Fan Controller Design Document

William Powell

Integrated Engineering (EE30186)

Word Count: 2725

Referencing Type: IEEE

November 2022

Contents

1 Product Description **1**
1.1 Overview . . . 1
1.2 Requirements . . . 1
1.3 Hardware . . . 1
1.4 Firmware . . . 1

2 User Experience **2**

3 Design **2**
3.1 Hardware Configuration . . . 2

4 Firmware **3**
4.1 Structure . . . 4
4.2 Fan Control . . . 5
4.2.1 Evaluation of Closed Loop Control Systems . . . 5
4.2.2 Implementation . . . 6
4.3 Rotary Encoder . . . 8
4.4 Additional Features . . . 8
4.4.1 LCD Display . . . 8
4.4.2 Data Logger . . . 8
4.4.3 Playing Music . . . 9

5 Design Constraints and Improvements **9**

6 Appendix **11**
6.1 User Interface Menu Navigation . . . 11
6.2 Equivalent Circuitry for 3 and 4 Wire Fans . . . 11

1 Product Description

1.1 Overview

Fan controllers are used in a variety of applications, from PC and server cooling systems to air-conditioning in vehicles. With the surge in consumer gaming there has been a demand for custom high-performance cooling fans, where users can control fan speed to maximise performance, whilst reducing noise. Temperature regulation requires a closed-loop system where fan speed is set according to the desired temperature of the CPU, however users may also want the ability to override this and set a desired speed, to limit noise for example.

1.2 Requirements

For this design, input will be given to the user through a user interface to select the desired speed of the fan and read the current speed. A control method will be used to reduce fan-speed error. In addition to these features, customers may also want to since firmware has no additional cost besides the research and development.

The fan controller should:

1. Maintain a desired fan speed with minimal error.
2. Run smoothly at low speeds.
3. Be intuitive to use, without requiring extensive instructions.
4. Minimise acoustic noise so as not to distract the user.
5. Be at a low cost to the consumer.
6. Stand out from other fan controllers on the market, by use of additional features.

1.3 Hardware

The microcontroller used is an STM32 with 32-bit 48MHz ARM processor. It is widely used in industry due to its low cost to performance ratio, high versatility and extensive documentation. It has 51 GPIO, with external interrupt support, to provide connection to peripherals communicating with SPI, I2C or other digital devices. It also contains a 12-bit, 16 Channel ADC to read analogue circuitry [1]. The peripherals used should be low cost and are detailed in section 3.1.

1.4 Firmware

MBED-OS has been chosen as it offers a suitable API to reduce the amount of Low-Level configuration required. This abstracts the requirements for using HAL, DMA and other low-level operations with the bare-metal implementation. It provides a well documented API [2] and official and open-sourced firmware and libraries.

All firmware will be written in C++, due to its high performance and ability to use object-orientation over C.

2 User Experience

The priority of the user experience is to be intuitive and easy to use. A rotary encoder and onboard push button will be used to interact with the fan controller. The LCD display will be used to go through the menu; select different modes and features; and display useful information, such as the fan speed.

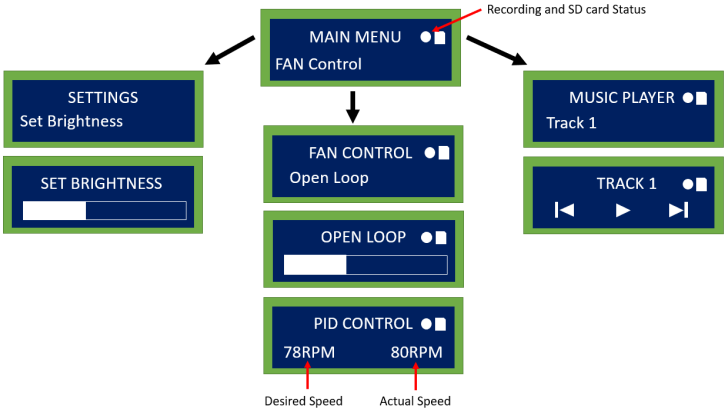

Figure 1: LCD User Interface

The user will utilise the encoder to scroll through the various the menu options and then select using the push button integrated onto the Nucleo board. A long press of the button will take the user to the previous menu.

See appendix 6.1 for the menu navigation options.

3 Design

3.1 Hardware Configuration

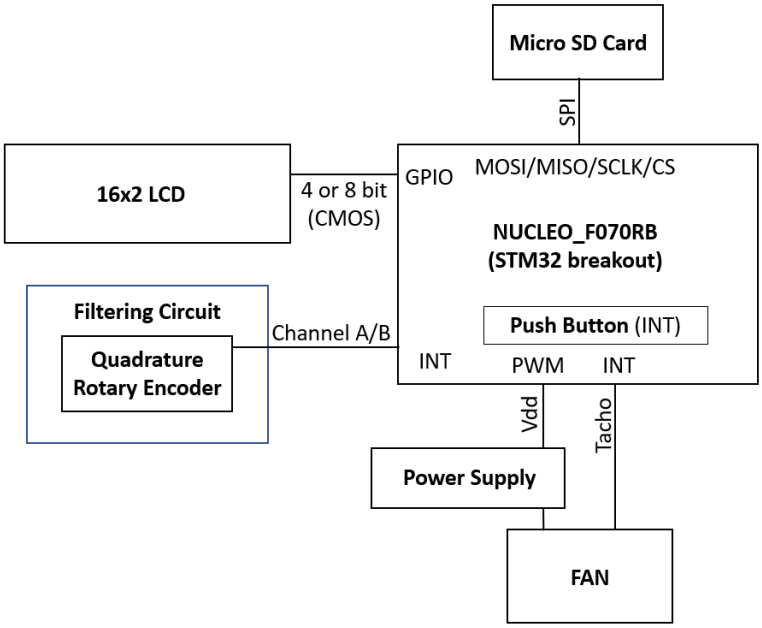

Figure 2: Circuit representation of the different peripherals that will be used and their subsequent connections on the microcontroller. Where INT and PWM represents a connection to a GPIO pin that has interrupt or PWM capabilities respectively.

4 Firmware

To ensure it will be able to be maintained and improved over time, the firmware will be written using object orientated programming (OOP). This is industry standard for C++ applications and have 3 key features:

1. **Modularity and Encapsulation** - each class aims to be as self-dependent and encapsulated as possible. This means that new improvements and features can be added or removed with little change to the existing system. It also ensures easy substitution of components and their subsequent drivers.
2. **Inheritance and Polymorphism** - by inheriting classes, methods and members variables can be reused, making the code less repetitive and bloated. Any subtle changes to the base class can be overwritten or appended to, allowing for simpler improvements and maintenance.
3. **Abstraction** - by using OOP, the level of abstraction can be much higher, reducing complexity. Higher level classes do not need to interact with all the methods and members of the lower level class. By 'hiding' methods, by setting them as private, it indicates to the programmer that these methods will not be used externally, thus improving readability and allowing for easier interaction with other classes.

Interface classes, or abstract classes, will be used to detach hardware dependencies, thus allowing for quick substitution of drivers. In this design, the peripheral drivers, such as the Encoder and LCD drivers, will inherit the interface (base) class and provide the implementation. However when abstracted, the interface will be called instead. The interface class thus acts as a template to adhere to, and in the event of component swapping, a new driver can be easily integrated into the current system without the concern of dependencies.

4.1 Structure

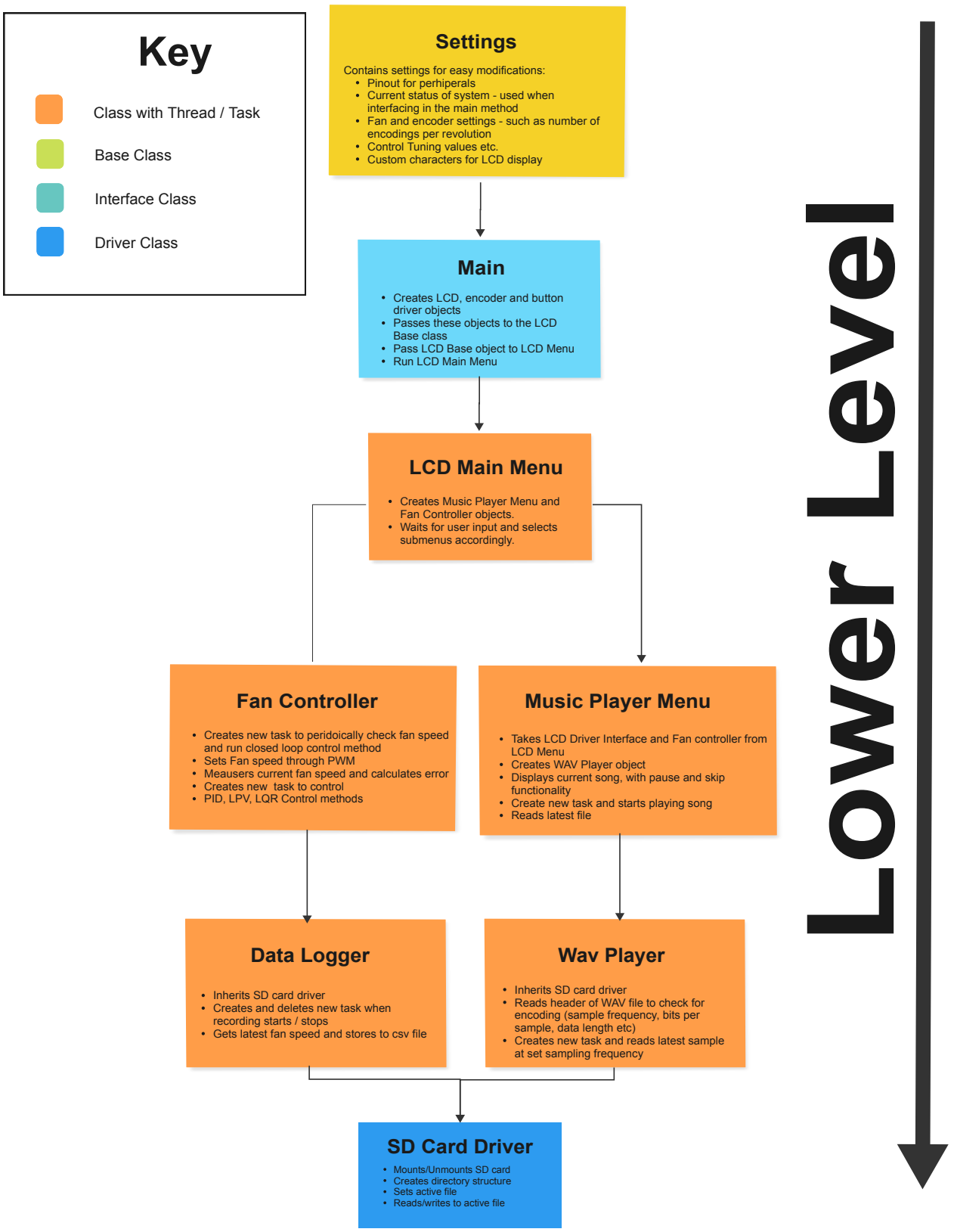

Figure 3: Structural Diagram illustrating the classes that will be used, their functionality and the layer of abstraction to higher level classes.

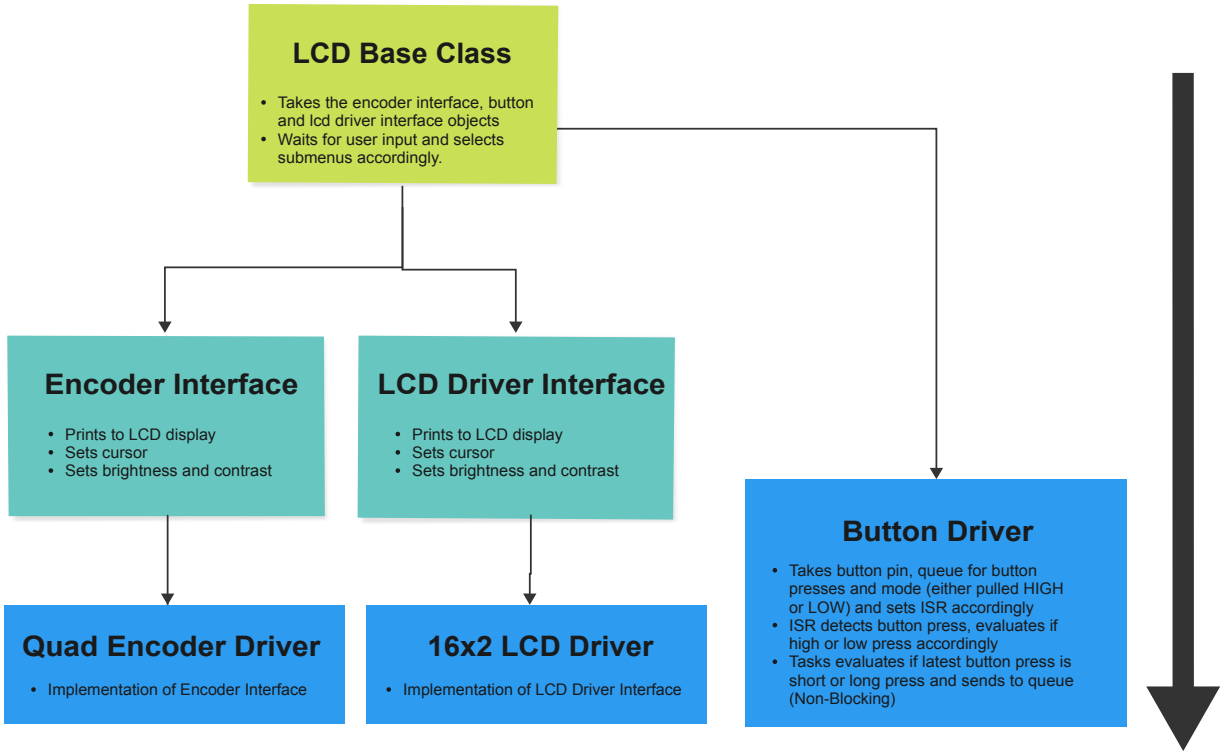

Figure 4: Structural diagram illustrating the composition of LCD Base Class.

Since the classes will distribute tasks into individual event loops, the firmware will use MBED-OS's Real Time Operating System (RTOS) to create and delete tasks; set priorities for these tasks; and use queue's when one thread interacts with another. This will ensure events occur without delay as long as it is the highest priority task requiring CPU time.

Events that require RTOS implementation include:

1. Writing to the SD Card at a fixed rate when data logger is active
2. Playing music at a fixed sampling frequency
3. Periodically calculating closed loop control. Increasing this task's priority level will improve the control system's performance
4. Updating the LCD when the user provides new inputs, such as the encoder and button.

4.2 Fan Control

4.2.1 Evaluation of Closed Loop Control Systems

To accurately maintain a desired fan speed, closed-loop control must be used.

First, it is important to identify the order of the fan controller system. With the assumption that the duty cycle of the PWM is proportional to voltage, equation for the fan speed can be derived [4]:

$$\text{PWM Duty cycle } \alpha \text{ Input Voltage } = iR + L\frac{di}{dt} + C_e\frac{d\theta}{dt} \tag{1}$$

where r is the resistance in the coil windings, L is the inductance of the motor, C_e is the EMF constant and θ is the rotational velocity of the motor.

Taking into account the resistive forces on the motor:

$$C_m i = J\frac{d^2\theta}{dt^2} + f\frac{d\theta}{dt} \tag{2}$$

where C_m is the electromagnetic torque constant, J is the moment of inertia of the fan and f is the viscous friction coefficient of the fan motor.

From this, a state space equation for the system can be obtained:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -f/J & C_m/J \\ 0 & -C_e/L & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix} u \tag{3}$$

Where x_1 is θ, x_2 is $\dot{\theta}$ and x_3 is i.

From this state equation, it is evident that the state is a first order system, thus control systems such as linear quadratic Gaussian (LQG) and . The most popular control systems that are appropriate for first order systems are discussed below.

Some fans have inbuilt analogue circuitry where the speed of the fan is fed back to the input power. However, these designs are more expensive and are are not required when a microcontroller is available.

PID Control is used in approximately 95% of industrial closed-loop controllers [5]. The error of the output is multiplied by a proportional, integral and derivative term and summated with the input. The key benefits of PID control are:

1. When correctly tuned, zero steady state error can be achieved due to the integral term.
2. Moderate peak overshoot, moderate stability.
3. Only requires a measurement device at the output and not internal states or disturbances.
4. Relatively simple implementation and high computational efficiency.

The main drawback of PID control is the considerable time to tune the device. Additionally, if there is a hardware change, such as a using a replacement fan, these terms may have to be re-tuned and the firmware updated.

LQR Control or LQR, is another widely used control system for first order systems. LQR uses a quadratic cost function to minimise the electrical power of error correction or large overshoots at the cost of slow error correction. This is implemented using a cost matrix that provides weightings for these cost parameters. For the fan controller, overshoot or power consumption is not an issue and would be of increasing complexity to PID. LQR can provide superior stability than PID for high-order systems, however for the first order system of the fan this would not be the case [6].

Feedforward Control requires the control system to have a measurable disturbance, which would then be compensated for before an output is produced. However, for the fan control system there is little disturbance, since there is no variation in resistive torque that would impede the output's variation. This approach is therefore not beneficial.

4.2.2 Implementation

The fan houses a brushless DC motor [3]. A PWM output on the STM32 will be connected through a transistor to Vdd of the fan, supplying power to the fan proportional to the PWM's duty cycle. The fan has a tachometer output that emits two pulses per revolution. This will be connected to an interrupt-enabled pin on the Nucleoboard and attached to an ISR (Interrupt Service Routine).

There are three methods to determine fan speed in the ISR:

1. Measuring the number of pulses in a given time interval. The velocity resolution is given by

$$V_{res} = \frac{\theta_{res}}{t_{interval}} \tag{4}$$

   Where θ_{res} is the rotational resolution. Since this method is independent of velocity, it is effective in measuring high velocity. However, at low speeds, this method is inaccurate due to a small number of integer pulses counted - the fractional element between pulses cannot be determined.
2. Measuring the time taken between individual pulses. The velocity resolution is given by

$$V_{res} = \frac{v^2 t_{res}}{\theta_{res}} \tag{5}$$

   where t_{res} is the time resolution of the clock. This is accurate at low speeds, however at high speeds the resolution becomes increasingly less accurate. This is due to it's dependency on clock resolution and the squared relationship with velocity.
3. Use a more sophisticated complementary filtering technique that combines both methods listed above.

The maximum speed of the motor is approximately 2300rpm. With two pulses per revolution, the rotational resolution is ($\theta_res = \Pi rad$). This results in a minimum of 208μs between pulses. Since the STM32 has a timer function to output to the nearest microsecond, high precision will still be obtained using method 2. However, to further improve the resolution, method 3. will be used. This uses method 2. to measure the time between individual pulses and approximates the fractional number of pulses in the time interval using method 1. Increasing the time interval will improve the resolution at the cost of a slower response to change. This will be optimised through experimentation for this system.

The fan controller will have an open loop control mode, where the encoder will be used to set the speed of the fan as a percentage, this will then set the PWM inverter's duty cycle.

Due to its benefits listed above 4.2.1, PID will be used as the closed loop control method. In the FanController class, a new thread will be created and the PID method will be called periodically, dependent on its priority level over other tasks. In this method, the error will be calculated as the difference between the desired input, set by the user, and the average speed since last polled, calculated in the ISR of the tachometer. This error will then be fed into the PID equation

$$u(t) = k_p e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt} \tag{6}$$

where $k_p, k_i and k_d$ are the gains for the proportional, integral and derivative terms respectively.

There are sophisticated PID methods however, this fan controller does not require critical control such as specific criterion for overshoots and settling time. Hence, PID values will be tuned using the Zeigler and Nichols method [8]. This method systematically tunes each value, starting with the proportional gain until the output shows consistent, stable oscillations to determine the ultimate gain, k_u and the oscillating time period, T_u. The table [**?**] is then used to calculate $\mathrm{k}_p, k_i and k_d$.

To implement this, a tuning method in FanController will be written to increment through proportional gain values. For each value, a step impulse will be set to instantaneously increase the desired speed from 0rpm to 2000rpm, whilst the data logger captures the actual speed change until steady state is reached. The oscillations will then be viewed from the log file to determine k_u and T_u, and thus the optimal gains.

4.3 Rotary Encoder

The encoder used is a 2 channel Rotary Encoder, see Datasheet [9]. A RC low-pass filter ($f_c = 1.6kHz$) stabilises the channel outputs to ensure there is no bouncing effect. When the encoder is rotated by one tick, the following output is obtained.

Figure 5: Logic Analyser Recording for the Encoder, when rotated clockwise and anticlockwise (top and bottom figure respectively) by one tick. Note when rotating clockwise, Channel B is 90°out of phase to Channel A. The opposite is the case in the anticlockwise direction. This follows Gray encoding where a subsequent value only changes via one bit.

To obtain maximum precision, x4 encoding shall be used, where both the rising and falling edges of both channels will be observed. This requires that the two encoder channels are connected to an interrupt pin. The corresponding ISR will be aware of the previous state of the encoder and thus when a bit changes on either channel A or B, the direction will be determined. This will then result in the incrementation or decrementation of the encoder positioning counter. Invalid states, such as when neither bits or both bits have changed since last ISR, will not be accounted for. An additional low-pass filter will also be implemented via code, ignoring any rise or fall cases that occurred in a given time-frame since the last rise/fall.

4.4 Additional Features

4.4.1 LCD Display

The display is a 16x2 LCD, communicating in either 4 or 8 bit mode to the CMOS registers. 8 bit mode is quicker to update, however uses more GPIO pins. For this application, refresh rate is not critical and using 8 GPIO pins would be excessive and limit the number of other peripherals that could be added.

Due to the popularity of this LCD display, an open sourced mbed-os based driver will be used.

4.4.2 Data Logger

Storing data of the fan speed is a useful feature for monitoring the performance of different control systems and evaluating their performance. It can also be used to monitor fan usage, such as how often it turns on and at what time, for cooling etc.

To achieve this the SD card will communicate on SPI, and the file will be written to using the SD card driver. Writing and reading uses the FAT file system, which works on a low level, using pointers to iterate through the memory addresses of the file. The fan speed will be polled at a set rate on a thread and stored to a CSV file.

4.4.3 Playing Music

Coil whine is an undesired acoustic noise that is electromagnetically induced. This can either be through an electrical field, a magnetic field, or in the case of the brushless dc motor in the fan, both. When the shape of the electromagnetic forces along the airgap of the stator match the stator's structural mode, vibrations and acoustic noise are amplified significantly [11].

This effect has led to companies designing fans at a frequency above the audible range, such as the most recent fan controllers from Analog Devices [7].

However this effect can also be utilised to create tones, and thus music. By experimenting with the PWM frequency that drives the fan, maximum resonance and thus volume can be achieved on a 'Silent' PC fan.

After the optimal PWM frequency is set, music can be read from the SD card using SPI. This is done in the WAVPlayer class that inherits the SD card driver. Using WAV files do not have compression and are therefore easier to read, with the header of the file detailing the sampling frequency, bit depth etc. These amplitudes can then be periodically outputted to the fan via PWM at the sampling frequency of the music. The amplitude will be mapped to a percentage, i.e. for 8 bit audio, 255, the maximum value, will correspond to 100% duty cycle outputted.

5 Design Constraints and Improvements

1. **Erroneous tachometer output at low frequency PWM** - when the PWM is at its low cycle, it is possible that the tachometer's Hall sensor will turn off, resulting in fewer tachometer pulses and thus an inaccurate fan speed measurement.

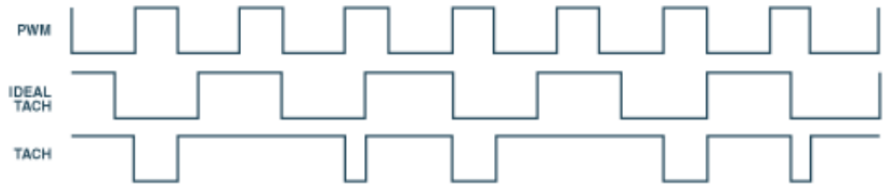

Figure 6: When the PWM frequency is low enough to turn periodically turn off the hall sensor, the tachometer will only output when the PWM is high and the fan rotates passed the Hall sensor [7].

Powering the fan with PWM relies that there is sufficient capacitance in the Hall sensor and wire to provide the a saturated value DC voltage capable of keeping the tachometer on till the next high pulse. There are several options to solve this: Increase the PWM frequency to the point where the Hall sensor obtains a saturated supply voltage; add an external capacitor after the transistor circuitry to act as a low-pass filter, smoothing out the square wave to a saturated supply; use pulse stretching (temporarily providing 100% duty cycle) to get an accurate tachometer output; or use a 4 wire fan to provide constant voltage to the Hall sensor, see Appendix .

2. **Computational performance** Performing additional tasks such as data logging, will put strain on critical tasks such as fan speed. Using MBED-OS's RTOS such as threads to set the priority levels and scheduling will ensure sufficient computational time for tasks is given. However, this will be at the cost of less prioritised tasks which will be given little time.
3. **Complexities in threading** Using MBED-OS's RTOS does result in added complexities, and often results in increased debugging time. Reducing the number of features that require threads will also reduce development time.

References

[1] *NUCLEO-F070RB Board*. Available at: https://os.mbed.com/platforms/ST-Nucleo-F070RB/ [Accessed 21 November 2022]

[2] *MBED-OS API*. Available at: https://os.mbed.com/docs/mbed-os/v6.15/introduction/index.html [Accessed 21 November 2022]

[3] *Brushless DC Fan*. Available at: https://www.farnell.com/datasheets/2630333.pdf [Accessed 22 November 2022]

[4] Zhang, D. *Control Engineering EE30041* University of Bath, 2022. Available at: https://www.analog.com/en/analog-dialogue/articles/how-to-control-fan-speed.html [Accessed 23 November 2022]

[5] Johan et Al, *PID Control*, CalTech, 2002. Available at: https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf [Accessed 23 November 2022]

[6] *Feedback Control with Linear Quadratic Regulator (LQR)*, Robotik. Available at: https://rrwiyatn.github.io/blog/robotik/2020/07/19/lqr.html [Accessed 23 November 2022]

[7] *3 and 4 Wire Fans*. Available at: https://www.analog.com/en/analog-dialogue/articles/how-to-control-fan-speed.html [Accessed 23 November 2022]

[8] Ellis, G. *Ziegler Nichols Method* Science Direct, 2012. Available at: https://www.sciencedirect.com/topics/computer-science/ziegler-nichols-method [Accessed 23 November 2022]

[9] *2 Channel Encoder*. Available at: https://www.bourns.com/docs/product-datasheets/pec11r.pdf [Accessed 22 November 2022]

[10] *16x2 LCD*. Available at: https://store.comet.bg/download-file.php?id=13475 [Accessed 22 November 2022]

[11] EOYMS. *Magnetic noise in electric machines*. Available at: https://e-nvh.eomys.com/magnetic-noise-and-vibrations-in-electrical-machines/ [Accessed 23 November 2022]

6 Appendix

6.1 User Interface Menu Navigation

1. Fan Control
    - 1.1 Open Loop
    - 1.2 Closed Loop (PID)
2. Start / Stop Data Logging
3. Music Player
    - 3.1 Track Player
4. Settings
    - 4.1 Set Brightness
    - 4.2 Set Contrast
    - 4.3 Mount SD Card
    - 4.4 Unmount SD Card

6.2 Equivalent Circuitry for 3 and 4 Wire Fans

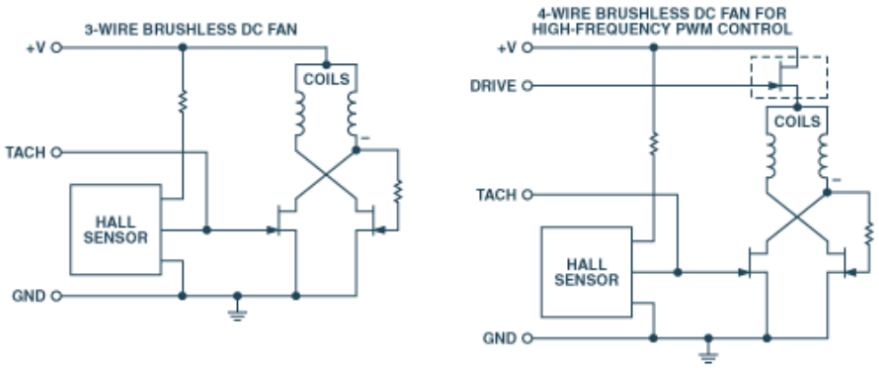

Figure 7: 4 wire Fan equivalent circuit which ensures the tachometer's Hall sensor has a continuous voltage supply rather than a square wave supply from PWM. [7]