ECG & Temperature Sensor with BLE Lab

BME554L - Spring 2026 - Palmeri

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

2026-02-16

Table of contents

This lab will be the final addition of functionality to your course firmware repository. You should take all functional specifications below to override and replace any previous lab functionality. You should refactor your code from all of the previous labs to demonstrate your mastery of best coding practices from this semester.

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.

Best Coding Practices

You should refactor your code from all of the previous labs to demonstrate your mastery of best coding practices from this semester. Some reminders:

- Functions should be short and do one thing. They should return an exit code that is checked in the calling function, indicating success or failure.
- MACROS! Avoid hard-coded values in your code.
- Use structs to organize related data.
- Use libraries for code that is self-contained.
- Use the `LOGGING` module to log errors, warnings, information and debug messages.
- You should not have any compiler/build warnings. The CI script will build against `v3.2.1` of the Zephyr SDK.

Firmware Functional Specifications

- Write all firmware using the state machine framework.
  - Do all device initialization in an `INIT` state.
  - Have an `IDLE` state when the device isn't making any measurements.
  - Have an `ERROR` state if any error exit codes are returned from any functions.
    - All 4 LEDs should blink at a 50% duty cycle (`ON:OFF` time), in-phase with each other, in the `ERROR` state.
    - An error condition should post an error-related event that causes the device to enter the `ERROR` state.

    - The error code should specify the error condition that caused the device to enter the `ERROR` state. For example, you may choose to have a bit array that can capture multiple error conditions.
    - A BLE notification should be sent with the error code (see BLE custom service/characteristic below).
    - If the device remains in the `ERROR` state for more than 2 minutes, this it should "gracefully" terminate the state machine function and go into a low-power mode.
  - Implement states of your choosing for the following measurements, calculations and BLE communications.

- Have a heartbeat `LED0` that blinks every 1 second with a 50% duty cycle (`ON:OFF` time) in all states.
- Implement functionality to measure a battery voltage (0-3.0 V) using `AIN0`:
  1. When the device first powers on, and then
  2. Every 1 minute thereafter, but only when in the `IDLE` state.
  3. You won't actually be connecting a battery to your device; you can use a power support or another voltage source to input a voltage to `AIN0` to simulate a battery level.
- If your battery level is <75% of the nominal voltage (3.0 V), then your device should "gracefully" terminate the state machine function and go into a low-power mode.
- Have the brightness of `LED1` linearly modulated by the percentage of the battery level.
- Implement functionality to make a temperature measurement with your `MCP9808` sensor (in degrees Celsius) after pressing `BUTTON1`.
- When you press `BUTTON2`, the device should measure "instantaneous" heart rate, expected to range from 40-200 BPM, using an ECG signal (ranging from -500
- 500 mV, note this is bipolar) from the function generator (see video on how to setup the function generator to output an ECG signal).
  - Pressing `BUTTON2` during ECG measurements will stop the "live" ECG measurements.
  - Blink `LED2` with a 25% duty cycle (`ON:OFF` time) at the average heart rate you have calculated (updated as often as you feel is appropriate).
  - You should decide how often the heart rate is updated in the blinking rate of `LED2` (e.g., every 1 second, every 5 seconds, etc.).

  - `BUTTON1` is allowed to be pressed during ECG measurements to take temperature measurements.

- Have Bluetooth notifications after the measurements are complete/updated and data have been processed, using the BLE services and characteristics described below.
  - Configure the **DIS (Device Information Service)** to report the device model as your name.
  - Set the **BAS (Battery Service)** to report the battery level of your device. (This isn't actually a battery level, but we're using the `AIN0` measurement as a surrogate for a battery level.)
  - Set the **Heart Rate Service** to report the average heart rate (see Resources section below.)
  - Setup a custom service with the following custom characterisitics:
    * `Temperature` for the I2C temperature sensor data in degrees Celcius.
    * `Error Code` for the error code that caused the device to enter the `ERROR` state.

- `BUTTON3` should be used to reset the device from the `ERROR` state or any measurement state and return to the `IDLE` state.
- `BUTTON0` should be used to return the device to the `IDLE` state from any measurement state without resetting the device (i.e., preserve all measured values that are stored in memory).
- Use timers, kernel events, work queues, threads and any other Zephyr RTOS features as needed to implement the above functionality.
- Any functionality not explicitly described above is left to your design discretion.

BLE Server (Mobile App)

- Your device can connect via BLE to a mobile app called nRF Connect.
- This app can be used to read the services and characteristics that your device is advertising.

State Diagram

- Generate a detailed state diagram that all states, events and actions for your firmware.
- You should add states, events and actions as needed to fully describe the functionality of your firmware.

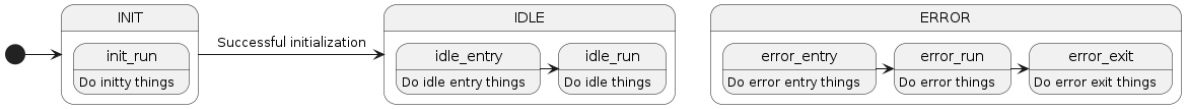

Figure 1: State Diagram

Testing & Verification

Create a new Jupyter notebook named `final_technical_report.ipynb` for your testing and verification technical report. The notebook should include the following sections:

Battery Voltage Measurement

Verify the accuracy of your battery level measurement (`AIN0`) for 0-3 V, as read through the nRF Connect app Bluetooth Battery Level GATT. Quantitative analysis should include:

- Linear regression analysis of the input voltage versus the battery level for both the `LOG_INF()` output, the oscilloscope measurement of the PWM duty cycle, and the nRF Connect app.
- 95% confidence intervals for the slope and intercept of the linear regression.

ECG Heart Rate Measurement

- Describe your method for calculating average heart rate from the ECG signal input. This should include any filtering, peak detection, and averaging methods used.
- Demonstrate that your average heart rate measurement is accurate, as recorded through `LOG_INF()` output, the oscilloscope measurements, and the nRF Connect app, relative to what was set on the function generator. This should be done for 40, 60, 120, 150, and 180 bpm. The 25% duty cycle should also be verified. Quantitative analysis should include:
  - Linear regression analysis of the recorded heart rate versus the set heart rate for both the `LOG_INF()` output, oscilloscope measurements, and the nRF Connect app.

  - 95% confidence intervals for the slope and intercept of the linear regression.
  - Verification that the duty cycle of out output is 25% for all heart rates.

Temperature Sensor Measurment

- Demonstrate that your temperature sensor can have a room temperature measurement read through `LOG_INF()` output and the nRF Connect app.
- Do this in 3 locations with different temperatures.
- You do not need to verify accuracy, just a reasonable measurements with agreement.

How to Generate an ECG Signal

- WaveStation 2012 AWG
- Digilent Waveforms Script

Grading

- This final project is worth 20% of your grade. *Absolutely no late submissions will be accepted.*
- Git version control will be graded based on best practices.
- Firmware will be graded based on all best practices taught throughout the semester.
- Code organization and coding best practices will be graded.
- State diagram will be graded based on completeness, accuracy and ease of interpretation.
- Testing and analysis technical report will be graded based on presentation, completeness, and accuracy.

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 all of your branches have been merged into the `main` branch.
- Create an annotated tag called `v4.0.0` to mark the commit that you want to be graded.
    - If you fix any bugs after creating this tag, you can create another tag called `v4.0.1`, etc.
    - Your latest tag will be the one that is graded up until the final due date/time of the project.
- Create an Issue assigned to Dr. Palmeri titled "Final Project Submission" and labelled `Review`.

> **!** Important
>
> All repositories will be cloned with the latest tag at the due date/time for grading. No repository commits after the due date/time will be accepted.

- Upload a PDF of your final technical report Jupyter notebook to Gradescope.

Resources

Heart Rate Service (GATT)

- BLE Sample: Peripheral Heartrate
- Zephyr Docs: BT Heartrate Service