Basics 02 : Firsts Steps

This document describes the first steps to work with the SLab system. That includes programming the hardware board and setting up the Python environment.

BOM	Hardware Board Communication cable

Index

Introduction

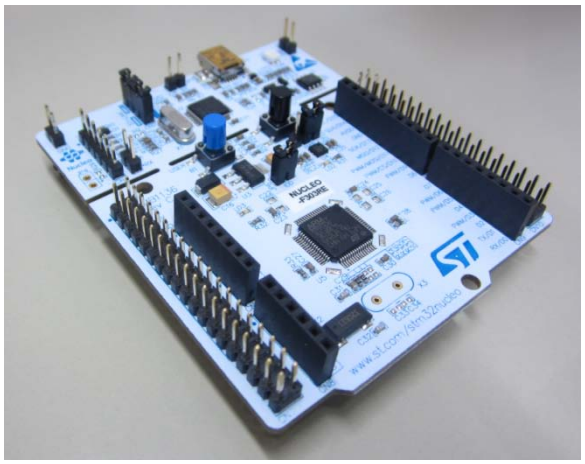
The hardware board is the basic instrument in the SLab Python project. It includes, at least, two DACs and four ADCs. A Digital to Analog Converter (DAC) is a device that, given an input digital number, can generate a voltage output proportional to this number. An Analog to Digital Converter (ADC) is a device that, given voltage a voltage in its input, measures it and returns a digital number that is proportional to the measured voltage. So, if we have a circuit to test, we can force two voltages using the two DACs and we can measure up to four node voltages using the four ADCs.

The board communicates with a PC using a serial COM link. Classic serial RS-232 links are not implemented in most modern PCs, so the serial connection is usually implemented inside the protocol of an USB link. The board is usually powered by the PC so no external power supplies are needed.

Any programmable board that features a COM link, and, at least, two DACs and four ADCs, can be used as the hardware board if it is programmed with the proper firmware. An Arduino UNO board won't do, for instance, because although it features several ADCs, it doesn't include any DAC.

The board firmware basically waits for a command request from the PC. When a command is received it interacts with the ADCs and DACs depending on the request. On some requests, information is brought back to the PC when the command ends.

In this document we will use the STM32 Nucleo64 F303RE demonstration board, shown in the figure below, as the hardware board. As the full name is too long, we will call it the F303RE Board in the rest of this document.



This is an official SLab hardware board in the sense that a firmware is provided for it. Consult the SLab Python Reference for up to date information of any other officially supported boards. You are free to use any non official board but, in this case, writing the firmware is up to you.

The F303RE Board is microcontroller board that ST provides as a way to demonstrate one of their ARM Cortex 32 bit microcontrollers, the STM32F403RE. The manufacturer produces the Nucleo Board and sells them practically at cost price to promote their microcontrollers. ST produces also the Discovery series of boards that usually include a microcontroller and some peripherals.

There are three series of Nucleo boards, the Nucleo 32 , Nucleo 64 and Nucleo 144 that feature 32 pin, 64 in or 144 pin microcontrollers. All the Nucleo boards are MBED enabled. That means that they all are programmable using a web based MBED compiler at its developer site. So you don't need to install any toolchain to program a Nucleo board. In fact, the current version of the SLab firmware have been compiled using MBED.

If at any point you are tired of the SLab work, you can start learning about MCU programming using the F303RE board in the MBED environment.

An MBED enabled board also features an easy way to program the board with a binary firmware file. The board, upon connection through USB, is seen as a mass storage disk. Just drag and drop the firmware file over the disk and you are done.

Connecting the board to the PC

In order to use the board, in our example, the F303RE Board, we need to connect it to a PC. In some cases it will be automatically detected and drivers will be loaded for it but your best bet is to install the drivers before connecting the board.

The Nucleo boards use a ST-LINK/V2-1 interface to communicate with the PC. The Windows drivers are on this page. You can also reach it from the F303RE Board page. If you want, you can also obtain some installation information on the MBED page of the F303RE Board.

The USB protocol enables a connected system to be seen as several different devices at the same time. If all goes well, after connecting the board to the PC, it will be seen with three different interfaces:

- A mass storage device (like an external hard disk)
- A serial communication COM port
- A debug port

We will use the storage device just once to transfer the firmware to the board. Most of the time we will use the COM port as this is the link for the requests sent from the Python code to the hardware board.

Most of the SLab work has been developed in a Windows machine, so no intensive checks have been performed on Linux and no test at all has been developed on MAC.

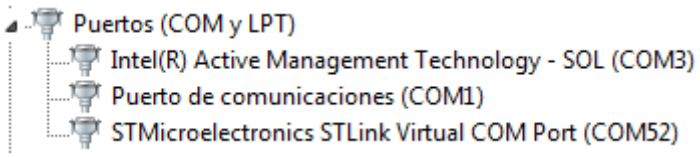
It has been tested that the drivers install ok, at least for the mass storage and COM port on Ubuntu Linux. At a minimum you shall have the COM port working as you can always reach a windows PC to do the firmware programming.

The Nucleo board includes two microcontrollers, one, a STM32F103 next to the USB connector, implements the ST-LINK/V2-1 that communicates with the PC. And the second one, the big STM32F303RE 64 pin chip at the center of the board, is where we store our firmware. So, in fact, we have two firmwares in the board, one for each microcontroller.

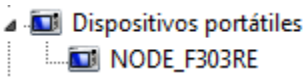
ST releases from time to time new firmwares for the STM32F103 microcontroller that takes care of the ST-LINK connection. You can find an installer program for this firmware on the F303RE board page. It is recommended to run this installer to guarantee that the STLINK firmware is up to date. Anyway, upgrading the firmware is the first step to take if you experience any communication problem with the board. As far as I know this installer only works on Windows, so you would need a Windows system to upgrade the STLINK firmware.

Uploading the SLab firmware

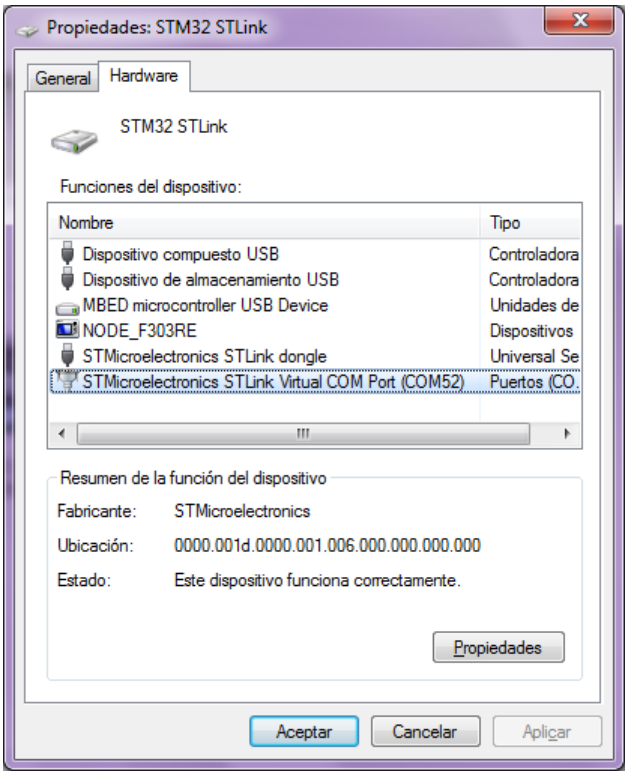
After the drivers installation, you shall see that the board is seen from the PC at least as a mass storage device, with name "NODE_F303RE" in the case of the F303RE Board, and as a COM serial link. In a Windows system you can see the COM port used in the device manager:



And also the mass storage device:



You can also see both in the STM32 STLink properties under "Devices and Printers" :



In a Linux machine, the board also is shown as a "NODE_F303RE" external storage unit and, in the machines tested, the serial is seen as **"/dev/ttyACM0"**.

Now it is time to upload the firmware on the board. For official boards, the firmware file has the name:

SLAB-Board Name-X.Y.bin

Where "Board Name" is the name of the board and X.Y are the major a minor versions of the firmware, so, version 1.0 of the F303RE Board will be:

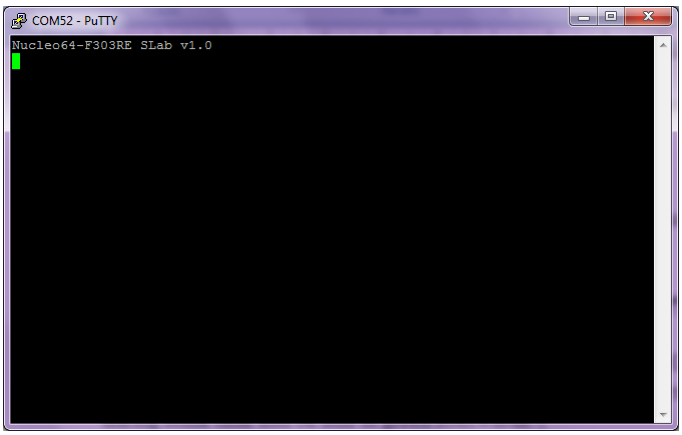
SLAB-Nucleo64-F303RE-1.0.bin

To program the firmware just drag and drop the file over the NODE_F303RE mass storage disk as if you where copying a file. You will see the board LED next to the USB connector blink during some time and be still in green afterwards.

If all goes ok, you have successfully written the firmware. This procedure has been tested both in Windows and Linux machines.

If you want to verify that the firmware is loaded and that the communication link is in operation, you can use a terminal emulator like Putty for that. This is not mandatory so you can skip to next section if you want.

If you connect to the board using its COM port at 38400 baud, 8 bits, no parity, you will see that every time you reset the board by hitting its black button, the board identifies itself with its name like in the following figure:



In a Linux machine you can use the screen program to access the serial port. Note, that you usually need to have root privileges to access the serial port. The following command has been tested to work to open the serial channel on a Linux Ubuntu machine:

```
> sudo screen /dev/ttyACM0 38400
```

That way you are sure that all the hardware operates as expected and you can confidently go to the next software section.

Setting Up the Software

As explained before, the SLab system use a slab.py module to communicate with the board. So you need Python, in particular, Python 2.7 to use the board. Moreover, the board relies on several sciPy modules, basically NumPy and matplotlib.

You can install Python from python.org and add those packages or you can go the easy way and install a bundle that incorporates the sciPy package like Anaconda that is available for Windows, Linux and Mac. Remember, to use the 2.7 installer, not the 3.x installer. It is possible to have Python 2.7 and 3.x at the same time on a system but the descriptions are beyond this document objectives.

You can also use SLab without having the SciPy packages installed in Python. In that case the module will have reduced features. No plotting, for instance. You will be informed about that when you import SLab with a message like:

```
Cannot load SciPy modules
Functionality will be reduced
```

In a proper Python installation you shall have Python 2.7 in the path so that you can call the python interpreter from any place.

When you run the Python interpreter, for instance by executing python.exe in windows, you should get, in the case of the Anaconda installation, something like:

```
Python 2.7.10 |Anaconda 2.3.0 (32-bit)| (default, May 28 2015, 17:02:00)
1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>>
```

Refer to your preferred Python documentation if you cannot get to this point.

The "`>>>`" is the prompt that ask you for commands to send.
Note that it says Python 2.7. This is the correct version of Anaconda.

Now it is time to verify that you have the needed packages. Write those lines (not the "`>>>`" characters as they are written by the interpreter):

```
>>> import numpy as np
>>> import pylab as pl
```

If the interpreter keeps silent it means that both the numpy and the pylab modules are installed, so far, so good. Now write down:

```
>>> x = np.arange(0,1,0.1)
>>> pl.plot(x,x)
```

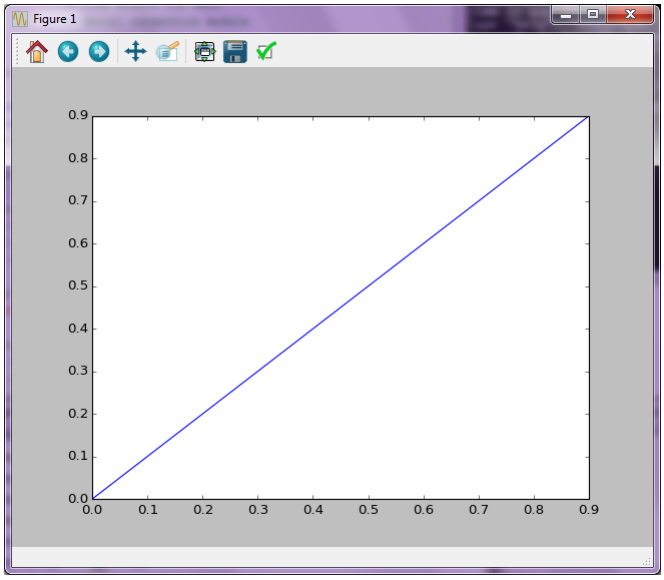
Python will respond with something like:

```
[<matplotlib.lines.Line2D object at 0x04EE6270>]
```

Now write:

```
>>> pl.show()
```

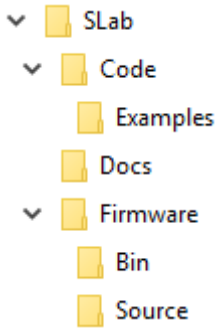
You shall get an image like the one below:



Dismiss the image and you are done.
If you have reached to this point, you should be able to use the SLab System.

First Steps on SLab

The SLab files are provided in a file ***"Slab vX.Y.zip"*** where X and Y are the major and minor versions of the release. Unzip this folder. We will refer to it as the SLab root folder. It contains several subfolders as seen in the following figure:



The **Docs** folder includes all SLab documentation including this file. The **firmware** folder includes the official SLab firmware, both in binary and source code, for the supported boards. Finally the **Code** folder is the location for the Python code so it will be our work folder. There is a ***Readme.txt*** file in the root SLab folder that describes all folder contents.

Open the ***Code*** folder, the one that contains the slab.py file, and other Python files, and call Python from this folder. In windows you can use the ***python.bat*** file present in this folder. If python.exe is not in the path, you can edit the batch file so that it points to the python.exe file wherever it is installed.

In the case of Linux, it is possible that you need to access Python as super user by using ***sudo*** in order to be able to connect with the board. To do that you can use a line like:

```
> sudo python
```

After calling the Python interpreter you should get to the Python prompt. Now write the following line and press enter. Remember that the ">>>" is written by Python, not by you.

```
>>> import slab
```

Python should respond something like that:

```
SLAB Module
Version 1.0 (22/2/2017)
Running interactively
```

If it gives any error, check that you have really executed the Python module executable from the Code folder that hosts the slab.py file.

Now, we want to connect with the board.

The slab.py module interacts with the board firmware using a serial COM port at 38400 baud speed. For the connection to work, the drivers of the board shall be installed so it is detected by the system.

In order to obtain a successful connection you usually need to know the name of the port. In windows it is usually "COMn" where n is a number and you can usually request its number on Devices and Printers on the windows menu or in the device administration control panel.
In Linux, the name can be something like "/dev/ttyACM0".
As you have seen on a previous figure in this document, the name is COM52 in my windows machine. In other operating systems knowing the port name will need a different method. You can refer to the Python pyserial documentation or the operating system information to obtain more details.

Once you have the port name of the hardware board, input the following command on the Python console, changing 'COM52' with whatever name your serial port has:

```
>>> slab.connect('COM52')
```

In Linux it could be:

```
>>> slab.connect('/dev/ttyACM0')
```

Python should respond with something like:

```
Getting board data
Connected to Nucleo64-F303RE SLab v1.0
No ADC calibration data found
No DAC calibration data found
```

If the port name is not a proper port it will respond something like the following two lines following with some more exception information:

```
** SLab exception
** Cannot open connection. Check port
```

If the port name exists on the system but it is not the board with the proper firmware loaded it will respond something like:

```
** Slab exception
** Board not responding. Check port and Firmware
```

Or like:

```
** Slab exception
** Bad magic from board. Check Firmware
```

In the first case there is not response from the com port after one second and in the second case the response from the board is wrong. The Slab board firmware contains a 4 byte magic code that guarantees that you only get a proper connection if you select a port that links to a board with the proper Slab firmware.

As you can see, the Python SLab module responds with an exception every time something goes wrong. That automatically aborts any running program. This is typical way to handle errors in Python.

If you are not able not obtain the proper name of the COM port used by the board, you can request the Slab module to autodetect the port. In order to do that, just request the connection without providing any port name:

```
>>>slab.connect()
```

That should usually work on Windows, probably on Linux and I don't know on a Mac. If the board can be detected you will see something like:

```
Board detected at port COM52
Getting board data
Connected to Nucleo64-F303RE SLab v1.0
No ADC calibration data found
No DAC calibration data found
```

You can take note of the COM port nave for successive connections but the Python SLab module will also remember this port for you. The last successfully COM port opened is stored in a “Last_COM.dat” file. Check that this file is generated in the Code folder.

Next time the board tries to auto detect the port, it will first try to use the last valid COM port like in the next example:

```
>>>slab.connect()

Trying last valid port COM52
Board detected
Getting board data
Connected to Nucleo64-F303RE SLab v1.0
No ADC calibration data found
No DAC calibration data found
```

Using auto detect is always slower than providing the proper port name but thanks to the “Last_COM.dat” file the full auto detect algorithm will only need to be carried out once. Moreover, the “Last_COM.dat” file is also stored for manually selected ports, so, except for the first time, you can always use the ***connect*** command without arguments.

The auto detect algorithm works by requesting a magic code from all connected devices. As this is an active search, it can affect the normal operation of the devices in the system. If you detect any problem, refrain from using auto detect and manually select the port at least until the "Last_COM.dat" file is generated.

If a board cannot be detected automatically you will get an exception:

```
** SLab exception
** COM Autodetect Fail
```

It usually means that there is a problem with the drivers, the STLINK firmware or the board SLab firmware. You can check using a terminal emulator on the board port at 38400 baud that the boards reports its name when you reset it, but if this works, the SLab connection will usually work too.

If the connection is established, you can disconnect from the board afterwards with the command:

```
>>> slab.disconnect()
```

Python will respond with something like:

```
Disconnected from the Board
```

As you have seen, and usual in python, we have used the slab module name for all the commands. All commands will have the structure:

```
var = slab.command_name(Arguments)
```

Where *command_name* is the name of the command and *Arguments* is an optional argument list for the command. Some commands return values. In those cases you can assign the result to a variable *var* or, in some cases, a list of variables. For commands that do not return any value, or for the cases that the command returns data you don't need, you can omit the "*var* =" part of the call. As this is standard in Python refer to the Python literature for more information about this language.

It is not recommended to remove the slab reference by using:

```
>>> from slab import *
```

It is bad Python practice and can break the operation of the program in unexpected ways. If you want to reduce the number of characters you are writing you'd better use an alias name for the module:

```
>>> import slab as sl
>>> sl.connect()
>>> sl.disconnect()
```

Before leaving Python, both in interactive mode or using a script, it is recommended to disconnect from the board issuing:

```
>>> slab.disconnect()
```

But probably Python will do that for you if you forget it.

At any time you can ask SLab for help using the ***help*** command:

```
>>> slab.help()
```

If you know a topic, like a command name for instance, you can directly ask for it. In the case of the ***connect*** command you can write:

```
>>> slab.help("connect")
```

At this point you have a nearly working SLab system. Next chapter will add some hardware to improve the performance of the hardware board.

References

STM32 Nucleo Page

Main page on the ST website about the STM32 Nucleo demonstration boards.
http://www.st.com/en/evaluation-tools/stm32-mcu-nucleo.html

MBED Developer Page

MBED developer page. This page gives access to the MBED compiler.
https://developer.mbed.org/

Python page:

Main page of the Python Software Foundation.
https://www.python.org/

Anaconda page:

Page where the Anaconda Python distribution can be downloaded.
https://www.continuum.io/downloads

SLab Python References

Those are the reference documents for the SLab Python modules. They describe the commands that can be carried out after importing each module.
They should be available in the **SLab/Doc** folder.