Spring 2021
EEE212 Microprocessors
Lab Assignment 4

In this lab, you are going to use Proteus Software to simulate your code written using MCU 8051 IDE Software. You will learn how to implement moving average filter that is commonly used for smoothing an array of sampled data. Please read the notes and the assignment requirements carefully since they are pretty important in terms of evaluation. The lab will have two questions, completing the first part will help you in the second part.

Important Notes:

- After completing the assignment, you need to get a check from one of the lab (grad) assistants (please do not ask to the tutors as they are not allowed to give checks by the rules). The check consists of explanation of the code and a small demonstration. Demonstration will be performed using **Proteus Software**.
- This is an individual lab. You can cooperate but you have to write your **OWN** code. Any kind of plagiarism will not be tolerated. Codes will be compared manually by assistants and by Turnitin software after the lab.

Q1: (40 pts)

Consider the following signals,

$$x[n] = 4 * \sin(2 * 2\pi * n * T_s)$$
$$y[n] = x[n] + \epsilon_n,$$

where $x[n]$ is the desired signal, T_s is the sampling period, ϵ_n is a sinusoidal noise, and $y[n]$ is the sampled data which contains both the desired signal $x[n]$ and the noise ϵ_n. Note that the frequency of $x[n]$ is 2 Hz. In this assignment our goal is to smoothen the sampled data $y[n]$ by using a moving average filter. In the dataset provided to you (`dataset.txt`), 250 samples of $y[n]$ are stored in 8-bit **signed** format. For easier implementation, we represent these data in hexadecimal format. The sampling frequency of the data is $f_s = 500$ Hz. In this part, you are only responsible of reading the data stored in the ROM and sending them to DAC to convert the digital data to an analog signal, continuously. This means, once you reach the last element of data (i.e n=249), you need to reset your data pointer to 0 and start reading the data from the beginning. Since the DAC accept only unsigned inputs, you need to transform each data by adding an offset value, 128, before sending them to DAC. please try to preserve the original data frequency (2 Hz) by including a proper delay equal to T_s=$1/f_s$ between each successive transmission of data to the DAC (delay=T_s=2ms). To demonstrate your work you will use the provided Proteus testbench (`testbench.pdsclip`).

Q2: (60 pts)

In this part, our aim is to process the data to obtain an estimate of the original signal, $\hat{x}[n]$, using the noisy observation $y[n]$. For this purpose, you could normally use a simple moving average filter of length 25 in the following form,

$$\hat{x}[k] = \frac{\sum_{i=0}^{24} y[k+i]}{25}.$$

However, to avoid floating point results and losing precision due to integer conversion, we will ignore the division, which will only change the scaling of the estimated signal while preserving its shape. That is, you will use the following filter

$$\hat{x}[k] = \sum_{i=0}^{24} y[k+i]. \tag{1}$$

For instance, you will estimate $x[5]$ as,

$$\hat{x}[5] = y[5] + y[6] + \ldots + y[29] \ .$$

To implement this step, you will need to modify your code from previous part to use the above filter before sending each data to the DAC. Basically you will perform the following steps for $k = 0, \cdots, 249$, continuously.

1. Estimate $\hat{x}[k]$ by summing $y[k], y[k+1], \cdots, y[k+24]$.
2. Carry $\hat{x}[k]$ to 8-bit unsigned number range by increasing its value with an offset (128).
3. Send the result of addition to P2.
4. Wait for t seconds.

Same as part one, we suggest that you adjust the delay t=T_s=2ms (sampling period) to see a signal close to the original wave frequency. Once you finished the above steps for $k = 249$, similar to the part one, you will start from $k = 0$ again and continue indefinitely.

Note: While implementing the filter and calculating the sum in Eq. 1, you do not need to worry about the overflow condition. The amplitude of the noisy data and the filter length are chosen such that the result will fit into an 8-bit register.