# Development and Hardware Implementation of a Phasor Measurement Unit using Microcontroller

*Thesis submitted to*

*National Institute of Technology, Rourkela*

*For award of the degree*

*of*

**Master of Technology**

*by*

**Debashish Mohapatra**

Under the guidance of

**Prof. Pravat Kumar Ray**

**DEPARTMENT OF ELECTRICAL ENGINEERING**
**NATIONAL INSTITUTE OF TECHNOLOGY**
**ROURKELA**

# CERTIFICATE

This is to certify that the thesis entitled DEVELOPMENT AND HARDWARE IMPLEMENTATION OF A PHASOR MEASUREMENT UNIT USING MICROCONTROLLER, submitted by DEBASHISH MOHAPATRA to National Institute of Technology, Rourkela, is a record of bona fide research work under my supervision and we consider it worthy of consideration for award of the degree of Master of Technology of the Institute.

Prof. Pravat Kumar Ray

(Supervisor)

# ACKNOWLEDGEMENTS

I express my sincere gratitude to my supervisor, Dr. Pravat Kumar Ray for his valuable guidance and suggestions without which this thesis would not be in its present form. I also thank him for his consistent encouragements throughout the work.

I express my gratitude to Prof. Anup kumar Panda, the Head of the Department of Electrical Engineering, NIT Rourkela for extending some facilities towards completion of this thesis. Thanks also to other faculty members in the department.

I express my gratitude to Mr. Rama Prasad Samantaray, Simulation and computing lab, Department of Electrical Engineering, NIT Rourkela for his help in setting up some facilities for the real time implementation and verification of the subject matter of this thesis.

I thank Abhilash, Abhishek, Sudipta, Anupam, Pavan and other fellow M.Tech scholars for their enjoyable and helpful company I had with.

My warmest thanks go to my family for their support, love, encouragement and patience.

Debashish Mohapatra
Rourkela

**to my family**

# 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 synchronise 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).

# Contents

# 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 |

| | | |
|---|---|---|
| LAN | : | Local Area Network |
| GUI | : | Graphical User Interface |
| CSV | : | Comma Separated Values |
| MSE | : | Mean Square Error |
| TVE | : | Total Vector Error |
| ISR | : | Interrupt Service Routine |
| CPU | : | Central Processing Unit |
| DMA | : | Direct Memory Access |
| RAM | : | Random Access Memory |
| FPGA | : | Field Programmable Gate Array |

# List of Figures

# List of Tables

CHAPTER 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 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 [23].

## 1.1 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 [3], [4], and parallel efforts by Bonanomi in 1981 [5].

However, the synchrophasor technology available today emerged from the early efforts by Phadke et al. at Virginia Tech as described in [6], [7]. Phadke demonstrated the first synchronized PMU in 1988, and in 1991 Macrodyne Inc. launched the first commercial PMU product [8]. 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 [9].

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 [10].

The standard for PMU devices is maintained by the IEEE C37.118 Working Group. IEEE Std. C37.118 [1] was released in 2005 and subsequently updated in 2011. The latest release comes in two parts; IEEE C37.118.1-2011 [1] describes how synchrophasors should be estimated and gives certification requirements while IEEE C37.118.2-2011 [2] describes data representation and data transfer. Concerns have been raised regarding the transient performance of PMUs under the 2005 standard [1], [11], [12]. 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 [1].

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 lowcost hardware, such as described in [13]. Two university projects are described in this section. Duplication of such work leads to lost time and resources. The OpenPMU project provides a common set of resources for PMU development and research collaboration. The successful open-source Phasor Data Concentrator, openPDC [14], is discussed, and the rational for using an open-source model is developed.

### 1.1.1 GridTRAK PMU

The GridTrak PMU was produced at Baltimore University by Stadlin [15]; 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 point-on-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.

### 1.1.2 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 [17]. 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 [18]. 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.

### 1.1.3 The openPDC

The openPDC was developed in the wake of the Northeast Blackout of 2003 [19]. 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 openPDC online community [14] has taken the venture of recording the history and development of openPDC and associated projects. The openPDC is utilized by the North American Synchrophasor Initiative.

The openPDC 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 openPDC system primarily operates between real-world telecommunications infrastructure and an archive DB. The adapters within openPDC 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. Output adapters can be used to forward data in a chosen communication language such as to emulate a physical PMU. In this way the openPDC can operate a diverse variety of user-specific configurations.

## 1.2 Shortcomings of currently developed PMUs

Commercial PMUs are already available in the market, but they often come with a very high price tag and strict copyright limitations. The schematics of the PMUs are not openly available as their business policies go against it. The algorithm or methodology of how a PMU actually operates, how it calculates the phasors of the sampled voltage and current signal, what sort of algorithm it follows etc. are also guarded by copyright laws. These PMUs do not allow to be used for educational or academic research purposes.

Therefore an open hardware platform is desired which can be reconfigured to suit the requirement of the client. Building a low cost hardware platform for the PMUs might have proven a costly endeavour in the past but recent advances and wide availability of low cost high performance micro-controller platforms have given rise to many possibilities which can be exploited to build the desired PMU. There has been a growing attempt to manufacture a cost effective open source PMU for research and academic purpose. Quite a few attempts to build open source PMU have been reported in literature.

An open source PMU has been reported in [24] which is built using LabVIEW environment. The open PMU adheres to the IEEE standards. Norwegian transmission system operator (Statnett SF) in collaboration with SmarTS lab at KTH Royal Institute of Technology developed a software development toolkit for synchro phasor application. The developed openPMU's hardware design and the firmwares are open-source [25], [26]. Since it uses LabVIEW and NI DAQ card for signal acquisition, which are property of National Instruments, the PMU itself is not completely open. Hence there is a need to develop both the hardware and the software of the PMU using Open-source Hardware and Software.

## 1.3 Current status of work done in the area

### 1.3.1 National status

- Tetra Tech, Bangalore is working on typical challenges associated with connecting Renewable Energy Sources to the Micro-grid using PMU and finding technical solution being explored on grid extension, conventional back up power, demand side management and in large scale electricity storage
- National Instruments Limited is involved in Micro-grid automation using communication technologies ,sensors and PMU
- GE (General Electric) Power deals with R & D Systems on Renewable Energy systems and on integration aspects of PMU in smart grid
- IIT Bombay has developed iPDC, a free Phasor Data Concentrator.

### 1.3.2 International status

- Queen's University Belfast, United Kingdom, KTH, Royal Institute of Technology, Sweden, Letterkenny Institute of Technology, Ireland jointly undertaking a project to develop an Open PMU Platform.
- Grid Protection Alliance (Open Source Software & Services for Electric Utilities), is developing OpenPDC (Open Source Phasor Data Concentrator).
- Sharif University of Technology, Tehran is doing his research on control design approach on three phase grid connected Renewable Energy Resources.
- Queens University Kingston is working on this area of filtering Techniques in three phase power systems.
- University Park, Notingham, UK is continuing his research on Control Design and Implementation for High Performance Shunt Active Filters in Aircraft Power Grids.

## 1.4 Concluding Remarks and Scope of the Present Work

The review undertaken leads to some open problems that appears not to have been addressed so far in literature. These are as follows.

- While quite a few attempts of building a OpenPMU have been made in the past, the idea of using Open Source Hardware to build a PMU has not been considered.
- Most Prototype PMUs described in literature do not consider the important features of a practical PMU such as use of a GPS synchronised clock, a true signal acquisition system and a user friendly PMU data presentation system.
- Lack of using generalised Hard-wares, which have a global presence, in the development of a PMU, makes it impossible to re-trace the phases of development and any opportunity to improve the design vanishes along with it.
- Hence there exists a need to build a cost-effective, Microcontroller based Phasor Measurement Unit which can report the Voltage and Current Phasors, satisfying the IEEE PMU Standards which can not only be used for R & D purposes but also be put to operation as the proven building block of a Smart Grid.
- This thesis discusses the design and implementation a low cost PMU using open hardware platform (Arduino) and openSource software platform (Python) as per existing IEEE standard for synchrophasor measurement (C37.118.1-2011). The proposed PMU estimates voltage phasors using 64 point DFT and the voltage signal is sampled at 3.2 kHz with 12 bit ADC resolution. A low cost microcontroller (ARM CORTEX M3) is used as the computational unit. Each phasor is time stamped with time sourced from GPS. The local communication is done using Universal Asynchronous Transmitter and Receiver (UART) which is a type of serial communication. The phasors are transmitted to remote location via Transmission Control Protocol (TCP) over Ethernet.

The remaining chapters of this thesis attempt to describe in detail the principles, design procedures, and experimental validation of the developed PMU.

**Chapter 2** provides the principles of estimation of the Phasors from sampled data. It also presents the a detailed description of hardware design and implementation implementation of a Phasor Measurement Unit. Each module is described separately to keep the re-usability of the specific work intact.

**Chapter 3** provides experimental results and looks in to the compliance issues. Comparative study of the performance of the PMU under various operating conditions have been described. Discussions have been made on what are the limitations of the current design, and new ideas have been proposed on how to improve the design, both in hardware and software or algorithm.

**Chapter 4** concludes the thesis and proposes some future research directions led by the present work.

**The Appendices** contain the source codes, both for the microcontrollers for developing the PMU and for the computer for developing the PDC, to be true to the Openness concept of this thesis.

CHAPTER 2

# Hardware Design and Implementation

This section describes the implementation of a Phasor measurement unit using open source hardware. The block diagram of such a unit is shown in Figure2.1.

## 2.1 Phasor Calculation for 3-phase system

Consider a balanced 3-phase power system operating at a nominal frequency of $f_0$, then the voltage waveform can be represented as

$$\begin{aligned} x_1(t) &= X_m cos(2\pi f_0 t + \phi_1) \\ x_2(t) &= X_m cos(2\pi f_0 t + \phi_2) \\ x_3(t) &= X_m cos(2\pi f_0 t + \phi_3) \end{aligned} \tag{2.1}$$

Here $X_m$ represents the maximum amplitude of the signal and $\Phi$ represents the phase angle. The phase angles are 120 degree or $\frac{2\pi}{3}$ radian apart. The time domain sample of the power system can be represented as

$$\begin{aligned}
X_{n1} &= X_m cos(\frac{2\pi n}{N} + \phi_1) \\
X_{n2} &= X_m cos(\frac{2\pi n}{N} + \phi_2) \\
X_{n3} &= X_m cos(\frac{2\pi n}{N} + \phi_3)
\end{aligned} \tag{2.2}$$

Here $N$ is the number of samples, which is an integer multiple of fundamental frequency $f_0$ 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

$$X = \frac{1}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N} - jsin\frac{2\pi n}{N}) \tag{2.3}$$

N-point DFT of the signal can be found out using

$$X_k = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N} - jsin\frac{2\pi n}{N}) \tag{2.4}$$

$$X_{nominal} = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N} - jsin\frac{2\pi n}{N}) \tag{2.5}$$

The real and imaginary part of the above expression can be rewritten as

$$X_{real} = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N}) \tag{2.6}$$

$$X_{img} = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N}) \tag{2.7}$$

The phasor estimate at nominal frequency is represented by this complex quantity $X_{nominal}$, whose magnitude $|X_{nominal}| = \sqrt{X_{real}^2 + X_{img}^2}$ gives the RMS magnitude of the signal. The phase angle can be computed using the trigonometric property,$\phi_{nominal} = atan(\frac{X_{img}}{X_{real}})$.

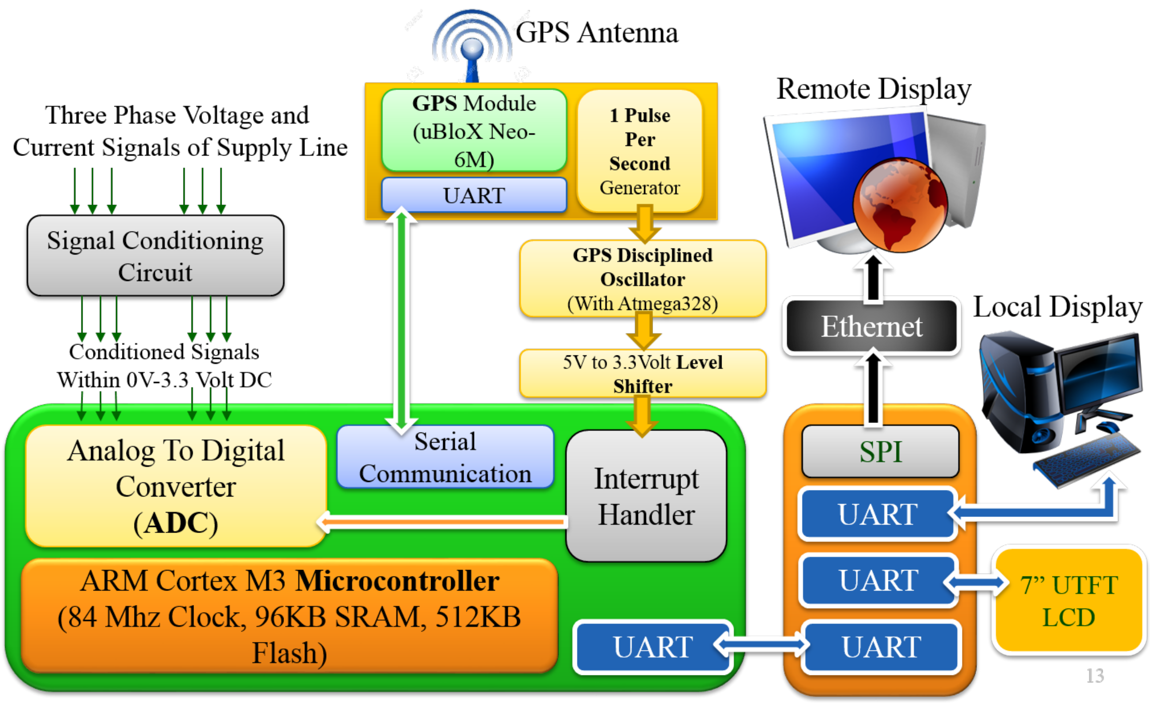

GPS Antenna
Three Phase Voltage and Current Signals of Supply Line
GPS Module (uBloX Neo-6M)
UART
1 Pulse Per Second Generator
Remote Display
Signal Conditioning Circuit
GPS Disciplined Oscillator (With Atmega328)
Ethernet
Local Display
Conditioned Signals Within 0V-3.3 Volt DC
5V to 3.3Volt Level Shifter
Analog To Digital Converter (ADC)
Serial Communication
Interrupt Handler
SPI
UART
UART
7" UTFT LCD
ARM Cortex M3 Microcontroller (84 Mhz Clock, 96KB SRAM, 512KB Flash)
UART
UART

Figure 2.1: Block diagram of the Proposed PMU

## 2.2 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.

### 2.2.1 Three Phase Signal Acquisition and Conditioning

The three phase voltages need to be measured by the microcontroller for estimation of the Phasors. However, the problem is the Microcontroller's ADC being able to only take in the signals in the range of 3.3 volt DC as input. To address this issue a suitable voltage signal conditioning circuit needed to be designed which can convert the 240 volt Phase voltages (or 679.21 volt Peak to Peak voltages) where the highest peak is 339.6 and lowest peak is -339.6 volt. For this purpose, a suitable voltage sensor has been designed using LEM LV-25P Hall Effect voltage sensor as shown in Figure2.4.

Table 2.1: Components required to build the Voltage Sensor Module

| Quantity | DETAILS | UNIT PRICE (in RS) | LINE TOTAL |
|---|---|---|---|
| 3 | LEM LV25P Hall Effect Voltage Sensor | 2400 | 7200 |
| 3 | 50K, 10 Watt Burden Resistor | 100 | 300 |
| 3 | 56K, 3 Watt Resistor | 5 | 15 |
| 10 | Screw Terminals | 10 | 100 |
| 3 | LM358 Dual Opamp Ics | 15 | 45 |
| 3 | 4Pin IC Base | 5 | 15 |
| 1 | PCB, Solder, Wires etc. | 500 | 500 |
| | | **Net Total** | **8175** |

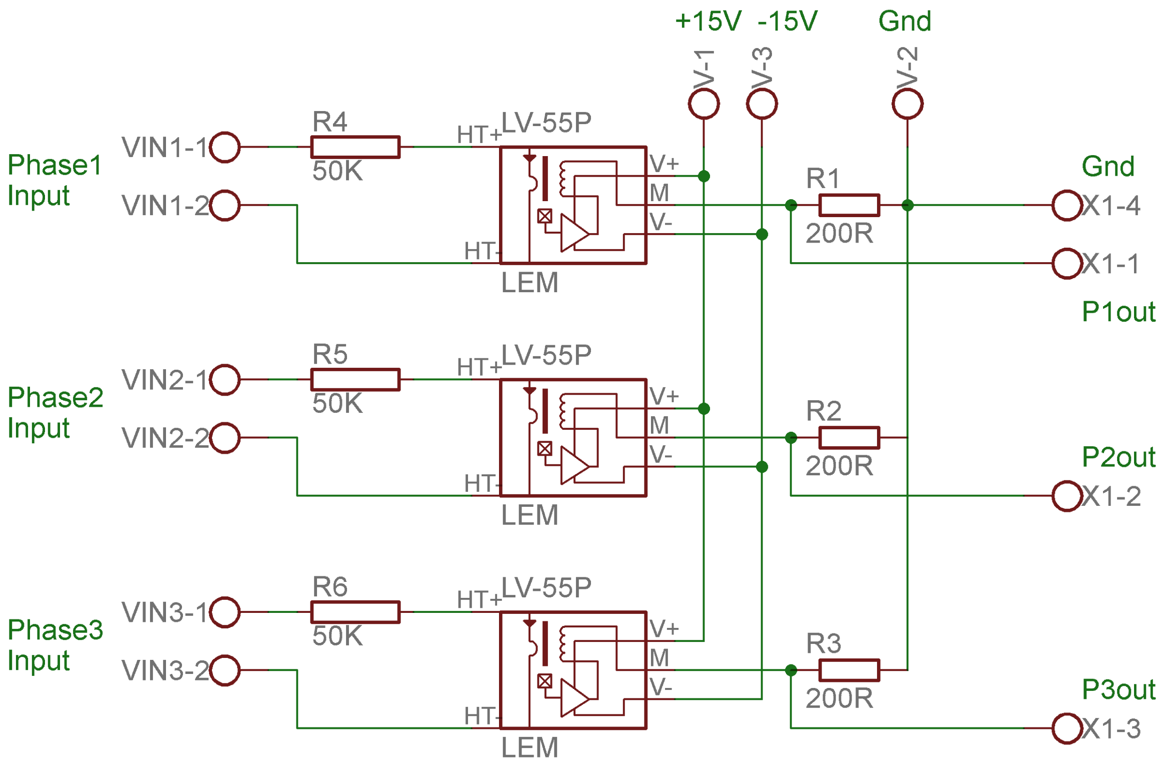


Figure 2.2: Schematics for 3 Phase signal acquisition board Part I

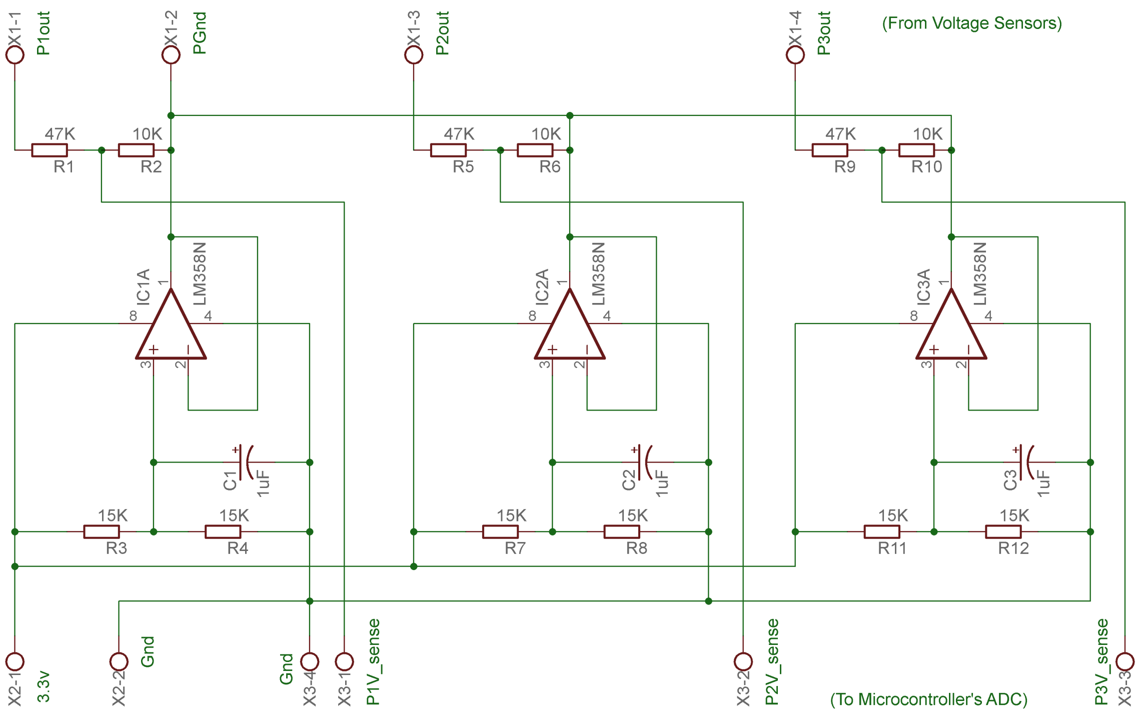


Figure 2.3: Schematics for 3 Phase signal acquisition board Part II

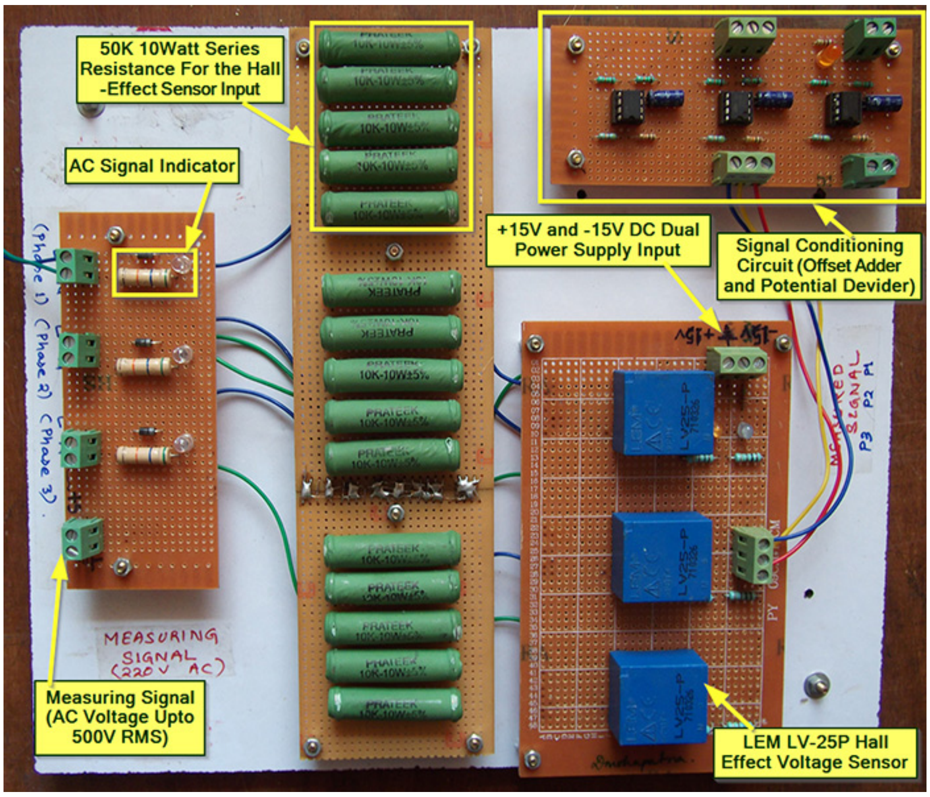

50K 10Watt Series Resistance For the Hall -Effect Sensor Input
AC Signal Indicator
+15V and -15V DC Dual Power Supply Input
Signal Conditioning Circuit (Offset Adder and Potential Devider)
Measuring Signal (AC Voltage Upto 500V RMS)
LEM LV-25P Hall Effect Voltage Sensor
(Phase 1) (Phase 2) (Phase 3)
MEASURING SIGNAL (220V AC)
P3 P2 P1

Figure 2.4: The developed 3 Phase signal acquisition board

### 2.2.2 Programmable 3-phase signal generator

Although three phase signal is available from the three phase supply i.e. 440V Line to Line or 220 Volt Line to Neutral at 50 Hz , it will not be adequate when building the Phasor processor, because of its fixed parameters. The analysis of the PMU incorporates the calculation/estimation the Phasors at nominal frequency, i.e. 50Hz, and off-nominal frequencies i.e. 49.5 Hz to 50.5 Hz. Hence a programmable 3-phase sine wave generator is desired, with which the generated sine waves' amplitude, frequency and phase shifts can be modified. There are two different methods to achieve the same, which are described as follows.

Method 1: The pre-calculated values (samples) of three phase signals generated in a third party program like MATLAB, SciLab or by some other application are stored in the memory (the area where the program is stored or ProgMEM) of the micro-controller. Then the phasors are calculated using these pre-sampled values. The benefit of this method is that there is no need to develop any signal generation, or acquisition hardware, for development and implementation of the Phasor calculation algorithms. However there is a disadvantage, which is by neglecting the data acquisition process, the computational time will differ. It happens so because the ADC conversion time is ignored in this process.

Method 2: A programmable sine wave generator is built, with which the generated sine waves amplitude, frequency and phase shifts can be altered to get the desired waveforms for analysis and estimation of phasors and other parameters. A pre-calculated lookup table is stored in the flash memory of the micro-controller of the programmable sine wave generator. The micro-controller fetches a value from the memory and writes it to one of its ports. A R-2R ladder Digital to Analog Converter converts this digital value into a analog voltage. Similarly, by subsequent conversions of the values stored in the look up table, a sine wave is generated. Three such look up tables are used to write data to three ports of the micro-controller, which makes the three phase signal generator. The circuit schematic of the programmable sine wave generator is shown in Figure 2.5 and Figure 2.6. The hardware realization of the programmable sine wave generator is shown in Figure2.7.

The arduino Mega 2560 Microcontroller board was used to generate three sinusoidal signals. Since the IO lines of this microcontroller board operate at 5 volt DC, a suitable signal conditioning circuit was designed with the help of an LM358 opamp to increase the current sourcing capability of the DAC and also to make the voltage level fall between 0 volt to 3.3 volt.

Table 2.2: Components required to build the programmable 3-phase sine wave generator

| Quantity | DETAILS | UNIT PRICE (in RS) | LINE TOTAL |
|---|---|---|---|
| 1 | Arduino Mega 2560 | 3400 | 3400 |
| 1 | Prototyping Shield for Mega2560 | 400 | 400 |
| 75 | 10K Resistances | 0.4 | 30 |
| 1 | Other resistors and Caps | 50 | 50 |
| 14 | Screw Terminals | 10 | 140 |
| 3 | Flat Ribbon Cables and connectors | 50 | 150 |
| 3 | LM358 Dual Opamp Ics | 15 | 45 |
| 3 | 4Pin IC Base | 5 | 15 |
| 1 | PCB, Solder, Wires etc. | 500 | 500 |
|  |  | **Net Total** | **4730** |

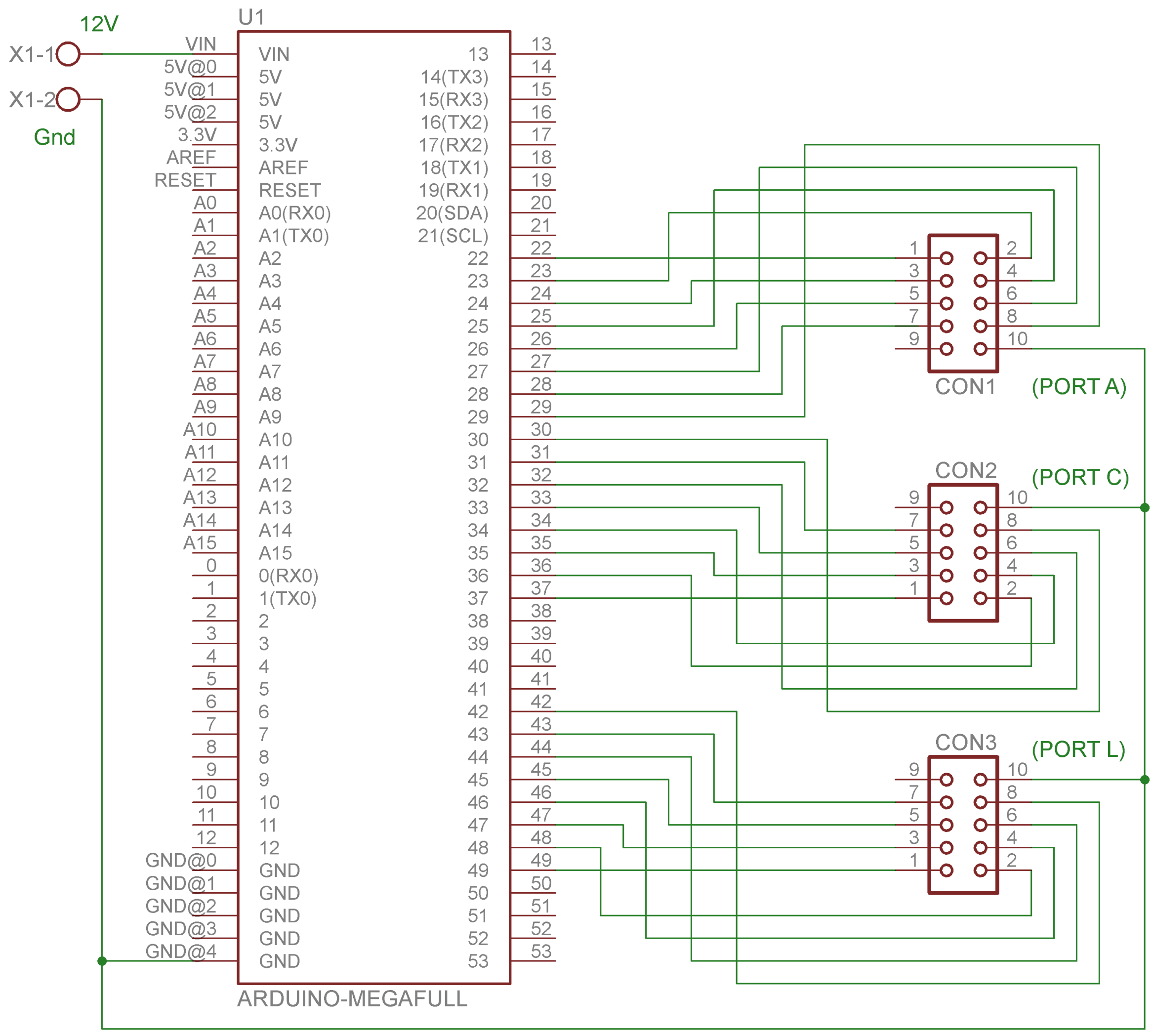


Figure 2.5: Schematics of the programmable 3 phase signal generator Part I

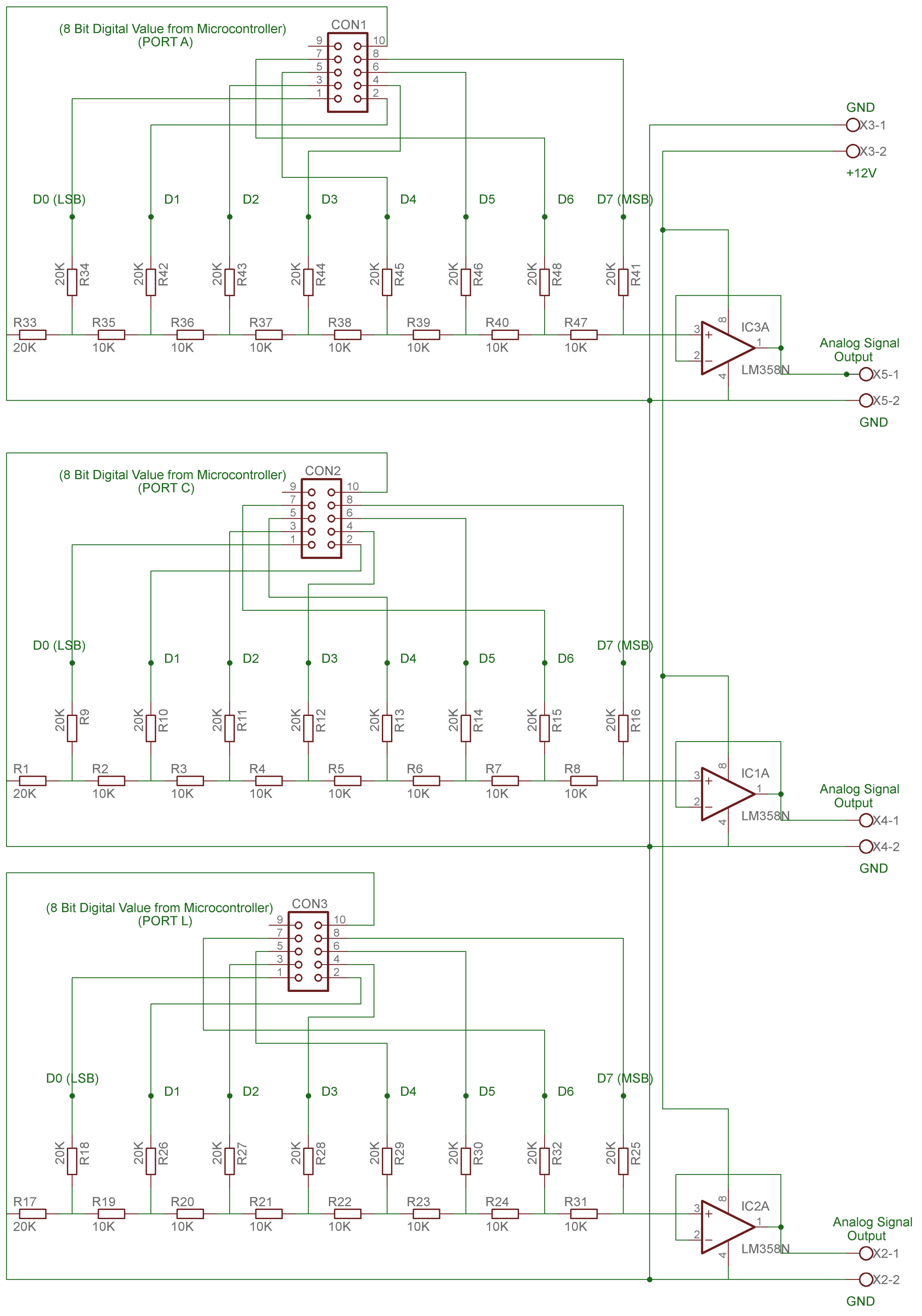


Figure 2.6: Schematics of the programmable 3 phase signal generator Part II

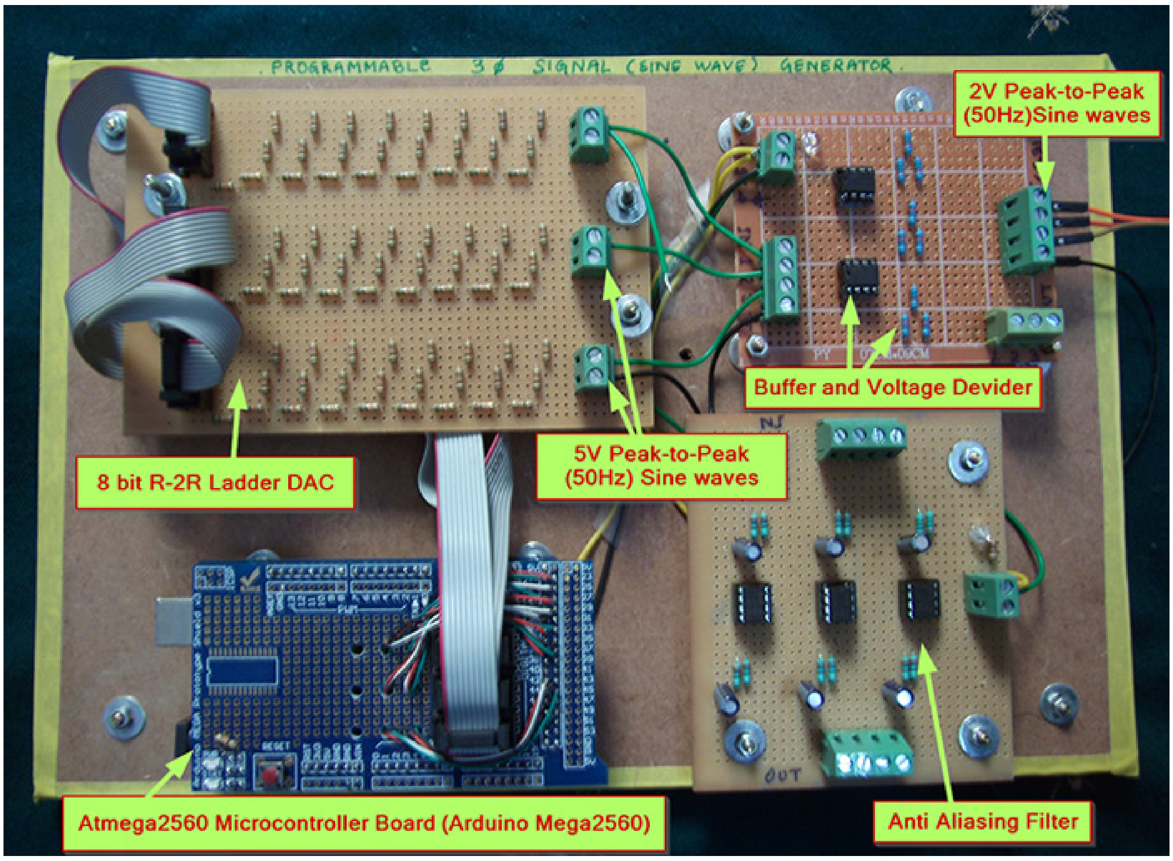

PROGRAMMABLE 3 ϕ SIGNAL (SINE WAVE) GENERATOR
2V Peak-to-Peak (50Hz)Sine waves
Buffer and Voltage Devider
8 bit R-2R Ladder DAC
5V Peak-to-Peak (50Hz) Sine waves
Atmega2560 Microcontroller Board (Arduino Mega2560)
Anti Aliasing Filter

Figure 2.7: Hardware of the programmable 3 Phase Signal generator

## 2.3 Method 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 is shown in Figure 2.8.

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.

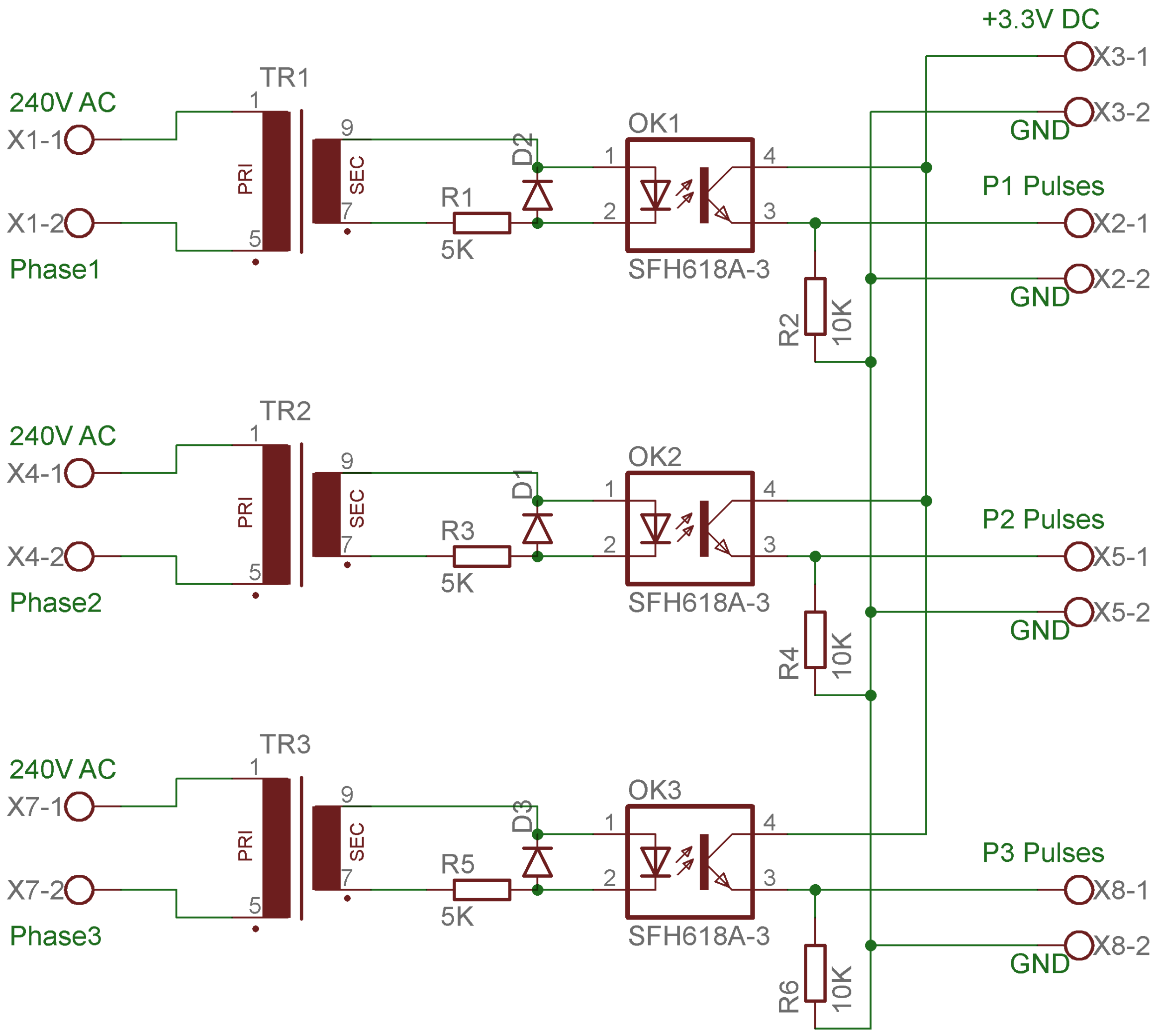


Figure 2.8: Schematics for AC to Square wave conversion

## 2.4 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 ClockIn. 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 $\mu$s width, separated by 400 $\mu$s interval. This results in generation of 3200 pulses per second. These pulses are available at the ClockOut pin. The flow chart of the programmed GPS disciplined oscillator is shown in Figure 2.10.

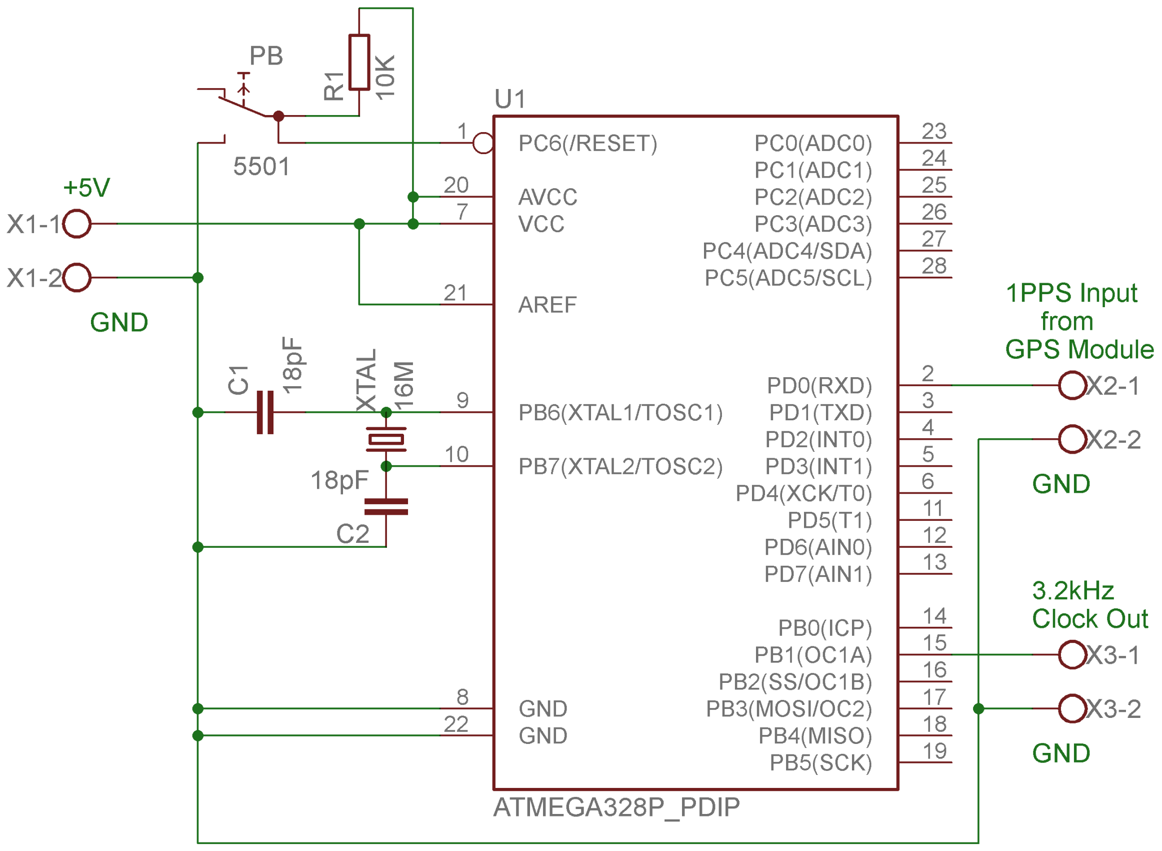


Figure 2.9: Schematics for the GPS Disciplined Oscillator

## 2.5 Sampling and Time Stamping

### 2.5.1 Sampling of the signals with in built ADC (Analog to Digital Converter)

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 $\mu$s. 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

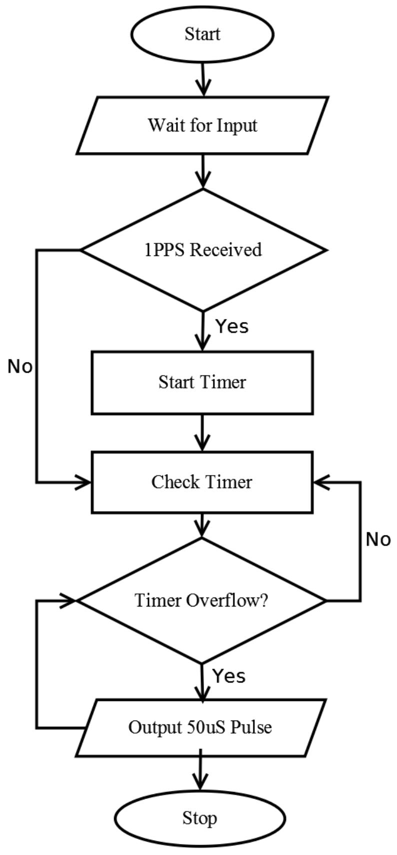


Figure 2.10: Flow Chart of the Programmed GPS Disciplined Oscillator

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.

### 2.5.2 Time Stamping with GPS Universal Coordinated Time(UTC)

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, whcih is then added to the phasor information during transmission. This method is known as the Time-Stamping of a Phasor.

## 2.6 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 in-built with the micro-controller which can very easily operate at 3200 samples per sec. The conversion time of ADC is 4 $\mu$s.

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, whcih 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.

Table 2.3: Components required to build the Phasor Processor

| **Quantity** | **DETAILS** | **UNIT PRICE (in RS)** | **LINE TOTAL** |
|---|---|---|---|
| 3 | Arduino Due Microcontroller Boards | 3700 | 11100 |
| 1 | Ublox Neo-6M GPS Module | 2800 | 2800 |
| 1 | Active 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** | **22700** |

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 .

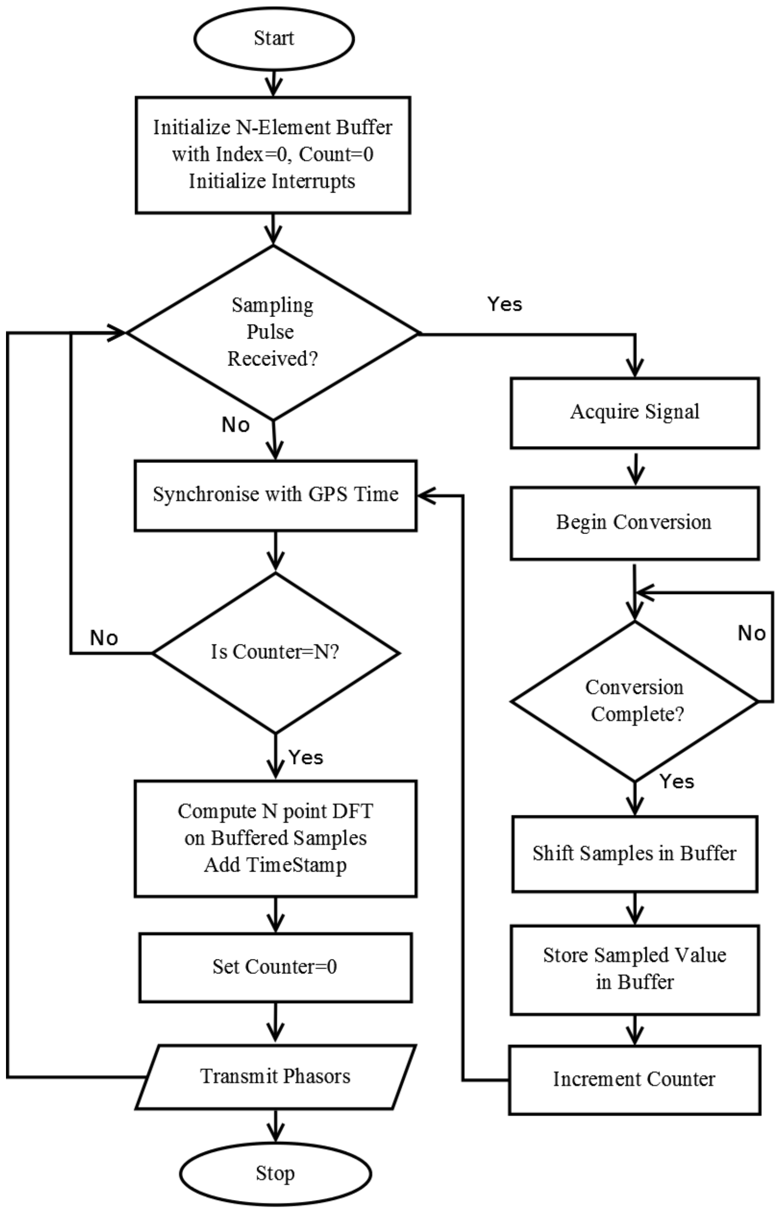


Figure 2.11: Flow Chart of the Phasor Processor

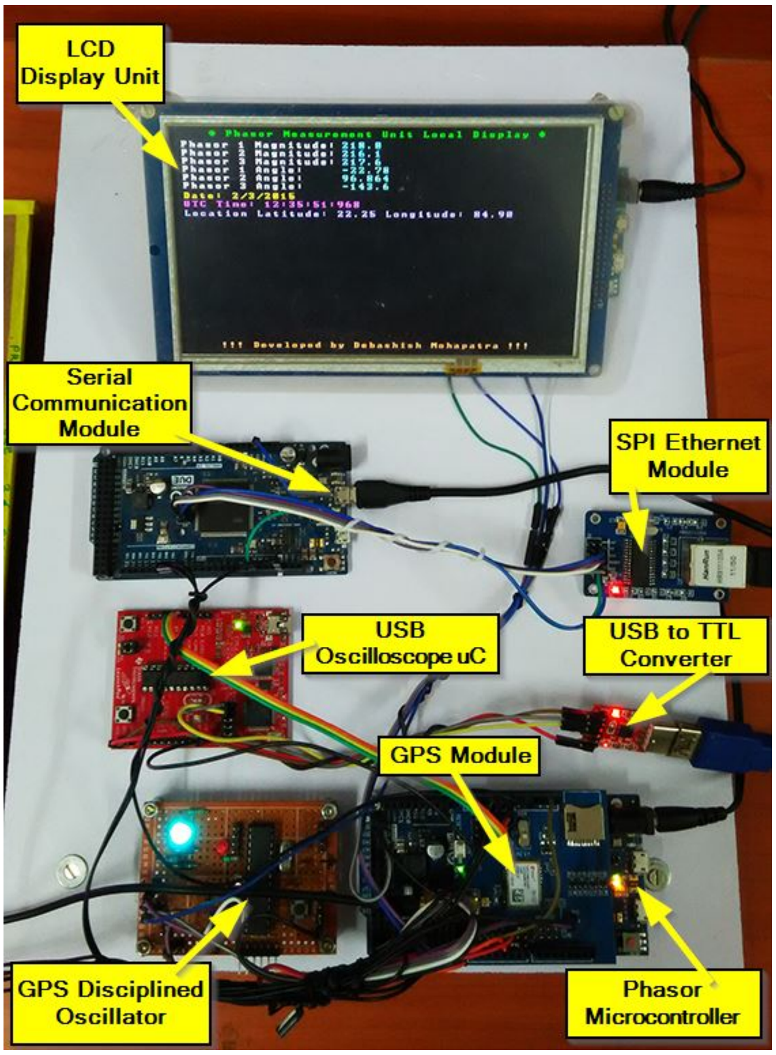


Figure 2.12: Hardware of the Phasor Processor

## 2.7 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 micro-controller, 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.

## 2.8 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.

### 2.8.1 Step-Down Transformers

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.

### 2.8.2 Diode Bridge Rectifiers

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, as shown in Figure 2.14.

### 2.8.3 Smoothing Capacitor

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.

### 2.8.4 Linear drop-out voltage regulator

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.

Table 2.4: Power requirement of the various module

| **Name of Module** | **Operating Voltage** |
|---|---|
| LEM LV-25P based voltage sensor module | +15V, -15V |
| Signal conditioning circuit | +3.3V |
| Sine to square wave converter circuit | +3.3V |
| PWM fan speed controller circuit | +5V |
| Cooling Fan (PWM) | +12V |
| Programmable sine wave generator | +12V |
| UTFT LCD Module | +12V |

Table 2.5: Components required to Build the Power Supply Unit

| Quantity | DETAILS | UNIT PRICE (in RS) | LINE TOTAL |
|---|---|---|---|
| 1 | 16x2 LCD | 180 | 180 |
| 1 | 12V 2200 RPM PWM Fan | 800 | 800 |
| 1 | Atmega328 Microcontroller Module | 250 | 250 |
| 1 | 16MHzCrystal, 22pF Caps, Push Button, IC Base etc. | 50 | 50 |
| 1 | Capaitors Pack (220uf, 3300uf, 10uf, .1uf, 2.2uf) | 200 | 200 |
| 6 | Heatsinks | 25 | 150 |
| 1 | 12-0-12V 5A Stepdown Transformer | 500 | 500 |
| 1 | 15-0-15 700mA Transformer | 100 | 100 |
| 2 | DCOM 6A Power Diodes | 8 | 16 |
| 7 | LDO ICs (7805,7812,Lm317,7815,7915) | 10 | 70 |
| 1 | DS18B20 Temperature Sensor | 400 | 400 |
| 10 | Screw Terminals | 10 | 100 |
| 1 | PCB, Solder, Wires etc. | 500 | 500 |
| | | **Net Total** | **3316** |

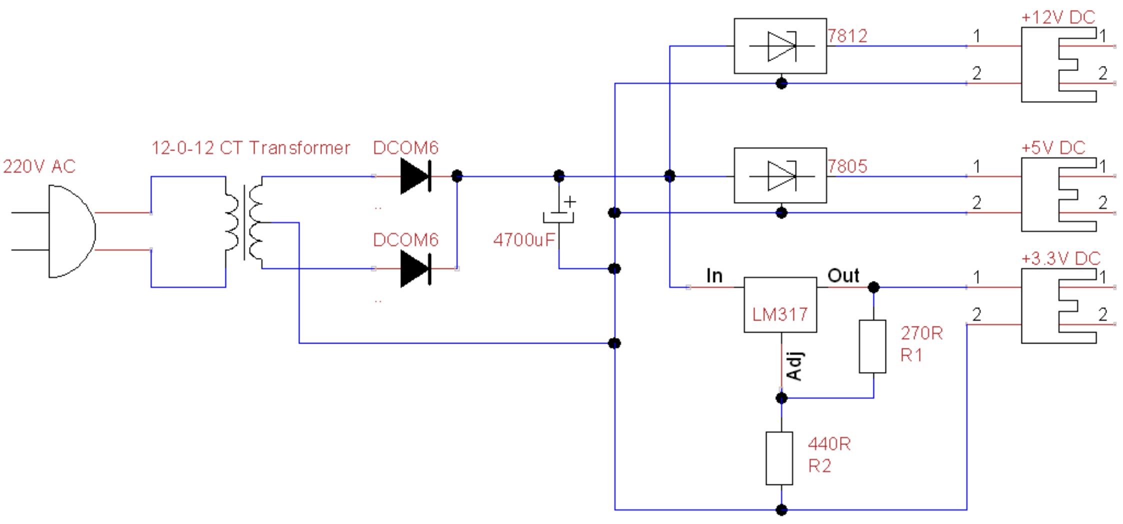


Figure 2.13: Schematics for the Positive Power Supply

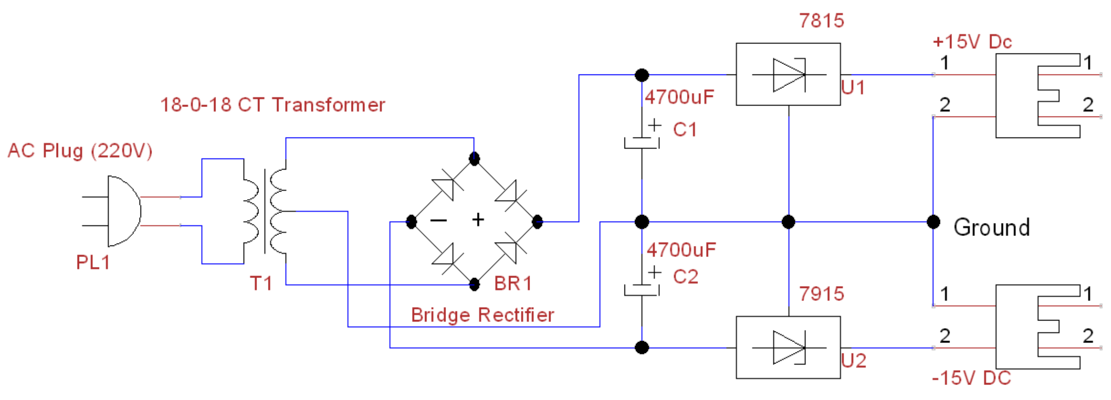


Figure 2.14: Schematics for the Dual Power Supply

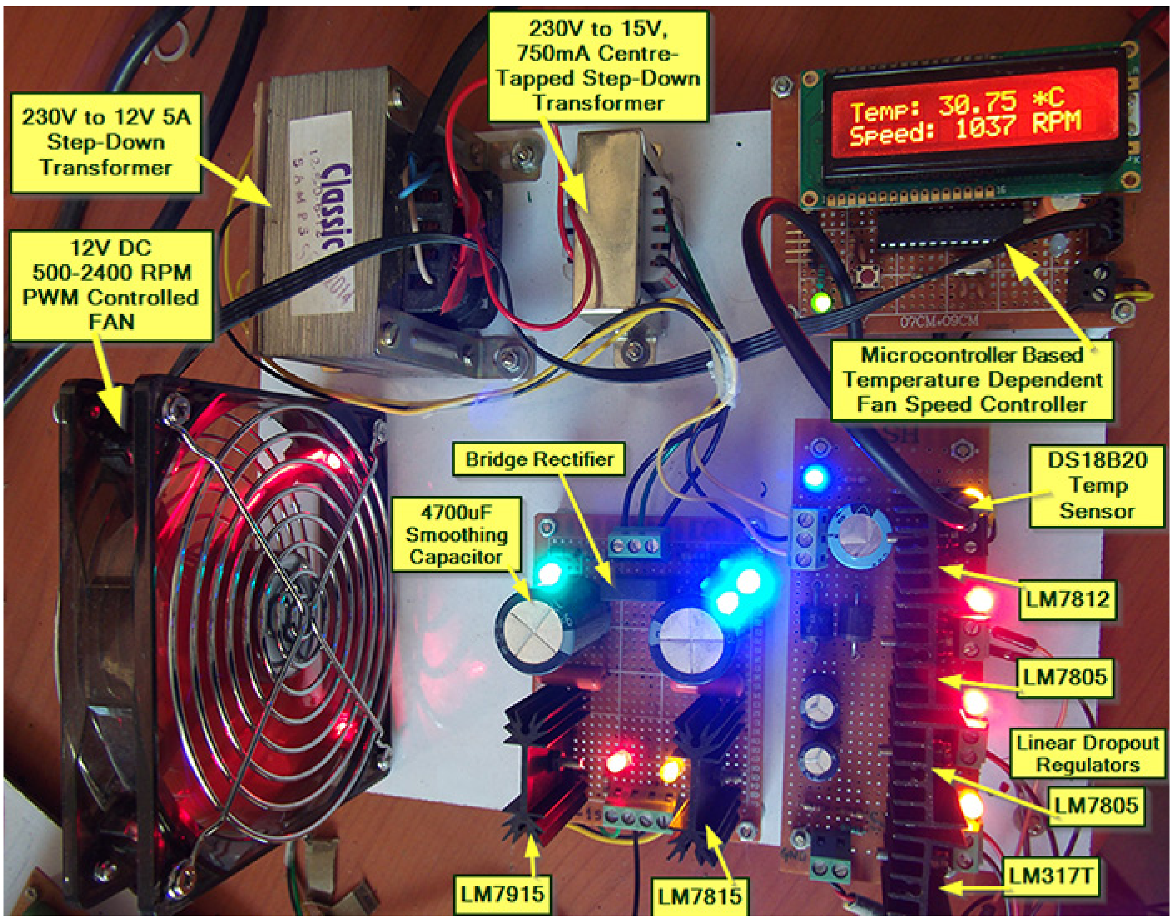

230V to 12V 5A Step-Down Transformer
230V to 15V, 750mA Centre-Tapped Step-Down Transformer
Temp: 30.75 *C
Speed: 1037 RPM
12V DC 500-2400 RPM PWM Controlled FAN
Microcontroller Based Temperature Dependent Fan Speed Controller
Bridge Rectifier
DS18B20 Temp Sensor
4700uF Smoothing Capacitor
LM7812
LM7805
Linear Dropout Regulators
LM7805
LM317T
LM7915
LM7815

Figure 2.15: The designed Power Supply Unit for PMU

CHAPTER 3

# Experimental Results and Discussion

This section describes the experimental setup and the results obtained from the developed laboratory prototype PMU.

## 3.1 The Experimental Setup for Testing of the PMU

A 3-Phase sine wave test signal generator is built with Atmel Atmega2560 8-bit AVR micro-controller. The micro-controller was chosen because of its availability of Input/Output Ports (six completely accessible ports), of which one port (8 pins) is required to generate one sine wave. In this prototype setup, only three of the ports were used to generate three sine waves whose magnitude and phases can be changed with programming. The Phasor micro-controller was connected to these signals, which after calculating the phasors and time stamping them reported to the local display, as well as to the computer which displays the data in real time. The Laboratory Setup for evaluating the PMU is shown in Figure 3.1.

After testing the PMU with the synthesized AC signal for various frequencies, the PMU was connected to actual three-phase four-wire AC supply system available in the Laboratory. The voltage signal acquisition board and the voltage to square wave converter circuit replaced the three phase sine wave generator. Then the MCB was turned on and the Phasors were monitored in the real-time GUI developed in Python.

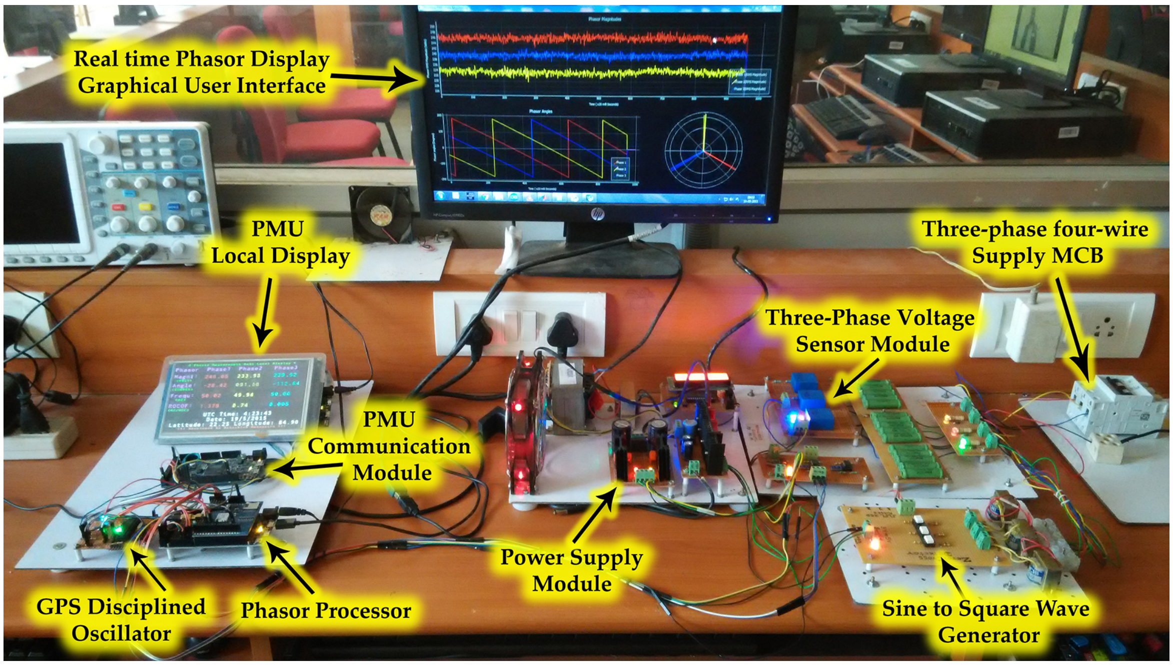


Figure 3.1: The Developed Laboratory Prototype of the PMU

The developed GPS Disciplined Oscillator(GPSDO) generates the required sampling pulses i.e. 3200 pulses per second, which are synchronised with the arrival of the 1 PPS signal from the GPS module. The pulse from the GPS module is shown in Figure 3.2(a) and the generated sampling pulses by the GPSDO are shown in Figure 3.2(b).

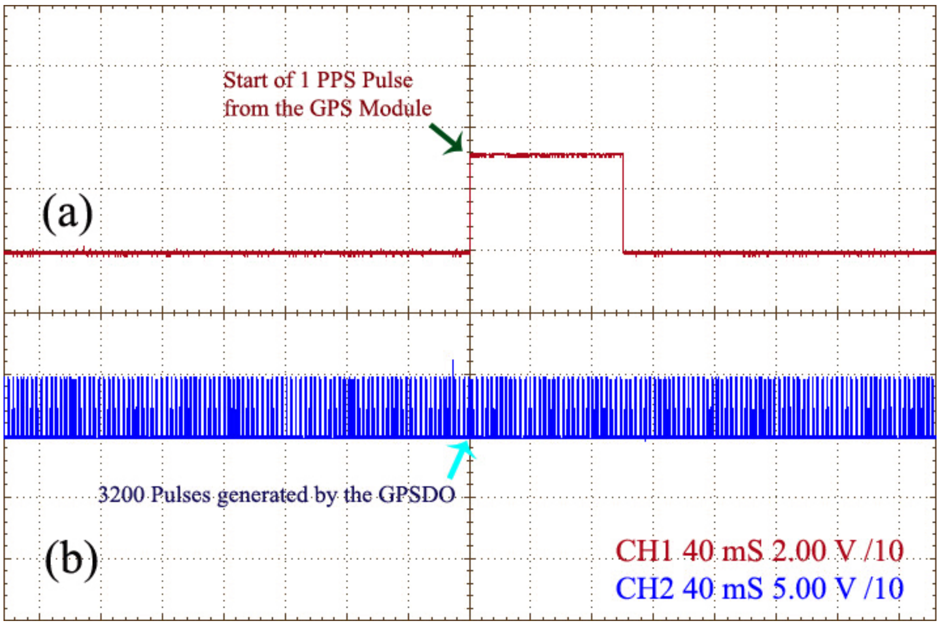


Figure 3.2: Pulses from the GPSDO

To measure the frequency of the three phases a sine wave to square wave converter was designed. The input and output waveforms are shown in Figure 3.3.

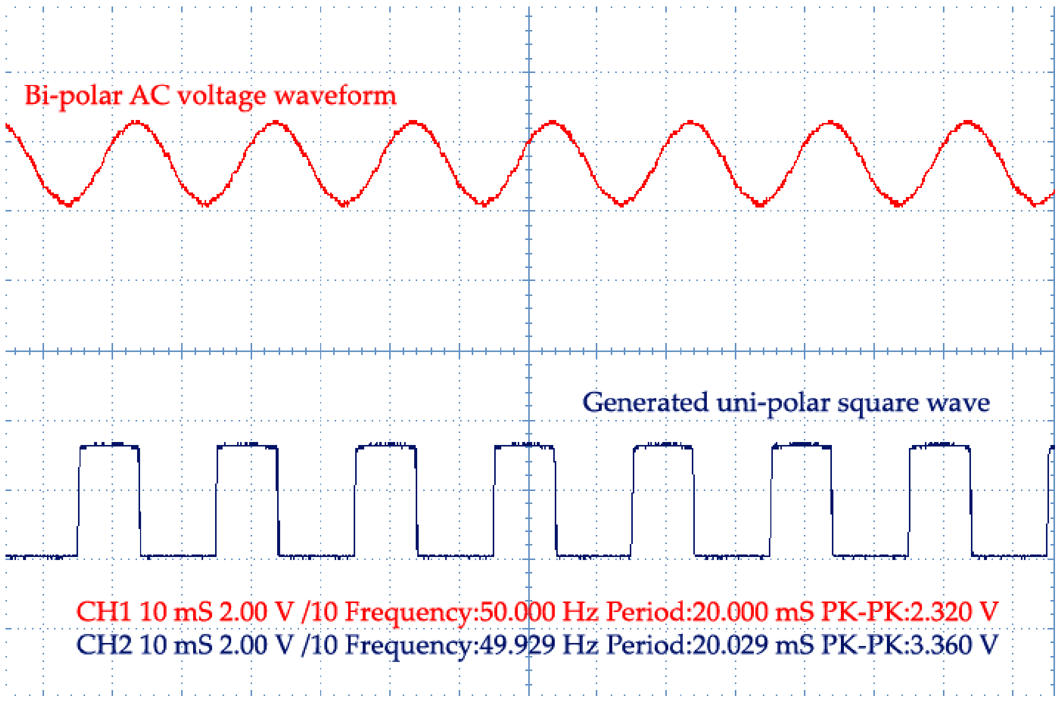
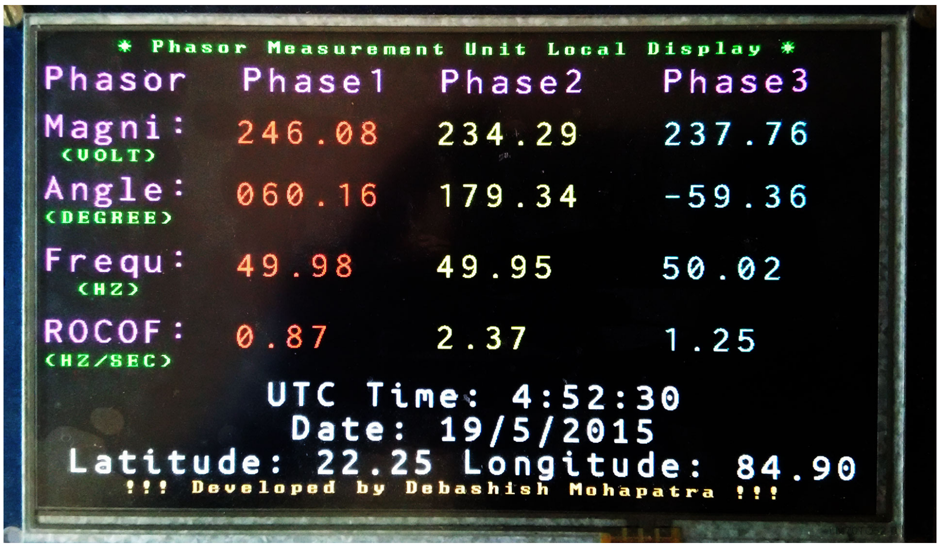


Figure 3.3: Square wave pulses for frequency measurement

## 3.2 Method of PMU Data acquisition and Plotting

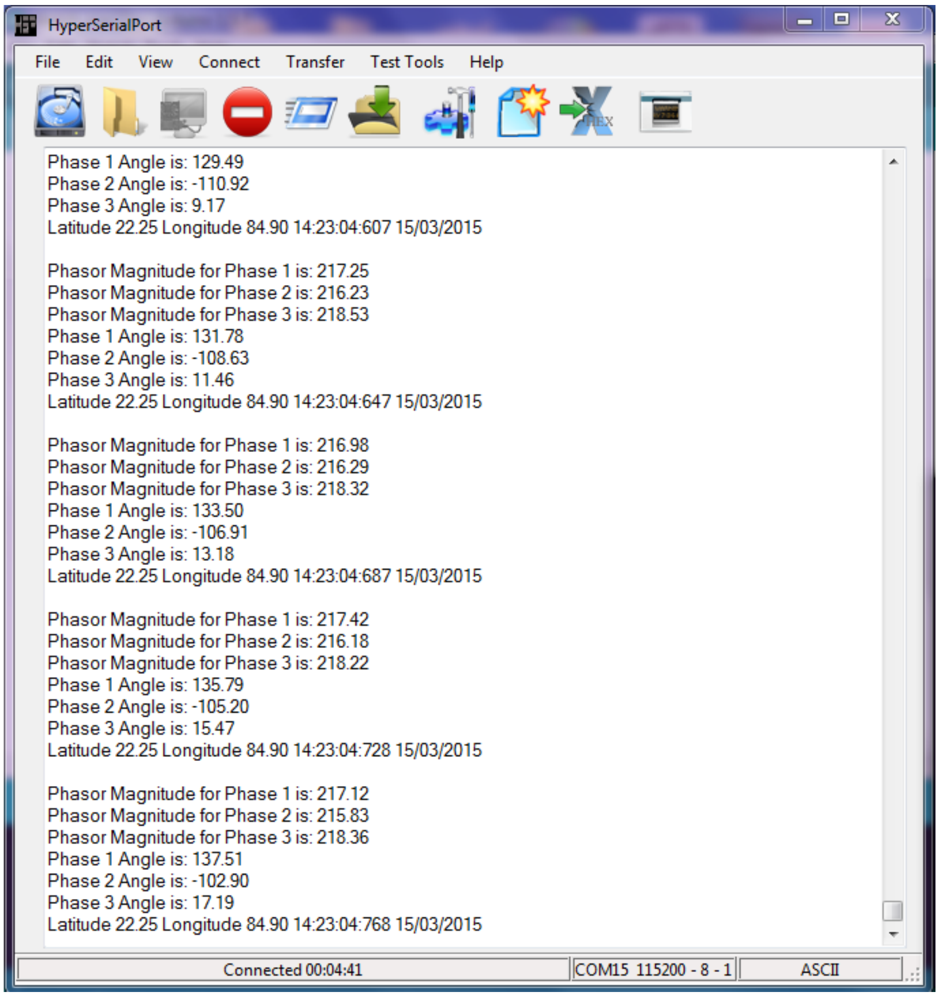
The developed PMU reports the calculated phasors at a rate of 50 phasors per second. The phasors can be seen on the LCD (PMU local display) as shown in Figure 3.4. This reporting includes the three-phase signals RMS magnitudes and the phase angles, the frequency and the ROCOF with a time stamp. This data is reported over serial terminal of the micro-controller, which is interfaced with the computer's USB Port.

The reported phasors from the PMU are displayed in readable text in a Serial Terminal software as shown in Figure 3.5. The phasors can also be accessed over a web interface with a web browser pointing to the IP address of the PMU in the Local Area Network(LAN) which is shown in Figure 3.6. The PMU is connected to the Network with the help of a Serial Peripheral Interface(SPI) Ethernet Module. The three-phase voltage signals, as acquired by another Microcontroller are shown in Figure 3.7. The X-axis represents the sampling instants and the Y-Axis displays the 10-bit integer value of the signal which is generated by the Analog to Digital Converter(ADC).



```
* Phasor Measurement Unit Local Display *
Phasor          Phase1    Phase2    Phase3
Magni:          246.08    234.29    237.76
<VOLT>
Angle:          060.16    179.34    -59.36
<DEGREE>
Frequ:          49.98     49.95     50.02
<HZ>
ROCOF:          0.87      2.37      1.25
<HZ/SEC>
             UTC Time: 4:52:30
              Date: 19/5/2015
 Latitude: 22.25 Longitude: 84.90
 !!! Developed by Debashish Mohapatra !!!
```

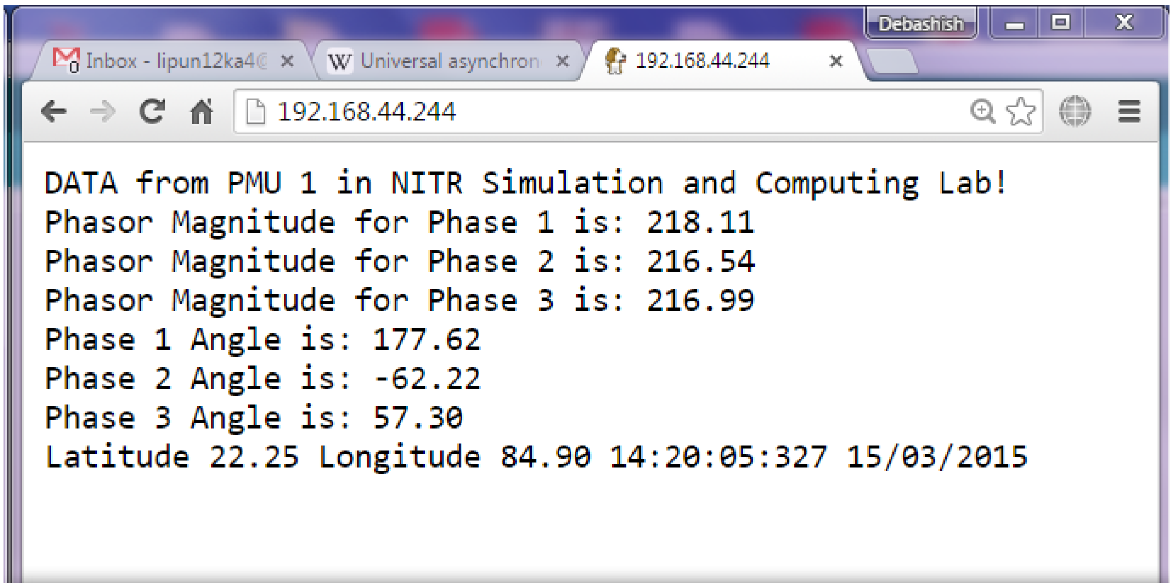
Figure 3.4: Phasors displayed on the PMU LCD



```
Phase 1 Angle is: 129.49
Phase 2 Angle is: -110.92
Phase 3 Angle is: 9.17
Latitude 22.25 Longitude 84.90 14:23:04:607 15/03/2015

Phasor Magnitude for Phase 1 is: 217.25
Phasor Magnitude for Phase 2 is: 216.23
Phasor Magnitude for Phase 3 is: 218.53
Phase 1 Angle is: 131.78
Phase 2 Angle is: -108.63
Phase 3 Angle is: 11.46
Latitude 22.25 Longitude 84.90 14:23:04:647 15/03/2015

Phasor Magnitude for Phase 1 is: 216.98
Phasor Magnitude for Phase 2 is: 216.29
Phasor Magnitude for Phase 3 is: 218.32
Phase 1 Angle is: 133.50
Phase 2 Angle is: -106.91
Phase 3 Angle is: 13.18
Latitude 22.25 Longitude 84.90 14:23:04:687 15/03/2015

Phasor Magnitude for Phase 1 is: 217.42
Phasor Magnitude for Phase 2 is: 216.18
Phasor Magnitude for Phase 3 is: 218.22
Phase 1 Angle is: 135.79
Phase 2 Angle is: -105.20
Phase 3 Angle is: 15.47
Latitude 22.25 Longitude 84.90 14:23:04:728 15/03/2015

Phasor Magnitude for Phase 1 is: 217.12
Phasor Magnitude for Phase 2 is: 215.83
Phasor Magnitude for Phase 3 is: 218.36
Phase 1 Angle is: 137.51
Phase 2 Angle is: -102.90
Phase 3 Angle is: 17.19
Latitude 22.25 Longitude 84.90 14:23:04:768 15/03/2015
```

Connected 00:04:41 | COM15 115200 - 8 - 1 | ASCII

Figure 3.5: A screenshot of the reported data from PMU over Serial Terminal



```
DATA from PMU 1 in NITR Simulation and Computing Lab!
Phasor Magnitude for Phase 1 is: 218.11
Phasor Magnitude for Phase 2 is: 216.54
Phasor Magnitude for Phase 3 is: 216.99
Phase 1 Angle is: 177.62
Phase 2 Angle is: -62.22
Phase 3 Angle is: 57.30
Latitude 22.25 Longitude 84.90 14:20:05:327 15/03/2015
```

Figure 3.6: A screenshot of the reported data from PMU over Ethernet(LAN)

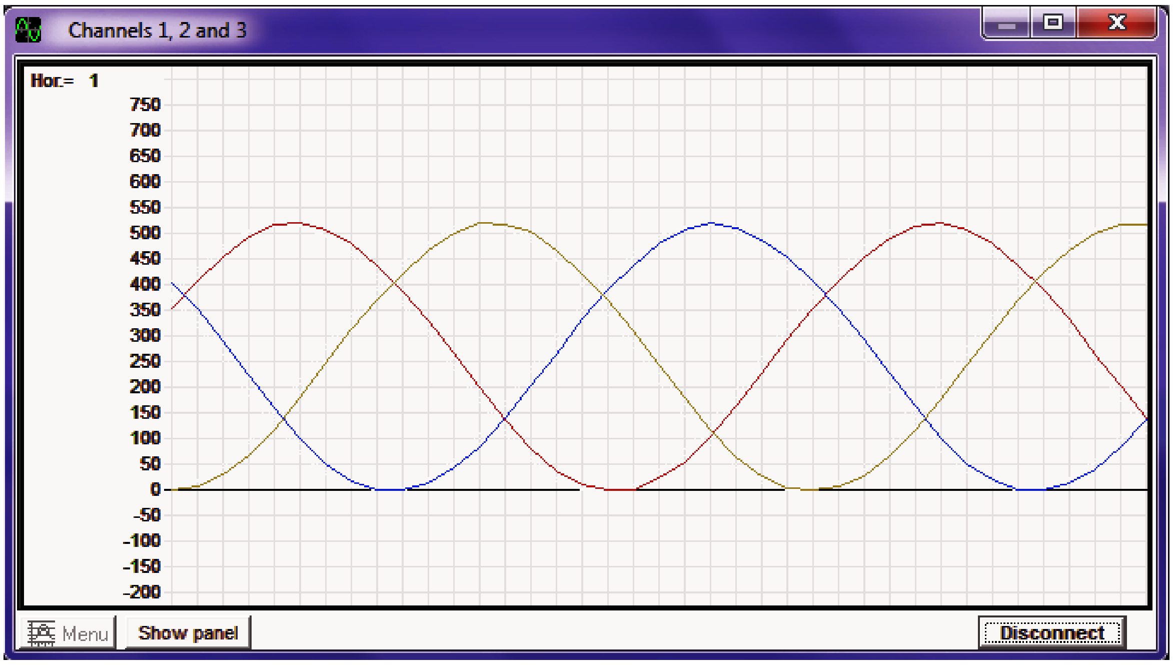


Figure 3.7: The acquired 3-phase voltage samples by the ADC

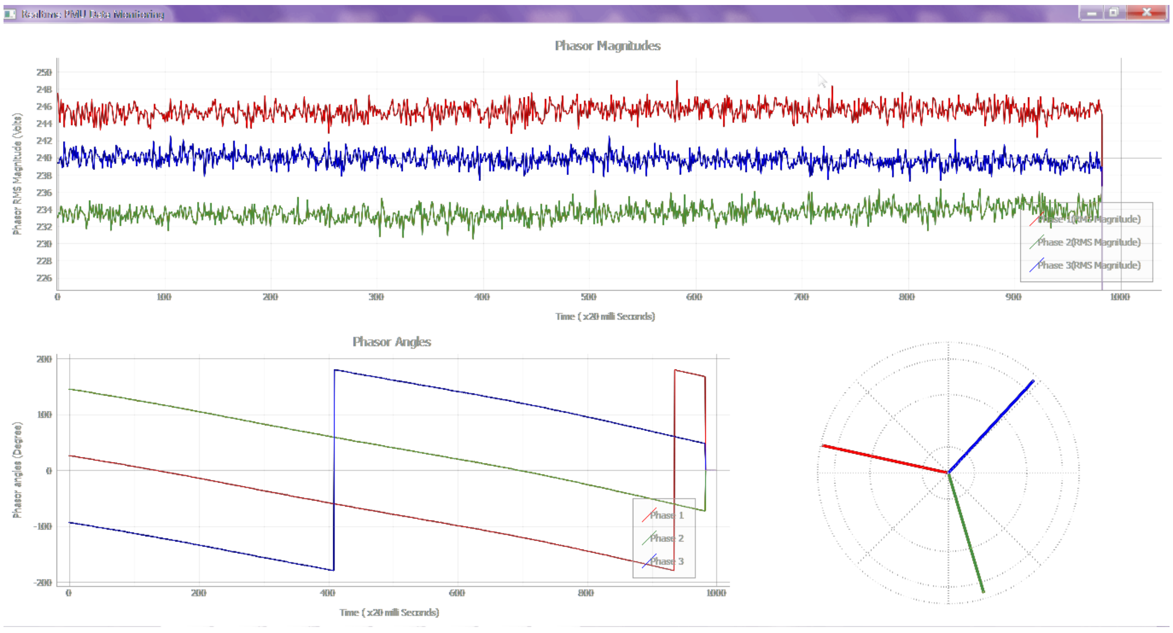


Figure 3.8: A screenshot of the Developed Python GUI for PMU

To acquire the data from the PMU, a program is developed in Python programming language. It opens the serial port to which the PMU is transmitting the Phasors, and reads the data stream. Then it separates the various parameters of the data stream into float data types and plots them in a Graphical User interface (GUI) created using the PyQtGraph library for Python. The GUI is shown in Figure 3.8.

The python program also logs the phasor data, which is being received from the PMU as Comma Separated Values (CSV) in a Text file, which can be further analysed in the future if needed. However for true reputability of the Phasor data in the future, it will require a lot of data logging or data storage space. The Text file or CSV file method of data logging will not be able to satisfy such a requirement. Hence a true database management system like Oracle or MySQL database systems need to be incorporated into the PDC system with terabytes of HDD to store the data.

To plot the logged data, another Python script is developed, which reads the Phsor data from the log file, separates the different phasor parameters and plots them using MatPlotLib library for Python.

## 3.3 Analysis of the Phasors reported by the PMU

Three test signals were generated by the designed 3 phase signal generator for frequencies of 49.95Hz, 49.65Hz and 50.30Hz. These signals were given as input to the PMU. The reported Phasor magnitudes(in RMS Volts) and the angles(in Degree) are shown in Figure 3.9, Figure 3.10 and Figure 3.11 respectively. The Mean Square Error of the reported phasor magnitudes for the three signals with different frequencies are given in Table 3.1.

The Phasors were also observed in real-time when the PMU was connected to the 3-phase ac supply available in the laboratory. Since the supply frequency is never quite constant, i.e. it keeps changing depending on the loads, and the compensating actions by the generating stations, the rotating rate of the phasors in the polar plot was observed to varying. However , At some instant of time, when the frequency is exactly 50.00 Hz, it was noticed that the ploar plot keeps stationary and the plot of phasor angles did not have a slope any more, rather they were completely horizontal lines. This phenomenon can be exploited to detect which lines are having the maximum frequency deviation, just by giving a quick look, into the phasor polar plot.

It was observed that the measurement error increases as the frequency of the signal deviates from the standard frequency, according to which the sampling window and the sampling intervals were chosen. To get more accurate measurement it is desired to develop an algorithm which takes into account the change of frequency while calculating the phasors and adjusts its sampling intervals and the sampling window so that using DFT based Phasor calculation would yield more accurate results. A FFT based algorithm will most certainly overcome these drawbacks of the DFT algorithm. However the instantaneous values of phasors, as reported by the DFT algorithm will be not be truely instantaneous any more. Moreover the choice of the sampling window type and size need to follow an adaptive algorithm which takes the changes in system parameters, especially frequency into account for calculating the phasor magnitudes so that it will produce more accurate results then the currently developed system.

## 3.4 Discussion

From Fig 3.10, Fig 3.9 and Fig 3.11 it can be observed that the more the deviation of the signal frequency from the nominal value, the more is the deviation of the measured phasor magnitudes. The calculated Mean Square Error is shown in Table 3.1.

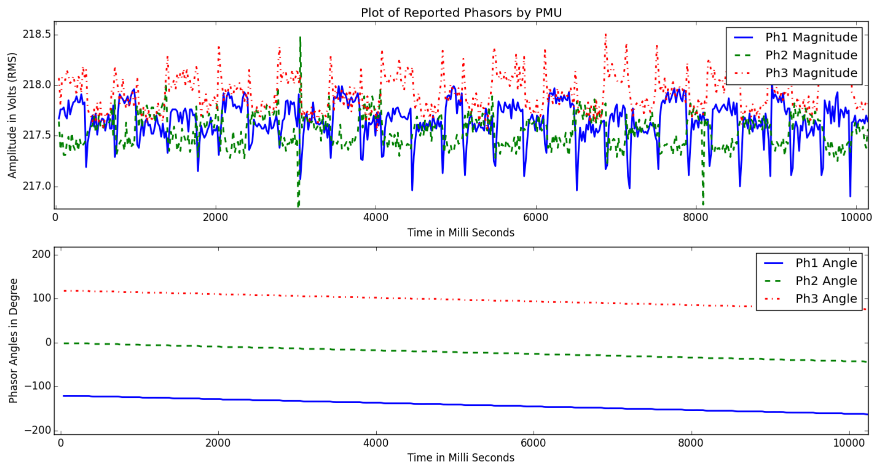


Figure 3.9: Reported Phasors by the PMU at a frequency of 49.95 Hz

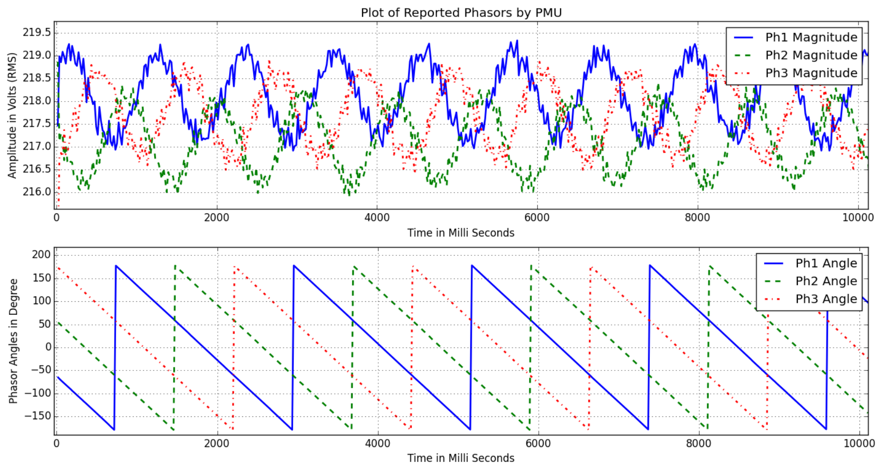


Figure 3.10: Reported Phasors by the PMU at a frequency of 49.65 Hz

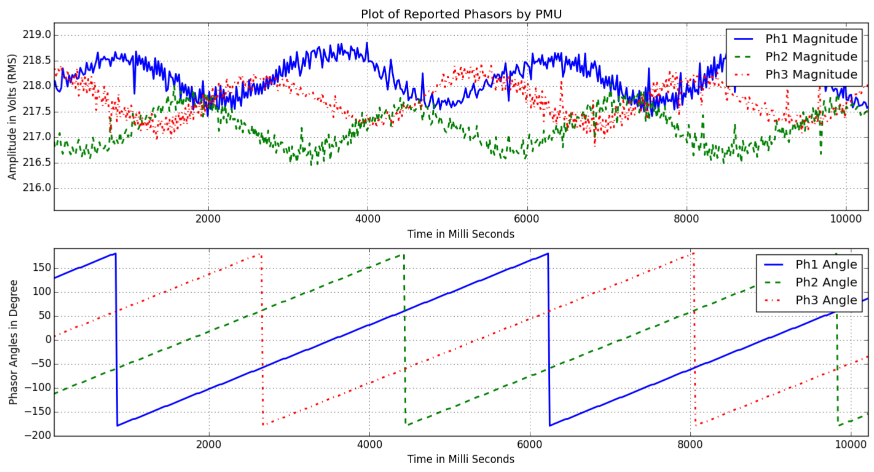


Figure 3.11: Reported Phasors by the PMU at a frequency of 50.30 Hz

The Mean Square Errors shown in Table 3.1 suggest that the magnitudes of error for all the three test signal frequencies of a balanced three phase system remains in the compliance boundaries as stated by the IEEE Standard of Synchrophasor Measurement.

Table 3.1: Mean Square Error of the Phasor Magnitudes

| **Signal Frequency** | **Mean Square Error (in Volts)** | | |
|---|---|---|---|
| **(in Hz)** | **Phase1** | **Phase 2** | **Phase 3** |
| 49.65 | 0.4736 | 1.2111 | 0.5745 |
| 49.95 | 0.1642 | 0.2708 | 0.0342 |
| 50.30 | 0.1527 | 0.8452 | 0.1964 |

The calculated phasors are precisely time stamped. By making the data acquisition (sampling) process handled by a ISR, the CPU is free to do other tasks like calculating the phasors, transmission/communication of the phasors, synchronising it's clock with GPS time etc., which enables increase in the reporting rate to 50 Phasors per second. Moreover, the communication being directly handled by the CPU can be made DMA enabled, which has the potential to free up the CPU even more. This extra CPU time can be used to further calculate more analytical parameters, such as, estimation of harmonics etc.

Moreover, use of a faster and yet, economical single board computing platform which incorporates more CPU Power, RAM and advanced communication features can be used along with a high speed external ADC to improve the accuracy of the Phasors being reported.The compliance with the IEEE Standard for Synchrophasor measurements is given in Table 3.2.

Table 3.2: IEEE Standard Compliance of the PMU

| Compliance Requirement | Balanced 3 Phase System | |
|---|---|---|
| | **Nominal Frequency** | **Off-Nominal Frequency** |
| Sampling Rate | Yes | Yes |
| Reporting Rate | Yes | Yes |
| Magnitude Error | Yes | Yes |
| Frequency | Yes | Yes |
| Rate of change of Frequency | Yes | Yes |
| Time Stamp | Yes | Yes |

Table 3.3: Expenditures in building a low-cost prototype PMU

| Name of Module | Cost (in INR) |
|---|---|
| LEM LV-25P based voltage signal acquisition board | 8175.00 |
| Sine to square wave converter circuit | 500.00 |
| PMU Main Unit | 22700.00 |
| Programmable sine wave generator | 4730.00 |
| Power Supply | 3316.00 |
| **Net Total** | **39421** |

The expenditures in building a low cost prototype PMU is given in Table 3.3. It is quite remarkable how using open source platforms for developing a new prototype can dramatically reduce the cost of a product. A PMU could be built only with approximately one-tenth of the price tags of the commercial PMUs available in the market.

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 user-friendly 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.

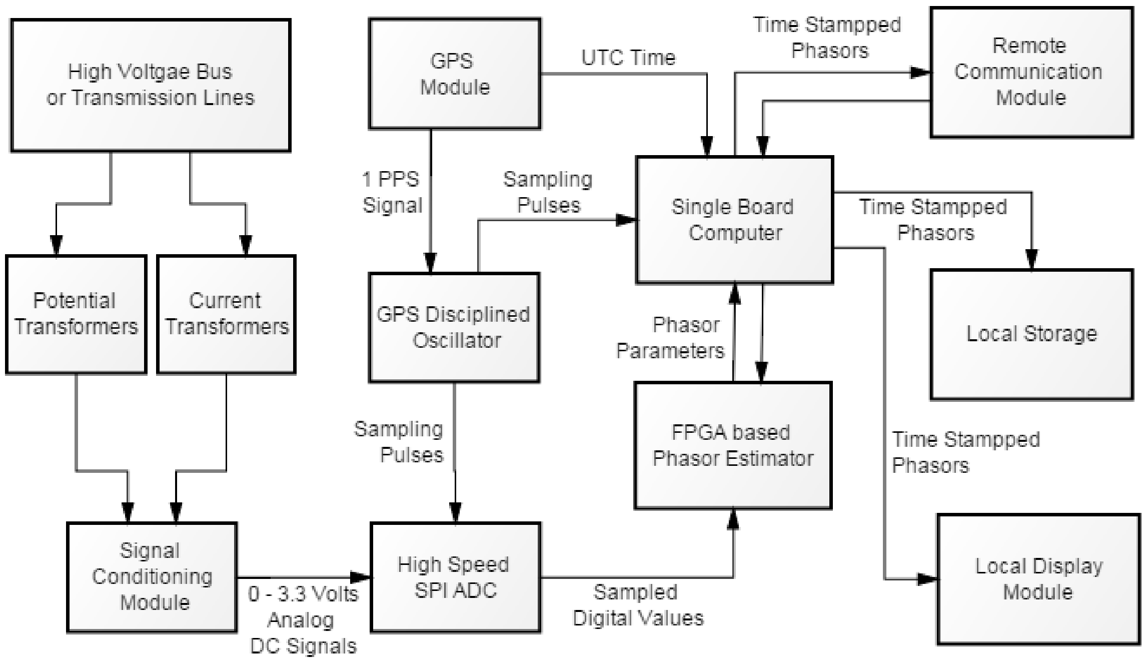


Figure 4.1: Revised Architechure of the Proposed PMU

## 4.1 Proposed directions 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.

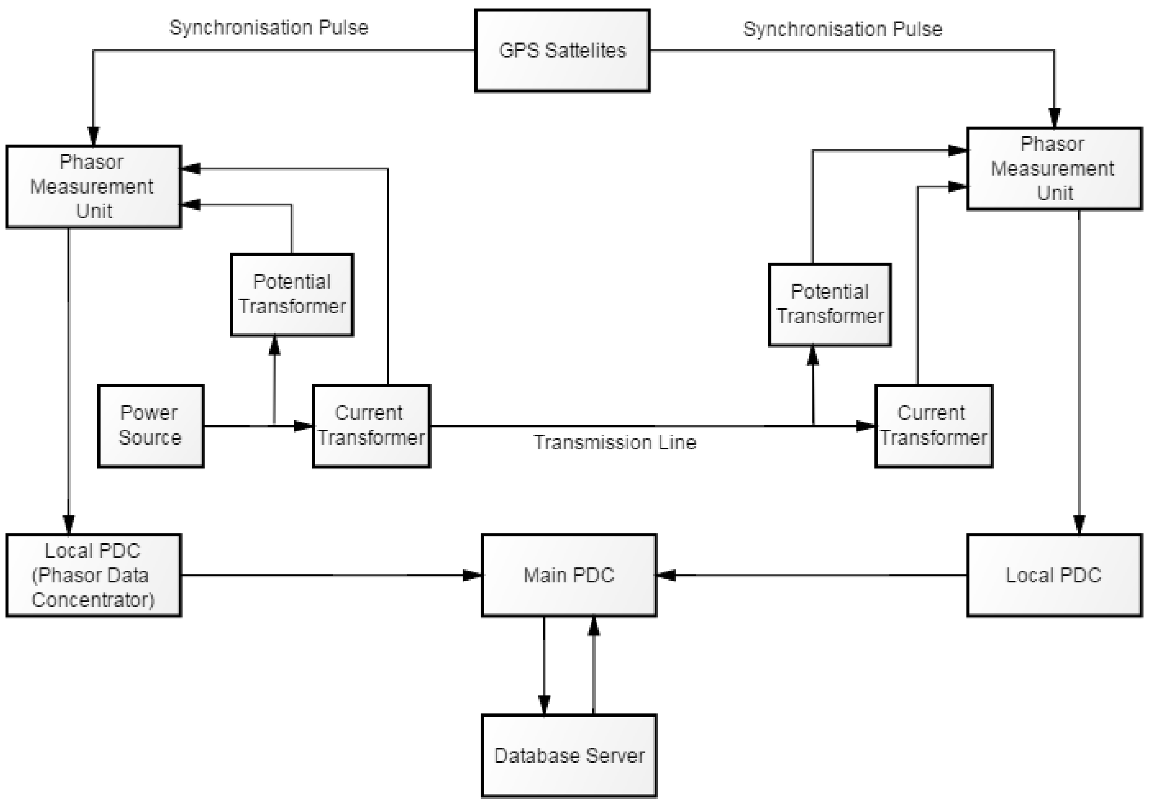


Figure 4.2: Proposed framework for testing the PMUs

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.

# References

[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.

[11] D. M. Laverty, D. J. Morrow, R. Best, and P. A. Crossley, Performance of phasor measurement units for wide area real-time control, in Proc. IEEE PES Gen. Meeting, Jul. 2630, 2009, pp. 1-5.

[12] A. J. Roscoe, I. F. Abdulhadi, and G. M. Burt, P-class phasor measurement unit algorithms using adaptive filtering to enhance accuracy at offnominal frequencies, in Proc. IEEE Int. Conf. SMFG, Nov. 14-16, 2011, pp. 51-58.

[13] A. Carta, N. Locci, C. Muscas, and S. Sulis, A flexible GPS-based system for synchronized phasor measurement in electric distribution networks, IEEE Trans. Instrum. Meas., vol. 57, no. 11, pp. 2450-2456, Nov. 2008.

[14] openPDC, Open Source Phasor Data ConcentratorGrid Protection Alliance, Codeplex Project Page. [Online]. Available: http://openpdc.codeplex.com/

[15] A. J. Stadlin, GridTrak Open Source Synchrophasor PMU Project, Codeplex Project Page, [Accessed: Feb. 15, 2012]. [Online]. Available: http://gridtrak.codeplex.com/

[16] Garcia-Valle et al., DTU PMU laboratory developmentTesting and validation, in Proc. IEEE ISGT Europe, Oct. 11-13, 2010, pp. 1-6.

[17] A. H. Nielsen, K. O. Helgesen Pedersen, and O. Samuelsson, An experimental GPS-based measurement unit, in Proc. Nordic Baltic Workshop Power Syst., Tampere, Finland, Feb. 4-5, 2002, pp. 1-6.

[18] L. Vanfretti, R. Garcia-Valle, K. Uhlen, E. Johansson, D. Trudnowski,J. W. Pierre, J. H. Chow, O. Samuelsson, J. stergaard, and K. E. Martin, Estimation of Eastern Denmarks electromechanical modes from ambient phasor measurement data, in Proc. IEEE PES Gen. Meeting, Jul. 25-29, 2010, pp. 1-8.

[19] A Report to Congress Pursuant to Section 1839 of the Energy Policy Act of 2005, US Dept. Energy Fed. Energy Regulat. Commiss., Washington, DC, USA, Feb. 2006.

[20] Peng Zhang; Ka Wing Chan, "Reliability Evaluation of Phasor Measurement Unit Using Monte Carlo Dynamic Fault Tree Method", IEEE Transactions on Smart Grid, vol.3, no.3, pp.1235-1243, Sept. 2012

[21] Murthy, C.; Mishra, A.; Ghosh, D.; Roy, D.S.; Mohanta, D.K., "Reliability Analysis of Phasor Measurement Unit Using Hidden Markov Model", IEEE Systems Journal, vol.8, no.4, pp.1293-1301, Dec. 2014

[22] Yang Wang; Wenyuan Li; Peng Zhang; Bing Wang; Jiping Lu, "Reliability Analysis of Phasor Measurement Unit Considering Data Uncertainty", IEEE Transactions on Power Systems, vol.27, no.3, pp.1503-1510, Aug. 2012

[23] V.K.Agrawal; P.K.Agarwal; Harish Rathour, "Application of PMU Based Information in Improving the Performance of Indian Electricity Grid", URL "http://indiwams.posoco.in/attachments/article/1/12144.pdf"

[24] Laverty, D.M.; Best, R.J.; Brogan, P.; Al Khatib, I.; Vanfretti, L.; Morrow, D.J., "The OpenPMU Platform for Open-Source Phasor Measurements", IEEE Transactions on Instrumentation and Measurement, vol.62, no.4, pp.701-709, April 2013

[25] Laverty, D.M.; Vanfretti, L.; Al Khatib, I.; Applegreen, V.K.; Best, R.J.; Morrow, D.J., "The OpenPMU Project: Challenges and perspectives", IEEE Power and Energy Society General Meeting (PES), 2013, vol., no., pp.1,5, 21-25 July 2013

[26] Laverty, D.M.; Vanfretti, L.; Best, R.J.; Morrow, D.J.; Nordstrom, L.; Chenine, M., "OpenPMU technology platform for Synchrophasor research applications", IEEE Power and Energy Society General Meeting, 2012, vol., no., pp.1,5, 22-26 July 2012

# Appendices

Appendix A

# Arduino Codes for Microcontrollers

## A.1 Arduino code for the GPS Disciplined Oscillator

```
#include <TimerOne.h>
// use this library to handle the Timer functionality
const int sampling_clock_out_pin = 9;
// the sampling pulses will be generated at this pin
void setup()
{
  pinMode(2, INPUT);
  // This is the pin where the 1 PPS pulse from GPS module is connected
  attachInterrupt(0, pulsePPS, RISING);//watch out for interrupt (1 PPS) on pin 2
  //Timer1.initialize(400);//for 2500 pulses per sec
  Timer1.initialize(312.5);//for 3200 pulses per sec
  Timer1.pwm(sampling_clock_out_pin, 100);//duty cycle of the pulse, i.e. about 100uS
}
void loop()
{ /* Since the microcontrollers timer operate independently without invoking the CPU, and the Interrupt handlers
  takes care of the ISR, there is nothing to do in the loop*/
}
void pulsePPS() //interrupt routine upon receiving PPS
{
  Timer1.restart();
  /*Just restart the timer, to keep it in sync with the GPS module's PPS pulses*/
}
```

## A.2 Arduino code for the 3 Phase Signal Generator

```
#include <JeeLib.h>
#include <avr/pgmspace.h>

//Define the sine wave look-up tables which caontains the 8-bit integers
byte sineR256[] PROGMEM = {
  128,131,134,137,140,143,146,149,152,155,158,162,165,167,170,173,
  176,179,182,185,188,190,193,196,198,201,203,206,208,211,213,215,
  218,220,222,224,226,228,230,232,234,235,237,238,240,241,243,244,
  245,246,248,249,250,250,251,252,253,253,254,254,254,255,255,255,
  255,255,255,255,254,254,254,253,253,252,251,250,250,249,248,246,
  245,244,243,241,240,238,237,235,234,232,230,228,226,224,222,220,
  218,215,213,211,208,206,203,201,198,196,193,190,188,185,182,179,
  176,173,170,167,165,162,158,155,152,149,146,143,140,137,134,131,
  128,124,121,118,115,112,109,106,103,100,97,93,90,88,85,82,79,76,
  73,70,67,65,62,59,57,54,52,49,47,44,42,40,37,35,33,31,29,27,25,
  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,
  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,
  29,31,33,35,37,40,42,44,47,49,52,54,57,59,62,65,67,70,73,76,79,
  82,85,88,90,93,97,100,103,106,109,112,115,118,121,124};

byte sineY256[] PROGMEM = {
  238,237,235,  234,232,230,228,226,224,222,220,218,215,213,211,
  208,206,203,201,198,196,193,190,188,185,182,179,176,173,170,
  167,165,162,158,155,152,149,146,143,140,137,134,131,128,124,
  121,118,115,112,109,106,103,100,97,93,90,88,85,82,79,76,73,70,
  67,65,62,59,57,54,52,49,47,44,42,40,37,35,33,31,29,27,25,23,21,
  20,18,17,15,14,12,11,10,9,7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,
  1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,17,18,20,21,23,25,27,29,
  31,33,35,37,40,42,44,47,49,52,54,57,59,62,65,67,70,73,76,79,82,
  85,88,90,93,97,100,103,106,109,112,115,118,121,124,128,131,134,
  137,140,143,146,149,152,155,158,162,165,167,170,173,176,179,182,
  185,188,190,193,196,198,201,203,206,208,211,213,215,218,220,222,
  224,226,228,230,232,234,235,237,238,240,241,243,244,245,246,248,
  249,250,250,251,252,253,253,254,254,254,255,255,255,255,255,255,
  255,254,254,254,253,253,252,251,250,250,249,248,246,245,244,243,
  241,240};

byte sineB256[] PROGMEM = {
  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,
  1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,17,18,20,21,23,25,27,29,31,
  33,35,37,40,42,44,47,49,52,54,57,59,62,65,67,70,73,76,79,82,85,
  88,90,93,97,100,103,106,109,112,115,118,121,124,128,131,134,137,
  140,143,146,149,152,155,158,162,165,167,170,173,176,179,182,185,
  188,190,193,196,198,201,203,206,208,211,213,215,218,220,222,224,
  226,228,230,232,234,235,237,238,240,241,243,244,245,246,248,249,
  250,250,251,252,253,253,254,254,254,255,255,255,255,255,255,255,
  254,254,254,253,253,252,251,250,250,249,248,246,245,244,243,241,
  240,238,237,235,234,232,230,228,226,224,222,220,218,215,213,211,
  208,206,203,201,198,196,193,190,188,185,182,179,176,173,170,167,
  165,162,158,155,152,149,146,143,140,137,134,131,128,124,121,118,
  115,112,109,106,103,100,97,93,90,88,85,82,79,76,73,70,67,65,62,
  59,57,54,52,49,47,44,42,40,37,35,33,31,29,27,25,23,21,20};

void setup () {
  for(int i=22;i<50;i++)//Define Port A,C and L as Output
  {
    pinMode(i,OUTPUT);
  }
}

void loop () {
  //the values from the look up tables are written to the ports
  //one by one
for (int i=0;i<256;i++) {
        PORTA = pgm_read_byte(sineR256 + i);//Red Phase
        PORTC = pgm_read_byte(sineY256 + i);//Blue Phase
        PORTL = pgm_read_byte(sineB256 + i);//Yellow Phase
        delayMicroseconds(76); //necessary delay to match the
                               //desired frequency
    }
}
```

## A.3 Arduino code for temperature controlled regulated power supply

```
#include <OneWire.h>
#include <LiquidCrystal.h>
#include <FreqMeasure.h>

OneWire  ds(3);  // on pin 3 (a 4.7K resistor is necessary)
LiquidCrystal lcd(A5, A4, A3, A2, A1, A0);
int pwm_fan = 6;
double sum = 0;
int count = 0, rpm = 0;

void setup(void) {
  lcd.begin(16, 2);
  Serial.begin(9600);
  FreqMeasure.begin();
  pinMode(pwm_fan, OUTPUT);
}

void loop(void) {
  byte i;
  byte present = 0;
  byte type_s;
  byte data[12];
  byte addr[8];
  float celsius, fahrenheit;

  if ( !ds.search(addr)) {
    Serial.println("No more addresses.");
    Serial.println();
    ds.reset_search();
    delay(250);
    return;
  }

  Serial.print("ROM =");
  for ( i = 0; i < 8; i++) {
    Serial.write(' ');
    Serial.print(addr[i], HEX);
  }

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

  // the first ROM byte indicates which chip
```

```
switch (addr[0]) {
  case 0x10:
    Serial.println("  Chip = DS18S20");  // or old DS1820
    type_s = 1;break;
  case 0x28:
    Serial.println("  Chip = DS18B20");
    type_s = 0;break;
  case 0x22:
    Serial.println("  Chip = DS1822");
    type_s = 0;break;
  default:
    Serial.println("Device is not a DS18x20 family device.");
    return;
}

ds.reset();
ds.select(addr);
ds.write(0x44, 1);        // start conversion, with parasite power
  on at the end
delay(1000);     // maybe 750ms is enough, maybe not
// we might do a ds.depower() here, but the reset will take care of
    it.
present = ds.reset();
ds.select(addr);
ds.write(0xBE);         // Read Scratchpad
Serial.print("  Data = ");
Serial.print(present, HEX);
Serial.print(" ");
for ( i = 0; i < 9; i++) {           // we need 9 bytes
  data[i] = ds.read();
  Serial.print(data[i], HEX);
  Serial.print(" ");
}
Serial.print(" CRC=");
Serial.print(OneWire::crc8(data, 8), HEX);
Serial.println();

/* Convert the data to actual temperature because the result is a
  16 bit signed integer, it should
  be stored to an "int16_t" type, which is always 16 bits even
     when compiled on a 32 bit processor.*/
int16_t raw = (data[1] << 8) | data[0];
if (type_s) {
  raw = raw << 3; // 9 bit resolution default
  if (data[7] == 0x10) {
    // "count remain" gives full 12 bit resolution
    raw = (raw & 0xFFF0) + 12 - data[6];
  }
} else {
  byte cfg = (data[4] & 0x60);
```

```
    // at lower res, the low bits are undefined, so let's zero them
    if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms
    else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
    else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
    //default is 12 bit resolution, 750 ms conversion time
  }
  celsius = (float)raw / 16.0;
  fahrenheit = celsius * 1.8 + 32.0;
  Serial.print("  Temperature = ");
  Serial.print(celsius);
  Serial.print(" Celsius, ");
  Serial.print(fahrenheit);
  Serial.println(" Fahrenheit");
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(celsius);
  lcd.print(" *C");
  if (FreqMeasure.available()) {
    // average several reading together
    sum = sum + FreqMeasure.read();
    count = count + 1;
    if (count > 15) {
      float frequency = FreqMeasure.countToFrequency(sum / count);
      rpm = frequency * (60 / 2);
      sum = 0;
      count = 0;
      Serial.print("Fan Speed");
      Serial.print(rpm);
      Serial.println("RPM");
      //rpm = 0;
    }
  }
  int fan_speed = map(celsius, 20, 45, 0, 255);
  if (fan_speed < 0)
  {
    fan_speed = 0;
  }
  else if (fan_speed > 255)
  {
    fan_speed = 255;
  }
  else
  {}
  analogWrite(pwm_fan, fan_speed);
  int desired_rpm = map(fan_speed, 0, 255, 800, 2200);
  lcd.setCursor(0, 1);
  lcd.print("Speed: ");
  lcd.print(rpm);
  lcd.print(" RPM");
}
```

## A.4 Arduino code for Phasor estimation using 64-Point DFT

```
#include <Time.h>          // Time Library
#include <TinyGPS++.h>     // GPS Library
#include <math.h>          // Math functions library

static const uint32_t GPSBaud = 38400;
boolean Calculate_A_Phasor = false;
boolean get_time_on_pps = false;
// The TinyGPS++ object
TinyGPSPlus gps;

// Serial connection to the GPS device
#define Serial_GPS Serial3
#define SerialTx Serial2
//#define SerialTx Serial

time_t prevDisplay = 0; // Count for when time last displayed
int Year;
byte Month;
byte Day;
byte Hour;
byte Minute;
byte Second;

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

long double adc_out_1[WindowSize], values_1[WindowSize];
long double adc_out_2[WindowSize], values_2[WindowSize];
long double adc_out_3[WindowSize], values_3[WindowSize];

long double Xi_1, Xr_1, Phasor_Magnitude_1, Phasor_Angle_1,
    Phasor_Angle_Degree_1;
long double Xi_2, Xr_2, Phasor_Magnitude_2, Phasor_Angle_2,
    Phasor_Angle_Degree_2;
long double Xi_3, Xr_3, Phasor_Magnitude_3, Phasor_Angle_3,
    Phasor_Angle_Degree_3;

unsigned long int calculation_start_millis;
unsigned long int calculation_finish_millis;

unsigned long int pps_time_millis;
unsigned long int phasor_stamp_millis;

//variables for frequency calculation
volatile long double P1_start_micros = 0, last_P1_start_micros = 0,
    P1_period = 0;
```

```
volatile long double P2_start_micros = 0, last_P2_start_micros = 0, P2_period = 0;
volatile long double P3_start_micros = 0, last_P3_start_micros = 0, P3_period = 0;

int P1_freq, P2_freq, P3_freq;
long double P1_freqf, P2_freqf, P3_freqf;
int P1_lf, P2_lf, P3_lf;// last frequencies
int P1_rocof, P2_rocof, P3_rocof;// rate of change of frequency df/dt
long double P1_rocoff, P2_rocoff, P3_rocoff;

void setup()
{ adc_setup();
  SerialTx.begin(921600);//for transmitting Phasors
  Serial_GPS.begin(GPSBaud); // Start GPS Serial Connection
  smartDelay(1000);
  delay(2000);
  analogReadResolution(12);
  attachInterrupt(22, aquire, RISING);// aquire voltage samples
  attachInterrupt(23, attach_pps_time, RISING);// get pulse per second time
  attachInterrupt(31, capture_P1_start, RISING);// get starting time of P1 waveform
  attachInterrupt(33, capture_P2_start, RISING);//"""""""""""""""""""" P2 waveform
  attachInterrupt(29, capture_P3_start, RISING);//"""""""""""""""""""" P3 waveform
}
// Get start Time of waves for calculation of frequency
void capture_P1_start() {
  P1_start_micros = micros();
  P1_period = P1_start_micros - last_P1_start_micros;
  last_P1_start_micros = P1_start_micros;
}
void capture_P2_start() {
  P2_start_micros = micros();
  P2_period = P2_start_micros - last_P2_start_micros;
  last_P2_start_micros = P2_start_micros;
}
void capture_P3_start() {
  P3_start_micros = micros();
  P3_period = P3_start_micros - last_P3_start_micros;
  last_P3_start_micros = P3_start_micros;
}
// Circular buffer, power of two.
#define BUFSIZE 0x40 //64 samples buffer
#define BUFMASK 0x3F
volatile int R [BUFSIZE] ;
volatile int Y [BUFSIZE] ;
volatile int B [BUFSIZE] ;
volatile int sptr = 0 ;
volatile int isr_count = 0 ;

void aquire() {
  ADC->ADC_CR |= 0b10; //start conversion
  while (!(ADC->ADC_ISR & 0b11100000)); //wait for conversion to end
  int Rval = ADC->ADC_CDR[7];
  int Yval = ADC->ADC_CDR[6];
  int Bval = ADC->ADC_CDR[5];

  R[sptr] = Rval;
  Y[sptr] = Yval;
  B[sptr] = Bval;
  sptr = (sptr + 1) & BUFMASK;
  isr_count ++ ;
}

void adc_setup()
{
  //ADC setup
  ADC->ADC_WPMR &= 0xFFFE; //disable write protect
  ADC->ADC_CHER = 0b11100000; //Enable AD7,AD6,AD5 or CH7,Ch6,Ch5 or PA16,PA24,PA23 or A0,A1 and A2      |
  ADC->ADC_MR &= 0b11111111000000000000011100000000;//Fast i.e. about 4mS for 2500 Conversions on three channels
  ADC->ADC_MR |= 0b00000000000100100000000000000000; //software trigger, hi res, no sleep, not free running
  ADC->ADC_IER = 0b11100000; //enable interrupt
  ADC->ADC_IMR = 0b11100000; //enable interrupt in mask
  ADC->ADC_CR |= 0b10; //start first conversion
}

void loop()
{
  if (get_time_on_pps == true)
  {
    pps_time_millis = millis();
    GPS_Timezone_Adjust();  // Call Time Adjust Function
    get_time_on_pps = false;
  }
  if (isr_count == 64)
  {
    Calculate_A_Phasor = true;
    isr_count = 0;
  }
  if (Calculate_A_Phasor == true)
  {
    calc_phasor();
    transmit_phasors_on_SerialTx();
    Calculate_A_Phasor = false;
  }
  smartDelay(0);
}

void attach_pps_time()
{
  get_time_on_pps = true;
}

void GPS_Timezone_Adjust() {

  Year = gps.date.year();
  Month = gps.date.month();
  Day = gps.date.day();
  Hour = gps.time.hour();
  Minute = gps.time.minute();
  Second = gps.time.second();

  // Set Time from GPS data string
  setTime(Hour, Minute, Second, Day, Month, Year);
  // Calc current Time Zone time by offset value

  if (timeStatus() != timeNotSet) {
    if (now() != prevDisplay) {
      prevDisplay = now();
    }
  }
  smartDelay(0);
}

static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (Serial_GPS.available())
      gps.encode(Serial_GPS.read());
  } while (millis() - start < ms);
}

//Phasor calculation function
void calc_phasor() {
  //copy buffer to SampleWindow for calculation
  for (int i = 0; i < 64; i++)
  {
    adc_out_1[i] = R[i];
    adc_out_2[i] = Y[i];
    adc_out_3[i] = B[i];
  }
  calculation_start_millis = millis();
```

```
for (int i = 0; i < N; i++)
{
  values_1[i] = map_double(adc_out_1[i], 1433, 2812, -347.25,
     347.25);// Phase A
  values_2[i] = map_double(adc_out_2[i], 1498, 2859, -336.78,
     336.78);// Phase B
  values_3[i] = map_double(adc_out_3[i], 1408, 2851, -344.70,
     344.70);// Phase C
  smartDelay(0);
}

//Calculate 64-Point DFT
Xr_1 = 0;    Xr_2 = 0;    Xr_3 = 0;
Xi_1 = 0;    Xi_2 = 0;    Xi_3 = 0;
Phasor_Magnitude_1 = 0;    Phasor_Magnitude_2 = 0;
   Phasor_Magnitude_3 = 0;
Phasor_Angle_1 = 0;    Phasor_Angle_2 = 0;    Phasor_Angle_3 = 0;
for (int n = 0; n < N; n++)
{
  Xr_1 = Xr_1 + values_1[n] * cos((2 * pi * n) / N);
  Xi_1 = Xi_1 + values_1[n] * sin((2 * pi * n) / N);

  Xr_2 = Xr_2 + values_2[n] * cos((2 * pi * n) / N);
  Xi_2 = Xi_2 + values_2[n] * sin((2 * pi * n) / N);

  Xr_3 = Xr_3 + values_3[n] * cos((2 * pi * n) / N);
  Xi_3 = Xi_3 + values_3[n] * sin((2 * pi * n) / N);
  smartDelay(0);
}

Xr_1 = (sqrt(2) / N) * Xr_1;
Xr_2 = (sqrt(2) / N) * Xr_2;
Xr_3 = (sqrt(2) / N) * Xr_3;

Xi_1 = -(sqrt(2) / N) * Xi_1;
Xi_2 = -(sqrt(2) / N) * Xi_2;
Xi_3 = -(sqrt(2) / N) * Xi_3;

Phasor_Magnitude_1 = sqrt(Xr_1 * Xr_1 + Xi_1 * Xi_1);
Phasor_Magnitude_2 = sqrt(Xr_2 * Xr_2 + Xi_2 * Xi_2);
Phasor_Magnitude_3 = sqrt(Xr_3 * Xr_3 + Xi_3 * Xi_3);

Phasor_Angle_1 = atan2(Xi_1, Xr_1); //double atan2(double y, double
    x)
Phasor_Angle_2 = atan2(Xi_2, Xr_2); //The atan2() function returns
   the arc tangent of y/x, in the range [-pi, +pi] radians.
Phasor_Angle_3 = atan2(Xi_3, Xr_3);

//Calculate Phasor Angle in Degree
Phasor_Angle_Degree_1 = (Phasor_Angle_1 * 4068) / 71;
```

```
  Phasor_Angle_Degree_2 = (Phasor_Angle_2 * 4068) / 71;
  Phasor_Angle_Degree_3 = (Phasor_Angle_3 * 4068) / 71;
  //
  calculation_finish_millis = millis();
  phasor_stamp_millis = calculation_start_millis - pps_time_millis;

  // Calculate frequency
  //long int P1_period = P1_end_micros - P1_start_micros;
  P1_freqf = 1000000 / P1_period;
  P2_freqf = 1000000 / P2_period;
  P3_freqf = 1000000 / P3_period;

  P1_rocoff = sqrt((P1_freqf - 50.00) * (P1_freqf - 50.00)) * 50;
  P2_rocoff = sqrt((P2_freqf - 50.00) * (P2_freqf - 50.00)) * 50;
  P3_rocoff = sqrt((P3_freqf - 50.00) * (P3_freqf - 50.00)) * 50;

  smartDelay(0);
}

float map_double(double x, double in_min, double in_max, double
   out_min, double out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
     out_min;
  smartDelay(0);
}
void transmit_phasors_on_SerialTx() {
  SerialTx.write('!');
  SerialTx.print(int(Phasor_Magnitude_1 * 100));
  SerialTx.write('"');
  SerialTx.print(int(Phasor_Magnitude_2 * 100));
  SerialTx.write('#');
  SerialTx.print(int(Phasor_Magnitude_3 * 100));
  SerialTx.write('$');

  //Calculate angle i.e. 2pi's complement to be sent
  if (Phasor_Angle_1 < 0)
    Phasor_Angle_1 = Phasor_Angle_1 + 6.286;
  if (Phasor_Angle_2 < 0)
    Phasor_Angle_2 = Phasor_Angle_2 + 6.286;
  if (Phasor_Angle_3 < 0)
    Phasor_Angle_3 = Phasor_Angle_3 + 6.286;

  SerialTx.print(int(Phasor_Angle_1 * 1000));
  SerialTx.write('%');
  SerialTx.print(int(Phasor_Angle_2 * 1000));
  SerialTx.write('&');
  SerialTx.print(int(Phasor_Angle_3 * 1000));
  SerialTx.write('(');

  SerialTx.print(day());
  SerialTx.write(')');
  SerialTx.print(month());
  SerialTx.write('*');
  SerialTx.print(year());

  SerialTx.write('+');
  SerialTx.print(hour());
  SerialTx.write(',');
  SerialTx.print(minute());
  SerialTx.write('-');
  SerialTx.print(second());
  SerialTx.write('.');
  SerialTx.print(phasor_stamp_millis);
  SerialTx.write('/');

  float Latitude = (gps.location.lat());
  float Longitude = (gps.location.lng());
  SerialTx.print(int(Latitude * 1000));
  SerialTx.write(':');
  SerialTx.print(int(Longitude * 1000));
  SerialTx.write(';');

  //Transmit Frequencies
  SerialTx.print(int(P1_freqf * 100));
  SerialTx.write('@');
  SerialTx.print(int(P2_freqf * 100));
  SerialTx.write('^');
  SerialTx.print(int(P3_freqf * 100));
  SerialTx.write('?');

  //Transmit ROCOF
  SerialTx.print(int(P1_rocoff * 100));
  SerialTx.write('[');
  SerialTx.print(int(P2_rocoff * 100));
  SerialTx.write(']');
  SerialTx.print(int(P3_rocoff * 100));
  SerialTx.write('|');

}
```

## A.5 Arduino code for PMU communication unit

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

int Year, Month, Day;
int Hour, Minute, Second, MilliSecond;

float Phasor_Magnitude_1, Phasor_Magnitude_2, Phasor_Magnitude_3;

float Phasor_Angle_1, Phasor_Angle_Degree_1;
float Phasor_Angle_2, Phasor_Angle_Degree_2;
float Phasor_Angle_3, Phasor_Angle_Degree_3;

float Latitude;
float Longitude;

float P1_freq, P2_freq, P3_freq;
float P1_rocof, P2_rocof, P3_rocof;

byte status = IDLE;
#define SerialRx Serial2
#define SerialLCD Serial3
void setup() {
  SerialLCD.begin(921600);
  SerialRx.begin(921600);
  Serial.begin(460800);
  //Serial.println("Ready");
}
int count = 0;
```

```
void loop() {

  if (SerialRx.available()) {
    int c = SerialRx.read();
    if (status == RECEIVING1 && c >= '0' && c <= '9') {
      Phasor_Magnitude_1 = Phasor_Magnitude_1 * 10 + (c - '0');
    } else if (status == RECEIVING2 && c >= '0' && c <= '9') {
      Phasor_Magnitude_2 = Phasor_Magnitude_2 * 10 + (c - '0');
    } else if (status == RECEIVING3 && c >= '0' && c <= '9') {
      Phasor_Magnitude_3 = Phasor_Magnitude_3 * 10 + (c - '0');
    } else if (status == RECEIVING4 && c >= '0' && c <= '9') {
      Phasor_Angle_1 = Phasor_Angle_1 * 10 + (c - '0');
    } else if (status == RECEIVING5 && c >= '0' && c <= '9') {
      Phasor_Angle_2 = Phasor_Angle_2 * 10 + (c - '0');
    } else if (status == RECEIVING6 && c >= '0' && c <= '9') {
      Phasor_Angle_3 = Phasor_Angle_3 * 10 + (c - '0');
    } else if (status == RECEIVING7 && c >= '0' && c <= '9') {
      Day = Day * 10 + (c - '0');
    } else if (status == RECEIVING8 && c >= '0' && c <= '9') {
      Month = Month * 10 + (c - '0');
    } else if (status == RECEIVING9 && c >= '0' && c <= '9') {
      Year = Year * 10 + (c - '0');
    } else if (status == RECEIVING10 && c >= '0' && c <= '9') {
      Hour = Hour * 10 + (c - '0');
    } else if (status == RECEIVING11 && c >= '0' && c <= '9') {
      Minute = Minute  * 10 + (c - '0');
    } else if (status == RECEIVING12 && c >= '0' && c <= '9') {
      Second = Second * 10 + (c - '0');
    }  else if (status == RECEIVING13 && c >= '0' && c <= '9') {
      MilliSecond = MilliSecond * 10 + (c - '0');
    }  else if (status == RECEIVING14 && c >= '0' && c <= '9') {
      Latitude = Latitude * 10 + (c - '0');
    }  else if (status == RECEIVING15 && c >= '0' && c <= '9') {
      Longitude = Longitude * 10 + (c - '0');
    }  else if (status == RECEIVING16 && c >= '0' && c <= '9') {
      P1_freq = P1_freq * 10 + (c - '0');
    }  else if (status == RECEIVING17 && c >= '0' && c <= '9') {
      P2_freq = P2_freq * 10 + (c - '0');
    }  else if (status == RECEIVING18 && c >= '0' && c <= '9') {
      P3_freq = P3_freq * 10 + (c - '0');
    } else if (status == RECEIVING19 && c >= '0' && c <= '9') {
      P1_rocof = P1_rocof * 10 + (c - '0');
    } else if (status == RECEIVING20 && c >= '0' && c <= '9') {
      P2_rocof = P2_rocof * 10 + (c - '0');
    } else if (status == RECEIVING21 && c >= '0' && c <= '9') {
      P3_rocof = P3_rocof * 10 + (c - '0');
    }

    else if (status == RECEIVING1 && c == '"') {
      status = RECEIVING2;
    } else if (status == RECEIVING2 && c == '#') {
      status = RECEIVING3;
    } else if (status == RECEIVING3 && c == '$') {
      status = RECEIVING4;
    } else if (status == RECEIVING4 && c == '%') {
      status = RECEIVING5;
    } else if (status == RECEIVING5 && c == '&') {
      status = RECEIVING6;
    } else if (status == RECEIVING6 && c == '(') {
      status = RECEIVING7;
    } else if (status == RECEIVING7 && c == ')') {
      status = RECEIVING8;
    } else if (status == RECEIVING8 && c == '*') {
      status = RECEIVING9;
    } else if (status == RECEIVING9 && c == '+') {
      status = RECEIVING10;
    } else if (status == RECEIVING10 && c == ',') {
      status = RECEIVING11;
    } else if (status == RECEIVING11 && c == '-') {
      status = RECEIVING12;
    } else if (status == RECEIVING12 && c == '.') {
      status = RECEIVING13;
    } else if (status == RECEIVING13 && c == '/') {
      status = RECEIVING14;
    } else if (status == RECEIVING14 && c == ':') {
      status = RECEIVING15;
    } else if (status == RECEIVING15 && c == ';') {
      status = RECEIVING16;
    } else if (status == RECEIVING16 && c == '@') {
      status = RECEIVING17;
    } else if (status == RECEIVING17 && c == '^') {
      status = RECEIVING18;
    } else if (status == RECEIVING18 && c == '?') {
      status = RECEIVING19;
    } else if (status == RECEIVING19 && c == '[') {
      status = RECEIVING20;
    } else if (status == RECEIVING20 && c == ']') {
      status = RECEIVING21;
    } else if (c == '|') {
      status = IDLE;

      //Remote value Received completely, Now display it

      //Calculate Phasor Angles into Float
      Phasor_Angle_1 = Phasor_Angle_1 / 1000;
      Phasor_Angle_2 = Phasor_Angle_2 / 1000;
      Phasor_Angle_3 = Phasor_Angle_3 / 1000;

      //Calculate Phasor Angle using reverse 2pi's complement
      if (Phasor_Angle_1 > 3.143)
```

```
          Phasor_Angle_1 = Phasor_Angle_1 - 6.286;
        if (Phasor_Angle_2 > 3.143)
          Phasor_Angle_2 = Phasor_Angle_2 - 6.286;
        if (Phasor_Angle_3 > 3.143)
          Phasor_Angle_3 = Phasor_Angle_3 - 6.286;

        //Calculate Angles in Degrees
        Phasor_Angle_Degree_1 = (Phasor_Angle_1 * 4068) / 71;
        Phasor_Angle_Degree_2 = (Phasor_Angle_2 * 4068) / 71;
        Phasor_Angle_Degree_3 = (Phasor_Angle_3 * 4068) / 71;

        Display_Phasors_on_Serial_Terminal();
        transmit_phasors_LCD();

      } else if (c == '!') {
        status = RECEIVING1;
        //Reset the variables to Zero
        Year = 0;
        Month = 0;
        Day = 0;
        Hour = 0;
        Minute = 0;
        Second = 0;
        MilliSecond = 0;
        Phasor_Angle_1 = 0;
        Phasor_Angle_2 = 0;
        Phasor_Angle_3 = 0;
        Phasor_Magnitude_1 = 0;
        Phasor_Magnitude_2 = 0;
        Phasor_Magnitude_3 = 0;
        Phasor_Angle_Degree_1 = 0;
        Phasor_Angle_Degree_2 = 0;
        Phasor_Angle_Degree_3 = 0;
        Latitude = 0;
        Longitude = 0;
        P1_freq = 0;
        P2_freq = 0;
        P3_freq = 0;
        P1_rocof = 0;
        P2_rocof = 0;
        P3_rocof = 0;
      }
    }
}
// Transmit the phasors to local PDC, where it can be plotted in real
   -time
void Display_Phasors_on_Serial_Terminal() {
  //Serial.print(millis());
  Serial.print(Day);
  Serial.print(" ");
  Serial.print(Month);
  Serial.print(" ");
  Serial.print(Year);
  Serial.print(" ");
  Serial.print(Hour);
  Serial.print(" ");
  Serial.print(Minute);
  Serial.print(" ");
  Serial.print(Second);
  Serial.print(" ");
  Serial.print(MilliSecond);
  Serial.print(" ");
  Serial.print(float(Phasor_Magnitude_1 / 100));
  Serial.print(" ");
  Serial.print(Phasor_Angle_Degree_1);
  Serial.print(" ");
  Serial.print(float(Phasor_Magnitude_2 / 100));
  Serial.print(" ");
  Serial.print(Phasor_Angle_Degree_2);
  Serial.print(" ");
  Serial.print(float(Phasor_Magnitude_3 / 100));
  Serial.print(" ");
  Serial.print(Phasor_Angle_Degree_3);
  Serial.print(" ");
  Serial.print(P1_freq / 100);
  Serial.print(" ");
  Serial.print(P2_freq / 100);
  Serial.print(" ");
  Serial.print(P3_freq / 100);
  Serial.print(" ");
  Serial.print(P1_rocof / 100);
  Serial.print(" ");
  Serial.print(P2_rocof / 100);
  Serial.print(" ");
  Serial.print(P3_rocof / 100);
  Serial.print("\n");
}

// Transmit the Phasor parameters to LCD Module
void transmit_phasors_LCD() {
  SerialLCD.write('!');
  SerialLCD.print(int(Phasor_Magnitude_1));
  SerialLCD.write('"');
  SerialLCD.print(int(Phasor_Magnitude_2));
  SerialLCD.write('#');
  SerialLCD.print(int(Phasor_Magnitude_3));
  SerialLCD.write('$');

  //Calculate angle i.e. 2pi's complement to be sent
  if (Phasor_Angle_1 < 0)
```

```
    Phasor_Angle_1 = Phasor_Angle_1 + 6.286;
  if (Phasor_Angle_2 < 0)
    Phasor_Angle_2 = Phasor_Angle_2 + 6.286;
  if (Phasor_Angle_3 < 0)
    Phasor_Angle_3 = Phasor_Angle_3 + 6.286;
  SerialLCD.print(int(Phasor_Angle_1 * 100));
  SerialLCD.write('%');
  SerialLCD.print(int(Phasor_Angle_2 * 100));
  SerialLCD.write('&');
  SerialLCD.print(int(Phasor_Angle_3 * 100));
  SerialLCD.write('(');

  SerialLCD.print(Day);
  SerialLCD.write(')');
  SerialLCD.print(Month);
  SerialLCD.write('*');
  SerialLCD.print(Year);

  SerialLCD.write('+');
  SerialLCD.print(Hour);
  SerialLCD.write(',');
  SerialLCD.print(Minute);
  SerialLCD.write('-');
  SerialLCD.print(Second);
  SerialLCD.write('.');
  SerialLCD.print(MilliSecond);
  SerialLCD.write('/');

  SerialLCD.print(int(Latitude));
  SerialLCD.write(':');
  SerialLCD.print(int(Longitude));
  SerialLCD.write(';');

  //Transmit Frequencies
  SerialLCD.print(P1_freq);
  SerialLCD.write('@');
  SerialLCD.print(P2_freq);
  SerialLCD.write('^');
  SerialLCD.print(P3_freq);
  SerialLCD.write('?');

  //Transmit ROCOF
  SerialLCD.print(P1_rocof);
  SerialLCD.write('[');
  SerialLCD.print(P2_rocof);
  SerialLCD.write(']');
  SerialLCD.print(P3_rocof);
  SerialLCD.write('|');
}
```

## A.6 Arduino code for local PMU data display unit

```
#include <UTFT.h>
extern uint8_t Grotesk32x64[];// Declare which fonts we will be using
extern uint8_t Ubuntubold[];
extern uint8_t Ubuntu[];
extern uint8_t franklingothic_normal[];
extern uint8_t Inconsola[];
extern uint8_t BigFont[];// Declare which fonts we will be using

UTFT PMU_LCD(CTE70, 25, 26, 27, 28);
bool display_now_on_lcd = false;

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

int Year, Month, Day;
int Hour, Minute, Second, MilliSecond;

float Phasor_Magnitude_1;
float Phasor_Magnitude_2;
float Phasor_Magnitude_3;

float Phasor_Angle_1, Phasor_Angle_Degree_1;
float Phasor_Angle_2, Phasor_Angle_Degree_2;
float Phasor_Angle_3, Phasor_Angle_Degree_3;

float Latitude;
float Longitude;

float P1_freq, P2_freq, P3_freq;
float P1_rocof, P2_rocof, P3_rocof;

byte status = IDLE;

void setup() {
  // Setup the LCD
  PMU_LCD.InitLCD();

  init_LCD();
  Serial3.begin(921600);
  attachInterrupt(8, display_on_lcd, RISING);

}
int count = 0;
void loop() {
  // put your main code here, to run repeatedly:
  if (Serial3.available()) {
    int c = Serial3.read();
    if (status == RECEIVING1 && c >= '0' && c <= '9') {
      Phasor_Magnitude_1 = Phasor_Magnitude_1 * 10 + (c - '0');
    } else if (status == RECEIVING2 && c >= '0' && c <= '9') {
      Phasor_Magnitude_2 = Phasor_Magnitude_2 * 10 + (c - '0');
    } else if (status == RECEIVING3 && c >= '0' && c <= '9') {
      Phasor_Magnitude_3 = Phasor_Magnitude_3 * 10 + (c - '0');
    } else if (status == RECEIVING4 && c >= '0' && c <= '9') {
      Phasor_Angle_1 = Phasor_Angle_1 * 10 + (c - '0');
    } else if (status == RECEIVING5 && c >= '0' && c <= '9') {
      Phasor_Angle_2 = Phasor_Angle_2 * 10 + (c - '0');
    } else if (status == RECEIVING6 && c >= '0' && c <= '9') {
      Phasor_Angle_3 = Phasor_Angle_3 * 10 + (c - '0');
    } else if (status == RECEIVING7 && c >= '0' && c <= '9') {
      Day = Day * 10 + (c - '0');
    } else if (status == RECEIVING8 && c >= '0' && c <= '9') {
      Month = Month * 10 + (c - '0');
    } else if (status == RECEIVING9 && c >= '0' && c <= '9') {
      Year = Year * 10 + (c - '0');
    } else if (status == RECEIVING10 && c >= '0' && c <= '9') {
      Hour = Hour * 10 + (c - '0');
    } else if (status == RECEIVING11 && c >= '0' && c <= '9') {
      Minute = Minute  * 10 + (c - '0');
    } else if (status == RECEIVING12 && c >= '0' && c <= '9') {
      Second = Second * 10 + (c - '0');
    }  else if (status == RECEIVING13 && c >= '0' && c <= '9') {
      MilliSecond = MilliSecond * 10 + (c - '0');
    }  else if (status == RECEIVING14 && c >= '0' && c <= '9') {
      Latitude = Latitude * 10 + (c - '0');
    }  else if (status == RECEIVING15 && c >= '0' && c <= '9') {
      Longitude = Longitude * 10 + (c - '0');
    } else if (status == RECEIVING16 && c >= '0' && c <= '9') {
      P1_freq = P1_freq * 10 + (c - '0');
    }  else if (status == RECEIVING17 && c >= '0' && c <= '9') {
      P2_freq = P2_freq * 10 + (c - '0');
    }  else if (status == RECEIVING18 && c >= '0' && c <= '9') {
      P3_freq = P3_freq * 10 + (c - '0');
    } else if (status == RECEIVING19 && c >= '0' && c <= '9') {
      P1_rocof = P1_rocof * 10 + (c - '0');
    } else if (status == RECEIVING20 && c >= '0' && c <= '9') {
      P2_rocof = P2_rocof * 10 + (c - '0');
    } else if (status == RECEIVING21 && c >= '0' && c <= '9') {
      P3_rocof = P3_rocof * 10 + (c - '0');
    }
    else if (status == RECEIVING1 && c == '"') {
      status = RECEIVING2;
    } else if (status == RECEIVING2 && c == '#') {
      status = RECEIVING3;
    } else if (status == RECEIVING3 && c == '$') {
      status = RECEIVING4;
    } else if (status == RECEIVING4 && c == '%') {
      status = RECEIVING5;
    } else if (status == RECEIVING5 && c == '&') {
      status = RECEIVING6;
    } else if (status == RECEIVING6 && c == '(') {
      status = RECEIVING7;
    } else if (status == RECEIVING7 && c == ')') {
      status = RECEIVING8;
    } else if (status == RECEIVING8 && c == '*') {
      status = RECEIVING9;
    } else if (status == RECEIVING9 && c == '+') {
      status = RECEIVING10;
    } else if (status == RECEIVING10 && c == ',') {
      status = RECEIVING11;
    } else if (status == RECEIVING11 && c == '-') {
      status = RECEIVING12;
    } else if (status == RECEIVING12 && c == '.') {
      status = RECEIVING13;
    } else if (status == RECEIVING13 && c == '/') {
      status = RECEIVING14;
    } else if (status == RECEIVING14 && c == ':') {
      status = RECEIVING15;
    } else if (status == RECEIVING15 && c == ';') {
      status = RECEIVING16;
    } else if (status == RECEIVING16 && c == '@') {
      status = RECEIVING17;
    } else if (status == RECEIVING17 && c == '^') {
      status = RECEIVING18;
    } else if (status == RECEIVING18 && c == '?') {
      status = RECEIVING19;
    } else if (status == RECEIVING19 && c == '[') {
      status = RECEIVING20;
    } else if (status == RECEIVING20 && c == ']') {
      status = RECEIVING21;
    } else if (c == '|') {
      status = IDLE;

      //Remote value Received completely, Now display it

      //Calculate Phasor Angles into Float
      Phasor_Angle_1 = Phasor_Angle_1 / 100;
      Phasor_Angle_2 = Phasor_Angle_2 / 100;
      Phasor_Angle_3 = Phasor_Angle_3 / 100;

      //Calculate Phasor Angle using reverse 2pi's complement
      if (Phasor_Angle_1 > 3.143)
        Phasor_Angle_1 = Phasor_Angle_1 - 6.286;
      if (Phasor_Angle_2 > 3.143)
        Phasor_Angle_2 = Phasor_Angle_2 - 6.286;
      if (Phasor_Angle_3 > 3.143)
        Phasor_Angle_3 = Phasor_Angle_3 - 6.286;

      //Calculate Angles in Degrees
      Phasor_Angle_Degree_1 = (Phasor_Angle_1 * 4068) / 71;
      Phasor_Angle_Degree_2 = (Phasor_Angle_2 * 4068) / 71;
      Phasor_Angle_Degree_3 = (Phasor_Angle_3 * 4068) / 71;

      //Interrupt Driven LCD Display
      if (display_now_on_lcd == true) {
        Display_on_LCD();
        display_now_on_lcd = false;
      }

    } else if (c == '!') {
      status = RECEIVING1;

      //Reset the variables to Zero
      Year = 0;
      Month = 0;
      Day = 0;
      Hour = 0;
      Minute = 0;
      Second = 0;
      MilliSecond = 0;
      Phasor_Angle_1 = 0;
      Phasor_Angle_2 = 0;
      Phasor_Angle_3 = 0;
      Phasor_Magnitude_1 = 0;
      Phasor_Magnitude_2 = 0;
      Phasor_Magnitude_3 = 0;
      Phasor_Angle_Degree_1 = 0;
      Phasor_Angle_Degree_2 = 0;
      Phasor_Angle_Degree_3 = 0;
      Latitude = 0;
      Longitude = 0;
      P1_freq = 0;
      P2_freq = 0;
      P3_freq = 0;
      P1_rocof = 0;
      P2_rocof = 0;
      P3_rocof = 0;
    }
  }
}

void init_LCD() {
  //LCD Size 800:480
  //0,0   799,0
  //0,479 799,479
  PMU_LCD.setFont(BigFont);
  PMU_LCD.clrScr();
  PMU_LCD.setColor(0, 255, 0);
  PMU_LCD.print("* Phasor Measurement Unit Local Display *", CENTER,
     1);
  PMU_LCD.setColor(255, 153, 51);
  PMU_LCD.print("!!! Developed by Debashish Mohapatra !!!", CENTER,
     462);

  // Print out Phase 1 phase 2 and Phase 3
  String Header1 = String("Phasor ") + String(" Phase1 ") + String("
     Phase2") + String("   Phase3");
  PMU_LCD.setColor(255, 0, 255);
  PMU_LCD.setFont(Inconsola);
  PMU_LCD.print(Header1, LEFT, 30);
  PMU_LCD.setColor(255, 0, 255);

  PMU_LCD.print("Magni:", LEFT, 80);
  PMU_LCD.print("Angle:", LEFT, 145);
  PMU_LCD.print("Frequ:", LEFT, 220);
  PMU_LCD.print("ROCOF:", LEFT, 295);

  PMU_LCD.setColor(0, 255, 0);
  PMU_LCD.setFont(BigFont);
  PMU_LCD.print(" (VOLT)", LEFT, 115);
  PMU_LCD.print("(DEGREE)", LEFT, 180);
  PMU_LCD.print("  (HZ)", LEFT, 255);
  PMU_LCD.print("(HZ/SEC)", LEFT, 330);
}

void Display_on_LCD() {
  // Print Phase1 Parameters
  PMU_LCD.setFont(Inconsola);
  PMU_LCD.setColor(255, 0, 0);// Red
  PMU_LCD.printNumF(Phasor_Magnitude_1 / 100, 2, 185, 85,  46, 5,48);
  PMU_LCD.print("        ", 185, 150);
  PMU_LCD.printNumF(Phasor_Angle_Degree_1, 2,  185, 150, 46, 6, 48);
  PMU_LCD.printNumF(P1_freq / 10000, 2,          185, 225, 46, 4,48);
  PMU_LCD.printNumF(P1_rocof / 10000, 2,         185, 300, 46, 4,48);

  PMU_LCD.setColor(255, 255, 0);// Yellow
  PMU_LCD.printNumF(Phasor_Magnitude_2 / 100, 2, 380, 85,  46, 5,48);
  PMU_LCD.print("        ", 380, 150);
  PMU_LCD.printNumF(Phasor_Angle_Degree_2, 2,  380, 150, 46, 6, 48);
  PMU_LCD.printNumF(P2_freq / 10000, 2,          380, 225, 46, 4,48);
  PMU_LCD.printNumF(P2_rocof / 10000, 2,         380, 300, 46, 4,48);

  PMU_LCD.setColor(127, 250, 250);// White-Blue
  PMU_LCD.printNumF(Phasor_Magnitude_3 / 100, 2, 600, 85,  46, 5,48);
  PMU_LCD.print("        ", 600, 150);
  PMU_LCD.printNumF(Phasor_Angle_Degree_3, 2,  600, 150, 46, 6, 48);
  PMU_LCD.printNumF(P3_freq / 10000, 2,          600, 225, 46, 4,48);
  PMU_LCD.printNumF(P3_rocof / 10000, 2,         600, 300, 46, 4,48);

  // Print GPS Information
  String Time = String("         UTC Time: ") + String(Hour) + ":" +
     String(Minute) + ":" + String(Second) + "   ";
  String Date = String("          Date: ") + String(Day) + "/" +
     String(Month) + "/" + String(Year);
  String Location = String(" Latitude: ") + String(Latitude / 1000) +
      String(" Longitude: ") + String(Longitude / 1000);

  PMU_LCD.setFont(Ubuntu);
  PMU_LCD.setColor(255, 255, 255);
  PMU_LCD.print(Time, LEFT, 360);
  PMU_LCD.print(Date, LEFT, 395);
  PMU_LCD .print(Location, LEFT, 430);
}
void display_on_lcd() { //ISR
  display_now_on_lcd = true;
}
```

Appendix B

# Python Codes for Computer

## B.1 Python program for real-time plotting and logging of the Phasors

```
from pyqtgraph.Qt import QtGui, QtCore
import pyqtgraph as pg
##import time
import numpy as np

import serial
ser = serial.Serial('com8', 460800, timeout=1)
# Connect to serial port at COM8, at 460800 bauds


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

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

win.setWindowTitle('Realtime PMU Data Monitoring')
# Set the window title

#Define first graph to show the phasor magnitudes
p1 = win.addPlot(title="Phasor Magnitudes",colspan=2)
p1.setRange(yRange=[215, 250],xRange=[0, 1000])
p1.setLabel('left', "Phasor RMS Magnitude", units='Volts')
p1.setLabel('bottom', "Time ( x20 milli Seconds)")
p1.showGrid(x=1, y=1, alpha=.5)
```

```
p1.addLegend(offset=[-10,-10])

win.nextRow()
#Define second graph to show the phasor angles
p2 = win.addPlot(title="Phasor Angles")
p2.setRange(yRange=[-200, 200],xRange=[0, 1000])
p2.setLabel('left', "Phasor angles", units='Degree')
p2.setLabel('bottom', "Time ( x20 milli Seconds)")
p2.showGrid(x=1, y=1, alpha=.5)
p2.addLegend(offset=[-40,-10])

#Define third graph to show the phasor polar plot
v = win.addViewBox()
v.setAspectLocked()
v.setFixedWidth(500)
p3 = pg.PlotItem()
p3.setRange(xRange=[-250,250],yRange=[-250, 250])
curvePen = pg.mkPen(color=(255, 255, 255), style=QtCore.Qt.DotLine)
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",)
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")
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")
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")
c7 = p3.plot(x=np.linspace(-177, 177, 500), y=np.linspace(-177, 177,
    500),pen=curvePen)
c8 = p3.plot(x=np.linspace(-177, 177, 500), y=np.linspace(177, -177,
    500),pen=curvePen)
c9 = p3.plot(x=np.linspace(-250, 250, 500), y=np.linspace(0, 0, 500),
    pen=curvePen)
c10 = p3.plot(x=np.linspace(0, 0, 500), y=np.linspace(-250, 250, 500)
    ,pen=curvePen)
p3.addLegend(offset=[-1,-1])

g = pg.GraphItem()
v.addItem(g)
v.addItem(c1)
v.addItem(c2)
v.addItem(c4)
v.addItem(c6)
v.addItem(c7)
v.addItem(c8)
v.addItem(c9)
v.addItem(c10)

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

```
curve3 = p1.plot(pen=(0, 0, 255),name="Phase 3(RMS Magnitude)")

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

# Read the serial data string coming in from the PMU
line1 = ser.readline()
# split the string and extract the phasor parameters
data1 = [float(val1) for val1 in line1.split()]

previous_minute = int(data1[4])

# define the log files, where the phasors will be stored
path_txt = 'pmu_data.txt'
path_txt_plot = 'pmu_data_plot.txt'
path_excel = 'pmu_data.csv'

now_min = "%s-%s-%s_%s-%s" %(int(data1[0]), int(data1[1]), int(data1
    [2]), int(data1[3]), int(data1[4]))
path_txt_n = '%s_%s' % (now_min, path_txt)
path_txt_plot_n = '%s_%s' % (now_min, path_txt_plot)
path_excel_n = '%s_%s' % (now_min, path_excel)

logfileExcel = open(path_excel_n, 'a')
logfileText = open(path_txt_n, 'a')
logfileTextPlot = open(path_txt_plot_n, 'a')

# define the read function to read the data stream and append the
# parameters to sepatrate arrays
def readfun():
  global data, current_minute, previous_minute, FORMAT, logfileText,
      logfileTextPlot, logfileExcel, path_txt, path_excel,
      path_txt_plot
  line = ser.readline()
  data = [float(val) for val in line.split()]

  current_minute = int(data[4])
  if current_minute == (previous_minute+1):
    now_m = "%s-%s-%s_%s-%s" %(int(data[0]), int(data[1]), int(data
        [2]), int(data[3]), int(data[4]))
    new_path_txt = '%s_%s' % (now_m, path_txt)
    new_path_txt_plot = '%s_%s' % (now_m, path_txt_plot)
    new_path_excel = '%s_%s' % (now_m, path_excel)

    logfileExcel.flush()
    logfileText.flush()
    logfileTextPlot.flush()
    logfileExcel.close()
    logfileText.close()
```

```python
    logfileTextPlot.close()
    logfileExcel = open(new_path_excel, 'a')
    logfileText = open(new_path_txt, 'a')
    logfileTextPlot = open(new_path_txt_plot, 'a')
    previous_minute = current_minute

  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")
  logfileExcel.write(a)
  logfileText.write(a)

  bs = int(data[5])
  bms = int(data[6])
  bmS = (bs*1000)+bms
  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")
  logfileTextPlot.write(b)
  return data[6],data[7],data[8],data[9],data[10],data[11],data[12]

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]

y2=np.zeros(1000,dtype=float)
y3=np.zeros(1000,dtype=float)
y4=np.zeros(1000,dtype=float)
y5=np.zeros(1000,dtype=float)
y6=np.zeros(1000,dtype=float)
y7=np.zeros(1000,dtype=float)

indx = 0
# define the update function to update the plots with the parameter arrays
def update():
  global curve1, curve2, curve3, indx, y2, y3, y4, y5, y6, y7 #y1

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

  y2[indx]=readData[1]
  y3[indx]=readData[2]
  y4[indx]=readData[3]
  y5[indx]=readData[4]
  y6[indx]=readData[5]
  y7[indx]=readData[6]

  Rx=y2[indx]*np.cos(np.deg2rad(y3[indx]))
  Ry=y2[indx]*np.sin(np.deg2rad(y3[indx]))
  Yx=y4[indx]*np.cos(np.deg2rad(y5[indx]))
  Yy=y4[indx]*np.sin(np.deg2rad(y5[indx]))
  Bx=y6[indx]*np.cos(np.deg2rad(y7[indx]))
  By=y6[indx]*np.sin(np.deg2rad(y7[indx]))

  pos = np.array([[0,0],[Rx,Ry],[Yx,Yy],[Bx,By]])
  adj = np.array([[0,1],[0,2],[0,3]])
  symbols = ['o','t','t','t']
  lines = np.array([(255,0,0,255,3),(255,255,0,255,3),(0,0,255,255,3)], dtype=[('red',np.ubyte),
        ('green',np.ubyte),('blue',np.ubyte),('alpha',np.ubyte),('width',float)])
  g.setData(pos=pos, adj=adj,pen=lines,size=1,symbol=symbols )

  if indx==999:
     y2=np.zeros(1000,dtype=float)
     y3=np.zeros(1000,dtype=float)
     y4=np.zeros(1000,dtype=float)
     y5=np.zeros(1000,dtype=float)
     y6=np.zeros(1000,dtype=float)
     y7=np.zeros(1000,dtype=float)
     indx = 0
  else:
     indx+=1

  curve1.setData(y2)# update magnitudes
  curve2.setData(y4)
  curve3.setData(y6)
  curve4.setData(y3)# update angles
  curve5.setData(y5)
  curve6.setData(y7)
  app.processEvents()

timer = QtCore.QTimer()
timer.timeout.connect(update)
timer.start()

if __name__ == '__main__':
  import sys
  if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_'):
    QtGui.QApplication.instance().exec_()
```

## B.2 Python program for offline plotting of phasor data

```python
import numpy as np
import matplotlib.pyplot as plt

with open("pmu_data_49_95_Hz.txt") as f:
    data = f.read()

data = data.split('\n')
x8 = [row.split(',')[0] for row in data]
x7 = [row.split(',')[1] for row in data]
x1 = [row.split(',')[2] for row in data]
x2 = [row.split(',')[3] for row in data]
x3 = [row.split(',')[4] for row in data]
x4 = [row.split(',')[5] for row in data]
x5 = [row.split(',')[6] for row in data]
x6 = [row.split(',')[7] for row in data]

fig = plt.figure()

ax1 = fig.add_subplot(211)

ax1.set_title("Plot of Reported Phasors by PMU")
ax1.set_xlabel('Time in Milli Seconds')
ax1.set_ylabel('Amplitude in Volts (RMS)')

ax1.plot(x7,x1,'-',c='r',linewidth=2.0, label='Ph1 Magnitude')
ax1.plot(x7,x3,'--',c='r',linewidth=2.0, label='Ph2 Magnitude')
ax1.plot(x7,x5,'-.',c='r',linewidth=2.0, label='Ph3 Magnitude')


leg = ax1.legend()

ax2 = fig.add_subplot(212)
ax2.set_xlabel('Time in Milli Seconds')
ax2.set_ylabel('Phasor Angles in Degree')

ax2.plot(x7,x2,'-',c='r',linewidth=2.0, label='Ph1 Angle')
ax2.plot(x7,x4,'--',c='y',linewidth=2.0, label='Ph2 Angle')
ax2.plot(x7,x6,'-.',c='b',linewidth=2.0, label='Ph3 Angle')

leg = ax2.legend()

plt.show()
```