Analog-to-Digital Conversion (ADC) Lab

BME554L - Fall 2025 - Palmeri

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

2025-10-15

Table of contents

Learning Objectives

- Implement single-sample and buffered ADC data acquisitions.
- Implement fixed-reference and differential ADC inputs.
- Implement synchronous and asynchronous ADC acquisitions.

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.

> **i** Note
>
> You will continue to build upon the firmware from your previous labs, but you are replacing all functional specifications with those in this lab.
> All states should be reconsidered given the new functional specifications.
> This significant change in functionality is represented with the version tags changing to a new major version number starting at `v2.0.0`.

ADC Single Sampling

Firmware Functional Specifications

- Do all development and testing for Part I on a developnment branch called `adc_single_sample`.
- `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.
- `BUTTON0` should be disabled while the ADC measurement and `LED1` blinking.
- All errors should result in a transition to an `ERROR` state that can only be exited by hitting a reset button.

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 by acquiring data to perform a linear regression of these variables and analyzing the resultant R^2, slope and intercept.

> **Tip**
>
> 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.

> **Tip**
>
> The `bme554.py` library has a functions called `plot_with_fit()` and `calculate_confidence_intervals()` that can help.

- Quantify the accuracy of the `LED1` 10% duty cycle for each frequency that you measure.
- Quantify the accuracy of `LED1` 5 second ontime duration.
- Present all of these data, your analysis and your interpretation in a new Jupyter notebook named `testing/testing_adc_single_sample.ipynb`.

Commit-n-Merge Single Sample Acquisition

- Update your state diagram.
- Merge your completed `adc_single_sample` 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 `v2.0.0`.
- Create an Issue **assigned to your teaching assistant** indicating that this tag is ready for review.

Buffered Differential ADC Sampling

Firmware Functional Specifications

In a new development branch called `diff_adc`, 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). 0 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. 0 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 dsplay a HEX array of the buffered ADC samples.

- Update your state diagram for this new functionality.

> **Warning**
>
> Be careful for the log message memory consumption! You may need to adjust the stack size of the appopriate thread if you are getting memory-related firmware faults.

> **Note**
>
> Remember that the data will be saved using two's complement.

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).

> **Tip**
>
> There are helper functions in `bme554.py` called `read_hex_data()` and `unwrap_twos_completely()`.
> 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.

- Discuss any differences between the input signal and your sampled signal.
- Calculate the frequency of your sampled signal and compare it to your input frequency.
- Add this analysis to a new Jupyter notebook named `testing/testing_adc_sync.ipynb`.

> **Warning**
>
> Merging Jupyter notebooks from different branches can be tricky and usually requires some extra tools or manual intervention. We are creating separate Jupyter notebooks for each part of this lab in case you jump to code development in a different part before finishing the testing for the previous part.

Commit-n-Merge Buffered Acquisition

- Merge your completed `diff_adc` 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 `v2.1.0`.
- Create an Issue **assigned to your teaching assistant** indicating that this tag is ready for review.

Asynchronous ADC Sampling

Firmware Functional Specifications

In a new development branch called `async_adc`, add the following functionality to your firmware:

- Refactor your blocking, synchronous buffered ADC sequence acquisition to an **asynchronous** acquisition that populates the same array that was populated with the synchronous approach.
- Utilize the 3 Zephyr enumerations presented in lecture to modulate the behaviour of the asynchronous callback function.
- Use a kernel poll (`k_poll`) to wait for the ADC sequence to complete.
- Implement a timeout mechanism to handle the case where the ADC sequence does not complete within a reasonable time frame.

Testing

- Repeat the testing you performed in a new Jupyter notebook named `testing/testing_adc_async.ipynb`.
- Discuss the benefits and drawbacks of the asyncronous ADC sequence sampling compared to the synchronous sampling.

Commit-n-Merge Asynchronous Sampling

- Merge your completed `async_adc` 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 `v2.2.0`.
- Create an Issue **assigned to your teaching assistant** indicating that this tag is ready for review.

Timer-Based Single Sampling to Array

Firmware Functional Specifications

In a new development branch called `timer_adc`, add the following functionality to your firmware:

- Refactor your firmware to use a kernel timer to do single ADC samples that are sequentially stored in the array that was associated with the ADC sequence in the previous two sections.

Testing

- Repeat the testing you performed in the asyncronous section for this timer-based implementation in a new Jupyter notebook named `testing/testing_adc_timer.ipynb`.
- Discuss the benefits and drawbacks of using a timer-based ADC sampling scheme compared to the previous approaches when trying to accurately sample a sinusoidal signal.

Commit-n-Merge Timer Sampling

- Merge your completed `timer_adc` 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 `v2.3.0`.
- Create an Issue **assigned to your teaching assistant** indicating that this tag is ready for review.

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.