Zephyr ADC & PWM Lab

BME554L - Fall 2025 - Palmeri

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

2025-06-24

Table of contents

Fork and Clone the Repository

- Fork this repository to your own GitLab account.
- Add Dr. Palmeri (`mlp6`) as a `Maintainer` to your GitLab repository. He will add TAs, graders and projects labels to your repository once this has been done.

Git Best Practices

- 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.

Firmware Expectations

- All firmware should be written using the **State Machine Framework**.

- Choose your states for each part as a firmware engineer would, using what you have learned so far this semester.
- Timers, work queues, callbacks, and interrupts should be used as appropriate.
- All good coding practices developed this semester should be followed.
- Use logging to display state information and other relevant information, warnings, and errors. Debugging log messages can remain in the code, but the logging level should be submitted at the `INF` level.
- Include a state diagram in your repository (`state_diagram.png`) using UML (`state_diagram.puml`) or some equivalent.

Part I: ADC Sampling

Firmware Functional Specifications

- Do all development and testing for Part I on a developnment branch called `part1`.
- `LED` and `BUTTON` number references below are based on Devicetree labels (not the annotation on the DK board itself).
- There should be a heartbeat LED (`LED0`) that blinks at 1 Hz with a 25% duty cycle at all times when the firmware is running.
- When `BUTTON0` is pressed:
  - Make a measurement using the `AIN0` channel of the ADC.
    - Use the `ADC_REF_INTERNAL` reference voltage.

    * Linearly map 0-3.0 V measured on the `AIN0` input to an `LED1` blink rate of 1-5 Hz (e.g., 0 V -> 1 Hz; 3.0 V -> 5 Hz) with a duty cycle of 10%.
    * `LED1` should remaining blinking for 5 seconds after the button has been pressed.

Testing

For the following testing, you can use a power supply to provide known voltage input to `AIN0`.

- Quantify how linear the relationship is between the voltage applied to `AIN0` and the `LED1` blinking frequency.
    - Not sure how to quantify linearity? Consider plotting the data and presenting the R^2 of a linear regression fit.
    - Remember that single data points for any single input voltage pair is not adequate; multiple measurements should be made and error bars presented on all plots.
- Quantify the accuract of the `LED1` 10% duty cycle for each frequency that you measure.
- Quantify the accuracy of `LED1` 5 sec ontime duration.
- Present all of these data, your analysis and your interpretation in a `technical_report.ipynb` Jupyter notebook.

Commit-n-Merge Part I

- Merge your completed `part1` branch into your `main` branch using a Merge Request on Gitlab.
- Create an annotated tag of your `main` branch with all part of this lab merged in called `part1`.
- Create an Issue for Dr. Palmeri to review Part I, assigning it the `Review` label.

Part II: Buffered Differential ADC Sampling

Firmware Functional Specifications

You will keep all of the functionality of Part I, and in a new development branch called `part2`, add the following functionality to your firmware:

- When you press `BUTTON1` , add an additional differential ADC measurement using `AIN1` and `AIN2`.
- Choose the reference voltage, gain, bit depth and acquisition time to adequately sample at least 20 cycles of a 10 Hz sinusoidal signal (V_{pp} = 2 V).

- Implement the `extra_sampling` buffering of the ADC for this differential sinusoidal signal measurement so that all of the data are stored in an array in a single `adc_read()` call (i.e., you are not using a kernel call for every sample).
- Disable the `BUTTON1` while your device is reading `AIN1` and `AIN2` and performing calculations below.
- Write a **library** called `calc_cycles` that calculates the number of sampled sinusoidal cycles in the buffered ADC samples.
  - This can be rounded to the nearest integer, depending on your algorithm.
  - Have this calculated number of cycles displayed using `LOG_INF()` in the console.
- Write log messages to your serial terminal that:
  - Display a HEX array of the buffered ADC samples.
  - :warning: Be careful for the log message memory consumption! Remember that the data will be saved using two's complement.
- Update your state diagram for this new functionality.

Testing

- Input a 10 Hz sinusoidal signal with a 2 V V_{pp} into the differential `AIN1` and `AIN2` inputs.
- Create a plot of your input signal and the buffered ADC samples (using the HEX array output).
  - The HEX data array from the terminal can be copied and pasted into a text file in your git repository to read into your Jupyter notebook for analysis.
  - Example code to decode your HEX data is included in the Jupyter notebook.
  - Discuss any differences between the input signal and your sampled signal.
- Compare your calculated number of cycles in the sampling period to the nominal value. Discuss any differences between your calculated number of cycles and the expected number of cycles.
- Add this analysis to testing/technical_report.ipynb.

Commit-n-Merge Part II

- Merge your completed `part2` branch into your `main` branch using a Merge Request on Gitlab.
- Create an annotated tag of your `main` branch with all part of this lab merged in called `part2`.
- Create an Issue for Dr. Palmeri to review Part II, assigning it the `Review` label.

Part III: Steady-State PWM Output

Using all of the the functionality of Parts I & II, in a new development branch called `part3`, add the following functionality to your firmware:

Firmware Functional Specifications

- Using the Part I functionality to read a DC voltage on `AIN0`, map this read DC voltage to scale the maximum brightness of `LED2` using a PWM output.
  - For example:
    - If `AIN0` = 0 V, then `LED2` should be off (0% duty cycle).
    - If `AIN0` = 1.5 V, then `LED2` should be at 50% brightness (50% duty cycle).
    - If `AIN0` = 3 V, then `LED2` should be at maximum brightness (100% duty cycle).
- Update your state diagram to include the new functionality.

Testing

- Quantify the linearity of the maximum brightness of `LED2` as a function of `AIN0` voltage ranging from 0-3 V.
- Present your data and analysis in the technical report Jupyter notebook.

Commit-n-Merge Part III

- Merge your completed `part3` branch into your `main` branch using a Merge Request on Gitlab.
- Create an annotated tag of your `main` branch with all part of this lab merged in called `part3`.
- Create an Issue for Dr. Palmeri to review Part III, assigning it the `Review` label.

Part IV: Sinusoidal Modulation of PWM Output

Using all of the the functionality of Parts I - III, in a new development branch called `part4`, add the following functionality to your firmware:

Firmware Functional Specifications

- Modulate the brighness of `LED3` to match the read sinusoidal voltage on `AIN1` and `AIN2` after pressing `BUTTON2`, as implemented and testing in Part II, **with as little latency (phase distortion) between the input and output signals**.
- The blocking nature of the buffered acquisition scheme of Part II will not be ameable to realtime modulation of the PWM output.
  - **Consider if the `async` / `callback` ADC sampling approach is appropriate for this task,** or
  - Consider a non-buffered ADC sampling approach that uses a timer to sample the ADC at a fixed rate.
- Set the minumum and maximum sinusoidal brightnesses to be PWM duty cycles of 0 and 100%, respectively.
- Update your state diagram to include the new functionality.

Testing

- Using the oscilloscope, measure your input sinusoidal signal on `AIN1` and `AIN2` and the output PWM signal on `LED3` at the same time.
- Save this acquired oscilloscope data to a CSV file using a USB memory stick.
- In your Jupyter notebook, plot the input and output signals.
- Calculate the frequency of your PWM signal and the phase difference between the PWM sinusoid and your input signal.
- Present your data and analysis in your Jupyter notebook.
- Discuss the latency of your system and how you could improve it in the future.

Commit-n-Merge Part IV

- Merge your completed `part4` branch into your `main` branch using a Merge Request on Gitlab.
- Create an annotated tag of your `main` branch with all part of this lab merged in called `part4`.
- Create an Issue for Dr. Palmeri to review Part IV, assigning it the `Review` label.

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.