Digital IO Test Sequence

1.1 Purpose

The Digital IO Test Sequence demonstrates the generation and measurement of digital state, pattern, clock, pulse, PWM signals with Digital Output and Digital Input 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\digital_io_tests"

1.2 Highlighted Features

- Digital State Test
  - Generates Digital High & Low state using Digital output resources and measures the digital states in test points using digital input resources. Libraries used in the example are "***StaticDigitalStateGeneration()"*** and "***StaticDigitalStateMeasurement()"***.
- Digital Pattern Test (with trigger)
  - Generates Digital Pattern using Digital output resources and measure the same using digital input resources. Libraries used in the example are "***DynamicDigitalPatternGeneration()"*** and "***DynamicDigitalPatternMeasurement()"***.
- Digital Clock Tests
  - Generates Digital Clock and measures the Digital Frequency of the same signal through counter-based measurements using Core Digital IO Resources. Libraries used in the example are "***DigitalClockGeneration()"*** and "***DigitalFrequencyMeasurement()"***.
- Digital PWM Test
  - Generates Digital pulse signal and measures the Digital PWM parameters of the same signal through counter-based measurements using Core Digital IO Resources. Libraries used in the example are "***DigitalPulseGeneration()"*** and "***DigitalPwmMeasurement()"***.
- Digital Count Event Tests - SW Timed (External Wait)
  - Generates Digital pulse signal and counts the number of Digital edges present in the same signal through counter-based measurements using Core Digital IO Resources. Digital edge counting is performed at Software timed using an external wait. Libraries used in this example are "***DigitalPulseGeneration()"*** and "***DigitalEdgeCountMeasurementUsingSoftwareTimer()"***.
- Digital Count Event Tests - HW Timed (With Trigger)
  - Generates Digital pattern and counts the number of Digital edges present in the same signal through counter-based measurements using Core Digital IO Resources. Digital edge counting is performed at Hardware timed using with Trigger to create a measurement window for fixed duration. Libraries used in this example are "***DynamicDigitalPatternGeneration()"*** and "***DigitalEdgeCountMeasurementUsingHardwareTimer()"***.

- Turn Off all DO Channels
    - Powers down all digital output channels by configuring the output state to LOW. Libraries used in the example is "***StaticDigitalStateGeneration()".***

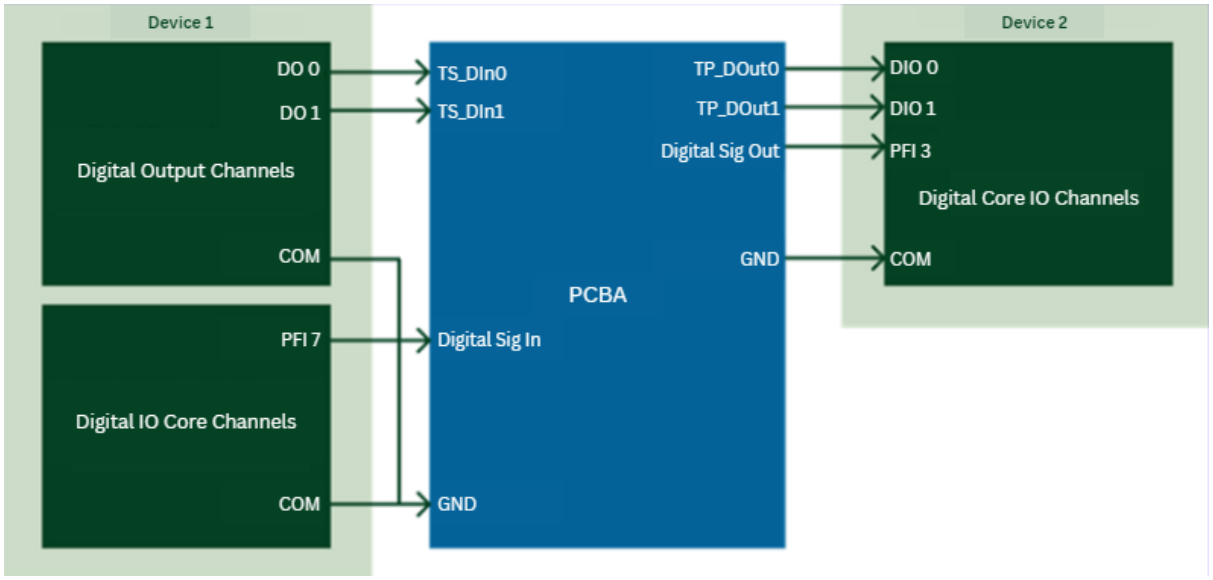
Refer this folder for more details on each Measurement library "\<venv>\Lib\site-packages\nipcbatt\pcbatt_library".

1.3 Prerequisites

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

1.4 Setup Diagram

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



Note:

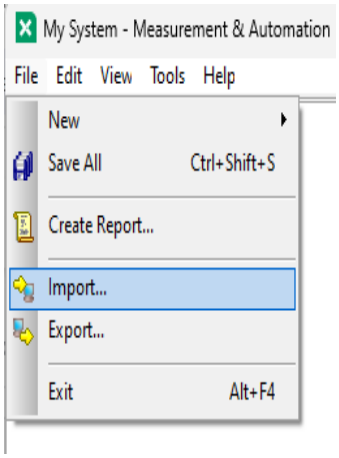
1. Make sure there is a common GND between different Digital resources involved.
2. In the above block diagram, usage of PFI3 as Input and PFI7 as Output is arbitrary. User can use any PFI terminals based on their custom use cases.

1.5 How to run this Example?

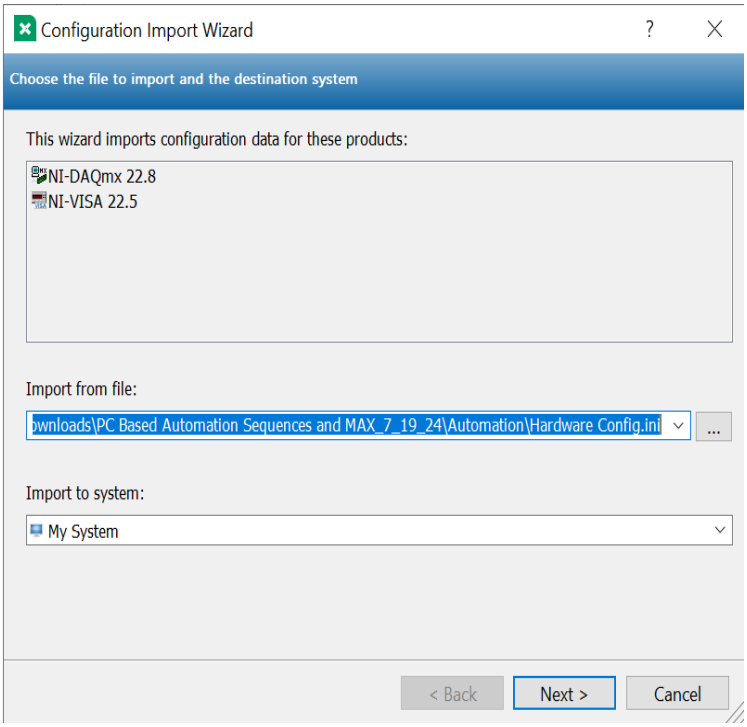
Complete the following steps to run the sequence.

1. First, we must configure NI-MAX to reflect the simulated virtual channels which will be used by the Python script names mentioned in ***digital_clock_test.py or digital_count_events_hw_timed.py*** or ***digital_count_events_sw_timed.py*** or ***digital_pattern_test.py*** or ***digital_pwm_test.py*** or ***digital_state_test.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 ***digital_clock_test.py, digital_count_events_hw_timed.py, digital_count_events_sw_timed.py, digital_pattern_test.py, digital_pwm_test.py*** and ***digital_state_test.py*** *fi*les will create log files in the form of a simple text (.txt) file. The default file path it will use is:

"C:\\Windows\\Temp\\digital_clock_test_results.txt"
"C:\\Windows\\Temp\\digital_edge_count_hw_timed_test_results.txt"
"C:\\Windows\\Temp\\digital_edge_count_sw_timed_test_results.txt"
"C:\\Windows\\Temp\\digital_pattern_test_results.txt"

"C:\\Windows\\Temp\\digital_pwm_test_results.txt"
"C:\\Windows\\Temp\\digital_state_test_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 ***digital_io_main_sequence.py*** *along with* ***digital_clock_test.py, digital_count_events_hw_timed.py, digital_count_events_sw_timed.py, digital_pattern_test.py, digital_pwm_test.py*** and ***digital_state_test.py*** in your IDE or text editor of choice. The following steps are performed within ***digital_io_main_sequence.py.***
    a. **Digital State Test** - Demonstrates static digital state generation and measurement using Digital output and input resources. Below are the steps included in the test.
        i. Initialize Static Digital State Generation and Static Digital State Measurement Libraries (Global Channels are inputs) by creating the instances of ***StaticDigitalStateGeneration()*** and ***StaticDigitalStateMeasurement()*** classes and then using ***initialize()*** method on each object***.***
        ii. Configures the Static Digital State Generation and start sourcing Digital State **HIGH.**
        iii. Static Digital State Measurement reads the Digital State.
        iv. Configures the Static Digital State Generation and start sourcing Digital State **LOW**
        v. Static Digital State Measurement reads the Digital State.
        vi. 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.

    b. **Digital Pattern Test (with Trigger)** - Demonstrates digital pattern generation and measurement using Digital input and output resources. Hardware Triggers are used to reduce the delay between Source and Measure. Below are the steps included in the test.
        i. Initialize Dynamic Digital Pattern Generation and Dynamic Digital Pattern Measurement Libraries (Global Channels are inputs) by creating the instances of ***DynamicDigitalPatternGeneration()*** and ***DynamicDigitalPatternMeasurement()*** classes and then using ***initialize()*** method on each object.
        ii. Configure Dynamic Digital Pattern Measurement to wait for Start Trigger from Digital Output Resource
        iii. Configure the Dynamic Digital Pattern Generation and start producing pulse train (in the backend, Digital Output resource sends the Trigger in the backplane once the Source started which in turn starts the measurement in Digital Input resource)
        iv. Fetch the pulse train measured from Digital Input Resource using Dynamic Digital Pattern Measurement library.
        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.

c. **Digital Clock Tests** - Demonstrates digital clock generation and frequency measurement through counter-based measurements using Core Digital IO Resources. Below are the steps included in the test.
    i. Initialize Digital Clock Generation and Digital Frequency Measurement Libraries (Physical Terminal and Counter are inputs) by creating the instances of ***DigitalClockGeneration()*** and ***DigitalFrequencyMeasurement()*** classes and then using ***initialize()*** method on each object.
    ii. Generates Digital Clock signal of fixed frequency using Digital Clock Generation Library.
    iii. Fetch and measures the frequency of generated Digital clock signal using Digital Frequency Measurement Library.
    iv. 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.

d. **Digital PWM Test** - Demonstrates digital pulse generation and PWM measurement through counter-based measurements using Core Digital IO Resources.
    i. Initialize Digital Pulse Generation and Digital PWM Measurment Libraries (Physical Terminal and Counter are inputs) by creating the instances of ***DigitalPulseGeneration()*** and ***DigitalPwmMeasurement()*** classes and then using ***initialize()*** method on each object.
    ii. Configure Digital PWM Measurement Library to capture Digital cycles/pulses using counter.
    Note: Digital PWM Measurement Library starts PWM measurements only on the arrival of first rising edge in the terminal input.
    iii. Generated required Digital pulses using Digital Pulse Generation Library.
    iv. Fetch and measure the PWM parameters like High Time(s), Low Time(s), Duty Cycle(%) and Frequency(Hz) from the captured Digital pulse signal.
    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.

e. **Digital Count Event Tests - SW Timed (External Wait)** - Demonstrates digital pulse generation and digital edge count measurement through counter-based measurements using Core Digital IO Resource. Digital edge counting is performed at Software timed using an external wait.
    i. Initialize Digital Pulse Generation and Digital Edge Count Measurment using Software Timer Libraries (Physical Terminal, Counter are inputs) by creating the instances of ***DigitalPulseGeneration()*** and ***DigitalEdgeCountMeasurementUsingSoftwareTimer()*** classes and then using ***initialize()*** method on each object.
    ii. Arms Digital Edge Count Measurement Library to measure Digital Edges measured on the terminal input.
    iii. Generates fixed number of Digital pulse signals using Digital Pulse Generation Library

iv. Add an external software wait for the generation to be completed by Digital Pulse Generation Library
v. Capture and measure the number of digital pulse edges obtained using Digital Edge Count Measurement Library.
vi. 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.

f. **Digital Count Event Tests - HW Timed (With Trigger)** - Demonstrates digital pattern generation and digital edge count measurement through counter-based measurements using Core Digital IO Resources. Digital edge counting is performed at Hardware timed using with Trigger to create a measurement window for fixed duration.
   i. Initialize Digital Pattern Generation and Digital Edge Count Measurement Hardware Timer Libraries (Physical Terminal, Counter are inputs) by creating the instances of ***DynamicDigitalPatternGeneration()*** and ***DigitalEdgeCountMeasurementUsingHardwareTimer()*** classes and then using ***initialize()*** method on each object.
   ii. Configure Digital Edge Count Measurement Library to measure Digital Edges for a fixed Measurement window upon Start Trigger.
   iii. Generates Digital pulse train signals using Digital Pattern Generation Library (in the backend, Digital Output resource sends the Trigger in the backplane once the Source started which in turn starts the measurement in Digital Input resource)
   iv. Capture and measure the number of digital pulse edges obtained using Digital Edge Count Measurement Library for the fixed measurement window.
   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.

g. **Turn Off all DO Channels** – Power downs all Digital output channels by configuring them to LOW state. Below are the steps included in the test.
   i. Initialize Static Digital State Generation Library by creating the instance of ***StaticDigitalStateGeneration()*** class and then using ***initialize()*** method on each object .
   ii. Configure the Static Digital State Generation to source state LOW in specified Digital Output channels.
   iii. Use the ***close()*** methods on both instances to close all tasks and release resources allocation

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 run with Hardware?

Digital IO 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. **Digital State Test**
      1. Step into "***digital_state_test.py***" sequence
      2. Update the "GENERATION_CHANNEL" input with Static Digital State Generation resource channel in the ***initialize()*** method of ***StaticDigitalStateGeneration()*** also update the "MEASUREMENT_CHANNEL" input with Static Digital State Measurement resource channel in the ***initialize()*** method of ***StaticDigitalStateMeasurement().***
      3. Open NI-MAX and update the physical Channel linked to the Global Channels – **TS_Din0, TS_Din1**, **TP_DOut0**, **TP_DOut1** (used in the initialize step of Static Digital State Generation and Measurement)
         Note: Sequence should be executed once in Simulation mode to create the required Global Virtual Channels in NI-MAX to modify.
      4. Review the Configurations of Digital Output and Digital Input Pins for the intended use case
   ii. **Digital Pattern Test (With Trigger)**
      1. Step into "***digital_pattern_test.py***" sequence.
      2. Update the "GENERATION_CHANNEL" input with Dynamic Digital Pattern Generation resource channel in the ***initialize()*** method of ***DynamicDigitalPatternGeneration()*** also update the "MEASUREMENT_CHANNEL" input with Dynamic Digital Pattern Measurement resource channel in the ***initialize()*** method of ***DynamicDigitalPatternMeasurement().***
      3. Open NI-MAX and update the physical Channel linked to the Global Channels – **TS_Din0, TS_Din1**, **TP_DOut0**, **TP_DOut1** (used in the initialize step - ignore if updated in previous steps)
      4. Update the Trigger settings in "Dynamic Digital Pattern Measurement - Configure TP" step – Set Trigger based on HW setup to capture digital signal as soon as pattern generated using Dynamic Digital Pattern Generation Library.

5. Update parameters in “Generate Port Digital Data” step based on digital pattern to be generated. This Step can also be replaced entirely with custom step to generate any custom digital pattern.
6. Verify if the sampling rate at measurement end should be same as the sampling rate at generation end (Onboard Clock for same Backplane or external PFI signals)
7. Review the configurations for the intended use case.

iii. **Digital Clock Test**

1. Step into “***digital_clock_test.py***” sequence
2. Update the “OUTPUT_TERMINAL” and “GEN_PHYSICAL_CHANNEL_COUNTER” inputs with Digital Clock Generation resource channel in the ***initialize()*** method of ***DigitalClockGeneration()*** also update the “INPUT_TERMINAL” and “MEAS_PHYSICAL_CHANNEL_COUNTER” inputs with Digital Frequency Measurement resource channel in the ***initialize()*** method of ***DigitalFrequencyMeasurement().***
3. Update Digital clock setting in “Digital Clock Generation - Configure and generate Digital clock” based on the required Digital Clock to generate for the intended use case.
4. Review the configurations for the intended use case.

iv. **Digital PWM Test**

1. Step into “***digital_pwm_test.py***” sequence
2. Update the “OUTPUT_TERMINAL” and “GEN_PHYSICAL_CHANNEL_COUNTER” inputs with Digital Pulse Generation resource channel in the ***initialize()*** method of ***DigitalPulseGeneration()*** also update the “INPUT_TERMINAL” and “MEAS_PHYSICAL_CHANNEL_COUNTER” inputs with Digital PWM Measurement resource channel in the ***initialize()*** method of ***DigitalPwmMeasurement().***
3. Update the cycles to capture in “Digital PWM Measurement – Configure_Only” step.
4. Update the Digital pulse settings in “Digital Pulse Generation - Generate Digital pulse signals” step based on the required digital signal to be generated.
5. Update the cycles to capture in “Digital PWM Measurement – Measure_Only” step
6. Review the configurations for the intended use case.

v. **Digital Count Event Tests - SW Timed (External Wait)**

1. Step into “***digital_count_events_sw_timed.py***” sequence.
2. Update the “OUTPUT_TERMINAL” and “GEN_PHYSICAL_CHANNEL_COUNTER” inputs with Digital Pulse Generation resource channel in the ***initialize()*** method of ***DigitalPulseGeneration()*** also update the “INPUT_TERMINAL” and “EDGE_COUNTER” inputs with Digital Edge Count Measurement Using Software Timer resource channel in the ***initialize()*** method of ***DigitalEdgeCountMeasurementUsingSoftwareTimer().***

3. Update the Digital pulse settings in “Digital Pulse Generation - Configure & Generate Digital pulse signals” step based on the required digital signal to be generated.
4. Review the configurations for the intended use case.

vi. **Digital Count Event Tests - HW Timed (With Trigger)**

1. Step into “***DigitalEdgeCountMeasurementUsingSoftwareTimer.py*** sequence.
2. Update the “GENERATION_CHANNEL” input with Dynamic Digital Pattern Generation resource channel in the ***initialize()*** method of ***DynamicDigitalPatternGeneration()*** also update the “INPUT_TERMINAL”, “EDGE_COUNTER” and “COUNTER_TIMER” inputs with Digital Edge Count Measurement Using Hardware Timer resource channel in the ***initialize()*** method of ***DigitalEdgeCountMeasurementUsingHardwareTimer().***
3. Update the Trigger settings in “Digital Edge Count Measurement - Configure TP” step – Set Trigger based on HW setup to capture digital edges as soon as pattern generated using Dynamic Digital Pattern Generation Library.
4. Update parameters in “Generate Port Digital Data” step based on digital pattern to be generated. This Step can also be replaced entirely with any other step to generate any custom digital pattern.
5. Review the configurations for the intended use case.

vii. **Turn Off DO Channel Sequence**

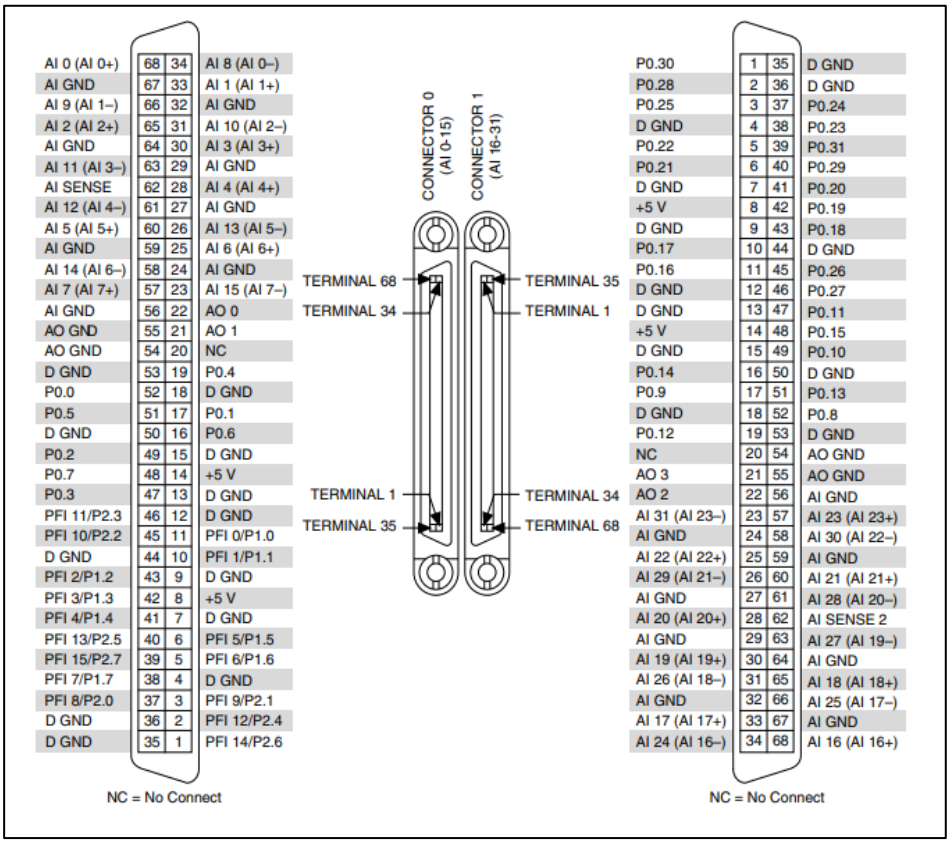
1. Step into the “***turn_off_all_do_channels.py***” sequence.
2. Update the “Global Channels” input with Digital Output Channels used in the initialize step of Turn Off All DO Channels sequence.
3. Review the Configurations of Digital Output Pins for the intended use case.

3. Hardware setup considerations

i. Make sure common ground connection provided between digital input and output resources.

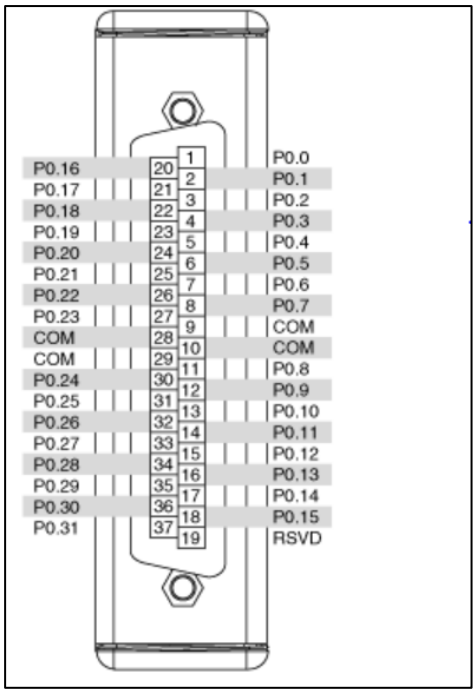
ii. In digital pattern measurement, make sure to use same sample clock rate in both digital input and output resources to extract the exact data from the testpoint.

1.6 Pinouts of PCIe-6323

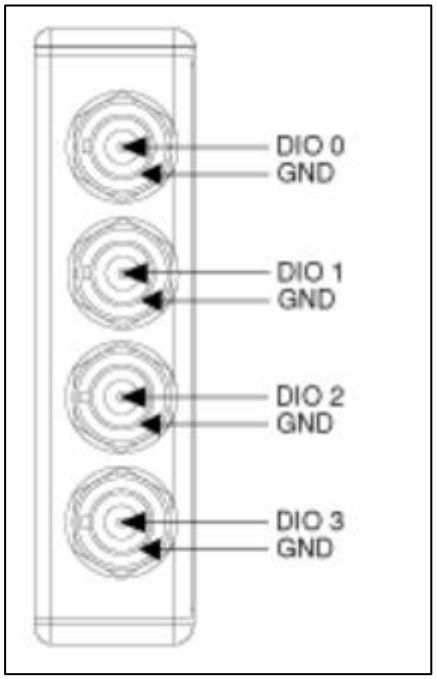


1.7 Pinouts of cDAQ Module

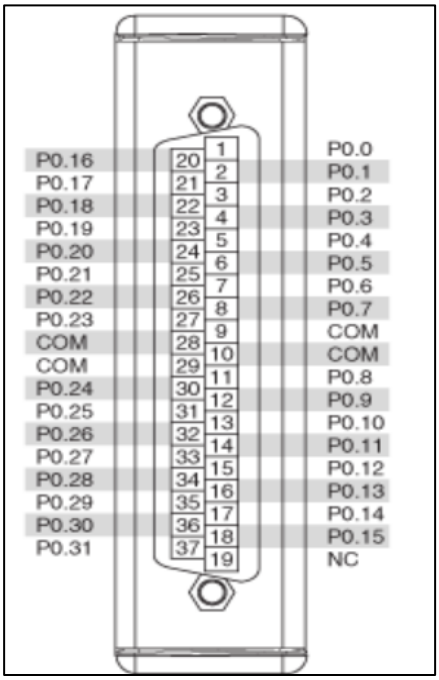
1. **Digital Input Output Module (NI-9403)**



2. **PFI Module (NI-9402)**

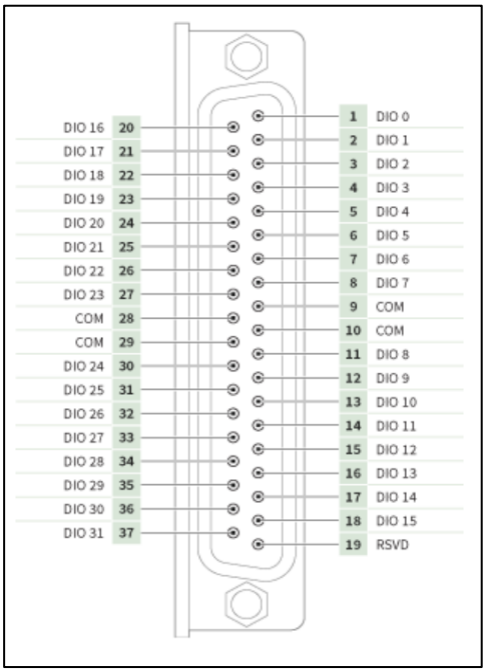


3. **Digital Output Module (NI-9477)**

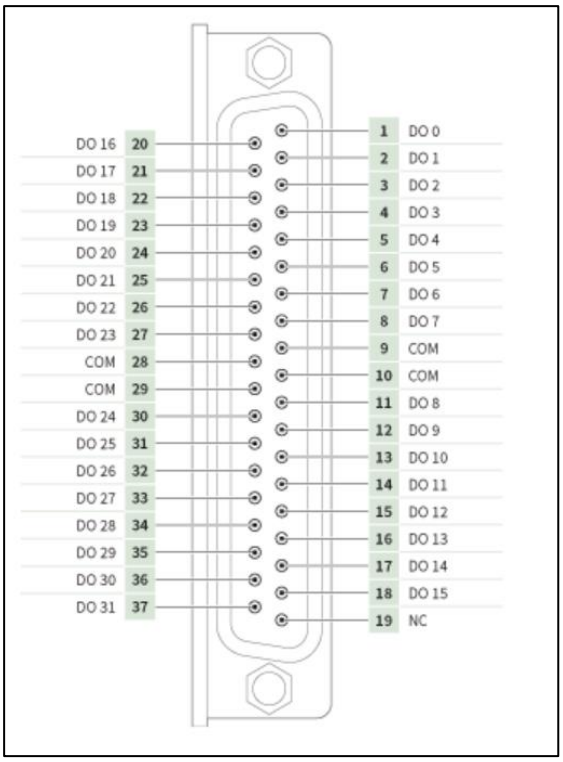


1.8 Pinouts of TestScale Modules

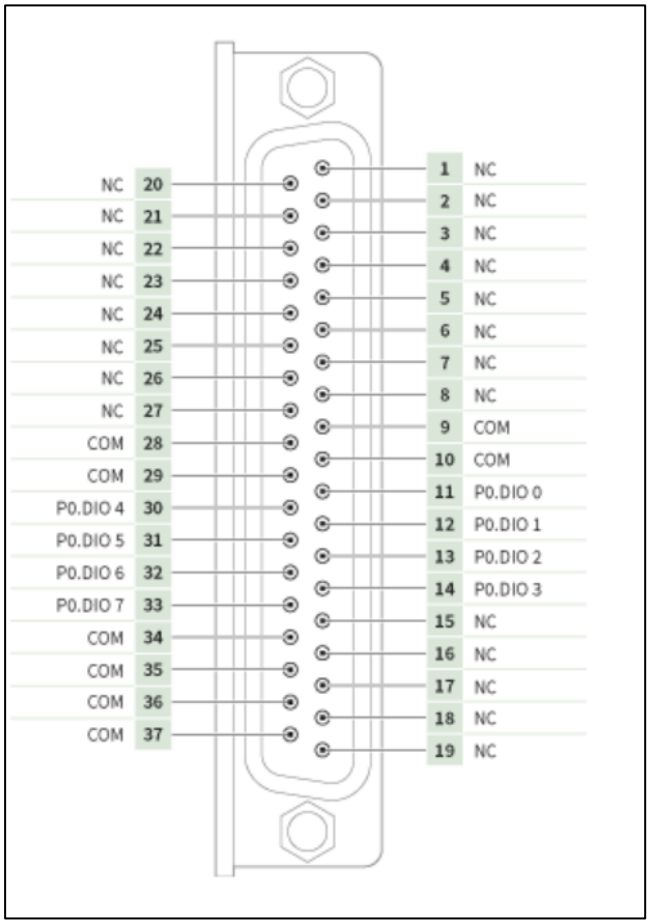
1. **Digital Input Output Module (TS-15120)**



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



3. **Core DIO Module (TS-15050)**



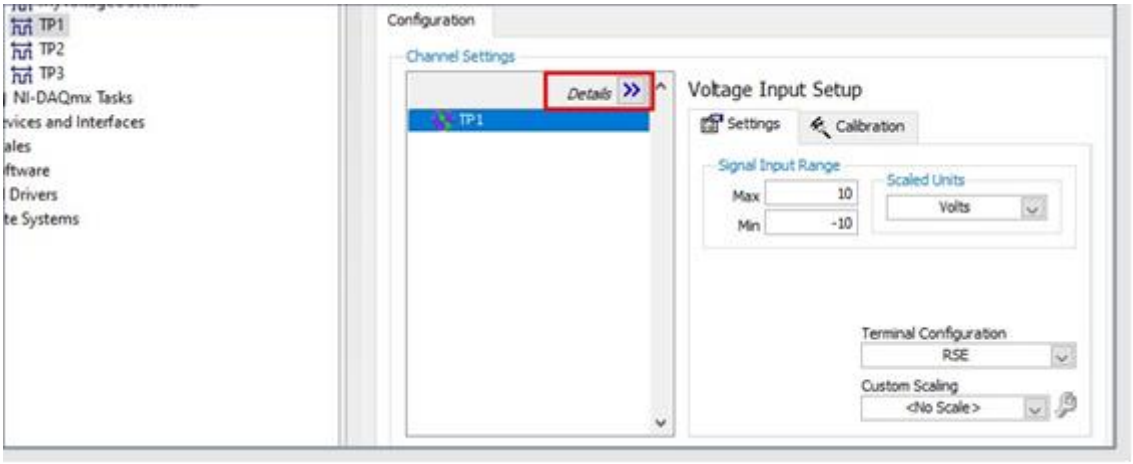
1.9 How to create/Modify Global Virtual Channels?

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.

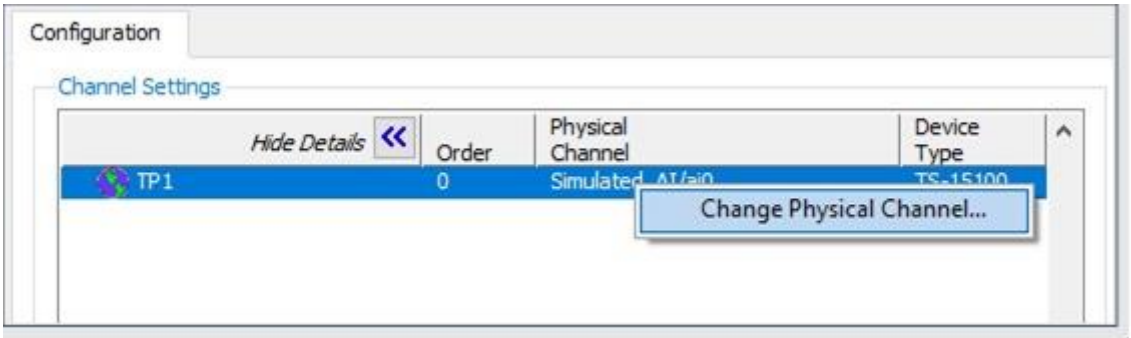
1. Launch NI-MAX
2. In NI-MAX, right-click **Data Neighbourhood** and select **Create New**
3. In the Create New window, select **NI-DAQmx Global Virtual Channel** and click **Next**. The DAQ Assistant opens.
4. Select an I/O type, such as analog input
5. Select the physical channel of Hardware
6. Type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