An Improved PMU Model for Voltage Response in Transient Stability Simulation

by

Adheesh Boratkar

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science (M.A.Sc.)
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

An Improved PMU Model for Voltage Response in Transient Stability Simulation

Adheesh Boratkar
Master of Applied Science (M.A.Sc.)
Graduate Department of Electrical and Computer Engineering
University of Toronto
2019

Large scale power networks are typically simulated only on transient stability analysis (TSA) software. This is sufficient when modelling slow moving ac quantities is of interest and is much faster than using time domain electromagnetic transient programs. However, TSA tools are not capable of modelling the subcycle dynamics well.

Since wide-area measurement systems are practically only modelled on TSA software, while evaluating potential phasor measurement unit (PMU) based control schemes, researchers/utilities depend on TSA results to approximate PMU measurements. This is not sufficient to ensure adequate real-world performance.

This research focused on creating a test bench for characterising the dynamic performance of PMUs. Using voltage magnitude data from this test bench, an improved PMU model for transient stability simulation has been created. The model achieves fits of over 97% across the test cases it was validated with, which is a substantial improvement over the raw TSA simulation results (63-77% accuracy).

I dedicate this thesis to my sister Aditi, and
my parents, Dr. Kumkum and Mr. Ravindra Boratkar.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Zeb Tate, for guiding me through the ups and downs of this journey (difficult courses, my first fracture and surgery, and the pursuit of the 60 Hz sine wave) with immense patience.

I would like to thank the members of the final oral examination committee for my thesis, Professor Joshua Taylor, Professor Aleksander Prodic and Professor Willy Wong, for their time and valuable feedback.

I would like to acknowledge the help and support I received from Professor Tate's doctoral student Zhen Dai. I am also grateful to my colleagues Bibin, Ram, Andres and Phil for their support in troubleshooting various issues along the way. My time at the university was memorable, thanks to my fellow students and friends here.

I would like to thank Dr. Milind Pimprikar, whose guidance was invaluable in my decision of moving to Canada for graduate studies. My special thanks to the Machado and Golawar families who helped with my transition here and continue to act as my safety net in Toronto.

I am eternally grateful to my parents and my sister, for their constant encouragement and support. Finally, I would like to extend my gratitude to my extended family, my friends back home, my former colleagues and teachers for their contribution to my overall success.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Synchronized phasor measurement units (PMUs), which were first introduced in the early 1980s, have been deployed in large numbers in major power systems across the globe over the last two decades [5] [6]. The value of data provided by PMUs was recognized with the occurrence of major power blackouts. This gave momentum to large-scale implementation of wide-area measurement systems (WAMS) using PMUs and phasor data concentrators (PDCs) [6]. As compared to traditional power system measurement information like SCADA, PMU data are more widely available in near real-time [7]. Multiple studies have been published on the optimal placement of PMUs for power system observability and their use in control applications [8] [9] [10].

While validating potential applications of PMUs on large systems is challenging, it is necessary to do so to ensure adequate real world performance. It should be noted that the steady-state behavior of PMUs is well modeled and standardized. On the other hand, the dynamic behavior of PMUs (i.e., what PMUs do outside of steady-state conditions) requires advanced modeling to ensure proper simulation.

In [11], wide-area protection schemes for controlled islanding of the Uruguayan electrical power system are described. The performance of the schemes is compared based on transient stability analysis simulations on the complete and official database of the Uruguayan network. Similarly, in [12], a new synchrophasor-assisted load shedding scheme for a 246-bus Indian system is compared against existing methods. In this work, TSA simulators are used to model PMUs. For such applications outside the steady-state, the use of TSA software to substitute PMU readings is not sufficient. This is explained in detail in Section 1.2.

1.2 Simulation of large transmission networks

Multiple classes of power systems simulation tools exist. Some tools like Simscape Power Systems support multiple simulation modes. Large power networks are typically simulated using phasor based transient stability analysis (TSA) tools like PSS/E. For a given time period of study, these tools solve much faster than the time domain electromagnetic transient programs (EMTP) like PSCAD.

While time domain EMTP tools accurately model all circuit components and lines in terms of their respective resistances, inductances and capacitances, the phasor TSA tools treat all components as static admittances/impedances calculated at the nominal/fundamental frequency of the system. Further, the time step used by phasor TSA simulators (typically about 2 ms) is much larger than that used by their time domain EMTP simulator counterparts (typically about 100 μs).

Therefore, the phasor TSA simulators fail to model some of the sub-cycle details. They are also not capable of modelling phenomenon like travelling waves on transmission lines. TSA simulators are then preferred when modelling slow-moving ac quantities are of interest. The key outputs of TSA software are the magnitude, frequency and phase of voltages on all nodes and currents on all lines at each time step, in addition to internal machine states (e.g., rotor speed, field current, etc.).

When an event that causes a slight change in the system frequency is simulated, the results produced by TSA simulators can drift from real life observations as well as from those calculated by their EMTP counterparts. This is because TSA simulators do not re-evaluate their admittance matrices at the slightly changed frequencies at small enough time intervals. They make this compromise to maintain their speed. In addition, TSA tools rely on simplified generator and load models in comparison to EMTP tools. Therefore, the phasors reported by TSA simulation tools often diverge from real life observations, e.g. from PMUs, during such events.

Figure 1.1 below demonstrates such divergence after a load switching event is simulated on an example network (shown in Figure 1.2) in the TSA simulator PSS/E and compared against the measurements reported by a PMU. While the voltage magnitude values before and some time after the event match closely, Figure 1.3 shows the extent of deviation in the moments just after the event, particularly in the rate of change of the voltage magnitude.

Time domain EMTP tools are capable of providing more accurate PMU simulation, but solving large electrical networks by this method is slow and often impractical. Perhaps more importantly, from a practical standpoint, is that utilities model their complete

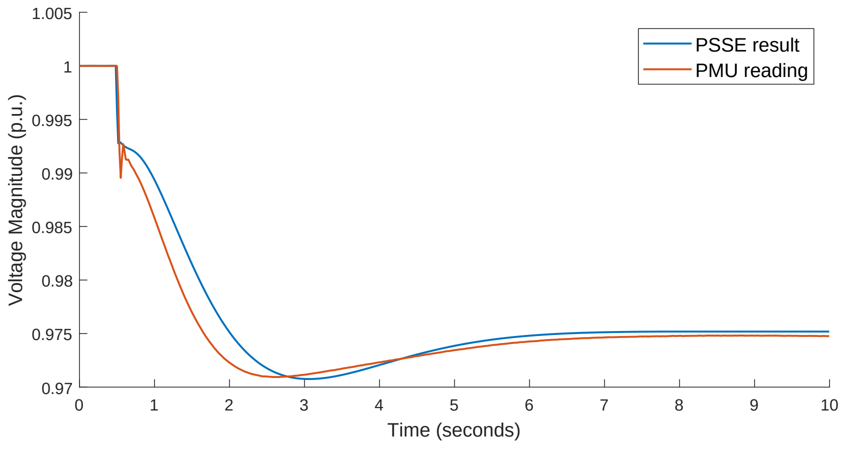


Figure 1.1: Divergence between TSA simulation results and PMU measurements after an event

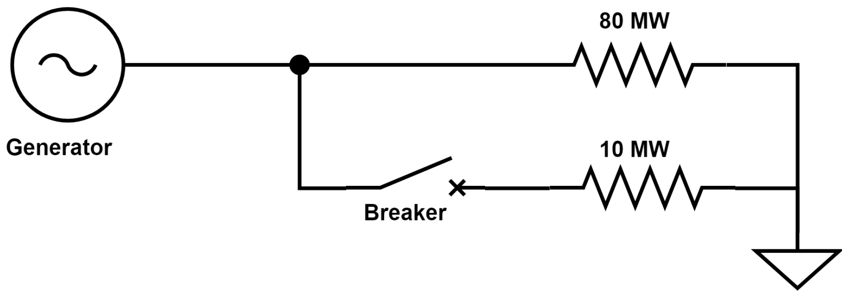


Figure 1.2: Block diagram of the example network used to generate the resistive load step event

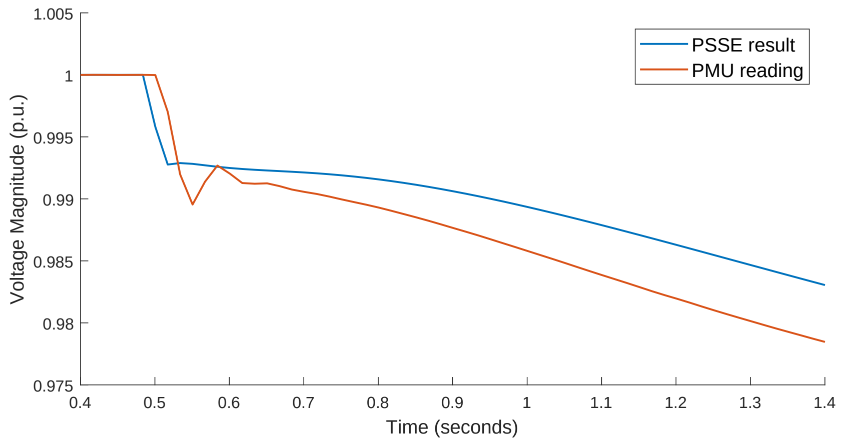


Figure 1.3: Zoomed in view of Figure 1.1 shows the extent of divergence between TSA simulation results and PMU measurements in the moments after the event

large electrical networks only in phasor TSA simulators like PSS/E. Therefore, due to lack of detailed network models, even if time is not a constraint, time domain electromagnetic simulations cannot easily be run for any real, large network.

This compels researchers to depend on the outputs from the phasor TSA simulators to estimate what a PMU on one of the nodes in the network would output. As demonstrated in Figure 1.1, this is not capable of providing an accurate simulation.

This makes evaluation of wide area controllers and monitoring inaccurate. Therefore, the behaviour of controllers produced in studies like [12] and [11] might deviate from the simulated expectations.

1.3 Research objective

This research aims to bridge this gap between the field measurements from devices like PMUs and the results obtained from TSA simulators. The objective is to develop a model of the PMU that can accept the results of a TSA simulator as its input and produce an output that is much closer to the observations reported by a real PMU (as shown in Figures 1.4 and 1.5).

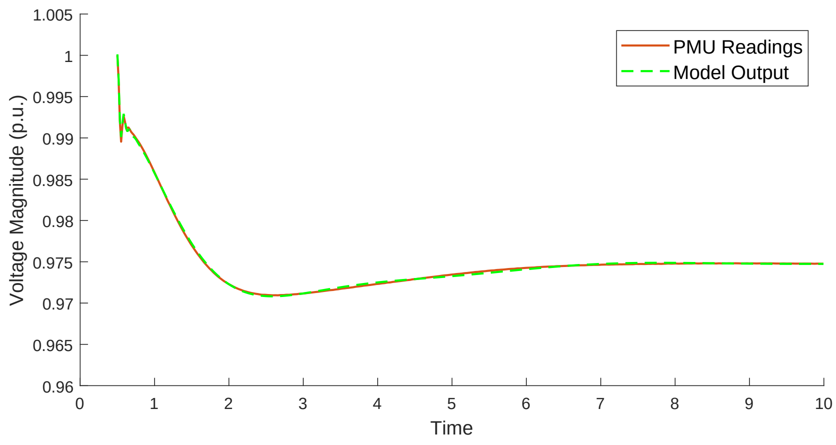


Figure 1.4: TSA simulation results processed through a PMU model compared against measurements from a PMU show a good match

1.4 Methodology overview

To create such a model, phasor TSA simulations as well as the readings from a PMU during the same events are required. Utilities can potentially generate such a dataset by matching their field PMU readings against the PSS/E or PSLF simulation of a historical

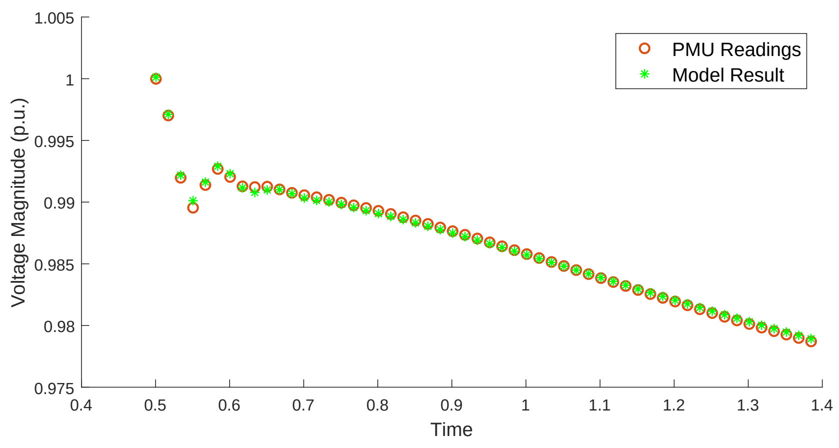


Figure 1.5: Zoomed in view of Figure 1.4 shows the TSA simulation results processed through a PMU model and the PMU measurements

event. In the absence of access to such data, appropriate conditions can also be created in a laboratory setting.

First, an equivalent electrical system is modelled and simulated in both a phasor TSA simulator and a time domain EMTP simulator. The results from the EMTP simulator closely approximate the physical quantities that a field PMU measures (and reports at 60 Hz). These are then fed to a PMU in the lab after appropriate time-synchronized digital-to-analog conversion. The measurements from the PMU and the results of the phasor TSA simulator are then processed using system identification tools to arrive at an improved PMU simulation model. A flowchart of this entire process is shown in Figure 1.6 below. Note that a variety of parameter variations need to be incorporated in event definitions to cross validate the model and prevent overfitting. This is a general procedure that could be repeated to calibrate the PMU model to different TSA simulators as well as measurement instruments from various manufacturers.

This thesis follows the following structure. Chapter 2 discusses the high level design decisions involved in developing the laboratory experiment/test bench. Chapter 3 discusses the grid emulation implemented on digital hardware in detail. Chapter 4 explains the data processing steps at various stages of this research and analyses the results.

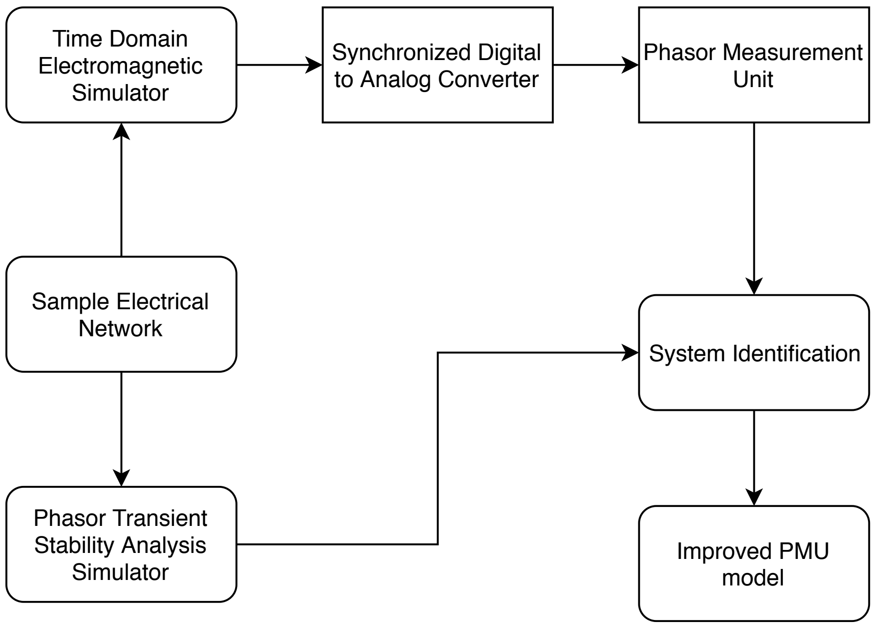

Time Domain Electromagnetic Simulator
Synchronized Digital to Analog Converter
Phasor Measurement Unit
Sample Electrical Network
System Identification
Phasor Transient Stability Analysis Simulator
Improved PMU model

Figure 1.6: Block diagram presents a brief overview of the procedure followed in this research

Chapter 2

Experiment Design

This chapter discusses the high level design decisions involved in developing the laboratory experiment in accordance with the flowchart from the previous chapter. Blocks of the flowchart are sub-tasks of this design process. Some data and models available from previous research greatly accelerated this research.

Since pairs of field PMU measurements and corresponding TSA simulation data from a utility were not available, the approach of recreating similar conditions in a lab was adopted.

2.1 Hardware components of the testbench

2.1.1 The PMU

At the University of Toronto's Energy Systems Lab, the SEL-451 PMU Relay (shown in Figure 2.1) manufactured by Schweitzer Engineering Laboratories (SEL) is available. It was used as the PMU for this research. SEL markets the SEL-451 Relay as a distribution relay featuring auto-reclosing with synchronism check, circuit breaker monitoring and circuit breaker failure protection. It supports extensive metering and data recording features [13].

Inputs to the PMU

The SEL-451 is typically connected to the grid via CTs and PTs. It is capable of accepting 5 A nominal CT inputs and 300 V phase-to-neutral wye configuration PT inputs [13]. The SEL-451 has internal transformers that step down the input from the 300 V PTs to a level acceptable by its internal DAC. The SEL-451 also provides direct access to these DACs via its Low Energy Analog (LEA) port which accepts 8 V ac inputs. Use of the

LEA port enables the experiment to be conducted with low voltage analog signals (of <8 V) instead of 300 V . This greatly simplifies the experiment. Therefore the LEA port was chosen as the input port for the PMU in this research.

Apart from the physical quantities they have to measure, PMUs require one other input, an accurate time reference. For this purpose, the SEL-451 accepts a 5 Vdc IRIG-B signal.

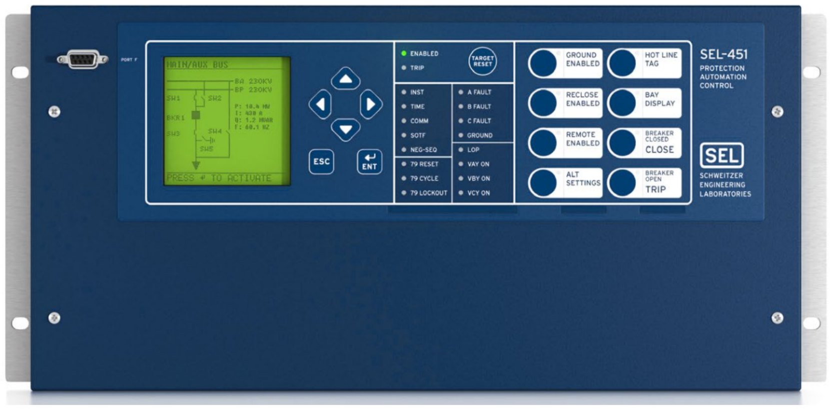


Figure 2.1: SEL-451 PMU relay [1]

The IRIG-B signal

The US military's Inter-Range Instrumentation Group (IRIG) publishes standards for transferring timing information. The standard was first published in 1960 and was last updated in 2016. IRIG-B is a serial time code format that has a rate of 100 pulse per second [14]. The SEL-451 decodes the second, minute, hour, and day fields and sets the internal time clock upon detecting valid IRIG-B time data on its time input port.

In the field, this signal is typically obtained via a GPS receiver. The SEL-2401 Satellite-Synchronized Clock (shown in Figure 2.2), which provides timing accuracy of $\pm 100ns$, is available in the University's energy systems lab. This interfaces with the PMU.

However, in a laboratory setting, the IRIG-B could be digitally generated, thereby eliminating the requirement for a GPS receiver/clock. It would have to be implemented on digital hardware such as an FPGA board with a precise clock. Dedicated logic can be built by following the standard that specifies how each of the 10 millisecond long pulses signify. For instance, the time-of-year and year information is encoded in BCD format, and seconds-of-day are encoded as "Straight binary seconds (SBS)", a 17-bit binary counter that counts from 0 to 86399 [14]. Significant additional effort would have been required to implement and verify such a set up due to the absence of an IRIG-B library in LabView; thus, it was decided to rely on a GPS-connected clock as the IRIG-B source.

Figure 2.2: SEL-2401 satellite-synchronized clock [2]

2.1.2 Specifications required for grid voltage emulation

The primary focus of this research is studying the deviation in the magnitude of the voltage reported in synchrophasor measurements. Therefore, the PMU's voltage ports need to be stimulated. As discussed in section 2.1.1, a decision was made to achieve this by providing a ≈ 8 V signal to the LEA port rather than a ≈ 300 V signal to the PT port. This decision simplified the design of this part of the work bench considerably as no relatively high voltages were involved.

This 8 V signal fed to the PMU needs to reflect events happening on the grid, which necessitates the ability to generate a signal that represents the stepped down version of the grid voltage. To generate this signal, the event itself is modelled and simulated in an EMTP simulators, with the time domain response captured in the standard common format for transient data exchange (COMTRADE) [15]

The next step is to translate the digital voltage waveform from the COMTRADE file into the appropriate input into the PMU's LEA ports.

The analog reconstruction of the voltage waveform must also be reproduced with accurate timestamps, in order to match (in time) the output of the EMTP tool. It therefore needs a DAC capable of producing 8 V and a mechanism of accurately timing the updates to the DAC. The system also needs to be capable of reproducing each cycle of the ac waveform with a sufficiently high number of samples to ensure the wave presented to the PMU is an accurate representation of the voltage waveform. Furthermore, this system should have the ability to time synchronize with the PMU.

In order to determine the minimum sampling rate of the grid voltage emulation signal, it is essential to know the rate at which the PMU samples the signals fed to it. The section on monitoring and metering in the PMU's datasheet specifies 8 kHz as the highest resolution at which the PMU can report data [13]. With 8 kHz as the sampling frequency of the PMU's internal ADC, the sampling rate of the DAC should be 8 kS/s or higher.

One way of achieving these functional requirements (GPS-synchronized +/-8V analog output from a COMTRADE file with waveforms sampled at over 8 kS/s) was to utilize the real time embedded industrial controller from National Instruments, the Compact RIO.

2.1.3 The CompactRIO

The National Instruments Compact Reconfigurable I/O system (cRIO) is used to generate the signal provided to the LEA input of the PMU. A cRIO consists of a processor that runs a real-time operating system (RTOS), a reconfigurable FPGA and multiple interchangeable I/O modules. The processor is used for controlling the overall behaviour of the cRIO unit while the FPGA is used to carry out smaller, detailed tasks which need to be executed with precise timing. A general block diagram of the cRIO architecture is shown in Figure 2.3 below.

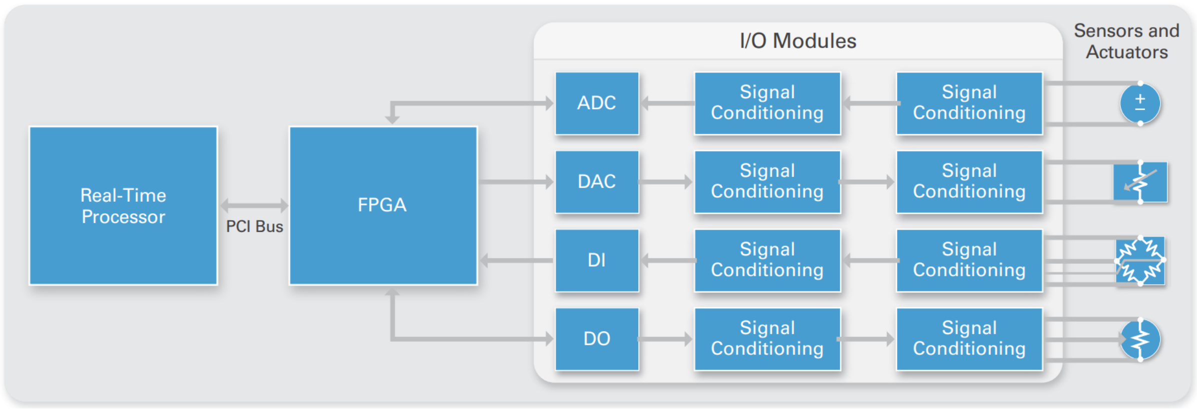


Figure 2.3: General architecture of the cRIO [3]

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is a system-design platform and development environment from National Instruments. The cRIO is configured using LabVIEW. LabVIEW programs/subroutines are termed Virtual Instruments (VIs). Multiple VIs are created and linked together to fulfill the task of generating a time synchronized LEA voltage input for the PMU based on the COMTRADE signal.

The specific model of the cRIO available in the Energy Systems Lab is the cRIO-9035 (shown in Figure 2.4 below). It is an 8 slot embedded real-time controller with a 1.33 GHz Dual-Core CPU, 1 GB DRAM, 4 GB Storage, and a Kintex-7 70T FPGA. Each of the 8 I/O slots can be populated with I/O modules from NI or third-party vendors [16].

Figure 2.4: NI cRIO 9035 [4]

For this research, three such modules were used. The analog output module NI9263 is a 4 channel voltage output module with a 16-bit DAC and a nominal voltage output range of 10 V. It is capable of updating each channel at 100 kS/s. NI9215 is a similar 4 channel analog input module with a 16-bit ADC and a nominal voltage input range of 10 V. It is capable of updating each channel at 100 kS/s as well. The NI 9467 is a stationary GPS timing module. It provides accurate timing and geographic location information to the cRIO host, which enables time synchronization of the signal generator and the PMU.

Since, the cRIO's GPS module NI 9467 also has the same accuracy as the PMU's GPS clock (100 ns), in interest of time and simplicity it was decided to use the individual GPS modules on these subsystems.

2.2 System simulation

For this research, equivalent example electrical systems needed to be modelled in both a phasor TSA simulator and a time domain EMTP simulator. From prior research carried out by Zhen Dai, models with matched behavior in PSS/E and PSCAD were available. In that study, PSS/E from Siemens was used as the phasor TSA simulator and EMTDC

PSCAD was used as the time domain electromagnetic simulator. For this research, these models were used with minor modifications.

2.2.1 The example electrical system

The example electrical system (shown in Figure 2.5) used in this study consists of a synchronous generator connected to an 80 MW resistive load. After an interval, an additional 10 MW resistive load is added on to the load bus via a breaker to create the event. The system and the machine are rated 22 kV line-to-line.

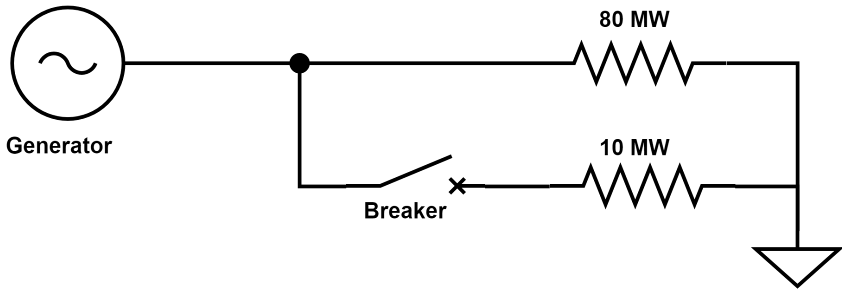


Figure 2.5: Block diagram of the example electrical network used to create the load switching event

PSCAD simulation

The PSCAD model of the example electrical system is shown in Figure 2.6. This simulation was run for 180 seconds at a time step of 10 μs. The generator stabilizes after start up within the first 5 seconds. At 100.5 seconds, the breaker is closed adding the 10 MW resistive load to the 3 phase system. This causes a transient in the voltage and frequency. The voltage and frequency stabilize within the 108^{th} second. For this research, the voltage time series from the 95 second mark till the 110 second mark was exported to a COMTRADE file. Data to this file was written at intervals of 100 ms. This was sufficient to capture all the dynamics of interest.

Since this research focuses on the deviation in the reported voltage amplitudes, the generators exciter time constant is varied from 0.5 s to 2.0 s in steps of 0.5 s to create multiple scenarios.

2.3 Brief overview of the data processing

The time series voltage data from PSCAD was resampled using the cubic spline interpolation in MATLAB to match the sampling frequency required by the cRIO and its

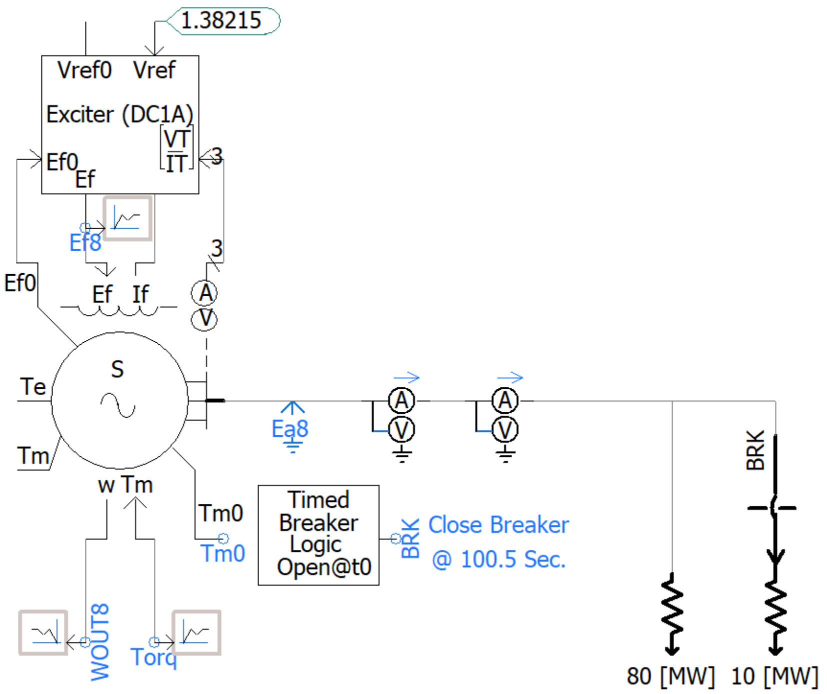


Figure 2.6: Model in PSCAD of the example electrical network used to create the load switching event

LabVIEW VIs. This suitable sampling frequency, which was empirically determined, was not a whole number μs. Therefore, data could not be directly sampled from PSCAD. This is explained in detail in Chapter 4. The cRIO reproduces the 15 second period of study from the PSCAD simulation and the PMU records the corresponding synchrophasor observations. This process was repeated for the other three values of the generators exciter time constant. Chapter 3 provides a detailed explanation of this process.

MATLABs System Identification Toolbox was used to compare the PMU outputs with the results from PSS/E across all the cases. Various models were fit on to this data to arrive at PMU like measurements directly from the PSS/E output. Details about the model selection and parameter fitting are discussed below, in Chapter 4.

Chapter 3

Detailed Implementation on the CompactRIO

This chapter discusses the programs deployed on the cRIO to make it emulate the grid voltage for the PMU. The software environment used to program the cRIO, LabVIEW, is a system-design platform and development environment from National Instruments which relies on a graphical programming structure. LabVIEW programs and subroutines are termed Virtual Instruments (VIs); the remainder of this chapter focuses on the VIs used for grid emulation, along with the design decisions made to enable accurate testing and identification of the PMU simulation model.

3.1 Basic structure of a LabVIEW VI

LabVIEW VIs are graphical programs, similar to MATLAB/Simulink. A cRIO LabVIEW project can consist of VIs running on three hierarchical levels: the host PC the cRIO is attached to, the cRIO chassis/processor, and the FPGA target, as shown in Figure 3.1. Each VI consists of a front panel for controls & displays and a back-end block diagram that captures the interactions between different components.

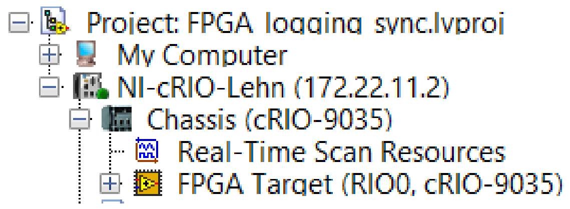

Project: FPGA logging sync.lvproj
My Computer
NI-cRIO-Lehn (172.22.11.2)
Chassis (cRIO-9035)
Real-Time Scan Resources
FPGA Target (RIO0, cRIO-9035)

Figure 3.1: LabVIEW Project Explorer shows the different levels at which VIs are deployed

To illustrate the basic setup of a VI, consider a simple example that produces a desired analog voltage (via the NI 9263 analog output module) and reads it back (via the NI 9215 analog input module). Figure 3.2 shows the front panel of the VI on the LabVIEW FPGA target. Right-clicking on the front panel opens the Controls menu from which various controls and displays can be added. In this example (see Figure 3.2), one numeric control (input), 'Target Voltage', is provided to set the desired analog output voltage. A numeric indicator display, "Voltage Readout", is also provided to display the value read from the analog input module.

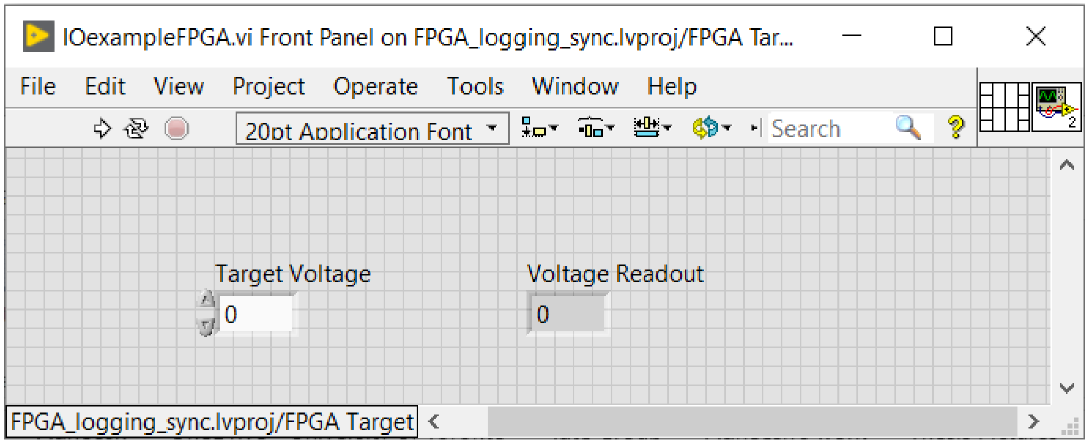

IOexampleFPGA.vi Front Panel on FPGA_logging_sync.lvproj/FPGA Tar...
File
Edit
View
Project
Operate
Tools
Window
Help
20pt Application Font
Search
Target Voltage
0
Voltage Readout
0
FPGA_logging_sync.lvproj/FPGA Target

Figure 3.2: Front Panel of the example VI

Figure 3.3 shows the corresponding Block Diagram that is automatically generated by LabVIEW as a result of adding these two components to the front panel. These two blocks must be connected to resources present on the cRIO, which are accessed via the Project Explorer panel under "FPGA Target" (as shown in Figure 3.4). Physically,

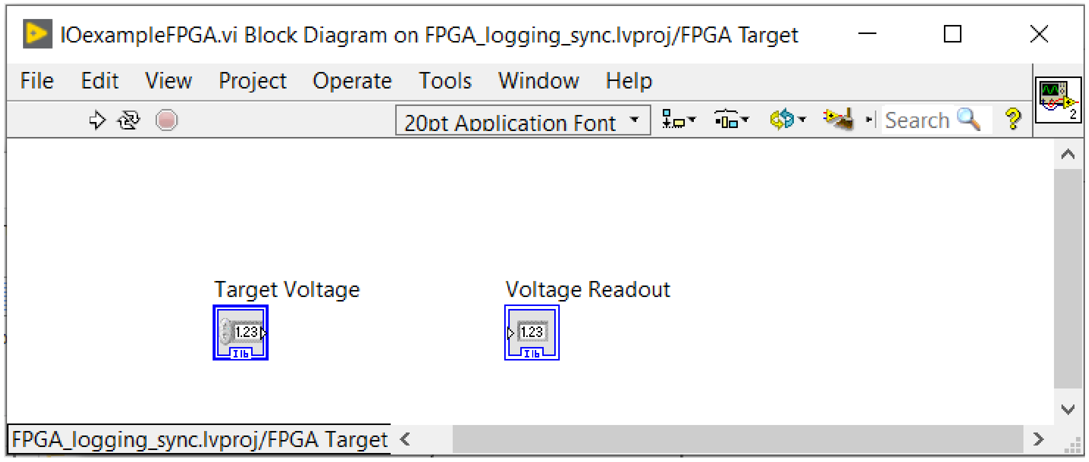
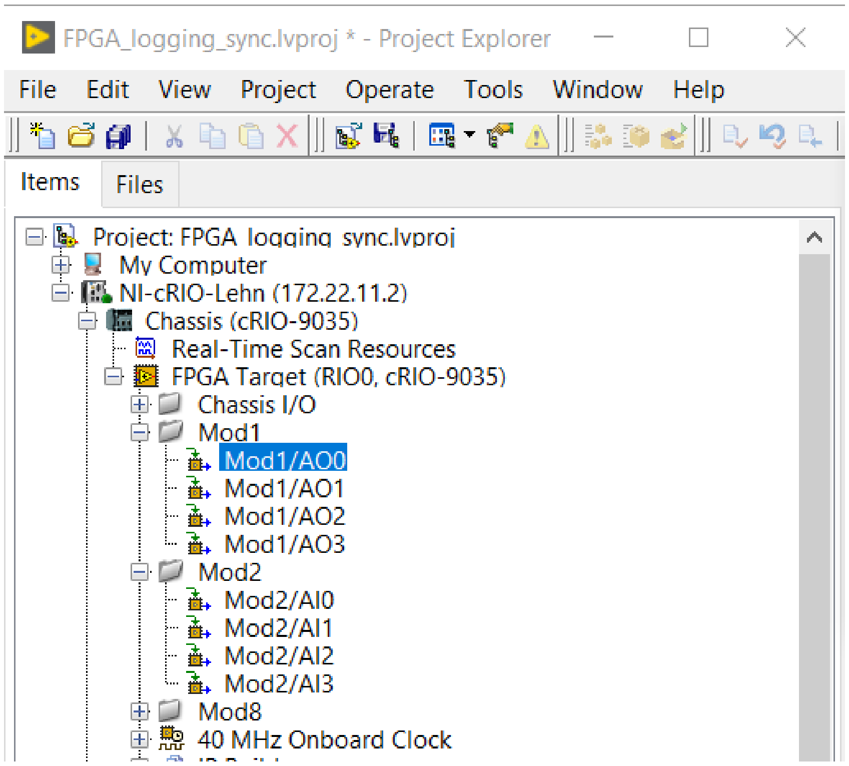

IOexampleFPGA.vi Block Diagram on FPGA_logging_sync.lvproj/FPGA Target
File
Edit
View
Project
Operate
Tools
Window
Help
20pt Application Font
Search
Target Voltage
1.23
Voltage Readout
1.23
FPGA_logging_sync.lvproj/FPGA Target

Figure 3.3: The auto-generated Block Diagram of the Example VI



```
Project: FPGA_logging_sync.lvproj
  My Computer
  NI-cRIO-Lehn (172.22.11.2)
    Chassis (cRIO-9035)
      Real-Time Scan Resources
      FPGA Target (RIO0, cRIO-9035)
        Chassis I/O
        Mod1
          Mod1/AO0
          Mod1/AO1
          Mod1/AO2
          Mod1/AO3
        Mod2
          Mod2/AI0
          Mod2/AI1
          Mod2/AI2
          Mod2/AI3
        Mod8
        40 MHz Onboard Clock
```

Figure 3.4: LabVIEW Project Explorer shows the I/O and clock resources on the FPGA

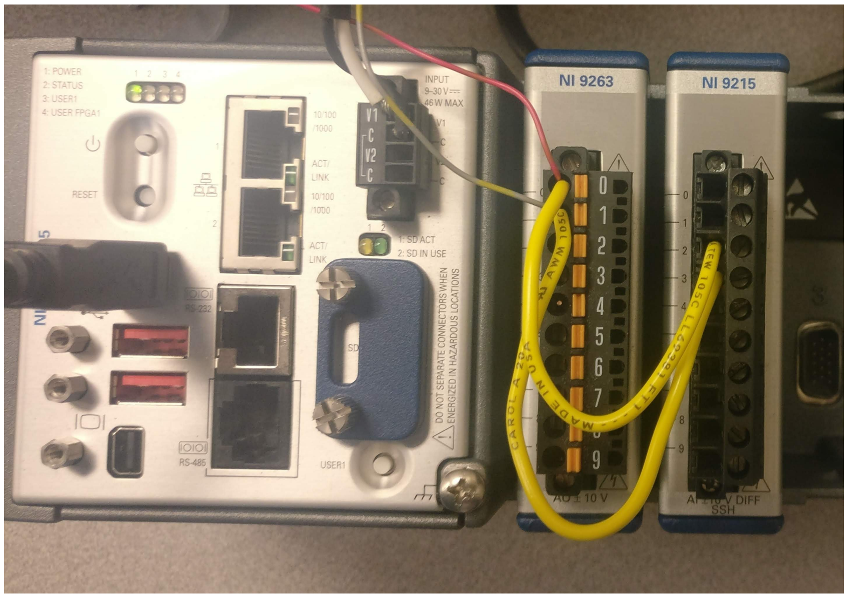


Figure 3.5: Physical connections made on the cRIO to read-back the AO0 port output on the AI1 port

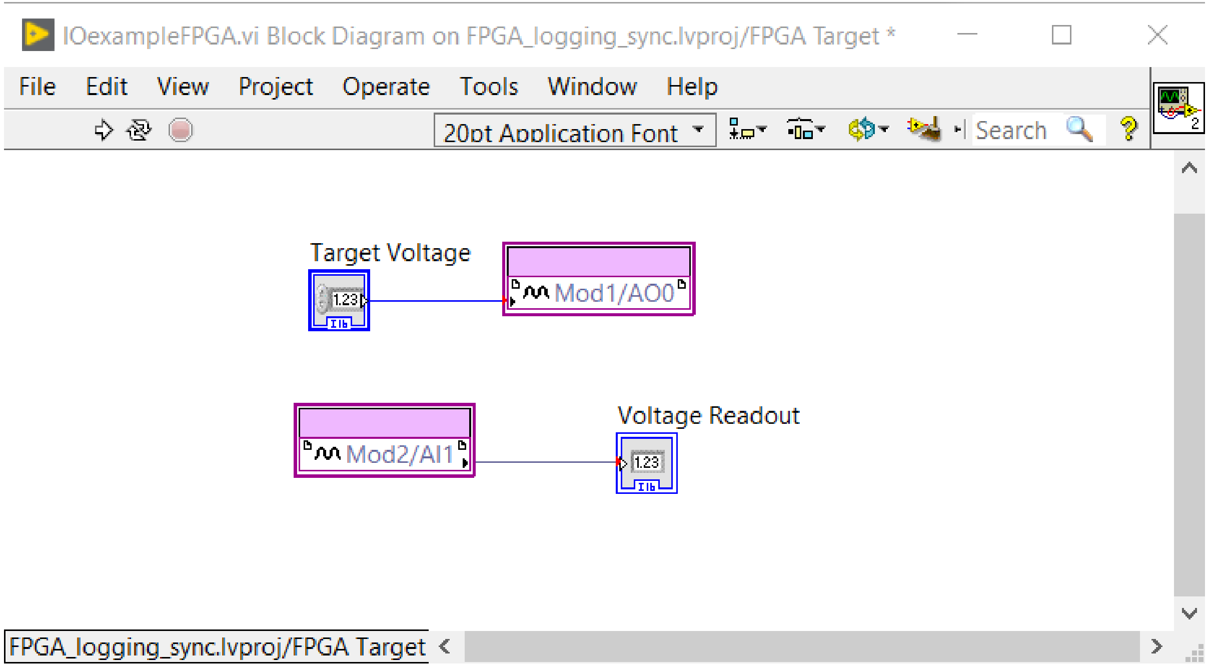
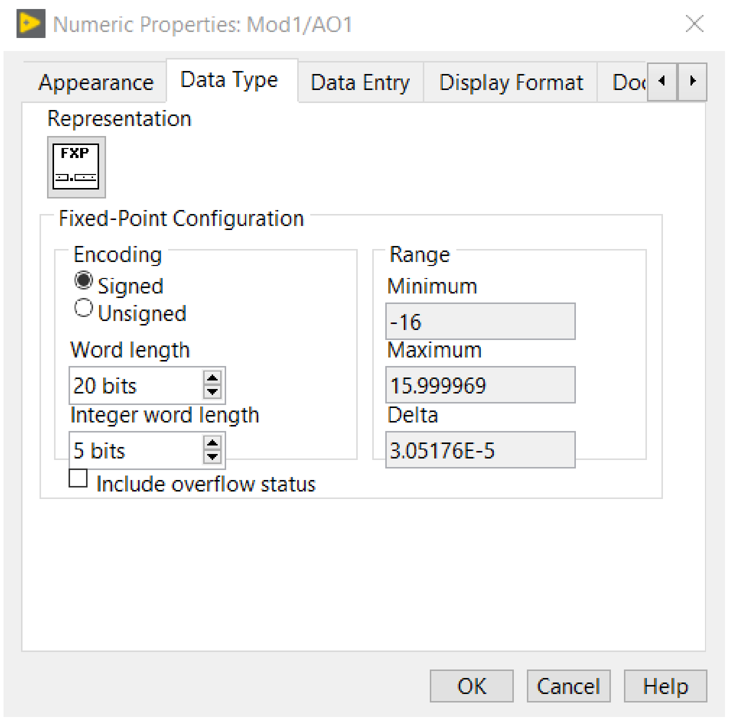

IOexampleFPGA.vi Block Diagram on FPGA_logging_sync.lvproj/FPGA Target *
File Edit View Project Operate Tools Window Help
20pt Application Font
Search
Target Voltage
Mod1/AO0
Mod2/AI1
Voltage Readout
FPGA_logging_sync.lvproj/FPGA Target

Figure 3.6: Block Diagram of the example VI, connections of the input and the display to the appropriate I/O module source and sink are shown

Channel 0 of the analog output module (AO-0) is connected to Channel 1 of the analog input module (AI-1) (see Figure 3.5). Therefore, in the Block Diagram, it is necessary to connect the numeric control “Target Voltage” to AO-0 and the numeric indicator “Voltage Readout” to AI-1, as shown in Figure 3.6. Notice the red dots at the end of the lines that connect the numeric control to the AO module and the AI module with the numeric display. These red dots indicate a data type mismatch. In Figure 3.6, the “I-16” written along the bottom of the “Target Voltage” and “Voltage Readout” blocks indicate that these two blocks, by default, expect to provide (or consume, in the case of the “Voltage Readout” block) 16-bit integer values. On the other hand, the AO and AI modules are configured to use a fixed-point, 20-bit representation of the voltage (see Figure 3.7). As indicated in the “Integer word length” field, the first 5 bits are used to represent the signed integer component of the data; the remaining 15 bits are used to represent the fractional component. Thus, the representable values range from -16 to $(16 - 2^{-15})$, and the smallest difference between two values (“Delta”) is 2^{-15}. Note that these values are higher than the capabilities of the hardware in terms of both range (+10 V to -10 V) and resolution (16-bit). The datatypes of the numeric control, “Target Voltage” and the numeric indicator “Voltage Readout”, therefore have to be changed from the default of I-16 to signed fixed-point with 20 bit total word length and 5 bit integer component. After this change, the notations below the “Target Voltage” and “Voltage


Numeric Properties: Mod1/AO1

Representation: FXP

Fixed-Point Configuration

Encoding: [✓ Signed] Unsigned

Word length: 20 bits

Integer word length: 5 bits

☐ Include overflow status

Range

Minimum: -16

Maximum: 15.999969

Delta: 3.05176E-5

Figure 3.7: Numeric data-type specification for the AI/AO modules

Readout" blocks change to "FXP" and the red circles indicating a type mismatch change to black (see Figure 3.8).

Like most programming languages, LabVIEW has flow control structures like while, for, case etc. Here, because the program should run continuously, the input/output connections are placed inside a a while loop. This is represented by the outer rectangle in Figure 3.8. The large red dot on the bottom right corner connected to a button labelled "Stop" provides a way to stop the continuous execution of this while loop from the front end (see Figure 3.9). Inside the while loop is a Flat Sequence Structure. While tasks in each individual section of this structure are executed in parallel, the tasks in the adjacent sections are run sequentially, i.e., tasks in the second section are run only after the fist section finishes execution. Here, in the first subsection, the Analog values are being both written and read from the I/O modules and the next section is merely a wait state. The amount of time the wait state is held for can be controlled from the front end by specifying the value of the "Tick Count" in milliseconds. Internally this is implemented by a 32-bit counter running at the FPGA's base frequency of 40 MHz.

In Figure 3.9, it can be noticed that when the target output voltage is set to 4 V and the readout is 3.996 V. The typical accuracy specification of these devices mentions that calibrated NI 9263 AO module and the NI 9215 AI module could suffer from 0.0214 V

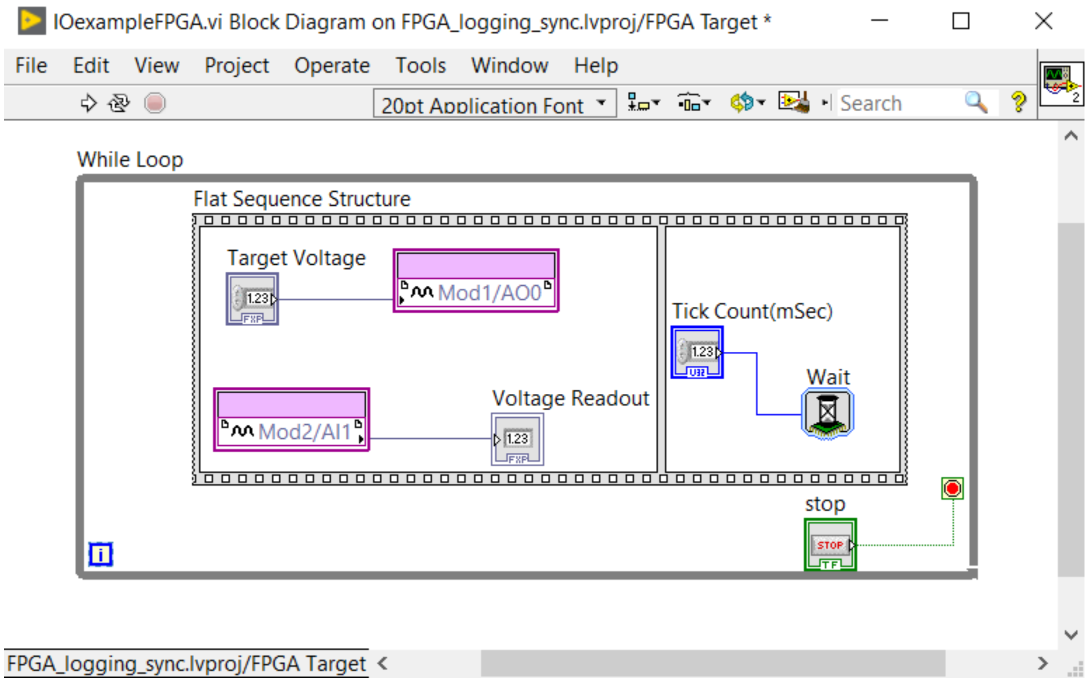

IOexampleFPGA.vi Block Diagram on FPGA_logging_sync.lvproj/FPGA Target *
File Edit View Project Operate Tools Window Help
20pt Application Font
Search
While Loop
Flat Sequence Structure
Target Voltage
Mod1/AO0
Mod2/AI1
Voltage Readout
Tick Count(mSec)
Wait
stop
FPGA_logging_sync.lvproj/FPGA Target

Figure 3.8: Block Diagram of the example VI. The data-types are now compatible and flow control structures have been implemented

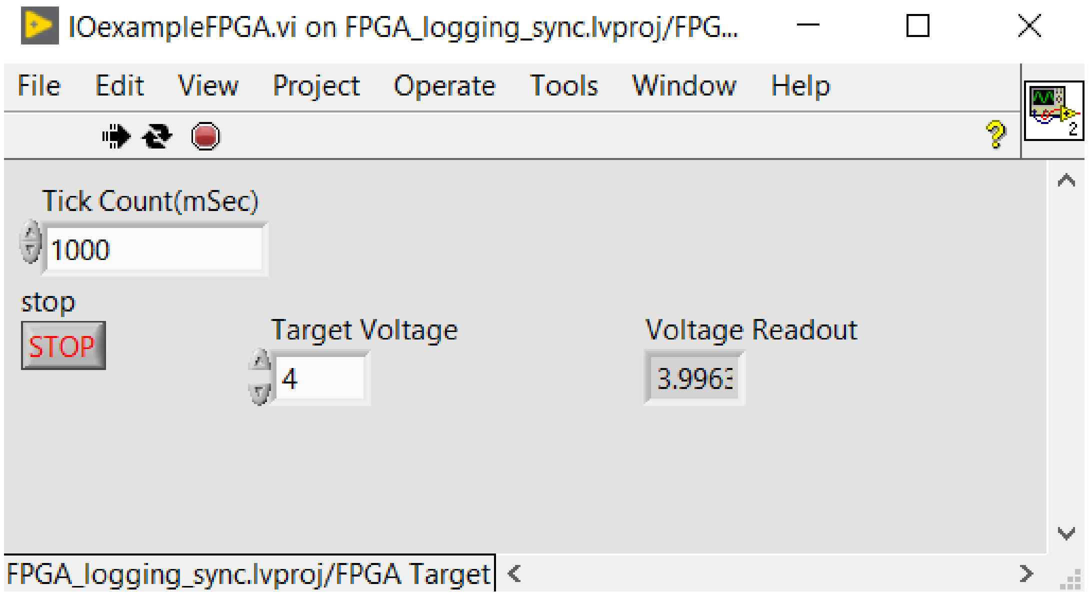
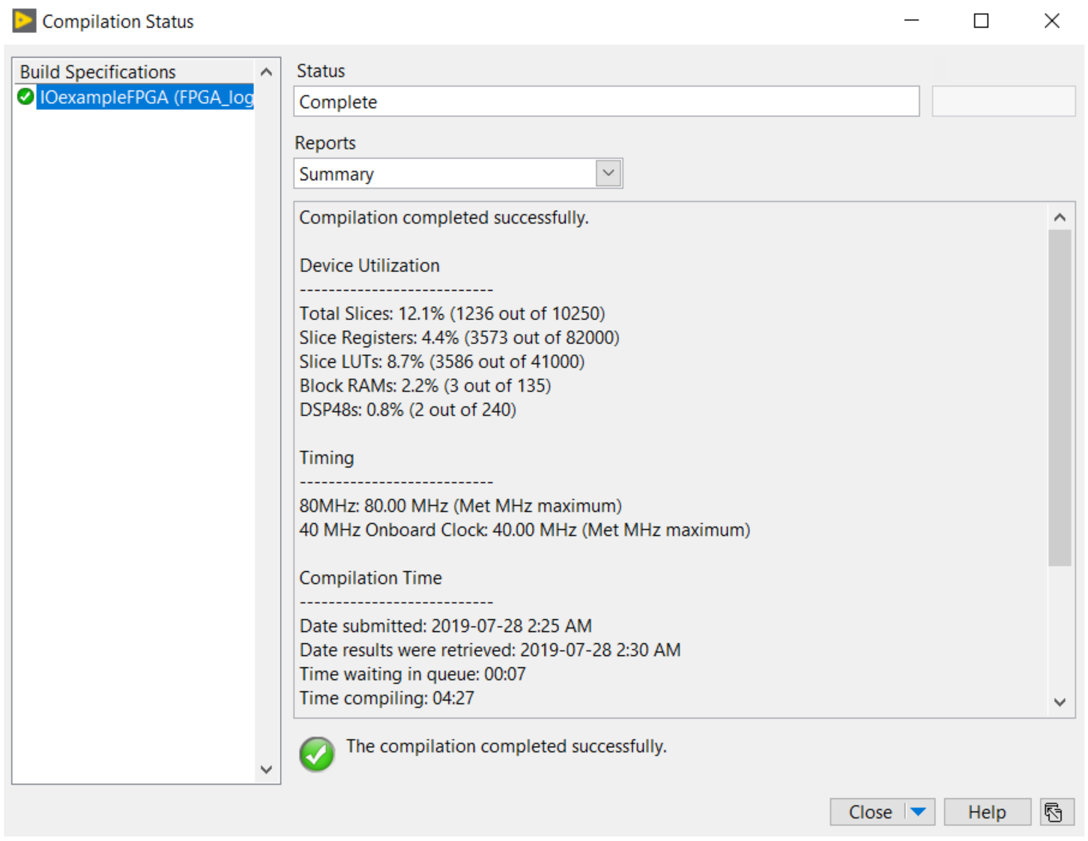

IOexampleFPGA.vi on FPGA_logging_sync.lvproj/FPG...
File Edit View Project Operate Tools Window Help
Tick Count(mSec)
1000
stop
STOP
Target Voltage
4
Voltage Readout
3.9963
FPGA_logging_sync.lvproj/FPGA Target

Figure 3.9: Example VI Front Panel shows the ‘Target Voltage’ input, the ‘Voltage Readout’ display, the ‘Tick Count’ hold time control, and the ‘STOP’ stop execution button

(0.1% of its typical range of ±10.7 V) and 0.002912 V (0.014% of its typical range of ±10.4 V) offset respectively. Therefore, the error of 0.004 V (= 4 - 9.996) observed here is within the 0.024312 V (= 0.0214 + 0.002912) offset the modules could cumulatively suffer when calibrated.

All VIs to be deployed on the FPGA need to be compiled. This process consumes time. A 64-bit Windows system with an Intel Xeon E5-16200 processor running at 3.6 GHz and 8 GB RAM was used for all of these experiments. Figure 3.10 shows the compilation summary, which includes detailed information on the FPGA utilization (number of slice registers, LUTs, block RAMs, and DSPs). Once the VI is compiled, it can then be deployed and executed on the cRIO.



Status: Complete

```
Compilation completed successfully.

Device Utilization
---------------------------
Total Slices: 12.1% (1236 out of 10250)
Slice Registers: 4.4% (3573 out of 82000)
Slice LUTs: 8.7% (3586 out of 41000)
Block RAMs: 2.2% (3 out of 135)
DSP48s: 0.8% (2 out of 240)

Timing
---------------------------
80MHz: 80.00 MHz (Met MHz maximum)
40 MHz Onboard Clock: 40.00 MHz (Met MHz maximum)

Compilation Time
---------------------------
Date submitted: 2019-07-28 2:25 AM
Date results were retrieved: 2019-07-28 2:30 AM
Time waiting in queue: 00:07
Time compiling: 04:27
```

The compilation completed successfully.

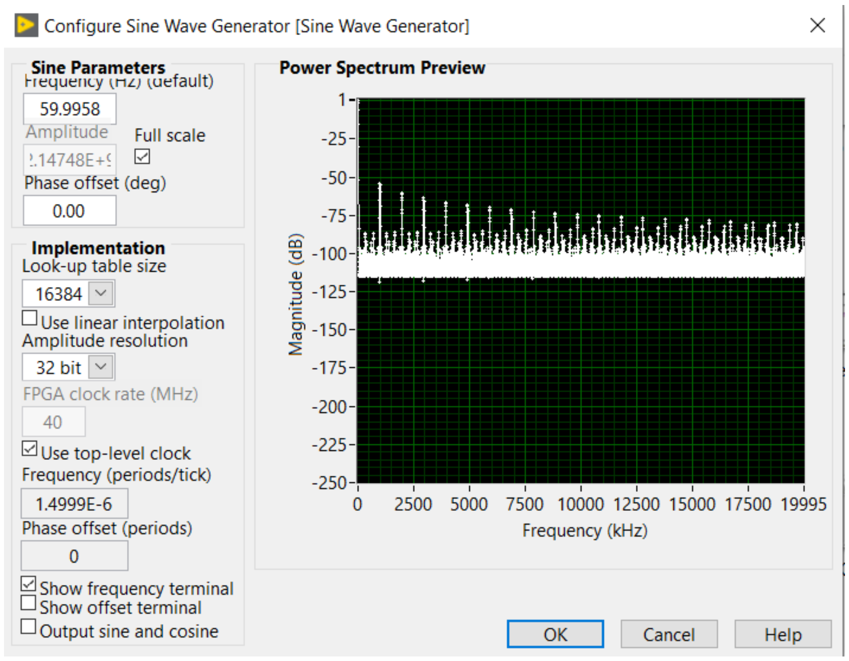
Figure 3.10: Summary of the FPGA compilation shows the FPGA resource utilization and the compilation time

3.2 Generating a 60 Hz sine wave

This subsection discusses the various ways of generating a sine wave at a desired frequency and amplitude on the cRIO. This capability allows for testing of the steady-state behavior of the PMU, which is well-defined by the C37.118 standard.

3.2.1 Method 1: The Sine Wave Generation block on the FPGA

LabVIEW comes with a built-in sine wave generation block (See Figure 3.11). This block generates a sine wave point-by-point using direct digital synthesis. For implementing the sine wave, it needs to be assigned a look up table size and an amplitude resolution. Tuning these parameters has an impact on the power spectrum of the sine wave. It is a tradeoff between the power spectrum of the resulting sine wave and the FPGA's resources–using a larger LUT results in better accuracy, but the physical implementation on the FPGA of a large LUT consumes more FPGA resources. For guidance, the expected power spectrum is updated on the display every time the amplitude resolution and LUT size are modified. In this example, the largest permissible values are chosen for the LUT size (16384) and the amplitude resolution (32 bit). The block also accepts the desired frequency in Hz and calculates a parameter "Frequency", with units of periods per tick. Here, one tick corresponds to one clock cycle and the period is the inverse of the desired sine wave frequency. This block can be configured to generate sine waves at designated frequencies.


Configure Sine Wave Generator [Sine Wave Generator]

Sine Parameters
Frequency (Hz) (default): 59.9958
Amplitude: .14748E+
Full scale ☑
Phase offset (deg): 0.00

Implementation
Look-up table size: 16384
☐ Use linear interpolation
Amplitude resolution: 32 bit
FPGA clock rate (MHz): 40
☑ Use top-level clock
Frequency (periods/tick): 1.4999E-6
Phase offset (periods): 0
☑ Show frequency terminal
☐ Show offset terminal
☐ Output sine and cosine

Figure 3.11: Sine Wave Generation block parameters

However, due to the way the sine wave generator is implemented, the frequency of the generated signal can only change in discrete steps. For example, in its current setting, it cannot generate a wave of exactly 60 Hz. Instead, it can generate waves of either 59.9958

Hz or 60.0051 Hz. LabVIEW contains methods of digitally deriving higher frequency clocks from the default clock, however they are not as accurate as the physical 40 MHz clock and hence were not used.

Figure 3.12 shows the block diagram of a VI that uses this Sine Wave Generator block. The block takes frequency (periods/tick) as an input. The FXP block at the output of the Sine Wave Generator block converts its 32-bit signed integer output to a fixed point value. This is then divided by its highest possible magnitude (2^{31}). At this point, we have a high resolution sine wave with a unit magnitude. The multiplication block takes an amplitude input from the front panel and generates a sine wave of the desired amplitude. This is again converted to a FXP specification that matches with the requirements of the analog output (AO) module. The same signal is fed to two channels of the 4-channel AO module. One is fed to the PMU and the other is connected to an oscilloscope to monitor the output. The instantaneous output is displayed on the Front Panel via the "instantaneous output" display block (see Figure 3.13).

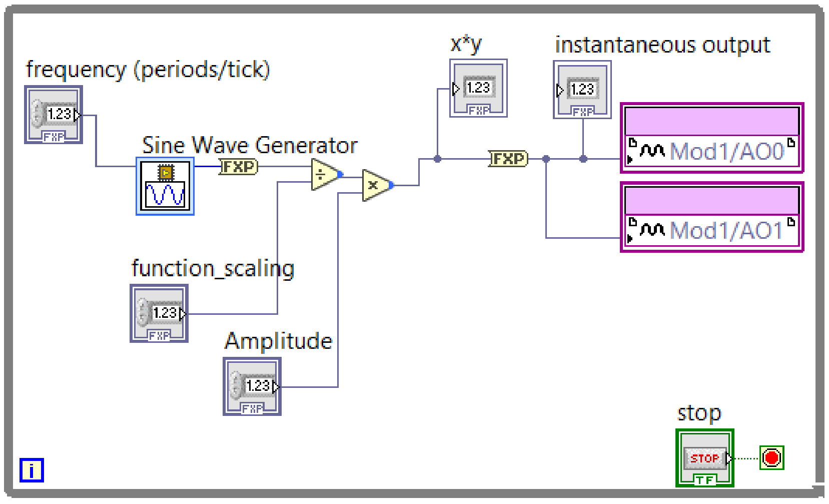
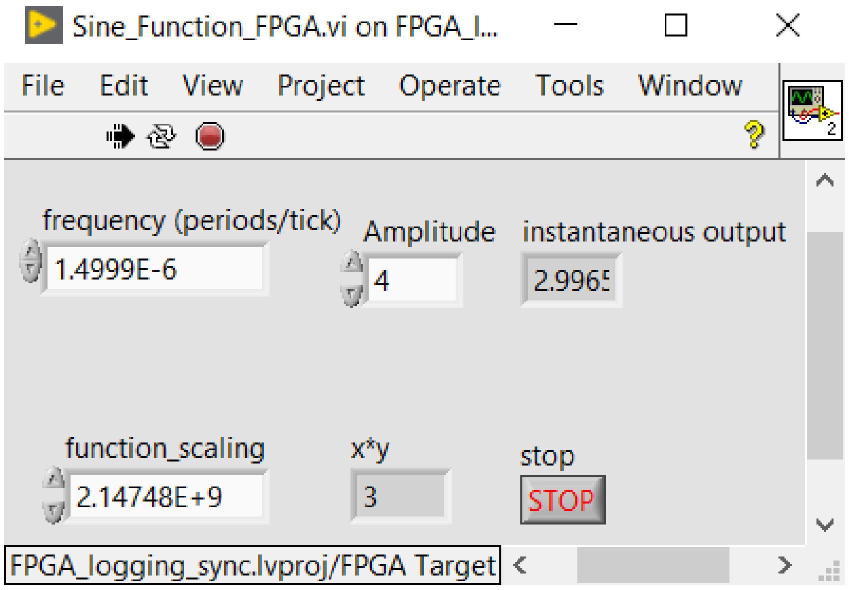


Figure 3.12: Block Diagram of the VI with Sine Wave Generation block feeding 2 AO ports

Figures 3.14 and 3.15 show the resulting sine wave and its FFT on an oscilloscope. The oscilloscope records the dominant frequency of the signal as 60.2 Hz. In Figure 3.15, in the FFT, we see that most of the power is concentrated near the peak around ~ 60 Hz.

There is a fundamental challenge using the increment-per-tick method described above. The tick length corresponding to the 40 MHz clock is 2.5×10^{-8} s, and the period of a 60 Hz sine wave is $0.01\overline{6}$ s. Here, we use an overline to indicate infinitely-repeated decimals. Therefore, in order to generate exactly 60 Hz from a 40 MHz clock,


Sine_Function_FPGA.vi on FPGA_I...
File
Edit
View
Project
Operate
Tools
Window
STOP
FPGA_logging_sync.lvproj/FPGA Target

frequency (periods/tick): 1.4999E-6
Amplitude: 4
instantaneous output: 2.9965
function_scaling: 2.14748E+9
x*y: 3
stop

Figure 3.13: Front Panel of the VI with Sine Wave Generation block: frequency and amplitude of the wave can be controlled

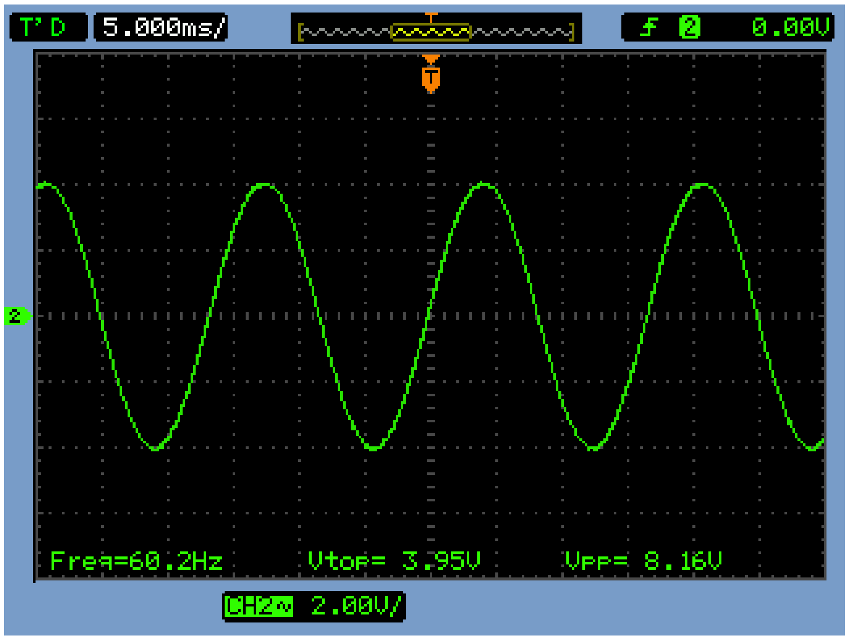


Figure 3.14: Oscilloscope trace of output from Sine Wave Generator VI shows a sine wave with a frequency close to 60 Hz

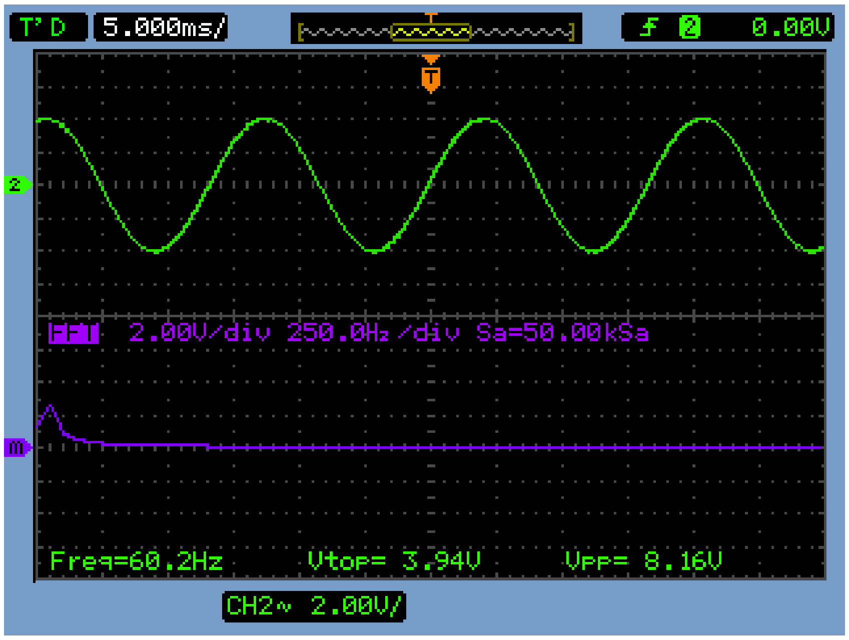

T'D 5.000ms/
0.00V
FFT 2.00V/div 250.0Hz/div Sa=50.00kSa
Freq=60.2Hz Vtop= 3.94V Vpp= 8.16V
CH2 2.00V/

Figure 3.15: Oscilloscope FFT of output from Sine Wave Generator VI: Most of the power is concentrated near 60 Hz

a non-integer number of ticks ($666666.\overline{6}$) are contained within one period of the desired sinusoid. The closest integer, 666666, result in a total sinusoidal period of 0.01666665 seconds, which in turn corresponds to a sinusoidal frequency of 60.00006000006 Hz (the next-highest integer value, 666667, corresponds to a frequency of 59.999970000015 Hz). Due to limitations of array sizes and the fact that executing an instruction, reading a value, processing it, and passing it to the AO module takes multiple clock cycles, the built-in Sine Wave Generation block cannot generate a precise 60 Hz sine wave but only either 59.9958 Hz or 60.0051 Hz. Therefor, an alternative sine wave generator was developed.

3.2.2 Sine wave generation using a circular array

One of the simplest ways of generating any repeating waveform on the cRIO is to store an array of constants in the FPGA and sequentially and cyclically passing the elements from the array to the AO module. When implemented in this fashion, the number of elements in the array define the smoothness of the resulting sine wave, and the number of wait state ticks between updates in the output value determine the waveform's frequency. The shape of the waveform is determined by the contents of the array. In this example (see Figure 3.16), an array of 1000 points forming a sine wave is used. Since the signal in PSCAD is sampled at about 167 samples per period, a waveform with 1000 points per period was thought to be adequate for this general test. A much larger number of samples was not chosen taking into consideration, the limited resources like LUTs on the FPGA. The values in the array are for a sine wave with an amplitude of 0.5. A For loop is used to read out one element at a time from the array. The counter variable for this loop can be controlled to produce repeating half or quarter sine waves, etc. Inside the for loop, the 0.5 amplitude sine wave is multiplied by 2 to make it a sine wave with an amplitude of 1. Next, it is multiplied by an input from the Front Panel (see Figure 3.17) to generate a sine wave with the desired amplitude. Each element is then passed in to the Flat Sequence Structure. Here, in the first frame, the elements are written to the AO module and, in the next frame, the program holds based on the wait state input from the front panel. During this wait time, the AO module holds its output voltage.

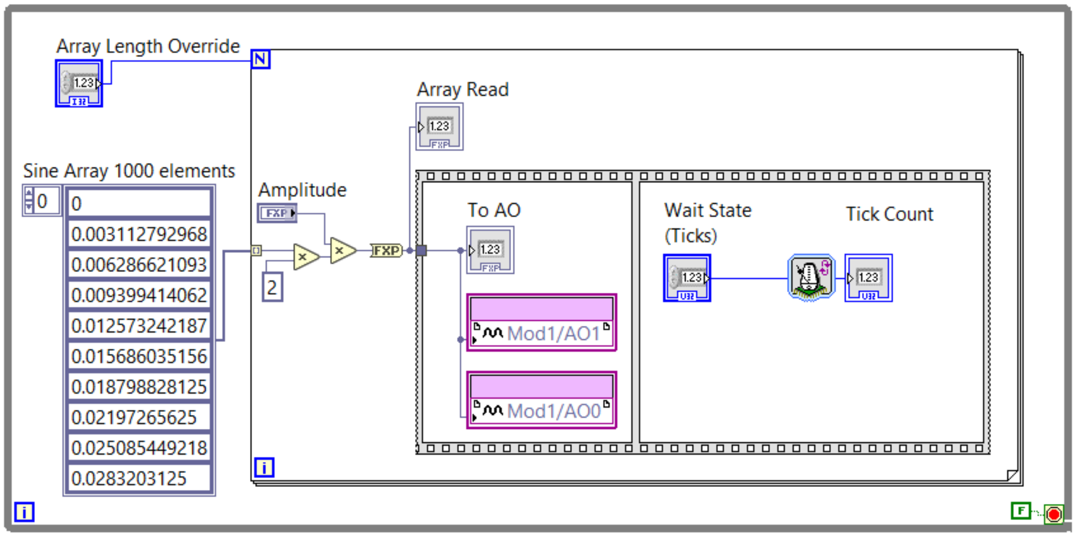
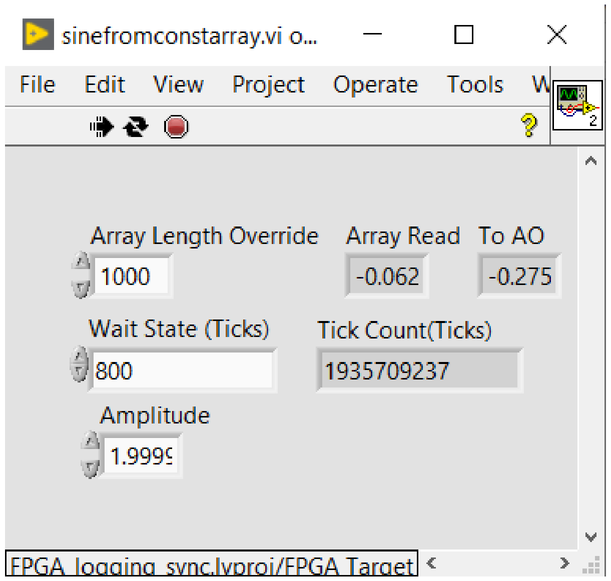


Figure 3.16: Block Diagram of the VI that generates waveforms from circular arrays

Figure 3.18 shows the results on an oscilloscope for the settings shown in Figure 3.17. The array length is 1000 and hence we see the full sine wave. The wait state is set to 800 ticks and this results in a frequency of 50.3 Hz. Changing the wait state to 670 ticks results in a waveform with a frequency of 59.9 Hz, as shown in Figure 3.19 (note: the frequency measurement on the oscilloscope is fairly coarse as compared to the PMU). Overriding the array length to 500 on the Front Panel results in a half wave as shown in Figure 3.20.


sinefromconstarray.vi o...
File
Edit
View
Project
Operate
Tools
W
FPGA logging sync.lvproj/FPGA Target

Array Length Override: 1000
Array Read: -0.062
To AO: -0.275
Wait State (Ticks): 800
Tick Count(Ticks): 1935709237
Amplitude: 1.9999

Figure 3.17: Front Panel of the VI that generates waveforms from circular arrays

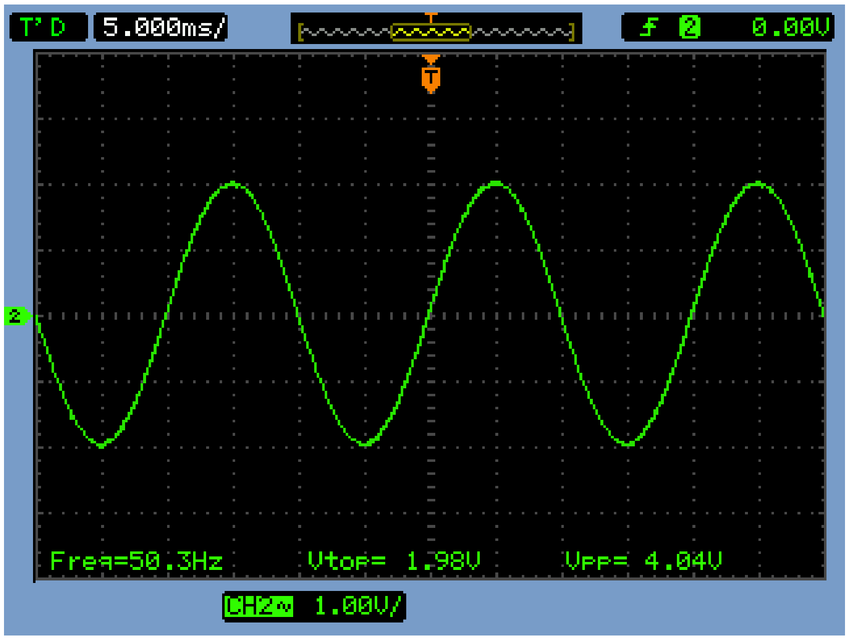


Figure 3.18: Sine wave of $\approx 50.3Hz$ generated with a 1000 point array and wait state of 800 ticks

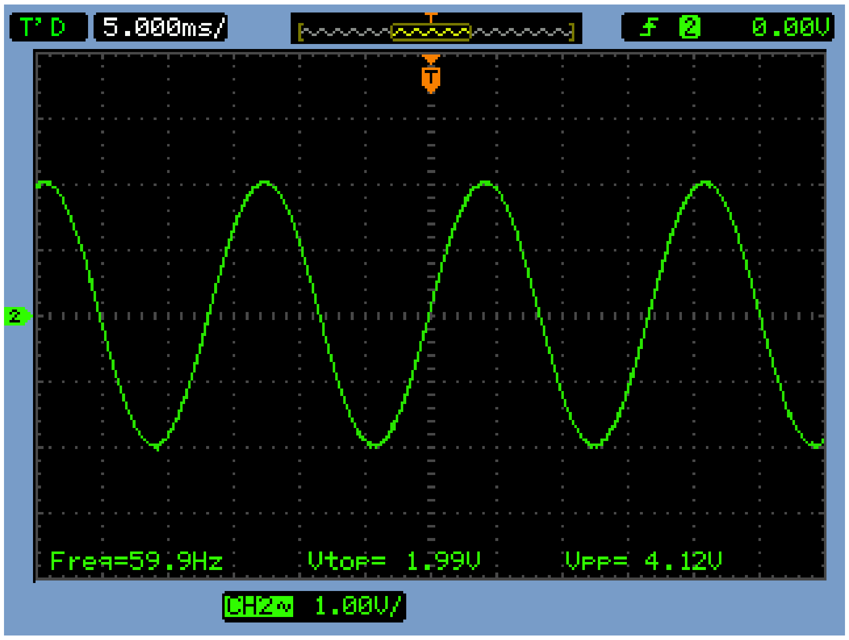


Figure 3.19: Sine wave of $\approx 59.9Hz$ generated with a 1000 point array and wait state of 670 ticks

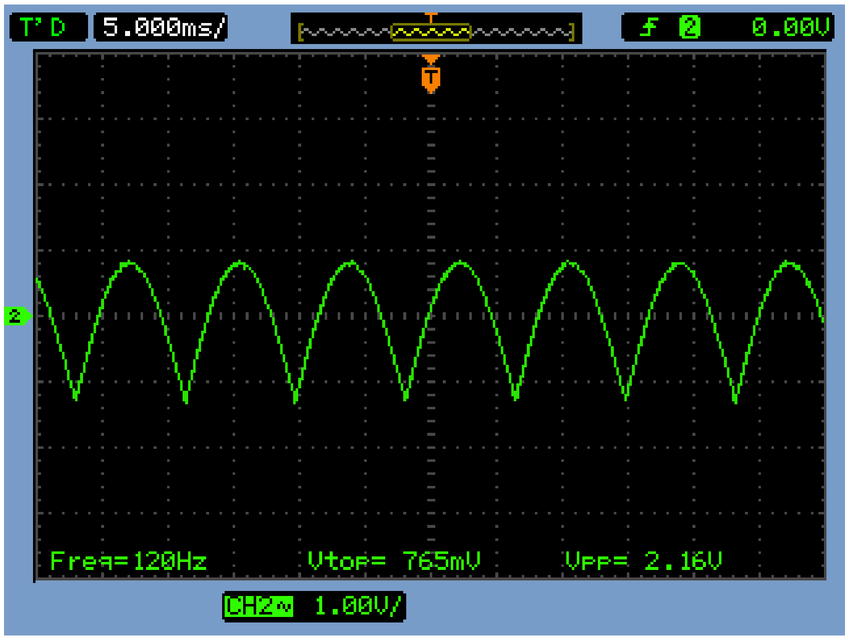


Figure 3.20: Half sine wave of $2 \times 59.9Hz \approx 120Hz$ generated with a $1000/2 = 500$ point array and wait state of 670 ticks

3.2.3 Measuring the frequency using the PMU

The oscilloscope used to make these previous measurements was Agilent DSO1014A which is rated at 100 MHz and 2GSa/s. However, the frequency it displayed kept on shifting every few seconds say between 60.0 and 60.2 etc. Closer observation of Figures 3.18 and 3.19 reveals that when the wait state is changed, without any change in the amplitude, the Vpp reading has moved from 4.04 V to 4.12 V. However, the Vtop remains at 1.99 in both cases. The oscilloscope was therefore deemed not accurate enough to measure the waveforms generated by the cRIO precisely. However, it was good enough to observe the expected trends that validated the proper functioning of the VIs running on the cRIO. Because of the limitations of the frequency measurement provided by the oscilloscope, it was decided to use the PMU to accurately measure the frequency of the sine wave generated by the cRIO (the IEEE Standard for Synchrophasor Measurements for Power Systems C37.118.1 requires PMUs to measure frequencies with an accuracy of better than 0.005 Hz, and published independent lab tests have shown much better steady state performances from PMUs of 0.0000345 Hz [17]).

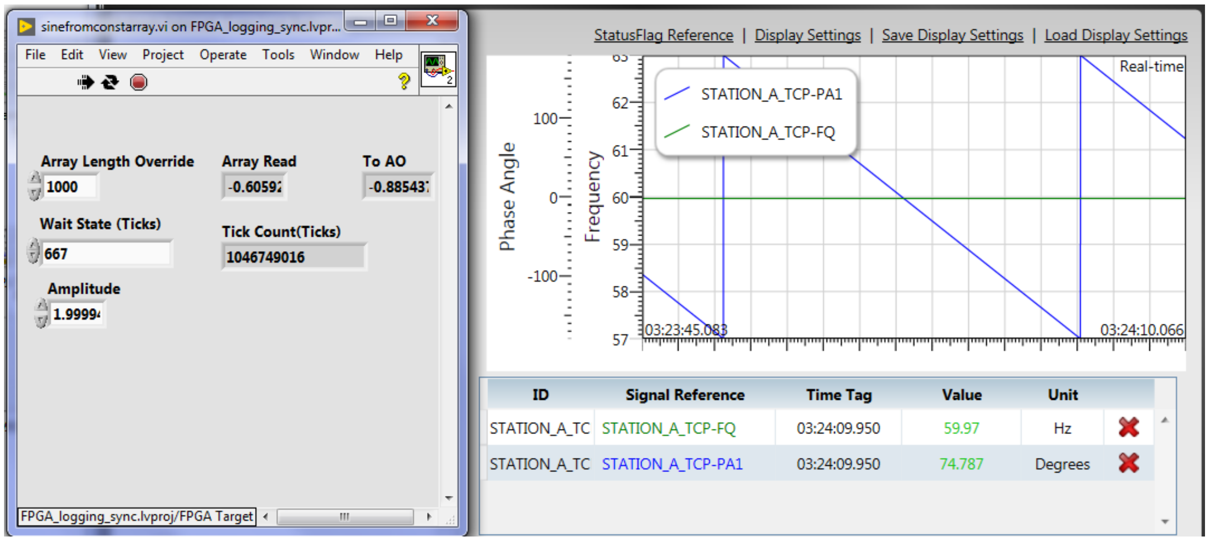
Figure 3.21 shows the PMU phase and frequency measurements for a sine wave constructed with an array of 1000 data points, with the wait state set to 667 ticks (FPGA clock cycles). The PMU records the frequency as 59.97 Hz. The green line on the graph shows the frequency and the blue line shows the phase. Since PMUs report phase with respect to a nominal frequency (60 Hz in this case), whenever the frequency deviates from 60 Hz, the phase of consecutive cycles keeps shifting. If a perfect 60 Hz wave were to be applied to the PMU, the blue line in Figure 3.21 would be flat.

PMUs report phase angles relative to a 60 Hz reference signal that is synchronized to UTC. When a PMU measures a signal, it compares it with the reference signal and reports the phase. If the frequency is exactly same as the nominal frequency, the phase difference between the measured signal and the reference signal would remain constant. However, if the frequency is constant but off from the nominal frequency, when the PMU measures the signal (at exact 60 Hz), the waveform will be measured at varying distances from the reference. Therefore, the phase reported begins to drift. [18]

Since the frequency is less than 60 Hz, we try to reduce the wait state by 1. Figure 3.22 captures the corresponding changes in the reported phase and frequency. The frequency measured by the PMU changes to 60.06 Hz, which corresponding to a constant, linear increase in the reported phase angle. Since neither of these wait states result in the desired signal frequency (60 Hz), it isn't possible to achieve a finer control on the sine wave generated by a 1000-element circular array on the cRIO.

It is in fact possible to deduce this analytically. The period of a 40 MHz clock is

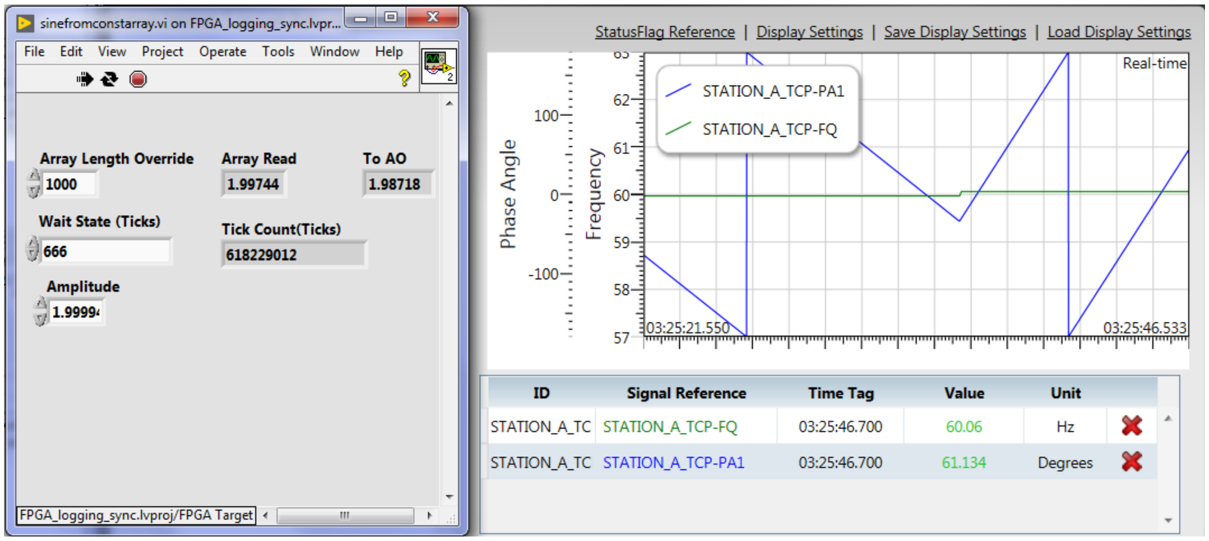
2.5×10^{-8} seconds. For the cRIO to output such a 1000 element circular array, with a 667 tick long wait state between each output, time required would be $1000 \times 2.5 \times 10^{-8} \times 667 = 0.016675$ seconds. This time period corresponds to a frequency of 59.97001499 Hz and the PMU reports 59.97 Hz. Similarly for a wait state of 666 ticks, the frequency, analytically should be $60.0\overline{600}$ Hz and the PMU correctly reports it to be 60.06 Hz.


sinefromconstarray.vi on FPGA_logging_sync.lvpr...
File Edit View Project Operate Tools Window Help
FPGA_logging_sync.lvproj/FPGA Target
StatusFlag Reference | Display Settings | Save Display Settings | Load Display Settings
Phase Angle
Frequency
100
0
-100
62
61
60
59
58
57
STATION_A_TCP-PA1
STATION_A_TCP-FQ
Real-time
03:23:45.083
03:24:10.066

Array Length Override: 1000
Array Read: -0.6059
To AO: -0.88543
Wait State (Ticks): 667
Tick Count(Ticks): 1046749016
Amplitude: 1.9999

ID	Signal Reference	Time Tag	Value	Unit	
STATION_A_TC	STATION_A_TCP-FQ	03:24:09.950	59.97	Hz	✗
STATION_A_TC	STATION_A_TCP-PA1	03:24:09.950	74.787	Degrees	✗

Figure 3.21: PMU measurements with 667 wait states results in a frequency of 59.97 Hz. Phase in blue keeps rolling down.


sinefromconstarray.vi on FPGA_logging_sync.lvpr...
File Edit View Project Operate Tools Window Help
FPGA_logging_sync.lvproj/FPGA Target
StatusFlag Reference | Display Settings | Save Display Settings | Load Display Settings
Phase Angle
Frequency
100
0
-100
62
61
60
59
58
57
STATION_A_TCP-PA1
STATION_A_TCP-FQ
Real-time
03:25:21.550
03:25:46.533

Array Length Override: 1000
Array Read: 1.99744
To AO: 1.98718
Wait State (Ticks): 666
Tick Count(Ticks): 618229012
Amplitude: 1.9999

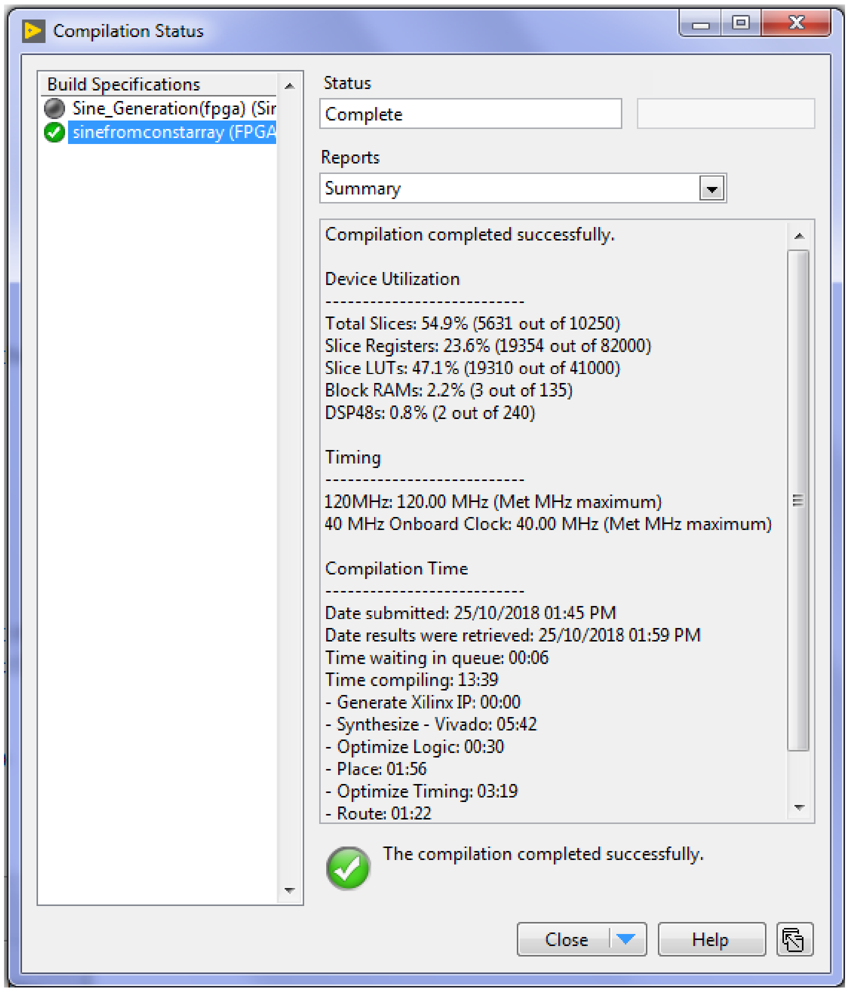
ID	Signal Reference	Time Tag	Value	Unit	
STATION_A_TC	STATION_A_TCP-FQ	03:25:46.700	60.06	Hz	✗
STATION_A_TC	STATION_A_TCP-PA1	03:25:46.700	61.134	Degrees	✗

Figure 3.22: PMU measurements when wait state is changed from 667 to 666. Frequency jumps up to 60.06 Hz and the phase starts climbing upwards.

3.2.4 Generating an accurate 60 Hz waveform

From Section 3.2.3, it is clear that generating an accurate 60 Hz waveform from a 1000 point sequence is not possible by varying the wait states of the FPGA. With a wait state of 667 ticks, the frequency was 59.97 Hz. The time period of this waveform is slightly longer than desired. Reducing the length of the array might help in achieving our desired waveform. However, even reducing the length by one, i.e. with an array of 999 data points, the analytically calculated frequency is 60.03, still not 60.00. This is not a viable solution, at least in the vicinity of the array length of 1000 data points. Therefore, generating an exact 60 Hz sine wave with an integer number of elements in the sequence is not possible/feasible. However, unlike wait states that need to be integers, the number of elements used to generate one sine wave can be, on average, fractional. This can happen when a large sequence that captures a waveform of multiple sine waves is used. For instance, if a sequence of 2001 points is used to generate 2 waves, then on average one wave is generated using 1000.5 data points. As the number of waves in the sequence, and the number of elements per wave are increased, a finer control over the frequency can be achieved. Figure 3.23 is the compilation report from the VI which stores 1000 elements in the FPGA's memory–note that 54.9 percent of the resources on the FPGA were consumed for this array length. For the purposes of experimentation, it was desired to create a 15-second output. If the entire waveform is stored on the FPGA, it would correspond to 900 waves, which is well beyond the local storage capabilities of the FPGA (since storing only one wave takes up 54.9 percent of the available resources). Fortunately, LabVIEW offers a solution to this problem in the form of dedicated FIFO buffers. The cRIO chassis has larger memories, and dedicated FIFOs can be set up from the chassis to the FPGA which transfer data at predictable data rates via DMA.

A FIFO (labelled FIFO 3) was set up to transfer data from the Host (processor controlled part of the cRIO chassis) to the Target (FPGA). The complete (900-period) sinusoidal signal array has a length of more than 30,000 elements. Each element in this array is a signed fixed point value with word length of 25 bits and integer word length of 5 bits, i.e., the FIFO is capable of transferring data with a higher precision than required by the cRIO's analog output module. During every read cycle on the FPGA, one element is transmitted from the FIFO to the FPGA. By controlling the number of elements used to generate the full (15-second) sequence, the frequency can be controlled to satisfaction. Figure 3.24 shows that the phase drift is much less pronounced with fine frequency control. Moreover, by using the FIFO buffer, arbitrary signals of relatively long lengths (e.g., COMTRADE output of a PSCAD simulation) can be directly passed to the FPGA with precise control of the timing between sample outputs. Note that the



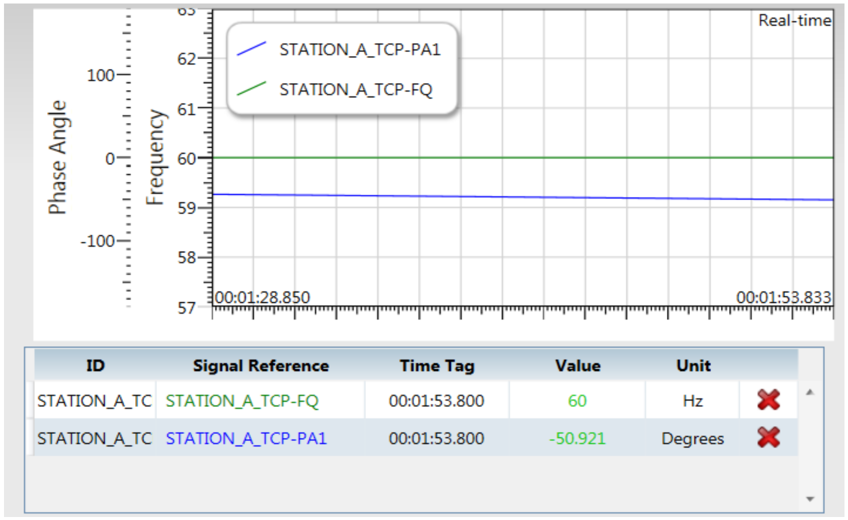
Complete

```
Compilation completed successfully.

Device Utilization
---------------------------
Total Slices: 54.9% (5631 out of 10250)
Slice Registers: 23.6% (19354 out of 82000)
Slice LUTs: 47.1% (19310 out of 41000)
Block RAMs: 2.2% (3 out of 135)
DSP48s: 0.8% (2 out of 240)

Timing
---------------------------
120MHz: 120.00 MHz (Met MHz maximum)
40 MHz Onboard Clock: 40.00 MHz (Met MHz maximum)

Compilation Time
---------------------------
Date submitted: 25/10/2018 01:45 PM
Date results were retrieved: 25/10/2018 01:59 PM
Time waiting in queue: 00:06
Time compiling: 13:39
- Generate Xilinx IP: 00:00
- Synthesize - Vivado: 05:42
- Optimize Logic: 00:30
- Place: 01:56
- Optimize Timing: 03:19
- Route: 01:22
```

The compilation completed successfully.

Figure 3.23: FPGA compilation report for the 1000 point array shows 54.9% device utilization

time scale here is same as that seen previously in Figures 3.21 and 3.22



ID	Signal Reference	Time Tag	Value	Unit	
STATION_A_TC	STATION_A_TCP-FQ	00:01:53.800	60	Hz	✖
STATION_A_TC	STATION_A_TCP-PA1	00:01:53.800	-50.921	Degrees	✖

Figure 3.24: Fine frequency control results in a steady 60Hz frequency and a slow phase drift. Note the time scale is same as Figures 3.21 and 3.22

Time synchronisation

The GPS module provides a timestamp value in terms of a count of nanoseconds after a known epoch. This time stamp can be compared with a predetermined value to trigger the start of execution of a conditional while loop in a LabVIEW VI. Similarly, the VI can be stopped after a particular predetermined time. Using such conditions, the 15 second period of interest from a PSCAD simulation can be emulated on the cRIO with precise start and stop times. Figure 3.25 shows the part of the VI deployed on the FPGA to achieve this.

In the upper part of the VI, we see two numeric inputs which provide the VI with the upper and lower bounds of the desired time period of operation entered in the front panel. The block labelled 'Get Time.vi' is the block that supplies the current GPS time stamp. This block is connected to two Boolean logic blocks (for the aforementioned comparison/or) that compare its value against the desired upper and lower bounds. The outputs of these blocks are in turn connected to a Boolean AND block. When both conditions are met, the output of this AND block keeps the while loop in operation. By adding an OR block in series, a provision for bypassing this timing condition from the front panel is also made available.

In the Flat Sequence Structure below, the first section takes the elements from the FIFO and supplies it to the AO modules. In the second section, the input labelled 'Count' determines how long the Wait block will hold program execution before the next element from the FIFO is passed on to the AO modules. The display labelled 'Tick count' reflects the action of the 'Wait' block on the front panel.

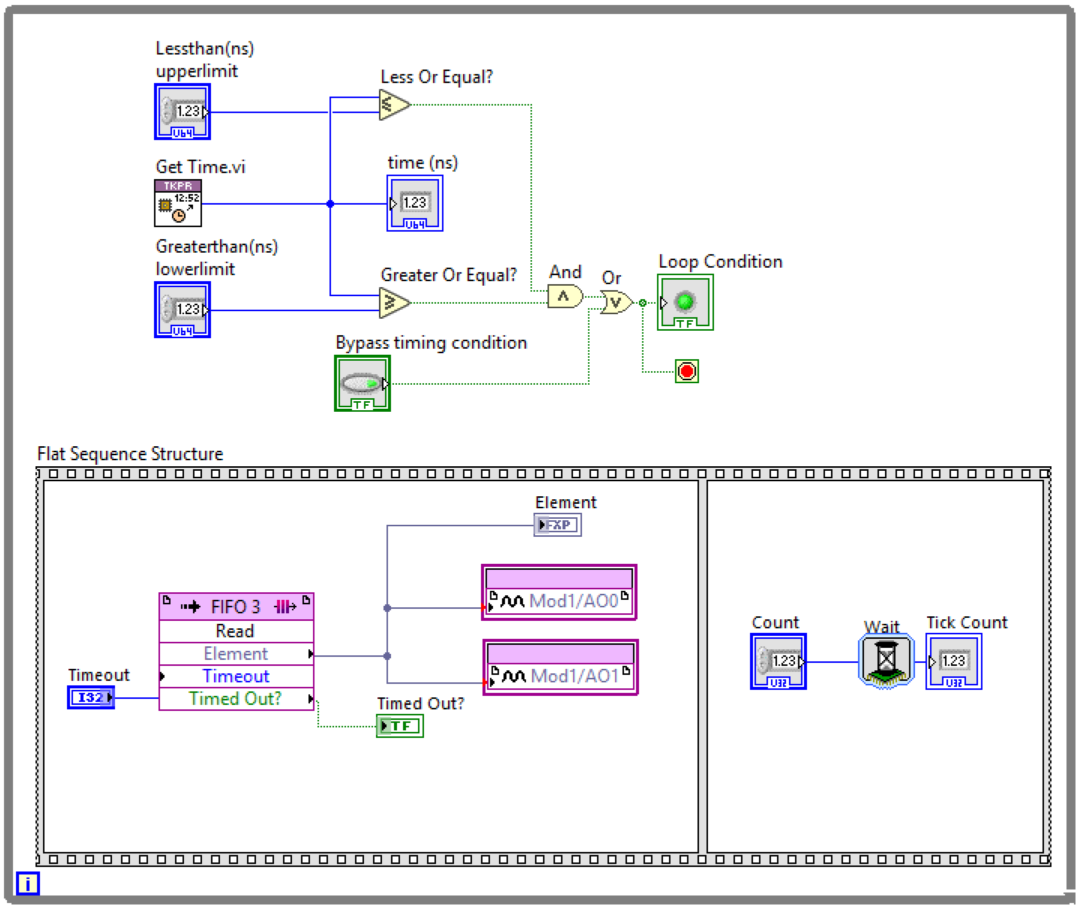


Figure 3.25: VI shows GPS triggered, timed implementation on the FPGA that utilises FIFO to transfer data from the cRIO processor

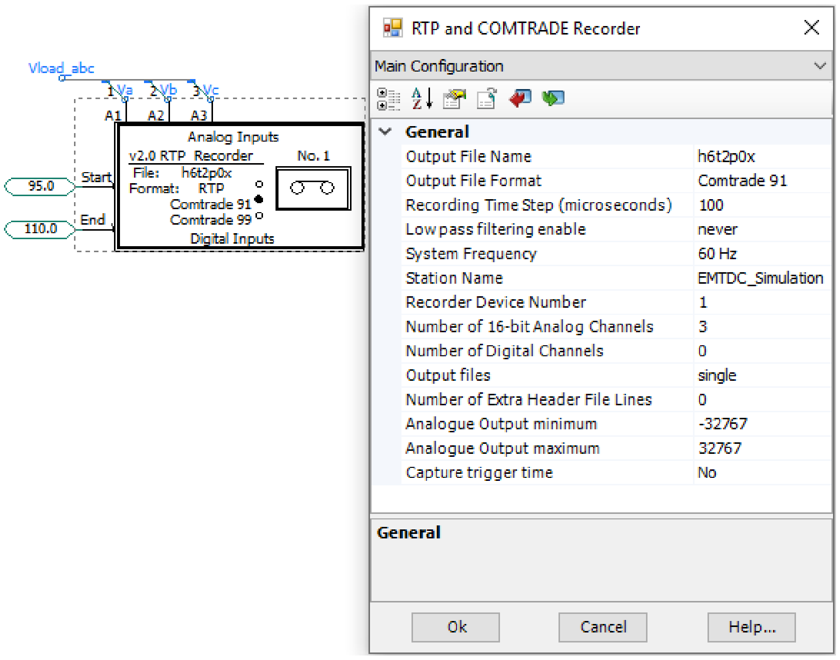
Chapter 4

Data Processing, Results, Analysis and Modelling

This chapter discusses the details of the steps involved in processing data at various stages of this research. Section 4.1 discusses how the data was extracted from PSCAD and processed to make it suitable for the cRIO. Section 4.2 explains how the readings from the PMU were acquired. Section 4.3 presents the results and Section 4.4 discusses the modelling of the PMU's voltage response.

4.1 Processing PSCAD data for the cRIO

As described in Section 2.2.1 and shown in Figure 4.1, the COMTRADE block is used in PSCAD to export the grid voltage waveform for the 15 second period of interest. This waveform is sampled at 100 μs. At this rate, each period, is constituted of about 167 samples (actually $166.\overline{6}$), which is sufficient to capture the dynamics of interest. While PSCAD can export this waveform at faster datarates, the sampling time needs to be an integer (in μs) and it cannot sample at datarates fast enough to be directly passed on to the cRIO for accurate reproduction. The .DAT file generated by the COMTRADE block for the 15 second period of interest contains 150,000 points. On the cRIO, after multiple trials, it was empirically determined that with the wait state = 1 tick, a circular array of 2574 elements produces a wave with a period that is closest to a frequency of 60 Hz. Therefore, for the cRIO to reproduce the PSCAD waveform faithfully/accurately, data for each individual sinusoidal wave needs to consist of 2574 points instead of the current $166.\overline{6}$. This necessitates resampling.



Output File Name	h6t2p0x
Output File Format	Comtrade 91
Recording Time Step (microseconds)	100
Low pass filtering enable	never
System Frequency	60 Hz
Station Name	EMTDC_Simulation
Recorder Device Number	1
Number of 16-bit Analog Channels	3
Number of Digital Channels	0
Output files	single
Number of Extra Header File Lines	0
Analogue Output minimum	-32767
Analogue Output maximum	32767
Capture trigger time	No

Figure 4.1: The COMTRADE Block in PSCAD used to export the time domain voltage signal for the 15 second period of interest

Interpolation of the COMTRADE signal

There are many mathematical methods and tools available for resampling data/signals. MATLAB was used for this purpose because of familiarity and its convenient vector and matrix manipulation capabilities. It also has easy to use system identification tools.

MATLAB has multiple functions for interpolation and resampling. To compare these, the following procedure was followed. Two sine waves, one consisting of 167 data points/samples per period and the other consisting of 2574 data points/samples per period were generated in MATLAB. From the 167 samples per period waveform, using various MATLAB interpolants, waveforms of 2574 samples per period were generated. These were compared with the ideal 2574 sample sine wave. The best method was chosen based on the mean absolute error.

The 'resample' function resamples data to a new fixed rate based on the ratio of two numerical inputs it accepts. It uses a polyphase anti-aliasing filter to resample data. However, when filtering, 'resample' assumes that the input sequence is zero before and after the samples it is given. Deviations from zero at the endpoints of the input signal result in unexpected values of the output signal. These edge effects are shown in Figure 4.2. Further, as seen in Figure 4.3, while interpolating near the peak of the sine wave, the 'resample' function creates small concave sections on the otherwise convex waveform. Therefore, this method too had to be rejected.

The ‘interp1’ and ‘spline’ (cubic spline) functions work even with non-uniformly sampled data. They accept the input signal vector along with its corresponding time vector. The functions then require a time vector corresponding to which the resampled data is calculated. By default, the ‘interp1’ function uses linear interpolation. While interpolating between two consecutive points, this can lead to sharp edges in the waveform (as can be seen in Figure 4.3). The ‘spline’ function uses cubic spline interpolation which creates piecewise cubic functions that are continuous and have continuous derivatives.

A review of the literature reveals that spline functions were developed specifically to fix some problems with polynomial interpolation [19]. What stands out most are their properties of continuity. For cubic spline functions, the functions themselves and their derivatives up to the second derivative are continuous [20]. The cubic spline functions are widely used in interpolation algorithms in medical imaging and chemical engineering[21].

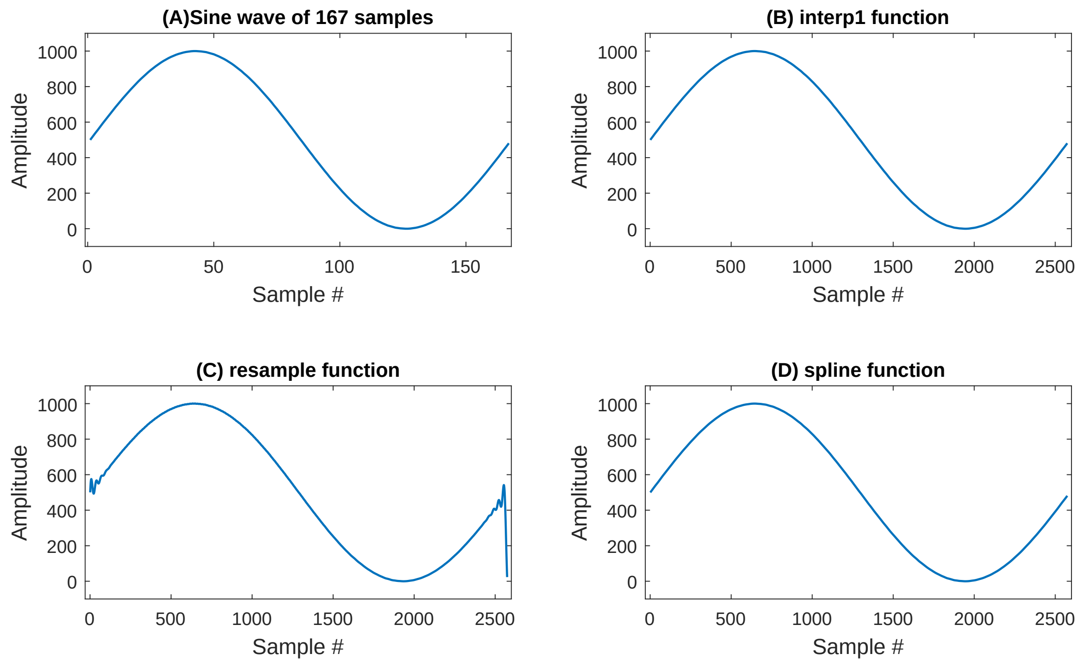


Figure 4.2: Comparison of the performance of various resampling functions in MATLAB used to interpolate a sine wave with 167 samples to 2574 samples

Figures 4.2 and 4.3 show the sine wave formed with 167 points alongside its 2574 point interpolated versions generated by using the ‘resample’, ‘interp1’ and ‘spline’ functions. Figure 4.2 displays the complete, full length signal sequence, here, all variants look more or less similar apart from the edge distortions brought about by the ‘resample’ function. Figure 4.3 shows a zoomed in view of the peaks of these waveforms. Here, the fine differences between each of the methods are prominently visible. Based on these graphs, cubic spline looks like the best method for resampling PSCAD waveforms for the cRIO.

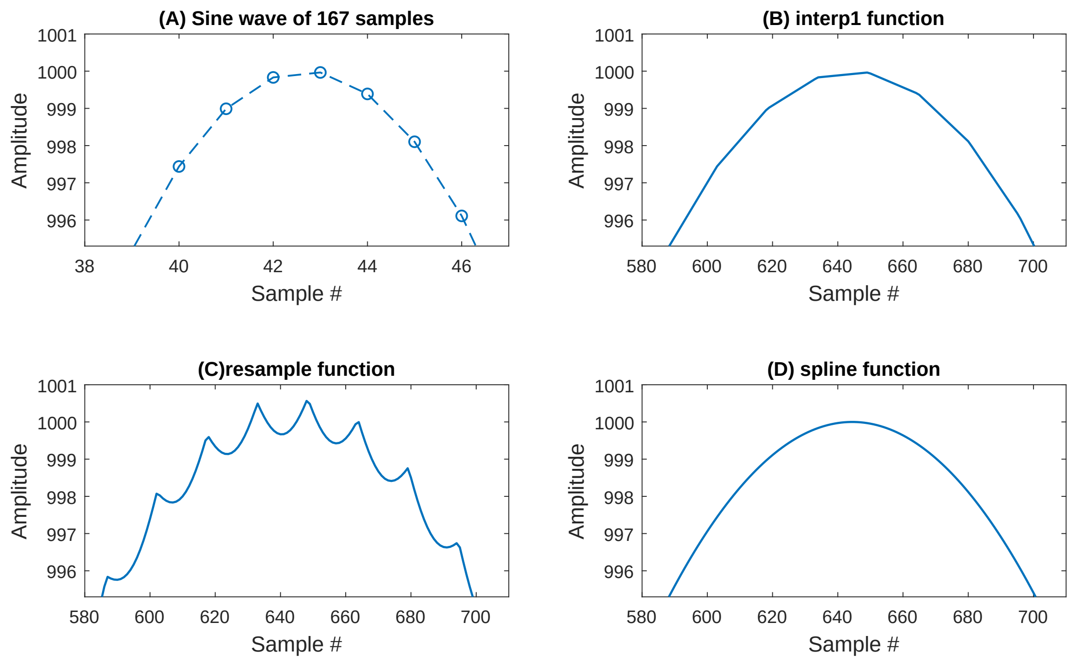


Figure 4.3: Comparison of the performance of various resampling functions in MATLAB used to interpolate a sine wave with 167 samples to 2574 samples. Zoomed in view of Figure 4.2 shows the performance near the peak of the wave.

These three resampled waveforms were compared against the ideal sine wave. The mean absolute errors, defined as

$$\sum_{i=1}^{2574} \frac{|IdealSignal[i] - InterpolatedSignal[i]|}{2574}$$

for the ‘resample’ function was 7.761, for the ‘interp1’ function was 0.038 and for the ‘spline’ function was 9.176×10^{-07}. Based on these mean absolute error values, it is clear that among the methods considered, cubic spline is the most suitable for resampling. Since voltages on power lines are ideally continuous sinusoidal signals this exercise with interpolation of sine waves is a good indicator and can be used to decide the best method for our application.

As discussed, the ‘spline’ (cubic spline) function accepts an input signal vector along with two time vectors, one each corresponding to the sampling of the available input and the desired output signal. The resampled signal contains the same number of samples as this second time vector. Since our signal in PSCAD is sampled at a uniform rate and the objective is to resample it as well at an uniform rate, the required time vectors can be created using the ‘linspace’ command in MATLAB.

The specification of 2574 samples per period for the cRIO was arrived at based on a circular array with data for one 60 Hz period only. As the simulation scenario consists of 900 such periods/cycles, there is scope for finer control over the frequency. Instead of 900×2574 data points in the final waveform, there can be $(900 \times 2574) \pm x$ data points. This x is named as the fine adjustment parameter in some of the MATLAB scripts (see Appendix A). Since this process of resampling had to be repeated for multiple PSCAD simulation scenarios, a MATLAB function, "PSCADtoCRIOfunction.m" was created. The function accepts an input file name (the COMTRADE file), an output file name and the fine adjustment parameter. The fine adjustment parameter was set to -14 for all test cases. Therefore, the waveform generated by the cRIO, representing the 15 second scenario consists of $(2574 \times 900) - 14 = 2316586$ data points.

4.2 Obtaining PMU data

The PMU communicates with the PC over the serial port. The openECA (Extensible Control and Analytics) Platform from the Grid Protection Alliance receives and displays the data from the PMU. The tool is connected to an SQL database. Every cycle, the PMU reports 27 unique quantities. These records can be exported to .csv files. Of these, only the voltage magnitude, phase and frequency of one particular channel are of interest for the experiment. A Python script was written to isolate the quantities of interest (See Appendix B).

While testing a particular case on the PMU, the 15 second period of interest was produced continuously on the cRIO in a cyclic manner. Therefore, the PMU recorded multiple instances of the experiment. The first instance was discarded as prior to its beginning the PMU reports invalid data. The second cycle was isolated and analyzed for each scenario.

4.3 Results

Of all the data collected, this research focuses on the voltage magnitude measurements. Figure 4.4 shows the voltage magnitude measured by the PMU as the exciter time constant is varied from 0.5 s to 2 seconds in steps on 0.5 s. It shows the changes in the dynamic response for each of these cases. As the exciter time constant is increased, the voltage magnitude dips further and takes longer to recover (as expected).

The PSS/E data for each of the cases is available for a period of 10 seconds with the event occurring at 0.5 seconds. Figure 4.5 shows the PSS/E results and the PMU

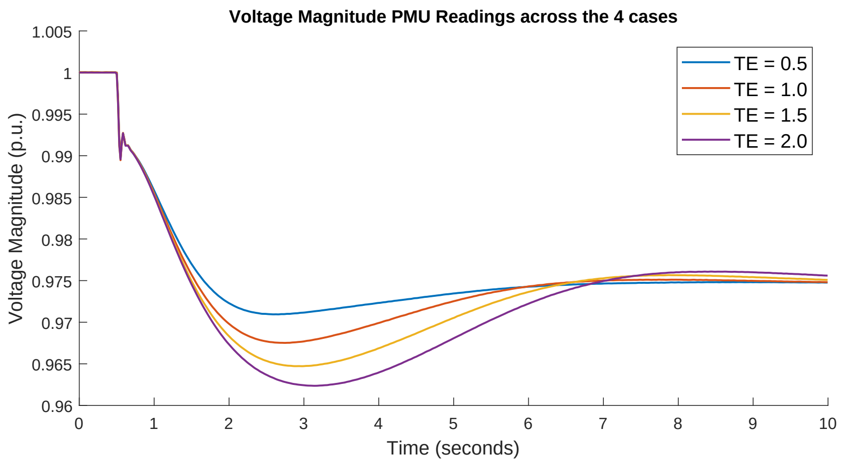


Figure 4.4: PMU readings: Voltage magnitudes of all four cases

readings for the case with exciter time constant 0.5s. The PSS/E simulation results do not drop as sharp as the the PMU readings after the event. The PMU readings rise up/ recover after the sharp drop and then fall again. This is captured in the zommed in view shown in Figure 4.6. The PMU readings are not in sync with the PSS/E results. Similar behaviour is seen in the other cases as well.

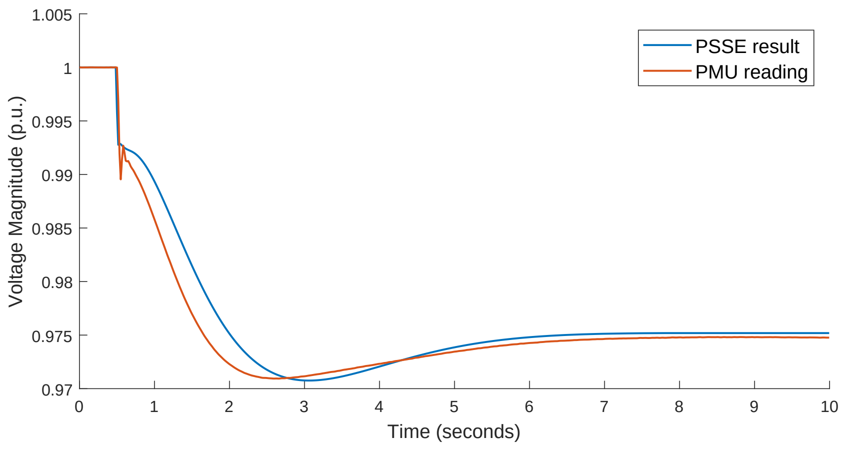


Figure 4.5: PMU readings and PSS/E Results for the voltage magnitudes, Case TE = 0.5 s

4.4 Modelling the PMU

Once the PMU and PSS/E results were obtained, the next step was to take the PSS/E results and process them to get a closer match with the PMU readings. Since both the PSS/E results and the PMU readings are available, a model of the PMU can be

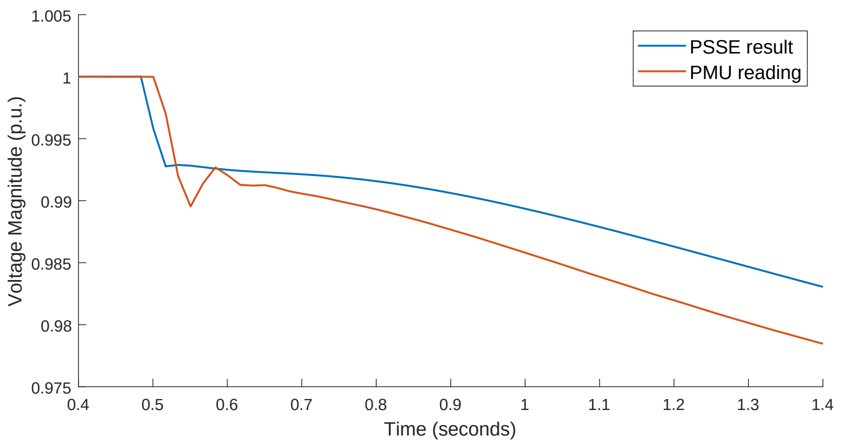


Figure 4.6: MU readings and PSS/E results for the voltage magnitudes, case TE = 0.5 s. Zoomed in view of Figure 4.5 shows dynamics just after the event.

generated. For this purpose, MATLAB's System Identification Toolbox was used.

The System Identification Toolbox provides MATLAB functions, Simulink blocks, and an app for constructing mathematical models of dynamic systems from measured input-output data. It facilitates creation of models of systems that are not easily modeled from first principles or specifications. Once both data sets are imported into the System Identification Toolbox, there are multiple options for estimating the system.

The System Identification Toolbox requires both the input and output signals to have been sampled at the same time. The PMU reports one reading per 60 Hz cycle; however, the PSS/E data available was at a faster rate. While the PMU data consists of 600 samples for the 10 second period, the PSSE data contains 9600 data points (for the same period). The PSS/E signal was therefore required to be downsampled. This was accomplished on MATLAB using the (cubic) 'spline' command. Figure 4.7 shows the original PSS/E results in blue and the PSS/E data downsampled at the timestep corresponding to the PMU readings in red crosses.

This exercise was carried out for the four cases. Graphs in Figure 4.8 plot this downsampled PSS/E data with the PMU readings. These were the input/output data pairs imported into the System Identification Toolbox. The PSS/E results and the PMU readings match before the load step/event, therefore those samples should not play a major role in estimating the system dynamics.

How much do PSS/E simulations deviate from the PMU readings?

For the four cases, Fit Percentages (normalized root mean squared error (NRMSE) expressed as a percentage) were computed. Fit Percentage for a given TE case is defined

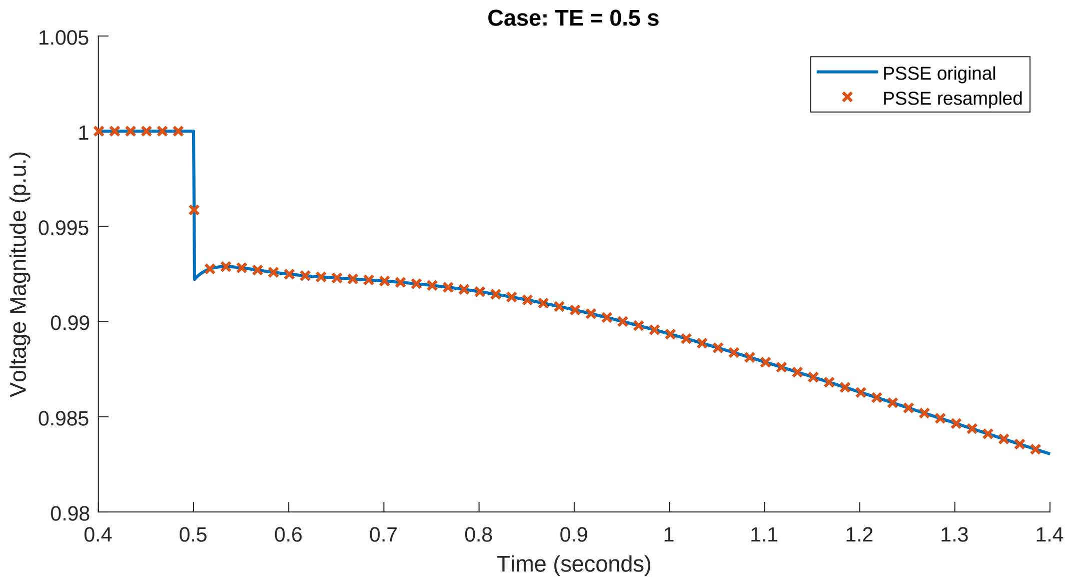


Figure 4.7: Down sampled PSS/E results plotted against the original data set show a good match

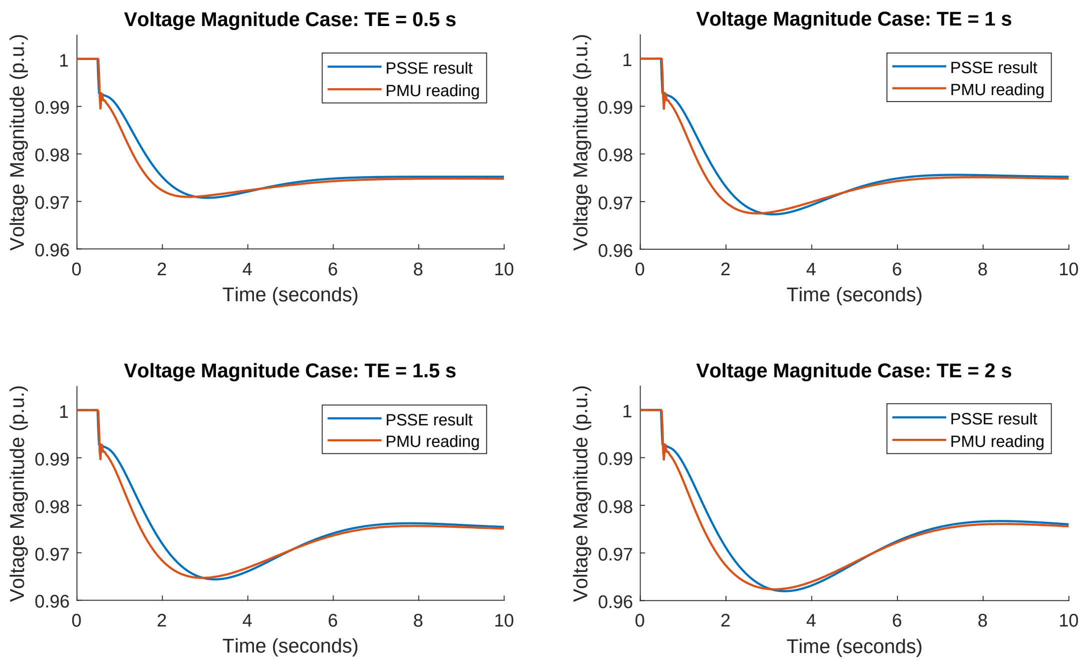


Figure 4.8: Comparing the PSS/E results and PMU readings for the four cases.

as:

$$FitPercent_{TE} = 100 \times \left(1 - \frac{\sqrt{\sum_{i=1}^{900}(VmPMU_{TE}[i] - VmPSS/E_{TE}[i])^2}}{\sqrt{\sum_{i=1}^{900}(VmPMU_{TE}[i] - \overline{VmPMU_{TE}})^2}}\right)$$

where $TE \in Cases = \{0.5, 1.0, 1.5, 2.0\}$, $VmPMU$ and $VmPSS/E$ are the voltage magnitude data for the 15 second scenario of interest and the overline represents mean. Since the PMU reports data once every cycle (60 Hz), for a 15 second scenario, there are $60 \times 15 = 900$ data-points. These are reported in Table 4.1. They vary from 63.77 % for $FitPercent_{0.5}$ to 77.27 % for $FitPercent_{2.0}$. This is a significant deviation.

Table 4.1: Deviation in PSS/E results from the PMU readings

Case	**Fit Percentage**
TE = 0.5s	63.77%
TE = 1.0s	69.02%
TE = 1.5s	73.78%
TE = 2.0s	77.27%

General layout of the System Identification Toolbox app

The left portion of Figure 4.9 shows the four input-output paired time domain datasets imported into the System Identification Toolbox. The case TE = 0.5 s has been chosen as the working data. All estimation operations for generating models are performed on this dataset. Options include using transfer functions, state space models, process models with integrators and delays, polynomial models like ARX and ARMAX, non linear ARX models and correlation models.

The right side of Figure 4.9 shows different models that have been estimated based on this dataset. At the bottom of the Figure, in the 'Validation Data' box, the case with TE = 1.5 s has been selected. This means that the models generated with the TE =0.5 case ('Working Data') above will be tested on the input-output pairs of the TE =1.5 case ('Validation Data').

The output in Figure 4.10 shows how well the input data from Validation Dataset (case TE = 1.5), when processed thorough the models shown in the right side of Figure 4.9 (based on the 'Working Data' or the case with TE = 0.5 s) matches with the output data (of the Validation Dataset). In this Figure, the System Identification Toolbox plots these model outputs (in colours corresponding to models in the previous Figure 4.9) and

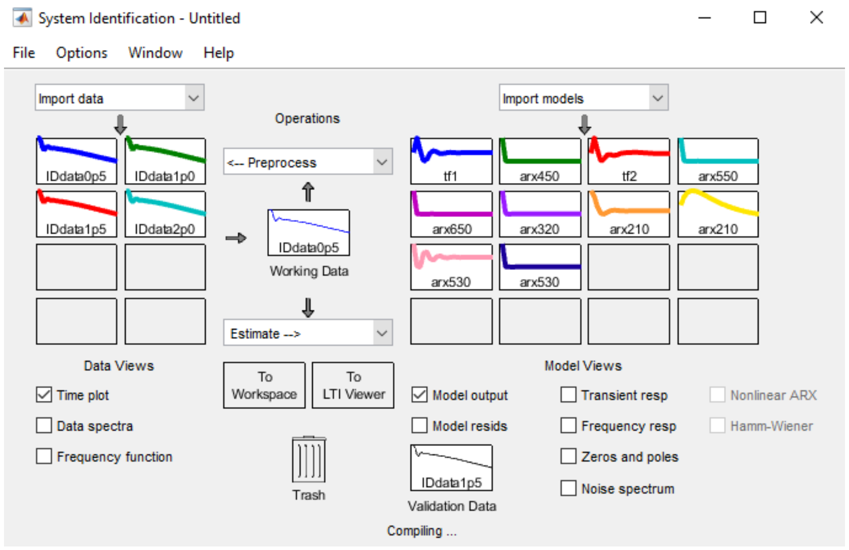


Figure 4.9: The System Identification Toolbox app

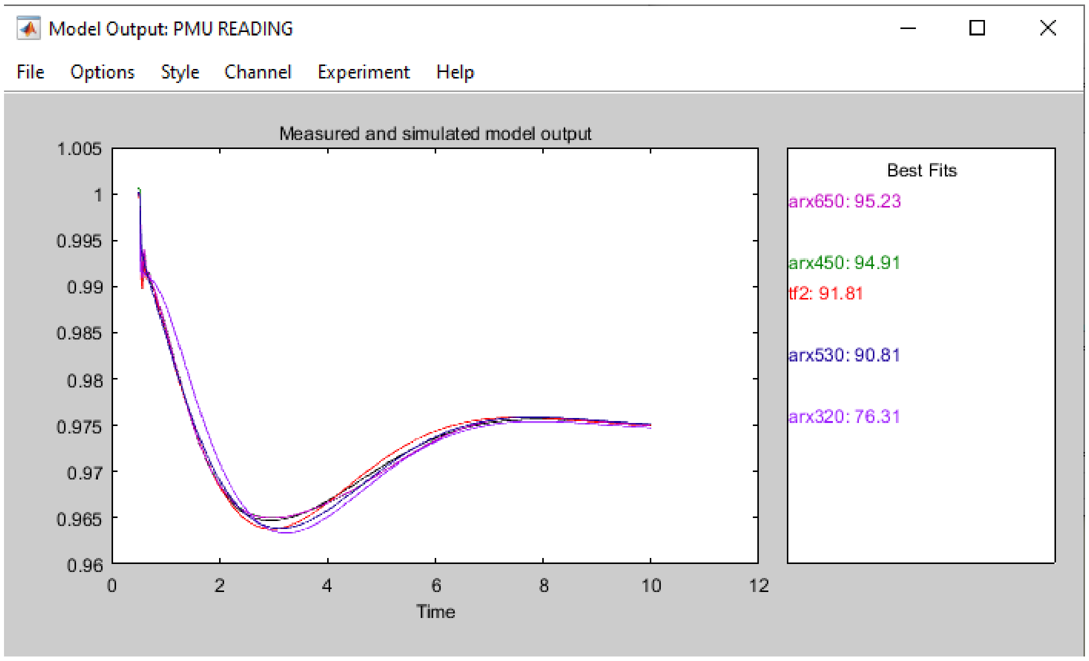


Figure 4.10: The System Identification Toolbox: Model outputs

the actual PMU reading (in black). The right side of the figure, i.e. the Best Fit legend lists the models in the order of their match percentages. Match percentages vary from 76.31% to 95.23%.

Therefore, once a model has been estimated, it can be validated against different sets of data. For instance a model can be created using the case with TE = 1 s and then input data from all other cases can be fed to the model and the accuracy of the model can be tested by comparing the observed outputs with the model's estimates. This exercise is important to detect overfitting.

The ARX model

In time series modeling, autoregressive exogenous (ARX) models are autoregressive (AR) models with exogenous inputs [22]. They are the simplest class of linear polynomial models that incorporate stimulus response [23].In AR models, the output and input are assumed to be correlated with one another. To identify the model, an assumption about how the variables are contemporaneously correlated with one another needs to be made. ARX models on the other hand use estimates of a given system of correlated variables and also exogenous variables. They allow outside shocks/factors to be taken into consideration [22].

The general structure of the ARX model is :

$$y[i] + a_1 y[i-1] + ... + a_{n_a} y[i-n_a] = b_1 u[i-n_k] + ... + b_{n_b} u[i-n_b-n_k+1] + e[i]$$

The parameters na and nb are the orders of the ARX model, and nk is the delay [24]. Symbols in the equations denote the following:
$y[i]$ - The current output.
n_a - Number of poles or the order of the polynomial $A(q)$.
n_b - Number of zeroes plus 1 or the order of the polynomial $B(q)+1$.
n_k - Number of input samples that occur before the input affects the output, also called the dead time in the system.
$y[i-1]...y[i-n_a]$ - Previous outputs on which the current output depends.
$u[i-n_k]...u[i-n_k-n_b+1]$ - Previous and delayed inputs on which the current output depends.
$e[i]$ - White-noise disturbance value [24].

A more compact way to write the difference equation is:

$$A[q]y[i] = B[q]u[i-nk] + e[i]$$

where q is the delay operator. And therefore,
$A[q] = 1 + a_1 q^{-1} + ... + a_{n_a} q^{n_a}$
$B[q] = b_1 + b_2 q^1 + ... + b_{n_b} q^{n_b+1}$

The complexity of the ARX modelling is determined by model orders n_a, n_b and n_k. The key problem is to determine the most suitable model complexity that can capture the dynamics of the system in general and be biased [25].

ARX in MATLAB

The 'arx' command in MATLAB estimates the parameters of ARX model using least squares. The input arguments for the command are the estimation data (input output time series pair represented as an iddata object), the polynomial orders of the model (n_a, n_b and n_k) and other estimation options (discussed in Section 4.4. The command returns an ARX model that fits the estimation data, in the form of a discrete-time idpoly object (polynomial model with identifiable parameters). Then the 'compare' command compares the response of the system SYS (created by the ARX command) to any validation data (the iddata objects created with the input output pairs of the 4 cases). It plots the graphs of the PMU output and compares it with the results obtained by processing the original PSS/E results with the polynomial generated by the ARX function. It also reports a fit percentage. Note that the dead time of the system, n_k was set to 0 for all models in this research.

Figure 4.11 below shows the raw PSS/E results in red (TSA simulation), the PMU readings in blue (ground truth) and the result of an ARX fit in green (model). For this model, data from the case TE = 0.5 s was used and the order of the coefficients of the ARX model were $n_a = 5$, $n_b = 4$, and $n_k = 0$. The fit percentage (normalized root mean squared error expressed as a percentage) is reported as 96.84%. For the discrete-time ARX model: $A[q]y[i] = B[q]u[i] + e[i]$, the following were the tuned model parameters.

$$\begin{aligned} y[i] = &-2.338y[i-1] + 2.192y[i-2] - 1.348y[i-3] + 0.492y[i-4] + \\ &1.004u[i] - 2.348u[i-1] + 2.197u[i-2] - 1.344z[i-3] + 0.4915u[i-4] \end{aligned}$$

Figure 4.12 shows a zoomed in view of the same data. Here points are individually plotted rather than joined by lines. It can be observed that the model output very closely matches the PMU readings point by point. What can also be seen is how distant the raw PSS/E results were from the PMU readings. This validates the model for this particular/specific case.

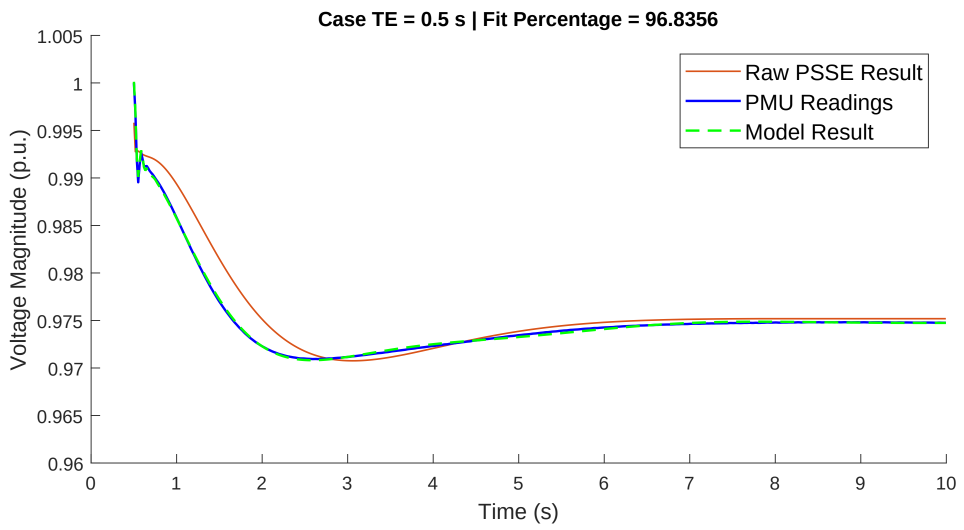


Figure 4.11: Comparison of the model output to PMU readings and raw PSS/E Results TE = 0.5 s for an ARX model with $n_a = 5$ and $n_b = 4$

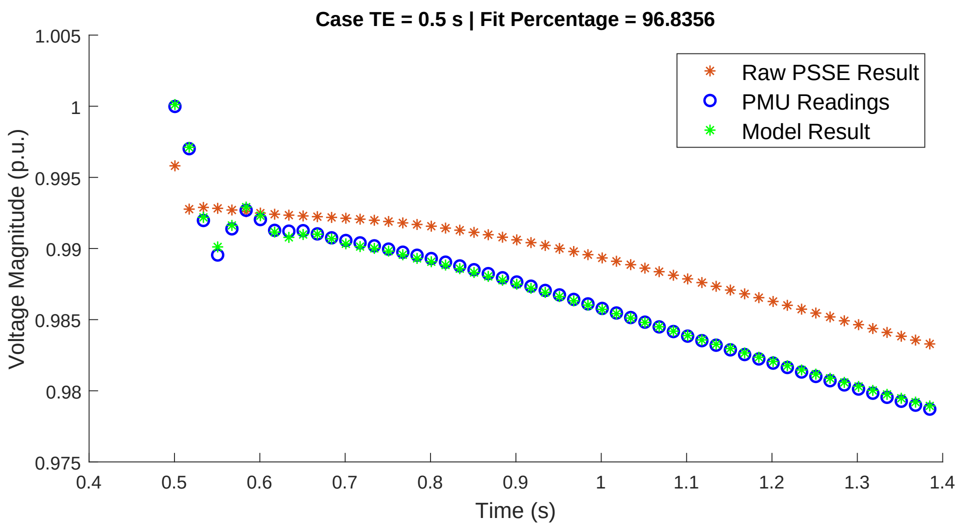


Figure 4.12: Zoomed in view of Figure 4.11. Comparison of the model output to PMU readings and raw PSS/E Results for the case TE = 0.5 s using an ARX model with $n_a = 5$ and $n_b = 4$.

Figure 4.13 shows the results when all 4 cases were validated with a model trained with data from the case with TE = 1.0 s. The ARX parameters used were $n_a = 4$ $n_b = 4$ and $n_k = 0$. With the lower order than the previous case, the percentage fit is lower as well. by analysing the average match/fit percentage and their spread within these 4 cases, an optimal order can be deduced.

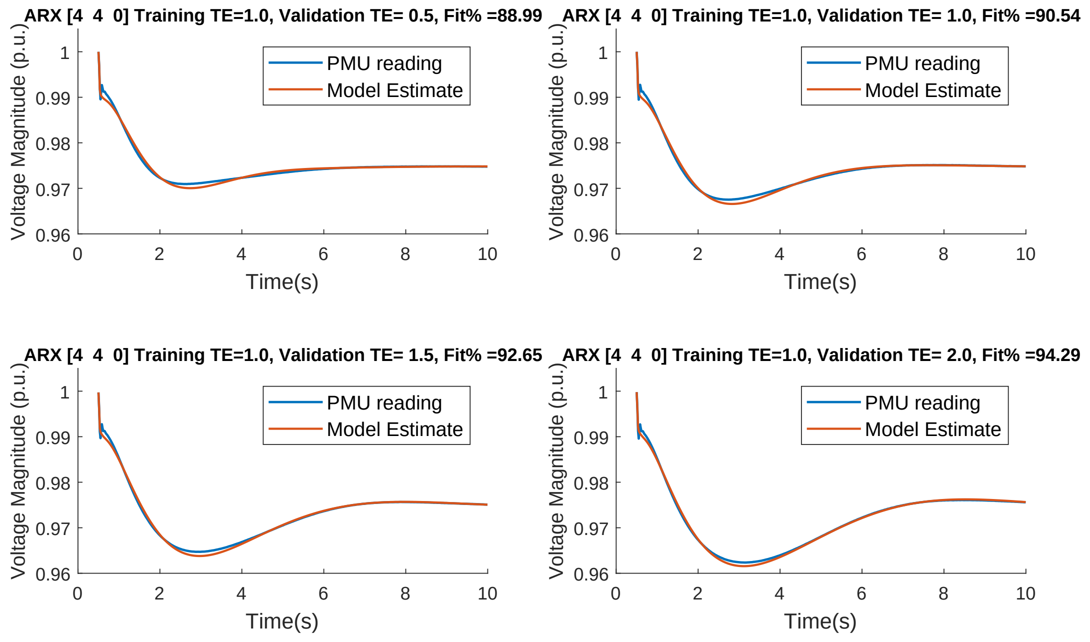


Figure 4.13: ARX model with $n_a = 4$ and $n_b = 4$ trained with data from the case TE = 1.0 s and validated with data from all four cases

ARX options

The 'arxOptions' command is used to create an 'option set' for the 'arx' command that specifies estimation options. The argument 'Focus' affects the formulation of the objective function. It specifies the error term to be minimized in the loss function during estimation.

When 'Focus' is set to the 'prediction' option, command computes the model response at some specified amount of time in the future. This is the default option and is used when the 'arx' command is used for analyzing and predicting traditional time series quantities like stocks etc. The one-step ahead prediction error between measured and predicted outputs is minimized during estimation.

When 'Focus' is set to the 'simulation' option, the simulation error between measured and simulated outputs is minimized during estimation. 'Simulation' computes the model response using input data and initial conditions only. This is unlike the 'prediction' mode where current and past values of the input data as well as the measured output are used.

This command also has options for adding input and output offsets in forms of coma separated values or column vectors. The estimation focuses on generating a good fit for simulation of model response with the current input. For this research, the 'arx' command was used in the 'simulation' mode.

4.4.1 Choosing the ARX model order

Determining the right order for the ARX as well as which of the four datasets should be chosen for the model is not obvious. Therefore a large number of tests were run for different values of n_a and n_b. For each of the n_a- n_b combinations, ARX models were fit on each of the four datasets. Each of these models was validated against the four datasets and their fit percentage was recorded. Each model yielded 4 such fit percentages, one each corresponding to every dataset (one of these four is from the same data that was used to create the model).

Any given fit percentage is characterised by the data-set used to create the model denoted by $TEM \in Cases = \{0.5, 1.0, 1.5, 2.0\}$, the orders of the model denoted by n_a and n_b and finally data-set with which the model was tested to calculate the fit, denoted by $TEV \in Cases = \{0.5, 1.0, 1.5, 2.0\}$. Therefore, a specific fit could be denoted by the following notation, $FitPercent^{TEV}_{TEM,na,nb}$

A good model is one which would produce a high percentage fit. However, producing a good fit for only the dataset from which the model is derived is not sufficient. A good model should not only capture the dynamics of the PMU but also be general enough to not adversely impact the fit when data from other tests cases is tested with the model. Therefore, a model may be considered good if the mean of the four Fit Percentages it produces is high. For the model data from case TEM, and a model with orders n_a and n_b, the Mean Fit Percentage is define as:

$$MeanFitPercent_{TEM,n_a,n_b} = \frac{\sum_{TEV \in Cases} FitPercent^{TEV}_{TEM,n_a,n_b}}{4}$$

This metric is analyzed in Figure 4.14.

The colour white represents a 100% fit and all matches 84% and below are shown in black. Brighter the colour, higher is the Fit Percentage. One of the first things that becomes evident is that increasing n_a and n_b does not always lead to better results. For

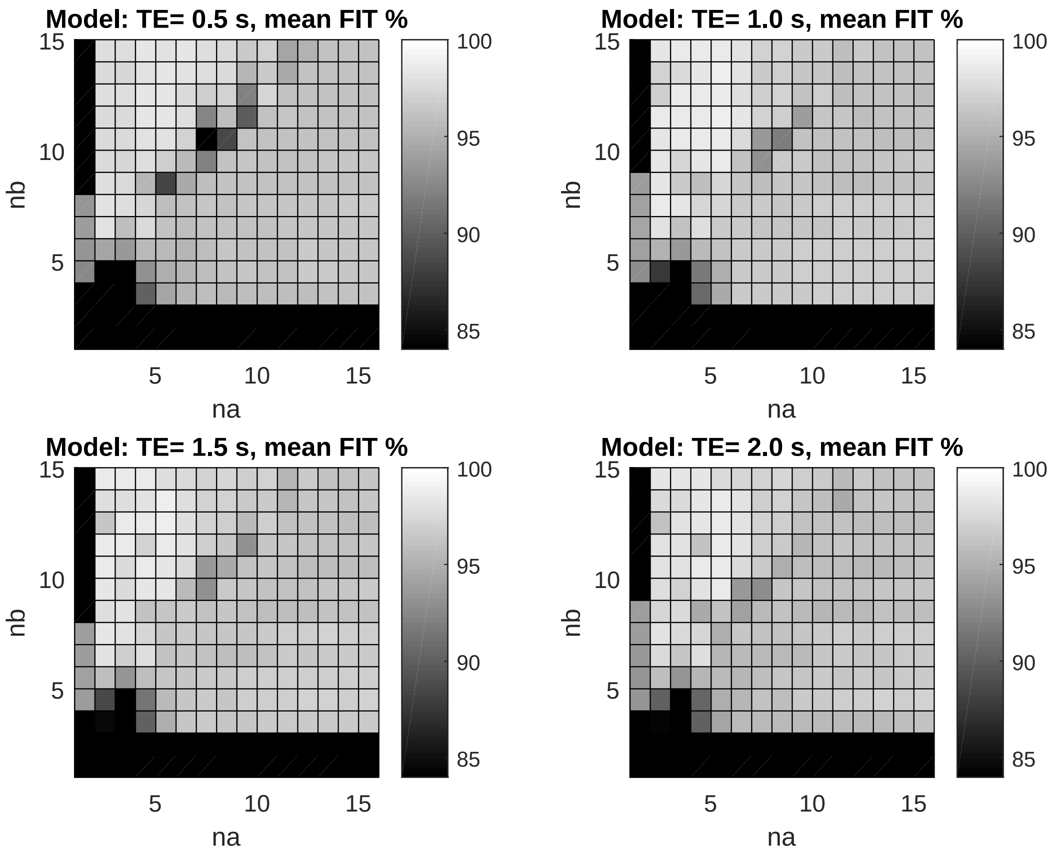


Figure 4.14: Mean Fit Percentages for models generated with the four datasets by varying n_a and n_b.

instance, consider the case of the model trained using data from the case TE= 0.5 s, shown on the top right corner of Figure 4.14. $MeanFitPercent_{0.5,2,6}$ (i.e. the mean fit percentage for the model prepared from data of the case TE= 0.5, and with polynomial orders $n_a = 2$ and n_b =6) is better than $MeanFitPercent_{0.5,14,14}$ a model created with higher order polynomials. Similar trends can be seen in the other models as well.

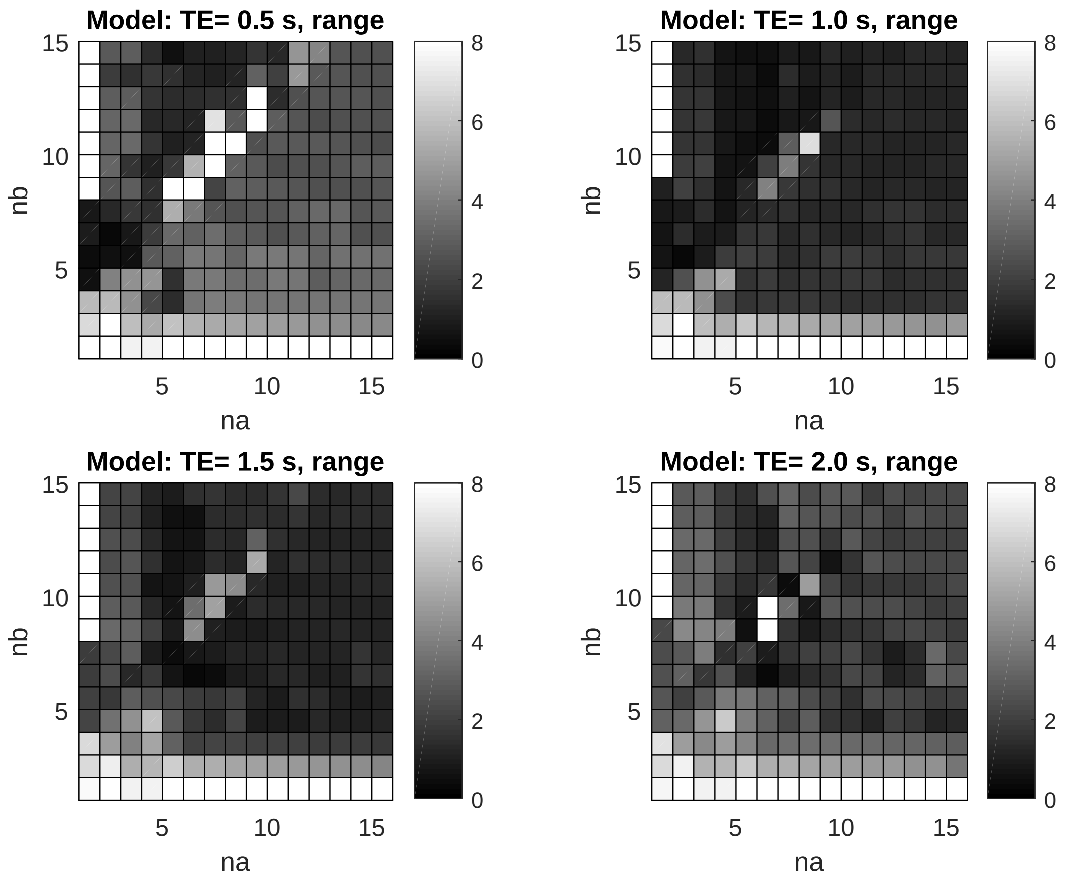


Figure 4.15: Fit Percentage Range for models generated with the four datasets by varying n_a and n_b.

Regardless of the match percent magnitude, it may also be desirable for a model to have similar percentage fits across the cases. This would indicate less bias towards a particular dataset and hence give confidence against the possibility of overfitting. This can be tested by comparing the Range, defined as the difference between the largest and the smallest percentage fit reported by a given model when applied to the four cases. Similarly, for a given dataset for the case TE = TEM used to create the model, and the polynomial orders of the model n_a and n_b, Range is denoted as:

$$Range_{TEM,n_a,n_b} = \max_{TEV \in Cases}(FitPercent^{TEV}_{TEM,n_a,n_b}) - \min_{TEV \in Cases}(FitPercent^{TEV}_{TEM,n_a,n_b})$$

The larger this range, the more biased the model is likely to be. This too was tested and documented in Figure 4.15. A smaller range (indicated by a darker gray) is more desirable. Again it is clear that arbitrarily increasing orders n_a and n_b do not always lead to more desirable outcomes. If that were the case, the top right corners of all the sub plots which represent both high values of n_a and n_b should have been the darkest.

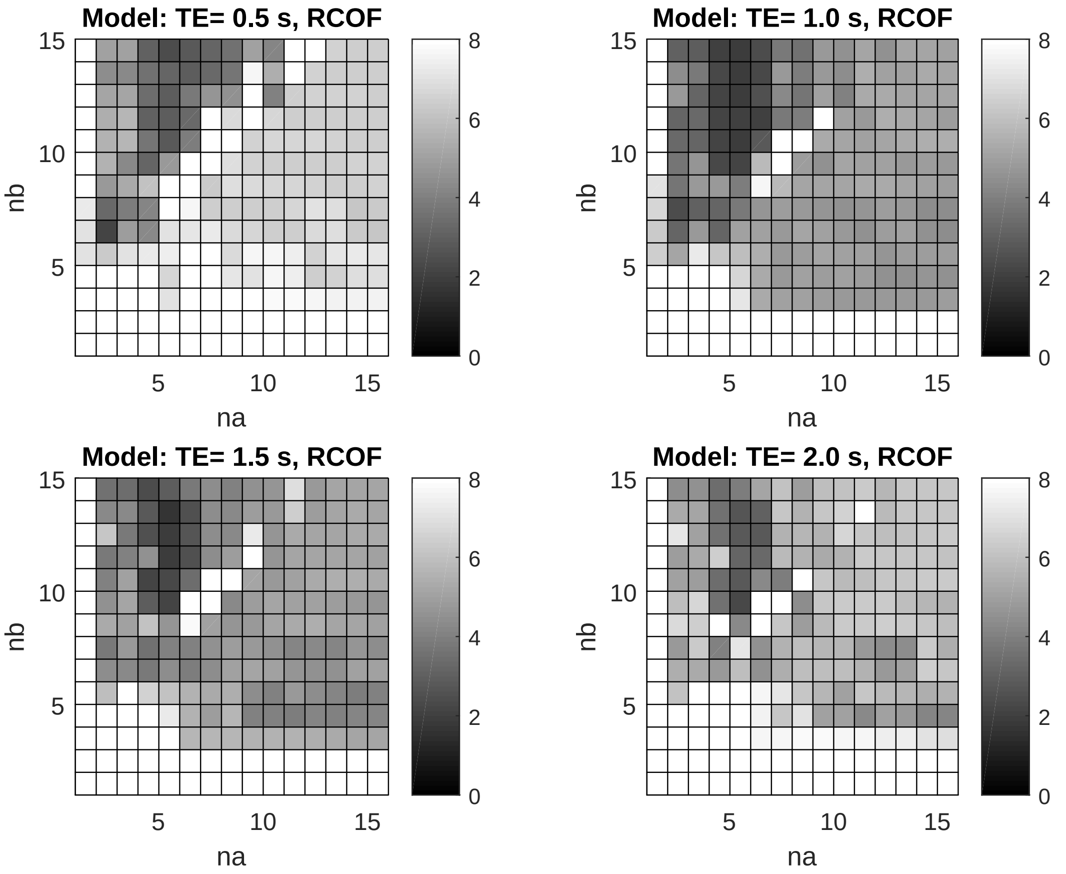


Figure 4.16: RCOF for models generated with the four datasets by varying n_a and n_b.

Fit percentages are always ≤ 100. The closer they are to 100, the better. For Range on the other hand, smaller is better. By using a combination of the two criteria discussed above, a third metric may be defined. Range Compensated Optimal Fit is defined as the sum of Range and the difference between 100 and the percentage fit. With such a definition, smaller RCOF is better.

$$RCOF_{TEM,n_a,n_b} = Range_{TEM,n_a,n_b} + (100 - MeanFitPercent_{TEM,n_a,n_b})$$

Therefore, in Figure 4.16, a darker shade is more desirable. By analyzing the models derived from the case TE = 0.5 s, shown on the top right corner, n_a=2 and n_b=6 i.e. $RCOF_{0.5,2,6}$ looks like a good choice for a model. From its shade of gray, $RCOF_{0.5,2,6} \leq 3$. That means, across the four cases, it should have greater than 97% match and the range too should be less than 3. Figure 4.17 shows the results. The lowest fit percentage is 97.98% while the highest is 98.25%. The range is 0.27. Figure 4.18 shows the zoomed in view of the half second just after the event. A good match is observed through most of the samples with only 3 to 4 points not exactly matching up, yet they are very close.

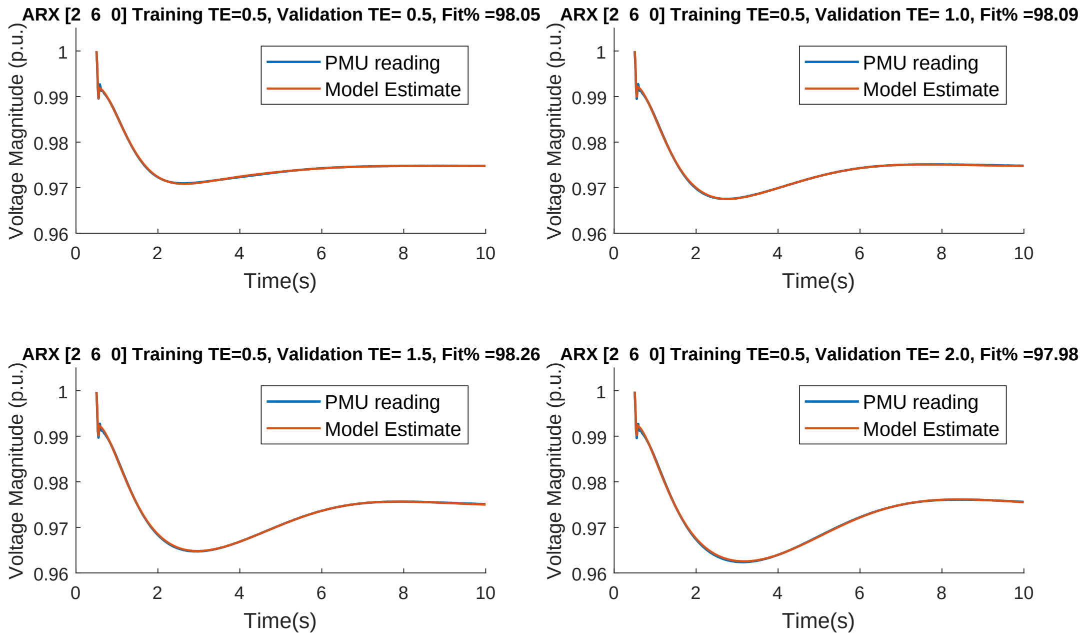


Figure 4.17: Results produced by a model based on the dataset with TE = 0.5 s, n_a=2 and n_b=6. A match% of over 97.98% is seen for all of the four datasets.

Thus, it is shown that the proposed model estimation framework accurately reproduces the PMU behaviour from a given set of PSCAD and PSS/E simulations.

4.4.2 Akaike's information criterion (AIC)

As discussed in Section 4.4.1, determining the order of a model is not trivial.

In a sequence of papers, starting in 1973, Akaike laid down the foundations of modern statistical model identification. He developed the entropy based Akaike's information criterion (AIC) for the identification of an optimal model from a class of competing

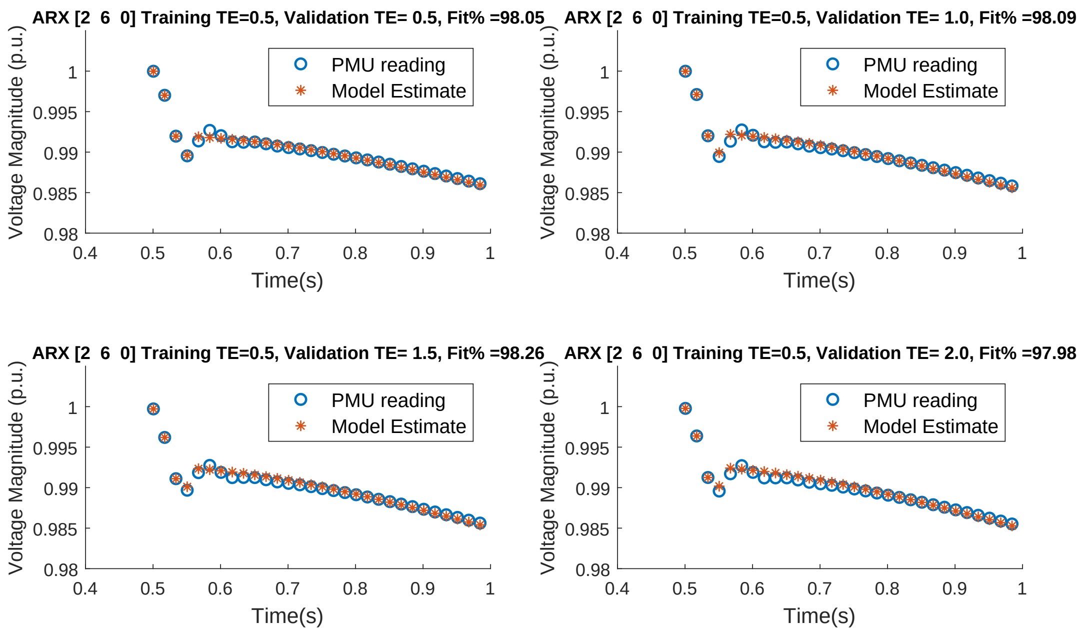


Figure 4.18: Zoomed in view of Figure 4.17 shows matching of data points during the half second after the initial transient. Results Produced by the Model based on the dataset with TE = 0.5 s, n_a=2 and n_b=6. A match% of over 97.98% is seen for all of the four datasets.

models [26]. This criterion accounts for the model complexity and adds penalties for higher orders. It is a trade-off between the accuracy of the fit and the simplicity of the model. Therefore, it addresses both underfiting and overfitting. It is defined as:

$$AIC(k) = 2k - 2ln(\hat{L})$$

where k is the number of estimated parameters for a model, in this case, $k = n_a + n_b + 1$ and $\hat{L}$ is the maximum likelihood function of the model.

While this criterion is useful in determining model orders, it was not used in this research as the data-set was fairly limited and the model was chosen based on the criterion described in the previous Section 4.4.1 by traversing through all of the individual cases.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this research, a test bench has been created, through which the dynamic performance of PMUs can be studied. This test bench accepts the results from EMTP programs using the standard COMTRADE format and emulates a time-synchronized low voltage grid for PMUs.

By utilizing features of the cRIO, like the FIFO buffer, for transferring large waveform sequences and the accurate analog output module, it was possible to create a very accurate representation of the voltage waveform into the PMU. Multiple such scenarios were emulated and the response of the PMU was recorded.

MATLAB's System Identification Toolbox was used to create ARX models for the voltage magnitude response of the PMU based on this data. Performance of these models was studied by varying the model parameters (orders of the polynomials). An optimal model was chosen based on a combination of metrics like high match percentages and low variation in these match percentages across the four test cases. The optimal model had match percentages of over 97% across all cases and a range of less than 0.5. This significantly outperforms a naive simulation that directly uses the PSS/E results (which has fit percentages in the range of 63-77%).

This research has successfully demonstrated a process that can be used to characterize the dynamic performance of PMUs.

5.2 Future work

This research used limited test scenarios and focused on the deviations in voltage magnitudes alone. One of the reasons that limited the number of cases was the difficulty in getting simulations on different EMTP and TSA software to closely match. In the future, more such test scenarios can be modelled and tested. Also, the load change modelled in the event was purely resistive. Reactive elements could be included in future research.

Others in the research group (Zhen Dai) have worked on the deviations in the frequency reported by PMUs. A model of the PMU that caters to the deviations across the parameters of magnitude, frequency and phase can also be created to provide a full virtual PMU.

In the course of this research, it was assumed that the analog source was completely accurate. Since this is unlikely to be the case, more effort could be put into to calibrating all the analog sources and measurement equipment used in this research.

The current set up relies on two separate GPS receivers in the PMU and the cRIO. While running the experiments, the antennas needed to be placed outside through windows. In the future, an IRIG B signal could be digitally synthesized which would not only lead to more accurate experiments but would also make the experiment more convenient to execute.

Finally, the impact of these model improvements in evaluating wide-area control systems needs to be studied.

Bibliography

[1] "451 front visual motion.png (JPEG Image, 1440660 pixels)." [Online]. Available: https://prodcdn.selinc.com/uploadedImages/Web/Products/Visuals/451%20front%20visual%20motion.png?n=63667465321000&preset=pattern-carousel&bp=lg

[2] "2401-with-9524a.png (JPEG Image, 380272 pixels)." [Online]. Available: https://prodcdn.selinc.com/uploadedImages/Web/Products/Popular/2XXX/2401-with-9524A.png?n=63623381516000&preset=size-col-4&bp=lg

[3] "NI LabVIEW for CompactRIO Developers Guide." [Online]. Available: http://www.ni.com/pdf/products/us/fullcriodevguide.pdf

[4] "cRIO9035photo." [Online]. Available: http://www.ni.com/en-ca/support/model.crio-9035.html

[5] J. D. L. Ree, V. Centeno, J. S. Thorp, and A. G. Phadke, "Synchronized Phasor Measurement Applications in Power Systems," *IEEE Transactions on Smart Grid*, vol. 1, no. 1, pp. 20–27, Jun. 2010.

[6] G. C. Patil and A. G. Thosar, "Application of synchrophasor measurements using PMU for modern power systems monitoring and control," in *2017 International Conference on Computation of Power, Energy Information and Commuincation (ICCPEIC)*, Mar. 2017, pp. 754–760.

[7] J. E. Tate and T. J. Overbye, "Line Outage Detection Using Phasor Angle Measurements," *IEEE Transactions on Power Systems*, vol. 23, no. 4, pp. 1644–1652, Nov. 2008.

[8] B. Gou, "Optimal Placement of PMUs by Integer Linear Programming," *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 1525–1526, Aug. 2008.

[9] J. S. Bhonsle and A. S. Junghare, "An optimal PMU-PDC placement technique in wide area measurement system," in *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, May 2015, pp. 401–405.

[10] S. Chakrabarti and E. Kyriakides, "Optimal Placement of Phasor Measurement Units for Power System Observability," *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 1433–1440, Aug. 2008.

[11] R. Franco, C. Sena, G. N. Taranto, and A. Giusto, "Using synchrophasors for controlled islanding - A prospective application for the Uruguayan power system," *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 2016–2024, May 2013.

[12] S. K, S. N. Singh, and S. C. Srivastava, "A Synchrophasor Assisted Frequency and Voltage Stability Based Load Shedding Scheme for Self-Healing of Power System," *IEEE Transactions on Smart Grid*, vol. 2, no. 2, pp. 221–230, Jun. 2011.

[13] "SEL-451-5 Data Sheet." [Online]. Available: https://cdn.selinc.com/assets/Literature/Product%20Literature/Data%20Sheets/451-5_DS_20190613.pdf?v=20190729-134306

[14] "Overview of IRIG-B Time Code Standard," TELECOMMUNICATIONS AND TIMING GROUP, TECHNICAL NOTE TN-102, May 2016. [Online]. Available: https://www.itsamerica.com/assets/publications/TN-102_IRIG-B.pdf

[15] "IEEE/IEC Measuring relays and protection equipment Part 24: Common format for transient data exchange (COMTRADE) for power systems - Redline," *IEEE Std C37.111-2013 (IEC 60255-24 Edition 2.0 2013-04) - Redline*, pp. 1–136, Apr. 2013.

[16] "NI cRIO-9035 User Manual - National Instruments," p. 40.

[17] T. Becejac and P. Dehghanian, "PMU Multilevel End-to-End Testing to Assess Synchrophasor Measurements During Faults," *IEEE Power and Energy Technology Systems Journal*, vol. 6, no. 1, pp. 71–80, Mar. 2019.

[18] "Phase Angle Calculations: Considerations and Use Cases," NASPI Engineering Analysis Task Team, Technical Paper, Sep. 2016. [Online]. Available: https://www.naspi.org/sites/default/files/reference_documents/naspi_2016_tr_006_phase_angle_calculations_final.pdf

[19] P. Thevenaz, T. Blu, and M. Unser, "Interpolation revisited [medical images application]," *IEEE Transactions on Medical Imaging*, vol. 19, no. 7, pp. 739–758, Jul. 2000.

[20] M. Unser, "Splines: a perfect fit for signal and image processing," *IEEE Signal Processing Magazine*, vol. 16, no. 6, pp. 22–38, Nov. 1999.

[21] L. Matiu-Iovan, F. M. Frigura-Iliasa, and S. Rancov, "A cubic B-spline interpolation algorithm that uses the first derivative values of the input function in the knots," in *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, Jul. 2013, pp. 709–712.

[22] "Autoregressive Exogenous Model Analytical Way." [Online]. Available: http://www.analyticalway.com/?p=62

[23] "ARX Model Definitions (System Identification Toolkit) - LabVIEW 2013 System Identification Toolkit Help - National Instruments." [Online]. Available: http://zone.ni.com/reference/en-XX/help/372458D-01/lvsysidconcepts/modeldefinitionsarx/

[24] "Estimate parameters of ARX or AR model using least squares - MATLAB arx." [Online]. Available: https://www.mathworks.com/help/ident/ref/arx.html

[25] Zulkeflee A.A., Sata S.A., and Aziz N., "Auto-regressive with exogenous input model predictive controller for water activity in esterification," *Chemical Engineering Transactions*, vol. 56, pp. 217–222, Apr. 2017. [Online]. Available: http://doi.org/10.3303/CET1756037

[26] H. Bozdogan, "Model Selection and Akaike's Information Criterion (AIC): The General Theory and Its Analytical Extensions," *Psychometrika; Colorado Springs, Colo.*, vol. 52, no. 3, p. 345, Sep. 1987. [Online]. Available: http://search.proquest.com/docview/1304598765?pq-origsite=summon

Appendices

Appendix A

MATLAB scripts

The following script was used to process multiple PSCAD COMTRADE output files to cRIO input data. This script merely calls the 'PSCADtoCRIOfunction' function which is given further below.

```
close all;
clear;
clc;
inputpath = '..\PSCAD\TwoBus_ideal_s_varyH.gf46\Rank_00001\Run_00001\';
inputfilelist = ['h0.dat' 'h1.dat' 'h2.dat' 'h3.dat' 'h4.dat' ...
    'h5.dat'...
    'h6.dat' 'h7.dat' 'h8.dat' 'h9.dat' 'h10.dat' 'h11.dat' 'h12.dat'...
    'h6TE0p5.dat' 'h6TE1p0.dat' 'h6TE1p5.dat' 'h6TE2p0.dat'];

outputpath = '..\PSCAD\';
outputfilelist = ['h0.csv' 'h1.csv' 'h2.csv' 'h3.csv' 'h4.csv' ...
    'h5.csv'...
    'h6.csv' 'h7.csv' 'h8.csv' 'h9.csv' 'h10.csv' 'h11.csv' ...
        'h12.csv' ...
    'h6TE0p5.csv' 'h6TE1p0.csv' 'h6TE1p5.csv' 'h6TE2p0.csv'];

frq_fine_adj = -14;

for i=1:size(inputfilelist)
    PSCADtoCRIOfunction( strcat( [inputpath inputfilelist(i)] ) ,...
        strcat( [outputpath outputfilelist(i)] ),frq_fine_adj );
end
```

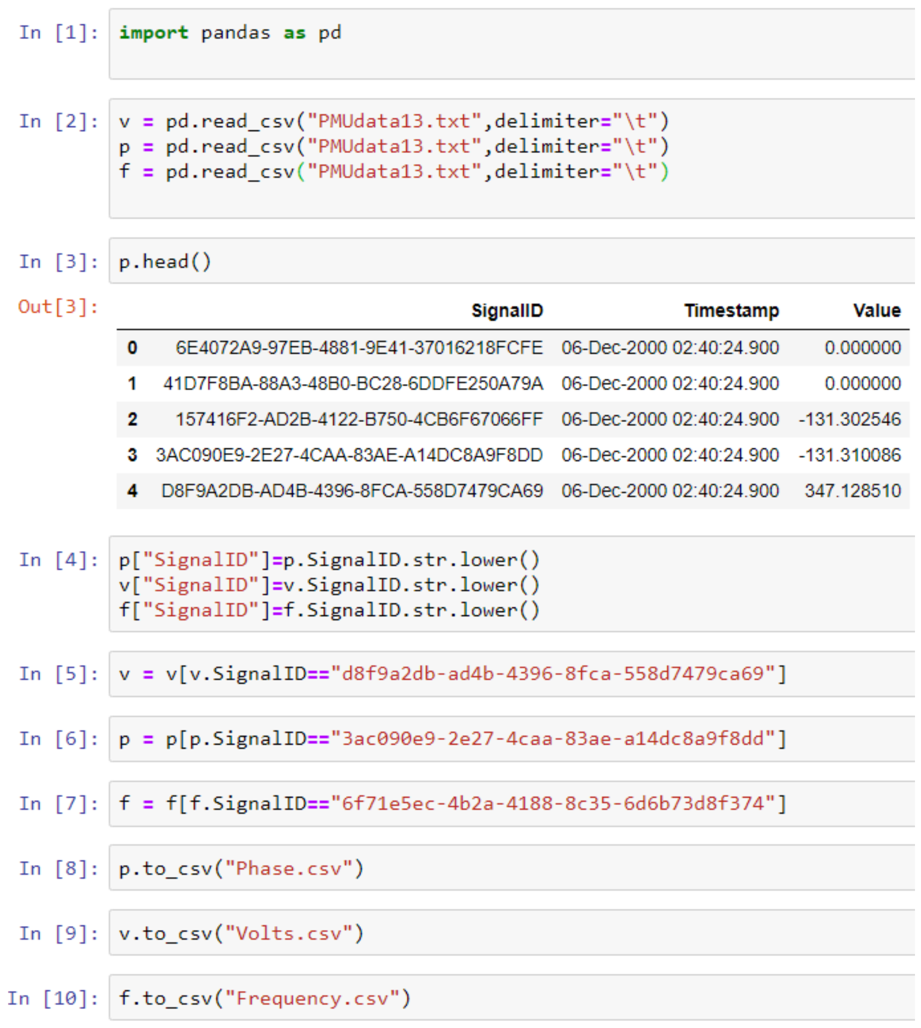
The function 'PSCADtoCRIOfunction' that resamples the PSCAD data for the PMU is given below.

```matlab
function [] = PSCADtoCRIOfunction(in_file,out_file,frq_fine_adj)


%% Importing Data
pscad_data = csvread(in_file);
pscad_data = pscad_data(:,3); %Isolating the relevant column from ...
    the COMTRADE file

%% Conversion
nominal_frequency = 60; %in Hz
time = 15; % in seconds
number_of_waves = time*nominal_frequency;
crio_pscad_factor = 2574; %Emperically Obtained

y_pscad_wave = pscad_data;

length_pscad_wave = length(y_pscad_wave);
length_crio_wave = (crio_pscad_factor*number_of_waves) + frq_fine_adj;

x_pscad = linspace(1, length_pscad_wave, length_pscad_wave);
x_crio = linspace(1, length_pscad_wave,(length_crio_wave ));


y_crio_wave = spline (x_pscad, y_pscad_wave, x_crio);


%% Exporting Output
csvwrite(out_file,y_crio_wave')

end
```

Appendix B

Python Script

The PMU reported a streams of 27 parameters for each time stamp. For this research, only the voltage magnitude, phase and the frequency for the one channel of interest needed to be isolated. This python script was used for that purpose.



```
In [1]: import pandas as pd
```

```
In [2]: v = pd.read_csv("PMUdata13.txt",delimiter="\t")
        p = pd.read_csv("PMUdata13.txt",delimiter="\t")
        f = pd.read_csv("PMUdata13.txt",delimiter="\t")
```

```
In [3]: p.head()
```

Out[3]:

	SignalID	Timestamp	Value
0	6E4072A9-97EB-4881-9E41-37016218FCFE	06-Dec-2000 02:40:24.900	0.000000
1	41D7F8BA-88A3-48B0-BC28-6DDFE250A79A	06-Dec-2000 02:40:24.900	0.000000
2	157416F2-AD2B-4122-B750-4CB6F67066FF	06-Dec-2000 02:40:24.900	-131.302546
3	3AC090E9-2E27-4CAA-83AE-A14DC8A9F8DD	06-Dec-2000 02:40:24.900	-131.310086
4	D8F9A2DB-AD4B-4396-8FCA-558D7479CA69	06-Dec-2000 02:40:24.900	347.128510

```
In [4]: p["SignalID"]=p.SignalID.str.lower()
        v["SignalID"]=v.SignalID.str.lower()
        f["SignalID"]=f.SignalID.str.lower()
```

```
In [5]: v = v[v.SignalID=="d8f9a2db-ad4b-4396-8fca-558d7479ca69"]
```

```
In [6]: p = p[p.SignalID=="3ac090e9-2e27-4caa-83ae-a14dc8a9f8dd"]
```

```
In [7]: f = f[f.SignalID=="6f71e5ec-4b2a-4188-8c35-6d6b73d8f374"]
```

```
In [8]: p.to_csv("Phase.csv")
```

```
In [9]: v.to_csv("Volts.csv")
```

```
In [10]: f.to_csv("Frequency.csv")
```