LAMPIRAN
PROGRAM

```
#include <LiquidCrystal.h>
#include "EmonLib.h"                   // Include Emon Library

#define led1  10
#define led2  11
#define led3  12

  EnergyMonitor phase1;                     // Create an instance
  EnergyMonitor phase2;
  EnergyMonitor phase3;

  LiquidCrystal lcd (2,3,4,5,6,8,9);

float Irms1,Irms2,Irms3;
float V1,V2,V3;
float P1,P2,P3;
float VA1,VA2,VA3;
float CP1,CP2,CP3;

unsigned long currentMillis;
long previousMillis = 0;

void setup() {
  Serial.begin(57600);
  lcd.begin(20,4);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  phase3.current(5, 60);//111.1);
  phase2.current(4, 60);//111.1);
  phase1.current(3, 60);//111.1);

  phase1.voltage(2,198.5, 1.2);
  phase2.voltage(1,198.5, 1.2);
  phase3.voltage(0,198.5, 1.2);
}

void sendData(){
  Serial.print(Irms1);
  Serial.print(',');
  Serial.print(V1);//(Veff[2]);
  Serial.print(',');
  Serial.print(P1);
  Serial.print(',');
  Serial.print(VA1);
  Serial.print(',');
  Serial.print(CP1);
  Serial.print(',');

  Serial.print(Irms2);
  Serial.print(',');
  Serial.print(V2);//(Veff[1]);
  Serial.print(',');
  Serial.print(P2);
  Serial.print(',');
  Serial.print(VA2);
  Serial.print(',');
```

```
  Serial.print(CP2);
  Serial.print(',');

  Serial.print(Irms3);
  Serial.print(',');
  Serial.print(V3);//(Veff[0]);
  Serial.print(',');
  Serial.print(P3);
  Serial.print(',');
  Serial.print(VA3);
  Serial.print(',');
  Serial.print(CP3);
  Serial.print('\n');
}


void loop() {
  phase1.calcVI(20,100);
  phase2.calcVI(20,100);
  phase3.calcVI(20,100);
  Irms1=phase1.Irms;
  Irms2=phase2.Irms;
  Irms3=phase3.Irms;

  V1=phase1.Voltage;
  V2=phase2.Voltage;
  V3=phase3.Voltage;

  P1=phase1.realPower;
  P2=phase2.realPower;
  P3=phase3.realPower;

  VA1=phase1.apparentPower;
  VA2=phase2.apparentPower;
  VA3=phase3.apparentPower;

  CP1=phase1.powerFactor;
  CP2=phase2.powerFactor;
  CP3=phase3.powerFactor;

  lcd.setCursor(3,0);
  lcd.print("Energy Monitor");

  lcd.setCursor(0,1);lcd.print("I1:");
  lcd.setCursor(4,1);lcd.print(Irms1);
  lcd.setCursor(9,1);lcd.print("V1:");
  lcd.setCursor(12,1);lcd.print(V1);

  lcd.setCursor(0,2);lcd.print("I2:");
  lcd.setCursor(4,2);lcd.print(Irms2);
  lcd.setCursor(9,2);lcd.print("V2:");
  lcd.setCursor(12,2);lcd.print(V2);

  lcd.setCursor(0,3);lcd.print("I3:");
  lcd.setCursor(4,3);lcd.print(Irms3);
  lcd.setCursor(9,3);lcd.print("V3:");
  lcd.setCursor(12,3);lcd.print(V3);
```

```
  if(V1>=10 || Irms1 >=1){
    digitalWrite(led1,HIGH);
  }
  else{
    digitalWrite(led1,LOW);
  }
  if(V2>=10 || Irms2 >=1){
    digitalWrite(led2,HIGH);
  }
  else{
    digitalWrite(led2,LOW);
  }

  if(V3>=10 || Irms3 >=1){
    digitalWrite(led3,HIGH);
  }
  else{
    digitalWrite(led3,LOW);
  }
  currentMillis = millis();
  if(currentMillis - previousMillis > 10)
  {
    sendData();
    previousMillis = currentMillis;
  }

}
```

```
/*
  Emon.cpp - Library for openenergymonitor
  Created by Trystan Lea, April 27 2010
  GNU GPL
  modified to use up to 12 bits ADC resolution (ex. Arduino Due)
  by boredman@boredomprojects.net 26.12.2013
  Low Pass filter for offset removal replaces HP filter 1/1/2015 -
RW
*/

//#include "WProgram.h" un-comment for use on older versions of
Arduino IDE
#include "EmonLib.h"

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


//--------------------------------------------------------------------
--------------------
// Sets the pins to be used for voltage and current sensors
//--------------------------------------------------------------------
--------------------
void EnergyMonitor::voltage(unsigned int _inPinV, double _VCAL,
double _PHASECAL)
{
   inPinV = _inPinV;
   VCAL = _VCAL;
   PHASECAL = _PHASECAL;
   offsetV = ADC_COUNTS>>1;
}

void EnergyMonitor::current(unsigned int _inPinI, double _ICAL)
{
   inPinI = _inPinI;
   ICAL = _ICAL;
   offsetI = ADC_COUNTS>>1;
}

//--------------------------------------------------------------------
--------------------
// Sets the pins to be used for voltage and current sensors based on
emontx pin map
//--------------------------------------------------------------------
--------------------
void EnergyMonitor::voltageTX(double _VCAL, double _PHASECAL)
{
   inPinV = 2;
   VCAL = _VCAL;
   PHASECAL = _PHASECAL;
   offsetV = ADC_COUNTS>>1;
```

```
}

void EnergyMonitor::currentTX(unsigned int _channel, double _ICAL)
{
   if (_channel == 1) inPinI = 3;
   if (_channel == 2) inPinI = 0;
   if (_channel == 3) inPinI = 1;
   ICAL = _ICAL;
   offsetI = ADC_COUNTS>>1;
}

//-------------------------------------------------------------------
--------------------
// emon_calc procedure
// Calculates realPower,apparentPower,powerFactor,Vrms,Irms,kWh
increment
// From a sample window of the mains AC voltage and current.
// The Sample window length is defined by the number of half
wavelengths or crossings we choose to measure.
//-------------------------------------------------------------------
--------------------
void EnergyMonitor::calcVI(unsigned int crossings, unsigned int
timeout)
{
   #if defined emonTxV3
      int SupplyVoltage=3300;
   #else
      int SupplyVoltage = readVcc();
   #endif

  unsigned int crossCount = 0;                             //Used to
measure number of times threshold is crossed.
  unsigned int numberOfSamples = 0;                        //This is
now incremented

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

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

  while(st==false)                                        //the while
loop...
  {
     startV = analogRead(inPinV);                         //using the
voltage waveform
     if ((startV < (ADC_COUNTS*0.55)) && (startV >
(ADC_COUNTS*0.45))) st=true;  //check its within range
     if ((millis()-start)>timeout) st = true;
  }
```

```
  //---------------------------------------------------------------
---------------------------------------------------
  // 2) Main measurement loop
  //---------------------------------------------------------------
---------------------------------------------------
  start = millis();

  while ((crossCount < crossings) && ((millis()-start)<timeout))
  {
    numberOfSamples++;                       //Count number of times
looped.
    lastFilteredV = filteredV;               //Used for delay/phase
compensation

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

    //---------------------------------------------------------------
---------------
    // B) Apply digital low pass filters to extract the 2.5 V or
1.65 V dc offset,
    //     then subtract this - signal is now centred on 0 counts.
    //---------------------------------------------------------------
---------------
    offsetV = offsetV + ((sampleV-offsetV)/1024);
      filteredV = sampleV - offsetV;
    offsetI = offsetI + ((sampleI-offsetI)/1024);
      filteredI = sampleI - offsetI;

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

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

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

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

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

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

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

  double V_RATIO = VCAL *((SupplyVoltage/1000.0) / (ADC_COUNTS));
  Vrms = V_RATIO * sqrt(sumV / numberOfSamples);

  double I_RATIO = ICAL *((SupplyVoltage/1000.0) / (ADC_COUNTS));
  Irms = I_RATIO * sqrt(sumI / numberOfSamples);
  //Calculation power values
  realPower = V_RATIO * I_RATIO * sumP / numberOfSamples;
  apparentPower = Vrms * Irms;
  powerFactor=(realPower / apparentPower)+0.06;
  Voltage=((Vrms*0.9973)-16.481);
     if(Irms<=0.20){
          realPower=apparentPower=powerFactor=0;
     }
```

```
      if(Voltage<0){
            Voltage=0;
      }
//    if(realPower<=0.50){
//          realPower= 0.00;
//    }
  //Reset accumulators
  sumV = 0;
  sumI = 0;
  sumP = 0;
//-------------------------------------------------------------------
--------------------
}

//-------------------------------------------------------------------
--------------------
double EnergyMonitor::calcIrms(unsigned int Number_of_Samples)
{

   #if defined emonTxV3
     int SupplyVoltage=3300;
   #else
     int SupplyVoltage = readVcc();
   #endif


  for (unsigned int n = 0; n < Number_of_Samples; n++)
  {
    sampleI = analogRead(inPinI);

    // Digital low pass filter extracts the 2.5 V or 1.65 V dc
offset,
     //  then subtract this - signal is now centered on 0 counts.
    offsetI = (offsetI + (sampleI-offsetI)/1024);
     filteredI = sampleI - offsetI;

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

  double I_RATIO = ICAL *((SupplyVoltage/1000.0) / (ADC_COUNTS));
  Irms = I_RATIO * sqrt(sumI / Number_of_Samples);

  //Reset accumulators
  sumI = 0;
//-------------------------------------------------------------------
--------------------

  return Irms;
}

void EnergyMonitor::serialprint()
```

```
{
//    Serial.print(realPower);
//    Serial.print(' ');
//    Serial.print(apparentPower);
//    Serial.print(' ');
    Serial.print(Voltage);
//    Serial.print(' ');
//    Serial.print(Irms);
//    Serial.print(' ');
//    Serial.print(powerFactor);
    Serial.println(' ');
    delay(100);
}

//thanks to http://hacking.majenko.co.uk/making-accurate-adc-
readings-on-arduino
//and JÃ©rÃ´me who alerted us to
http://provideyourown.com/2012/secret-arduino-voltmeter-measure-
battery-voltage/

long EnergyMonitor::readVcc() {
  long result;

  //not used on emonTx V3 - as Vcc is always 3.3V - eliminates
bandgap error and need for calibration
http://harizanov.com/2013/09/thoughts-on-avr-adc-accuracy/

  #if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328__) ||
defined (__AVR_ATmega328P__)
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  #elif defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__)
|| defined(__AVR_ATmega2560__) || defined(__AVR_AT90USB1286__)
  ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) |
_BV(MUX1);
  ADCSRB &= ~_BV(MUX5);   // Without this the function always
returns -1 on the ATmega2560
http://openenergymonitor.org/emon/node/2253#comment-11432
  #elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) ||
defined(__AVR_ATtiny84__)
  ADMUX = _BV(MUX5) | _BV(MUX0);
  #elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) ||
defined(__AVR_ATtiny85__)
  ADMUX = _BV(MUX3) | _BV(MUX2);

  #endif


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

```
  return result;
 #elif defined(__arm__)
  return (3300);                                    //Arduino Due
 #else
  return (3300);                                    //Guess that other
un-supported architectures will be running a 3.3V!
 #endif
}
```

```
/*
  Emon.h - Library for openenergymonitor
  Created by Trystan Lea, April 27 2010
  GNU GPL
  modified to use up to 12 bits ADC resolution (ex. Arduino Due)
  by boredman@boredomprojects.net 26.12.2013
  Low Pass filter for offset removal replaces HP filter 1/1/2015 -
RW
*/

#ifndef EmonLib_h
#define EmonLib_h

#if defined(ARDUINO) && ARDUINO >= 100

#include "Arduino.h"

#else

#include "WProgram.h"

#endif

// define theoretical vref calibration constant for use in readvcc()
// 1100mV*1024 ADC steps http://openenergymonitor.org/emon/node/1186
// override in your code with value for your specific AVR chip
// determined by procedure described under "Calibrating the internal
reference voltage" at
// http://openenergymonitor.org/emon/buildingblocks/calibration
#ifndef READVCC_CALIBRATION_CONST
#define READVCC_CALIBRATION_CONST 1126400L
#endif

// to enable 12-bit ADC resolution on Arduino Due,
// include the following line in main sketch inside setup()
function:
//  analogReadResolution(ADC_BITS);
// otherwise will default to 10 bits, as in regular Arduino-based
boards.
#if defined(__arm__)
#define ADC_BITS    12
#else
#define ADC_BITS    10
#endif

#define ADC_COUNTS  (1<<ADC_BITS)


class EnergyMonitor
{
  public:

    void voltage(unsigned int _inPinV, double _VCAL, double
_PHASECAL);
```

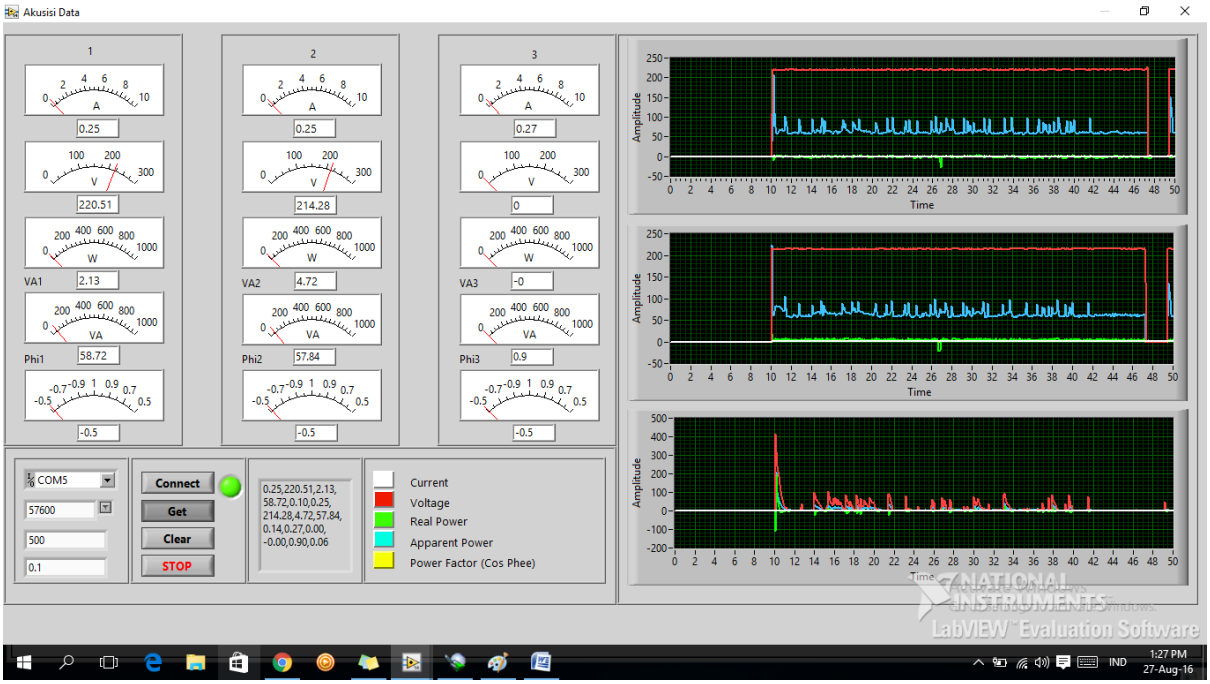
```
    void current(unsigned int _inPinI, double _ICAL);

    void voltageTX(double _VCAL, double _PHASECAL);
    void currentTX(unsigned int _channel, double _ICAL);

    void calcVI(unsigned int crossings, unsigned int timeout);
    double calcIrms(unsigned int NUMBER_OF_SAMPLES);
    void serialprint();

    long readVcc();
    //Useful value variables
    double realPower,
       apparentPower,
       powerFactor,
       Vrms,
       Irms,
       Voltage;

  private:

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

    //-------------------------------------------------------------------
------------------------
    // Variable declaration for emon_calc procedure
    //-------------------------------------------------------------------
------------------------
      int sampleV;                                        //sample_
holds the raw analog read value
      int sampleI;

      double lastFilteredV,filteredV;          //Filtered_ is the
raw analog value minus the DC offset
      double filteredI;
      double offsetV;                          //Low-pass filter
output
      double offsetI;                          //Low-pass filter
output

      double phaseShiftedV;                               //Holds the
calibrated phase shifted voltage.

      double sqV,sumV,sqI,sumI,instP,sumP;             //sq =
squared, sum = Sum, inst = instantaneous

      int startV;
//Instantaneous voltage at start of sample window.
```

```
     boolean lastVCross, checkVCross;                  //Used to
measure number of times threshold is crossed.


};

#endif
```

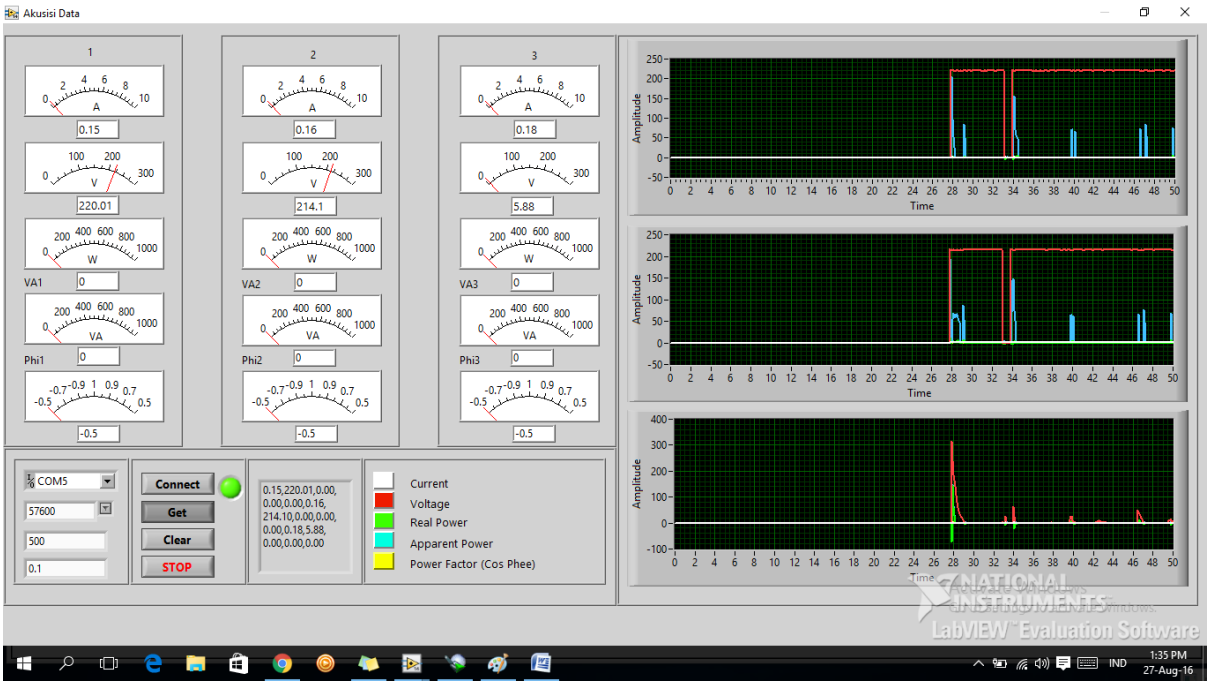
```
// EmonLibrary examples openenergymonitor.org, Licence GNU GPL V3

#include "EmonLib.h"                   // Include Emon Library
EnergyMonitor emon1;                   // Create an instance

void setup()
{
  Serial.begin(9600);

  emon1.current(1, 111.1);             // Current: input pin, calibration.
}

void loop()
{
  double Irms = emon1.calcIrms(1480);  // Calculate Irms only

  Serial.print(Irms*230.0);          // Apparent power
  Serial.print(" ");
  Serial.println(Irms);              // Irms
}
//current and voltage
// EmonLibrary examples openenergymonitor.org, Licence GNU GPL V3

#include "EmonLib.h"             // Include Emon Library
EnergyMonitor emon1;             // Create an instance

void setup()
{
  Serial.begin(9600);

  emon1.voltage(2, 234.26, 1.7);  // Voltage: input pin, calibration,
phase_shift
  emon1.current(1, 111.1);       // Current: input pin, calibration.
}

void loop()
{
  emon1.calcVI(20,2000);         // Calculate all. No.of half wavelengths
(crossings), time-out
  emon1.serialprint();           // Print out all variables (realpower,
apparent power, Vrms, Irms, power factor)

  float realPower       = emon1.realPower;        //extract Real Power into
variable
  float apparentPower   = emon1.apparentPower;    //extract Apparent Power
into variable
  float powerFActor     = emon1.powerFactor;      //extract Power Factor
into Variable
  float supplyVoltage   = emon1.Vrms;             //extract Vrms into
Variable
  float Irms            = emon1.Irms;             //extract Irms into
Variable
}
```

LAMPIRAN GAMBAR

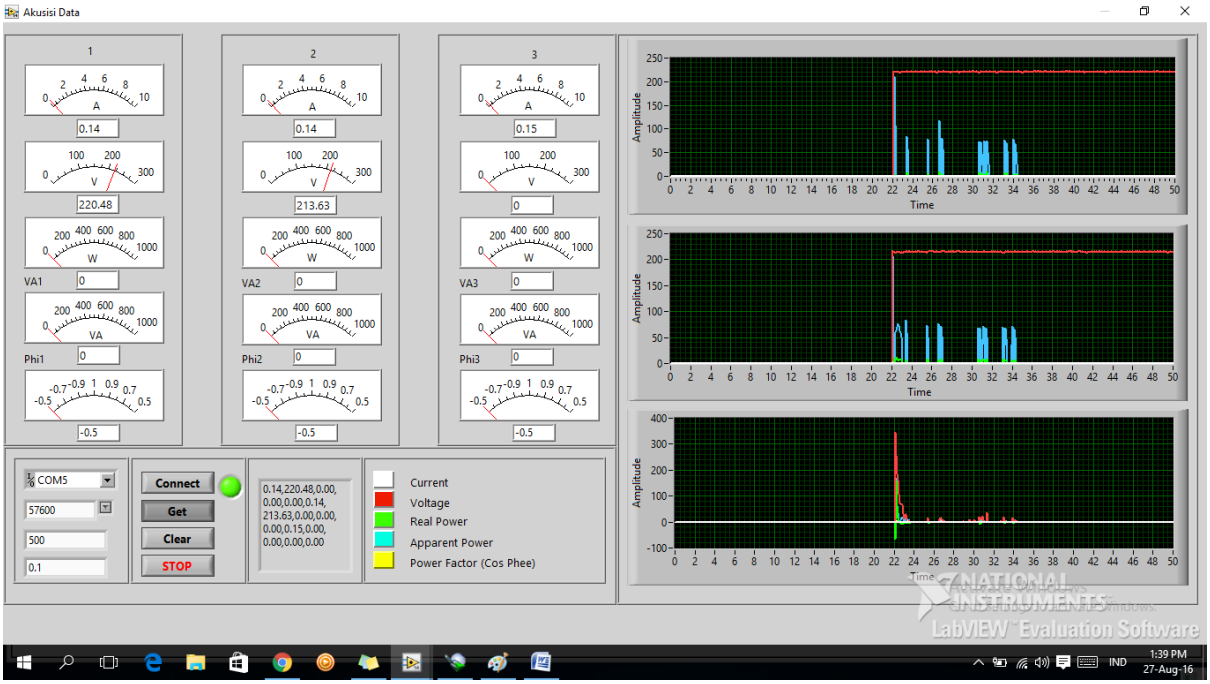
PENGUKURAN 1



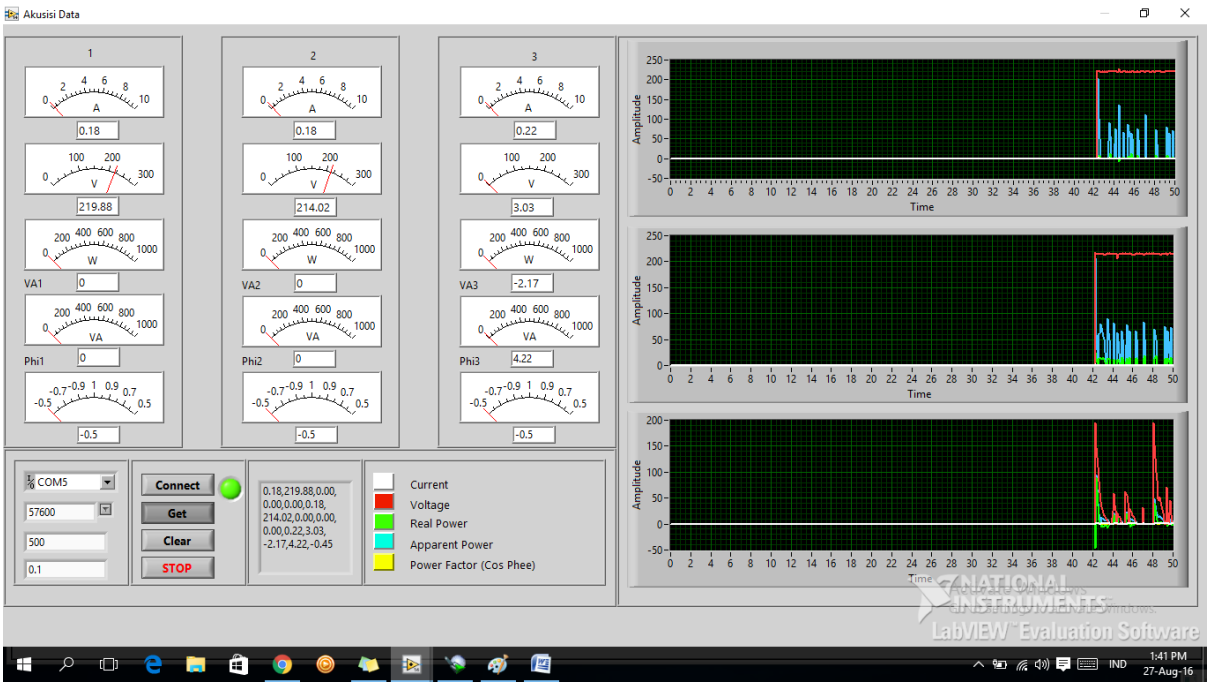
PENGUKURAN 2



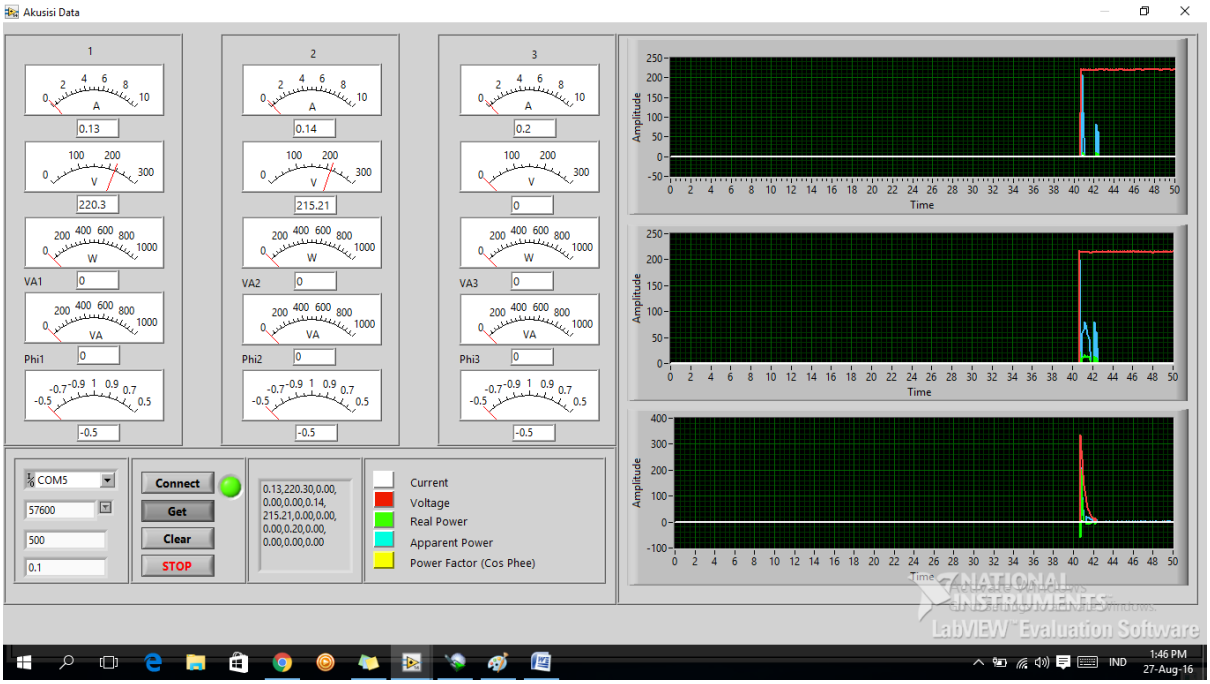
PENGUKURAN 3



Pengukuran 4

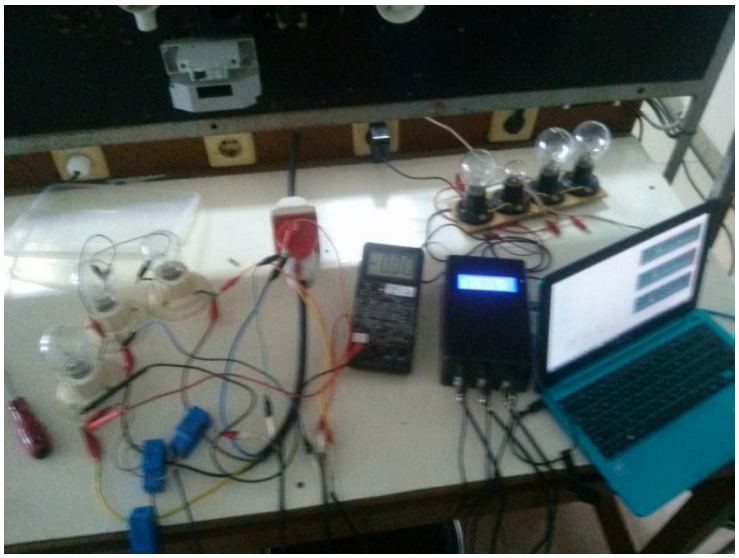


Pengukuran 5



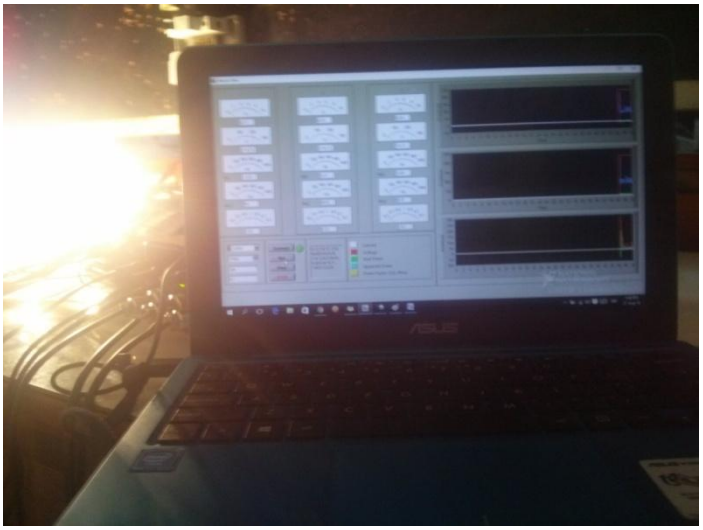
Dokumentasi

Gambar 1



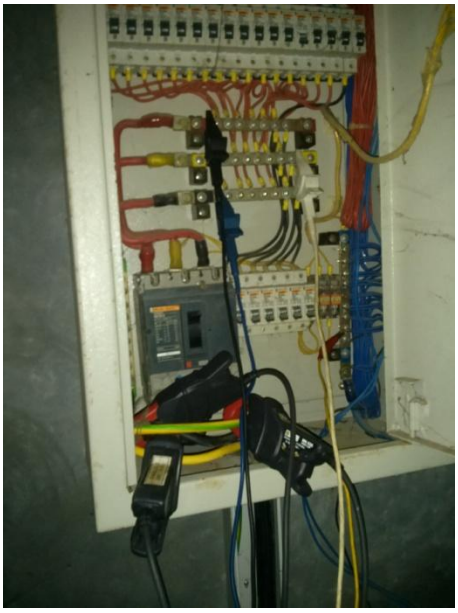
Gambar 2

Gambar 3

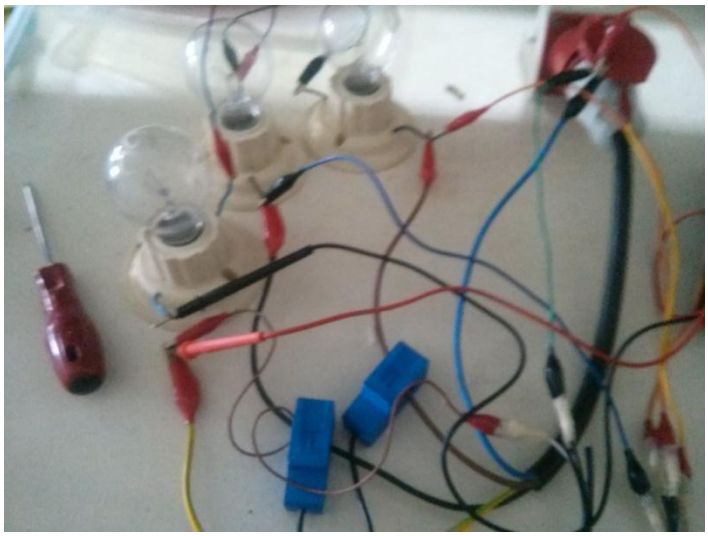


Gambar 4

Gambar 5



Gambar 6



Gambar 7

Gambar 8