last update 7-17-17

LL_USB_python Python sample app for Linux – overview

Sample app "LL_USB_python" makes use of "libLL_USB30x.so" built and tested in Oracle VM VirtualBox, within Win7, running ubuntu 14.04 LTS. Built with installed library **lusb1.0** and library, **pthread 1.1**. "libLL_USB30x.so" will need to be built and installed as per the instructions accompanying it in order for this app to make use of it.

The library makes use of many of the functions mentioned in the documentation for our Windows DLL API which can be found at our website. The link below provides the API as well as much more information that further explains using the functions, and some information concerning interaction with the hardware. Below is the link:

lawsonlabs.com web link

The functions currently available can be found in the "main.h" header file distributed with the library source code, many of which are recognized by the "EX_" preceding the rest of the function name. Other functions within the library, not preceded by "EX_" which wrap, extend, and are called by those functions, can also be called by themselves.

Sample app LL_USB_python makes use of only a handful of the functions available, in order to demonstrate their usage in connecting to the device and performing some basic tasks. The app connects to the device, reads/displays the voltage from the default channel 0, sets channel 6, and then reads/displays the voltage again. Next the app sets DAC 0 to 3.5 volts and changes to channel 3. Please connect DAC 0 to channel 3 (analog out 1 to 4+ and GD to 4-). The app reads the voltage from that channel (should read approx. 3.5 volts). It then demonstrates how to get the "actual rate" the board will run at based on a requested rate, since the board is not capable of always running at the precise rate one might request. It sets a temporary rate variable to 1000Hz and then calls into the library to get what the "actual board rate" would be (approx. 1000.651466Hz) at that requested rate. Note that trying to set the rate, for example with a call to **EX_SendRate(...)** with the rate that was returned by that call, could then set it to a different rate. That returned rate should only be used within an app for timing within the app. Finally, the app does a single-channel 100Hz scan on channel 3, displaying 25 voltages, then exits.

It makes use of the following function calls within the library:

EX_ConnectOneDevice(...)
EX_GetOneConversion(...)
EX_SendChan(...)
EX_SendDAC(...)
EX_GetCalculatedRate(...)

the following is also used by this app but is not shown in the Windows DLL API at our website as it is exclusive to our Linux library and is the function passed to the scan thread:

doScan(...)

The device ID is hard-coded to 5206 at the top of the "main()" function call so that will need to be changed to "your" device ID before running your Linux system.

The next page shows how it looks when run, which may require super user rights, "*sudo ./LL_USB_python*." Note that the single-channel scan data is for channel 3, which was the last channel selected, and DAC0 was connected to it with it's output set to 3.5 volts. Also note that some of the output shown is being printed by the library since ***AllowPrintf(...)*** has been called by the application to set a flag that enables that option.

C coded header file, "main.h" from the library, is included within the archive for reference to all variables and functions available to the python developer from the libLL_USB30x.so library. It can be viewed in the default linux/ubuntu text editor.

```
signing on (may take a while) - WAIT . . .
  signed on

getting a conversion - WAIT . . .
  got volts: -0.000012

changing channel - WAIT . . .
  sendChan success, lastDigin: 0

getting a conversion - WAIT . . .
  got volts: 4.999999

setting DAC0 to 3.5 volts - WAIT . . .
  sendDAC success

changing channel 3 (should be DAC0 volts) - WAIT . . .
  sendChan success, lastDigin: 0

getting a conversion - WAIT . . .
  got volts: 3.510315

checking 'actual rate' when 1000Hz is requested - WAIT . . .
  1000Hz would be: 1000.651466

starting scan

here comes the data
getting scan data
scan: 0 volts:   3.510501
scan: 1 volts:   3.510427
scan: 2 volts:   3.510458
scan: 3 volts:   3.510668
scan: 4 volts:   3.510732
scan: 5 volts:   3.510554
scan: 6 volts:   3.510596
scan: 7 volts:   3.510728
scan: 8 volts:   3.510654
scan: 9 volts:   3.510392
scan: 10 volts: 3.510401
scan: 11 volts: 3.510525
scan: 12 volts: 3.510537
scan: 13 volts: 3.510496
scan: 14 volts: 3.510472
scan: 15 volts: 3.510501
scan: 16 volts: 3.510501
scan: 17 volts: 3.510599
scan: 18 volts: 3.510549
scan: 19 volts: 3.510644
scan: 20 volts: 3.510746
scan: 21 volts: 3.510732
scan: 22 volts: 3.510563
scan: 23 volts: 3.510654
scan: 24 volts: 3.510838
starting wait for end-scan codes echo
end-scan codes: 0, 23, 3, 2, 1
done scan
```