ARISTOTLE UNIVERSITY OF THESSALONIKI

Diploma Thesis

Design of Fault Detection, Isolation and Recovery

in the

AcubeSAT NanoSatellite

Author: Konstantinos Kanavouras (8824)

Supervisor: Prof. Alkiviadis Hatzopoulos

A thesis submitted in fulfillment of the requirements
for the diploma in Electrical & Computer Engineering

in the

Faculty of Engineering
Electrical & Computer Engineering Department

June 22, 2021

PUBLISHED BY THE ARISTOTLE UNIVERSITY OF THESSALONIKI

HTTPS://GITHUB.COM/KONGR45GPEN/THESIS-LATEX

The AcubeSAT project is carried out with the support of the Education Office of the European Space Agency, under the educational Fly Your Satellite! programme.

Rendered on June 22, 2021 (38d8e32: Add I2C failure detection method)

Contents

List of Figures

List of Tables

Acronyms

ADCS Attitude Determination and Control Subsystem 12–15
CAN Controller Area Network 15
CCSDS The Consultative Committee for Space Data Systems 13
CDR Critical Design Review 12
COBS Consistent Overhead Byte Stuffing 26
COMMS Communications 13, 14
COTS Commercial Off-The-Shelf 12, 14, 17
ECSS European Cooperation for Space Standardization 21, 25, 26, 30, 38, 39
EMC Electromagnetic Compatibility 14
EPS Electrical Power Subsystem 14
ETL Embedded Template Library 27
FDIR Fault Detection, Isolation and Recovery 11–13, 18, 21, 23–27, 29–31, 33, 35–39
FMEA Failure Mode and Effects Analysis 17, 31–33
GS Ground Station 13, 14, 22
HAL Hardware Abstraction Library 28
HSIA Hardware/Software Interaction Analysis 32, 33, 36
I²C Inter-Integrated Circuit 26, 28, 29, 32
IC Integrated Circuit 26
IDE Integrated Development Environment 28
ISM Industrial, Scientific, Medical 14
LEO Low Earth Orbit 18
MAIV Maintenance, Assembly, Integration and Verification 17
MCU microcontroller (MicroController Unit) 15, 26–28
MPPT Maximum Power Point Tracking 14
MRAM Magnetoresistive Random-Access Memory 15
OBC On-Board Computer 14, 15
OBDH On-Board Data Handling 15
OBSW On-Board Software 15
OPS Operations 15

Abstract

Space is not a welcoming environment; while the aerospace engineering community has managed to reliably operate thousands of satellites in orbit, CubeSats, the most popular class of nanosatellite, only have a 50% success rate. Low costs, lack of strict technical requirements and scarcity of publicly available documentation often drives up the risks for educational, scientific and commercial CubeSats. This thesis investigates a configurable and modular Fault Detection, Isolation and Recovery (FDIR) architecture that uses the ECSS Packet Utilisation Standard. This FDIR concept, along with the provided open-source software implementation, can be used by CubeSat missions to increase the reliability of their design and chances of mission success, by autonomously responding to on-board errors. The thesis also includes background information regarding CubeSat reliability, and explores the software and hardware used to implement the proposed FDIR design on the AcubeSAT mission, currently under design by students of the Aristotle University of Thessaloniki.

KAPOU EDW tha mpei kai to abstract sta ellhnika

1

Introduction

1.1 Thesis research area

1-2 pages

1.2 Thesis goals & contribution

1.3 Thesis structure

2

Reliability Engineering in CubeSat Systems

2.1 Introduction to space engineering

cubesat reliablity [1]

2.2 Nanosatellites & CubeSats

2.3 CubeSat reliability

Fault Detection, Isolation and Recovery (FDIR)

3

The AcubeSAT mission

3.1 The mission

Here we write the history of the project and a description of the mission and the satellite...

3.2 Subsystems

The AcubeSAT nanosatellite is technically and programmatically split into 11 different subteams or **subsystems**, each responsible for a different section of the satellite, and made up out of 2 to 9 dedicated members.

In the following sections, a brief introduction on the function and design of each subsystem is presented. As the Systems Engineering and Product Assurance process is inherently connected to the function of all subsystems, the details relevant to FDIR are also mentioned. For more detailed information, the reader is encouraged to refer to AcubeSAT's website[1], or to the publicly available Critical Design Review (CDR) documents[2].

[1] https://acubesat.spacedot.gr/subsystems/

[2] https://gitlab.com/acubesat/documentation/cdr-public

3.2.1 Attitude Determination and Control Subsystem (ADCS)

The ADCS subteam is responsible for controlling the **attitude** and orientation of the spacecraft in orbit. This is achieved using a series of Commercial Off-The-Shelf (COTS) control actuators (one 3-axis magnetorquer board, one 1-axis reaction wheel), sensors for attitude determination (1 gyroscope and 2 magnetometers), and filtering, determination & control algorithms.[2]

The ADCS can operate under different pointing profiles for different attitude needs:

1. **Detumbling**, where the satellite is attempting to minimize its angular acceleration close to 0, to ensure a stable communications link, prevent detachment of parts and allow easier regaining of pointing.

   Detumbling mode is implemented in the simplest possible way, using only one of the 2 redundant magnetometers and a simple control algorithm. It is activated when there is no need to apply any of the specific pointing modes, or if the spacecraft angular rate is too high. In AcubeSAT, system-wide *Safe Mode*, *Commissioning Mode* and *Science Mode* apply detumbling.

2. **Nadir pointing**, where the satellite points the $+X$ side towards the Earth. This profile is used during *Nominal Mode* on passes over the Ground Station, where the directional patch antenna needs direct visibility.

3. **Sun pointing**, where the satellite points two sides to the sun with a 45° angle of incident each, in order to maximise solar panel input. This profile is used during *Nominal Mode*, between GS passes, in order to ensure a positive power budget.

The performance of the ADCS system can be summarised using performance metrics such as the ones presented in Table 3.1.

Table 3.1: Maximum ADCS error values after stabilisation

Error	Value
Absolute Performance Error	$< 30°$
Absolute Knowledge Error	$< 1°$

3.2.2 *Communications (COMMS)*

The communications subsystem is responsible for transmitting data between the Earth and the spacecraft in orbit. The transmitted data is split into 3 different categories[3]:

- **Telecommands (TC)**: Commands from the Earth to the satellite. They can be used to request information, or to perform specific spacecraft actions.

- **Telemetry (TM)**: Information sent from the satellite towards Earth, typically including vital information such as sensor values, system status, timestamps and events.

- **Science data**: The scientific data generated by the payload. These are the highest-volume data and represent the main scientific output of the mission.

It is important to mention that the satellite orbit only allows for a very short visibility duration every day, increasing the needs for on-board autonomy and the importance of a correctly implemented FDIR method.

The main component of the COMMS subsystem is the **SatNOGS**

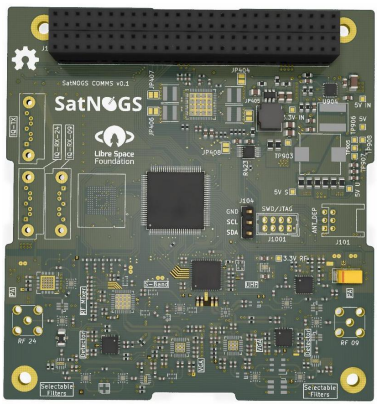


Figure 3.1: The SatNOGS COMMS board

COMMS board [4], an open-source RF transceiver developed by the LibreSpace Foundation, based on CCSDS telecommunications standards.

Communication will take place using 2 frequency bands on the ISM range, namely 436.5 MHz and 2.425 GHz, supported by a deployable turnstile and a directional patch antenna respectively. The use of ISM frequencies allows easy radio-amateur access to the satellite. The first (UHF) band also emits a periodic **beacon**, listing information about satellite status.

The communications subsystem is also responsible for the Electromagnetic Compatibility (EMC) analysis and interference mitigation, as well as the design and construction of the satellite Ground Station. The Ground Station will be part of **SatNOGS**[5], a global network of satellite ground stations based on open technologies and open data.

3.2.3 *Electrical Power Subsystem (EPS)*

The EPS is the subsystem responsible for the generation, distribution and storage of electrical power of the spacecraft. It is a critical aspect of the spacecraft due to the direct dependence of all subsystems to the high power needs of many CubeSat subsystems, and is theorised to be the most common reason for CubeSat failure.[6]

AcubeSAT has opted for a COTS subsystem approach for the EPS:[7]

- **Solar panels** are procured from EnduroSat. Four 3U panels cover the X and Y faces of the satellite, and one 1U panel covers the $-Z$ face.
- The **Power Conditioning & Distribution Unit (PCDU)** is procured from NanoAvionics and offers 10 switched channels with overcurrent protection over 4 voltage rails, as well as 4 Maximum Power Point Tracking (MPPT) converters.
- The **battery pack**, also procured from NanoAvionics, contains 4 18650 Li-Ion cells in a 2S2P[3] configuration.

[3] 2 series, 2 parallel

A dynamic approach is taken with regards to power budget calculation:

1. The in-orbit power generation is calculated for the duration of the mission using the **STK** software, taking into account satellite orientation, pointing profiles and eclipse, with a 1 min resolution.
2. The power consumption of the system is calculated on average for each different operational mode.
3. MPPT efficiencies and battery charge level are calculated for

Table 3.2: AcubeSAT nominal mode power budget

Consumer	**Power**
ADCS	1.10 W
COMMS	0.85 W
EPS	0.99 W
OBC	0.12 W
SU	0.25 W
Total	3.30 W
Orbit Average Power	4.24 W

each timepoint, assuming worst-case thermal and electrical conditions
4. A system-wide 10% margin is applied to the results

We have created a Python library consolidating the above steps[4] and producing the necessary outputs to prove the adequacy of the design.

[4] https://gitlab.com/acubesat/eps/power-budget

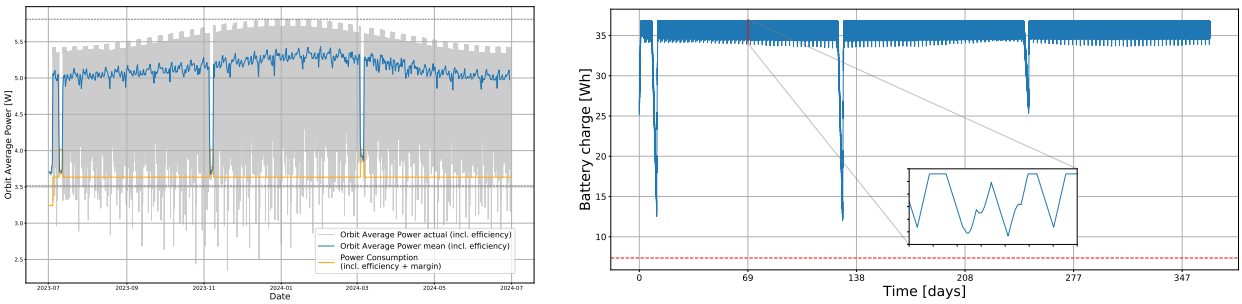


Figure 3.2: Dynamic power budget analysis. Left: Power consumption & generation for the orbit. Right: Battery discharge level throughout the mission

3.2.4 *On-Board Data Handling (OBDH)*

The OBDH subsystem is responsible for the design of the spacecraft's data interfaces, as well as the design of the **On-Board Computer (OBC)** board, tasked with controlling the basic spacecraft functions.[8]

The OBC board contains the main OBC logic, and is based around a Microchip SAMV71Q21RT[5] radiation-tolerant microcontroller and an MRAM memory, used to store critical data. The board also hosts the in-house implemented components of the ADCS subsystem, as a space-saving measure.

[5] https://www.microchip.com/wwwproducts/en/SAMV71Q21RT

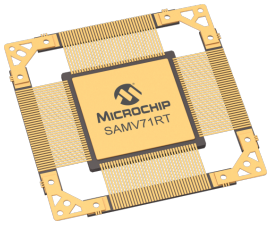


Figure 3.3: The `SAMV71Q21RT` microcontroller

AcubeSAT's data interface is using a cold-redundant Controller Area Network (CAN) bus to facilitate cross-subsystem communication, selected due to its robustness and reliability. [9] AcubeSAT boards implement the PC/104 mechanical interface.[10]

3.2.5 *On-Board Software (OBSW)*

The OBSW subsystem is responsible for the design and development of the nanosatellite's software. The language chosen to be used in the system's 4 microcontrollers (MCUs) is a reduced form of **C++**, and the code is guarded under a number of standards, static checkers and unit tests.[11] All software runs under the FreeRTOS operating system.

3.2.6 *Operations (OPS)*

The operations subsystem is responsible for the devising the operational modes & procedures of the spacecraft, and ensuring the functionality, commandability and observability of the satellite before and during its mission.

During flight, AcubeSAT can remain within one of the following **system modes**:[12]

- **Launch/Off mode**: During this mode, the satellite is turned completely off, and no subsystems are energised. This is used to represent the state of the spacecraft inside the deployer, where no electronics are allowed to be energised,[13, req. 3.3.3] and the CubeSat is in a completely dormant state.

- **Commissioning mode**: This mode is initiated as soon as the CubeSat exits the deployer, meaning that launch is complete. It contains the initial startup actions of the spacecraft, including detumbling and antenna deployment. No science takes place during commissioning mode.

- **Nominal mode**: This mode represents the state where the CubeSat will spend most of the time on. Apart from the necessary autonomy functions and battery charging, the CubeSat will also downlink telemetry and science data. No science takes place during nominal mode, except for health checks commanded from the ground. Nominal mode is also the only mode where the satellite performs nadir or sun pointing (Section 3.2.1).

- **Science mode**: This is where the main experiment takes place and payload data are generated. This mode includes operation of the fluidic system, control of the microfluidic chip, reinvigoration of the cells, and periodic acquisition of pictures using the miniaturised microscope.

  AcubeSAT has split science mode into **3 distinct occurrences**, termed sub-experiments α, β and γ, lasting 72 hours each, and taking place at different points of the mission to investigate the time-dependence of the observed results.

- **Safe mode**: It is common for spacecraft systems to include a *safe mode*,[14, p. 385] where the spacecraft switches off all non-essential systems and function, in order to respond to major malfunctions that cannot be corrected by autonomous procedures. Safe mode is intended as a well-defined and well-tested mode which is easy to maintain and reduces risk of any malfunction.

  On AcubeSAT, spacecraft functionality is significantly reduced, and the attitude profile includes pointing only. However, UHF communication and beacon transmission are still active for observability purposes.

Each mode is associated with a **functional flow** diagram, showing a high-level description of the spacecraft operation during this mode.[15]

Function	Launch	Commissioning	Nominal	Science	Safe
ADCS	Off	Detumbling	Pointing	Detumbling	Detumbling
COMMS	Off	UHF only	UHF and S-Band	UHF only	UHF only
EPS	Off	On	On	On	On
OBC	Off	On	On	On	On
SU	Off	Off	Maintenance & data only	On	Maintenance only

Table 3.3: Overview of AcubeSAT functionality on different modes

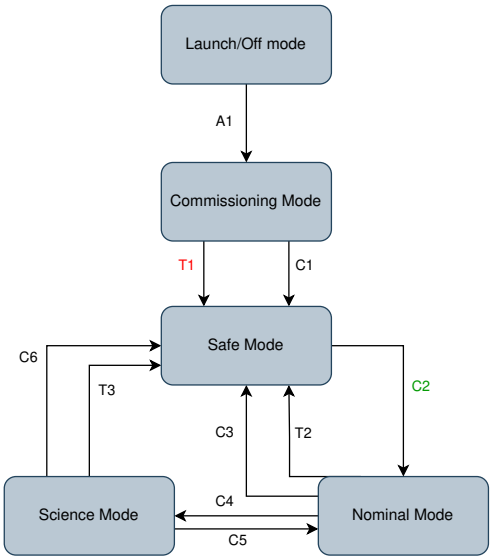


Figure 3.4: All transitions between system modes

3.2.7 *Structural*

The Structural subsystem has taken over:

- The analysis and configuration of the COTS 3-unit structure (Figure 3.5) housing all the CubeSat's components. Vibration analyses are especially important, as they serve to investigate whether the CubeSat can withstand the launcher's loads.

- The complete design, manufacturing and assembly of the **payload container** and its unibody, hosting the scientific experiment of the mission (Figure 3.7).

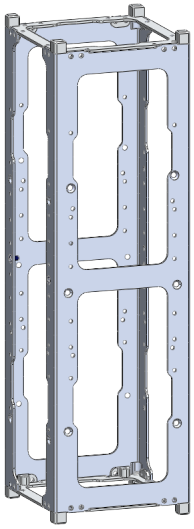


Figure 3.5: The CubeSat's 3U COTS structure

3.2.8 *Systems Engineering (SYE)*

The Systems Engineering subteam serves as the technical authority for the satellite. It is responsible for coordinating the developments and interfaces between subsystems, ensuring the conformance to standards and technical requirements, and identifying & resolving all issues arising from the complex multi-discipline design of the CubeSat.

Additionally, the SYE team is responsible for some specific technical developments that do not belong in any of the other subsystems, such as Reliability, Availability, Maintainability and Safety (RAMS), Failure Mode and Effects Analysis (FMEA), harnessing and the Main-

tenance, Assembly, Integration and Verification plan.

3.2.9 *Science Unit (SU)*

The Science Unit subteam is responsible for the conceptualisation and implementation of the mission's scientific payload, namely the high-throughput study of the effects of Low Earth Orbit (LEO) environments on yeast cells.

The payload is composed of the following functional parts:[16]

- The **payload container**, an almost 2U aluminum structure, pressurised at standard atmospheric pressure, and designed to host all the payload instrumentation. The container also accomodates a unibody which mechanically supports all SU components.
- A **microfluidic chip** based on Polydimethylsiloxane (PDMS), hosting 384 chambers capable of probing 190 distinct strains of *Saccharomyces Cerevisiae* for each subexperiment.
- A **fluidic system** composed from 2 pumps, 8 latching solenoid valves, 6 non-latching solenoid valves, and 3 fluid medium containers
- An **imaging system** operating as a microscope, containing a camera and a series of lights, filters and a lens
- A number of **heaters** to control component operational temperatures
- A number of redundant **sensors** for environmental measurements
- A Printed Circuit Board (PCB) containing the microcontroller and rest of the control components of the payload

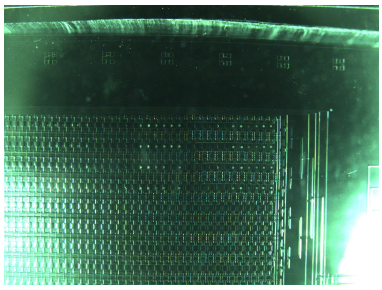


Figure 3.6: Example mission image output ([16])

There is a number of constraints that the payload imposes on FDIR design:

1. **Interruption** of one of the three 72-hour subexperiments during execution may mean complete loss of the subexperiment. The impact of such an event depends on the duration and timepoint of its occurence. In any case, adequate observability will allow the ground-control experiment to mimic the in-orbit conditions as much as possible.

2. **Freezing** of liquids inside the chip and fluidic tubes may lead to permanent damage on the setup. As such, heater operation may be required even during safe mode, after the first sub-experiment has been performed and liquid has flown into the system. This characteristics imposes further restrictions on the minimum availability of the system, depending on the thermal conditions.

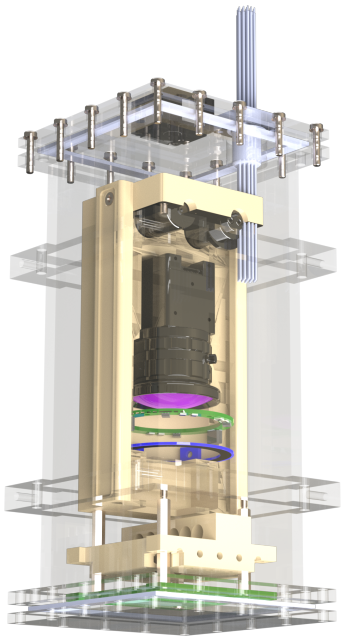


Figure 3.7: Transparent view of the payload container and its internals

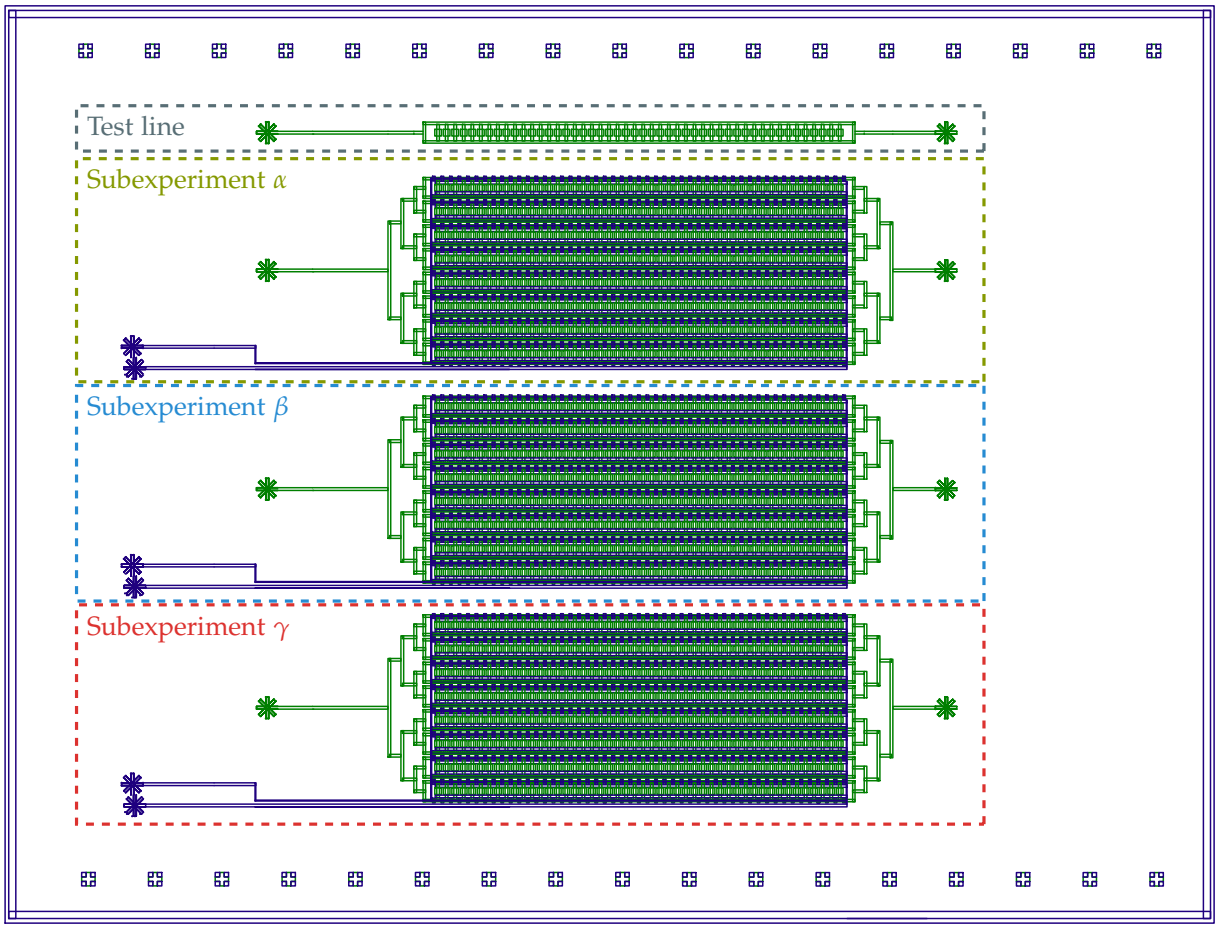


Figure 3.8: The microfluidic chip and its separation into 3 subexperiments and 1 test line. The fluid inlets are shown on the left side of the chip, while the outlets are on the right. *Green:* flow layer. *Blue:* control layer.

3.2.10 *Thermal*

The Thermal subteam is responsible for the thermal analysis of the satellite, where the solar and earth albedo conditions are combined with the components' heat dissipation to determine the worst-case temperatures experienced by the satellite in hot and cold conditions.

The results of thermal analysis typically lead to implementation of passive or active thermal control methods. Notably, in AcubeSAT, 3 electronically-controlled heaters are used for the batteries, PDMS chip and valves.

3.2.11 *Trajectory*

The Trajectory subteam is responsible for the analysis of the spacecraft's orbit, the calculation of radiation effects, the satellite's compliance to space debris regulations, and the estimation of its orbital lifetime.

AcubeSAT's requirements do not dictate use of thrusters, meaning that the satellite's orbit will be determined solely by the launcher, and cannot be altered in flight. As the selected launcher opportunity is unknown until some time before satellite delivery, a number of sensitivity analyses are executed to determine the allowed orbits.

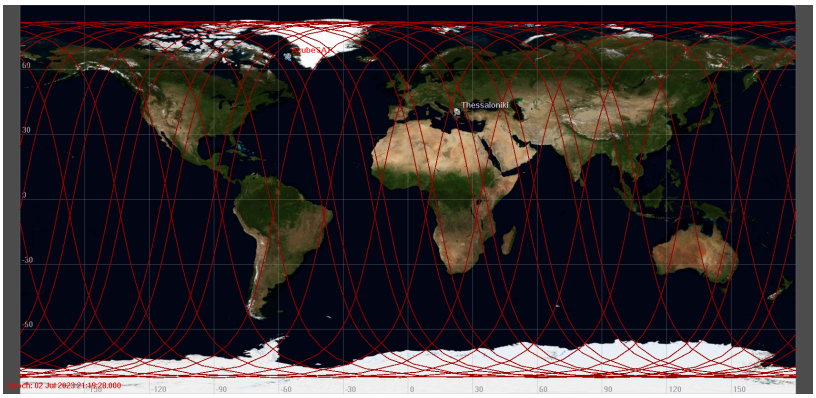


Figure 3.9: Ground track of an example AcubeSAT orbit, generated using NASA's General Mission Analysis Tool

3.3 Tools used

Here we write a small paragraph about OCDT and other MBSE tools and utilities used by AcubeSAT or developed in-house...

4

The SAVOIR FDIR concept

4.1 The ECSS Packet Utilisation Standard

4.1.1 The ECSS services

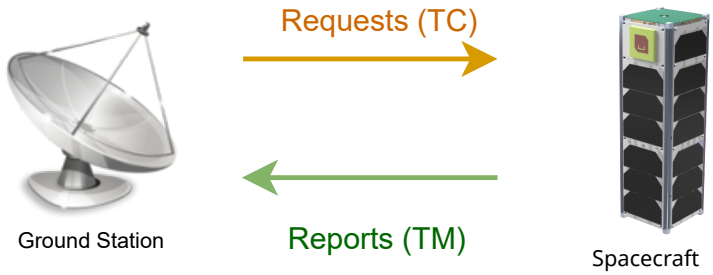


Figure 4.1: The PUS data transfer model

blablablab [17–19]

- **`ST[01]`: Request verification**
  Provides acknowledgement or failure reports for executed commands. This service essentially informs the operators about the status of TC sent to the spacecraft, and reports any occurred errors during parsing or execution.
- **`ST[02]`: Device access**
  Allows toggling, controlling and reconfiguring any on-board peripherals that do not support the PUS paradigm, but rely on simpler protocols to communicate.
- **`ST[03]`: Housekeeping**
  Produces periodic reports containing values of on-board parameters. This service essentially composes the periodic RF **beacon** of the satellite, by storing and transmitting parameter values without any prior TC request.
- **`ST[04]`: Parameter statistics reporting**
  Allows reporting statistics (min, max, mean, standard deviation) for specific parameters over specified intervals. This is a memory-efficient alternative to the `ST[03]` housekeeping service.
- **`ST[05]`: Event reporting**

Generates reports when notable occurrences take place on-board, such as:
  - Autonomous on-board actions
  - Detected failures or anomalies
  - Predefined steps during an operation
- **`ST[06]`: Memory management**
  Allows writing and reading directly from an on-board memory unit. This can be useful for debugging and investigative purposes, fetching mission data, or uploading new software to the spacecraft avionics. The service also provides for downlinking and uplinking files in a file system.
- **`ST[07]`: Task management** *(deprecated)*
  Allows stopping, suspending or resuming software tasks in case of contingency. This service has been removed from the standard and is only mentioned as a reference.
- **`ST[08]`: Function management**
  Provides the capability of running predefined actions that can receive further parameters. These actions can correspond to payload, platform, or any other functionality.
- **`ST[09]`: Time management**
  Allows periodic reporting of the current absolute spacecraft time for observability and correlation purposes.
- **`ST[10]`: Time packet** *(deprecated)*
  Used in the past for time packet generation. This service has been removed from the standard and is only mentioned as a reference.
- **`ST[11]`: Time-based scheduling**
  Allows the operators to **"time-tag"** telecommands for execution at future timestamps, instead of immediately.
- **`ST[12]`: On-board monitoring**
  This service allows checking parameter values to ensure that they remain within configurable limits. Whenever a violation occurs, an `ST[05]` can be optionally generated for further processing.
- **`ST[13]`: Large packet transfer**
  Provides a method of message segmentation, for message payloads that are too large to fit within the maximum allowed length for TC or TM.
- **`ST[14]`: Real-time forwarding control**
  This service is responsible of controlling which types of generated reports are immediately transmitted to the Ground Station.
- **`ST[15]`: On-board storage and retrieval**
  This service allows storing generated report on-board, as well as their commanded mass retrieval when the spacecraft has Ground Station visibility.
- **`ST[16]`: On-board traffic management** *(deprecated)*
  Allows monitoring the status and load of an on-board data bus and provides commands for resolution of errors. This service has been removed from the standard and is only mentioned as a reference.
- **`ST[17]`: Test**

This service allows performing on-board connection and "are-you-alive" checks.

- **`ST[18]`: On-board operations procedure**
  Allows loading, controlling (start, suspend, resume, abort) and configuring On-Board Control Procedures, which are sequences of commands written in an application-specific language.
- **`ST[19]`: Event-action**
  Provides the operators with the capability of autonomously executing TCs when an `ST[05]` event is triggered.
- **`ST[20]`: On-board parameter management**
  Provides the capability of reading and setting on-board parameters. Parameters are some of the most important entities defined in the PUS, and can represent:
  - Read-write configuration variables for the system or lower-level components
  - Read-only sensor and other telemetry values
  - FDIR results and diagnostics
- **`ST[21]`: Request sequencing**
  Allows operators to load series of TCs to be executed in a sequential order.
- **`ST[22]`: Position-based scheduling**
  Provides the capability of executing TCs when the spacecraft reaches a specific point in its orbit.
- **`ST[23]`: File management**
  Provides the capability of managing on-board file systems, with functions such as *copy*, *move*, *delete*, or *create directory*.

4.2 The SAVOIR standard

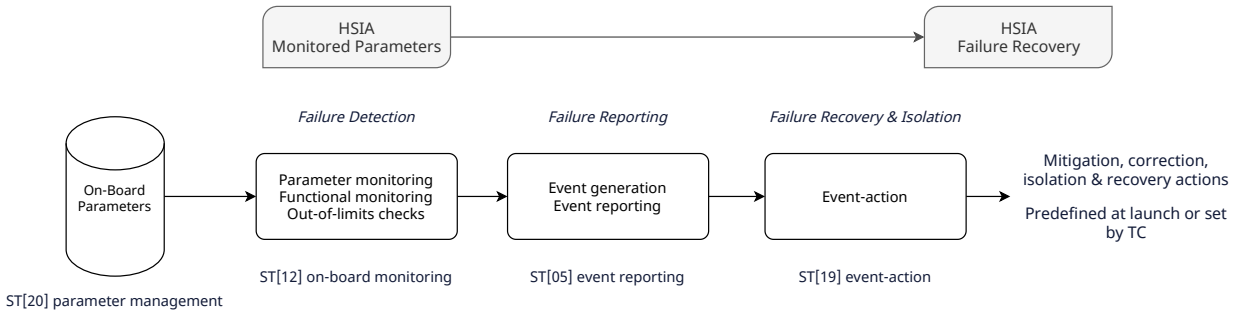


Figure 4.2: blablabla

5

FDIR in AcubeSAT

5.1 FDIR principles

- The 7 AcubeSAT FDIR principles
- SAVOIR FDIR requirements and compliance

5.2 Investigation on different architectures

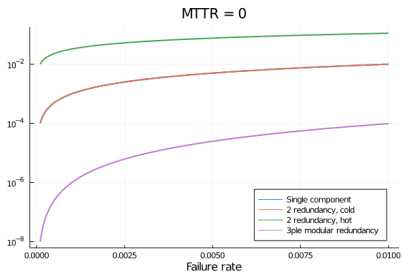


Figure 5.1: Error rates

5.3 Failure causes and recovery actions

5.3.1 Failure causes

5.3.2 Preventive actions

5.3.3 Corrective actions

5.4 FDIR operating modes

6

Practical demonstration of FDIR

As a proof of concept for AcubeSAT's FDIR implementation, a practical setup to simulate the satellite's behaviour was prepared (Figure 6.1). Key elements of the setup are:

- A Cortex-M7 **microcontroller**, used to simulate a **satellite subsystem**
- A number of redundant **sensors** used as the potential failure points
- The accompanying **software** that includes an implementation of the ECSS services and the Space AVionics Open Interface aRchitecture (SAVOIR) FDIR methodology
- A desktop computer, serving as the **ground station** to provide the necessary commanding and observing capabilities

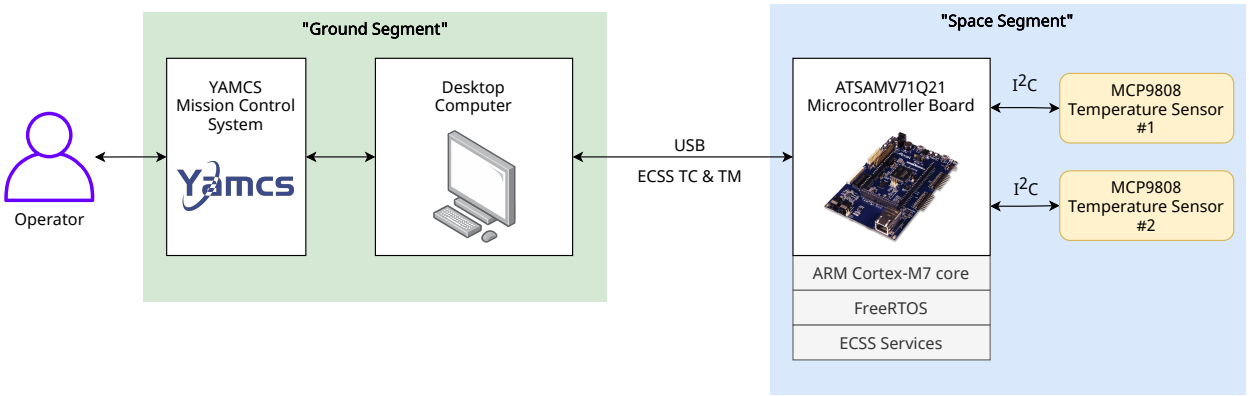


Figure 6.1: High-level block diagram of the demonstration system

6.1 System Description

6.1.1 Functionality

In order to emulate the most core functions of a spacecraft subsystem, we implemented a system with a single functional requirement:

`RQ-010`: The system shall measure and report the ambient tempera-

ture.

In order to justify implementing an FDIR approach for this system, we will introduce a reliability requirement:

RQ-020: No single failure on any measuring component shall lead to loss of system functionality

The detailed design of this simple demonstration system is presented in the following sections, and was architected to match the functionality, interfaces, design and software of the AcubeSAT nanosatellite as much as possible.

6.1.2 *Hardware*

The centre of the demonstration system is the MCU used to simulate the design and functionality of one of AcubeSAT's subsystems (Section 3.2.4). The selected MCU is the Atmel ATSAMV71Q21 hosted on the ATSAMV71-XULT[1] development board. This MCU is functionally identical to the one that will be used in orbit, featuring a 32-bit ARM Cortex-M7 core with 2 MiB of flash and 384 KiB of SRAM memory, and a maximum clock speed of 300 MHz.

[1] https://www.microchip.com/Developmenttools/ProductDetails/ATSAMV71-XULT

The MCU is accompanied by two **temperature sensors** which are used to simulate subsystem components which are prone to failure. The selected sensor is the Microchip MCP9808[2] which offers an accurate and frequent temperature readout over an Inter-Integrated Circuit (I^2C) bus. The maximum acquisition interval for the sensor is 250 ms.

[2] https://www.microchip.com/wwwproducts/en/MCP9808

The two sensors are wired in a **hot-redundant** configuration, due to their extremely low operating power. The two sensors are connected to different I^2C buses, so that the failure of one bus will not affect the operation of the other sensor.

Cold redundancy: Only one component is operating, while the other is not.

Warm redundancy: One component is operating fully, and the other is operating with reduced functionality.

Hot redundancy: Two or more simultaneously active parts operate in parallel. [20]

USB interface In order to receive Telemetry and transmit Telecommands to the demonstrative space segment, a USB link with a desktop computer is integrated into the design. This link uses the UART peripheral of the microcontroller, which solely transfers ECSS messages between the microcontroller and the desktop. These messages include the typical TM reports and TC requests, but also log messages intended for diagnostic purposes.

As the UART protocol does not offer packetisation facilities by default, Consistent Overhead Byte Stuffing (COBS) encoding [21] is implemented for all transmitted messages.

Connections The development board and sensors were laid out and connected directly. The sensor Integrated Circuits (ICs) were sol-

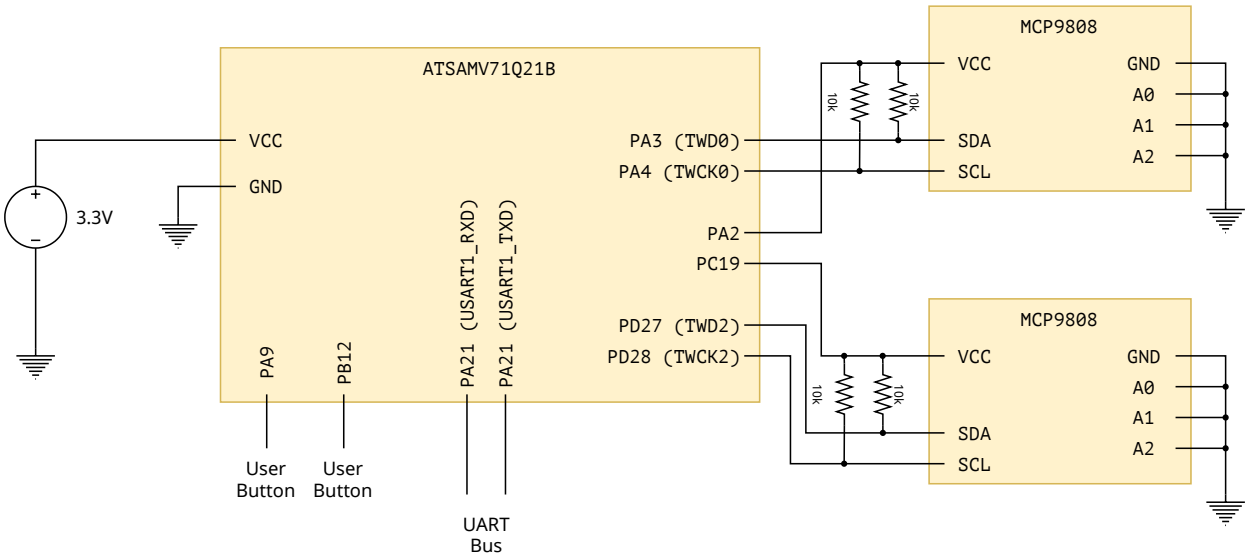


Figure 6.2: Simplified electrical schematic of the implementation. The two sensors are wired via I²C with the necessary pull-up resistors. Their power is controlled directly by microcontroller outputs, allowing the user to completely cut the power of the sensors if needed. The A pins of the sensors are arbitrarily set to select the I²C address of each device. The same address was used, as the two I²C buses are separate. Decoupling capacitors, crystals and other boilerplate components are not shown, as they are already included in the MCU development board.

dered on breakout PCBs which were manufactured based on Jaroslav Sýkora's design.https://www.jsykora.info/2019/06/pcb-panel-\protect\discretionary{\char\hyphenchar\font}{}{}of-smt-breakout-boards-with-kicad/

6.2 Software

6.2.1 *Flight Segment*

All FDIR operations and functionality are developed using the **C++** language[3] on the MCU. All developments rely on free and open source software which is freely available for download and modification. The Git[4] version control is used for all software projects.

The FDIR functionality is structured around the **ECSS-E-ST-70-41C PUS implementation created by the AcubeSAT team**[5], which offers a modular implementation of the standard utilising modern C++.

The MCU software and business logic are built on **FreeRTOS**[6], a low-footprint real-time operating system targeted towards embedded devices. FreeRTOS provides the capability of safe concurrency, along with support for well-controlled tasks and a number of synchronisation primitives.

To ensure high reliability and a low resource footprint of the software, the following **constraints** are enacted in C++ development:

1. Dynamic memory allocation[7] is banned completely from the code.
2. Item 1 means that standard C++ containers cannot be used. Instead, the Embedded Template Library (ETL)[8] is integrated

[3] https://isocpp.org/

[4] https://git-scm.com/

[5] https://gitlab.com/acubesat/obc/ecss-services

[6] https://www.freertos.org/

[7] Use of `malloc`, `new` etc.

[8] https://www.etlcpp.com/

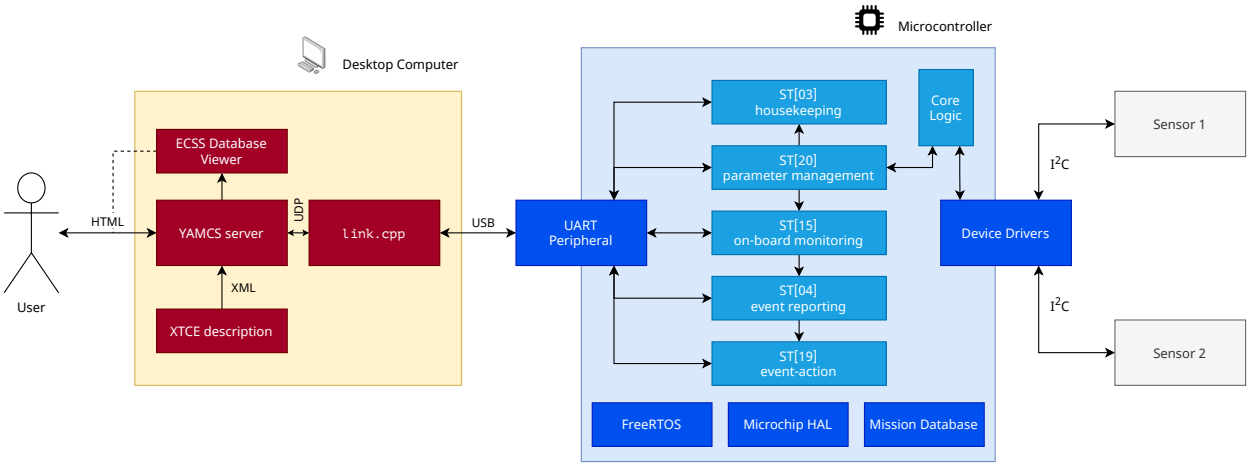


Figure 6.3: View of the software components and the data flow between them

in the software.

3. "Expensive" features such as Run-Time Type Inference, Dynamic Casts or exceptions are also prohibited.

The MPLAB Harmony[9] Hardware Abstraction Library (HAL) and configurator are used to interface with all the MCU's peripherals. CLion[10] has been selected as the IDE, along with the CMake[11] build system.

All software developed in the scope of this thesis is available online, and listed in Table 6.1. The most critical parts of the developed software are expanded in Appendix B. A summary of used libraries can also be found in Table 6.2.

[9] https://www.microchip.com/en-us/development-tools-tools-and-software/embedded-software-center/mplab-harmony-v3

[10] https://www.jetbrains.com/clion/

[11] https://cmake.org/

Library name	Description	URL
fdir-demo	Full microcontroller code for the demonstration	https://github.com/kongr45gpen/fdir-demo
fdir-demo-yamcs	Ground segment software: files and source code for YAMCS integration and MCU communications	https://github.com/kongr45gpen/fdir-demo-yamcs

Table 6.1: List of new software developed for this thesis

6.2.2 *I²C Failure Detection*

It is of interest to investigate how the hardware I²C peripheral provided by the microcontroller can be used to monitor the symptoms of bus failure.

1. **Microcontroller peripheral errors**

   The microcontroller's I²C peripheral is capable of generating an error whenever *the communicating peripheral does not set the "acknowledge" bit*. Failure to set this bit might be a result of disconnection of the clock or data lines, failure of the bus, or non-operation of the peripheral itself.

To ensure detection of peripheral errors, it is important to make sure that the error status of the peripheral is not discarded after every operation. Interrupts and the `[[nodiscard]]` C++ feature are used to ensure that all errors are caught.

Library name	Description	Modifications
FreeRTOS	Real-time operating system for embedded devices	
`ecss-services`	C++ implementation of the ECSS-E-ST-70-41C Packet Utilisation Standard	Missing services were implemented and some interface improvements were made to improve integration with the MCU `https://gitlab.com/kongr45gpen/ecss-services/-/tree/fdir`
ETL	C++ library (including containers, algorithms and other utilities) for applications with embedded constraints	
YAMCS	Software framework for mission & spacecraft control	
`cobs-c`	Implementation of COBS	

Table 6.2: List of used "off-the-shelf" software libraries

2. **Response timeout**

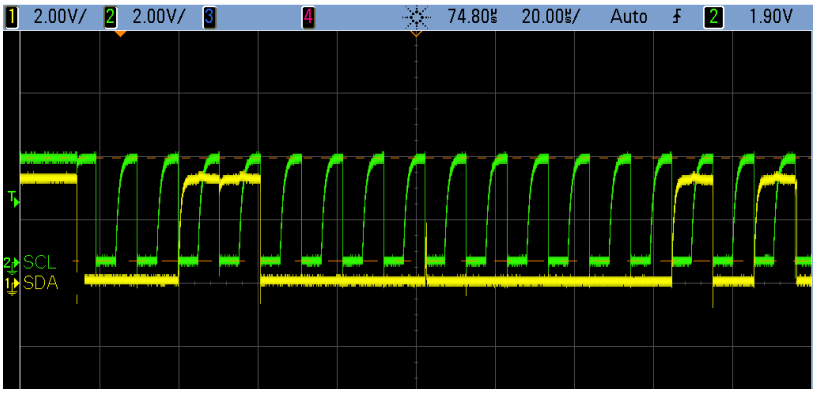
   For some adverse I²C bus conditions, such as incorrectly chosen pull-up resistor values or large amounts of capacitive load, I²C signal integrity may be lost (Section 6.2.2). This error is not detected by the microcontroller's peripheral directly, but can be diagnosed by adding a timeout to the sensor's data readouts.

3. **Chip ID**

   It is possible that the sensor suffers a severe fault but is still able to respond to I²C commands or set the *acknowledge* bit. For this reason, a more rigid check of the sensor's status is performed before each temperature read: the *permanently-set* registers of the peripheral contain values which, under nominal conditions, are hard-coded and will not change, barring hardware failure. If any value other than the expected one is return, the peripheral or the bus are assumed to have a fault.

Table 6.3: Read-only registers for the MCP9808

Address	Register Name	Value
`0x06`	Manufacturer ID	`0x0054`
`0x07`	Device ID	`0x0400`



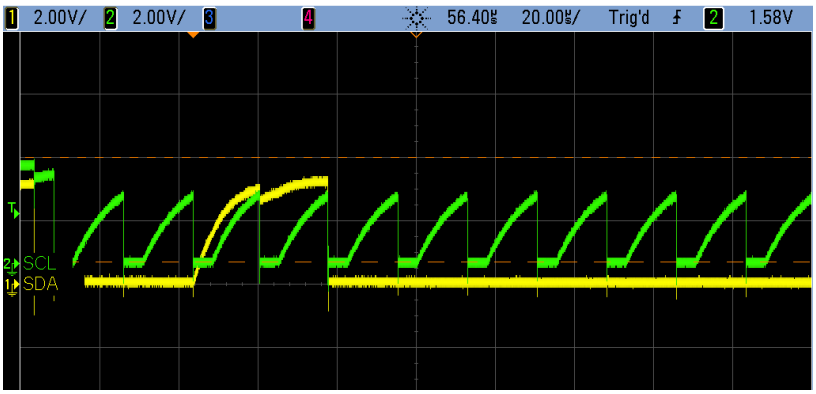


Figure 6.4: blablabla

6.2.3 *Mission Control*

The ground segment desktop computer needs to parse received packets and send commands, as well as display information about the sensor measurements, FDIR status and health of the connected flight system.

The YAMCS [22] framework was selected to cover the aforementioned needs, and has been tailored to provide the capabilities needed for this demonstration.

The ECSS protocol[17] is not supported by YAMCS by default. However, the necessary commands have been integrated into the system using the supported XML Telemetric and Command Exchange (XTCE) specification.[23] The source code of the integration can be found in Appendix B.

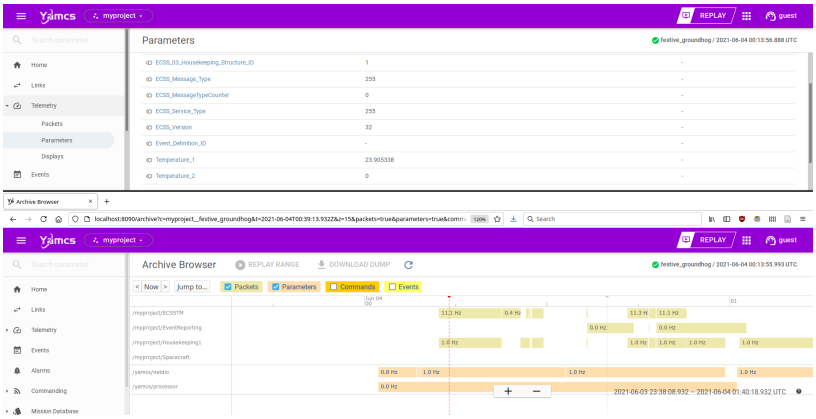


Figure 6.5: YAMCS parameter and archive views

6.2.4 *PUS database*

In

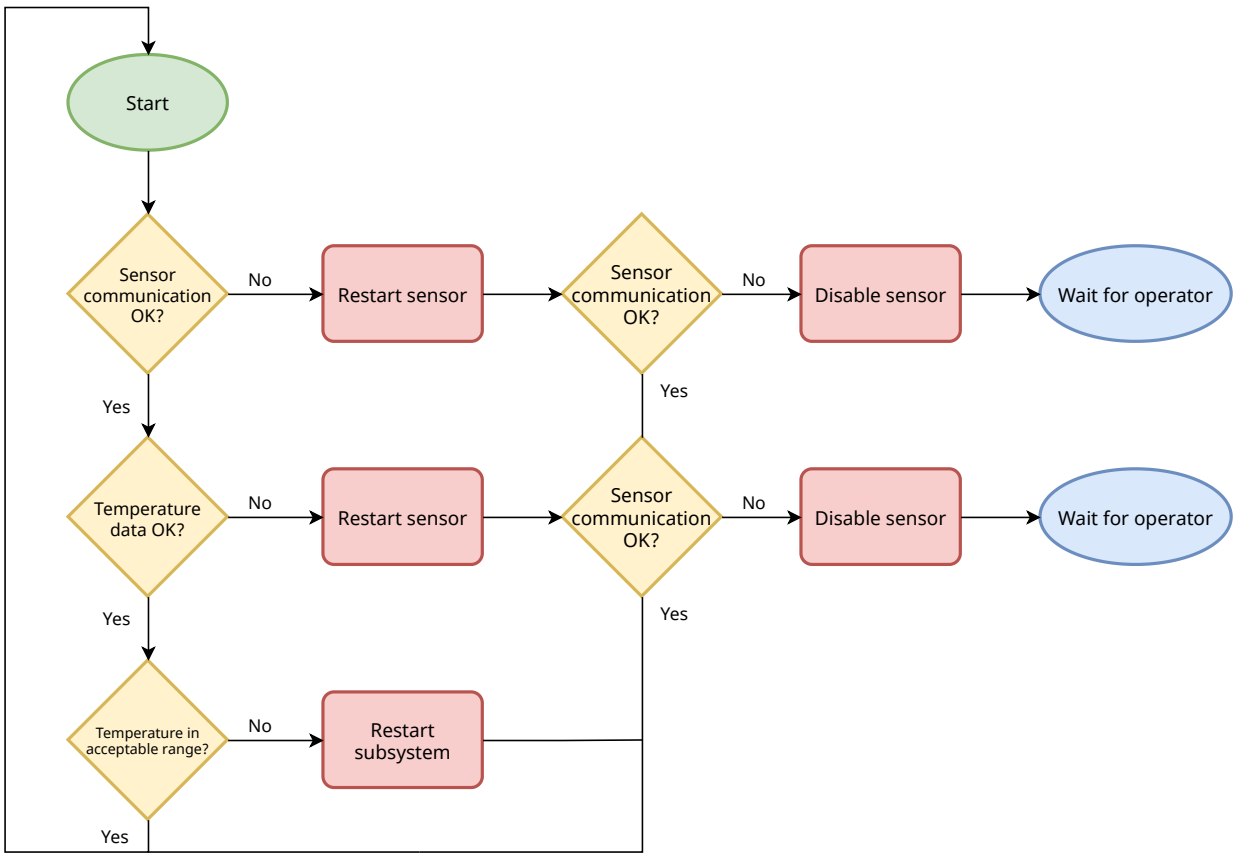
6.3 FDIR setup

The purpose of the test setup is to observe the response of the system to any failure of the two "vulnerable" temperature sensors. For this purpose, we will attempt to simulate every failure mode of each sensor, and design an FDIR implementation that anticipates all those failures.

As a required input for the FDIR detailed design, a FMEA analysis needs to be prepared for the system. All FMEA performed follows the requirements of the ECSS-Q-ST-30-02C standard.[24] The analysis is based on the FMEA performed for the AcubeSAT nanosatellite,[25] and can be seen in Table 6.4 for the reduced system. All expected failure modes for the two sensors are investigated, as well as a failure mode for the entire system that can be detected by correctly-operating sensors. Each of those failure modes will be later verified by injecting software or hardware modifications.

Double failures are not investigated in the scope of this document, as they are mostly considered only for launchers and manned missions [20].

6.3.1 *FDIR detailed design*



The FDIR of the sensors follows the hierarchical process described in Section 5.4, which is shown in Figure 6.6 as tailored for the sensors.

Figure 6.6: Overview of the temperature sensor FDIR process. For every process executed and failure detected, corresponding telemetry is generated as well.

The approach acts using the following steps, which are constantly being executed in the background:

1. The I²C bus health status is monitored. Any failure or lack of communication indicates a failure in the sensor's electronics.
2. The temperature output of the sensors is monitored. Any values which are outside the bounds of the physically possible thermal conditions are considered to indicate sensor or communication failures. Any values outside the operational limits of the subsystem's electronics are considered to indicate overheating and lead to a subsystem restart.
3. In order to recover from the failures, first a restart of the sensor is attempted. If the erroneous values still persist, then the sensor cannot recover from the failure, but is just isolated and disabled instead.

For this demonstration, we will not investigate failures of the microcontroller itself and its internals. Refer to [25] for AcubeSAT's FMEA and HSIA on the microcontrollers.

ID	Failure Mode	Failure Cause(s)	Mission Phase	Failure effects: Local	Failure effects: End effects	Failure Detection/Observable symptoms	Severity level	Compensating provisions
Temperature sensor MCP9808 #1								
F-010	Temporary loss of function	Intrinsic, Radiation	All	No temperature measurement from this sensor	None	No communication via I²C	4	Dual-redundant temperature sensor
F-020	Permanent loss of function	Intrinsic, Radiation	All	No temperature measurement from this sensor	None	No communication via I²C	4	Dual-redundant temperature sensor
F-030	Short Circuit between power pins	Intrinsic, Radiation	All	No temperature measurement from this sensor	None	No communication via I²C	4	Current-limiting resistor
F-040	Temporary Value Shift	Intrinsic, Radiation	All	Incorrect temperature readings	None	Temperature difference between 2 redundant sensors bigger than a safety value	4	Dual-redundant temperature sensor
F-050	Permanent Value Shift	Intrinsic, Radiation	All	Incorrect temperature readings	None	Temperature difference between 2 redundant sensors bigger than a safety value	4	Dual-redundant temperature sensor
F-060	I²C bus pin output stuck	Intrinsic, Radiation	All	Inability to communicate with both sensors	None	No communication via I²C for all temperature sensors	4	Sensors wired on separate I²C buses
Temperature sensor MCP9808 #2								
F-070	Temporary loss of function	Intrinsic, Radiation	All	No temperature measurement from this sensor	None	No communication via I²C	4	Dual-redundant temperature sensor
F-080	Permanent loss of function	Intrinsic, Radiation	All	No temperature measurement from this sensor	None	No communication via I²C	4	Dual-redundant temperature sensor
F-090	Short Circuit between power pins	Intrinsic, Radiation	All	No temperature measurement from this sensor	None	No communication via I²C	4	Current-limiting resistor
F-100	Temporary Value Shift	Intrinsic, Radiation	All	Incorrect temperature readings	None	Temperature difference between 2 redundant sensors bigger than a safety value	4	Dual-redundant temperature sensor
F-110	Permanent Value Shift	Intrinsic, Radiation	All	Incorrect temperature readings	None	Temperature difference between 2 redundant sensors bigger than a safety value	4	Dual-redundant temperature sensor
F-120	I²C bus pin output stuck	Intrinsic, Radiation	All	Inability to communicate with both sensors	None	No communication via I²C for all temperature sensors	4	Sensors wired on separate I²C buses
Subsystem								
F-130	Overheating	Short circuit, Environmental	All	Vulnerable component failure	Loss of subsystem functionality	Measured temperature outside expected range	3	Thermal analysis with uncertainty margins, overcurrent protection

Table 6.4: FMEA on demonstration system

Hardware/Software Interaction Analysis After defining the possible failure modes and logic of the system, we are ready to analyse each entry to provide the correct software inputs to implement the required FDIR. More specifically, for each failure, the following must be defined: [20, p. 84]

- Parameters to be monitored for detection
- Value ranges to be used to warn that a parameter is exceeding a specified range
- Isolation and reconfiguration actions to prevent failure propagation and, if possible, bring the system to a well operating state

The above information is listed in the Hardware/Software Interaction Analysis (HSIA) table, which links each failure (identified in the Failure Mode and Effects Analysis) to the corresponding low-level software parameters (Table 6.5).

For the purposes of this demonstration, we will consider that all FMEA items can lead to *feared events*, i.e. that FDIR will anticipate all identified possible failures.

The HSIA adds the following significant pieces of information to the FMEA:[24]

- The **monitored parameters** and the **conditions** that trigger the recovery action.

  These are listed here in terms of PUS parameters which can be easily identified and compared by software. This list is then linked to the corresponding `ST[12]` *on-board monitoring* service definitions, listed in Table 6.7.

  It is important to note that no single out-of-bounds value can trigger a recovery action. This is recommended in order to prevent transient errors or temporary communication protocol issues from triggering an unnecessary response which might disable a well-functioning component.

  The distinction between *temporary* and *permanent* failures is made by **counting the number of consecutive failures**. Before a permanent action to disable a sensor is enacted, at least 5 attempts to communicate must have failed.

  The use of 2 hot-redundant sensor also allows us to investigate the difference between the two temperatures to deduce the health of the produced data. If the absolute difference between the 2 values is too high, assuming the sensors are placed close enough together, an error condition is assumed. However, as triple modular redundancy was not implemented for this system, it is not

Temporary failure: Caused by events such as bit flips, noise or transients, and can be corrected without intervention, or with a simple restart/power-cycle.

Permament failure: A failure (typically in hardware, e.g. due to latchups or short-circuits) that cannot be corrected autonomously.

possible to deduce which sensor has failed.

- The **failure recovery and isolation actions**.

  These are listed in terms of PUS events (*ST[05] event reporting*) and event-actions (*ST[19] eventaction*), which are linked to Table 6.8.

  The selected recovery actions match the approach shown in Figure 6.6. Additionally, when it is not known which sensor has failed, a fail-safe approach of restarting or disabling both sensors is used.

ID	Failure Mode	Monitored Parameters	Monitoring ID	Event ID	Failure Recovery Action
Temperature sensor MCP9808 #1					
F-010	Temporary loss of function	`T1_Status` = `TIMEOUT` for 2 times	0	0	Power-cycle sensor 1
F-020	Permanent loss of function	`T1_Status` = `TIMEOUT` for 5 times	2	2	Ignore sensor 1 values
F-030	Short Circuit between power pins	`T1_Status` = `TIMEOUT` for 5 times	2	2	Ignore sensor 1 values
F-040	Temporary Value Shift	$\begin{cases} \lvert\Delta T\rvert & > 20\,^\circ\text{C or} \\ T_1 & > 100\,^\circ\text{C or} \\ T_1 & < -40\,^\circ\text{C} \end{cases}$ for 2 times	4, 6	0, 4	Power-cycle sensor 1
F-050	Permanent Value Shift	$\begin{cases} \lvert\Delta T\rvert & > 20\,^\circ\text{C or} \\ T_1 & > 100\,^\circ\text{C or} \\ T_1 & < -40\,^\circ\text{C} \end{cases}$ for 5 times	7, 9	2, 5	Ignore sensor 1 values
F-060	I²C bus pin output stuck	`T1_Status` = `TIMEOUT` for 5 times	2	2	Ignore sensor 1 values
Temperature sensor MCP9808 #2					
F-070	Temporary loss of function	`T2_Status` = `TIMEOUT` for 2 times	1	1	Power-cycle sensor 2
F-080	Permanent loss of function	`T2_Status` = `TIMEOUT` for 5 times	3	3	Ignore sensor 2 values
F-090	Short Circuit between power pins	`T2_Status` = `TIMEOUT` for 5 times	3	3	Ignore sensor 2 values
F-100	Temporary Value Shift	$\begin{cases} \lvert\Delta T\rvert & > 20\,^\circ\text{C or} \\ T_2 & > 100\,^\circ\text{C or} \\ T_2 & < -40\,^\circ\text{C} \end{cases}$ for 2 times	5, 6	1, 4	Power-cycle sensor 2
F-110	Permanent Value Shift	$\begin{cases} \lvert\Delta T\rvert & > 20\,^\circ\text{C or} \\ T_2 & > 100\,^\circ\text{C or} \\ T_2 & < -40\,^\circ\text{C} \end{cases}$ for 5 times	8, 9	3, 5	Ignore sensor 2 values
F-120	I²C bus pin output stuck	`T2_Status` = `TIMEOUT` for 5 times	3	3	Ignore sensor 2 values
Subsystem					
F-130	Overheating	$T_1 > 40\,^\circ\text{C}$ or $T_2 > 40\,^\circ\text{C}$	10, 11	6	Restart subsystem

Table 6.5: HSIA table

Parameters After investigating the Hardware/Software Interaction Analysis, we can list the scalar parameters for the *ST[20] parameter management* service[17] (Table 6.6).

Beyond the actual temperature values, we have included the **difference** (Δ) between the two temperatures, since the PUS standard does not provide ways to perform arithmetic calculations without hard-coded software. Additionally, the status of each sensor peripheral is included, which is an enumerated value.[12]

[12] One of `NOMINAL`, `TIMEOUT` or `DISABLED`

These parameters are telemetered to the ground segment and displayed to the user periodically, via the *ST[03] housekeeping* PUS service. However, during flight, the ground segment will not have constant access, since the satellite will not be visible from ground stations throughout its entire orbit.

Param. ID	Parameter	Units	Type	Type
0	Temperature 1	°C	`float`	R
1	Temperature 2	°C	`float`	R
2	Δ Temperature	°C	`float`	R
3	Temperature 1 Status		Enumerated	RW
4	Temperature 2 Status		Enumerated	RW
5	Temperature 1+2 Status		Enumerated	R

Table 6.6: List of `ST[20]` parameters
R: Read-only
RW: Read-write

On-board monitoring definitions After defining the parameters, the monitoring definitions that establish the acceptable and actionable ranges for each parameter should be outlined. These are then managed by the *ST[12] on-board monitoring* PUS service.

The *check validity conditions* are important to mention in this case, as they use the three *Status* parameters to prevent executing checks for disabled peripherals. As such, sensors that are not providing values due to hardware issues, FDIR actions or operator intervention, will not be able to trigger FDIR actions.

		Check validity condition				Check	
Definition ID	Monitored Parameter	Validity Parameter	Expected Value	Monitoring Interval	Repetition Number	Check Type	Criteria
0	Temp. 1 Status			500 ms	2	Expected Value	$\neq$ `TIMEOUT`
1	Temp. 2 Status			500 ms	2	Expected Value	$\neq$ `TIMEOUT`
2	Temp. 1 Status			500 ms	5	Expected Value	$\neq$ `TIMEOUT`
3	Temp. 2 Status			500 ms	5	Expected Value	$\neq$ `TIMEOUT`
4	Temperature 1	Temp. 1 Status	`NOMINAL`	500 ms	2	Range	$-40\,°C < t < 100\,°C$
5	Temperature 2	Temp. 2 Status	`NOMINAL`	500 ms	2	Range	$-40\,°C < t < 100\,°C$
6	Δ Temperature	Temp. 1+2 Status	`NOMINAL`	500 ms	2	Range	$-20\,°C < \Delta < 20\,°C$
7	Temperature 1	Temp. 1 Status	`NOMINAL`	500 ms	5	Range	$-40\,°C < t < 100\,°C$
8	Temperature 2	Temp. 2 Status	`NOMINAL`	500 ms	5	Range	$-40\,°C < t < 100\,°C$
9	Δ Temperature	Temp. 1+2 Status	`NOMINAL`	500 ms	5	Range	$-20\,°C < \Delta < 20\,°C$
10	Temperature 1	Temp. 1 Status	`NOMINAL`	500 ms	5	Range	$t < 50\,°C$
11	Temperature 2	Temp. 2 Status	`NOMINAL`	500 ms	5	Range	$t < 50\,°C$

Table 6.7: List of `ST[12]` monitoring definitions. t and Δ are used as placeholders for parameter values.

Event-action definitions The final part of the puzzle are the event-action definitions, which can be seen on Table 6.8 and combine:

- PUS events (`ST[05]` service) which are automatically generated when an on-board monitoring definition goes out of limits
- PUS event-action definitions (`ST[19]` service), which link every occurrence of an event to a stored TC that is immediately executed. They define the FDIR recovery actions.

All the aforementioned definitions can be easily modified by operators using single commands that affect the on-board stored PUS database.

Event ID	Monitoring Definitions	Action
0	0, 4	Restart sensor 1
1	1, 5	Restart sensor 2
2	2, 7	Set sensor 1 status = `DISABLED`
3	3, 8	Set sensor 2 status = `DISABLED`
4	6	Restart all sensors
5	9	Set all sensors status = `DISABLED`
6	10, 11	Restart system

Table 6.8: List of `ST[19]` event-action definitions

6.4 FDIR validation

6.4.1 *Nominal operation*

During nominal operation, no failures are simulated. The system is connected to power and outputs ECSS telemetry (Figure 6.7), which includes parameter values parsed by (Figure 6.8). Additionally, a log file is kept for diagnostic purposes (Figure 6.9). The system is compliant with its requirements (Section 6.1.1) and can transmit the temperature to the ground station as expected.

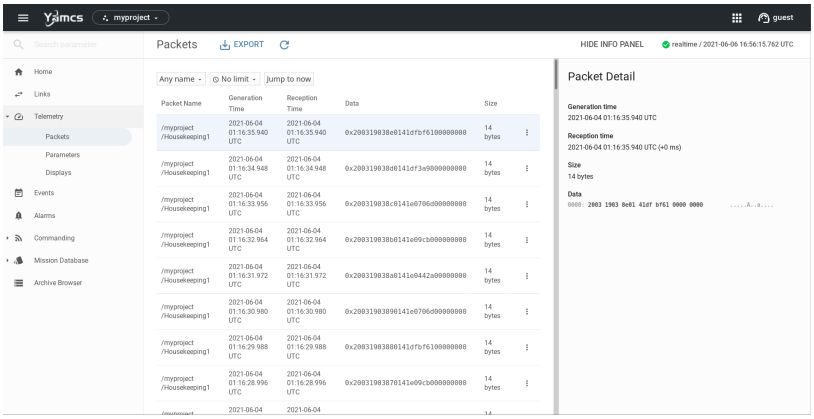


Figure 6.7: View of housekeeping packets generated by the MCU in YAMCS

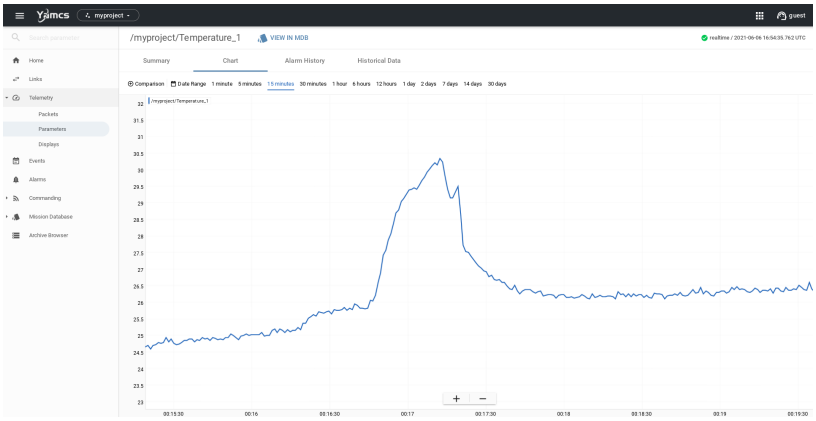


Figure 6.8: View of one temperature parameter in YAMCS

6.4.2 *Simulation of failures*

The following methods were used to simulate each failure:

- For component communication failures, each sensor was just physically disconnected from the system.
- For small changes in temperature, a hot air gun or other hot air source was pointed towards each sensor. Care was taken to not exceed the rated temperatures of the affected components.
- For changes in temperature which were impractical to simulate, the failure was injected in software, by manually modifying the

```
909601  [debug  ] T1 = 28.18
909701  [debug  ] T1 = 28.08
909801  [debug  ] T1 = 28.14
909901  [debug  ] T1 = 27.97
910000  [trace  ] New TM [3,25]
910001  [debug  ] T1 = 28.10
910101  [debug  ] T1 = 28.03
910201  [debug  ] T1 = 28.12
910301  [debug  ] T1 = 27.90
910401  [debug  ] T1 = 28.08
```

Figure 6.9: Log output during nominal operation. The first number is the MCU tick counter in milliseconds since boot.

variable.

The two buttons present on the development board were used as a simple interface to "inject" extreme temperature values.

6.4.3 *Real-time modification of FDIR*

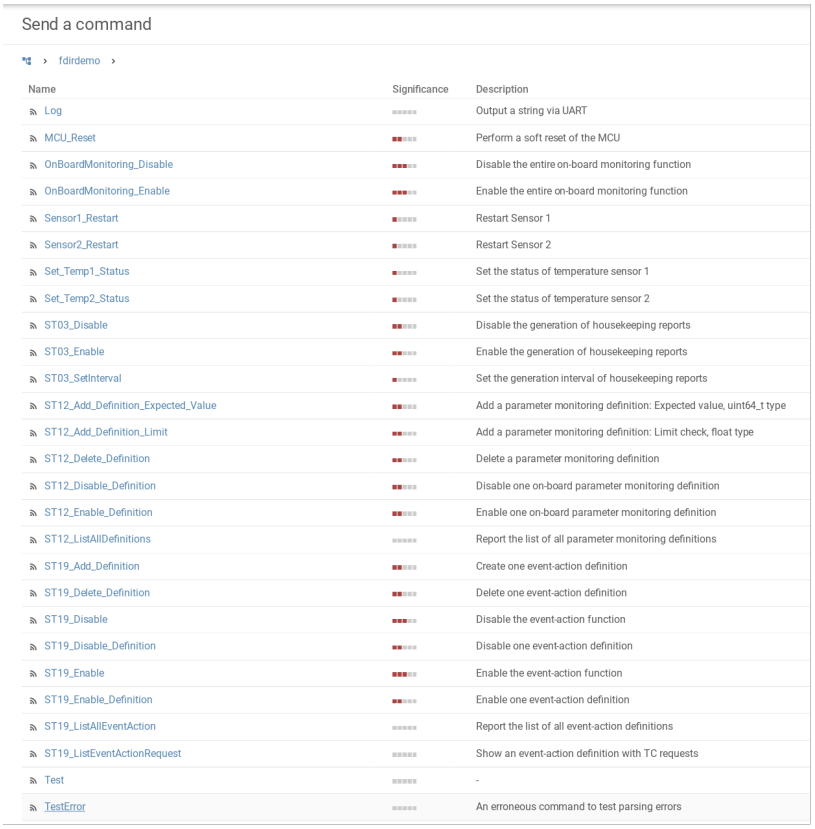
The power of SAVOIR's FDIR approach lies in the fact that the entire FDIR concept can be safely modified via TC without requiring any modifications or reuploads of the spacecraft's software. The application of the ECSS services paradigm allows customising all on-board FDIR functions and associated services using predefined TC structures. In this section, ECSS commands will be executed to demonstrate these capabilities and their control from the Ground Station.

The level of reconfigurability possible is defined by requirements listed in [20, 26]. The tailored version used for our experiments is listed in Table 6.9.

#	Requirement	Service	Details
RC-010	The reporting of FDIR activities shall contain all information for failure analysis (e.g. time of occurrence, parameter out-of-limit, switching performed).	ST[12] on-board monitoring	Via periodic *check transition reports*
RC-020	The capability shall be provided to enable and to disable any on-board FDIR function by telecommand.	ST[12] on-board monitoring	Via *enable/disable parameter monitoring definitions* commands
RC-030	The capability shall be provided to add and delete parameters from the on-board monitoring list.	ST[12] on-board monitoring	Via *add/delete parameter monitoring definitions* commands
RC-040	The capability shall be provided to modify the failure recovery actions associated with each on-board monitoring definition.	ST[19] event-action	Via *add*, *delete*, *enable* and *disable event-action definitions* commands

Table 6.9: FDIR reconfigurability requirements

The YAMCS platform was once again used to generate the wanted commands (Figure 6.10).



Send a command

› fdirdemo ›

Name	Significance	Description
Log	▪▫▫▫▫	Output a string via UART
MCU_Reset	▪▪▫▫▫	Perform a soft reset of the MCU
OnBoardMonitoring_Disable	▪▪▪▫▫	Disable the entire on-board monitoring function
OnBoardMonitoring_Enable	▪▪▪▫▫	Enable the entire on-board monitoring function
Sensor1_Restart	▪▫▫▫▫	Restart Sensor 1
Sensor2_Restart	▪▫▫▫▫	Restart Sensor 2
Set_Temp1_Status	▪▫▫▫▫	Set the status of temperature sensor 1
Set_Temp2_Status	▪▫▫▫▫	Set the status of temperature sensor 2
ST03_Disable	▪▪▫▫▫	Disable the generation of housekeeping reports
ST03_Enable	▪▪▫▫▫	Enable the generation of housekeeping reports
ST03_SetInterval	▪▫▫▫▫	Set the generation interval of housekeeping reports
ST12_Add_Definition_Expected_Value	▪▪▫▫▫	Add a parameter monitoring definition: Expected value, uint64_t type
ST12_Add_Definition_Limit	▪▪▫▫▫	Add a parameter monitoring definition: Limit check, float type
ST12_Delete_Definition	▪▪▫▫▫	Delete a parameter monitoring definition
ST12_Disable_Definition	▪▪▫▫▫	Disable one on-board parameter monitoring definition
ST12_Enable_Definition	▪▪▫▫▫	Enable one on-board parameter monitoring definition
ST12_ListAllDefinitions	▫▫▫▫▫	Report the list of all parameter monitoring definitions
ST19_Add_Definition	▪▪▫▫▫	Create one event-action definition
ST19_Delete_Definition	▪▪▫▫▫	Delete one event-action definition
ST19_Disable	▪▪▪▫▫	Disable the event-action function
ST19_Disable_Definition	▪▪▫▫▫	Disable one event-action definition
ST19_Enable	▪▪▪▫▫	Enable the event-action function
ST19_Enable_Definition	▪▪▫▫▫	Enable one event-action definition
ST19_ListAllEventAction	▫▫▫▫▫	Report the list of all event-action definitions
ST19_ListEventActionRequest	▫▫▫▫▫	Show an event-action definition with TC requests
Test	▫▫▫▫▫	-
TestError	▫▫▫▫▫	An erroneous command to test parsing errors

Figure 6.10: List of commands in the YAMCS interface. These commands can initiate spacecraft actions, enable/disable sensors, or reconfigure the on-board ECSS services databases. The list of commands is provided programmatically in XTCE format (Appendix B).

7

Conclusions and Future Work

7.1 Future work

A

Bibliography

[1] O. Durou, V. Godet, L. Mangane, D. Pérarnaud, and R. Roques, "Hierarchical fault detection, isolation and recovery applied to cof and atv avionics," *Acta Astronautica*, vol. 50, no. 9, pp. 547–556, May 1, 2002, ISSN: 0094-5765. DOI: 10.1016/S0094-5765(01)00212-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576501002120 (visited on 04/03/2021).

[2] G. Savvidis, R. Voulgarakis, T. Papafotiou, V. Moustakas, E. Mylonas, and V. Pappa, *AcubeSAT AOCS DDJF*, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/DDJF/DDJF_AOCS.pdf.

[3] K. Kapoglis and E. Chatziargyriou, *AcubeSAT TTC DDJF*, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/DDJF/DDJF_TTC.pdf.

[4] M. Surligas, "SatNOGS-COMMS," presented at the CubeSat Developers Workshop, Apr. 12, 2021, p. 22.

[5] D. White, C. Shields, P. Papadeas, A. Zisimatos, M. Surligas, M. Papamatthaiou, D. Papadeas, and E. Kosmas, "Overview of the Satellite Networked Open Ground Stations (SatNOGS) Project," *Small Satellite Conference*, Aug. 8, 2018. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2018/all2018/313.

[6] M. Langer and J. Bouwmeester, "Reliability of CubeSats – Statistical Data, Developers' Beliefs and the Way Forward," *Proceedings of the 30th Annual AIAA/USU Conference on Small Satellites*, 2016. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A4c6668ff-c994-467f-a6de-6518f209962e (visited on 05/20/2021).

[7] A.-F. Retselis, K. Kanavouras, and G. Pavlakis, *AcubeSAT System DDJF*, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/DDJF/DDJF_SYS.pdf.

[8] K. Kanavouras and G. Pavlakis, *AcubeSAT OBDH DDJF*, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/DDJF/DDJF_OBDH.pdf.

[9] J. Bouwmeester, M. Langer, and E. Gill, "Survey on the implementation and reliability of CubeSat electrical bus interfaces," *CEAS Space Journal*, vol. 9, no. 2, pp. 163–173, Jun. 1, 2017, ISSN: 1868-2510. DOI: 10.1007/s12567-016-0138-0. [Online]. Available: https://doi.org/10.1007/s12567-016-0138-0 (visited on 05/20/2021).

[10] PC/104 Embedded Consortium, "PC/104 Specification," Aug. 13, 2008. [Online]. Available: https://pc104.org/wp-content/uploads/2015/02/PC104_Spec_v2_6.pdf.

[11] K. Kanavouras, I. Kozaris, A. Theocharis, G. Pavlakis, and D. Stoupis, *AcubeSAT OBSW DDJF*, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/DDJF/DDJF_OBSW.pdf.

[12] A. Zaras, K. Kapoglis, M. Georgousi, T. Papafotiou, M. Chadolias, A. Anthopoulos, A.-F. Retselis, A. Arampatzis, K.-O. Xenos, and E. Christidou, *AcubeSAT Mission Description & Operations Plan*, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/MDO%20file/MDO.pdf.

[13] California Polytechnic State University, "CubeSat Design Specification Rev. 13," Feb. 20, 2014. [Online]. Available: https://www.cubesat.org/s/cds_rev13_final2.pdf.

[14] M. A. Aguirre, *Introduction to Space Systems: Design and Synthesis*, ser. Space Technology Library. New York: Springer-Verlag, 2013, ISBN: 978-1-4614-3757-4. DOI: 10.1007/978-1-4614-3758-1. [Online]. Available: https://www.springer.com/gp/book/9781461437574 (visited on 05/23/2021).

[15] (May 8, 2021). AcubeSAT Functional Architecture, [Online]. Available: https://gitlab.com/acubesat/systems-engineering/functional-architecture.

[16] A. Arampatzis, A. Zaras, P. Matsatsos, O. Ousoultzoglou, D. Nikolopoulou, and E. Sandaltzopoulou, *AcubeSAT Payload DDJF*, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/DDJF/DDJF_PL.pdf.

[17] ECSS Secretariat, "ECSS-E-ST-70-41C – Telemetry and telecommand packet utilization," European Space Agency, Apr. 15, 2016. [Online]. Available: https://ecss.nl/standard/ecss-e-st-70-41c-space-engineering-telemetry-and-telecommand-packet-utilization-15-april-2016/.

[18] ——, "ECSS-E-70-41A – Telemetry and telecommand packet utilization," European Space Agency, Jan. 30, 2003. [Online]. Available: https://ecss.nl/standard/ecss-e-70-41a-ground-systems-and-operations-telemetry-and-telecommand-packet-utilization/ (visited on 05/24/2021).

[19] J.-F. Kaufeler, "The ESA standard for telemetry and telecommand packet utilisation: PUS," Nov. 1, 1994. [Online]. Available: https://core.ac.uk/download/pdf/42783096.pdf (visited on 05/24/2021).

[20] Space Avionics Open interface Architecture, "SAVOIR FDIR Handbook," European Space Agency, SAVOIR-HB-003, Oct. 2019. [Online]. Available: https://essr.esa.int/project/savoir.

[21] S. Cheshire and M. Baker, "Consistent overhead byte stuffing," in *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '97, New York, NY, USA: Association for Computing Machinery, Oct. 1, 1997, pp. 209–220, ISBN: 978-0-89791-905-0. DOI: 10.1145/263105.263168. [Online]. Available: https://doi.org/10.1145/263105.263168 (visited on 06/02/2021).

[22] A. Sela, "Yamcs - A Lightweight Open-Source Mission Control System," in *SpaceOps 2012 Conference*, ser. SpaceOps Conferences, 0 vols., American Institute of Aeronautics and Astronautics, Jun. 11, 2012. DOI: 10.2514/6.2012-1280790. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2012-1280790 (visited on 06/05/2021).

[23] G. Simon, E. Shaya, K. Rice, S. Cooper, J. Dunham, and J. Champion, "XTCE: A standard XML-schema for describing mission operations databases," in *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*, vol. 5, Mar. 2004, 3313–3325 Vol.5. DOI: 10.1109/AERO.2004.1368138.

[24] ECSS Secretariat, "ECSS-Q-ST-30-02C – Failure modes, effects (and criticality) analysis (FMEA/FMECA)," European Space Agency, Mar. 6, 2009. [Online]. Available: https://ecss.nl/standard/ecss-q-st-30-02c-failure-modes-effects-and-criticality-analysis-fmeafmeca/.

[25] A.-F. Retselis and K. Kanavouras. (Dec. 1, 2020). AcubeSAT FMEA Worksheet, GitLab, [Online]. Available: https://gitlab.com/acubesat/systems-engineering/fmea (visited on 06/05/2021).

[26] ECSS Secretariat, "ECSS-E-ST-70-11C – Space segment operability (31 July 2008)," European Space Agency, Jul. 31, 2008. [Online]. Available: https://ecss.nl/standard/ecss-e-st-70-11c-space-segment-operability/ (visited on 06/21/2021).

B

Source code

`main.cpp`

`xtce.xml`

fixme