# Tick-Tock: Executing Instructions at Regular Intervals

**A little demo workshop, fall 2020 (Engn/Phys 207), by JE**

## Introduction

In many instances, electronics measurement devices need to repeat a multiple tasks (execute instructions) at regular intervals. For example, in our PPG circuit, let's say we want to acquire data at a sampling rate at $f_s = 100$ Hz (possibly overkill for this application, you can adjust to taste). This means that every $T_s = 1/100$ s = 10 ms = 10000 μs, the microcontroller needs to do the following going around the `loop()`:

1. Turn on the red LED
2. Take an analog reading
3. Turn off the red LED
4. Do the same for the IR LED
5. Do some math to compute convert from binary (0 – 1023) to physical voltage
6. Do some more math: compute peak-to-peak voltage, average voltage for both red and IR data streams
7. Print the result to the Serial monitor

Phew, and here I thought it was bad when I used to get as a kid: *"Brush your teeth; get dressed; put on your shoes."* And now back to our regularly schedule programming…!

The fundamental question is: How can we properly program the Feather 32u4 co repeat this set of instructions accurately and precisely? ***Accurately*** means that we are very close to our $T_s = 10$ ms = 10000 μs target prescribed above on average (or whatever you ultimately choose it to be). ***Precisely*** means that there is very little variation in this timing between loop cycles. For example, it might be 10004 μs one loop, then 10008 μs the next, then 10002 μs the next, but there are never really numbers that stray from the target---most of them hit very near bullseye.

## To delay() or not to delay()

We are familiar with `delay()` function in Arduino. We have used it repeatedly this term to control the approximate timing of a `loop()`, hence one way to control the sampling rate, in theory. But what about in practice? Look up the Arduino documentation (linked in first sentence or just google it), and carefully read and take heed of the "Notes and Warnings". What is the major potential drawback to using `delay()`?

1. Use the code provided **`delayLoopTiming.ino`** to assess the accuracy and precision of the actual sampling period compared to the desired sampling period. Note this code uses the delay() function to set the regular interval at which the loop executes. (Make sure to study the code to understand where and how this is happening!). To make this concrete, per our example above, we want a 10000 μs interval to achieve a sampling rate of 100 Hz.
    a. Is the actual, measured interval 10000 μs? How accurate and precise is the loop timing? Quantify this by computing the mean and standard deviation loop interval. You can do this by copy and pasting the contents printed in the serial monitor to a computer program of your choice (Excel, Matlab, or other).
    b. Compute the % difference between the actual loop time relative to the desired loop time.
    c. If the loop time is "significantly" different than desired, what would one approach be to correct that issue? Try your solution and see if it fixes the issue
    d. Now add some basic lines of code to toggle LEDs. Does your solution from part 1c still work? Hint: probably you are missing the 10000 μs target again, by a noticeable amount.

## A better way: elapsedMillis and elapsedMicros

2. You will likely have found that `delay()` isn't really your friend here. You may come up with a workable solution to part 1c, but there really is a better, cleaner way! Enter stage left: `elapsedMillis()`. Note: you may need to install the `elapsedMillis` library using the library manager in the Arduino IDE (see figure below). Once the library manager pops up, Search for elapsedMillis. Install the version by Paul Stoffregen (one of the original Arduino developers-legit!).

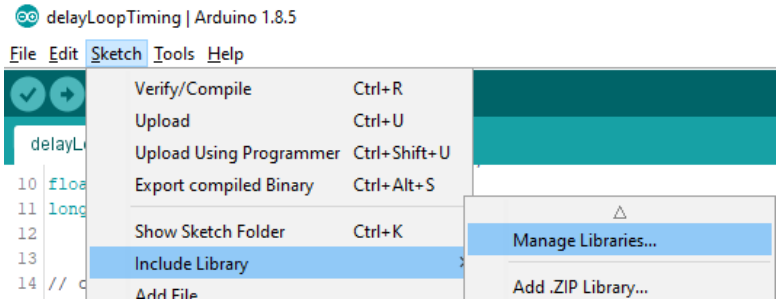


*Figure 1. Using library manager to install the **elapsedMillis** library.*

a. Use the code provided **`elapsedMicrosTimingDemo.ino`** to assess the accuracy and precision of the actual sampling period compared to the desired sampling period. Note this code uses the elapsedMicros variable is being used to set the regular interval at which the loop executes. (Make sure to study the code to understand where and how this is happening!). To make this concrete, per our example above, we want a 10000 μs interval to achieve a sampling rate of 100 Hz.
b. Is the actual, measured interval 10000 μs? How accurate and precise is the loop timing? Quantify this by computing the mean and standard deviation loop interval. You can do this by copy and pasting the contents printed in the serial monitor to a computer program of your choice (Excel, Matlab, or other).
c. Compute the % difference between the actual loop time relative to the desired loop time.
d. If the loop time is "significantly" different than desired, what would one approach be to correct that issue? If not, great. If yes, try your solution and see if it fixes the issue.
e. Now add some basic lines of code to toggle LEDs. Does your solution from part 2d still work? Hint: probably should be hitting the 10000 μs target again with high accuracy and precision.

## Conclusion

We often need to complete a set of tasks at regular repeating intervals. Use the `elapsedMillis()` and `elapsedMicros()`!! They are easy to use and way avoid a lot of pitfalls associated with the (not so) trusty old delay().

You may note that there IS a small error between the desired and actual timing achieved, even with elapsedMicros(). If you need *really* accurate timing, there are solutions to doing so, namely using ***Interrupts***. We'll save formal presentation, workshop, and discussion of that topic for another day (or take Electronics! ☺). For the curious cat, check out: https://learn.adafruit.com/multi-tasking-the-arduino-part-2/what-is-an-interrupt and https://www.pjrc.com/teensy/td_timing_IntervalTimer.html