Zephyr: Timers Lab

BME554L - Fall 2025 - Palmeri

Dr. Mark Palmeri, M.D., Ph.D.

2025-09-22

Table of contents

Refactor your GPIO Lab Code 1
Git Version Control 1
Test the Accuracy of your Timing 2
Methods 2
Logging Statements 2
Oscilloscope Measurements 3
Discussion 3
How to Ask for Help 4
What to Submit 4

Refactor your GPIO Lab Code

You will be refactoring your code from the GPIO lab to now use timers. Do all of the refactoring below on a new branch called `timers`.

- Eliminate any conditional logic (`if/else` statements) to test for elapsed time to toggle/set LED output states.
- Use timers for all LED illumination events.
- You do not have to use work queues if your timer handler functions run quickly and do not block. However, if your handler functions are more complex, you should use work queues to submit the work to the system work queue.

Git Version Control

- Use best practices for version control (branching, commit messages, etc.).
- Do all development on a dedicated branch that is merged into `main` once it is functional.

- Commits should be very specific to the changes/additions you are making to your code. This will help you and others understand what you did and why you did it.

- On a given development branch, try to implement one small piece of functionality at a time, commit it, and then move on to the next piece of functionality.

> **Important**
>
> You do not want one, monolithic git commit right before you submit your project.

Test the Accuracy of your Timing

For the measurements below, record and analyze your data in the technical report captured in a Jupyter notebook called `testing/timer_testing.ipynb`. You can use the Jupyter notebook included in the lab repository (`testing/technical_report.ipynb`), which includes some example code, as a starting point, but be sure to rename the notebook in the `testing/` directory.

Your technical report should include:

Methods

A brief Methods section describing what you did (e.g., how you captured the timing via logging or how you measured timing on the oscilloscope).

For each section below, all raw data should be read in from a dedicated file (e.g., a CSV file).

> **Warning**
>
> Do *not* manually type in data in your Jupyter notebook.

Logging Statements

- Use logging statements to test the accuracy of your heartbeat and action LED timing for the (a) default, (b) fastest and (c) slowest blink rate of your action LEDs.

> **Note**
>
> Note that logging statements, by default, are low priority and non-blocking (asynchronous).

- Given the latency of when a log message may be printed relative to the event you are trying to get the timing of, you should save the execution time of the timing handler to be log printed later.
- Save your logging output data to a CSV file to be read into your Jupyter notebook. There is no elegant way to directly save your data to a CSV file directly from the Terminal in VS Code.

> **Note**
>
> Two possible approaches: 1. You can save the entire Terminal output to a file and parse it in your Jupyter notebook, or 2. You can cut-and-paste the relevant values from the Terminal output to a CSV file.

- Estimate the 95% confidence interval (2x standard deviation for normally-distributed data) for your timing relative to the nominal specification.

Oscilloscope Measurements

- Repeat your timing accuracy analysis you performed with logging statements using an oscilloscope directly measuring the GPIO pin signals. *Remember that all of the GPIO pins are accessible as female header sockets on the development kit; you do not need to try to directly connect to the pads of the LEDs and buttons!*
- Save your oscilloscope measurements to a CSV file to be read into your Jupyter notebook.
- Use the oscilloscope cursors to measure the timing intervals.

> **Note**
>
> Remember that you will need multiple independent measurements to calculate confidence intervals.

- Quantify a 95% confidence interval for your timing relative to the nominal specification using these oscillosocpe measurements.

Discussion

Discuss how well the measurements match their nominal values, and if significant deviation has occurred, discuss why this may have happened and how it could be improved moving forward.

How to Ask for Help

1. If you have a general / non-coding question, you should ask your TAs / Dr. Palmeri on Ed to allow any of them to respond in a timely manner.
2. Push you code to your GitLab repository, ideally with your active development on a non-`main` branch.
3. Create an Issue in your repository.
    - Add as much detail as possible as to your problem, and add links to specific lines / section of code when possible.
    - Assign the label "Bug" or "Question", as appropriate.
    - Be sure to specify what branch you are working on.
    - Assign the Issue to one of the TAs.
    - If your TA cannot solve your Issue, they can escalate the Issue to Dr. Palmeri.
4. You will get a response to your Issue, and maybe a new branch of code will be pushed to help you with some example syntax that you can use `git diff` to visualize.

What to Submit

- Make sure that your complete `timing_analysis.ipynb` notebook is in your `timers` branch in a directory called `testing/`, which also includes all raw CSV data.
- Make sure that your Jupyter notebook reads in data from relative paths for the repository, **not** absolute paths on your local machine.
- Push your `timers` branch to GitLab.
- Create a new Merge Request to merge `timers` into your forked repository's `main` branch.

> **Warning**
>
> Do not create a Merge Request in to the parent repository that you forked from!

- Merge your `timers` branch into your `main` branch.
- Create an annotated tag for that merge commit called `v1.1.0`.
- Create a new Issue assigned to Dr. Palmeri to request a code review for `v1.1.0`.

> **Note**
>
> If you fix any bugs after creating the original annotated tag, increment up the patch number with a new annotated tag version number, and update the Issue to reflect which annotated tag should be reviewed.