Project on

**Prototype Implementation of Phasor Measurement Unit**

Submitted in partial fulfillment of the requirement Of the degree of

Bachelor of Electrical Engineering By

E-20-0319 Omkar Prakash Sawant E-20-0027 Sahil Vishwas Chalke

E-20-0316 Nikil Santosh Ghadge E-20-0218 Amey Ajit Palande

Under the guidance of Prof.Priya Potdar



# Department of Electrical Engineering Finolex Academy of Management and Technology

**University of Mumbai 2023-24**

Project on

**Prototype Implementation of Phasor Measurement Unit**

By

E-20-0319 Omkar Prakash Sawant E-20-0027 Sahil Vishwas Chalke

E-20-0316 Nikil Santosh Ghadge E-20-0218 Amey Ajit Palande

Semester – VIII



# Department of Electrical Engineering Finolex Academy of Management and Technology

**MIDC , Mirjole Block , Ratnagiri District : Ratnagiri , PIN :415639 2023-24**

**Certificate**

This is to certify that the following students have satisfactorily completed project work on **Protoype Implementation of Phasor Measurement Unit** submitted to **University of Mumbai** in partial fulfillment of the Bachelor’s Degree in Electrical Engineering course of **Semester VIII.**

Name of students

E-20-0319 Omkar Prakash Sawant E-20-0027 Sahil Vishwas Chalke

E-20-0316 Nikil Santosh Ghadge E-20-0218 Amey Ajit Palande

(Prof.Priya Potdar) Guide

(Dr.Jayant Mane) Dr.Kaushal Prasad Head Of Department Principal

Finolex Academy of Management and Technology

District : Ratnagiri , PIN :415639

# Department of Electrical Engineering Approval Sheet

**Prototype Implementation of Phasor Measurement Unit**

Submitted by

E-20-0319 Omkar Prakash Sawant E-20-0027 Sahil Vishwas Chalke

E-20-0316 Nikil Santosh Ghadge E-20-0218 Amey Ajit Palande

In partial fulfillment of the Degree of B.E.(Semester VIII) in Electrical Engineering during Academic year 2023-24 approved.

(Prof.Priya Potdar) Examiner Guide 1.

2.

(Dr.Jayant Mane) (Dr.Kaushal Prasad) Head Of Department Principal

**Declaration**

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of students Signature

E-20-0319 Omkar Prakash Sawant E-20-0027 Sahil Vishwas Chalke

E-20-0316 Nikil Santosh Ghadge E-20-0218 Amey Ajit Palande

# Date :

**Acknowledgement**

I express my sincere gratitude to my supervisor, Prof. P.A.Potdar for his valuable guidance and suggestions without which this thesis would not be in its present form. I also thank her for his consistent encouragements throughout the work. I express my gratitude to Dr. J.J.Mane, the Head of the Department of Electrical Engineering,FAMT for extending some facilities towards completion of this thesis. Thanks also to other faculty members in the department.

Sawant Omkar Prakash Chalke Sahil Vishwas Ghadge Nikil Santosh Palande Amey Ajit

# Contents

|  |  |  |  |
| --- | --- | --- | --- |
| Abstract | | | **i** |
| List of Figures | | | **ii** |
| List of Acronyms | | | **iii** |
| Chapter | Chapter Name | | Page |
| 1 | Introduction | |  |
|  | 1.1 | Introduction |  |
|  | 1.2 | Motivation |  |
|  | 1.3 | Aim and Objective |  |
|  | 1.4 | Problem definition |  |
|  | 1.5 | Expected outcome |  |
| 2 | Literature Review | |  |
|  | 2.1 | Grid Trak PMU |  |
| 2.2 | DTU PMU |
| 2.3 | The open PDC |
| 3 | Overview of System and Software development | |  |
|  | 3.1 | Block Diagram |  |
|  | 3.2 | Phasor Calculation For 3 phase system |  |
|  | 3.3 | Signal Acquisition |  |
|  | 3.4 | Methods of frequency estimation |  |
|  | 3.5 | GPS Disciplined Oscillator (GPSDO) |  |
|  | 3.6 | Sampling and Time Stamping |  |
|  | 3.7 | Phasor Calculation |  |
|  | 3.8 | Local Display Terminal |  |
|  | 3.9 | Power Supply Unit |  |
|  | 3.10 | MATLAB Simulation |  |
| 4 | Conclusions | |  |
|  | 4.1 | Proposed direction for Future Work |  |
|  | References | |  |
|  | Appendix – A | |  |

**Abstract**

As the world continues to move towards a Smarter Grid day by day, it has become the necessity to incorporate real-time monitoring of the grid wherein the instantaneous snapshot of the health of the grid can be made available. No other parameters than the Instantaneous Phasors, considered to be the heart- beats of the Electrical Grid, can represent the complete health status of the grid. This paper discusses how an Open Hardware Platform (Arduino Due with ARM Cortex M3 Micro-controller) can be used to estimate the phasors of a three phase system in real-time. The Pulse Per Second (PPS) signal from a GPS module is used to generate the sampling pulses. These pulses synchronize the sampling process by the Analog to Digital Converters (ADC), used by the PMU throughout the globe because of the high accuracy of the atomic clocks in the GPS satellites. The microcontroller uses a 64-Point DFT algorithm to estimate the phasors. The reference time is obtained from the GPS module which is the UTC time, with which the phasors are time stamped and displayed in a real-time Graphical User Interface (GUI) designed using Python (another open source programming language).

# List Of Figures

|  |  |  |
| --- | --- | --- |
| Sr. no. | Figure name | Page no |
| 3.1 | Block Diagram |  |
| 3.2 | Programmed GPS Disciplined Oscillator |  |
| **3.3** | Flow Chart of the Phasor Processor |  |
| **3.4** | MATLAB Simulation |  |
| **3.5** | **Simulation Result** |  |
|  |  |  |

**List Of Acronyms**

PMU : Phasor Measurement Unit COF : Change of Frequency

ROCOF : Rate Of Change Of Frequency GPS : Global Positioning System GPSDO : GPS Disciplined Oscillator PPS : Pulse Per Second

ADC : Analog to Digital Converter SPI : Serial Peripheral Interface PROGMEM : Program Memory

I2C : Inter Integrated Circuit

UART : Universal Asynchronous Receiver and Transmitter TFT : Thin Film Transistor

UTFT : Universal TFT

UTC : Universal Coordinated Time ARM : Advanced RISC Machines

LDO : Linear(Low) Dropout Regulator USB : Universal Serial Bus

# Chapter 1 Introduction

* 1. **Introduction**

The load dispatch centres in a large power system supervise and control over the transmission network and it takes preventive actions to avoid any sort of system failure which can hamper electricity distribution. With ever increasing size and complexity of the power system, the ability to detect any faults in the power system is heavily dependent on the real time information available to the operator. Traditionally, analog and digital information (status of circuit breaker, power flow and frequency) is measured at the substation level and transmitted to load dispatch center using supervisory control and data acquisition system (SCADA) or energy management system (EMS). The major limitation of SCADA or EMS is the inability to accurately calculate the phase angle between a pair of substations. In SCADA or EMS, phase angle is either estimated from available data or is calculated offline. Phasor Measurement Units (PMU) overcome the limitations of SCADA and EMS by accurately calculating the phase angle between a pair of grid. Synchronized phasor measurement units were introduced in the mid-1980s as a solution for the need of more efficient and safer monitoring devices for Electric Power Systems(EPS). Since then, measuring Electric Power System (EPS) parameters of voltage and current in relatively distant buses has received great attention from researchers. Such measurements are performed by phasor measurement units (PMUs), synchronized by Global Positioning System (GPS) satellites. The advantage of referring phase angle to a global reference time is helpful in capturing the wide area snap shot of the power system. Effective utilization of this technology is very 2 Introduction useful in mitigating blackouts and learning the real time behavior of the power system. Since the bus voltage angle of a power system is very closely linked with the behavior of a network, its real time measurement is a powerful tool for operating a network. A commercial PMU measures the voltage and angle of a particular grid at 25 samples per second. The phase information is synchronized with Global Positioning Systems (GPS) satellite and is transmitted to Phasor Data

Concentrator (PDC) through a high speed communication network. The time stamped phase information is called synchrophasor. There are several benefits of PMU such as monitoring of EPS and network protection. The measurement of voltage and current in remote bus allows the operator to make a concrete decision about the maintenance and security of the system in the face of various uncertainties. As on 31st May 2012, fourteen PMUs are commissioned in India.

# Motivation

Phasor Measurement Units (PMUs) are crucial in power systems for real-time monitoring, control, and protection. They provide precise information about voltage, current, frequency, and phase angle, enabling operators to optimize grid stability, manage power flow, and detect and mitigate disturbances swiftly. Ultimately, PMUs enhance grid reliability, efficiency, and resilience, which are vital for modern electrical infrastructure.

# Aim and Objective

* + - Real-time monitoring: Continuously monitor the state of the power grid to detect abnormalities and potential issues.
    - Grid stability: Facilitate rapid identification of disturbances and enable fast corrective actions to maintain grid stability.
    - Power system control: Assist in the effective control of power flow, voltage levels, and frequency within the grid.

# Problem Definition

* + - Accuracy and Synchronization: Ensuring PMU measurements are highly accurate and synchronized across the grid, especially in large-scale systems with diverse sources of generation and varying loads.
    - Data Management: Handling the large volume of data generated by PMUs efficiently, including storage, processing, and analysis, to extract actionable insights in real-time.
    - Cybersecurity: Protecting PMUs from cyber threats to prevent unauthorized access, data manipulation, or disruption of grid operations.
    - Cost-effectivness: Balancing the benefits of PMU deployment with the associated costs, including equipment installation, maintenance, and operational expenses, to justify investments in grid modernization efforts.

# Expected Outcomes

* + - Improved grid stability: PMUs enable real-time monitoring and control of grid parameters, allowing operators to take proactive measures to maintain stable voltage levels, frequency, and power flow within the grid.
    - Faster Fault detection: PMUs enable rapid detection of faults and grid disturbances, fascilitating quicker restoration of service through automated or manual interventions.
    - Optimized power system operation: With comprehensive data on grid conditions, operators, can make informed decisions to optimize power flow, manage congestion, and minimize losses, leading to improved efficiency and cost savings.

Chapter 2

# Literature Review

The measurement of voltage phase angles using synchronized clocks for power system applications dates back to the early 1980s when measurements of voltage phase angles were carried out between Montreal and SEPT-ILES and parallel efforts by Bonanomi in 1981. However, the synchrophasor technology available today emerged from the early efforts by Phadke et al. at Virginia Tech as described. Phadke demonstrated the first synchronized PMU in 1988, and in 1991 Macrodyne Inc. launched the first commercial PMU product. Due to the cost of early PMU devices, PMU technology has historically been limited to transmission system applications where the business case justified expensive phasor analysis equipment. One of the early applications that is important to mention is the implementation of the wide-area protection system Syclopes in France in the early 1990s, which was the first functional application of early forms of PMUs. The cost of the the components from which PMUs are assembled (such as GPS receivers, microprocessors, and storage devices) have been dropped significantly due to the Recent developments across the electronics sector. As a consequence, PMUs have reached price points that have made them an attractive tool for the distribution systems and embedded generation. Many PMUs are sold as dedicated devices which offer event recorder type functionality. Costs for such units vary between US $6000 and US

$15000 depending on the specification. Many equipment vendors have begun to offer PMU functionality as a supplementary feature on other products in their range, such as protection relays.

The standard for PMU devices is maintained by the IEEE C37.118 Working Group. IEEE Std. C37.118 was released in 2005 and subsequently updated in 2011. The latest release comes in two parts; IEEE C37.118.1-2011 describes how synchrophasors should be estimated and gives certification requirements while IEEE C37.118.2-2011 describes data representation and data transfer. Concerns have been raised regarding the transient performance of PMUs under the 2005 standard. These concerns are addressed in the 2011 release of the standard. IEEE C37.118.1-2011 states that it defines synchrophasors, frequency, and rate- of-change-of frequency measurement under all operating conditions. A significant barrier regarding the use of PMU technology in research is the closed philosophy under which commercial PMU devices are developed and sold. Commercial vendors tightly guard their hardware and software designs,

meaning that the measurement processes and algorithms are not known to researchers. This has led to some research departments developing their own PMU systems. Many designs utilize low cost hardware, such as described in .Two university projects are described in this section. Duplication of such work leads to lost time and resources. The Open PMU project provides a common set of resources for PMU development and research collaboration. The successful opensource Phasor Data Concentrator, open PDC , is discussed, and the rational for using an open-source model is developed.

* 1. **Grid Trak PMU**

The GridTrak PMU was produced at Baltimore University by Stadlin; subsequently, the design has been published under open-source license. The aim of GridTrak is to produce an inexpensive PMU that can be widely distributed, among researchers and amateur enthusiasts, allowing widespread monitoring of the distribution network. The design works via a zero crossings technique, making the unit simple and robust; however, the loss of pointon-wave information reduces GridTraks applications. The GridTrak hardware converts the ac measurement signal into three square waves triggered at the crossing of reference voltages. Frequency estimation is determined by the interval between the crossings while voltage is estimated by imposing a perfect sine wave on the full set of crossing points and determining the magnitude. The GridTrak incorporates a GPS module from which it derives time and estimates phase angle. This design is limited to single phase measurements, and all point-on- wave and harmonic information is lost.

## DTU PMU

The DTU PMU [16] was produced in several stages at the Technical University of Denmark. The DTU PMU utilizes two PCs to monitor the ac voltage signal, actively synchronizing the sample rate to 64 or 128 samples per cycle, to fit the waveform. The first PC runs MS-DOS in a near real-time state, stripping out background programs that might interrupt measurements. These measurements are packeted and exported to the second PC at intervals of 20 ms. The second PC runs Labview; in this environment waveform parameters are estimated and the information is archived locally as well as exported in IEEE C37.118 format to a central location. The PMU was thoroughly tested in house

before ten models were installed across the Danish electricity transmission and distribution grid including wind farms and consumer supply. Through ambient monitoring, this wide-area monitoring system has successfully detected many transient system events as-well-as identifying a 0.8 Hz inter-area oscillation, believed to arise from rotor interaction between generators in Sweden and Eastern Denmark.

## The open PDC

The open PDC was developed in the wake of the Northeast Blackout of 2003. After the blackout, there were recommendations of many grid improvements including increased real time observability. The Super PDC began its development by the Tennessee Valley Authority in 2004 to monitor and archive the PMUs installed by itself. The code was released in 2010, under an open- source license, and the Grid Protection Alliance took on responsibility of developing the program and entered into a contract with the North American Electric Reliability Council. The open PDC online community has taken the venture of recording the history and development of open PDC and associated projects. The open PDC is utilized by the North American Synchrophasor Initiative. The open PDC runs as a Windows Service programmed in the Microsoft Visual Basic Studio (Linux versions are also available). It exists as a modular set of programs that can be combined in different forms to achieve different results. Modules, or Adapters as they are called, are activated through Structured Query Language (SQL) commands in the assigned database (DB) and can be reprogrammed through Visual Basic. The open PDC system primarily operates between real-world telecommunications infrastructure and an archive DB. The adapters within open PDC can be subdivided into three groups: Input, Action, and Output adapters. The input adapters receive the raw telecommunications information (in any of the major synchrophasor communication standards including C37.118), process it to extract the relevant data, and then send it for processing or archival in the SQL DB. Action Adapters can process in real time or post event as well as fulfilling the concentrate/compress functions in the DB. Furthermore, Action Adapters can introduce new measurements, for example by importing Comma Separated Value (CSV) files into an existing archive DB.

Chapter 3

# Overview of System and Software Development

* 1. **Block Diagram**

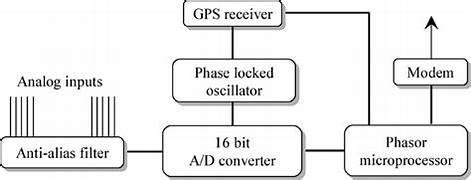


Fig 3.1 Block Diagram of proposed PMU

# Phasor Calculation for 3 Phase-system

Consider a balanced 3-phase power system operating at a nominal frequency of f0, then the voltage waveform can be represented as

x1(t) = Xmcos(2πf0t + φ1) x2(t) = Xmcos(2πf0t + φ2)

x3(t) = Xmcos(2πf0t + φ3) (2.1)

Here Xm represents the maximum amplitude of the signal and Φ represents the phase angle. The phase angles are 120 degree or 2π 3 radian apart. The time domain sample of thepower system can be represented as

Xn1 = Xmcos( 2πn/N + φ1) Xn2 = Xmcos( 2πn/N + φ2)

Xn3 = Xmcos( 2πn/N + φ3) (2.2)

Here N is the number of samples, which is an integer multiple of fundamental frequency f0 and n represents the sample index in the array which ranges from 0 to N − 1. The generalized expression for N-point can be represented as

X = 1/N N X−1 n=0 xn(cos 2πn/N − jsin 2π/N (2.3) N-point DFT of the signal can be found out using

Xk = √ 2/N N X−1 n=0 xn(cos 2πn/N − jsin 2πn/N ) (2.4)

Xnominal = √ 2 N/N X−1 n=0 xn(cos 2πn/N − jsin 2πn/N ) (2.5) The real and imaginary part of the above expression can be rewritten as

Xreal = √ 2 N/N X−1 n=0 xn(cos 2πn/N ) (2.6)

Ximg = √ 2 / N X−1 n=0 xn(cos 2πn/N ) (2.7)

The phasor estimate at nominal frequency is represented by this complex quantity Xnominal, whose magnitude |Xnominal| = q X2 real + X2 img gives the RMS magnitude of the signal. The phase angle can be computed using the trigonometric property,φnominal = atan( Ximg/Xreal ).

# Signal Acquisition

For the calculation of a phasor, the data (i.e. the sampled voltage signal) must be acquired. When the PMU is tested in real-world scenarios a means of getting the signals from the transmission lines is necessary, which is accomplished using a Potential Transformer (PT) and a Current Transformer (CT) in the substations. This signal is further stepped down using the Hall Effect voltage sensors as described in the next section.

* 1. **Methods of frequency estimation**

A PMU must be able to report the frequency of the phases. It should also report the rate of change of frequency. There are roughly two methods to achieve this task. The first one being calculating the frequency by some mathematical formula or doing some computation to find the time derivative of the signal. The second option is modify the input sinusoidal wave into a square wave whose frequency can be simply calculated by measuring the time difference between arrival of two consecutive pulses. The later method is adopted in this PMU design. The circuit to achieve this task of converting sine wave to square wave pulses. The output of this circuit is designed to be a square wave pulse stream, whose voltage levels lie between 0 volt to 3.3 volt. These signals are given to three interrupt pins of the Arduino Due microcontroller, which is acting as the phasor processor. The microcontroller records the time stamp every time a rising edge is detected on the interrupt pins. These time stamps are stored in three variables dedicated to each phase. Upon receiving the second rising edge, again a time stamp is taken and the difference between the previous capture time and the current capture time is calculated. This difference gives the Time period of the signal. The instantaneous frequency is calculated by taking the inverse of this period. The advantage of using this method of frequency calculation is no optimization needs to be done to accurately calculate the frequency, even if there is large deviation from the nominal frequency, like that of estimating phasors using DFT over the sampled signal.

# GPS Disciplined Oscillator (GPSDO)

Recent developments in the PMU has been made possible only because of easy available of the synchronizing pulses which are derived from the GPS modules. The GPS satellites have multiple number of Atomic clocks on board, which gives them the capability of accurately tracking time. Thanks to the low cost GPS modules being available now a days, anyone with a GPS module with a price tag ranging between INR 2000 to INR 10000, can access this time source accurate to only a few microseconds. The GPS module for a PMU serves three purposes such as • After a successful fix with at least three satellites, the GPS module provides a Pulse per Second(PPS) signal, which is given to the GPSDO to generate the pulses to trigger sampling. • It also provides Universal Coordinated Time (UTC), reference time received from the satellites to the micro-controller according to which the calculated phasors are time stamped and can be synchronized irrespective of their origin and the time delay which may incur between their transmission and reception at the Phasor Data Concentrator Unit. • Since a GPS module can also report the Geographical co-ordinates,i.e. latitudes and longitudes of itself, this data can be transmitted to the PDC where the location of the PMU is mapped to the map of the grid, and it can show a clear picture of the health status of the grid in the geographical area. The operating voltage of GPS module used in this paper is 3.3V DC. The transmitter and receiver serial pins of the GPS module are connected to the phasor processor from which the UTC time,the Latitude and the Longitude are derived. The 1 PPS pin on the module is used by the GPS disciplined oscillator to generate the sampling initialization clocks. According to IEEE standard for phasor measurement, for a system with 50 Hz frequency it is desired that upto 50 phasors should be calculated per second. The primary objective of GPS disciplined oscillator is to take 1 PPS signal (generated from GPS unit) and generate predefined set of equidistant pulses per second. According to which the sampling is initiated and the phasors are calculated and time stamped. Here an Atmel ATmega328 AVR Microcontroller is used to generate the required number of pulses based on the 1 PPS signal from the GPS module at its pin Clock In. An Interrupt Service Routine (ISR) inside the microcontroller sets a flag upon receiving the 1 PPS signal. When this flag is set the microcontroller runs the code which first of all clears the flag and then generates pulses of 50 µs width, separated by 400 µs interval. This results in generation of 3200 pulses per second. These pulses are available at the Clock Out pin. The flow chart of the programmed GPS disciplined oscillator.

# Sampling and Time Stamping

The ARM Cortex M3 Microcontroller has a 12 Bit ADC with 12 available channels. The time taken by each conversion for three channels is 1.5 µs. The ADC conversion is started on reception of an interrupt from the GPS Disciplined Oscillator (GPSDO). Once the data is ready it is stored in a sixty-four element circular buffer. This conversion process and storing of the ADC value happens inside an Interrupt Service Routine (ISR) so as to allow

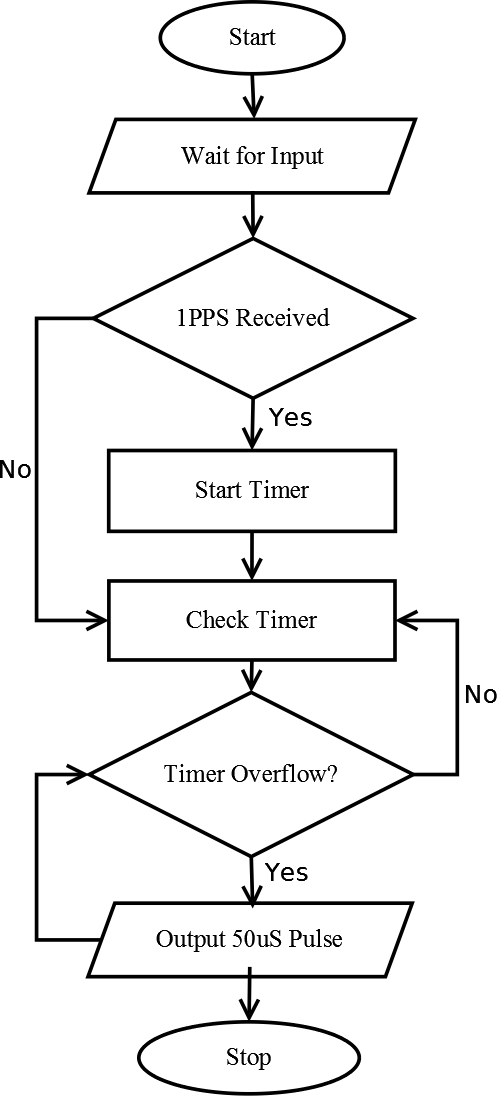


Fig 3.2 Flow Chart of the Programmed GPS Disciplined Oscillator

the micro-controller to perform other tasks while not using the ADC. Since the ADC in a Arduino Due microcontroller is a 12-bit ADC, it gives a reading of 0 to

4095 for an input voltage range of 0 volt to 3.3 volt. To map these readings with the actual measured values, it was first necessary to calculate the positive and negative peaks of the system. Then a continuous stream of ADC values were taken for about 10 complete cycles. The minimum and maximum of these values were found out. These values were mapped to the positive and negative voltage peaks of the AC signal being measured.

A local clock has been programmed in the microcontroller to keep its time synchronised with the UTC time reported by the GPS module read over UART. The time with a resolution of milliseconds is recorded at the time of phasor calculation, which is then added to the phasor information during transmission. This method is known as the Time-Stamping of a Phasor.

# Phasor Calculation

To implement the phasor calculation unit, the Arduino Due micro-controller with ARM Cortex M3 core has been used as the computational unit. Arduino Due has 32 bit ARM core micro-controller with 54 digital IO pins, 12 analog inputs, 4 UARTs and a 84MHz clock. It has 96 KB SRAM, 512 KB FLASH memory and a DMA controller. A 12 bit ADC is inbuilt with the micro-controller which can very easily operate at 3200 samples per sec. The conversion time of ADC is 4 µs. For a 3- phase system, the voltage samples are stored in the micro-controller’s RAM as a 64-element buffer which keeps updating every time a new sample comes, which is controlled by the GPSDO. A counter in the microcontroller keeps track of the number of accumulated samples. when this number reaches 64, i.e. a full cycle of signal is present in the buffer the phasor calculation task is initiated which is indicated by a flag and the counter is reset to Zero. When the flag to calculate the phasor is set, a 64 point DFT algorithm is used to calculate the three phasors. After calculation of phasor, the flag is reset and the time stamp is added to the calculated phasor and is transmitted to the display unit. The Phasor magnitudes, angles, frequencies, rate of change of frequencies, phasor time stamp and geographical coordinates are transmitted to the communication module. The communication module handles the PMU’s communication tasks, which is to report the Phasors to the display module for displaying them locally and also transmitting the phasors over a pre-selected channel to the local PDC. In this design the Phasors are transmitted over UART through a USB cable to a local PDC which has a python script running to capture the Phasors and further

process them. The algorithm of the Phasor calculation is shown in Figure 2.11 and the developed hardware is shown in Figure 2.12 .

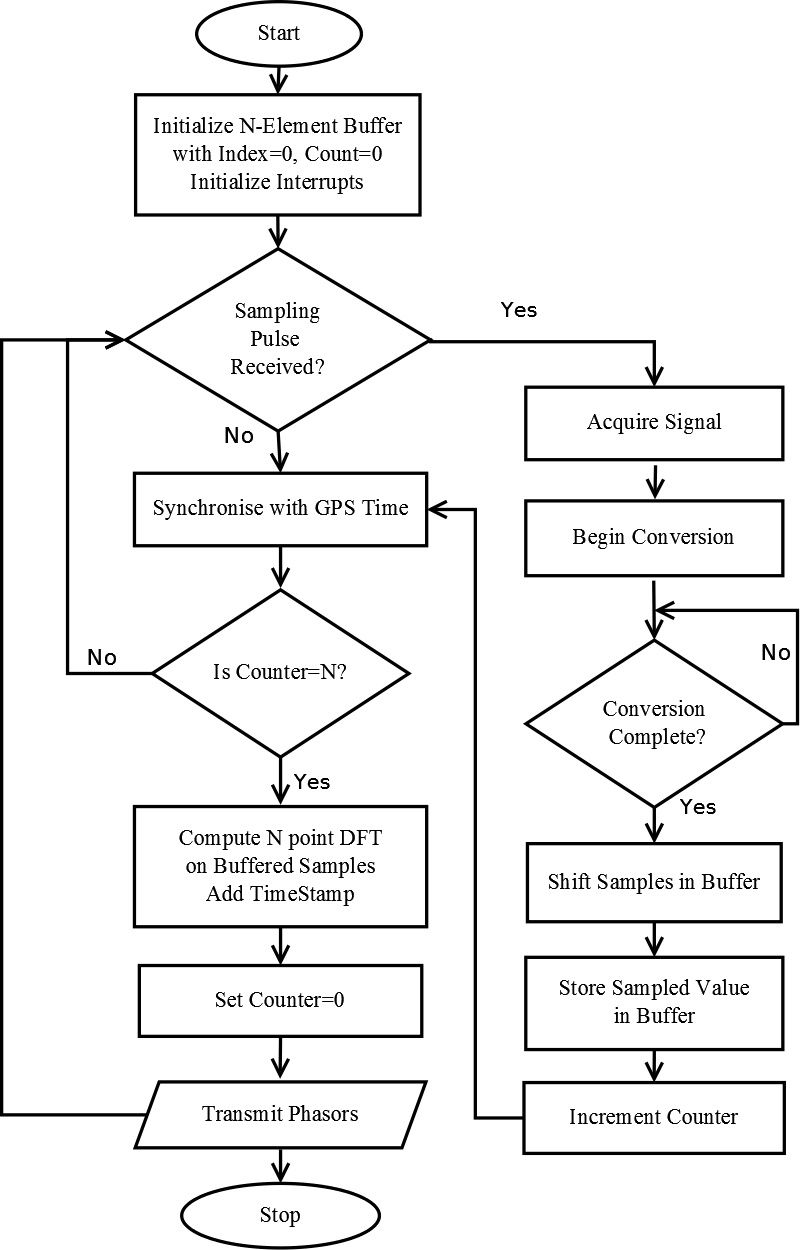


Figure 3.3: Flow Chart of the Phasor Processor

* 1. **Local Display Terminal**

Although, the Phasors are reported to the PDC at a rate of 25 or more phasors per second, a local terminal is needed to display the information for in-field debugging and verification by human operators. Since our eyes can not see the change of phasors if it keeps updating the display at the reporting rate, we need a more slower refresh rate like only one phasor per second which we can detect. For this purpose a local display terminal is built which consists of one Arduino Due microcontroller board and a 7 inch TFT LCD screen to display all the parameters of PMU. The Phasor microprocessor sends the data to be displayed by serial communication channel at 921600 baud. The arduino due microcontroller, which receives the data stream sent by the phasor processor, decodes the data, extracts the various parameters like the Phase voltages, angles, frequencies and ROCOFs. The microcontroller, then displays the data on the 7-inch LCD using UTFT library, upon receiving the same 1 PPS signal from the GPS module. Although, the displaying of the data gets triggered upon arrival of the 1 PPS signal, it takes about 300 ms to update the display completely.

# Power Supply Unit

A power supply unit was developed to satisfy the various power supply requirements by the different modules of the PMU. The required voltage supplies of the Hall Effect sensors are +15 Volt and -15 Volt, for the micro- controllers +3.3 Volt and +5 Volt, the LCD is +5 Volt, for the cooling Fan +12 Volt. All these voltage levels are provided by the Power Supply Unit (PSU) built with step-down transformers, diode bridge rectifiers and Low Drop Out (LDO) voltage regulators. The power requirement of the various modules is shown in Table 2.4.The components required for building the power supply module are listed in Table 2.5.A simple LDO based power supply unit has been designed. Description of the various sections of the power supply uni is given here.

Two step-down transformers have been used. One steps down the voltage from 230 Volt AC to 12 volt AC. This transformer is of 5 Ampere current rating, so as to provide enough power for all the modules of the PMU .The other steps down the voltage from 230 volt AC to 15 volt DC, and is of center tapped type, which is necessary to facilitate both positive and negative voltage for the dual power supply.

A full wave rectifier made of two diodes rectify the 12 volt AC of the center tapped transformer to give 12 volt DC as shown in Figure 2.13. Another full wave rectifier, a diode bridge rectifier made of four diodes is used with the 15 volt center tapped transformer to provide a positive and a negative power supply referenced to the center tapping of the transformer.

Smoothing capacitors are used throughout the power supply design to filter out the ripples present in the power supply after rectification. The capacitor charges up at the start of a positive half cycle which is available at the rectifier output, and discharges from the middle of the positive half cycle towards the end. The result is a smooth power delivered to the load even with the inherent pulsating nature of the DC available after the rectifier. Moreover, when a sufficiently large capacitor is used, it compensates for the sudden draw of current by some device in the circuit by providing the additional power from the charge stored in the capacitor rather than directly from the primary source, i.e. the transformer.

A voltage regulator eliminates any ripple present in the Voltage supply after the capacitor so that a maximum allowable ripple of 1% of the rated voltage is present at the output, i.e. the output is close to pure DC, suitable for the sensitive microcontrollers. The following are the LDOs used in this power supply to generate the various voltages required by the components of the PMU. The developed power supply module is shown in Figure2.15 • CD7805 for +5 volt at a maximum 1 ampere current. • CD7812 for +12 volt at a maximum 1 ampere current. • CD7815 for +15 volt at a maximum 1 ampere current. • CD7915 for - 15 volt at a maximum 800 milli ampere current. • LM317T adjustable voltage regulator for +3.3 volt at a maximum 1.5 ampere current.

# MATLAB Simulation and Result

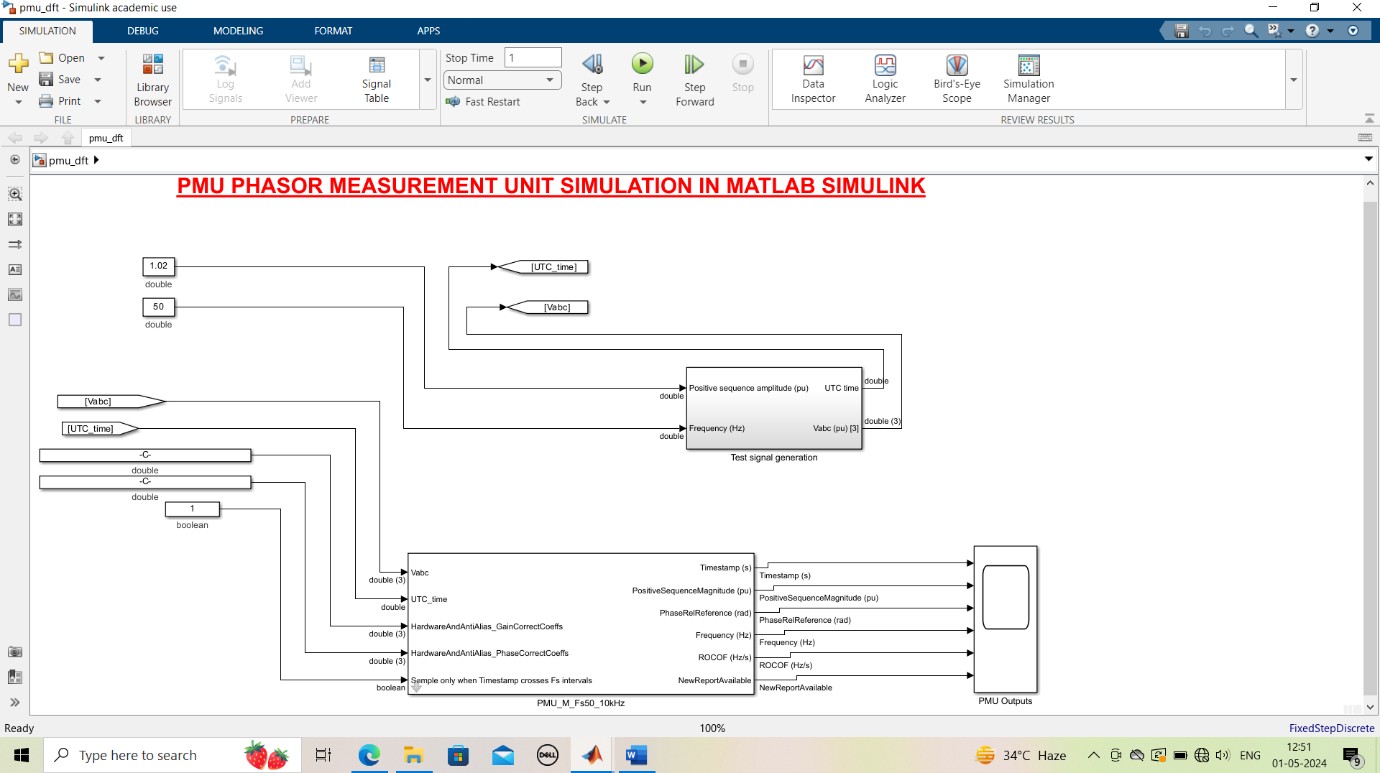


Fig 3.4 MATLAB Simulation

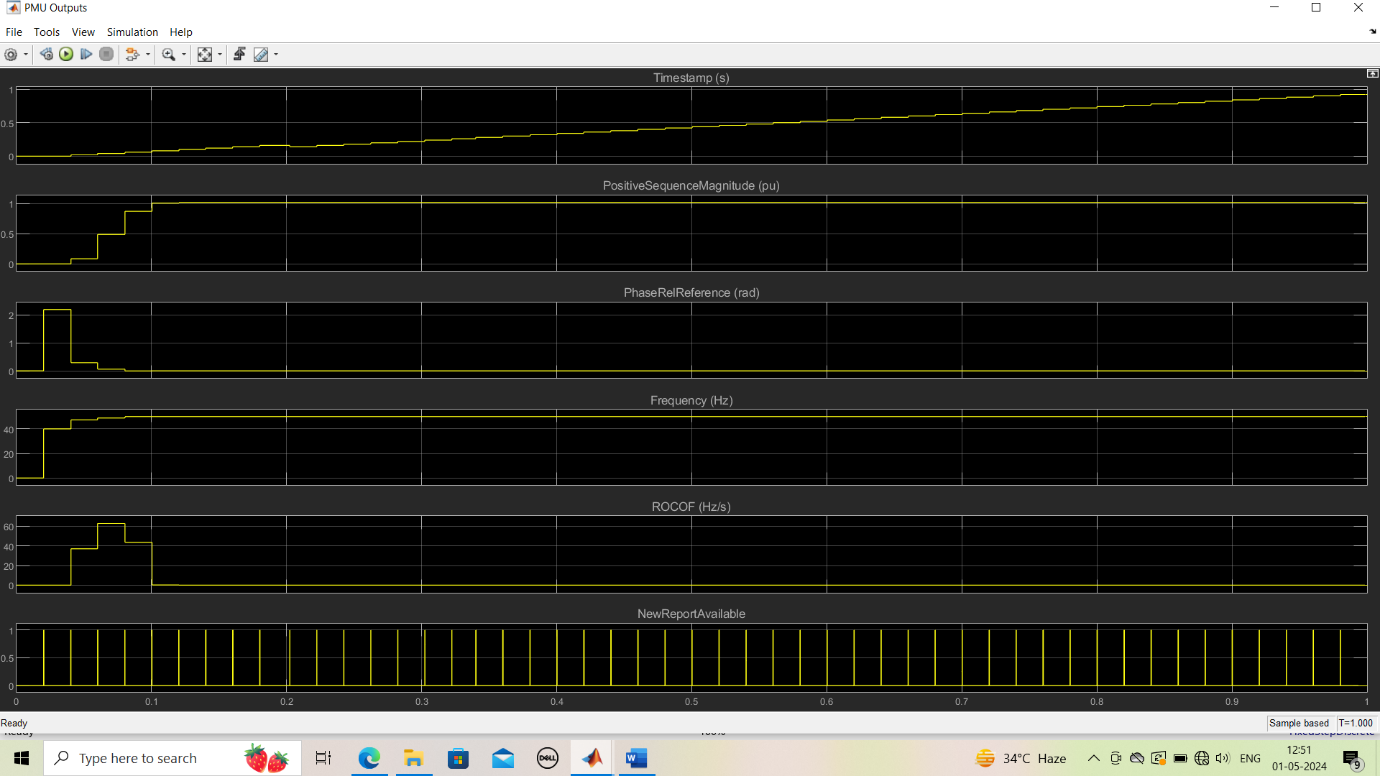


Fig 3.5 Simulation Result

# Components required to build Phasor Processor

|  |  |  |  |
| --- | --- | --- | --- |
| Quantity | Details | Unit Price (in Rs) | Line Total |
| 1 | Arduino Due Microcontroller Boards | 4500 | 4500 |
| 1 | Ublox Neo-6M GPS Module | 2800 | 2800 |
| 1 | Acitve GPS Antenna | 1000 | 1000 |
| 1 | 7-inch UTFT LCD with Shield for Due | 7000 | 7000 |
| 1 | GPSDO with Atmega328 uC | 300 | 300 |
| 1 | PCB, Solder, Jumper Wires etc. | 500 | 500 |
|  |  | **Net Total** | **16100** |

**Chapter 4**

# Conclusion

The Phasor Measurement Units are going to be the basic building blocks for monitoring the Smart Grid of the future. With the increase in the number of active PMUS in the Electric Grids day by day, the Real-Time monitoring of the Health of the Grid is going to be a reality sooner than expected. Since a lot of manufacturers are going to build their own versions of PMU, the much needed IEEE Standard C37.118.X.2011 is definitely a welcome guidance to make the different PMUs compatible with each other and with the PDC. The goal of developing a low cost PMU was not to compete with other manufacturers who provide commercial PMUs, but to facilitate the research in the academics and the R&D organizations which incorporates the data from the PMU to design and simulate the various projects related to the Electrical Grids. The objective was to simplify the hardware implementation process of Phasor measurements such that a PMU can be built using the many economic open hardware computing platforms available now a days. The Arduino Due (with ARM Cortex M3 Microcontroller) development board was chosen because of its userfriendly development environment both in terms of hardware and software. It facilitates the use of the device both by a beginner as well as an expert. Voltage phasors for the nominal frequency were found to be more accurate than phasors computed for off-nominal frequencies. This was because of the nature of the DFT and the sampling window used to calculate the phasors. However the reported error is well within the IEEE Standard compliance range of 1 % of the actual magnitude.

# 4.1 Proposed direction for Future Work

Step 1: Develop a Prototype PMU which is Cost Effective, but satisfies the IEEE Satndards for Phasor Measurements. A High Speed (250 Mega Samples Per Second), High Resolution (24 Bit), SPI Analog to digital Converter(ADC) will convert the input signals i.e. the three phase voltage and current signals and store the digital samples in a circular buffer. A High Speed FPGA (Spartan LX9 or better) will take/read these samples from the buffer into its own memory and save it as a 32 bit array. A DSP Processor developed in the FPGA Hardware will do the DFT/FFT (like Radix 4 or Radix 8 FFT) on these samples and report the phasor. The FPGA will output the various parameters such as Voltage and Current Phasors, Frequency, change of frequency (COF), Rate of change of frequency (ROCOF) and the Harmonic Components present in the Signals. The calculated values will be time stamped, with the time from a GPS Module, along with the Latitudes and longitudes and will be sent out through a Communication Interface (i.e. a GSM 3G Modem), which is handled by a High Speed (1 GHz or more) Single Board Computer with a High Performance Multicore Modern ARM Processor (Dual Core like A20 or Octa-core like A80) which will be a running a Real Time Operating System. The architecture of such a system is shown in Figure 4.1

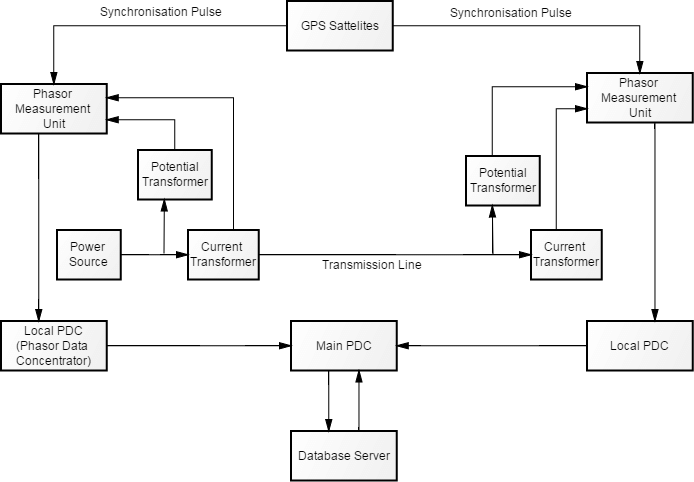


Figure 4.1: Proposed framework for testing the PMUs

Step 2: Develop a PMU Connection tester to test the PMU Which is installed in Field and is Remote A software will be developed which will integrate a data acquisition system to collect the data being transmitted from the PMU, a Graphical User Interface to display the data in a User Friendly Format. The software will also have a Database to store the PMU Data for future retrieval for analytical purposes. Then the software will be tested with the prototype PMU, and any necessary adjustments/developments will be done. Step 3: Development and Optimal Placement of Multiple Prototype PMUs. Four more Such Prototypes will be developed, and installed in the locality of the Institute in Remote locations, in different segments of the Electrical Grid. A block diagram of such a prototype grid for testing of the low cost PMUs is shown in Figure 4.2. Step 4: Integration of a Phasor Data Concentrator A Phasor Data Concentrator (PDC) will be either developed from scratch, or the existing OpenPDC Project will be used with little Modifications so as to make it compatible with the Developed Prototypes. The PDC will collect all the information from the Prototypes and display in the Graphical User Interface and store in a Database.

The work done for developing a low cost PMU has opened up new windows for developments. At present the hardware meets only one standard i.e. C37.118.1- 2011, and a way needs to be developed to meet the second part of the standard C37.118.2-2011 which describes how the PMUs must communicate.Also the method of acquiring the time from GPS and synchronizing the local clock according to it, needs more improvement so that the time stamps of the phasors can be made more accurate than it is. The method of using Open Hardware Platforms, not just limited to Arduino Due but Raspberry Pi, Beaglebone, Intel Galileo etc, for developing some real-time monitoring devices like a PMU, which has not just to be used in the laboratory but can be put to operation, is definitely a great venture because of the wide availability of such platforms and a huge community wherein everyone can contribute to the development of something important for the betterment of the society

# Refrence

1. IEEE Standard for Synchrophasor Measurements for Power Systems,IEEE Std. C37.118.1-2011. [Online]. Available: <http://standards.ieee.org/findstds/standard/C37.118.1-2011.html>
2. IEEE Standard for Synchrophasor Data Transfer for Power Systems,IEEE Std. C37.118.2- 2011. [Online]. Available: <http://standards.ieee.org/findstds/standard/C37.118.2->2011.html
3. G. Missout and P. Girard, Measurement of bus voltage angle between montreal and SEPT-ILES, IEEE Trans. Power App. Syst., vol. PAS-99, no. 2, pp. 536-539, Mar. 1980.
4. G. Missout, J. Beland, G. Bedard, and Y. Lafleur, Dynamic measurement of the absolute voltage angle on long transmission lines, IEEE Trans. Power App. Syst., vol. PAS-100, no. 11, pp. 4428-4434, Nov. 1981.
5. P. Bonanomi, Phase angle measurements with synchronized clocksprinciple and applications, IEEE Trans. Power App. Syst., vol. PAS-100, no. 12, pp. 5036- 5043, Dec. 1981.
6. A. G. Phadke and J. S. Thorp, History and applications of phasor measurements, in Proc. IEEE PES PSCE, 2006, pp. 331-335.
7. A. G. Phadke, Synchronized phasor measurementsA historical overview, in Proc. IEEE/PES Transmiss. Distrib. Conf. Exhib. Asia Pacific, Oct. 6-10, 2002, vol. 1, pp. 476-479.
8. A. G. Phadke and J. S. Thorp, Synchronized Phasor Measurements and Their Applications. New York, NY, USA: Springer-Verlag, 2008.
9. P. Denys, C. Counan, L. Hossenlopp, and C. Holweck, Measurement of voltage phase for the French future defence plan against losses of synchronism, IEEE Trans. Power Del., vol. 7, no. 1, pp. 62-69, Jan. 1992.
10. B. Kasztenny and M. Adamiak, Implementation and performance of synchrophasor function within microprocessor based relays, in Proc. 61st Annu. Georgia Tech. Protect. Relaying Conf., Atlanta, GA, USA, May 2-4, 2007, pp. 1-43.

# Appendix A

## Arduino Codes for Microcontrollers

* 1. **Arduino code for the GPS Disciplined Oscillator**

1. # include < TimerOne .h >
2. // use this library to handle the Timer functionality 3 const int sampling\_clock\_out\_pin = 9;

4 // the sampling pulses will be generated at this pin 5 void setup ()

6 {

1. pinMode (2 , INPUT ) ;
2. // This is the pin where the 1 PPS pulse from GPS module is connected
3. attachInterrupt (0 , pulsePPS , RISING ) ;// watch out for interrupt (1 PPS) on pin 2 10 // Timer1 . initialize (400) ;// for 2500 pulses per sec
4. Timer1 . initialize (312.5) ;// for 3200 pulses per sec
5. Timer1 . pwm ( sampling\_clock\_out\_pin , 100) ;// duty cycle of the pulse , i.e. about 100 uS 13 }
6. void loop ()
7. { /\* Since the microcontrollers timer operate independently without invoking the CPU , and the Interrupt handlers 16 takes care of the ISR , there is nothing to do in the loop \*/

17 }

18 void pulsePPS () // interrupt routine upon receiving PPS 19 {

1. Timer1 . restart () ;
2. /\* Just restart the timer , to keep it in sync with the GPS module ’s PPS pulses \*/

22 }

## Arduino code for the 3 Phase Signal Generator

1. # include < JeeLib .h >
2. # include < avr / pgmspace .h > 3

4 // Define the sine wave look -up tables which caontains the 8 -bit integers 5 byte sineR256 [] PROGMEM = {

6 128 ,131 ,134 ,137 ,140 ,143 ,146 ,149 ,152 ,155 ,158 ,162 ,165 ,167 ,170 ,173 ,

7 176 ,179 ,182 ,185 ,188 ,190 ,193 ,196 ,198 ,201 ,203 ,206 ,208 ,211 ,213 ,215 ,

8 218 ,220 ,222 ,224 ,226 ,228 ,230 ,232 ,234 ,235 ,237 ,238 ,240 ,241 ,243 ,244 ,

9 245 ,246 ,248 ,249 ,250 ,250 ,251 ,252 ,253 ,253 ,254 ,254 ,254 ,255 ,255 ,255 ,

10 255 ,255 ,255 ,255 ,254 ,254 ,254 ,253 ,253 ,252 ,251 ,250 ,250 ,249 ,248 ,246 ,

11 245 ,244 ,243 ,241 ,240 ,238 ,237 ,235 ,234 ,232 ,230 ,228 ,226 ,224 ,222 ,220 ,

12 218 ,215 ,213 ,211 ,208 ,206 ,203 ,201 ,198 ,196 ,193 ,190 ,188 ,185 ,182 ,179 ,

13 176 ,173 ,170 ,167 ,165 ,162 ,158 ,155 ,152 ,149 ,146 ,143 ,140 ,137 ,134 ,131 ,

14 128 ,124 ,121 ,118 ,115 ,112 ,109 ,106 ,103 ,100 ,97 ,93 ,90 ,88 ,85 ,82 ,79 ,76 ,

15 73 ,70 ,67 ,65 ,62 ,59 ,57 ,54 ,52 ,49 ,47 ,44 ,42 ,40 ,37 ,35 ,33 ,31 ,29 ,27 ,25 ,

16 23 ,21 ,20 ,18 ,17 ,15 ,14 ,12 ,11 ,10 ,9 ,7 ,6 ,5 ,5 ,4 ,3 ,2 ,2 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,

17 0 ,0 ,1 ,1 ,1 ,2 ,2 ,3 ,4 ,5 ,5 ,6 ,7 ,9 ,10 ,11 ,12 ,14 ,15 ,17 ,18 ,20 ,21 ,23 ,25 ,27 ,

18 29 ,31 ,33 ,35 ,37 ,40 ,42 ,44 ,47 ,49 ,52 ,54 ,57 ,59 ,62 ,65 ,67 ,70 ,73 ,76 ,79 ,

19 82 ,85 ,88 ,90 ,93 ,97 ,100 ,103 ,106 ,109 ,112 ,115 ,118 ,121 ,124};

20

21 byte sineY256 [] PROGMEM = { 22

238 ,237 ,235 , 234 ,232 ,230 ,228 ,226 ,224 ,222 ,220 ,218 ,215 ,213 ,211 ,

23 208 ,206 ,203 ,201 ,198 ,196 ,193 ,190 ,188 ,185 ,182 ,179 ,176 ,173 ,170 ,

24 167 ,165 ,162 ,158 ,155 ,152 ,149 ,146 ,143 ,140 ,137 ,134 ,131 ,128 ,124 ,

25 121 ,118 ,115 ,112 ,109 ,106 ,103 ,100 ,97 ,93 ,90 ,88 ,85 ,82 ,79 ,76 ,73 ,70 ,

26 67 ,65 ,62 ,59 ,57 ,54 ,52 ,49 ,47 ,44 ,42 ,40 ,37 ,35 ,33 ,31 ,29 ,27 ,25 ,23 ,21 ,

27 20 ,18 ,17 ,15 ,14 ,12 ,11 ,10 ,9 ,7 ,6 ,5 ,5 ,4 ,3 ,2 ,2 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,

28 1 ,1 ,1 ,2 ,2 ,3 ,4 ,5 ,5 ,6 ,7 ,9 ,10 ,11 ,12 ,14 ,15 ,17 ,18 ,20 ,21 ,23 ,25 ,27 ,29 ,

29 31 ,33 ,35 ,37 ,40 ,42 ,44 ,47 ,49 ,52 ,54 ,57 ,59 ,62 ,65 ,67 ,70 ,73 ,76 ,79 ,82 ,

30 85 ,88 ,90 ,93 ,97 ,100 ,103 ,106 ,109 ,112 ,115 ,118 ,121 ,124 ,128 ,131 ,134 ,

31 137 ,140 ,143 ,146 ,149 ,152 ,155 ,158 ,162 ,165 ,167 ,170 ,173 ,176 ,179 ,182 ,

32 185 ,188 ,190 ,193 ,196 ,198 ,201 ,203 ,206 ,208 ,211 ,213 ,215 ,218 ,220 ,222 ,

33 224 ,226 ,228 ,230 ,232 ,234 ,235 ,237 ,238 ,240 ,241 ,243 ,244 ,245 ,246 ,248 ,

34 249 ,250 ,250 ,251 ,252 ,253 ,253 ,254 ,254 ,254 ,255 ,255 ,255 ,255 ,255 ,255 ,

35 255 ,254 ,254 ,254 ,253 ,253 ,252 ,251 ,250 ,250 ,249 ,248 ,246 ,245 ,244 ,243 ,

36 241 ,240};

37

38 byte sineB256 [] PROGMEM = {

39 18 ,17 ,15 ,14 ,12 ,11 ,10 ,9 ,7 ,6 ,5 ,5 ,4 ,3 ,2 ,2 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,

40 1 ,2 ,2 ,3 ,4 ,5 ,5 ,6 ,7 ,9 ,10 ,11 ,12 ,14 ,15 ,17 ,18 ,20 ,21 ,23 ,25 ,27 ,29 ,31 ,

41 33 ,35 ,37 ,40 ,42 ,44 ,47 ,49 ,52 ,54 ,57 ,59 ,62 ,65 ,67 ,70 ,73 ,76 ,79 ,82 ,85 ,

42 88 ,90 ,93 ,97 ,100 ,103 ,106 ,109 ,112 ,115 ,118 ,121 ,124 ,128 ,131 ,134 ,137 ,

43 140 ,143 ,146 ,149 ,152 ,155 ,158 ,162 ,165 ,167 ,170 ,173 ,176 ,179 ,182 ,185 ,

44 188 ,190 ,193 ,196 ,198 ,201 ,203 ,206 ,208 ,211 ,213 ,215 ,218 ,220 ,222 ,224 ,

45 226 ,228 ,230 ,232 ,234 ,235 ,237 ,238 ,240 ,241 ,243 ,244 ,245 ,246 ,248 ,249 ,

46 250 ,250 ,251 ,252 ,253 ,253 ,254 ,254 ,254 ,255 ,255 ,255 ,255 ,255 ,255 ,255 ,

47 254 ,254 ,254 ,253 ,253 ,252 ,251 ,250 ,250 ,249 ,248 ,246 ,245 ,244 ,243 ,241 ,

48 240 ,238 ,237 ,235 ,234 ,232 ,230 ,228 ,226 ,224 ,222 ,220 ,218 ,215 ,213 ,211 ,

49 208 ,206 ,203 ,201 ,198 ,196 ,193 ,190 ,188 ,185 ,182 ,179 ,176 ,173 ,170 ,167 ,

50 165 ,162 ,158 ,155 ,152 ,149 ,146 ,143 ,140 ,137 ,134 ,131 ,128 ,124 ,121 ,118 ,

51 115 ,112 ,109 ,106 ,103 ,100 ,97 ,93 ,90 ,88 ,85 ,82 ,79 ,76 ,73 ,70 ,67 ,65 ,62 ,

52 59 ,57 ,54 ,

52 ,49 ,47 ,44 ,42 ,40 ,37 ,35 ,33 ,31 ,29 ,27 ,25 ,23 ,21 ,20};

53

54 void setup () { 55 for (int i =22; i

## Arduino code for temperature controlled regulated power supply

1. # include < OneWire .h >
2. # include < LiquidCrystal .h > 3 # include < FreqMeasure .h > 4

5 OneWire ds (3) ; // on pin 3 (a 4.7 K resistor is necessary ) 6 LiquidCrystal lcd ( A5 , A4 , A3 , A2 , A1 , A0 ) ;

1. int pwm\_fan = 6;
2. double sum = 0;
3. int count = 0 , rpm = 0;

10

11 void setup ( void ) { 12 lcd . begin (16 , 2) ;

13 Serial . begin (9600) ; 14 FreqMeasure . begin () ;

15 pinMode ( pwm\_fan , OUTPUT ) ; 16 }

17

18 void loop ( void ) { 19 byte i ;

20 byte present = 0; 21 byte type\_s ;

1. byte data [12];
2. byte addr [8];
3. float celsius , fahrenheit ; 25
4. if ( ! ds . search ( addr ) ) {
5. Serial . println ("No more addresses .") ; 28 Serial . println () ;

29 ds . reset\_search () ; 30 delay (250) ;

31 return ;

32 }

33

34 Serial . print ("ROM =") ; 35 for ( i = 0; i < 8; i ++) {

1. Serial . write (’ ’) ;
2. Serial . print ( addr [ i ] , HEX ) ; 38 }

39

40 if ( OneWire :: crc8 ( addr , 7) != addr [7]) { 41 Serial . println ("CRC is not valid !") ;

42 return ;

43 }

44 Serial . println () ; 45

46 // the first ROM byte indicates which chip 47 switch ( addr [0]) {

1. case 0 x10 :
2. Serial . println (" Chip = DS18S20 ") ; // or old DS1820 50 type\_s = 1; break ;
3. case 0 x28 :
4. Serial . println (" Chip = DS18B20 ") ; 53 type\_s = 0; break ;
5. case 0 x22 :
6. Serial . println (" Chip = DS1822 ") ; 56 type\_s = 0; break ;
7. default :
8. Serial . println (" Device is not a DS18x20 family device .") ; 59 return ;

60 }

61

1. ds . reset () ;
2. ds . select ( addr ) ;
3. ds . write (0 x44 , 1) ; // start conversion , with parasite power on at the end 65 delay (1000) ; // maybe 750 ms is enough , maybe not

66 // we might do a ds. depower () here , but the reset will take care of it. 67 present = ds . reset () ;

1. ds . select ( addr ) ;
2. ds . write (0 xBE ) ; // Read Scratchpad 70 Serial . print (" Data = ") ;

71 Serial . print ( present , HEX ) ; 72 Serial . print (" ") ;

73 for ( i = 0; i < 9; i ++) { // we need 9 bytes 74 data [ i ] = ds . read () ;

75 Serial . print ( data [ i ] , HEX ) ; 76 Serial . print (" ") ;

77 }

1. Serial . print (" CRC =") ;
2. Serial . print ( OneWire :: crc8 ( data , 8) , HEX ) ; 80 Serial . println () ;

81

82 /\* Convert the data to actual temperature because the result is a 16 bit signed integer , it should 83 be stored to an " int16\_t " type , which is always 16 bits even when compiled on a 32 bit processor .\*/

84 int16\_t raw = ( data [1] << 8) | data [0]; 85 if ( type\_s ) {

86 raw = raw << 3; // 9 bit resolution default 87 if ( data [7] == 0 x10 ) {

88 // " count remain " gives full 12 bit resolution 89 raw = ( raw & 0 xFFF0 ) + 12 - data [6];

90 }

1. } else {
2. byte cfg = ( data [4] & 0 x60 ) ;
3. // at lower res , the low bits are undefined , so let ’s zero them 94 if ( cfg == 0 x00 ) raw = raw & ~7; // 9 bit resolution , 93.75 ms 95 else if ( cfg == 0 x20 ) raw = raw & ~3; // 10 bit res , 187.5 ms 96 else if ( cfg == 0 x40 ) raw = raw & ~1; // 11 bit res , 375 ms

97 // default is 12 bit resolution , 750 ms conversion time 98 }

1. celsius = ( float ) raw / 16.0;
2. fahrenheit = celsius \* 1.8 + 32.0; 101 Serial . print (" Temperature = ") ; 102 Serial . print ( celsius ) ;

103 Serial . print (" Celsius , ") ; 104 Serial . print ( fahrenheit ) ;

105 Serial . println (" Fahrenheit ") ; 106 lcd . setCursor (0 , 0) ;

107 lcd . print (" Temp : ") ; 108 lcd . print ( celsius ) ; 109 lcd . print (" \*C") ;

110 if ( FreqMeasure . available () ) { 111 // average several reading together 112 sum = sum + FreqMeasure . read () ; 113 count = count + 1;

1. if ( count > 15) {
2. float frequency = FreqMeasure . countToFrequency ( sum / count ) ; 116 rpm = frequency \* (60 / 2) ;
3. sum = 0;
4. count = 0;
5. Serial . print ("Fan Speed ") ; 120 Serial . print ( rpm ) ;

121 Serial . println ("RPM ") ; 122 // rpm = 0;

123 }

124 }

1. int fan\_speed = map ( celsius , 20 , 45 , 0 , 255) ;
2. if ( fan\_speed < 0)

127 {

128 fan\_speed = 0;

129 }

130 else if ( fan\_speed > 255) 131 {

132 fan\_speed = 255;

133 }

134 else

135 {}

1. analogWrite ( pwm\_fan , fan\_speed ) ;
2. int desired\_rpm = map ( fan\_speed , 0 , 255 , 800 , 2200) ;
3. lcd . setCursor (0 , 1) ; 139 lcd . print (" Speed : ") ; 140 lcd . print ( rpm ) ;

141 lcd . print (" RPM") ; 142 }

## Arduino code for Phasor estimation using 64-Point DFT

1. # include < Time .h > // Time Library
2. # include < TinyGPS ++. h > // GPS Library
3. # include < math .h > // Math functions library 4

5 static const uint32\_t GPSBaud = 38400; 6 boolean Calculate\_A\_Phasor = false ;

7 boolean get\_time\_on\_pps = false ; 8 // The TinyGPS ++ object

9 TinyGPSPlus gps ; 10

11 // Serial connection to the GPS device 12 # define Serial\_GPS Serial3

13 # define SerialTx Serial2 14 //# define SerialTx Serial 15

16 time\_t prevDisplay = 0; // Count for when time last displayed 17 int Year ;

18 byte Month ; 19 byte Day ; 20 byte Hour ;

21 byte Minute ; 22 byte Second ; 23

1. // Phasor Estimation Variable Declaration
2. # define WindowSize 64 //i.e. 64 samples per second 26 int N = WindowSize ; // Sampling frequency 3200 Hz 27 long double pi = 3.143;

28

29 long double adc\_out\_1 [ WindowSize ] , values\_1 [ WindowSize ]; 30 long double adc\_out\_2 [ WindowSize ] , values\_2 [ WindowSize ];

31 long double adc\_out\_3 [ WindowSize ] , values\_3 [ WindowSize ]; 32

33 long double Xi\_1 , Xr\_1 , Phasor\_Magnitude\_1 , Phasor\_Angle\_1 , Phasor\_Angle\_Degree\_1 ; 34 long double Xi\_2 , Xr\_2 , Phasor\_Magnitude\_2 , Phasor\_Angle\_2 , Phasor\_Angle\_Degree\_2 ; 35 long double Xi\_3 , Xr\_3 , Phasor\_Magnitude\_3 , Phasor\_Angle\_3 , Phasor\_Angle\_Degree\_3 ; 36

37 unsigned long int calculation\_start\_millis ; 38 unsigned long int calculation\_finish\_millis ; 39

1. unsigned long int pps\_time\_millis ;
2. unsigned long int phasor\_stamp\_millis ; 42
3. // variables for frequency calculation
4. volatile long double P1\_start\_micros = 0 , last\_P1\_start\_micros = 0 , P1\_period = 0; 45 volatile long double P2\_start\_micros = 0 , last\_P2\_start\_micros = 0 , P2\_period = 0; 46 volatile long double P3\_start\_micros = 0 , last\_P3\_start\_micros = 0 , P3\_period = 0; 47
5. int P1\_freq , P2\_freq , P3\_freq ;
6. long double P1\_freqf , P2\_freqf , P3\_freqf ; 50 int P1\_lf , P2\_lf , P3\_lf ;// last frequencies

51 int P1\_rocof , P2\_rocof , P3\_rocof ;// rate of change of frequency df/dt 52 long double P1\_rocoff , P2\_rocoff , P3\_rocoff ;

53

54 void setup () 55 { adc\_setup () ;

1. SerialTx . begin (921600) ;// for transmitting Phasors
2. Serial\_GPS . begin ( GPSBaud ) ; // Start GPS Serial Connection 58 smartDelay (1000) ;

59 delay (2000) ;

1. analogReadResolution (12) ;
2. attachInterrupt (22 , aquire , RISING ) ;// aquire voltage samples
3. attachInterrupt (23 , attach\_pps\_time , RISING ) ;// get pulse per second time
4. attachInterrupt (31 , capture\_P1\_start , RISING ) ;// get starting time of P1 waveform 64 attachInterrupt (33 , capture\_P2\_start , RISING ) ;// """""""""""""""""""" P2 waveform

65 attachInterrupt (29 , capture\_P3\_start , RISING ) ;// """""""""""""""""""" P3 waveform 66 }

67 // Get start Time of waves for calculation of frequency 68 void capture\_P1\_start () {

1. P1\_start\_micros = micros () ;
2. P1\_period = P1\_start\_micros - last\_P1\_start\_micros ; 71 last\_P1\_start\_micros = P1\_start\_micros ;

72 }

1. void capture\_P2\_start () {
2. P2\_start\_micros = micros () ;
3. P2\_period = P2\_start\_micros - last\_P2\_start\_micros ; 76 last\_P2\_start\_micros = P2\_start\_micros ;

77 }

1. void capture\_P3\_start () {
2. P3\_start\_micros = micros () ;
3. P3\_period = P3\_start\_micros - last\_P3\_start\_micros ; 81 last\_P3\_start\_micros = P3\_start\_micros ; 82 }
4. // Circular buffer , power of two .
5. # define BUFSIZE 0 x40 // 64 samples buffer 85 # define BUFMASK 0 x3F

86 volatile int R [ BUFSIZE ] ; 87 volatile int Y [ BUFSIZE ] ; 88 volatile int B [ BUFSIZE ] ; 89 volatile int sptr = 0 ;

90 volatile int isr\_count = 0 ; 91

1. void aquire () {
2. ADC - > ADC\_CR |= 0 b10 ; // start conversion
3. while (!( ADC - > ADC\_ISR & 0 b11100000 ) ) ; // wait for conversion to end
4. int Rval = ADC - > ADC\_CDR [7];
5. int Yval = ADC - > ADC\_CDR [6];
6. int Bval = ADC - > ADC\_CDR [5];

98

99 R [ sptr ] = Rval ; 100 Y [ sptr ] = Yval ; 101 B [ sptr ] = Bval ;

102 sptr = ( sptr + 1) & BUFMASK ; 103 isr\_count ++ ;

104 }

105

106 void adc\_setup () 107 {

1. // ADC setup
2. ADC - > ADC\_WPMR &= 0 xFFFE ; // disable write protect
3. ADC - > ADC\_CHER = 0 b11100000 ; // Enable AD7 ,AD6 ,AD5 or CH7 ,Ch6 ,Ch5 or PA16 ,PA24 ,

PA23 or A0 ,A1 and A2 |

111 ADC - > ADC\_MR &= 0 b11111111000000000000011100000000 ;// Fast i.e. about 4mS for 2500

Conversions on three channels

112 ADC - > ADC\_MR |= 0 b00000000000100100000000000000000 ; // software trigger , hi res , no sleep , not free running

1. ADC - > ADC\_IER = 0 b11100000 ; // enable interrupt
2. ADC - > ADC\_IMR = 0 b11100000 ; // enable interrupt in mask 115 ADC - > ADC\_CR |= 0 b10 ; // start first conversion

116 }

117

118 void loop () 119 {

120 if ( get\_time\_on\_pps == true ) 121 {

1. pps\_time\_millis = millis () ;
2. GPS\_Timezone\_Adjust () ; // Call Time Adjust Function
3. get\_time\_on\_pps = false ; 125 }

126 if ( isr\_count == 64)

127 {

128 Calculate\_A\_Phasor = true ; 129 isr\_count = 0;

130 }

131 if ( Calculate\_A\_Phasor == true ) 132 {

1. calc\_phasor () ;
2. transmit\_phasors\_on\_SerialTx () ; 135 Calculate\_A\_Phasor = false ;

136 }

137 smartDelay (0) ;

138 }

139

140 void attach\_pps\_time () 141 {

142 get\_time\_on\_pps = true ; 143 }

144

145 void GPS\_Timezone\_Adjust () { 146

1. Year = gps . date . year () ;
2. Month = gps . date . month () ; 149 Day = gps . date . day () ;
3. Hour = gps . time . hour () ;
4. Minute = gps . time . minute () ; 152 Second = gps . time . second () ; 153
5. // Set Time from GPS data string
6. setTime ( Hour , Minute , Second , Day , Month , Year ) ; 156 // Calc current Time Zone time by offset value

157

158 if ( timeStatus () != timeNotSet ) { 159 if ( now () != prevDisplay ) {

160 prevDisplay = now () ; 161 }

162 }

163 smartDelay (0) ;

164 }

165

166 static void smartDelay ( unsigned long ms ) 167 {

168 unsigned long start = millis () ; 169 do

170 {

171 while ( Serial\_GPS . available () ) 172 gps . encode ( Serial\_GPS . read () ) ; 173 } while ( millis () - start < ms ) ;

174 }

175

176 // Phasor calculation function 177 void calc\_phasor () {

178 // copy buffer to SampleWindow for calculation for (int i = 0; i < 64; i ++)

180 {

181 adc\_out\_1 [ i ] = R [ i ]; 182 adc\_out\_2 [ i ] = Y [ i ]; 183 adc\_out\_3 [ i ] = B [ i ]; 184 }

1. calculation\_start\_millis = millis () ;
2. for (int i = 0; i < N ; i ++) 187 {

188 values\_1 [ i ] = map\_double ( adc\_out\_1 [ i ] , 1433 , 2812 , -347.25 , 347.25) ;// Phase A 189 values\_2 [ i ] = map\_double ( adc\_out\_2 [ i ] , 1498 , 2859 , -336.78 , 336.78) ;// Phase B 190 values\_3 [ i ] = map\_double ( adc\_out\_3 [ i ] , 1408 , 2851 , -344.70 , 344.70) ;// Phase C 191 smartDelay (0) ;

192 }

193

194 // Calculate 64 - Point DFT 195 Xr\_1 = 0; Xr\_2 = 0; Xr\_3 = 0;

196 Xi\_1 = 0; Xi\_2 = 0; Xi\_3 = 0;

1. Phasor\_Magnitude\_1 = 0; Phasor\_Magnitude\_2 = 0; Phasor\_Magnitude\_3 = 0;
2. Phasor\_Angle\_1 = 0; Phasor\_Angle\_2 = 0; Phasor\_Angle\_3 = 0; 199 for (int n = 0; n < N ; n ++)

200 {

201 Xr\_1 = Xr\_1 + values\_1 [ n ] \* cos ((2 \* pi \* n ) / N ) ; 202 Xi\_1 = Xi\_1 + values\_1 [ n ] \* sin ((2 \* pi \* n ) / N ) ; 203

204 Xr\_2 = Xr\_2 + values\_2 [ n ] \* cos ((2 \* pi \* n ) / N ) ; 205 Xi\_2 = Xi\_2 + values\_2 [ n ] \* sin ((2 \* pi \* n ) / N ) ; 206

207 Xr\_3 = Xr\_3 + values\_3 [ n ] \* cos ((2 \* pi \* n ) / N ) ; 208 Xi\_3 = Xi\_3 + values\_3 [ n ] \* sin ((2 \* pi \* n ) / N ) ; 209 smartDelay (0) ;

210 }

211

212 Xr\_1 = ( sqrt (2) / N ) \* Xr\_1 ;

213 Xr\_2 = ( sqrt (2) / N ) \* Xr\_2 ;

214 Xr\_3 = ( sqrt (2) / N ) \* Xr\_3 ;

215

216 Xi\_1 = -( sqrt (2) / N ) \* Xi\_1 ;

217 Xi\_2 = -( sqrt (2) / N ) \* Xi\_2 ;

218 Xi\_3 = -( sqrt (2) / N ) \* Xi\_3 ;

219

220 Phasor\_Magnitude\_1 = sqrt ( Xr\_1 \* Xr\_1 + Xi\_1 \* Xi\_1 ) ; 221 Phasor\_Magnitude\_2 = sqrt ( Xr\_2 \* Xr\_2 + Xi\_2 \* Xi\_2 ) ; 222 Phasor\_Magnitude\_3 = sqrt ( Xr\_3 \* Xr\_3 + Xi\_3 \* Xi\_3 ) ; 223

1. Phasor\_Angle\_1 = atan2 ( Xi\_1 , Xr\_1 ) ; // double atan2 ( double y, double x)
2. Phasor\_Angle\_2 = atan2 ( Xi\_2 , Xr\_2 ) ; // The atan2 () function returns the arc tangent of y/x, in the range [ -pi , +pi] radians .
3. Phasor\_Angle\_3 = atan2 ( Xi\_3 , Xr\_3 ) ; 227
4. // Calculate Phasor Angle in Degree
5. Phasor\_Angle\_Degree\_1 = ( Phasor\_Angle\_1 \* 4068) / 71;
6. Phasor\_Angle\_Degree\_2 = ( Phasor\_Angle\_2 \* 4068) / 71;
7. Phasor\_Angle\_Degree\_3 = ( Phasor\_Angle\_3 \* 4068) / 71;

232 //

1. calculation\_finish\_millis = millis () ;
2. phasor\_stamp\_millis = calculation\_start\_millis - pps\_time\_millis ; 235
3. // Calculate frequency
4. // long int P1\_period = P1\_end\_micros - P1\_start\_micros ; 238 P1\_freqf = 1000000 / P1\_period ;

239 P2\_freqf = 1000000 / P2\_period ; 240 P3\_freqf = 1000000 / P3\_period ; 241

242 P1\_rocoff = sqrt (( P1\_freqf - 50.00) \* ( P1\_freqf - 50.00) ) \* 50; 243 P2\_rocoff = sqrt (( P2\_freqf - 50.00) \* ( P2\_freqf - 50.00) ) \* 50; 244 P3\_rocoff = sqrt (( P3\_freqf - 50.00) \* ( P3\_freqf - 50.00) ) \* 50; 245

246 smartDelay (0) ;

247 }

248

249 float map\_double ( double x , double in\_min , double in\_max , double out\_min , double out\_max )

250 {

251 return ( x - in\_min ) \* ( out\_max - out\_min ) / ( in\_max - in\_min ) + out\_min ; 252 smartDelay (0) ;

253 }

1. void transmit\_phasors\_on\_SerialTx () {
2. SerialTx . write (’!’) ;
3. SerialTx . print (int( Phasor\_Magnitude\_1 \* 100) ) ;
4. SerialTx . write (’"’) ;
5. SerialTx . print (int( Phasor\_Magnitude\_2 \* 100) ) ;
6. SerialTx . write (’#’) ;
7. SerialTx . print (int( Phasor\_Magnitude\_3 \* 100) ) ;
8. SerialTx . write (’$’) ;
9. // Calculate angle i.e. 2pi ’s complement to be sent
10. if ( Phasor\_Angle\_1 < 0)
11. Phasor\_Angle\_1 = Phasor\_Angle\_1 + 6.286;
12. if ( Phasor\_Angle\_2 < 0)
13. Phasor\_Angle\_2 = Phasor\_Angle\_2 + 6.286;
14. if ( Phasor\_Angle\_3 < 0)
15. Phasor\_Angle\_3 = Phasor\_Angle\_3 + 6.286;
16. SerialTx . print (int( Phasor\_Angle\_1 \* 1000) ) ;
17. SerialTx . write (’%’) ;
18. SerialTx . print (int( Phasor\_Angle\_2 \* 1000) ) ;
19. SerialTx . write (’&’) ;
20. SerialTx . print (int( Phasor\_Angle\_3 \* 1000) ) ;
21. SerialTx . write (’(’) ;
22. SerialTx . print ( day () ) ;
23. SerialTx . write (’)’) ;
24. SerialTx . print ( month () ) ;
25. SerialTx . write (’\*’) ;
26. SerialTx . print ( year () ) ; 283
27. SerialTx . write (’+’) ;
28. SerialTx . print ( hour () ) ;
29. SerialTx . write (’,’) ;
30. SerialTx . print ( minute () ) ;
31. SerialTx . write (’-’) ;
32. SerialTx . print ( second () ) ;
33. SerialTx . write (’.’) ;
34. SerialTx . print ( phasor\_stamp\_millis ) ; 292 SerialTx . write (’/’) ;

293

294 float Latitude = ( gps . location . lat () ) ; 295 float Longitude = ( gps . location . lng () ) ; 296 SerialTx . print (int( Latitude \* 1000) ) ; 297 SerialTx . write (’:’) ;

1. SerialTx . print (int( Longitude \* 1000) ) ;
2. SerialTx . write (’;’) ;
3. // Transmit Frequencies
4. SerialTx . print (int( P1\_freqf \* 100) ) ;
5. SerialTx . write (’@’) ;
6. SerialTx . print (int( P2\_freqf \* 100) ) ;
7. SerialTx . write (’^’) ;
8. SerialTx . print (int( P3\_freqf \* 100) ) ;
9. SerialTx . write (’?’) ;
10. // Transmit ROCOF
11. SerialTx . print (int( P1\_rocoff \* 100) ) ;
12. SerialTx . write (’[’) ;
13. SerialTx . print (int( P2\_rocoff \* 100) ) ;
14. SerialTx . write (’]’) ;
15. SerialTx . print (int( P3\_rocoff \* 100) ) ;
16. SerialTx . write (’|’) ;

317 }

# Arduino code for PMU communication unit

1. # define IDLE 0
2. # define RECEIVING1 1
3. # define RECEIVING2 2
4. # define RECEIVING3 3
5. # define RECEIVING4 4
6. # define RECEIVING5 5
7. # define RECEIVING6 6
8. # define RECEIVING7 7
9. # define RECEIVING8 8
10. # define RECEIVING9 9
11. # define RECEIVING10 10
12. # define RECEIVING11 11
13. # define RECEIVING12 12
14. # define RECEIVING13 13
15. # define RECEIVING14 14
16. # define RECEIVING15 15
17. # define RECEIVING16 16
18. # define RECEIVING17 17
19. # define RECEIVING18 18
20. # define RECEIVING19 19
21. # define RECEIVING20 20
22. # define RECEIVING21 21

23

1. int Year , Month , Day ;
2. int Hour , Minute , Second , MilliSecond ; 26

27 float Phasor\_Magnitude\_1 , Phasor\_Magnitude\_2 , Phasor\_Magnitude\_3 ; 28

29 float Phasor\_Angle\_1 , Phasor\_Angle\_Degree\_1 ; 30 float Phasor\_Angle\_2 , Phasor\_Angle\_Degree\_2 ;

31 float Phasor\_Angle\_3 , Phasor\_Angle\_Degree\_3 ; 32

33 float Latitude ; 34 float Longitude ; 35

1. float P1\_freq , P2\_freq , P3\_freq ;
2. float P1\_rocof , P2\_rocof , P3\_rocof ; 38
3. byte status = IDLE ;
4. # define SerialRx Serial2 41 # define SerialLCD Serial3 42 void setup () {
5. SerialLCD . begin (921600) ;
6. SerialRx . begin (921600) ;
7. Serial . begin (460800) ;
8. // Serial . println (" Ready "); 47 }

48 int count = 0; 49 void loop () { 50

51 if ( SerialRx . available () ) { 52 int c = SerialRx . read () ;

53 if ( status == RECEIVING1 && c >= ’0’ && c <= ’9’) {

1. Phasor\_Magnitude\_1 = Phasor\_Magnitude\_1 \* 10 + ( c - ’0’) ;
2. } else if ( status == RECEIVING2 && c >= ’0’ && c <= ’9’) {
3. Phasor\_Magnitude\_2 = Phasor\_Magnitude\_2 \* 10 + ( c - ’0’) ; 57 } else if ( status == RECEIVING3 && c >= ’0’ && c <= ’9’) {

58 Phasor\_Magnitude\_3 = Phasor\_Magnitude\_3 \* 10 + ( c - ’0’) ; 59 } else if ( status == RECEIVING4 && c >= ’0’ && c <= ’9’) {

1. Phasor\_Angle\_1 = Phasor\_Angle\_1 \* 10 + ( c - ’0’) ;
2. } else if ( status == RECEIVING5 && c >= ’0’ && c <= ’9’) {
3. Phasor\_Angle\_2 = Phasor\_Angle\_2 \* 10 + ( c - ’0’) ;
4. } else if ( status == RECEIVING6 && c >= ’0’ && c <= ’9’) {
5. Phasor\_Angle\_3 = Phasor\_Angle\_3 \* 10 + ( c - ’0’) ;
6. } else if ( status == RECEIVING7 && c >= ’0’ && c <= ’9’) {

66 Day = Day \* 10 + ( c - ’0’) ;

67 } else if ( status == RECEIVING8 && c >= ’0’ && c <= ’9’) {

68 Month = Month \* 10 + ( c - ’0’) ;

69 } else if ( status == RECEIVING9 && c >= ’0’ && c <= ’9’) {

70 Year = Year \* 10 + ( c - ’0’) ;

71 } else if ( status == RECEIVING10 && c >= ’0’ && c <= ’9’) {

72 Hour = Hour \* 10 + ( c - ’0’) ;

1. } else if ( status == RECEIVING11 && c >= ’0’ && c <= ’9’) {
2. Minute = Minute \* 10 + ( c - ’0’) ;
3. } else if ( status == RECEIVING12 && c >= ’0’ && c <= ’9’) {
4. Second = Second \* 10 + ( c - ’0’) ;
5. } else if ( status == RECEIVING13 && c >= ’0’ && c <= ’9’) {
6. MilliSecond = MilliSecond \* 10 + ( c - ’0’) ;
7. } else if ( status == RECEIVING14 && c >= ’0’ && c <= ’9’) {
8. Latitude = Latitude \* 10 + ( c - ’0’) ;
9. } else if ( status == RECEIVING15 && c >= ’0’ && c <= ’9’) {
10. Longitude = Longitude \* 10 + ( c - ’0’) ;
11. } else if ( status == RECEIVING16 && c >= ’0’ && c <= ’9’) {

84 P1\_freq = P1\_freq \* 10 + ( c - ’0’) ;

85 } else if ( status == RECEIVING17 && c >= ’0’ && c <= ’9’) {

86 P2\_freq = P2\_freq \* 10 + ( c - ’0’) ;

87 } else if ( status == RECEIVING18 && c >= ’0’ && c <= ’9’) {

88 P3\_freq = P3\_freq \* 10 + ( c - ’0’) ;

89 } else if ( status == RECEIVING19 && c >= ’0’ && c <= ’9’) {

90 P1\_rocof = P1\_rocof \* 10 + ( c - ’0’) ;

91 } else if ( status == RECEIVING20 && c >= ’0’ && c <= ’9’) {

92 P2\_rocof = P2\_rocof \* 10 + ( c - ’0’) ;

93 } else if ( status == RECEIVING21 && c >= ’0’ && c <= ’9’) {

94 P3\_rocof = P3\_rocof \* 10 + ( c - ’0’) ;

95 }

96

1. else if ( status == RECEIVING1 && c == ’"’) {
2. status = RECEIVING2 ;
3. } else if ( status == RECEIVING2 && c == ’#’) {
4. status = RECEIVING3 ;
5. } else if ( status == RECEIVING3 && c == ’$’) {
6. status = RECEIVING4 ;
7. } else if ( status == RECEIVING4 && c == ’%’) {
8. status = RECEIVING5 ;
9. } else if ( status == RECEIVING5 && c == ’&’) {
10. status = RECEIVING6 ;
11. } else if ( status == RECEIVING6 && c == ’(’) {
12. status = RECEIVING7 ;
13. } else if ( status == RECEIVING7 && c == ’)’) {
14. status = RECEIVING8 ;
15. } else if ( status == RECEIVING8 && c == ’\*’) {
16. status = RECEIVING9 ;
17. } else if ( status == RECEIVING9 && c == ’+’) {
18. status = RECEIVING10 ;
19. } else if ( status == RECEIVING10 && c == ’,’) {
20. status = RECEIVING11 ;
21. } else if ( status == RECEIVING11 && c == ’-’) {
22. status = RECEIVING12 ;
23. } else if ( status == RECEIVING12 && c == ’.’) {
24. status = RECEIVING13 ;
25. } else if ( status == RECEIVING13 && c == ’/’) {
26. status = RECEIVING14 ;
27. } else if ( status == RECEIVING14 && c == ’:’) {
28. status = RECEIVING15 ;
29. } else if ( status == RECEIVING15 && c == ’;’) {
30. status = RECEIVING16 ;
31. } else if ( status == RECEIVING16 && c == ’@’) {
32. status = RECEIVING17 ;
33. } else if ( status == RECEIVING17 && c == ’^’) {
34. status = RECEIVING18 ;
35. } else if ( status == RECEIVING18 && c == ’?’) {
36. status = RECEIVING19 ;
37. } else if ( status == RECEIVING19 && c == ’[’) {
38. status = RECEIVING20 ;
39. } else if ( status == RECEIVING20 && c == ’]’) {
40. status = RECEIVING21 ;

137 } else if ( c == ’|’) { 138 status = IDLE ; 139

140 // Remote value Received completely , Now display it 141

1. // Calculate Phasor Angles into Float
2. Phasor\_Angle\_1 = Phasor\_Angle\_1 / 1000;
3. Phasor\_Angle\_2 = Phasor\_Angle\_2 / 1000;
4. Phasor\_Angle\_3 = Phasor\_Angle\_3 / 1000;
5. // Calculate Phasor Angle using reverse 2pi ’s complement
6. if ( Phasor\_Angle\_1 > 3.143)
7. Phasor\_Angle\_1 = Phasor\_Angle\_1 - 6.286;
8. if ( Phasor\_Angle\_2 > 3.143)
9. Phasor\_Angle\_2 = Phasor\_Angle\_2 - 6.286;
10. if ( Phasor\_Angle\_3 > 3.143)
11. Phasor\_Angle\_3 = Phasor\_Angle\_3 - 6.286;
12. // Calculate Angles in Degrees
13. Phasor\_Angle\_Degree\_1 = ( Phasor\_Angle\_1 \* 4068) / 71;
14. Phasor\_Angle\_Degree\_2 = ( Phasor\_Angle\_2 \* 4068) / 71;
15. Phasor\_Angle\_Degree\_3 = ( Phasor\_Angle\_3 \* 4068) / 71;
16. Display\_Phasors\_on\_Serial\_Terminal () ; 161 transmit\_phasors\_LCD () ;

162

163 } else if ( c == ’!’) {

1. status = RECEIVING1 ;
2. // Reset the variables to Zero 166 Year = 0;
3. Month = 0;
4. Day = 0;
5. Hour = 0;
6. Minute = 0;
7. Second = 0;
8. MilliSecond = 0;
9. Phasor\_Angle\_1 = 0;
10. Phasor\_Angle\_2 = 0;
11. Phasor\_Angle\_3 = 0;
12. Phasor\_Magnitude\_1 = 0;
13. Phasor\_Magnitude\_2 = 0;
14. Phasor\_Magnitude\_3 = 0;
15. Phasor\_Angle\_Degree\_1 = 0;
16. Phasor\_Angle\_Degree\_2 = 0;
17. Phasor\_Angle\_Degree\_3 = 0;
18. Latitude = 0;
19. Longitude = 0;
20. P1\_freq = 0;
21. P2\_freq = 0;
22. P3\_freq = 0;
23. P1\_rocof = 0;
24. P2\_rocof = 0;
25. P3\_rocof = 0;

190 }

191 }

192 }

193 // Transmit the phasors to local PDC , where it can be plotted in real - time 194 void Display\_Phasors\_on\_Serial\_Terminal () {

195 // Serial . print ( millis ()); 196 Serial . print ( Day ) ;

1. Serial . print (" ") ;
2. Serial . print ( Month ) ; 199 Serial . print (" ") ;

200 Serial . print ( Year ) ; 201 Serial . print (" ") ; 202 Serial . print ( Hour ) ; 203 Serial . print (" ") ;

204 Serial . print ( Minute ) ; 205 Serial . print (" ") ;

206 Serial . print ( Second ) ; 207 Serial . print (" ") ;

208 Serial . print ( MilliSecond ) ; 209 Serial . print (" ") ;

210 Serial . print ( float ( Phasor\_Magnitude\_1 / 100) ) ; 211 Serial . print (" ") ;

212 Serial . print ( Phasor\_Angle\_Degree\_1 ) ; 213 Serial . print (" ") ;

214 Serial . print ( float ( Phasor\_Magnitude\_2 / 100) ) ; 215 Serial . print (" ") ;

1. Serial . print ( Phasor\_Angle\_Degree\_2 ) ;
2. Serial . print (" ") ;
3. Serial . print ( float ( Phasor\_Magnitude\_3 / 100) ) ; 219 Serial . print (" ") ;

220 Serial . print ( Phasor\_Angle\_Degree\_3 ) ; 221 Serial . print (" ") ;

222 Serial . print ( P1\_freq / 100) ; 223 Serial . print (" ") ;

224 Serial . print ( P2\_freq / 100) ; 225 Serial . print (" ") ;

226 Serial . print ( P3\_freq / 100) ; 227 Serial . print (" ") ;

228 Serial . print ( P1\_rocof / 100) ; 229 Serial . print (" ") ;

230 Serial . print ( P2\_rocof / 100) ; 231 Serial . print (" ") ;

232 Serial . print ( P3\_rocof / 100) ; 233 Serial . print ("\n") ;

234 }

235

236 // Transmit the Phasor parameters to LCD Module 237 void transmit\_phasors\_LCD () {

1. SerialLCD . write (’!’) ;
2. SerialLCD . print (int( Phasor\_Magnitude\_1 ) ) ;
3. SerialLCD . write (’"’) ;
4. SerialLCD . print (int( Phasor\_Magnitude\_2 ) ) ;
5. SerialLCD . write (’#’) ;
6. SerialLCD . print (int( Phasor\_Magnitude\_3 ) ) ;
7. SerialLCD . write (’$’) ;
8. // Calculate angle i.e. 2pi ’s complement to be sent
9. if ( Phasor\_Angle\_1 < 0)
10. Phasor\_Angle\_1 = Phasor\_Angle\_1 + 6.286;
11. if ( Phasor\_Angle\_2 < 0)
12. Phasor\_Angle\_2 = Phasor\_Angle\_2 + 6.286;
13. if ( Phasor\_Angle\_3 < 0)
14. Phasor\_Angle\_3 = Phasor\_Angle\_3 + 6.286; 253 SerialLCD . print (int( Phasor\_Angle\_1 \* 100) ) ; 254 SerialLCD . write (’%’) ;
15. SerialLCD . print (int( Phasor\_Angle\_2 \* 100) ) ;
16. SerialLCD . write (’&’) ;
17. SerialLCD . print (int( Phasor\_Angle\_3 \* 100) ) ;
18. SerialLCD . write (’(’) ;
19. SerialLCD . print ( Day ) ;
20. SerialLCD . write (’)’) ;
21. SerialLCD . print ( Month ) ;
22. SerialLCD . write (’\*’) ; 264 SerialLCD . print ( Year ) ; 265

266 SerialLCD . write (’+’) ; 267 SerialLCD . print ( Hour ) ; 268 SerialLCD . write (’,’) ;

1. SerialLCD . print ( Minute ) ;
2. SerialLCD . write (’-’) ;
3. SerialLCD . print ( Second ) ;
4. SerialLCD . write (’.’) ;
5. SerialLCD . print ( MilliSecond ) ;
6. SerialLCD . write (’/’) ;
7. SerialLCD . print (int( Latitude ) ) ;
8. SerialLCD . write (’:’) ;
9. SerialLCD . print (int( Longitude ) ) ;
10. SerialLCD . write (’;’) ;
11. // Transmit Frequencies 282 SerialLCD . print ( P1\_freq ) ; 283 SerialLCD . write (’@’) ;
12. SerialLCD . print ( P2\_freq ) ;
13. SerialLCD . write (’^’) ;
14. SerialLCD . print ( P3\_freq ) ; 287 SerialLCD . write (’?’) ;

288

1. // Transmit ROCOF
2. SerialLCD . print ( P1\_rocof ) ;
3. SerialLCD . write (’[’) ;
4. SerialLCD . print ( P2\_rocof ) ;
5. SerialLCD . write (’]’) ;
6. SerialLCD . print ( P3\_rocof ) ;
7. SerialLCD . write (’|’) ;

296 }

# Arduino code for local PMU data display unit

1. # include
2. extern uint8\_t Grotesk32x64 []; // Declare which fonts we will be using 3 extern uint8\_t Ubuntubold [];
3. extern uint8\_t Ubuntu [];
4. extern uint8\_t franklingothic\_normal []; 6 extern uint8\_t Inconsola [];

7 extern uint8\_t BigFont []; // Declare which fonts we will be using 8

9 UTFT PMU\_LCD ( CTE70 , 25 , 26 , 27 , 28) ;

10 bool display\_now\_on\_lcd = false ; 11

1. # define IDLE 0
2. # define RECEIVING1 1
3. # define RECEIVING2 2
4. # define RECEIVING3 3
5. # define RECEIVING4 4
6. # define RECEIVING5 5
7. # define RECEIVING6 6
8. # define RECEIVING7 7
9. # define RECEIVING8 8
10. # define RECEIVING9 9
11. # define RECEIVING10 10
12. # define RECEIVING11 11
13. # define RECEIVING12 12
14. # define RECEIVING13 13
15. # define RECEIVING14 14
16. # define RECEIVING15 15
17. # define RECEIVING16 16
18. # define RECEIVING17 17
19. # define RECEIVING18 18
20. # define RECEIVING19 19
21. # define RECEIVING20 20
22. # define RECEIVING21 21

34

1. int Year , Month , Day ;
2. int Hour , Minute , Second , MilliSecond ; 37

38 float Phasor\_Magnitude\_1 ; 39 float Phasor\_Magnitude\_2 ; 40 float Phasor\_Magnitude\_3 ; 41

42 float Phasor\_Angle\_1 , Phasor\_Angle\_Degree\_1 ; 43 float Phasor\_Angle\_2 , Phasor\_Angle\_Degree\_2 ; 44 float Phasor\_Angle\_3 , Phasor\_Angle\_Degree\_3 ; 45

46 float Latitude ; 47 float Longitude ; 48

1. float P1\_freq , P2\_freq , P3\_freq ;
2. float P1\_rocof , P2\_rocof , P3\_rocof ; 51

52 byte status = IDLE ; 53

54 void setup () { 55 // Setup the LCD

56 PMU\_LCD . InitLCD () ;

57

1. init\_LCD () ;
2. Serial3 . begin (921600) ;
3. attachInterrupt (8 , display\_on\_lcd , RISING ) ; 61

62 }

63 int count = 0; 64 void loop () {

65 // put your main code here , to run repeatedly : 66 if ( Serial3 . available () ) {

67 int c = Serial3 . read () ;

68 if ( status == RECEIVING1 && c >= ’0’ && c <= ’9’) {

69 Phasor\_Magnitude\_1 = Phasor\_Magnitude\_1 \* 10 + ( c - ’0’) ; 70 } else if ( status == RECEIVING2 && c >= ’0’ && c <= ’9’) {

1. Phasor\_Magnitude\_2 = Phasor\_Magnitude\_2 \* 10 + ( c - ’0’) ;
2. } else if ( status == RECEIVING3 && c >= ’0’ && c <= ’9’) {
3. Phasor\_Magnitude\_3 = Phasor\_Magnitude\_3 \* 10 + ( c - ’0’) ; 74 } else if ( status == RECEIVING4 && c >= ’0’ && c <= ’9’) {
4. Phasor\_Angle\_1 = Phasor\_Angle\_1 \* 10 + ( c - ’0’) ;
5. } else if ( status == RECEIVING5 && c >= ’0’ && c <= ’9’) {
6. Phasor\_Angle\_2 = Phasor\_Angle\_2 \* 10 + ( c - ’0’) ;
7. } else if ( status == RECEIVING6 && c >= ’0’ && c <= ’9’) {
8. Phasor\_Angle\_3 = Phasor\_Angle\_3 \* 10 + ( c - ’0’) ;
9. } else if ( status == RECEIVING7 && c >= ’0’ && c <= ’9’) {

81 Day = Day \* 10 + ( c - ’0’) ;

82 } else if ( status == RECEIVING8 && c >= ’0’ && c <= ’9’) {

83 Month = Month \* 10 + ( c - ’0’) ;

84 } else if ( status == RECEIVING9 && c >= ’0’ && c <= ’9’) {

85 Year = Year \* 10 + ( c - ’0’) ;

86 } else if ( status == RECEIVING10 && c >= ’0’ && c <= ’9’) {

87 Hour = Hour \* 10 + ( c - ’0’) ;

1. } else if ( status == RECEIVING11 && c >= ’0’ && c <= ’9’) {
2. Minute = Minute \* 10 + ( c - ’0’) ;
3. } else if ( status == RECEIVING12 && c >= ’0’ && c <= ’9’) {
4. Second = Second \* 10 + ( c - ’0’) ;
5. } else if ( status == RECEIVING13 && c >= ’0’ && c <= ’9’) {
6. MilliSecond = MilliSecond \* 10 + ( c - ’0’) ;
7. } else if ( status == RECEIVING14 && c >= ’0’ && c <= ’9’) {
8. Latitude = Latitude \* 10 + ( c - ’0’) ;
9. } else if ( status == RECEIVING15 && c >= ’0’ && c <= ’9’) {
10. Longitude = Longitude \* 10 + ( c - ’0’) ;
11. } else if ( status == RECEIVING16 && c >= ’0’ && c <= ’9’) {

99 P1\_freq = P1\_freq \* 10 + ( c - ’0’) ;

100 } else if ( status == RECEIVING17 && c >= ’0’ && c <= ’9’) {

101 P2\_freq = P2\_freq \* 10 + ( c - ’0’) ;

102 } else if ( status == RECEIVING18 && c >= ’0’ && c <= ’9’) {

103 P3\_freq = P3\_freq \* 10 + ( c - ’0’) ;

104 } else if ( status == RECEIVING19 && c >= ’0’ && c <= ’9’) {

105 P1\_rocof = P1\_rocof \* 10 + ( c - ’0’) ;

106 } else if ( status == RECEIVING20 && c >= ’0’ && c <= ’9’) {

107 P2\_rocof = P2\_rocof \* 10 + ( c - ’0’) ;

108 } else if ( status == RECEIVING21 && c >= ’0’ && c <= ’9’) {

109 P3\_rocof = P3\_rocof \* 10 + ( c - ’0’) ;

110 }

1. else if ( status == RECEIVING1 && c == ’"’) {
2. status = RECEIVING2 ;
3. } else if ( status == RECEIVING2 && c == ’#’) {
4. status = RECEIVING3 ;
5. } else if ( status == RECEIVING3 && c == ’$’) {
6. status = RECEIVING4 ;
7. } else if ( status == RECEIVING4 && c == ’%’) {
8. status = RECEIVING5 ;
9. } else if ( status == RECEIVING5 && c == ’&’) {
10. status = RECEIVING6 ;
11. } else if ( status == RECEIVING6 && c == ’(’) {
12. status = RECEIVING7 ;
13. } else if ( status == RECEIVING7 && c == ’)’) {
14. status = RECEIVING8 ;
15. } else if ( status == RECEIVING8 && c == ’\*’) {
16. status = RECEIVING9 ;
17. } else if ( status == RECEIVING9 && c == ’+’) {
18. status = RECEIVING10 ;
19. } else if ( status == RECEIVING10 && c == ’,’) {
20. status = RECEIVING11 ;
21. } else if ( status == RECEIVING11 && c == ’-’) {
22. status = RECEIVING12 ;
23. } else if ( status == RECEIVING12 && c == ’.’) {
24. status = RECEIVING13 ;
25. } else if ( status == RECEIVING13 && c == ’/’) {
26. status = RECEIVING14 ;
27. } else if ( status == RECEIVING14 && c == ’:’) {
28. status = RECEIVING15 ;
29. } else if ( status == RECEIVING15 && c == ’;’) {
30. status = RECEIVING16 ;
31. } else if ( status == RECEIVING16 && c == ’@’) {
32. status = RECEIVING17 ;
33. } else if ( status == RECEIVING17 && c == ’^’) {
34. status = RECEIVING18 ;
35. } else if ( status == RECEIVING18 && c == ’?’) {
36. status = RECEIVING19 ;
37. } else if ( status == RECEIVING19 && c == ’[’) {
38. status = RECEIVING20 ;
39. } else if ( status == RECEIVING20 && c == ’]’) {
40. status = RECEIVING21 ;

151 } else if ( c == ’|’) { 152 status = IDLE ; 153

154 // Remote value Received completely , Now display it

155

1. // Calculate Phasor Angles into Float
2. Phasor\_Angle\_1 = Phasor\_Angle\_1 / 100;
3. Phasor\_Angle\_2 = Phasor\_Angle\_2 / 100;
4. Phasor\_Angle\_3 = Phasor\_Angle\_3 / 100;
5. // Calculate Phasor Angle using reverse 2pi ’s complement
6. if ( Phasor\_Angle\_1 > 3.143)
7. Phasor\_Angle\_1 = Phasor\_Angle\_1 - 6.286;
8. if ( Phasor\_Angle\_2 > 3.143)
9. Phasor\_Angle\_2 = Phasor\_Angle\_2 - 6.286;
10. if ( Phasor\_Angle\_3 > 3.143)
11. Phasor\_Angle\_3 = Phasor\_Angle\_3 - 6.286;
12. // Calculate Angles in Degrees
13. Phasor\_Angle\_Degree\_1 = ( Phasor\_Angle\_1 \* 4068) / 71;
14. Phasor\_Angle\_Degree\_2 = ( Phasor\_Angle\_2 \* 4068) / 71;
15. Phasor\_Angle\_Degree\_3 = ( Phasor\_Angle\_3 \* 4068) / 71;
16. // Interrupt Driven LCD Display 175 if ( display\_now\_on\_lcd == true ) { 176 Display\_on\_LCD () ;

177 display\_now\_on\_lcd = false ; 178 }

179

180 } else if ( c == ’!’) {

181 status = RECEIVING1 ;

182

183 // Reset the variables to Zero 184 Year = 0;

1. Month = 0;
2. Day = 0;
3. Hour = 0; 4
4. Minute = 0;
5. Second = 0;
6. MilliSecond = 0;
7. Phasor\_Angle\_1 = 0;
8. Phasor\_Angle\_2 = 0;
9. Phasor\_Angle\_3 = 0;
10. Phasor\_Magnitude\_1 = 0;
11. Phasor\_Magnitude\_2 = 0;
12. Phasor\_Magnitude\_3 = 0;
13. Phasor\_Angle\_Degree\_1 = 0;
14. Phasor\_Angle\_Degree\_2 = 0;
15. Phasor\_Angle\_Degree\_3 = 0;
16. Latitude = 0;
17. Longitude = 0;
18. P1\_freq = 0;
19. P2\_freq = 0;
20. P3\_freq = 0;
21. P1\_rocof = 0;
22. P2\_rocof = 0;
23. P3\_rocof = 0;

208 }

209 }

210 }

211

212 void init\_LCD () { 213 // LCD Size 800:480

214 // 0 ,0 799 ,0

215 // 0 ,479 799 ,479

1. PMU\_LCD . setFont ( BigFont ) ;
2. PMU\_LCD . clrScr () ;
3. PMU\_LCD . setColor (0 , 255 , 0) ;
4. PMU\_LCD . print ("\* Phasor Measurement Unit Local Display \*", CENTER , 1) ; 220 PMU\_LCD . setColor (255 , 153 , 51) ;

221 PMU\_LCD . print ("!!! Developed by Debashish Mohapatra !!! ", CENTER , 462) ; 222

1. // Print out Phase 1 phase 2 and Phase 3
2. String Header1 = String (" Phasor ") + String (" Phase1 ") + String (" Phase2 ") + String (" Phase3 ") 225 PMU\_LCD . setColor (255 , 0 , 255) ;
3. PMU\_LCD . setFont ( Inconsola ) ;
4. PMU\_LCD . print ( Header1 , LEFT , 30) ;
5. PMU\_LCD . setColor (255 , 0 , 255) ;
6. PMU\_LCD . print (" Magni :", LEFT , 80) ;
7. PMU\_LCD . print (" Angle :", LEFT , 145) ;
8. PMU\_LCD . print (" Frequ :", LEFT , 220) ;
9. PMU\_LCD . print (" ROCOF :", LEFT , 295) ;
10. PMU\_LCD . setColor (0 , 255 , 0) ;
11. PMU\_LCD . setFont ( BigFont ) ;
12. PMU\_LCD . print (" ( VOLT )", LEFT , 115) ;
13. PMU\_LCD . print ("( DEGREE )", LEFT , 180) ;
14. PMU\_LCD . print (" (HZ)", LEFT , 255) ;
15. PMU\_LCD . print ("(HZ/SEC )", LEFT , 330) ;

241 }

242

243 void Display\_on\_LCD () { 244 // Print Phase1 Parameters

1. PMU\_LCD . setFont ( Inconsola ) ;
2. PMU\_LCD . setColor (255 , 0 , 0) ;// Red
3. PMU\_LCD . printNumF ( Phasor\_Magnitude\_1 / 100 , 2 , 185 , 85 , 46 , 5 ,48) ;

248 PMU\_LCD . print (" ", 185 , 150) ;

249 PMU\_LCD . printNumF ( Phasor\_Angle\_Degree\_1 , 2 , 185 , 150 , 46 , 6 , 48) ;

250 PMU\_LCD . printNumF ( P1\_freq / 10000 , 2 , 185 , 225 , 46 , 4 ,48) ;

251 PMU\_LCD . printNumF ( P1\_rocof / 10000 , 2 , 185 , 300 , 46 , 4 ,48) ;

252

1. PMU\_LCD . setColor (255 , 255 , 0) ;// Yellow
2. PMU\_LCD . printNumF ( Phasor\_Magnitude\_2 / 100 , 2 , 380 , 85 , 46 , 5 ,48) ; 255 PMU\_LCD . print (" ", 380 , 150) ;

256 PMU\_LCD . printNumF ( Phasor\_Angle\_Degree\_2 , 2 , 380 , 150 , 46 , 6 , 48) ;

257 PMU\_LCD . printNumF ( P2\_freq / 10000 , 2 , 380 , 225 , 46 , 4 ,48) ;

258 PMU\_LCD . printNumF ( P2\_rocof / 10000 , 2 , 380 , 300 , 46 , 4 ,48) ;

259

1. PMU\_LCD . setColor (127 , 250 , 250) ;// White - Blue
2. PMU\_LCD . printNumF ( Phasor\_Magnitude\_3 / 100 , 2 , 600 , 85 , 46 , 5 ,48) ; 262 PMU\_LCD . print (" ", 600 , 150) ;

263 PMU\_LCD . printNumF ( Phasor\_Angle\_Degree\_3 , 2 , 600 , 150 , 46 , 6 , 48) ;

264 PMU\_LCD . printNumF ( P3\_freq / 10000 , 2 , 600 , 225 , 46 , 4 ,48) ;

265 PMU\_LCD . printNumF ( P3\_rocof / 10000 , 2 , 600 , 300 , 46 , 4 ,48) ;

266

1. // Print GPS Information
2. String Time = String (" UTC Time : ") + String ( Hour ) + ":" + String ( Minute ) + ":" + String ( Second ) + " ";
3. String Date = String (" Date : ") + String ( Day ) + "/" + String ( Month ) + "/" + String ( Year ) ;
4. String Location = String (" Latitude : ") + String ( Latitude / 1000) + String (" Longitude : ") + String ( Longitude / 1000) ;
5. PMU\_LCD . setFont ( Ubuntu ) ;

273 PMU\_LCD . setColor (255 , 255 , 255) ;

1. PMU\_LCD . print ( Time , LEFT , 360) ;
2. PMU\_LCD . print ( Date , LEFT , 395) ;
3. PMU\_LCD . print ( Location , LEFT , 430) ; 277 }

278 void display\_on\_lcd () { // ISR 279 display\_now\_on\_lcd = true ; 280 }

# Appendix B

## Python program for real-time plotting and logging of the Phasors

1 from pyqtgraph . Qt import QtGui , QtCore 2 import pyqtgraph as pg

1. ## import time
2. import numpy as np 5
3. import serial
4. ser = serial . Serial (’com8 ’, 460800 , timeout =1)
5. # Connect to serial port at COM8 , at 460800 bauds 9

10

11 pg . setConfigOptions ( antialias = True ) 12 # Enable antialiasing for prettier plots 13

14 app = QtGui . QApplication ([]) 15 win = pg . GraphicsWindow () 16

1. win . setWindowTitle (’Realtime PMU Data Monitoring ’)
2. # Set the window title 19
3. # Define first graph to show the phasor magnitudes
4. p1 = win . addPlot ( title =" Phasor Magnitudes ", colspan =2) 22 p1 . setRange ( yRange =[215 , 250] , xRange =[0 , 1000])

23 p1 . setLabel (’left ’, " Phasor RMS Magnitude ", units =’Volts ’) 24 p1 . setLabel (’bottom ’, " Time ( x20 milli Seconds )")

25 p1 . showGrid ( x =1 , y =1 , alpha =.5) 26 p1 . addLegend ( offset =[ -10 , -10]) 27

1. win . nextRow ()
2. # Define second graph to show the phasor angles 30 p2 = win . addPlot ( title =" Phasor Angles ")

31 p2 . setRange ( yRange =[ -200 , 200] , xRange =[0 , 1000]) 32 p2 . setLabel (’left ’, " Phasor angles ", units =’Degree ’) 33 p2 . setLabel (’bottom ’, " Time ( x20 milli Seconds )")

34 p2 . showGrid ( x =1 , y =1 , alpha =.5) 35 p2 . addLegend ( offset =[ -40 , -10]) 36

37 # Define third graph to show the phasor polar plot 38 v = win . addViewBox ()

39 v . setAspectLocked () 40 v . setFixedWidth (500) 41 p3 = pg . PlotItem ()

42 p3 . setRange ( xRange =[ -250 ,250] , yRange =[ -250 , 250])

1. curvePen = pg . mkPen ( color =(255 , 255 , 255) , style = QtCore . Qt . DotLine )
2. c1 = p3 . plot ( x =218\* np . cos ( np . linspace (0 , 2\* np . pi , 360) ) , y =218\* np . sin ( np . linspace (0 , 2\* np . pi , 360) ) , pen = curvePen , name ="218V" ,)
3. c2 = p3 . plot ( x =50\* np . cos ( np . linspace (0 , 2\* np . pi , 360) ) , y =50\* np . sin ( np . linspace (0

, 2\* np . pi , 360) ) , pen = curvePen , name ="50V")

1. c4 = p3 . plot ( x =150\* np . cos ( np . linspace (0 , 2\* np . pi , 360) ) , y =150\* np . sin ( np . linspace (0 , 2\* np . pi , 360) ) , pen = curvePen , name ="150V")
2. c6 = p3 . plot ( x =250\* np . cos ( np . linspace (0 , 2\* np . pi , 360) ) , y =250\* np . sin ( np . linspace (0 , 2\* np . pi , 360) ) , pen = curvePen , name ="250V")

48 c7 = p3 . plot ( x = np . linspace ( -177 , 177 , 500) , y = np . linspace ( -177 , 177 , 500) , pen = curvePen )

49 c8 = p3 . plot ( x = np . linspace ( -177 , 177 , 500) , y = np . linspace (177 , -177 , 500) , pen = curvePen )

50 c9 = p3 . plot ( x = np . linspace ( -250 , 250 , 500) , y = np . linspace (0 , 0 , 500) , pen = curvePen ) 51 c10 = p3 . plot ( x = np . linspace (0 , 0 , 500) , y = np . linspace ( -250 , 250 , 500) , pen = curvePen ) 52 p3 . addLegend ( offset =[ -1 , -1])

53

54 g = pg . GraphItem () 55 v . addItem ( g )

56 v . addItem ( c1 ) 57 v . addItem ( c2 ) 58 v . addItem ( c4 ) 59 v . addItem ( c6 ) 60 v . addItem ( c7 ) 61 v . addItem ( c8 ) 62 v . addItem ( c9 ) 63 v . addItem ( c10 ) 64

1. # plot the curves in the graph areas
2. curve1 = p1 . plot ( pen =(255 , 0 , 0) , name =" Phase 1( RMS Magnitude )") 67 curve2 = p1 . plot ( pen =(255 , 255 , 0) , name =" Phase 2( RMS Magnitude )") 68 curve3 = p1 . plot ( pen =(0 , 0 , 255) , name =" Phase 3( RMS Magnitude )") 69

70 curve4 = p2 . plot ( pen =(255 , 0 , 0) , name =" Phase 1") 71 curve5 = p2 . plot ( pen =(255 , 255 , 0) , name =" Phase 2") 72 curve6 = p2 . plot ( pen =(0 , 0 , 255) , name =" Phase 3") 73

74 # Read the serial data string coming in from the PMU 75 line1 = ser . readline ()

76 # split the string and extract the phasor parameters 77 data1 = [ float ( val1 ) for val1 in line1 . split () ]

78

79 previous\_minute = int( data1 [4])

80

1. # define the log files , where the phasors will be stored
2. path\_txt = ’pmu\_data .txt ’
3. path\_txt\_plot = ’ pmu\_data\_plot .txt ’ 84 path\_excel = ’pmu\_data .csv ’

85

1. now\_min = "%s -%s -% s\_%s -%s" %( int ( data1 [0]) , int ( data1 [1]) , int ( data1 [2]) , int ( data1 [3]) , int ( data1 [4]) )
2. path\_txt\_n = ’%s\_%s’ % ( now\_min , path\_txt ) 88 path\_txt\_plot\_n = ’%s\_%s’ % ( now\_min ,

path\_txt\_plot )

89 path\_excel\_n = ’%s\_%s’ % ( now\_min , path\_excel )

90

91 logfileExcel = open ( path\_excel\_n , ’a’) 92 logfileText = open ( path\_txt\_n , ’a’)

93 logfileTextPlot = open ( path\_txt\_plot\_n , ’a’)

94

95 # define the read function to read the data stream and append the 96 # parameters to sepatrate arrays

1. def readfun () :
2. global data , current\_minute , previous\_minute , FORMAT , logfileText , logfileTextPlot , logfileExcel , path\_txt , path\_excel , path\_txt\_plot
3. line = ser . readline ()
4. data = [ float ( val ) for val in line . split () ] 101
5. current\_minute = int ( data [4])
6. if current\_minute == ( previous\_minute +1) :
7. now\_m = "%s -%s -% s\_%s -%s" %( int( data [0]) , int( data [1]) , int( data [2]) , int ( data [3]) , int ( data [4]) )
8. new\_path\_txt = ’%s\_%s’ % ( now\_m , path\_txt )
9. new\_path\_txt\_plot = ’%s\_%s’ % ( now\_m , path\_txt\_plot ) 107 new\_path\_excel = ’%s\_%s’ % ( now\_m , path\_excel )

108

109 logfileExcel . flush () 110 logfileText . flush () 111 logfileTextPlot . flush () 112 logfileExcel . close () 113 logfileText . close () 114 logfileTextPlot . close ()

115 logfileExcel = open ( new\_path\_excel , ’a’) 116 logfileText = open ( new\_path\_txt , ’a’)

1. logfileTextPlot = open ( new\_path\_txt\_plot , ’a’)
2. previous\_minute = current\_minute 119

120 a = "%s -%s -%s, %s:%s:%s:%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s , %s, %s, %s" % (int( data [0]) , int( data [1]) , int( data [2]) , int( data [3]) , int( data [4]) , int ( data [5]) , int ( data [6]) , data [7] ,

data [8] , data [9] , data [10] , data [11] , data [12] , data [13] , data [14] , data [15] , data [16] , data

[17] , data [18] ,"\n")

121 logfileExcel . write ( a ) 122 logfileText . write ( a ) 123

124 bs = int ( data [5]) 125 bms = int( data [6])

126 bmS = ( bs \*1000) + bms

127 b = "%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s %s" % ( bmS , data [7] , data [8] , data [9] ,

data [10] , data [11] , data [12] , data [13] , data [14] , data [15] , data [16] , data [17] , data [18] , "\n")

1. logfileTextPlot . write ( b )
2. return data [6] , data [7] , data [8] , data [9] , data [10] , data [11] , data [12]

131 readData = [0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0]

132

133 y2 = np . zeros (1000 , dtype = float ) 134 y3 = np . zeros (1000 , dtype = float ) 135 y4 = np . zeros (1000 , dtype = float ) 136 y5 = np . zeros (1000 , dtype = float ) 137 y6 = np . zeros (1000 , dtype = float ) 138 y7 = np . zeros (1000 , dtype = float ) 139

1. indx = 0
2. # define the update function to update the plots with the parameter arrays 142 def update () :

143 global curve1 , curve2 , curve3 , indx , y2 , y3 , y4 , y5 , y6 , y7 #y1 144

145 readData = readfun () # function that reads data from the sensor it returns a list of 6 elements as the y- coordinates for the updating plots

146

147 y2 [ indx ]= readData [1] 148 y3 [ indx ]= readData [2] 149 y4 [ indx ]= readData [3] 150 y5 [ indx ]= readData [4] 151 y6 [ indx ]= readData [5] 152 y7 [ indx ]= readData [6] 153

154 Rx = y2 [ indx ]\* np . cos ( np . deg2rad ( y3 [ indx ]) ) 155 Ry = y2 [ indx ]\* np . sin ( np . deg2rad ( y3 [ indx ]) ) 156 Yx = y4 [ indx ]\* np . cos ( np . deg2rad ( y5 [ indx ]) ) 157 Yy = y4 [ indx ]\* np . sin ( np . deg2rad ( y5 [ indx ]) ) 158 Bx = y6 [ indx ]\* np . cos ( np . deg2rad ( y7 [ indx ]) ) 159 By = y6 [ indx ]\* np . sin ( np . deg2rad ( y7 [ indx ]) ) 160

161 pos = np . array ([[0 ,0] ,[ Rx , Ry ] ,[ Yx , Yy ] ,[ Bx , By ]])

162 adj = np . array ([[0 ,1] ,[0 ,2] ,[0 ,3]])

163 symbols = [’o’,’t’,’t’,’t’]

164 lines = np . array ([(255 ,0 ,0 ,255 ,3) ,(255 ,255 ,0 ,255 ,3) ,(0 ,0 ,255 ,255 ,3) ] , dtype =[( ’red ’,

np . ubyte ) ,

1. (’green ’, np . ubyte ) ,(’blue ’, np . ubyte ) ,(’alpha ’, np . ubyte ) ,(’ width ’,float ) ])
2. g . setData ( pos = pos , adj = adj , pen = lines , size =1 , symbol = symbols ) 167
3. if indx ==999:
4. y2 = np . zeros (1000 , dtype = float ) 170 y3 = np . zeros (1000 , dtype = float ) 171 y4 = np . zeros (1000 , dtype = float ) ] 172 y5 = np . zeros (1000 , dtype = float )

173 y6 = np . zeros (1000 , dtype = float ) 174 y7 = np . zeros (1000 , dtype = float ) 175 indx = 0

1. else :
2. indx +=1
3. curve1 . setData ( y2 )# update magnitudes 180 curve2 . setData ( y4 )
4. curve3 . setData ( y6 )
5. curve4 . setData ( y3 )# update angles 183 curve5 . setData ( y5 )

184 curve6 . setData ( y7 ) 185 app . processEvents () 186

1. timer = QtCore . QTimer ()
2. timer . timeout . connect ( update ) 189 timer . start ()

190

1. if name == ’ main ’:
2. import sys
3. if ( sys . flags . interactive != 1) or not hasattr ( QtCore , ’PYQT\_ ’) :
4. QtGui . QApplication . instance () . exec\_ ()

# Python program for offline plotting of phasor data

1. import numpy as np
2. import matplotlib . pyplot as plt 3
3. with open (" pmu\_data\_49\_95\_Hz .txt ") as f :
4. data = f . read () 6
5. data = data . split (’\n’)
6. x8 = [ row . split (’,’) [0] for row in data ] 9 x7 = [ row . split (’,’) [1] for row in data ] 10 x1 = [ row . split (’,’) [2] for row in data ] 11 x2 = [ row . split (’,’) [3] for row in data ] 12 x3 = [ row . split (’,’) [4] for row in data ] 13 x4 = [ row . split (’,’) [5] for row in data ] 14 x5 = [ row . split (’,’) [6] for row in data ] 15 x6 = [ row . split (’,’) [7] for row in data ] 16

17 fig = plt . figure () 18

19 ax1 = fig . add\_subplot (211) 20

1. ax1 . set\_title (" Plot of Reported Phasors by PMU")
2. ax1 . set\_xlabel (’Time in Milli Seconds ’)
3. ax1 . set\_ylabel (’Amplitude in Volts (RMS)’)

24

25 ax1 . plot ( x7 , x1 ,’-’,c =’r’, linewidth =2.0 , label =’Ph1 Magnitude ’) 26 ax1 . plot ( x7 , x3 ,’--’,c =’r’, linewidth =2.0 , label =’Ph2 Magnitude ’) 27 ax1 . plot ( x7 , x5 ,’ -.’,c =’r’, linewidth =2.0 , label =’Ph3 Magnitude ’) 28

29

30 leg = ax1 . legend ()

31

1. ax2 = fig . add\_subplot (212)
2. ax2 . set\_xlabel (’Time in Milli Seconds ’)
3. ax2 . set\_ylabel (’Phasor Angles in Degree ’)

35

36 ax2 . plot ( x7 , x2 ,’-’,c =’r’, linewidth =2.0 , label =’Ph1 Angle ’) 37 ax2 . plot ( x7 , x4 ,’--’,c =’y’, linewidth =2.0 , label =’Ph2 Angle ’) 38 ax2 . plot ( x7 , x6 ,’ -.’,c =’b’, linewidth =2.0 , label =’Ph3 Angle ’) 39

40 leg = ax2 . legend () 41

42 plt . show ()