Programming the Tabor Proteus Arbitrary Waveform Generator (AWG) and Arbitrary Waveform Transceiver (AWT) using Python

Applies to all models – P1282M, P1284M, P2582M, P2584M, P9082M, P9484M, P1282D, P1284D, P1288D, P12812D, P2582D, P2584D, P2588D, P25812D, P9082D, P9084D, P9086D, P9484D, P9488D, P94812D, P1282B, P1284B, P1288B, P12812B, P2582B, P2584B, P2588B, P25812B, P9082B, P9084B, P9086B, P9484B, P9488B, P94812B.

The Tabor Proteus family of Arbitrary Waveform Generators (AWG) and Arbitrary Waveform Transceivers (AWT) are programmed using the industry standard instrument programming language SCPI.

How to connect to the instrument using Python.

Proteus is a modular system, the B and D models have an embedded instrument controller, with a SCPI server that allows for physical connectivity via LAN or USB. The M models are designed to operate in the PXIe standard Chassis and connectivity is direct through the PCIe physical interface. In both cases commands are sent to the Instruments DLL, that should be installed on your local PC.

First and most important step: Install Wave Design Studio, this will install the DLL, and the appropriate VISA drivers. Once this is installed you are ready to program your instrument. Open your favorite IDE or text editor and ensure the appropriate libraries will be imported.

```
import os
import sys
srcpath = os.path.realpath('SourceFiles')
sys.path.append(srcpath)
```

Ensure SourcesFiles.py path is correctly defined. Connect the D or B models using a LAN or USB cable. If Proteus is an M version ensure the module is installed in a PXIe chassis, and the chassis is connected to your PC.

Determine the instruments address: Run Wave Design Studio (WDS) – this will search for all compatible instrument IP address or PXIe slot numbers. Make a note of the instruments IP address, or Slot number, and close the application. If no instrument is found, then please check connectivity.

For connecting via IP address (B and D models) add the following lines of code.

```
from tevisainst import TEVisaInst
inst_addr = 'TCPIP::169.254.169.109::5025::SOCKET' # <- use the IP address WDS found.
inst = TEVisaInst(inst_addr)
resp = inst.send_scpi_query("*IDN?")
print('connected to: ' + resp)
```

For connecting via PCIe in a PXIe modular system add the following lines of code instead.

```
from teproteus import TEProteusAdmin as TepAdmin
admin = TepAdmin() #required to control PXI module
sid = 12 #PXI slot WDS found
inst = admin.open_instrument(slot_id=sid)
```

```
resp = inst.send_scpi_query("*IDN?")
print('connected to: ' + resp) # Print *IDN
```

Save the file as connect.py and on the command line type python connect.py. This will run the script. If connection is successful the Instruments model number serial number, etc. will be return. And you have a pointer for further programming called inst. From here on programming of the instrument is the same.

Make an Arbitrary Waveform using Python.

Let's make a sine wave: In the following code we define a 10-cycle sine wave of 1024 points.

```
import matplotlib.pyplot as plt
import numpy as np

sampleRateDAC = 1E9
amp = 1
cycles = 10
segLen = 1024 # must be a multiple of 64
time = np.linspace(0, segLen-1, segLen)
w = 2 * np.pi * cycles
dacWave = amp * np.sin(w*time/segLen)
print('Frequency {0} Hz'.format(sampleRateDAC*cycles/segLen))
```

Note the frequency of the sine is the sample rate of the instrument, multiplied by the number of cycles over the segment length. The max value is 1, minimum value is -1 and the DC point is 0. We need to scale this to the DAC, (Digital to Analog Converter) which is 16 bits and covert to type uint16.

```
max_dac=65535 # Max Dac
half_dac=max_dac/2 # DC Level
data_type = np.uint16 # DAC data type
dacWave = ((dacWave) + 1.0) * half_dac
dacWave = dacWave.astype(data_type)
```

The minimum value should now be zero and the maximum value with be two to the power of 16.

Programming the Tabor Proteus

We are now ready to program the instrument. Initialize the instrument, then select the instrument channel you which to use, set the sample rate, clear the memory and set the mode to continuous playback.

```
inst.send_scpi_cmd('*CLS; *RST') # Init instrument
#AWG channel
ch = 1 # everything after relates to CH 1
cmd = ':INST:CHAN {0}'.format(ch)
inst.send_scpi_cmd(cmd)
cmd = ':FREQ:RAST {0}'.format(sampleRateDAC)
inst.send_scpi_cmd(cmd)
cmd = ':TRAC:DEL:ALL' # Clear CH 1 Memory
inst.send_scpi_cmd(cmd)
```

```
cmd = ':INIT:CONT ON' # play waveform continuously
inst.send_scpi_cmd(cmd)
```

We are now ready to download the waveform into the instrument. This process involves selecting a memory location and memory size to store the waveform in. The size of the waveform is 1024 points, and we will store that in memory location number 1, or what we call waveform segment 1.

```
segnum = 1

cmd = ':TRAC:DEF {0}, {1}'.format(segnum, len(dacWave)) # memory location and length
inst.send_scpi_cmd(cmd)
 # Select the segment
cmd = ':TRAC:SEL {0}'.format(segnum)
inst.send_scpi_cmd(cmd)
```

Sometimes waveforms can be very large so its good practice to increase the instrument timeout length, before downloading the date.

```
inst.timeout = 30000 #increase
inst.write_binary_data('*OPC?; :TRAC:DATA', dacWave) # write, and wait while *OPC completes
inst.timeout = 10000 # return to normal
```

The instrument now has a waveform stored in its internal segment memory location number 1. The next step is to play it out. This is simply done by selecting segment and switching the instruments output on.

```
cmd = ':FUNC:MODE:SEGM {0}'.format(segnum)
inst.send_scpi_cmd(cmd)
cmd = ':OUTP ON'
rc = inst.send_scpi_cmd(cmd)
```

The waveform is now playing on channel one.

A more complex sequence of waveforms can be realized using the task table. Let's create a DC waveform. And load it into segment memory location number 2.

```
segnum = 2

dacWaveDC = np.ones(segLen)
dacWaveDC = dacWaveDC * half_dac  # scale
dacWaveDC = dacWaveDC.astype(data_type)

cmd = ':TRAC:DEF {0}, {1}'.format(segnum, len(dacWaveDC)) # memory location and length
inst.send_scpi_cmd(cmd)
cmd = ':TRAC:SEL {0}'.format(segnum)
inst.send_scpi_cmd(cmd)
inst.timeout = 30000 #increase
inst.write_binary_data('*OPC?; :TRAC:DATA', dacWaveDC) # write, and wait while *OPC completes
inst.timeout = 10000 # return to normal
```

Now we can create a task table to play the sine wave once and the DC times then repeat. Replace the two following lines of code:

```
cmd = ':FUNC:MODE:SEGM {0}'.format(segnum)
inst.send_scpi_cmd(cmd)
```

With the task table definition as follows:

```
cmd = ':TASK:COMP:LENG 2' # set task table length
inst.send_scpi_cmd(cmd)

cmd = ':TASK:COMP:SEL 1' # set task 1
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:SEGM 1'
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:NEXT1 2'
inst.send_scpi_cmd(cmd)

cmd = ':TASK:COMP:SEL 2' # set task 2
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:SEGM 2'
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:LOOP 10'
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:NEXT1 1'
inst.send_scpi_cmd(cmd)

cmd = ':TASK:COMP:WRITE'
inst.send_scpi_cmd(cmd)
cmd = ':SOUR:FUNC:MODE TASK'
inst.send_scpi_cmd(cmd)
```

Run your python script and you should see a pulsed sine wave.

Programing and Using the Digital IQ Modulator

The Proteus AWG has a digital implementation of an IQ modulator and Local Oscillator. The Oscillator is digital implemented using an NCO (Numerically Controlled Oscillator). Instead of generating a sine wave in segment one, lets use the NCO and pulse it on and off. The advantage of this over creating a pulse is that the NCO, or our LO becomes coherent.

First let’s change segment one to generate a pulse, modify the code as follows:

```
segnum = 1

dacWave = np.ones(segLen)
dacWave = dacWave * max_dac
dacWave = dacWave.astype(data_type)

cmd = ':TRAC:DEF {0}, {1}'.format(segnum, len(dacWave)) # memory location and length
inst.send_scpi_cmd(cmd)
cmd = ':TRAC:SEL {0}'.format(segnum)
inst.send_scpi_cmd(cmd)
inst.timeout = 30000 #increase
inst.write_binary_data('*OPC?; :TRAC:DATA', dacWave) # write, and wait while *OPC completes
inst.timeout = 10000 # return to normal
```

If you now run the script, you will see on a scope an unmodulated pulse. Now we must modify the above script so we can make segment one and two IQ data. To do that the dacWave array needs to be reshaped so the instrument accepts IQIQIQIQIQ data in the segment.

```
# reshapes for IQIQIQIQIQ....
arr_tuple = (dacWave, dacWave)
dacWave = np.vstack(arr_tuple).reshape((-1,), order='F')
```

Also do the same for the DC pulse, after you defined the DC pulse and before you download segment two add the following code.

```
# reshapes for IQIQIQIQIQ....
arr_tuple = (dacWaveDC, dacWaveDC)
dacWaveDC = np.vstack(arr_tuple).reshape((-1,), order='F')
```

After the second segment download code insert the following code to enable Digital Upconverting Mode (DUC) and use the instruments interpolator to provide a low pass filter function before the modulator.

```
cmd = ':SOUR:MODE DUC'
resp = inst.send_scpi_cmd(cmd)
cmd = ':SOUR:IQM ONE' #
resp = inst.send_scpi_cmd(cmd)
# Set x8 interpolation
cmd = ':SOUR:INT X8'
resp = inst.send_scpi_cmd(cmd)
# Reset sample rate to interpolated rate
sampleRateDACInt = sampleRateDAC * 8
cmd = ':FREQ:RAST {0}'.format(sampleRateDACInt)
resp = inst.send_scpi_cmd(cmd)
#Set the LO (NCO)
ncoFreq = 10E6
cmd = ':SOUR:NCO:CFR1 {0}'.format(ncoFreq)
resp = inst.send_scpi_cmd(cmd)
```

Run this modified code and you should see a pulse modulated waveform at the frequency set by ncoFreq.

Now it’s time to digitize the signal.

Acquiring Signals with the Proteus Arbitrary Waveform Transceiver. (AWT)

We can deterministically capture the signal we just generated using the instruments optional digitizer, option number AWT. This is a two-channel digitizer. That can be operated in both dual and single modes. In the single mode the digitizers sample rates doubles, at the cost of losing a channel. Connect Channel 1 of the AWG to Channel 1 of the digitizer using a short SMA cable.

We are going to trigger the digitizer from the task table, to do this we need to modify the above table and change task one by adding the digitizer trigger command.

```
cmd = ':TASK:COMP:SEL 1'
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:SEGM 1'
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:DTR ON' # Trigger the Digitizer
inst.send_scpi_cmd(cmd)
cmd = ':TASK:COMP:NEXT1 2'
inst.send_scpi_cmd(cmd)
```

Now we can go ahead and setup the digitizer. We will add the following code to the digitizer. Capturing is performed using a frame system, you define the number of frames you want to capture and the length of each frame. This is especially useful if you are capturing pulses, you can capture the 'on' portion of the signal only and store it as a frame. In this example we will capture a single frame of length 9600 (note length must be a multiple of 96. We will also reserve some memory to place the capture in of type unit16 and set the sample rate to 1GS/s

```
numframes, framelen = 1, 9600
totalLen = numframes * framelen
wav1 = np.zeros(framelen, dtype=np.uint16)
inst.send_scpi_cmd(':DIG:MODE DUAL') # enables CH1 and CH2, max rate is 2.5GS/s
sampleRateADC = 1E9
cmd = ':DIG:FREQ  {0}'.format(sampleRateADC)
inst.send_scpi_cmd(cmd)
```

The next step is to select and enable channel one, set the trigger, define the frame memory and length, specify the first and last frame to capture and take the acquisition.

```
inst.send_scpi_cmd(':DIG:CHAN:SEL 1')
inst.send_scpi_cmd(':DIG:CHAN:STATE ENAB')
inst.send_scpi_cmd(':DIG:TRIG:SOURCE TASK1')
cmd = ':DIG:ACQuire:FRAM:DEF {0},{1}'.format(numframes, framelen)
inst.send_scpi_cmd(cmd)
capture_first, capture_count = 1, numframes
cmd = ':DIG:ACQuire:FRAM:CAPT {0},{1}'.format(capture_first, capture_count)
inst.send_scpi_cmd(cmd)
# Start the digitizer's capture machine
inst.send_scpi_cmd(':DIG:INIT ON')
inst.send_scpi_cmd(':DIG:TRIG:IMM')
inst.send_scpi_cmd(':DIG:INIT OFF')
```

The acquired data is now in the memory of the Proteus Arbitrary Waveform Generator. The next step is to tranter the data to your local machine and plot the results of the acquisition. The following codes tells the AWT that you want readback all the data, and readback the frame (you can also read a time stamped header)

```
inst.send_scpi_cmd(':DIG:DATA:SEL ALL')
inst.send_scpi_cmd(':DIG:DATA:TYPE FRAM')
resp = inst.send_scpi_query(':DIG:DATA:SIZE?')
```

```
rc = inst.read_binary_data(':DIG:DATA:READ?', wav1, totalLen)
plt.plot(wav1)
plt.show()
```

If the plot shows a modulated pulse, well done! You can use some of the instrument’s core features. More advanced features are documented in the manuals, along with further examples on our GitHub page. The complete script code set is available to download in our download section.