NTNU
Kunnskap for en bedre verden

Department of ICT and Natural Sciences

AIS2201 - Digital Signal Processing

Portfolio Project Description
Part 3: Implementation

Kai Erik Hoff

Date

Overview

This document describes part 3 out of 3 for the portfolio project for AIS2201 - Digital Signal Processing, autumn 2025. The focus of part 3 of the project is to convert the signal processing algorithm developed in part 2 to an embedde program which can run on a microprocessor in real-time.

Table of Contents

1 Checklist from part 2 of the project

Were you able to do the following?

1. Develop a prototype heart rate detection system which makes use of DSP principles to ensure robust heart rate detection in the presence of signal interference.
2. Use your simulated test signals and evaluation setup from part 1 to measure the robustness of your heart rate detection system.
3. Complete a status meeting with lecturer to receive feedback on your progress on part 2.

If not: **do not** proceed with part 3 of the project until the above list is completed.

2 Part 3: Implementation

At this stage of the project, you should have developed a prototype heart rate detection algorithm and a clear picture of how well the algorithm can detect heart rate in the presence of signal interference using simulated ECG data. The final stage of the project involves implementing the algorithm on a microcontroller, and using it to conduct heart rate measurements on real-world ECG signals in real time.

The main task for part 3 is converting your prototype algorithm developed in Python/Matlab to a embedded program which includes the buffer management eneded to make the algorithm work with a input signal being sampled in real-time. While the STM32 has been a staple of the assignments it is not a requirement that you use this specific microcontroller or the CubeIDE development environment for your project. However, your choice of programming language will reflect on the difficulty involved in implementing your system, and therefore determine whether your solution constitutes Level 1 **or** Level 2:

2.1 Level 1: Arduino C

Arduino C offers a simple-to use API for sketching embedded code without needing to concern oneself with hardware peripheral configuration etc. which most of you will already be familiar with. Due to it's popularity, open source projects allowing the use of Arduino C to program a wide range of microcontrollers are available (STM32 included[1]), and there are also libraries available offering support for calculating Fast Fourier Transforms[2]. If you find the prospect of implementing your heart rate monitor on an STM32 using the HAL API and CMSIS-DSP daunting, consider selecting Level 1 for this part of the project.

2.2 Level 2: "Hardware-specific" embedded C

The default choice for level 2 is implementing your heart rate monitor algorithm on an STM32F446re microcontroller using the STM32 CubeIDE programming tool. However, you are not absolutely required to use CubeIDE or even an STM32 for that matter. The main requirement which needs to be met to complete level 2 is that your program interacts with peripheral hardware components with a degree of abstraction similar to or lower than that in a STM32 C project. If you are very confident in your embedded programming abilities you are free to branch select any platform you wish, but beware that you will be on your own as teaching staff will likely not be able to assist if you run into problems.

[1] Getting started with stm32duino: https://github.com/stm32duino/Arduino_Core_STM32/wiki/Getting-Started
[2] ArduinoFFT: https://github.com/kosme/arduinoFFT

Aside from the tool/language used for embedded implementation, the architecture of the implemented DSP system into code will also be an opportunity to collect credits during grading. In applied DSP a lot of thought often goes to algorithm optimization with the aim of reducing the computational load. Examples from class include use of fast convolution instead of direct convolution and multistage decimation.

2.3 System testing with real ECG signals

When implementing your system, it is recommended that you use test signals from a signal generator or similar as an initial test. Once you have worked past all the hurdles of writing your algorithm in C, it is time to hook up the microcontroller to an actual ECG sensor and make heart rate measurements on a living person.

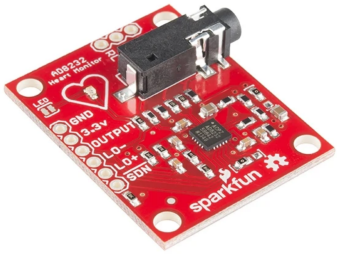

Figure 1: AD8232 Sparkfun Single Lead Heart Rate Monitor

The ECG sensor we will use is an AD8232 SparkFun Single Lead Hart Rate Monitor[3] shown in figure 1. The sensor has three output connectors which need to be connected to the microcontroller in addition to the power supply and ground connectors. These are:

1. `OUTPUT`: analog signal output for ECG signal
2. `LO+`: Digital "Leads Off Comparator Output" indicating if the Left Arm Electrode is disconnected.
3. `LO-`: Digital "Leads Off Comparator Output" indicating if the Right Arm Electrode is disconnected.

It is recommended you consult the hookup guide for the sensor[4] for information on how to connect the sensor module. Use the `LO+`/`LO-` signals in your program to drive an indicator (LED etc.) that the electrodes are disconnected, and need to be inspected before any actual heart rate can be measured.

WARNING!

To safeguard against the unlikely event of electrical shocks caused by faulty electronics, MAKE SURE THERE IS ABSOLUTELY NO POSSIBLE CONDUCTIVE PATH BETWEEN THE ECG ELECTRODES AND A MAINS POWER OUTLET! This means, if the microcontroller taking measurements is connected to a PC, do not connect the PC to a charger. Anything connected to the ECG sensor must solely battery powered.

Once you are hooked up to the ECG sensor, it is time to test your heart rate monitor with som real-world measurements. You can influence both heart rate and noise level by your activities while recording data:

[3]Link to product page: https://www.sparkfun.com/sparkfun-single-lead-heart-rate-monitor-ad8232.html
[4]Link to hookup guide: https://learn.sparkfun.com/tutorials/ad8232-heart-rate-monitor-hookup-guide

- Measuring heart rate while sitting/lying completely still will likely provide measurements with very little noise.
- Exaggerated heavy and slow breathing should provoke baseline drift to some degree.
- Moving the AD8232 sensor board close to e.g. a AC/DC charger should induce some very noticeable mains interference.
- Physical activity will provoke a lot of muscle activity aside from the heart beats, adding noise with an approximately white spectrum to the ECG signal.

It is recommended you use a UART protocol (such as the one presented in assignment 10) to stream both raw signal data and registered heartbeats to the PC, so you can verify through graphical signal inspection the degree to which your implemented heartbeat monitor is functioning as intended.

3 Project Report

By the end of Part 3: Implementation you should have all the parts required to complete the project report. This includes the remaining chapters listed below:

- **Chapter 1: Introduction**

  This should be a standard project introduction, giving a largely non-technical summary of the problem to be solved, as well as an overview of the project's objectives and methodology.

- **Chapter 5: Implementation**

  In the project template[5], this chapter is set aside for both a description of how the heartbeat detection algorithm was implemented, as well as results from real-world testing. Make sure you describe any changes and/or concessions that needed to be made in order for the algorithm to be a adapted for real-time processing. Remember that the presentation of results should also include some commentary. While thorough analysis of the results typically belong in the Discussion chapter, you should bring attention to key bits of information in the figures/tables etc.

- **Chapter 6: Discussion**

  This chapter is set aside for more in-depth analysis of your results, both using simulated data and real-world testing. Did the heartbeat detection system perform as well as you hypothesized? Were there any significant deviations between performance using simulated data and performance using real-world data? Remember that a good discussion chapter will not only identify discrepancies between theoretical behavior and real-world results, but also investigate what may be the cause of these discrepancies. Beware: vague, hand-waving references to e.g. "probably imprecise measurements etc." doesn't quite cut it.

- **Chapter 7: Conclusion**

  Summary of the points from the discussion, and relates them to the problem statement in the introduction chapter. If you have thoughts about alternative approaches that may have merit, this is a good place to add a sentence or two on the topic.

[5] Overleaf project report template