

Zephyr: State Machine Framework Refactor Lab

BME554L - Fall 2025 - Palmeri

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

2025-10-06

Table of contents

| | |
|---|---|
| Git Version Control | 1 |
| Refactor: State Machine Framework | 1 |
| How do I ask for help? | 2 |
| What to Submit | 2 |

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.

Refactor: State Machine Framework

1. Create a new development branch in your forked repository you have used the last few weeks called `refactor_smf`.

2. Refactor your code to use the [Zephyr State Machine Framework \(SMF\)](#).
 - Use **entry** and **exit** substates to handle state transitions where something should be done “once” when entering or exiting a state.
 - Refactor code to make sure that state-specific things are done in that state. For example, if an error LED should only be on in the **ERROR** state, then the code to turn on the LED should be in the **ERROR** entry state and not done in the exit of the previous state.
 - Make sure that your device initialization is done in an **INIT** state (so that, in the future, a soft reset can allow the device to be completely re-initialized).
3. Make sure that your state diagram is updated to reflect these entry / exit actions. This diagram should exactly match your firmware implementation of the SMF.
4. Qualitatively make sure that your device continues to function as expected.

How do I 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 your 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

1. As with the previous labs, push this development branch to Gitlab and merge it into **main**.
2. Create an annotated tag for that merged commit called **v1.4.0**.
3. Create an Issue assigned to Dr. Palmeri to review your code for **v1.4.0**. (Teaching team will upload a zip archive of your project to Gradescope.)