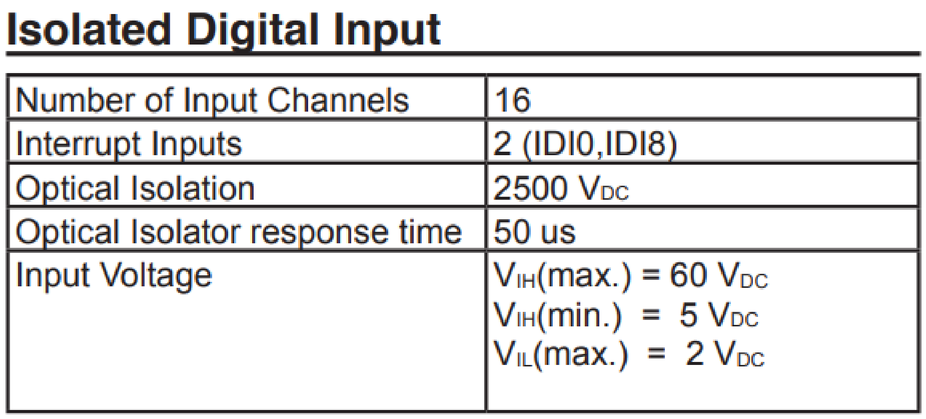
Advantech – USB 4750 Documentation

Specifications of device: :

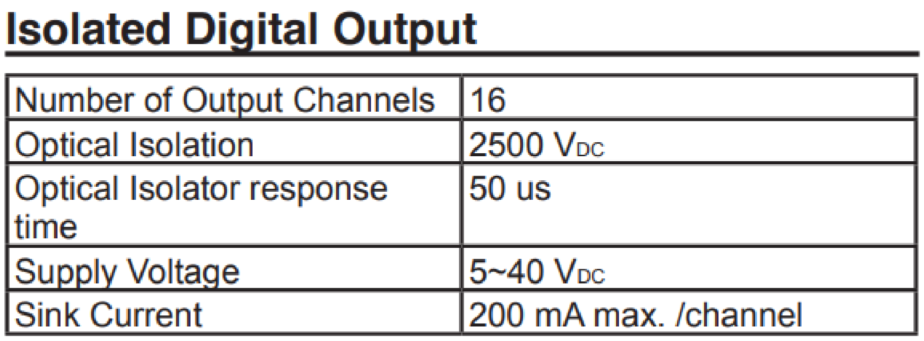
Isolated Digital Input



Isolated Digital Input

Number of Input Channels	16
Interrupt Inputs	2 (IDI0,IDI8)
Optical Isolation	2500 V_{DC}
Optical Isolator response time	50 us
Input Voltage	V_{IH}(max.) = 60 V_{DC} V_{IH}(min.) = 5 V_{DC} V_{IL}(max.) = 2 V_{DC}

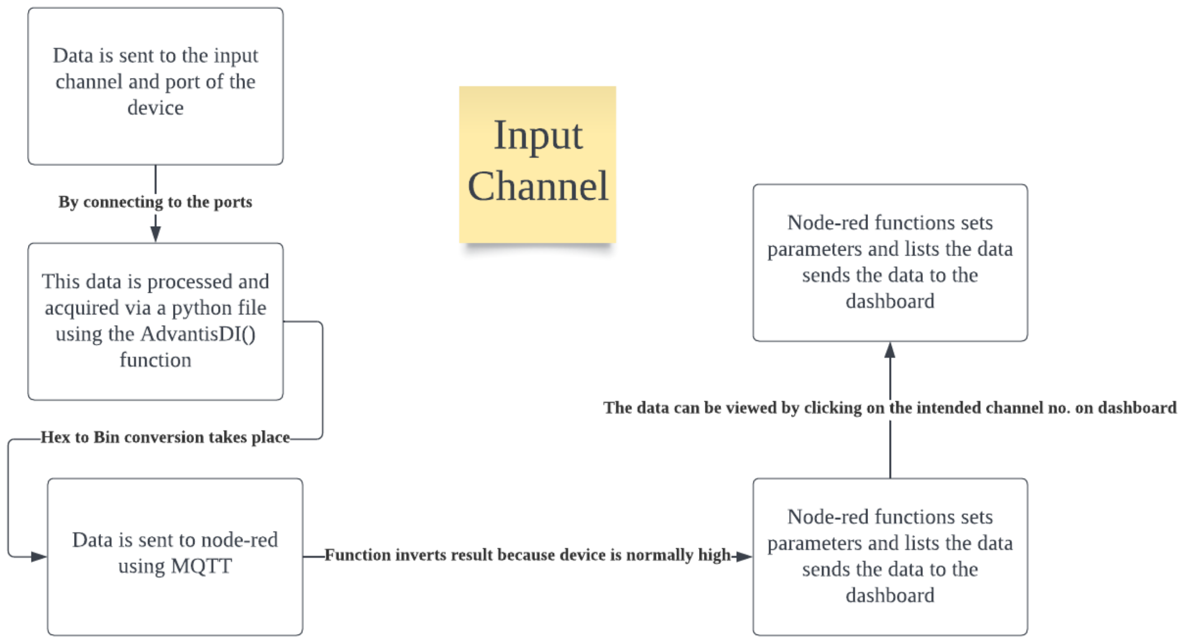
Isolated Digital Output



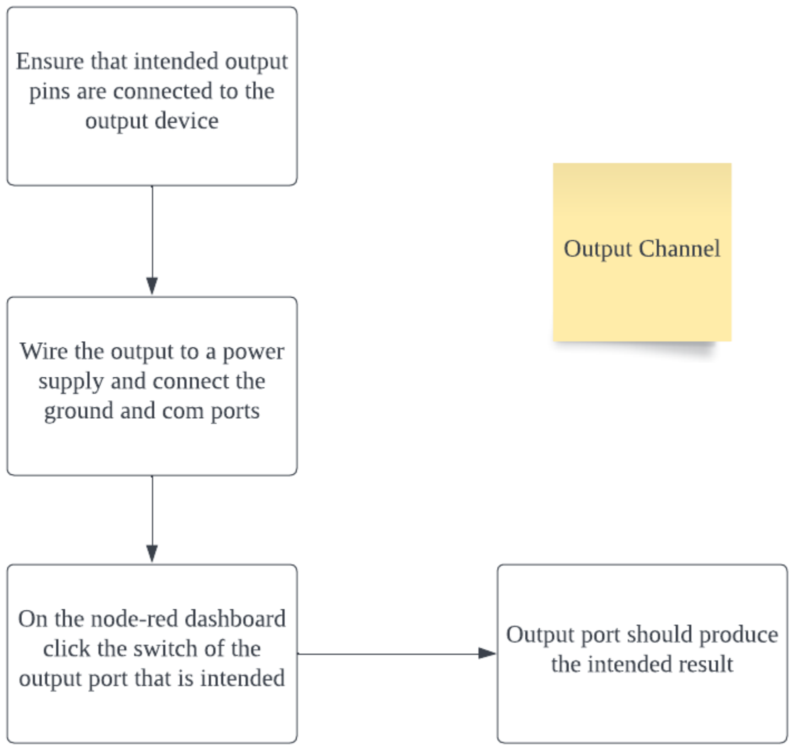
Isolated Digital Output

Number of Output Channels	16
Optical Isolation	2500 V_{DC}
Optical Isolator response time	50 us
Supply Voltage	5~40 V_{DC}
Sink Current	200 mA max. /channel

A basic flowchart summarizing the entire process is being shown below:-


Data is sent to the input channel and port of the device
By connecting to the ports
This data is processed and acquired via a python file using the AdvantisDI() function
Hex to Bin conversion takes place
Data is sent to node-red using MQTT
Function inverts result because device is normally high
Node-red functions sets parameters and lists the data sends the data to the dashboard
The data can be viewed by clicking on the intended channel no. on dashboard
Node-red functions sets parameters and lists the data sends the data to the dashboard
Input Channel

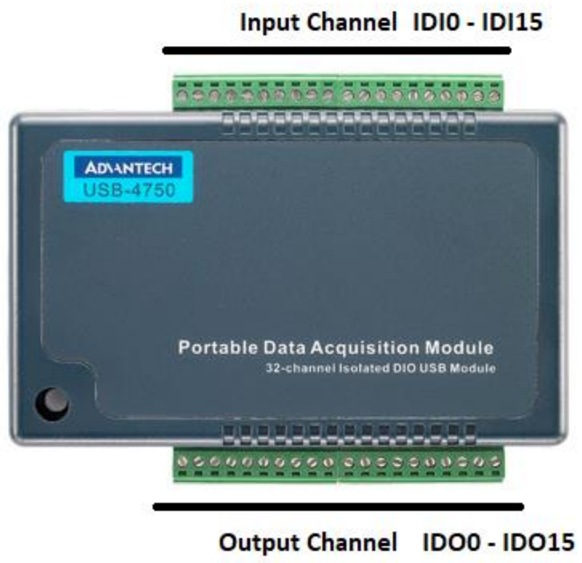
Output Channel:-


Ensure that intended output pins are connected to the output device
Wire the output to a power supply and connect the ground and com ports
On the node-red dashboard click the switch of the output port that is intended
Output port should produce the intended result
Output Channel

Step by Step process of extracting data from Inputs

Each input channel will accept a 5 V- 60 V Direct Current

It is important to note that the numbering of the ports is as follows


Input Channel IDI0 - IDI15
ADVANTECH
USB-4750
Portable Data Acquisition Module
32-channel Isolated DIO USB Module
Output Channel IDO0 - IDO15

Each channel is separated into 2 ports

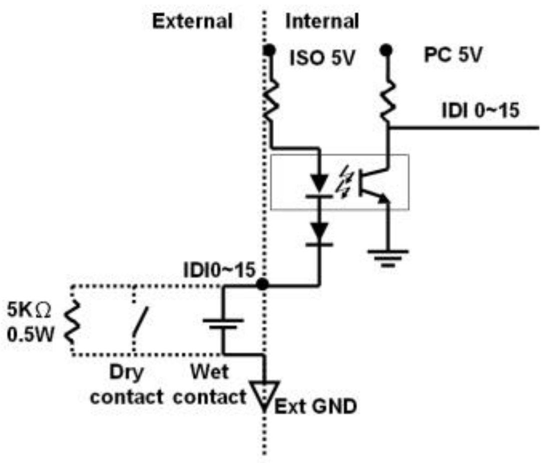
For the input channel :- Port 1 (IDI0 – IDI7)

Port 2 (IDI8 – IDI15)

For the output channel :- Port 1 (IDO0 – IDO7)

Port 2 (ID08 – IDO15)

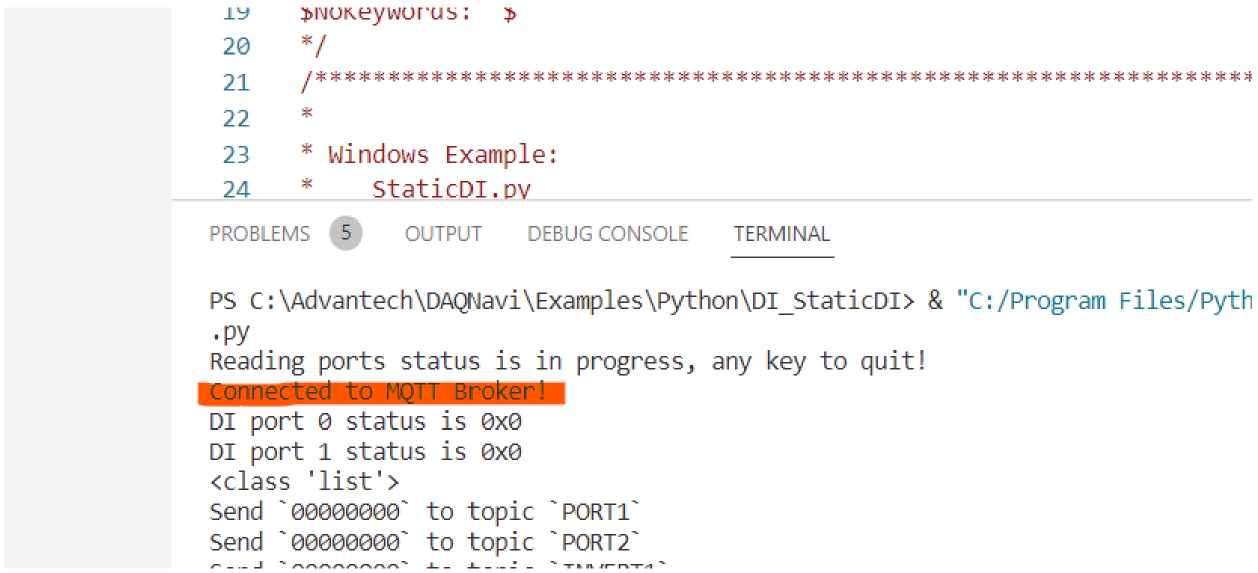
<u>Step 1</u> :- Connect ground of input pin to the GND channel in the device and connect the other pin to the desired channel



Step 2 :- Connect the device to the computer with all the drivers installed and run the python program provided

On successful connection with the device the terminal should give an output of

‘Connected to MQTT broker’ as shown below:-

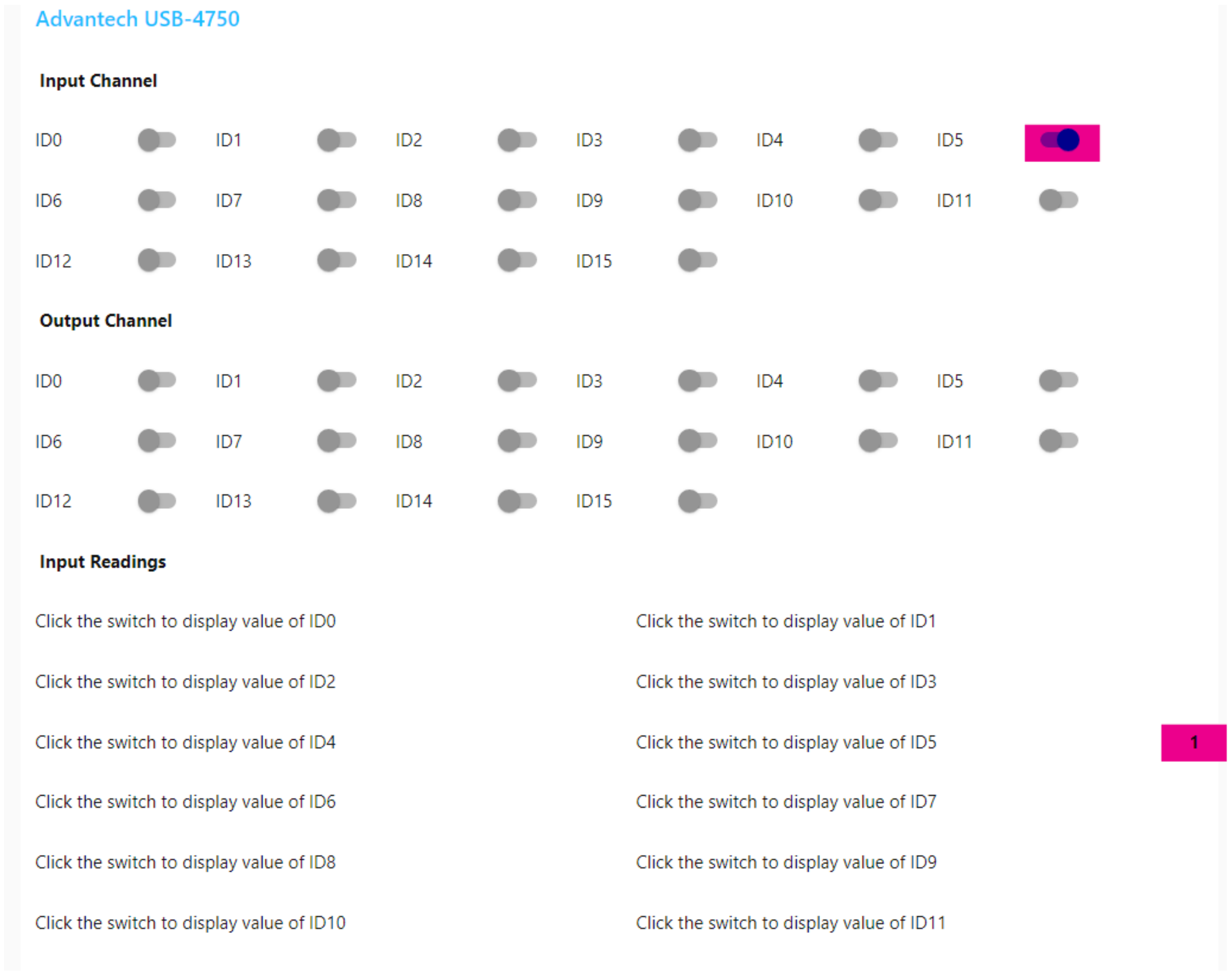


```
$NoKeywords:  $
*/
/***********************************************************************
*
* Windows Example:
*     StaticDI.py
```

```
PS C:\Advantech\DAQNavi\Examples\Python\DI_StaticDI> & "C:/Program Files/Pyth
.py
Reading ports status is in progress, any key to quit!
Connected to MQTT Broker!
DI port 0 status is 0x0
DI port 1 status is 0x0
<class 'list'>
Send `00000000` to topic `PORT1`
Send `00000000` to topic `PORT2`
```

Step 4:- Once connected, proceed to the node-red dashboard and toggle the switch on the channel number of the port that you have inserted the input to and the result should be displayed on the screen

Important - The input being triggered through a channel will only be displayed and shown by toggling the switch linked that channel and port number



Advantech USB-4750

Input Channel

ID0	ID1	ID2	ID3	ID4	ID5
ID6	ID7	ID8	ID9	ID10	ID11
ID12	ID13	ID14	ID15		

Output Channel

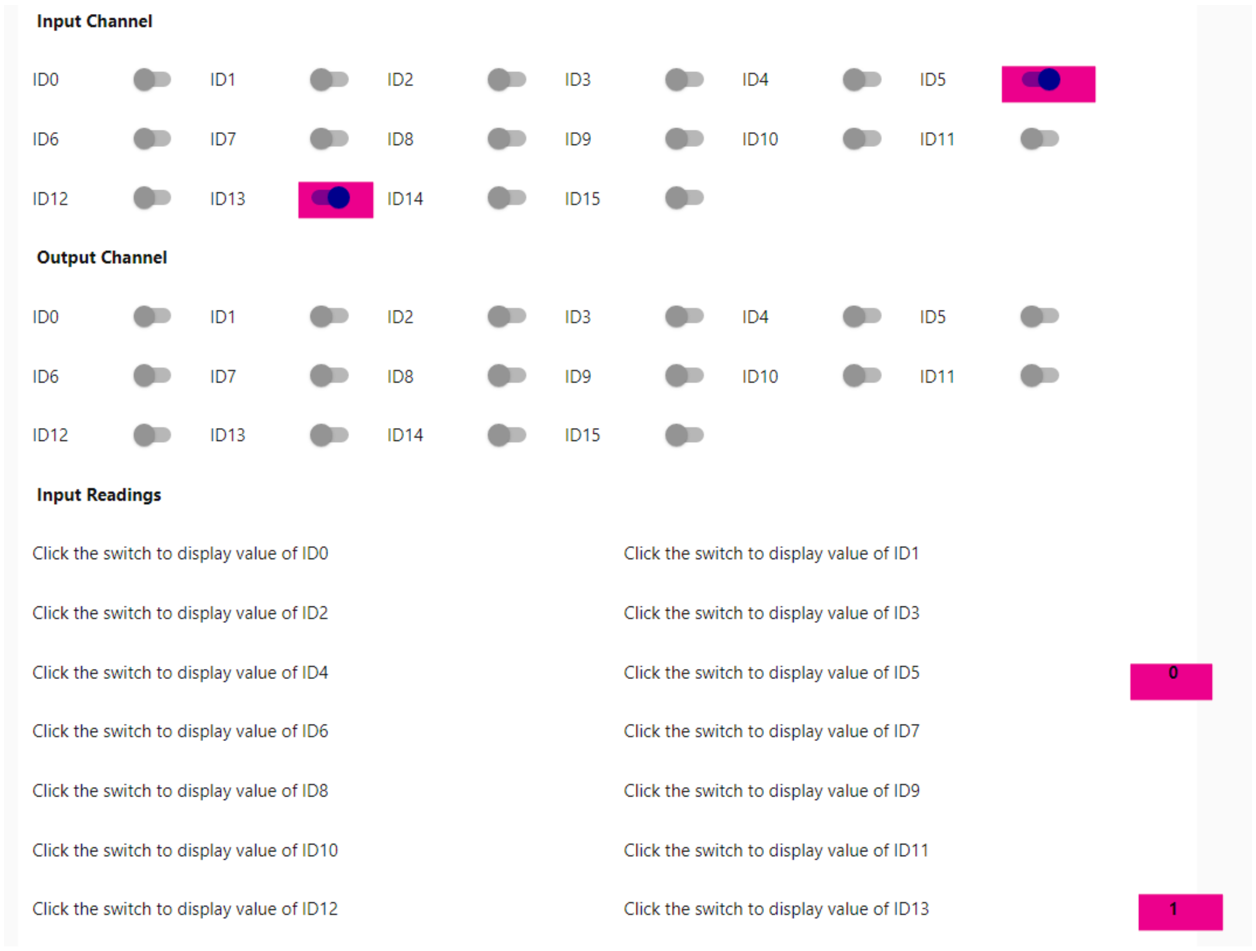
ID0	ID1	ID2	ID3	ID4	ID5
ID6	ID7	ID8	ID9	ID10	ID11
ID12	ID13	ID14	ID15		

Input Readings

Click the switch to display value of ID0	Click the switch to display value of ID1	
Click the switch to display value of ID2	Click the switch to display value of ID3	
Click the switch to display value of ID4	Click the switch to display value of ID5	1
Click the switch to display value of ID6	Click the switch to display value of ID7	
Click the switch to display value of ID8	Click the switch to display value of ID9	
Click the switch to display value of ID10	Click the switch to display value of ID11	

To connect multiple inputs and display the results on node-red dashboard:-

To connect multiple inputs, connect the input pins and the grounds to their respective channel number in the device and toggle the switch on the node-red dashboard and the result will be displayed for a result showing multiple inputs



Input Channel

ID0 ID1 ID2 ID3 ID4 ID5

ID6 ID7 ID8 ID9 ID10 ID11

ID12 ID13 ID14 ID15

Output Channel

ID0 ID1 ID2 ID3 ID4 ID5

ID6 ID7 ID8 ID9 ID10 ID11

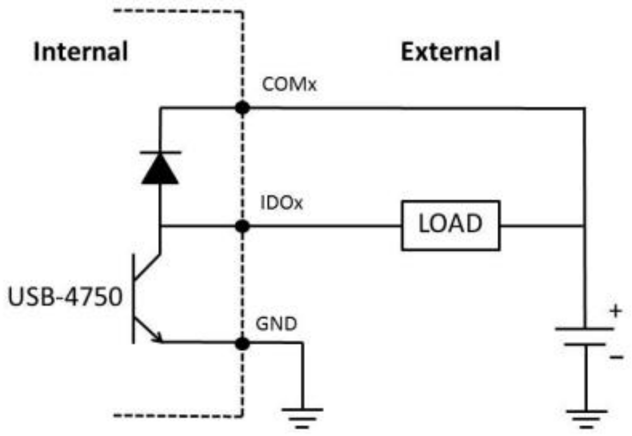
ID12 ID13 ID14 ID15

Input Readings

Click the switch to display value of ID0	Click the switch to display value of ID1	
Click the switch to display value of ID2	Click the switch to display value of ID3	
Click the switch to display value of ID4	Click the switch to display value of ID5	0
Click the switch to display value of ID6	Click the switch to display value of ID7	
Click the switch to display value of ID8	Click the switch to display value of ID9	
Click the switch to display value of ID10	Click the switch to display value of ID11	
Click the switch to display value of ID12	Click the switch to display value of ID13	1

Step by step process of writing data to output:-

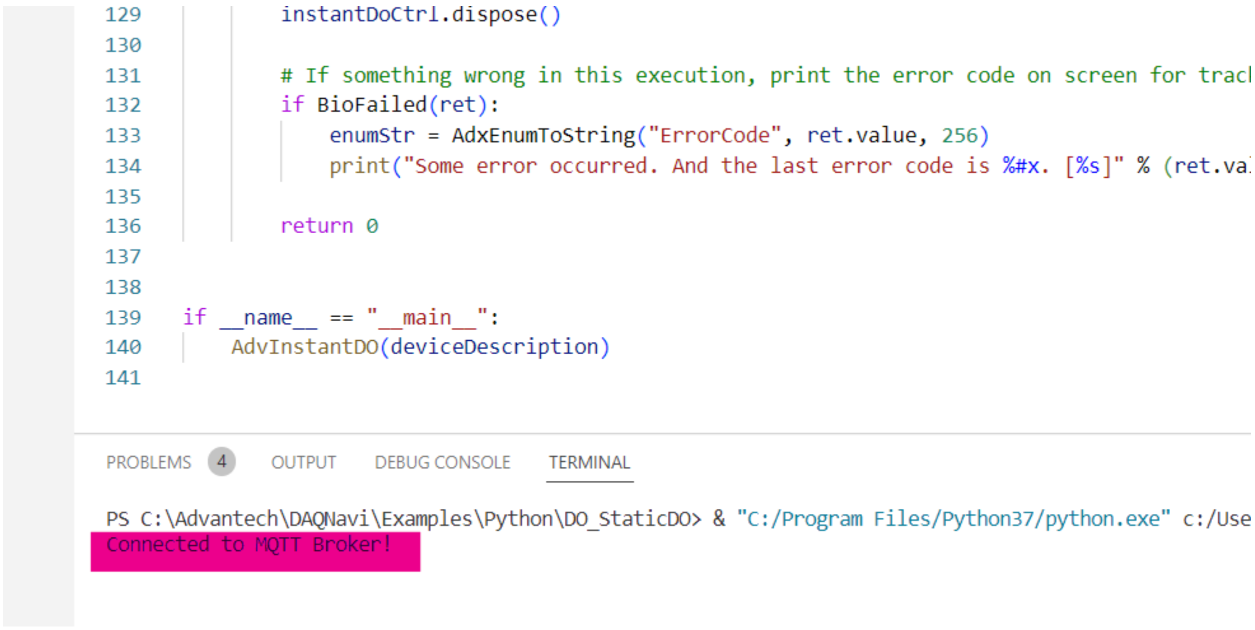
Step 1:- Please make sure that connecting the output devices to the device should be done by following the wiring schematic of the circuit given below



Step 2 :- Connect the device to the computer with all the drivers installed and run the python program provided

On successful connection with the device the terminal should give an output of

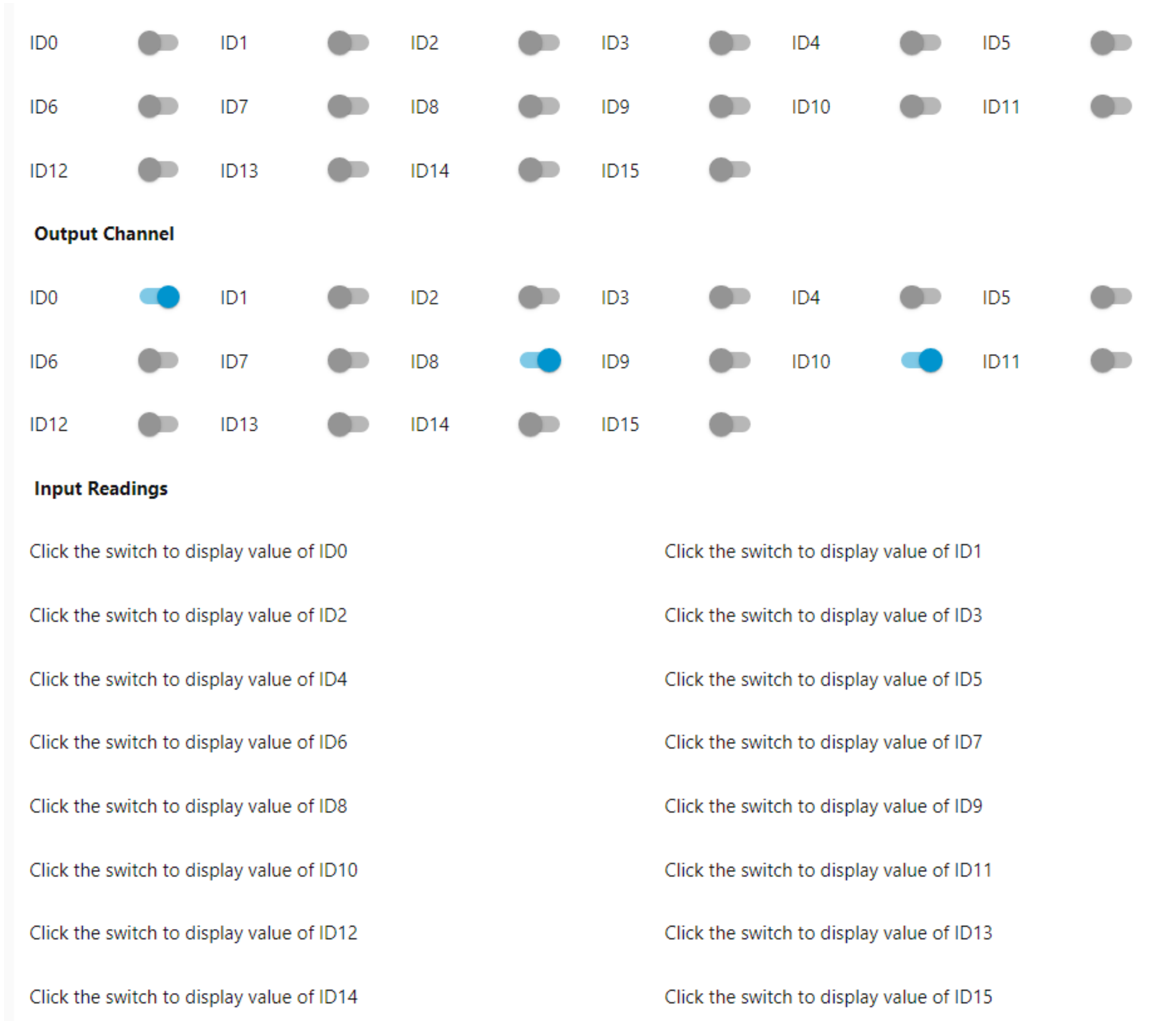
'Connected to MQTT broker' as shown below:-



```
            instantDoCtrl.dispose()

            # If something wrong in this execution, print the error code on screen for tracl
            if BioFailed(ret):
                enumStr = AdxEnumToString("ErrorCode", ret.value, 256)
                print("Some error occurred. And the last error code is %#x. [%s]" % (ret.va

            return 0


if __name__ == "__main__":
    AdvInstantDO(deviceDescription)

```

```
PS C:\Advantech\DAQNavi\Examples\Python\DO_StaticDO> & "C:/Program Files/Python37/python.exe" c:/Use
Connected to MQTT Broker!
```

Step 3 :- Go to the node-red dashboard and the port and channel number needed to be activated can be activated by pushing the switch and toggling the switch to enable the output to be sent for a single input or any number of input



ID0	○	ID1	○	ID2	○	ID3	○	ID4	○	ID5	○
ID6	○	ID7	○	ID8	○	ID9	○	ID10	○	ID11	○
ID12	○	ID13	○	ID14	○	ID15	○				

Output Channel

ID0	●	ID1	○	ID2	○	ID3	○	ID4	○	ID5	○
ID6	○	ID7	○	ID8	●	ID9	○	ID10	●	ID11	○
ID12	○	ID13	○	ID14	○	ID15	○				

Input Readings

Click the switch to display value of ID0	Click the switch to display value of ID1
Click the switch to display value of ID2	Click the switch to display value of ID3
Click the switch to display value of ID4	Click the switch to display value of ID5
Click the switch to display value of ID6	Click the switch to display value of ID7
Click the switch to display value of ID8	Click the switch to display value of ID9
Click the switch to display value of ID10	Click the switch to display value of ID11
Click the switch to display value of ID12	Click the switch to display value of ID13
Click the switch to display value of ID14	Click the switch to display value of ID15

Advantech USB-4750 Hardware Manual Links

Startup Manual Link

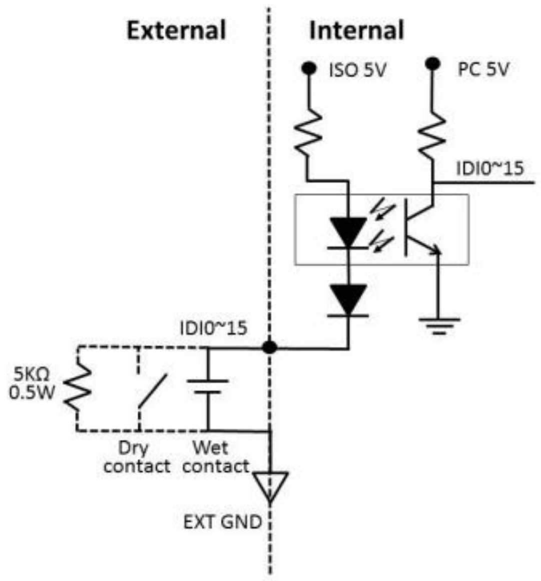
https://advdownload.advantech.com/productfile/Downloadfile2/1-14HO1G7/USB-4750_Startup%20Manaul_Ed2-2.PDF

User Manual Link

https://advdownload.advantech.com/productfile/Downloadfile2/1-2A7WYPD/USB-4750_User_Manual_Ed.3_FINAL.pdf

Input Contacts

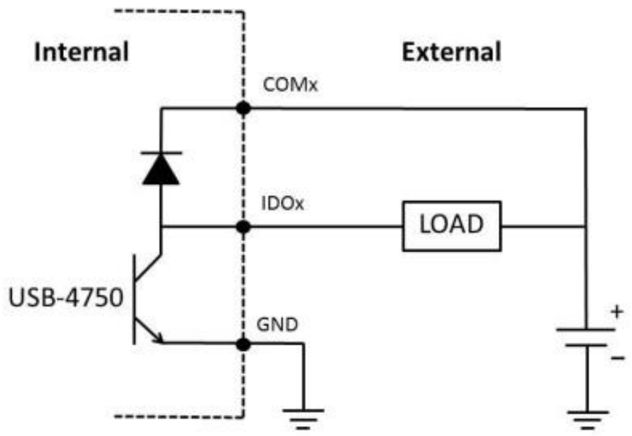
The circuits for the dry/wet contacts for the inputs is as follows:-



Each input can support up to 5-60V DC, the dry contact capability will allow the channel to respond to changes in external circuitry when no voltage is present in the external circuit

Output Contacts:

The circuitry contacts for the inputs is as follows:-



Appendix (Code)

InputDI

```
"""
import sys
sys.path.append('..')
from CommonUtils import kbhit
import time

from Automation.BDaq import *
from Automation.BDaq.InstantDiCtrl import InstantDiCtrl
from Automation.BDaq.BDaqApi import AdxEnumToString, BioFailed

import random

from paho.mqtt import client as mqtt_client

broker = 'localhost'
port = 1883
topic = "PORT1"
topic_1 = "PORT2"
reverse_invert = "INVERT1"
reverse_invert1 = "INVERT2"

deviceDescription = "USB-4750,BID#0"
profilePath = u"..\\..\\profile\\USB_4750.xml"
startPort = 0
portCount = 2



def AdvInstantDI():
    ret = ErrorCode.Success

    # Step 1: Create a 'InstantDiCtrl' for DI function.
     # Select a device by device number or device description and specify the access
mode.
    # In this example we use ModeWrite mode so that we can fully control the device,
    # including configuring, sampling, etc.
    instantDiCtrl = InstantDiCtrl(deviceDescription)

    for _ in range(1):
            instantDiCtrl.loadProfile = profilePath
```

```
# Step 2: Read DI ports' status and show.
    print("Reading ports status is in progress, any key to quit!")

    def connect_mqtt():
        def on_connect(client, userdata, flags, rc):
            if rc == 0:
                print("Connected to MQTT Broker!")
            else:
                print("Failed to connect, return code %d\n", rc)

        client = mqtt_client.Client()
        client.on_connect = on_connect
        client.connect(broker, port)
        return client

    def reverse_binary_string(s):
         # Convert the string to a list of characters
        char_list = list(s)

         # Reverse the list
        char_list.reverse()

         # Join the list back into a string and return it
        return ''.join(char_list)

    while not kbhit():
        def publish(client):
            msg_count = 0
            while True:
                time.sleep(1)
                ret, data = instantDiCtrl.readAny(startPort, portCount)
                if BioFailed(ret):
                    break

                for i in range(startPort, startPort + portCount):
                   print("DI port %d status is %#x" % (i, data[i-startPort]))

                msg_1 = data[startPort]
                msg_1 = '{0:08b}'.format(msg_1)
                msg = msg_1
                binary_string = msg
                reversed_string = reverse_binary_string(binary_string)
                invert_1 = reversed_string
```

```
msg_2 = data[i-startPort]
print(type([i-startPort]))
msg_2 = '{0:08b}'.format(msg_2)
binary_string_1 = msg_2
reversed_string_1 = reverse_binary_string(binary_string_1)
invert_2 = reversed_string_1




result = client.publish(topic, msg)
# result: [0, 1]
status = result[0]
if status == 0:
    print(f"Send `{msg}` to topic `{topic}`")
else:
    print(f"Failed to send message to topic {topic}")


result = client.publish(topic_1, msg_2)
# result: [0, 1]
status = result[0]
if status == 0:
    print(f"Send `{msg_2}` to topic `{topic_1}`")
else:
    print(f"Failed to send message to topic {topic_1}")

result = client.publish(reverse_invert, invert_1)
# result: [0, 1]
status = result[0]
if status == 0:
    print(f"Send `{invert_1}` to topic `{reverse_invert}`")
else:
 print(f"Failed to send message to topic {reverse_invert}")

result = client.publish(reverse_invert1, invert_2)
# result: [0, 1]
status = result[0]
if status == 0:
    print(f"Send `{invert_2}` to topic `{reverse_invert1}`")
else:
                 print(f"Failed  to  send  message  to  topic
{reverse_invert1}")
```

```
                        msg_count += 1
                        time.sleep(1)

    def run():
        client = connect_mqtt()
        client.loop_start()
        publish(client)

    if __name__ == '__main__':
        run()

# Step 3: Close device and release any allocated resource
    instantDiCtrl.dispose()

    # If something wrong in this execution, print the error code on screen for tracking.
    if BioFailed(ret):
        enumStr = AdxEnumToString("ErrorCode", ret.value, 256)
       print("Some error occurred. And the last error code is %#x. [%s]" % (ret.value,
enumStr))

    return 0

if __name__ == '__main__':
    mainData = AdvInstantDI()
```

InputDO

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

from paho.mqtt import client as mqtt_client
broker = 'localhost'
port = 1883
topic = "DAQ/DO"
MSG = ""

"""
/*******************************************************************************
Copyright (c) 1983-2021 Advantech Co., Ltd.
********************************************************************************
THIS IS AN UNPUBLISHED WORK CONTAINING CONFIDENTIAL AND PROPRIETARY INFORMATION
WHICH IS THE PROPERTY OF ADVANTECH CORP., ANY DISCLOSURE, USE, OR REPRODUCTION,
WITHOUT WRITTEN AUTHORIZATION FROM ADVANTECH CORP., IS STRICTLY PROHIBITED.

================================================================================
REVISION HISTORY
--------------------------------------------------------------------------------
$Log:  $

--------------------------------------------------------------------------------
$NoKeywords:  $
*/

/******************************************************************************
*
* Windows Example:
*    StaticDO.py
*
* Example Category:
*    DIO
*
* Description:
*    This example demonstrates how to use Static DO function.
*
* Instructions for Running:
*    1. Set the 'deviceDescription' for opening the device.
*    2. Set the 'profilePath' to save the profile path of being initialized
device.
*    3. Set the 'startPort'as the first port for Do .
*    4. Set the 'portCount'to decide how many sequential ports to operate Do.
```

```
*
* I/O Connections Overview:
*    Please refer to your hardware reference manual.
*
******************************************************************************/
"""
import sys
sys.path.append('..')

from Automation.BDaq import *
from Automation.BDaq.InstantDoCtrl import InstantDoCtrl
from Automation.BDaq.BDaqApi import AdxEnumToString, BioFailed

deviceDescription = "USB-4750,BID#0"
profilePath = u"..\\..\\profile\\DemoDevice.xml"
startPort = 0
portCount = 1


def connect_mqtt() -> mqtt_client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)

    client = mqtt_client.Client('')
    client.on_connect = on_connect
    client.connect(broker, port)
    return client


def subscribe(client: mqtt_client,deviceDescription):
    def on_message(client, userdata, msg):
        print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
        temp(deviceDescription,msg.payload)

    client.subscribe(topic)
    client.on_message = on_message

def run(instantDoCtrl):
    client = connect_mqtt()
    subscribe(client,instantDoCtrl)
    client.loop_forever()
```

```
def AdvInstantDO(deviceDescription):

    ret = ErrorCode.Success

    # Step 1: Create a instantDoCtrl for DO function.
    # Select a device by device number or device description and specify the
access mode.
    # In this example we use ModeWrite mode so that we can fully control the
device,
    # including configuring, sampling, etc.

    run(deviceDescription)



def temp(deviceDescription,msg):
    instantDoCtrl = InstantDoCtrl(deviceDescription)

    for _ in range(1):
        instantDoCtrl.loadProfile = profilePath
        print(msg)

        # Step 2: Write DO ports
        dataBuffer = [0] * portCount
        for i in range(startPort, portCount + startPort):
            # inputVal = input("Input a 16 hex number for DO port %d to
output(for example, 0x00): " % i)
            inputVal = msg
            if not isinstance(inputVal, int):
                inputVal = int(inputVal, 16)

            dataBuffer[i-startPort] = inputVal

        ret = instantDoCtrl.writeAny(startPort, portCount, dataBuffer)
        if BioFailed(ret):
            break
            print("DO output completed!")

        # Step 3: Close device and release any allocated resource.
        instantDoCtrl.dispose()

        # If something wrong in this execution, print the error code on screen
for tracking.
        if BioFailed(ret):
            enumStr = AdxEnumToString("ErrorCode", ret.value, 256)
            print("Some error occurred. And the last error code is %#x. [%s]" %
(ret.value, enumStr))

        return 0


if __name__ == "__main__":
    AdvInstantDO(deviceDescription)
```

In the input section:

The function:-

connect_mqtt() – Set ups the connection between node-red and the python file using MQTT

AdvantisDI() – Processes the data, converts it from hexa-decimal to binary to make it easier for node-red to read the data and process it

In the output section:-

connect_mqtt() – Set ups the connection between node-red and the python file using MQTT

AdvantisDO() – Receives the push signal from node-red using MQTT and lights up the intended port in the device