Document Information

This document serves to provide a comprehensive overview of the Australis Firmware system architecture and its implementation. Descriptions are provided of the firmware architecture and its components, implementations of internal systems, and justifications for the design.

Additional information is included on the future of the system and where developments may lead.

Related Documents

Available from `https://github.com/RMIT-Competition-Rocketry/`:

- AV2 Hardware Reference

Acknowledgements

Thank you to Aurora & Legacy project team leads Patrick Underwood and Brodie Alexander for providing the opportunity and environment to work on these rockets as part of the team, as well as to Dr. Glenn Matthews for his enourmous help in getting this project up off the ground and his pivotal involvement with the avionics team in our initial year.

And thank you to everyone on the Aurora team who helped make it all a reality!

Key Contributors

Thank you to these contributors, without whom the Australis Firmware would not exist:

- Matthew Ricci – *Principal firmware developer*
- William Houlahan – *Initial driver implementations*
- Benjamin Wilsmore – *Initial driver implementations*

Special Thanks

Other members of the Aurora V Avionics team:

- Hugo Begg – *avionics hardware*
- Jonathan Chandler – *ground control station*
- Jeremy Timotius – *data analysis*
- Lucas Webb – *ground control station*

Australis Documentation

Firmware Design Document

Version 1.0
Last Updated 2025/04/20
Date Created 2025/03/16
Original Author ..... Matthew Ricci

Contents

Important Terms

Australis	SRAD flight computer platform for high power rockets.
Australis Core	An internal component providing the base API and critical logic; stylised as `core`.
Australis Extra	An internal component providing modular systems that may be optionally included to extend core functionality; stylised as `extra`.
Australis Firmware	Flight computer firmware system for high power rockets; designed for, but not limited to, deployment on Australis targets.
Component	A collection of semantically related code groups and files within the Australis Firmware ecosystem.
Device	A hardware element external to the controller that provides additional functionality via a connected interface.
Driver	Software implementation of a device or peripheral interface.
Peripheral	A hardware element internal to the controller that provides important extensions to the feature set of its core processor.
Submodule	An isolated source group packaged within `extra`, that extends system functionality to target code. Submodules may only depend on the `core` API.
Subtarget	A single connected hardware unit as part of a target, consisting of one or more unique chain-programmable controllers.
Target	A hardware platform on which the Australis Firmware operates.

Abbreviations

A3[1]	Aurora 3
API	Application Programming Interface
AV2	Australis Version 2
COTS	Commercial Off The Shelf
SRAD	Student Researched And Developed

[1] Also refers to version 1 of the Australis flight computer hardware.

1 System Overview

A Note on Endianness

All data in this document is presented in Big-endian notation, unless otherwise specified.

In reference to byte order, this means that the lower address of a datum represents the least significant byte of the complete value.

1.1 Firmware Architecture

Australis Firmware is a software platform for implementing FreeRTOS task systems designed for high power rockets based on STM32 hardware. It is packaged as two internal components, `core` and `extra`, which can be integrated by developers to create a complete flight computer system for their required target.

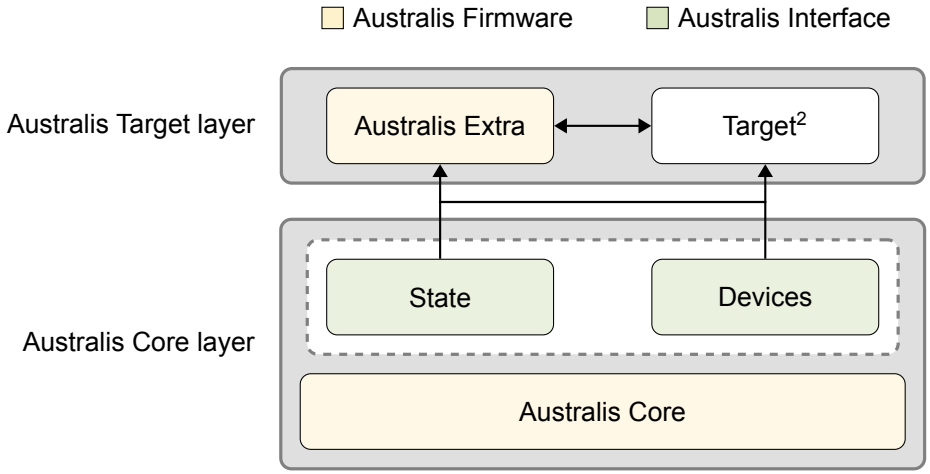

Figure 1.1: Australis Firmware system architecture

Australis Firmware implements a layered architecture (pictured Fig. 1.1), where `core` defines the base which provides the system API and critical logic, with the target layer on top integrating both `extra` submodules and target source code.

1.2 Components

1.2.1 Core

State

Devices

1.2.2 Extra

1.3 Data Storage and Interpretation

Data is stored in hardware-provided external flash as a series of dataframes which indicate the type of data and its contents. At present, there are three key dataframes that are defined:

- **High resolution data** contains the raw data collected from the high resolution sensors (accelerometers, gyroscope)

[2] Target component is implementation specific.

- **Low resolution data** contains the raw data collected from the low resolution sensors (barometer)
- **Event timestamps** contain an event identifier (e.g. apogee reached) and a time-of-flight millisecond counter to provide timing information for post-flight analysis

These dataframes are organised by header and payload, as described in Figure 1.2, where frame headers define the boundaries of each payload.

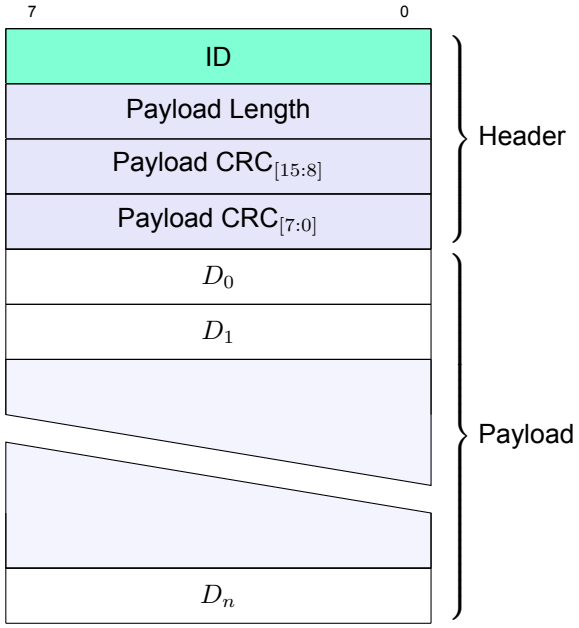

Figure 1.2: Dataframe structure for avionics

A dataframe header consists of four bytes. The first byte defines the frame identifier, indicating the type of dataframe. The second byte defines the payload length, which describes how many bytes are contained within the payload of the dataframe.

The last two bytes in the header contain a CRC encoding of the payload data. This allows for post-processing analysis to better identify and reject invalid or corrupt data.

1.3.1 Interpreting Binary Data

Algorithm 1 provides a simple, pseudocode defined implementation for dataframe validation when performing post-flight data analysis.

This approach simply iterates through every byte of data retrieved from the binary, extracts the bytes that *would* contain the payload length and CRC in a valid dataframe, and compares the expected CRC to the calculated CRC of the assumed payload.

Algorithm 1 : Simple dataframe validation

```
Assume crc16 is a function that accepts some arbitrary binary
series and its length, and returns its 16-bit CRC encoding.

input:  An array of 8-bit binary values, data
output: A valid data payload

foreach byte in data do:
    id[0]     ← *(byte)
    length[0] ← *(byte + 1)
    crc[1:0]  ← *(byte + 2)

    payload ← (byte + 4)

    if crc is crc16(payload, length) then:
        return payload

    else: continue
```

Full implementation details are beyond the scope of this document and may not entirely pertain to the definition of Algorithm 1.

More efficient methods of analysis are possible, this simply serves as a high level informational breakdown of the process of interpreting the binary data.

2 Implementation Details

3 Future Progress

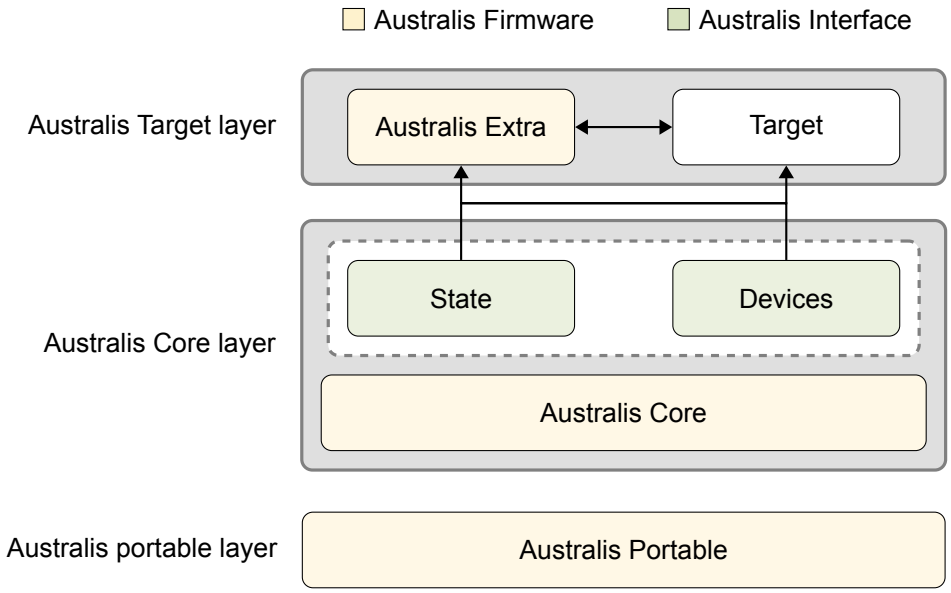

Figure 3.1: Proposed system architecture

4 Document History

Date	Changes Made	Made By
2025/04/20	Update data storage and interpretation	Matthew Ricci
2025/03/16	Create initial document	Matthew Ricci

Appendix

Appendix A: Appendix Item