Lab 0x00 - Serial Communication and ADC Reading

This assignment is to be completed in lab groups of two. Your team will use an analog to digital converter, commonly referred to as an ADC, to measure the capacitor voltage in an RC circuit. ADC data will be collected at a frequency of 1000 Hz while a step change is applied to the input of the RC circuit. You will then generate a step response plot showing the ADC voltage plotted against time. Finally you will determine a time constant based on the step response data and then compare this value to one calculated from the component values in your RC circuit.

Familiarization

Before you begin the assignment, familiarize yourself with the hardware by connecting your Nucleo board and running some commands from the REPL[1]. The following instructions are written assuming you are using a PC running Microsoft Windows. If you are using a different platform it will be your responsibility to adapt the instructions to your own platform.

1. Connect your Nucleo board to a PC using a USB Mini-B cable; be sure to plug into the USB port on the "Shoe of Brian", not the one on the Nucleo itself. You can run code on the Nucleo board using a serial monitor like PuTTY; the following steps will assume you are using PuTTY specifically on a Windows PC.

   (a) With the Shoe of Brian connected to a PC through USB, navigate to "Device Manager" and browse to "Ports (COM & LPT)". In the drop-down look for a device listed as "USB Serial Device (COMxx)" and make note of the port number, probably COM3 or COM4.

   (b) Open up PuTTY and navigate to the serial connection settings. Configure the serial monitor with the following settings:

Serial line to connect to	COMxx
Baudrate	115200
Data Bits	8
Stop Bits	1
Parity	None
Flow control	None

   (c) Navigate back to the "Session" category in PuTTY and change the "Connection type" to "Serial". To avoid having to reconfigure these settings in the future, highlight the saved session called "Default Settings" and click "Save" to store the COM port settings.

   (d) Click on the "Open" button to connect to the hardware and open the REPL. Confirm your connection is working properly by pressing Ctrl-C and then Ctrl-D to first cancel any running code and then soft-reset the device. You should be presented with the MicroPython REPL, indicated by the three right-facing carets, `>>>`. You can also learn a little about MicroPython and the REPL by running the `help()` function.

[1] The REPL or read-evaluate-print-loop is what we use as a commandline to interact with MicroPython.

2. Start by interacting with some of the basic features built into the Nucleo board, like the User LED. Try creating a pin object associated with pin PA5.

```
from pyb import Pin

PA5 = Pin(Pin.cpu.A5, mode=Pin.OUT_PP)
```

   You can then turn the LED on and off using the state of PA5.

```
PA5.high() # Turn LED on
PA5.low()  # Turn LED off
```

3. The pin, PA5, associated with the LED is one of the pins associated with the timer peripherals on the STM32 microcontroller; specifically PA5 is connected to Timer 2 Channel 1. Information about which peripherals connect to which pins can be found in the datasheet for the microcontroller.

   Search online for "STM32L476 Datasheet" and look for a PDF document provided by ST Microelectronics. Be careful that you find the datasheet for that precise part number or the information will likely not match the lab hardware. You should then skim through the datasheet until you find something called the "Alternate Function" table. This is a matrix showing what features each pin is capable of on the STM32L476. This particular document (and table) will be one of the most common references for you throughout the term, so it may be smart for you to bookmark or otherwise retain access to the document.

   Try adjusting the brightness of the LED by creating a timer object, and from that a channel object. The code below shows how to set up a timer channel to toggle the LED pin at 1Hz; is that fast enough for human eyes to perceive a reduction in brightness? Experiment with the timer frequency until you determine a minimum frequency that looks to the eye as a dimmed LED instead of one that blinks noticeably. Once you figure out the right frequency you can adjust the pulse-width-percent, also referred to as the duty cycle, of the timer to adjust the apparent brightness of the LED.

```
from pyb import Timer, Pin
tim2 = Timer(2, freq=1)
t2ch1 = tim2.channel(1, pin=Pin.cpu.A5, mode=Timer.PWM, pulse_width_percent=50)
```

4. Now try quickly interfacing with the User button on the Nucleo through pin PC13. First reset the Nucleo by using "Ctrl-D" at the REPL. Then you can set up a simple callback function as follows:

```
from pyb import ExtInt, Pin

PA5 = Pin(Pin.cpu.A5, mode=Pin.OUT_PP)

button_int = ExtInt(Pin.cpu.C13, ExtInt.IRQ_FALLING,
                    Pin.PULL_NONE, lambda p: PA5.value(0 if PA5.value() else 1))
```

You should now be able to toggle the LED state by pressing the blue user button on your Nucleo board. The preceding code uses what is called a "Lambda Function" also called an anonymous function in other programming languages. This lambda function is just a snippet of code that is a very concise way of writing a short function.

```
lambda p: PA5.value(0 if PA5.value() else 1)
```

The preceding snippet defines a function that takes in a parameter representing a pin object and then depending on the state of the pin, it will set the LED pin to the opposite value. Why may it be necessary to flip the logic between the button and LED?

It may be more clear to read through the function below which accomplishes the same thing as the more concise lambda function.

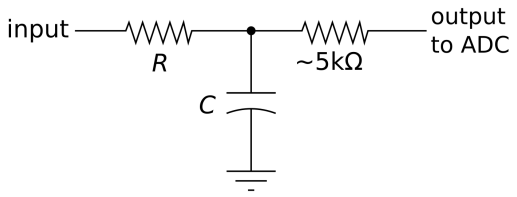
```
def button_LED_toggle(the_pin):
    if the_pin.value():
        PA5.value(0) # or PA5.low()
    else:
        PA5.value(1) # or PA5.high()
```

Assignment

Before you begin the assignment, read through all of the steps below. The steps outline the general process needed to complete the lab, but with many gaps you will have to fill in on your own.

It is **strongly** recommended that you build your program incrementally, slowly adding features and lines of code as you build and test the functionality. If you attempt to write the finished program line-by-line top-to-bottom you will have a much harder timer than building functionality in small pieces that can then be assembled into a working program.

1. Disconnect all power (the USB Cable) from your Nucleo board.
2. Build the following RC circuit using components found in your lab kit.



   You should find a capacitor in the range of 2 to 10 μF and a 100 $K\Omega$ resistor. These parts *should* be in good condition, but you may need to find replacement resistors from one of the blue cabinets in lab. Aim for an RC time constant near 0.2 or 0.3 seconds. The additional 5 $K\Omega$ resistor is to protect the Nucleo in case your circuit is connected incorrectly, but in normal operation should have no effect on the ADC readings.

3. Connect the input to the RC circuit to your Nucleo so that you can trigger a step response by changing the state of a digital pin on the Nucleo. Use pin `PC1` as your digital output, connected to the input of your RC circuit. The inside lid of each kit has a figure showing the pin names as available through the top connectors on the Nucleo. You can find the same figures here: https://os.mbed.com/platforms/ST-Nucleo-L476RG/.

   Always reference pins by their MCU pin name in the form "Pxy" where 'x' is the port, and 'y' is the pin. For example, `PC1` is pin 1 on port C.

4. Connect the output from the RC circuit to your Nucleo so that you can measure the capacitor voltage using the built in ADC. Use pin `PC0` for your ADC connection. Why does the 5 $K\Omega$ resistor have little to no effect on the measurement?

5. Reconnect your USB cable and use the Python REPL in PuTTY to read the ADC values manually and confirm that you are able to measure the voltage on the RC circuit correctly. You can make an ADC object using the `pyb.ADC` class. Refer to the ADC documenation on the MicroPython website: https://docs.micropython.org/en/latest/library/pyb.ADC.html.

6. Set up a timer to trigger callbacks at a rate of 1 KHz using an object of the `pyb.Timer` class. Timer 6 or 7 would be a good choice. Write your callback function so that it reads a single value from the ADC and adds it to an array of voltages. Since this function will be more complex than appropriate for a Lambda function, you will need to write a standard Python function. Start with the following skeleton below.

```
def tim_cb(tim):
```

```
    global data, idx
    # Code goes here
```

This skeleton sets up a function that takes in a single input argument representing a timer object. The function must be defined with this input parameter because when the timer callback is triggered the timer object is automatically passed into the callback function. The `global` keyword indicates that the variables `data` and `idx` are used both within and outside the function definition.

To set up the callback to run at a regular interval, first create a timer object and then assign to it the callback function you want to run. For example, to run the preceding function at 100Hz with Timer 7, you could use the following code shown below.

```
tim7 = Timer(7, freq=100) # create the timer object
tim7.callback(tim_cb)     # assign the callback
```

You can also turn off a callback using a similar line of code as shown below.

```
tim7.callback(None)      # disable the callback
```

7. Write code that enables the callback, toggles the input to the RC circuit, and collects data over a long enough time window to fully capture your step response. This should be roughly 4 or 5 times your estimated time constant.

   There are a variety of ways to store data in Python, but for this assignment it will be best to work with a numerical array; read about them here: https://docs.python.org/3/library/array.html. Use some code like the example below to create an initialize an array.

   ```
   from array import array
   data = array('H', 1000*[0])
   ```

   The array initializer requires two inputs, first, the type code, selected as `'H'` in this example, and an initial value for the array, selected as a Python List made up of 1000 zeros. Why is `'H'` the best choice of data type for this assignment?

   You can access or assign data to array elements using standard indexing with square brackets. For example, the following code snippet will assign the value 42 to the element of the array called `data` at position `idx` (counting from zero). Make sure that you aren't indexing out of bounds!

   ```
   data[idx] = 42
   ```

   If you want your program to simply wait for a period of time while the step response is running you can use one of the sleep commands. For example, the following code sleeps for 1500ms. For most assignments this quarter we will avoid using any sort of blocking code, like sleep or delay commands; however, for the scope of this initial assignment you can write blocking code.

   ```
   from time import sleep_ms
   sleep_ms(1500)
   ```

8. Once the data collection has been completed, print the data to the Python REPL in a comma separated format with time in the left column and ADC reading in the right column. For the sake of this assignment you may assume that the timer runs at an exact interval matching the selected timer frequency. In future labs you will timestamp data as it is collected.

To print data in a comma separated format you can simply loop through each index of the data and then print a formatted string. For example, the following code will print a single row of comma separated data to PuTTY. Formatted strings, like the f-string used in the example, will be very useful throughout the quarter.

```
print(f"{idx}, {data[idx]}")
```

When you print the data make sure to convert the numerical values into appropriate units to use for your plot. That is, convert values to seconds or milliseconds and volts before you send the data to your PuTTY window.

You can copy and paste the comma separated values into a text editor and then save the file with a name ending in ".CSV". The CSV file can then be used with programs like Excel, MATLAB, or with a Python script running on a PC to produce the deliverables described below.

Note: to copy in PuTTY all you need to do is highlight text, do not use Ctrl-C, as that keystroke is used to cancel code running on the Nucleo. Similarly, if you wish to paste into PuTTY you simply need to right click inside the terminal window and the text from your clipboard will be pasted into PuTTY.

Requirements and Deliverables

Once completed you should be able to trigger a step response automatically by pressing the blue user button on your Nucleo board or by interacting with your Nucleo through PuTTY. Once you've collected your data and imported the data into another tool on your PC, complete the following tasks listed below.

- Create a figure with the step response plotted with voltage on the vertical axis and time on the horizontal axis.
- Create a second figure with the step response plotted on a log scale so that the response appears as a straight line.
- Determine the slope of the line from your second figure and use that to calculate the time constant associated with the RC circuit. Annotate the plot with time constant.
- Compare the value of your time constant, as calculated from your log plot, to your estimated time constant, as calculated from the resistor and capacitor values used in your circuit. Is the difference in these values commensurate with the tolerances of the resistor and capacitor used to build the circuit?

You will submit a brief memo describing your setup and program flow along with the required plots and some commentary on your results. Include your Python source code as attachments to the memo. You must use a standard memo format for your submission; if you are unfamiliar refresh yourself on proper memo formatting before submission. The memo will be submitted through Canvas as a single PDF document once per group.

Your instructor may also ask you to demonstrate the functionality of your program at the beginning of class the week after this lab was assigned.