**IMON Calibration Software User Guide**

[v-stpurc@microsoft.com](mailto:v-stpurc@microsoft.com)

**1.0 Overview:**

The IMON calibration software is written in Python 2 in the Spider IDE. The application is a PyQt dialog box that spawns a background thread that handles the actual work load.

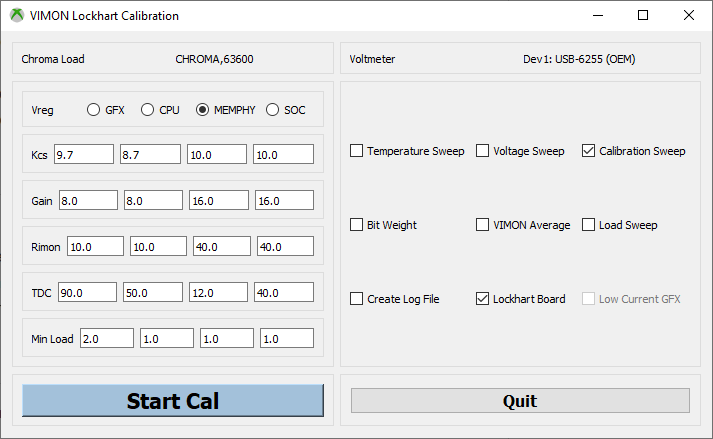


Figure IMON Calibration Dialog Box

The IMON calibration dialog box allows the user to choose various functions depending on their needs. It does however step through a typical calibration sequence automatically when used without making changes.

At the top of the dialog box the two instruments required are shown: the Chroma load and the voltmeter. If the instruments do not appear as shown then the application will not run. Instrument setup details will not be covered in this document.

The Vreg radio buttons indicate which of the voltage rails will be calibrated. Below that are the default values that will be used in the calibration. The complete details of calibration will not be covered in this document. The user should refer to AMD documentation to understand calibration fully.

In addition to default values the user can modify the values used in a calibration should that be needed for special circumstances.

In the right hand pane are the basic functions of the application that are available to the user.

* Temperature Sweep is a function used to capture IMON variation over temperature at a given load. This functionality typically requires code modifications for a specific situation and is not used during calibration.
* Voltage Sweep is a standard part of a calibration routine and is used to determine the error delta over voltage changes.
* Calibration Sweep does the actual setting of the gain and offset registers. It will set the load and measure VIMON adjusting the register values as necessary.
* Bit Weight is a function designed to determine the LSB step size across the range of the offset register. This is not normally used during calibration.
* VIMON Average is used to determine the long term stability and total variation to be expected from VIMON at a fixed load. This also is not typically used during calibration.
* Load Sweep is the final step in a calibration when accuracy across load is determined.
* Create Log File creates and separate text file with the log information. Log information is included in the output spreadsheets by default.
* LockHart Board selects between LockHart and Anaconda boards.
* Low Current GFX is a stand alone IMON setting used only on Anaconda boards.

**1.1 Process:**

The IMON calibration software is automated to some degree in that the sequence of Vregs and steps on each happens automatically as the user proceeds through the process. Where user input is required is in setting up the board, moving the load cables from rail to rail, and starting each phase of the test.

A five bay Chroma main frame electronic load is required for calibration. A two bay load could be used for the LockHart board but would require the code to be modified to support it.

Five 63640-80-80 modules are necessary for calibrating the Anaconda board. A set of cables as heavy and as short as possible is required, as is an Ironwood interposer and interposer board. The photo below shows an Anaconda board with GFX hooked up and ready for calibration.

The USB cable needs to be connected as does the SCSII cable from the NI USB-6255 to the DAP connector.

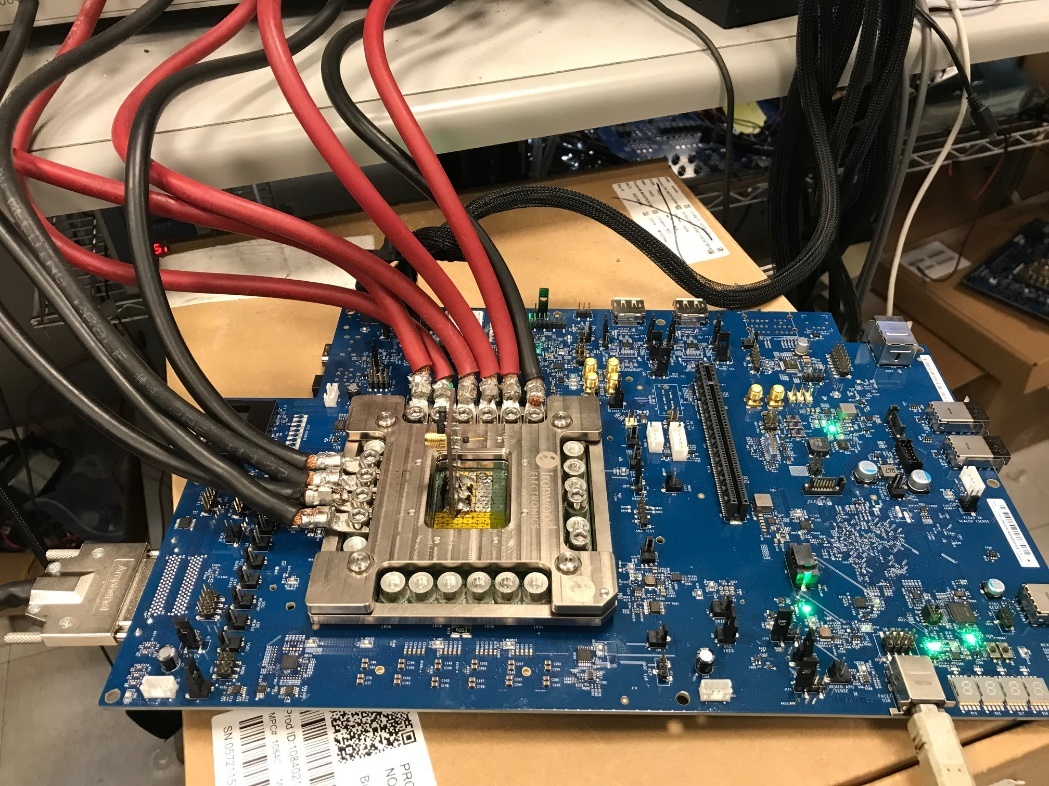


Figure Anaconda board with GFX connected

The calibration proceeds in the following order: MEMPHY, SOC, CPU, GFX. This is to put the GFX load sweep at the end of the process to minimize heating. For each rail the steps are as follows: Calibration Sweep, Voltage Sweep, Load Sweep.

This process is followed automatically by pressing the Start Cal button and waiting for the each step to complete.

When the user presses ‘Start Cal’ for the first time the DUT Info dialog box pops up. Here the user can enter the Config ID, PCBA Serial #, Product Serial #, and Run Notes. This information is included in the output results spreadsheet.

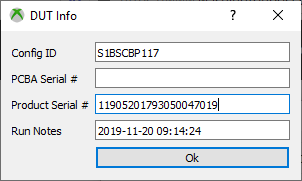


Figure DUT Info Dialog Box

After clicking Ok here the Start Cal button changes to the Abort Cal button and the calibration routing for the first rail: MEMPHY typically, begins.

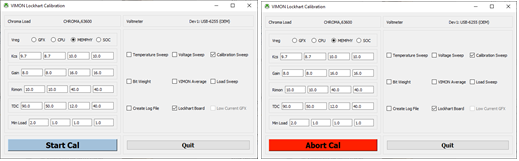


Figure Start Cal state to Abort Cal state

At his point the user can follow the progress in the Spider IDE Ipython console window. Any problems or issues with the calibration can be determined by becoming familiar with the output from the code that displays in this window.

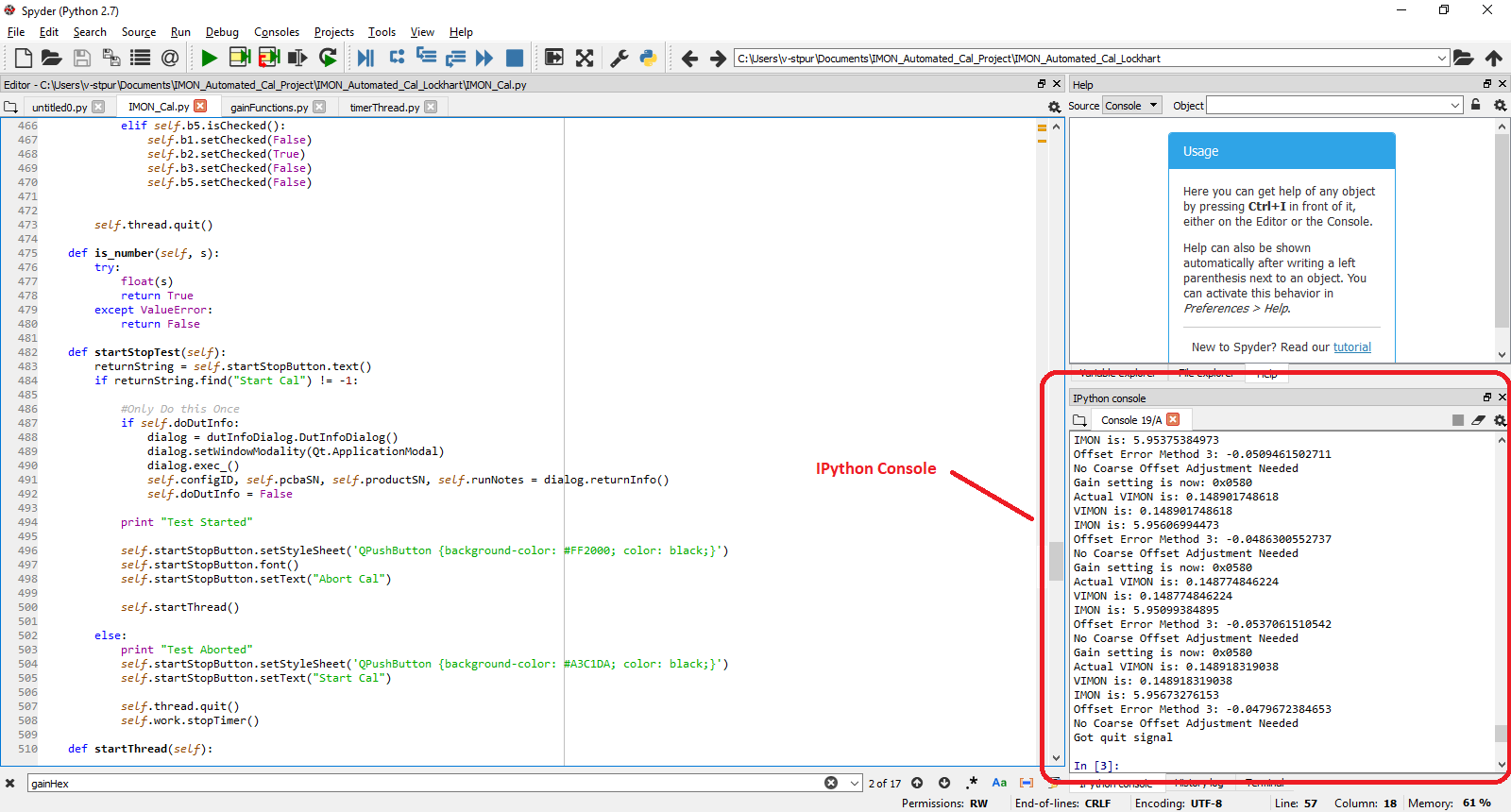


Figure Spider IDE showing IPython console window

If the calibration is successful the Abort Cal button will change back to the Start Cal button and the check box selection will change to Voltage Sweep. If considerable time elapses and the calibration step does not complete then the user should check that the cabling is done correctly and the DAP connector is plugged in.

After the Voltage Sweep completes the check box for Load Sweep will be selected. At each step the user must press the Start Cal button to move on. After the three steps for each rail the next rail is selected and the user must move the cables to the new rail.

At the end of the process the check box for the starting Vreg, MEMPHY will again be selected. At this point the user should press the Quit button and the application will complete.

**2.0 Code Structure Details:**

The basic structure of the IMON Calibration code is based on a model that originated with the PyQt GUI for the load slammer board.

This idea is that a dialog box GUI is the main control interface for the application and that this will then spawn a thread that runs in the background. Control is via messages sent between the main GUI and the worker thread.

**2.1 Timer Thread:**

The reason for this structure is so that the user interface doesn’t appear to lock as tasks are completed. The class for the worker thread is called TimerThread to invoke the idea that a loop governed by a timer could be implemented this way and mimic a real time system. This is similar in concept to an Arduino where the user puts all their code in the ‘loop’ function.

In our case the loop function is called ‘run’ in the body of the TimerThread class. We also have considerable functionality happening outside the run function. However, the logic of how the IMON cal is done resides in the ‘run’ function.

In the case of the IMON Calibration software each time the ‘Start Cal’ button is pressed a new thread is spawned. When that thread completes it’s duties and the button changes back to the ‘Start Cal’ state that thread is killed.

Each Vreg has a set of global variables in the timerThread.py file scope. As each thread completes it’s functionality these global variable are updated. When the entire sequence is completed each of these should have meaningful values. In that case when the thread is stopped for the last time critical values in FLASH will be read out and compared to values in the global variables to verify that everything is correct.

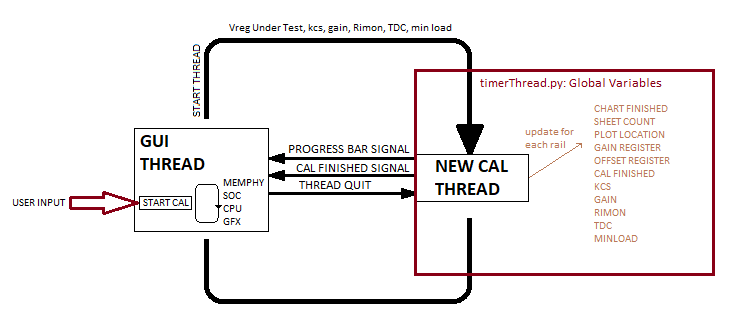


Figure Gui Thread spawns TimerThread objects

The TimerThread class uses several helper classes. These are initialized in the timer thread \_init function. Some of these classes import timerThread.py to get access to it’s global variables.

**2.2 Gain Functions:**

The actual functionality of calibration occurs in functions that are members of the **GainFunctions** class. There are three primary functions in this class: **gainLoopMPS, offsetLoopMPS,** and **imonCheck**.

Both the what is called the gain loop and the offset loop are modeled on formulas in a spreadsheet provided by MPS as an IMON calibration calculator.

Calibration is done in three steps, once through the offset loop, once through the gain loop, then back through the offset loop.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Step 1 | C5h=0x0800 ---> add +10uA and gain=0% | | | | | | | | |
| Step 2 | measure the VIMON voltage for two Iout current. 0A and TDC | | | | | | | | |
|  | Iout | VIMON (mV) |  | gain |  | gain error | gain adj |  |  |
|  | 0 | 30 |  | 2.7 | mV/A | 1.65% | -1.65% |  |  |
|  | 190 | 543 |  |  |  |  |  |  |  |
| Step 3 | adjust gain --> new C5h (C5h=0x0822) in this example, adjust gain for -1.6% | | | | | | | | |
| Step 4 | measure the VIMON voltage @ 0A | | | | | | | | |
|  | Iout | VIMON (mV) |  |  |  | offset error (uA) | offset adj (uA) |  |  |
|  | 0 | 29.5 |  |  |  | 0.5875 | -0.5875 |  |  |
| Step 5 | adjust offset --> new C5h (C5h=07A2) in this example, adjust offset for -0.625uA | | | | | | | | |
| Step 6 | measure the VIMON voltage for two Iout current. 0A and TDC | | | | | | | | |
|  | Iout | VIMON (mV) | abs error (A) | gain |  | gain error | gain adj |  |  |
|  | 0 | 26.7 | 0.052 | 2.65842105 | mV/A | 0.08% | -0.08% | no adjust if < 0.5% |  |
|  | 190 | 531.8 | 0.207 |  |  | offset error (uA) | offset adj (uA) |  |  |
|  |  |  |  |  |  | 0.0275 | -0.0275 | no adjust if < 0.2uA |  |
| Step 7 | adjust the gain/offset for +/-LSB if needed and final | | | |  |  |  |  |  |

**2.2.1 offsetLoopMPS**

The first thing in the offset routine is to set it to a fixed large value, 0x0800 for GFX/  
CPU and 0x0500 for MEMPHY/SOC. This is to improve accuracy at low loads because at low loads the shift from the 1.2V reference of the intelliphase chips to the ground reference of the IMON circuit results in the loss of intelliphase signals that go below the 1.2V reference. By add the fixed offset this signal is entirely above ground.

The offset error is to be adjusted until it is within 1LSB of the target. However the LSB size has been found to be variable and over time the tolerance has been opened up to accommodate this. At this time the error targets are as follows: GFX = 0.33 uA, CPU = 0.29uA, SOC = 0.23uA, and MEMPHY = 0.21uA.

The symptom for needing the error target to be opened up is that the offset loop will never complete and the bit can be seen to being repeatedly set then cleared with the error switching between positive and negative values that are greater than the error target.

The code for the offset loop is relatively simple and is diagramed below:

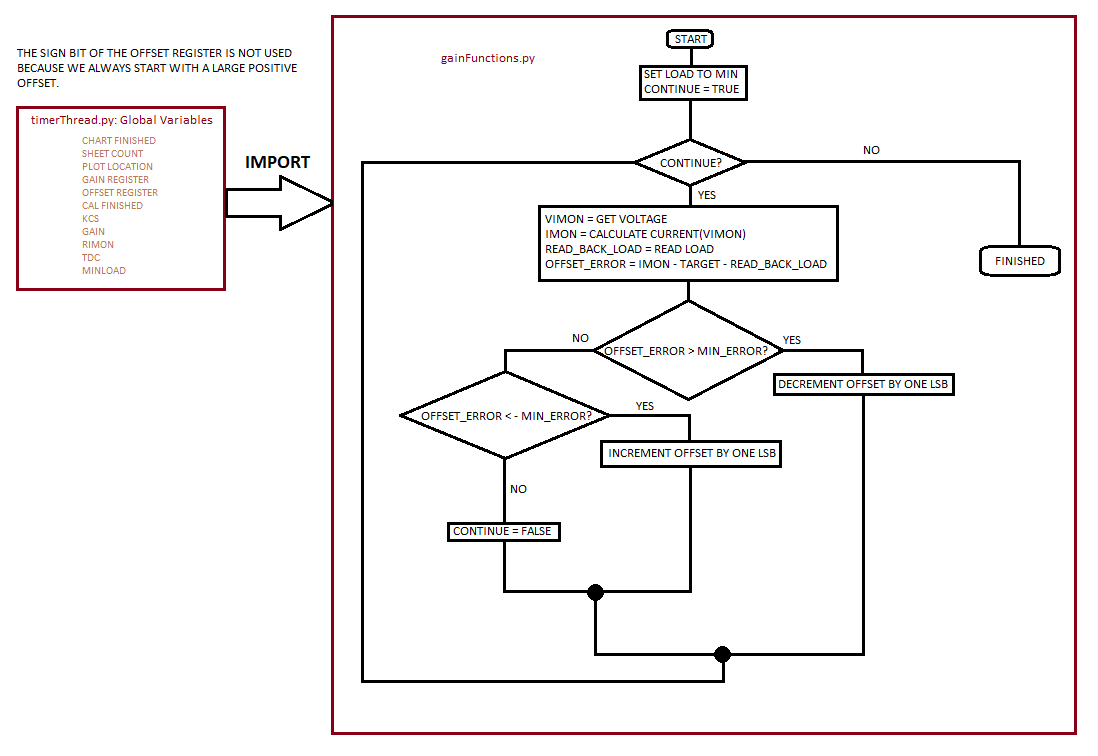


Figure Offset Loop Diagram

**2.2.2 gainLoopMPS**

The gain portion of the register setting starts at zero and is unmodified by the offset portion of the code. The gain loop is slightly more complex than the offset loop due to the need for a sign bit to be set in some cases.

The gain loop works similarly to the offset loop except the target is a percentage of the ideal gain rather than a current. It is diagrammed below:

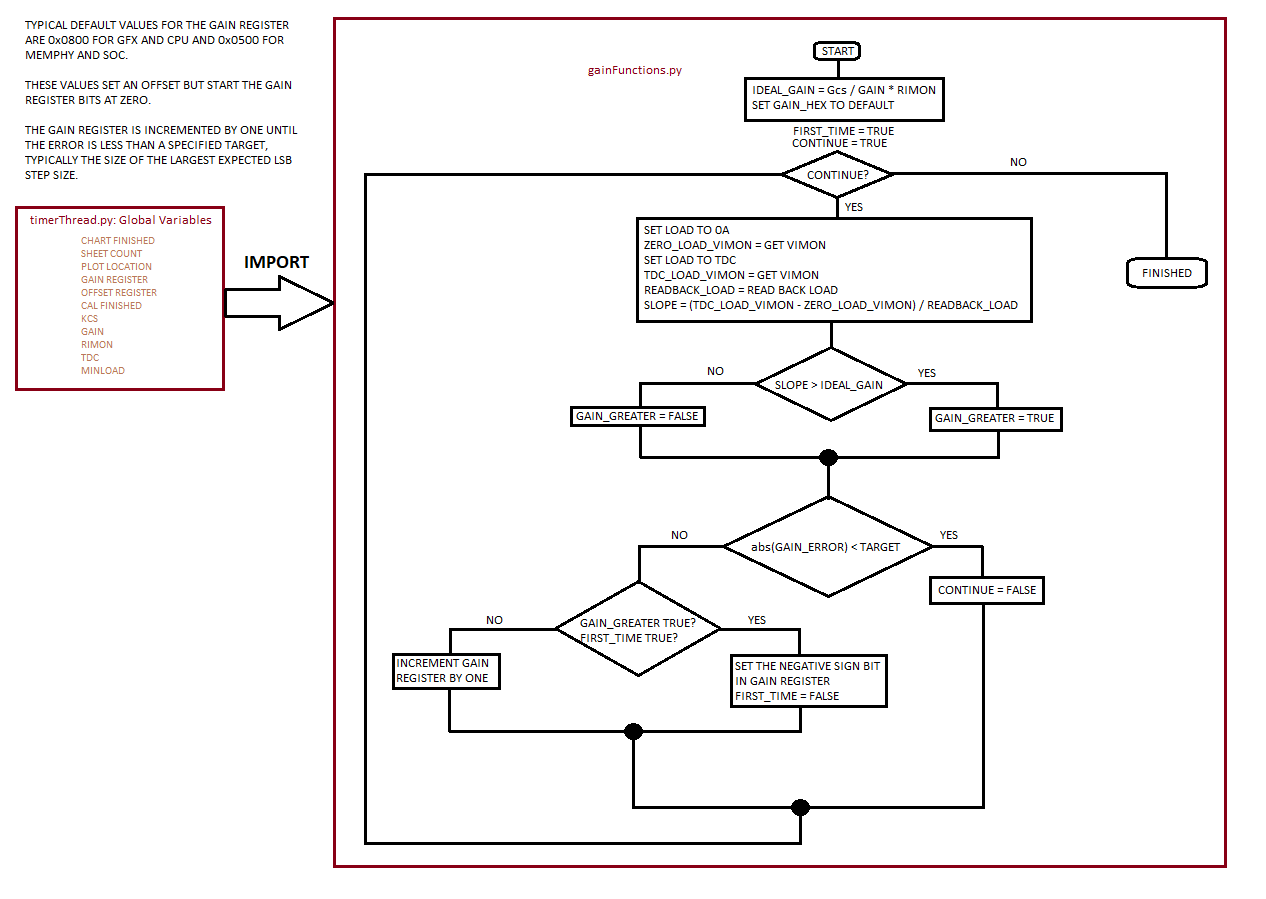
****

Figure Gain Loop Diagram

**2.2.3 imonCheck**

The imonCheck function is relatively straight forward and is not diagramed here. It’s basic functionality is to step the Chroma load and read back and compare this load to that of the IMON pin accounting for the fixed offset.

This information is used to create a plot of the accuracy over load and the accuracy reported on the summary page of the results is from this data at minimum load.

**2.3 Get Voltage:**

The voltmeter used for IMON calibration is the National Instruments USB-6255. This instrument does not use the VISA interface but rather a custom API provided by National Instruments.

Python for this interface is provided by NI\_DAQmx Python API. Basic usage can be found at:

<https://nidaqmx-python.readthedocs.io/en/latest/>

The relevant C API reference is part of the NI software package and can be found here:

http://zone.ni.com/reference/en-XX/help/370471AM-01/cdaqmx/help\_file\_title/

The USB-6255 has a 100mil pitch headers and needs to be adapted to the SCSII DAP connector on the LockHart and Anaconda boards.

This adaptation is done by modifying an off the shelf inteposer board designed to adapt SCSII cable to 100 mil headers.

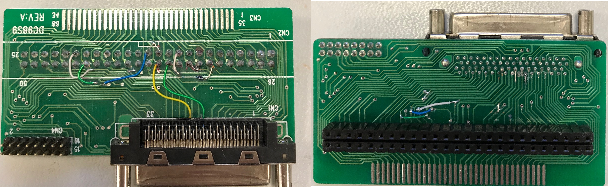


Figure DAP to USB-6255 Interposer

The USB-6255 is a high speed converter and the signal on the VIMON pin contains high frequency signals so averaging is done to the VIMON signal.

The DAP connector provides both the VIMON signals and the Vout values. The channels are set up with the following (simplified) code. For a different interposer board signals could be routed differently.

#VIMON

self.taskGFX.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai67", max\_val=2, min\_val=-2, DIFFERENTIAL)

self.taskCPU.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai66", max\_val=2, min\_val=-2, DIFFERENTIAL)

self.taskMEMIO.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai64", max\_val=1, min\_val=-1, DIFFERENTIAL)

self.taskMEMPHY.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai65", max\_val=2, min\_val=-2, DIFFERENTIAL)

self.taskSOC.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai53", max\_val=2, min\_val=-2, DIFFERENTIAL)

#V Sense

self.taskGFX\_Sense.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai52", max\_val=2, min\_val=-2, DIFFERENTIAL)

self.taskCPU\_Sense.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai51", max\_val=2, min\_val=-2, DIFFERENTIAL)

self.taskMEMPHY\_Sense.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai49", max\_val=2, min\_val=-2, DIFFERENTIAL)

self.taskSOC\_Sense.ai\_channels.add\_ai\_voltage\_chan(self.niDeviceName + "/ai50", max\_val=2, min\_val=-2, DIFFERENTIAL)

Things to notice here are the max and min values. The user needs to ensure that at the maximum load the range will cover the expected values. Also note these are differential signals even though the signal being measured is ground referenced.

**2.3.1 getVoltage:**

To properly implement the NI instrument it’s best to have the NI-DAQmc C Reference Help software installed. This provides the functional details that the python oriented documentation lacks. To find the relevant documentation it’s necessary to map as best you can the name in the python API to a name in the C API. For example we want to understand the channel setup done in the init section of Get Voltage.

There we see the following line:

self.taskGFX\_Sense.timing.cfg\_samp\_clk\_timing(100000, "", Edge.RISING, AcquisitionType.FINITE, 50000)

We look through the C help and we find: DAQmxCfgSampClkTiming which agrees closely with our python name. Opening that we find the function prototype that helps us interpret the argument to the python function.

int32 DAQmxCfgSampClkTiming (TaskHandle taskHandle, const char source[], float64 rate, int32 activeEdge, int32 sampleMode, uInt64 sampsPerChanToAcquire);

We ignore the task handle and source. Our interest starts with the rate argument. For us that will be 100kHz, then we sample on the rising edge and only on acquisition rather than a continuous one. In our case this is not a real time acquisition so we are indifferent to buffer overruns or buffer empty situations. The final argument is the number of samples to acquire which we set at 50,000.

So we are collecting a half second of data at a time.

When we get the voltage value we send ‘getVoltage’ an integer which is the number of half second intervals to average for the result.

getVoltage loops through the number of intervals requested collecting 50,000 samples each time with the following code:

data = self.taskGFX.read(number\_of\_samples\_per\_channel=50000)

For comparisons sake with the C API we would find the function “DAQmxReadAnalogF64” to find what the arguments should be. In this case the help file give the C prototype as:

int32 DAQmxReadAnalogF64 (TaskHandle taskHandle, int32 numSampsPerChan, float64 timeout, bool32 fillMode, float64 readArray[], uInt32 arraySizeInSamps, int32 \*sampsPerChanRead, bool32 \*reserved);

Again we’re not interested in the task handle so the first argument of interest is numSampsPerChan, which we set to 50000 in the python code.

The read returns an array of 50000 numbers which we collapse into a single sum:

for sumIndex in range (0,50000-1):

dataSum = dataSum + float(data[sumIndex])

At the end of the **for** loop the average is calculated based on the number of intervals requested. This value is returned by the function.

dataFloat = dataSum / (averageRuns \* 50000)

return dataFloat

**2.4 xlsxwriter**

The IMON Calibration software uses the python library xlsxwriter to create the output spreadsheet.

xlsxwriter should be thought of as obsolete for xbox test automation since it only provides writing functionality and is superseded by the more standard openpyxl.

Help and examples for xlsxwriter are widely available online and the code in this application is based on online example usage. Creation and manipulation of the output results spreadsheet constitutes the bulk of the complexity in the IMON Calibration software.

**2.4.1 File and Sheet Creation**

The output file workbook and required sheets are created in the GUI init function. The first sheets created are: SummarySheet, PlotSheet, and LogSheet. Each of these only has one instance possible. The workbook and sheets are passed to the thread when it is spawned.

The only other manipulation of the workbook in the GUI thread is that the file is closed when the user clicks the quit button.

**2.4.2 Timer Thread Manipulation of the workbook**

timerThread.py has a number of global variable defined for use in manipulating the workbook. It has variables that keep track of which parts of a standard test are complete, a count of how many load sweep and voltage sheets there are, and cell locations for the plots.

It’s possible to run the voltage and load sweep multiple times although this functionality hasn’t been used or tested extensively.

Each time the timerThread goes through it’s init function is will be working on a specific Vreg but using common variable names. These global variables used in the calibration process are initialized in the timerThread init function.

After initializing variables timerThread sets a number of things in the summary sheet. These things are mostly redundant and this functionality should probably be moved to the GUI thread.

**2.4.2.1 stopTimer**

Charts are largely created in the GainFunctions and TemperatureFunctions classes. However they are not inserted into the workbook until the user presses the Quit button which invoked the stopTimer function. If a chart for a specific Vreg has been created it’s noted by a flag variable being set and then inserted on the plot sheet.

**2.4.2.2 run**

As the calibration proceeds gain and offset values as well as temperature are captured and entered on the summary sheet.

For both voltage and load sweeps the called function actually creates the output sheet to hold the data as well as creating the charts and the number of sheets created is saved in a global variable in the timerThread.py space.

**2.4.3 GainFunctions Manipulation of the Workbook**

The offset calibration routine captures the offset error and enters it on the summary sheet where it is used for accuracy determination.

**2.4.3.1 imonCheck**

imonCheck is where voltage and load data is captured and entered onto a new sheet and into charts to determine calibration accuracy. Only one chart is created per Vreg so this is only done for the first iteration through imonCheck. Data from subsequent load sweeps will be added to the charts created, rather than having a separate chart.

In the **for loop** that steps through the chroma load voltage data is captured and the formulas for turning that into current and for error calculations are entered. The output sheet consists mostly of formulas and references values on the summary sheet.

Since the ranges for the output values are always fixed they are hard coded and used to add the series to the charts created.

**2.4.4 TemperatureFunctions Manipulation of the Workbook**

The IMON Calibration software developed over time and grew out of a number of experiments and discoveries. For this reason the voltage sweep functionality is contained in the TemperatureFunctions class even though the original intent of this class is now rarely used.

**2.4.4.1 longTermTemp**

longTermTemp is used to determine the temperature coefficient for IMON error. It creates a new sheet called ‘averageSheet’ that wouldn’t typically be found in a calibrations results workbook.

This function requires the user to modify code to make best use of it and will not be covered further here.

**2.4.4.2 longTermVolts**

longTermVolts is where the voltage is swept at minimum load to determine the change in offset error.

Originally this function determined long term voltage variation in VIMON but that functionality was moved to the BitCharacterization class.

Charts are created in this function as well as the determination of a slope and intercept for the region below 1V output. The slope and intercept are used for voltage corrections and the values are entered on the summary sheet and subsequently referenced in the formulas on the load sweep sheets.

For this reason the voltage sweep is done after calibration but before the load sweep.

This process is done twice and the results averaged for determination of the slope and intercept values.