Class:	ENCE361
Project:	Step Counter Read Me
Date:	22 / 5 / 2025
Group number:	Group 44

Name:	Will Fraser	**Student number**:	87858780
Name:	Justin France	**Student number:**	75349339

Description of overall project

The purpose of this project was to create a step counter to track and record a user's steps and display useful information through the display LEDs and buzzer on the board. The step counter is built around the STM32C071 nucleo development board with an expansion board added to interface with the user. The STM32CubeIDE was used to program the board and develop the step counting software for this embedded system.

This step counter included all the basic information that a user would require when counting their steps. Major features of this project included a LCD display where the user could toggle between the step count, distance travelled and the goal progress by moving the joystick left or right. Different units for the corresponding screens could also be seen when the user moved the joystick up. The user is able to select a goal between 500 and 15000 steps depending on their abilities. LED lights provide the user with visual feedback on their progress towards this goal and a buzzer is used to notify the user upon completion. A test mode is also built into the program so the user can test the functionality of the system which can simulate steps without actually moving. These features integrate the hardware and software components of this system to provide the user with an intuitive and interactive experience.

Justification and description of modularisation

Modularisation consists of three layers the application layer, device layer and the hardware layer as seen in Figure 1. Application layer is the topmost layer and consists of the user interfacing modules. This layer handles all the tasks which include the major hardware elements on our board. app.c is at the top of this layer and its primary task is to determine what frequency each task is running at. Below the application layer is the device layer which acts as an intermediary step between the application layer and HAL. The device layer includes all the drivers and specific functions to make the hardware run. This makes it easier to update the program if a component is changed without affecting the higher-level application code. The HAL provides an interface between

operating system and the hardware which was auto generated by STM32cubeIDE when the ioc file was set up.

Each layer should only rely on the layer below it and this is successfully carried out in our program. The exception to this is the reliance tasks blinky, buzzer and buttons has on HAL. This is not ideal due to the use of the HAL get_tick function to time different aspects in each of these tasks. This function could be easily replaced but is not a big issue since the tasks are not heavily reliant on HAL.

Laying the program out like this allows for good reusability of the code as the device layer modules can be reused on different projects. It also allows for easier readability and maintenance as the code is separated out into digestible modules containing one functionality each. This also makes the project easier to scale and build upon as each feature is separated nicely and can be added without affecting other features too much.

The modules in our program have relatively high cohesion as each task is nicely separated to service one part of the hardware and are reliant on a select few modules, that are relevant to carry out each of there functions. numbers.c is the module with the lowest cohesion and would be a good place to improve the program. It currently handles the step detection, step count and test mode which are all relevant to the processing of the user's steps could be separated into different modules to improve the cohesion.

Our program has relatively low coupling. Modules in the same layer have low reliance on each other. The biggest exception to this is the state.c module as interaction is needed between it and the buttons module to change the state and disable some inputs depending on the current state. Having low coupling meant that it was easier to test the modules independently of one and other and this made debugging and maintenance of the code easier.

By passing information between the modules with getters and setters this improves the encapsulation of these modules. By exposing specific methods to read or modify the data, reduces unintended interactions between the modules.

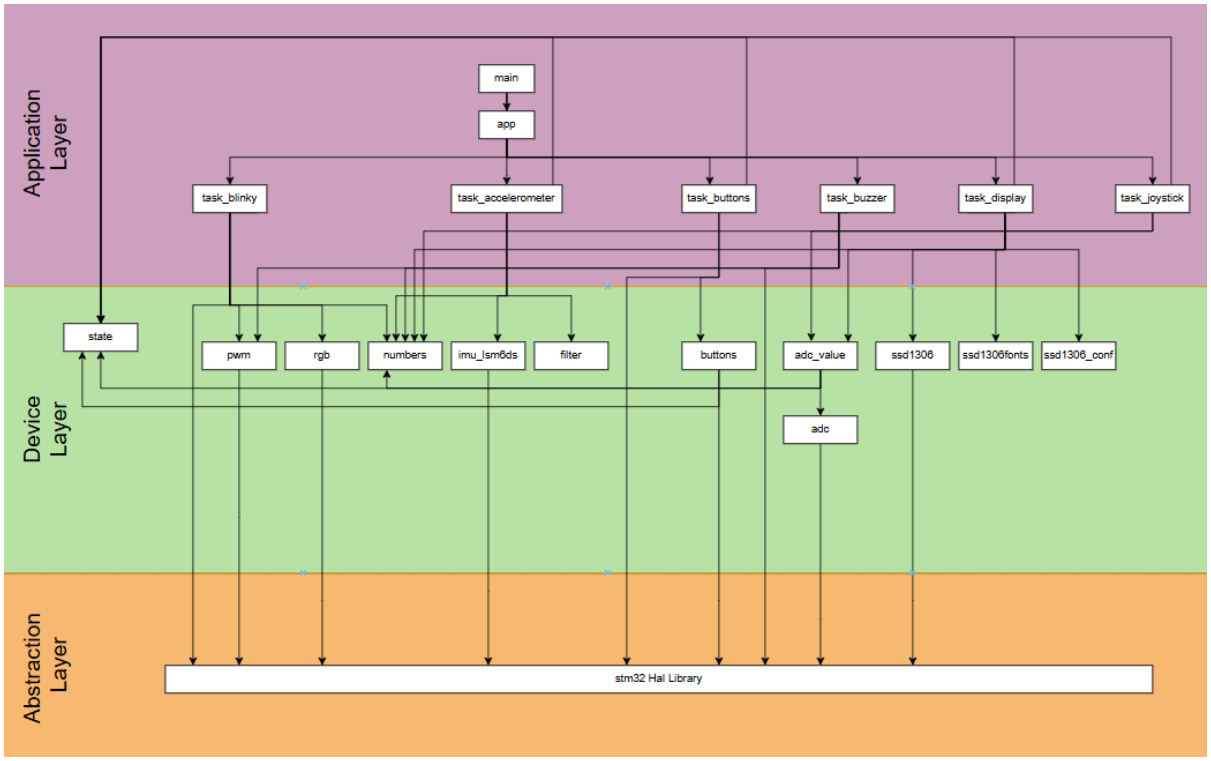

Figure 1: Dependency diagram of step counter program

Analysis of firmware operation

To ensure an accurate step detection a finite impulse response filter (FIR) was implemented to deal with the undesirable stochastic noise whilst still enabling step detection. Careful consideration of the FIR buffer size and sampling rates was essential as buffer sizes too large or sampling rates too low caused latency in the step detection response. The FIR utilised a circular buffer of size 32 that was sampled at a frequency of 100 Hz to attenuate noise frequencies from the raw accelerometer data these values were found using equation 1.

$$\frac{f_s}{M} = f_{noise} \qquad (1)$$

Threshold with hysteresis was implemented for the step detection logic where, a rearranged version of the sample variance (Nvar) was used to avoid microcontroller memory overhead, see equation 2 below. Due to the magnitude of the nvar values these were read as uint64_t and bit shifted to avoid overflow issues.

$$N\sigma^2 = \sum(x_i - \mu)^2 \qquad (2)$$

From step testing a value of $1 * 10^{15}$ for the step detection threshold was determined, this provided the required sensitivity for reliable step detection.

To allow for enough time for Nvar value to decay after the initial detection of a step, the hysteresis margin was set to $2.8 * 10^9$ which provided the necessary delay by allowing the average magnitude to dip below this margin before counting again to avoid double counting of the current step. A graph of the magnitude response to three steps is shown below.

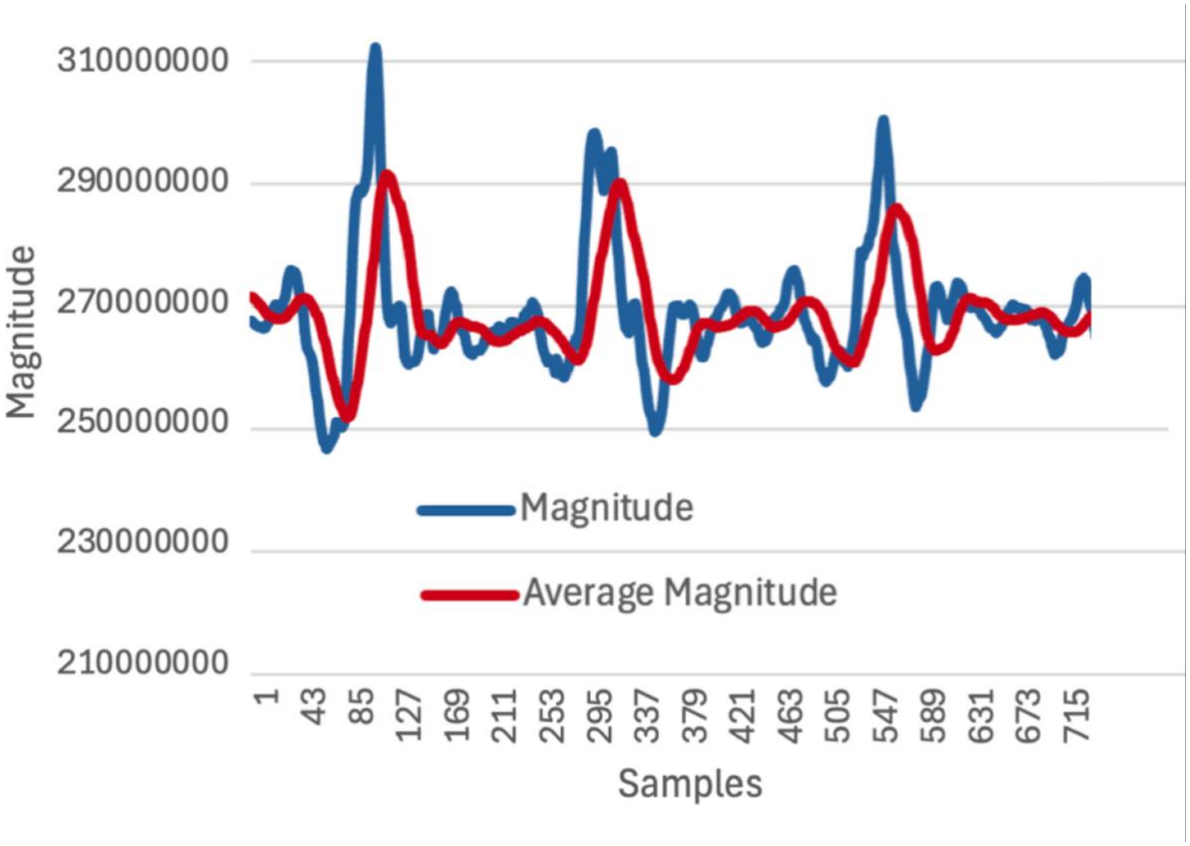

Figure 2: Graph of accelerometer magnitudes showing 3 steps

A step delay of 500ms provided further robustness and allowed cycling of the threshold and detect functions at a frequency, similar to human walking speed. Careful consideration of the step delay was considered as a time delay that was to short resulted in double counting and a time delay that was to long resulted in missing steps.

The program ran off a polling based scheduler where each task ran off different frequency's these were chosen based on the task being performed and proved effective in our program. Thorough testing was carried out and all the tasks performed without any missed functionality so specific profiling was not necessary.

Conclusions

A program was developed which successfully tracked a user's steps and provided the user with some basic analytics and inputs surrounding this data. Clean, well-structured code was essential to manage a project of this scale and having good modularisation of the program allowed for easy debugging, maintenance and the addition of features. The modularisation of the program proved to be effective and low coupling and high cohesion was achieved with only a few minor areas identified that could be improved upon in the future. The step counter implemented a robust method of step detection. A FIR filter was used to attenuate unwanted noise and a sample variance was implemented to provide a threshold in order to reliably detect steps. The program operated on a polling-based scheduler assigning appropriate frequency's to each task so they could carry out their functions. Overall this project provided a valuable and insightful experience in embedded system development resulting in a functional step counter.