**Lab-2: First-order Transient Response**

In this lab we will be looking at the Transient Response of First-order circuits. Objectives are:

1- Understand the transient response of RC circuits.

2- How can such elements act bring non-idealities to circuit design. (Propagation Delay)

**Lab report structure:** please submit a document or a pdf file. Follow the structure of the lab itself. Respond to each step in the design and the experimental sections, one by one.

**Design**

**Step.1:** Consider an RC circuit as follows. Derive an expression for Vout as a function of an input step wave that goes from 0 to 5 volts. Consider both charging and discharging cases (0 -> 5 v step and 5 v -> 0 step).



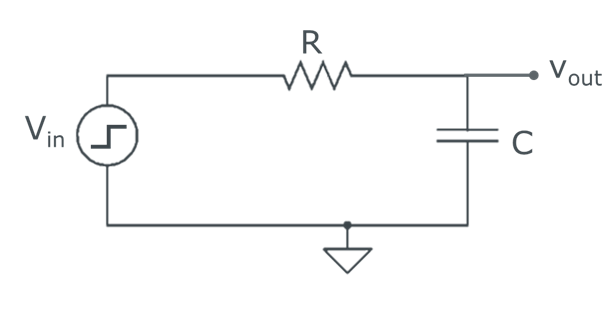
****

Figure. 1a. RC series circuit.

**Step.2:** Since our scope’s sampling interval is 1ms. Considering a fixed R = 1kΩ, find upper or lower bounds for C, for which the transient response of the system will be detectable by our scope (a good case can be made just by comparing the time constant with the sampling rate).

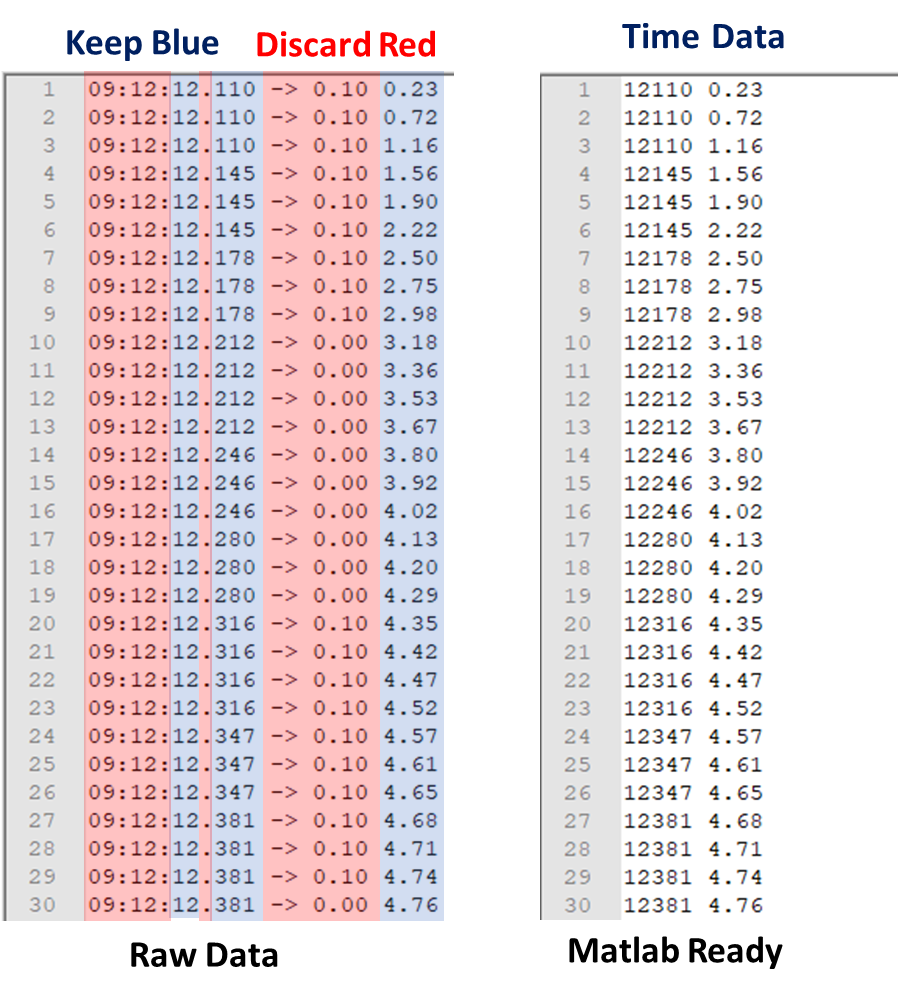
**Experiment**

**Step.3: Extracting time-constant of an RC series circuit**

Setup the configuration shown in Fig.1a, (R = 1kΩ, C= [10uf, 47uf ,100uF]), on your breadboard. Revisit lab.1 to connect and configure your Arduino to send a 1hz square wave as input (Vin) and scope the output using the Arduino’s oscilloscope. Calculate the time-constant of the transient response (time it takes for the output to reach 63% of its final value). Compare this with the theoretical value. Is it what you expected? Why do you think it might differ? (Suggestion: Set the sampling rate on your scope to 100 to get a better view of the signal, however, keep it on 0.1 when extracting data for the actual measurments)

**[Side Note 1]:** Tips: The time-scale of your x-axis in the serial plotter depends on the value of your sample interval, but usually for the suggested minimum sampling interval, your x-axis has a unit of ~1 millisecond. For larger values of your sample interval (~100), your x-axis timescale will change (~10ms). You can always confirm your timescale by generating a 1Hz signal (1s period) and probing it with your scope. I want to emphasize that your x-axis on serial-plotter is not very accurate! It’s always preferred to get the actual data points from Tools 🡪 Serial monitor, make sure to enable the timestamps view so you can have your x-axis. Copy the data array to Matlab and use that for accurate analysis. You can also use [Engauge Digitizer](http://markummitchell.github.io/engauge-digitizer/)  to extrapolate the data directly from an snapshot of the wave, but as discussed, this is not very precise)

**[Side Note 2]:** To make the data in the serial monitor easily compatible with Matlab, I suggest you do the following. Download and install [Notepad++](https://notepad-plus-plus.org/) ( a great free source replacement for notepad editor for people who code). From serial monitor enable timestamps, wait a little while so data from enough oscillations are collected. Disable auto-scroll and copy the data to a notepad++ blank page. Your data should look similar to left handside of the figure below (ofcourse without my false coloring). Now we need to discard the parts that are shown in red. This changes the timestamps to unit of miliseconds, which can now be used as your x-axis and also discards the sampling signal (0.10,0.00 datapoints) markers. To do so, it will be much faster to choose individual columns by holding alt+ click-drag selecting the column and then simply deleting them. Now you should have your Matlab compatible data points. To plot them, open Matlab, copy the data and then input the following the command window:

**Sidenote Figure.** Left shows the raw data and right shows our targeted format.

a = [ *copy your data by ctrl + v* ]

plot(a(:,1),a(:,2))

which well then plot your data points on the second column as a function of your data points. Also you can use other methods such as saving the Matlab ready file and importing it using import data tab in Matlab. Or preparing a script like:

clc

clear

a = [ *copy your data by ctrl + v* ]

plot(a(:,1),a(:,2))

in Matlab and then copying your data to script and then running it. Please make sure to add the figure axis names and appropriate units to your figure. Also make sure to collect and plot enough oscillations to ensure your data visualization looks good and supports your conclusions.

**[Side Note 3]**: Although you do not need to know about capacitors in detail, you need to know enough information to build the circuit in step 3. There are many types of capacitors including polarized and non-polarized types (samples of each capacitor and their schematic symbols are shown in the figure above). For the polarized capacitors, you need to connect the positive/negative terminal to the higher/lower voltage whereas, for the non-polarized ones, the order of terminals does not matter. In step 3, you will use a polarized capacitor, so you need to connect its terminals to the correct polarity, meaning the positive terminal to the power and the negative terminal to the ground (GND). For a polarized capacitor, you can find the positive terminal either by selecting the longer terminal or looking at the "+" mark above it.



Side-note Figure. A sample of a polarized capacitor and its schematic symbol (left). A sample of a non-polarized capacitor and its schematic symbol (right).

**Step.4: Life is complicated at high frequencies**

Increase the frequency of the input square wave from 1hz to 10hz. (Repeat this for all three C = [10uf, 47uf,100uf] capacitors) What do you observe at the output for each case? Why do you think this is happening? Derive an analytical expression describing the output and confirm its validity by comparing the outputs of the three circuits.

**Step.5: High RC lines and their consequences in propagation delay**

In today's VLSI technology, the maximum speed of the chip becomes limited by the delay of the interconnect system. You can think of this in the following simple example. There are certain conditions where a simple square wave (usually the clock) must feed thousands of other logic gates (Figure.2a). Each gate carries a little capacitance, but if you sum them all up, they can add up to a considerable amount of capacitance. Moreover, the interconnect wires are also non-ideal and have non-zero resistances. These wires can be very long and also add up to a noticeable series resistance. A simplified scenario is presented in Figure.2b. Let’s see what complications such non-idealities can introduce.

As you saw in Step.2, as the time constant of an RC circuit becomes larger than frequency of the input signal, output signal will not resemble the input anymore. This has consequences in logic circuits.

Use the [CD4011BE chip](https://www.ti.com/lit/ds/symlink/cd4011b.pdf?ts=1598818336519&ref_url=https%253A%252F%252Fwww.ti.com%252Fstore%252Fti%252Fen%252Fp%252Fproduct%252F%253Fp%253DCD4011BE) (NAND) to construct a simple inverter gate by connecting the inputs of a single NAND gate together. Look up the datasheet, and power the chip using Arduino’s supply rails accordingly (Use 5V rail for the VDD). Test the circuit by feeding your 1Hz square wave to the input and observing the output. Output should go high when input is low and vice versa.

Suggestion: use beams=2 on the probe, so you can probe both input and output of the gate using two scope channels on pin-A0 and pin-A1, at the same time. Connect A0 to input and A1 to output (or vice-versa). If you are confused which signal is input or output, just disconnect one of the pins for a second and look at the serial plotter.

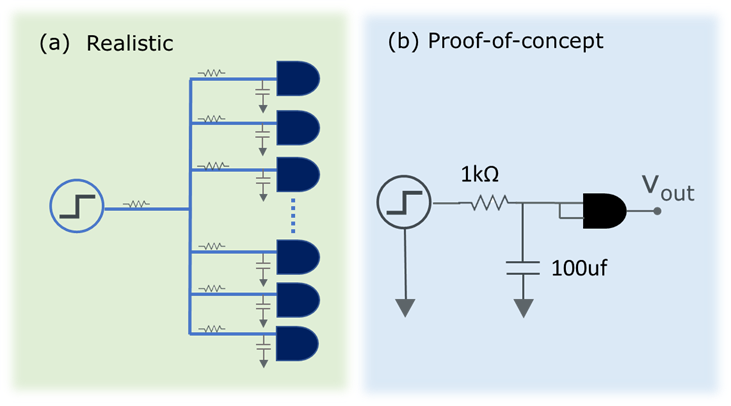


Figure.2. **(a)** In VLSI logic, certain signals must drive many logic gates at the same time. Each gate carries an specific amount of capacitance. Moreover, the interconnect wiring is also not perfect. **(b)** The non-idealities in Fig2.a can be simplified in a signal driving an inverter with an RC series in its route.

Construct the circuit in Figure.2b. Here, R=1kΩ and C=100uf resemble the large RC of a highly integrated circuit as seen in the Fig2. a. Probe the signal simultaneously at the input and output again.

**[Side Note4]:** When going to 2 probe mode, your y-axis scale changes by half. You might notice that in 2 probe mode, one signal oscillates between 0 and 2.5, and the other between 2.5 and 5. However, in reality, both signals are oscillating between 0 and 5, the program is just rescaling your y-axis to visualize the data.

You will notice that input and output do not start changing at the same time. It seems there is a delay between the two signals. This delay is called “propagation delay”.

* Estimate the propagation delay using the scope.
* Explain the mechanism behind propagation delay and the role that RC passives play in it. You might be interested in observing the signal on the capacitor.
* Increase the input frequency to 100 hz. Probe and explain the behavior at the output of the inverter.

Propagation delay of logic gates is one of the central concepts in circuit design. This delay can cause significant problems where different modules of a circuit become out-of-sync with each other and cause logic errors (you’ll learn more about this in ECE152A). Note that the value of R and C are usually orders of magnitude smaller, and consequently the propagation delay. However, keep in mind that current state-of-the-art processors run at clock frequencies up to 4.4 Ghz and propagation delay still poses a lot of limitations and regulations on appropriate logic designs. Also, in this lab, we looked at only one of the non-ideal effects that RC parasitic causes. Interconnects also have non-zero inductance which adds RL and RLC non-linearities to the design complexity. If you are interested, see chapter 4 in [Digital Integrated Circuits: A design perspective by M. Rabaey](https://www.pearson.com/us/higher-education/program/Rabaey-Digital-Integrated-Circuits-2nd-Edition/PGM263532.html) or come have a chat with me.