Audio Test Sequence

1.1 Purpose

The Audio Tests Sequence performs frequency domain measurements of sine tones using the Analog Input and Analog Output Resources. This example sequence can be executed in a custom Python sequence script using the measurement libraries written in Python.

Example File Location

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

1.2 Highlighted Features

- Audio Line Check
  - Sends single-tone sine waveform through Analog Output resource, captures it with Analog input resource and performs Frequency domain measurements to validate the Audio amplifier path. Hardware Triggers are used to reduce the delay between generation and capture of signals. Libraries used in the example are "***SignalVoltageGeneration()***" and "***FrequencyDomainMeasurement()***".
- Audio Filter Check
  - Sends multi-tone sine wave through Analog Ouput resource, captures it with Analog input resource and extract the detected tones to verify the filter setup of the DUT. Hardware Triggers are used to reduce the delay between generation and capture of signals. Libraries used in the example are "***SignalVoltageGeneration()***" and "***FrequencyDomainMeasurement()***".
- Turn Off all AO Channels
  - Powers down all analog output channels by configuring the output voltage as 0 Volts. Libraries used in the example is "***DcVoltageGeneration()***".

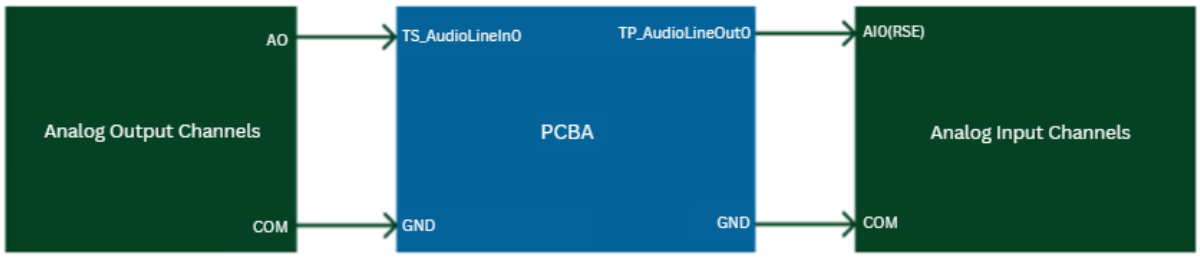
Refer this folder for more details on each Measurement library "\<venv>\Lib\site-packages\nipcbatt\pcbatt_library".

1.3 Prerequisites

- Python – 3.9 to 3.12
- DAQmx Driver – 2023 Q3 or later

1.4 Setup Diagram

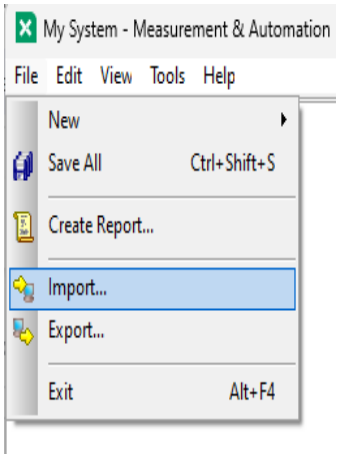
Represents the hardware setup used in this example sequence. Pin Outs of each resource is added below.



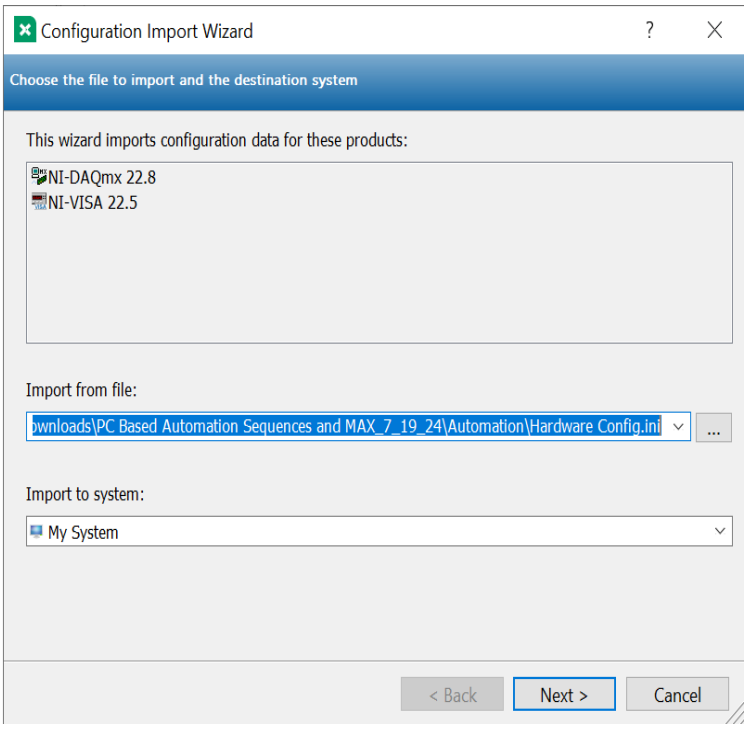
1.5 How to run this Example?

Complete the following steps to run the sequence.

1. First, we must configure NI-MAX to reflect the simulated virtual channels which will be used by the Python script names mentioned in ***audio_filter_check.py*** or ***audio_line_check.py:***
   a. A hardware configuration file for NI-MAX is required to run this example. The configuration file contains a set of predefined global channel names which are used by the nidaqmx driver to communicate with the Python scripts.
   b. To import the "Hardware Config" open NI-MAX.
   c. Click on File -> Import to open the Configuration Import Wizard



   d. In the Configuration Import Wizard window, click on the Browse (...) button and locate the *Hardware Config.ini* file in *"\<venv>\Lib\site-packages\nipcbatt\pcbatt_automation".* Then click on *Next -> Import -> Finish*



   e. NI-MAX now holds the same virtual channel name references contained in the examples provided

The ***audio_filter_check.py*** and ***audio_line_check.py*** files will create log files in the form of a simple text (.txt) file. The default file path it will use is

"C:\\Windows\\Temp\\power_supply_test_with_trigger_results.txt"
"C:\\Windows\\Temp\\power_supply_test_without_trigger_results.txt"

If you wish to create this file in a different location on your PC, change the value of the string variable *DEFAULT_FILEPATH*.

2. Open the Python scripts ***audio_test_main_sequence.py*** *along with* ***audio_filter_check.py*** and ***audio_line_check.py*** in your IDE or text editor of choice. The following steps are performed within ***audio_test_main_sequence.py.***
    a. **Audio Line Check** - demonstrates Frequency domain measurements of captured single tone Analog (Audio) signal generated by Analog Output resource with Hardware Trigger. Below are the steps included in the test.
        i. Initialize Signal Voltage Generation and Frequency Domain Measurement Libraries by creating the instances of ***SignalVoltageGeneration()*** and ***FrequencyDomainMeasurement()*** classes and then using ***initialize()*** methods on each object.
        ii. Configure Frequency Domain Measurement to wait for Start Trigger from Signal Voltage Generation.
        iii. Configure the Signal Voltage Generation to start sourcing Sine Tone. Here the example generates **Sine wave of 1kHz with Amplitude of 1V.** (in the backend, Signal Voltage Generation resource sends the Trigger through the backplane during sourcing starts which in-turns starts the measurement in Analog Input resource).
        iv. Fetch the Voltage waveforms measured and return the Frequency measurements.
        v. Use the ***close()*** methods on both instances to close all tasks and release resources allocation.

        Refer the help/comments in the sequence for more details to know more about trigger configuration.

    b. **Audio Filter Check** - demonstrates Frequency domain measurements of captured multi tone Analog (Audio) signal generated by Analog Output resource with Hardware Trigger. Below are the steps included in the test.
        i. Initialize Signal Voltage Generation and Frequency Domain Measurement Libraries by creating the instances of ***SignalVoltageGeneration()*** and ***FrequencyDomainMeasurement()*** classes and then using ***initialize()*** methods on each object.
        ii. Configure Frequency Domain Measurement to wait for Start Trigger from Signal Voltage Generation.
        iii. Configure the Signal Voltage Generation to start sourcing Sine wave with multi-tones. Here the example generates **Multi tone Sine wave of 1Hz, 100Hz, 1kHz and 10kHz with Amplitude of 1V** (in the backend, Signal Voltage Generation resource sends the Trigger through the backplane

during sourcing starts which in-turns starts the measurement in Analog Input resource)

iv. Fetch the Voltage waveforms measured and return the detected frequency and amplitude of tones.

v. Use the ***close()*** methods on both instances to close all tasks and release resources allocation.

Refer the help/comments in the sequence for more details to know more about trigger configuration.

c. **Turn Off all AO Channels** – Power downs all Analog output channels by configuring them to 0 Volts. Below are the steps included in the test. These steps are accomplished within *turn_off_all_ao_channels.py*

i. Initialize the DC Voltage Generation library by creating an instance of ***DcVoltageGeneration()*** class and then using ***Initialize()*** method.

ii. Configure the DC Voltage Generation to source 0 Volts in specified Analog Output channels by calling the ***configure_and_generate()*** method using the default parameters.

iii. Use ***close()*** instance after setting AO channels to 0 Volts.

3. When the execution completes, review the results on the *.txt* files generated by the logger at the specified location.
   a. The report has the configurations and Measurement values captured (runs with simulated instrument by default)
   b. Verify the Measurement and data formats returned by the Measurement library

1.5.1 How to enable the Hardware?

Audio Test sequence runs with simulated hardware by default. Once the hardware setup is available, you can do the below changes to enable running the test with the hardware.

Note : In this example, physical and global virtual channels are used to configure the terminal or pin to perform the instrument actions. Global Virtual Channels are software entities that encapsulate the physical channel along with other channel specific information—range, terminal configuration, and custom scaling. Global Channels can be created in NI-MAX and called in Measurement Libraries

1. Skip the first step which imports simulated virtual channels in MAX as in section 1.5. If already done, you can simply update the channel names (physical or virtual) in the ***initialize()*** step of each automation sequence to match the hardware connected/detected in NI-MAX.
   Note: Please ensure correct trigger sources as mentioned in the steps below.

2. Follow the below steps for each sequence. Refer "**Note to run with Hardware**" labels in the sequence.
   i. **Audio Line Check**
      1. Step into the "***audio_line_check.py***" sequence
      2. Update the "GEN_CHANNEL" input with Signal Voltage Generation resource channel in the ***initialize()*** method of ***SignalVoltageGeneration()*** also update the "MEAS_CHANNEL" input with Frequency Domain Measurement resource channel in the ***initialize()*** method of ***FrequencyDomainMeasurement()***.

3. Open NI-MAX and update the physical Channel linked to the Global Channels – **TS_AudioLineIn0, TP_AudioLineOut0** (called in the initialize step of Signal Voltage Generation and Frequency Domain Measurement)
4. Update the "*digital_start_trigger_source*" based on NI-MAX Hardware in the *"DigitalStartTriggerParameters*" configured in the ***configure_and_measure()*** method.
5. Review the Configurations of Analog Output and Analog Input Pins for the intended use case.

ii. **Audio Filter Check**

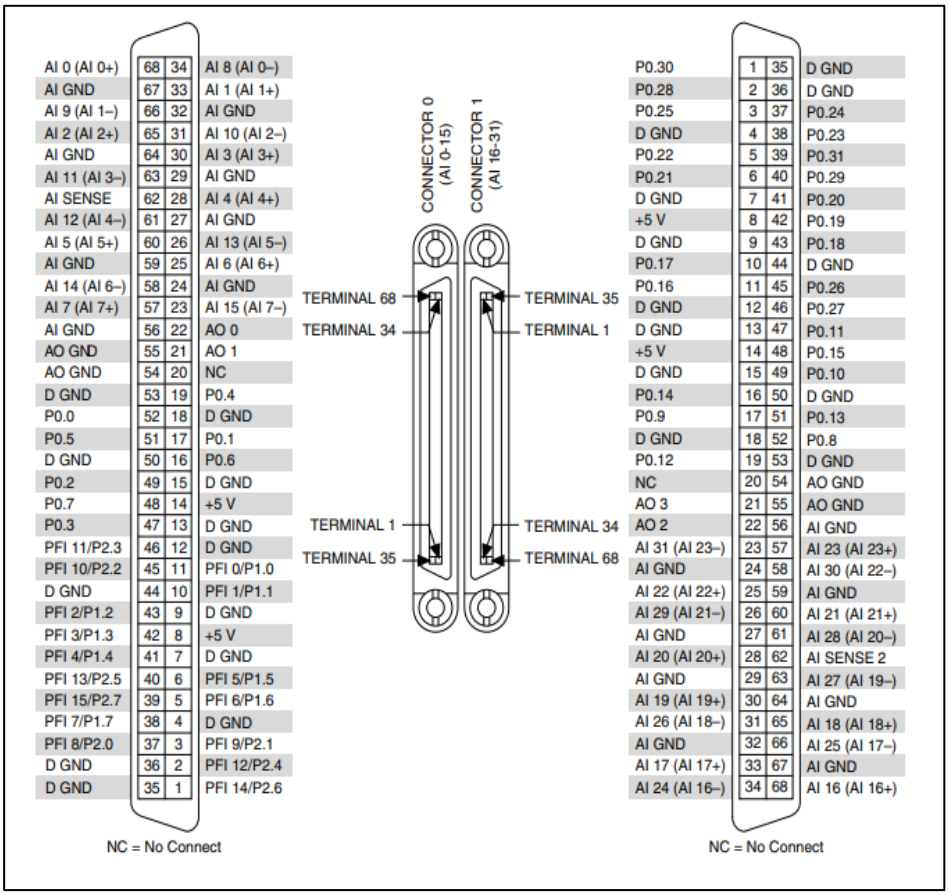
1. Step into the "***audio_filter_check.py***" sequence
2. Update the "GEN_CHANNEL" input with Signal Voltage Generation resource channel in the ***initialize()*** method of ***SignalVoltageGeneration()*** also update the "MEAS_CHANNEL" input with Frequency Domain Measurement resource channel in the ***initialize()*** method of ***FrequencyDomainMeasurement()***.
3. Open NI-MAX and update the physical Channel linked to the Global Channels – **TS_AudioLineIn0, TP_AudioLineOut0** (called in the initialize step of Signal Voltage Generation and Frequency Domain Measurement) – Ignore if updated in previous step
4. Update the "*digital_start_trigger_source*" based on NI-MAX Hardware in the *"DigitalStartTriggerParameters*" configured in the ***configure_and_measure()*** method.
5. Review the Configurations of Analog Output and Analog Input Pins for the intended use case

iii. **Turn Off AO Channel**

1. Step into the "***turn_off_all_ao_channels.py***" example.
2. Update the physical channels input with Analog Output Channel in the initialize step of ***Turn_off_all_ao_channels.***
3. Review the Configurations of Analog Output for the intended use case

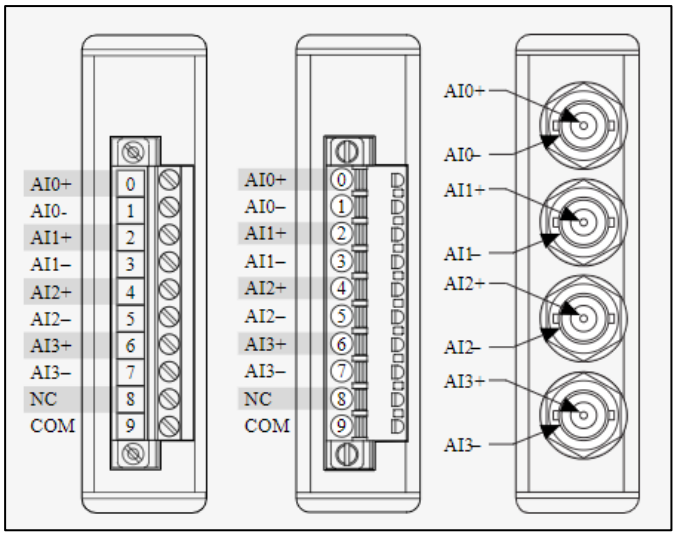
3. Review the **generate and range settings of Analog Output Pins and Analog input Pins** based on the DUT and Connections before running with Hardware.

1.6 Pinouts of PCIe-6323

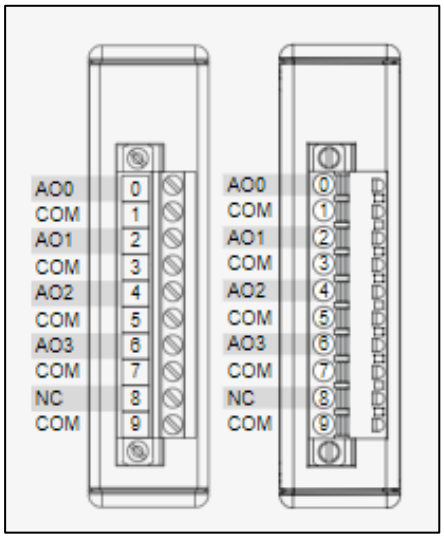


1.7 Pinouts of cDAQ Modules

1. **Analog Input Module (NI-9215)**

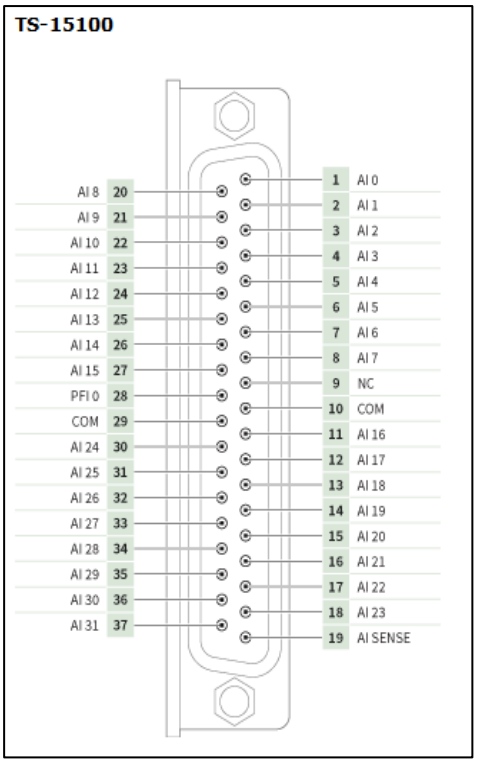


2. **Analog Output Module (NI-9263)**

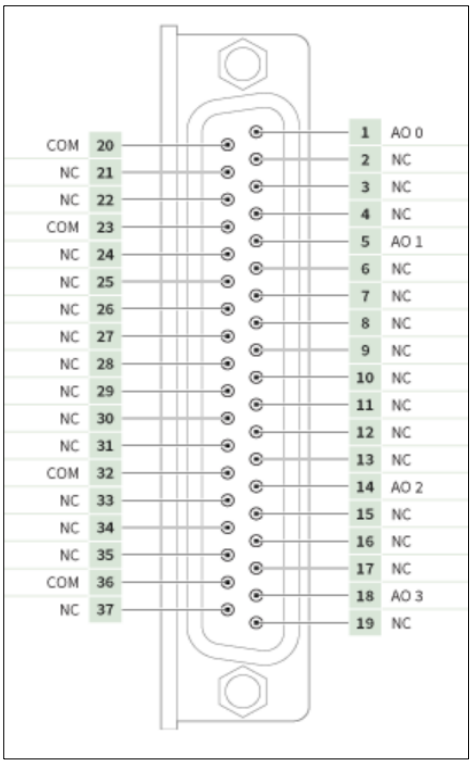


1.8 Pinouts of TestScale Modules

1. **Analog Input Module (TS-15100)**



2. **Analog Output Module (TS-15110)**



1.9 How to create/Modify Global Virtual Channels?

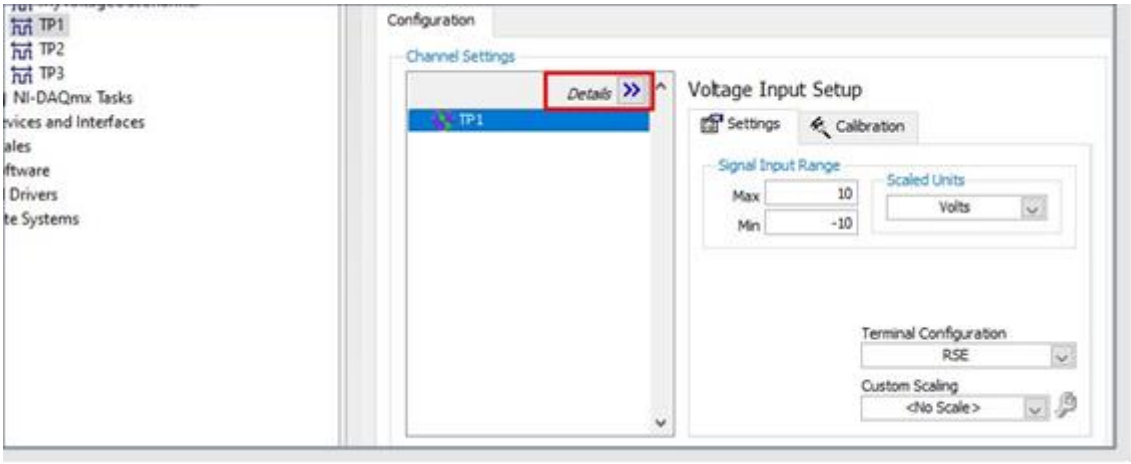
A virtual channel is a collection of settings such as a name, a physical channel, input terminal connections, the type of measurement or generation, and can include scaling information. A virtual channel created outside a task is a Global Virtual Channel.Follow the below steps to **create Global Virtual Channel** in NI-MAX.

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

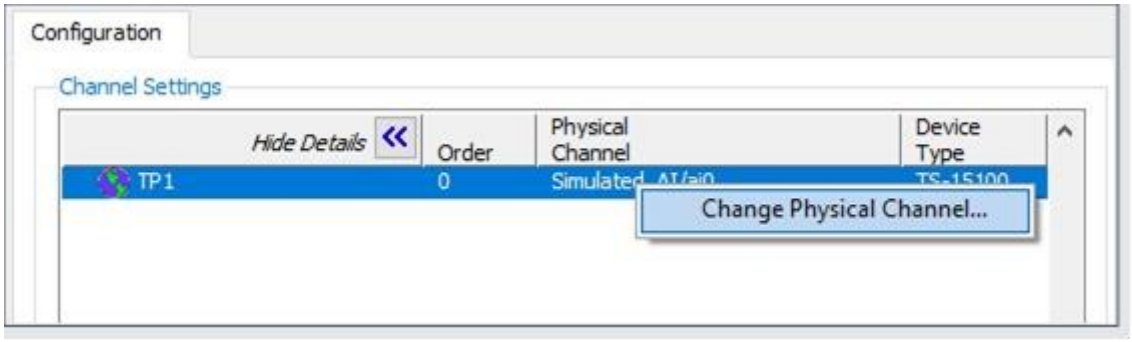
1. Launch NI-MAX
2. In NI-MAX, right-click **Data Neighbourhood** and select **Create New**
3. In the Create New window, select **NI-DAQmx Global Virtual Channel** and click **Next**. The DAQ Assistant opens.
4. Select an I/O type, such as analog input
5. Select the physical channel of Hardware
6. Type 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