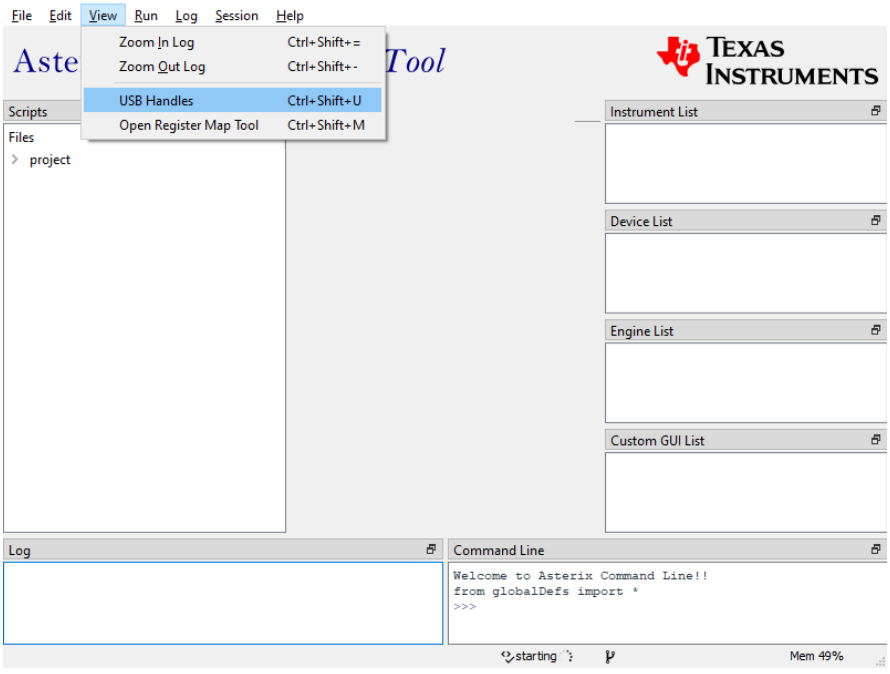
Python 3 script for controlling Ultrasound TX EVMs

Files

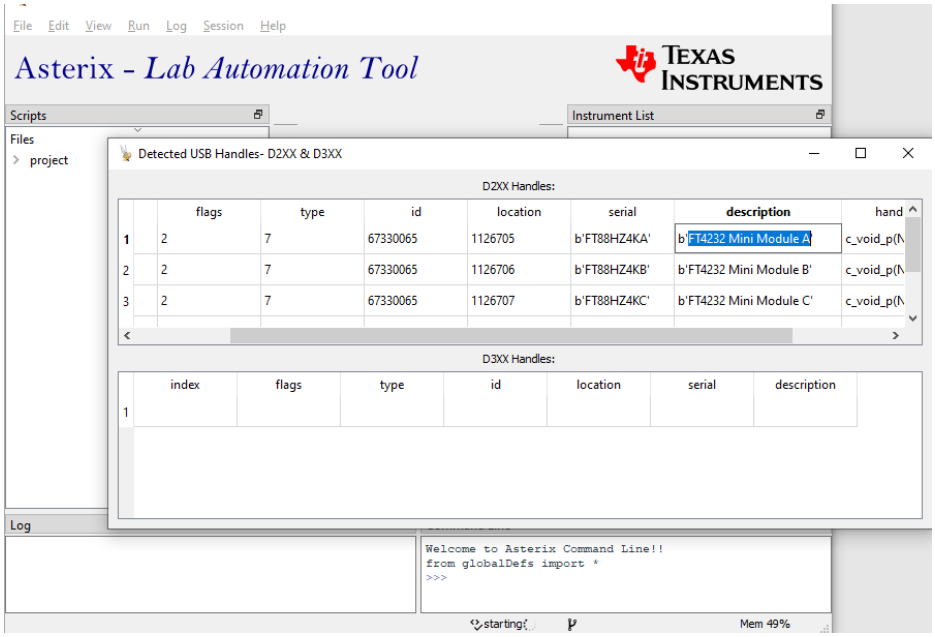
- **deviceController.py** – The main file containing the core read and write functions as well as the device initialization function.
- **Example.py** – This file provides an example code on how to initialize the device object as well as read and write to the device
- **Requirements.txt** – This file contains the list of python libraries that are mandatory to run the device. The version numbers are recommendations. You can try with other versions of these libraries as well.

How to find the device address

1. Make sure the EVM is connected to the PC
2. Open the GUI. (No need to run devInit.py)
3. Click on View-> USB Handles



4. In the window that opens, the address is visible under **description** column as the first row (assuming no other EVMs are connected). For example, If the description is b'FT4232 Mini Module A', the address you will need to use in the code is **FT4232 Mini Module A**. Here, **FT4232 Mini Module** is the address of the FTDI device and **A** corresponds to the port used. Most of our EVMs utilize port A of the FTDI device.



Using the device functions

The basic device read and write functions are defined in **deviceController.py** file inside the **USBQPort** class. Essentially these functions are all you need to communicate with the device once **USBQPort** object is initialized.

- **writeReg(addr,val)** – This function takes 2 arguments
  - **addr –** the integer address
  - **val –** the integer value

  This writes the **val** into the register address **addr**

- **readReg(addr)** – This function takes 1 argument
  - **addr** – the integer address to be read back from

  This function reads from the device register at address **addr** and returns the inter value read.

  Note that for the read function to work, the read enable register (Register 0 bit 1) needs to be set to 1. Otherwise the read back value will always be zero. The register can be set using **writeReg** as given in the example script

The **example.py** script contains the code to instantiate the device object. This device object will be needed throughout for communication. To initialize the device, you will need to specify the device address as a function parameter. Steps on how to get the device address is explained in the previous session.

The **example.py** script also contains the following function definitions

- **deviceWrite(address,data,pageSelect)**
- **deviceRead(address,pageSelect)**

which, acts as a one-shop read and write functions and takes care of the page select in case you wish to write to memory pages, and also takes care of the read enable register setting so that single function call to **deviceRead** will read back and return data from GBL_PAGE registers or even the memory.

The pageSelect should be a 16 bit number with each bit corresponding to a page, starting from LSB – Page 1 to MSB – Page 16 (Refer datasheet).

You can either use the **USBQPort** class's **writeReg** and **readReg** functions or the **deviceWrite** and **deviceRead** functions in your code to communicate with the device.

To configure the SPI port, the following variables can be changed in the **deviceController.py** file's **USBQPortController** class

```
    msbFirst = 1
    '''MSB first or LSB first'''
    clkEdge = 1
    '''Clock detected on positive rising edge (1) or falling edge(0)'''
    readClkEdge = 0
    '''Clock detected on positive rising edge (1) or falling edge(0) (Read
mode)'''
    packetLen = 44
    '''Total packet length. address + data lengths'''
    addressLen = 12
    '''Length of address in bits'''
    packetOrder = 0
    '''Address first (0) or data first (1)'''

    enableBit = 4
    '''Which pin of the port is SEN'''
    clkBit = 1
    '''Which pin of the port is clk'''
    dataBit = 2
    '''Which pin of the port is data'''
    dataOutBit = 0
    '''Which pin is SDOUT from device'''
    enableHigh = 0
    '''SEN is active high or active low'''
    readOutMode = 0
    '''1 -> 3 wire SPI'''
```