not have been possible.

We are also thankful to Risul Karim and Jonayet Hossain for helping us with the laboratory work.

Abstract

Vital to the evolution of a modern civilization is uninterrupted supply of electricity. However, in a resource constrained utility the large gap between demand and generation entails load shed i.e. tripping a feeder from the substation causing a total service disruption to the consumers under that feeder for an interval. An alternative is load management through demand response i.e. consumers will willingly reduce their consumptions on getting a signal on price or other incentives from the utility so that there is no need for shedding the total load of a consumer. This is one of the objectives of a Smart grid that aims to overlay ICT in the existing power grid and co-ordinate energy usage of individual consumers so that the generation and demand is balanced.

In order to familiarize the smart grid concept to the consumers in general and future engineers in particular this thesis addresses the load management aspect through developing a laboratory model. Here the consumer end smart meter is represented by an Arduino Mega2560 microcontroller equipped with a NRF905 transceiver and interfaced with LEDs (which simulate loads). Another similar set of Arduino is interfaced with a computer to simulate the utility server. From the server a signal on the percentage of load to be turned off is transmitted to the client end. Within 10 seconds of receiving the signal a suitable combination of loads greater than or equal to the instructed percentage needs to be turned off by the consumer else the equivalent load will be turned off automatically by the smart meter. In either case the client (smart meter) transmits a feedback signal to the server. The way the model has been built and various codes are developed for control and communication is illustrated in this thesis.

Table of Contents

Chapter 1 Introduction 1

1.1 Background 1

1.2 Literature Review 1

1.3 Objectives 2

Chapter 2 Methodology 3

2.1 Control center 4

2.2 Client end 7

2.3 Load Control 9

Chapter 3 Implementation 11

3.1 NRF905 Communication 11

3.2 Control Center Code 143

3.3 LCD connection and code 16

3.4 Client end connection and code 18

3.4.1 Detect Load Use 22

3.4.2 Keep Load Under Check 23

3.4.3 Central Client Code 25

Chapter 4 Conclusion 29

4.1 Conclusion 29

4.2 Suggestions for further work: 29

References 31

Appendixes 32

A.1 Server code 32

A.2 Client Code 35

Chapter 1

Introduction

1.1 Background

Smart grid is a vision for transforming the conventional electric power system into a more reliable, energy efficient, customer friendly and fault tolerant self-healing system through integrating distributed energy generation and storage devices at the demand side (consumer end) besides large centralized power plants and deploying ICT (information and communication technology) equipment massively so that demand response i.e. feedback from consumers can be acquired and utilized in real time for balancing the generation and demand in the power system. Demand of electricity usage changes in real time due to diversity in human behavior and activities. If a power grid fails to cope with the demand in any interval of time, system stability needs to be compromised else load shedding needs to be enforced. Smart grid introduces a distributed and client-centric load management system based on a two way communication between the client and the control center.

1.2 Literature Review

A review of literature [1-8] reveals that since the introduction of the smart grid concept in USA by EPRI (Electric Power Research Institute) in the year 2000 many other institutions in USA, Europe, Japan, China and India proliferated the concept and put forward various visionary

models over the last decade. These concepts include mainly renewable energy sources and plug-in hybrid electric vehicle storage at the consumer end coupled with a two way communication system between the utility server and each consumer which would utilize all the available media including the internet as well as intranets. However, most of the works are still in research stage awaiting full scale commercial implementation and acceptance by all the stakeholders due to various reasons e.g. question of affordability or wide-scale availability of broadband communication infrastructure and smart consumer appliances. Moreover, still much effort needs to be put to familiarize the smart grid concept to the general mass.

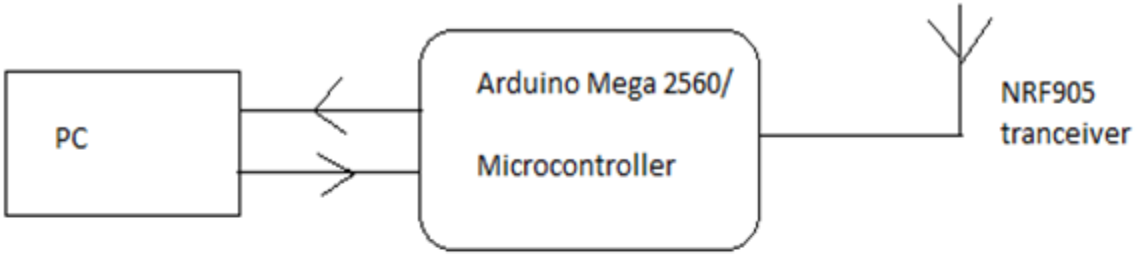
1.3 Objectives

This thesis has addressed one of the functions of smart grid to familiarize this to future engineers in particular and people in general. It aims at developing a laboratory scale model on a load management system where consumer will willingly curtail their electricity consumption following a signal from the utility server whenever necessary from frequency control, price or other points of view. If the consumer does not turn off load within a given time (e.g.10 seconds at the simulation stage), the client end controller will automatically turn off equivalent or greater amount of load. The shed load magnitude is communicated back to the server. For implementation of the model two sets of Arduino Mega 2560 have been used, one as utility server and another as client end meter where some LEDs are connected to simulate loads. The communication between server and client has been developed by one pair of NRF905 transceivers.

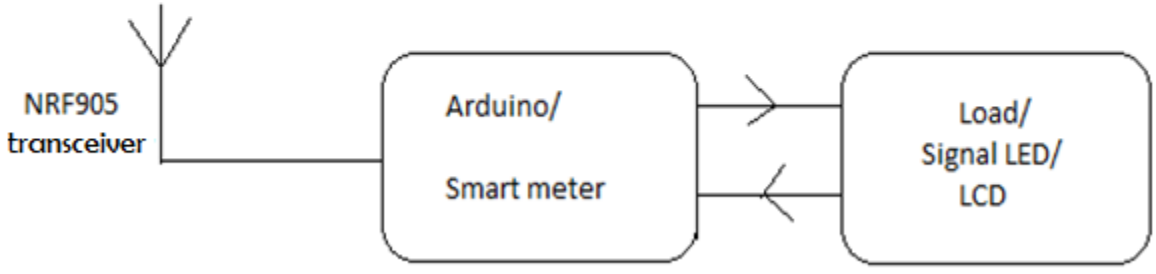
Chapter 2

Methodology

For the operation of smart grid load management system at least the information on percentage of load to be reduced needs to be exchanged between the server and the clients. In the designed model a client end and its loads are represented by an Arduino Mega2560 equipped with NRF905 [9] and few LEDs. A similar set of Arduino is connected to a PC with standard USB cable (A-plug to B-plug) to simulate the control center. The whole system is illustrated in Fig.2.1.



Block Diagram at Server

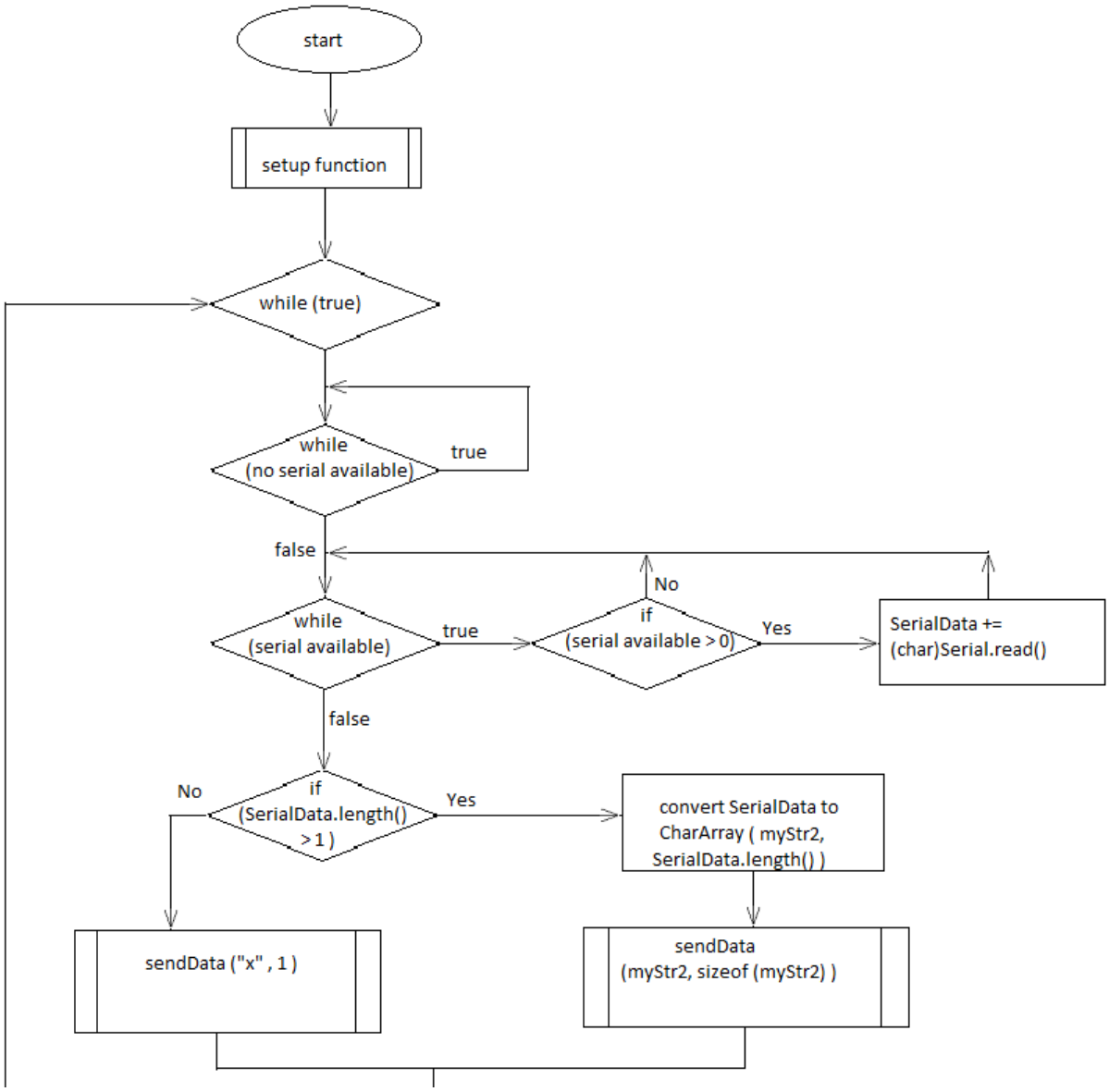


Block Diagram at Client **End**

Fig. 2.1 The developed laboratory model

2.1 Control center

The information to be sent from the control center is entered into the serial monitor tool of Arduino software via the keyboard. Ten seconds after sending the signal the control center receives feedback from a client on how much load had been shed. The magnitude of reduced load is displayed in the serial monitor after filtering the unwanted signals. A simplified flowchart for the process at control center is shown in Fig. 2.2.


start
setup function
while (true)
while
(no serial available)
true
false
while
(serial available)
true
if
(serial available > 0)
Yes
No
SerialData +=
(char)Serial.read()
false
No
if
(SerialData.length()
> 1)
Yes
convert SerialData to
CharArray (myStr2,
SerialData.length())
sendData ("x" , 1)
sendData
(myStr2, sizeof (myStr2))

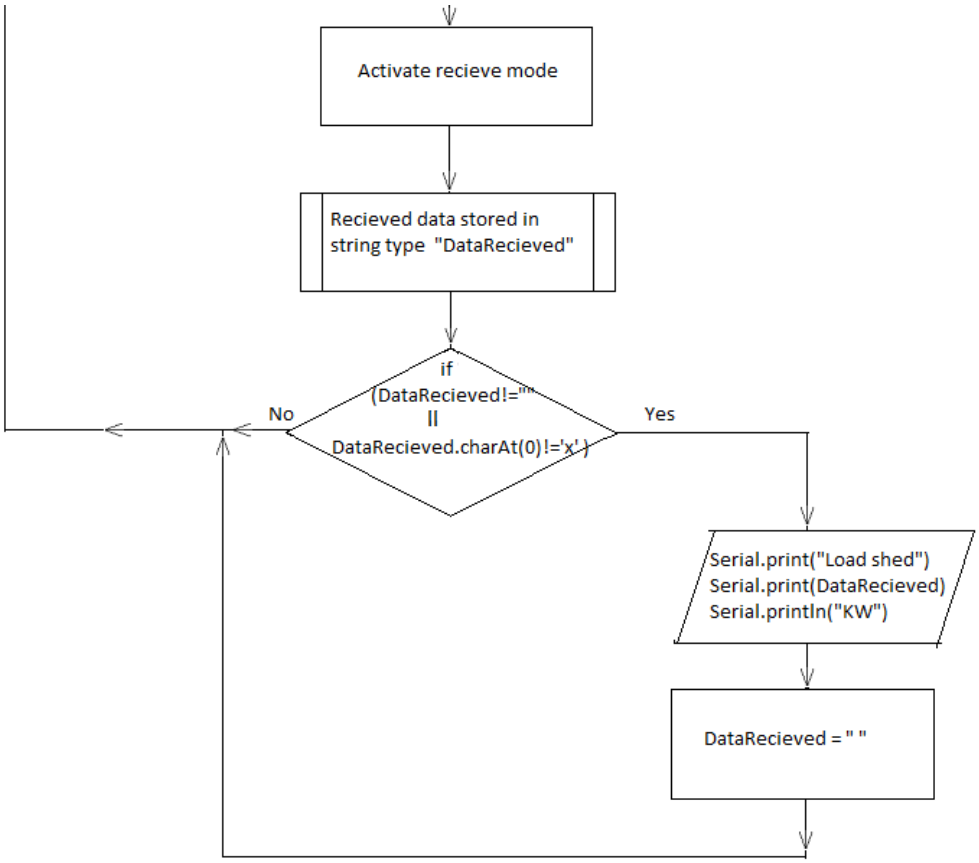
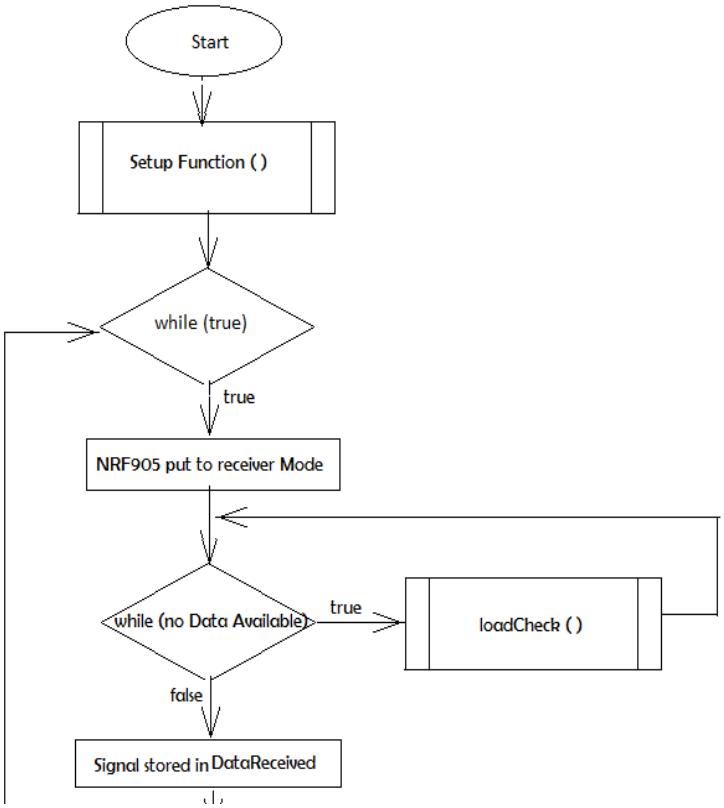
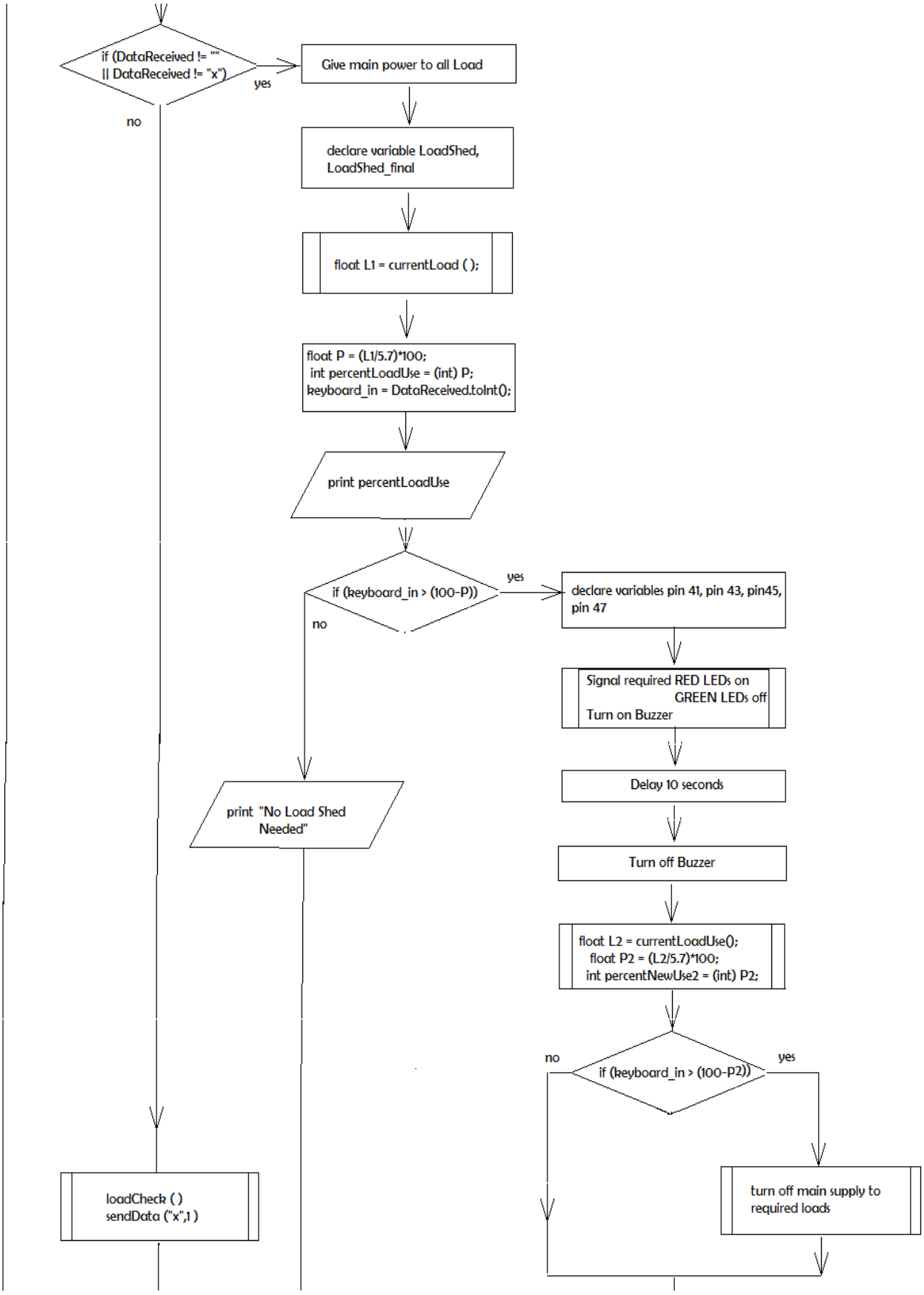


Fig. 2.2 Flowchart showing the actions at the control center

2.2 Client end

Once the information about percentage of load to be shed is received at the client end microcontroller it turns on a buzzer signaling LEDs for 10 seconds to instruct the consumer to turn off corresponding loads to keep the load within the prescribed limit. The information on current load use and load to be shed is displayed in the LCD screen besides the signaling LEDs in the signaling board. If the consumer does not turn off load voluntarily the client end Arduino turns off the supply to certain LEDs (‘loads’) to shed the required amount of load and returns the value of shed load to the control center. If the server instructed percentage of load to be shed is less than the load not being used at the client end then a character ‘x’ is transmitted to the control center to block transmission of garbage and to inform that no load shed is needed. The simplified flowchart for the code at client end is shown in Fig. 2.3.




if (DataReceived != ""
|| DataReceived != "x")
yes
no
Give main power to all Load
declare variable LoadShed,
LoadShed_final
float L1 = currentLoad ( );
float P = (L1/5.7)*100;
int percentLoadUse = (int) P;
keyboard_in = DataReceived.toInt();
print percentLoadUse
if (keyboard_in > (100-P))
yes
no
declare variables pin 41, pin 43, pin45,
pin 47
Signal required RED LEDs on
GREEN LEDs off
Turn on Buzzer
Delay 10 seconds
Turn off Buzzer
print "No Load Shed
Needed"
float L2 = currentLoadUse();
float P2 = (L2/5.7)*100;
int percentNewUse2 = (int) P2;
no
if (keyboard_in > (100-P2))
yes
turn off main supply to
required loads
loadCheck ( )
sendData ("x",1)

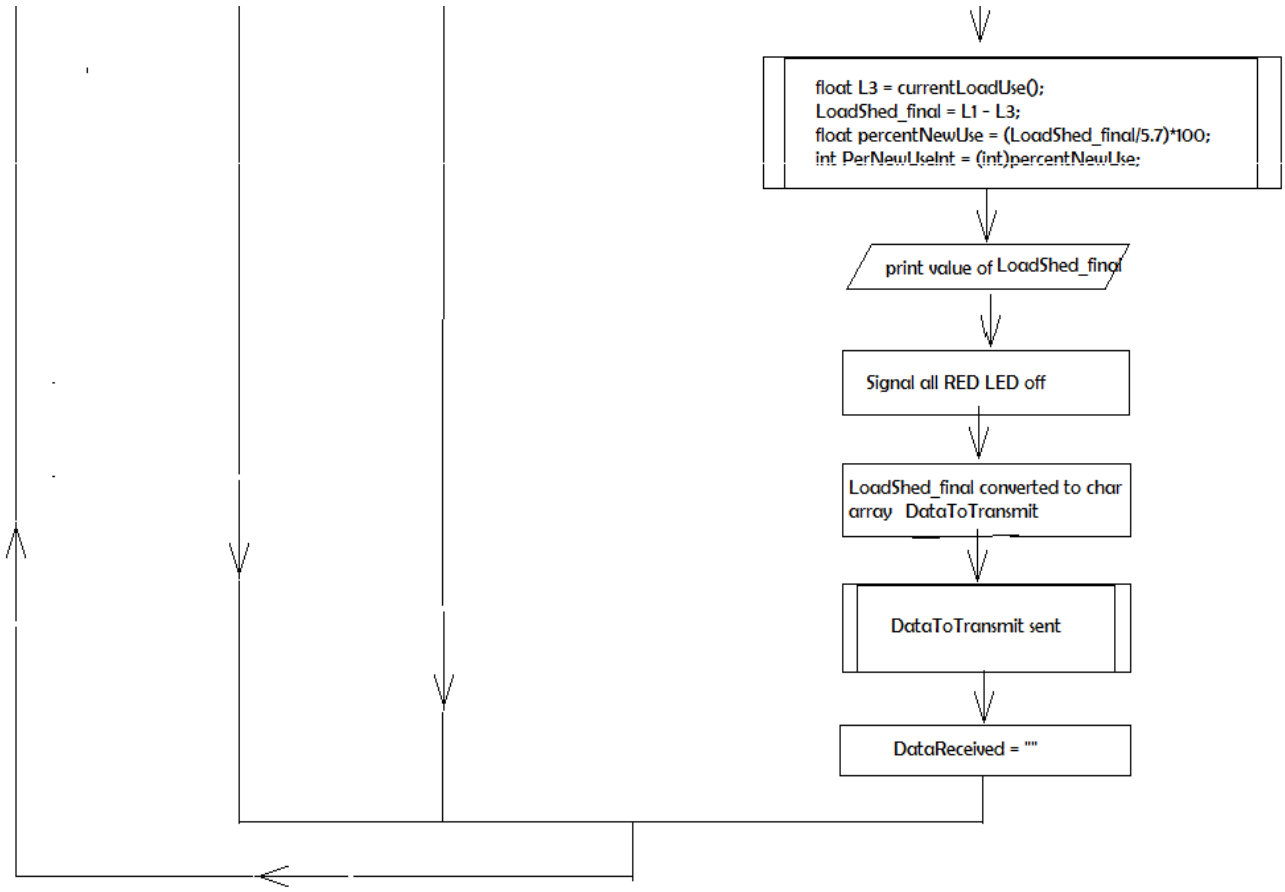


Fig. 2.3 flow chart showing the client end process

2.3 Load Control

For designing the algorithm for load control every possible combination of the controllable loads and their corresponding percentage value of the total load are listed in a lookup table (Table 2.1). In the designed model it is assumed that besides essential lighting, the other loads in a house are the 0.5kW washing machine, 0.7kW motor, 1.0kW oven and the 3.5kW AC. This makes a total controllable load of 5.7kW.

Table 2.1 Look up table showing controllable loads

LOAD VALUE	COMBINATION OF LOAD	PERCENTAGE VALUE OF LOAD
0.5	0.5	9
0.7	0.7	12
1.0	1.0	17
1.2	0.5 + 0.7	21
1.5	0.5 + 1.0	26
1.7	0.7 + 1.0	29
2.2	0.5 + 0.7 +1.0	38
3.5	3.5	61
4.0	0.5 +3.5	70
4.2	0.7 +3.5	73
4.5	1.0 + 3.5	78
4.7	0.5 + 0.7 + 3.5	82
5.0	0.5 + 1.0 + 3.5	87
5.2	0.7 + 1.0 +3.5	91
5.7	0.5 + 0.7 + 1.0 +3.5	100

The algorithm designed signals to turn off load greater than or equal to the instructed percentage. So if the server sends instruction to shed 14% load 1.0kW load will be signaled to turn off, for 75% 1.0kW and 3.5kW load will be signaled to be turned off, and so on.

Chapter 3

Implementation

3.1 NRF905 Communication

The control center needs to communicate information to and from the client. It is achieved by using NRF905 transceiver. The NRF905 is a low power consumption device which communicates at a rate of 50kbps in a working frequency range of 422.4-473.5MHZ. It has good indoor range of 300m making it suitable for the model in this work. The connection between Arduino Mega2560 pins and NRF905 is shown in the Table 3.1.

Table 3.1 Connection between NRF905 and Arduino

NRF905	Arduino Mega2560	Description
VCC	3.3V	Power (3.3V)
CE	7	Stand by- High = TX/RX mode, Low = Standby
TXE	9	Transmit or Receive Mode – High = transmit, Low = receive
PWR	8	Power up – High = on, Low = off
CD	2	Carrier detect- High when signal is detected for collision avoidance
AM	-	Address Match – High when receiving a packet that has the same address as the one set for this device, optional since state is stored in register, not used by this library

DR	3	Data Ready – High when finished transmitting/High when new data received, optional since state is stored in register, if interrupts are used this pin must be connected
SO	50	SPI MISO
SI	51	SPI MOSI
SCK	52	SPI SCK
CSN	10	SPI SS
GND	GND	Ground

Fig. 3.1 further illustrates the connection between the Arduino and the NRF905

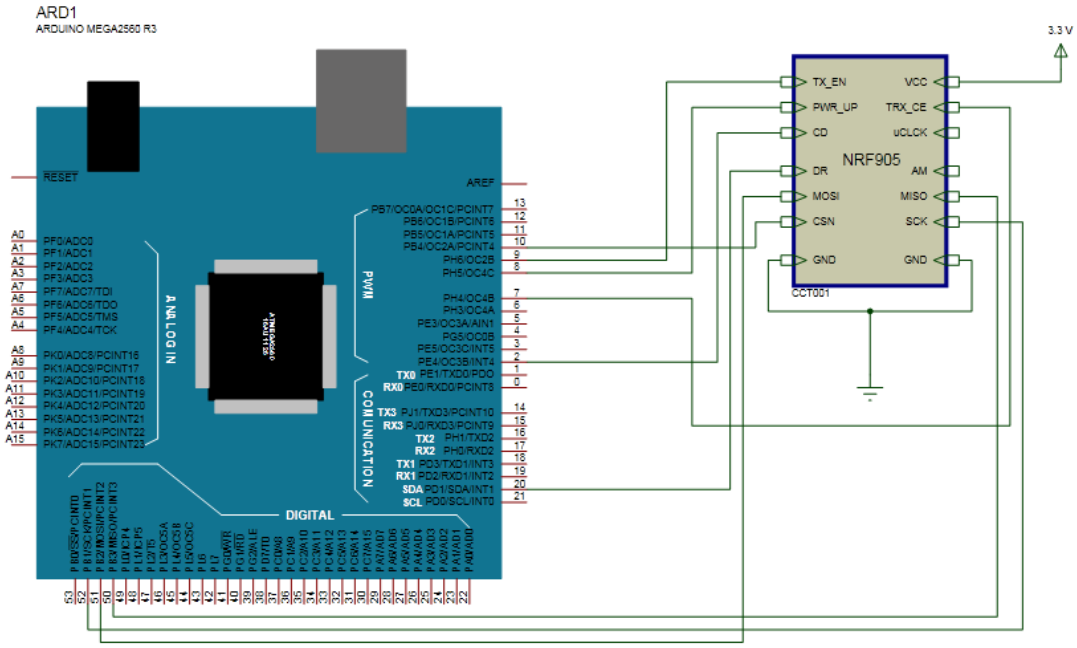


Fig. 3.1 NRF905 and Arduino board to board connection

The minimum code needed for the communication is quoted in Fig.3.2. At the beginning of the code NRF905 library and the SPI (Serial Peripheral Interface) library needs to be included and the address of the transmitting and receiving device is set. A byte array named buffer of maximum size of 32 byte is created to store the data received by the NRF905 (byte buffer [NRF905_MAX_PAYLOAD];)

```
#include <nRF905.h>                          //nrf905 library included
#include <SPI.h>                             // SPI library included
#define RXADDR {0x58, 0x6F, 0x2E, 0x10}      // Address of this device (4 bytes) declared
#define TXADDR {0xFE, 0x4C, 0xA6, 0xE5}      // Address of device to send to (4 bytes) declared

// Transmitter and Receiver address stored in different Byte Array
byte RxAddr[] = RXADDR;
byte TxAddr[] = TXADDR;
byte buffer[NRF905_MAX_PAYLOAD];
```

Fig. 3.2a: Code for communication between Arduino and NRF905-Part 1

The setup function is called when the sketch starts. It runs only once after each power up or reset of Arduino board, and it is where the baud rate is specified for serial communication with Arduino board, initializes the NRF905 library and puts the NRF905 in receiver mode at startup.

```
Void setup(){
nRF905_init();
nRF905_setRXAddress(RxAddr);
 nRF905_receive();
Serial.begin(9600);
}
```

Fig. 3.2b: Code for communication between Arduino and NRF905-Part 2

The loop function runs continuously as long as the Arduino is powered up. To send data to another NRF905 the transmitting address is set and the data(which must be generated as an character array) is sent by calling the sendData(parameters) method which had been created for easy implementation. When it is called with the parameters the NRF905 is switched to

transmitter mode and transmitter address is set and the data is sent and returns to the original function. Once the data had been sent the NRF905 is brought back to receiver mode and the data received is stored in the buffer created before, which is then converted into character array and stored in the globally declared string name DataReceived

```
void loop()
{
  // generate Character Array "myStr" which will be transmitted to other nrf905
sendData(myStr,sizeof(myStr));

  nRF905_receive();
  while(!nRF905_getData(buffer, sizeof(buffer))) {}
DataReceived = (char*)buffer;
}
```

```
void sendData(char DataToSend[], byte length)
{
      nRF905_setTXAddress(TxAddr); //
transmitting address set
      nRF905_setData(DataToSend, length); //
data being sent
      while(!nRF905_send());
}
```

Fig. 3.2c: Code for communication between Arduino and NRF905-Part 3

A similar minimum code must be uploaded to the Arduino connected to the other NRF905 for the two transceivers to communicate with each other.

3.2 Control Center Code

After the communication is set up between the client and the server the information to be sent needs to be entered into the serial monitor tool of the Arduino software via the keyboard. Fig. 3.3 shows this.

The number of characters entered in the serial monitor is stored in the predefined Serial.available() function and each individual character is read by Serial.read(). The program waits in infinite loop as long as no input is available. When input is available

(ie.Serial.available() > 0) it stores the characters available serially in the globally declared SerialData string.

```
while(!Serial.available())
 {
 }
    while(Serial.available())
  {
      delay(3);  //delay to allow buffer to fill
      if (Serial.available() > 0)
      {
          //gets one byte from serial buffer and make the string SerialData
SerialData += (char) Serial.read();
      }
  }
```

Fig. 3.3a: Code for entering information into the serial monitor-Part 1

The length of the SerialData string is checked. If the length is greater than one it implies that some value had been saved in it thus the string is converted to character array and transmitted through the transceiver by calling the sendData function. After transmission of signal the SerialData string is cleared to make ir ready for next incoming value. Otherwise “x” is sent to eliminate garbage transmission.

```
if (SerialData.length() > 1)
  {
  char myStr2[SerialData.length()];
  SerialData.toCharArray(myStr2, SerialData.length());
  sendData(myStr2, sizeof(myStr2));
  SerialData = "";
  }
  else
  {
  sendData("x", 1);
  }
```

Fig. 3.3b: Code for entering information into the serial monitor-Part 2

The NRF905 is put back to receiver mode and the program waits as long as data is not received from the other device. A blank string or “x” is sent from the client whenever there is no data to be sent. So, after checking that the characters receives is neither blank nor “x ” the characters received is printed in the serial monitor.

```
nRF905_receive();
while(!nRF905_getData(buffer, sizeof(buffer))) {}

// convert data received to string
DataRecived = (char*)buffer;
if(DataRecived != "")
{
    if(DataRecived.charAt(0) != 'x')
    {
    Serial.print("Load Shed ");
    Serial.print(DataRecived);
    Serial.println(" kW");
    }
}
```

Fig. 3.3c: Code for entering information into the serial monitor-Part 3

3.3 LCD connection and code

At the client end of the smart grid a NRF905, LCD display and few LEDs need to be connected to the Arduino. For the interfacing of the LCD +5V dc voltage was applied to the VDD and LED+ pin and ground the VSS, RW and LED- pins. A resistor (R) was inserted between the V0 and ground to adjust the contrast of the LCD display. The RS, E, D4, D5, D6 and D7 pins were connected to the digital I/O pins of the Arduino Mega board. The pin connection between the LCD display and Arduino Mega2560 is illustrated in Fig. 3.4.

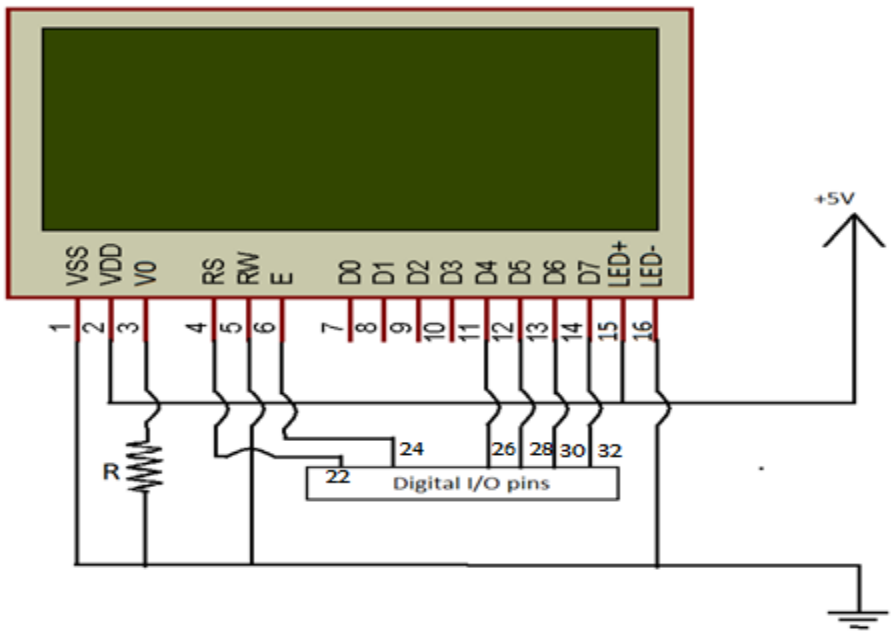


Fig. 3.4 The pin connection between the LCD display and Arduino Mega2560

The necessary codes needed for interfacing of LCD is quoted in Fig. 3.5. The liquid crystal library needs to be included and LCD interface pins in Arduino are initialized. In the setup loop the dimension (column, row) of LCD display is declared. Before printing any string in the LCD screen we need to set the cursor to certain position where the string will be printed. If the cursor is not set, any string sent for printing will be printed after the last printed character.

```
#include <LiquidCrystal.h>
LiquidCrystallcd(22, 24, 26, 28, 30, 32);

void setup() {
lcd.begin(20, 4);
}
void loop() {
lcd.setCursor(5,0);
lcd.print("Smart Grid");
}
```

Fig. 3.5 Code for interfacing LCD display

3.4 Client end connection and code

The overall idea of the client end code is given in the flowchart in Chapter 2. In the setup function pins are set at either input or output mode for signaling, controlling and checking purposes.

```
pinMode(41, INPUT);
pinMode(46, OUTPUT);
```

The digital input mode pins of Arduino can only check the state of the pin and return either true or false depending on the voltate of the point in the circuit where the pin is connected to.

```
digitalRead(41);
```

The digital output mode pins control the voltage at the output from the pin. It can either supply 5V or ground the point in the circuit it is connected to.

```
digitalWrite(46,HIGH);
digitalWrite(46,LOW);
```

The output pin 40,42,44,46 controls the RED signaling LED, pin 31,33,35,37 controls the power supply to individual loads, pin 34,36,38,39 controls the GREEN signaling LED and input pin 41,43,45,47 checks the state of the load (BLUE LED). Pin 40,31,34 and 41 are used to signal and control the 0.5kW load. The table below maps the connection of the pins to different loads.

Table 3.1 Mapping of LEDs connection corresponding to controllable loads

	Controls RED signal LED	Controls power supply to load	Controls GREEN signal LED	Checks state of load
0.5kW	40	31	34	41
0.7kW	42	33	36	43
1.0kW	44	35	38	45
3.5kW	46	37	39	47

The connection of the pins corresponding to 0.5kW load is illustrated as for instance in Fig. 3.6.

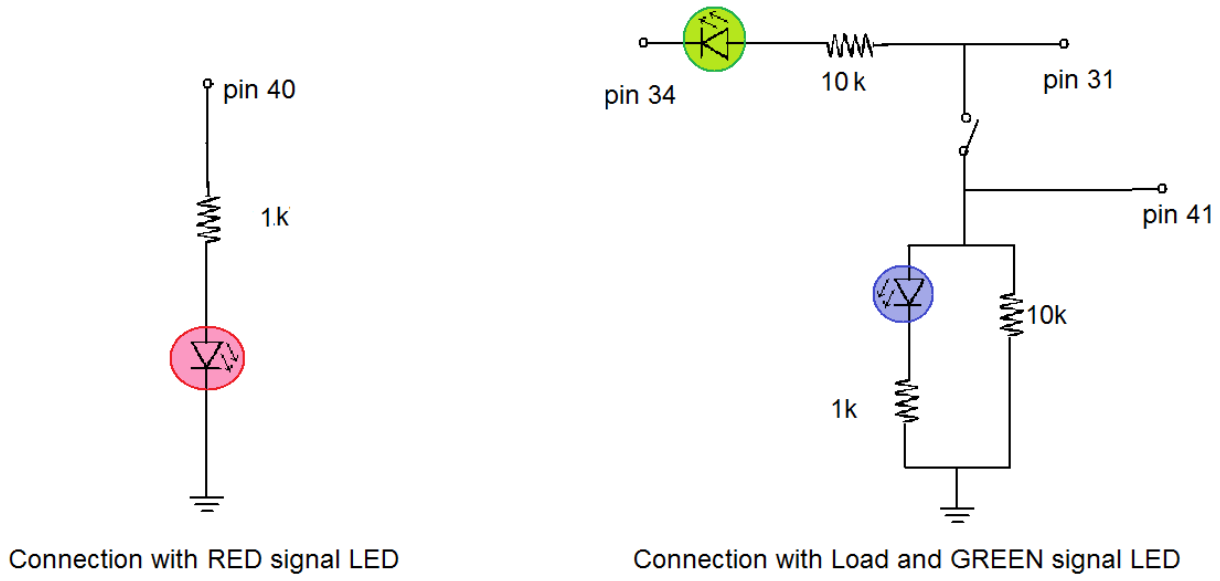


Fig. 3.6 Illustration of LEDs pin connection in Arduino board at the client end for a load

The full schematic of the physical connection is shown in Fig. 3.7. The blue LEDs represent the load which can be turned on and off by normal switch or by command from the microcontroller. The red LEDs are signaling LED which instruct user to turn off certain specific load(s) when

information is received from server. Greens LEDs are placed beside each switch and gives continuous signal to the user about which load can be used. When the load corresponding to that switch is not needed to be turned off for the current allowance of electricity, the green LED beside that switch is kept on indicating that this load can be turned on. If the load needs to be turned off according to the lookup table then the green LED is turned off. The resistors are added for preventing overflow of current through the diodes.

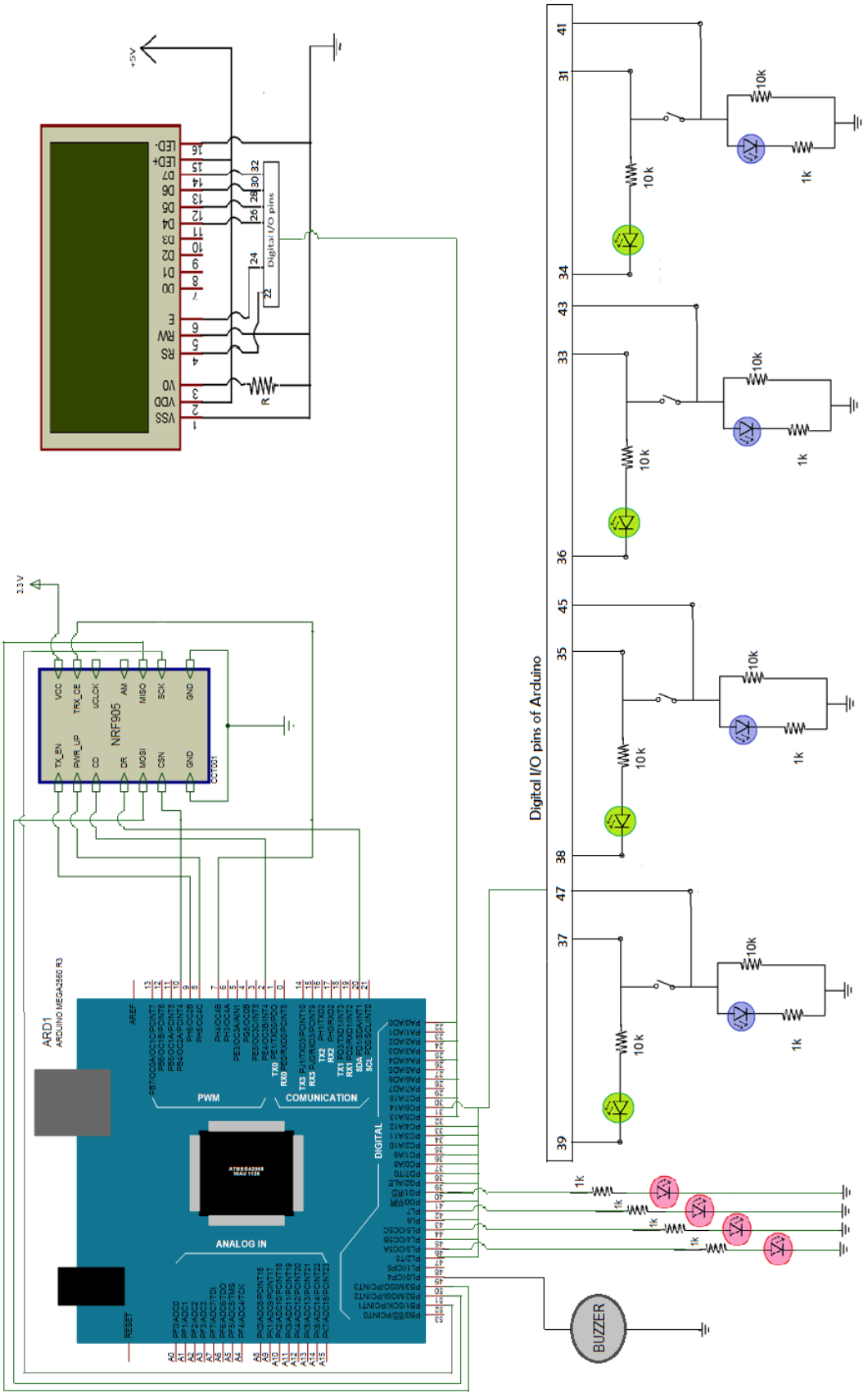


Fig. 3.7 Complete Connection at Client End

3.4.1 Detect Load Use

The pins mentioned in Table 3.1 are used for checking the current use of load and keeping the load use under check. The currentLoadUse() function checks the state (ON/OFF) of the actual loads and stores the state condition of the load in variables pin41, pin43, pin45, pin47. The variable a1, b1, c1, d1 (corresponding to 0.5kW, 0.7kW, 1.0kW and 3.5kW respectively) are adjusted according to the load being used. If the 0.5kW and 1.0kW load is being used the value of the variables will be changed as a1=0.5, b1=0, c1=1.0, d1=0 Adding the values of all the variables the total value of load used is obtained which is returned to the main function when the currentLoadUse() function ends.

```
Float currentLoadUse (){
int pin41 = digitalRead(41);
int pin43 = digitalRead(43);
int pin45 = digitalRead(45);
int pin47 = digitalRead(47);

  float a1;
  float b1;
  float c1;
  float d1;

  if (pin41==true){a1 = 0.5;} else {a1=0;}
  if (pin43==true){b1 = 0.7;} else {b1=0;}
  if (pin45==true){c1 = 1.0;} else {c1=0;}
  if (pin47==true){d1 = 3.5;} else {d1=0;}

  float L1 = a1 + b1 + c1 + d1;

  return L1;
}
```

3.4.2 Keep Load Under Check

The loadCheck() function sends signal to the GREEN LED beside each switch signaling which loads can be kept on for the current allowance of electricity. This function also use the currentLoadUse() function to determine the current load use, converts it to percentage of total load and compares it with the last received percentage value from server which had been saved in the global variable server_data. If the percentage of load being used is more than the allowed percentage load that can be used then the main supply to individual loads will be turned off as per requirement.

```
void loadCheck()
{
  if(server_data<= 9)
   {
   .
   .
   .
   }
   else if(server_data <= 12)
   {
     digitalWrite(31,HIGH);
     digitalWrite(33,LOW);
     digitalWrite(35,HIGH);
     digitalWrite(37,HIGH);   }
   else if(other condition)
   {
   }


  float L1 = currentLoadUse();
  float P = (L1/5.7)*100;
  intpercentLoadUse = (int) P;

  if(server_data > (100-percentLoadUse))
  {
```

```
  lcd.setCursor(0,0);
  lcd.print("ExcessUse: AUTO OFF");
  lcd.setCursor(0,2);
  lcd.print("LoadUse= ");
  lcd.print(percentLoadUse);
  lcd.print("% ");
  lcd.print(L1);
  lcd.print("kW");

  if(server_data <= 9)
   {
   .
   .
   .
   }
   else if(server_data <= 12)
   {
   digitalWrite(31,HIGH);
   digitalWrite(33,LOW);
   digitalWrite(35,HIGH);
   digitalWrite(37,HIGH);
   }
   else if(other condition)
   {
   }
  }
}
```

3.4.3 Central Client Code

The loop function runs continuously and calls the receiveData() function each time it runs.

```
void loop()
{
receieveData();
}
```

Inside the receiveData() function nrf905 is first set to receiver mode and the program waits for data from the server as long as it don't get any value and at the same time it keeps the load under check.

```
nRF905_receive();
    // wait until data recived
    while(!nRF905_getData(buffer, sizeof(buffer)))
    {
    loadCheck();
    }
```

Once any value is received it is stored in the globally declared string DataReceived. After checking that the data received is neither an empty string nor "x" the power to all the load is turned on and some variables are declared for load calculation. The current load use value is stored in L1, it is converted to percentage of total load and is then converted to integer form.

```
DataReceived = (char*)buffer;
if(DataReceived != "")
    {
          if(DataReceived.charAt(0) != 'x')
      {
      digitalWrite(31,HIGH);
      digitalWrite(33,HIGH);
      digitalWrite(35,HIGH);
      digitalWrite(37,HIGH);
```

```
float LoadShed;
float LoadShed_final;
float L1 = currentLoadUse();
float P = (L1/5.7)*100;
intpercentLoadUse = (int) P;
server_data = DataReceived.toInt();
```

Whether the requested amount of load shed from the server is greater than the percentage of load not being used, is checked. If the condition is true the client is signaled to turn off specific loads, otherwise they are informed that no load shed is needed right away and "x" is transmitted to server to block garbage transmission.

```
if(server_data > (100-percentLoadUse))
{
.
.
.
} else
{
lcd.setCursor(0,3);
lcd.print("No Load Shed needed ");
sendData("x", 1);
}
```

Once inside the if(server_data > (100-percentLoadUse)) condition loop it stores the state of each load in variable pin42, pin43, pin 45 and pin47. And in accordance with the value of server_data specified RED LED are turned on for signaling to shed load; but before turning on the RED LED it is checked whether the load is being used or not. Load is signaled to turnoff only if it is being used. The total amount of load that needs to be shed is also displayed in the LCD screen.

```
int pin41 = digitalRead(41);
```

```
    int pin43 = digitalRead(43);
    int pin45 = digitalRead(45);
    int pin47 = digitalRead(47);

    lcd.setCursor(0,1);
    lcd.print("Shed load ");
    lcd.print(server_data);
    lcd.print("% ");

          if(server_data <= 9)
          {
             .
             .
             .
          }
          else if(server_data <= 12)
          {
              digitalWrite(40,LOW);
              if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
              digitalWrite(44,LOW);
              digitalWrite(46,LOW);
              lcd.print("0.7 kW");
              Serial.println("inside keyboard_in<= 12 loop");

              digitalWrite(34,LOW);
              digitalWrite(36,HIGH);
              digitalWrite(38,LOW);
              digitalWrite(39,LOW);
          }else if (Other condition)
          {

    }
```

After sending the signal the program turns on the buzzer to get the client's attention, waits for 10 seconds to allow client to turn off load voluntarily then turns the buzzer off. The positive pin of the buzzer is connected to pin49 so making pin49 high turns on the buzzer. The current load use is converted to percentage.

```
    digitalWrite(49,HIGH);
    delay(10000);
```

```
digitalWrite(49,LOW);
lcd.setCursor(0,0);
lcd.print("Disconnecting timeup");
float L2 = currentLoadUse();
float P2 = (L2/5.7)*100;
int percentNewUse2 = (int) P2;
```

If the required percentage load shed is greater that the percentage of current unused load the main supply to individual load will be turned off similarly as it had been turned off in the loadCheck () function.

After a final check of current load use the load shed upon receiving the signal is calculated, displayed in the LCD screen and all the RED signaling LEDs are turned off.

```
float L3 = currentLoadUse();
LoadShed_final = L1 - L3;

float percentNewUse = (LoadShed_final/5.7)*100;
intPerNewUseInt = (int)percentNewUse;

lcd.setCursor(0,3);
lcd.print("LoadSheded ");
lcd.print((PerNewUseInt));
lcd.print("% ");
lcd.print(LoadShed_final);
lcd.setCursor(18,3);
lcd.print("kW");

digitalWrite(40,LOW);
digitalWrite(42,LOW);
digitalWrite(44,LOW);
digitalWrite(46,LOW);
```

The valued of load shed is converted to character array and transmitted to the server by calling the sendData function. After transmission the string DataReceived is changed back to blank string making it ready to save the next signal.

```
char buff[5];
String transmit = dtostrf(LoadShed_final, 1, 3, buff);
char DataToTransmit[transmit.length()];
transmit.toCharArray(DataToTransmit, transmit.length());
sendData(DataToTransmit, sizeof(DataToTransmit));
DataReceived = "";
```

It is where the receiveData() function ends and goes back to the next line of the loop function from where it was called.

Chapter 4

Conclusion

4.1 Conclusion

A laboratory model has been developed using microcontrollers, trans-receiving antennas, LED and LCDs to simulate the way load management will be done in a real-life smart grid with two way communication between server and client. The prototype successfully demonstrates the concept and is easy to be visualized by the future engineers so that the general public can be made aware of the smart grid vision.

4.2 Suggestions for further work:

For real life implementations of the developed laboratory model there are many points to focus.

a. Several number of Arduino can be used to represent multiple numbers of clients.
b. Multiple information e.g. the unit price for the electricity, amount of load needed to be shed, system frequency etc. all together can be transmitted as a single massage by modifying the code on a higher degree following the available protocols that the present mobile company uses.
c. Other microcontroller interfaced with mobile phone SIM can be used so that info can be sent to and fro between server and multiple clients.

For a developing country like Bangladesh also the following points are worth consideration as primary steps for introducing smart grid.

a. Undertaking first a pilot project on smart grid load management in a small area of a distribution utility e.g. DESCO/DPDC and interfacing the client end ‘smart meter’ (Arduino) with real loads.

b. Sharing the knowledge and expertise of academia on smart grid with utility engineers under a collaboration framework.

References

1. Sinha, A. ; Neogi, S. ; Lahiri, R.N. ; Chowdhury, S. ; Chowdhury, S.P. ; Chakraborty, N. ”Smart grid initiative for power distribution utility in India”, Power and Energy Society General Meeting IEEE, 2011, 24-29 July 2011,pp. 1 – 8.

2. Misra, Satyajayant ; Xue, Guoliang ; Yang, Dejun “Smart Grid — The New and Improved Power Grid: A Survey”, Communications Surveys & Tutorials IEEE, 2012, Volume:14 , Issue: 4, pp. 944 – 980.

3. Niyato, D.; Ping Wang “Cooperative transmission for meter data collection in smart grid”, Communications Magazine IEEE, 2012, Volume:50 , Issue: 4 pp.90 – 97.

4. Sahu, N. ; Dehalwar, V “Intelligent machine to machine communication in home area network for smart grid”, International Conference on Computing Communication & Networking Technologies (ICCCNT), 26-28 July 2012, pp.1 – 6.

5. Xiang Yuan ; ZhenXing Qian ; Yang Zhou ; You Wang ; Mingming Yan “Discussion on the development trend of smart grid and its key technology**”,** Conference on Electricity Distribution (CICED), 10-14 Sept. 2012, pp.1 – 8.

6. M. Fadaeenejada,n, A.M.Saberiana,n, Mohd.Fadaeeb, M.A.M.Radzia, H. Hizama, M.Z.A.AbKadir. ” The present and future of smart power grid in developing countries”,Renewable and Sustainable Energy Reviews, Volume 29, January 2014, pp. 828–834.

7. I. Khan, A. Mahmood, N. Javaid, S. Razzaq, R. D. Khan, M. Ilahi “Home Energy Management Systems in Future Smart Grids”, J. Basic. Appl. Sci. Res., 3(3),2013, pp.1224-1231.

8. Erol-Kantarci, M.; Sarker, J.H. ; Mouftah, H.T. “Communication based plug in hybrid electric vehicle load management in the smart grid”, Computers and Communications (ISCC), IEEE Symposium on , June 28 -July 1 2011, pp.404 – 409.

9. http://blog.zakkemble.co.uk/nrf905-avrarduino-librarydriver/

Appendixes

The following codes were developed in the software Arduino, version 1.0.5-r2 for the server and client. The library for NRF905 transceiver was downloaded from http://blog.zakkemble.co.uk/nrf905-avrarduino-librarydriver/

A.1 Server code

```
//takes keyboard value and sends it.

//receives returning signal and prints it in serial monitor


#include <nRF905.h>

#include <SPI.h>

#define RXADDR {0x58, 0x6F, 0x2E, 0x10} // Address of this device (4 bytes)

#define TXADDR {0xFE, 0x4C, 0xA6, 0xE5} // Address of device to send to (4 bytes)


byte RxAddr[] = RXADDR;

byte TxAddr[] = TXADDR;

byte buffer[NRF905_MAX_PAYLOAD];

String SerialData;

String Datareceived;


void setup()

{

        nRF905_init();

        nRF905_setRXAddress(RxAddr);

      nRF905_receive();

      Serial.begin(9600);
```

```
        Serial.println("Server started");
}

void sendData(char DataToSend[], byte length)
{
      nRF905_setTXAddress(TxAddr);
      nRF905_setData(DataToSend, length);
      while(!nRF905_send());
}

void loop()
{
    // wait for serial data
    while(!Serial.available())
  {
  }

    // read serial data
    while(Serial.available())
    {
        delay(3);  //delay to allow buffer to fill
        if (Serial.available() > 0)
        {
            //gets one byte from serial buffer and make the string SerialData
            SerialData += (char) Serial.read();
```

```
    }
}

if (SerialData.length() > 1)
{
    char myStr2[SerialData.length()];
    SerialData.toCharArray(myStr2, SerialData.length());
    //Serial.println(myStr2);

    sendData(myStr2, sizeof(myStr2));

    // make SerialData clear
    SerialData = "";
}
else
{
    // send null data
    sendData("x", 1);
}

// wait until data recived
nRF905_receive();
while(!nRF905_getData(buffer, sizeof(buffer))) {}

// get data as string
```

```
    Datareceived = (char*)buffer;


    if(Datareceived != "")
    {
        if(Datareceived.charAt(0) != 'x')
        {
          Serial.print("Load Shed ");
          Serial.print(Datareceived);
          Serial.println(" kW");


        }
    }
}
```

A.2 Client Code

```
#include <nRF905.h>
#include <SPI.h>
#include <LiquidCrystal.h>


#define RXADDR {0xFE, 0x4C, 0xA6, 0xE5} // Address of this device (4 bytes)
#define TXADDR {0x58, 0x6F, 0x2E, 0x10} // Address of device to send to (4 bytes)


// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(22, 24, 26, 28, 30, 32);
```

```
byte RxAddr[] = RXADDR;
byte TxAddr[] = TXADDR;
byte buffer[NRF905_MAX_PAYLOAD];
String DataReceived;
int server_data;

void setup()
{
    // Start up
    nRF905_init();
    nRF905_setRXAddress(RxAddr);
    nRF905_receive();
    Serial.begin(9600);
    Serial.println("Client started");

    lcd.begin(20, 4);
    lcd.setCursor(5,0);
    lcd.print("Smart Grid");

//set pin modes
    pinMode(40, OUTPUT);
    pinMode(42, OUTPUT);
    pinMode(44, OUTPUT);
    pinMode(46, OUTPUT);
```

```
    pinMode(34, OUTPUT);
    pinMode(36, OUTPUT);
    pinMode(38, OUTPUT);
    pinMode(39, OUTPUT);

    pinMode(41, INPUT);
    pinMode(43, INPUT);
    pinMode(45, INPUT);
    pinMode(47, INPUT);

    pinMode(31, OUTPUT);
    pinMode(33, OUTPUT);
    pinMode(35, OUTPUT);
    pinMode(37, OUTPUT);

    pinMode(49, OUTPUT);

// turn on main supply to all load
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,HIGH);
    digitalWrite(37,HIGH);

// signal GREEN LEDs off
    digitalWrite(34,LOW);
```

```
    digitalWrite(36,LOW);
    digitalWrite(38,LOW);
    digitalWrite(39,LOW);
}

void loop()
{
    receiveData();
}



void receiveData()
{
    nRF905_receive();

    // wait until data recived
    while(!nRF905_getData(buffer, sizeof(buffer)))
    {
    loadCheck();
    }

    // get data as string
    DataReceived = (char*)buffer;
```

```
if(DataReceived != "")
{
  if(DataReceived.charAt(0) != 'x')
  {
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,HIGH);
    digitalWrite(37,HIGH);

    Serial.println(DataReceived);

    Serial.print("Shed ");
    Serial.print(DataReceived);
    Serial.println("% load");

    float LoadShed;
    float LoadShed_final;

    float L1 = currentLoadUse();
    float P = (L1/5.7)*100;
    int percentLoadUse = (int) P;
    server_data = DataReceived.toInt();

    Serial.print("server_data =");
    Serial.println(server_data);
```

```
Serial.print("P =");
Serial.println(P);
Serial.print("PercentLoadUse =");
Serial.println(percentLoadUse);

lcd.setCursor(0,2);
lcd.print("LoadUse= ");
lcd.print(percentLoadUse);
lcd.print("% ");
lcd.print(L1);
lcd.print("kW");

lcd.setCursor(0,1);
lcd.print("Shed load ");
lcd.print(server_data);
lcd.print("% ");
   if(server_data <= 9)
   {
    if (server_data == 0)
      {
       lcd.print("0 kW");
      }
      lcd.print("0.5 kW");
    }
    else if(server_data <= 12)
```

```
{
  lcd.print("0.7 kW");
}
else if(server_data <= 17)
{
  lcd.print("1.0 kW");
}
else if(server_data <= 21)
{
  lcd.print("1.2 kW");
}
else if(server_data <= 26)
{
  lcd.print("1.5 kW");
}
else if(server_data <= 29)
{
  lcd.print("1.7 kW");
}
else if(server_data <= 38)
{
  lcd.print("2.2 kW");
}
else if(server_data <= 61)
{
```

```
  lcd.print("3.5 kW");
}
else if(server_data <= 70)
{
  lcd.print("4.0 kW");
}
else if(server_data <= 73)
{
  lcd.print("4.2 kW");
}
else if(server_data <= 78)
{
  lcd.print("4.5 kW");
}
else if(server_data <= 82)
{
  lcd.print("4.7 kW");
}
else if(server_data <= 87)
{
  lcd.print("5.0 kW");
}
else if(server_data <= 91)
{
  lcd.print("5.2 kW");
```

```
        }
        else
        {
          lcd.print("5.7 kW");
         }

//control RED and GREEN signal LED according to server data
      if(server_data > (100-percentLoadUse))
      {
        Serial.println("inside server_data < percentLoadUse loop");

        int pin41 = digitalRead(41);
        int pin43 = digitalRead(43);
        int pin45 = digitalRead(45);
        int pin47 = digitalRead(47);


        if(server_data <= 9)
        {
          if (server_data == 0)
          {
            digitalWrite(40,LOW);
            digitalWrite(42,LOW);
            digitalWrite(44,LOW);
            digitalWrite(46,LOW);
```

```
        lcd.print("0 kW");
        Serial.println("inside server_data == 0 loop");

        digitalWrite(34,LOW);
        digitalWrite(36,LOW);
        digitalWrite(38,LOW);
        digitalWrite(39,LOW);
    }

    if (pin41 == true){digitalWrite(40,HIGH);}else{digitalWrite(40,LOW);}
    digitalWrite(42,LOW);
    digitalWrite(44,LOW);
    digitalWrite(46,LOW);
    lcd.print("0.5 kW");
    Serial.println("inside server_data <= 9 loop");

    digitalWrite(34,HIGH);
    digitalWrite(36,LOW);
    digitalWrite(38,LOW);
    digitalWrite(39,LOW);


}
else if(server_data <= 12)
{
```

```
    digitalWrite(40,LOW);
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    digitalWrite(44,LOW);
    digitalWrite(46,LOW);
    lcd.print("0.7 kW");
    Serial.println("inside server_data <= 12 loop");

    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,LOW);

}
else if(server_data <= 17)
{
    digitalWrite(40,LOW);
    digitalWrite(42,LOW);
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    digitalWrite(46,LOW);
    lcd.print("1.0 kW");
    Serial.println("inside server_data <= 17 loop");

    digitalWrite(34,LOW);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
```

```
    digitalWrite(39,LOW);


}
else if(server_data <= 21)
{
    if (pin41 == true){digitalWrite(40,HIGH);} else{digitalWrite(40,LOW)}
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    digitalWrite(44,LOW);
    digitalWrite(46,LOW);
    lcd.print("0.7 kW");
    Serial.println("inside server_data <= 21 loop");


    digitalWrite(34,HIGH);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,LOW);


}
else if(server_data <= 26)
{
    if (pin41 == true){digitalWrite(40,HIGH);} else{digitalWrite(40,LOW);}
    digitalWrite(42,LOW);
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    digitalWrite(46,LOW);
    lcd.print("1.5 kW");
```

```
    Serial.println("inside server_data <= 26 loop");


    digitalWrite(34,HIGH);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
    digitalWrite(39,LOW);


}
else if(server_data <= 29)
{
    digitalWrite(40,LOW);
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    digitalWrite(46,LOW);
    lcd.print("1.7 kW");
    Serial.println("inside server_data <= 29 loop");


    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,HIGH);
    digitalWrite(39,LOW);


}
else if(server_data <= 38)
{
```

```
    if (pin41 == true){digitalWrite(40,HIGH);} else{digitalWrite(40,LOW);}
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    digitalWrite(46,LOW);
    lcd.print("2.2 kW");
    Serial.println("inside server_data <= 38 loop");

    digitalWrite(34,HIGH);
    digitalWrite(36,HIGH);
    digitalWrite(38,HIGH);
    digitalWrite(39,LOW);

}
else if(server_data <= 61)
{
    digitalWrite(40,LOW);
    digitalWrite(42,LOW);
    digitalWrite(44,LOW);
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("3.5 kW");
    Serial.println("inside server_data <= 61 loop");

    digitalWrite(34,LOW);
    digitalWrite(36,LOW);
    digitalWrite(38,LOW);
```

```
    digitalWrite(39,HIGH);


}
else if(server_data <= 70)
{
    if (pin41 == true){digitalWrite(40,HIGH);} else{digitalWrite(40,LOW);}
    digitalWrite(42,LOW);
    digitalWrite(44,LOW);
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("4.0 kW");
    Serial.println("inside server_data <= 70 loop");


    digitalWrite(34,HIGH);
    digitalWrite(36,LOW);
    digitalWrite(38,LOW);
    digitalWrite(39,HIGH);


}
else if(server_data <= 73)
{
    digitalWrite(40,LOW);
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    digitalWrite(44,LOW);
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("4.2 kW");
```

```
    Serial.println("inside server_data <= 73 loop");


    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,HIGH);


}
else if(server_data <= 78)
{
    digitalWrite(40,LOW);
    digitalWrite(42,LOW);
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("4.5 kW");
    Serial.println("inside server_data <= 78 loop");


    digitalWrite(34,LOW);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
    digitalWrite(39,HIGH);


}
else if(server_data <= 82)
{
```

```
    if (pin41 == true){digitalWrite(40,HIGH);} else{digitalWrite(40,LOW);}
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    digitalWrite(44,LOW);
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("0.7 kW");
    Serial.println("inside server_data <= 82 loop");

    digitalWrite(34,HIGH);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,HIGH);

}
else if(server_data <= 87)
{
    if (pin41 == true){digitalWrite(40,HIGH);} else{digitalWrite(40,LOW);}
    digitalWrite(42,LOW);
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("5.0 kW");
    Serial.println("inside server_data <= 87 loop");

    digitalWrite(34,HIGH);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
```

```
    digitalWrite(39,HIGH);


}
else if(server_data <= 91)
{
    digitalWrite(40,LOW);
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("5.2 kW");
    Serial.println("inside server_data <= 91 loop");


    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,HIGH);
    digitalWrite(39,HIGH);


}
else
{
    if (pin41 == true){digitalWrite(40,HIGH);} else{digitalWrite(40,LOW);}
    if (pin43 == true){digitalWrite(42,HIGH);} else{digitalWrite(42,LOW);}
    if (pin45 == true){digitalWrite(44,HIGH);} else{digitalWrite(44,LOW);}
    if (pin47 == true){digitalWrite(46,HIGH);} else{digitalWrite(46,LOW);}
    lcd.print("5.7 kW");
```

```
            Serial.println("inside server_data else loop");

            digitalWrite(34,HIGH);
            digitalWrite(36,HIGH);
            digitalWrite(38,HIGH);
            digitalWrite(39,HIGH);

        }

        lcd.setCursor(0,0);
        lcd.print("Please disconnect  ");
        digitalWrite(49,HIGH);
        delay(10000);
        digitalWrite(49,LOW);
        lcd.setCursor(0,0);
        lcd.print("Disconnecting timeup  ");
        Serial.println("Timeup for disconnecting");

        float L2 = currentLoadUse();
        float P2 = (L2/5.7)*100;
        int percentNewUse2 = (int) P2;

// turn off main supply to respective load if user don't shed enough load
        if ((100 - percentNewUse2) < server_data){
```

```
Serial.println("Not enought load shed... Load being Shed automatically");
lcd.setCursor(0,0);
lcd.print("ExcessUse: AUTO OFF ");
 if(server_data <= 9)
 {
    if(server_data == 0)
    {
       digitalWrite(31,HIGH);
       digitalWrite(33,HIGH);
       digitalWrite(35,HIGH);
       digitalWrite(37,HIGH);
    }
    else
    {
       digitalWrite(31,LOW);
       digitalWrite(33,HIGH);
       digitalWrite(35,HIGH);
       digitalWrite(37,HIGH);
    }
 }
 else if(server_data <= 12)
 {
     digitalWrite(31,HIGH);
     digitalWrite(33,LOW);
     digitalWrite(35,HIGH);
```

```
    digitalWrite(37,HIGH);
}
else if(server_data <= 17)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,LOW);
    digitalWrite(37,HIGH);
}
else if(server_data <= 21)
{
    digitalWrite(31,LOW);
    digitalWrite(33,LOW);
    digitalWrite(35,HIGH);
    digitalWrite(37,HIGH);
}
else if(server_data <= 26)
{
    digitalWrite(31,LOW);
    digitalWrite(33,HIGH);
    digitalWrite(35,LOW);
    digitalWrite(37,HIGH);
}
else if(server_data <= 29)
{
```

```
    digitalWrite(31,HIGH);
    digitalWrite(33,LOW);
    digitalWrite(35,LOW);
    digitalWrite(37,HIGH);
}
else if(server_data <= 38)
{
    digitalWrite(31,LOW);
    digitalWrite(33,LOW);
    digitalWrite(35,LOW);
    digitalWrite(37,HIGH);
}
else if(server_data <= 61)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);
}
else if(server_data <= 70)
{
    digitalWrite(31,LOW);
    digitalWrite(33,HIGH);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);
```

```
}
else if(server_data <= 73)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,LOW);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);

}
else if(server_data <= 78)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,LOW);
    digitalWrite(37,LOW);
}
else if(server_data <= 82)
{
    digitalWrite(31,LOW);
    digitalWrite(33,LOW);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);
}
else if(server_data <= 87)
{
```

```
            digitalWrite(31,LOW);
            digitalWrite(33,HIGH);
            digitalWrite(35,LOW);
            digitalWrite(37,LOW);
        }
        else if(server_data <= 91)
        {
            digitalWrite(31,HIGH);
            digitalWrite(33,LOW);
            digitalWrite(35,LOW);
            digitalWrite(37,LOW);
        }
        else
        {

            digitalWrite(31,LOW);
            digitalWrite(33,LOW);
            digitalWrite(35,LOW);
            digitalWrite(37,LOW);
        }
}
    float L3 = currentLoadUse();
    LoadShed_final = L1 - L3;

    float percentNewUse = (LoadShed_final/5.7)*100;
```

```
int PerNewUseInt = (int)percentNewUse;


Serial.print("Loadsheded = ");
Serial.print(LoadShed_final);
Serial.print("kW      ");
Serial.print(percentNewUse);
Serial.println("%");


lcd.setCursor(0,3);
lcd.print("LoadSheded ");
lcd.print((PerNewUseInt));
lcd.print("% ");
lcd.print(LoadShed_final);
lcd.setCursor(18,3);
lcd.print("kW");


digitalWrite(40,LOW);
digitalWrite(42,LOW);
digitalWrite(44,LOW);
digitalWrite(46,LOW);


char buff[5];
String transmit = dtostrf(LoadShed_final, 1, 3, buff);
char DataToTransmit[transmit.length()];
transmit.toCharArray(DataToTransmit, transmit.length());
```

```
        Serial.println("String value: ");
        Serial.println(transmit);

        // send response
        sendData(DataToTransmit, sizeof(DataToTransmit));
        DataReceived = "";
    }
    else
    {
        Serial.println("No Load shed needed");

        lcd.setCursor(0,3);
        lcd.print("No Load Shed needed ");

        lcd.setCursor(0,0);
        lcd.print("     SMART GRID     ");

        sendData("x", 1);
    }
}
else
{
    loadCheck();
    sendData("x", 1);
```

```
      }
    }
    else
    {
      loadCheck();
      sendData("x", 1);
    }
}

void loadCheck(){
// control GREEN LED
if(server_data <= 9)
 {
     if(server_data == 0)
     {
        digitalWrite(34,LOW);
        digitalWrite(36,LOW);
        digitalWrite(38,LOW);
        digitalWrite(39,LOW);
     }
     else
     {
        digitalWrite(34,HIGH);
        digitalWrite(36,LOW);
        digitalWrite(38,LOW);
```

```
      digitalWrite(39,LOW);
    }
}
else if(server_data <= 12)
{
    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,LOW);
}
else if(server_data <= 17)
{
    digitalWrite(34,LOW);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
    digitalWrite(39,LOW);
}
else if(server_data <= 21)
{
    digitalWrite(34,HIGH);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,LOW);
}
else if(server_data <= 26)
```

```
{
    digitalWrite(34,HIGH);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
    digitalWrite(39,LOW);
}
else if(server_data <= 29)
{
    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,HIGH);
    digitalWrite(39,LOW);
}
else if(server_data <= 38)
{
    digitalWrite(34,HIGH);
    digitalWrite(36,HIGH);
    digitalWrite(38,HIGH);
    digitalWrite(39,LOW);
}
else if(server_data <= 61)
{
    digitalWrite(34,LOW);
    digitalWrite(36,LOW);
    digitalWrite(38,LOW);
```

```
    digitalWrite(39,HIGH);
}
else if(server_data <= 70)
{
    digitalWrite(34,HIGH);
    digitalWrite(36,LOW);
    digitalWrite(38,LOW);
    digitalWrite(39,HIGH);
}
else if(server_data <= 73)
{
    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,HIGH);

}
else if(server_data <= 78)
{
    digitalWrite(34,LOW);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
    digitalWrite(39,HIGH);
}
else if(server_data <= 82)
```

```
{
    digitalWrite(34,HIGH);
    digitalWrite(36,HIGH);
    digitalWrite(38,LOW);
    digitalWrite(39,HIGH);
}
else if(server_data <= 87)
{
    digitalWrite(34,HIGH);
    digitalWrite(36,LOW);
    digitalWrite(38,HIGH);
    digitalWrite(39,HIGH);
}
else if(server_data <= 91)
{
    digitalWrite(34,LOW);
    digitalWrite(36,HIGH);
    digitalWrite(38,HIGH);
    digitalWrite(39,HIGH);
}
else
{

    digitalWrite(34,HIGH);
    digitalWrite(36,HIGH);
```

```
    digitalWrite(38,HIGH);
    digitalWrite(39,HIGH);
  }

  float L1 = currentLoadUse();
  float P = (L1/5.7)*100;
  int percentLoadUse = (int) P;

// turn off main supply to load if excess is being used
  if(server_data > (100-percentLoadUse)){

    Serial.println("Load Use beyond Limit ");

    lcd.setCursor(0,0);
    lcd.print("ExcessUse: AUTO OFF ");
    lcd.setCursor(0,2);
    lcd.print("LoadUse= ");
    lcd.print(percentLoadUse);
    lcd.print("% ");
    lcd.print(L1);
    lcd.print("kW");
    if(server_data <= 9)
      {
        if(server_data == 0)
        {
```

```
        digitalWrite(31,HIGH);
        digitalWrite(33,HIGH);
        digitalWrite(35,HIGH);
        digitalWrite(37,HIGH);
    }
    else
    {
        digitalWrite(31,LOW);
        digitalWrite(33,HIGH);
        digitalWrite(35,HIGH);
        digitalWrite(37,HIGH);
    }
}
else if(server_data <= 12)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,LOW);
    digitalWrite(35,HIGH);
    digitalWrite(37,HIGH);
}
else if(server_data <= 17)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,LOW);
```

```
    digitalWrite(37,HIGH);
}
else if(server_data <= 21)
{
    digitalWrite(31,LOW);
    digitalWrite(33,LOW);
    digitalWrite(35,HIGH);
    digitalWrite(37,HIGH);
}
else if(server_data <= 26)
{
    digitalWrite(31,LOW);
    digitalWrite(33,HIGH);
    digitalWrite(35,LOW);
    digitalWrite(37,HIGH);
}
else if(server_data <= 29)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,LOW);
    digitalWrite(35,LOW);
    digitalWrite(37,HIGH);
}
else if(server_data <= 38)
{
```

```
    digitalWrite(31,LOW);
    digitalWrite(33,LOW);
    digitalWrite(35,LOW);
    digitalWrite(37,HIGH);
}
else if(server_data <= 61)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);
}
else if(server_data <= 70)
{
    digitalWrite(31,LOW);
    digitalWrite(33,HIGH);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);
}
else if(server_data <= 73)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,LOW);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);
```

```
}
else if(server_data <= 78)
{
    digitalWrite(31,HIGH);
    digitalWrite(33,HIGH);
    digitalWrite(35,LOW);
    digitalWrite(37,LOW);
}
else if(server_data <= 82)
{
    digitalWrite(31,LOW);
    digitalWrite(33,LOW);
    digitalWrite(35,HIGH);
    digitalWrite(37,LOW);
}
else if(server_data <= 87)
{
    digitalWrite(31,LOW);
    digitalWrite(33,HIGH);
    digitalWrite(35,LOW);
    digitalWrite(37,LOW);
}
else if(server_data <= 91)
{
    digitalWrite(31,HIGH);
```

```
            digitalWrite(33,LOW);
            digitalWrite(35,LOW);
            digitalWrite(37,LOW);
        }
        else
        {
            digitalWrite(31,LOW);
            digitalWrite(33,LOW);
            digitalWrite(35,LOW);
            digitalWrite(37,LOW);
        }
  }
}

float currentLoadUse (){
  int pin41 = digitalRead(41);
  int pin43 = digitalRead(43);
  int pin45 = digitalRead(45);
  int pin47 = digitalRead(47);

  float a1;
  float b1;
  float c1;
  float d1;
```

```
  if (pin41==true){a1 = 0.5;} else {a1=0;}
  if (pin43==true){b1 = 0.7;} else {b1=0;}
  if (pin45==true){c1 = 1.0;} else {c1=0;}
  if (pin47==true){d1 = 3.5;} else {d1=0;}

  float L1 = a1 + b1 + c1 + d1;

  return L1;

}

void sendData(char DataToSend[], byte length)
{
      nRF905_setTXAddress(TxAddr);
      nRF905_setData(DataToSend, length);
      while(!nRF905_send());
}
```