EZ Module Reference

This document is the reference for the "Easy" EZ sub module of the Python SLab system.

Index

Introduction

The "SLab System" is a software solution that provides a low cost access to circuit measurements by the use of low cost development boards.
The complete system has a natural tendency to feature creeping. In order to make the dimension of the project easy to manage, it has been divided in several modules.
The main module, associated to the file slab.py, contains all the code that directly accesses the Hardware Board and the most immediate low complexity commands.

The functionality of the main module is extended by the use of additional modules. This document describes the Easy EZ module, associated to the file slab_ez.py that includes a set of functions to ease the use of the SLab System. The EZ module provides the easiest way to access the SLab system. That easiness comes at a cost, however, as the commands in the EZ module are far less powerful than the ones in the other modules.

If you want to use the EZ module you don't need to import any other module as the functions of the main SLab module needed for this module to operate are duplicated in the EZ module.
There are two ways to use the EZ module. The first one is to use a normal import.

```
>>> import slab_ez as ez
```

Observe that we have used an alias name for the module namespace, so, a command with the name "command" will be accessed as "ez.command".
The second way is to import the module in the global namespace:

```
>>> from slab_ez import *
```

That way you don't need to access any namespace qualifier to access the module function. So, the commands will be accessed with their own name without any namespace qualifier. Although it is not usually recommended to use this kind of import, it can ease the use of the module. For all examples in this document we will suppose that you have imported the module using this second mode so no name space qualifier will be used.

In either case, the board will be connected in auto detect mode automatically during the import so be sure to have the board plugged before importing this module.

There is an even easier way to use the EZ module in the second flat namespace mode. The SLab\Code folder contains a **SLab_EZ.bat** windows script file. This file executes the command:

```
python.exe -i slab_ez.py
```

So, by double clicking over **SLab_EZ.bat** you open a python window, import the functions of the EZ module, connect to the board, and get a python prompt.

Management Commands

This section describes the easy commands that manage the board in general.

help(*topic*)

Arguments	topic	Requested topic (Defaults to "ez")	V1
Returns		Nothing	

Auxiliary

Gives help information about one topic. If no topic is provided, goes to the main help page of the EZ module. This is the same command of the main SLab module except that it uses the default "ez" topic instead of "root".

Example:
Get help on the EZ module

Just issue the command:

```
>>> help()
```

You will get something like:

```
-----------------------------
 EZ Submodule

 Commands common to slab.py:
    readVoltage
    setVoltage
    dcPrint
    zero

 EZ command topics:
    liveVoltage
    sweepPlot
    ioCurve
    bridgeCurve
    sineResponse
    triangleResponse
    squareResponse
    sineBridgeResponse

-----------------------------
root topic goes to main page
Just hit return to exit help

Help topic :
```

connect()

Arguments	None	V1
Returns	Nothing	

Base "M" "F" "I" "L"

Connects with a Hardware Board using auto detect.
You don't usually need to connect when using the EZ module as this module connect to the board during the import.

Autodetect works by exploring all possible COM ports, and sending a magic code request using the char 'M' until a proper response if obtained. That means that only the first board detected is found. Moreover, as the search for the port is active (sends **"M"** to all available ports) it can disrupt the operation of other devices connected to the PC.
Last valid COM port is stored in a **"Last_COM.dat"** file, so auto detect always first tries the last valid COM port.
If the board is already connected, disconnects from it and reconnects. This procedure is useful if you reset the board and want to restart the connection.
This is the same command of the main SLab module except that it always use auto detect.

Example:
Reconnect with the hardware board

Just issue the command:

```
>>> connect()
```

DC Commands

This section describes the easy commands that operate in DC.

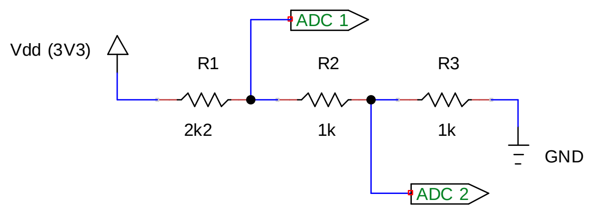
readVoltage(ch1,ch2)

Arguments	Ch1	ADC to read as (+) 1, 2, 3, 4 or 0 (GND)	V1
	Ch2	ADC to read as (-) 1, 2, 3, 4 or 0 (GND) (Defaults to GND)	
Returns		Voltage read on the ADC	

Reads the voltage of one of the four ADC inputs ADC1, ADC2, ADC3 and ADC4 expressed in volts. If two parameters are given, reads the voltage difference between the channels.
Channel number 0 is considered to be GND.
This is the same command of the main SLab module.

Example 28:
Check a voltage divider

We first implement the following circuit:



We can get the voltages at the ADC1 and ADC2 nodes.

```
>>> readVoltage(1)
1.5895649015717992
>>> readVoltage(2)
0.7929267489711933
```

We can see that ADC1 voltage doubles ADC2 voltage.
We can also check that R2 and R3 voltages are the same. R3 voltage is the value previously measured at ADC2. R2 voltage is the voltage we can measure between ADC1 and ADC2.

```
>>> readVoltage(1,2)
0.7960297635452641
```

setVoltage(channel,value)

Arguments	channel	DAC to set 1, 2 or 3*	V1
	value	Voltage between 0.0 and Vdd	
Returns		Nothing	

Set one DAC output of the board to the selected voltage.
*All SLab compatible boards feature at least 2 DACs, some could feature more DACs.
This is the same command of the main SLab module.

Example:
Set voltage at DAC 2 to 1V

```
>>> slab.setVoltage(2,1.0)
```

dcPrint()

Arguments	None	V1
Returns	Nothing	

Reads the voltage of the four ADC inputs ADC1, ADC2, ADC3 and ADC4 expressed in volts and show the results on screen.
This is the same command of the main SLab module.

Example 02:
Show the voltage at all four ADCs

```
>>> slab.dcPrint()
```

The ***dcPrint*** command will show on screen something like:

```
ADC DC Values
  ADC1 = 0.533 V
  ADC2 = 0.131 V
  ADC3 = 0.237 V
  ADC4 = 0.384 V
```

zero()

Arguments	None	V1
Returns	Nothing	

Set all DAC channels to zero.
It is recommended to only modify the circuit connections when it is not powered. That means that you shall remove the Vdd lead before performing circuit modifications. The problem is that if DACs are set to a non zero value, you could power the circuit through the DACs. That could yield important problems because active circuits don't usually like to have input voltages when they are not powered.
The ***zero*** command set all DACs to its minimum value so that you can remove the power supply without producing any hazard on the active devices on the circuit.
The ***zero*** command can also be used as a shortcut to set all DACs to zero at the same time. Note, however, that this command doesn't use the DAC calibration tables, so you will get zero at the DAC outputs of the board. This is not always the same value at the buffer outputs.
This is the same command of the main SLab module.

Example:
Perform a secure modification of the circuit
First we issue the zero command

```
>>> slab.zero()
```

Then we can remove the Vdd lead that powers the circuit and the ADCs and DACs buffers.

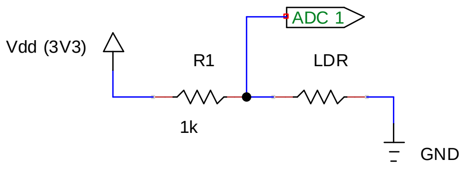
liveVoltage()

Arguments	None	V1
Returns	Nothing	

This command shows the live values of the four ADCs.
It writes on screen the values of the ADCs waits 0.2 seconds and repeat the readings.
Use CTRL+C to exit the command.

Example 29:
Check a LDR operation

We start building the following circuit that includes a Light Dependent Resistor (LDR).



Then we issue the ***liveVoltage*** command:

```
>>> liveVoltage()
```

We will see the voltage at all four ADCs and we can check that the ADC1 voltage changes when we change the light that reaches the LDR.
Use CTRL+C to exit the command.

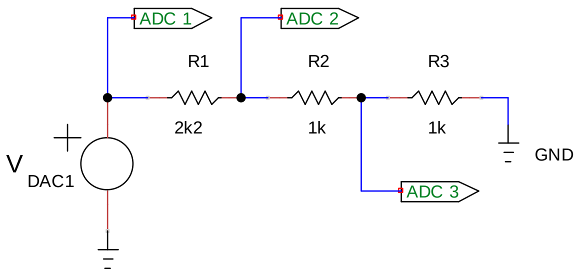
sweepPlot(v1, v2, vi)

Arguments	v1	Start voltage (Defaults to 0 V)	V1
	v2	End voltage (Defaults to 3 V)	
	vi	Voltage step (Defaults to 0.1 V)	
Returns		Nothing. Draws a plot	

Performs a DC sweep plot.
Measures all four ADCs for each DAC 1 value between v1 and v2 in vi steps.
After the measurements, obtained data is plotted.

Example 30:
Check the response of a voltage divider

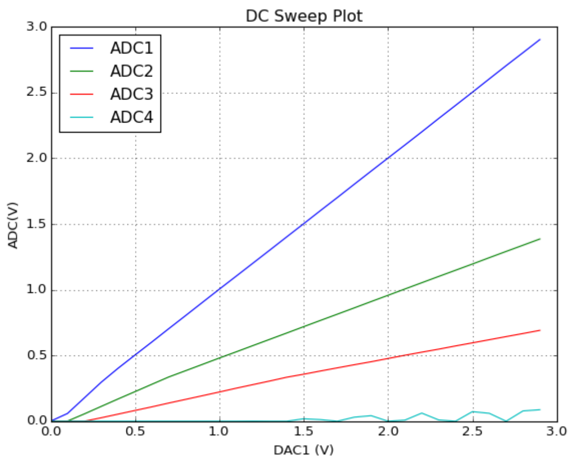
We start building the following circuit that includes a three series resistors connected to DAC1.



Then we can obtain the DC response when we sweep the DAC1 value.

```
>>> sweepPlot()
```

That will generate the following drawing:



As ADC4 is not connected, its curve has no meaning.
Note that the DAC is not able to operate at very low voltages.

ioCurve()

Arguments	None	V1
Returns	Nothing	

Obtains the Input to Output DC curve of a circuit.
ADC1 and DAC1 must be connected to the input and ADC2 to the output. Both signals are ground referenced.
DAC1 will sweep values between 0 V and 3 V.

bridgeCurve()

Arguments	None	V1
Returns	Nothing	

Obtains the Input to Output DC curve of a circuit in bridge mode.
The input positive terminal must be connected to DAC1 and ADC1, the input negative terminals must be connected to DAC2 and ADC2.
The output will be measured between ADC3 (positive) and ADC4 (negative).
DAC1 and 2 will sweep voltages between 01 V and 3.1 V for a total bridge range of ±3 V.

trendPlot(n)

Arguments	n	Number of ADCs to show (Defaults to 4)	V1
Returns		Nothing. Draws a plot	

Shows the evolution of the ADCs with time.
If the optional n value is provided, it indicates the number of ADCs to show.
In order to end the command, close the graph window. A new static graph window will all the data will be shown. Close also this graph to end the command.

Transient Commands

This section describes the easy commands that operate in time dependent transients.

sineResponse(vmin, vmax, f)

Arguments	vmin	Minimum voltage (Defaults to 1 V)	V1
	vmax	Maximum voltage (Defaults to 2 V)	
	f	Frequency (Defaults to 100 Hz)	
Returns		Nothing. Draws a plot	

Generates five sine waves on DAC 1 and measures all four ADC channels. Afterwards, the measurements are shown in a plot.
Using the optional arguments you can set the minimum, maximum and frequency of the wave.
If the minimum is above the maximum, phase is reversed.
As the wave always has 100 sample points, the maximum frequency is limited to150 Hz.

triangleResponse(vmin, vmax, f)

Arguments	v1	Minimum voltage (Defaults to 1 V)	V1
	v2	Maximum voltage (Defaults to 2 V)	
	f	Frequency (Defaults to 100 Hz)	
Returns		Nothing. Draws a plot	

Generates five triangle waves on DAC 1 and measures all four ADC channels. Afterwards, the measurements are shown in a plot.
Using the optional arguments you can set the minimum, maximum and frequency of the wave. If the minimum is above the maximum, phase is reversed.
As the wave always has 100 sample points, the maximum frequency is limited to150 Hz.

squareResponse(v1, v2, f)

Arguments	v1	Start value (Defaults to 1 V)	V1
	v2	Value after first edge (Defaults to 2 V)	
	f	Frequency (Defaults to 100 Hz)	
Returns		Nothing. Draws a plot	

Generates five square waves on DAC 1 and measures all four ADC channels. Afterwards, the measurements are shown in a plot.
Using the optional arguments you can set the voltage levels of the wave.
As the wave always has 100 sample points, the maximum frequency is limited to150 Hz.

sineBridgeResponse(vmax, f)

Arguments	vmax	Maximum voltage (Defaults to 3 V)	V1
	f	Frequency (Defaults to 100 Hz)	
Returns		Nothing. Draws a plot	

Generates five sine waves in bridge mode between DAC 1 and measures all four ADC channels with Vi between ADC 1 and ADC2 and Vo between ADC3 and ADC4. Afterwards, the measurements are shown in a plot.
If you use reactive components, you must guarantee that ADC voltages are inside of the supply limits for the board.
Using the optional arguments you can set maximum at +vmax, the minimum at -vmax and the frequency.
As the wave always has 100 sample points, the maximum frequency is limited to150 Hz.

References

STM32 Nucleo Page
http://www.st.com/en/evaluation-tools/stm32-mcu-nucleo.html

MBED Developer Page
https://developer.mbed.org/

Python page:
https://www.python.org/

Anaconda download page:
https://www.continuum.io/downloads

TinyCad
https://sourceforge.net/projects/tinycad/
Circuit images on this document have been drawn using the free software TinyCad