	# Basics 05 : The SLab Workflow

This document describes the basic workflow in the SLab experiments.	
BOM	2x 1 kΩ and 1x 2,2 kΩ resistors

Index

Introduction

This is the last document in the SLab Basics series. Here, the basic workflow in the SLab experiments will be described together with some common semantics.
The SLab project is about learning electronics by building circuits and performing measurements on them. Each document will pivot around a particular concept and will include several numbered tasks starting on "1".

There are two kinds of tasks: Theoretical and Practical. Theoretical tasks will be marked in blue as in the example task "1" below and require you to work without interacting with any circuit. That could be, for instance, perform a calculation or do a data search like in the example below:

	1	Obtain the datasheet of the MCP6002 dual operational amplifier chip. Locate and take note of its Slew Rate specification.

Practical tasks will be marked in red as in the example task "2" below and require you to interact with an electronic circuit. That can be building a circuit, setting some input voltages or performing a measurement like in the example below:

	1	Mount the proposed circuit. Measure Vo with ADC 1. ¿Is it the expected value?

Practical tasks are usually more fun than theoretical ones but the real understanding comes from using your head. You have to know what to expect on an experiment if you want to learn from it.

This document will develop as a set of simple experiments. That will introduce you to the semantics and how to interact with an electronic circuit using the SLab environment.

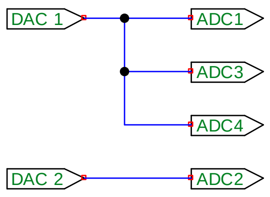
Checking the Setup

In order to perform the next experiments you need to have a working SLab environment. That includes:

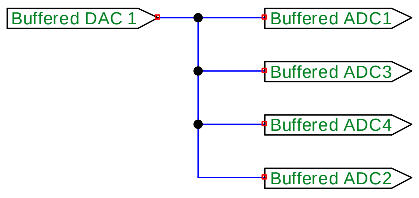
- A hardware board, like the STM32 Nucleo64 F303RE, with the SLab firmware loaded and connected to a PC.
- A breadboard with the ADCs and DACs buffers.
- Jumper wires for all eight connections between the hardware board and the breadboard: Vdd, GND, 4 unbuffered ADCs and 2 unbuffered DACs.
- A working Python enviroment that includes SciPy.
- The SLab root directory and its subdirectories.

🛠	**1**	Mount the SLab setup. Check that the red LED is active.

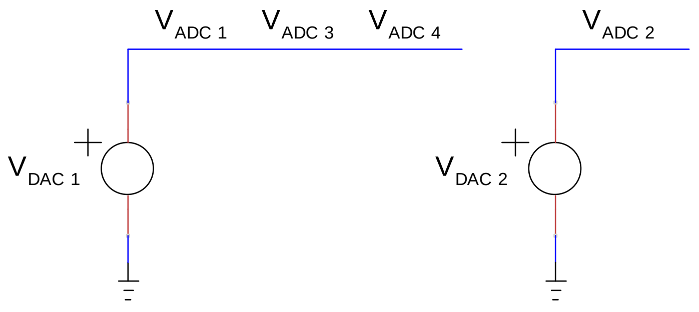
The first action we will perform is to check the board calibration. We will connect the DAC outputs to the ADC inputs as shown in the figure.



In our experiments we will always use the buffered ADC and DAC connections. We will only use the unbuffered lines to connect the hardware board with the breadboard buffers. That way, the above schematic is equivalent to the one on the figure below.

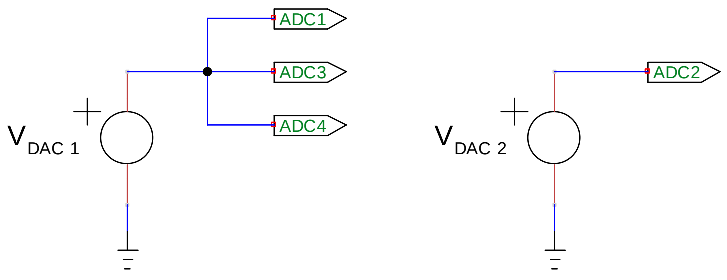


The above schematic defines the DACs as inputs and the ADCs as outputs. As DACs force voltages on the circuits, they can be considered as voltage sources. As ADCs measure voltages respect to the common ground, they can be indicated by a V_{ADC} measurement text over a node. That means that the above schematic can also be written as:



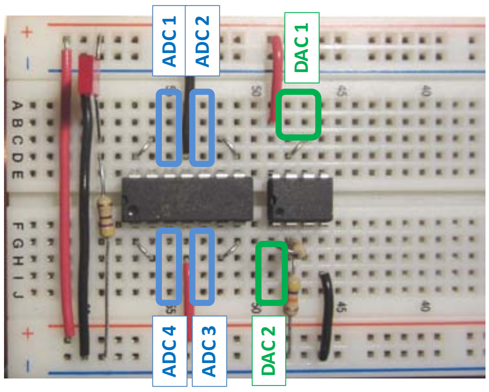
Note the ground symbol. All connections to the ground symbol should go to the negative blue GND rail at either side of the breadboard. We can always consider that all ground symbols correspond to the same node so they have the same voltage.

We can also use a mixed combination drawing the DACs as voltage sources and the ADCs as outputs like in the figure below:



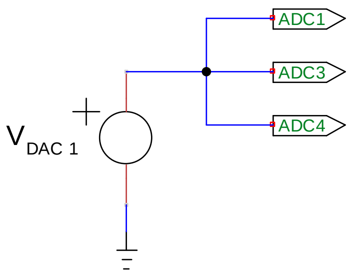
All the above schematics mean the same and are interchangeable.

If you have used the proposed breadboard buffer configuration, the buffered ADC and DAC lines are in the location indicated in the figure below. This figure won't be shown again in other SLab documents.

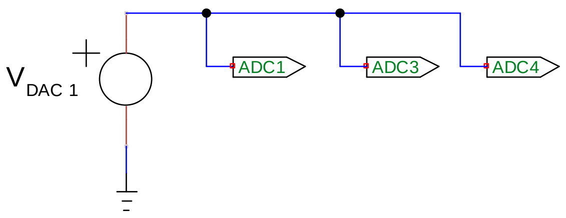


A **node** is the combination of all points that are connected together with conductors like the jumper wires. So the circuit in the previous schematic has only three nodes: GND, the node that joins DAC 1, ADC 1, ADC 3 and ADC 4 and the node that joins ADC 2 and DAC 2.

A circuit operation is independent on how you wire the nodes. The DAC 1 node can be implemented connecting three jumper wires from the pins next to the DAC 1 output to all three ADCs:



Or you can use a chain: Connect DAC 1 output to ADC 1, ADC 1 with ADC 2 and ADCA 2 with ADC 3.



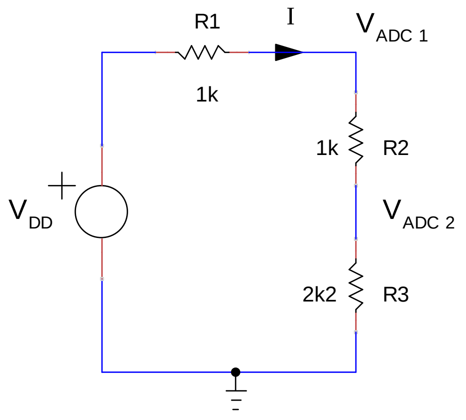
The circuit operation is the same because we conserve its topology. In general we won't care on the differences. If we care in any experiment we will make it explicit.

🛠	2	Implement the proposed DAC to ADC connections. Execute the ***Calibrate4.py*** script on the SLab/Code folder. Check that the four lines feature the same X and Y values.

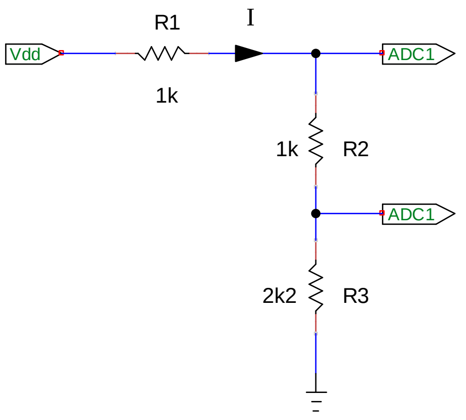
Don't continue if the calibration result is wrong. Return to the previous calibration document if needed.

A simple resistor chain

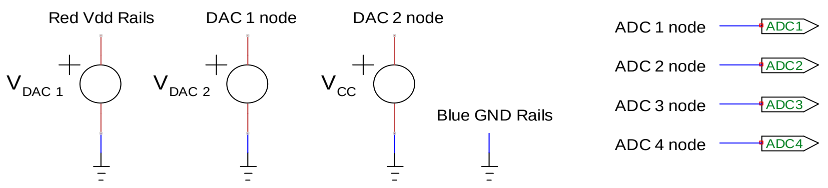
After checking the calibration, we will start measuring a simple three resistor chain circuit. The circuit includes five elements: A Vdd voltage source, three resistors R1, R2 and R3 and the ground connection. The circuit also indicates that there is a current I flowing through the supply and all three resistors.



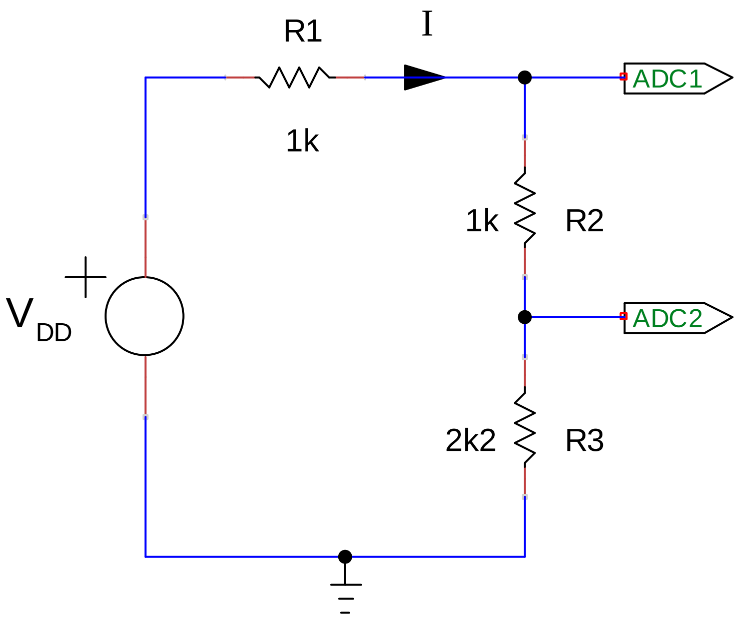
The circuit could also be drawn as shown in the next figure.



In general we prefer the first drawing because it makes evident that the current flows through Vdd. In the second figure it is not evident how the current returns to Vdd.
We can usually assume that no current enters the ADC buffered inputs so the ADCs are not considered as components but as voltage measurement points. Vdd and the DACs, however can supply or drain current so they behave like voltage source components. To make that explicit, the preferred symbol for Vdd and the DACs will be that of a voltage source. That gives us the preferred symbols we will use on our circuits for any connection related to the hardware board:



Observe that all Vdd, DAC 1 and DAC 2 sources refer to GND so they shall always have their negative terminal connected to the GND node. With those symbols, our resistor chain will be drawn as:



This is a good schematic for the circuit we see that there is an explicit path for all currents. As no current enters the ADCs, there is only one current I that flows through V_{DD}, R1, R2 and R3. Note that no current flows inside the ground terminal. It is only used for reference.

🛠	3	Implement the proposed circuit in the breadboard. You should only need the three resistors and two jumper wires.

Now it's time to compare theory and practice.

	4	Analyze the circuit and obtain the current I and the voltages at resistors R2 and R3. Consider that Vdd has a 3,3V value. If you use a board different from the F303RE board, use its supply voltage instead.

We will use a Python interactive window called from the ***SLab/Code*** directory to request the measurements.
We want to first measure the voltage on R3. As R3 is connected between the ADC 2 node and ground, we can issue the following commands. The first line imports the SLab module, the second line connects to the board and the third one reads and print the voltage.

```
>>> import slab
>>> slab.connect()
>>> slab.readVoltage(2)
```

Al your calculations depend on the Vdd value (3.3V on the F303RE board). If you want to have more exact values in your calculations you can request the board calibrated Vdd value:

```
>>> slab.vdd
```

	5	Perform the requested measurements. Do they coincide with the theoretical values? Use the calibrated Vdd value if needed.

We now want to check the voltage at R2. As neither terminal of this resistance is grounded, we need to calculate its voltage as the difference between the voltages of the ADC 1 and ADC 2.

$$V_{R2} = V_{ADC\ 1} - V_{ADC\ 2}$$

There are many ways to skin this cat.
We can obtain both voltages and perform the subtraction ourselves:

```
>>> slab.readVoltage(1)
>>> slab.readVoltage(2)
```

We can be lazy and show all four ADC values and look only after ADC 1 and ADC 2:

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

We can let Python do the calculations for us:

```
>>> slab.readVoltage(1) - slab.readVoltage(2)
```

We can also ask for the voltage difference between ADC 1 and ADC 2:

```
>>> slab.readVoltage(1,2)
```

6 Obtain V_{R2} in all proposed ways.
Check they all give the same expected value.

To end the circuit measurements we want to obtain the I current. The SLab system does not include any current measurement element, so we will need to work out the current from a voltage measurement.

We know that the current is the same on all three resistors as ADC input currents are zero. So, we can compute the current from the voltage in R3.

```
>>> slab.readVoltage(2) / 2200
```

You can also use check the current at R2:

```
>>> slab.readVoltage(1,2) / 1000
```

The SLab module also includes a command to calculate Ohm's law for you. You supply the resistor value and the two ADC numbers connected to a resistor to nodes (+) and (-) and it gives you the current:

```
>>> slab.rCurrent(1000,1,2)
```

You can also calculate the current at R3, as this resistor is grounded, just don’t provide the second node:

```
>>> slab.rCurrent(2200,2)
```

Finally you can calculate the current at R1:

```
>>> (slab.vdd – slab.readVoltage(1)) / 1000
```

Perhaps the above commands are too long for you. In that case you can create a short alias version of the ***readVoltage*** command that is just one letter like “***v***”.

```
>>> from slab import readVoltage as v
```

As this is an explicit import of a function, you don't need to use the ***slab*** namespace.

```
>>> v(2)
```

Calculation of the current at R2 will be:

```
>>> v(1,2) / 1000
```

You can create alias for any other command:

```
>>> from slab import dcPrint as dcp
>>> dcp()
```

🛠	7	Perform all the proposed measurements and compare the current with the expected value. Try also using alias for the commands. Some deviation is expected from theoretical values as resistors have some tolerance on its values. If you have a multimeter, you can check the real value of the resistors to obtain better data for your calculations.

What if something goes wrong?

Sometimes the system won't work as expected. In can be due to user errors, electrical glitches or bugs in the board firmware or the SLab Python code. If the system is unresponsive or you cannot get back the Python prompt, this section will give some options to try.

Problems at the Python level

Sometimes things go wrong on the Python SLab software layer. If you cannot get back the prompt, try these options:

- Check if there is a drawing generated by Python. When Python generates a graph, it shifts the focus to this graph so you can interact with it. You won't recover the prompt until you close the graph.

- Use CTRL+C to stop the Python script. This should be enough to stop any problem associated to the SLab Python code. Most times you could continue working after recovering the control but, in the worst cases, perhaps you need to reconnect to the board.

Let’s try to lock SLab generating a code that takes too much time to complete. Disconnect any circuit from the board and execute the following command. Don’t worry, you don’t need to understand it.

```
>>> slab.dcSweep(1,1,2,0.1,100)
```

SLab will write:

```
Performing mesurements...
```

But you won’t recover the prompt until about 16 minutes. This is because the previous command instructs SLab to obtain several DC readings taken 100 seconds apart from each other. As the wait is generated inside the SLab Python code, you can interrupt the command issuing CTRL+C. Try it.
You should recover the prompt with a message like:

```
KeyboardInterrupt
>>>
```

Sometimes, more text is printed, but you will recover the prompt.

Problems at the board firmware level

Things can also go wrong at the board firmware level too. If CTRL+C don’t work, that usually means that the problem lies on the board. Things that you can try in this case:

- Push the **HALT** button. The HALT button, the blue user one on the 303RE board, instructs the firmware to stop what it is doing, send an abort error code, and wait for new commands. That is mostly useful against user errors like, for instance, asking to wait for a condition that never takes place or asking for a too long transient measurement. Usually you can continue with what you were doing before aborting the current command.

- Push the **RESET** button. The RESET button, the black one on the 303RE board, restarts the firmware on the board. It is similar to unplug and plug again the board. That means any configuration will be lost and you would need to reconnect to the board.

- Connect and disconnect the board. Sometimes things can go really wrong. In those cases a **RESET** won’t do. Disconnect the board from the computer and connect again. Sometimes the problem can be related to the circuit under test, so, is best to disconnect any circuit from the board before reconnecting.

Let’s try the HALT button.

Execute the following code:

```
>>> slab.transientTriggered(1)
```

SLab will respond with something like:

```
Performing transient triggered measurement...
```

But you won’t recover the prompt because this command instruct the board to wait for a condition that never takes place.
To recover the board, just push the blue **HALT** button.
You will get something like:

```
** SLab exception
** Halt from board
```

And some trace text but you will recover the prompt.

Commands learned so far

Before this document we knew about ***connect*** and ***disconnect*** commands. In this document we have learned about other commands:

- ***readVoltage***
- ***dcPrint***
- ***rCurrent***

We have also learned about the internal ***vdd*** variable that holds the calibrated supply voltage value and some ways to recover from a unresponsive SLab system.

You can find more information about the commands in the SLab Python reference document located in the ***SLab/Docs*** folder.

References

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.

TinyCad

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

SciPy

All the functions plots have been generated using the Matplotlib SciPy package.
https://www.scipy.org/