



# PYTHON PCB ASSEMBLY TEST TOOLKIT

## User Manual

## Legal Information

### Copyright

© 2004–2026 National Instruments Corporation. All rights reserved.

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

NI respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

## Contents

1	About This Manual.....	5
2	Overview.....	5
3	Software setup.....	5
4	Launch Python PCB Assembly Test Toolkit .....	5
5	Measurement Libraries .....	6
5.1	Structure of Library .....	10
5.2	Physical and Virtual Channels.....	10
5.2.1	Create/Modify Global Virtual Channels.....	10
5.2.2	Calibration.....	11
5.3	Execution Options.....	12
5.4	Limitations.....	13
6	Libraries .....	14
6.1	DMM Measurements.....	16
6.1.1	DC RMS Voltage Measurements.....	16
6.1.2	DC RMS Current Measurements.....	17
6.1.3	Resistance Measurement .....	18
6.1.4	Mixed Measurement.....	19
6.2	DAQ Measurements.....	20
6.2.1	Power Supply Source and Measure .....	20
6.2.2	DC-RMS Voltage Measurement.....	23
6.2.3	DC Voltage Generation.....	25
6.2.4	DC-RMS Current Measurement.....	26
6.2.5	Time Domain Measurement .....	28
6.2.6	Frequency Domain Measurement.....	30
6.2.7	Signal Voltage Generation .....	32
6.2.8	Static Digital State Measurement.....	34
6.2.9	Static Digital State Generation .....	35
6.2.10	Dynamic Digital Pattern Measurement.....	36
6.2.11	Dynamic Digital Pattern Generation .....	38
6.2.12	Digital Clock Generation .....	40
6.2.13	Digital Pulse Generation .....	42
6.2.14	Digital Frequency Measurement .....	44
6.2.15	Digital PWM Measurement.....	46
6.2.16	Digital Edge Count Measurement Using Hardware Timer.....	49
6.2.17	Digital Edge Count Measurement Using Software Timer.....	52

6.2.18	Synchronization .....	55
6.2.19	Temperature RTD Measurement.....	57
6.2.20	Temperature Thermistor Measurement .....	60
6.2.21	Temperature Thermocouple Measurement.....	64
6.3	Communications Libraries .....	67
6.4	SWITCH Measurements.....	70
6.4.1	Static Digital Path Generation .....	70
6.5	DMM SCAN Measurements .....	72
6.5.1	DMM Scan Library.....	72
6.5.2	DMM_Scan PXI Mux PXI Shunt 16V-15C Library .....	73
7	Automation Test Sequences .....	76
7.1	Execution with Simulated Hardware.....	77
8	Functional Test Demo sequences.....	77
9	Device Synchronization Example.....	80
9.1	How to achieve synchronization?.....	80
10	Developing Test Programs .....	80
10.1	Validation examples:.....	81
10.2	Automation Sequences:.....	81
10.3	FT Demo Sequence:.....	81
11	Modify nipcbatt source code and rebuild distribution package .....	83
12	Errors and Troubleshooting .....	84
13	Known Issues List.....	85
14	Related Documents.....	86
7.	References .....	87

## 1 About This Manual

This manual contains information about the Python PCB Assembly Test Toolkit, including a brief overview of each library and automation sequences, how to develop tests using libraries, limitations and troubleshooting guidelines.

## 2 Overview

Python PCB Assembly Test Toolkit is a collection of Measurement Library, Automation Sequences, Validation Examples, Functional Test Demo Sequence along with Documentation for PCB Assembly electrical functional test.

Python PCB Assembly Test Toolkit provides support for DAQmx, niDMM, niSWITCH, and compatible with PXI, PC Based DAQ, cDAQ, TestScale and high level enough to be applicable or scalable for other instruments with similar functionality later. Libraries are built in Python, designed, and structured to use readily with sequencers. Example test sequences built with the libraries are added to demonstrate basic Functional test in Python.

## 3 Software setup

Please refer to the **Getting Started Guide** for installation and setup procedures of Python PCB Assembly Test Toolkit. In the downloaded source code archive, it is available at the location

*"\\nipcbatt-2.x\\src\\docs\\Python PCB Assembly Test Toolkit - Getting Started.pdf".*

## 4 Launch Python PCB Assembly Test Toolkit

Start testing with Python PCB Assembly Test Toolkit with the steps below:

1. Refer to the Getting Started guide to install the Python toolkit from PyPI. To go to the source folder which includes Library and tests through the installed library, refer to the following path:

*"<venv>\\Lib\\site-packages\\nipcbatt\\pcbatt\_library"*

Or if have downloaded source code from GitHub, refer to this path:

*"\\nipcbatt-main\\src\\nipcbatt\\pcbatt\_library"*

Or if have downloaded source code from PyPI website, refer to this path:

*"\\nipcbatt-2.x\\src\\nipcbatt\\pcbatt\_library"*

2. Go to the following location to copy the individual and pair examples into your workspace:

*"<venv>\\Lib\\site-packages\\nipcbatt\\pcbatt\_validation\_examples"*

Or

*"\\nipcbatt-main\\src\\nipcbatt\\pcbatt\_validation\_examples"*

Or

*"\\nipcbatt-2.x\\src\\nipcbatt\\pcbatt\_validation\_examples"*

3. Go to the following location to locate the example automation sequences:

*"<venv>\\Lib\\site-packages\\nipcbatt\\pcbatt\_automation"*

Or

*"\\nipcbatt-main\\src\\nipcbatt\\pcbatt\_automation"*

Or

*"\\nipcbatt-2.x\\src\\nipcbatt\\pcbatt\_automation"*

4. The FT Demo sequence can be found in the location:

"\<venv>\Lib\site-packages\nipcbatt\pcbatt\_ft\_demo\_test\_sequence"

Or

"\nipcbatt-main\src\nipcbatt\pcbatt\_ft\_demo\_test\_sequence"

Or

"\nipcbatt-2.x\src\nipcbatt\pcbatt\_ft\_demo\_test\_sequence"

## 5 Measurement Libraries

Python PCBA Measurement Library is located inside "\nipcbatt-main\src\nipcbatt\pcbatt\_library" or "\nipcbatt-2.x\src\nipcbatt\pcbatt\_library" folder. This folder contains all the measurement libraries and common reusables. Refer the table below for the list of available libraries.

The **daq** measurement libraries are found in the **nipcbatt\pcbatt\_library\daq directory**:

Measurement Library	Overview
<b>power_supply_source_and_measurements</b>	<i>PowerSupplySourceAndMeasure</i> class methods can be used to initialize, configure, source, measure and close on user configurable power supply pins. This library is applicable for TestScale power supply module (TS-15200).
<b>dc_rms_voltage_measurements</b>	<i>DcRmsVoltageMeasurement</i> class methods can be used to initialize, configure, measure and close on user configurable Analog input pins. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b>dc_voltage_generations</b>	<i>DcVoltageGeneration</i> class methods can be used to initialize, configure, generate and close on user configurable Analog output pins for DC voltage generation. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b>dc_rms_current_measurements</b>	<i>DcRmsCurrentMeasurement</i> class methods can be used to initialize, configure, measure and close on user configurable analog input voltage pins for current measurements. This library applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b>time_domain_measurements</b>	<i>TimeDomainMeasurement</i> class methods can be used to initialize, configure, measure and close on user configurable Analog input pins and derive time domain measurements for the measured waveforms. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b>frequency_domain_measurements</b>	<i>FrequencyDomainMeasurement</i> class methods can be used to initialize, configure, measure and close on user configurable Analog input pins and derive frequency domain measurements for the measured waveforms. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.

<b><i>signal_voltage_generations</i></b>	<i>SignalVoltageGeneration</i> class method provides options to generate different waveform voltage signals tones (single/multi) over a given generation time(s) on analog output terminals of DAQmx. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>static_digital_state_measurements</i></b>	<i>StaticDigitalStateMeasurement</i> class method measures the static digital data on all configured digital channel lines in the DAQmx task. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>static_digital_state_generations</i></b>	<i>StaticDigitalStateGeneration</i> class method generates the static digital data on all configured digital channel lines in the DAQmx task. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>dynamic_digital_pattern_measurements</i></b>	<i>DynamicDigitalPatternMeasurement</i> class methods helps to take dynamic digital measurements on a single/multiple digital channel lines or on an entire port in a digital module. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>dynamic_digital_pattern_generations</i></b>	<i>DynamicDigitalPatternGeneration</i> class methods can be used to initialize, configure, generate and close on user configurable Digital output pins and generate digital pattern in specified lines. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>digital_clock_generations</i></b>	<i>DigitalClockGeneration</i> class methods can be used to initialize, configure, generate and close on user configurable terminals using counters. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>digital_pulse_generations</i></b>	<i>DigitalPulseGeneration</i> class methods can be used to initialize, configure, generate and close on user configurable terminals using counters. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>digital_frequency_measurements</i></b>	<i>DigitalFrequencyMeasurement</i> class methods can be used to initialize, configure, measure and close on user configurable PFI lines using selected counter for digital frequency measurement. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>digital_pwm_measurements</i></b>	<i>DigitalPwmMeasurement</i> class methods can be used to initialize, configure, measure and close on counter input task assigned with an input terminal. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b><i>digital_edge_count_measurements</i></b>	<i>DigitalEdgeCountMeasurementUsingHardwareTimer</i> and <i>DigitalEdgeCountMeasurementUsingSoftwareTimer</i>

	class methods can be used to initialize, configure, measure and close on user configurable PFI lines using selected counters for digital events/edges counting. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b>synchronizations</b>	<i>SynchronizationSignalRouting</i> class methods can be used to route signals between specified source signal and output terminals for the given DAQmx Task. This library is applicable for PC Based DAQ, TestScale and cDAQ hardware.
<b>temperature_measurements</b>	<p><i>TemperatureMeasurementUsingRtd</i> class methods can be used to initialize, configure, measure and close on user configurable Analog input pins to derive temperature measurements from RTDs (Resistance Temperature Detector). This library is applicable for C Series Temperature Input Modules.</p> <p><i>TemperatureMeasurementUsingThermistor</i> class methods can be used to initialize, configure, measure and close on user configurable Analog input pins to derive temperature measurements from voltage excited NTC typed Thermistor devices. This library is applicable for PC based DAQ devices, TestScale analog input modules, C Series Voltage Input Modules.</p> <p><i>TemperatureMeasurementUsingThermocouple</i> class methods can be used to initialize, configure, measure and close on user configurable Analog input pins to derive temperature measurements from Thermocouples. This library is applicable for C Series Temperature Input Modules.</p>

The **communications** measurement libraries are found in the ***nipcbatt\pcbatt\_library\communications*** directory:

Measurement Library	Overview
<b>serial_communications, i2c_communications, spi_communications</b>	These libraries contain classes and simple read and write methods for I2C, SPI and Serial mode of communications. This library is applicable for USB-8452 and USB-232.

The **dmm** measurement libraries are found in the ***nipcbatt\pcbatt\_library\dmm*** directory:

Measurement Library	Overview
<b>dc_rms_voltage_measurements</b>	<i>DcRmsVoltageMeasurement</i> class methods can be used to initialize, configure, measure and close on user configurable Analog input pins. This library is applicable for DMM hardware.



<b><i>dc_rms_current_measurements</i></b>	<i>DcRmsCurrentMeasurement</i> class methods can be used to initialize, configure, measure and close on user configurable analog input voltage pins for current measurements. This library applicable for DMM hardware.
<b><i>mixed_measurements</i></b>	<i>MixedMeasurement</i> class methods can be used to class methods can be used to initialize, configure, measure and close on user configurable analog input voltage pins for current, voltage, or resistance measurements. This library is applicable for DMM hardware.
<b><i>resistance_measurements</i></b>	<i>DcRmsResistanceMeasurement</i> class methods can be used to class methods can be used to initialize, configure, measure and close on user configurable analog input voltage pins for resistance measurements. This library is applicable for DMM hardware.

The **switch** measurement libraries are found in the ***nipcbatt\pcbatt\_library\switch directory***:

Measurement Library	Overview
<b><i>static_digital_path_generations</i></b>	<i>StaticDigitalPathGeneration</i> class methods can be used to class methods can be used to initialize, configure, measure and close paths between user configurable switch channels. This library is applicable for SWITCH hardware.

The **dmm\_scan** measurement libraries are found in the ***nipcbatt\pcbatt\_library\dmm\_scan directory***:

Measurement Library	Overview
<b><i>dmm_scan_pmps_16V_15C</i></b>	<i>DmmScanPMPS</i> class methods can be used to initialize, configure, and execute a mixed measurement scan that automatically produces a voltage, current, or resistance measurement by coordinating the switch channels for a given configuration. This library is applicable for the combination of DMM and SWITCH hardware.

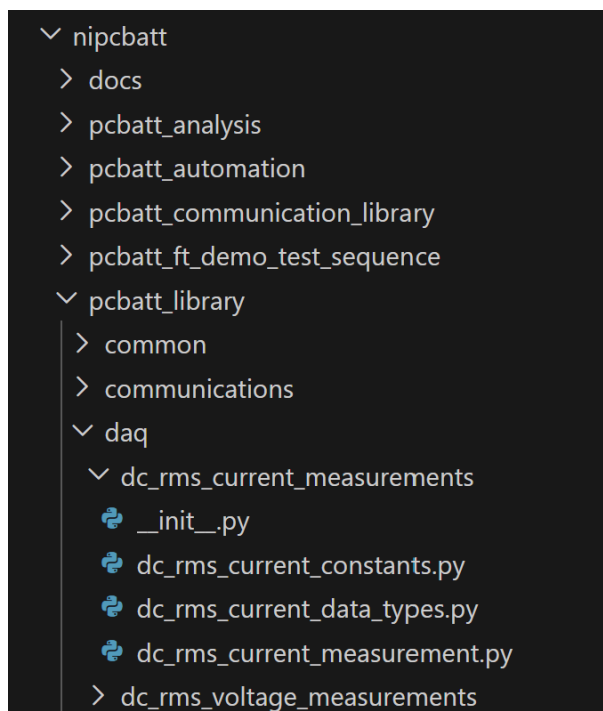


#### NOTE

Detailed Help on how to use the above measurement libraries can be found in the [Libraries](#) section.

## 5.1 Structure of Library

All the measurement libraries mentioned above are present in the *nipcbatt\pcbatt\_library* folder:



Each library/class has three main methods as below,

Method	Overview
<b>initialize()</b>	Used to initialize a DAQmx, NI-DMM, or NI-SWITCH session using either physical or global channels provided to perform the respective task.
<b>configure_and_measure()/configure_and_generate()</b>	Configures, Initiates and Measure/Generate for an input/output task respectively. Also, can return raw data for external custom post analysis and measurements from embedded analysis(selectable/optional)
<b>close()</b>	Closes the session and clears resources.

## 5.2 Physical and Virtual Channels

All Libraries (except power\_supply\_source\_and\_measurements) support [Global Virtual Channels](#) created using NI-MAX. User can either provide physical channels from the module directly or use virtual channels. If both are provided, Virtual channels will take precedence and will be used while initializing the task.



### NOTE

The channel settings of Global Virtual Channels set in NI-MAX will be overwritten using Configure and Measure.

### 5.2.1 Create/Modify Global Virtual Channels

A virtual channel is a collection of settings such as a name, a physical channel, input terminal connections, the type of measurement or generation, and can include scaling information. A virtual

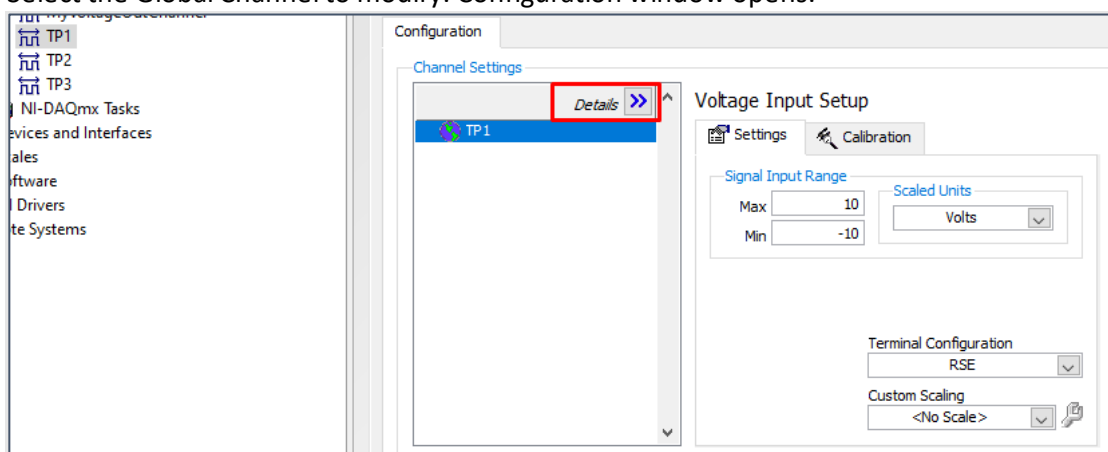
channel created outside a task is a Global Virtual Channel. For detailed definitions and example codes, refer to [Creating a Virtual Channel in NI-DAQmx and Using it in LabVIEW](#).

Follow the below steps to **create Global Virtual Channel** in NI-MAX.

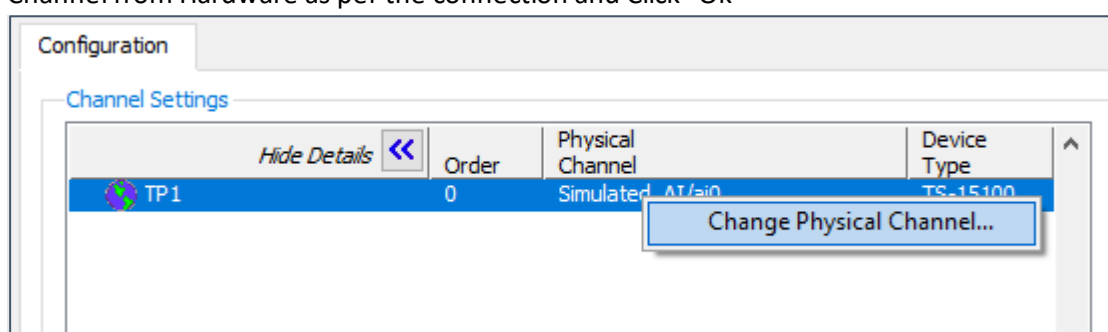
1. Launch NI-MAX
2. In NI-MAX, right-click **Data Neighbourhood** and select **Create New**
3. In the Create New window, select **NI-DAQmx Global Virtual Channel** and click **Next**. The DAQ Assistant opens.
4. Select an I/O type, such as analog input
5. Select the physical channel of Hardware
6. Type the global virtual channel [name](#). Click **Finish**
7. Save your configuration.

Follow the below steps to **modify the existing Global Virtual Channel** in NI-MAX.

1. Launch NI-MAX
2. In NI-MAX, expand **Data Neighbourhood > NI-DAQmx Global Virtual Channel**
3. Select the Global Channel to modify. Configuration window opens.



4. Click on “Details >>” as highlighted above to view the Physical Channel
5. Right click and **Change Physical Channel** to update the Physical Channel. Select the Physical Channel from Hardware as per the connection and Click “Ok”



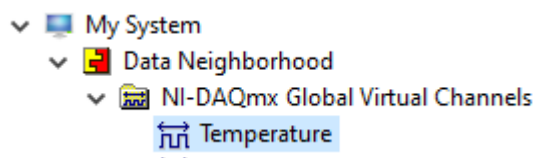
6. **Save** your configuration

### 5.2.2 Calibration

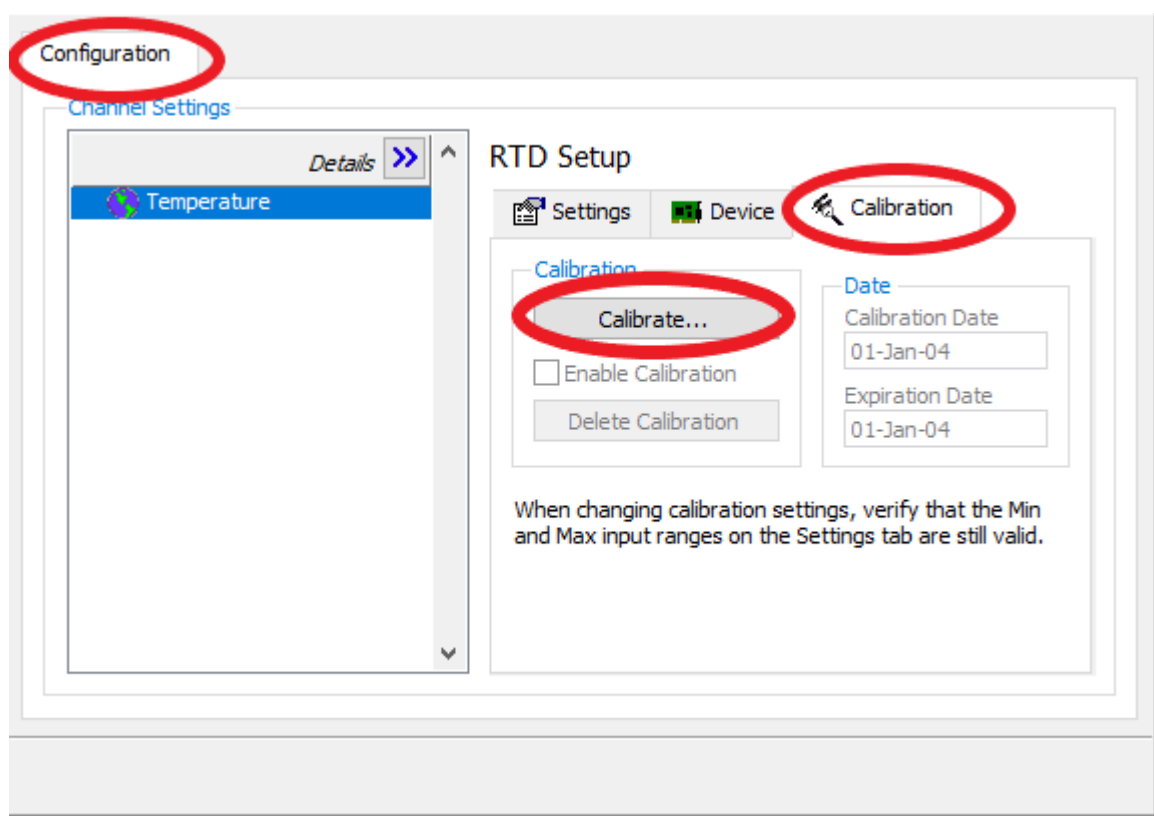
Calibrate a channel to increase the accuracy of a measurement by compensating for errors due to cabling, wiring, or sensors. NI MAX provides calibration option for DAQ devices virtual global channels.

With the use of Virtual global channels, we can calibrate input measurements individually, save the calibration, edit them later or recalibrate it. It can be useful for sensors calibrations to compensate sensors errors. To calibrate a virtual global channel, follow the below steps,

1. Open NI MAX
2. Create/Select the Virtual global channel to calibrate under **My System»Data Neighborhood»NI-DAQmx Global Virtual Channels»YourChannelName** from the **Configuration** view in NI MAX.



3. Select **Configuration»Calibration»Calibrate...** in the NI\_DAQmx Global Channel view.



4. In Channel Calibration Wizard, enter channel calibration details.

For more details on performing calibration of virtual global channels, refer [Performing DAQmx Channel Calibration in MAX Using Wizard](#).

### 5.3 Execution Options

The execution option mechanism allows to use a sequential single thread for synchronization by triggers.

Configure and Measure can be done in a single or multiple steps. Refer below for three execution options available in configuration.

Execution Option	Overview
<b>CONFIGURE_AND_MEASURE</b>	Configures the instrument and performs the measurement
<b>CONFIGURE_ONLY</b>	Sets the configuration to the instrument and initiate but skips the fetch and measure. This option can be used for synchronization where the channel can be configured to wait for the trigger to start the measure
<b>MEASURE_ONLY</b>	Fetch the data captured in the instrument and performs the processing to return the measurements. Please note that Measure only cannot be repeated without preceding with the "CONFIGURE_ONLY "

## 5.4 Limitations

1. **MEASURE\_ONLY** cannot be repeated without preceding with the **Configure Only** option. **CONFIGURE\_ONLY** is necessary to initiate a measurement task each time.
2. Limitations of Drivers and Hardware are applicable for Libraries. A Task/Session can control multiple lines from different DAQ modules, but multiple tasks/sessions cannot be initiated for a single DAQ, DMM, or Switch Module.
3. Timed digital input/output restrictions:
  - a) You cannot use parallel and serial modules together on the same hardware timed task, unless they are in separate DAQ chassis using multi-chassis device tasks.
  - b) You cannot use serial modules for triggering.
  - c) You cannot do both static and timed tasks at the same time on a single serial module.
  - d) You can only do hardware timing in one direction at a time on a serial module.

## 6 Libraries

For ease of use, inputs parameters for the library are set to default values and are configurable by the user:

### Default Values for DAQ measurements:

- All functions are set to measure or generate over a 0.1s finite timing
- Maximum sampling rate and timings are optimized for resolution, configurable for each function
- All analog inputs range are defined to +-10V
- All Terminal Configuration Modes are set to RSE (Referenced Single Ended), can be set individually (each analog input line) to other Terminal Configuration Modes.
- Sample timing engine set to AUTO
- All Triggers Controls are set to Trigger Type: No trigger, Digital Source: Empty, Digital Start trigger Edge: Rising
- Execution option set to Configure & Measure by default.
- Skip Analysis will be disabled by default.

### Default Values for DMM measurements:

- All functions are set to measure over an Auto Aperture Time (-1.0s) and Auto Settle Time (-1.0s)
- Powerline frequency is set to 50.0 Hz by default
- Default resolution is set to 5.5 digits for all measurement functions
- All measurement ranges (Voltage, Current, Resistance) are set to Auto Range by default
- AC measurements are set to a minimum frequency of 40.0 Hz by default
- All Triggers Controls are set to Trigger Source: Immediate, Trigger Delay: Auto (-1.0s), Trigger Slope: Rising, and Enable Trigger: False
- Execution option set to Configure & Measure by default.
- 

### Default Values for Switch measurements:

- Device reset is enabled by default during initialization (reset\_device=True)
- Simulation mode is disabled by default (simulate=False)
- Maximum debounce wait time is set to 5000 ms to ensure path stability after connection or disconnection operations.

### Default Values for DMM Scan (PMPS) measurements:

- Default Powerline frequency is set to 50.0 Hz
- Measurement aperture, settle time, and trigger delay are all set to Auto (-1.0s) by default
- Default resolution for each point in the scan is 5.5 digits
- Default relay topologies are pre-configured (e.g., "2527/2-Wire Dual 16x1 Mux") to ensure immediate compatibility with standard PMPS hardware layouts
- All shunt relays are set to "Close All" by default to ensure current paths are maintained during scan initiation

- All Triggers Controls follow DMM defaults: Trigger Source: Immediate, Trigger Slope: Rising, and Enable Trigger: False
- Execution option set to Configure & Measure by default.

## 6.1 DMM Measurements

### 6.1.1 DC RMS Voltage Measurements

#### Overview

Use the ***DcRmsVoltageMeasurement*** class methods to initialize, configure, measure and close DC or AC RMS voltage using a DMM instrument. Applicable for NI-DMM instruments.

#### Validated Hardware

- PXI-4065

#### Instructions

1. Create an instance of the ***DcRmsVoltageMeasurement()*** class using the library.
2. Initialize the DMM session by calling the ***initialize(self, dmm\_resource\_name: str, powerline\_frequency: float)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: DcRmsVoltageMeasurementConfiguration)*** method with the following parameters:
  - a. ***execution\_type (MeasurementExecutionType):***
    - Specify the mode of execution: Configure & Measure, Configure only, or Measure only.
  - b. ***measurement\_function\_parameters (DcRmsVoltageMeasurementFunctionParameters):***
    - Set parameters to configure the measurement function (DC or AC RMS voltage) and the voltage range, and the resolution in digits for the measurement.
  - c. ***trigger\_parameters (TriggerParameters):***
    - Configure either no trigger or an enabled trigger. If using a trigger, provide a suitable trigger source and set the trigger slope as Rising or Falling.
  - d. ***timing\_parameters (TimingParameters):***
    - Set the aperture time and settle time for the measurement. Use the default value (-1) to allow the driver to manage these settings automatically.
  - e. ***ac\_min\_frequency (float):***
    - Specify the minimum frequency in Hz for AC voltage measurements. This setting is ignored for DC voltage measurements.
  - f. The output data includes (***DcRmsVoltageMeasurementResultData***):
    - The measured voltage value, unit, and formatted measurement result.
    - The DMM execution settings applied during the measurement.



4. Finally close the DMM session using the ***close(self)*** method.

#### Notes:

- For higher accuracy, use a higher resolution in digits settings.

## 6.1.2 DC RMS Current Measurements

### Overview

Use the ***DcRmsCurrentMeasurement*** class methods to initialize, configure, measure and close DC or AC RMS current on user configurable DMM channels. Applicable for NI-DMM instruments.

### Validated Hardware

- PXI-4065

### Instructions

1. Create an instance of the ***DcRmsCurrentMeasurement()*** class using the library.
2. Initialize the DMM session by calling the ***initialize(self, dmm\_resource\_name: str, powerline\_frequency: float)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: DcRmsCurrentMeasurementConfiguration)*** method with the following parameters:
  - a. ***execution\_type (MeasurementExecutionType):***
    - Specify the mode of execution: Configure & Measure, Configure only, or Measure only.
  - b. ***measurement\_function\_parameters (DcRmsCurrentMeasurementFunctionParameters):***
    - Set parameters to configure the measurement function (DC or AC RMS current) and the current range, and the resolution in digits for the measurement.
  - c. ***trigger\_parameters (TriggerParameters):***
    - Configure either no trigger or an enabled trigger. If using a trigger, provide a suitable trigger source and set the trigger slope as Rising or Falling.
  - d. ***timing\_parameters (TimingParameters):***
    - Set the aperture time and settle time for the measurement. Use the default value (-1) to allow the driver to manage these settings automatically.
  - e. ***ac\_min\_frequency (float):***
    - Specify the minimum frequency in Hz for AC current measurements. This setting is ignored for DC current measurements.
  - f. The output data includes (***DcRmsCurrentMeasurementResultData***):

- The measured current value, unit, and formatted measurement result.
  - The DMM execution settings applied during the measurement.
4. Finally close the DMM session using the ***close(self)*** method.

## Notes

- For higher accuracy, use a higher resolution in digits setting.

## 6.1.3 Resistance Measurement

### Overview

Use the ***DcRmsResistanceMeasurement*** class methods to initialize, configure, measure, and close resistance measurements on user configurable DMM channels. Supports both 2-wire and 4-wire resistance measurement modes. Applicable for NI-DMM instruments.

### Validated Hardware

- PXI-4065

### Instructions

1. Create an instance of the ***DcRmsResistanceMeasurement()*** class using the library.
2. Initialize the DMM session by calling the ***initialize(self, dmm\_resource\_name: str, powerline\_frequency: float)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: ResistanceMeasurementConfiguration)*** method with the following parameters:
  - a. ***execution\_type (MeasurementExecutionType):***
    - Specify the mode of execution: Configure & Measure, Configure only, or Measure only.
  - b. ***measurement\_function\_parameters (ResistanceMeasurementFunctionParameters):***
    - Set parameters to configure the measurement function (2-wire or 4-wire resistance) and the resistance range, and the resolution in digits for the measurement.  
Use ***ResistanceRangeAndFunctions.TWO\_W\_Resistance\_Auto\_Range*** or ***FOUR\_W\_Resistance\_Auto\_Range*** for automatic range selection, or choose a fixed range (e.g., ***TWO\_W\_RES\_1k\_Ohm***, ***FOUR\_W\_RES\_10k\_Ohm***).
  - c. ***trigger\_parameters (TriggerParameters):***
    - Configure either no trigger or an enabled trigger. If using a trigger, provide a suitable trigger source and set the trigger slope as Rising or Falling.
  - d. ***timing\_parameters (TimingParameters):***
    - Set the aperture time and settle time for the measurement. Use the default value (-1) to allow the driver to manage these settings automatically.
  - e. The output data includes (***ResistanceMeasurementResultData***):

- The measured resistance value, unit, and formatted measurement result.
  - The DMM execution settings applied during the measurement.
4. Finally close the DMM session using the ***close(self)*** method.

### Notes

- For higher accuracy, use a higher resolution in digits setting.

## 6.1.4 Mixed Measurement

### Overview

Use the ***MixedMeasurement*** class methods to initialize, configure, measure and close DC/AC voltage, DC/AC current, or 2-wire/4-wire resistance using a DMM instrument. Applicable for NI-DMM instruments.

### Validated Hardware

- PXI-4065

### Instructions

1. Create an instance of the ***MixedMeasurement()*** class using the library.
2. Initialize the DMM session by calling the ***initialize(self, dmm\_resource\_name: str, powerline\_frequency: float)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: MixedMeasurementConfiguration)*** method with the following parameters:
  - a. ***execution\_type (MeasurementExecutionType):***
    - Specify the mode of execution: Configure & Measure, Configure only, or Measure only.
  - b. ***measurement\_function\_parameters (MixedMeasurementFunctionParameters):***
    - Set parameters to configure the measurement function and range using ***MixedRangeAndFunctions***. Supported functions include DC/AC voltage, DC/AC current, and 2-wire/4-wire resistance. Also set the resolution in digits for the measurement.
  - c. ***trigger\_parameters (TriggerParameters):***
    - Configure either no trigger or an enabled trigger. If using a trigger, provide a suitable trigger source and set the trigger slope as Rising or Falling.
  - d. ***timing\_parameters (TimingParameters):***
    - Set the aperture time and settle time for the measurement. Use the default value (-1) to allow the driver to manage these settings automatically.
  - e. ***ac\_min\_frequency (float):***

- Specify the minimum frequency in Hz for AC voltage or AC current measurements. This setting is ignored for DC and resistance measurements.
- f. The output data includes (***MixedMeasurementResultData***):
- The measured value, unit, and formatted measurement result.
  - The DMM execution settings applied during the measurement.
4. Finally close the DMM session using the ***close(self)*** method.

**Notes:**

- For higher accuracy, use a higher resolution in digits setting.
- "Measure Only" cannot be performed without preceding it with "Configure Only".

## 6.2 DAQ Measurements

### 6.2.1 Power Supply Source and Measure

**Overview**

Use the ***PowerSupplySourceAndMeasure*** class methods to initialize, configure, source, measure and close on user configurable power supply pins. Applicable for TestScale hardware.

**Validated Hardware**

- cDAQ and PXI-based DAQ modules

**Instructions**

1. Create an instance of the ***PowerSupplySourceAndMeasure()*** class using the library.
2. Initialize the DAQmx task by calling the ***initialize(self, power\_channel\_name:str)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: PowerSupplySourceAndMeasureConfiguration)*** method with the following parameters:
  - a. Configure the ***measurement\_options (MeasurementOptions)*** property:
    - Set ***execution\_option (MeasurementExecutionType)*** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
    - Set ***Measurement\_analysis\_requirement*** to determine whether post-analysis of the measured data is required. Select SKIP\_ANALYSIS to skip analysis or PROCEED\_TO\_ANALYSIS to perform analysis.

**Note:** "Measure Only" cannot be performed without preceding with "Configure Only".

- b. Configure ***terminal\_parameters (PowerSupplySourceAndMeasureTerminalParameters)***:

- Set parameters to configure the terminal parameters such as voltage setpoint, current setpoint, power sense mode, idle output behaviour of the channel.
- c. Configure ***sample\_clock\_timing\_parameters (SampleClockTimingParameters)***:
  - Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.

Acquisition time = Number of Samples / Sampling Rate.

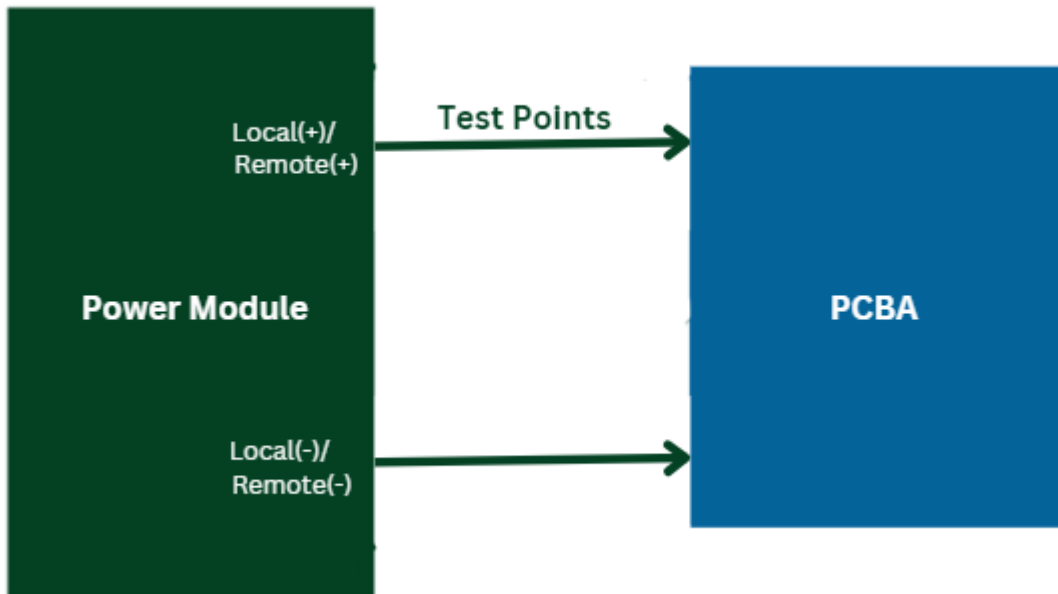
**Note:** Maximum sampling rate for TS-15200 is 10 kS/s. Setting values more than that will generate an error.

- d. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:
    - Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
  - e. The Output data includes (***PowerSupplySourceAndMeasureResultData***):
    - Voltage and Current Waveforms - Displays the captured voltage and current waveform during sourcing power supply
    - Power Measurements - Provides the derived post-analysed data from the captured waveforms.
    - Acquisition Time – Calculated acquisition time for the measurement.
4. Finally close the power supply measurement task using the ***close(self)*** method

#### Notes:

- For fast measurements, set the maximum sample rate possible for the given "Sample Clock Source" input and assign a smaller "Number of Samples" to capture. The minimum sample size is 2.
- For better accuracy, configure the instrument to read more samples, e.g., 1000 samples.

#### Block Diagram



## 6.2.2 DC-RMS Voltage Measurement

### Overview

Use the ***DcRmsVoltageMeasurement*** class methods to initialize, configure, measure, and close on user-configurable analog input pins. Applicable for PC based DAQ, DMM, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15100
- NI-9205, NI 9215
- PCIe-6323
- PXI-4065

**Note:** This method is compatible with other similar DAQmx supported hardware having basic analog input resources.

### Instructions

1. Create an instance of the ***DcRmsVoltageMeasurement()*** class using the library.
2. Initialize the DAQmx task by calling the ***initialize(self, analog\_input\_channel\_expression:str)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration:DcRmsVoltageMeasurementConfiguration)*** method with the following parameters:
  - a. Configure the ***measurement\_options (MeasurementOptions)*** property:
    - Set ***execution\_option (MeasurementExecutionType)*** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
    - Set ***Measurement\_analysis\_requirement*** to determine whether post-analysis of the measured data is required. Select SKIP\_ANALYSIS to skip analysis or PROCEED\_TO\_ANALYSIS to perform analysis.

**Note:** “Measure only” cannot be performed without preceding with “Configure Only”.

- b. Configure ***global\_channel\_parameters (VoltageRangeAndTerminalParameters)***:
  - Set parameters to configure the range and terminal settings for all analog input voltage channels, excluding channels specified in the specific channel settings.
- c. Configure ***specific\_channels\_parameters (list[VoltageMeasurementChannelAndTerminalRangeParameters])***:
  - Set parameters for specific channels to a particular limit. If Global Virtual Channel name is provided in initialize(), provide the Virtual Channel name in Specific Channel Parameters as well. Similarly, if a Physical Channel name is provided in initialize(),

provide the Physical Channel name in `SpecificChannelParameters`. If no channels are specified, global settings are applied.

d. Configure ***sample\_clock\_timing\_parameters (SampleClockTimingParameters)***:

- Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.

Acquisition time = Number of Samples / Sampling Rate.

**Notes:**

- The maximum aggregate sampling rate for TS-15100 / NI-9205 / PCIe-6323 is 250 kS/s values exceeding this limit will generate an error.
- PCIe supports only the Auto Mode of the Timing Engine.

e. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:

- Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.

f. The output data includes (***DcRmsVoltageMeasurementResultData***):

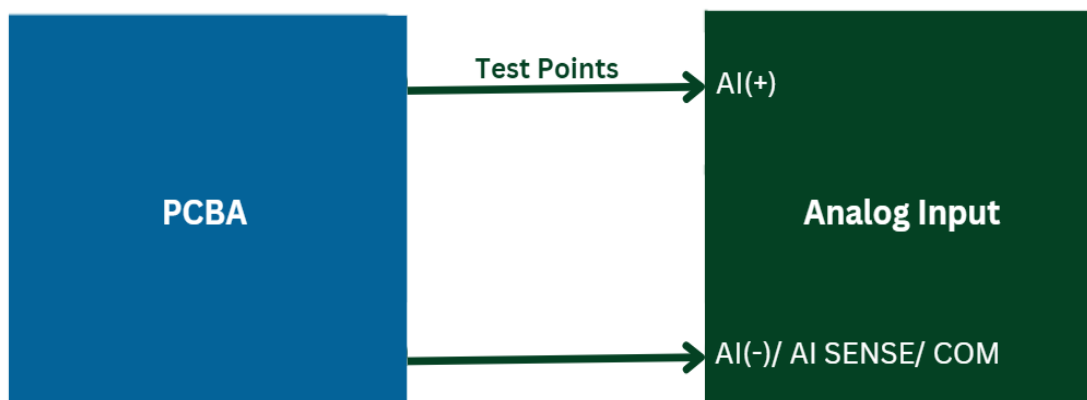
- Provide DC-RMS Voltage Measurements derived from post-analysis.
- Provide Average DC Voltage calculated from post-analysis.

4. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

- For fast measurements, set the maximum sample rate possible for the given "Sample Clock Source" input and assign a smaller "Number of Samples" to capture. The minimum sample size is 2.
- For better accuracy, configure the instrument to read more samples, e.g., 1000 samples.

**Block Diagram**





### 6.2.3 DC Voltage Generation

#### Overview

Use the **DcVoltageGeneration** class methods to initialize, configure, generate, and close on user-configurable analog input pins. Applicable for PC based DAQ, TestScale and cDAQ hardware.

#### Validated Hardware

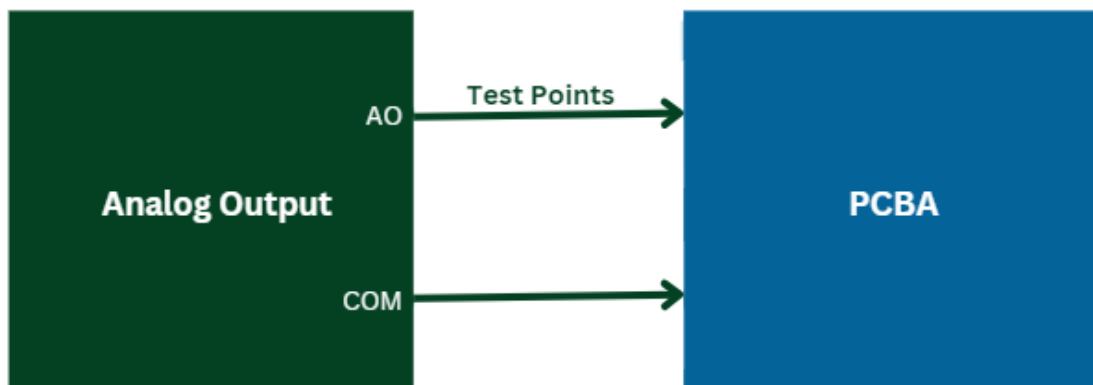
- TS-15110
- NI-9263
- PCIe-6323

#### Instructions

1. Create an instance of the **DcVoltageGeneration()** class using the library.
2. Initialize the DAQmx task by calling the **initialize(self, analog\_output\_channel\_expression: str)**
3. Configure settings by calling the **configure\_and\_generate(self, configuration: DcVoltageGenerationConfiguration)** method with the following parameters:
  - a. Configure **voltage\_generation\_range\_parameters()** property:
    - Set the **range\_min\_volts** to declare the minimum possible value of the generated voltage.
    - Set the **range\_max\_volts** to declare the maximum possible value of the generated voltage.
  - b. Configure the **output\_voltages** to set the voltage that must be generated.
4. Finally close the DAQmx task using **close(self)** method.

**Note:** Upon completion of the task, it is crucial to set the output voltages of the power supply to zero. Failure to do so may result in the device continuing to generate the last specified voltage. To ensure that the output voltage is correctly reset set **output\_voltages** to zero and run the Program.

#### Block Diagram



## 6.2.4 DC-RMS Current Measurement

### Overview

Use the **DcRmsCurrentMeasurement** class methods to initialize, configure, measure, and close on user-configurable analog input pins. Applicable for PC based DAQ, DMM, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15100
- NI-9205, NI 9215
- PCIe-6323
- PXI-4065

### Instructions

1. Create an instance of the **DcRmsCurrentMeasurement()** class using the library.
2. Initialize the DAQmx task by calling the **initialize(self, analog\_input\_channel\_expression: str, use\_specific\_channel: bool)** method on the class instance.  
**Note:** If Global channel(s) is/are used, it should be declared as "Current".
3. Configure settings by calling the **configure\_and\_measure(self, configuration: DcRmsCurrentMeasurementConfiguration)** method with the following parameters:
  - a. Configure the **measurement\_options (MeasurementOptions)** property:
    - Set **execution\_option (MeasurementExecutionType)** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
    - Set **measurement\_analysis\_requirement** to determine whether post-analysis of the measured data is required. Select SKIP\_ANALYSIS to skip analysis or PROCEED\_TO\_ANALYSIS to perform analysis.**Note:** "Measure only" cannot be performed without preceding with "Configure Only".
  - b. Configure **global\_channel\_parameters (VoltageRangeAndTerminalParameters)**:
    - Configure the required analog input channel settings. The channel settings include Max and Min Current range, Terminal configuration and Shunt resistor value.
  - c. Configure **specific\_channels\_parameters**:
    - Set parameters for specific channels to a particular limit. If Global Virtual Channel name is provided in initialize(), provide the Virtual Channel name in Specific Channel Parameters as well. Similarly, if a Physical Channel name is provided in initialize(), provide the Physical Channel name in Specific Channel Parameters.**Note:** "**use\_specific\_channel**" in **initialize()** has to be declared as **True** for using Specific channel settings. For more information refer to the [known issues](#) section.
  - d. Configure **sample\_clock\_timing\_parameters (SampleClockTimingParameters)**:
    - Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.  
 Acquisition time = Number of Samples / Sampling Rate.

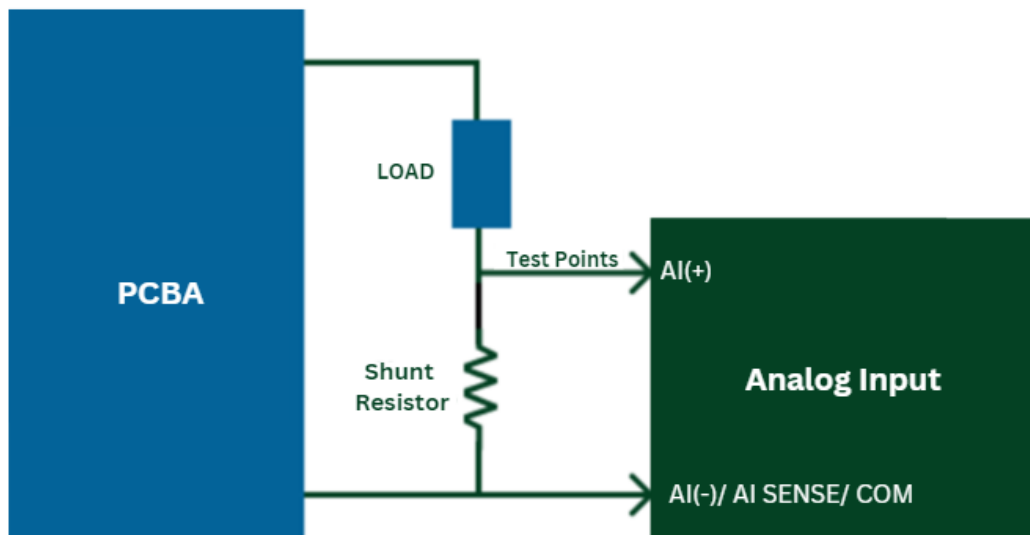
**Notes:**

- The maximum aggregate sampling rate for TS-15100 / NI-9205 / PCIe-6323 is 250 kS/s. Values exceeding this limit will generate an error.
  - PCIe supports only the Auto Mode of the Timing Engine.
  - e. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:
    - Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
  - f. The Output data includes
    - Current Waveforms - Displays the captured DC current waveforms.
    - DC-RMS Current Measurements - Provides the derived post-analyzed data from the captured DC waveforms.
4. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

- For fast measurements, set the maximum sample rate possible for the given "Sample Clock Source" input and assign a smaller "Number of Samples" to capture. The minimum sample size is 2.
- For better resolution, configure the instrument to read more samples e.g., 1000 samples.

**Block Diagram**



## 6.2.5 Time Domain Measurement

### Overview

Use the ***TimeDomainMeasurement*** class methods to initialize, configure, measure and close on user configurable Analog input pins and derive time domain measurements for the measured waveforms. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15100
- NI-9205, NI 9215
- PCIe-6323

### Instructions

1. Create an instance of the ***TimeDomainMeasurement()*** class using the library.
2. Initialize the DAQmx task by calling the ***initialize(self, analog\_input\_channel\_expression: str)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: TimeDomainMeasurementConfiguration)*** method with the following parameters:
  - a. Configure the ***measurement\_options (MeasurementOptions)*** property:
    - Set ***execution\_option (MeasurementExecutionType)*** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
    - Set ***Measurement\_analysis\_requirement*** to determine whether post-analysis of the measured data is required. Select SKIP\_ANALYSIS to skip analysis or PROCEED\_TO\_ANALYSIS to perform analysis.

**Note:** “Measure only” cannot be performed without preceding with “Configure Only”.

- b. Configure ***global\_channel\_parameters (VoltageRangeAndTerminalParameters)***:
    - Set parameters to configure the range and terminal settings for all analog input voltage channels, excluding channels specified in the specific channel settings.
  - c. Configure ***specific\_channels\_parameters (list[VoltageMeasurementChannelAndTerminalRangeParameters])***:
    - Set parameters for specific channels to a particular limit. If Global Virtual Channel name is provided in initialize(), provide the Virtual Channel name in Specific Channel Parameters as well. Similarly, if a Physical Channel name is provided in initialize(), provide the Physical Channel name in Specific Channel Parameters. If no channels are specified, global settings are applied.
  - d. Configure ***sample\_clock\_timing\_parameters (SampleClockTimingParameters)***:
    - Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.  
Acquisition time = Number of Samples / Sampling Rate.

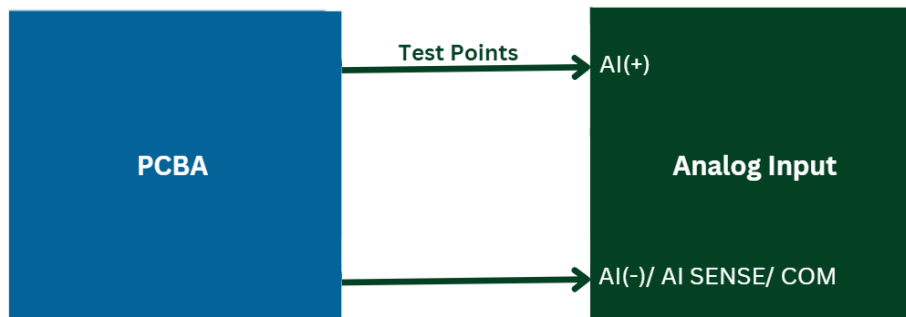
**Notes:**

- The maximum aggregate sampling rate for TS-15100 / NI-9205 / PCIe-6323 is 250 kS/s values exceeding this limit will generate an error.
  - PCIe supports only the Auto Mode of the Timing Engine.
- e. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:
- Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
- f. The output data includes (***TimeDomainMeasurementResultData***):
- Mean Dc Voltage Values derived from post-analysis.
  - Peak-to-peak amplitude voltage calculated from post-analysis.
  - Voltage waveforms frequencies calculated from post-analysis.
  - Voltage waveform duty cycle calculated from post-analysis.
4. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

- For fast measurements, set the maximum sample rate possible for the given "Sample Clock Source" input and assign a smaller "Number of Samples" to capture. The minimum sample size is 2.
- For better accuracy, configure the instrument to read more samples, e.g., 1000 samples.

**Block Diagram**



## 6.2.6 Frequency Domain Measurement

### Overview

Use the ***FrequencyDomainMeasurement*** class methods to initialize, configure, measure and close on user configurable Analog input pins and derive frequency domain measurements for the measured waveforms. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15100
- NI-9205, NI 9215
- PCIe-6323

### Instructions

1. Create an instance of the ***FrequencyDomainMeasurement()*** class using the library.
2. Initialize the DAQmx task by calling the ***initialize(self, analog\_input\_channel\_expression: str)*** method on the class instance.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: FrequencyDomainMeasurementConfiguration)***
  - a. Configure the ***measurement\_options (MeasurementOptions)*** property:
    - Set ***execution\_option (MeasurementExecutionType)*** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
    - Set ***Measurement\_analysis\_requirement*** to determine whether post-analysis of the measured data is required. Select SKIP\_ANALYSIS to skip analysis or PROCEED\_TO\_ANALYSIS to perform analysis.
  - Note:** "Measure only" cannot be performed without preceding with "Configure Only".
  - b. Configure ***global\_channel\_parameters (VoltageRangeAndTerminalParameters)***:
    - Set parameters to configure the range and terminal settings for all analog input voltage channels, excluding channels specified in the specific channel settings.
  - c. Configure ***specific\_channels\_parameters (list[VoltageMeasurementChannelAndTerminalRangeParameters])***:
    - Set parameters for specific channels to a particular limit. If Global Virtual Channel name is provided in ***initialize()***, provide the Virtual Channel name in Specific Channel Parameters as well. Similarly, if a Physical Channel name is provided in ***initialize()***, provide the Physical Channel name in Specific Channel Parameters. If no channels are specified, global settings are applied.
  - d. Configure ***sample\_clock\_timing\_parameters (SampleClockTimingParameters)***:
    - Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.  
Acquisition time = Number of Samples / Sampling Rate.

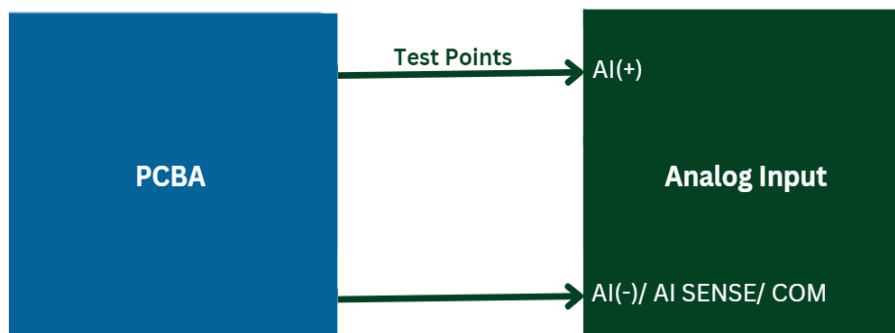
**Notes:**

- The maximum aggregate sampling rate for TS-15100 / NI-9205 / PCIe-6323 is 250 kS/s values exceeding this limit will generate an error.
  - PCIe supports only the Auto Mode of the Timing Engine.
  - e. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:
    - Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
  - f. The output data includes (***FrequencyDomainMeasurementResultData***):
    - Voltage Waveforms - Displays the captured voltage waveforms.
    - Frequency Domain Measurements - Provides the derived post-analysed data from the captured Frequency domain waveforms.
4. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

- For fast measurements, set the maximum sample rate possible for the given "Sample Clock Source" input and assign a smaller "Number of Samples" to capture. The minimum sample size is 2.
- For better accuracy, configure the instrument to read more samples, e.g., 1000 samples.

**Block Diagram**



## 6.2.7 Signal Voltage Generation

### Overview

**SignalVoltageGeneration** class provides options to generate different waveform voltage signals tones (single/multi) over a given generation time(s) on analog output terminals of DAQmx. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15110
- NI-9263
- PCIe-6323

### Instructions

1. Create an instance of the class **SignalVoltageGeneration()** class using the library.
2. Initialize the DAQmx task by calling the **initialize(self, analog\_output\_channel\_expression: str)**
3. Configure settings by calling the **configure\_and\_generate\_sine\_waveform()** for generating sine wave or **configure\_and\_generate\_square\_waveform()** for generating square wave or **configure\_and\_generate\_multiple\_tones\_waveform()** for generating sine wave with multiple tones method with the following parameters:
  - a. Configure **voltage\_generation\_range\_parameters** property:
    - Set the **range\_min\_volts** to declare the minimum possible value of generated voltage.
    - Set the **range\_max\_volts** to declare the maximum possible value of generated voltage.
  - b. Configure the **timing\_parameters** property:
    - Set the **sample\_clock\_source** parameter to give the source of the sample clock.
    - Adjust the **sampling\_rate\_hertz** parameter to configure the sampling rate.
    - Adjust the **generated\_signal\_duration\_seconds** parameter to configure the time for which the signal must be generated.
  - c. Configure the **digital\_start\_trigger\_parameters** property:
    - Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
  - d. For **configure\_and\_generate\_sine\_waveform()**:
    - Set the **generated\_signal\_tone\_parameters** which includes the frequency of the generated waveform, amplitude of the generated waveform and the phase of the generated waveform.
    - Set the **waveform\_parameters** which includes the offset voltage for the generated waveform.
  - e. For **configure\_and\_generate\_square\_waveform()**:

**Note:** The maximum aggregate sampling rate for TS-15100 / NI-9205 / PCIe-6323 is 250 kS/s values exceeding this limit will generate an error.



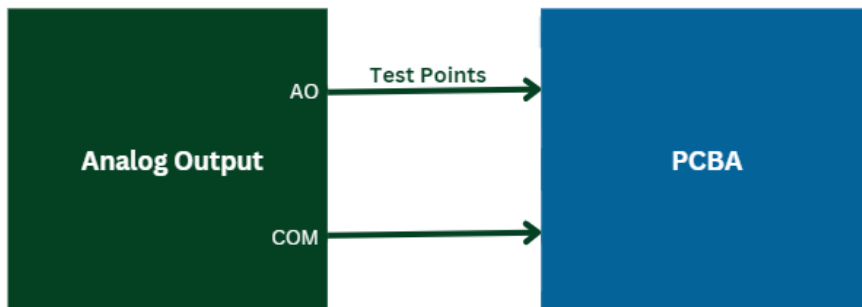
- Set the ***waveform\_parameters*** which includes the amplitude, duty cycle, frequency, phase and the voltage offset of the generated waveform.
- f. For ***configure\_and\_generate\_multiple\_tones\_waveform()***:
- Use ***multiple\_tones\_parameters(ToneParameters)*** to set the required waveform parameters.

4. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

- It is recommended to select the generation time (s) as an integral multiple of the time-period ( $1/\text{Frequency (Hz)}$ ) such that complete cycles of each tone are generated.
- For Multitone Generation, it is required that the generation time (s) is an integral multiple of the time-period ( $1/\text{Frequency (Hz)}$ ) of all the tones.

**Block Diagram**



## 6.2.8 Static Digital State Measurement

### Overview

Use the ***StaticDigitalStateMeasurement*** class methods to initialize, configure, measure and close on user configurable Digital input pins. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15120
- NI-9403
- PCIe-6323

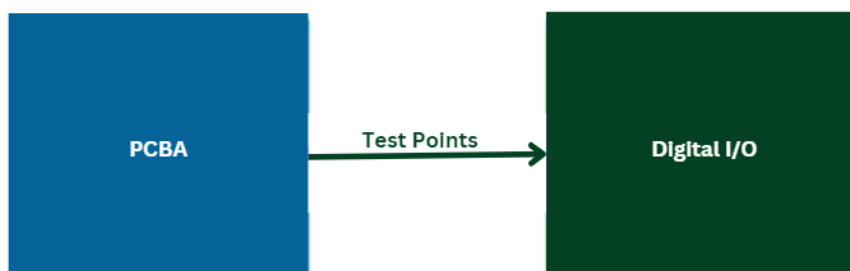
### Instructions

1. Create an instance of the ***StaticDigitalStateMeasurement()*** class using the library.
2. Initialize the DAQmx task by calling the ***initialize(self, channel\_expression: str)***.
3. The output data includes (***StaticDigitalStateMeasurementResultData***)
  - a. Channel Line Identifiers – returns the digital channels lines in the task.
  - b. Line States – outputs the acquired states of digital lines configured in the DAQmx Task. Each element in this array maps linearly to a line in the task which is returned in **Channel line identifier** array.
4. Finally close the DAQmx task using ***close(self)*** method.

### Note:

- Digital port is not supported in this Static Digital State Measurement Library, and it returns an error. Users can either use **Dynamic Digital Pattern Measurement** Library to support port-based inputs or specify the ports in **line-based format** (Ex: *DIO/port0/line0:31*) with this current library.

### Block Diagram



## 6.2.9 Static Digital State Generation

### Overview

Use the **StaticDigitalStateGeneration** class methods to initialize, configure, generate, and close on user configurable Digital output pins. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15120
- NI-9403, NI-9477
- PCIe-6323

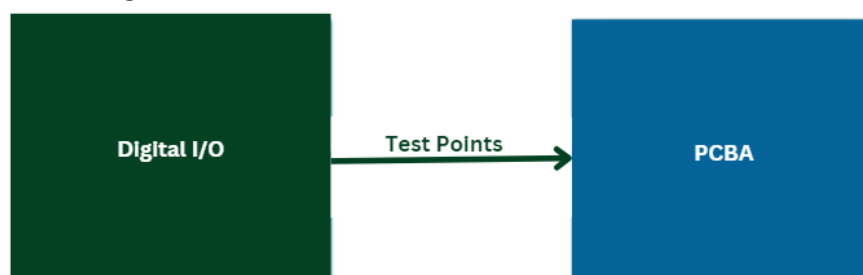
### Instructions

1. Create an instance of the class **StaticDigitalStateGeneration()** class using the library.
2. Initialize the DAQmx task by calling **initialize(self, channel\_expression: str)**.
3. Configure settings by calling the **configure\_and\_generate(self, configuration: StaticDigitalStateGenerationConfiguration )** method with the following parameters:
  - a. Configure the **StaticDigitalStateGenerationConfiguration()** to set the data to write. This represents the digital data to be output. Each element in the list corresponds to a line. The number of elements in the line must match the number of lines configured to write data.
4. Finally close the DAQmx task using **close(self)** method.

### Note:

- Digital port is not supported in this Static Digital State Generation Library, and it returns an error. Users can either use **Dynamic Digital Pattern Measurement** Library to support port-based inputs or specify the ports in **line-based format** (Ex: *DIO/port0/line0:31*) with this current library.

### Block Diagram



### 6.2.10 Dynamic Digital Pattern Measurement

#### Overview

**DynamicDigitalPatternMeasurement** class provides options to measure digital patterns through the specified lines of DAQmx. Applicable for PC based DAQ, TestScale and cDAQ hardware.

#### Validated Hardware

- TS-15050, TS-15120
- NI-9403
- PCIe-6323

#### Instructions

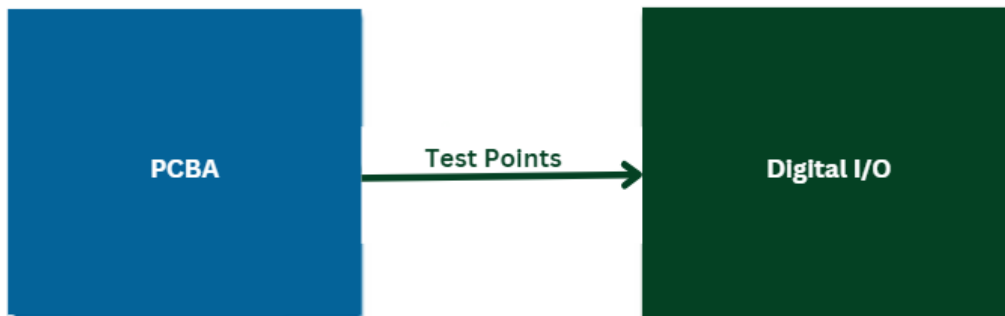
1. Initialize the DAQmx task by calling ***initialize(self, channel\_expression: str)***
2. Configure measurement settings by calling the ***configure\_and\_measure(self, configuration: DynamicDigitalPatternMeasurementConfiguration)***
3. Configure the ***DynamicDigitalPatternMeasurementConfiguration*** to set the configurations for pattern measurements.
  - a. Configure the ***measurement\_options (MeasurementOptions)*** property:
    - Set ***execution\_option (MeasurementExecutionType)*** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
    - Set ***Measurement\_analysis\_requirement*** to determine whether post-analysis of the measured data is required. Select SKIP\_ANALYSIS to skip analysis or PROCEED\_TO\_ANALYSIS to perform analysis.
  - b. Configure ***timing\_parameters*** to set sample clock source, sample rate, number of samples per channel and active edge.
 

**Note:** Maximum sampling rate for TS-15120 is 142 kS/s. Setting values more than that will generate an error.
  - c. Configure ***trigger\_parameters*** to set the trigger type, digital start trigger source and edge.
  - d. The Output data includes,
    - Measured Digital Pattern - Displays the captured digital voltage waveforms.
    - Port Digital Data - Provides the derived post-analyzed port digital data in U32 format from the captured digital patterns.
4. Finally close the DAQmx task using ***close(self)*** method.

#### Notes:

- Dynamic Digital Pattern Measurement Library supports **both Port and Digital lines** from a single module to measure dynamic digital patterns. But **multi-module support is not applicable** on this library and throws errors.
- For Fast measurements, set maximum sample rate possible for the given "Sample Clock Source" input and assign smaller "Number of Samples" to capture. Min sample size is 2.
- For better resolution, configure the instrument to read more samples e.g., 1000 samples.

### Block Diagram



### 6.2.11 Dynamic Digital Pattern Generation

#### Overview

**DynamicDigitalPatternGeneration** class provides options to generate digital patterns, an array of digital samples in the specified IO lines of DAQmx. Applicable for PC based DAQ, TestScale and cDAQ hardware.

#### Validated Hardware

- TS-15050, TS-15120
- NI-9403
- PCIe-6323

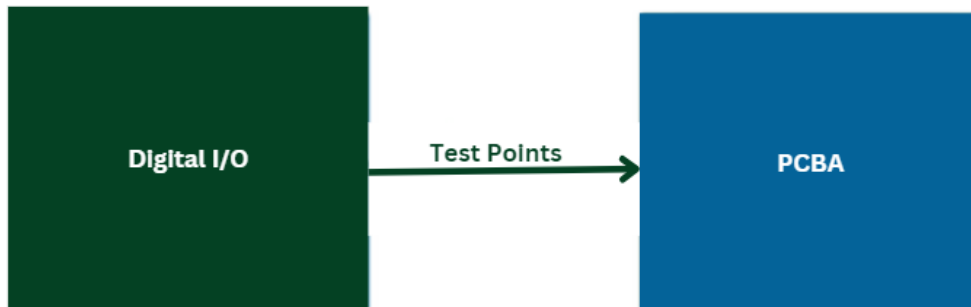
#### Instructions

1. Create an instance of the **DynamicDigitalPatternGeneration()** class using the library.
2. Initialize the DAQmx task by calling **initialize(self, channel\_expression: str)**
3. Configure settings by calling the **configure\_and\_generate(self, configuration: DynamicDigitalPatternGenerationConfiguration, pulse\_signal: np.ndarray)**
4. Configure the **DynamicDigitalPatternGenerationConfiguration** to set the configurations for pattern generation.
  - a. Configure **timing\_parameters** to set sample clock source, sample rate, number of samples per channel and active edge.  
 Generation time = Number of Samples/Sample rate  
**Note:** Maximum sampling rate for TS-15120 is 142 kS/s. Setting values more than that will generate an error.
  - b. Configure **trigger\_parameters** to set the trigger type, digital start trigger source and edge.
5. Configure **pulse\_signal** to set the digital pattern to be written.  
**Note:** DAQmx APIs doesn't evaluate the size of the port but internally ignores the other bits.
6. The Output data includes,
  - a. Generation Time: Total generation time in seconds.
7. Finally close the DAQmx task using **close(self)** method.

**Notes:**

- Dynamic Digital Pattern Generation Library supports **both Port and Digital lines** from a single module to generate dynamic digital patterns. But **Multi-module support is not applicable** on this library and throws errors.
- Triggers are not supported for serial modules (TS-15120 & TS-15130). Refer [Timed digital input/output restrictions](#) to know all the limitations to be considered while performing hardware timed digital tasks.

**Block Diagram**



## 6.2.12 Digital Clock Generation

### Overview

Use the ***DigitalClockGeneration()*** class methods to initialize, configure, Generate and close on user configurable terminals using counters. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15050
- NI-9402
- PCIe-6323

### Instructions

1. Create an instance of the class ***DigitalClockGeneration()*** class using the library.
2. Initialize the DAQmx task by calling ***initialize(self, channel\_expression: str, output\_terminal\_name: str)***

#### Notes:

- Only one counter can be used for a task.
- Provide the Output Terminal on which the signal is to be generated. Refer below on the default terminals of each counter. Users can configure different PFI lines as well.

Counter Selected	Ctr0	Ctr1	Ctr2	Ctr3
Default Terminal for TestScale	PFI3	PFI7	PFI1	PFI5
Default Terminal for cDAQ	PFI3	PFI7	PFI1	PFI5
Default Terminal for PCIe	PFI12	PFI13	PFI14	PFI15

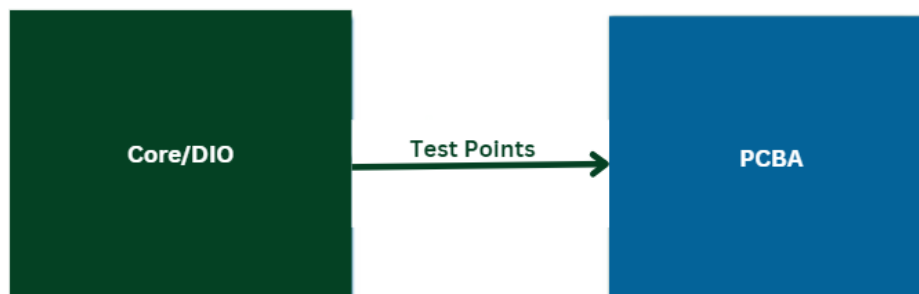
3. Configure settings by calling ***configure\_and\_generate(self, configuration: DigitalClockGenerationConfiguration)*** method with the following parameters:
  - a. Configure the ***channel\_parameters*** to set the frequency and duty cycle.
  - b. Configure the ***clock\_timing\_parameters*** to set the ***clock\_duration\_seconds*** which is the generation time of the clock.
  - c. The output data ***DigitalClockGenerationData()*** includes:
    - Actual Timebase
    - Actual Clock Frequency
    - Actual Clock Duty Cycle
    - Actual Clock Duration Seconds
4. Finally close the DAQmx task using ***close(self)*** method.



**Notes:**

- Multiple counters cannot be initialized in the same task, Use a separate task for each counter.
- Frequency Generator(FreqOut) is not supported as Generation Time and Duty Cycle cannot be controlled. Refer [here](#) for more details.
- It is recommended to set the generation time (s) as an integral multiple of the time period ( $1/\text{Frequency (Hz)}$ ) such that complete cycles are generated.

**Block Diagram**



### 6.2.13 Digital Pulse Generation

#### Overview

Use the **DigitalPulseGeneration** class methods to initialize, configure, generate and close on user configurable terminals using counters. Applicable for PC based DAQ, TestScale and cDAQ hardware.

#### Validated Hardware

- TS-15050
- NI-9402
- PCIe-6323

#### Instructions

1. Create an instance of the class **DigitalPulseGeneration()** class using the library.
2. Initialize the DAQmx task by calling **initialize(self, channel\_expression: str, output\_terminal\_name: str)**

#### Notes:

- Only one counter can be used for a task.
- Provide the Output Terminal on which the signal is to be generated. Refer below on the default terminals of each counter. Users can configure different PFI lines as well.

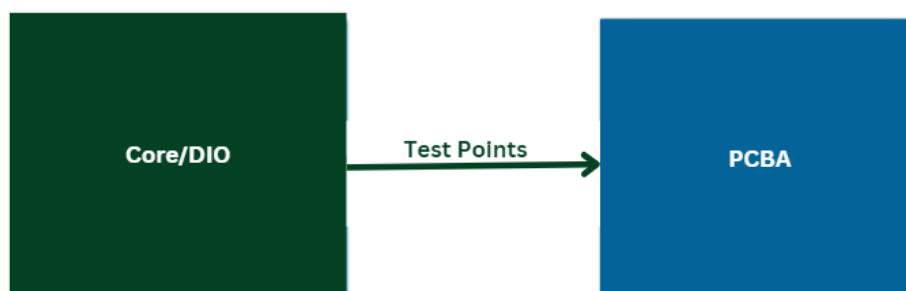
Counter Selected	Ctr0	Ctr1	Ctr2	Ctr3
Default Terminal for TestScale	PFI3	PFI7	PFI1	PFI5
Default Terminal for cDAQ	PFI3	PFI7	PFI1	PFI5
Default Terminal for PCIe	PFI12	PFI13	PFI14	PFI15

3. Configure settings by calling the **configure\_and\_generate(self, configuration: DigitalPulseGenerationConfiguration)** method with the following parameters:
  - a. Configure the **channel\_parameters** to set the low time, high time and the idle state of the pulse.
  - b. Configure the **pulse\_timing\_parameter** to set the pulse count that gives the number of pulses that has to be generated.
  - c. The output data includes (**DigitalPulseGenerationData**)
    - Actual high time
    - Low time
    - Generation time of generated digital signal from instrument.
4. Finally close the DAQmx task using **close(self)** method.

**Notes:**

- Multiple counters cannot be initialized in the same task, Use a separate task for each counter.
- Supported frequency range to generate clock depends on the available Counter Timebase. The available timebases for TestScale are 80 MHz, 20 MHz, and 100 kHz. For PCIe, the time-bases are 100 MHz, 20 MHz, and 100 kHz. For cDAQ, the time-bases are 80 MHz, 20 MHz, and 100 kHz and it is selected based on the input configurations to generate pulse of required frequency.
- Users can choose “Digital Pulse Generation” or “Digital Clock Generation” library based on the input options to match the requirement.

**Block Diagram**



## 6.2.14 Digital Frequency Measurement

### Overview

Use the **DigitalFrequencyMeasurement** class methods to initialize, configure, measure and close on user configurable PFI line using the selected counter for digital frequency measurement. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15050
- NI-9402
- PCIe-6323

### Instructions

1. Create an instance of the class **DigitalFrequencyMeasurement()** class using the library.
2. Initialize the DAQmx task by calling **initialize(self, channel\_expression: str, input\_terminal\_name: str)**

#### Notes:

- Only one counter can be used for a task.
- Counter 0 is always paired with Counter 1. Counter 2 is always paired with Counter 3 for the measurement method used here ([Large Range \(2 Counters\)](#)). Paired counters cannot be used in different tasks.
- Provide input Terminal on which the signal to be measured. Refer below on the default terminals of each counters. User can configure different PFI lines as well.

Counter Selected	Ctr0	Ctr1	Ctr2	Ctr3
Default Terminal for TestScale	PFI3	PFI7	PFI1	PFI5
Default Terminal for cDAQ	PFI3	PFI7	PFI1	PFI5
Default Terminal for PCIe	PFI12	PFI13	PFI14	PFI15

3. Configure settings by calling the **configure\_and\_measure(self, configuration: DigitalFrequencyMeasurementConfiguration)** method with the following parameters:
  - a. Configure the **range\_parameters** to set the minimum and maximum frequency that can be read.
  - b. Configure the **counter\_channel\_configuration\_parameters** to set the range parameter, input divisor for frequency measurement and measurement duration.
  - c. The output data includes:
    - Detected frequency

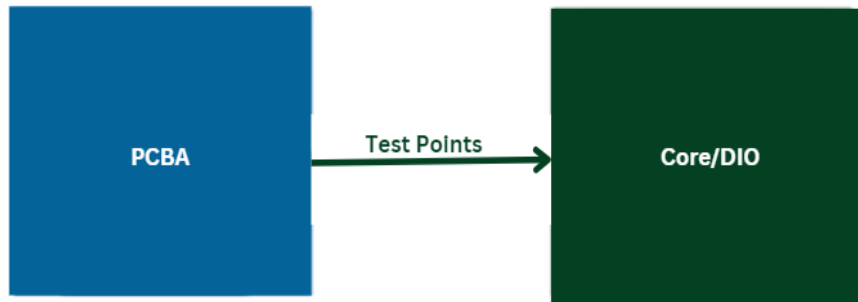
**Note:** Keep in mind that for very low frequencies, actual time taken for the measurement will be N times of a single input cycle.

4. Finally close the DAQmx task using **close(self)** method.

**Notes:**

- There cannot be multiple counters in the same task for input operations. Use a separate task for each counter.
- Large Range (2 counter) measurement method will be used for this measurement. Uses one counter to divide the frequency of the input signal by **Divisor**, creating a lower-frequency signal, more easily measured by the second counter.

**Block Diagram**



## 6.2.15 Digital PWM Measurement

### Overview

Use the **DigitalPwmMeasurement** class methods to initialize, configure, measure and close on user configurable PFI line using the selected counter for digital PWM measurement. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15050
- NI-9402
- PCIe-6323

### Instructions

1. Create an instance of the class **DigitalPwmMeasurement()** class using the library.
2. Initialize the DAQmx task by calling **initialize(self, channel\_expression: str, input\_terminal\_name: str)**

#### Notes:

- Only one counter can be used for a task.
- Global Channel (Counter) input terminal supports only **Semi-Period Counter Input global channels** from NI MAX. The example returns error for other invalid global channels.
- Provide Input Terminal on which the signal to be measured. Refer below on the default terminals of each counter. Users can configure different PFI lines as well.

Counter Selected	Ctr0	Ctr1	Ctr2	Ctr3
Default Terminal for TestScale	PFI3	PFI7	PFI1	PFI5
Default Terminal for cDAQ	PFI3	PFI7	PFI1	PFI5
Default Terminal for PCIe	PFI12	PFI13	PFI14	PFI15

3. Configure settings by calling the **configure\_and\_measure(self, configuration: DigitalPwmMeasurementConfiguration)** method with the following parameters:
  - a. Configure the **range\_parameters** to set the **semi\_period\_maximum\_value\_seconds** and **semi\_period\_minimum\_value\_seconds**. User can determine the Timebase clocks assigned to the counter by updating the Minimum and Maximum Semi-Period(s) values. For more details on TestScale Counter Timebase, visit [TestScale clock routing](#), for more information on PCIe Counter Timebase, visit [X Series User Manual - National Instruments \(ni.com\)](#), for more information on cDAQ-9189 Timebase, visit [cDAQ-9185/9189 User Manual - NI](#).

#### Notes:

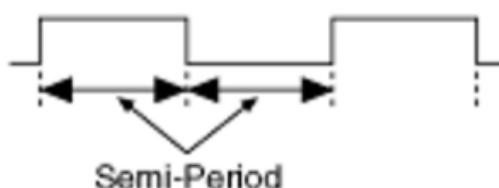
- Initially the counters are reserved with default timebase (For TestScale, the default timebase is **80MHz**, for PCIe the default timebase is **100MHz**, for cDAQ the default timebase is **80MHz**). TestScale, PCIe and cDAQ support three Timebase clocks. The below

table shows the relation between different Timebase and PWM Semi-period duration in TestScale, PCIe and cDAQ.

*Relation between Timebase and semi-period duration*

Devices	Default Timebase	Minimum Semi-Period	Maximum Semi-Period
TestScale	80 MHz	25 ns	53.687091 s
PCIe-6323	100MHz	20 ns	42.949672 s
cDAQ-9189	80MHz	25 ns	53.687091 s

**Note:** “Semi-period” denotes the time between either a rising and falling edge, or a falling and rising edge in a PWM signal.



- b. Configure the **timing\_parameters** to set the **semi\_period\_counter\_wanted\_cycles\_count**.

**Note:** The Default timeout is set to **10 seconds**. If the time elapses before the requested cycles are captured, the program will return error.

- c. Configure the **counter\_channel\_parameters** to set the **semi\_period\_counter\_starting\_edge**.
- d. Configure the **dpwmm\_configuration** to set the **measurement\_options** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.

**Notes:**

- “Measure only” cannot be performed without preceding with “Configure Only”.
- Execution options can be used to capture the entire signal by setting the channel to wait for the first rising edge using Configure Only mode before the generation of input signal. After capturing the cycles, Measure Only mode can be used to read the measured data.

- e. The output of **DigitalPwmMeasurementResultData** returns

- Detected High Time(s)
- Low Time(s)
- Duty Cycle (%)
- Frequency (Hz)

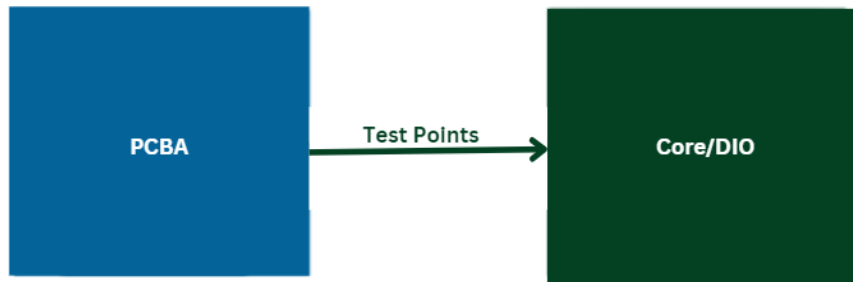
on the terminal. These measurements are averaged results from the captured PWM cycles. Additionally, output returns the actual no of cycles captured during the measurement.

4. Finally close the DAQmx task using **close(self)** method.

**Notes:**

- Multiple counters cannot be initialized in the same task, Use a separate task for each counter.
- NI recommends using **Digital Frequency Measurement** Library for more accurate PWM Frequency measurements.
- Low time for a single pulse cannot be determined as it does not have a clear ending. So, **Minimum cycles required for PWM Measurement is 2.**

**Block Diagram**





## 6.2.16 Digital Edge Count Measurement Using Hardware Timer

### Overview

Use the **DigitalEdgeCountMeasurementUsingHardwareTimer** class methods to initialize, configure, measure and close on user configurable PFI line using the selected counter for Edge Counting. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15050
- NI-9402
- PCIe-6323

### Instructions

1. Create an instance of the **DigitalEdgeCountMeasurementUsingHardwareTimer()** class using the library.
2. Initialize the DAQmx task by calling the **initialize(self, measurement\_channel\_expression: str, measurement\_input\_terminal\_name: str, timer\_channel\_expression: str)** method on the class instance.
  - Two tasks are used for Count Digital Events library. Counter task is used for edge counting and Timer task is used to control measurement time precisely, used only for hardware timed wait type.
  - For **Counter task**, provide a counter resource and Input terminal to listen for digital events / edges. Refer below on the default terminals of each counter. Users can configure different PFI lines as well.

Counter Selected	Ctr0	Ctr1	Ctr2	Ctr3
Default Terminal for TestScale	PFI3	PFI7	PFI1	PFI5
Default Terminal for cDAQ	PFI3	PFI7	PFI1	PFI5
Default Terminal for PCIe	PFI12	PFI13	PFI14	PFI15

- For **Timer task**, provide a counter resource to generate the measurement window for **hardware timed wait type**. No external physical connections are required. Generated pulse will be internally used by Counter task.
  - Get both Counter and Timer tasks from DAQmx Tasks.
3. Configure settings by calling the **configure\_and\_measure(self, configuration: DigitalEdgeCountHardwareTimerConfiguration)** method with the following parameters:
    - a. Configure the **decm\_configuration** to set the **measurement\_options** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.  
**Note:** "Measure only" cannot be performed without preceding with "Configure Only".
    - b. Configure the **counter\_channel\_parameters** to set the **edge\_type**. User can set the edge type either to RISING or to FALLING.
    - c. Configure the **timing\_parameters** to set the **edge\_counting\_duration** which is the time for which the library will be counting the edges.

- d. Configure the **trigger\_parameters** to set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
- e. The Output data (**DigitalEdgeCountMeasurementResultData**) returns the number of edge counts at the end of measurement duration.

**Note:** Refer next section for more details on the configurations.

4. Finally close the DAQmx task using **close(self)** method.

#### Notes:

- Measure Only call for software timed wait type will not wait for the specified duration. Instead, it will return the current counter value immediately.

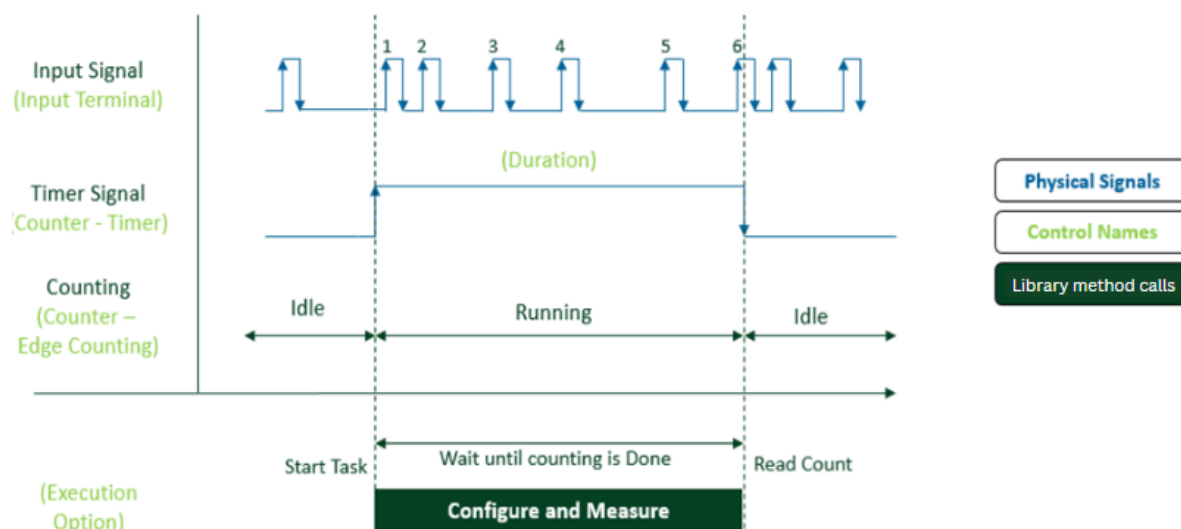
### Hardware Timed Digital Edge Counting

Timer Task is used to control the count measurement time precisely by generating a single pulse of specified duration which is used to start/stop the counting process. This is very useful in the case of high frequency inputs where we need the precise measurement time for edge counting.

#### Without Trigger

Below timing diagram explains the hardware timed digital edge counting. Here, “Counter – Timer” is used to control the measurement time by generating the pulse signal of “Duration” specified by the user. “Counter – Edge Counting” is used to count the edges in the input signal.

**Configure and Measure** call starts the edge counting, wait for the given duration to complete (counting happens until the timer pulse signal goes low) and measures the counter value at the end.

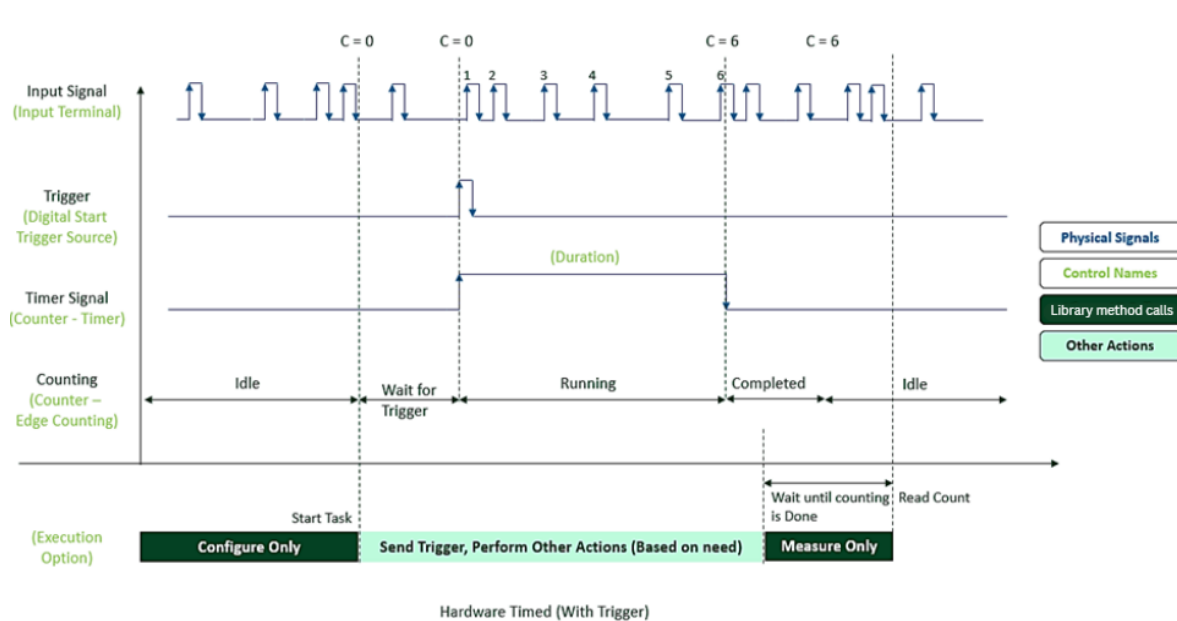


## With Trigger

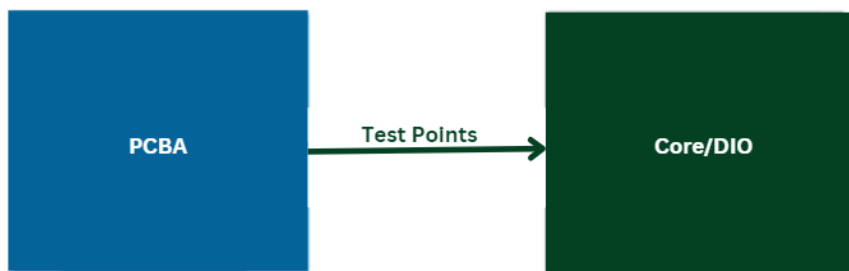
Below timing diagram explains the hardware timed digital edge counting in triggered scenario. Here, the start of the timer (pulse signal generation) is controlled by the user using “Digital Start Trigger Source”.

**Configure Only** call configures the trigger, starts the task and waits for the trigger to start the timer and edge counting. User can now send the trigger and perform any other actions.

**Measure Only** call waits for the measurement duration to complete and reads the counter value.



## Block Diagram



## 6.2.17 Digital Edge Count Measurement Using Software Timer

### Overview

Use the ***DigitalEdgeCountMeasurementUsingSoftwareTimer*** class methods to initialize, configure, measure and close on user configurable PFI line using the selected counter for Edge Counting. Applicable for PC based DAQ, TestScale and cDAQ hardware.

### Validated Hardware

- TS-15050
- NI-9402
- PCIe-6323

### Instructions

1. Create an instance of the ***DigitalEdgeCountMeasurementUsingSoftwareTimer()*** class using the library.
2. Initialize the DAQmx task by calling the ***initialize(self, measurement\_channel\_expression: str, measurement\_input\_terminal\_name: str)*** method on the class instance.
  - For **Counter task**, provide a counter resource and Input terminal to listen for digital events / edges. Refer below on the default terminals of each counter. Users can configure different PFI lines as well.

Counter Selected	Ctr0	Ctr1	Ctr2	Ctr3
Default Terminal for TestScale	PFI3	PFI7	PFI1	PFI5
Default Terminal for cDAQ	PFI3	PFI7	PFI1	PFI5
Default Terminal for PCIe	PFI12	PFI13	PFI14	PFI15

- Get both Counter and Timer tasks from DAQmx Tasks.
3. Configure settings by calling the ***configure\_and\_measure(self, configuration: DigitalEdgeCountSoftwareTimerConfiguration)*** method with the following parameters:
    - Configure the ***decn\_configuration*** to set the ***measurement\_options*** to specify the mode of execution: Configure & Measure, Configure only, or Measure only.

**Note:** “Measure only” cannot be performed without preceding with “Configure Only”.

- Configure the ***counter\_channel\_parameters*** to set the ***edge\_type***. User can set the edge type either to RISING or to FALLING.
- Configure the ***timing\_parameters*** to set the ***edge\_counting\_duration*** which is the time for which the library will be counting the edges.
- Configure the ***trigger\_parameters*** to set triggers to start measurement on various trigger events. Configure no trigger for Software Timer.
- The Output data(***DigitalEdgeCountMeasurementResultData***) returns the number of edge counts at the end of measurement duration.

**Note:** Refer next section for more details on the configurations.

- Finally close the DAQmx task using ***close(self)*** method.

**Note:**

- Measure Only call for software timed wait type will not wait for the specified duration. Instead, it will return the current counter value immediately.

### Software Timed Digital Edge Counting

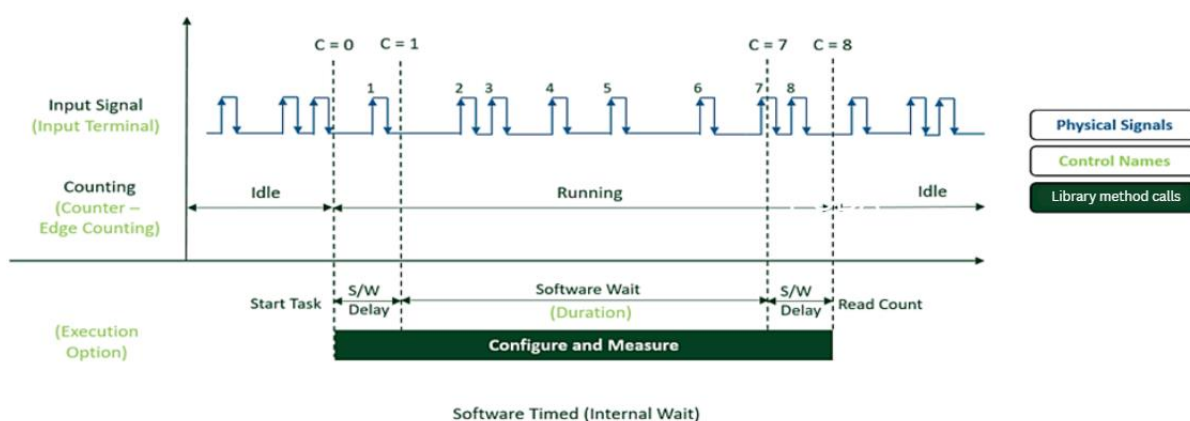
In this method, Counting starts as soon as the task is started. There should be a software wait for the “Duration” before reading the counter value. This can be done internally or externally which are explained below.

#### Internal Wait

Below timing diagram explains the software timed digital edge counting where software wait is internally added by the methods.

**Configure and Measure** call starts the edge counting, waits for given duration (software wait) and measures the counter value at the end. Here the delays produced by the software can affect the counting process and count might not be precise.

This is very useful in the case of low frequency inputs where we don’t need the precise measurement time for edge counting and save the extra counter resources by skipping the “Counter-Timer” input required for hardware timed execution.



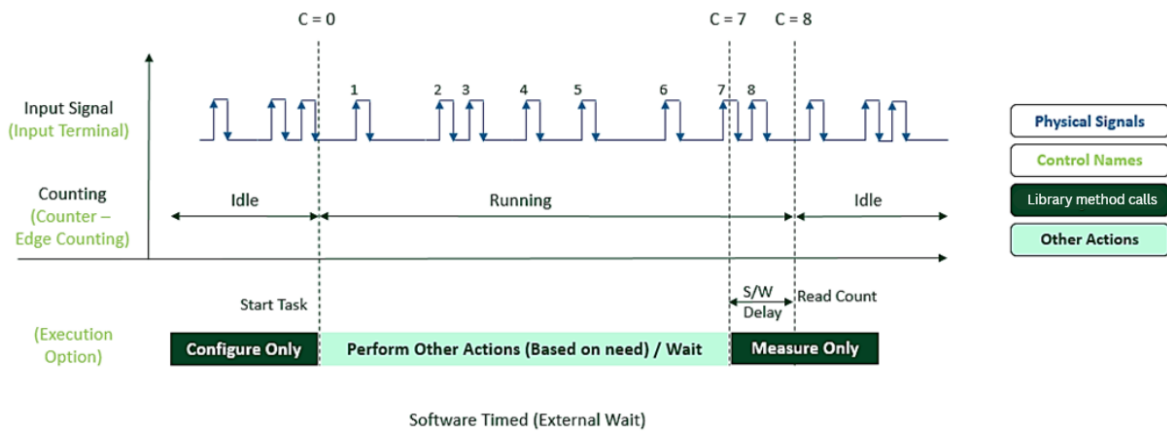
#### External Wait

Below timing diagram explains the software timed digital edge counting where software wait is externally added by the user. Methods does not wait during measurement.

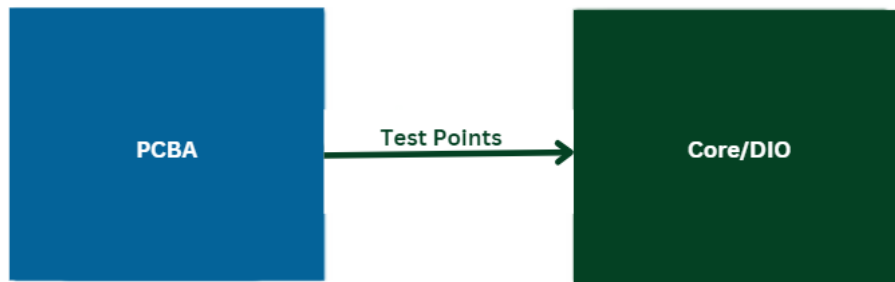
**Configure Only** call starts edge counting. User can wait for the required time in the test layer (or perform any other actions) and call the **Measure Only** to read count.

**Measure Only** call reads the current counter value and return the output.

This is much useful in the case of low frequency inputs where the measurement time is higher. User can start the task, perform any other tasks and come back to measure later (wait and measure can also be done in separate thread).



## Block Diagram



### 6.2.18 Synchronization

#### Overview

Use the ***SynchronizationSignalRouting()*** class methods to route signals between specified source and output terminals for the given DAQmx Task.

#### Validated Hardware

- PCIe-6323
- cDAQ NI 9402
- TS-15050
- TS-15100
- TS-15110
- NI TS-15120
- TS-15130

#### Instructions

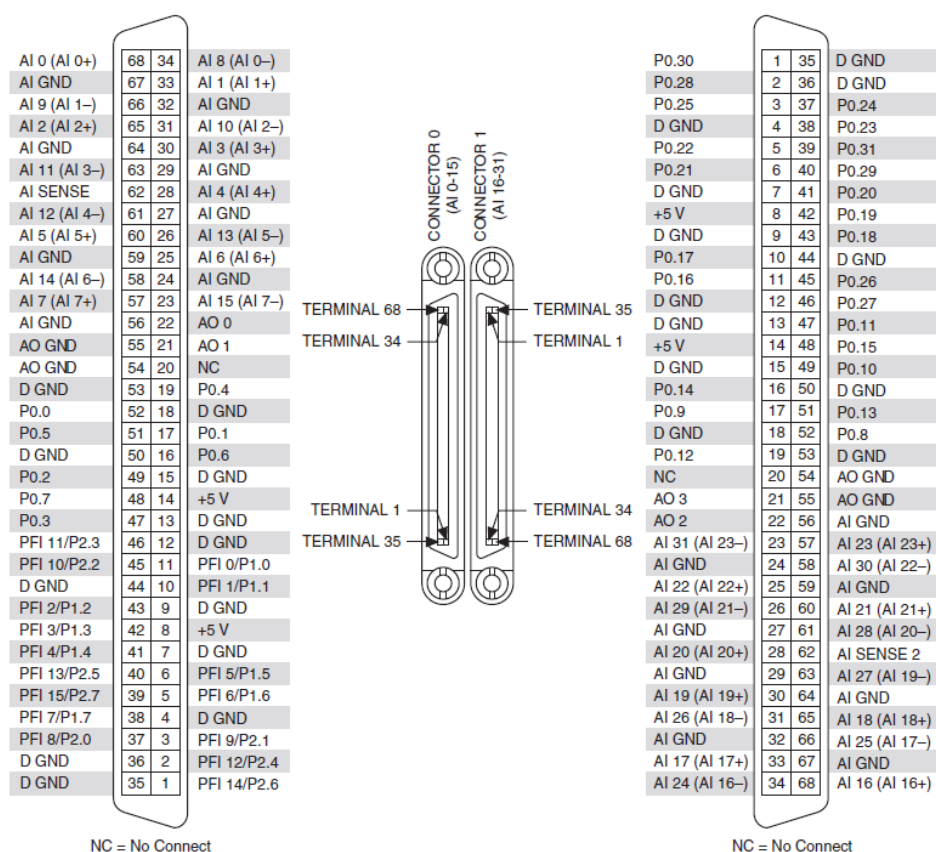
1. Create an instance of any class ***SignalVoltageGeneration()***, ***DynamicDigitalPatternGeneration()*** ) to configure source device.
2. Call the ***initialize(\*args)*** method to initialize the DAQ device for generation using the class object.
3. Use the Synchronization class methods available in the class to configure settings mentioned below:
  - a. ***route\_start\_trigger\_signal\_to\_terminal(terminal\_name:str)*** method which is used to route the start trigger signal to the specified terminal.
  - b. ***route\_sample\_clock\_signal\_to\_terminal(terminal\_name:str)*** method which is used to route the sample clock signal to the specified terminal.
  - c. The routes created by these methods are embedded in a task. Once the DAQmx task is committed, the route is committed to the targeted device. When the task is cleared, the route is unreserved. This routing method is termed as [Task-Based Routing](#).
  - d. Refer to the Device Routes table in Measurement and Automation Explorer to determine eligible signals for routing for the targeted device.
4. Use the ***configure\_and\_measure()*** and ***close()*** methods to configure DAQ settings using the respective class objects.

#### Notes:

- Refer “synchronization\_tests” sequence for an example demonstrating synchronization using the library.
- For more details on Signal Routing refer [DAQmx Signal Routing](#) concepts

- The source signal and output terminals can be on different devices as long as a connecting public bus, such as TestScale backplane, connects the devices.
- In PC Based DAQ, external **PFI lines** can be found on the connector. For more details for each device, refer to the [X Series User Manual](#).
- In CompactDAQ, you can use and select the **PFI lines** available at the front panel of some backplanes or add a [NI 9402](#) BNC high speed digital module in the backplane to route and select PFI lines for trigger and Sample clock externally. For more details refer digital PFI section [of CompactDAQ backplane user manual](#).
- In TestScale, certain internal signals can be routed and exposed to external **PFI Lines**. For more details refer [Digital Routing](#) concepts in TestScale.
- AI or DI Sample Clock and few Triggers can be route to any output PFI terminal. Sample Clock is an active high pulse by default. For more details, refer [AI and DI Timing Signals](#).
- AO or DO Sample Clock and few Triggers can be route to any output PFI terminal. Sample Clock is an active high pulse by default. For more details, refer [AO and DO Timing Signals](#).

## PCIe-6323 Pinout



### Note:

- In PC Based DAQ Devices, for example, as shown in this PCIe-6323 block diagram, **PFI 0...15** channels are mapped to **P1.0 to P1.7** and **P2.0 to P2.7** terminals. For more details regarding Digital I/O/PFI characteristics, visit [PCIe-6323 specifications](#).



## 6.2.19 Temperature RTD Measurement

### Overview

Use ***TemperatureMeasurementUsingRtd*** class methods to initialize, configure, measure and close on user configurable Analog input pins to derive temperature measurements from RTDs (Resistance Temperature Detector). This library is applicable on C Series Temperature Input Modules.

### Validated Hardware

- NI C Series/cDAQ Temperature Input Module (NI-9217)

### Instructions

1. The library requires an internal/external **current excitation (I<sub>ex</sub>)** to be applied across the RTD for temperature measurements. For more info on connections, refer the [Block Diagram](#) section.

2. Create an instance of the ***TemperatureMeasurementUsingRtd()*** class using the library.

3. Initialize the DAQmx task by calling the ***initialize(self, channel\_expression: str)*** method on the class instance.

**Note:** Global channel specified in the ***initialize()*** can be calibrated in NI MAX to provide more accurate measurements. For more details refer, [Calibration](#) section.

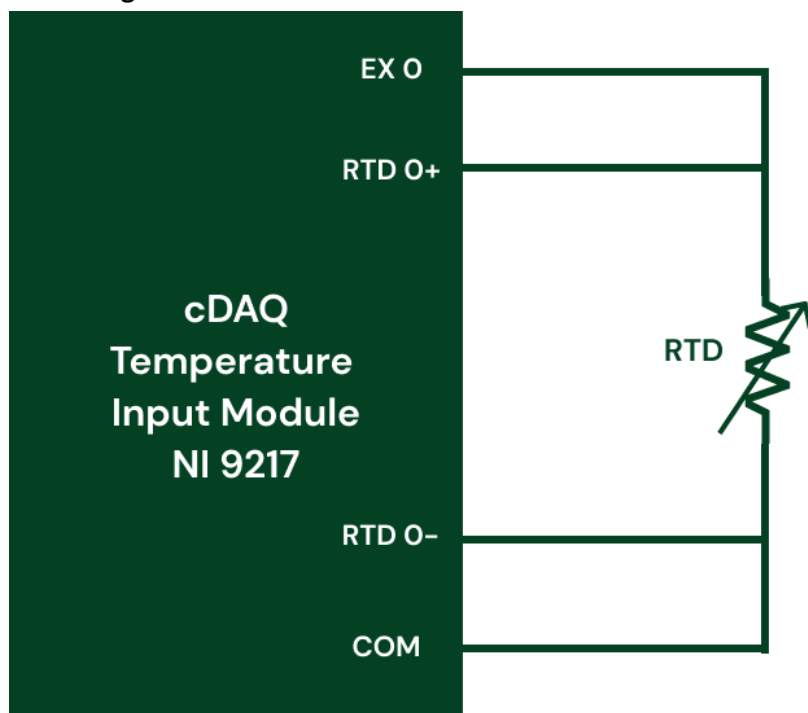
4. Configure settings by calling the ***configure\_and\_measure(self, configuration: TemperatureRtdMeasurementConfiguration)*** method with the following parameters:
  - a. Configure the ***measurement\_execution\_type (MeasurementExecutionType)*** property to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
  - b. Configure ***global\_channel\_parameters (TemperatureRtdMeasurementTerminalParameters)***:  
Set parameters to configure the range and terminal settings for all analog input temperature channels, excluding channels specified in the specific channel settings.
  - c. Configure ***specific\_channels\_parameters (List[TemperatureRtdMeasurementChannelParameters])***:  
Set parameters for specific channels individually. If Global Virtual Channel name is provided in ***initialize()***, provide the Virtual Channel name in Specific Channel Parameters as well. Similarly, if a Physical Channel name is provided in ***initialize()***, provide the Physical Channel name in Specific Channel Parameters. If no channels are specified, global settings are applied.  
**Note:** RTD parameters can be configured globally or specifically to each channel.
  - d. Configure ***sample\_clock\_timing\_parameters (SampleClockTimingParameters)***:  
Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.  
Acquisition time = Number of Samples / Sampling Rate.  
**Note:** Maximum aggregate sampling rate for NI 9217 is 400 S/s. Setting values more than that will generate an error.

- e. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:  
Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
  - f. The Output data includes (***TemperatureMeasurementResultData***):
    - Temperature Waveforms - Displays the captured Temperature waveforms for each channel configured in the DAQmx Task.
    - Temperature Measurements - Provides the derived post-analyzed data from the captured Temperature waveforms to output Averaged Temperature in deg Celsius and Kelvin representations.
    - Acquisition Time – Calculated acquisition time for the measurement.
5. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

- The library only supports external/internal current excitation for RTDs. Voltage excitation is not supported by this library.
- Any other external source can be used for current excitation in RTD Temperature Measurements.
- For Fast measurements, set maximum sample rate possible for the given "Sample Clock Source" input and assign smaller "Number of Samples" to capture. Min sample size is 2.
- For better resolution, configure the instrument to read more samples e.g., 1000 samples.
- NI recommends using 4-wire method for measuring voltage drop across RTD to derive more accurate temperature measurements. For more details, visit [4 - Wire Resistance](#).
- Library supports only C-Series Temperature Input devices. For more details on Thermistor and RTD Measurements, refer [Making an RTD or Thermistor Measurement in LabVIEW](#).

### Block Diagram



## 6.2.20 Temperature Thermistor Measurement

### Overview

Use *TemperatureMeasurementUsingThermistor* class methods to initialize, configure, measure and close on user configurable Analog input pins to derive temperature measurements from voltage excited NTC typed Thermistor devices. This library is applicable for TestScale analog input modules, C Series Voltage Input Modules, PC based devices with Input channels.

### Validated Hardware

- TestScale Analog Input Module (TS-15100)
- C Series/cDAQ Voltage Input Module (NI-9215)
- PC based DAQ (PCIe-6323)

### Instructions

1. Library requires an **external Voltage excitation (Vex)** to be applied across the Thermistor for temperature measurements. The voltage excitation must be applied to the Thermistor through a **Dropping resistor (R1)**. For more info on connections, refer the [Block Diagram](#) section.
2. Create an instance of the *TemperatureMeasurementUsingThermistor()* class using the library.

3. Initialize the DAQmx task by calling the *initialize(self, channel\_expression: str)* method on the class instance.

**Note:** Global channel specified in the *initialize()* can be calibrated in NI MAX to provide more accurate measurements. For more details refer, [Calibration](#) section.

4. Configure settings by calling the *configure\_and\_measure(self, configuration: TemperatureThermistorMeasurementConfiguration)* method with the following parameters:
  - a. Configure the *measurement\_execution\_type (MeasurementExecutionType)* property to specify the mode of execution: Configure & Measure, Configure only, or Measure only.
  - b. Configure *global\_channel\_parameters (TemperatureThermistorRangeAndTerminalParameters)*:  
Set parameters to configure the range and terminal settings for all analog input temperature channels, excluding channels specified in the specific channel settings.

**Note:** The library supports both **A B C parameter** and **Beta parameter** for Thermistor Temperature measurements. For more details, refer the [Measurement Details](#) section.

- c. Configure *specific\_channels\_parameters (List[TemperatureThermistorChannelRangeAndTerminalParameters])*:  
Set parameters for specific channels individually. If Global Virtual Channel name is provided in *initialize()*, provide the Virtual Channel name in Specific Channel Parameters as well. Similarly, if a Physical Channel name is provided in *initialize()*,

provide the Physical Channel name in `SpecificChannelParameters`. If no channels are specified, global settings are applied.

**Note:** Thermistor parameters and Voltage excitation settings can be configured globally or specifically to each channel.

d. Configure ***sample\_clock\_timing\_parameters (SampleClockTimingParameters)***:

- Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.  
Acquisition time = Number of Samples / Sampling Rate.

**Notes:**

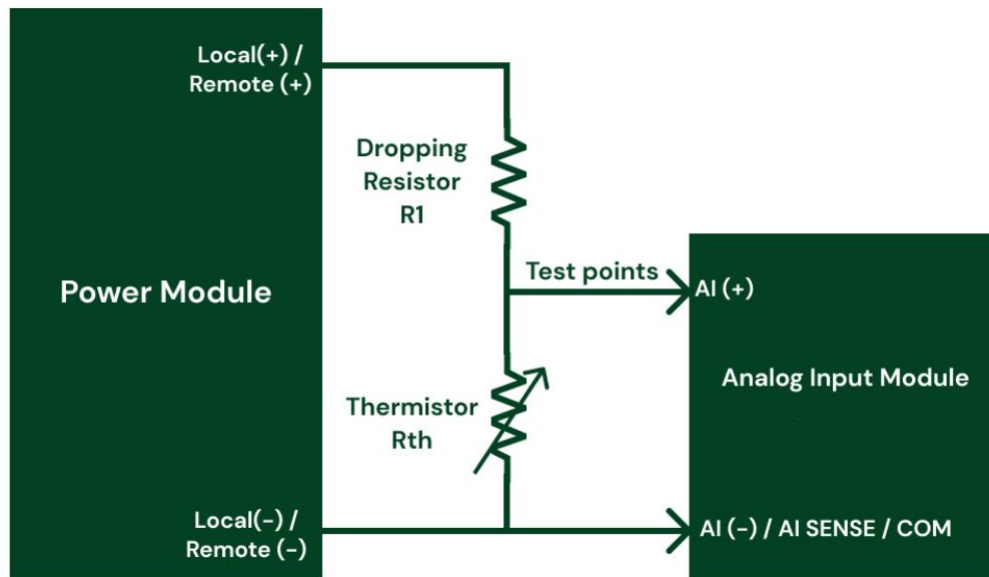
- The timing engine configurations are the same for both TestScale and cDAQ chassis. For more details refer [cDAQ-91xx and TestScale Chassis Timing Engines](#).
  - PCIe does not support sample timing engine
- e. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:  
Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger. If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
- f. The Output data includes ***(TemperatureMeasurementResultData)***:
- Temperature Waveforms - Displays the captured Temperature waveforms for each channel configured in the DAQmx Task.
  - Temperature Measurements - Provides the derived post-analyzed data from the captured Temperature waveforms to output Averaged Temperature in deg Celsius and Kelvin representations.
  - Acquisition Time – Calculated acquisition time for the measurement.

5. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

- The library only supports external voltage excitation for the thermistor temperature measurements. Current excitation is not supported by this library.
- Any other external source can be used for Voltage excitation in Thermistor measurements.
- For Fast measurements, set maximum sample rate possible for the given "Sample Clock Source" input and assign smaller "Number of Samples" to capture. Min sample size is 2.
- For better resolution, configure the instrument to read more samples e.g., 1000 samples.
- NI recommends using remote sensing method in power supply modules for more reliable voltage excitation values. For more details, visit [Remote Sense](#).
- Library extends support for C-Series, PC based DAQ and TestScale devices.
- For more details on Thermistor and RTD Measurements, refer [Making an RTD or Thermistor Measurement in NI-MAX - NI](#).

## Block Diagram



For more details regarding specifications, refer to [References section](#).

## Measurement Details

Library derives Temperature measurements from Thermistor parameters using different methodologies. They are,

### 1. A B C Type

- a. Specifies the Steinhart-Hart Equation parameters used to perform temperature measurements.
- b. It includes three parameters – A, B and C. These parameters are provided by the thermistor manufacturers. Refer the thermistor datasheet to get its respective values.
- c. The parameters are substituted in the below Steinhart-Hart Equation,

$$\frac{1}{T} = A + B (\ln R) + C (\ln R)^3$$

Where,

T = Temperature in Kelvins

R = Measured Thermistor Resistance in Ohm

A, B, C = Constants provided by the thermistor manufacturer

## 2. Beta Type

- a. Characterised with the  $\beta$  (Beta) parameter equation which is essentially Steinhart-Hart equation where A, B and C constants are substituted with  $\beta$ ,  $T_0$  and  $R_0$ . Refer below,

$$A = \frac{1}{T_0} - \left(\frac{1}{\beta}\right) \ln R_0$$

$$B = \frac{1}{\beta}$$

$$C = 0$$

Substituting the above values in Steinhart-Hart equation gives the Beta parameter equation,

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{\beta} \ln \left(\frac{R}{R_0}\right)$$

Where,

T = Temperature in Kelvin

R = Measured Thermistor Resistance in Ohm

$\beta$  = Beta parameter provided by thermistor manufacturers in Kelvin

$R_0$  = Thermistor Resistance (in Ohm) in temperature  $T_0$

$T_0$  = 298.15 Kelvin (equivalent to 25°C)

### Note:

- **A B C parameter-based** Temperature measurements gives **more accurate results** compared to Beta parameter method. Since the parameter “C” constant is substituted to “Zero” in Steinhart-Hart equation for approximation purposes during Beta parameter conversions.

For more details, refer [Thermistor-Wiki](#)

## 6.2.21 Temperature Thermocouple Measurement

### Overview

Use *TemperatureMeasurementUsingThermocouple()* class methods to initialize, configure, measure and close on user configurable Analog input pins to derive temperature measurements from Thermocouples. This library is applicable for C Series Temperature Input Modules.

### Validated Hardware

- NI C Series/cDAQ Temperature Input Module (NI-9211)

### Instructions

1. Create an instance of the *TemperatureMeasurementUsingThermocouple* class using the library.
2. Initialize the DAQmx task by calling the *initialize(self, channel\_expression: str, cold\_junction\_compensation\_channel: str, cold\_junction\_compensation\_source: nidaqmx.constants.CJCSource)* method on the class instance.

#### Notes:

- The *initialize()* method requires Cold Junction Compensation details for accurate Temperature measurement. User can use Built In / Constant / Channel based CJC Inputs.
- To setup external channel based CJC measurements, refer [Cold Junction Compensation \(CJC\) on DAQ Hardware Without Built in CJC](#).
- Global channel specified in the Initialize VI can be calibrated in NI MAX to provide more accurate measurements. For more details refer, [Calibration](#) section.

3. Configure settings by calling the *configure\_and\_measure(self, configuration: TemperatureThermocoupleMeasurementConfiguration)* method with the following parameters:
  - a. Configure the *measurement\_execution\_type (MeasurementExecutionType)* property to specify the mode of execution: Configure & Measure, Configure only, or Measure only.

**Note:** Measure only cannot be repeated without preceding with the "Configure Only".

- b. Configure *global\_channel\_parameters (TemperatureThermocoupleMeasurementTerminalParameters)*:  
Set parameters to configure the range and terminal settings for all analog input temperature channels, excluding channels specified in the specific channel settings.

#### Notes:

- CJCSource and CJC Channel are specified for Cold Junction Compensation (CJC).
- Autozero is provided to compensate for internal offsets during temperature measurements. The compensation in turns increases the accuracy of measurements.
- c. Configure *specific\_channels\_parameters (List[TemperatureThermocoupleChannelRangeAndTerminalParameters])*:



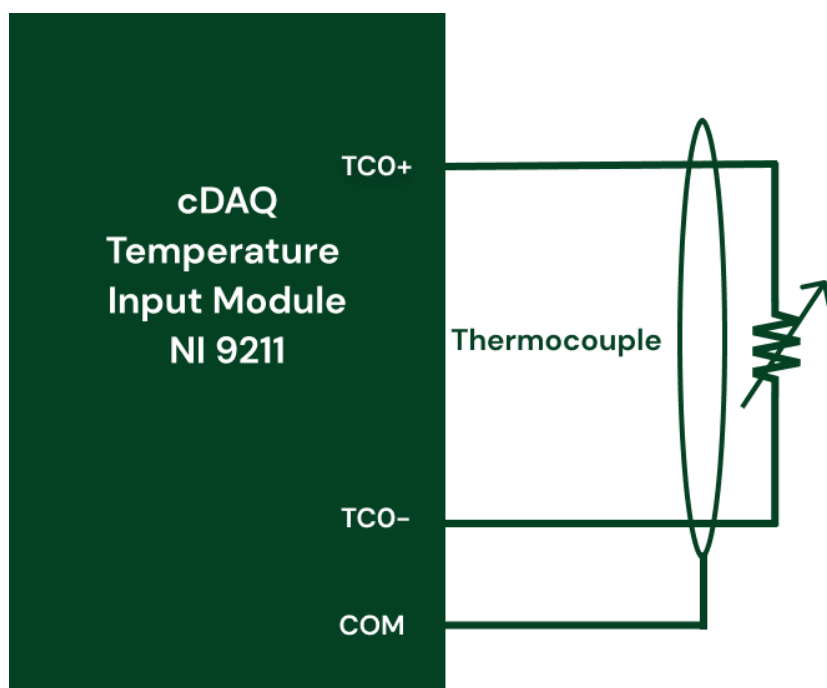
- Set parameters for specific channels individually. If Global Virtual Channel name is provided in ***initialize()***, provide the Virtual Channel name in Specific Channel Parameters as well. Similarly, if a Physical Channel name is provided in ***initialize()***, provide the Physical Channel name in Specific Channel Parameters. If no channels are specified, global settings are applied.
  - d. Configure ***sample\_clock\_timing\_parameters (SampleClockTimingParameters)***:
    - Set the timing configuration for sample clock source input, sampling rate, number of samples, and sample timing engine used for the DAQmx task. Use the "Auto" setting to automatically select the timing engine.  
Acquisition time = Number of Samples / Sampling Rate.

**Note:** Maximum aggregate sampling rate for NI 9211 is 14 S/s. Setting values more than that will generate an error.
  - e. Configure ***digital\_start\_trigger\_parameters (DigitalStartTriggerParameters)***:
    - Set triggers to start measurement on various trigger events. Configure either no trigger or a digital trigger.
    - If using a digital trigger, provide a suitable digital start trigger source. Set the digital start trigger edge as Rising or Falling.
  - f. The Output data includes ***(TemperatureMeasurementResultData)***:
    - Temperature Waveforms - Displays the captured Temperature waveforms for each channel configured in the DAQmx Task.
    - Temperature Measurements - Provides the derived post-analyzed data from the captured Temperature waveforms to output Averaged Temperature in deg Celsius and Kelvin representations.
    - Acquisition Time – Calculated acquisition time for the measurement.
4. Finally close the DAQmx task using ***close(self)*** method.

**Notes:**

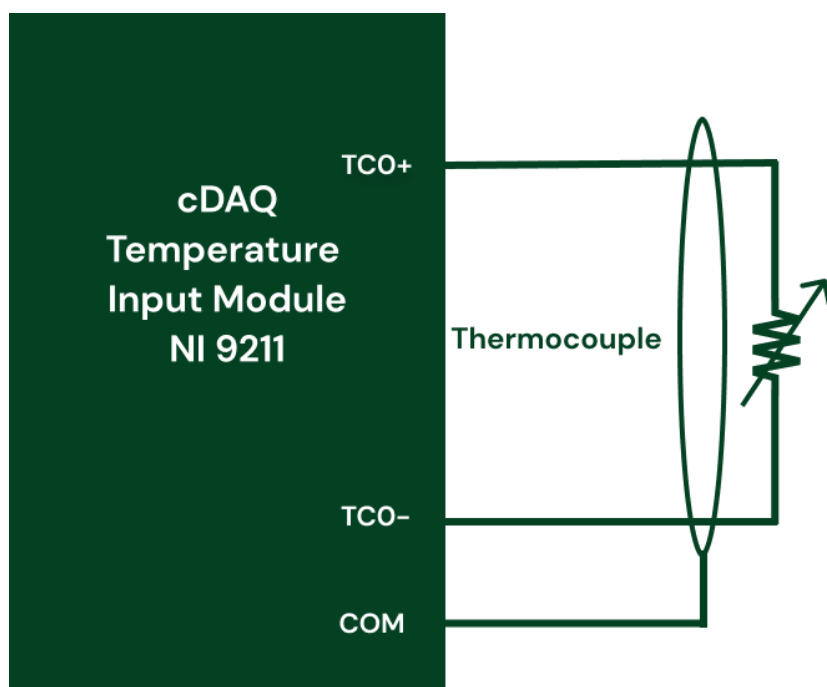
- For Fast measurements, set maximum sample rate possible for the given "Sample Clock Source" input and assign smaller "Number of Samples" to capture. Min sample size is 2.
- For better resolution, configure the instrument to read more samples e.g., 10 samples. Maximum aggregate sampling rate for cDAQ Temperature input modules like NI 9211 is 14 S/s. Hence, requesting more samples to read will increase the measurement acquisition time.
- Library supports only C-Series Temperature Input devices. NI TestScale Modules are not supported by this library.
- NI recommends following various signal conditioning requirements for accurate temperature measurements from Thermocouple devices. For more details refer, [Signal Conditioning Requirements for Thermocouples](#).
- For more details, on Thermocouple measurements using NI DAQ devices. Refer [Taking a Thermocouple Measurement in LabVIEW](#).
- Use of Physical channels is recommended for this library as Global virtual channel properties cannot be overwritten due to lack of setter method in nidaqmx-python.

### Block Diagram



For more details regarding specifications, refer to [References section](#).

### Block Diagram



For more details regarding specifications, refer to [References section](#).

## 6.3 Communications Libraries

### Overview

Communication Library can be used to perform data read and write operations through I2C, SPI and Serial Comm Port. This library is applicable for NI Hardware's like USB-8452 and USB-232.

### Hardware Requirements

- USB-8452
- USB-232

### Software Requirements

- NI 845x Driver 2022 or later
- NI Serial Driver 2023 or later
- NI VISA Driver 2023 or later

### Instructions

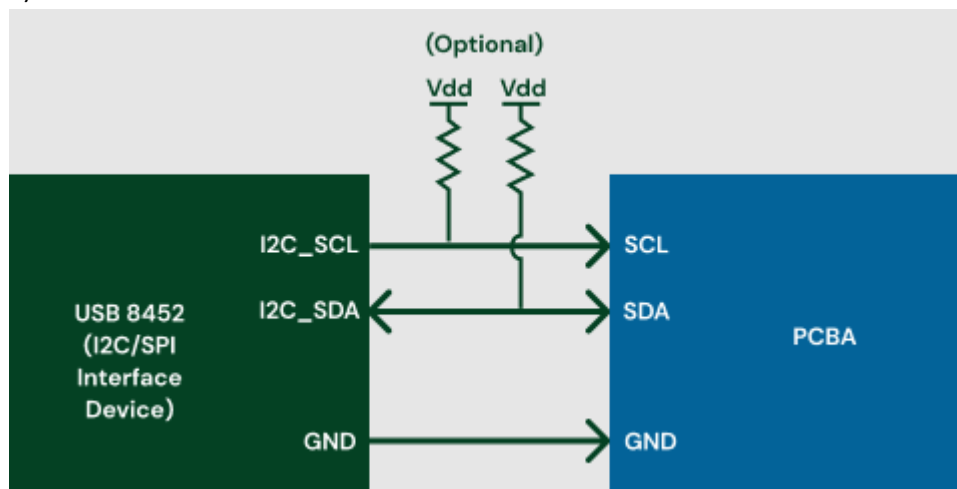
1. Classes *ni\_845x\_i2c\_communication\_devices* and *ni\_845x\_spi\_communication\_devices* in the communication library can be used for Read/Write operations through I2C, SPI and Serial Comm Port.
2. Use "*I2cReadCommunication()*" and "*I2cWriteCommunication()*" class methods for simple read and write data operations through I2C protocol communication using NI 845x Device.
  - a. Initialize the i2c communication library by creating the instances of *I2cReadCommunication()* and *I2cWriteCommunication()* and using *initialize()* method on the objects.
  - b. Configure the setting to read data by calling the *configure\_and\_read\_data(self, configuration: I2cReadCommunicationConfiguration)* method with *device\_parameters*, *communication\_parameters* and *read\_parameters*.
  - c. Configure settings to write data by calling the *configure\_and\_write\_data(self, configuration: I2cWriteCommunicationConfiguration)* method with *device\_parameters*, *communication\_parameters* and *write\_parameters*.
  - d. Use the *close()* method to close all tasks and release allocated resources.
3. Use "*SpiReadCommunication()*" and "*SpiWriteCommunication()*" class methods for simple read and write data operations through I2C protocol communication using NI 845x Device.
  - a. Initialize the i2c communication library by creating the instances of *SpiReadCommunication()* and *SpiWriteCommunication()* and using *initialize()* method on the objects.
  - b. Configure settings by calling the *configure\_and\_read\_data(self, configuration: SpiReadCommunicationConfiguration,)* method with *device\_parameters*, *communication\_parameters* and *read\_parameters*.

- c. Configure settings by calling the ***configure\_and\_write\_data(self, configuration: SpiWriteCommunicationConfiguration)*** method with ***device\_parameters***, ***communication\_parameters*** and ***write\_parameters***.
  - d. Use the ***close()*** method to close all tasks and release allocated resources.
4. Use ***SerialCommunication()*** class methods for simple read and write data operations through Serial communication using USB-232 Device.
  - a. Initialize the DAQmx task by calling the ***initialize (self, serial\_device\_name: str)*** method on the class instance.
  - b. Configure and then perform serial communication operations according to specific configuration using the ***configure\_then\_send\_command\_and\_receive\_response (configuration: SerialCommunicationConfiguration)*** method.
    - i. Configure settings by using the ***SerialCommunicationConfiguration(communication\_parameters: SerialCommunicationParameters, command\_to\_send: str)***.
  - c. The Output data includes (***SerialCommunicationData***) response in the str format.
  - d. Use the ***close()*** method to close all tasks and release allocated resources.

### Block Diagram

Refer the following Block diagram applicable for each communication,

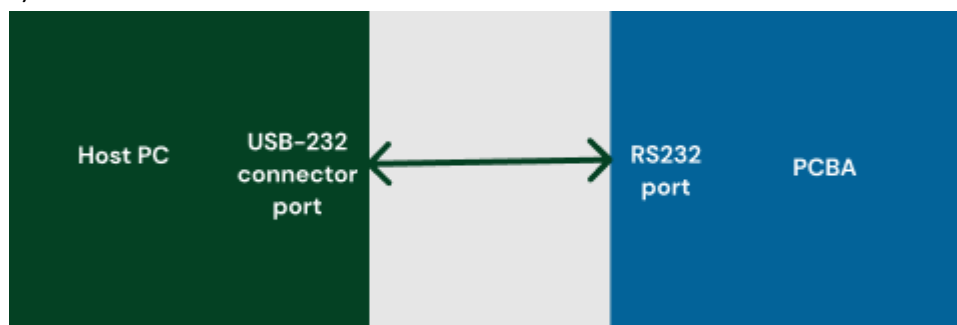
#### a) I2C Communication



#### b) SPI Communication

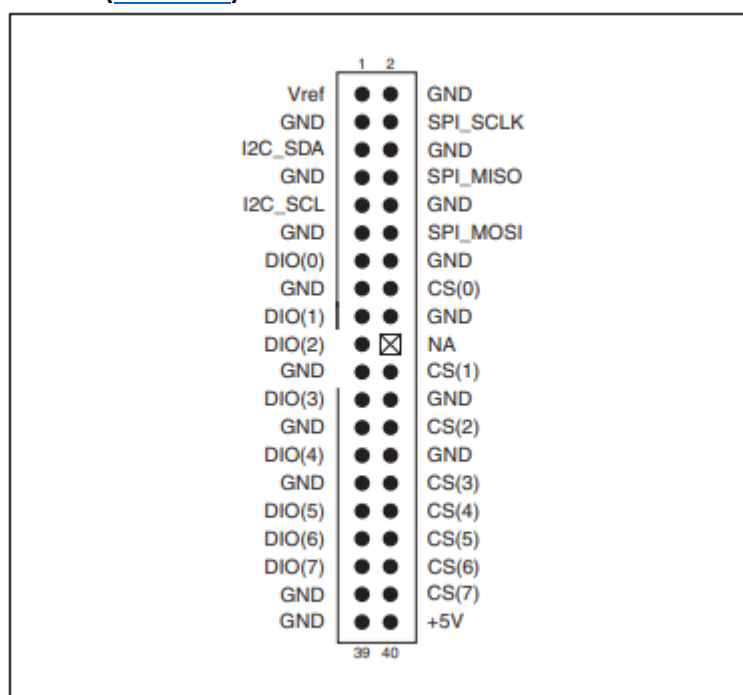


### c) Serial Communication



### Pinouts of Device

#### I2C/SPI Interface Device ([USB-8452](#))



For more details refer – [NI-845x Hardware and Driver Software Getting Started Guide](#)

### References

Below are few other references for communication library,

1. NI 845x Software and Hardware Installing Procedure - <https://www.ni.com/docs/en-US/bundle/ni-845x-software-hardware-installing/resource/371708c.pdf>
2. NI 845x Example location - <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000wyG5CAI&l=en-IN>
3. NI USB-232 Serial Getting Started Guide - <https://www.ni.com/docs/en-US/bundle/ni84xx-usb-232-485-getting-started/resource/371583h.pdf>
4. Set Up Communication with Serial Interface - [Set Up Communication with Serial Instruments in LabVIEW using NI-VISA - NI](#)

## 6.4 SWITCH Measurements

### 6.4.1 Static Digital Path Generation

#### Overview

Use the ***StaticDigitalPathGeneration*** class methods to initialize, configure, generate, and release NI-SWITCH compatible devices.

#### Validated Hardware

- NI-2527, NI-2568

#### Instructions-

1. Create an instance of the ***StaticDigitalPathGeneration()*** class using the library.
2. Initialize the switch session by calling the ***initialize(self, switch\_resource\_name: str, topology\_name: str, reset\_device: bool, simulate: bool)*** method on the class instance.
  - ***switch\_resource\_name*** (str): The resource name of the NI-SWITCH instrument.
  - ***topology\_name*** (str): The topology to apply to the switch.
  - ***reset\_device*** (bool): Whether to reset the device on initialization. Defaults to True.
  - ***simulate*** (bool): Whether to use a simulated instrument. Defaults to False.
3. Connect or disconnect paths by calling the ***configure\_and\_generate(self, configuration: StaticDigitalPathGenerationConfiguration)*** method with the following parameters:
  - a. ***terminal\_and\_state\_settings (StaticDigitalPathGenerationTerminalAndStateSettings)***:  
Aggregates the channel and connection state settings into a single object:
    - ***channel\_parameters (StaticDigitalPathGenerationChannelParameters)***:  
Specify the two channel names involved in the switch connection path.
      - ***channel\_one*** (str): Name of the first channel.
      - ***channel\_two*** (str): Name of the second channel.
    - ***connection\_state (StaticDigitalPathGenerationStateParameters)***:  
Specify whether to make or break the connection.
      - ***connect*** (bool): True to connect the path, False to disconnect it.
  - b. ***timing\_settings (StaticDigitalPathGenerationTimingParameters)***:  
Controls how long to wait for the relay to settle after switching.
    - ***max\_debounce\_wait*** (int): Maximum time in milliseconds to wait for the relay to debounce. Must be  $\geq 0$ .
  - c. The output data includes ***(StaticDigitalPathGenerationPathStatus)***:
    - ***path\_status (niswitch.PathCapability)***: Indicates the capability of the requested path — whether it was available, already existed, was unsupported, or involved a resource conflict.
5. Finally close the switch session using the ***close(self)*** method. This will automatically disconnect all active connections before releasing the instrument resource.

**Notes:**

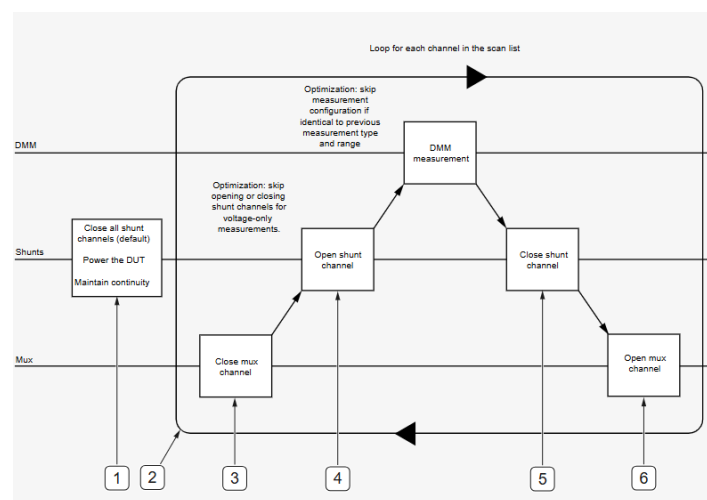
- If ***can\_connect()*** reports the path is neither available nor already existing (e.g., a resource conflict or unsupported path), no connect/disconnect action is taken and the path status is still returned for inspection.
- A connect value of True creates the path; False breaks it. Ensure the path exists before attempting to disconnect.
- ***reset\_device=True (default)*** ensures a clean relay state at the start of each session.
- Use ***simulate=True*** during development or testing when hardware is not present.

## 6.5 DMM SCAN Measurements

### 6.5.1 DMM Scan Library

#### Overview

The Python PCB Assembly Test Toolkit includes a DMM Scan library to facilitate parametric tests that can be used to rapidly test a variety of different measurements and settings. The figure below describes the general software structure and configuration for the DMM Scan library. The DMM Scan scan takes in a scan configuration consisting of a list of <channel, function & range, resolution> elements.



1. By default, during the initialize function, all shunt channels are closed to maintain continuity of current to power the DUT.
2. Each element (channel, function & range, digits) from the scan configuration list is passed sequentially to the main loop.
3. The mux channels for voltage-type measurements are closed based on the scan configuration for that path.
4. If the channel is current-type, then shunt channel is opened to enable a DMM current measurement.
5. The shunt channel is closed to maintain continuity between scans.
6. The mux channel is opened

The following two optimizations are executed:

- Do not open or close the shunt channel relay for voltage-only functions.
- Do not reconfigure the DMM if the next channel configuration is the same. Instead, simply execute the measurement using the existing configuration parameters.

To save time, we recommend grouping measurements with the same function and range together which will enable the optimizations. To precisely calculate the total reduced time, consult your DMM specifications and relay operation time values listed in the switch specifications or external relay documentation.



## 6.5.2 DMM\_Scan PXI Mux PXI Shunt 16V-15C Library

### Overview

The **DmmScanPMPS** class initializes, configures, scans, and measures a list of channels across multiple DMM functions, and closes sessions on the NI-DMM, SWITCH MUX, and SWITCH Shunt relay instruments.

This library performs a specific scan configuration that is assigned in a random order to a combination of 16 voltage type (maximum) and 15 current type (maximum) measurements in a 2-wire (differential) configuration.

This library supports 16 voltage type differential measurements, 15 current differential measurements, and 15 free SPST relays.

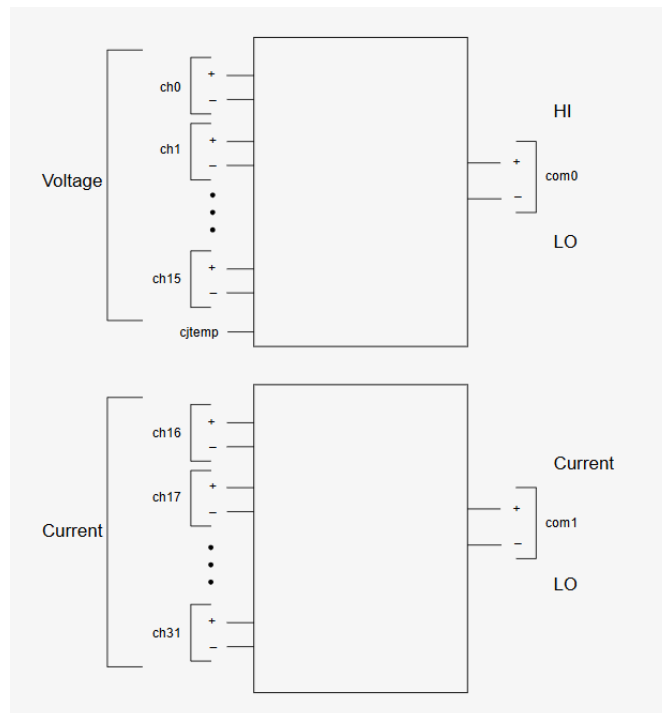
### Recommended Hardware

- PXI chassis (hybrid) and controller
- PXI-4065 or PXIe-408x DMM
- PXIe-2527 (or similar multiplexer switch module) with TB-2627 terminal block
- HV6-BAN4 cable to connect the DMM to the TB-2627 terminal block
- PXI-2568 or similar relay module with individual SPST relays for the current shunt continuity function

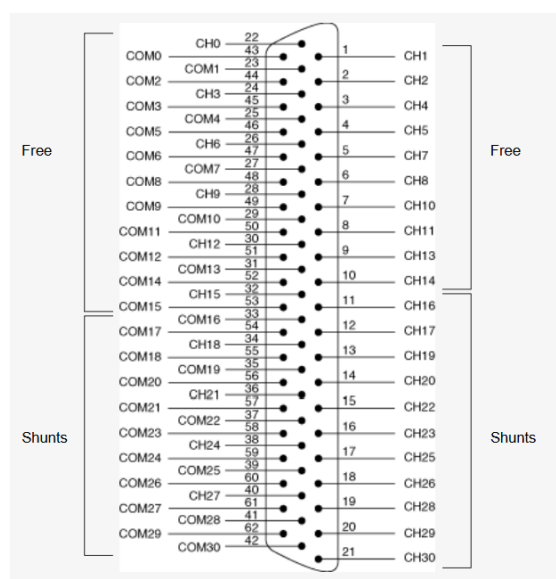
### Recommended Hardware Configuration

This section outlines the recommended connections, using the PXIe-2527 or similar multiplexer switch module configured in a dual topology:

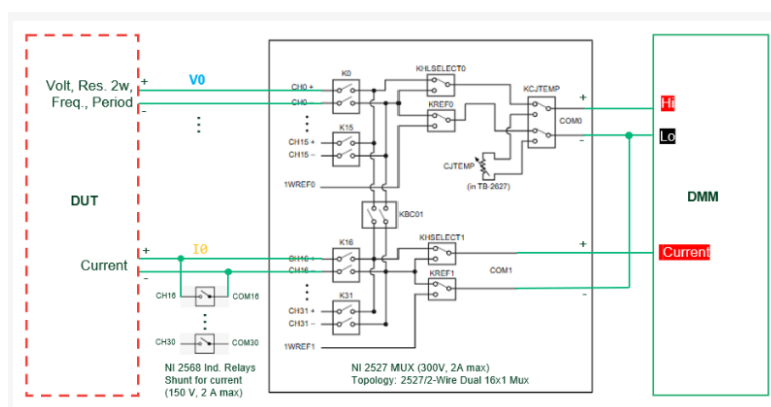
- 16 voltage-based 2-wire measurements (ch0 to ch15) connected to the DMM HI/LO
- 15 current-based 2-wire measurements (ch16 to ch30) connected to the same DMM to CURRENT/LO



The following figure depicts the connections for the PXI-2568 or similar relay module, as shunt, dedicated to 15 current-based 2-wire measurements (ch16 to ch30). Use a configuration in parallel with the current channels to simplify connections and maintain continuity of DUT power during all measurements.



The following figures illustrate example PXI Mux PXI Shunt connections for a DMM SCAN using the PXIe-2527 and PXI-2568.



## Software Recommendations

For voltage and current measurements, you must individually declare, in any order, the channels, functions and ranges, and digits in the scan list.

For timing optimization, the software removes the configuration phase between multiplexer scans when sequential channels in the scan list use the same configuration.

By default, all shunt relays are closed by the Initialize method. This behavior can be disabled with the *Close All Shunts* parameter.

To measure only a list of resistances, disable the *Close All Shunts* by setting it to False. The DUT does not receive power in this instance because all shunt relays are open.

## Instructions

1. Create an instance of the **DmmScanPMPS()** class using the library.

2. Initialize the switch and DMM sessions by calling the ***initialize(self, mux\_resource\_name: str, mux\_topology\_name: str, shunt\_resource\_name: str, shunt\_topology\_name: str, dmm\_resource\_name: str, powerline\_freq: int, close\_all\_shunts: bool)*** method on the class instance.
3. Configure and execute the scan by calling the ***configure\_and\_measure(self, resource\_handles: ScanResources, scan\_configuration: list, verbose: bool)*** method with the following parameters:
  - a. ***resource\_handles (ScanResources):***
    - Pass the session handles returned by ***initialize()***. Contains the MUX, shunt, and DMM sessions.
  - b. ***scan\_configuration (list):***
    - Populate this list with the measurements to perform. Each entry is a list in the format:  
***[channel (int), range\_and\_function (MixedRangeAndFunctions), resolution (ResolutionInDigits)]***
    - Channels 0–15 are voltage measurements (routed via com0); channels 16–30 are current measurements (routed via com1).
  - c. ***verbose (bool):***
    - Set to True to print scan results to the console. Set to False to suppress output.
  - d. The output data includes (***MeasurementResult***):
    - **sessions** – The switch and DMM session handles used during the scan.
    - **scan\_time** – Total elapsed wall-clock time of the scan in seconds.
    - **formatted\_measurements** – List of (channel, formatted value, elapsed time) per measurement.
    - **execution\_settings** – The DMM execution settings applied during each measurement.
    - **raw\_measurements** – List of (channel, raw value, measurement type) per measurement.
4. Finally close all sessions using the ***close(self, resource\_handles: ScanResources)*** method.

**NOTE**

Logger functionality is not supported by DMM Scan PMPS measurement.

The default powerline frequency value is 50 Hz. Specify 60 Hz for North or South America.

## 7 Automation Test Sequences

Test Sequences built in Python using Python PCBA measurement library demonstrates basic electrical functional test which can be reused or modified based on the procedure. Test Sequences are placed in “\nipcbatt-main\src\nipcbatt\pcbatt\_automation” or “\nipcbatt-2.x\src\nipcbatt\pcbatt\_automation” folder.

Below are the example tests added,

Test Sequence	Overview
<b>power_supply_tests</b>	Power Supply Tests source the Voltage from Power (TS-15200) Module and measure all Test points in the PCB simultaneously with Analog Input Module.
<b>led_tests</b>	LED Tests demonstrate the measurement of differential voltage across Anode and Cathode of LED and the validation of PWM signal with Analog Buffer.
<b>action_button_tests</b>	Action Button Tests demonstrate DC-RMS Voltage Measurements by performing button actions (generating DC Voltages) on specific test points in PCB simultaneously by using Analog Output and Analog Input Modules.
<b>digital_io_tests</b>	Digital IO Tests demonstrate the generation and measurement of Digital state, pattern, clock, pulse, PWM signals with Digital Output and Digital Input Modules.
<b>communication_tests</b>	Communication Tests demonstrate data read and write operations through various communication protocols like I2C, SPI and Serial Port Interactions by using NI Hardware’s like USB-845x and USB-232. The Automation sequences for SPI and I2C serve as examples of communication libraries with generic memory and registers.
<b>audio_tests</b>	Audio Tests demonstrate the frequency domain measurements of audio tones captured from the Audio amplifier path.
<b>microphone_tests</b>	Microphone Tests demonstrate Frequency domain measurements of captured Analog (Audio) signal generated by Analog Output (generates sine wave tones) module with Hardware Trigger using Analog Output and Analog Input Modules.
<b>sensor_tests</b>	Sensor Tests demonstrate the Temperature measurements captured by Thermistor, RTD and Thermocouple sensing devices using C Series Temperature Input Modules (cDAQ-9211, cDAQ-9217) and Analog Input Modules.



### NOTE

Detailed Help on how to use the above test sequences can be found in **ReadMe.pdf** placed parallel to the sequences.

**Naming Convention:** Test sequences follow a naming convention for Virtual Channel names where Sourcing points in the PCBA Board has “TS\_” prefix added and Test points to be measured has “TP\_” prefix added. And when multiple channels are involved, each channel has a common name and numbering starting with 0. For Example, Digital IO test has input channels – TS\_DIn0, TS\_DIn1 and Output channels – TP\_DOut0, TP\_DOut1

## 7.1 Execution with Simulated Hardware

Follow below steps to run the automation sequences using simulated hardware:

- Import NI MAX configuration file to get the simulated hardware. The configuration file is located at:  
`"<venv>\Lib\site-packages\nipcbatt\pcbatt_automation\Hardware Config.ini"`  
OR  
`"\nipcbatt-main\src\nipcbatt\pcbatt_automation\Hardware Config.ini"`  
OR  
`"\nipcbatt-2.x\src\nipcbatt\pcbatt_automation\Hardware Config.ini"`
- Open the validation examples/sequences in any IDE of your choice.
- Run the main sequence file present in folder of any sequence with a VENV having nipcbatt library installed.
- Results can be obtained from the .txt files created by the logger. Edit the file path in the sequences to save the results in any location of your choice.



### NOTE

To run with hardware, Refer **ReadMe.pdf** inside respective test folders

## 8 Functional Test Demo sequences

PCBA FT Demo Test Sequences demonstrates application testing of PCBA DUTs using various simulated hardware. It is a generic example for any PCBA DUT. It demonstrates how to use the PCBA Measurement Library, and how to extract test data and compare it against the expected result to determine the failures.

Demo Sequences can be found at `"\nipcbatt-main\src\nipcbatt\pcbatt_ft_demo_test_sequence"` or `"\nipcbatt-2.x\src\nipcbatt\pcbatt_ft_demo_test_sequence"`. This example sequence can be executed in Python using the Measurement libraries.

Refer below table for detailed test scenario,

PCBA FT DEMO	LSL	USL	Units	Timing (s)	Test Point	V Mode	Analysis Value	Trigger	Procedure/Condition	Measurement Library
<b>Power Diagnostics</b>										
To power up and measure Start-up Transition Max Current, Idle power Consumption and DC Regulators										
Start-up Transition max current	0	1	A	0.1	Simulated_Power/power	Ref	Max Current	Maintain Existing value	Power On Voltage 6V, 3A, Maintain Existing Value, Measure TP_ Max Current (Peak Transition)	Power Supply Source and Measure
Idle Power Consumption	4.5	5.5	W	0.1	Simulated_Power/power	Ref	Idle Watt	Maintain Existing value	Measure Idle Watt, Power voltage already On	Power Supply Source and Measure
DC Regulators	4.9	5.2	V	0.1	TP_REG0	Ref	Average DC Voltage	No	Measure Regulator Voltage 5V	DC Voltage Measurement
-	3.1	3.4	V	-	TP_REG1	Ref	Average DC Voltage	No	Measure Regulator Voltage 3.3V	-
<b>Reset and Self-Test</b>										
To simulate Push Reset Button ON/OFF condition for 0.1 sec followed by max 30 sec timeout wait for reboot and waiting check Status Activity LED ON										
Push Reset Button			Digital Level	0.1	TS_RESETO	Ref		No	Switch ON 0.1 sec then Switch OFF with Digital Output	Static Digital State Generation
Activity LED0 status			Digital Level	0.1 (30 max until reboot)	TP_ACT_LED0	Ref		No	Wait activity LED On, Timeout elapsed time <30 sec maximum	Static Digital State Measurement
<b>Animation and Sound User Input Test</b>										
To simulate the push on user action button to generate a sound and a light animation on 3 LEDs										
Push Action Button			Digital Level	0.5	TS_BUTTON0	Ref		No	Generate Digital Level for 0.5 sec to Simulate Button ON/OFF	Digital State Generation
Animation 3 LEDs			Digital Pattern	0.5	TP_AN_LED0:2	Ref	Custom Array Sample Analysis	SW Trigger after Button	Capture 200 samples, check states levels (by pointers) from Pattern animation array	Digital Pattern Measurement
Tone Action Sound Check	990	1010	Hz	0.5	TP_TWEET0	Ref	TDM Voltage Frequency	SW Trigger after Button	Measure 1 Khz Frequency on Tweeter	Time Domain Measurement
<b>Audio Filter Test</b>										
To test if measured Tones Frequency are within +-10% tolerance and Amplitudes Level are the same within +-10% tolerance (Filter Flat)										
Send Multi Tone Audio			V	0.1	TS_LINE_IN0	Ref			Send 4 tones: 10 Hz, 100 Hz, 1kHz, 10 Khz, 1V Sine Signal	Signal Voltage Generation
Measure Tones Frequencies	-10%	+10%	Hz	0.1	TP_LINE_OUT0	Ref	Detect Tones Frequencies	HW Trigger From Signal generation	Measure 4 Tones: 10 Hz, 100 Hz, 1kHz, 10 Khz	Frequency Domain Mesurement
Measure Tones Amplitudes	-10%	+10%	V	0.1	-	Ref	Detect Tones Amplitudes	-	Measure 4 Voltage Amplitudes (Same Level)	-

### Power Diagnostics

- Sources the supply voltage with a power supply resource and measures the DC Regulator Test Points simultaneously utilizing analog input. Libraries used in the example are “Power Supply Source and Measure” and “DC-RMS Voltage Measurement”.

### Reset and Self-Test

- Sources static DC state with a digital output resource to perform push reset button actions. Measures activity on a status LED using digital input resource. This example includes a 30 second wait for the DUT (timeout) to reboot. Libraries used in the example are “Static Digital State Generation” and “Static Digital State Measurement”.

### Animation and Sound User Input Test

- Sources DC voltage with analog output resource to perform push button actions. Takes time domain measurements using an analog input resource to measure a tweeter sound wave of 1 KHz. Uses a digital pattern input resource to acquire an array of LEDs performing an animation to be analyzed. Software Triggers are used between button actions and measurement captures. Libraries used in the example are “DC Voltage Generation”, “Digital Pattern Measurement” and “Time Domain Measurement”.

### Audio Filter Test

- Sends a multi tone sine wave through analog output module, captures it with the analog input module and extracts the detected tones to verify the expected frequency and amplitude. Hardware triggers are used to reduce the delay between generation and capture of signals. Libraries used in the example are “Signal Voltage Generation” and “Frequency Domain Measurement”

#### Turn Off all AO Channels

- Powers down all analog output channels by configuring the output voltage to 0 Volts. The library used in the example is “DC Voltage Generation”.

#### Power Down Supply

- Powers down all power supplies by disabling the output. The library used in the example is “DC Voltage Generation”



#### NOTE

Refer “**How to enable the hardware?**” section in the Help file that comes with Demo sequence, for how to use it / modify it to apply to real world UUTs.

## 9 Device Synchronization Example

**Synchronization\_tests** sequence demonstrates multiple devices synchronization using Generation and Measurement Library targeting both Analog and Digital IO modules.

### Example File Location

`"\nipcbatt-main\src\nipcbatt\pcbatt_automation\synchronization_tests"`

OR

`"\nipcbatt-2.x\src\nipcbatt\pcbatt_automation\synchronization_tests"`

### Note:

Refer ReadMe.pdf inside the above folder to know more about the example and instructions on how to run the example sequence.

### 9.1 How to achieve synchronization?

To achieve better synchronization between multiple devices and resources, NI suggests following the below standards to get optimum results,

1. During multi-device synchronization, make sure the external cables used to route signals like Sample clock and Start Trigger between systems are length matched. This prevents signal skew between trigger and clock.
2. Configure the generation and measurement ends to function at same sample clock rate.  
**Note:** Hardware devices will not divide down external sample clock signals and would need actual sample rate input from the user to provide correct timing details in the measurement results.
3. During multi-device synchronization for Digital Signals, Generation of Data in System 1 and Measurement of Data in System 2 are performed with same shared clock signal, there are chances of incorrect data measurements due to metastability caused by setup and hold time violation. To prevent such scenario, NI recommends the following configurations,
  - a) In Measurement end, set the 'Active Edge' parameter in Timing settings as 'Falling Edge' and in Generation end, set the same parameter as 'Rising Edge'. By doing so, an offset of  $\frac{1}{2}$  Sampling Time Period is added to measurement end to prevent setup and hold violations.
4. For more info on Synchronization, refer [Synchronization Explained](#) topic in NI Website.
5. To choose your custom synchronization technique for cDAQ/TestScale Devices, refer [Choosing a CompactDAQ Synchronization Technology](#) topic in NI Website.

## 10 Developing Test Programs

You can develop a test program in python using the libraries in nipcbatt.

`"\<venv>\Lib\site-packages\nipcbatt\pcbatt_library"`

*For example, to generate a sine waveform and measure a time domain waveform at any test point in the PCB, you could use the below two libraries:*



- i. *Use Signal Voltage Generation (SVG) library to generate the waveform of your choice by configuring the Signal Voltage Generation parameters*
- ii. *Then use Time Domain Measurement (TDVM) library to make time domain measurement of waveform signals by configuring the Time Domain Measurement parameters*

Please refer to the validation examples to gain an understanding of the library's functionality and how to utilize it to develop other examples.

### 10.1 Validation examples:

Validation examples are created as examples for testing a pair of libraries together, one library for generation and another for measurement.

Please refer to the validation examples in this location.

`"\<venv>\Lib\site-packages\nipcbatt\pcbatt_validation_examples"`

*Or refer to the location given below if you are using the downloaded source code from GitHub.*

`"\nipcbatt-main\src\nipcbatt\pcbatt_validation_examples"`

*Or refer to the location given below if you are using the downloaded source code from PyPI.*

`"\nipcbatt-2.x\src\nipcbatt\pcbatt_validation_examples"`

### 10.2 Automation Sequences:

*Automation sequences are examples of using libraries for real time scenarios like microphone tests, LED tests and so on. Automation sequences are tested in simulation mode.*

Please refer to the Automation Sequences in this location.

`"\<venv>\Lib\site-packages\nipcbatt\pcbatt_automation"`

*Or refer to the location given below if you are using source code downloaded from GitHub.*

`"\nipcbatt-main\src\nipcbatt\pcbatt_automation"`

*Or refer to the location given below if you are using source code downloaded from PyPI.*

`"\nipcbatt-2.x\src\nipcbatt\pcbatt_automation"`

*Please refer to the documentation Readme.pdf parallel to each automation sequence file for more details on each sequence.*

### 10.3 FT Demo Sequence:

*FT demo sequences is an example for creating a test sequence using libraries with applying test limits on the results to determine whether the test is a pass or a fail.*

Please refer to the FT Demo Sequence in the location.

`"<venv>\Lib\site-packages\nipcbatt\pcbatt_ft_demo_test_sequence"`

*Or refer to the location given below if you are using source code downloaded from GitHub.*

`"nipcbatt-main\src\nipcbatt\pcbatt_ft_demo_test_sequence"`

*Or refer to the location given below if you are using source code downloaded from PyPI.*

`"nipcbatt-2.x\src\nipcbatt\pcbatt_ft_demo_test_sequence"`

## 11 Modify nipcbatt source code and rebuild distribution package

This feature allows users to add new functions or modify existing library functions to adapt them to their specific needs. Refer to [Download source code](#) from section 5.1 to download the source code for the python library. Follow the below steps to create custom builds:

1. Extract the nipcbatt source files present in the downloaded archive “nipcbatt-main” zip file:

- nipcbatt-main
  - src
    - nipcbatt
  - PKG-INFO
  - pyproject.toml
  - docs
  - README.md
  - LICENSE
  - requirements.txt
  - requirements\_for\_packaging.txt

OR

Extract the nipcbatt source files present in the downloaded archive “nipcbatt-2.x.tar.gz”:

- nipcbatt-2.x
  - src
    - nipcbatt
  - PKG-INFO
  - pyproject.toml
  - docs
  - README.md
  - LICENSE
  - requirements.txt
  - requirements\_for\_packaging.txt

2. Install ***requirements\_for\_packaging.txt*** dependencies using the following command:  
*“pip install -r requirements\_for\_packaging.txt”*

Note: It is recommended to create a separate venv with the mentioned requirements and use it for the build process.

3. Modify source files contained in src folder and rebuild package by running inside folder containing src this command line:

*“python -m build”*

## 12 Errors and Troubleshooting

Common problems and troubleshooting techniques can be found below.

### 1. Channel Settings of Virtual Channels lost/not used when running test sequences Problem

The channel settings of virtual channels set using NI-MAX are replaced with different values after running the test sequences.

#### Solution

It is expected. Configuration VI will override the channel settings set by NI-MAX or Settings defined in Hardware Config.ini. Virtual channels are only used to map respective physical channels. All channel settings should be configured using VIs by the users.

### 2. Not enough waveform cycles (-20308)

#### Problem

When executing TDV Meas Configure and Measure, the error below occurs when the measured waveform is not expected to perform Time Domain analysis.

*“The waveform did not cross the mid reference level enough times to perform this measurement. Check the signal length, reference level and ref level units.”*

#### Solution

To perform time domain analysis (waveform period and duty cycle), more than one full cycle of the waveform is required so that there will be two points in the waveform which crosses mid reference level to calculate frequency and Duty cycle.

This error will occur while running the test sequence with simulated hardware or when DC Voltage is measured instead of waveform.

To skip the time domain analysis, set **“Skip Analysis?” to True**.

### 3. Redundant bits in the port data of Pattern Gen library

#### Problem

There is no error when we place 255 in port data even if we have 4 lines as input in the Core module.

#### Solution

Here each 8 bit corresponds to a line on the port and DAQmx APIs doesn't evaluate the size of the port but internally ignores the other bits.

Redundant line data can be either 0 or 1. Anyway those bits will be masked and will not be applied to terminals (lines). Refer Help of Port Digital Data control for more details on how to frame the output data for any requirements.

## 13 Known Issues List

### 1. Frequency Domain Measurements

Frequency Domain Measurement Library returns inaccurate results for lower voltage readings (below 1.5V).

### 2. Synchronization

During multi-devices synchronization for Digital Signals, Generation of Data in T1 system and Measurement of Data in T2 system are performed with same shared clock signal, there are chances of incorrect data measurements due to metastability caused by setup and hold time violation.

To prevent such scenario, NI recommends the following configurations. In Measurement end, set the 'Active Edge' parameter in Timing settings as 'Falling Edge' and in Generation end, set the same parameter as 'Rising Edge'. By doing so, an offset of  $\frac{1}{2}$  Sampling Time Period is added to measurement end to prevent setup and hold violations

### 3. Digital Pulse Width Measurement

The Digital Pulse Width Measurement library is designed to function solely with the default timebases of PCIe, cDAQ, and TestScale, specifically adhering to the default maximum and minimum semi-period values.

For further information on default maximum and minimum values of semi-periods please refer to "[Relation between Timebase and semi-period duration](#)" table.

### 4. Thermocouple Temperature Measurement

Global virtual channel properties cannot be overwritten due to the absence of setter method for `ai_thrmcpl_cjc_src` in nidaqmx-python. So, it runs with default settings of the global virtual channel which are defined in NI-MAX.

Use of Physical channels is recommended for this library to avoid this issue.

### 5. DC-RMS Current Measurement

Nidaqmx-python does not allow current min and max range parameters to be overwritten. In this library, there is an additional parameter present in the `initialize()` function, `use_specific_channel: bool` which allows the library to correctly perform current measurements while using `specific_channel_parameters`. This will skip the initialization of the channels not mentioned in the `specific_channel_parameters`.

Users should set this parameter to True while using specific channel settings.

## 14 Related Documents

- Refer to the getting started pdf for installation procedure in the below location:
  - “*nipcbatt-main\src\docs\Python PCB Assembly Test Toolkit - Getting Started.pdf*”.
  - “*nipcbatt-2.x\src\docs\Python PCB Assembly Test Toolkit - Getting Started.pdf*”.
- Detailed Help on how to use the example test sequences can be found in **ReadMe.pdf** inside each Automation Sequences.
- Refer to the schematics of the PCIe setup in the below location:
  - *Location in installed library:*  
“*<venv>\Lib\site-packages\nipcbatt\pcbatt\_validation\_examples*”
  - Location in downloaded source code from GitHub:  
“*nipcbatt-main\src\nipcbatt\pcbatt\_validation\_examples*”
  - Location in downloaded source code from PyPI:  
“*nipcbatt-2.x\src\nipcbatt\pcbatt\_validation\_examples*”

## 7. References

- NI DMM Overview - [NI-DMM Overview - NI](#)
- PXI-4065 Specifications - [PXI-4065 Specifications - NI](#)
- PXI-2527 Overview - [PXI-2527 Overview - NI](#)
- PXI-2527 Specifications - [PXI-2527 Specifications - NI](#)
- PXI-2568 Overview - [PXI-2568 Overview - NI](#)
- PXI-2568 Specifications - [PXI-2568 Specifications - NI](#)
- PC Based X series DAQ - [X series User manual](#)
- cDAQ AI Module - [NI 9205](#)
- cDAQ AI Module - [NI 9215](#)
- cDAQ AO Module - [NI 9263](#)
- cDAQ RTD Module - [NI 9217](#)
- cDAQ TC Module - [NI 9211](#)
- cDAQ DIO Module - [NI 9402](#)
- cDAQ DIO Module - [NI 9403](#)
- cDAQ DO Module - [NI 9477](#)
- I2S SPI device - [USB-8452](#)
- Serial device - [USB-232](#)
- TestScale Help - [What is TestScale? - NI](#)
- TestScale Module TS-15050 Overview - [TS-15050 Overview - NI](#)
- TestScale Module TS-15050 Specifications - [TS-15050 Specifications - NI](#)
- TestScale Module TS-15100 Overview - [TS-15100 Overview - NI](#)
- TestScale Module TS-15100 Specifications - [TS-15100 Specifications - NI](#)
- TestScale Module TS-15110 Overview - [TS-15110 Overview - NI](#)
- TestScale Module TS-15110 Specifications - [TS-15110 Specifications - NI](#)
- TestScale Module TS-15120 Overview - [TS-15120 Overview - NI](#)
- TestScale Module TS-15120 Specifications - [TS-15120 Specifications - NI](#)
- TestScale Module TS-15130 Overview - [TS-15130 Overview - NI](#)
- TestScale Module TS-15130 Specifications - [TS-15130 Specifications - NI](#)
- TestScale Module TS-15200 Overview - [TS-15200 Overview - NI](#)
- TestScale Module TS-15200 Specifications - [TS-15200 Specifications - NI](#)
- DAQmx Help - [NI-DAQmx Help - NI](#)