2016

Phasor Measurement Unit Testing and Related Topics

Joe Perigo
Montana Tech of the University of Montana

Phasor Measurement Unit Testing and Related Topics

by

Joe Perigo

Senior Design Final Report

Montana Tech of The University of Montana

2016

Abstract

Phasor Measurement Units (PMUs) are devices that take three phase voltages as inputs and output A/B/C phase magnitudes, angles, frequencies, and time stamp data that correlates to these outputs. PMU functionality is typically built into protection relays which are used to protect sensitive areas of the power grid such as transformers and substations. PMU's then become useful tools that can provide critical insight into the power grid, and may have applications in feedback controls as a sensor.

The main goal of this project was to obtain the delay time inherent in SEL-421 and SEL-487 PMU hardware. To test these units I needed to construct a test bench of hardware and software for signal generation, signal monitoring, signal amplification, data acquisition, data parsing, and calculation software to analyze the results of my tests. My research focuses on a signals and systems approach to testing this hardware. By inputting various signals such as phase ramps (frequency steps), amplitude steps, amplitude modulation and phase modulation I am able to determine characteristics of the PMU hardware such as delay time. By inputting a frequency spectrum of phase modulation signals I was able to calculate the transfer function and obtain a bode plot for the SEL 421 which also yields characteristics of the PMU hardware. These results are a critical part of a much larger grid protection project, and the accuracy of these results are paramount.

Keywords: Phasor Measurement Unit (PMU), Delay Time, Signal Generation, Transfer Function, Bode Plot.

Table of Contents

List of Figures

1. Introduction

The process required to run tests on the SEL-421 and the SEL-487 Phasor Measurement Unit (PMU) hardware requires several steps and an understanding of the block diagram shown in Figure 1. The first section of this paper will explain each block in Figure 1 as it pertains to testing PMUs. At this point a step by step procedure will be explained for running a typical test on the PMU. To understand the mathematics required for signal generation a discussion of the equations will be covered. A detailed reference to the math document for this project is also given in Appendix E. The next section of the paper will detail some of the Matlab code used for signal generation, data parsing, and calculation. At this point a section of the paper will be dedicated to baseline testing. This will help ensure that the PMU is functioning as expected and that future testing will yield good results. From here issues related to PMU testing will be covered to aid future testing. The last topic covered will look at our plans for streamlining the test bench and future testing.

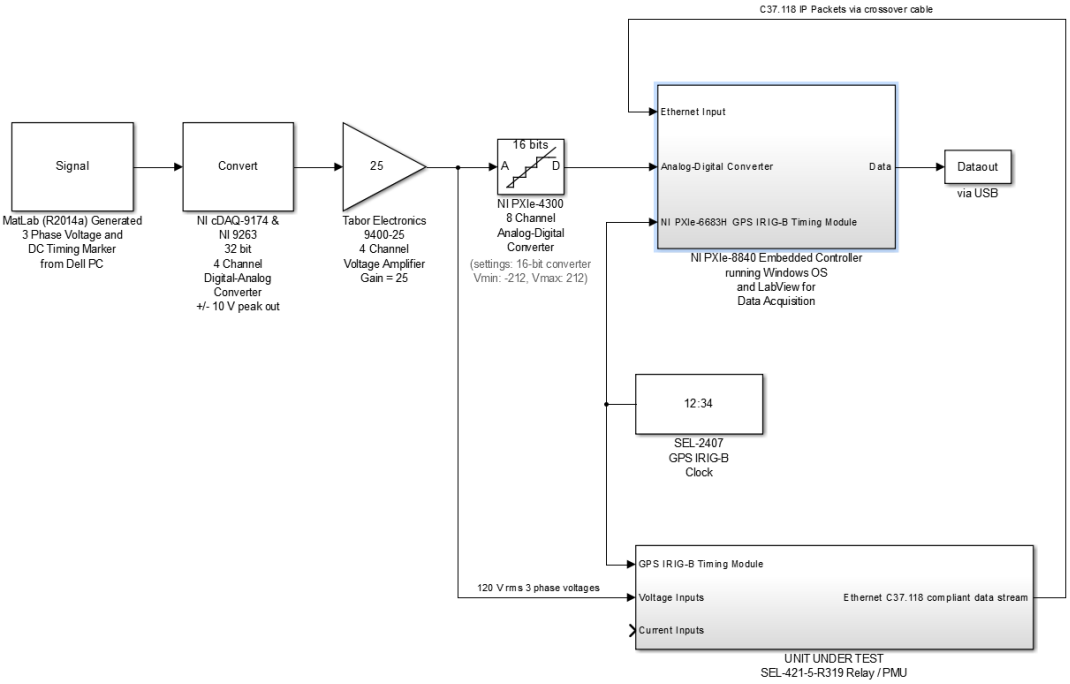


Figure 1: Test Bench Block Diagram

2. Walkthrough of the Block Diagram

2.1. The Signal and Convert Blocks

The signal block in figure 1 is a PC with Matlab setup to communicate with the NI-cDAQ-9174. The NI-cDAQ-9174 is a digital to analog converter (DAC) and is connected to the PC via USB. The sample rate of the NI-cDAQ-9174 can be adjusted up to 100,000 sps. At the time of this writing the sample rate for this device is set to 40,000 sps. In a script file the desired signal can be constructed with the code necessary to communicate with the DAC. Examples of signal generation code with the DAC communication code is provided in Appendix B.

2.2. The Gain Block

Once the signal generation code is compiled Matlab will communicate with the DAC to create an analog signal which is then amplified via the Tabor 9400-25. This is a four channel wide band signal amplifier with an ideal gain of 25. There are a few important things to note about this amplifier. On the back panel of this module there is a *'Unipolar Mode'* option. This turns the amplifier into a two input / four output rectified amplifier. It is important for PMU testing that this option be turned off.

If the PMU is opened up you'll notice that the first thing the input sees is a set of transformers. This means that the input impedance to PMU is inductive. So at low frequencies the input impedance to the PMU is small. The Tabor 9400 is only rated to output 50mA of current but at low frequencies the PMU is drawing more than this. If sinusoid signals are fed to the PMU with frequencies below 30Hz the fuse in the Tabor 9400 amplifier will blow. For PMU testing the frequency range should not drop below 50 Hz. If a fuse is blown there is a cylindrical tab on the back of the module which can be released with a flat head screw driver. The fuses needed for this amplifier are rated for 2 Amps at 250 Volts.

2.3. The SEL 2407 GPS IRIG-B Block

This block represents a GPS clock which provides an absolute time that is used by both the NI PXIe and the SEL PMU hardware in time stamping data packets. When looking at the front panel of this module you will notice a set of LED indicators. For testing purposes it is important that the green LEDs for '*Enabled'* and '*Satellite Lock'* are on. If the orange LED for '*Holdover'* is on that means that the unit has lost its satellite connection. An antenna is connected to the side of the Main Hall building that provides a clear view of the sky and the GPS clock should not have trouble with receiving time from the satellite.

2.4. The NI PXIe-4300 and NI PXIe-8840 Embedded Controller Blocks

The NI PXIe-8840 is a data acquisition system operating with Window OS. LabVIEW is used on this platform to obtain both analog data from the NI PXIe-4300 and phasor data from the PMU. The PXIe-4300 is an analog to digital convertor (ADC) and is plugged directly into the PXIe-8840 chassis. The analog data is time stamped by LabVIEW using the NI PXIe-6683H GPS IRIG-B Timing Module. To check that PMU data packets are arriving through the network as expected it is necessary to open Wireshark on the PXIe-8840 and monitor the appropriate IP address. The time stamps applied to these packets can be saved as a '*.csv* '. Importing this time data into Matlab makes it easy to check for consistency in the arrival time of the PMU data packets.

By opening up the '*PMU C37 Reader.vi'* project in Lab View you will be able to save both a file for the input analog data to the PMU and the output phasor data from the PMU. These files will be saved as a '*.csv* 'and can then be imported to Matlab for processing.

2.5. The SEL-421 and SEL-487 PMU

This block is the unit being tested. SEL relays are used for various protection schemes in the power industry and have additional functionality built into them. The PMU functionality of these devices is the main focus of this paper. Three phase voltage is input into the PMU. Output from this unit includes timestamp data, A/B/C phasor voltages and angles, and a frequency associated with these signals.

3. PMU Testing Procedure

A typical testing procedure is outlined to give a reference for discussions in this paper and will also facilitate more in depth subjects to be addressed in later sections of this document.

3.1. Precautionary Steps

1. Check the SEL-2407 GPS clock to make sure that the '*Satellite Lock*' is on (LED indicator will be green). If '*Satellite Lock*' is off and the '*Hold over*' LED is showing orange it can't be guaranteed that time stamp data will be accurate.
2. Make sure that the '*Unipolar Mode*' on the back of the Tabor 9400 amplifier is off.
3. Check the signal generation code to ensure that frequencies below 30Hz are not being sent to the amplifier. This is assuming that the PMU is being fed the amplified signals. See the description of the '*Gain Block*' above for details.
4. The signal generation code should also be checked to make sure that the signals are zeroed out at the end of the code. This is due to the fact that the DAC will not zero out signals after the signal generation code is complete. For Example, if you run a plain 60Hz sinusoid through the DAC and at the end of the code the sinusoid ends at 20 Volts, then instead of the signal dropping to 0 Volts it will instead sit at 20 Volts DC. This is bad since the PMU will be drawing more current than the amplifier can supply, and a blown fuse will result.

3.2. Start Up

The PC with Matlab will need to be booted and the desired signal generation code ready to go. The NI PXIe-8840 will also need to be booted up with LabVIEW loaded and

ready to go. Below is a detailed description on how to load the *'PMU C37 Reader.vi'* project in LabVIEW.

3.3. LabVIEW

Launch LabVIEW and select the '*Open Existing'* option. In the file directory select '*Montana Tech EE PMU Testing Station'*, then select *'pmuTestingStation'* from the project explorer window. Finally select *'PMU C37 Reader.vi'* from the new project explorer window. Once LabVIEW is ready to go it will be necessary to name the data files. In the Graphic User Interface (GUI) associated with this project there will be two options, '*Phasor Data'* and *'Analog Data'*. *'Phasor Data'* refers to the output from the PMU. *'Analog Data'* refers to the signal received by the NI PXIe-8840 from the NI PXIe-4300. By clicking on the folder icon you will be taken to a save screen. It is important that the naming convention be chosen with some thought as it might make life easier when importing these files into Matlab later. For example, if a frequency sweep test is being run on the PMU a naming convention for the files might look like this:

Phasor File Naming Convention for Frequency Sweep Test:

Phasor_FreqSweep_00.1Hz.csv

Phasor_FreqSweep_00.2Hz.csv

Phasor_FreqSweep_00.3Hz.csv

Analog File Naming Convention for Frequency Sweep Test:

Analog_FreqSweep_00.1Hz.csv

Analog_FreqSweep_00.2Hz.csv

Analog_FreqSweep_00.3Hz.csv

It is necessary to put the file extension *'.csv'* in the file name. The only thing that changes in the file names is the frequency value that is being sent to the NI PXIe-8840 and the PMU. When these files are imported into Matlab for analysis, instead of importing each file one at a time, it will be possible to write a for loop that will import all the files at once. This is made easier with naming conventions similar to the one shown above.

With Matlab and LabVIEW ready to go it is now possible to run tests. The PMU has a *'config'* frame that starts at the beginning of each minute on the GPS clock. This means that data packets will be sent out to the NI PXIe-8840 at the beginning of each minute. This also means that the PMU will not record data until this *'config'* frame starts. The PMU was designed to be receiving 3 phase power 24-7. So with this in mind it is good practice that the input signal be running before the *'config'* frame starts. A typical test might be run as follows:

1. Have Matlab and LabVIEW ready to go as described above
2. While looking at the GPS clock wait for the 45th second. At this time hit play on the LabVIEW GUI to start recording analog data. Remember that phasor data won't record until the config frame starts at the beginning of the next minute.
3. While looking at the GPS clock wait until the 50th second. At this time hit play on the Matlab script to generate the test signal. This will give Matlab a few seconds to compile the signal generation code and send it out before the config frame starts. This will ensure that the PMU is seeing 3 phase signals before the config frame starts.
4. Once the Matlab signal generation code has completed running then hit stop on the LabVIEW GUI to stop recording data.
5. Transfer *'.csv'* data files to a PC with Matlab for processing.

3.4. Phasor and Analog .csv File Contents

Once the phasor and analog files have been saved they can then be imported into Matlab. Before doing this it might be useful to open up the raw *'.csv'* files in both Microsoft Excel and in Notepad for viewing. In Excel the data in both the phasor and analog files will be contained in a single column vector. This knowledge will be helpful when parsing the data with the '*textscan'* function in Matlab. When the files are viewed in Notepad there will be separate columns with

headers that aren't quite lined up. With a little effort the headers can be adjusted. An example of these files is referenced in Appendix A.

4. Math Document

To summarize and prepare the mathematics for this project a math document was developed. The basis of this document are the equations presented in '*Frequency Estimation for Inter-Area Oscillation Feedback Damping Control'* (Trudnowski, Hill, Wold, 2015). The equations are listed in their base form and are then expanded into a testing form to further aid the process of writing test code. Before running signals or writing signal generation code a fundamental knowledge of this document and all the equations listed is paramount. A copy of this document is referenced in Appendix F.

5. Walkthrough of Matlab Code

5.1. Signal Generation Code

The signal generation script file will contain code that generates 3 phase signals as well as code for a timing trigger. The timing trigger will be key in detecting when changes in the input signal occur such as steps in frequency or when switching between frequency modulation and phase modulation. When the data is parsed out and plots are made it will be possible to see exactly when the input signal changed and the PMU's response to this change. The signal generation script file will also need to contain the commands necessary for communication to the NI-cDAQ-9174 as seen in Appendix B.

5.2. Data Parsing Code

As mentioned above the Analog and Phasor *'.csv'* files contain a single column vector. Within this vector is contained various time stamp, voltage, and frequency data. One method to parse this data is to use the *'textscan'* function in Matlab. An example of a data parsing script is

given in Appendix C. This script file is particularly useful for parsing multiple data files. It uses a for loop to step through the naming convention mentioned above. At the end of the for loop the data of interest is then isolated and saved to a '*.mat'* file. This option is only useful if the user wishes to do further calculation in a separate script file, such as calculating transfer functions, and bode plotting. Otherwise simple plotting can be done directly in the data parsing script, such as plotting the analog A phase input against the A phasor magnitude output to see the PMU's response to a particular input.

5.3. Data Loader / Transfer Function Calculator / Bode Plotter

One of the desired goals of this project was to obtain a transfer function for the PMU unit. The method for calculating this transfer function is based on equation 6 listed in Appendix F. With this transfer function a bode plot for the PMU can be used to visualize PMU characteristics. To accomplish this a frequency spectrum of phase modulation signals were input to the PMU. Data parsing code written specifically for this testing sequence is then used to obtain the A phase magnitude, angle, and frequency from each test in the spectrum. An example of this code is referenced in Appendix D with details commented out pertaining to the various calculations.

6. Baseline Test

A large amount of testing on the PMU was run to ensure that the device was behaving as expected. Once it was determined that the devices were giving consistent results a series of base line tests were developed for pre-testing purposes. Only after reproduceing this baseline test should futrue testing be conducted. It was decided the best baseline test would consist of a mixed signal with at least 30 seconds of data. The signal for baseline testing consists of a mix of steps, AM, FM, and PM. By sending the PMU a good mixed signal any red flags that may exsist

in the device should become apparent when running this signal. The signal generation code for baseline testing is referenced in Appendix E.

7. Issues Related to PMU Testing

7.1. Time

The analog and phasor *'.csv'* files will contain three timestamp column vectors. The analog *'.csv'* file contains a timestamp column vector that corresponds to the analog signals being received by both the NI PXIe-8840 and the PMU. The phasor *'.csv'* file contains two timestamp column vectors. The *'PMU_Timestamp'* signify the internal timestamps made by the PMU which correspond to the output of this unit. The *'C37.118_Timestamp*' signifies the timestamps made by LabVIEW as the data packets are received from the PMU which also correspond to the output of this unit. Ideally the *'PMU_Timestamp*' column vector and the *'C37.118 Timestamp*' column vectors should line up perfectly and correspond to the output of the PMU with exact reference to GPS time. Unfortunately they don't.

7.2. Analog Time

With uncertainty in the '*PMU_Timestamp*' and *'C37.118_Timestamp*' data one of the first things we wanted to test was to make sure we could trust the analog GPS timestamp data. To do this two tests were run. The first was to run a 60 Hz sinusoid through the PMU. The input was then captured on an oscilliscope and the data saved to a *'.csv'*. The analog data was then saved from the NI PXIe-8840 and imported into Matlab along with the oscilliscope data. The signals were then plotted on top of eachother. The results of this test show that the oscilliscope data and the NI PXIe-8840 data line up right on top of eachother. The figure below shows a sample of these results.

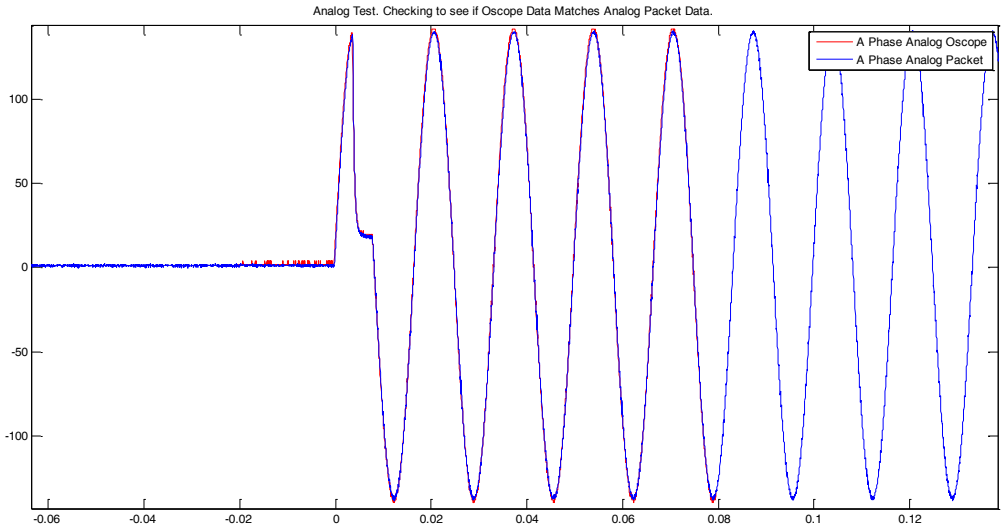


Figure 2: Analog Check Test

The next test that was run involved sending three signals out to the NI PXIe-8840 that were all 60Hz sinusoids in phase with eachother. The output data from the NI PXIe-8840 shows

that these signals lie right on top of eachother as expected. With these results it was concluded that the analog data could be trusted.

7.3. PMU Time

As mentioned above the '*PMU_Timestamp'* column vector represents the timestamps recorded internally by the PMU which are then sent to the NI PXIe-8840 in data packets through the network. The PMU samples at 60 sps and when the data is plotted the data points are consitent with this sample rate. An easy way to check this is to use the '*diff*' funciton in Matlab to plot the difference between data points in the PMU time vector. The issue with the PMU time vector is that it shows non-causal responses in our tests. This implies that that the PMU can see into the future or that this unit is back dating data. After reviewing the data and discusions with Dr. Trudnowski it has been concluded that the PMU looks at a 6 cycle window of data and puts a timestamp in the middle of this window instead of at the end of the window. An example of this response can be seen in the figure below.

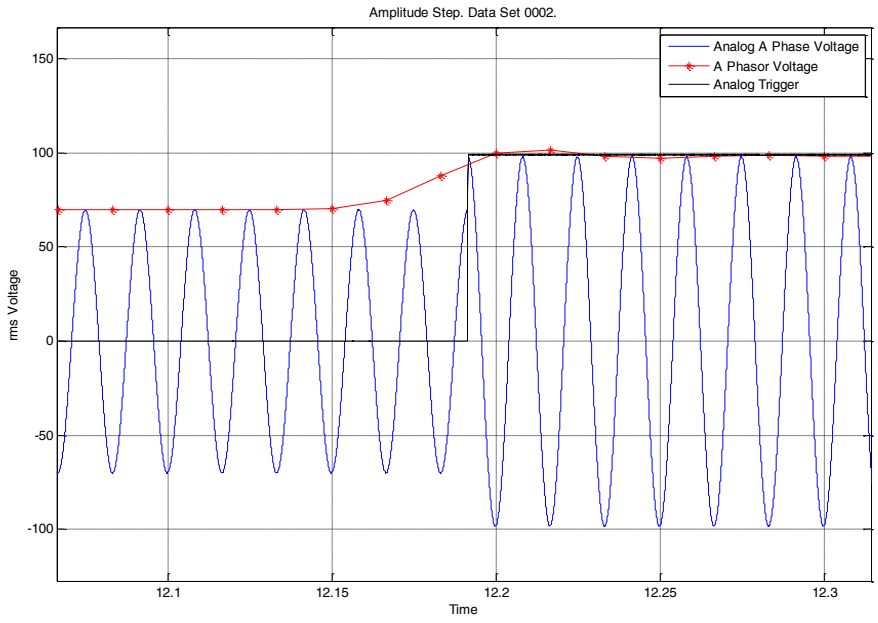


Figure 3: Non-Causal Response Plotted Using PMU Time

The plot above shows the PMU's response to an amplitude step of a 60Hz sinusoid made at the peak. The blue data shows the analog input to the PMU. The black data plot shows the timing trigger that is used in various tests to show exactly when points of interest occur in our tests. The red data corresponds to the PMU's A phasor output. As can be seen in the plot the time data shows that the PMU reacts three samples before the amplitude step even occurs. This is consistent with our knowledge that the PMU is taking a six cycle window of data and applying the timestamp in the middle of this window.

7.4. C37.118 Arrival Time

The *'C37.118_Timestamp'* refers to the timestamps made by LabVIEW as data packets are received at the PXIe through an ethernet connection. These timestamps correspond to the output of the PMU. The issue with this time stamp data is that the timestamps are made inconsistently and end up distorting the data. To see this it is necessary to show individual data points when plotting so that it is possible to see each timestamp. There were two main issues that came up when using the *'C37.118_Timestamp'* data. The first is that the data showed a certain amount of *'jitter'*. This means that the time stamps would show some regularity and then at somewhat constant intervals the timestamps would get backed up in the system. The second major issue with *'C37.118_Timestamp'* data is that the timestamps would be made at regular intervals and then the data would become stretched. This is not simply a delay, but an increasing delay with time causeing the data to stretch out. An example of the jitter issue can be seen

below.

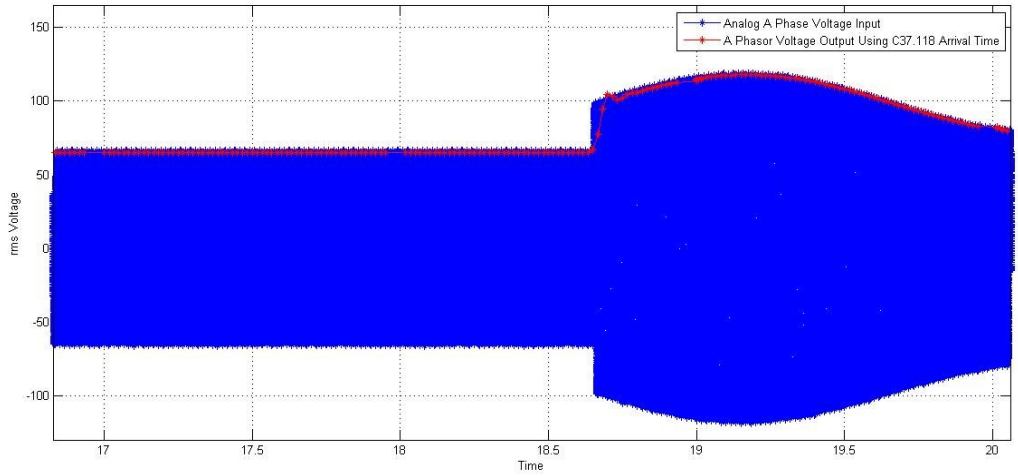


Figure 4: *'C37.118_Timestamp'* Jitter

The above example shows a sinusoidal signal stepping up to an AM signal at approximately 18.6 seconds. When looking closely at the output data from the PMU it is clear that there are gaps in the data at somewhat regular intervals. There are in fact no missing time stamps. When zooming in on the gaps you can see how there are two to three data points that lie on top of each other as shown below.

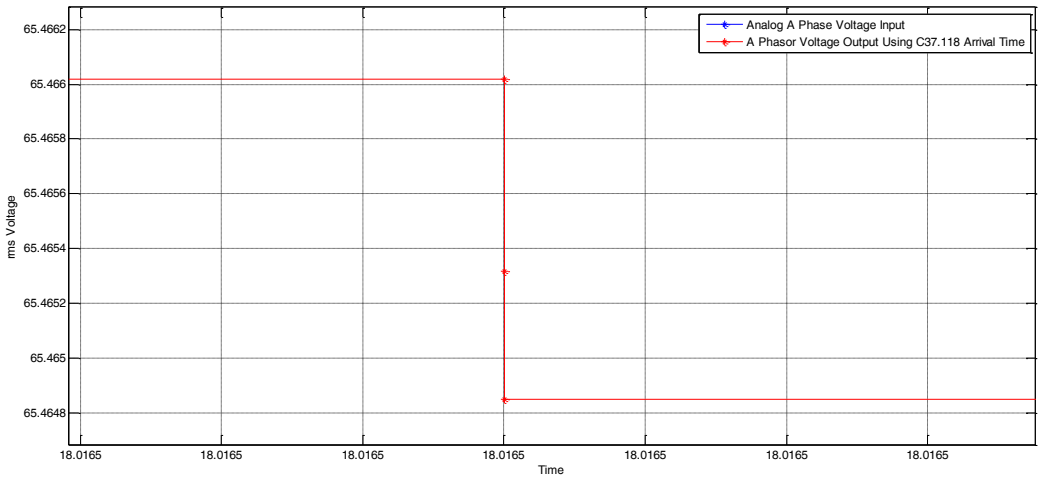


Figure 5: Stacked Data Points

The second issue with the *'C37.118_Timestamp'* data is the stretching effect. This is more of a rare occurrence as it only occurred a handful of times throughout three months of testing. At a glance it can be seen that the data is not being just delayed but stretched as the interval between each time stamp grows.

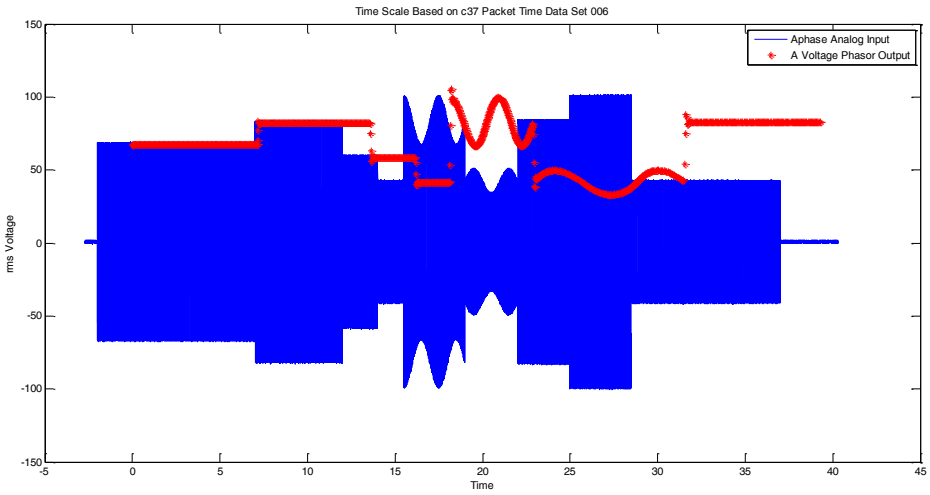


Figure 6: Data Stretching

The stretching is most obvious towards the end of the data set where the smaller AM signal is stretched out compared to the original input signal.

To try and solve the issues with the *'C37.118_Timestamp'* a number of possible solutions were implemented. In discussions with the team it was speculated that the stretching effect and even the jitter might be caused from the PXIe being connected to the school network. To try and solve this we disconnected the PXIe from the network and wired it directly to the PMU through a cross over cable. This didn't seem to fix our issues as the jitter continued as well as the stretching effect. To minimize any delays in the network the PXIe and the PMU remained connected through the cross over cable.

There are a few possible reasons why the *'C37.118_Timestamp'* are having these issues. One reason might be how LabVIEW runs on Windows OS. Windows is somewhat notorious for having issues when networking time is critical. The idea is that data packets are arriving at a buffer in the PXIe, and instead of being timestamped immediately the timestamps are being made after Windows performs other critical tasks. Another possibility is that the LabVIEW code itself might have issues as to how this timestamping is being conducted in the loops contained in the code. The Analog timestamps are also being made by LabVIEW and they don't display any of the jitter or stretching issues associated with the *'C37.118_Timestamp'*.

It was decided that the PMU timestamp data would be used since these timestamps were solid in their consistency.

8. Streamlining the Test Bench and Future Testing

To enhance usability the test bench will be integrated into the rack with the SEL hardware. The main goal was to be able to remote desktop into the test bench and run signals into system and pull data from the PXIe from anywhere on campus. A server specific for this task was considered but the price tag associated with this server made us explore other options. After some consideration the best option turned out to be a rack mounted PC which could then be connected to the network switch in lab along with the PXIe. With this setup it is possible to use the remote desktop application to tap into the test bench. A block diagram of the new test set

up is shown in the figure below.

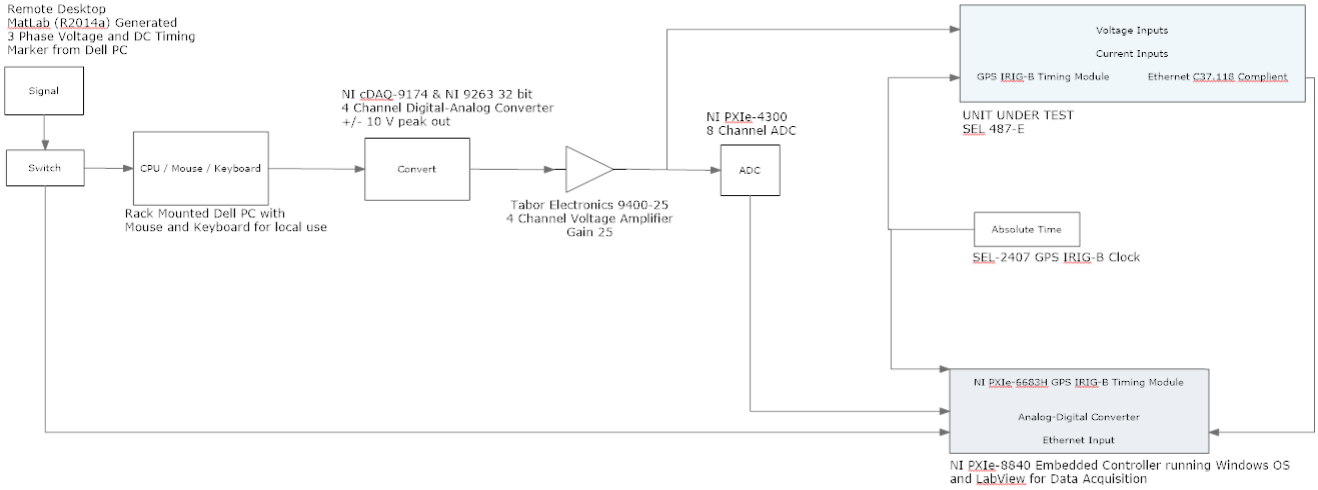


Figure 7: Streamlined Test Bench

9. The Team

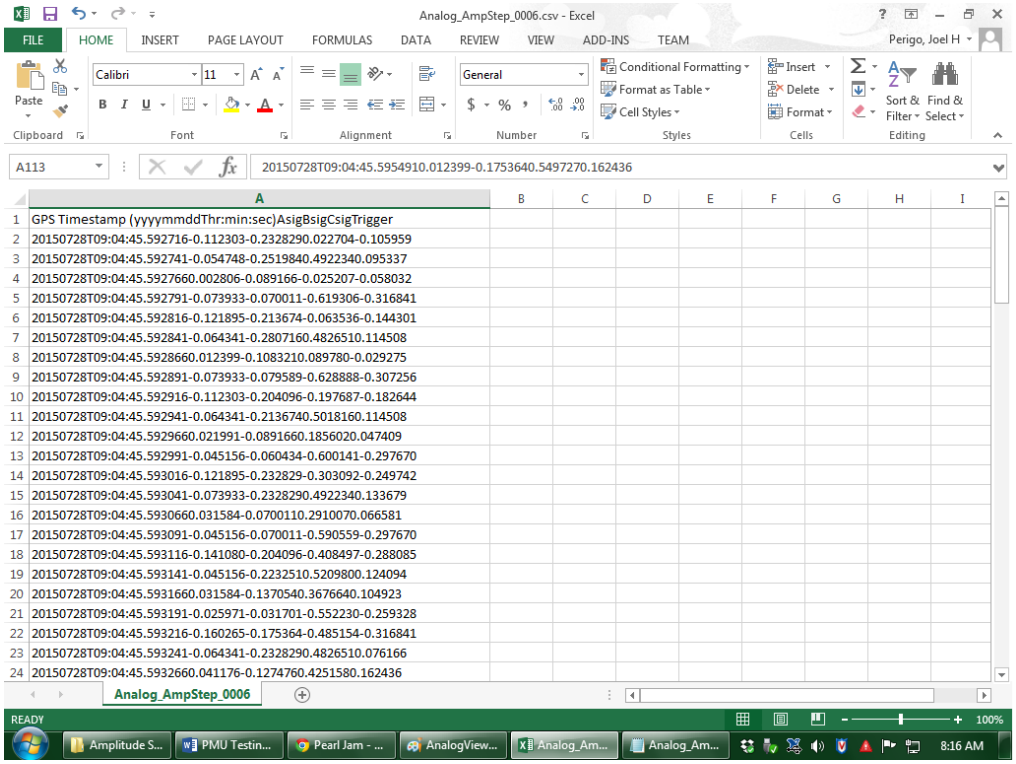
I would like to mention that I would not have been able to accomplish this project without a lot of help from the team that worked with me. James Colwell, Dr. Dan Trudnowski, Dr. Josh Wold, Matt Stajcar, Dr. Bryce Hill, and Dr. Matt Donnelly not only helped me get the job done, but also taught me a lot along the way and I would like to thank them for this.

References Cited

Wold, J., Trudnowski, D., Hill, B. (2015). Frequency Estimation for Inter-Area Oscillation Damping Controller. Not Published.

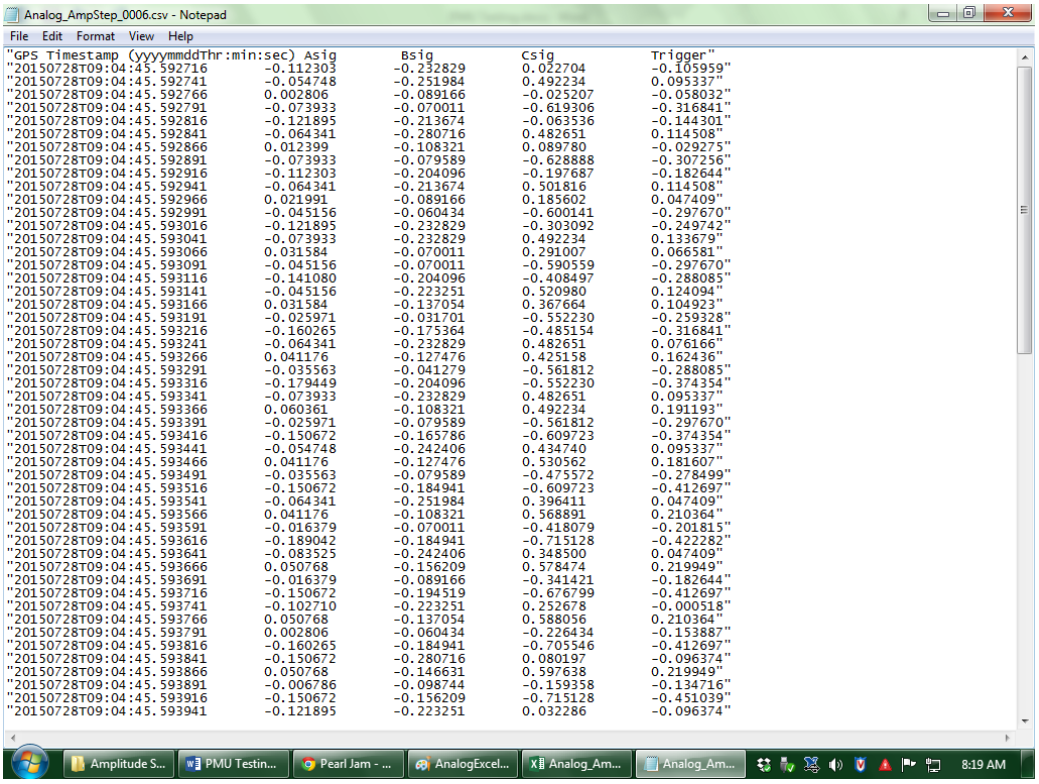
Appendix A: Phasor and Analog .csv contents

Analog File Contents:



Analog_AmpStep_0006.csv - Excel

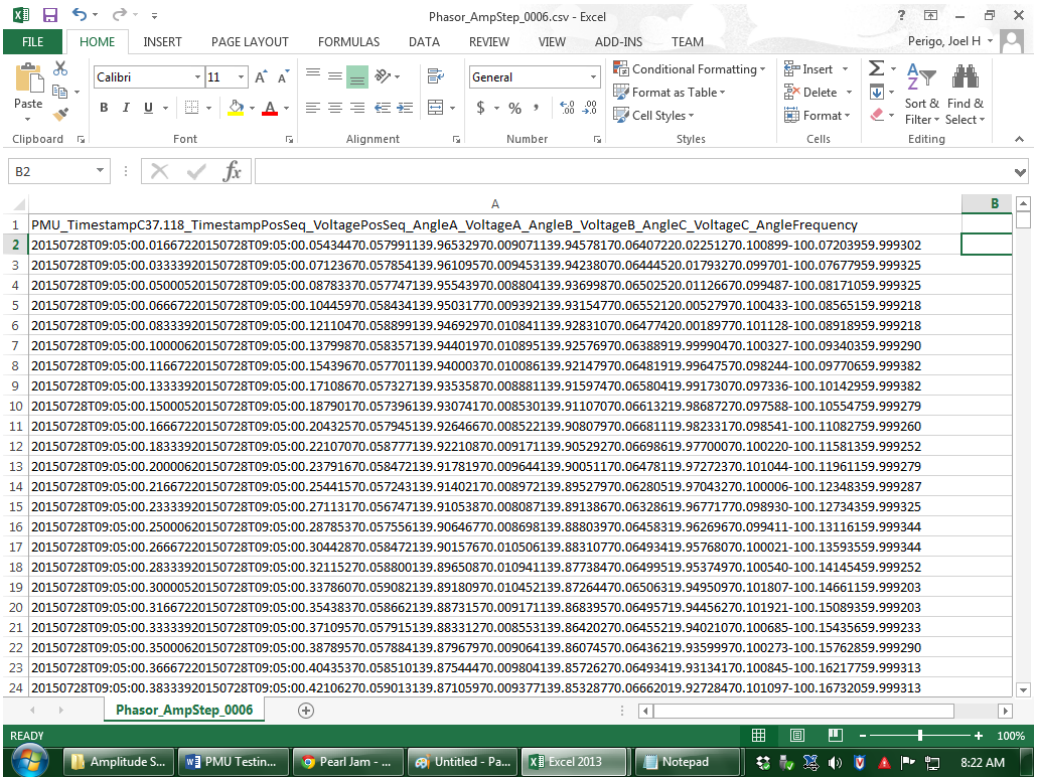
	A
1	GPS Timestamp (yyyymmddThr:min:sec)AsigBsigCsigTrigger
2	20150728T09:04:45.592716-0.112303-0.2328290.022704-0.105959
3	20150728T09:04:45.592741-0.054748-0.2519840.4922340.095337
4	20150728T09:04:45.5927660.002806-0.089166-0.025207-0.058032
5	20150728T09:04:45.592791-0.073933-0.070011-0.619306-0.316841
6	20150728T09:04:45.592816-0.121895-0.213674-0.063536-0.144301
7	20150728T09:04:45.592841-0.064341-0.2807160.4826510.114508
8	20150728T09:04:45.5928660.012399-0.1083210.089780-0.029275
9	20150728T09:04:45.592891-0.073933-0.079589-0.628888-0.307256
10	20150728T09:04:45.592916-0.112303-0.204096-0.197687-0.182644
11	20150728T09:04:45.592941-0.064341-0.2136740.5018160.114508
12	20150728T09:04:45.5929660.021991-0.0891660.1856020.047409
13	20150728T09:04:45.592991-0.045156-0.060434-0.600141-0.297670
14	20150728T09:04:45.593016-0.121895-0.232829-0.303092-0.249742
15	20150728T09:04:45.593041-0.073933-0.2328290.4922340.133679
16	20150728T09:04:45.5930660.031584-0.0700110.2910070.066581
17	20150728T09:04:45.593091-0.045156-0.070011-0.590559-0.297670
18	20150728T09:04:45.593116-0.141080-0.204096-0.408497-0.288085
19	20150728T09:04:45.593141-0.045156-0.2232510.5209800.124094
20	20150728T09:04:45.5931660.031584-0.1370540.3676640.104923
21	20150728T09:04:45.593191-0.025971-0.031701-0.552230-0.259328
22	20150728T09:04:45.593216-0.160265-0.175364-0.485154-0.316841
23	20150728T09:04:45.593241-0.064341-0.2328290.4826510.076166
24	20150728T09:04:45.5932660.041176-0.1274760.4251580.162436

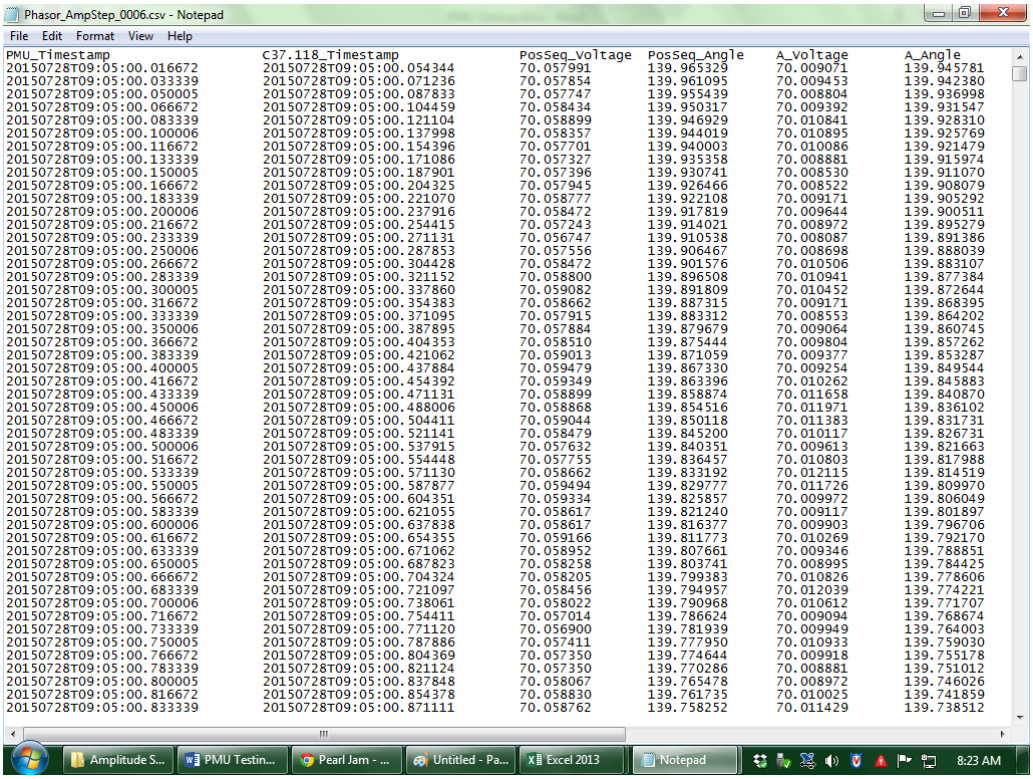


Analog_AmpStep_0006.csv - Notepad

```
"GPS Timestamp (yyyymmddThr:min:sec) Asig        Bsig        Csig        Trigger"
"20150728T09:04:45.592716       -0.112303       -0.232829       0.022704        -0.105959"
"20150728T09:04:45.592741       -0.054748       -0.251984       0.492234        0.095337"
"20150728T09:04:45.592766       0.002806        -0.089166       -0.025207       -0.058032"
"20150728T09:04:45.592791       -0.073933       -0.070011       -0.619306       -0.316841"
"20150728T09:04:45.592816       -0.121895       -0.213674       -0.063536       -0.144301"
"20150728T09:04:45.592841       -0.064341       -0.280716       0.482651        0.114508"
"20150728T09:04:45.592866       0.012399        -0.108321       0.089780        -0.029275"
"20150728T09:04:45.592891       -0.073933       -0.079589       -0.628888       -0.307256"
"20150728T09:04:45.592916       -0.112303       -0.204096       -0.197687       -0.182644"
"20150728T09:04:45.592941       -0.064341       -0.213674       0.501816        0.114508"
"20150728T09:04:45.592966       0.021991        -0.089166       0.185602        0.047409"
"20150728T09:04:45.592991       -0.045156       -0.060434       -0.600141       -0.297670"
"20150728T09:04:45.593016       -0.121895       -0.232829       -0.303092       -0.249742"
"20150728T09:04:45.593041       -0.073933       -0.232829       0.492234        0.133679"
"20150728T09:04:45.593066       0.031584        -0.070011       0.291007        0.066581"
"20150728T09:04:45.593091       -0.045156       -0.070011       -0.590559       -0.297670"
"20150728T09:04:45.593116       -0.141080       -0.204096       -0.408497       -0.288085"
"20150728T09:04:45.593141       -0.045156       -0.223251       0.520980        0.124094"
"20150728T09:04:45.593166       0.031584        -0.137054       0.367664        0.104923"
"20150728T09:04:45.593191       -0.025971       -0.031701       -0.552230       -0.259328"
"20150728T09:04:45.593216       -0.160265       -0.175364       -0.485154       -0.316841"
"20150728T09:04:45.593241       -0.064341       -0.232829       0.482651        0.076166"
"20150728T09:04:45.593266       0.041176        -0.127476       0.425158        0.162436"
"20150728T09:04:45.593291       -0.035563       -0.041279       -0.561812       -0.288085"
"20150728T09:04:45.593316       -0.179449       -0.204096       -0.552230       -0.374354"
"20150728T09:04:45.593341       -0.073933       -0.232829       0.482651        0.095337"
"20150728T09:04:45.593366       0.060361        -0.108321       0.492234        0.191193"
"20150728T09:04:45.593391       -0.025971       -0.079589       -0.561812       -0.297670"
"20150728T09:04:45.593416       -0.150672       -0.165786       -0.609723       -0.374354"
"20150728T09:04:45.593441       -0.054748       -0.242406       0.434740        0.095337"
"20150728T09:04:45.593466       0.041176        -0.127476       0.530562        0.181607"
"20150728T09:04:45.593491       -0.035563       -0.079589       -0.475572       -0.278499"
"20150728T09:04:45.593516       -0.150672       -0.184941       -0.609723       -0.412697"
"20150728T09:04:45.593541       -0.064341       -0.251984       0.396411        0.047409"
"20150728T09:04:45.593566       0.041176        -0.108321       0.568891        0.210364"
"20150728T09:04:45.593591       -0.016379       -0.070011       -0.418079       -0.201815"
"20150728T09:04:45.593616       -0.189042       -0.184941       -0.715128       -0.422282"
"20150728T09:04:45.593641       -0.083525       -0.242406       0.348500        0.047409"
"20150728T09:04:45.593666       0.050768        -0.156209       0.578474        0.219949"
"20150728T09:04:45.593691       -0.016379       -0.089166       -0.341421       -0.182644"
"20150728T09:04:45.593716       -0.150672       -0.194519       -0.676799       -0.412697"
"20150728T09:04:45.593741       -0.102710       -0.223251       0.252678        -0.000518"
"20150728T09:04:45.593766       0.050768        -0.137054       0.588056        0.210364"
"20150728T09:04:45.593791       0.002806        -0.060434       -0.226434       -0.153887"
"20150728T09:04:45.593816       -0.160265       -0.184941       -0.705546       -0.412697"
"20150728T09:04:45.593841       -0.150672       -0.280716       0.080197        -0.096374"
"20150728T09:04:45.593866       0.050768        -0.146631       0.597638        0.219949"
"20150728T09:04:45.593891       -0.006786       -0.098744       -0.159358       -0.134716"
"20150728T09:04:45.593916       -0.150672       -0.156209       -0.715128       -0.451039"
"20150728T09:04:45.593941       -0.121895       -0.223251       0.032286        -0.096374"
```

Phasor File Contents:





Appendix B: Signal Generation Code

Sinusoid Signal Generation Example

```
% This code generates a simple sinusoid. The frequency can be adjusted via
% the frequency variable "f".  If this signal is being amplified via the
% Tabor 9400 and then sent to the Relay/PMU it is important to keep the
% frequency above 30Hz.

clear all;
close all;
clc;
tic

% Signal Parameters
f = 60; % in Hz

% 'sigmag' refers to the signal magnitude in the sinusoids below.  To
% calculate this take 6.5*25 = 162 Volts. The '25' refers to the gain of
% the Tabor 9400.  We want to keep our signal below 170 Volts so setting
% sigmag = 6.5 should be good.
sigmag = 6.5;

% 'DACfreq' refers to the sample rate of the NI-cDAQ-9174.  In the time
% vector 't' it is pointless to try and sample any higher than 1/DACfreq
% since the NI-cDAQ-9174 will only sample at 1/40000.  The sample rate of
% the NI-cDAQ-9174 can be adjusted up to 100000 sps.  If this adjustment is
% made to the hardware make sure to adjust the code appropriately.
DACfreq = 40000; % in Hz
tend = 45; % in seconds
t = 0:1/DACfreq:tend;

% Signal generation code for A, B, and C phases.  Signal code for a timing
% trigger is also included. A, B, and C phases are basic sinusoids.  The
% step fun command is used to turn on 0 volts DC at 0 seconds and off at 5
% seconds.  Then a basic sinusoid is turned on at 5 seconds and turned off
% at 40 seconds.  The signals are then zeroed at 40 seconds until tend.  It
% is important to zero the signals out due to the fact that the
% NI-cDAQ-9174 will hold the final sinusoid value as a DC signal at the end
% of the code.  The PMU draws more current at low frequencies than the
% Tabor 9400 can supply especially at DC.
Asig = 0.00.*(stepfun(t,0)-stepfun(t,5.00))+...
       sigmag*sin(2*pi*f*t).*(stepfun(t,5.00)-stepfun(t,40.00))+....
       0.00.*(stepfun(t,40.00)-stepfun(t,tend));

Bsig = 0.00.*(stepfun(t,0)-stepfun(t,5.00))+...
       sigmag*sin(2*pi*f*t - 2*pi/3).*(stepfun(t,5.00)-stepfun(t,40.00))+...
       0.00*sin(2*pi*60*t - 2*pi/3).*(stepfun(t,40.00)-stepfun(t,tend));

Csig = 0.00.*(stepfun(t,0)-stepfun(t,5.00))+...
       sigmag*sin(2*pi*f*t + 2*pi/3).*(stepfun(t,5.00)-stepfun(t,40.00))+...
       0.00.*(stepfun(t,40.00)-stepfun(t,tend));
```

```
Trig = 0.00.*(stepfun(t,0)-stepfun(t,5.00))+...
       6.5*(stepfun(t,5.00)-stepfun(t,40.00))+...
       0.00.*(stepfun(t,40.00)-stepfun(t,tend));
```

DAQ communication code

The following code is used to communicate with the NI-cDAQ-9174.

```
dataOut = [Asig' Bsig' Csig' Trig'];

% Create the data acquisition session.
cDAQ = daq.createSession('ni');

% Create analog output channel on board ID 'cDAQ1Mod1', channel # 'ao0', with signal type
'Voltage');
cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao0', 'Voltage');
cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao1', 'Voltage');
cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao2', 'Voltage');
cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao3', 'Voltage');

cDAQ.Rate = DACfreq; % Refresh rate of DAQ [Hz].

queueOutputData(cDAQ,dataOut);
startForeground(cDAQ);

% Clean up and release hardware.
cDAQ.release();
delete(cDAQ);
clear cDAQ;
toc

%
```

Appendix C: Data Parsing Code

Data Parsing Code

This code is used to retrieve data from the phasor and analog csv files which are saved from LabView. The format of the csv files requires that various time stamp values, voltages and phase angels are parsed out using the textscan function. More detail on this function is provided below.

```
close all;
clear all;
clc;

% 'freq_mod_value' describes a list of modulation frequencies(Hz) used in a
% phasse modulation test.  By putting these values in a cell array we are
% able to use a for loop to parse the data out of multiple files all at
% once rather than individually. The for loop will use 'freq_mod_value' for
% indexing.
```

```
freq_mod_value={'00.1','00.2','00.3','00.4','00.5','00.6','00.7','00.8',...
    '00.9','01.0','01.1','01.2','01.3','01.4','01.5','01.6','01.7',...
    '01.8','01.9','02.0','02.1','02.2','02.3','02.4','02.5','02.6',...
    '02.7','02.8','02.9','03.0','03.1','03.2','03.3','03.4','03.5',...
    '03.6','03.7','03.8','03.9','04.0','04.1','04.2','04.3','04.4',...
    '04.5','04.6','04.7','04.8','04.9','05.0','06.0','07.0','08.0',...
    '09.0','10.0','12.0','14.0','16.0','18.0','20.0','22.0','24.0',...
    '26.0','28.0','30.0','35.0','40.0','45.0','50.0','55.0','60.0'}
```

Phasor Data Parsing

Using fopen is necessary for the textscan function to get data out of the file. The parameters of the textscan function are the fileID variable, followed by several character types which appear in a specific order relative to how the data is organized in the csv file. The textscan function also requires you to define how many header lines are in the csv file and how it is delimited.

```
for k=1:length(freq_mod_value)

    infileID=['Phasor_FreqSweep_' freq_mod_value{k} 'Hz.csv'];
    fileID = fopen(infileID);
    CP = textscan(fileID,...
        '%f%c%f:%f:%f%f%c%f:%f:%f %f %f %f %f %f %f %f %f %f ',...
        'Headerlines',1,'Delimiter',',');
    fclose(fileID);

    % The Data is now parsed and seperated out in large columns that are
    % contained in cells.  To make this data more accesable and easier to know
    % exactly what the data represents it will be removed from the cells to be
    % stand alone vectors that are named accordingly.

    % Define PMU Time Stamp Variables
    PMUtStampYMD = CP{1};
    PMUtStampHour = CP{3};
    PMUtStampMin = CP{4};
    PMUtStampSec = CP{5};

    % Define c37 Time Stamp Variables
    c37tStampYMD = CP{6};
    c37tStampHour = CP{8};
    c37tStampMin = CP{9};
    c37tStampSec = CP{10};

    % Define Voltage Phase Varables
    posSeqVol = CP{11};
    posSeqAng = CP{12};
    PhasorAVol = CP{13};
    PhasorAAng = CP{14};
    PhasorBVol = CP{15};
    PhasorBAng = CP{16};
    PhasorCVol = CP{17};
    PhasorCAng = CP{18};
```

```
PhasorFreq = CP{19};

% This section of the code is used to obtain the time the test was run.
% There are two sets of time stamp data.  Total test time will be
% determined for both the PMU time stamps and the c37 time stamps.  Each
% set of data will also have a time vector used for plotting.

% This section is used to shift the data to start at time = 0 seconds.
zeroPMUPhasorTime = (PMUtStampHour(1) * 60 * 60) +...
    (PMUtStampMin(1) * 60) + (PMUtStampSec(1));

zeroC37PhasorTime = (c37tStampHour(1) * 60 * 60) +...
    (c37tStampMin(1) * 60) + (c37tStampSec(1));

zeroTime1 = zeroPMUPhasorTime;

% New time vectors.  Takes the time vectors from the hour, min, and seconds
% and then converts all to seconds.  Then we shift the scale to start at
% zero.
PMUtimeVector = (PMUtStampHour * 60 * 60) + (PMUtStampMin * 60) +...
    PMUtStampSec - zeroTime1;
c37TimeVector = (c37tStampHour * 60 * 60) + (c37tStampMin * 60) +...
    c37tStampSec - zeroTime1;
```

Analog Data Parsing

See the above comment for a description on the textscan function

```
infileID=['Analog_FreqSweep_' freq_mod_value{k} 'Hz.csv'];
fileID = fopen(infileID);
CA = textscan(fileID,'%f%c%f:%f:%f %f %f %f %f ','Headerlines',1,...
    'Delimiter',',');
fclose(fileID);

% As described above this section of the code is pulling the data columns
% out of cells and naming them for easier access and use.
AnalogtStampHour = CA{3};
AnalogtStampMin = CA{4};
AnalogtStampSec = CA{5};

% This section is used to make sure the plots start at time zero
zeroAnalogTime = (AnalogtStampHour(1) * 60 * 60) +...
    (AnalogtStampMin(1) * 60) + AnalogtStampSec(1);

% New time vectors.  Takes the time vectors from the hour, min, and seconds
% and then converts all to seconds.  Then we shift the scale to start at
% zero.
AnalogTimeVector = (AnalogtStampHour * 60 * 60) +...
    (AnalogtStampMin * 60) + AnalogtStampSec -...
    zeroTime1;

% Give variable names to the 3 phases
```

```
    AnalogAphase = CA{6};
    AnalogBphase = CA{7};
    AnalogCphase = CA{8};
    AnalogTrig = CA{9};

%     Voltage plot.  Good for checking that the data is ok.
%     figure
%     plot(PMUtimeVector,PhasorAVol,'*-r')
%     hold on
%     plot(AnalogTimeVector,AnalogAphase/sqrt(2),'b')
%     xlabel('Time')
%     ylabel('Magnitude')
%     legend('A Phasor Voltage','Analog A Phase')
%     grid
%     title(freq_mod_value{k})
%     axis([-10 60 -150 150])

    %  This code is used to find a specific voltage magnitude for the
    %  timing trigger.  By finding the index of the trigger at the desired
    %  point we can capture a known input signal time length and a known
    %  PMU output time length.
    DataStart = find(AnalogTrig>139.8);
    DataStart = DataStart(1);
    % Use the time vectors to relate Analog and Phasor data.
    PMUstart = find(PMUtimeVector>AnalogTimeVector(DataStart));
    % Starting point of interst for PMU data.
    PMUstartIndx = PMUstart(1)

    % Isolating Phasor frequency, A phase magnitude, and A phase angle data
    % of interest from PMU.
    PhasorFreq_Freq = PhasorFreq(PMUstartIndx:PMUstartIndx + 1800);
    PhasorAVol_Mag = PhasorAVol(PMUstartIndx:PMUstartIndx + 1800);
    PhasorAAng_Ang = PhasorAAng(PMUstartIndx:PMUstartIndx + 1800);
    % Using the for loop index to step through 'freq_mod_value' and save
    % the data above as '.mat' files.
    outfileMag=['PhasorFreqSweep_Mag_' freq_mod_value{k} 'Hz.mat'];
    outfileAng=['PhasorFreqSweep_Ang_' freq_mod_value{k} 'Hz.mat'];
    outfileFreq = ['PhasorFreqSweep_Freq_' freq_mod_value{k} 'Hz.mat'];
    save(outfileMag,'PhasorAVol_Mag')
    save(outfileAng,'PhasorAAng_Ang')
    save(outfileFreq,'PhasorFreq_Freq') end;
```

Appendix D: Data Loader / Transfer Function Calculator / Bode Plotter

Data Loader / Transfer Function Calculator / Bode Plotter

This script loads the A phase voltage, A phase angle, and frequency output data from the PMU. From here a numerical derivative is done on the A phase angle to get frequency. Then the fft is done on the frequency to get eqn. 4 from the *'Frequency Estimation for Inter-Area Oscillation Feedback Damping Controller*' paper. The input term in the denominator is simple since we know the derivative of the frequency/phase term in the phase modulation signal which was sent to the PMU for processing. With this derivative we also know the fft of this will be Aphi*wm < wm*to.

```
close all;
clear all;
clc;
```

Setting up an array for the modulation values.

Set up frequency sweep for the phase modulation in a cell array.

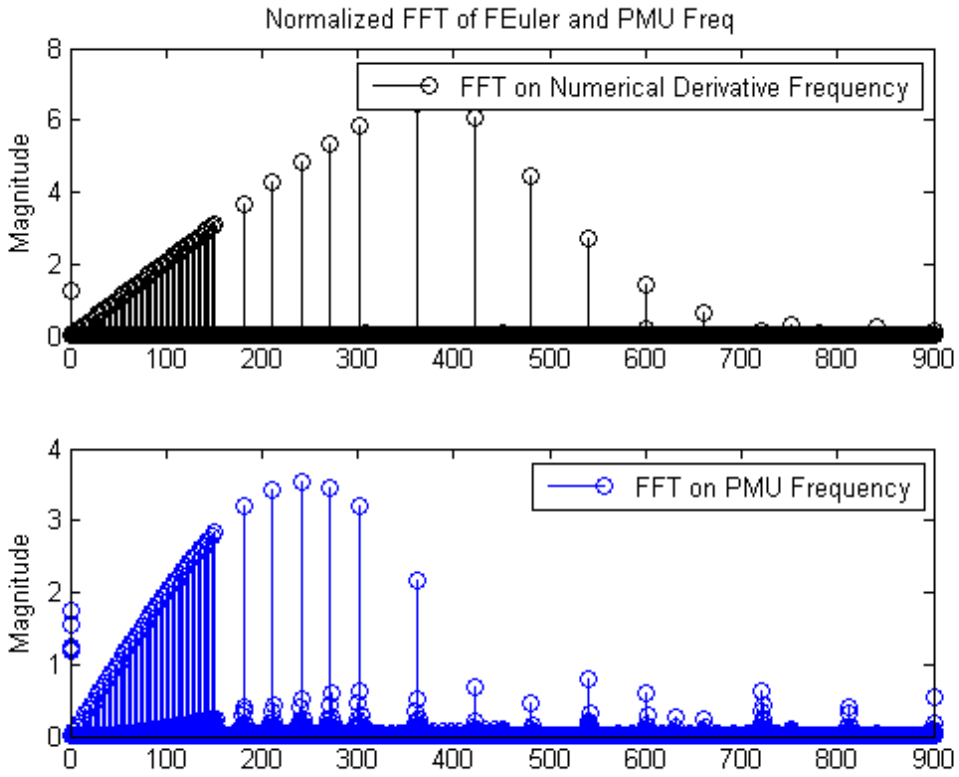
```
freq_mod_value={'00.1','00.2','00.3','00.4','00.5','00.6','00.7','00.8',...
    '00.9','01.0','01.1','01.2','01.3','01.4','01.5','01.6','01.7',...
    '01.8','01.9','02.0','02.1','02.2','02.3','02.4','02.5','02.6',...
    '02.7','02.8','02.9','03.0','03.1','03.2','03.3','03.4','03.5',...
    '03.6','03.7','03.8','03.9','04.0','04.1','04.2','04.3','04.4',...
    '04.5','04.6','04.7','04.8','04.9','05.0','06.0','07.0','08.0',...
    '09.0','10.0','12.0','14.0','16.0','18.0','20.0','22.0','24.0',...
    '26.0','28.0','30.0','35.0','40.0','45.0','50.0','55.0','60.0'};
```

Transfer Function Calculation

```
for k=1:length(freq_mod_value)-1 % Steps through each frequency.

    % Assigns a variable with index of k to a file location which also
    % steps through the naming convention using ' freq_mod_value{k} '
    Angstr{k}=['C:\Users\jhperigo\Desktop\PMU Testing\New PMU with Updated Firmware\Transfer
Function Test 1\PhasorFreqSweep_Ang_' freq_mod_value{k} 'Hz.mat'];
    Magstr{k}=['C:\Users\jhperigo\Desktop\PMU Testing\New PMU with Updated Firmware\Transfer
Function Test 1\PhasorFreqSweep_Mag_' freq_mod_value{k} 'Hz.mat'];
    Freqstr{k} = ['C:\Users\jhperigo\Desktop\PMU Testing\New PMU with Updated Firmware\Transfer
Function Test 1\PhasorFreqSweep_Freq_' freq_mod_value{k} 'Hz.mat'];

    % Sets up modulation frequency by pulling values out of freq_mod_value
    % and converting them to with str2num.
    modfreq(k)=str2num(freq_mod_value{k}); % Hz

    % Now sets a variable for Magnitude and Angle which loads from the
    % variable set up at lines 32 and 33.
    AMag{k} = load(Magstr{k});
    AMag{k} = AMag{k}.PhasorAVol_Mag;
    AAng{k} = load(Angstr{k});
    AAng{k} = AAng{k}.PhasorAAng_Ang;
    PMUfreq{k} = load(Freqstr{k});
    PMUfreq{k} = (PMUfreq{k}.PhasorFreq_Freq-60)*2*pi; % In rads/sec

    % Taking the numerical derivative of the AAng{k} to get frequency.
    Theta = AAng{k};
    Theta = pi/180.*Theta; % Radians
    X = angle(exp(1i*Theta(2:end))./exp(1i*Theta(1:end-1)));
    %FEuler{k} = [0;X]*60; % in radians/sec: Used for comparing freqs.
    FEuler{k} = X*60; % in radians/sec: Used for bode plotting

    % Doing fft of FEuler{k} to get to the transfer function mentioned
```

```
    % above. Then we break up the fft vector into its magnitude and angle
    % so that the division of phasors can be done.
    FFTofFhat{k} = 2/length(FEuler{k})*fft(FEuler{k}); %exp(-1*2*pi*modfreq(k)*3/60);
    FFTofFhatMag{k} = abs(FFTofFhat{k});
    FFTofFhatAngle{k} = angle(FFTofFhat{k});

    % Now for the frequency that was calculated by the PMU we'll call the
    % variable PMUfreq and then use the fft funtion on it.  Then as above
    % I'll split the complex number into it's magnitude and angle to make
    % the polar division easier.
    FFTofPMUfreq{k} = 2/length(PMUfreq{k})*fft(PMUfreq{k}); %*exp(-1i*2*pi*modfreq(k)*3/60);
    FFTofPMUfreqMag{k} = abs(FFTofPMUfreq{k});
    FFTofPMUfreqAngle{k}= angle(FFTofPMUfreq{k});

    % If the FFTofFhatMag is plotted there will be two spikes that
    % correspond to the modulation frequency being fed to the PMU.  'i' is
    % used to find the index of these peaks so that the magnitude and angle
    % at this index can be easily aquired.
    i(k) = round(length(FFTofFhatMag{k})/60*modfreq(k)+1);

    % Transfer function Division 1.
    % The following 4 lines are calculating the transfer function using the
    % frequency values that were calculated using the numerical derivative.
    % 'GMag' is doing the division in the transfer function.
    GMag(k) = FFTofFhatMag{k}(i(k))/(0.1*2*pi*modfreq(k));
    % 'GAng' is doing the subtraction in the transfer function.
    GAng(k) = FFTofFhatAngle{k}(i(k)) - (2*pi*modfreq(k))*(22) - pi/2 - 2*pi*modfreq(k)*3/60;


    % Transfer function Division 2.
    % The following 4 lines are calculating the transfer function using the
    % frequency values that were calculated by the PMU.
    % 'GMag_PMUfreq' is doing the division in the transfer function
    GMag_PMUfreq(k) = FFTofPMUfreqMag{k}(i(k))/(0.1*2*pi*modfreq(k));
    % 'GAng_PMUfreq' is doing the subtraction in the transfer function.
    GAng_PMUfreq(k) = FFTofPMUfreqAngle{k}(i(k)) - (2*pi*modfreq(k))*(22) - pi/2 -
2*pi*modfreq(k)*3/60;

    % Plotting the magnitudes of the fft's both from the numerical
    % derivative frequency and the PMU frequency to check for leakage.
    subplot(211)
    stem(FFTofFhatMag{k},'k')
    hold on
    title('Normalized FFT of FEuler and PMU Freq')
    ylabel('Magnitude')
    legend('FFT on Numerical Derivative Frequency')
    xlim([0 900])
    grid
    subplot(212)
    stem(FFTofPMUfreqMag{k},'b')
    hold on
    ylabel('Magnitude')
    legend('FFT on PMU Frequency')
```

```
    xlim([ 0 900])
    grid


end
```



Dan's Code

```
% Plot Freq resp
% clear all; close all; clc

% load('FreqResp1A','Test','fSample2'); %kx=0.1, ka=0.1, sinusoid model,Nonlin.nCycles = 1,
nCycles=1
% load('FreqResp1B','Test','fSample2'); %kx=0.1, ka=0.1, rampled sinusoid model, Nonlin.nCycles =
1, nCycles=1
% load('FreqResp1C','Test','fSample2'); %kx=0.1, ka=0.1, rampled sinusoid model, Nonlin.nCycles =
2, nCycles=1
load('FreqResp1D','Test','fSample2'); %kx=0.1, ka=0.1, rampled sinusoid model, Nonlin.nCycles =
2, nCycles=2
```

Create spectrum data

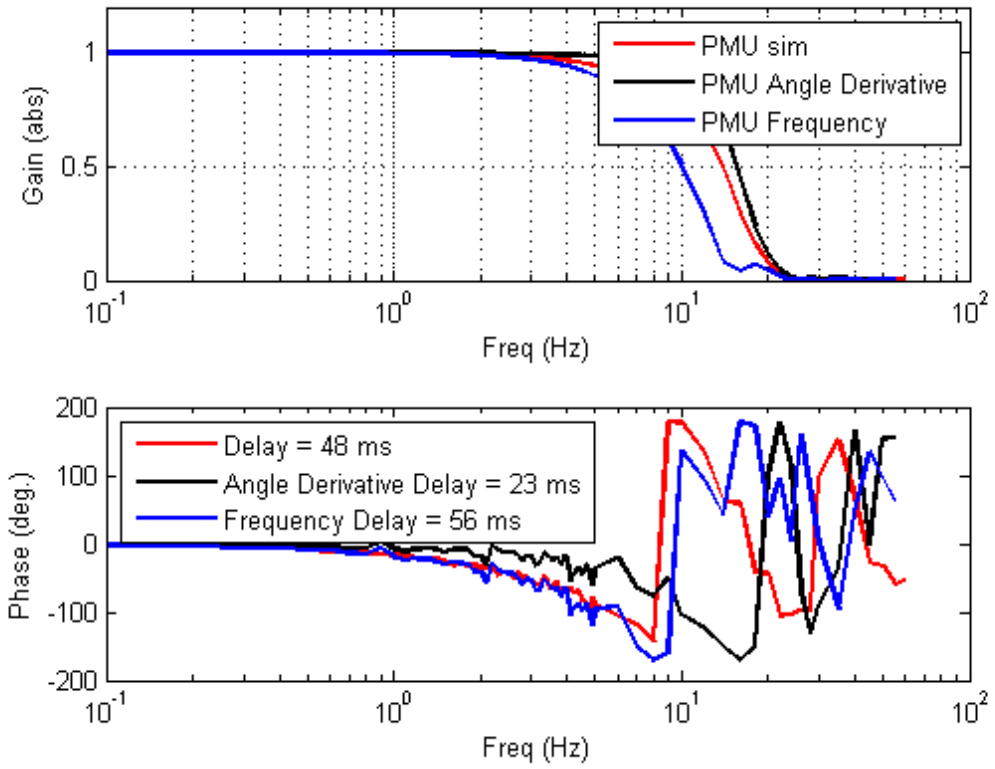
```
kC = [1:71]';
Gbode.PMU.F = zeros(length(kC),1);
```

```
Gbode.PMU.FB = zeros(length(kC),1);
Gbode.f = zeros(length(kC),1);
for k=1:length(kC)
    if Test{kC(k)}.Sig.Param.fm>7
        nCy = ceil([0.2*Test{kC(k)}.Sig.Param.fm; 0.8*Test{kC(k)}.Sig.Param.fm]);
        if abs(round(nCy(1)/2)-nCy(1)/2) > 0.1; nCy(1) = nCy(1) + 1; end
        if abs(round(nCy(2)/2)-nCy(2)/2) > 0.1; nCy(2) = nCy(2) + 1; end
    else
        nCy = [3;7]';
    end
    tR = nCy./Test{kC(k)}.Sig.Param.fm; %Range for plotting in sec.
    nSig = [round(tR(1)*Test{k}.Sig.fSample):1:round(tR(2)*Test{k}.Sig.fSample)-1]';
    tSig = (1/Test{k}.Sig.fSample)*[0:length(Test{kC(k)}.Sig.f)-1]';
    ne = round(((nCy(2)-nCy(1))/Test{kC(k)}.Sig.Param.fm)*fSample2*60);
    [~,ns] = min(abs(tR(1)-Test{1}.PMU.t));
    nPMU = [ns:1:ne+ns-1]';
    clear ns ne tR

    %Freq fesp
    Gbode.f(k) = Test{kC(k)}.Sig.Param.fm;
    f = (fSample2*60/length(nPMU))*[0:length(nPMU)-1]';
    [~,n] = min(abs(Gbode.f(k)-f));
    HPMU = (2/length(nPMU))*fft(Test{kC(k)}.PMU.FEuler(nPMU));
    HPMU = HPMU(n);
    HPMUBD = (2/length(nPMU))*fft(Test{kC(k)}.PMU.FBesselD(nPMU));
    HPMUBD = HPMUBD(n);
    f = (Test{kC(k)}.Sig.fSample/length(nSig))*[0:length(nSig)-1]';
    [~,n] = min(abs(Gbode.f(k)-f));
    Hact = (2/length(nSig))*fft(Test{kC(k)}.Sig.f(nSig));
    Hact = Hact(n);
    Gbode.PMU.F(k) = HPMU/Hact;
    Gbode.PMU.FB(k) = HPMUBD/Hact;
    clear n f HPMUBD HPMU HZeroCrossB HZeroCross HNonlinB HNonlin nZer nSig nNon nPMU nCy
end
clear k

%1-Hz Delay
[~,n] = min(abs(Gbode.f-1));
d = unwrap(angle(Gbode.PMU.F(1:n)));
Gbode.PMU.Fdelay = -(d(end)-d(1))/(2*pi*(Gbode.f(n)-Gbode.f(1)));
clear n d

D = unwrap(GAng(1:10));
derivDelay = -(D(end)-D(1))/(2*pi*(1-0.1));

E = unwrap(GAng_PMUfreq(1:10));
freqDelay = -(E(end)-E(1))/(2*pi*(1-0.1));
```

Plot bodes

No Bessel filter defining the modulation frequency sweep for plotting below.

```
FreqSweep = [0.1:0.1:4.9, 5:1:10, 12:2:30, 35:5:60];

figure
subplot(211)
semilogx(Gbode.f,abs(Gbode.PMU.F),'r','Linewidth',2)
hold on
semilogx(FreqSweep(1:end -1),GMag,'k','Linewidth',2)
semilogx(FreqSweep(1:end -1),GMag_PMUfreq,'b','Linewidth',2)
hold off
ylim([0 1.2])
xlabel('Freq (Hz)')
ylabel('Gain (abs)')
legend('PMU sim','PMU Angle Derivative','PMU Frequency')
grid
subplot(212)
semilogx(Gbode.f,(180/pi)*angle(Gbode.PMU.F),'r','Linewidth',2)
hold on
semilogx(FreqSweep(1:end -1),wrapToPi(GAng)*180/pi,'k','Linewidth',2)
semilogx(FreqSweep(1:end -1),wrapToPi(GAng_PMUfreq)*180/pi,'b','Linewidth',2)
hold off
ylabel('Phase (deg.)')
xlabel('Freq (Hz)')
legend(['Delay = ' num2str(round(1e3*Gbode.PMU.Fdelay)) ' ms'],...
       ['Angle Derivative Delay = ' num2str(round(1e3*derivDelay)) ' ms'],...
       ['Frequency Delay = ' num2str(round(1e3*freqDelay)) ' ms'],...
        'Location','NorthWest');
grid

% Loading Frequency for comparision between PMU, Numerical Derivative
% Frequency and Matlab.
load('PhasorFreqSweep_Freq_01.0Hz.mat')

% Creating a typical input signal with desired frequency for comparison.
t = 0:1/60:30;
% V(t) = 169.6*cos(2*pi*60*t+0.1*cos(2*pi*0.1*t)); % typical PM voltage
dangdt = 59.9993-0.1*sin(2*pi*1*t+1*pi/72); % DC and phase adjusted 29 0.1^2

% This plot is comparing the numerical derivative frequency, PMU frequency,
% and Matlab frequency. For this plot to work "FEuler{k} = X*60" needs to
% be replaced with "FEuler{k} = [0;X]*60; % in radians/sec".  This is so
% the vector lengths will match up.
% figure
% plot(t,PhasorFreq_Freq-0.1,'k',t,FEuler{10}/(2*pi)+60-0.1,'r',t,dangdt,'b')
% title('Frequency Comparison')
% legend('PMU Frequency','Derivative Frequency', 'Matlab Frequency')
% ylabel('Frequency (Hz)')
% xlabel('Time (sec)')
% % ylim([59.985 60.015])
grid


%
```



Appendix E: Baseline Testing Code

Baseline Testing Signal Generation.

Combines Steps/AM/FM/PM signals ensure PMU functionality.

```
close all;
clear all;
clc;

tic
```

Signal Parameters

```
fbase = 60; % In Hz
Tbase = 1/fbase; % in seconds
amodfreq = 0.50; % in Hz
fmodfreq = 1.00; % in Hz
sigmag = 6.70; % rms voltage
ampgain = 1.00;
pmodfreq = 0.25;
```

```
% Time Vector
DACfreq = 40000; % In Hz
tend = 40; % in seconds
t = 0:1/DACfreq:tend;
```

A Phase

```
Astep = 0.00.*...
                (stepfun(t,1)-stepfun(t,4.00))+...
        sigmag/3.5*sin(2*pi*60*t).*...
                (stepfun(t,4.00)-stepfun(t,6.00))+...
        sigmag/2*sin(2*pi*60*t).*...
                (stepfun(t,6.00)-stepfun(t,8.00))+...
        sigmag*(sin(60*2*pi*t)).*...
                (stepfun(t,8.00)-stepfun(t,10.00));

% A AM
Aamod = sigmag/2.5*(sin(60*2*pi*t)).*...
                (stepfun(t,10.00)-stepfun(t,12.00))+...
        sigmag/2.5*(1+.2*sin(amodfreq*2*pi*t)).*(sin(60*2*pi*t)).*...
                (stepfun(t,12.00)-stepfun(t,14.00))+...
        sigmag/2.5*(1+.2*sin(amodfreq*2*pi*t)).*(sin(60*2*pi*t)).*...
                (stepfun(t,14.00)-stepfun(t,16.00))+...
        sigmag/2.5*(sin(60*2*pi*t)).*...
                (stepfun(t,16.00)-stepfun(t,18.00));

% A FM
Afmod = sigmag*sin(2*pi*t.*(60-2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,18.00)-stepfun(t,20.00))+...
        sigmag*sin(2*pi*t.*(60-2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,20.00)-stepfun(t,22.00))+...
        sigmag*(sin(60*2*pi*t)).*...
                (stepfun(t,22.00)-stepfun(t,24.00));

% A PM
Apmod = sigmag*(sin(60*2*pi*t+.2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,24.00)-stepfun(t,26.00))+...
        sigmag*(sin(60*2*pi*t+.2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,26.00)-stepfun(t,28.00))+...
        sigmag/1.5*(sin(60*2*pi*t)).*...
                (stepfun(t,28.00)-stepfun(t,30.00))+...
        0.00.*...
                (stepfun(t,30.00)-stepfun(t,tend));

Asig = Astep + Aamod + Afmod + Apmod;
```

B phase

```
Bstep = 0.00.*...
                (stepfun(t,1)-stepfun(t,4.00))+...
        sigmag/3.5*sin(2*pi*60*t-2*pi/3).*...
                (stepfun(t,4.00)-stepfun(t,6.00))+...
        sigmag/2*sin(2*pi*60*t-2*pi/3).*...
                (stepfun(t,6.00)-stepfun(t,8.00))+...
        sigmag*(sin(60*2*pi*t-2*pi/3)).*...
                (stepfun(t,8.00)-stepfun(t,10.00));

% B AM
Bamod = sigmag/2.5*(sin(60*2*pi*t-2*pi/3)).*...
                (stepfun(t,10.00)-stepfun(t,12.00))+...
        sigmag/2.5*(1+.2*sin(amodfreq*2*pi*t)).*(sin(60*2*pi*t-2*pi/3)).*...
                (stepfun(t,12.00)-stepfun(t,14.00))+...
        sigmag/2.5*(1+.2*sin(amodfreq*2*pi*t)).*(sin(60*2*pi*t-2*pi/3)).*...
                (stepfun(t,14.00)-stepfun(t,16.00))+...
        sigmag/2.5*(sin(60*2*pi*t-2*pi/3)).*...
                (stepfun(t,16.00)-stepfun(t,18.00));


% B FM
Bfmod = sigmag*sin(2*pi*t.*(60-2*sin(fmodfreq*2*pi*t))-2*pi/3).*...
                (stepfun(t,18.00)-stepfun(t,20.00))+...
        sigmag*sin(2*pi*t.*(60-2*sin(fmodfreq*2*pi*t))-2*pi/3).*...
                (stepfun(t,20.00)-stepfun(t,22.00))+...
        sigmag*(sin(60*2*pi*t-2*pi/3)).*...
                (stepfun(t,22.00)-stepfun(t,24.00));
% B PM
Bpmod = sigmag*(sin(60*2*pi*t-2*pi/3+.2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,24.00)-stepfun(t,26.00))+...
        sigmag*(sin(60*2*pi*t-2*pi/3+.2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,26.00)-stepfun(t,28.00))+...
        sigmag/1.5*(sin(60*2*pi*t-2*pi/3)).*...
                (stepfun(t,28.00)-stepfun(t,30.00))+...
        0.00.*...
                (stepfun(t,30.00)-stepfun(t,tend));

Bsig = Bstep + Bamod + Bfmod + Bpmod;
```

C Phase

```
Cstep = 0.00.*...
                (stepfun(t,1)-stepfun(t,4.00))+...
        sigmag/3.5*sin(2*pi*60*t+2*pi/3).*...
                (stepfun(t,4.00)-stepfun(t,6.00))+...
        sigmag/2*sin(2*pi*60*t+2*pi/3).*...
                (stepfun(t,6.00)-stepfun(t,8.00))+...
        sigmag*(sin(60*2*pi*t+2*pi/3)).*...
                (stepfun(t,8.00)-stepfun(t,10.00));
```

```
% C AM
Camod = sigmag/2.5*(sin(60*2*pi*t+2*pi/3)).*...
                (stepfun(t,10.00)-stepfun(t,12.00))+...
        sigmag/2.5*(1+.2*sin(amodfreq*2*pi*t)).*(sin(60*2*pi*t+2*pi/3)).*...
                (stepfun(t,12.00)-stepfun(t,14.00))+...
        sigmag/2.5*(1+.2*sin(amodfreq*2*pi*t)).*(sin(60*2*pi*t+2*pi/3)).*...
                (stepfun(t,14.00)-stepfun(t,16.00))+...
        sigmag/2.5*(sin(60*2*pi*t+2*pi/3)).*...
                (stepfun(t,16.00)-stepfun(t,18.00));

% C FM
Cfmod = sigmag*sin(2*pi*t.*(60-2*sin(fmodfreq*2*pi*t))+2*pi/3).*...
                (stepfun(t,18.00)-stepfun(t,20.00))+...
        sigmag*sin(2*pi*t.*(60-2*sin(fmodfreq*2*pi*t))+2*pi/3).*...
                (stepfun(t,20.00)-stepfun(t,22.00))+...
        sigmag*(sin(60*2*pi*t+2*pi/3)).*...
                (stepfun(t,22.00)-stepfun(t,24.00));

% C PM
Cpmod = sigmag*(sin(60*2*pi*t+2*pi/3+.2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,24.00)-stepfun(t,26.00))+...
        sigmag*(sin(60*2*pi*t+2*pi/3+.2*sin(fmodfreq*2*pi*t))).*...
                (stepfun(t,26.00)-stepfun(t,28.00))+...
        sigmag/1.5*(sin(60*2*pi*t+2*pi/3)).*...
                (stepfun(t,28.00)-stepfun(t,30.00))+...
        0.00.*...
                (stepfun(t,30.00)-stepfun(t,tend));


Csig = Cstep + Camod + Cfmod + Cpmod;
```

Setup Trigger

```
Trig = 0.00.*...
                (stepfun(t,0.00)-stepfun(t,4.00))+...
       6.50.*...
                (stepfun(t,4.00)-stepfun(t,6.00))+...
       0.00.*...
                (stepfun(t,6.00)-stepfun(t,8.00))+...
       6.50.*...
                (stepfun(t,8.00)-stepfun(t,10.00))+...
       0.00.*...
                (stepfun(t,10.00)-stepfun(t,12.00))+...
       6.50.*...
                (stepfun(t,12.00)-stepfun(t,14.00))+...
       0.00.*...
                (stepfun(t,14.00)-stepfun(t,16.00))+...
       6.50.*...
                (stepfun(t,16.00)-stepfun(t,18.00))+...
       0.00.*...
                (stepfun(t,18.00)-stepfun(t,20.00))+...
       6.50.*...
```

```
                (stepfun(t,20.00)-stepfun(t,22.00))+...
        0.00.*...
                (stepfun(t,22.00)-stepfun(t,24.00))+...
        6.50.*...
                (stepfun(t,24.00)-stepfun(t,26.00))+...
        0.00.*...
                (stepfun(t,26.00)-stepfun(t,28.00))+...
        0.00.*...
                (stepfun(t,28.00)-stepfun(t,30.00));

% Plot checking signals before being sent to Relay/PMU
% figure
% plot(t,Asig,'r')
% hold on
% plot(t,Bsig,'g')
% plot(t,Csig,'b')
% plot(t,Trig,'k')
% legend('Asig','Bsig','Csig','Trig')
% grid
```

DAQ Communication

Setup Session, Add Channels and Configure Parameters.

```
% Create the data acquisition session.
% cDAQ = daq.createSession('ni')
% Dataout = [Asig' Bsig' Csig' Trig'];
% % Create analog output channel on board ID 'cDAQ1Mod1', channel # 'ao0', with signal type
'Voltage');
% cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao0', 'Voltage');
% cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao1', 'Voltage');
% cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao2', 'Voltage');
% cDAQ.addAnalogOutputChannel('cDAQ1Mod1','ao3', 'Voltage');
%
% cDAQ.Rate = DACfreq; % Refresh rate of DAQ [Hz].
%
% queueOutputData(cDAQ,dataOut);
% startForeground(cDAQ);
%
% % Clean up and release hardware.
%
% cDAQ.release();
% delete(cDAQ);
% clear cDAQ;
% toc


% End
```

Appendix E: Baseline .csv Reader

Baseline .csv Reader

Takes two sets of data and compares the two side by side. An established baseline data set can be compared against a new baseline test using the signal generation code provided in the report and then loaded using the following code.

```
close all;
clear all;
clc;
```

Established Phasor Baseline Data Set

```
fileID = fopen('PhasorStep_AM_FM_PM_0003.csv');
CP1 = textscan(fileID,...
               '%f%c%f:%f:%f%f%c%f:%f:%f %f %f %f %f %f %f %f %f %f ',...
               'Headerlines',1,'Delimiter',',');
fclose(fileID);

% The Data is now parsed and seperated out in large columns that are
% contained in cells.  To make this data more accesable and easier to know
% exactly what the data represents it will be removed from the cells to be
% stand alone vectors that are named accordingly.

% Define PMU Time Stamp Variables
PMUtStampYMD1 = CP1{1};
PMUtStampHour1 = CP1{3};
PMUtStampMin1 = CP1{4};
PMUtStampSec1 = CP1{5};

% Define c37 Time Stamp Variables
c37tStampYMD1 = CP1{6};
c37tStampHour1 = CP1{8};
c37tStampMin1 = CP1{9};
c37tStampSec1 = CP1{10};

% Define Voltage Phase Varables
posSeqVol1 = CP1{11};
posSeqAng1 = CP1{12};
aVol1 = CP1{13};
aAng1 = CP1{14};
bVol1 = CP1{15};
bAng1 = CP1{16};
cVol1 = CP1{17};
cAng1 = CP1{18};
freq1 = CP1{19};

% This section is used to make sure the plots start relative to time zero.
zeroPMUPhasorTime1 = (PMUtStampHour1(1) * 60 * 60) +...
                     (PMUtStampMin1(1) * 60) + (PMUtStampSec1(1));
```

```
zeroC37PhasorTime1 = (c37tStampHour1(1) * 60 * 60) +...
                     (c37tStampMin1(1) * 60) + (c37tStampSec1(1));

% New time vectors.  Takes the time vectors from the hour, min, and seconds
% and then converts all to seconds.  Then we shift the scale to start at
% zero.
PMUtimeVector1 = (PMUtStampHour1 * 60 * 60) + (PMUtStampMin1 * 60) +...
                  PMUtStampSec1 - zeroPMUPhasorTime1;
c37TimeVector1 = (c37tStampHour1 * 60 * 60) + (c37tStampMin1 * 60) +...
                  c37tStampSec1 - zeroPMUPhasorTime1;
```

New Phasor Data for Baseline Comparison

```
fileID = fopen('PhasorStep_AM_FM_PM_0008.csv');
CP2 = textscan(fileID,...
              '%f%c%f:%f:%f%f%c%f:%f:%f %f %f %f %f %f %f %f %f ',...
              'Headerlines',1,'Delimiter',',');
fclose(fileID);
% Define PMU Time Stamp Variables
PMUtStampYMD2 = CP2{1};
PMUtStampHour2 = CP2{3};
PMUtStampMin2 = CP2{4};
PMUtStampSec2 = CP2{5};
% Define c37 Time Stamp Variables
c37tStampYMD2 = CP2{6};
c37tStampHour2 = CP2{8};
c37tStampMin2 = CP2{9};
c37tStampSec2 = CP2{10};

% Define Voltage Phase Varables
posSeqVol2 = CP2{11};
posSeqAng2 = CP2{12};
aVol2 = CP2{13};
aAng2 = CP2{14};
bVol2 = CP2{15};
bAng2 = CP2{16};
cVol2 = CP2{17};
cAng2 = CP2{18};
freq2 = CP2{19};

% This section is used to make sure the plots start at time zero
zeroPMUPhasorTime2 = (PMUtStampHour2(1) * 60 * 60) +...
                     (PMUtStampMin2(1) * 60) + (PMUtStampSec2(1));

zeroC37PhasorTime2 = (c37tStampHour2(1) * 60 * 60) +...
                     (c37tStampMin2(1) * 60) + (c37tStampSec2(1));

% New time vectors.  Takes the time vectors from the hour, min, and seconds
% and then converts all to seconds.  Then we shift the scale to start at
% zero.
PMUtimeVector2 = (PMUtStampHour2 * 60 * 60) + (PMUtStampMin2 * 60) +...
```

```
                PMUtStampSec2 - zeroPMUPhasorTime2;
c37TimeVector2 = (c37tStampHour2 * 60 * 60) + (c37tStampMin2 * 60) +...
                c37tStampSec2 - zeroPMUPhasorTime2;
```

Established Analog Baseline Data Set

See the above comment for a description on the textscan function

```
fileID = fopen('AnalogStep_AM_FM_PM_0003.csv');
CA1 = textscan(fileID,'%f%c%f:%f:%f %f %f %f %f ','Headerlines',1,...
              'Delimiter',',');
fclose(fileID);

% As described above this section of the code is pulling the data columns
% out of cells and naming them for easier access and use.

AnalogtStampHour1 = CA1{3};
AnalogtStampMin1 = CA1{4};
AnalogtStampSec1 = CA1{5};

% This section is used to make sure the plots start at time zero
zeroAnalogTime1 = (AnalogtStampHour1(1) * 60 * 60) +...
                (AnalogtStampMin1(1) * 60) + AnalogtStampSec1(1);

% New time vectors.  Takes the time vectors from the hour, min, and seconds
% and then converts all to seconds.  Then we shift the scale to start at
% zero.
AnalogTimeVector1 = (AnalogtStampHour1 * 60 * 60) +...
                    (AnalogtStampMin1 * 60) + AnalogtStampSec1 -...
                    zeroPMUPhasorTime1;

% Give variable names to the 3 phases
Aphase1 = CA1{6};
Bphase1 = CA1{7};
Cphase1 = CA1{8};
```

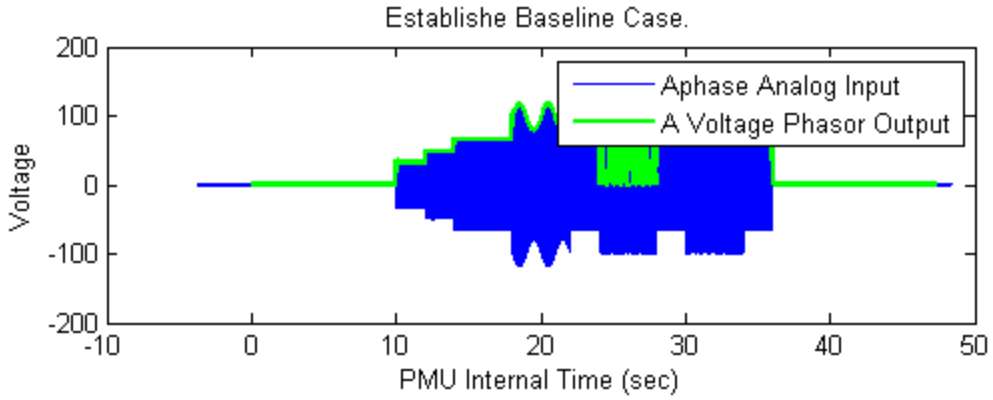
New Analog Data for Baseline Comparison

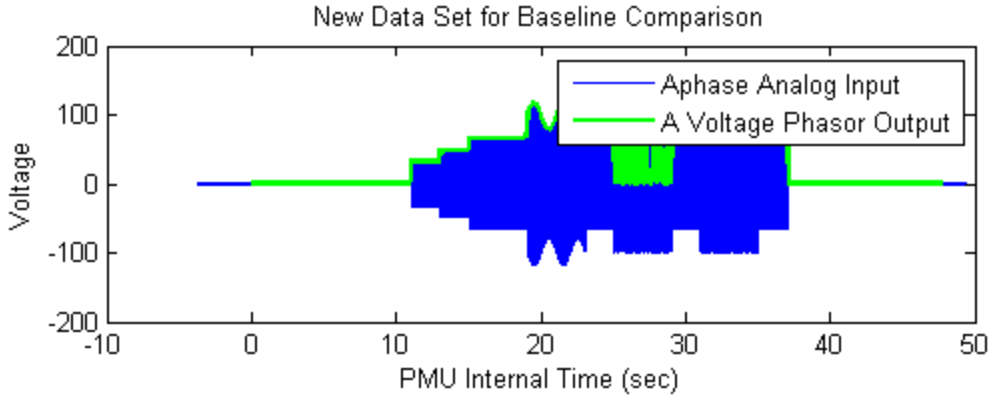
```
fileID = fopen('AnalogStep_AM_FM_PM_0008.csv');
CA2 = textscan(fileID,'%f%c%f:%f:%f %f %f %f %f ','Headerlines',1,...
              'Delimiter',',');
fclose(fileID);

% As described above this section of the code is pulling the data columns
% out of cells and naming them for easier access and use.

AnalogtStampHour2 = CA2{3};
AnalogtStampMin2 = CA2{4};
AnalogtStampSec2 = CA2{5};

% This section is used to make sure the plots start at time zero
zeroAnalogTime2 = (AnalogtStampHour2(1) * 60 * 60) +...
```

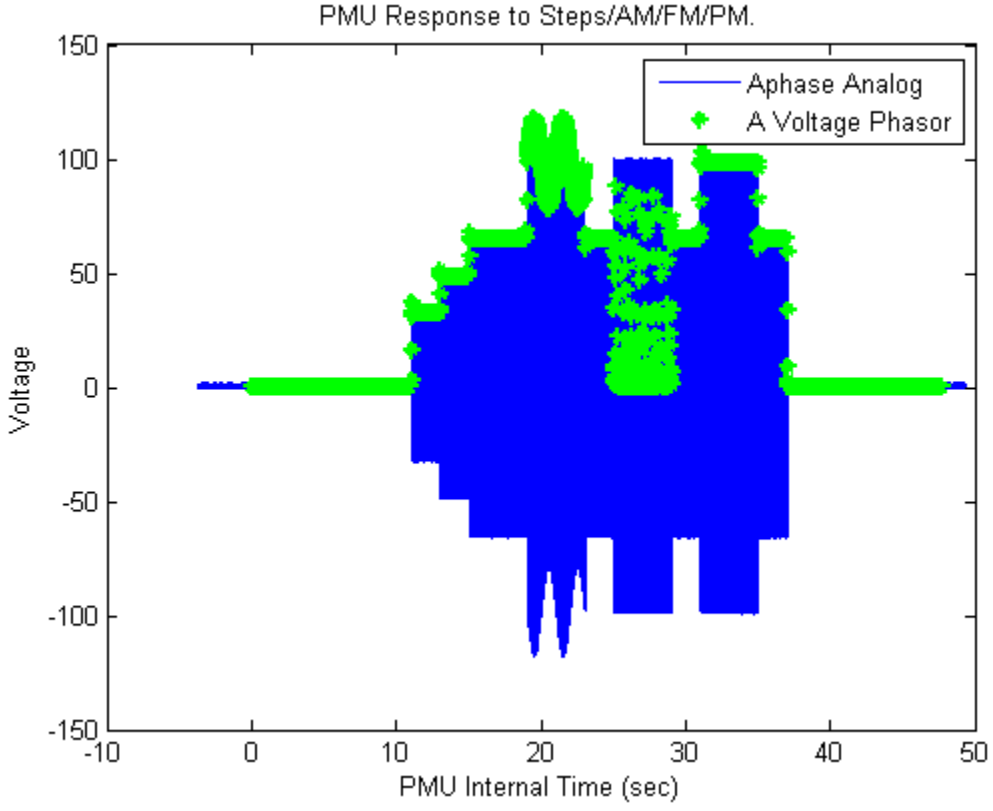
```
                (AnalogtStampMin2(1) * 60) + AnalogtStampSec2(1);

% New time vectors.  Takes the time vectors from the hour, min, and seconds
% and then converts all to seconds.  Then we shift the scale to start at
% zero.
AnalogTimeVector2 = (AnalogtStampHour2 * 60 * 60) +...
                    (AnalogtStampMin2 * 60) + AnalogtStampSec2 -...
                     zeroPMUPhasorTime2;

% Give variable names to the 3 phases
Aphase2 = CA2{6};
Bphase2 = CA2{7};
Cphase2 = CA2{8};
```

Plotting For Comparison

```
figure
subplot(211)
plot(AnalogTimeVector1,Aphase1/sqrt(2),'b')
hold on
plot(PMUtimeVector1,aVol1,'LineWidth',2,'Color',[0 1 0])
legend('Aphase Analog Input','A Voltage Phasor Output')
title('Establishe Baseline Case.')
xlabel('PMU Internal Time (sec)')
ylabel('Voltage')
subplot(212)
plot(AnalogTimeVector2,Aphase2/sqrt(2),'b')
hold on
plot(PMUtimeVector2,aVol2,'LineWidth',2,'Color',[0 1 0])
legend('Aphase Analog Input','A Voltage Phasor Output')
title('New Data Set for Baseline Comparison')
xlabel('PMU Internal Time (sec)')
ylabel('Voltage')

figure
plot(AnalogTimeVector2,Aphase2/sqrt(2),'b')
hold on
plot(PMUtimeVector2,aVol2,'*','LineWidth',2,'Color',[0 1 0])
legend('Aphase Analog','A Voltage Phasor')
title('PMU Response to Steps/AM/FM/PM.')
xlabel('PMU Internal Time (sec)')
ylabel('Voltage')

%end
```


Establishe Baseline Case.
Aphase Analog Input
A Voltage Phasor Output
Voltage
PMU Internal Time (sec)


New Data Set for Baseline Comparison
Aphase Analog Input
A Voltage Phasor Output
Voltage
PMU Internal Time (sec)


PMU Response to Steps/AM/FM/PM.
Aphase Analog
A Voltage Phasor
Voltage
PMU Internal Time (sec)

Appendix F: Math Document

Math Doc for PMU Testing Project **Version: 4**
Date: 2/20/16

Terms

$A_0 = DC\ offset$
$A_1 = Amplitude\ of\ the\ fundamental$
$A_i\ for\ i > 1\ are\ harmonic\ terms$
$\varphi = phase\ angle$
$n_m = measurement\ noisse$
$t = time$
$\Delta f_r =\ is\ a\ constant$ Frequency for phase ramp term
$u(t - t_r) = unit\ step\ function\ switching\ at\ time\ t_r$
$\hat{f}(t) = estimated\ frequency\ calculated\ from\ measurement\ of\ v(t)$
$R_f = Ramp\ rate\ in \frac{Hz}{s}$
$R_f = Equivalent\ Effective\ Ramp\ Rate$
$f_m = frequency\ for\ phase\ modulation\ term$

Below is a list of all the equations as they appear in "*Frequency Estimation for Inter-Area Oscillation Feedback Damping Control*" and how they will appear in testing form.

Equation 1 Paper Form

Point on Wave (POW) model. This is the fundamental form for our test signals which will be translated to three phase. The equation in its current form would represent the A phase. The B and C phases will need to be shifted in phase by 2π/3 rad. For our current studies we are neglecting the harmonic and noise part of the signal contained in the summation.

$$v(t) = A_0 + A_1(t)\cos\big(2\pi 60t + \varphi(t)\big) + \sum_{i=2}^{\infty} A_i \cos(i2\pi 60t + i\varphi(t) + \varphi_{0i}) + n_m(t)$$

Equation 1 Testing Form

$$Va(t) = [A_{offset} + A_1(t)\cos(2\pi f_o t + \varphi(t)]\,[u(t - t_{start}) - u(t - t_{end})]$$

Equation 2a Paper Form

$$A_1(t) = A_{1,0}(1 + k_x \cos(2\pi f_m t))$$

Equation 2a Testing Form

$$A_1(t) = A_{1,0}\{1 + k_x \cos(2\pi f_m t)\left[u(t - t_{AmodOn}) - u\big(t - t_{AmodOff}\big)\right] + \cdots$$
$$\ldots + A_{step}[u\big(t - t_{AmpStepOn}\big) - u\big(t - t_{AmpStepOff}\big)]\}$$

Equation 2b Paper Form

$$\varphi(t) = \varphi_0 + 2\pi\Delta f_r(t - t_r)u(t - t_r) + k_a \cos(2\pi f_m t)$$

Equation 2b Testing Form

$$\varphi(t) = \varphi_{offset} + 2\pi\Delta f_r\left(t - t_{PhaseRampOn}\right)\left[u\left(t - t_{PhaseRampOn}\right) - u\left(t - t_{PhaseRampOff}\right)\right] + \cdots$$
$$\ldots + k_a\cos(2\pi f_m t)[u(t - t_{PhaseModOn}) - u\left(t - t_{PhaseModOff}\right)]$$

Equation 3

Point on wave instantaneous frequency

$$f(t) = 60 + \left(\frac{1}{2\pi}\right)\frac{d\varphi(t)}{dt}\ Hz$$

Equation 4

Actual Phase Angle, for frequency ramp test.

$$\varphi(t) = \pi R_f t^2\ rad$$

Equation 5

Estimated Phase Angle, for frequency ramp test.

$$\hat{\varphi}(t) = \pi\widehat{R_f} t^2\ rad$$

Equation 6

Transfer Function

$$G(f_m) = \frac{F\{\hat{f}(t)\}}{F\{f(t)\}}$$