

Signal Conversion for Sampling a Sinewave with the Arduino Yun

Application Note

Michigan State University
Senior Design – ECE 480 – Team 3

Facilitator

Dean Aslam

Team Member

Victor Tobenna Ezenwoko

November 4, 2015



Executive Summary

The purpose of this project is to develop a three-phase four-wire AC power monitoring system capable of data logging voltage phase-phase, voltage phase-ground, current, and power. The system will alert a user at the occurrence of any transient event, as well as permit the user the ability to change the sampling rate, and possess enough memory to log data for a week with a sampling rate of 100 milliseconds. The system will also have a user interface to allow data viewing, changes for monitoring purposes, and the ability to export the logged data with a USB memory stick.

One of the most vital parts in achieving our project is designing the circuit and codes which would enable our chosen microcontroller, the Arduino Yun, to take an input voltage or current signal and convert it to a format it can sample.

Keywords

- Arduino yun
- Voltage & current signals
- Signal conversion
- Sinewave sampling

Table of Contents

➤ Background	4
➤ Arduino Yun	5
➤ Signal Conversion	6
○ Circuit Schematic	7
➤ Programming	8
➤ Recommendations	11
➤ References	12

Background

Great Lakes Control & Engineering were recently contracted with the task of developing a power monitoring system that would be applied in factories which deal with 3 phase power that could possess voltage imbalances and a poor power factor. Voltage imbalances damage delicate machinery which could be expensive to repair or replace. Poor power factors result in additional charges from power utilities.

Voltage imbalance is a condition when the voltage to the major 3-phase components is significantly different between the 3 phases. Three phase power consists of 3 “hot” wires each having full line voltage with respect to the other two. These three voltages should be nearly, if not exactly, equal in voltage to each other. If the voltages are too far out of balance, components (like motors and compressors) will start to overheat. Motors and compressors operating with too high a voltage imbalance may continue to run but at elevated motor winding temperatures. This, in turn, will reduce the life of that component. ^[1] Voltage unbalance exceeding more than 2 percent in three-phase systems can cause current unbalance among the windings. These, in turn, can cause an increase in winding temperature and an overheating problem that can be harmful to the motor. ^[2] In addition, many solid-state motor controllers and inverters include components that are especially sensitive to voltage imbalances. ^[3] The adverse effect of unbalanced voltages on induction motors stem from the fact that the unbalanced voltage breaks down into two opposing components that can be described using a method of symmetrical components. Through the use of symmetrical components, we can describe an unbalanced three-phase system of voltage or current phasors using three balanced systems of phasors termed positive, negative and zero sequence. Positive sequence voltage produces the desired positive torque, but the negative sequence voltage produces an air gap flux rotating against the rotation of the rotor producing an unwanted negative torque. ^[4]

In order to achieve this project, it is beyond critical we design a circuit that would handle the signal conversion, as well as program the microcontroller with the ability to detect and sample the input voltage and current signals.

Arduino Yun

Arduino is an open-source platform used for constructing and programming of electronics. It can receive and send information to most devices, and even through the internet to command the specific electronic device. ^[5] The Arduino Yún is a microcontroller board based on the ATmega32u4 and the Atheros AR9331. The Atheros processor supports a Linux distribution based on OpenWrt named OpenWrt-Yun. The Yún distinguishes itself from other Arduino boards in that it can communicate with the Linux distribution onboard, offering a powerful networked computer with the ease of Arduino. ^[6] In addition to Linux commands like cURL, you can write your own shell and python scripts for robust interactions.

Although Arduino is typically seen as a hobbyist controller, given our short timeframe for the project and the inexperience of the team in terms of computer programming, this microcontroller served as the best choice for this project. There are many open source codes and information on the internet pertaining to our project like for sending emails, writing to LCD displays, writing to USB, to name a few. The board has built-in Ethernet and WiFi support, a USB-A port, micro-SD card slot, 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, an ICSP header, and a 3 reset buttons.



Figure 1. Arduino Yun microcontroller

Signal Conversion

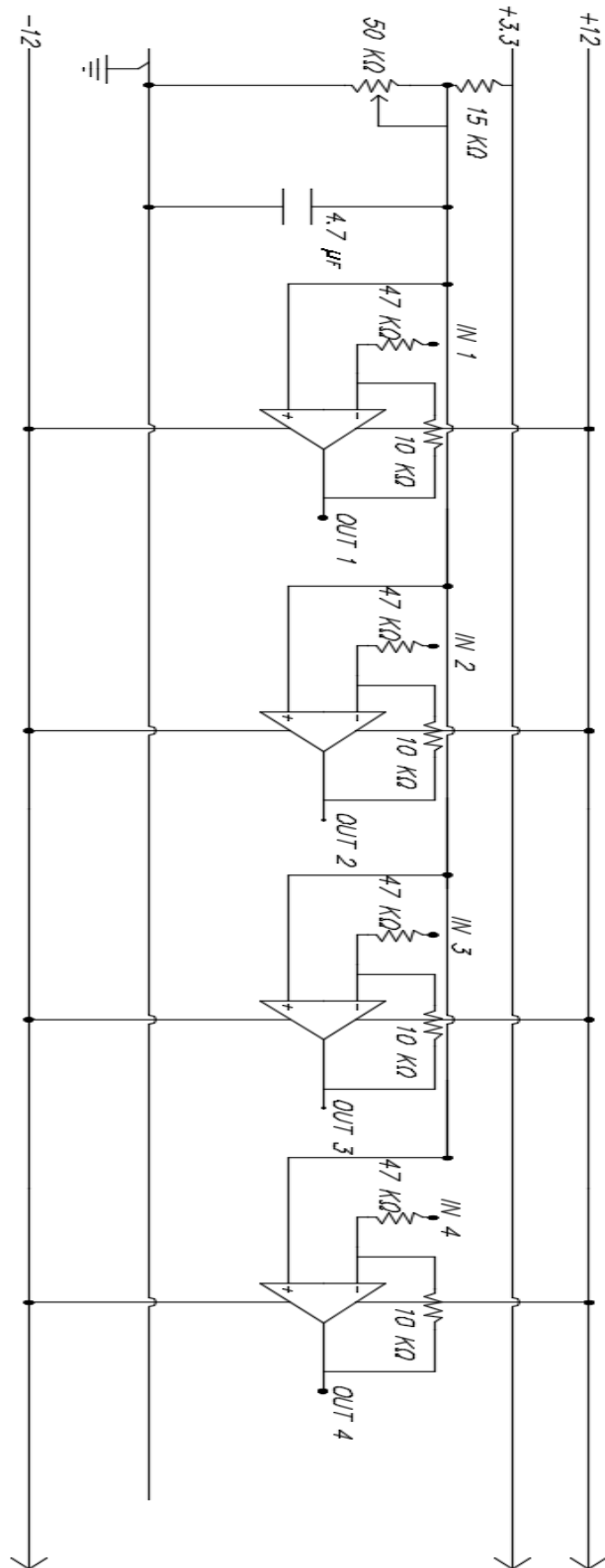
The most vital part of our design is the circuit's ability to take an input voltage or current signal and convert it to a format our microcontroller can sample. Our microcontroller can only read voltages ranging from 0 to 5 V. ^[7] To solve this issue, we designed a summing amplifier that would scale down the input signal and also add a 2.5 V DC offset, which would enable us to properly see the sine wave of the voltage or current input signal. ^[8] A feature of the design is having a feedback of power from the output circuit to the input circuit of the amplifier of such magnitude and phase as to reduce the input impedance of the amplifier to a small value and to make the over-all gain of the amplifier a predetermined quantity. As the effective gain for each source is controlled by the ratio of the feedback resistor to the input resistor, the voltages from the sources may have different gains, thus multiplying or dividing one voltage with respect to the others. ^[9]

We plan to use transformers that will step down the signals to a 5 Vrms voltage; therefore, we will have an input of roughly 15 Vpp. If we build an inverting summer, the signal will be inverted which would not distort any information as long as each signal was inverted. We can also choose resistors such that the input signal is scaled down by a value of 4.7. If we are able to get a 5 Vpp input to the microcontroller, it would mean our input to the signal conversion circuit would be, $4.7 \times 5 = 23.5$ Vpp, well suited to handle the expected 15 Vpp signal. We used to quad operational amplifiers and 47k Ω and 10k Ω resistors to realize this design as can be seen in the circuit schematic on page 7.

To create the DC offset, we used a voltage divider with a variable pot to grant us the ability to control the output offset through calibration. The input terminals of the operational amplifier have a high impedance so there is little concern of loading effects while using a voltage divider and to create a stiff voltage, we used 4.7 μ f capacitors tied to ground.

All the necessary parts used in creating this circuit was found in the senior design laboratory, while the basic circuit knowledge I used in designing this circuit were concepts I learned in ECE 302 & ECE 402 with Professor Wierzba. ^[10] In building the circuit, I tested each step using the Tested using the Agilent E3611A Power Supply and the Digital Multimeter in the laboratory. I tested the circuit and all tests provided a DC offset ranging from 2.493 V to 2.515 V, which are within error range for our desired DC offset voltage.

Circuit Schematic for the Signal Conversion Unit



Programming

After designing and testing the signal conversion circuit, the next step was to ensure the signals are read properly by the microcontroller, as well as programming the microcontroller to correctly sample the signals.

Here's a compilation of different sections of the code regarding sampling the sinewave from a converted input signal. The code shown programs the microcontroller to detect and read the voltage/current signal, as well as carry out the necessary computation to get the rms voltage, rms current, real power, reactive power, and power factor. It is worth mentioning that the microcontroller was programmed to take samples every 1ms.

Although parts of the codes, mainly the libraries, were found online, it was developed and tested using basic coding principles learned in CSE 251, and basic power calculation formulas learned in multiple classes.

```
#include <Wire.h>
#include <FileIO.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd1(0x25, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
LiquidCrystal_I2C lcd2(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
LiquidCrystal_I2C lcd3(0x23, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
LiquidCrystal_I2C lcd4(0x26, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

unsigned long int taskTime[10];
unsigned long int taskPeriod[10];
int Voltage_X = A0; //Read voltage through A0
int Voltage_Y = A1;
int Voltage_Z = A2;
int Current_X = A3;
int Current_Y = A4;
int Current_Z = A5;
uint16_t VoltageRead_X, VoltageRead_Y, VoltageRead_Z;
uint16_t CurrentRead_X, CurrentRead_Y, CurrentRead_Z;
int VoltageAC_X, VoltageAC_Y, VoltageAC_Z;
int CurrentAC_X, CurrentAC_Y, CurrentAC_Z;
unsigned long int PowerInst_X, PowerInst_Y, PowerInst_Z;
unsigned long int VoltageSquared_X, VoltageSquared_Y, VoltageSquared_Z;
unsigned long int CurrentSquared_X, CurrentSquared_Y, CurrentSquared_Z;
unsigned long int VoltageSum_X, VoltageSum_Y, VoltageSum_Z;
unsigned long int CurrentSum_X, CurrentSum_Y, CurrentSum_Z;
unsigned long int PowerSum_X, PowerSum_Y, PowerSum_Z;
int VoltageRMS_X, VoltageRMS_Y, VoltageRMS_Z;
int CurrentRMS_X, CurrentRMS_Y, CurrentRMS_Z;
float PowerRealAverage_X, PowerRealAverage_Y, PowerRealAverage_Z;
unsigned long int PowerApparent_X, PowerApparent_Y, PowerApparent_Z;
float PowerFactor_X, PowerFactor_Y, PowerFactor_Z;
float VoltageReal_X, VoltageReal_Y, VoltageReal_Z; //VoltageRMS converted to real voltage
float CurrentReal_X, CurrentReal_Y, CurrentReal_Z;
float PowerReal_X, PowerReal_Y, PowerReal_Z;
int LowVoltage_X, LowVoltage_Y, LowVoltage_Z;
int LowCurrent_X, LowCurrent_Y, LowCurrent_Z;
int NumSamples = 1000; //Number of samples to calculate RMS
int ArrayLocation = 0;
int VoltageCutoff = 128;
```

```

int CurrentCutoff = 61;
int VoltageScale = 340;
int CurrentScale = 50;

//*****

void setup()
{
  Serial.begin(9600);
  Bridge.begin();
  FileSystem.begin();
  lcd1.begin(16,2);
  lcd2.begin(16,2);
  lcd3.begin(16,2);
  lcd4.begin(16,2);
  taskPeriod[0] = NumSamples; //how often it prints VoltageRMS to LCD
  taskPeriod[1] = 1; //read a new value of array Sin5V every 1ms
  taskPeriod[3] = NumSamples;
  pinMode(Voltage_X, INPUT);
  pinMode(Voltage_Y, INPUT);
  pinMode(Voltage_Z, INPUT);
  pinMode(Current_X, INPUT);
  pinMode(Current_Y, INPUT);
  pinMode(Current_Z, INPUT);
  //*****
  taskPeriod[2] = 20;
  pinMode(ButtonOne, INPUT);
  digitalWrite(ButtonOne, HIGH);
  pinMode(ButtonTwo, INPUT);
  digitalWrite(ButtonTwo, HIGH);
  OldButtonOneState = digitalRead(ButtonOne);
  OldButtonTwoState = digitalRead(ButtonTwo);
  //*****
}

void(* resetFunc)(void) = 0;

void loop(){
  unsigned long int loopTime = millis();
  String dataString; //Used to format data for MicroSD
  //*****
  CurrentButtonOneState = digitalRead(ButtonOne);
  CurrentButtonTwoState = digitalRead(ButtonTwo);
  //*****

  // *****
  // *****Read Voltage*****
  // *****
  if(loopTime - taskTime[1] > taskPeriod[1]){
    taskTime[1] = loopTime-1;
    VoltageRead_X = analogRead(Voltage_X);
    VoltageRead_Y = analogRead(Voltage_Y);
    VoltageRead_Z = analogRead(Voltage_Z);
    CurrentRead_X = analogRead(Current_X);
    CurrentRead_Y = analogRead(Current_Y);
    CurrentRead_Z = analogRead(Current_Z);
    VoltageAC_X = VoltageRead_X-512;
    VoltageAC_Y = VoltageRead_Y-512;
    VoltageAC_Z = VoltageRead_Z-512;
  }
}

```

```

CurrentAC_X = CurrentRead_X-512;
CurrentAC_Y = CurrentRead_Y-512;
CurrentAC_Z = CurrentRead_Z-512;
PowerInst_X = (float)VoltageAC_X*(float)CurrentAC_X;
PowerInst_Y = (float)VoltageAC_Y*(float)CurrentAC_Y;
PowerInst_Z = (float)VoltageAC_Z*(float)CurrentAC_Z;
VoltageSquared_X = sq((float)VoltageAC_X);
VoltageSquared_Y = sq((float)VoltageAC_Y);
VoltageSquared_Z = sq((float)VoltageAC_Z);
CurrentSquared_X = sq((float)CurrentAC_X);
CurrentSquared_Y = sq((float)CurrentAC_Y);
CurrentSquared_Z = sq((float)CurrentAC_Z);
VoltageSum_X = VoltageSum_X + VoltageSquared_X;
VoltageSum_Y = VoltageSum_Y + VoltageSquared_Y;
VoltageSum_Z = VoltageSum_Z + VoltageSquared_Z;
CurrentSum_X = CurrentSum_X + CurrentSquared_X;
CurrentSum_Y = CurrentSum_Y + CurrentSquared_Y;
CurrentSum_Z = CurrentSum_Z + CurrentSquared_Z;
PowerSum_X = PowerSum_X + PowerInst_X;
PowerSum_Y = PowerSum_Y + PowerInst_Y;
PowerSum_Z = PowerSum_Z + PowerInst_Z;
ArrayLocation = ArrayLocation + 1;
if(ArrayLocation == NumSamples){
VoltageRMS_X = sqrt(VoltageSum_X/NumSamples);
VoltageRMS_Y = sqrt(VoltageSum_Y/NumSamples);
VoltageRMS_Z = sqrt(VoltageSum_Z/NumSamples);
CurrentRMS_X = sqrt(CurrentSum_X/NumSamples);
CurrentRMS_Y = sqrt(CurrentSum_Y/NumSamples);
CurrentRMS_Z = sqrt(CurrentSum_Z/NumSamples);
PowerRealAverage_X = PowerSum_X/NumSamples;
PowerRealAverage_Y = PowerSum_Y/NumSamples;
PowerRealAverage_Z = PowerSum_Z/NumSamples;
PowerApparent_X = (float)VoltageRMS_X*(float)CurrentRMS_X;
PowerApparent_Y = (float)VoltageRMS_Y*(float)CurrentRMS_Y;
PowerApparent_Z = (float)VoltageRMS_Z*(float)CurrentRMS_Z;
PowerFactor_X = PowerRealAverage_X/PowerApparent_X;
PowerFactor_Y = PowerRealAverage_Y/PowerApparent_Y;
PowerFactor_Z = PowerRealAverage_Z/PowerApparent_Z;

//*****

```

Recommendations

After conducting research and carrying out multiple circuit trials, I was successful in converting the input signals for sampling a sinewave with our microcontroller, the Arduino Yun. However, there are a few minor future suggestions and recommendation I had in mind during and after the creation of this design.

Firstly, I would suggest more trial and error and calculations to obtain resistor values, to replace the current ones, which would not be too far apart from each other, provide exactly the desired 2.5V DC offset and also increase the resistive tolerance of the circuit.

Secondly, I would recommend further design implementations on the codes for sampling the DC voltage. For future designs, I would suggesting looking into more efficient digital ways in sampling the sinewave signal. In addition, we use 512 to represent our V_{cc} which is meant to be 5 V, but 512 gives approximately 4.92 v. Although 4.92 V is pretty close to our desired value, the less error achievable the better. A possible solution would be programming our microcontroller to take an average of the last set of samples taken and use that as a reference for subtracting when acquiring the voltage/current AC values; we currently use 512, and taking that average value would provide more accurate results.

Finally, I would suggest switching to a better and more technologically advanced microcontroller. We selected the Arduino Yun due our lack of programming experience; therefore, a higher level microcontroller could potentially provide more accurate results, as well as, add more functionality to our device.

References

- [1] Voltage Imbalance in 3-Phase Systems, York Central Tech Talk.
<https://yorkcentraltechtalk.wordpress.com/2012/07/03/voltage-imbalance-in-3-phase-systems/> 07/03/2012.
- [2] Three-Phase Motor Voltage Unbalance, The NEWS Magazine.
<http://www.achrnews.com/articles/93649-three-phase-motor-voltage-unbalance>
04/04/2005. Authorship by John Tomczyk, professor of HVACR at Ferris State University.
- [3] The Basics of Voltage Imbalance, Electrical Construction & Maintenance (EC&M)
<http://ecmweb.com/content/basics-voltage-imbalance> 09/01/1999. Authorship by Paul Frank.
- [4] A. von Jouanne and B. B. Banerjee, *Assessment of voltage unbalance*, IEEE Transactions on Power Delivery, IEEE, Volume: 16, Issue: 4, page 782-790, 2001.
- [5] Badamasi, Y.A., *The working principle of an Arduino*, Electronics, Computer and Computation (ICECCO), IEEE, page 1-4, 2014.
- [6] Barrett, S., "Arduino Microcontroller:Processing for Everyone! Second Edition," in *Arduino Microcontroller:Processing for Everyone! Second Edition* , 1, Morgan & Claypool, pp.371, IEEE, 2012.
- [7] Arduino Yun Datasheet. <https://www.arduino.cc/en/Main/ArduinoBoardYun>.
- [8] 1. S.J.K. D, Summing ampli_er, June 11 1946, US Patent 2,401,779.
- [9] Gray, Paul R., Paul Hurst, Robert G. Meyer, and Stephen Lewis. *Analysis and design of analog integrated circuits*. Wiley, 2001.
- [10] ECE 402 Applications of Analog Integrated Circuits Course e-Notes, 2015. Authorship by Gregory M. Wierzba.