Aurora V Avionics Documentation

Avionics dataframe structure & synchronisation

RMIT University

April 5, 2024

1 Overview

This document provides a high level introduction to the generic structure of avionics data as it is being stored in flash memory, with an overview of how synchronisation is achieved to match up high and low resolution data together with that logged by the Blue Raven altimeters.

Section 2 details the dataframe structure and its constituent bytes, with Section 3 describing the data synchronisation logic.

2 Dataframe structure

Figure 1 provides a bytefield structure for the dataframes being constructed from the on-board avionics sensors. Each frame contains a two-byte header that identifies the type of data recorded in the payload (e.g. high resolution, low resolution or rocket payload data), describes the number of bytes contained within the payload, and provides a synchronisation byte for post processing.

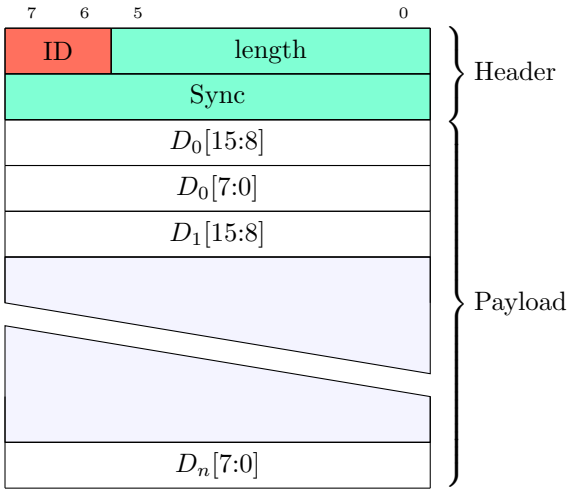

Figure 1: Dataframe structure for avionics

These dataframes will be stored within a `uint8_t` buffer as opposed to a 16-bit equivalent. This is because some sensor data is contained within 24 bits per sample, so to make efficient use of space while maintaining simplicity of implementation 8-bit nibbles are desirable.

2.1 Data typing

Data stored by the avionics system will retain the raw structure as output by the on-board sensors, written as 8-bit unsigned blocks. This is to minimise the amount of storage used with each frame, as well as to avoid any processing overhead from computing real data.

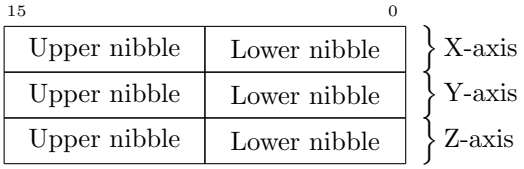

Figure 2: Example structure for a single sensor payload

Processing can then be done offline during analysis to obtain real data, for example by multiplying out the sensitivity of a given sensor into a floating point variable.

The payload of each dataframe will consist of consecutive groupings of sensor data, an example of which is pictured in Figure 2. No delimiter will be used to separate these groupings and instead processing will be performed by using an agreed upon structure.

Code that processes this data will need to read in each dataframe from memory, using the length field in the header to delimit where the data ends. Based on the type information from the frame header, each variable of data can then be read.

3 Synchronisation

According to the Blue Raven user's guide [1], synchronisation between high and low resolution data is achieved through a "sync code" that is stored with the data. This code is essentially a counter that increments with each millisecond, with overflow at a count of 250.

To maintain synchronisation both with our own sets of data, as well as with the Blue Raven data (for the sake of comparison), an identical solution will be implemented within our own logging system. As outlined in Section 2, this sync code is included as a full byte in the frame header, following the id and frame length. This byte will be included with all forms of saved data to maintain synchronisation parity across the board.

3.1 Example synchronisation code

```
/*======================
          TODO
======================*/
```

References

[1] A. Adamson, "Blue raven user's guide." https://www.apogeerockets.com/downloads/PDFs/09170-blue_raven_users_manual_september_15.pdf, 09 2023.