Objective

To design an automatic power factor improvement system.

Introduction

In electrical engineering, the power factor of an AC electrical power system is defined as the ratio of the real power absorbed by the load to the apparent power flowing in the circuit. Here real power means the actual power consumed by the equipment to do useful work. It is measured in watts. The apparent power is the product of the RMS voltage and the RMS current. It is measured in kVA. And the reactive power is the resultant power of an AC circuit when the current waveform is out of phase with the waveform of the voltage. It is measured in kVAR. Real power is distinguished from apparent power by eliminating the reactive power component that may be present. These three types of power can be represented in a triangular form which is named "Power Triangle". A generalized power triangle is shown in figure 1.

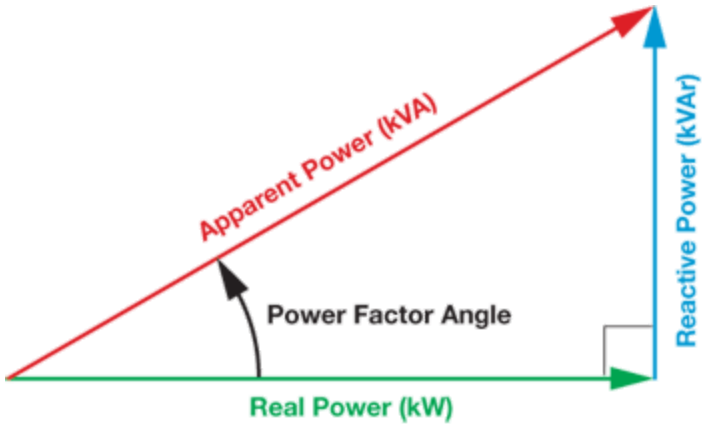


Figure: 1

If the power factor angle is denoted as $\boldsymbol{\theta}$, then we can say that the power factor is equal to cos. Here $\boldsymbol{\theta}$ can also be defined as the difference of phase angle of the voltage and current of the system. When the impedance of the system is purely resistive, the apparent power is the same as the real power and there is no phase difference between voltage and current. In that case, the power factor becomes 1.

This situation is called the “unity power factor”. But when reactance exists, the apparent power is greater than the true power and a phase difference is created between voltage and current. If the load is of capacitive type then the current will lead the voltage. And for inductive loads current lags the voltage. Thus the power factor value falls below 1. The maximum value of the power factor can be 1. It is always desired to keep the power factor at unity. Because low power factor creates some problems. Firstly low power factor means high reactive power. Reactive power is the unused power that is pushed forth and back. We can not use this power to do useful work. Besides reactive power causes an unwanted increase in the current of the transmission line. Consequently, losses on AC transmission lines increase. This can be understood easily from the following mathematical equations:

We know that the power of a specific system can be expressed as

$$P = VI\cos\theta$$

Now the current I can be shown as

$$I = \frac{P}{V\cos\theta}$$

For a specific system power (P) and supply voltage (V) is constant. So now if $\cos\theta$ i.e. power factor decreases, then the current increases. As a result loss(P_L) in the transmission line increases according to the following relationship of power loss and current:

$$P_L = I^2R$$

So we can say that if the power factor of a system is not unity, there will be huge loss in transmission. For this reason, every system needs to have a mechanism that will make sure that the power factor is close to unity. Even if the power factor decreases, the mechanism will take measures to correct the power factor and take its value near unity. This correction can be done in many ways. Usually, most of the loads that we use are inductive. Inductive loads create a lagging power factor as the current lags the voltage. In this case, capacitors are connected in parallel to

correct the power factor. This is done because the current leads the voltage for capacitors and thus the lagging current for inductive loads is compensated. In most cases, capacitor banks, which can provide different values of capacitance, are used to correct the power factor of different inductive loads. In our project, we have also used this method of using a capacitor bank to improve the power factor of a system that has inductive loads connected with it. We used "Arduino Uno" to measure the power factor and control the capacitor bank. This project mainly focuses on software simulation. The project has two parts. The first part is the circuit part and the second part is the coding part. The circuit was constructed in Proteus 7 and coding was done in Arduino IDE 1.8.10. Both of these parts are discussed in detail in the following parts.

Circuit in Proteus

Figure 2 shows the full circuit of our project. This circuit is divided into few parts for the ease of explanation. The parts are:

- ➢ Source
- ➢ Transformers
- ➢ Op-Amps / Zero Detection Circuit
- ➢ XOR Gate
- ➢ Arduino
- ➢ LCD
- ➢ Load
- ➢ Capacitor Bank

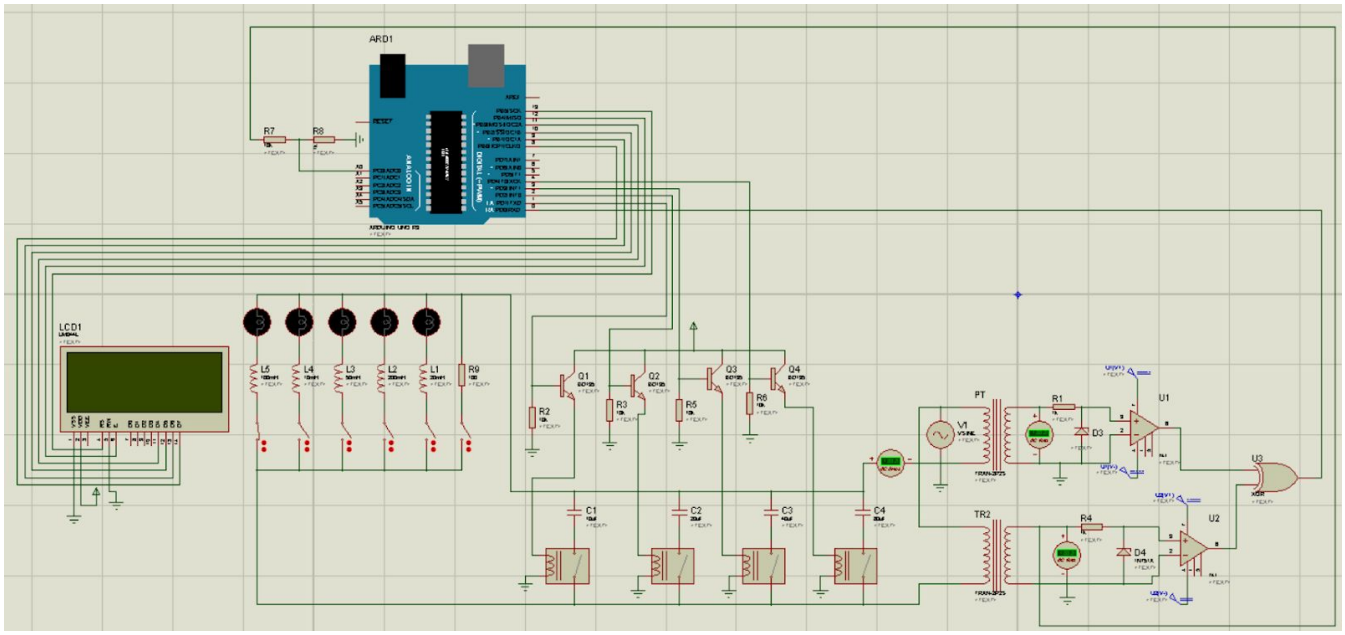


Figure: 2

Source: An AC sinusoidal source of 312V and 50Hz frequency is used. The RMS value of the source is approximately 220V.

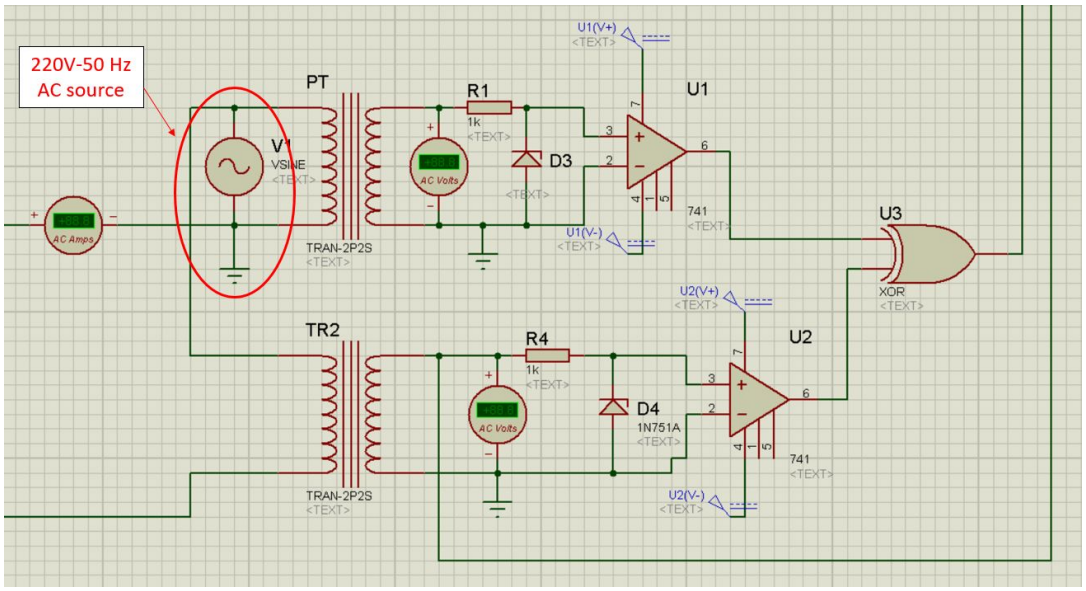


Figure-3: Source, PT & CT, Zero Detection Circuit and XOR gate

Transformers: Two step-down transformers are used to step down the high voltage of 220V to a lower voltage of approximately 7V to measure voltage and current. This stepping down is needed because the voltage and the current both are fed to two Op-Amps and the voltage required for the operation of the Op-Amps is very low (5V). Two transformers are called Potential Transformer (PT) and Current Transformer (CT). The PT is connected in parallel with the source and it measures the voltage. To determine the ratio of the transformer to convert 220V into 7V, we changed the inductance value of the primary and secondary coils of the transformers. For PT, the primary inductance is set to 0.9H and secondary inductance is 0.001H. This is done using the trial and error method.

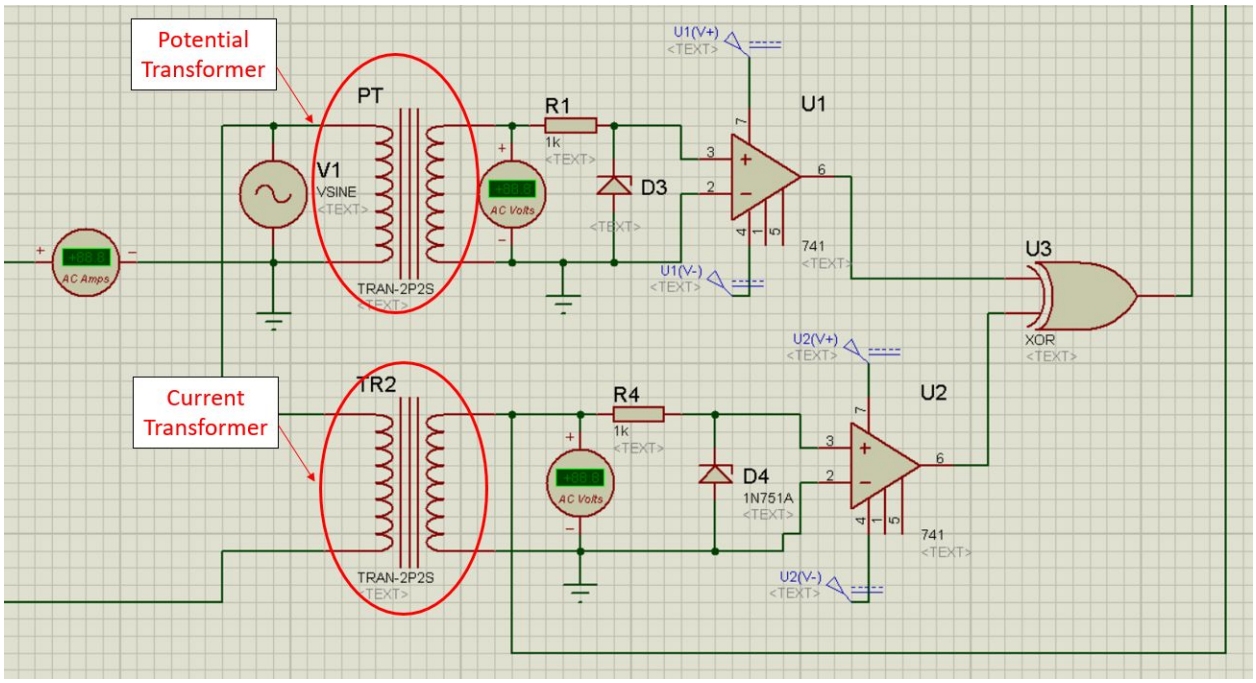
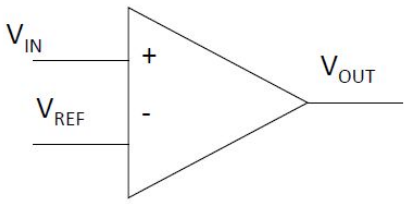


Figure-4: Potential & Current Transformers

The current is measured using the CT. This transformer is connected with the circuit in series. Thus current flowing through the transformer will induce a voltage in the primary coil and eventually in the secondary coil. The value of the current is very low. So we kept the transformer ratio as 1 by setting the inductance value of both the coils to 0.001H. The outputs of the secondary side of both PT and CT are fed to two "Zero Detection Circuits".

Zero Detection Circuit: A "Zero Detection Circuit" is used for detecting the zero-crossing point of any signal. This circuit is made using Op-Amp. We used the LM741 model Op-Amp in our project. In our project, we used two Zero Detection Circuits. One is for the voltage and the other is for the current. In both the Zero Detection Circuit, Op-Amps are used in comparator mode.



If	Output
$V_{IN} > V_{REF}$	High
$V_{IN} < V_{REF}$	Low

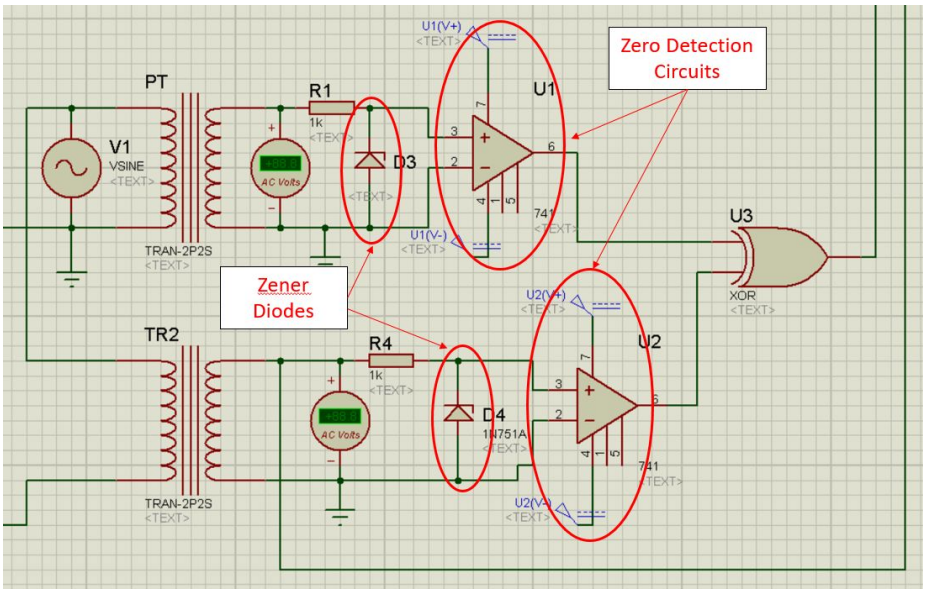


Figure-5: Zero Detection Circuits

The inverting terminals are grounded. So whenever the voltages on the non-inverting terminals are greater than zero, the Op-Amps will give 5V in the output terminal. And when the voltages are below zero i.e in the negative half cycle, the outputs of the Op-Amps will be zero. Thus the Op-Amps could detect zero crossings of both voltage and current. Now the Op-Amps will give us a square pulse of 5V whenever the voltage and current will be in the positive half cycle. If

there is a phase difference between current and voltage, the square pulses will not start at the same time. The phase angle can be determined from the delay between starting of the square pulses of current and voltage. To measure this delay, square pulses of the voltage and the current are passed through an XOR gate.

Two Zener diodes, which have a reverse breakdown voltage of 5V, are connected across both of the Op-Amps. This is done to limit the voltage level at 5V across the Op-Amps. When the voltage is above 5V, the Zener diode works as a voltage regulator and provides a constant voltage of 5V. Voltage for Op-Amps is limited to 5V because the Arduino works with 5V.

XOR Gate:

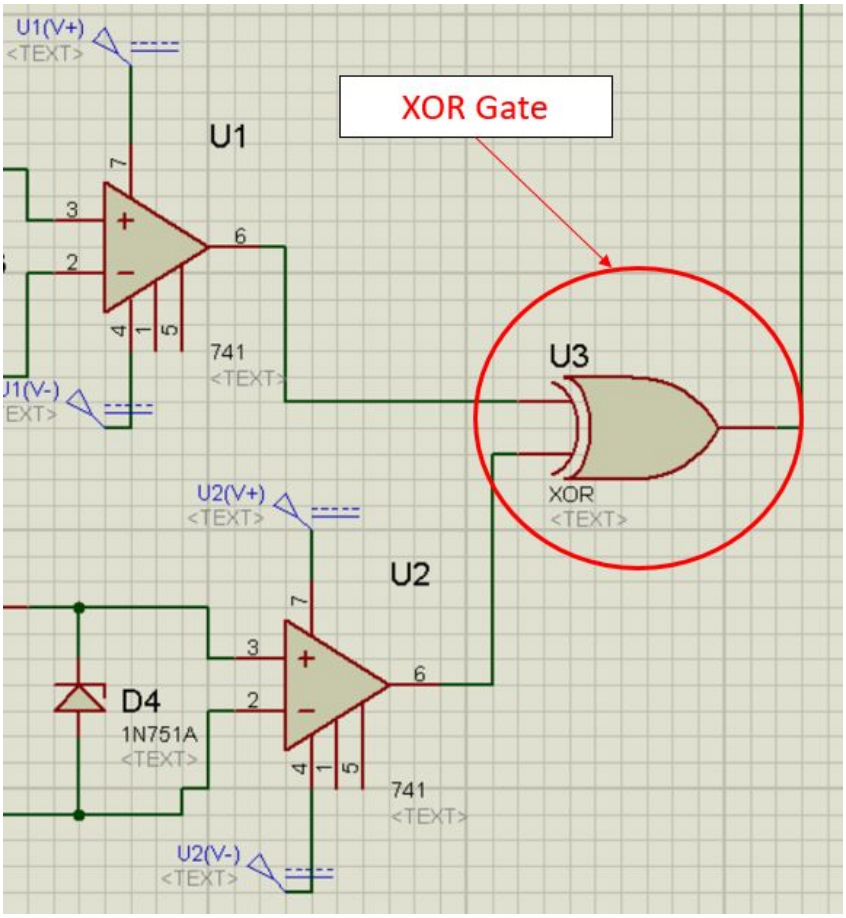
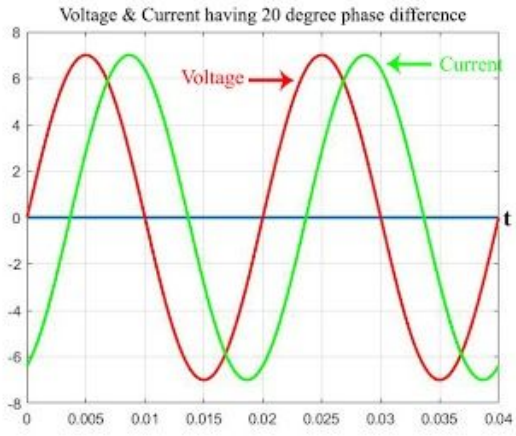


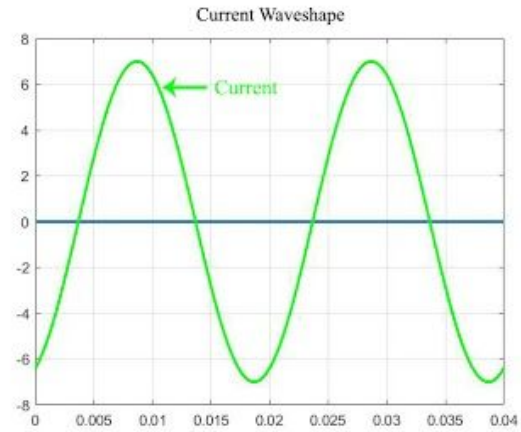
Figure-6: XOR gate

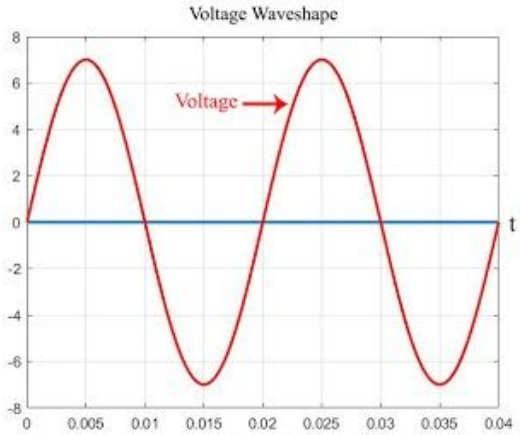
Output terminals of the Op-Amps are connected to the two terminals of an XOR gate. The working principle of an XOR gate can be understood from the truth table given below.

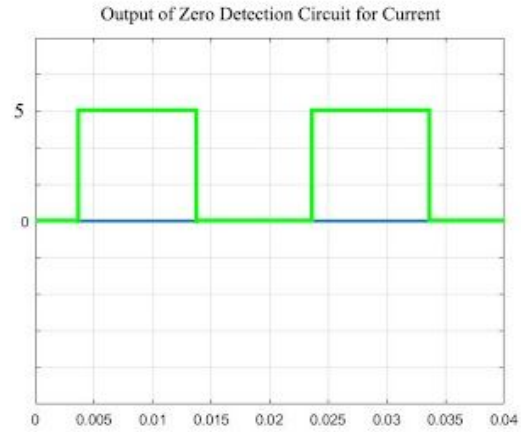
Truth Table of XOR Gate		
Input 1	**Input 2**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

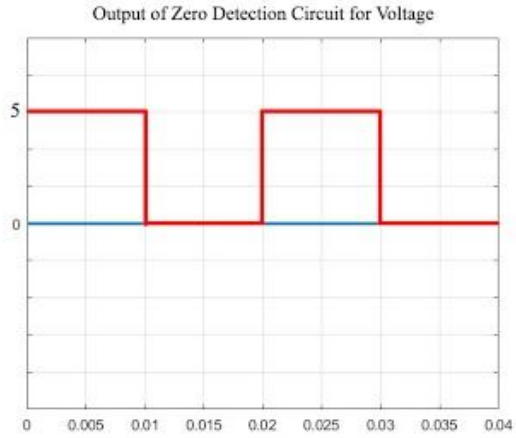
When the outputs of both the Op-Amps are zero, XOR inputs are both zero. Thus XOR output is zero. Whenever voltage or current any one of the two is above zero, the output of the corresponding Op-Amp is 5V i.e one of the XOR inputs is 5V. Thus XOR output is 5V. Then when the other signal becomes more than zero, the other Op-Amp starts giving 5V to the other input of the XOR which makes the XOR output zero. Thus for the phase difference part, XOR gives a square pulse of 5V.

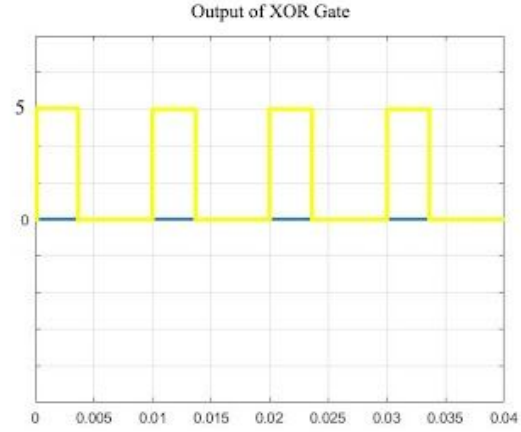












<u>Arduino:</u> In this project, we used an "Arduino Uno" to measure the phase difference between current and voltage, to measure the power factor from the phase angle, to display the power factor value and to decide which combination of the capacitors should be used to improve the power factor if it falls below a preset

value. To do all these things, the Arduino needs to collect data from the circuit and send data to the LCD and capacitor banks.

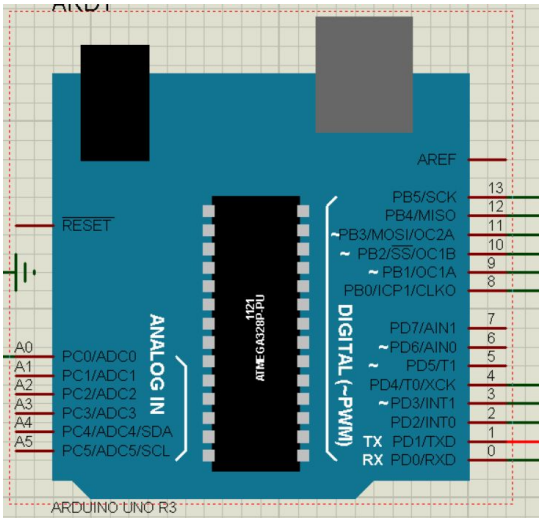


Figure-7: Arduino

The measuring and decision-making procedures will be discussed in the coding part. Here we will explain the hardware connections only. Connection of different pins of the Arduino is as follows:

➔ **Pin 0:** The output of the XOR gate is connected to this pin. This pin measures the duration of the output pulse of the XOR gate and then converts it to phase angle.

➔ **Pin 1-4:** These 4 pins are connected to the base of 4 transistors. These pins are used to turn on the capacitors.

➔ **Pin 8-13:** These pins are connected to the LCD and are used to control the LCD.

➔ **Pin A0:** This pin is used to measure the current in the circuit. Pin A0 is an analog input pin. It can measure values starting from 0 to 1023. The voltage from the CT is divided using a voltage divider and then passed to the pin A0. If there's any change in the current, the voltage will change. Thus the change in current can be detected using this pin. This pin is mainly used to sense the open circuit situation. Whenever the circuit has no load, there will be no current flow. Thus voltage in pin A0 will be zero. Thus the Arduino can know about the open circuit situation.

LCD:

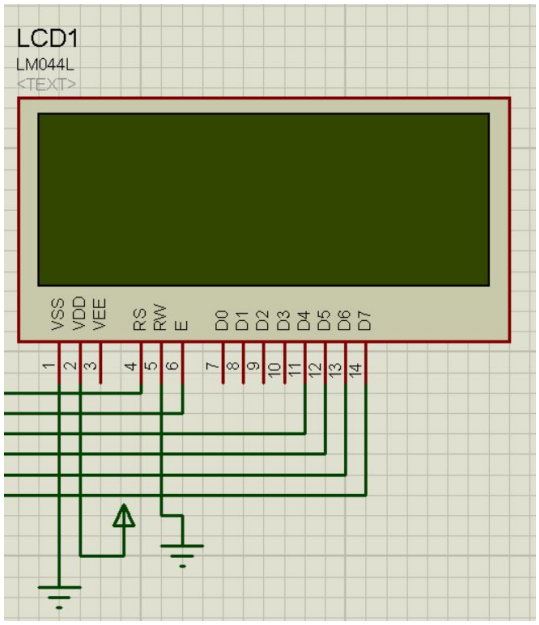


Figure-8: LCD

A 20x4 LCD is connected to the Arduino. It is used to display the Phase angle, Power factor, the combination of the capacitor bank used. The RS and RW pins of the LCD are connected to the pins 13 and 12 of the Arduino. To send data to LCD from Arduino, only 4 pins are needed. So the pins D4-D7 of the LCD are connected to the pins 11-8 of the Arduino. The VDD pin of the LCD is provided with 5V and the VSS and RW pins are grounded. The RW pin is grounded because we will be only writing data to the LCD which requires RW to be 0. As we will not read data from LCD, RW doesn't need to be 1 and thus it is grounded.

Load:

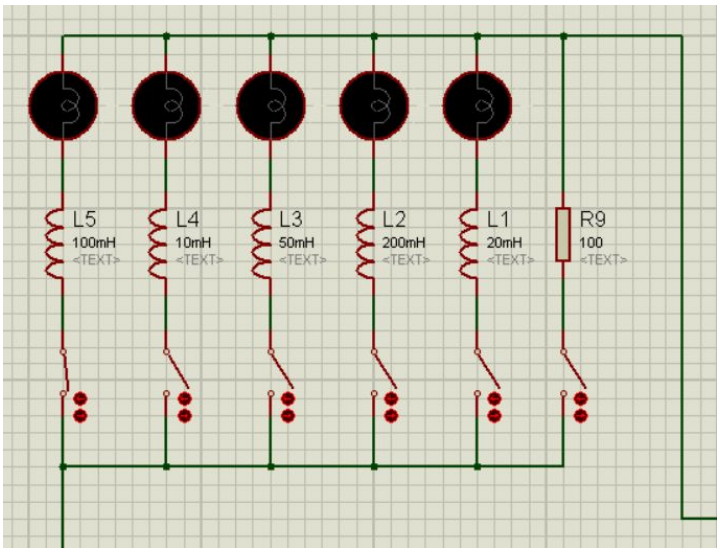


Figure-9: Load

In our project, we connected five lamps as loads. Each of these lamps have an inductor connected in series with it. The values of these inductances are not equal. They are deliberately made different so that we can have different combinations of load. All these loads are connected in parallel and have a switch that will determine whether the load is connected or not. Using the switches we will choose our preferred load combination. As calculating equivalent inductances follows the same rule as resistances, adding inductances in parallel will decrease the value of the total load inductance.

Capacitor Bank: Four capacitors are connected with the loads for power factor correction. The capacitors are operated with relays which are energized by the transistors. All the collectors of the transistors are connected to a 5V supply and the emitters are connected to the four relays. The base of the transistors (Q1, Q2, Q3, Q4) are connected to the pins D1, D2, D3, D4 of the Arduino. Whenever the Arduino provides 5V to any of the pins, the base of the transistor connected to that corresponding pin gets 5V. Thus the emitter and collector of that transistor gets shorted and 5V is passed to the relay connected to that transistor. As a result, the

relay gets energized and connects the capacitor corresponding to that relay to the circuit. In this way, all four capacitors are controlled.

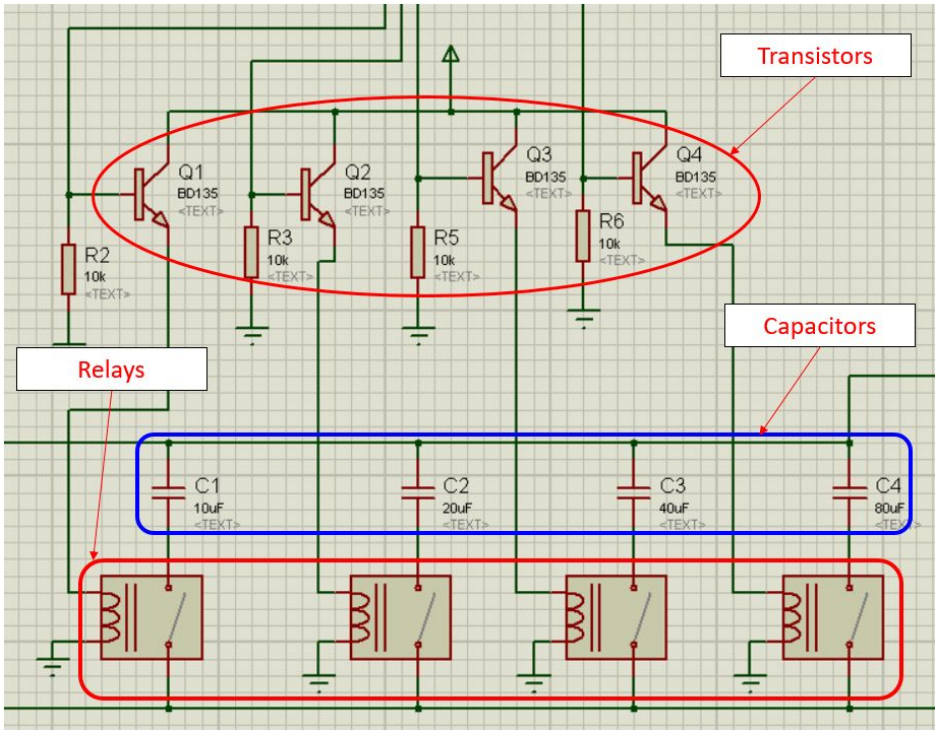


Figure-10: Capacitor Bank

The capacitor values are chosen as C1=10uF, C2=20uF, C3=40uF, C4=80uF. These values are chosen to get the advantage of increasing the capacitance value by 10uF in each step. When there is no need for power factor correction, the ports D4, D3, D2, D1 are set as 0000 and 0V are provided to the base of the transistors. The transistor remains open-circuited and no current flows through the transistor to the relay. So the relay doesn't energize and the capacitors are disconnected. For D4 D3 D2 D1 = 0001, the 10uF capacitor gets connected in parallel with the circuit. When D4 D3 D2 D1 = 0010, the 20uF capacitor gets connected in parallel with the circuit. And for D4 D3 D2 D1 = 0011, the 10uF and 20uF capacitors get connected in parallel with the circuit. Thus the equivalent capacitance becomes 30uF. In this way, there are 16 different combinations starting from 0000 to 1111. There 16 combinations will provide capacitance starting from 10uF to 150uF with a step size of 10uF.

Arduino Code

The full code of Arduino is as follows:

```
void Measure();
void Correction(int x);
void Print();
void OC_Print();
void CB_Print();

#include<LiquidCrystal.h>
LiquidCrystal LCD(13,12,11,10,9,8);

float PF, Angle;
int dur = 0;
int Case = 0;
float IL,IL1;

int R1 = 1;
int R2 = 2;
int R3 = 3;
int R4 = 4;
int ReadDur = 0;

void loop() {

  Case=0;
  Correction(Case);
  Measure();
  delay(500);

  while(IL<1 && IL1<1)
 {
  LCD.clear();
  OC_Print();
  delay(300);
  Measure();
  delay(500);
  }

  Case=0;
  Correction(Case);
  Measure();
  delay(500);

  LCD.clear();
  Print();
  CB_Print();

while(PF<0.95)
    {
     LCD.setCursor(0,2);
     LCD.print("Correcting PF");
     Case++;
     if (Case>15)
     {Case=0;
     break;}
     Correction(Case);
     delay(100);
     Measure();
     delay(500);
     Print();
     CB_Print();
     if(Angle>20 || PF>0.95)
     break;
     delay(400);
    }

    delay(400);

    LCD.setCursor(0,2);
    LCD.print("                    ");
    LCD.setCursor(0,2);
    LCD.print("WAIT");
    delay(1000);
    Print();
    CB_Print();
```

```
    while(1)
    {
      if(PF<0.95)
      {
        break;
      }

      LCD.setCursor(0,2);
      LCD.print("                    ");
      LCD.setCursor(0,2);
      LCD.print("CORRECTION DONE!");

      delay(700);
      Measure();
      Print();
      CB_Print();
      }

      delay(500);


}

void Measure()
{
  dur = pulseIn(ReadDur, HIGH);
  Angle = dur * 0.018;
  Angle = Angle-90;
  PF = cos(Angle*0.0174533);
 IL= analogRead(A0);
 delay(500);
 IL1= analogRead(A0);
}

void Print()
{
  LCD.setCursor(0,3);
  LCD.print("                    ");
  LCD.setCursor(0,0);
  LCD.print("Angle = ");
  LCD.print(Angle);
  LCD.setCursor(0,1);
  LCD.print("PF = ");
  LCD.print(PF);

}
```

```
void CB_Print()
{
  LCD.setCursor(0,3);
  LCD.print("CB Combination=");
  LCD.print(Case);
}

  void OC_Print()
{
  LCD.clear();
  LCD.setCursor(0,1);  //setCursor(x,y)
  LCD.print("       NO LOAD");
  LCD.setCursor(0,2);
  LCD.print("      CONNECTED");
  }



void Correction(int x)
{
  if(x==1)
  {
    digitalWrite(1, HIGH);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
  }

  else if(x==2)
  {
  digitalWrite(1, LOW);
  digitalWrite(2, HIGH);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);

  }

  else if(x==3)
  {
     digitalWrite(R1, HIGH);
    digitalWrite(R2, HIGH);
    digitalWrite(R3, LOW);
    digitalWrite(R4, LOW);
  }

  else if(x==4)
  {
     digitalWrite(R1, LOW);
    digitalWrite(R2, LOW);
   digitalWrite(R3, HIGH);
    digitalWrite(R4, LOW);
  }
```

```
else if(x==5)
{
  digitalWrite(R1, HIGH);
 digitalWrite(R2, LOW);
  digitalWrite(R3, HIGH);
 digitalWrite(R4, LOW);
}

else if(x==6)
{
  digitalWrite(R1, LOW);
 digitalWrite(R2, HIGH);
  digitalWrite(R3, HIGH);
  digitalWrite(R4, LOW);
}

else if(x==7)
{
  digitalWrite(R1, HIGH);
  digitalWrite(R2, HIGH);
  digitalWrite(R3, HIGH);
  digitalWrite(R4, LOW);
}

else if(x==8)
{
  digitalWrite(R1, LOW);
  digitalWrite(R2, LOW);
 digitalWrite(R3, LOW);
  digitalWrite(R4, HIGH);
}
else if(x==9)
{
  digitalWrite(R1, HIGH);
  digitalWrite(R2, LOW);
  digitalWrite(R3, LOW);
  digitalWrite(R4, HIGH);
}
else if(x==10)
{
  digitalWrite(R1, LOW);
  digitalWrite(R2, HIGH);
  digitalWrite(R3, LOW);
  digitalWrite(R4, HIGH);
}
else if(x==11)
{
  digitalWrite(R1, HIGH);
  digitalWrite(R2, HIGH);
  digitalWrite(R3, LOW);
  digitalWrite(R4, HIGH);
}
  else if(x==12)
  {
    digitalWrite(R1, LOW);
    digitalWrite(R2, LOW);
    digitalWrite(R3, HIGH);
    digitalWrite(R4, HIGH);
  }
  else if(x==13)
  {
    digitalWrite(R1, HIGH);
    digitalWrite(R2, LOW);
    digitalWrite(R3, HIGH);
    digitalWrite(R4, HIGH);
  }
  else if(x==14)
  {
    digitalWrite(R1, LOW);
    digitalWrite(R2, HIGH);
    digitalWrite(R3, HIGH);
    digitalWrite(R4, HIGH);
  }
  else if(x==15)
  {
    digitalWrite(R1, HIGH);
    digitalWrite(R2, HIGH);
    digitalWrite(R3, HIGH);
    digitalWrite(R4, HIGH);
  }

  else
  {
    digitalWrite(R1, LOW);
    digitalWrite(R2, LOW);
    digitalWrite(R3, LOW);
    digitalWrite(R4, LOW);
  }
}
```

Now different parts of the code are explained one by one.

```
void Measure();
void Correction(int x);
void Print();
void OC_Print();
void CB_Print();
```

At the beginning of our code, we are doing 'Function Prototyping'. Function prototyping means declaring any user-defined function at the beginning of the program. Inside the main program when we will call these functions then that particular task, for which these functions are made, will be done.

Here 'Measure' will measure some parameters that we have discussed below in detail. When we will call 'Correction' function then it will try to correct the power factor using the capacitor bank. The input will be given in decimal form from 0 to 15 and the corresponding binary number will be used to turn on the capacitors of the capacitor bank. E.g if we give 2 means 0010 which will turn on the 20uF capacitance. 'Print' will print on display. 'OC_Print' will print 'No Load Connected' in display in case of an open-circuit case. And using 'CB_Print' we will show which capacitor combination we are using.

```
#include<LiquidCrystal.h>
LiquidCrystal LCD(13,12,11,10,9,8);
```

Here we are including LCD's library here. After that, we are defining an object name 'LCD'. By calling LCD we will access the LCD. Then we are writing which pins of Arduino we are using for accessing LCD.

After that, we are defining the data types of different variables. Here time duration is defined by ‘dur’. From ‘0’ pin we will get the time duration. And initially, we are setting ‘case’ = 0 means we don’t need any correction. IL and IL1 will calculate current to find if it is an open circuit or not.

```
float PF, Angle;
int dur = 0;
int Case = 0;
float IL,IL1;

int R1 = 1;
int R2 = 2;
int R3 = 3;
int R4 = 4;
int ReadDur = 0;
```

Here R1, R2, R3, R4 are four capacitors of the capacitor bank. Then for accessing capacitor bank we are defining here that pin 1 is for R1, pin 2 is for R2, pin 3 is for R3, and pin 4 is for R4. By ‘ReadDur’ we are reading the value of time duration from pin 0.

Void setup ( )

```
void setup() {

  LCD.begin(20, 4);

  pinMode(ReadDur, INPUT);
  pinMode(A0,INPUT);
  pinMode(R1, OUTPUT);
  pinMode(R2, OUTPUT);
  pinMode(R3, OUTPUT);
  pinMode(R4, OUTPUT);

  digitalWrite(R1, LOW);
  digitalWrite(R2, LOW);
  digitalWrite(R3, LOW);
  digitalWrite(R4, LOW);

}
```

We are initializing here the 20 characters and 4 rows of our LCD first. Then by using pinMode we are defining our particular Arduino pins as input or output. As

we don’t know the initial condition, we will not use any capacitor. So we are setting the pins 1-4 as LOW which will disconnect all the capacitors.

Void measure ( )

```
void Measure()
{
  dur = pulseIn(ReadDur, HIGH);
  Angle = dur * 0.018; //time to angle (in degree) convert
  Angle = Angle-90; // (-90) is for offseting the angle
  PF = cos(Angle*0.0174533); //angle conversion from degree to radian, then PF calcultion
 IL= analogRead(A0);
 delay(500);
 IL1= analogRead(A0);
}
```

In this subroutine, we are using a variable named ‘dur’. As long as we will get a High pulse in the ‘Readdur’ pin which is pin 0, it will count the time and will save it in the ‘dur’ variable. Then we will convert it into angle. As we are getting 90 degrees for the pure resistive load due to the transformer inductance, so to compensate that we are subtracting 90 from angle. Then we are converting the angle from degree to radian as Arduino does calculations of trigonometric functions in radian. Then we will measure the current value in the A0 analog pin. For accuracy, we are taking values twice giving a 500ms time delay.

Void CB_Print ( )

```
void CB_Print()
{
  LCD.setCursor(0,3);
  LCD.print("CB Combination=");
  LCD.print(Case);
}
```

Going to the last line we are printing here the capacitor bank combination which will be obtained from the ‘case’.

Void Print ( )

```
void Print()
{
  LCD.setCursor(0,3);
  LCD.print("                    ");
  LCD.setCursor(0,0);  //setCursor(x,y)     x=column  y=row
  LCD.print("Angle = ");
  LCD.print(Angle);
  LCD.setCursor(0,1);
  LCD.print("PF = ");
  LCD.print(PF);


}
```

First, we are setting our cursor at the 4th (last) line of LCD and will print spaces means it will clear the last line. Then it will go to first-line and will print ‘Angle’ and then it will print the value of the angle. Then it will go to the next line and will print PF and its value.

Void OC_Print ( )

```
  void OC_Print()
{
  LCD.clear();
  LCD.setCursor(0,1);  //setCursor(x,y)     x=column  y=row
  LCD.print("       NO LOAD");
  LCD.setCursor(0,2);
  LCD.print("      CONNECTED");
  }
```

Here we are printing ‘NO LOAD CONNECTED’ for open circuit case if there is no current flow.

Void correction (int x)

Here it will check the value of ‘x’. The value of ‘x’ will be determined from the value of ‘case’. If x = 1 then it will set “R4 R3 R2 R1” = 0001 which means pin 1 will become high. Thus the capacitor of 10uF will be connected. If x=2 then “R4 R3 R2 R1” will be 0010 and pin 2 will become high and the 20uF capacitor will be connected. In this way, it will continue up to 15. Connected capacitance value will always be 10 times the value of ‘x’ which is nothing but the value of ‘case’.

```
void Correction(int x)
{
  if(x==1)
  {
    digitalWrite(1, HIGH);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
  }

  else if(x==2)
  {
  digitalWrite(1, LOW);
  digitalWrite(2, HIGH);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);

  }
```

```
void loop() {


  while(IL<1 && IL1<1)
 {
  LCD.clear();
  OC_Print();
  delay(300);
  Measure();
  delay(500);
  }
```

The power factor correcting algorithm of our system is in this main loop “void loop( )”. This is an unconditional loop. Thus the tasks defined inside this loop will be repeated again and again.

First, we check whether our IL & IL1 both are less than 1 or not. IL and IL1 are nothing but the values representing the current in the circuit. If these values are less than 1 that means there is no current flow. So we clear the display and call the sub-function “OC_Print ( )” which will print that there is no load connected. Then we call “Measure” to update the IL and IL1 values. As long as the current is zero, the program will stay inside this while loop and keep printing “NO LOAD CONNECTED” and keep checking whether there is any current in the circuit or not. If there is any current then the values of IL and IL1 will go above 1 and the program will come out of the while loop.

```
Case=0;
Correction(Case);
Measure();
delay(500);


LCD.clear();
Print();
CB_Print();
```

Then we make case=0 and call the “Correction” function. This function will turn on the capacitors depending on the “case” value. Here “case” is made zero because we assumed that initially there is no need for correcting the power factor. Then we give a delay of 500ms. Throughout the whole program, we have used several delays. These delays are given so that the Arduino gets enough time to implement whatever it is supposed to implement.

After turning off all the capacitors we call “Measure” to update the angle and power factor values. Throughout the whole program, we will be calling this function to update these values frequently so that any change in the circuit at any

moment can be detected easily. After updating the values we clear the whole display and call the print functions to display the values.

Now we check whether the PF value if less than 0.95 or not.

```
while(PF<0.95)
    {
     LCD.setCursor(0,2);
     LCD.print("Correcting PF");
     Case++;
     if (Case>15)
     {Case=0;
     break;}
     Correction(Case);
     delay(100);
     Measure();
     delay(500);
     Print();
     CB_Print();
     if(Angle>20 || PF>0.95)
     break;
     delay(400);
    }
```

This loop starts executing when our measured PF is less than 0.95. In that case, we start correcting PF value by adding capacitors to the circuit. LCD displays that the correction process is ongoing. For correction, we increase the ‘case’ value by one and check whether it has gone above 15 or not. If the ‘case’ value is above 15, we break out of this loop as we don’t have any combination of capacitors for ‘case’ value greater than 15. If the ‘case’ value is less than 15, we call the ‘correction’ function and measure and print the PF for that combination of ‘case’. If PF goes above 0.95, this loop breaks. Otherwise, this loop continues until the PF goes above 0.95. Here we used another condition that will break the loop if the angle value goes beyond 20. For the loads we have used in our project, the angle value should remain negative if no correction is done. But during correction this might happen that we have connected more than necessary capacitance. In that case angle will become positive but PF will still remain less than 0.95. In such a situation if the loop is not broken, then the system will keep adding capacitors to correct the

PF. But adding more capacitors will make the angle even more positive and decrease the PF. So if angle goes beyond positive 20, the loop must be broken.

After coming out of this loop, we give a delay to let the system become stable and measure everything again and display the values.

```
delay(400);

LCD.setCursor(0,2);
LCD.print("                ");
LCD.setCursor(0,2);
LCD.print("WAIT");
delay(1000);
Print();
CB_Print();


while(1)
{
  if(PF<0.95)
  {
    break;
  }

  LCD.setCursor(0,2);
  LCD.print("                 ");
  LCD.setCursor(0,2);
  LCD.print("CORRECTION DONE!");

  delay(700);
  Measure();
  Print();
  CB_Print();
  }
```

This while loop has no initial condition. No matter what the program will enter this loop. When the loop starts executing we check if the PF is less than 0.95 or not. If yes then we break out of this loop. If not then proceed further in this loop. We then keep measuring the values and keep printing them. As long as PF stays above 0.95, it keeps printing “CORRECTION DONE!” and this loop doesn’t break. If the PF goes below 0.95 due to a change of load on the system, we break out of this loop.

After breaking out of this loop, the program starts from the beginning of the 'void loop ( )' and keeps correcting the PF until it goes above 0.95.

Problems Faced and Drawbacks

This project might look very simple. But we faced a lot of problems while doing this. The first problem that we faced was finding out the voltage and current waveshapes. Using the PT and CT isn't that easy. We used the trial and error method to find out the primary and secondary coil inductances of the transformers. It took a huge effort and time to find out the suitable values. Then we faced a problem in implementing the open-circuit situation. As the current kept fluctuating, it showed an open circuit when many loads were connected. Measuring the current twice decreased the frequency of this error. Yet this error occurs sometimes. Then we tried to show the short circuit situation. But we had to drop that part as too many errors were occurring. Then the correction algorithm didn't work at the beginning. We couldn't figure out what part of the code wasn't correct. After spending a huge amount of time we found out that a lot of delays are needed. Otherwise, the algorithm simply didn't work. So the same code started working after putting some delays.

Future Prospects

In this project, we used 4 capacitors to provide capacitance of 10uF to 150uF with a step size of 10uF i.e we got 16 combinations. We still have 3 pins of Arduino unused. Using each extra pin to include one more capacitor will multiply the combination by 2. That means if we use 5 pins, the combination will be 32 and for 6 pins it will be 64. Thus we can have a huge range of capacitance with low step sizes. And if we use Arduino Mega instead of Arduino UNO, then we will have more pins. So we can then set the standard PF value more than 0.95.

Discussion

Our project has 5 loads of different inductance values and they can be connected in parallel. We have tested all the possible combinations of these loads and checked whether the system can improve the power factor to 0.95 or not. What we observed, no matter what the load combination is, the system corrected the power factor whenever it was below 0.95. And for all the cases, the system was able to detect the change in load almost instantly. Thus we can say that the objective of this project to correct the power factor is achieved.