# Aurora V Avionics Documentation

## FreeRTOS Task Architecture

RMIT University

Version ..... 1.0
Last Updated ..... 2024–07–14
Major Contributors ..... Matthew Ricci

| Date | Changes Made | Made By |
|---|---|---|
| 2024–07–14 | Create initial document | Matthew Ricci |

# Contents

# 1 Architecture Overview

The Aurora V avionics firmware package implements FreeRTOS for task scheduling. The firmware is designed to provide logical implementation of hardware interfaces for communications and telemetry, state calculations, data logging, and more, while adhering to stringent timing requirements.

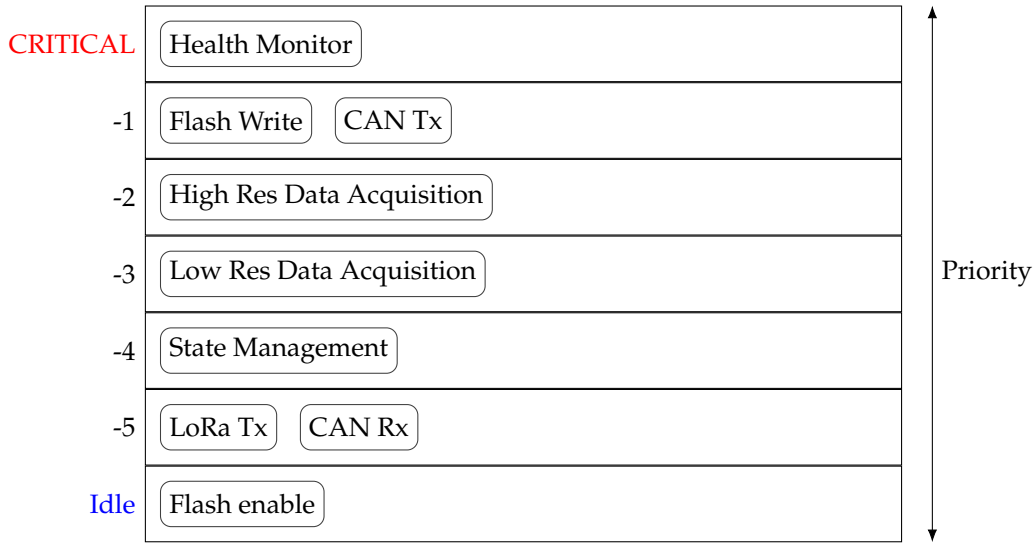


Figure 1: Avionics RTOS task hierarchy

Figure 1 illustrates a high-level overview of the task architecture. Tasks with higher priority are designed to execute promptly whenever they are not blocked, ensuring critical operations are performed without delay.

Lower priority tasks are scheduled with more flexible timing but may enter critical sections to prevent preemption during essential operations.

# 2 Task Description

## 2.1 Health Monitor

<To be designed>

## 2.2 Flash (impl. Appendix A)

Flashing of data primarily occurs during idle time on the CPU. This is managed through a task group flag that enables the operation, unblocking the task that handles writing of flash.

### 2.2.1 Flash Enable

The avionics firmware implements a FreeRTOS idle hook that runs when no tasks are occupying time on the CPU.

Listing 1: Flash idle logic

```
// Write if a page is available in the buffer
if (currentState == LAUNCH && mem.pageReady)
  xEventGroupSetBits(xTaskEnableGroup, 0x01);
```

If a launch event has occurred, the memory buffer is polled to determine if data is ready to be written. When this is the case an event group flag is set to enable the flash write task.

### 2.2.2 Flash Write

Listing 2: Flash write logic

```
  // Wait for write flag to be ready, clear flag on exit
  EventBits_t uxBits = xEventGroupWaitBits(xTaskEnableGroup, 0x01,
      pdTRUE, pdFALSE, timeout);
  if ((uxBits & 0x01) == 0x01) {
    // Flush data to output buffer
    bool success = mem.readPage(&mem, outBuff);
    if (success) {
      // Write data to flash memory
      write_data_spi(outBuff, &hspi4, pageAddr, spi4_cs);
      pageAddr += 0x100;
    }
  }
```

The write task blocks on the task group bit that is set by the idle hook. When the flag is set, the task will attempt to flush the memory buffer, writing the full page to flash and incrementing the write address if successful.

## 2.3 Communications (impl. Appendix ??)

Communications on-board Aurora V are handled both internally and externally. Telemetry to the ground, as well as ground station commands to the rocket, are handled wirelessly over LoRa. Internal communications to other subsystems implement a CAN bus interface.

Within the RTOS firmware, these communications handle in- and out-bound messages through queues.

### 2.3.1 LoRa

foo

### 2.3.2 CAN

bar

## 2.4 Data Acquisition

### 2.4.1 High Resolution

### 2.4.2 Low Resolution

## 2.5 State Management

# Appendix A

```
// Use idle time to write flash
void vApplicationIdleHook(void) {
  // Write if a page is available in the buffer
  if (currentState == LAUNCH && mem.pageReady)
    xEventGroupSetBits(xTaskEnableGroup, 0x01);
}

void vFlashBuffer(void *argument) {
  const TickType_t timeout = pdMS_TO_TICKS(1);
  uint32_t pageAddr        = 0;
  for (;;) {
    // Wait for write flag to be ready, clear flag on exit
    EventBits_t uxBits = xEventGroupWaitBits(xTaskEnableGroup, 0x01, pdTRUE
        , pdFALSE, timeout);
    if ((uxBits & 0x01) == 0x01) {
      // Flush data to output buffer
      bool success = mem.readPage(&mem, outBuff);
      if (success) {
        // Write data to flash memory
        write_data_spi(outBuff, &hspi4, pageAddr, spi4_cs);
        pageAddr += 0x100;
      }
    }
  }
}
```

Listing A.1: Flash task implementation