NUVATION ENGINEERING
ELECTRONIC PRODUCT DESIGN AND MANUFACTURING

SILICON VALLEY HEADQUARTERS
151 GIBRALTAR CT
SUNNYVALE, CA 94089 USA
408.228.5580

WATERLOO DESIGN CENTER
332 MARSLAND DR., SUITE 200
WATERLOO, ON N2J 3Z1 CANADA
519.746.2304

WWW.NUVATION.COM

WNG01: ACTUATOR CONTROLS Firmware Software Design Description (SDD)

For Michael Malcolm

Last Updated: *September 11, 2019,* ***Revision*** *1.7*

Approvals for Rev 1.7

NUVATION ENGINEERING

Michael Malcolm	Nuvation
_____	_____
Name	Name
_____	_____
Signed	Signed
_____	_____
Title	Title
_____	_____
Date	Date

Revision History

Revision	Date	Description	By
0.1	2017 – 12 – 15	Initial Draft	R. Chung - Nuvation
1.0	2018 – 01 – 29	Updates from client feedback	R. Chung - Nuvation
1.1	2018 – 03 – 26	Added system function overview table	R. Chung - Nuvation
1.2	2018 – 06 – 26	Updated document based on current state of the firmware	R. Chung - Nuvation
1.3	2018 – 12 – 03	Added value defines for de-energized coils Added defines for actuator error and output control parameters Added defines for MCU UID Added temperature ADC filter window, control loop interval, and flight registration parameters to the configuration sector Added new events to support changes	J. Teng - Nuvation
1.4	2019 – 01 – 31	Added WOW logging support	J. Teng – Nuvation
1.5	2019 – 05 – 22	Added Pre-bias defines	J. Teng – Nuvation
1.6	2019 – 06 – 18	Added offset defines	J. Teng – Nuvation
1.7	2019 – 09 – 10	Update description of control algorithm, add references to WOW in actuator responsibilities	J. Chinnick – Nuvation

Table of Contents

List of Figures

List of Tables

Glossary

Term	Description
ADC	Analog to Digital Converter
API	Application Programming Interface
BLE	Bluetooth Low Energy
CDC	Communication Device Class
DMA	Direct Memory Access
GPIO	General Purpose Input Output
FW	Firmware
HAL	Hardware Abstraction Layer
HDD	Hardware Design Description
ISR	Interrupt Service Routine
OS	Operating System
OTA	Over-The-Air
PI	Proportional-Integral
QSPI	Quad (4 lane) Serial Peripheral Interface
RTOS	Real Time Operating System
SDD	Software Design Description
TBO	Time Before Overhaul
TLV	Time-Length-Value
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VM	Virtual Machine
WOW	Weight-On-Wheels

1 Introduction

The North American P51 Mustang has two thermostatic actuators that open and close the air outlet doors of the coolant and oil radiators. Adjusting the positions of these doors controls the amount of air flowing through the radiator and thus the operating temperatures of the engine coolant and oil.

Each thermostatic actuator is operated automatically using an electro-mechanical control system which is comprised of relays, switches and a diatstat. The diastat is used to sense the temperature of the coolant or oil. It is becoming increasingly difficult to find diastats that still function properly.

1.1 Purpose

The purpose of this project is to modify the existing thermostatic actuators with a digital controller that replaces the existing electro-mechanical controller. This document describes the architecture and design of the firmware for this digital controller. It provides an overview of the firmware implementation, the control algorithm used to control the coolant and oil temperatures, and the communication protocol to an iOS application. The iOS application will be referred to as the mobile application from here on in this document.

1.2 Scope

This document primarily describes the firmware on the controller itself. It does not describe any software design related to the mobile application. The mobile application design can be found in the Mobile SDD. In this document, the use of the term software refers to the firmware described in this document and does not refer to the mobile device software unless otherwise stated.

1.3 Assumptions

The following is a list of assumptions made about the system and its requirements during the writing of this document:

- No security provisions made over the BLE interface
- Coolant and oil actuator doors are assumed to be in the same position as the last recorded value on controller startup
- The coolant and oil cooling systems have negligible effects on each other
- Changes in the plant (the physical system in control terminology), are a lot slower (at least x10) than the update rate of the controller
- Hold and sample time in the control loop can be ignored because the temperature sensor device and controller react a lot faster (at least x10) than the changes in the plant
- DO-278C for Level D shall apply

1.4 Reference Documentation

The following documentations are referenced by this document:

- Digital Coolant and Oil Actuator Controller v.3
- STM32F412 advanced ARM®-based 32-bit MCUs RM0402
- STM32Cube USB Host library User Manual UM1720

1.5 Related Documentation

The following documentations are related to this document:

- WNG01_Mobile_SDD
- WNG01_HDD

2 System Overview

This section gives an overview of the system and highlights the role of the firmware in relation to other components. In Figure 1 below, the overall system diagram is given.

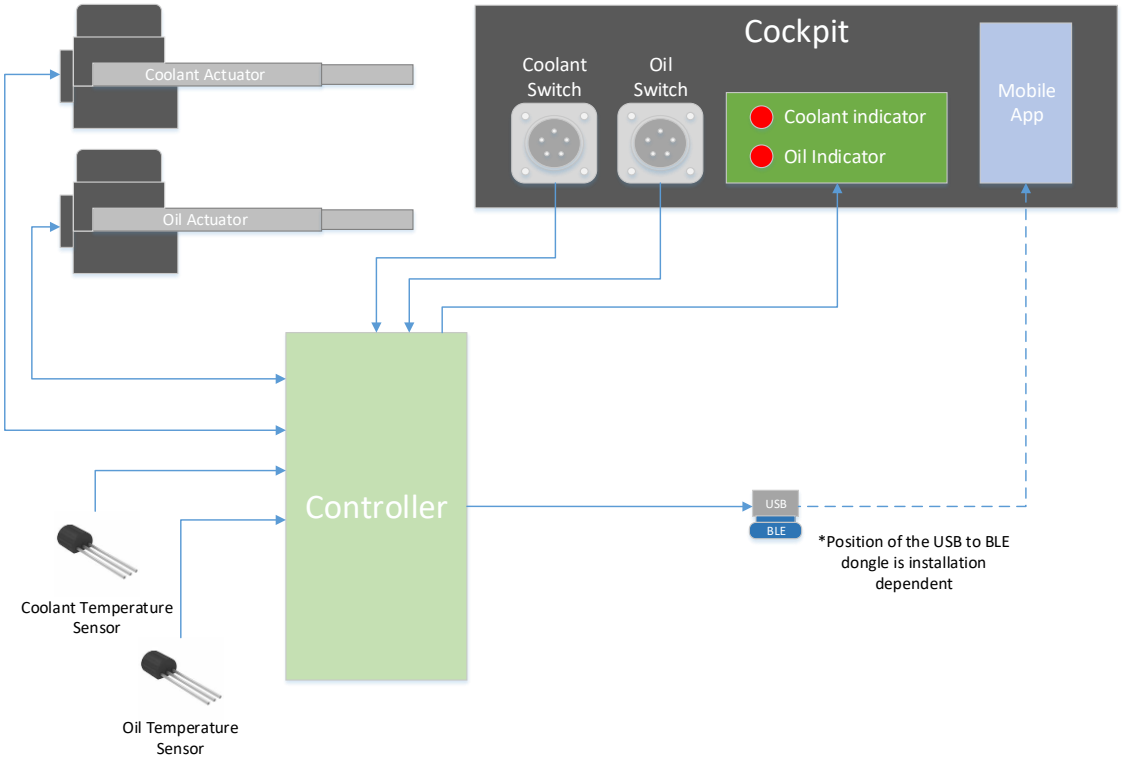

Figure 1: System diagram

There are two main digital components in the system: the controller and the mobile application. The controller is located in the hell hole of the P-51 fuselage and contains a microcontroller with the firmware described by this document. The mobile app lives on a mobile device located inside the cockpit. The mobile application is not required during flight; the controller manages the system control independently of the mobile application.

The controller controls the coolant and oil actuators depending on the coolant and oil temperatures. These values are measured by the coolant and oil temperature sensors. In addition, the controller interfaces with the coolant and oil control mode switches, light indicators, and a USB to BLE dongle.

The USB to BLE dongle enables the controller to communicate over Bluetooth to the mobile app for maintenance operations. The location of the USB to BLE dongle may differ from one installation to another.

2.1 Controller Overview

The controller comprises a microcontroller and an external flash storage used to store flight logs. Figure 2 below shows a more detailed view of the controller including the type of inputs and outputs connected to it.

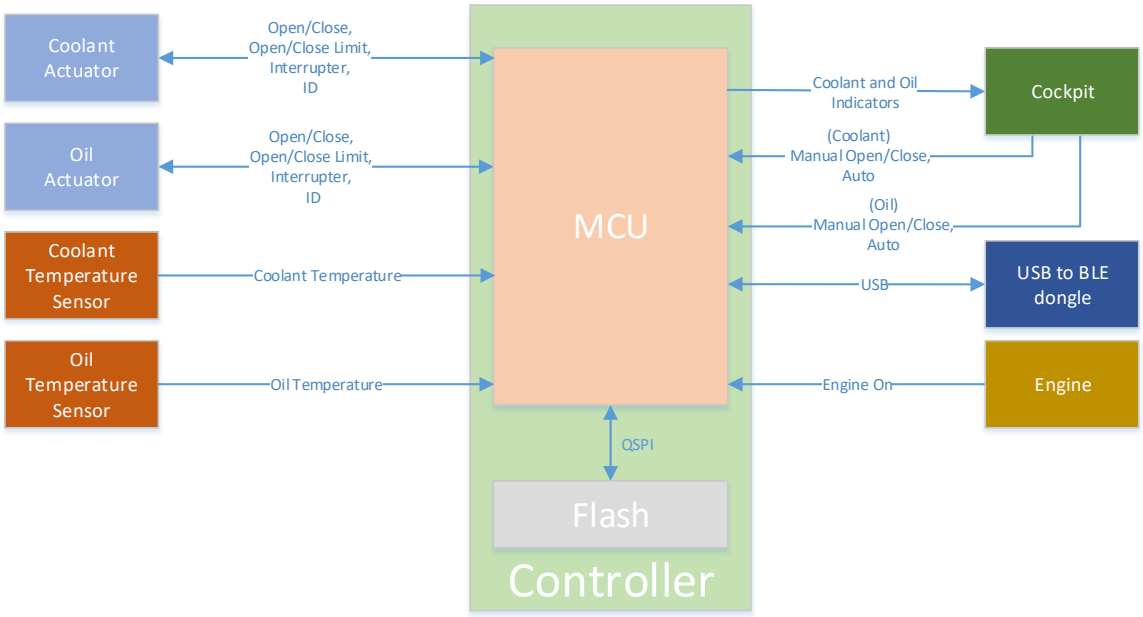

Figure 2: Control box overview

The microcontroller has interfaces to control the oil and coolant actuators and the temperature sensors. There is also a USB connection from the microcontroller which is connected to the USB to BLE dongle.

2.2 Hardware Components

The follow hardware components are used for the controller:

- STM32F412 processor
- Cypress Semiconductor S25FL256LAGMFN000 External Flash
- Bluegiga BLED112 BLE Dongle

2.3 System Function Overview

The functional overview of the system is shown below in Table 1.

System Components	Functions	Function Description
System Manager	Reads the engine on/off state	
	Reads the controller temperature	
	Reports a controller reset	
	Reports oil/coolant/system warnings	Powers the oil and coolant indicators if a warning has occurred
Oil Actuator	Determines if the actuator is connected	
	Monitors WOW sensor	Affects behavior of Auto mode
	Monitors oil temperature	
	Monitors actuator operation mode	Auto, manual neutral, manual close, manual open
	Monitors interrupter counts	
	Monitors open limit switch	
	Monitors close limit switch	
	Energizes the open coil to open the actuator doors	
	Energizes the close coil to close the actuator doors	
	Regulates the temperature of the oil	Uses a control loop to regulate the temperature to a set temperature point. Only active in “auto” mode
Coolant Actuator	Determines if the actuator is connected	
	Monitors WOW sensor	Affects behavior of Auto mode
	Monitors coolant temperature	
	Monitors actuator operation mode	Auto, manual neutral, manual close, manual open
	Monitors interrupter counts	
	Monitors open limit switch	
	Monitors close limit switch	
	Energizes the open coil to open the actuator doors	
	Energizes the close coil to close the actuator doors	
	Regulates the temperature of the oil	Uses a control loop to regulate the temperature to a set temperature point. Only active in “auto” mode
Logging	Log messages into non-volatile storage	Refer to Table 14 for messages logged.
Watchdog	Resets the system if any software service is unresponsive	
	Kicks the external HW watchdog	

Mobile Communication	Communicates system and actuator information to a mobile application	Communicates over BLE to the mobile application
	Performs maintenance operations	
Maintenance [1]	Calibrate oil actuator door	Executes calibration procedure described in Section 5.4
	Set oil temperature set point	
	Set oil actuator control loop parameters	
	Calibrate coolant actuator door	Executes calibration procedure described in Section 5.4
	Set coolant temperature set point	
	Set coolant actuator control loop parameters	
	Update firmware	
	Retrieve log data from non-volatile flash	

Table 1: System function overview

[1] Maintenance functions are accessible only in maintenance mode. Refer to Section 5.1 for more information.

3 Software Architecture

This section explains the software architecture of the controller firmware. Figure 3 below shows the high-level organization of the software.

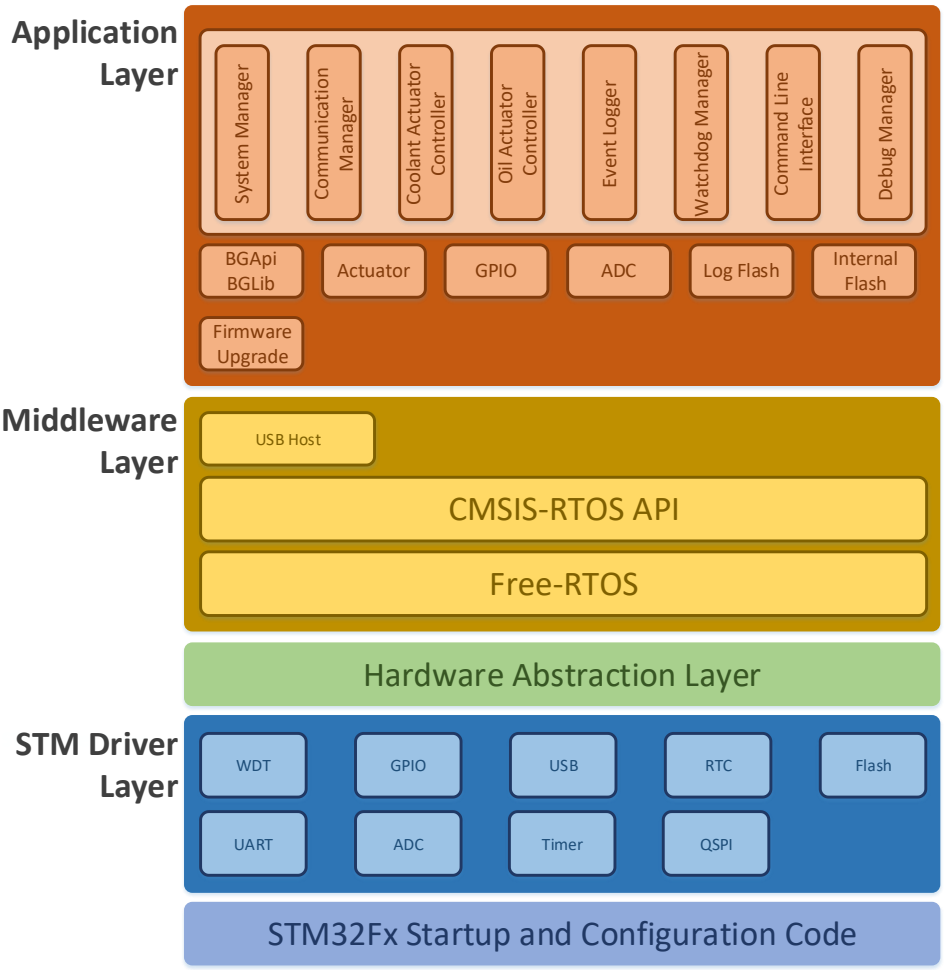

Figure 3: High-level software architecture stack

The majority of the software design effort is concentrated on the application layer. This is because the layers of software below it are auto-generated using a tool provided by STMicroelectronics called STM32CubeMX.

3.1 Application Layer

The application layer consists of multiple software services that handle the program logic of the software. Each service is responsible for managing a specific high-level function of the software. The services are:

- System Manager
- Communication Manager

- Coolant Actuator Controller
- Oil Actuator Controller
- Event Logger
- Watchdog Manager
- Command Line Interface
- Debug Manager

In terms of implementation, each software service is implemented as a separate thread managed by the underlying operating system. In addition to the mentioned software services, the application layer also contains software libraries that provide functionality convenient to a service. These libraries include interfaces to the following:

- Internal flash
- BlueGiga BGAPI™ and BGLib™ (BLE Dongle API) for the *Communication Manager*
- Actuators for the *Coolant Actuator Controller and Oil Actuator Controller*
- External flash for the *Event Logger*
- GPIO and ADC libraries for the *Actuator Library* and other services
- Firmware Upgrade to handle firmware upgrade related tasks

3.1.1 Software Services

The implementation of services is created as separate threads. Figure 5 below shows the hierarchy and creation of threads.

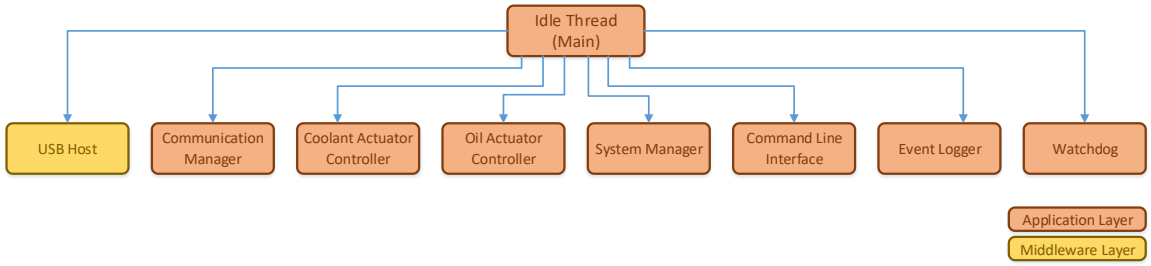

Figure 4: Thread hierarchy

These services mostly communicate with each other through messaging queues with most services exposing one messaging queue. Messaging queues are used so that communication from multiple services to a common service can be serialized. Event messages are placed into messaging queues to indicate to a specific service to act on an event. The only exception to this is the Watchdog Manager which does not have a messaging queue. This is further explained in Section 3.1.6. Figure 5 below shows the simplified communication between the services that are necessary for the system.

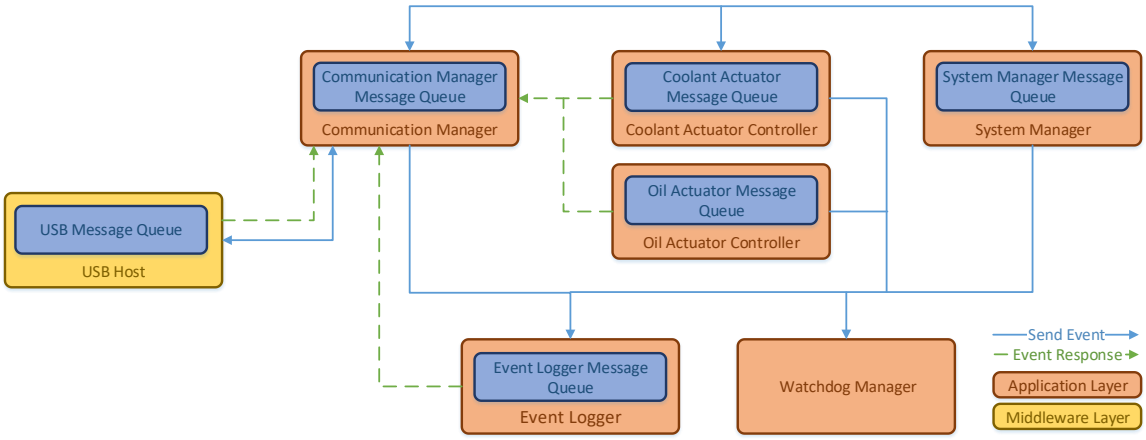

Figure 5: Inter-service communication (simplified)

Communication between services to initiate a request or respond to a request is done directly as opposed to through a central message dispatching service. This approach was chosen because there are only a small number of services in the system, as such adding an extra central messaging service would have overcomplicated the design.

The blue arrows in Figure 5 above indicate the direction of communication for requests from one service to another. These requests occur as event messages to the target service. The green arrows indicate the direction of a corresponding response message to a received event. The corresponding response messages are also sent as event message.

Event messages should not be missed by any service. If an event message is missed and therefore dropped, this indicates a software fault which will be logged for debugging purposes. In addition, the controller will reset to hopefully restore the controller to a working state. The only exception is the Debug Manager messaging queue. A dropped message is not considered a critical error to the rest of the system. However a provision for missed messages exists and is described in Section 0 .

The full communication diagram is shown below in Figure 6.

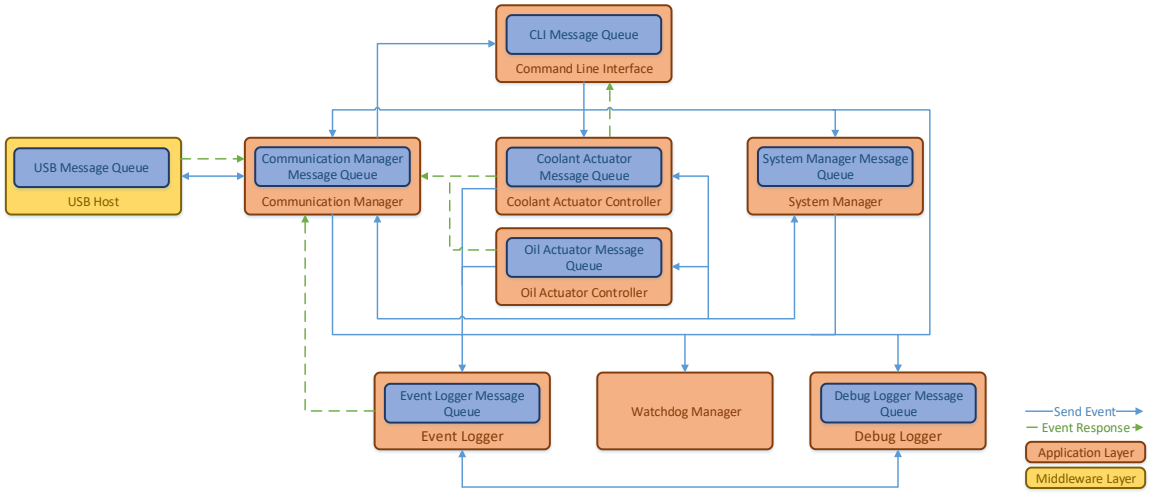

Figure 6: Inter-service communication (complete)

The Command Line Interface and the Debug Logger components are added in this figure. Both of these services are important during the development process and testing but do not affect the functionality of the controller.

3.1.2 System Manager

The role of the System Manager is to handle system state changes. The System Manager does the following:

- Reads the engine state
- Keeps track of the current flight number
- Reads the controller temperature
- Reports a controller reset
- Reports a controller malfunction
- Performs firmware upgrade operations
- Resets the system
- Checks in with the Watchdog Manager

3.1.2.1 Internal Flash Library

The internal flash library interfaces with the processors internal flash memory. All accesses to the internal flash are synchronized through the System Manager. The internal flash library is used to write configurations into flash as mentioned in Section 4.1 as well as to write new firmware.

3.1.2.2 Event Messages

The System Manager responds to the event messages listed in Table 2 below.

Event Name	Event Generation Source
ENGINE_ON	GPIO library
ENGINE_OFF	GPIO library
CLEAR_FLIGHT_NUM	Event Logger
BLE_CONNECTED	Communication Manager
BLE_DISCONNECTED	Communication Manager
SET_OIL_CTRL_PARAM	Oil Actuator Controller
SET_OIL_ALARM	Oil Actuator Controller
CLEAR_OIL_ALARM	Oil Actuator Controller
SET_COOLANT_CTRL_PARAM	Coolant Actuator Controller
SET_ COOLANT _ALARM	Coolant Actuator Controller
CLEAR_ COOLANT _ALARM	Coolant Actuator Controller
SET_EPOCH_TIME	Communication Manager
START_FW_TRANSFER	Communication Manager
FINISH_FW_TRANSFER	Communication Manager
ABORT_FW_TRANSFER	Communication Manager
FW_PACKET	Communication Manager
START_CTRL_TEMP_BLE_UPDATE	CLI
STOP_CTRL_TEMP_BLE_UPDATE	CLI

Table 2: System Manager event messages

3.1.3 Communication Manager

The role of the Communication Manager is to handle communication between the mobile application and the controller. The Communication Manager communicates with other services to act upon the commands received from the mobile application. Specifically, the Communication Manager does the following:

- Parse and generate USB packets
- Parse and generate Bluegiga API packets
- Parse and generate messages from the mobile device
- Request information from the Actuator Controller
- Initiate writing firmware upgrades to the internal flash
- Checks in with the Watchdog Manager

3.1.3.1 Bluegiga BGAPI™ and BGLib™

The BLED112 USB BLE dongle is used by the controller to communicate to the mobile application via Bluetooth. The BLED112 dongle implements a full Bluetooth stack removing the firmware from having to implement it. However, the firmware needs to communicate with the dongle through the Bluegiga API (BGAPI™) and library (BGLib™). The BGAPI™ provides an API to communicate with the dongle while the BGLib™ provides functionality to parse the information received from the dongle. The software for this is included in the SDK from Silicon Labs, the vendor for the Bluegiga chips. Version 1.6 of the SDK is used for this project.

3.1.3.2 Event Messages

The Communication Manager responds to the event messages listed in Table 3 below.

Event Name	Event Generation Source
UPDATE_SYSTEM_ENGINE_POWER	System Manager
UPDATE_SYSTEM _FLIGHT_NUM	System Manager
UPDATE_SYSTEM_CTRL_TEMP	System Manager
UPDATE_SYSTEM_ALARM	System Manager
UPDATE_OIL_OP_MODE	Oil Actuator Controller
UPDATE_OIL_TEMP	Oil Actuator Controller
UPDATE_OIL_DOOR_POS	Oil Actuator Controller
UPDATE_OIL_ALARM	Oil Actuator Controller
UPDATE_COOLANT_OP_MODE	Coolant Actuator Controller
UPDATE_COOLANT _TEMP	Coolant Actuator Controller
UPDATE_COOLANT _DOOR_POS	Coolant Actuator Controller
UPDATE_COOLANT _ALARM	Coolant Actuator Controller
UPDATE_CONFIG_OIL_SET_TEMP	Oil Actuator Controller
UPDATE_CONFIG_COOLANT_SET_TEMP	Coolant Actuator Controller
UPDATE_CONFIG_OIL_P_GAIN	Oil Actuator Controller
UPDATE_CONFIG_OIL_I_GAIN	Oil Actuator Controller
UPDATE_CONFIG_OIL_D_GAIN	Oil Actuator Controller
UPDATE_CONFIG_OIL_HYSTERESIS	Oil Actuator Controller
UPDATE_CONFIG_COOLANT_P_GAIN	Coolant Actuator Controller
UPDATE_CONFIG_COOLANT _I_GAIN	Coolant Actuator Controller
UPDATE_CONFIG_COOLANT _D_GAIN	Coolant Actuator Controller
UPDATE_CONFIG_COOLANT _HYSTERESIS	Coolant Actuator Controller
UPDATE_CONFIG_CTRL_LOOP_INTERVAL	Oil/Coolant Actuator Controller
UPDATE_FW_VERSION	Communication Manager
READ_SYSTEM_ENGINE_POWER	CLI
READ_SYSTEM _FLIGHT_NUM	CLI
READ _SYSTEM_CTRL_TEMP	CLI
READ _SYSTEM_ALARM	CLI
READ _OIL_OP_MODE	CLI
READ _OIL_TEMP	CLI
READ _OIL_DOOR_POS	CLI
READ _OIL_ALARM	CLI
READ _COOLANT_OP_MODE	CLI
READ _COOLANT _TEMP	CLI
READ _COOLANT _DOOR_POS	CLI
READ _COOLANT _ALARM	CLI
READ _CONFIG_OIL_SET_TEMP	CLI
READ _CONFIG_COOLANT_SET_TEMP	CLI
READ _CONFIG_OIL_P_GAIN	CLI
READ _CONFIG_OIL_I_GAIN	CLI
READ _CONFIG_OIL_D_GAIN	CLI
READ _CONFIG_OIL_HYSTERESIS	CLI
READ _CONFIG_COOLANT_P_GAIN	CLI
READ _CONFIG_COOLANT _I_GAIN	CLI
READ _CONFIG_COOLANT _D_GAIN	CLI
READ _CONFIG_COOLANT _HYSTERESIS	CLI
READ_CONFIG_CTRL_LOOP_INTERVAL	Oil/Coolant Actuator Controller
READ _FW_VERSION	CLI
BLE_CONNECT	Communication Manager
BLE_DISCONNECT	Communication Manager

Table 3: Communication Manager event messages

3.1.4 Coolant and Oil Actuator Controller

The Coolant Actuator Controller and the Oil Actuator Controller are responsible for interfacing with the coolant and oil actuators respectively. The two services are identical in the functionality and the event messages they expose. The only difference is that one is controlling the coolant actuator while the other is controlling the oil actuator. The two services are not dependent on each other and do not communicate with each other. Each actuator controller service performs the following:

- Determines if the actuator is connected
- Determines if WOW is asserted
- Reads the coolant/oil temperature
- Reads the actuator operation mode
- Responds to the open and close limit switches and interrupter when triggered
- Calibrates the position of the actuator door
- Keeps track of the position of the coolant door
- Reports an actuator malfunction
- Writes temperature and operation events into the logs
- Responds to request for actuator information
- Checks in with the Watchdog Manager

3.1.4.1 Actuator Library

The Actuator Library provides an interface in the form of function handles for the Coolant and Oil Actuator Controller services to implement. These function handles are called by the Actuator Library when there is a change in the physical actuators. Table 4 below lists the interface provided by the Actuator Library.

Function Handle Name	Description
ActuatorConnectHandle	Called when an actuator is connected
AcutatorDisconnectHandle	Called when an actuator is disconnected
ActuatorOperationModeChangeHandle	Called when there is a change in the operation mode of the actuator (manual neutral, manual open, manual close, auto)
ActuatorDoorActuationHandle	Called when the open/close coil is energized
ActuatorInterrupterTriggerHandle	Called when the interrupter is triggered
ActuatorLimitTriggerHandle	Called when a limit switch is hit
ActuatorErrorHandle	Called when an actuator malfunction occurs.
ActuatorMoveChangeHandle	Called when an actuator has changed its movement
ActuatorWOWConnectHandle	Called when an actuator has a new WOW connect status

Table 4: Actuator Library interface description

3.1.4.2 Controller

Both the Oil and Coolant actuators are controlled by independent control algorithms that are based on the same principles. They each have their own set of parameters and variables for managing their actuators.

The software mimics the original control system designed for the P51 Mustang with an additional feature to modulate the operating temperature to a chosen set point.

Here is described how the control system mimics the original control system.

For each door position there is an upper and lower temperature threshold. The controller keeps track of which door position the door is currently at. Periodically, at the loop polling interval, the controller compares the current temperature of the fluid to the thresholds for the current door position. If the temperature of the fluid being controlled exceeds the upper threshold, then the door is opened to the next door position. If the temperature is below the lower threshold and the temperature is not rising quickly, then the door is closed to the previous position.Note the check for temperature rising quickly is an additional software check that is not present in the original design. This helps reduce the door movements and temperature spikes during takeoff.

During flight the door will open / close to the point where the energy dumped by the radiator approximately equals the energy input by the engine. Since the door positions are discrete, most of the time this balance point will be between two door positions. The thresholds associated with these two door positions will determine the control band for the temperature.

Changes to the operating conditions, such as engine power and ambient air temperature, will impact the energy equation, which ultimately determines which door positions will balance the equation. These operating conditions are dynamic during any particular flight and change through the seasons and local climate conditions.

In software, these thresholds are stored in tables specialized for coolant, Table 5, and oil, Table 6. The active tables are indexed by the door position. The values for the tables were derived from data extracted from the service manual for the P51 relating to calibrating the actuators combined with additional assumptions; that the threshold are equally spaced and that the difference between the upper and lower thresholds for a particular door position is fixed at 1.59 C for coolant and 2.50 C for oil. This allows for a control band between two door positions of about 1.2 C for coolant and 2.0 C for oil.

Interrupter Count	When temperature goes lower than this move to next lower range	When temperature exceeds this move to next higher range
31	115.30	116.89
30	114.91	116.50
29	114.51	116.11
28	114.12	115.71
27	113.73	115.32
26	113.33	114.93
25	112.94	114.53
24	112.55	114.14
23	112.15	113.75
22	111.76	113.35
21	111.37	112.96
20	110.97	112.57
19	110.58	112.17
18	110.19	111.78
17	109.79	111.39
16	109.40	110.99
15	109.01	110.60
14	108.61	110.21
13	108.22	109.81
12	107.83	109.42
11	107.43	109.03
10	107.04	108.63
9	106.65	108.24
8	106.25	107.85
7	105.86	107.45
6	105.47	107.06
5	105.07	106.67
4	104.68	106.27
3	104.29	105.88
2	103.89	105.49
1	103.50	105.09
0	103.11	104.70

Table 5 - Coolant Thermal Control Table

Interrupter Count	When temperature goes lower than this move to next lower range	When temperature exceeds this move to next higher range
23	83.00	85.50
22	82.50	85.00
21	82.00	84.50
20	81.50	84.00
19	81.00	83.50
18	80.50	83.00
17	80.00	82.50
16	79.50	82.00
15	79.00	81.50
14	78.50	81.00
13	78.00	80.50
12	77.50	80.00
11	77.00	79.50
10	76.50	79.00
9	76.00	78.50
8	75.50	78.00
7	75.00	77.50
6	74.50	77.00
5	74.00	76.50
4	73.50	76.00
3	73.00	75.50
2	72.50	75.00
1	72.00	74.50
0	71.50	74.00

Table 6 - Oil Thermal Control Table

The primary benefit of using this control mechanism is that it minimizes the differences from the original design. The original design takes into account the delays in the system and allows the system to respond quickly to changes in the energy equation.

The energy equation is balanced based on the door position, but the temperature is controlled by the thresholds in the table.

In the original design the temperature thresholds could be adjusted by up to about 5 C by adjusting a setscrew. In software this can be accomplished by adding an offset to the table. And because it is software, this offset can be dynamically adjusted during the flight.

The offset applied to the table is limited to make sure that:

- the set point does not go beyond the table
- the door will be fully open if the temperature approaches the maximum operating temperature (120 C for coolant, 95 C for oil)

The offset is set to the minimum value that can be applied while the WOW signal is asserted. This pre-biases the offset for the takeoff condition when the energy input to the system is typically running at or near maximum power. While WOW is asserted and Auto mode is active, the doors will be commanded to open until the limit switch is reached.

When WOW is de-asserted the offset will start adjusting based on the PID loop starting at this minimum value, initially calling for maximum cooling.The output of a discrete-time PID controller monitoring the error between the current temperature and the set point is used to adjust the offset that is applied to the table. A PID controller was chosen because of the following reasons:

- Proportional gain to drive the system to the desired temperature
- Integral gain to eliminate steady-state error
- Differential gain to respond to quick changes in system
- Simple control system to implement in discrete-time

The following assumptions are made about the control system:

- The coolant and oil cooling systems have negligible effects on each other
- Changes in the plant (the physical system in control terminology), are a lot slower (at least x10) than the update rate of the controller
- Hold and sample time can be ignored because the temperature sensor device and controller react a lot faster (at least x10) than the changes in the plant

The reduced control system based on the above assumptions is shown below in Figure 7.

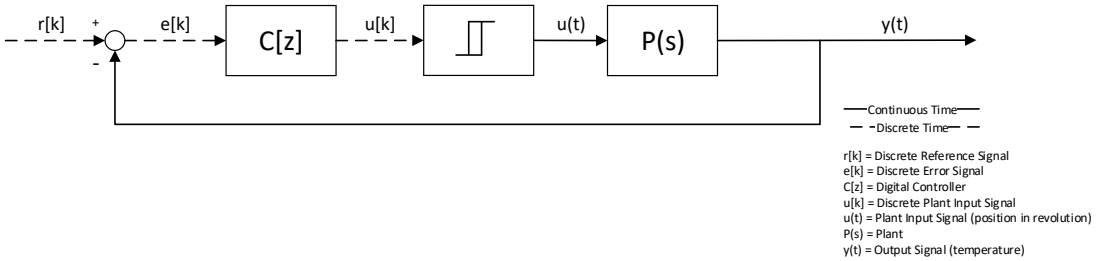

Figure 7: Controller block diagram

The discrete PID controller is represented as a difference equation that is implemented in software. The difference equation is show below in Equation 1:

$$u[k] = K_p * e[k] + K_i * \sum_{i=0}^{k} e[k] + K_d * (e[k] - e[k-1])$$

where K_p = proportional gain
K_i = integral gain
K_d = integral gain
u[k] = input to the plant at time k
e[k] = error at time k

Equation 1: Discrete PID controller difference equation

In addition to applying the difference equation above, the output of the controller is also applied through a hysteresis block which is used to prevent the controller from adjusting the offset when the system is nearly stable and operating at a temperature near the set point.

The update frequency was chosen to be a little faster than the fastest door movement rate to allow the controller to move the doors at the maximum rate. The maximum theoretical rate of door movements is approximately 2 Hz. Setting the loop update rate to 3 Hz ensures that the control loop has been evaluated at least once before the door movement completes so that a determination can be made whether the door should continue to move or not when the interrupter pulse is received.

The controller parameters are determined first by using a simulation of the system and further tuned during test flights.

The integral term (I) is limited so that its contribution will be less than or equal to half the magnitude of the hysteresis block.

When operating near equilibrium we can assume that the contribution for the differential term (D) is close to zero.

Therefore, assuming the system is near equilibrium, as long as the contribution from the proportional term (P) remains less than half the magnitude of the hysteresis block (H), then the offset will not be adjusted.

Through experimentation and analysis we determined that maintaining a relationship, described below, between the P and H settings based on the control table control band size helps to prevent extra door movements and provide a very predictable temperature control band during flight.

For coolant, the table control band is 1.2 C between two adjacent door positions. Setting the hysteresis band to be 0.8 C bigger will create a 2.0 C band of temperatures where the offset is not adjusted. The door will move when the measured temperature exceeds 0.6 C from the set point. The temperature can continue to move up to 1.0 C away from the set point before the table offset will start to be adjusted. This allows time for the system to respond to the door position change, so that it can stabilize between two door positions.

For oil, the table control band is 2.0 C between two adjacent door positions. Setting the hysteresis band to be 0.8 C bigger will create a 2.8 C band of temperatures where the offset is not adjusted.

Equation relating H and P for coolant:

$$H = (1.2 + 0.8) * P$$

Equation relating H and P for oil:

$$H = (2.0 + 0.8) * P$$

The P, I, and D gains may need to be adjusted to account for differences between airplane configurations, such as radiator efficiency, engine power, or other physical properties that affect system responsiveness. Therefore these parameters are configurable through the maintenance interface.

Setting the proportional term to 0 will activate the classic algorithm that will use a fixed table that approximates the original electro-mechanical control system.

3.1.4.3 Event Messages

The Coolant and Oil Actuator Controller services responds to the event messages listed in Table 7 below.

Event Name	Event Generation Source
READ_STATE	CLI
READ_TEMP	CLI
READ_DOOR_POS	CLI
READ_OP_MODE	CLI
READ_LIMIT_DIR	CLI

READ_CTRL_PARAM	CLI
WRITE_CTRL_PARAM_P_GAIN	Communication Manager
WRITE_CTRL_PARAM_I_GAIN	Communication Manager
WRITE_CTRL_PARAM_D_GAIN	Communication Manager
WRITE_CTRL_PARAM_HYSTERESIS	Communication Manager
WRITE_CTRL_PARAM_SET_POINT	Communication Manager
WRITE_CTRL_PARAM_PRE_BIAS	Communicaton Manager
WRITE_CTRL_LOOP_INTERVAL	Communication Manager
START_BLE_UPDATE	CLI
STOP_BLE_UPDATE	CLI
START_CTRL_LOOP	System Manager
STOP_CTRL_LOOP	System Manager
MOVE	CLI
ACTUATOR_CONNECT	Actuator Library Callback
ACTUATOR_DISCONNECT	Actuator Library Callback
OP_MODE_CHANGE	Actuator Library Callback
DOOR_ACTUATION	Actuator Library Callback
INTERRUPTER_TRIG	Actuator Library Callback
SET_ACTUATOR_ALARM	Actuator Library Callback
CLEAR_ACTUATOR_ALARM	Actuator Library Callback
START_CALIBRATION	System Manager

Table 7: Actuator Controller event messages

3.1.5 Event Logger

The event logger service is in charge of the following:

- Write system events to external flash
- Read logs from the external flash
- Formats and un-formats the logs to and from the external flash
- Keeps track of the next location to write the logs
- Checks in with the Watchdog Manager

The event logger uses the log flash library to perform the reads and writes to the flash.

3.1.5.1 Log Flash Library

The log flash library provides the functionality to read and write to the external flash. It abstracts the underlying logic of interacting with the specific flash device. It provides the following functions:

- Write byte(s)
- Read byte(s)
- Erase the whole flash

3.1.5.2 Event Messages

The Event Logger responds to the event messages listed in Table 8 below.

Event	Event Generation Source
SYSTEM_START	System Manager
SYSTEM_ENGINE_ON	System Manager
SYSTEM_ENGINE_OFF	System Manager
SYSTEM_RESET	System Manager
SYSTEM_ALARM	System Manager
SYSTEM_MCU_UID_1	System Manager
SYSTEM_MCU_UID_2	System Manager
SYSTEM_MCU_UID_3	System Manager
CTRL_LOOP_INTERVAL	System Manager
WOW_SETTING	System Manager
OIL_TEMP	Oil Actuator Controller
OIL_ACTUATION	Oil Actuator Controller
OIL_POS_UPDATE	Oil Actuator Controller
OIL_POS_CORRECTION	Oil Actuator Controller
OIL_OP_MODE	Oil Actuator Controller
OIL_LIMIT_TRIG	Oil Actuator Controller
OIL_SET_POINT	Oil Actuator Controller
OIL_CTRL_PARAM_P_GAIN	Oil Actuator Controller
OIL_CTRL_PARAM_I_GAIN	Oil Actuator Controller
OIL_CTRL_PARAM_D_GAIN	Oil Actuator Controller
OIL_CTRL_PARAM_HYSTERESIS	Oil Actuator Controller
OIL_CTRL_VALUE_ERROR	Oil Actuator Controller
OIL_CTRL_VALUE_OUTPUT	Oil Actuator Controller
OIL_CTRL_VALUE_OFFSET	Oil Actuator Controller
OIL_CTRL_ALARM	Oil Actuator Controller
COOLANT_TEMP	Coolant Actuator Controller
COOLANT _ACTUATION	Coolant Actuator Controller
COOLANT _POS_UPDATE	Coolant Actuator Controller
COOLANT _POS_CORRECTION	Coolant Actuator Controller
COOLANT _OP_MODE	Coolant Actuator Controller
COOLANT _LIMIT_TRIG	Coolant Actuator Controller
COOLANT _SET_POINT	Coolant Actuator Controller
COOLANT _CTRL_PARAM_P_GAIN	Coolant Actuator Controller
COOLANT _CTRL_PARAM_I_GAIN	Coolant Actuator Controller
COOLANT _CTRL_PARAM_D_GAIN	Coolant Actuator Controller
COOLANT _CTRL_PARAM_HYSTERESIS	Coolant Actuator Controller
COOLANT_CTRL_VALUE_ERROR	Coolant Actuator Controller
COOLANT_CTRL_VALUE_OUTPUT	Coolant Actuator Controller
COOLANT_CTRL_VALUE_OFFSET	Coolant Actuator Controller
COOLANT _CTRL_ALARM	Coolant Actuator Controller
MOBILE_CONNECT	Communication Manager
MOBILE_DISCONNECT	Communication Manager
MOBILE_TIMESTAMP	Communication Manager
MOBILE_READ_LOG	Communication Manager
MOBILE_UPDATE_FW	Communication Manager
READ_LOGS	Communication Manager
CLEAR_LOGS	Communication Manager
READ_PROPERTY	CLI

Table 8: Event Logger event messages

3.1.6 Watchdog Manager

The Watchdog Manager is responsible for the following:

- Kicking the internal watchdog timer
- Book keeping of services checking in

Timing for the Watchdog Manager is a hard requirement. The Watchdog Manager must kick the watchdog peripheral at the set interval otherwise the device will reset.

On each cycle that the Watchdog Manager executes, it determines whether it should kick the watchdog based on when services have last checked in with it. This is done by keeping track of a service's last check in-time. If the time difference between the current time and the service's last check-in time exceeds the service's set time limit, the Watchdog Manager will stop kicking the watchdog and the system will reset. Table 9 below illustrates the table that the Watchdog Manager bookkeeps.

Service	Last Check-in [ticks]	Check-in Time Limit [ms]
System Manager	0	150
Communication Manager	0	150
Actuator Controller (Coolant)	0	150
Actuator Controller (Oil)	0	150
Event Logger	0	370000
Command Line Interface	0	10000
Test Alarm Lights	0	4000
Debug Manager	0	150

Table 9: Watchdog Manager check-in table

Each thread has access to this table directly in order to minimize latency of a service checking in with the Watchdog Manager. The check-in times and watchdog timeout are set such that the time limit is greater than 100 ms, which is the time between watchdog kicks.

3.1.7 Command Line Interface

The Command Line Interface (CLI) service is responsible for the following:

- Providing information about and from the different services to the developer
- Running unit and system tests on the system

As a debugging and testing service, the CLI sends event messages to services while imitating another service. For example, the CLI can imitate the Oil Actuator Controller and send event logging information to the Event Logger service. This is useful because the CLI can then check the response of the Event Logger (ex. request for its logging content) so that it can compare the response with an expected value. By doing this, the CLI is used to perform system level testing for the different services.

3.1.7.1 Event Messages

This section lists the different event message the CLI will listen to. Because the CLI imitates other services, the event messages that it will listen to is a combination of all the events from the services it imitates.

3.1.8 Debug Logger

The Debug Logger service is responsible for serializing messages to be printed to the debug console. It serializes messages through the use of the message queue. All other services use the Debug Logger in order to print debug messages, useful during development. Instead of a queue of event messages, the debug logger contains a queue of the messages that it writes in the order in which the messages are received.

If the queue is full, the debug manager will increase a counter that keeps track of the number of missed messages. A message will then be sent to the debug console with the number of messages that have been missed.

3.2 Middleware Layer

The middleware layer contains hardware agnostic software modules which are used by the application. This section describes the modules in this layer.

3.2.1 CMSIS-RTOS and Free-RTOS

The application software interfaces with the CMSIS-RTOS API. CMSIS-RTOS is a vendor-independent hardware abstraction layer for the Cortex-M processor series. The underlying RTOS is Free-RTOS v9.0.0 with the default configurations that the STM32CubeMX application defines.

3.2.2 STM32Cube USB Host Library Module

The controller only acts as the host on the USB interface for this application. As such, only the USB host library is selected as part of the STM32CubeMX setup. The STM32Cube USB host library module is organized into two main components: the USB host core and the USB host class drivers. Figure 8 below is taken from the *STM32Cube USB Host Library User Manual UM1720* user manual which shows the USB host library architecture.

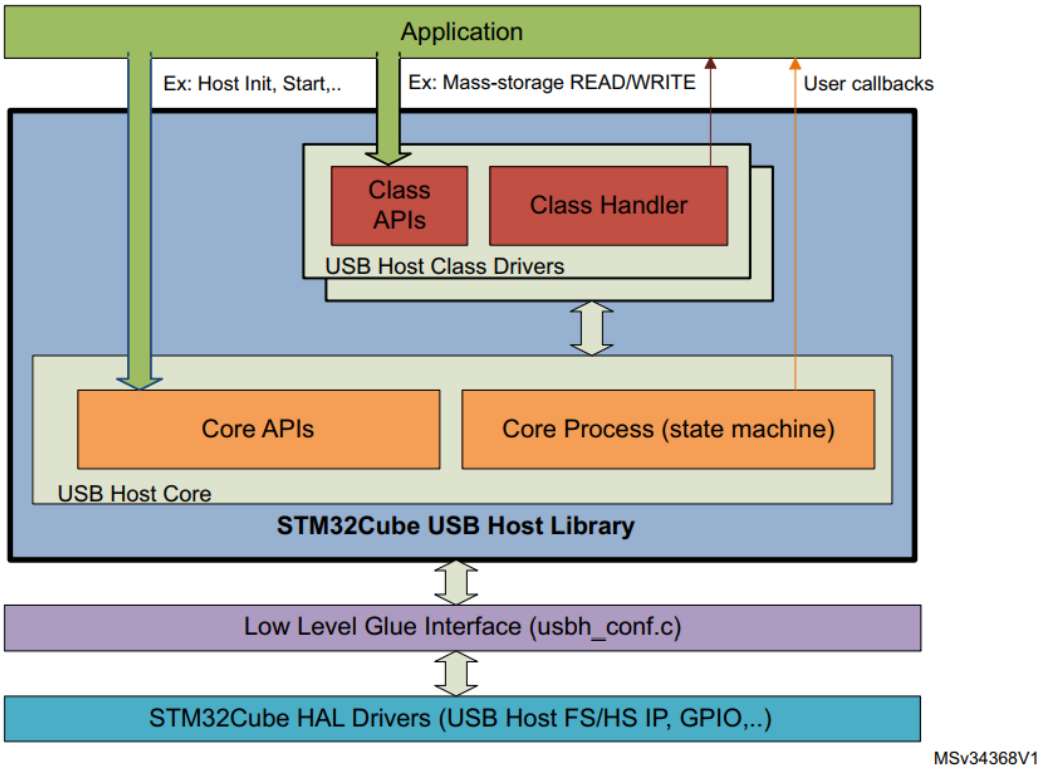

Figure 8: USB host library architecture

The USB host library provides two APIs for the application layer: the Core APIs and the Class APIs. The Core APIs are used to enable basic USB functionality such as initializing the USB host stack, enumerating the USB device, and registering the class of USB devices. The Class APIs provide functionality to communicate with specific USB device classes.

The only class of USB devices the controller communicates with at the moment is the Communication Device Class (CDC). Accordingly, only the CDC files are used for the application. More information about the STM32Cube USB Host library can be found in the *STM32Cube USB Host Library User Manual UM1720* document.

3.3 Hardware Abstraction Layer (HAL)

The hardware abstraction layer is provided by STM through the STM32CubeMX application. It provides wrappers and convenience functions to the lower level driver code so that application code can be separated from the underlying hardware. The HAL provides three operation modes (polling, interrupt, and DMA) for applicable peripherals. Table 10 below lists the modes of operation for the applicable peripherals for this application.

Peripheral	Operation Mode	Comments
GPIO	Normal and Interrupt	• Interrupt mode is used for peripherals such as limit switch and interrupter inputs. • Pins are normally read and set

		otherwise
UART Receive	DMA	• DMA with idle line detect
UART Transmit	DMA	
ADC	DMA	• Continuous multichannel conversion with DMA
QSPI	Polling	• Polling because HAL layer interrupts causes qspi to hang sometime

Table 10: Peripheral operation mode

3.4 STM32 Driver Layer and STM32Fx Startup and Configuration Code

STM32CubeMX provides low level drivers which the HAL builds on top of. The application can also use these low level drivers directly as well. In addition the STM32CubeMX application generates the startup and configuration code for the project. This includes setting up the appropriate pin multiplexing and clock tree for the system. Firmware version 1.17.0 was used.

4 System-Level Considerations

This section specifies system design considerations that are made based on certain constraints in the system.

4.1 Internal Flash Partitioning

The internal flash memory on the STM32F412 microcontroller is 512Kbytes. The memory partitioning is show in Table 11 below.

Sector	Size	Field
Sector 0	16 KBytes	Bootloader
Sector 1	16 KBytes	Reserved
Sector 2	16 KBytes	Control Block
Sector 3	16 KBytes	Configuration
Sector 4	64 KBytes	Reserved
Sector 5	128 KBytes	Image A
Sector 6	128 KBytes	Reserved
Sector 7	128 KBytes	Image B

Table 11: Internal flash partition

The configuration sector contains configurable data that must persist between restarts of the system. The configuration registers are listed in Table 12 below.

Configuration Type	Offset	Size in bytes	Description
Header	0x000	4	MAJOR.MINOR.PATCH MAJOR is at the least significant byte.
	0x1FC	4	Crc32
System	0x200	2	Flight Number
	0x202	2	Reserved
	0x204	4	Temperature ADC filter window
	0x208	4	Control loop interval
	0x20C	8	Flight registration
	0x214	4	WOW Setting
	0x3FC	4	Crc32
Oil	0x400	4	Oil controller proportional gain
	0x404	4	Oil controller integral gain
	0x408	4	Oil controller differential gain
	0x40C	4	Oil controller hysteresis
	0x410	2	Oil controller set point
	0x412	2	Oil controller pre bias
	0x5FC	4	Crc32
Coolant	0x600	4	Coolant controller proportional gain
	0x604	4	Coolant controller integral gain
	0x608	4	Coolant controller differential gain
	0x60C	4	Coolant controller hysteresis

	0x610	2	Coolant controller set point
	0x612	2	Coolant controller pre bias
	0x7FC	4	Crc32

Table 12: Configuration data

Each field will be discussed in detail in Section 5.

4.2 External Flash Partitioning

The external flash used is the Cypress Semiconductor S25FL256LAGMFN000. This component has 256Mbit (32 Mbyte) of flash memory and is byte programmable. The log records are fixed length records with the following information: flight number, system time, event type, data, crc16. The flight number field is incremented whenever the engine is started. A flight is defined as the period between an engine on and engine off signal. An engine signal that is on, when the controller starts up, will indicate that the controller has restarted and the flight number should be kept.

The system time field is defined as the number of seconds the controller has been awake for. This time is started at 0 every time the controller starts up and will count up to 1677215 seconds which is equivalent to around 4660 hours. The actual flight time is not recorded in the logs. However, it can be calculated as the difference between the system time of an engine on and off event. A log record is shown in Table 13 below.

	Event Type	System Time	Data Value	Flight Number	Crc16
Size in Byte(s)	1	3	4	2	2

Table 13: Generic event log structure

The crc code is crc of the preceding bytes in the log record. The log records are page aligned to the flash's page boundary. In the case of the selected flash, page alignment is on every 256 bytes. The total number of records that can be stored in the flash is:

$$\#\ of\ records = 131072\ pages * \left\lfloor \frac{256\ bytes\ per\ page}{12\ bytes\ per\ record} \right\rfloor = 2752512\ records$$

The majority of the logs come from the regular temperature and actuator offset logs. The requirement for logging temperature is one measurement every 5 seconds. Taking into account the TBO of the engines to be around 600 hours, the total number of temperature and actuator offset measurements can be calculated

$$\#\ of\ temp\ \&\ actuator\ offset\ readings = 2 * [2\ actuators * \ 600hr * \frac{3600s}{1hr} * \frac{1\ reading}{5s}] = 1728000\ readings$$

The other logs are based on changes in the actuator, firmware changes, or controller resets which are mostly infrequent changes. This should occur less than once every 5 second with a reasonable controller. Keeping this into account, there is still roughly a third of the flash space

left. Keeping into account the TBO safety factor already calculated, the size of the flash selected is sufficient.

4.2.1 Event Log Format

The format for each type of event log is shown in Table 14 below.

Component	Event Name	Event Description	Event Type Code	Value	Value Description
Reserved	-	-	0x00 – 0x0F	-	-
System	System start	-	0x10	Uint32 MAJOR.MINOR.PATCH MAJOR is at the least significant byte.	Firmware version MAJOR and MINOR are 1 byte while PATCH is 2 bytes
	Engine On	-	0x11	Uint32	Time from epoch (only going forward)
	Engine Off	-	0x12	Uint32	Time from epoch (only going forward)
	Reset	-	0x13	Uint8 0x00 – Normal power up 0x01 – FW update 0x02 – Watchdog tripped	Reason for reset
	Alert	-	0x14	Uint8 0x00 – None 0x01 – Controller high temperature 0x02 – Watchdog tripped 0x03 – Power good tripped	-
	MCU UID 1	-	0x15	Uint32	1st 32-bits of the MCU UID
	MCU UID 2	-	0x16	Uint32	2nd 32-bits of the MCU UID
	MCU UID 3	-	0x17	Uint32	3rd 32-bits of the MCU UID
	CTRL_LOOP_INTERVAL		0x19	Uint32	Min: 100 Max: 2000
	Reserved	-	0x1A – 0x1F	-	-
Oil	Temperature Update	-	0x20	Int16	Value is measured as 1/100th of a degree (°C)
	Door actuation	Logged when either the open or close	0x21	Uint8 0x00 – Actuator close coil energized	

		coils are energized.		0x01 – Actuator open coil energized 0x02 – Actuator close coil de-energized 0x03 – Actuator open coil de-energized	
	Door position update	-	0x22	UInt8	Measured as revolutions from closed position.
	Door position correction	Logged when an end limit is hit and door position differs than the end limit position.	0x23	Int8	Measured in number of revolutions that must be compensated to match the real position.
	Operation mode change	-	0x24	Uint8 0x00 – Manual neutral 0x01 – Manual open 0x02 – Manual close 0x03 – Auto 0x04 - WOW	-
	Limit switch hit	-	0x25	Uint8 0x00 – Close limit hit 0x01 – Open limit hit	-
	Set point change	Logged when the temperature set point is changed	0x26	Int16	Value is measured as $1/100^{th}$ of a degree (˚C)
	Controller P Gain	-	0x27	Int32_t	Value of proportional gain
	Controller I Gain	-	0x28	Int32_t	Value of integral gain
	Controller D Gain	-	0x29	Int32_t	Value of differential gain
	Controller Hysteresis	-	0x2A	Int32_t	Value of hysteresis
	Alert	-	0x2B	Uint8	-

				0x00 – None 0x01 – High temperature 0x02 – Interrupter missed 0x03 – Open and close limit hit 0x04 – Disconnected	
	Controller Error	-	0x2C	Float (4 bytes)	Value of error
	Controller Output	-	0x2D	Float (4 bytes)	Value of output
	Pre-bias	-	0x2E	Int16	Value is measured as $1/100^{th}$ of a degree (˚C)
	Controller Offset	-	0x2F	Float (4 bytes)	Value of offset
Coolant	Temperature Update	-	0x30	Int16	Value is measured as $1/100^{th}$ of a degree (˚C)
	Door actuation	Logged when either the open or close coils are energized.	0x31	Uint8 0x00 – Actuator close coil energized 0x01 – Actuator open coil energized 0x02 – Actuator close coil de-energized 0x03 – Actuator open coil de-energized	
	Door position update	-	0x32	Uint8	Measured as revolutions from closed position.
	Door position correction	Logged when an end limit is hit and door position differs than the end limit position.	0x33	Int8	Measured in number of revolutions that must be compensated to match the real position.
	Operation mode change	-	0x34	Uint8 0x00 – Manual neutral 0x01 – Manual open 0x02 – Manual close	-

				0x03 – Auto 0x04 - WOW	
	Limit switch hit	-	0x35	Uint8 0x00 – Close limit hit 0x01 – Open limit hit	-
	Set point change	Logged when the temperature set point is changed	0x36	Int16	Value is measured as $1/100^{th}$ of a degree (˚C)
	Controller P Gain	-	0x37	Int32_t	Value of proportional gain
	Controller I Gain	-	0x38	Int32_t	Value of integral gain
	Controller D Gain	-	0x39	Int32_t	Value of differential gain
	Controller Hysteresis	-	0x3A	Int32_t	Value of hysteresis
	Alert	-	0x3B	Uint8 0x00 – None 0x01 – High temperature 0x02 – Interrupter missed 0x03 – Open and close limit hit 0x04 – Disconnected	-
	Controller Error	-	0x3C	Float (4 bytes)	Value of error
	Controller Output	-	0x3D	Float (4 bytes)	Value of output
	Pre-bias	-	0x3E	Int16	Value is measured as $1/100^{th}$ of a degree (˚C)
	Controller Offset	-	0x3F	Float (4 bytes)	Value of offset
Mobile Application	Connection	-	0x40	Uint32	Uuid of the connected device (generated from the app)
	Disconnection	-	0x41	Uint32	Uuid of the connected device (generated from the app)
	Timestamp	-	0x42	Uint32	Time from epoch in

					seconds (only going forward)
	Read logs	-	0x43	Uint16	Flight number of the requested logs
	Update FW	-	0x44	Uint32 MAJOR.MINOR.PATCH	Firmware version MAJOR and MINOR are 1 byte while PATCH is 2 bytes
	Reserved	-	0x45 – 0x4F	-	-
Reserved	-	-	0x50 – 0xFF	-	-

Table 14: List of event log structure

5 System Operation

The section covers the system operation of the software.

5.1 Maintenance Mode

Maintenance mode is entered when the following conditions are met:

- The engine pressure signal is low, indicating that the engine is off
- The mobile application is connected to the firmware

Maintenance mode puts the controller in a state where the following operations can be performed via the mobile application:

- Update oil and coolant temperature set points
- Update oil and coolant control loop parameters
- Calibrate oil and coolant actuator position
- Update firmware
- Retrieve log data from non-volatile flash

When the engine pressure signal goes high, the mobile app is locked out from being able to perform the above operations.

5.2 Bootloader

The bootloader is the first point of entry into the software after a reset or power on of the system. The bootloader performs the following operations:

- Check control block information
- Determine FW application to boot from
- Determine if FW update needs to occur
- Update pending FW
- Jump to FW application
- Kick HW watchdog

When the bootloader is started, it first reads the state of the status byte in the control block to determine what it should do. The control block contains the information shown in Table 15.

Byte	Purpose	Possible Values	Meaning
0	Area use for validation	0x0F	If any other value, image A and B are in an unknown state
1	Status	0xFF 0xF0 0x00	New image downloaded Error in new image Original State – alternate image not downloaded
2	Boot image	0xFF 0xF0	Image A Image B

3	Reserved		Reserved byte
4:7	Crc		Crc32 of bytes 0-3

Table 15: Control block information

On a normal boot-up, the status flag should be set to its original state, and the bootloader will jump to the FW application indicated by the boot image byte and start execution from there.

If the status flag indicates that a new image has been downloaded, the bootloader will check the firmware application control header of the selected image. The firmware control header is shown in Table 16 below.

Byte	Purpose	Meaning
0:3	Version	Version of the FW (MAJOR.MINOR.PATCH) MAJOR and MINOR are 1 byte while PATCH is 2 bytes. MAJOR is at the least significant byte.
4:7	Length in bytes	Length of the image data in bytes
8	Image Type	Image A (0xFF) or Image B(0xF0)
9:11	Reserved	
12:15	CRC	CRC32 of the image data
16:511	Reserved	

Table 16: FW application control header

A CRC will be applied to the firmware to make sure that the executing code is valid. If it is, the bootloader will jump to the beginning of the firmware and start execution. Otherwise the bootloader will execute the previous firmware.

Before jumping to the application code, the bootloader will set the state of the status byte to 0xF0 to indicate an error. This error status code should be cleared by the executing application at an appropriate point to indicate that it is error free. This is so that if the executing application is bad, the bootloader can revert to loading the old application.

5.3 Over-The-Air (OTA) Firmware Update Process

The OTA firmware update process is initiated by the mobile application. The mobile application sends the firmware to the Communication Manager which will then dispatch the information to the System Manager for writing into internal flash.

Upon receiving the blocks of firmware from the Communication Manager, the Communication Manager will send the packets of data to the System Manager. The System Manager will then write the data into internal flash. Once the entirety of the firmware is written into the internal flash, the System Manager initiates a CRC check on the firmware. If the CRC check is invalid, the System Manager communicates the error to the Communication Manager and the mobile application will be notified. If the CRC check is valid, the System Manager updates the control block information and resets the system.

5.3.1 Firmware Versioning

The firmware versioning will comply with Semantic Versioning. This gives a version number in the form of MAJOR.MINOR.PATCH which is used by the firmware and mobile app to check for compatibility. The MAJOR, MINOR, and PATCH values represent:

- MAJOR version bump indicates an incompatible API change with previous versions of the mobile app
- MINOR version bump indicates backwards compatible features with previous versions of the mobile app
- PATCH version bump indicates backwards compatible bug fixes with previous versions of the mobile app

A MAJOR version bump will occur in this application when the API between the controller and mobile device changes. This also includes any changes in the event log format.

The firmware is not expected to be updated often during the lifetime of the device. Therefore the version number will be represented as three bytes with each byte representing either the MAJOR, MINOR, or PATCH number.

5.4 Door Position Calibration

A door position calibration is needed for controller to determine the position of the air door positions. The door positions are calibrated based on the operation and considerations below:

- The doors and the software will be calibrated and the door position will be remembered in non-volatile memory. The calibration process should be performed when:
    - A controller is installed or replaced
    - An actuator is installed or replaced
- The controller will always record the door position in non-volatile memory as indicated by the interrupter switch
- The controller will rely on the stored position of the door for calibration on startup
- The controller PID loop is not dependent on the absolute door position. The controller uses the error in the desired and actual temperature to either open the door more, or close it more.
- If the controller encounters an end-stop limit switch earlier than expected (fully closed or fully open door) due to an error in stored position, the internal state is automatically updated to the true position of the door. In this way the controller is self-correcting and the position of the door stays accurate over time.
- In the worst case scenario, the stored value could differ from the actual value if during shutdown while in automatic mode the door was operating and the interrupter was almost triggered and the power shutoff. The interrupter may register a revolution, but the controller was powered down and as such did not record the interrupter event.

5.5 Controller Fault and Alert Behavior

The response of the controller based on certain fault and alert conditions are listed below in Table 17.

Event	Turn on Coolant Alert Light	Turn on Oil Alert Light
Coolant temperature exceeds maximum temperature	x	
Oil temperature exceeds maximum temperature		x
Coolant actuator fails	x	
Oil actuator fails		x
Controller malfunctions	x	x

Table 17: Controller fault and alert responses

5.6 Mobile Application Communication

The controller and the mobile application communicate with one another through BLE. The controller exposes certain services and characteristics that the mobile application will subscribe to. The communication protocol between the two devices can be found in the Mobile SDD.

6 Tools

This section describes the tools that are used during the development process.

6.1 STM32CubeMX

The STM32CubeMX tool is a graphical interface application supplied by STM which aids developers in generating code for setting the desired pin muxing, peripheral drivers, hardware abstraction layer interfaces, and a real time operating system. It is part of the STMCube™ initiative originated by STMicroelectronics to ease developers in writing controller specific drivers and setup code so that more time can be spent in writing the actual application.

6.2 System Workbench for STM32

The System Workbench for STM32 is a free Eclipse IDE that integrates a complete code editor, compilation tools, and remote-debugging tools for STM32 devices. The System Workbench will primarily be used as a debugger for the firmware but may also be used to build and flash the firmware onto the microcontroller.

6.3 Software Development Environment

The software development environment is set up using a Linux operating system running the latest Ubuntu 16.04 LTS distribution. The Linux OS is running inside a virtual machine (VM) from the Windows host operating system. A VM is useful in containing development packages so that the same development environment can be shared easily between developers.

6.4 STM32F412G Evaluation Board

The STM32F412G evaluation board is used during the implementation phase of development. This board contains the same processor and package type as the one that is used in the final design. This means that very minimal software work is required to have the code that is developed on the evaluation board to work on the final board.

7 Licenses

Software	License
STM33CubeF4	Open-source BSD
FreeRTOS v8.0.0	GPL 2
CMSIS	Apache 2.0
BGAPI™ and BGLib™	Free license
USB host library	MCD-ST Liberty SW License Agreement V2