Design, development and testing of flight software for EIRSAT-1: a university-class CubeSat enabling astronomical research

Maeve Doyle,[a,b,*] Andrew Gloster,[c] Meadhbh Griffin,[a] Mike Hibbett,[d] Jack Kyle,[a] Conor O'Toole,[c] Joseph Mangan,[a,b] David Murphy,[a,b] Nicholas L. Wong,[a] Sai Krishna Reddy Akarapu,[a,b] Rachel Dunwoody,[a,b] Jessica Erkal,[a,b] Gabriel Finneran,[a,b] Jack Reilly,[a,b] Lána Salmon,[a,b] Joseph Thompson,[b,e] Sarah Walsh,[a,b] Brian Shortt,[f] Antonio Martin-Carrillo,[a,b] Sheila McBreen,[a,b] David McKeown,[b,e] William O'Connor,[e] Alexey Uliyanov,[a,b] Ronan Wall,[a,b] and Lorraine Hanlon[a,b]

[a]University College Dublin, School of Physics, Dublin, Ireland
[b]University College Dublin, Centre for Space Research, Dublin, Ireland
[c]University College Dublin, School of Mathematics and Statistics, Dublin, Ireland
[d]Irish Manufacturing Research, Dublin, Ireland
[e]University College Dublin, School of Mechanical and Materials Engineering, Ireland
[f]European Space Agency, ESTEC, Directorate of Science, Future Missions Department, The Netherlands

Abstract. The capabilities of CubeSats have grown significantly since the first of these small satellites was launched in the early 2000s. These capabilities enable a wide range of mission profiles, with CubeSats emerging as viable platforms for certain space-based astronomical research applications. The Educational Irish Research Satellite (EIRSAT-1) is a CubeSat being developed as part of the European Space Agency's Fly Your Satellite! program. In addition to its educational aims, the mission is driven by several scientific and technological goals, including goals related to a novel gamma-ray instrument for the detection of bright transient astrophysical sources, such as gamma-ray bursts. This work provides a detailed description of the software development life-cycle for EIRSAT-1, addressing the design, development and testing of robust flight software, aspects of payload interfacing, and risk mitigation. The described design-to-testing approach was implemented to establish, prior to launch, that EIRSAT-1 can perform its intended mission. Constraints and challenges typically experienced by CubeSat teams, which can impact the likelihood of mission success, are considered throughout this work, and lessons learned are discussed. The aim of this work is to highlight the advanced capabilities of CubeSats while providing a useful resource for teams implementing their own flight software. © *The Authors. Published by SPIE under a Creative Commons Attribution 4.0 International License. Distribution or reproduction of this work in whole or in part requires full attribution of the original publication, including its DOI.* [DOI: 10.1117/1.JATIS.9.1.017002]

Keywords: CubeSat; flight software; EIRSAT-1.

Paper 22085G received Sep. 5, 2022; accepted for publication Feb. 7, 2023; published online Mar. 19, 2023.

1 Introduction

The concept of CubeSats—nanosatellites measured in units (U) of $10\ \text{cm} \times 10\ \text{cm} \times 10\ \text{cm}$ and weighing $\sim 1.3\ \text{kg/U}$[1]—was first proposed in the late-1990s as a teaching tool for university-level students.[2,3] More than 20 years and 1800 launches later,[4] CubeSats are now considered by many to be a disruptive space innovation.[5–7] The CubeSat design specification defines a compact satellite form factor,[1] facilitating the use of commercial off-the-shelf components (COTS) and

*Address all correspondence to Maeve Doyle, maeve.doyle.1@ucdconnect.ie drastically reducing the cost of space entry compared with conventional approaches, which typically employ large, one-off spacecraft.[8] The resulting accessibility of CubeSats drives their growing popularity, with each five-year period since 2000 witnessing 24, 78, 415, and 1109 CubeSats launches, respectively.[4] In parallel, and as a result of recent advances in technology miniaturization, the capabilities of CubeSats have also grown such that their mission objectives now span technology demonstration, communications, Earth observation, and science.[7] Their usefulness for diverse research areas in astrophysics, heliophysics, and deep space exploration are becoming more widely understood and acknowledged.[9–12]

As one example, the multi-messenger era of astronomy, in which astrophysical sources are studied through electromagnetic, gravitational wave (GW), and particle channels, was boosted in 2017 with the first joint detection of a gamma-ray burst (GRB)[13] and a GW signal from the merger of two neutron stars.[14]

There are significant observational challenges to making progress beyond this first detection. The gamma-ray sources are short-lived (tens of seconds long), and their locations are unpredictable. To maximize the chances of more joint GRB and GW detections, all-sky coverage of the gamma-ray sky is required. Although this is mostly achieved by the current fleet of γ-ray satellites (e.g., INTEGRAL,[15] the Neil Gehrels Swift Observatory,[16] Fermi,[17,18] and ASTROSAT[19]), these missions are operating well beyond their intended lifetimes. For example, INTEGRAL was launched in 2002 with a 2-5 year mission lifetime.

For the detection of bright, transient sources such as GRBs, CubeSats offer a low-cost, fast-paced alternative to traditional large gamma-ray astronomy missions. One example is GRBAlpha —a 1U CubeSat that was launched in March 2022 and is equipped with a miniaturized γ-ray sensor[20] to detect GRBs. GRBAlpha also aims to test the in-flight performance of technologies for a constellation of CubeSats, known as CAMELOT.[21,22] As of June 2022, GRBAlpha has successfully detected at least five GRBs. Other examples of GRB-dedicated CubeSat missions include VZLUSAT-2,[23] BurstCube,[24] and HERMES.[25]

To advance the use of CubeSats as viable platforms for gamma-ray astrophysics research, this work presents details on CubeSat flight software (Sec. 2). More specifically, a detailed description of the design (Secs. 3 and 4) and the development and testing (Sec. 5) of flight software for the EIRSAT-1 mission and its primary γ-ray detector payload is presented. Aspects of the software development process dedicated to improving reliability through extensive testing methods are discussed in Sec. 5. Finally, key lessons learned from developing flight software for a CubeSat mission are presented in Sec. 6.

1.1 *Educational Irish Research Satellite*

The Educational Irish Research Satellite (EIRSAT-1), as shown in Fig. 1, is a 2U CubeSat being developed by a student-led team at University College Dublin (UCD).[26] The project, which aims to launch EIRSAT-1 as Ireland's first satellite, is supported by the Education Office of the European Space Agency (ESA), under the second round of the Fly Your Satellite! (FYS!) program.[27] The main objective of the mission is to enhance the capabilities of the national higher education sector in space science and engineering. However, the project is also motivated by a number of technology demonstration and scientific aims,[26] to be achieved with three novel payloads. The ENBIO Module (EMOD) is a thermal materials experiment,[28,29] Wave-Based Control (WBC) is a software-based attitude control test-bed,[30] and the Gamma-Ray Module (GMOD), discussed in Sec. 1.2, is a miniaturized γ-ray detector.[31,32]

The EIRSAT-1 spacecraft consists of custom hardware that was designed in-house for the EMOD and GMOD payloads. An antenna deployment module (ADM),[33] which is responsible for deploying the mission's Ultra High Frequency (UHF)/Very High Frequency (VHF) antenna elements on-orbit, was also developed in-house. The remainder of the spacecraft consists of COTS components supplied by AAC Clyde Space, including the battery, electrical power system (EPS), solar cells, radio transceiver, attitude determine and control system (ADCS), and onboard computer (OBC). Two versions of the EIRSAT-1 spacecraft—an engineering qualification model (EQM) and a near-identical flight model (FM)—are being built as part of this project to reduce the level of risk arising from the team's initial lack of experience with spacecraft development as well as the complexity of the mission's payloads/objectives.[34]

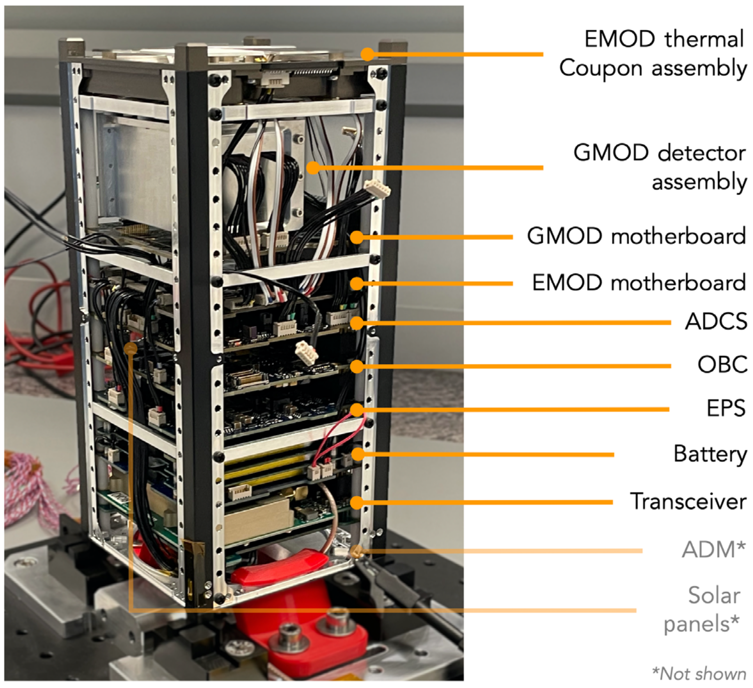

Fig. 1 FM of EIRSAT-1, showing the satellite's constituent components.

1.2 *Gamma-Ray Module*

The primary aims of the GMOD payload are to detect GRBs, assess the on-orbit performance of novel gamma-ray detection technology, and demonstrate the role of CubeSats in the multi-messenger era of astrophysics.

GMOD (Fig. 1) is a miniaturized γ-ray detector that was developed in-house. The payload comprises a motherboard, with the necessary electronics to operate the payload, and a detector assembly.[31,32] This assembly consists of a 25 mm × 25 mm × 40 mm cerium bromide ($CeBr_3$) scintillator crystal, an array of silicon photomultipliers (SiPMs), and low-power read-out electronics, which are all contained within a compact, light-tight enclosure. For each high-energy photon incident on the detector assembly, a time-tagged event (TTE) is produced, with an ADC value corresponding to the photon's energy.[32] The TTEs produced by GMOD on-orbit will be used to understand the payload's γ-ray environment and to study high-energy events, such as GRBs.[32,35,36]

GMOD is expected to detect between 11 and 14 GRBs per year (in an energy range from tens of keV to a few MeV), at a significance greater than 10σ (and up to 32 at 5σ).[31]

2 CubeSat Flight Software

Flight software generally refers to the onboard software responsible for the on-orbit behavior of the complete satellite system, performing a wide range of tasks from subsystem interfacing to data handling and fault management. Its reliable performance is essential to mission success. Although CubeSats are much smaller than conventional spacecraft that are typically employed for astronomical research, the required flight software in many ways faces similar needs and challenges to that of larger missions—flight software complexity is not proportional to the satellite volume.[37] To implement this software in a way that is consistent with the low-cost, fast-paced CubeSat design philosophy, several platforms that assist the software development process (also known as frameworks) have been developed. Some examples are

- **NASA's core Flight System (cFS):**[38] an open-source software development platform created by NASA's Goddard Space Flight Center. It was originally developed to facilitate software development for larger missions, based on the heritage of previously successful NASA missions, but later adapted for Small/CubeSats.[39]

- **KubOS:**[40] a CubeSat-specific framework created by a Texas-based space software company in 2014. Provides tools and libraries to aid quick development projects. It is primarily open-source; however, additional features and support can also be procured.
- **GenerationOne flight software development kit (FSDK):**[41] a framework created by Scottish company Bright Ascension, which is discussed in Sec. 3.3. It is also focused on CubeSats.

These frameworks, some of which are reviewed in comparison studies by Refs. 37 and 42, aim to reduce project costs and development times through software reusability, similar to the approach driving the use of standardized COTS hardware. Alternative approaches to the use of frameworks include re-use of software that was previously developed in-house, use of open-source materials, such as code provided by the Libre Space Foundation,[43] and in-house development from scratch.

However, the lack of specialized literature on the topic has been identified as a potential challenge for CubeSat teams when developing flight software.[44] To address this deficit in knowledge dissemination around flight software development for CubeSats, this paper documents the approach adopted for EIRSAT-1, complementing the existing literature on other CubeSat missions, including Refs. 44–47.

3 Design Drivers

There are several factors that drive a CubeSat's flight software design and development. In the case of EIRSAT-1, these are (i) the mission requirements (Sec. 3.1), (ii) the available resources (Secs. 3.2 and 3.3), and (iii) the planned strategy for operating the mission (Sec. 3.4).

3.1 *Requirements*

Requirements concisely state some goal that "shall" be achieved. More formally, the European Cooperation for Space Standardisation (ECSS) define a requirement as a "documented demand to be complied with."[48] A project's requirements are typically defined during the very early project phase and provide the foundation for the CubeSat design, including the flight software design. Requirements capture, at a high-level, what the satellite system must do to achieve the mission's objectives and the constraints within which it must perform.

Two sets of requirements are associated with EIRSAT-1.

- **Mission specific requirements:** created by the EIRSAT-1 team with the mission's primary objectives in mind. Approximately 200 of these requirements are collated in the EIRSAT-1 technical specification (EIR_TS). These requirements were developed prior to, and then reviewed as part of, the mission's critical design review with ESA and FYS!.
- **Programmatic and technical requirements:** mandatory for all missions participating in the FYS! program. The FYS! 2 design specification (FDS) contains ∼150 requirements. The FDS draws from a number of sources that are important for conventional spacecraft development, as well as CubeSat development, and include the COTS, launcher and deployer specifications, mission registration guidelines, and ESA policies.

Table 1 shows some of the requirements that must be satisfied (at least in part) by the flight software. Prior to launch, the capability of the mission to satisfy these requirements must be formally verified. This is primarily achieved via testing (Sec. 5.3), with the test setup, test procedures, and as-run results being extensively documented.

In addition to the strict set of requirements set out in the EIR_TS and FDS, the project also follows the guidance of ECSS standards, which act as a coherent, single set of standards for use in European space activities. These standards provide exhaustive lists of requirements and recommendations encouraging best practices and are presented in several documents that are specific to different areas of space mission development. Given the extent of the information

Table 1 Example requirements from the EIR_TS and the FDS.

Type	Requirement
EIR_TS	The OBC shall collate and process raw data from the science payloads.
EIR_TS	The OBC shall be able to process raw GMOD data into binned spectra and light curves.
EIR_TS	The OBC shall be capable of protecting requested data, such as potential GRB data.
FDS	The CubeSat shall allow for reprogramming of the onboard software during processing on ground… Note: preferably the CubeSat should also allow for—through a safe and robust procedure—software changes in orbit.
FDS	The CubeSat shall have a timer, set to a minimum of 30 min, before satellite operation or deployment of appendages such as booms, antennas, and solar panels.

provided within these documents, ECSS standards are typically tailored and used by teams in a way that best suits their projects.[44,49,50] For EIRSAT-1, this tailoring is decided on with input from FYS!, but it is largely driven by what is feasibly achievable given the timeline and resources available to the CubeSat project.

3.2 *Computer Hardware*

A spacecraft's hardware design is largely defined during a very early project phase and typically precedes any major software development activities. The flight software design is then driven by the hardware architecture for which it is being developed. For EIRSAT-1, the flight software can be categorized based on hardware, as given in the following.[51]

- **Platform:** flight-ready firmware provided with each of the COTS components by AAC Clyde Space to operate the subsystem, implementing key functions of the subsystem and providing an interface to these functions via I2C communication.
- **Payload:** firmware developed in-house for the GMOD and EMOD microprocessors (i.e. Texas Instruments MSP 430s). The firmware for each payload implements the respective experiment functionality and provides an interface to this functionality via I2C and serial communication.
- **OBC:** Software developed in-house to run on the COTS OBC. This software is responsible for interfacing with the spacecraft's subsystems and controlling the behavior of the satellite system as a whole.

The below work is primarily focused on the OBC-run software, which is referred to as the "main" flight software. The OBC for which this software is developed is provided by AAC Clyde Space. It is built around a MicroSemi SmartFusion2 System-on-Chip and incorporates an ARM Cortex M3 processor. The memory bus consists of 8 MB of magnetoresistive RAM (MRAM) storage, as well as 4 GB of flash memory, both of which are protected against radiation effects via an error detection and correction mechanism. The OBC is equipped with multiple onboard data buses, including I2C and serial/UART.

The AAC Clyde Space OBC was primarily chosen for EIRSAT-1 given its flight heritage as well as the possibility of procuring all COTS components from a single supplier. As an additional advantage, a FSDK supports software development for this OBC.

3.3 *Resources*

As this is Ireland's first satellite, there was no in-house developed software with flight heritage. To reduce risk, the Bright Ascension GenerationOne FSDK was selected for this project. This kit allows for rapid development of flight software, in the C programming language, using a model-based development approach known as component-based development. In this context, a component is a reusable, standalone software module in which a certain set of functions are

Table 2 Advantages ('✓') and potential disadvantages ('×') of kit-driven development.[51]

✓	Schedule	Resources provided with a kit generally support rapid learning and implementation. This benefits a student-led project in particular when the time required to build knowledge and expertise can present a challenge to project schedules.
×	Flexibility	To fully benefit from a kit, a software project should ideally implement many of the services (e.g., code, frameworks, development, test tools, etc.) being provided with the kit. This did not pose a major issue for EIRSAT-1. However, for projects that are less flexible (e.g., with regards to the implementation of certain functionalities), some kits may not be well suited.
✓	Risk	The Bright Ascension FSDK has flown on several missions, gaining many of the kit's components flight heritage. On-orbit validation of the software's performance improves reliability and reduces risk.
×	Cost (of re-use)	A consequence of using a proprietary kit is that no in-house intellectual property is developed with regards to a reusable software framework. A license must be procured for each mission that makes use of such a kit.
✓	Cost (of resources)	The predicted cost of resources that would be required to build the necessary expertise to produce a flight-ready software image without the support of a kit can outweigh the actual cash cost, if any, of the kit. Given the initial lack of experience within the team, as well as the benefits offered by the Bright Ascension FSDK, this was considered to be the case for the EIRSAT-1 project.

implemented. Components are then integrated to form a mission's flight software image. The FSDK is provided with libraries of prevalidated components—many with flight heritage—including components that provide low-level, supplier-specific hardware interfacing through to high-level functionality. The kit is also equipped with development and test tools, such as those for automatic code generation.

Although this kit and kit-driven development in general were considered a good fit for EIRSAT-1 over other development options, some of which are discussed in Sec. 2, this is not necessarily the case for all CubeSat teams. Table 2 highlights the key advantages and potential disadvantages of this approach, which ultimately favored the use of the Bright Ascension FSDK for the EIRSAT-1 project.[51]

3.4 *Concept of Operations*

A system's concept of operations (ConOps) describes the strategy for operating the system from the point of view of the user/operator. Many aspects of a system's ConOps must be implemented, or at least accounted for, in the system's software design. For example, a common ConOps strategy used for space missions is to use operational modes, whereby the mission operator gains an initial understanding about the state of the spacecraft based on the current mode.[44,52–54] Modes also generally dictate which of the satellite's subsystems are active and what activities are ongoing at any point in time. The software implementation of operational modes and mode management for EIRSAT-1 is discussed in Sec. 4.

4 Main Flight Software Design

The OBC's memory is mapped to hold more than one mission image at a time. Embedded flash memory is used to hold a 256 KB "failsafe" image, and part of MRAM is used for two 1 MB "primary' images."[55] The primary image slots are intended for images compiled with all of the components that allow EIRSAT-1 to fully achieve its mission objectives. Two images slots are provided to facilitate on-orbit image updates without the need to overwrite the existing/active image. In contrast, the failsafe image slot is intended to hold a simplified image, with fewer components and only the functionality needed to keep the mission in a stable, operable state.

At start-up, the OBC's bootloader determines which of these images to load based on configurable data stored in MRAM. In the event of multiple or unsuccessful reboots (e.g., as a result of an unstable image), the OBC defaults to operating in failsafe.

The software architecture of the primary mission image is presented in Sec. 4.1, followed by a more detailed description of key components related to onboard operational state management (Sec. 4.2), subsystem interfacing (Sec. 4.3), and fault protection (Sec. 4.4).

4.1 *Flight Software Architecture*

Figure 2 presents a block diagram of the main flight software architecture, in which software components are grouped and placed into tiers based on their function.[51]

- **Management tier:** responsible for managing when the system's functions are called.
- **Capabilities tier:** provides access to the system's main functions.
- **Hardware tier:** facilitates low-level interfaces to the system hardware.

Much of the functionality provided by the two lower tier components in this architecture is standard across different space missions. For instance, all missions require at least some data aggregation and logging functionality as part of their onboard data handling (OBDH) software. As another example, missions employing AAC Clyde Space subsystems must implement similar software to interface with this hardware. To address this, the Bright Ascension FSDK is provided with much of this functionality implemented in (near) flight-ready components. As a result, the in-house development time required for different aspects of the flight software is well represented by the architecture in Fig. 2, with the management tier requiring the most effort.[51] This is for two reasons: (i) in some instances, the management tier functions required more mission-specific software (e.g., to implement EIRSAT-1's ConOps approach [Sec. 3.4]) and (ii) many of the management tier functions are critical to mission success. Therefore, for cases in which kit-provided software was available, significant efforts were still given to developing a

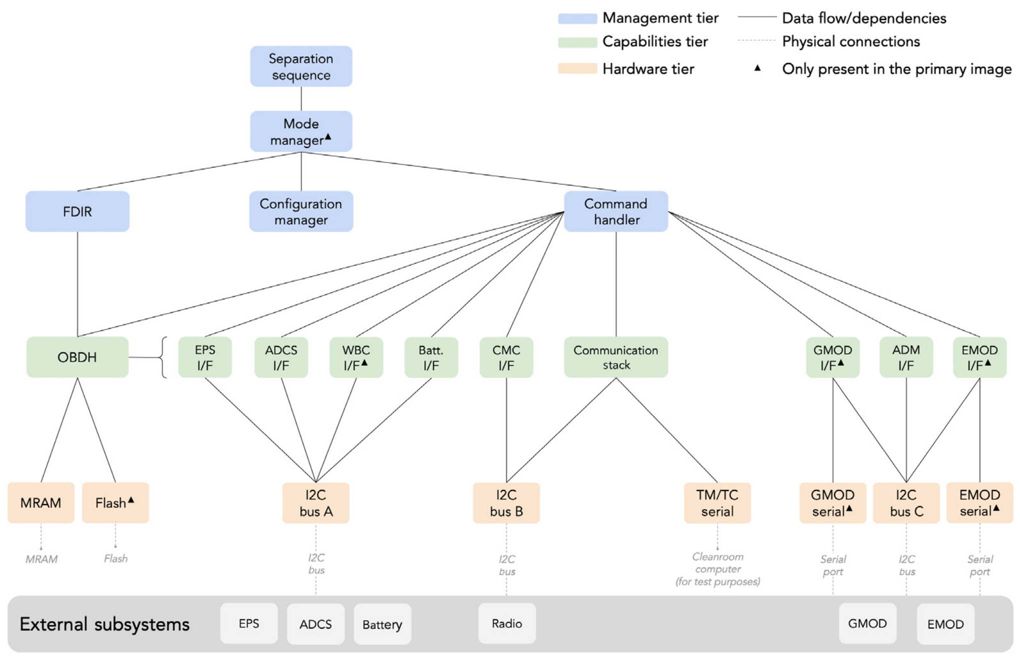

Fig. 2 High-level overview of the main flight software architecture, where solid lines represent the general flow of data/data dependencies in the system, FDIR = fault detection, isolation, and recovery, OBDH = onboard data handling, and I/F = interface, with these components providing an interface to an external subsystem (via physical connections that are given as dashed lines), TM/TC = telemetry/telecommand and "external subsystems" refers to microprocessors on the EPS, ADCS, battery, radio transceiver, GMOD, and EMOD subsystems. Components that are not present in the failsafe image are identified with a triangle (▲).

more in-depth understanding of the software, to ensure that its functionality was used and tested correctly. This was particularly the case when modifications to the software were required. For the capabilities tier, the in-house development time was primarily dedicated to payload software development. The remaining efforts then, across all tiers, was focused on any mission-specific updates required to kit-provided components and bug fixes.

Software components from the management and capabilities tiers that are essential to achieving EIRSAT-1's objectives are now discussed in more detail.

4.2 *Operational State Management*

4.2.1 *Configuration manager*

The configuration manager component manages the persistence of data through OBC reboots. When its functions are called by other components within the image, the configuration manager stores a select amount of data (i.e., stores values of a set number of parameters) to the OBC's non-volatile MRAM. Following a reboot, the persisted data are then loaded from MRAM.

The usefulness of this functionality is best demonstrated by the mode manager component (Sec. 4.2.3), which uses the last known u8_Mode parameter value to determine which operational mode should be loaded following a reboot. Caution must be taken with this functionality as reboots are an important method of recovering software to a known, working state in the event of an anomaly. The more data that are persisted by this component, the less effective a reboot is at recovery. Therefore, only a small sub-set of parameters with persisted states that serve a functional purpose (e.g., u8_Mode) are persisted within the main flight software image.

4.2.2 *Separation sequence*

This component is responsible for the satellite's immediate post-launch behavior—referred to as the "separation sequence"—and is designed to reliably enable two-way communications with the spacecraft via a series of automatic onboard operations. In software, this component is built as a basic finite state machine, in which the required operations occur as part of different states. The sequence of states and their primary tasks are as follows:

1. u8_State : initialization—a number of initialization tasks are first performed. For instance, watchdog functionality (discussed in Sec. 4.4) is enabled to ensure that the ADM firmware is responsive throughout the sequence.
2. u8_State : post-launch wait—to satisfy a mission requirement (Sec. 3.1), a 45-min wait period is observed before antenna deployment is attempted.
3. u8_State : −Y/+Y/−X/+X primary burn—deployment attempts are initiated via I2C commands to the ADM firmware, which is responsible for EIRSAT-1's primary antenna release mechanism.[33] This mechanism involves the 'burning' of resistors on the ADM to melt lines holding the antenna in a stowed state. Through these states, the resistors that are responsible for the release of the four antennas are burned individually, each for a 30-second period.
4. u8_State : between burn wait—radio transmission is enabled if either element door switch on the ADM indicates that the downlink/UHF antennas are released, or n deployment attempts have occurred. These conditions are implemented to prevent radio transmissions while antennas are still stowed as power radiated into the transceiver could damage the component. However, the conditions also eventually prioritize radio transmission to favor the likelihood of acquiring a signal (AOS) from the satellite.

   A 25- or 100-min wait period is then observed. The duration is dictated by the status of the door switches on the ADM, with a longer wait-time being implemented if all antennas are likely deployed.
5. u8_State : secondary burn—a deployment attempt is initiated via I2C communication with the EPS, which is responsible for EIRSAT-1's secondary antenna release mechanism. In this case, the EPS powers one of its power distribution modules, which burns four resistors on the ADM at once, for a 30-second period.

During the separation sequence, steps 3 to 5 are repeated. Deployment attempts are periodically carried out, alternating between the primary and secondary antenna release mechanisms, and are separated by a wait period (that is not a multiple of the orbital period so that deployment attempts occur at different ambient temperatures). This chain of functionality is represented in Fig. 3. The sequence will be configured to initialize following the first-boot on-orbit. A telecommand is required to terminate the sequence, following which the sequence's u8_State: finished state will be persisted by the configuration manager.

Given the importance of this component in achieving AOS, its reliability in the event of a fault is essential, motivating a number of design decisions. For instance, (i) the ADM watchdog functionality ensures the ADM firmware can reliably be commanded throughout the sequence, ensuring the primary antenna release mechanism is operable; (ii) the separation sequence only ceases to cycle through deployment attempts in response to a telecommand, instead of automatic onboard logic, which might use, for example, the status of the ADM's door switches. This approach removes any reliance on additional functionality/hardware and reduces the risk of the sequence ceasing early, for example prior to a successful deployment due to a faulty switch reading; (iii) the wait time implemented between deployment attempts allows for time for battery charging between resistor burns but also ensures an eventual deployment attempt in favorable temperature conditions; and (iv) both the failsafe and primary images are equipped with this component and are configured at launch to enter the sequence's u8_State : initialization state.

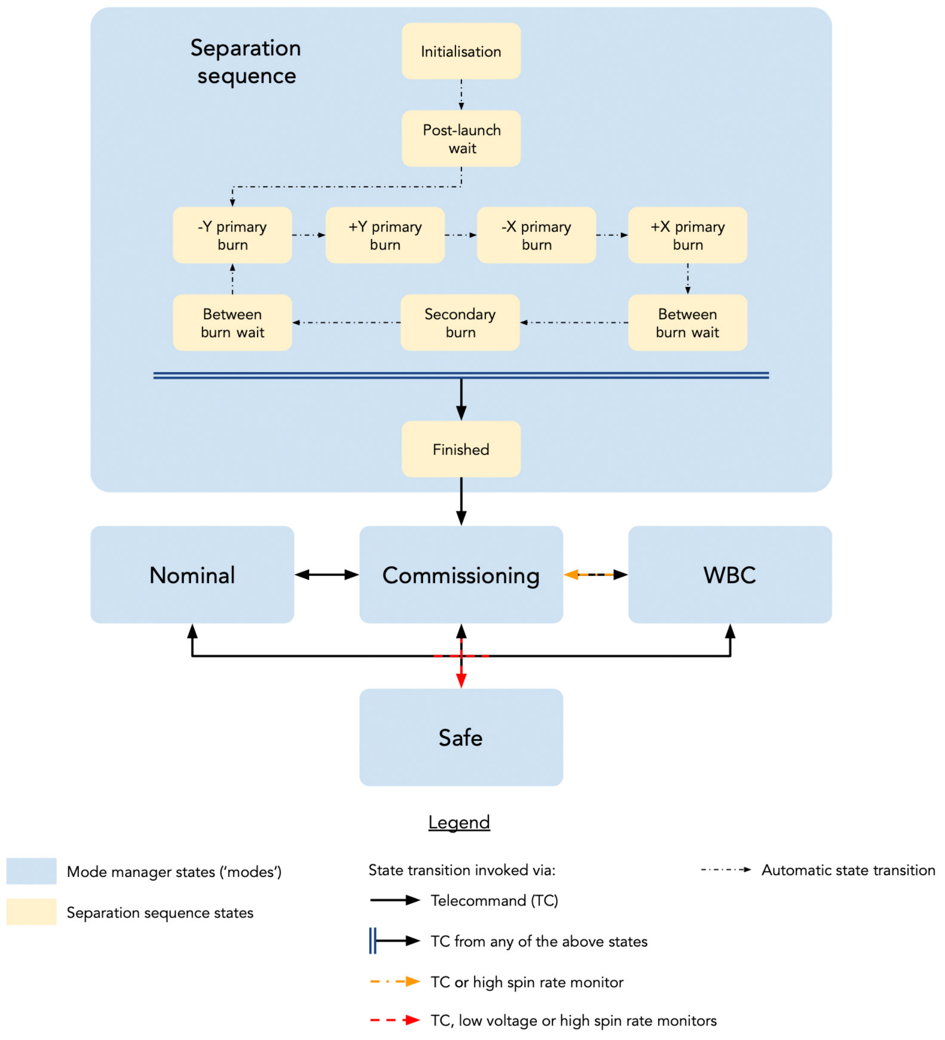

Fig. 3 Separation sequence states and operational modes.

4.2.3 *Mode manager*

The mode manager component is responsible for determining and setting the operational mode of EIRSAT-1. Modes allow operators to rapidly assess the operational state of the spacecraft and determine which subsystems are active and what, if any, autonomous activities have recently occurred. The mode manager is implemented in software as a finite state machine that sets up a defined configuration while transitioning between states via mode-specific exit and entry functions. The modes implemented in EIRSAT-1's mode manager are given in the following:

- u8_Mode : separation sequence—this mode is entered following the first-boot on-orbit, while the persisted mode parameter remains the u8_Mode : separation sequence. Its key purpose is to power on/enable any subsystems/functions needed for the separation sequence. The spacecraft continues to operate in this mode, with the separation sequence on-going, until a telecommand is received instructing a mode change, at which point the mode is exited and the separation sequence is put into its u8_State : finished state.
- u8_Mode : commissioning—this mode is primarily intended for use during the post-launch and early operations phase following exit from u8_Mode : separation sequence. For this reason, to ensure that repeated antenna deployment attempts have ceased, the separation sequence is again placed into its u8_State : finished state on entry to this mode (this action is solely implemented for added redundancy as it is also carried out on exit from u8_Mode : separation sequence). Fault detection, isolation and recovery (FDIR) functionality (Sec. 4.4) is also enabled on entry to this mode to allow for automatic safe mode transitions, or "triggers."
- u8_Mode : nominal—the software implementation of this mode is similar to that of commissioning mode; however, u8_Mode : nominal indicates that nominal mission operations (i.e., running of the GMOD and EMOD payloads) are on-going.
- u8_Mode : WBC—WBC is one of EIRSAT-1's three primary experiments (Sec. 1.1), the aim of which is to test the performance of a novel attitude control algorithm, implemented in the main flight software, with COTS ADCS hardware. The software implementation of u8_Mode : WBC is similar to that of commissioning and nominal mode. However, on entry to this mode, control of the spacecraft's attitude is transferred from the COTS ADCS controller to the custom WBC algorithm. On exit from u8_Mode : WBC, attitude control reverts back to the COTS ADCS. Implementing this functionality within a mode allows FDIR mechanisms to be implemented on entry to/exit from the mode (Sec. 4.4). From a ConOps perspective, this approach also provides clarity to operators when the WBC experiment is active.
- u8_Mode : safe—safe mode's primary purpose is to act as a recovery mode to which the mode manager automatically defaults in response to anomalous behavior or following a reboot in which the previously persisted mode $\neq$ u8_Mode : separation sequence. Therefore, on entry to this mode, the mode manager places the satellite into a "stable" state in which non-critical subsystems/functions are deactivated. For instance, the GMOD and EMOD payloads are powered off, and the ADCS is set to operate in standby mode. As Fig. 3 shows, safe mode is accessible from all modes, excluding u8_Mode : separation sequence. This exclusion is made to ensure that the separation sequence proceeds in full post launch.

Unlike the separation sequence, in which state changes are almost entirely autonomous, mode transitions are invoked via both autonomous and commanded operations in the mode manager component and do not follow a set sequence. The different modes defined for EIRSAT-1 and the operations that invoke mode transitions are shown in Fig. 3.

4.3 *Subsystem Interfacing*

The Bright Ascension FSDK comes with software components for interfacing with some supplier-specific COTS hardware, including AAC Clyde Space subsystems. Therefore, kit-provided components were implemented in the main flight software's capabilities tier to interface with the

EPS, ADCS, transceiver, and battery, with only minor updates made, e.g., to implement bug fixes. In contrast, software components for the custom payloads were developed entirely in-house.

Section 4.3.1 provides an overview of the GMOD software component, which provides an interface to the payload and performs key tasks associated with the payload's scientific objectives. Further details about the GMOD firmware with which this component interfaces are provided in Refs. 32, 35, and 36.

4.3.1 *GMOD*

GMOD (Sec. 1.2) is the primary scientific payload, with a scientific goal of detecting GRBs. Key requirements driving the design of the software components are given in Table 1. To satisfy these requirements, the GMOD software component is designed with the following functionality.

Data handling and storage. When configured to run its primary experiment, the GMOD payload collates 256-byte 'pages' of TTE data, which are stored to its flash memory.[32] The expected rate at which data will be produced on-orbit ranges from ∼2 to 30 pages/second, which correspond to the γ-ray background (i.e., a ∼50 Hz TTE rate) and a 20σ GRB, respectively. A parameter in the payload firmware, that is settable via I2C communication, controls whether GMOD periodically sends, or "streams," data from GMOD's flash memory to the OBC. The fastest period at which this can occur is also controlled by a configurable parameter (0.2 s is configured by default, which allows the page streaming rate to keep pace with potential page production rates). This streaming approach was chosen over an implementation in which the OBC requested pages one at a time via I2C commands to reduce the computational resources required to transfer science data to the OBC.

Although I2C communication is adopted for all commanded operations of GMOD, the pages of TTE data are transferred to the OBC via serial, with each payload having access to an independent port. This implementation was chosen to minimize traffic on the payloads' I2C bus, which is shared by both EMOD and GMOD. As pages are received, GMOD's main flight software component first assesses the validity of the data (e.g., by comparing a checksum transferred with the data to a calculated checksum) before parsing the data into individual TTEs, each with a 4-byte coarse timestamp, 4-byte fine timestamp, and 12-bit ADC value.[32] These TTEs are then passed to a number of different functions within the component, as described below and summarized in Fig. 4.

Finally, this software component collates the TTE data received into 16-page "sectors," before storing the data to the OBC's flash memory for later downlink to the ground station. Approximately 1.25 GiB of memory was allocated for this purpose; this is sufficient for holding $\sim 35-40$ days worth of data.

Using calibration data obtained prior to launch, TTE ADC values will be converted to physical energy units on the ground following downlink of the data. GMOD's time is synced with that of the OBC during the experiment setup and will be maintained using the output of the OBC's pulse per second pin. TTE timestamps will be converted to UTC on the ground from the onboard time (OBT) frame, using OBT and GPS time data that is periodically logged to onboard storage as well as transmitted as part of a beacon. Because GMOD's main flight software component uses OBT and raw ADC values in its operation, these TTE conversions do not need to be implemented onboard the spacecraft.

Light curve and spectrum generation. The mission will make use of one primary ground station, located at UCD (53.3065°N, 6.2255°W), allowing for ∼30 min of communication time with the spacecraft per day. As a result, when prolonged nominal operations are achieved, GMOD will likely produce more TTE data than is feasible to downlink. For this reason, in addition to logging sectors of "raw" TTE data for downlinking, GMOD's main flight software component also packs the TTE data into more compressed data products, namely light curves and spectra (Fig. 5).

Two buffers are defined in the software to build these light curves and spectra. The light curve buffer is 2000 bytes in size to provide 1000 uint16 time bins, and the spectrum buffer is

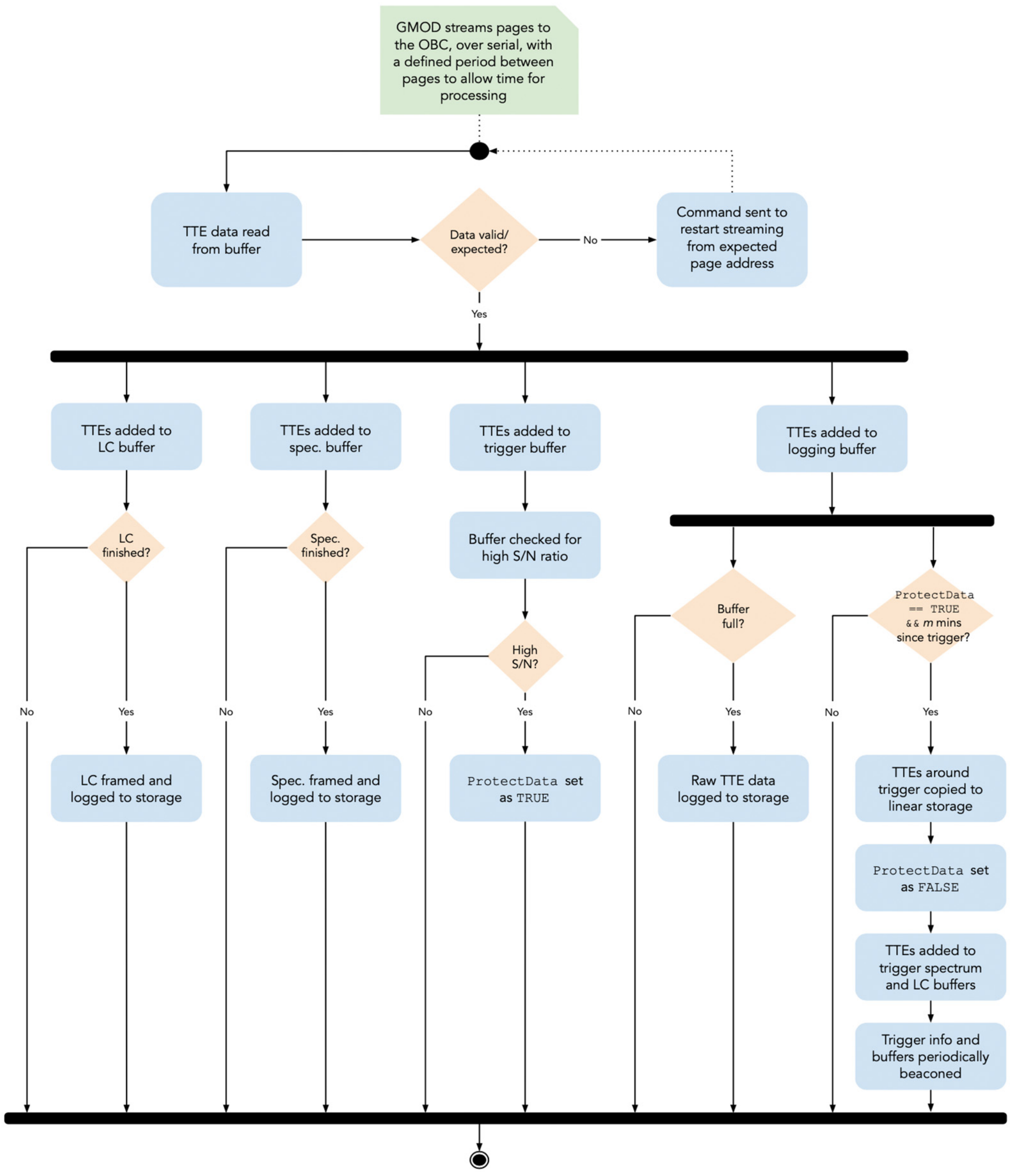
GMOD streams pages to the OBC, over serial, with a defined period between pages to allow time for processing
TTE data read from buffer
Data valid/ expected?
No
Command sent to restart streaming from expected page address
Yes
TTEs added to LC buffer
TTEs added to spec. buffer
TTEs added to trigger buffer
TTEs added to logging buffer
LC finished?
Spec. finished?
Buffer checked for high S/N ratio
High S/N?
Buffer full?
ProtectData == TRUE && m mins since trigger?
No
Yes
LC framed and logged to storage
Spec. framed and logged to storage
ProtectData set as TRUE
Raw TTE data logged to storage
TTEs around trigger copied to linear storage
ProtectData set as FALSE
TTEs added to trigger spectrum and LC buffers
Trigger info and buffers periodically beaconed

Fig. 4 Data acquisition/parsing process implemented in GMOD's main flight software component.

480-bytes, with 240 uint16 ADC/energy bins. As page data are received and parsed into individual TTEs, this component adds to these buffers, separating events into bins based on their timestamp and ADC/energy. By default, the GMOD component generates 20-min light curves and spectra; however, the time bin widths and integration times used to generate these products are configurable with millisecond and second resolution, respectively.

When the buffers are "full" (i.e., when the last time bin of the light curve is reached, and the spectrum integration time has elapsed), frame data are appended to the buffers with information on the generated data products before the data are logged to the OBC's flash memory for later downlink.[32] Following logging, a new light curve and spectrum are started.

To manage constraints on two-way communication time with the spacecraft, these light curves and spectra will regularly take downlink preference over the sectors of raw TTE data. However, in some cases, the TTE data will be assigned higher priority. Examples of such cases include (i) during commissioning, when the less compressed data type will be more useful for assessing the payload's health and performance; (ii) when performing a more detailed analysis on EIRSAT-1's γ-ray environment; (iii) if anomalous results/features are observed in the data; and (iv) when a GRB is detected (discussed below).

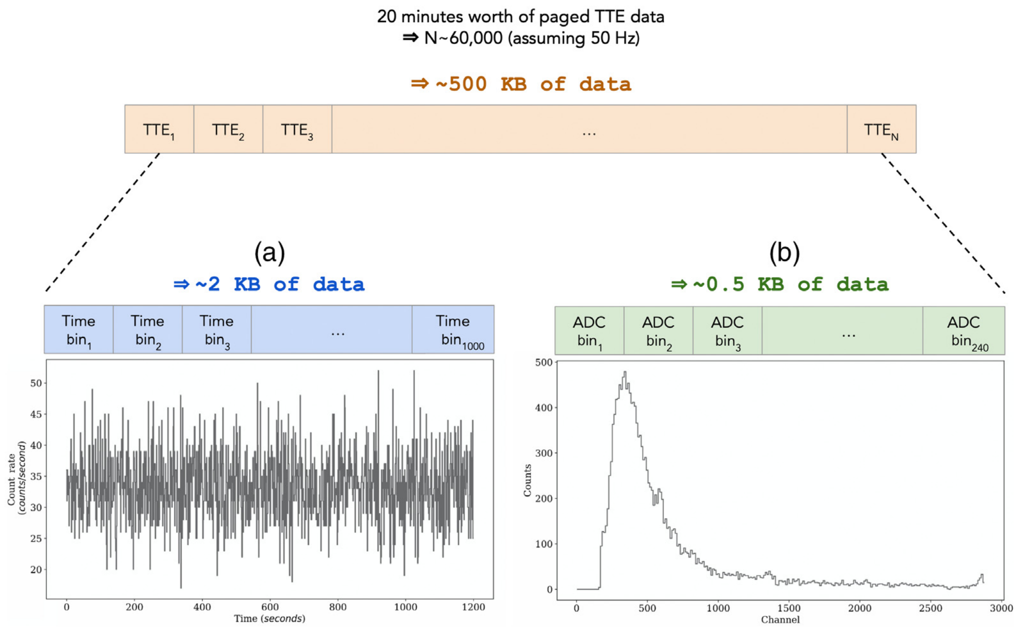

Fig. 5 Data compression carried out by the GMOD main flight software component, showing an example 20-min light curve (a) and spectrum (b) produced by the component from raw/paged TTE data.

GRB triggering. An additional capability of the GMOD main flight software component is triggering, in which the software can identify and act on high signal-to-noise (S/N) ratios within the TTE data. To achieve this, the parsed page data are also added to a separate "triggering" light curve buffer, measuring up to 8192 uint16 time bins. Unlike the light curves discussed above, which are built, logged, and restarted every n minutes, the triggering light curve is implemented as a circular buffer, which is continuously updated with newer data at the expense of the oldest data in the buffer.

The time bins comprising this light curve are divided into three segments, referred to as the "background window," the "signal window" and the "signal window offset," which separates the background and signal segments by a configurable number of bins. As event data are added to the triggering light curve, the sum of events in these segments is calculated and passed to a triggering algorithm

$$\text{u32\_Sig} = \frac{(N \times \text{Signal}) - \text{Background}}{\sqrt{(N^2 \times \text{Signal}) + \text{Background}}} \tag{1}$$

where u32_Sig is the calculated significance of the signal within the signal window, Signal $= \Sigma$ (TTEs in the signal window bins), Background $= \Sigma$ (TTEs in the background window bins), and $N = \frac{\text{Background window size}}{\text{Signal window size}}$ (i.e., a scaling factor given the ratio of the background window size, in terms of bins, compared with the signal window).

If a high S/N ratio is detected in the signal window through this algorithm (i.e., if u32_Sig > u32_SigThreshold, where u32_SigThreshold is configurable), the component marks that a trigger has occurred. The data acquisition process then continues as normal for a configurable time period to allow the OBC to acquire some post-trigger data before the following tasks are performed:

- m sectors of TTE data, that were logged to the OBC's flash memory as part of the normal data acquisition process, are copied (or 'protected') to a separate location in flash memory. This memory is dedicated to potential GRB data and can store data associated with up to 40 triggers at a time. The aim is to isolate high priority data for downlinking and protect it against the possibility of being overwritten by less significant, background TTE data.

Configurable parameters in the software specify the time period around the trigger and the number of sectors (i.e. m) to be protected.

- As m sectors of TTE data are read in by the component (i.e., as they are copied from one location in flash memory to another), a lightweight light curve and spectrum are generated. Information about the trigger (e.g., the peak S/N ratio recorded, its duration, and where the protected TTE data are located in flash memory), as well as the light curve and spectrum, are logged to the OBC's flash memory to enable rapid downlinking and analyses. In contrast to the non-triggered light curves and spectra (Fig. 5), the start and end times of the triggered light curve and spectrum are centered around the high S/N event.
- To rapidly communicate the trigger to the astronomical community (e.g., through the Gamma-ray Coordination Network[56]), the light curve and spectrum are also transmitted by the spacecraft as part of a periodic telemetry beacon that is automatically transmitted every 1.5 min. This allows key information about the detection to be disseminated rapidly. It also provides advanced notice to the mission control team, so preparations can be made to prioritize downlinking of GRB data during the next available communication windows. In particular, the m sectors of TTE data "protected" in the OBC's flash memory will be downlinked such that a more comprehensive analysis of the GRB can be conducted on the ground.

More information about the planned on-orbit operation of GMOD is provided in Ref. 57.

4.4 *Fault Protection*

The Bright Ascension FSDK is equipped with a number of features and components intended to help implement autonomous FDIR methods, including:[55]

- **Events:** data products produced to indicate the occurrence of a significant event on board.
- **Monitors:** periodically poll and compare parameter values with user-defined upper and lower limits, raising an event if either limit is exceeded.
- **Event actions:** automatically trigger some defined functionality in response to a specific onboard event.
- **Periodicactions:** periodically trigger some defined functionality.

Table 3 provides examples of the integration of these autonomous FDIR features into the main flight software to reduce the risk of mission loss.

Fault tree analyses were performed to (i) better understand the spacecraft's expected on-orbit behavior in the event of a fault; (ii) determine how the flight software could be improved to ensure a robust system; and (iii) assess the mission's ability to recover, at least to a "safe" state, using the FDIR functions.

5 Development and Testing

Development and testing of the main flight software largely proceeded in parallel from the outset. Introducing software test and verification activities from an early stage in the development process has several advantages, including (i) improved understanding of the expected performance of the software as part of the full satellite system through hardware-in-the-loop (HIL) testing; (ii) rapid identification of bugs and design issues; and (iii) building confidence in the flight software before proceeding to more formal test and requirement verification campaigns.

5.1 *Setup and Tools*

5.1.1 *Development environment*

The Eclipse[58] integrated development environment for C/C++ developers was chosen to implement the flight software. This choice was driven by the availability of Eclipse plug-ins, which

Table 3 Examples of FDIR functionality implemented in EIRSAT-1's main flight software, where TC = telecommand, S/C = spacecraft, and *n* denotes parameters that are configurable via TC.

Name	Description	Enabled
ADM alive check	A PeriodicAction polls the ADM firmware for its firmware version. If no response is received following *n* consecutive attempts to poll the parameter, the microcontroller hosting the firmware is power-cycled.	While u8_Mode = separation sequence.
Safe mode entry triggering	A monitor periodically polls and checks the battery voltage and gyro rate parameters against predefined upper/lower limits. If a limit is exceeded, an event is raised, which then triggers a transition to safe mode via an EventAction.	While u8_Mode = commissioning, nominal, and WBC.
WBC mode exit triggering	A monitor periodically polls and checks the gyro rate parameters against predefined upper/lower limits. If a limit is exceeded, an event is raised, which then triggers a transition to commissioning mode via an EventAction.	While u8_Mode = WBC.
No TC watchdog	A PeriodicAction checks the number of TCs (N_{TC}) received by the S/C and resets a timer if N_{TC} has increased since the last check. However, if N_{TC} does not increase and the timer reaches *n* days, a full S/C power-cycle is invoked.	Throughout the mission.
EPS watchdog	A PeriodicAction commands the EPS to reset a watchdog timer (managed by the EPS firmware), preventing a watchdog (implemented in the firmware) from power-cycling the S/C.	Throughout the mission.

provide automatic-code generation capabilities, with the Bright Ascension FSDK. This tooling also generates "spacecraft databases," which list all of the accessible parameters and functions provided by the flight software. These databases can then be used with mission control software to interface with the spacecraft.

5.1.2 *Version control*

Git[59] and GitHub[60] are used to track day-to-day software development activities. Git releases mark more significant milestones in the development process. Software releases are made prior to the project's formal test and requirement verification campaigns, with a given software version being used throughout the campaign. A software configuration file is separately maintained in Google sheets to capture, at a high-level, any updates made to the software components between releases. Google sheets were adopted for this purpose over other, more conventional methods (e.g., a GitHub wiki or more detailed Git release notes) primarily as the Bright Ascension FSDK auto-generates documentation that includes a csv file listing an image's constituent components. The presentation of EIRSAT-1's main flight software as a simplified list of components has been particularly useful for communicating summaries of software updates to ESA/FYS!. Furthermore, Google sheets are easily shared (with limited access/permissions) to facilitate reviews by external ESA experts. In the absence of further software updates following testing, the released software will be used for flight.

5.1.3 *Continuous integration*

Using a Docker image with the necessary build tools, CircleCI[61] was integrated with GitHub to automatically build the flight software and perform basic checks each time a commit is made to the Git repository. If the build succeeds and checks are passed, merging of Git branches into the main development branch is "unlocked." This process alerts developers to issues with their code and protects the main development branch. For EIRSAT-1, these checks include standard compiler checks and unit tests (discussed in Sec. 5.3). However, more rigorous checks, for instance, using static code analysis tools, can also be implemented. The flight software images built as part of this process are used for HIL test activities to ensure that images programmed onto the OBC are generated using a consistent build environment.

5.1.4 *Testing*

The following setups and tools are used for testing throughout the software development process.

- A unit testing framework (Sec. 5.3) is provided with the Bright Ascension FSDK.
- Software developed within the FSDK can be built for a variety of platforms, including Linux. As a result, platform-independent functionality (i.e., functionality that does not require an AAC Clyde Space hardware interface) was tested locally on developers' desktops, reducing the demand on the CubeSat hardware.
- HIL setups are available for testing when the necessary hardware is not otherwise in use. HIL testing primarily makes use of a FlatSat configuration, in which the subsystems of the satellite are assembled on a single large motherboard or series of motherboards laid out horizontally.[62] From a software development perspective, a key benefit of the FlatSat is that elements of the hardware/software can be tested individually. For debugging purposes, the tabletop configuration also ensures that components are easily accessible throughout testing.

  During HIL tests, software images are programmed onto the OBC via a JTAG programmer, and communication with the OBC is primarily achieved via a serial link to an external desktop (rather than a radio link via the spacecraft's transceiver/antenna). Debug output by the software image (over a serial port on the OBC) is a key functionality during test activities.

  HIL tests are also carried out on the fully assembled satellite "stack," as shown in Fig. 1. In this case, the connections to the OBC are achieved via an umbilical cable that connects to an external port on the spacecraft.

Where required, mission control software provided by Bright Ascension, along with a spacecraft database, is used to facilitate TM/TC communication with the flight software.

5.2 *Life Cycle*

The main flight software was produced following an iterative and incremental development process. The timeline over which this development took place is strongly tied to the mission's two-model philosophy. More specifically, a flight-worthy version of the software was "frozen" prior to the EQM test and requirement verification campaign. By aiming to freeze a (near) flight-ready version of the software for EQM testing, the main flight software is extensively tested twice, through both the EQM and FM test campaigns, providing confidence in the software design and implementation. The risk of project delays arising from software development is also reduced by this parallel development approach.

5.3 *Tests*

Testing of the main flight software was undertaken throughout the development process.

5.3.1 *Unit testing*

In unit testing, individual units, corresponding to an individual software component, of source code are tested. Unit testing allows the behavior of blocks or units of code to be verified prior to further development or other test activities. It also provides testing for code that otherwise might not be readily assessed. A component's run-time environment is mimicked, and its functions are then called, with inputs where required. The resulting output, or behavior, of the functions is then compared with the expected output. Unit testing is implemented using a framework provided with the Bright Ascension FSDK that makes use of the CMock and Unity[63,64] testing tools.

A further benefit of unit testing relates to code changes, in which previously implemented tests can be run after a change to inspect if the performance of the component was impacted and whether the impact was as expected. For EIRSAT-1, this functionality is implemented using CirceCI (Sec. 5.1.3), in which unit tests are automatically compiled and run and the results are checked each time a commit is made to the Git repository in which the software was developed. When a compile or unit test fails, CircleCI automatically emails the software developer to alert them to this failure, so it can be resolved before the code is merged into the main software development branch.

For kit-provided components (many with flight heritage), in which unit tests were also provided, test coverage ranges between ∼10 and 100% and averages ∼70%. Due to resource constraints, unit test development has focused on in-house developed components that are critical to EIRSAT-1's on-orbit performance and safety, particularly the mode manager and separation sequence components. Using the gcov test coverage program to assess the quality/completeness of the tests, 76 – 96% coverage was achieved for these components. Lower coverage metrics are currently achieved for less critical in-house developed components, including those that implement the GMOD, EMOD, and WBC experiments. Testing of these components through, for example, HIL testing is described in Sec. 5.3.2.

5.3.2 *Integration and HIL testing*

During integration tests, software components are brought together to form a software image. This image is then run on either the developer's desktop or on the OBC to test the functionality of the integrated components.

Integration testing also commonly includes other spacecraft hardware to allow for HIL testing. As many of the main flight software components interact with, and use data from, the spacecraft's hardware, HIL tests provide the most representative environment to test the software's performance.

In addition to HIL tests performed during the software development process, the main flight software was also tested with the spacecraft's hardware as part of test campaigns during which the mission's requirements were formally verified.[34] Although these tests were primarily focused on the mission's performance as a whole, their success required the reliable performance of the main flight software. The following tests were included.

- **Functional testing:** in which key functions of the satellite's subsystems are tested.[65]
- **Mission testing:** in which the performance of the complete satellite system, as configured for flight, is tested as part of a mission simulation.[66,67]
- **Environmental testing:** in which the satellite is subject to representative spaceflight conditions, including vibration and thermal vacuum testing, with functional checks prior to, during, and after the tests.[68]

6 Lessons Learned

EIRSAT-1's main flight software was developed with an overarching aim of maximizing the likelihood of mission success given the project's highly constrained resources and initial lack of expertise. The success of the approaches taken was demonstrated through a test campaign with an EQM of the spacecraft, in which the flight software largely performed as intended.[69] Further confidence in the software's performance will be achieved via the FM test campaign. Key lessons learned from the experience to date are given in the following.

1. **Early and continuous unit testing:** the benefits of unit testing are discussed in Sec. 5.3. Although unit testing requires time and resources to implement, which typically poses a challenge to CubeSat teams, it should not be overlooked as it is a highly effective tool for improving software reliability.

2. **Early HIL testing:** many of the more significant software bugs and issues were identified during HIL testing. In some cases, these bugs/issues took months of effort to overcome. Had these been identified at a later stage, the project's schedule may have been impacted.
3. **Development options:** the EIRSAT-1 team chose an FSDK to overcome a number of challenges associated with the project (e.g., student-led with limited resources, no in-house experience from previous missions, and multiple novel and complex payloads).[51] Using the FSDK allowed for more focus on the development of software for the mission's payloads. Not all teams may face the same challenges, and many software development options are available and should be assessed early in a project to ensure that the correct approach is implemented from the outset.[37,42,51]
4. **Wider knowledge about the flight software behavior within the team:** given the resource constraints typically experienced by CubeSat projects, only a small sub-set of team members generally work on the flight software. However, in many cases, EIRSAT-1's flight software team benefited from the wider team's input to the software design, development, and testing. For example, meetings with the wider team were held to discuss, at a high-level, the design of the separation sequence to ensure that the design of this software component was robust against any foreseeable failure scenarios. In many circumstances (e.g., during testing) the team must be knowledgeable about operating and interfacing with the software/spacecraft via telecommands. For this reason, efforts should be made wherever possible to incorporate the wider team in the software development process.

7 Conclusions

For certain applications, CubeSats offer an alternate and more accessible route to space in comparison with conventional satellites, which typically require highly specialized facilities and are more costly to develop. Advances in miniaturized technologies have allowed the capabilities of CubeSats to grow, widening the scope of this small satellite platform well beyond education to encompass Earth observation, communications, astrophysics, and heliophysics. To further advance the use of CubeSats as capable and reliable platforms for scientific research, this work presents key details on the design, development, and testing of robust flight software for the EIRSAT-1 mission. The reliable performance of the software methods implemented was demonstrated through a comprehensive test campaign with an EQM of the EIRSAT-1 spacecraft. An FM test campaign is the next step in the verification of the flight software for the mission.

Acknowledgments

The EIRSAT-1 project is supported by ESA Education, under the second round of the FYS! program. The authors thank the Bright Ascension support team for their continued support and guidance. The project acknowledges all students who have contributed to the development of EIRSAT-1 and support from Parameter Space Ltd. M.D., R.D., D.M., L.S., and J.T. acknowledge support from the Irish Research Council (Grant Nos. GOIPG/2018/2564, GOIPG/2019/2033, GOIPG/2014/453, GOIPG/2017/1525, and GOIPG/2014/684, respectively). N.W. acknowledges support from Science Foundation Ireland (SFI) (Grant No. 19/FFP/6795). G.F. acknowledges support from a scholarship associated with the UCD Ad Astra fellowship program. D.M., A.U., J.M., and S.M. acknowledge support from SFI (Grant No. 17/CDA/4723). S.W. acknowledges support from ESA (PRODEX contract number 400012071). J.E. and J.R. acknowledge scholarships from the UCD School of Physics. L.H. acknowledges support from SFI (Grant No. 19/FFP/6777) and support from EU H2020 AHEAD2020 project (Grant No. 871158). This study was supported by ESA's Science program (contract 4000104771/11/NL/Cbi) and by ESA's PRODEX program (contract number C 4000124425). This work is based on content previously reported at the SPIE Software and Cyberinfrastructure for Astronomy VII conference, and will appear in the conference proceedings.

References

1. California Polytechnic State University, "CubeSat design specification (CDS) Rev 13, CP-CDS-R13," (2015).
2. H. Heidt et al., "CubeSat: a new generation of picosatellite for education and industry low-cost space experimentation," in *Proc. 14th Annu. AIAA/USU Conf. Small Satellites, SSC00-V-5* (2000).
3. National Aeronautics and Space Administration (NASA), *CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers*, 1st ed., NASA CubeSat Launch Initiative (2017).
4. E. Kulu, "Nanosats database," https://www.nanosats.eu/ (accessed 11 May 2022).
5. L. Summerer, "Signs of potentially disruptive innovation in the space sector," *Int. J. Innov. Sci.* **3**(3), 127–140 (2011).
6. T. H. Zurbuchen, "Achieving science with CubeSats: thinking inside the box (conference presentation)," *Proc. SPIE* **9978**, 997802 (2016).
7. T. Villela et al., "Towards the thousandth cubesat: a statistical overview," *Int. J. Aerosp. Eng.* **2019**, 1–13 (2019).
8. C. Cappelletti, S. Battistini, and B. Malphrus, *CubeSat Handbook: From Mission Design to Operations*, Elsevier (2020).
9. E. L. Shkolnik, "On the verge of an astronomy CubeSat revolution," *Nat. Astron.* **2**(5), 374–378 (2018).
10. K. Woellert et al., "Cubesats: cost-effective science and technology platforms for emerging and developing nations," *Adv. Space Res.* **47**(4), 663–684 (2011).
11. A. Poghosyan and A. Golkar, "CubeSat evolution: analyzing CubeSat capabilities for conducting science missions," *Progr. Aerosp. Sci.* **88**, 59–83 (2017).
12. P. F. Bloser et al., "CubeSats for gamma-ray astronomy," arXiv:2212.11413 (2022).
13. P. Mészáros, "Gamma-ray bursts," *Rep. Progr. Phys.* **69**(8), 2259–2321 (2006).
14. B. P. Abbott et al., "Gravitational waves and gamma-rays from a binary neutron star merger: GW170817 and GRB 170817A," *ApJL* **848**, L13 (2017).
15. C. Winkler et al., "The INTEGRAL mission," *Astron. Astrophys.* **411**, L1–L6 (2003).
16. N. Gehrels et al., "The swift gamma-ray burst mission," *Astrophys. J.* **611**(2), 1005–1020 (2004).
17. C. Meegan et al., "The fermi gamma-ray burst monitor," *Astrophys. J.* **702**(1), 791–804 (2009).
18. W. B. Atwood et al., "The large area telescope on the fermi gamma-ray space telescope mission," *Astrophys. J.* **697**(2), 1071–1102 (2009).
19. K. P. Singh et al., "ASTROSAT mission," *Proc. SPIE* **9144**, 517–531 (2014).
20. A. Pál et al., "GRBAlpha: a 1U CubeSat mission for validating timing-based gamma-ray burst localization," *Proc. SPIE* **11444**, 114444V (2020).
21. N. Werner et al., "CAMELOT: cubesats applied for measuring and localising transients mission overview," *Proc. SPIE* **10699**, 106992P (2018).
22. M. Ohno et al., "CAMELOT: design and performance verification of the detector concept and localization capability," *Proc. SPIE* **10699**, 1069964 (2018).
23. J. Ripa et al., "*GRB 220320A: detection by VZLUSAT-2*," GRB Coordinates Network 31803, p. 1 (2022).
24. J. S. Perkins et al., "BurstCube: a CubeSat for gravitational wave counterparts," *Proc. SPIE* **11444**, 114441X (2020).
25. F. Fiore et al., "The HERMES-technologic and scientific pathfinder," *Proc. SPIE* **11444**, 114441R (2020).
26. D. Murphy et al., "EIRSAT-1 – The Educational Irish Research Satellite," in *Proc. 2nd Symp. Space Educ. Activities (SSEA), SSEA-2018-73*, Budapest, Hungary, (2018).
27. J. Vanreusel, "Fly your satellite! The ESA Academy CubeSats programme," in *ITU Symp. & Workshop on Small Satellite Regulation and Commun. Syst.* (2016).
28. K. Doherty et al., "High-temperature solar reflector coating for the solar orbiter," *J. Spacecraft Rockets* **53**, 1–8 (2016).
29. K. A. J. Doherty et al., "Flat absorber coating for spacecraft thermal control applications," *J. Spacecraft Rockets* **53**(6), 1035–1042 (2016).

30. D. Sherwin et al., "Wave-based attitude control of EIRSAT-1, a 2U CubeSat," in *Proc. 2nd Symp. Space Educ. Activities, SSEA-2018-93*, Budapest, Hungary (2018).
31. D. Murphy et al., "A compact instrument for gamma-ray burst detection on a CubeSat platform I," *Exp. Astron.* **52**(1–2), 59–84 (2021).
32. D. Murphy et al., "A compact instrument for gamma-ray burst detection on a CubeSat platform II," *Exp. Astron.* **53**, 961–990 (2022).
33. J. Thompson et al., "Double-dipole antenna deployment system for EIRSAT-1, a 2U CubeSat," in *Proc. 2nd Symp. Space Educ. Activities (SSEA), SSEA-2018-78*, Budapest, Hungary (2018).
34. S. Walsh et al., "Assembly, integration, and verification activities for a 2U CubeSat, EIRSAT-1," in *Proc. 3rd Symp. Space Educ. Activities (SSEA)*, Leicester, United Kingdom (2019).
35. J. Mangan et al., "Embedded firmware development for a novel CubeSat gamma-ray detector," in *8th IEEE Int. Conf. Space Mission Challenge* (2021).
36. J. Mangan et al., "Experiences in firmware development for a CubeSat instrument payload," in *Proc. 4th Symp. Space Educ. Activities (SSEA)*, Barcelona, Spain (2022).
37. D. Miranda et al., "A comparative survey on flight software frameworks for 'new space' nanosatellite missions," *J. Aerosp. Technol. Manage.* **11** (2019).
38. NASA, "Core flight system," https://cfs.gsfc.nasa.gov/ (accessed 17 February 2023).
39. A. Cudmore et al., "Big software for SmallSats: adapting CFS to CubeSat missions," in *Annu. AIAA/USU Conf. Small Satellites* (2015).
40. Kubos, "Kubos," https://docs.kubos.com/ (accessed 17 February 2023).
41. Bright Ascension, "Flight software development kit," https://brightascension.com/products/flight-software-development-kit/ (accessed 17 February 2023).
42. O. Quiros-Jimenez and D. D'Hemecourt, "Development of a flight software framework for student CubeSat missions," *Revista Tecnología en Marcha* **6**, 180–197 (2019).
43. https://upsat.gr/.
44. I. Latachi et al., "Reusable and reliable flight-control software for a fail-safe and cost-efficient CubeSat mission: design and implementation," *Aerospace* **7**(10), 146 (2020).
45. S. F. Hishmeh, T. J. Doering, and J. E. Lumpp, "Design of flight software for the KySat CubeSat bus," in *IEEE Aerosp. Conf.*, pp. 1–15 (2009).
46. H. Askari et al., "Software development for galassia CubeSat-design, implementation and in-orbit validation," in *Joint Conf. 31st Int. Symp. Space Technol. And Sci. (ISTS)* (2017).
47. K. V. C. K. de Souza, Y. Bouslimani, and M. Ghribi, "Flight software development for a CubeSat application," *IEEE J. Miniaturization Air Space Syst.* **3**(4), 184–196 (2022).
48. European Cooperation for Space Standardization, "ECSS-S-ST-00-01C – Glossary of terms," Standard (2012).
49. European Space Agency, "Tailored ECSS engineering standards for in-orbit demonstration CubeSat projects," TEC-SY/128/2013/SPD/RW (2016).
50. B. Tiseo et al., "Tailoring of ECSS standard for space qualification test of CubeSat nano-satellite," *Int. J. Aerosp. Mech. Eng.* **13**(4), 295–302 (2019).
51. M. Doyle et al., "Flight software development for the EIRSAT-1 mission," in *Proc. 3rd Symp. Space Educ. Activities (SSEA)*, Leicester, United Kingdom (2019).
52. R. Mahmood, K. Khurshid, and Q. U. Islam, "Institute of space technology CubeSat: ICUBE-1 subsystem analysis and design," in *Aerosp. Conf.*, pp. 1–11 (2011).
53. ATHENA Study Team, "ATHENA – concept of operations," ATHENA-ESA-DD-0001 (2015).
54. M. W. Smith et al., "On-orbit results and lessons learned from the ASTERIA space telescope mission," in *32nd Annu. AIAA/USU Conf. Small Satellites* (2018).
55. Bright Ascension, "User manual – generationone flight software development kit," BAL-GEN1-OBSW-UM (2018).
56. https://gcn.gsfc.nasa.gov/.
57. R. Dunwoody et al., "Validation of the operations manual for EIRSAT-1, a 2U CubeSat with a novel gamma-ray burst detector," *Proc. SPIE* **12186**, 121861B (2022).
58. Eclipse, "Eclipse," https://www.eclipse.org/ (accessed 17 February 2023).
59. Scott Chacon, "Git," https://git-scm.com/ (accessed 17 February 2023).

60. GitHub, “GitHub,” https://github.com/ (accessed 17 February 2023).
61. CircleCI, “CircleCI,” https://circleci.com/ (accessed 17 February 2023).
62. J. Reilly et al., “EIRFLAT-1: a FlatSat platform for the development and testing of the 2U CubeSat EIRSAT-1,” in *Proc. 4th Symp. Space Educ. Activities (SSEA)*, Barcelona, Spain (2022).
63. ThrowTheSwitch.org, “CMock,” http://www.throwtheswitch.org/cmock (accessed 17 February 2023).
64. ThrowTheSwitch.org, “Unity,” http://www.throwtheswitch.org/unity (accessed 17 February 2023).
65. S. Walsh et al., “Development of the EIRSAT-1 CubeSat through functional verification of the engineering qualification model,” *Aerospace* **8**(9), 254 (2021).
66. M. Doyle et al., “Mission testing for improved reliability of CubeSats,” *Proc. SPIE* **11852**, 118526M (2021).
67. M. Doyle et al., “Mission test campaign for the EIRSAT-1 engineering qualification model,” *Aerospace* **9**(2), 100 (2022).
68. R. Dunwoody et al., “Thermal vacuum test campaign of the EIRSAT-1 engineering qualification model,” *Aerospace* **9**(2), 99 (2022).
69. M. Doyle et al., “Update on the status of The Educational Irish Research Satellite (EIRSAT-1),” in *Proc. 4th Symp. Space Educ. Activities (SSEA)*, Barcelona, Spain (2022).

Biographies of the authors are not available.