



Dipl.-Ing. Maximilian Henkel, BSc

# Using a Reconfigurable FPGA for Emerging Space Applications

Dissertation

presented in partial fulfillment of the  
requirements for the academic degree of

Doctor of Engineering Sciences

submitted to

Graz University of Technology

Supervisor

Em.Univ.-Prof. Dipl.-Ing. Dr.techn. Otto Koudelka

Assessor

Assoc.Prof. Dipl.-Ing. Dr.techn. Wilfried Gappmair

Institute of Communication Networks and Satellite Communications

Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Klaus Witrissal

Co-Supervisor

David Evans, MSc

European Space Operations Center

European Space Agency

Advisor

Prof. Dr.-Ing. René Laufer

Luleå University of Technology

Graz, April 2024

## Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present thesis.

---

Date

---

Signature

This document was written with Vim, compiled with Lua $\text{\LaTeX}$  and Biber, and typeset in Latin Modern font. It uses the KOMA script book class, and the title as well as this page from the  $\text{\LaTeX}$  template by Karl Voit, which is available at <https://github.com/novoid/LaTeX-KOMA-template>

# Preface

Fortunately, I seized the opportunity and joined the OPS-SAT team in late 2015 at the Institute of Communication Networks and Satellite Communications at my alma mater. My choice was based on the expectation to work in a team of experienced colleagues and to get confronted with technically diverse and challenging issues. My expectations were more than fulfilled and I enjoyed and still enjoy solving them in a creative and use-oriented way.

When I joined, the OPS-SAT project was already well underway. Although everyone in the team focused on specific topics, everyone was able to take on any task. Therefore, I limit my specific tasks to the ones mentioned in the present thesis, like the software and boot design of the processing platform, the communication chain design and the radio tests, and the in-orbit commissioning and operations. With the project progressing Prof. Koudelka, David Evans and myself realised that the software and firmware developments for the processing platform, in particular its integrated Field-Programmable Gate Array (FPGA), demanded more intense and deeper engagement. Fortunately, the work on this thesis could be supported by European Space Agency's (ESA) Network Partnering Initiative (NPI) programme, which allowed me to elaborate further on many unknown aspects of the envisioned use of OPS-SAT as an in-orbit experimental platform.

Throughout the project, I saw many concepts, software and hardware designed to minimise the risk of failure. This is a very valuable goal. However, many of these approaches are neither state-of-the-art nor use-oriented. This work can therefore also be seen as a counter-thesis to such (sometimes too) established practices: rather than merely avoiding failure, it may show, based on real experience of operating a spacecraft, that accepting and working with failure can lead to further progress.

## *Preface*

The real-world experience of using reconfigurable computing hardware on a spacecraft could be valuable for space professionals of any field. Therefore explanations on the evolution of the presented approaches, their implications, and the general considerations are all described in this thesis. Furthermore, the technical aspects of the presented approaches may encourage developers to use (more) reconfigurable FPGA fabrics in any phase of a project. Specifically, developments for remote systems could benefit from the gained experience. For this reason, specific details have been avoided and the focus is instead on the underlying concepts and the potential applications.

This thesis gives another view on the OPS-SAT spacecraft and its safety concepts. In particular, it explains how the reconfiguration can be integrated into the live spacecraft operations and what such a reconfigurable computing platform can offer. The covered topics range from sophisticated recovery to advanced debugging techniques. Some of these topics were already published. Specifically, Chapter 6 contains content directly from [38] and Chapter 8 from [37].

A significant amount of the work presented here was carried out during the numerous pandemic-related lockdowns in Austria. For example, the recovery presented in Chapter 4, which allowed OPS-SAT to continue its experimental mission, and the in-orbit SpaceWire implementation in Chapter 7 were implemented by me while being locked up with only a development board at hand. During this period, the daily (remote) contact with my colleagues was an enormous support. From another perspective, this also illustrates the rapid and agile development cycle from my desk at home to the execution of synthesised hardware and software in orbit.

# Acknowledgements

First of all, I am grateful to Prof. Otto Koudelka for giving me the opportunity to be a part of the OPS-SAT project<sup>1</sup>, and who initiated, together with Mr. David Evans, the financial support for the work on this thesis<sup>2</sup>. I would also like to thank Prof. Wilfried Gappmair, Prof. Klaus Witrissal and Prof. René Laufer for their support and valuable comments.

In addition, I am grateful to the ESA executives involved for initiating this exceptional project, and made the funding possible. I am also grateful to the Austrian Aeronautics and Space Agency (ALR), in particular Dr. Stephan Mayer, for supporting the project. Especially, I would like to express my sincere appreciation to Mr. David Evans, the responsible technical officer at ESA, for his continuous support. His unwavering dedication to the progress of the mission encouraged new ideas and was instrumental in the success of the entire project, and the results presented here. I would also like to express my gratitude to the current and former members of the OPS-SAT Mission Control Team for their excellent work and the enriching collaboration. I am also grateful for the opportunity to work with the exceptional team at the Institute: the dedication, the interdisciplinary approach, the open exchange, and the mutual support are further building blocks to the project's success. This thesis has benefited greatly from that, as well as from our discussions and the proofreading.

My heartfelt thanks go to my family, who were most affected by my lack of that most scarce of resources, time, and who have always supported me. Especially Kerstin, for her utmost patience and for giving me the room I needed for this whole endeavour.

---

<sup>1</sup>The OPS-SAT project was carried out under ESA Contract No. 4000113614/15/D/SR and was funded through the ESA General Support Technology Programme (GSTP).

<sup>2</sup>The work on this thesis was supported by ESA's NPI programme under ESA Contract No. 4000122372/17/D/MB



# Abstract

Modern satellites offer an extraordinary opportunity for flying new innovative applications and new concepts. A wealth of state-of-the-art, commercially available hardware, supported by software written by an open community of developers, could revolutionise the idea of what a satellite can do. However, as satellites operate in a harsh environment with limited visibility and accessibility, there is a healthy scepticism about flying such new developments when not essential to the mission. The aim of the OPS-SAT mission is the synthesis of this apparent contradiction. The spacecraft allows a large variety of configurations and experimental applications. To facilitate this, at the centre of the payload is a powerful processing platform which is well connected and equipped with a reconfigurable Field-Programmable Gate Array (FPGA). However, there is little experience with reconfigurable computing in space applications. This thesis discusses and describes the operational challenges faced on OPS-SAT and the resulting solutions, e.g. advanced recovery scenarios, adaptive testing and debugging capabilities, and new (hardware) implementations, all of which were implemented in orbit. Furthermore, it demonstrates the high potential and the increased resilience to failures achieved due to these reconfiguration capabilities. It also shows how the availability of such a platform can change the on-board development process to become faster, more adaptable and more agile. These findings could be valuable for upcoming missions in providing experience and ideas for an efficient and safe use of this technology. Furthermore, it may encourage the space industry to utilise more reconfigurable hardware in future missions, as it has now been demonstrated to offer enhanced adaptability throughout all phases of a mission. At another scale, it shows how reconfigurable computing has the potential to expand the solution space for any hard-to-access system, including but not limited to, space applications.

### *Acknowledgements*

**Keywords:** Space technology, Artificial satellites, CubeSat, Electrical engineering, Low-earth-orbit (LEO) satellites, Reconfigurable computing, Reconfigurable devices, Reconfigurable logic, System-on-chip, Field-Programmable Gate Arrays (FPGA), Space communications, Satellite communication, Radio links, Failure analysis, Fault diagnosis, Logic testing, Data buses.

**Simple summary:** Reconfigurable hardware, and in particular field-programmable gate arrays, offers the potential to extend the capabilities of a (flying) spacecraft in an adaptive manner. This is demonstrated on ESA's OPS-SAT mission with several distinctive, novel in-orbit applications.



# Contents

<b>Preface</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>vii</b>
<b>1. Introduction</b>	<b>1</b>
<b>2. OPS-SAT Mission and Spacecraft</b>	<b>7</b>
2.1. Mission Features . . . . .	7
2.2. Communications . . . . .	9
2.3. Experimental Processing Platform . . . . .	10
2.4. Provided Software . . . . .	11
2.5. Payloads . . . . .	13
<b>3. Initial Design</b>	<b>17</b>
3.1. Experimental Categories . . . . .	18
3.2. System Level Fault Detection, Isolation and Recovery . . . . .	19
3.3. Processing Platform Safe Boot Strategy . . . . .	24
3.4. FPGA Reconfiguration . . . . .	34
3.5. Brief History of OPS-SAT Operations . . . . .	39
<b>4. Recovering SDRAM</b>	<b>41</b>
4.1. The Broken SDRAM . . . . .	44
4.2. Reviving the Second Core . . . . .	49
4.3. Conclusion . . . . .	50
<b>5. Boot Strategy Revisited</b>	<b>51</b>

## Contents

<b>6. Testing Bypass Mode</b>	<b>57</b>
6.1. Pre-Launch Testing . . . . .	58
6.2. In-Orbit Testing . . . . .	62
6.3. Control and Monitoring . . . . .	67
6.4. Results . . . . .	68
6.5. Conclusion . . . . .	71
<b>7. Implementing SpaceWire</b>	<b>73</b>
7.1. Evolution of an Alternative Link . . . . .	74
7.2. The SpaceWire Standard . . . . .	75
7.3. IP Core Selection . . . . .	76
7.4. Implementation and Deployment . . . . .	78
7.5. Results . . . . .	81
7.6. Conclusion . . . . .	84
<b>8. Debugging SpaceWire</b>	<b>85</b>
8.1. System-level Debugging Framework . . . . .	86
8.2. Integration on OPS-SAT . . . . .	89
8.3. Deployment . . . . .	92
8.4. Measurement Results . . . . .	93
8.5. Conclusion . . . . .	96
<b>9. Final Remarks and Outlook</b>	<b>99</b>
9.1. Summary and Lessons Learnt . . . . .	99
9.2. Conclusion . . . . .	103
<b>A. Kernel SMP Bringup</b>	<b>107</b>
<b>B. Supplementary Figures</b>	<b>109</b>
B.1. Spare Unit's eMMC Erase Counters . . . . .	109
B.2. X-band Signal Analysis . . . . .	110
B.3. Full SpaceWire Traces . . . . .	112
<b>Contributions</b>	<b>119</b>
<b>Bibliography</b>	<b>121</b>

## List of Figures

2.1.	OPS-SAT spacecraft standing in the clean room . . . . .	8
2.2.	OPS-SAT system architecture . . . . .	12
3.1.	Simplified architectural diagram of OPS-SAT . . . . .	20
3.2.	Simplified schematic diagram of the Cyclone V SoC . . . . .	25
3.3.	General boot flow of the Cyclone V SoC . . . . .	26
3.4.	Erase counters of the eMMC memory on OPS-SAT . . . . .	28
3.5.	Simplified program flow of the Cyclone V bootROM . . . . .	29
3.6.	Boot flow including possible fallback scenarios . . . . .	31
3.7.	Partitioning layout at launch time on the non-volatile memories	33
3.8.	General boot flow for a warm boot from OCRAM. . . . .	37
3.9.	Hardware boot partition on eMMC after SDRAM recovery. .	38
4.1.	Power consumption of the HPS in the early boot phase . . .	43
4.2.	Schematic diagram of the nominal and the rewired SDRAM structure . . . . .	45
4.3.	Revised boot flow after SDRAM recovery . . . . .	47
5.1.	Overlay filesystem . . . . .	52
5.2.	Revised partitioning layout on NOR flash. . . . .	54
5.3.	Revised partitioning layout on eMMC . . . . .	55
6.1.	Examples for test scenarios for ground and space application with the CCSDS Engine. . . . .	59
6.2.	Schematic test setup for the X-band radio test on ground . .	60
6.3.	Power spectrum of the X-band radio transmitting 50 MSym/s OQPSK . . . . .	61
6.4.	Schematic loop connection over the ground and space segment	63
6.5.	Diagram of the protocol encapsulation in the CADU. . . . .	64
6.6.	Simplified schematic diagram of the loop-back setup . . . . .	66

## List of Figures

6.7. $E_b/N_0$ and RSSI on ground and on the satellite. . . . .	69
7.1. Initial SpaceWire architecture on the processing platform. . .	79
7.2. High-level software architecture of the SpW implementation. . .	80
7.3. Revised SpaceWire architecture on the processing platform . .	81
7.4. Aggregated long-term per pass up- and downlink statistics of the SpW interface . . . . .	83
8.1. System architecture of an SLD-enabled remote debugging sys- tem. . . . .	88
8.2. System architecture for autonomously capturing SpaceWire operation. . . . .	89
8.3. Application flow of the trace collector. . . . .	91
8.4. Data-flow of the autonomous logic analyser function. . . . .	93
8.5. Two selected sequences of a logic analyser trace of the Space- Wire signal lines . . . . .	94
B.1. Erase counters of the Micron MTFC16GJDEC-4M eMMC on spare model . . . . .	109
B.2. Complex samples of the captured signal from the X-band transmitter. . . . .	110
B.3. Eye diagram of the captured signal from the X-band transmitter	111
B.4. SpaceWire logic analyser trace on 16/07/2021 00:37:14 . . . .	113
B.5. SpaceWire logic analyser trace on 29/10/2021 16:42:37 . . . .	114

# CHAPTER 1

## Introduction

The environment in space is harsh. There can be extreme and rapid temperature changes plus exposure to many types of radiation from which the Earth's atmosphere protects us. For electronic components on spacecraft, this can lead to accelerated ageing and increased likelihood of failure unless special care is taken. In addition, replacing or upgrading a failed hardware component on a spacecraft is usually not an option. As a result, electronic components for space are thoroughly tested and only selected if they are reliable enough to function in this harsh environment. In many cases, flight heritage is given as a selection requirement that narrows down the choice to a small group of - sometimes even obsolete - components.

All the hardware components of a spacecraft are thoroughly tested prior to launch. Numerous test campaigns take place at various functional levels to identify as many potential problems as possible that could occur during operation in orbit. These range from the component and subsystem level to the fully integrated spacecraft. This is necessary because simple hardware failures in orbit can have catastrophic consequences, including mission failure. *What if hardware functionality could be replaced or adapted on the fly? What would that change?*

The classic solution is to fly two identical redundant units, either in a hot standby configuration, where both units are kept operational and the redundant unit can take over quickly, or as a cold spare, where the other unit is switched on when required. However, technology has progressed to the point where it is now possible to use reconfigurable electronic hardware in space in the form of Field-Programmable Gate Arrays (FPGAs).

## *Chapter 1. Introduction*

In this case the redundancy is provided down to the electronic gate level, i.e. an array of logic gates (logic modules) and their interconnections, and not just at the unit level. FPGAs therefore can be seen as a logical extension to redundancy, but they have even more advantages, including the possibility to implement new hardware based functionality in flight.

FPGAs are already widely used for spacecraft, but usually their configuration is fixed at the time of launch. In-flight reconfiguration has been demonstrated, e.g. with the Configurable Fault-Tolerant Processor (CFTP) [69] used on MidSTAR-1 and NPSat [33], FedSat [21], CFESat [14], plus some recent and upcoming missions are using them [78, 41, 62, 65]. However, there is currently very little operational experience in routinely reconfiguring an FPGA in orbit.

This is where the European Space Agency's (ESA) OPS-SAT satellite comes in. This mission offers a unique platform for testing new operational and experimental ideas. To allow as many ideas as possible to be tested, one of OPS-SAT's central concepts is reconfigurability. This is provided by a wealth of different peripherals connected to the Satellite's Experimental Processing Platform (SEPP) which consists of a tightly-coupled FPGA and Central Processing Unit (CPU). The CPU can reconfigure the FPGA at run-time. So for the first time there is a mission which is flying a reconfigurable FPGA and a mission concept in which it will be reconfigured routinely throughout its lifetime. The OPS-SAT as a mission is described in detail in Chapter 2, and the initial design to enable it in a safe way is described in Chapter 3.

Besides the advantages of reconfiguring FPGAs in-field there are also inherent risks. One risk is an incomplete or wrong configuration of the FPGA, or that the applied configuration might show an unintended behaviour. Both cases can result in hardware damage of the FPGA, of the connected peripherals or even for both. Therefore, it is important to be able to revert from a new or incomplete configuration back to a known safe state as fast as possible to safeguard the computing platform. Section 3.4 presents the concepts and implementations to make the FPGA reconfiguration process safe and sufficiently volatile on the processing platform for OPS-SAT. These implementations allowed FPGA reconfiguration to become a routine operation.

Even with having appropriate support in-place for routine reconfiguration, the operational experience may be different than anticipated. Section 3.5 focuses on the operational reality and presents a chronological summary of selected events that occurred on the spacecraft while in orbit. These events are described in the subsequent chapters in a thematic order.

A failure in a satellite can seriously affect its potential performance. This is considered in every design phase and countermeasures are prepared. However, it is only possible to prepare countermeasures for expected failures. Therefore, unexpected problems in orbit cannot always be solved out of the box and the chances of finding a solution depends largely on the adaptability of the overall system. Chapter 4 describes how the presence of a reconfigurable FPGA coupled with redundancy measures helped to overcome even serious defects. Subsequent to these changes and due to recurring issues with the non-volatile memory, the boot strategy had to be revised. Chapter 5 describes the relevant changes.

As well as solving problems, adding new components can also be used to enhance a working system. Such components could be the solution to a new task or improve the performance of an existing function. Enhancing hardware functionality for a flying satellite is usually impossible. However, a spacecraft's functionality can be changed or enhanced with an in-flight reconfigurable FPGA. Chapter 6 outlines a radio commissioning procedure tailored to OPS-SAT's communication chain. Chapter 7 presents an in-orbit implementation of a SpaceWire functionality that provides higher communication bandwidth on board and, consequently, to the ground data systems. Chapter 8 presents a solution for an autonomous logic analysis on the spacecraft. All these implementations are results of real in-orbit scenarios and implementations.

In summary, the methods and implementations for FPGA reconfiguration used on OPS-SAT are described. These techniques were used to mitigate mission ending scenarios and circumvent severe damage. Furthermore, they helped to extend and modify the functionality of the satellite. This increased hardware flexibility extended the overall lifetime of this satellite mission, and it is hoped that this experience can inspire innovation in the space industry without compromising safety.

## Evolution of FPGAs in Space Applications

ESA's Rosetta mission used 45 FPGAs in total even for crucial functions covering control, data management and power units [30]. This shows how much potential in terms of processing power and adaptability this technology could bring to the space sector. Nowadays, FPGAs are widely used in space applications, especially on CubeSats, but the essential difference is that OPS-SAT allows routine reconfiguration of its FPGA in flight. By loading custom configurations to the FPGA it can be configured and tailored to virtually any use-case. It is only limited by its external connections to the buses. This provides a unique way of using an FPGA in an experimental context in the space environment. The potential of this technology ranges from enabling advanced mission operations using custom protocols to hardware-level adaptations at runtime with an unprecedented level of flexibility.

One of the first uses of space-borne reconfigurable FPGAs was on Fed-Sat, which was launched 2002 and carried on its HPC-I payload a Xilinx XC2V1000. It was used for on-board image processing [21, 19] and real-time signal processing [20, 77]. In 2007, the United States Department of Defense Space Test Program-1 (STP-1) [33] launched six satellites into Lower Earth Orbit (LEO). Among them were MidSTAR-1 and NPSat. Both were carrying the CFTP, developed by the Naval Postgraduate School, as a payload [69]. The CFTP consists of two reconfigurable FPGAs (Xilinx XQVR600), a flash configuration memory, and an SDRAM. One FPGA is dedicated as 'configuration controller' and the other one is a 'configurable processor' [23]. Another satellite on this STP-1 launch was the Cibola Flight Experiment satellite (CFESat). It was built by Los Alamos National Laboratory and was used for ionosphere and lightning studies. CFESat is based on a platform by Surrey Satellite Technology Ltd. and uses a reconfigurable FPGA for processing the captured signals [14]. The Ingenuity Mars Helicopter Team<sup>1</sup> utilised an on-board FPGA for control and switch-over between redundant units [8, 7].

Recent developments by leading manufacturers have leaned towards providing integrated hard processing capabilities, i.e. having a hard-synthesised

---

<sup>1</sup>The SpaceOps Awards for Outstanding Achievement 2023 were presented to the Ingenuity Mars Helicopter Team and the OPS-SAT Team.



processor core and peripherals, together with an FPGA. This provides the flexibility and high-performance parallel processing capability of an FPGA with the power-efficiency and reliability of hard Intellectual Property (IP) cores. For example, two of the missions OPS-SAT was launched with (Eye-Sat and Angels) carry a Xilinx Zynq Z-7030 System-on-Chip (SoC), containing a reconfigurable FPGA, as their On-Board Computer (OBC) [62, 65].

A driving domain which typically combines high processing needs with rapidly evolving standards is telecommunications. Including a reconfigurable FPGA in the communication chain could allow a mission to benefit from the flexibility and to implement and test new protocols and communication standards while the mission is in progress. The FPGA can efficiently process the data from a Software-Defined Radio (SDR) receiver, but still keep the adaptability through hardware reconfiguration options. One example of a mission which specifically targets this type of application is the Fraunhofer On-Board Processor (FOBP) on the Heinrich-Hertz satellite (H2Sat) which was launched on 06/07/2023. It contains two radiation-hardened Xilinx Virtex-5QV FPGAs directly connected to the receive and transmit radio frontends and thus presents a versatile and powerful SDR platform. [41, 42, 64]

What makes the OPS-SAT mission different from many of the above examples is that it was not constructed to perform a pre-defined set of tasks but instead designed with a set of principles in mind. In order to maximise the offering to experimenters the satellite had to be powerful, reconfigurable, safe and open. During the design phase of the mission executed in 2013, the reconfigurability element was only software-based. However, in 2015, before Phase B2, ESA decided to expand that into hardware by incorporating two more requirements into the statement of work [10]. Accordingly, the processing platform should consist of two processors and at least one FPGA, and this FPGA should be reconfigurable in flight.

This change offered attractive possibilities, e.g. an experimenter could potentially change the satellite's original function, however, it also became obvious that there are inherent risks. These risks conflicted with another of OPS-SAT's design principles which was the satellite should be safe to operate by third party experimenters. Therefore it was necessary to develop

## *Chapter 1. Introduction*

well-designed procedures and safety concepts to provide both the reconfigurability and safety at the same time. As a first step, the reconfiguration has to be performed with appropriate checks, tailored to the specific platform, to verify successful programming. Later on, the FPGA's function needs to be monitored and appropriate counter-measures have to be performed in the event of problems. This is particularly important in the space environment where radiation and large temperature variations can affect the process.

For OPS-SAT, FPGA reconfiguration needed to be seen as a routine operation for all operational and experimental use cases. Furthermore, direct access to this powerful tool and indeed to the whole satellite platform, needed to be enabled for a growing community of experimenters. This approach and this open access are unique for a satellite in-flight. This thesis describes how this was achieved with emphasis on the computing platform, especially regarding its reconfiguration abilities.

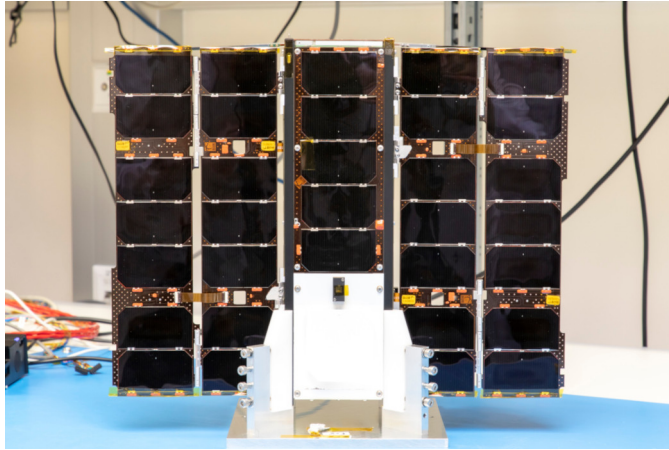
# CHAPTER 2

## OPS-SAT Mission and Spacecraft

OPS-SAT is a flying hardware and software laboratory in a form of a triple unit CubeSat. It was designed and built for ESA by a consortium of European partners, i.e. GMV Poland, Gomspace, Berlin Space Technologies, Magna Steyr Aerospace, MEW Aerospace, the Polish Space Research Centre and Creotech Instruments, led by Graz University of Technology and Unitel as prime contractor. It was launched in December 2019 and operations of the satellite are carried out by European Space Operations Center (ESOC) from the SMILE Lab ground station located in Darmstadt[57]. Occasionally, Graz University of Technology supported tests using the ground station in Graz. The mission's primary goal is to break the "has never flown, therefore it will never fly" cycle, preventing many ideas from ever reaching the real world environment, as they are deemed too risky, or too expensive to implement and test. In order to achieve this, OPS-SAT provides a set of generic capabilities and subsystems that are commonly found in space missions, large and small. Not only are these capabilities made available to the experimenters, but also the level, easiness, and lack of restrictions to make use of OPS-SAT's capabilities are brought to the next level through provision of an open-source space and ground software framework [27, 58].

### 2.1. Mission Features

To facilitate a broad range of potential applications, the spacecraft includes both conventional and novel subsystems. The subsystems comprise a programmable Satellite's Experimental Processing Platform (SEPP) with a dual



Credit: TU Graz - Lunghammer

Figure 2.1.: OPS-SAT spacecraft standing in the clean room facility at Graz University of Technology with its solar panels deployed.

core ARM CPU and an FPGA, a coarse and high-precision Attitude Determination and Control System (ADCS), a Global Navigation Satellite System (GNSS) receiver, a High Definition (HD) camera, an Ultra High Frequency (UHF) SDR receiver, an optical receiver, an S-band radio transceiver, an X-band transmitter, and a retroreflector [27]. Some of the components and subsystems used are Commercial Off-The-Shelf (COTS) products. The integration of these components has posed challenges, which have been and are being addressed through the use of the reconfigurable processing platform or, where available, their own reprogramming capabilities.

To date, already more than 250 experiments were executed on behalf of a growing community of experimenters. Among them are space agencies, universities, multi-national corporations, and one-man shows [39].

For such a diverse community of experimenters with different backgrounds, resources, and areas of interest, multiple access mechanisms are provided. These mechanisms range from remote execution of experiments with real-time telemetry and forwarding of telecommand connections to higher-level application interfaces, such as file uplink and downlink or data mirroring from space to ground [58].

## 2.2. Communications

To simplify the testing and validation process required to qualify the experiment for execution in orbit, scheduled remote access times to the satellite's engineering model on the ground are also provided. The access slots can be used to test software, simulate operations, and become familiar with the operating environment [58].

## 2.2. Communications

The radio system is an integral part of any satellite, but is even more important for OPS-SAT's experimental mission with high upload and download volumes. In addition, experimental use of it is also planned. The primary communication chain consists of an X-band transmitter (Syrlinks EWC27) and an S-band transceiver (Syrlinks EWC31), both interfaced to the "CCSDS Engine" responsible for Consultative Committee for Space Data Systems (CCSDS) compatible framing.

### CCSDS Engine

The CCSDS Engine is at the centre of the nominal high data rate communication chain and transparently handles the protocol translation. It has been jointly developed by the Space Research Centre of the Polish Academy of Sciences and Creotech Instruments SA. Its processing unit is a Microsemi ProASIC3 FPGA A3PE3000L. It is designed to reliably transmit space packets to and from the radios with a transmission and coding layer that conforms to the specifications of the CCSDS. It acts as a bridge between the radios and the satellite buses. On the satellite bus, the CCSDS Engine accepts and transmits data primarily over the Controller Area Network (CAN) bus, and thus provides radio connectivity to the OBC and the processing platform. For the processing platform it also provides a direct Low-Voltage Differential Signalling (LVDS) interface, which is configured for SpaceWire operation by default, but for experimental purposes it can also work in a "bypass mode". Then the entire protocol implementation is bypassed and direct access to the radios enables the use of custom protocols on the communication links [11, p. 217].

## *Chapter 2. OPS-SAT Mission and Spacecraft*

### S-band Telemetry and Telecommand subsystem

The primary radio on OPS-SAT for space-to-ground communication is the S-band link. The full-duplex Syrlinks EWC31 S-band transceiver provides the capability for uploading software to the spacecraft and to download large data volumes from on-board experiments with a fixed data rate of 256 kbit/s on the uplink and three selectable data rates on the downlink: 200 kbit/s, 500 kbit/s, and 1 Mbit/s. The data rate can be configured via the I<sup>2</sup>C interface. The output power is 31 dBm at the output of the duplexer. The transceiver is fully compatible to the CCSDS standards and to the ESA S-band ground segment. The radio system also contains a duplexer for allowing full duplex operations, and a power coupler for attaching two antennas. The antennas are mounted on opposite faces to obtain a nearly omnidirectional coverage. The data interface from the radio to the CCSDS Engine is an RS-422 point-to-point link [11, p. 222].

### X-band Telemetry Transmitter

The Syrlinks EWC 27 HDR-TM X-band communication system provides a high data rate downlink ranging from 3 Mbit/s up to 50 Mbit/s. The X-band transmitter is fully compatible with the ESA X-band ground segment. The transmitter is accompanied by a suitable antenna on the  $-Z$  face. It uses LVDS as data interface to the CCSDS Engine and I<sup>2</sup>C as control interface [11, p. 255].

## 2.3. Experimental Processing Platform

The Satellite's Experimental Processing Platform (SEPP) is the core of the experimental capabilities provided by the platform. The board was developed by Graz University of Technology. Exceptional computational power is achieved with the Intel/Altera Cyclone V SX SoC. The SoC has a hard-synthesised portion, the Hard Processing System (HPS), featuring a dual core ARM Cortex-A9 CPU with 800 MHz clock rate, and a Cyclone V FPGA [11, pp. 173]. The

## 2.4. Provided Software

- HPS has 1.5 GB (1 GB with Error Correction Code (ECC) enabled) Double Data Rate 3 Low-power SDRAM (DDR3L SDRAM), and the
- FPGA has 512 MB of the same type of Random-Access Memory (RAM)

accessible. As non-volatile storage media it hosts a

- Micron N25Q00AA NOR flash (128 MB) and a
- MTFC16GJDEC-4M embedded MultiMediaCard (eMMC) (16 MB).

On this processing platform the experimental software and firmware is loaded and executed. The overall system diagram of OPS-SAT with its payloads and buses is shown in Figure 2.2.

## 2.4. Provided Software

The software available to efficiently operate the processing platform comprises

- a reference Altera Cyclone V SoC design
- a baseline configuration and FPGA IP cores handling the processing platform interfaces like CAN, I<sup>2</sup>C, Serial Peripheral Interface (SPI)
- an embedded Linux system image based on Ångström containing
- Python 3.5
- Java SE Embedded Runtime Environment Version 8 Update 131
- a protocol bridge applications allowing access the CCSDS-compliant data streams on CAN and SpaceWire buses through a Transmission Control Protocol (TCP) socket<sup>1</sup>
- a set of user space libraries with drivers for all of the payloads connected to the processing platform and the
- NanoSat MO Framework (NMF)

---

<sup>1</sup>The SpaceWire bridge application is described in Chapter 7.

## Chapter 2. OPS-SAT Mission and Spacecraft

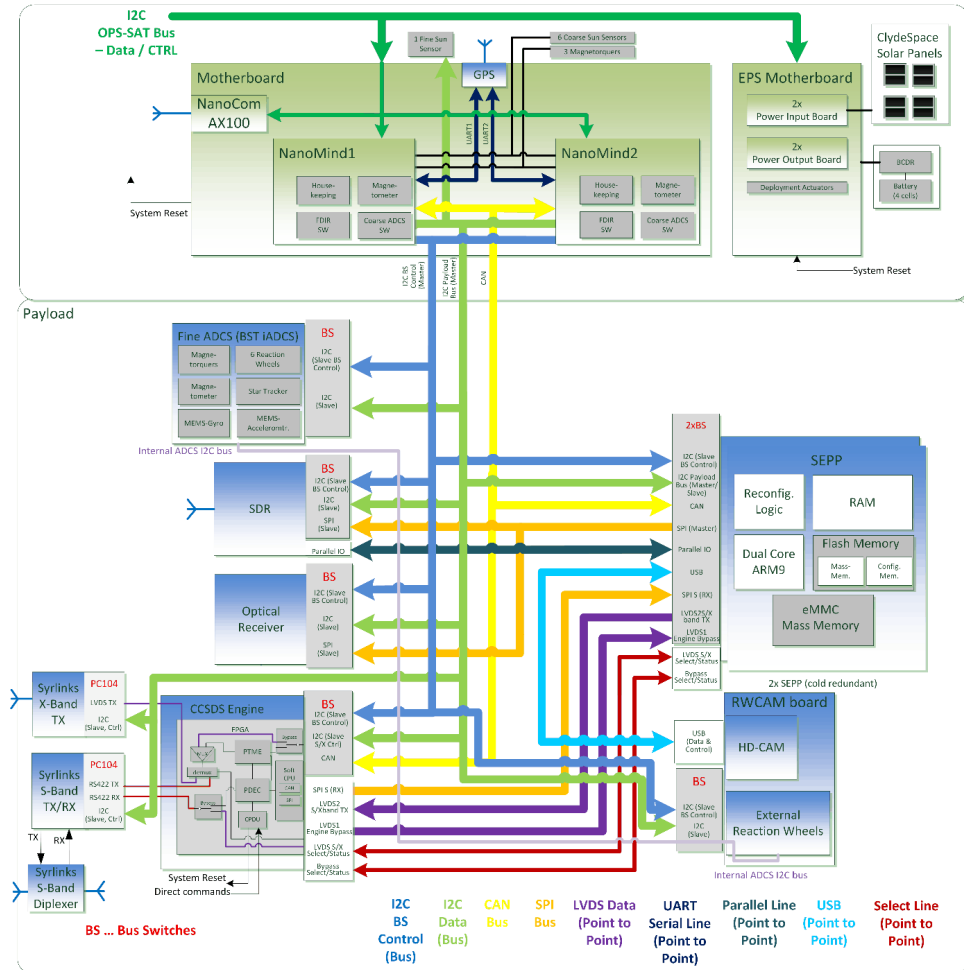


Figure 2.2.: OPS-SAT system architectural diagram [11] showing the core bus (top) and the payload (bottom) section.



## 2.5. Payloads

The NMF is a set of Java libraries and applications to quickly develop, test, and deploy custom applications to be run on the spacecraft. This is achieved with an abstraction layer of the hardware interfaces on the spacecraft, and a communication stack compliant with the CCSDS Mission Operations Monitor & Control Services standard [59] to interact with the application. This allows experimental software to be written independently of the actual satellite environment, to be tested in a simulated environment, and to communicate using the appropriate protocols. The core component on-board, i.e. the NMF supervisor, accepts and handles requests from ground and provides the primary experiment execution mechanism on OPS-SAT [58].

## 2.5. Payloads

OPS-SAT offers a variety of payloads that allow a wide range of experimental operations to be performed. These include a Global Positioning System (GPS) module, an independent high performance ADCS, retroreflectors, an optical receiver, a camera, a SDR receiver and a communication subsystem consisting of a unified, standards-compliant interface to an S-band transceiver and an X-band transmitter. The payloads are complemented by an experimental processing platform that provides outstanding computing power and through which they are interacted with and controlled. Each of these payloads is briefly described in the following paragraphs.

### GPS

Directly connected to the OBC is the NovaTel OEM615 GPS module, which is accessible via a Universal Asynchronous Receiver-Transmitter (UART) interface and supports the core ADCS and time-keeping tasks. [11, p. 233].

### Fine ADCS

The iADCS-100 from Berlin Space Technologies (BST) provides an additional ADCS solution, that has finer control than the core ADCS and is independent. It is available to the experimenters to perform their own ADCS

## *Chapter 2. OPS-SAT Mission and Spacecraft*

control of the satellite. It contains a set of sensors and actuators such as the ST-200 star tracker and miniature reaction wheels. It is connected to the payload I<sup>2</sup>C bus. [11, p. 234] It has the following characteristics:

- a star tracker (BST IMS-200),
- magnetometer and 3 gyroscopes,
- 6 reaction wheels,
- 3 independent magnetorquers, and
- autonomous operating modes such as Detumbling, Sun Pointing, Target Pointing and Nadir Pointing.

### Retroreflectors

As an alternative method for localisation and attitude determination from ground, OPS-SAT has retro reflectors to allow laser ranging. As retro reflectors are passive components, they can be used beyond the mission lifetime, e.g. for orbital decay monitoring. The reflector configuration is a pyramid arrangement with four of them on the  $-Z$  face. Combined, they provide a 180° aperture angle easing acquisition from ground. [11, p. 267]

### Optical Receiver

An active payload for receiving the laser ranging signal mentioned before is the optical receiver. It acts as an endpoint of an optical uplink from a laser ranging station modulating a data stream on its beam. The optical receiver contains a multi-pixel photon counter module to receive the incoming optical signal. The data rate is in the range of several kbit/s. The board is controlled by the SEPP via SPI and I<sup>2</sup>C interfaces. [11, p. 262]

### High-Definition Camera

Another optical payload is the IMS-100 HD camera from BST. The HD Camera uses the same miniaturized electronics as the ST-200 star tracker family, which is also used in the fine ADCS. The sensor and microcontroller of the ST-200 have been tested with proton irradiation of 130 MeV for a Total Ionising Dose (TID) of 10 krad. It can provide still images as well as video. In contrast to the star tracker, a colour sensor with RGGB bayer pattern is used. [11, p. 244] It has the following specifications:

- resolution of up to  $2048 \text{ pixels} \times 1944 \text{ pixels}$ ,
- frame rate up to 5 Hz with full resolution, and
- ground resolution of 60 m/pixel.

### Software Defined Radio Receiver

In addition to the three other radios, OPS-SAT also includes a Lime Microsystems LMS6002D field programmable Radio Frequency (RF) chip with a suitable RF frontend. This provides a multi-band, multi-standard transceiver. On OPS-SAT only the receive operation is foreseen and therefore the transmit function is disabled in hardware. The SDR board was designed and integrated as part of the processing platform stack to allow a seamless access to it. Combined with the processing capabilities on the processing platform it can be used as a powerful SDR solution. It has an

- input frequency range of 0.3 GHz to 1.575 GHz, a
- sensitivity of  $-98 \text{ dBm}$ , a
- sampling rate of 1.5 MHz to 40 MHz at 12 bit depth, and an
- LNA gain for 430 MHz to 440 MHz of 19.5 dB to 20 dB (including LPF).

The SDR is optimised to receive data in the 70 cm band and connected to a dipole UHF antenna. The antenna's centre frequency is at 425 MHz, and has a 10 dB bandwidth of 75.9 MHz. [11, p. 258]



# CHAPTER 3

## Initial Design

OPS-SAT acts as a flying laboratory, allowing experimentation using any of its diverse payloads in-orbit. Thus, OPS-SAT can be used to test and demonstrate new technologies in the field and let experimental technology gain flight heritage without risking a large, expensive satellite. The mission objective below summarises the overall goal of the OPS-SAT mission:

The OPS-SAT Mission provides a cost effective and reconfigurable platform allowing ESA and its designated experimenters for in-orbit demonstration of mission critical software, operations concepts and new technologies in order to minimise risks and to foster and accelerate innovations for other ESA missions.

[9, OPS-SAT Mission Objectives, p.8]

Although the objective is clear in a general sense, there is no statement of a *specific* task to be performed. Rather, OPS-SAT is intended to provide a highly flexible platform for a variety of hardware/software experiments, e.g. in the areas of on-board autonomy, telecommunications, remote sensing, artificial intelligence, attitude control or network security. However, since the mission is not limited to a specific task, the system requirements, such as power consumption or temperature limits, cannot be derived from the mission requirements. Therefore, only the outmost limits of the spacecraft were defined and then kept within bounds by monitoring and operational measures.

The future use of the spacecraft cannot be predicted, but it is possible to categorise future experiments according to their potential impact on the

spacecraft. This helps to plan the necessary operational and monitoring measures and to organise their implementation accordingly.

### 3.1. Experimental Categories

The OPS-SAT mission statement is further refined in the ESA mission requirements document [9] to "OPS-SAT shall provide a safe, hard/software laboratory, flying in a LEO orbit, reconfigurable at every layer from channel coding upwards". It should be "available for authorised experimenters to demonstrate innovative new mission operation concepts."

In this sense, an **experiment** can range from software only to a hardware design implemented in the FPGA. Also, even though only the communication chain is implied with "channel coding upwards", the experimenters can potentially access any payload from the processing platform and operate it. This permits a high degree of freedom for a prospective experiment. The problem space is therefore extremely large. The only thing that two experiments might have in common is they are based on an idea and are looking for a platform in space where it could be tried for real. A key requirement of the mission is the ability to return the spacecraft to a safe and known configuration in the event of a malfunctioning experiment, but also from externally induced events like single event-upsets. To prevent any damage to the satellite multiple layers of safeguards have been put in place, but critically the effort required will vary depending on the experiment category.

As a way to operationally distinguish experiments and determine the necessary safeguards, they are categorised into three groups. Each is described below starting with those with the least potential risk and ending with the highest risk. Firstly, a (software) experiment can run in a prepared, common framework to seamlessly use the resources on-board. This framework is implemented in Java and called the NMF [16]. All experiments executed therein belong to the group of **NMF experiments**. They benefit from a software development kit, plus a test and supervising execution environment. In addition, the remote interaction is performed according to the CCSDS Mission Operations (MO) Services standards [60]. Examples of this type of

### 3.2. System Level Fault Detection, Isolation and Recovery

experiment are the machine learning algorithms to classify pictures on-board taken with the on-board camera [56, 32].

Secondly, an experiment can run directly as a user process in the Linux operating system. This provides a more direct access to the peripherals without a supervising layer. The increased flexibility also bears the risk of unintended configurations and higher requirements on the software implementation. In addition, there is no specific mechanism for a direct interaction. In (lingual) contrast to the previous type, they are called **non-NMF experiments**. An example of this type of experiment is the search and rescue beacons (Cospas-Sarsat) decoder implemented with GNURadio and the SDR receiver [61].

The third type of experiment are ones that provide their own FPGA configuration and therefore need the hardware reconfiguration capability. This type allows the most low-level access to the platform and its capabilities. This requires a higher effort and a deeper understanding of the FPGA and the platform hardware to implement on the experiment side and also to check on the operations side. These are termed **FPGA experiments**. Examples of this type of experiment are an in-space SpaceFibre demonstrator [15] and picture classifiers implemented in the FPGA [49, 29].

### 3.2. System Level Fault Detection, Isolation and Recovery

In order to ensure a safe environment for experimentation, OPS-SAT implements many layers of protection. The system level Fault Detection Isolation and Recovery (FDIR) mechanisms described in this section are the result of multiple design iterations in coordination and collaboration with the customer. The concepts and mechanisms on system level are described in the system design report [11, ch. 4, 5], and as an overview they are summarised in this section.

Hierarchically, the highest protection layer is managed by the Electronic Power System (EPS) as depicted in Figure 3.1. OPS-SAT uses a Gomspace NanoPower P60 to supply all payloads. It monitors over-current or short-circuit events and, depending on the configuration, turns off, or power cycles, the affected devices. In addition, all experimental payloads have power bus

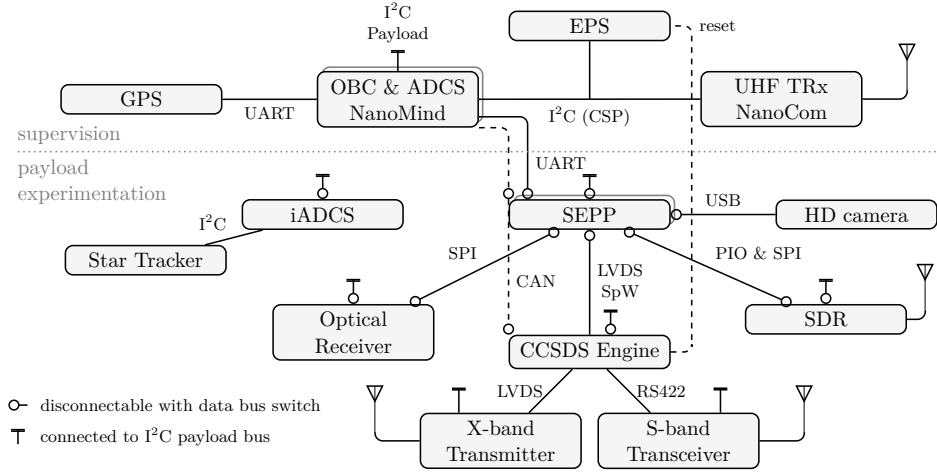


Figure 3.1.: Simplified architectural diagram of OPS-SAT omitting control busses. The top, hierarchically highest, region contains the subsystems providing the core satellite functions. They supply, monitor and control the safe execution of the payloads accessible for experimentation. The experimental subsystems in the lower region are supervised with the omitted bus switches. Only the data bus switch separation function is depicted with circles. Most of the peripherals are accessible over the I<sup>2</sup>C Payload bus.



### *3.2. System Level Fault Detection, Isolation and Recovery*

switches on their boards which can monitor and eventually isolate the payload from the system if triggered or commanded. Thereby, they also allow a more fine-grained view and control of the internal power distribution and dissipation. The bus switches are operated over a dedicated I<sup>2</sup>C bus, which is controlled by the OBC. Tightly integrated in this safety scheme is the Gomspace NanoMind OBC. It is solely responsible for controlling the operation and configuration of the above mentioned power bus switches. In a similar fashion to the power bus switches, data bus switches are also integrated on to the payload boards allowing the OBC to isolate them from each respective bus. Taken together, this means a payload can be completely separated, in terms of power supply and data lines, from the system.

Apart from the bus switch control, the OBC is also responsible for basic attitude control, temperature and spin rate monitoring of the spacecraft. In case of an anomaly it puts the spacecraft into a safe mode, which powers all experimental payloads off and activates its own detumbling mode. Therefore, a misbehaving experiment which could potentially harm the spacecraft, i.e. thermally, power-wise or attitude related, will trigger an autonomous reaction, and this returns the spacecraft to a safe known state. This transition and experiment abort mechanism is categorised as a soft safety trigger [11, p. 52].

In case the OBC could not detect the anomaly or is not operational itself, ground-driven safety mechanisms are available. The most obvious one is direct control of the EPS via the NanoCom UHF transceiver. This separate access route can be used to investigate the status of the core peripherals and configure the EPS without the help of the OBC [11, p. 53].

The EPS, the OBC and the UHF transceiver, together with the bus switch concept, form the core bus of the spacecraft. Together they provide the basic power, monitoring and control functions for the entire system. Thus, they could be seen as an independent spacecraft inside the spacecraft. Actually, the tasks of electrical power generation, storage and distribution, radio communication, monitoring of temperature and rotation, and attitude control, are all managed by this core bus. In addition, and supported by the power and data bus switches, they supervise the operation of the experimental section. As its function is not essential for the spacecraft's safety, the experimental section has relaxed requirements regarding reliability. Thus, it

### *Chapter 3. Initial Design*

is possible to include less tested and experimental payloads there. The payloads interact with the core bus, however, they can be interpreted as just an extension to it.

For example, in case the UHF chain is unavailable, then the EPS can also be reset over the S-band chain. A dedicated command can be issued via this channel to pull the EPS reset line. Eventually, the whole satellite, including the EPS, will reboot and come up in safe mode. In contrast to the software-controlled transition to safe mode by the OBC, this is categorised as a hard safety trigger [11, p. 53].

The core of the experimental section of the spacecraft, connecting all the aforementioned peripherals, is the processing platform, and it is therefore also termed Satellite’s Experimental Processing Platform (SEPP). It enables control of the experimental peripherals and is well connected, with unique direct connections to the SDR and the CCSDS Engine, which have the highest data bandwidth requirements.

Similar to the spacecraft’s bus switch concept, the processing platform is also equipped with bus switches to monitor and control the internal power supply, protecting the system from failures and to guarantee a safe and reliable system operation. Data bus switches can be operated to connect or disconnect interfaces to the spacecraft’s bus, as needed. The bus switch concept also enables to connect several processing platform modules in cold redundancy to the same power supply and data lines. Therefore, OPS-SAT contains two identical, stacked processing platform modules and a respective processing platform can be activated by enabling its corresponding power bus switches [11, pp. 172].

Due to its central role, the processing platform is implemented with reliability and performance in mind. To meet the spacecraft CubeSat form factor, it is implemented as a compact Printed Circuit Board (PCB). Outstanding computational power is achieved with the Intel Cyclone V SX SoC. It consists of a hard-synthesised portion, commonly referred to as HPS, featuring a dual core ARM Cortex-A9 CPU with 800 MHz clock rate, and a Cyclone V FPGA [81].

The HPS and FPGA portions are connected to the peripheral hardware as depicted in Figure 3.2. The HPS has a DDR3L SDRAM of 1.5 GB, with

### 3.2. System Level Fault Detection, Isolation and Recovery

512 MB dedicated to ECC, attached. The FPGA portion has 512 MB of the same type of RAM accessible. Moreover, it contains two non-volatile memories for operating system and application data: A 16 GB eMMC memory (NAND flash) and a 1 Gbit NOR flash attached to the Quad Serial Peripheral Interface (QSPI) controller.

In addition to the memory controllers, there are also peripheral controllers that allow connections to the spacecraft bus. The central I<sup>2</sup>C payload bus and the CAN bus are directly accessible on the chip. The CCSDS Engine can be accessed via pins connected to the FPGA portion. The data interface to the SDR is solved in the same way. However, only a suitable FPGA design can make use of the connections to the FPGA portion. Fortunately, it is possible to forward pins of the HPS peripheral controllers, the Input/Output (IO) blocks and its interface, to the FPGA fabric and vice versa. Then the switching matrix of the fabric can be used to connect the pins exported in this way to any FPGA pins or to use them within the fabric. This offers a high degree of flexibility for rewiring, for example.

Apart from the exported pins, the functionality implemented in the FPGA can be connected to the HPS part via bridges. There is a light-weight, in terms of its bus width, HPS-to-FPGA bridge (LWH2F) accessible via the L3 slave peripheral switch. It provides a simple and easy to implement slave memory interface to the FPGA. The more powerful variant is the normal HPS-to-FPGA (H2F) bridge. To enable access from the FPGA to the L3 main switch in the HPS part, the FPGA-to-HPS (F2H) bridge can be used. In this way, the fabric can be given configurable access to the entire HPS and its peripherals. In addition, there is also an FPGA-to-SDRAM (F2S) bridge that allows direct access to the HPS Synchronous Dynamic Random-Access Memory (SDRAM) controller.

Although these bridging functions allow each portion of the SoC to access the other one, an additional software implementation effort is also required. OPS-SAT uses an embedded Ångström Linux distribution and additional user software stored on the eMMC flash memory. In such a multitasking and dual-core environment with independent read and write access by multiple processes, coherent memory access, controlled bridge access and effective buffer back-pressure mechanisms are essential for interfacing with the real-time FPGA portion.

Another notable feature regarding the following section on booting is the integrated bootROM and On-Chip Random Access Memory (OCRAM). They provide the necessary machine code and memory to bring up the Microprocessor Unit (MPU) without the SDRAM controller being properly configured in the first place. The early boot stages executed from these on-chip memories allow the SDRAM to be initialised, make it accessible, and then to load the following boot stages to it.

### 3.3. Processing Platform Safe Boot Strategy

After power-up of the processing platform it is important to reach a known and operationally usable state. The visibility of the boot process and the possibility of influencing it is limited to the passes, i.e. to the times when the satellite is in the visibility of the ground station. In the case of OPS-SAT, this is a maximum of ten minutes per pass, with a maximum of six passes per day. Therefore, the whole boot flow should take autonomous decisions, but at the same time, should act deterministic. After completion, it should come up in a state where it is ready to communicate with the outside world. If not, then it should be clearly visible where and why the boot process stopped, to allow further action to be taken. Developing and implementing an appropriate strategy fulfilling these requirements was one of the initial objectives of this thesis. This was achieved through a combination of the manufacturer's chip specifications [34] and custom enhancements.

Resilience to physical and configuration errors is crucial for achieving such an autonomous boot process. The processing platform and the bootloader used<sup>1</sup> already offer several mechanisms to accomplish this, such as error correcting memory interfaces, multiple redundant boot sources, fallback boot device or scripted boot flows in redundant storage areas.

At the basis of a cold boot flow, as depicted in Figure 3.3, is a non-volatile storage media containing the software to be run eventually. The Cyclone V SoC on the processing platform supports as boot sources its FPGA portion, if it was configured before the HPS starts, and external flash memories. In

---

<sup>1</sup>"Das U-Boot"[74] was chosen as bootloader, because it provides a mature code-base and is maintained regularly by the manufacturer of the SoC.

### 3.3. Processing Platform Safe Boot Strategy

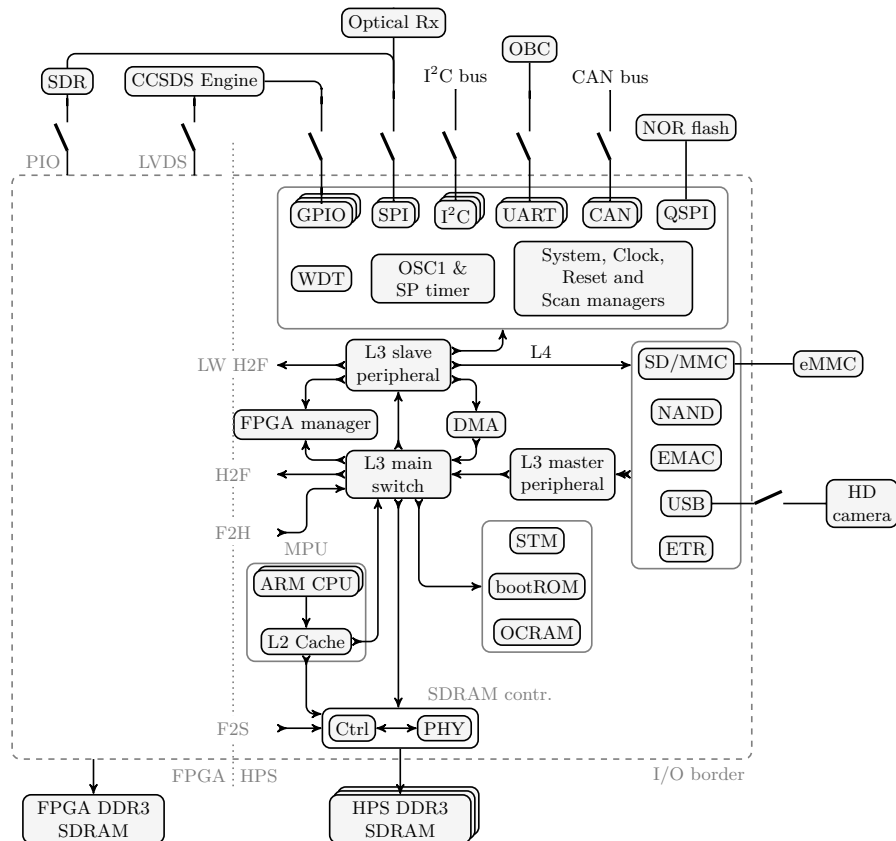


Figure 3.2.: Simplified schematic diagram of the Cyclone V SoC focussing on the HPS portion [34, p.1-4, p.7-2], the directly connected peripherals, and peripheral controllers. The peripheral controllers are placed next to the I/O border. Apart from the SDRAM controller, they are accessed over the L4 bus. The L4 bus is accessible via the L3 slave peripheral controller. Performant data transfers from peripherals are possible over the L3 master peripheral controller. The central element of interconnection is the L3 main switch facilitating direct access from the CPU and to the on-chip memories (bootROM and OCRM).



Figure 3.3.: General boot flow of the Cyclone V SoC. After reset, the bootROM is the first boot stage. It is responsible for loading the first custom bootloader, the preloader or Secondary Program Loader (SPL). Starting with the SPL, the customisable boot stages follows (marked in grey). Usually the flow consists of the SPL, a second level bootloader and the operating system.

the OPS-SAT flight configuration two flash memories are available as boot media: the eMMC attached to the SD/MMC controller, and the NOR flash attached to the QSPI controller. Before considering the boot process itself, these potential boot sources are described in more detail.

From a complexity point of view the simpler storage media is the Micron N25Q00AA NOR flash attached to the QSPI Flash Controller. It consists internally of four 256 MBit dies and provides a total capacity of 1 GBit (128 MB). The memory is addressable and readable down to single byte level [2].

For write operations, raw flashes (NOR- and NAND-technology) have to be erased first, because the individual bits can only be flipped in one direction. To revert the bit state, an erase operation has to be applied to a block, called erase block, which is 64 KB in size for the used NOR flash [2]. Consequently, data should be aligned to these erase block boundaries to save (block) space, and prevent unnecessary erase cycles.

Another notable aspect is the wear imposed on each erase block of the memory with every Program/Erase (P/E) cycle. As well as avoiding any changes, e.g. write accesses, it is also beneficial to spread the erase operations over a larger group of blocks, thereby spreading the wear. This wear-levelling keeps track of the erase counters and reassigns or allocates blocks as necessary. One software implementation, called Unsorted Block Images (UBI) [75], is readily supported by the bootloader used and the Linux kernel.

In contrast to this raw flash, these management operations can also be done by a controller in hardware, and integrated into a *managed* flash memory. An

### 3.3. Processing Platform Safe Boot Strategy

eMMC memory, which is also available on the processing platform, belongs to this class of managed flash memories. The interface, and its features are standardised [25]. It contains an integrated controller taking care of the previously described low-level accesses to the physical flash memory technology. The physical and logical erase block mapping and counters are depicted in Figure 3.4. The memory management handles tasks like block relocation for damaged areas, wear levelling or write protection settings.

Furthermore, eMMC memory can be split into hardware partitions. Each of these partitions has a dedicated memory management. Hardware partitions are classified as user data, enhanced user data, boot, Replay Protected Memory Block (RPMB) and general purpose. User partitions are the most generic ones. They are visible by default and provide the least features. Extended features are provided by enhanced user partitions. They have additional functions like fine-grained write protection and enhanced reliability settings. Boot partitions are intended for the early boot process. The memory can be switched to boot mode (usually at boot time) and the contents of the (selected) boot partition will be streamed out in a simplified fashion [25].

The eMMC standard [25] also defines several types of write protection. The default state of a memory chip is to have no write protection at all. Temporary write protection allows to protect certain parts of the memory even across resets or power-loss. This protection setting can be changed at any time. For power-on write protection it is different. It can not be changed while the memory is powered, but after a reset or power-loss the protected areas will appear as unprotected. The strongest variant of write protection is the permanent one. It can not be changed and survives resets and power-loss [25].

After power-up and releasing reset, the Cyclone V default configuration maps address zero to the bootROM. The first core of the ARM Cortex-A9 MPU then starts executing instructions from there. The bootROM startup code only runs on CPU core 0. If CPU core 1 comes up, the bootROM code directs it to continue from the address stored in register `cpu1startaddr`. The bootROM code, schematically depicted in Figure 3.5, initialises the necessary peripherals to find the next boot stage, termed preloader or SPL, at specific locations on a non-volatile storage media or on the H2F bridge

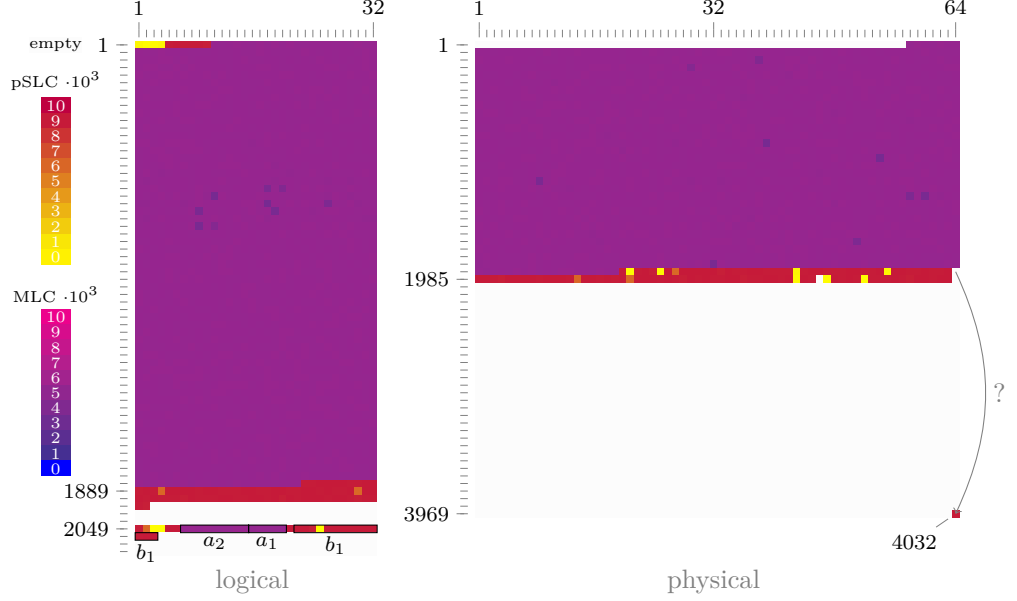


Figure 3.4.: Erase counters of the Micron MTFC16GJDEC-4M eMMC on OPS-SAT arranged by logical (left) and physical (right) blocks (8 MB each). The data was acquired on OPS-SAT on 13/01/2023 4:33:38 Coordinated Universal Time (UTC) with a custom utility. The beginning of the logical block area (top left) contains the two hardware boot partitions allocated as 10 pSLC blocks. After them follows the hardware user partition consisting of 1868 MLC type blocks. At the end are the areas  $a_1$ ,  $a_2$  and  $b_1$ . They are used as circular buffers for each type of area. Every write access causes a shift of the affected block into the circular buffer and the block shifted out gets written with the new data. This strategy maintains (at least part of) the wear-levelling mechanism. The wear-levelling is quite effective for the hardware user partition. Only the first boot partition is less well levelled, as it has been written to less frequently. When comparing the physical block mapping with another chip of this series (Figure B.1), one block (1984) is deallocated and instead another block (4032) is used.



### 3.3. Processing Platform Safe Boot Strategy

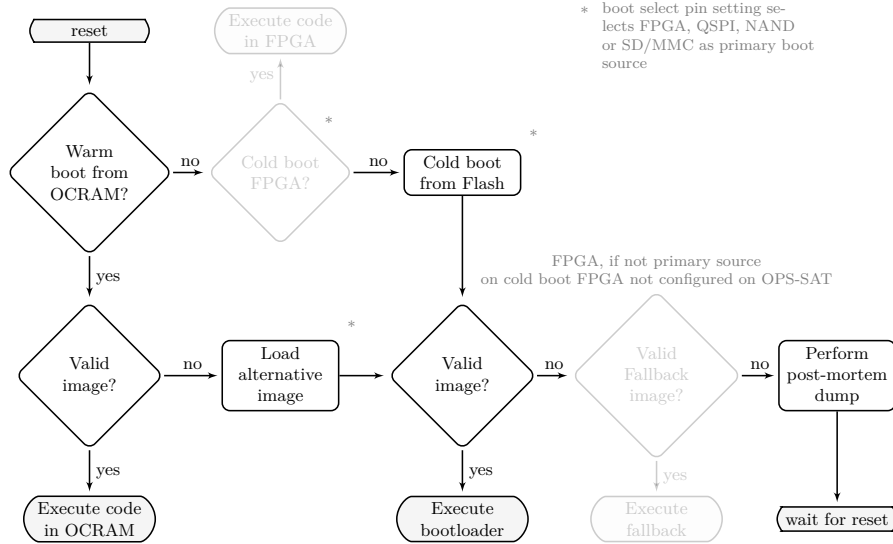


Figure 3.5.: Simplified flow diagram of the Cyclone V bootROM for CPU core 0. Loosely modelled after [34, p. A-27]. Depending on the type of boot (warm or cold) and the warm boot register (`warmramgrp`), the bootROM searches for a valid image in OCRAM. If it is not enabled or found, the search continues on alternative media. In the cold boot case and depending on the bootsel pin configuration, the FPGA is queried via H2F bridge for a valid image. If non-volatile storage media (QSPI or SD/MMC) was selected, it is searched, validated, loaded and executed. If the bootROM code is run by CPU core 1, it is directed to the address stored in `cpu1startaddr`.

exposed from the FPGA part. Its boot source selection depends on the setting of the boot select pins at power-up. It allows to select different storage locations like Secure Digital/Multimedia Card (SD/MMC), i.e. eMMC, QSPI attached (NAND/NOR) flash, direct NAND flash or the H2F bridge. Subsequently, it loads the found bootloader image to OCRM and jumps there [34, 37].

Given that the SD/MMC controller is selected as the primary boot source, the bootROM tries to locate a valid bootloader image on SD/MMC, i.e. eMMC memory. First, it tries to read the Master Boot Record (MBR) from the hardware user partition and locate a special partition with type 0xa2. Then it searches from the beginning four times in 64KB steps for a valid bootloader. Second, if no valid bootloader could be found, the search continues in the same way from the beginning of the hardware user partition (raw). If this was still not successful, a fallback image is searched on the H2F bridge and on error a post-mortem dump is placed in OCRM and it awaits reset. The boot process for a QSPI attached storage media is similar, except that the first step, i.e. the partition table search, is omitted. The attached flash memory is just searched from the beginning in 64KB steps for a valid bootloader. On error, the same strategy as for SD/MMC is followed [34, p. A-13].

If all went well, the bootROM has loaded a valid bootloader image to OCRM and handed control over to it. However, the OCRM provides only 64KB of storage, which is too small for boot data like a Linux kernel or in fact any advanced bootloader. Therefore, the primary task for the first-stage bootloader is the initialisation of the main SDRAM. The SDRAM is initialised with a dedicated procedure<sup>2</sup>, to calibrate the signal timing corrections, as they are susceptible to process, voltage, and temperature (PVT) variations.

However, the presented ideal multi-stage boot approach can potentially fail at several points. Apart from severe faults, i.e. inaccessible non-volatile storage media, there are also more light-weight corruptions which may prevent a successful boot of the processing platform. Therefore, a selection of different error conditions is examined and depicted in Figure 3.6. The boot from the

---

<sup>2</sup>Source code for Intel SoC Generation V SDRAM initialisation [74, v2022.10, `drivers/ddr/altera/sdram_gen5.c`]

### 3.3. Processing Platform Safe Boot Strategy

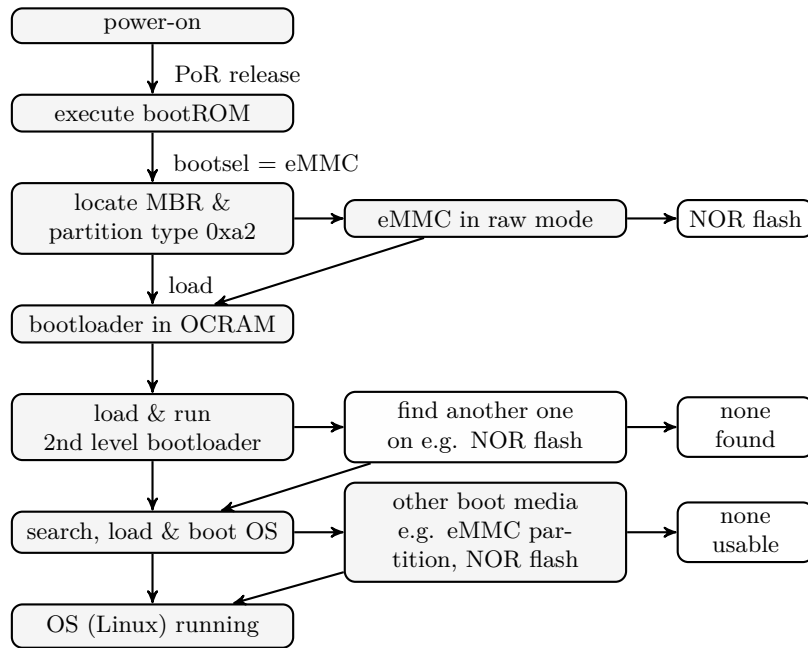


Figure 3.6.: Boot flow including possible fallback scenarios

### Chapter 3. Initial Design

H2F bridge is left-out due to unavailability in the present implementation. Furthermore, the analysis focuses on eMMC memory as it is the primary boot source.

In the bootROM phase, the first critical point is the detection of the dedicated boot partition containing the first-stage bootloader images. This depends on an intact MBR. In case its signature or partition information is damaged, the bootROM will fall back to raw mode and will stop on error. This case is accounted for with a valid image in the QSPI-attached NOR flash. It can be recovered by setting the boot select pins to boot from QSPI and issuing a platform reset.

The first stage bootloader, loaded and executed in OCRAM, provides more customisation options. Since it also initialises the main SDRAM and the interfaces for loading the next stage bootloader, it can also perform tasks like additional safety checks or a more advanced boot device selection. Here, it uses the same boot device as the bootROM and fails on error. On error, as with the bootROM, this can be handled with a change of the primary boot device. However, if the SDRAM calibration fails, as detailed in Chapter 4, there is by default no recovery possibility available.

The search for the next boot stage by the bootROM, as described above, and the resulting requirements are reflected in the memory layout. The memory layout of the non-volatile memories is depicted in Figure 3.7. The partitioning information of the NOR flash is not stored in a partition table on the medium, but is instead stored in the boot loader, its environment, or in its device tree. The start sectors of the flash contain four consecutive images, each 64 KB, of SPL code. The next boot stage, the full U-Boot bootloader, is located in a 512 KB area and its environment has a dedicated 64 KB space. The Linux kernel and its Device Tree Binary (DTB), containing the platform specific settings, also have dedicated memory areas. The main UBI filesystem is in one contiguous 90 MB area.

On eMMC, the bootROM's search is limited to the hardware user partition, i.e. no other *hardware* partitions are examined. The custom type `0xa2` partition is located via the partitioning information stored in the MBR. This partition also contains the four consecutive SPL images in the beginning and the next boot stage after that. The Linux kernel, its DTB and FPGA images are stored on a separate FAT-formatted 'boot' partition. The ext4 root

### 3.3. Processing Platform Safe Boot Strategy

Address	NOR flash	Size	Block	eMMC hardware user partition	Size
0	u-boot spl	256 KB	0	MBR	512 B
40000	env	64 KB	34	env	8 KB
50000	dtb	64 KB	50		
60000	u-boot	512 KB	2048	custom	1 MB
e0000	kernel	10 MB	4095		
			8192	boot	256 MB
ae0000	UBI fs	90 MB	532479		
			638976	root	4 GB
64e0000	FPGA	10 MB	9027583		
			9445376	boot backup	256 MB
6ee0000	spare	~17 MB	9969663		
			10076160	root backup	1 GB
8000000			12173311		

Figure 3.7.: Partitioning layout at launch time on the non-volatile memories, i.e. NOR flash and eMMC, of the processing platform.

filesystem lives on the 'root' partition. For redundancy these two partitions are replicated in corresponding backup partitions.

In summary, the boot process realised on the non-volatile memories of the processing platform consists of multiple independent stages, each of which can fail. After a cold start the HPS executes the hard-coded bootROM. It initialises the necessary controllers, and then loads the SPL bootloader stage from the non-volatile memory selected. The SPL initialises the SDRAM and loads the main bootloader. The bootloader fully initialises the system, and loads and boots the Linux kernel. However, there is still one open issue which is particularly important for the mission: the integration of the FPGA (re-)configuration into the boot process.

### 3.4. FPGA Reconfiguration

What has been left out in the discussion of the general boot strategy in the preceding section is the actual reconfiguration of the FPGA portion of the SoC. As this is a core function of the spacecraft, it must meet the safety and quality requirements of the spacecraft. However, reprogrammability is both a blessing and a curse: It offers the desired flexibility, but at the same time carries the risk of misconfiguration. An unintended or partial configuration of the gate array puts it into an undefined state. In the worst case, especially when configuring the output pins, this might lead to physical destruction. Also, excessive use of the FPGA fabric, e.g. in case of a misbehaving experiment, can lead to thermal destruction of the chip.

These types of faults can also be induced externally by radiation or temperature changes. This can affect the Cyclone V FPGA configuration stored in Static Random-Access Memory (SRAM) cells. The manufacturer states that "soft errors, which are caused by an ionizing particle, are not common in Altera devices" [22, p.8-2], but in general SRAM is highly susceptible to radiation [66, p.86],

One mitigation technique for radiation effects is partial reconfiguration or scrubbing of configuration memory [66, p.156], but, unfortunately, the Cyclone V version used on OPS-SAT does not support these functions, i.e. it has no 'SC' suffix in its part number [18, p.5]. However, partial reconfiguration could be synthesised as IP core and the error detection circuitry is available. Therefore, and because one of the mission's goals is to favour industrial components, another approach was chosen for improving the thermal behaviour and radiation hardness [83, 11, p.173]: the whole computing platform is enclosed in an aluminium housing and is thus less susceptible to radiation events<sup>3</sup>. So the focus is not on radiation hardness per se, but rather on improving radiation tolerance. This can be supported by reconfiguration in error cases. Reconfiguration is also seen as part of an operational strategy, because dedicated experiments will reconfigure the FPGA to use it for custom test cases.

In the case of a severe malfunction leading to a high current consumption,

---

<sup>3</sup>A summary of shielding and its effectiveness regarding radiation is given in [66, p. 176].

### 3.4. FPGA Reconfiguration

either the EPS or the power bus switch protection triggers and forces an autonomous power cut-off. In order to make it easier for experimenters to use the system by avoiding partial reconfiguration, which is not officially supported anyway, and to make the entire FPGA area available to them, only full reconfiguration is performed.

Regarding the FPGA configuration, there are two areas requiring a setup after power-up: The FPGA fabric itself and its interfaces to the outside. The former one will be covered later, and the latter one consists mainly of the pin configuration applied via the scan manager, and the HPS bridge configuration. This configuration is performed by the first bootloader stage, i.e. the preloader or SPL<sup>4</sup>. The rationale behind that is that in particular the SDRAM controller and the connected F2S bridge should be kept idle during the configuration. Failure to do so seems to result in a locked SDRAM controller.<sup>5</sup> This idle state can be achieved (almost exclusively) while the system is entirely running from OCRAM as in the SPL case.

There are different possibilities to configure the FPGA, but in the case of OPS-SAT, the FPGA configuration is performed over the HPS. This provides high flexibility as well as fast and reliable programming. The FPGA manager integrated in the HPS portion exposes all programming interfaces in a convenient way to allow live reconfiguration from the HPS with a read-back of the FPGA status. [34] So, after the bridge configuration is in place, the actual FPGA configuration can be done from the second-stage bootloader or at a later point. However, the most exclusive and direct control over the system, with only a single execution flow, is during the bootloader phase. Initially, all accesses from the HPS portion are disabled with a dedicated register. Then, the F2S bridge is put in reset as well as the other bridges, i.e. H2F, LWH2F and F2H bridge. The latter bridges are then unmapped from the L3 interconnect. Now, the FPGA portion is effectively isolated and the configuration procedure is safe to continue via the FPGA manager. After the configuration has completed, the bridges are activated in the reverse order.<sup>6</sup>

---

<sup>4</sup>see SPL code in `arch/arm/mach-socfpga/spl_gen5.c` [74].

<sup>5</sup>Although there is limited information by the manufacturer, there were discussions regarding the relevant U-Boot source code.[73]

<sup>6</sup>The entry point for the actual FPGA reconfiguration procedure in U-Boot is in function `socfpga_load` in `drivers/fpga/socfpga_gen5.c` [74].

### *Chapter 3. Initial Design*

Since the FPGA reconfiguration for experiments happens autonomously on the spacecraft, it has to be safe and revertible at any time. The general idea is to load a safe default configuration for the FPGA on boot and apply subsequent configurations volatile. To fulfil these requirements, a special boot mode of the Cyclone V bootROM allows to keep the entire configuration volatile.

The initial FPGA programming after a cold start is done as described above during the usual boot flow. Subsequent reprogramming employs the warm-boot method of the Cyclone V bootROM. The bootROM's flow is depicted in Figure 3.8. This allows a bootloader to be executed that was previously loaded to the OCRM. The bootROM is instructed with a magic number in a dedicated register to use it [5]. Hence, the (first-stage) bootloader is the first part, which is placed in volatile memory before a warm reboot.

However, this bootloader is limited to 64 KB due to the OCRM size. Therefore a secondary bootloader can be placed in the RAM or the existing one on non-volatile media is used. The second-stage bootloader is then loaded and executed by the first stage, and performs the FPGA reconfiguration either from an image stored in the RAM as well, or from a non-volatile memory. Subsequently, the usual boot process can continue.

Summarising the above, the full FPGA reconfiguration needs a first-stage bootloader in internal OCRM, and a second-stage bootloader in the main memory. The FPGA image can be either placed there as well or in a non-volatile memory. As the whole flow logic, i.e. the first boot stage, is only present in volatile memory, it is possible at any time to revert back to the non-volatile default configuration with a processing platform power-cycle.

Furthermore, the reconfiguration of the FPGA and the specific time when it happens in the boot process became even more important for recovering one of the processing platforms. Due to the size of the FPGA configuration, the above strategy relies on the availability of RAM. But in the case described in Chapter 4, RAM is only available once the FPGA is configured. Therefore, the configuration has to be performed in the first bootloader phase to allow loading the second one. However, this requires more effort due to the limited memory. In consequence, the storage format for the image had to be simplified. This was achieved by storing it in the first hardware boot partition of the non-volatile eMMC memory at a fixed location, as depicted



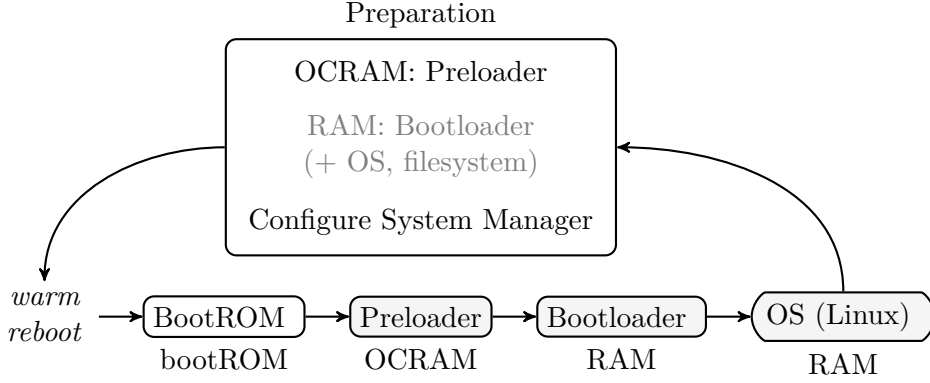


Figure 3.8.: General boot flow for a warm boot from OCRAM. It is similar to the general boot flow, but requires a preparation step. This instructs the bootROM to use the bootloader image preloaded to OCRAM as the next boot stage on a warm boot. The mechanism enables a volatile boot configuration change.

in Figure 3.9. The SPL can load it from there, and the entire user hardware partition can be left unmodified. This approach is further detailed in Section 4.1. [37]

However, also FPGA experiments should be executed on the platform and these need to change the configuration image. This was achieved by storing the FPGA image in the second hardware boot partition, using an SPL configured to use it, instead of the first one. Since the alternative FPGA configuration is based on changes in non-volatile memory only, the original boot process is thus preserved after a cold reset.

In conclusion, the described strategy allows a safe (re-)configuration of the FPGA fabric. Thus, the mission’s objective of a (routine) reconfiguration capability is possible. Also the critical parts of the reconfiguration, like the bridge and pin settings, were addressed and considered for the present implementation of the boot strategy. However, the IO and bridge configuration of a particular FPGA design have to be checked manually, i.e. by design inspection or the compiler output. A more automated and precise way to check the proper FPGA ‘environment’ could be a binary inspection of the

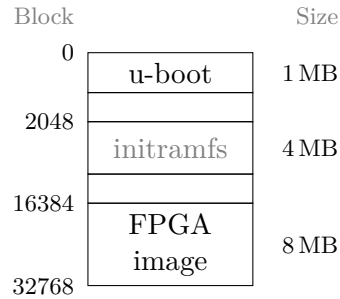


Figure 3.9.: Hardware boot partition layout on eMMC after SDRAM recovery. The bootloader and an FPGA image are stored in this separate area of memory, while leaving the hardware user partition as is. Subsequently, an initial 'ramfs' for allowing a minimal system boot was added.

programming files, if the format would be known.

A potentially unknown FPGA configuration also implies a risk to the spacecraft on system level like high power consumption, and in consequence a higher temperature. Thereby, also timing issues in the fabric can arise. These might also lead to an unpredictable behaviour of the design, and potentially of the controlled peripherals.

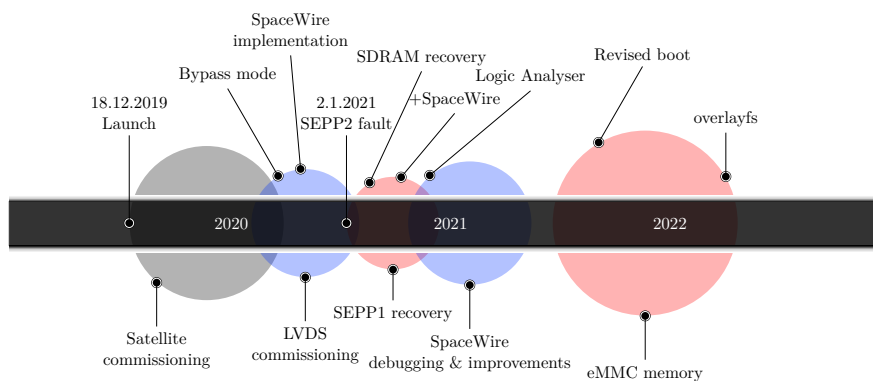
For these reasons, careful testing on ground is a prerequisite. An engineering model on the ground, kept in the same configuration as the spacecraft, is always used as a test platform for experiments before flight. By monitoring the resources consumed, such as power consumption, the behaviour can at least be estimated. However, the actual behaviour is unknown a priori, but determining it is the central aim of conducting experiments.

### 3.5. Brief History of OPS-SAT Operations

Unexpected behaviours and failures in OPS-SAT were analysed and, at least to some extent, taken into account in the early design phases and operational procedures. Hardware, software and operations were designed to ensure the safety of the spacecraft even under unfavourable conditions. However, as is often the case, reality caught up with these preparations, as the historical lifetime of OPS-SAT shows.

After OPS-SAT's launch on 18/12/2019, its commissioning phase started. The radio communication was established and step by step all the payloads were tested for their correct function. However, the bypass mode of the CCSDS Engine, directly forwarding the radio interfaces to the processing platform, required special treatment as the FPGA had to be included in the test. Therefore, an approach tailored to the ground and space requirements was implemented and executed on 04/09/2020. Chapter 6 discusses this in more detail. With the end of the commissioning phase, the execution of software experiments started.

It soon became clear, that the default communication over the CAN bus was insufficient to handle the high data loads associated with experimental usage. As an alternative communication channel, a SpaceWire interface was implemented and tested in-flight on the FPGA portion of the SoC, and is described in Chapter 7.



In early 2021 a severe failure occurred on one of the processing platforms: It could not successfully boot up anymore. To make things worse, the redundant unit was only partially operational. It suffered from a defect discovered during late ground testing. The approach for recovering this partially operational unit in-orbit is described in Chapter 4. After extensive testing, it could finally be recovered on 24/02/2021 and almost the full functionality of the spacecraft was restored. However, this recovery had side effects invalidating the SpaceWire implementation. Hence the SpaceWire design was reworked and, additionally, one month later a software extension was implemented to allow IP-based communication from ground with the processing platform.

During the summer of 2021, the SpaceWire link became increasingly unstable, with long periods of downtime. Actions were started to investigate the cause, but no clear correlations to external or internal events could be found. To gain more insight on the internal signals and the actual error behaviour, a special measurement circuitry, a logic analyser, was implemented on the FPGA. Its design and implementation is described in Chapter 8.

In early 2022 errors on the filesystem of the processing platform were experienced. The first countermeasures were filesystem repairs and partial re-deployments. However, the issues recurred and the performance degraded. Therefore, several concurrent strategies were followed. One approach was the avoidance of write accesses to the eMMC. Another strategy was to revive the alternative NOR flash as it was not up to date with the processing platform recovery measures carried out in 2021. Chapter 5 explains the changed boot strategy and the necessary changes.

The versatility of the satellite platform and the integrated reconfigurable FPGA proved essential tools in each of these experienced recoveries and enhancements. The capabilities that have already been utilised range from advanced diagnostics to disaster recovery. However, this could represent only a small fraction of the cases in which these new techniques could prove useful in dealing with unforeseen use cases on already implemented hardware.

The following chapters deliberately present these approaches in thematic rather than chronological order. This is done in order to shift the focus away from operational reality to the application examples that demonstrate the possibilities of reconfigurable hardware on a spacecraft.

## CHAPTER 4

### Recovering SDRAM

After OPS-SAT had been in orbit for more than a year, one of its main processing boards failed. An autonomous power supply reset occurred in early 2021, and after recovery, the processing platform could not boot up anymore. Investigations were therefore initiated to determine the cause and possible mitigative measures.

In the first attempt, the processing platform was turned on as usual. As soon as it was powered, the radio downlink was showing an unexpected signal and no data could be decoded. This problem could be traced down to the CCSDS Engine accepting a mode switch, i.e. bypass mode, on its dedicated pins getting probably misoperated by the processing platform. Therefore, for the next try, the CCSDS Engine was locked to its current mode with an override from ground.

After locking the CCSDS Engine mode, the processing platform was switched on again. The radio signal was stable this time, but the telemetry stopped completely, and the packets appeared to be queued up. It is possible that the CAN bus was occupied by the processing platform, as the queued telemetry packets were downlinked in a burst after turning off the platform. To resolve this issue, the corresponding data bus switch was configured to isolate the platform from the CAN bus.

As the CAN bus is the primary communication interface, it was then also not possible to communicate with the processing platform. However, it allowed the ground to power it on without affecting the spacecraft operation. In addition, power supply telemetry could be queried and collected. Monitoring

#### Chapter 4. Recovering SDRAM

Temp. °C	Input V	1.1 V mW	1.35 V mW	2.5 V mW	3.3 V mW
13.02	5.0	30.15	8.27	348.02	
13.06	5.0	30.27	9.96	348.79	326.91
13.05	5.0	29.81	9.59	348.49	324.56
12.98				347.36	328.75

Table 4.1.: Power consumption of the failing processing platform for the individual HPS supply rails 10.368 s after power-up.

Temp. °C	Input V	1.1 V mW	1.35 V mW	2.5 V mW	3.3 V mW
	5.0	29.89			
13.72	5.0	29.97	9.82	352.59	318.59
13.67	5.0	30.80	8.77	346.35	325.03
13.69	5.0	30.08	8.41	349.52	323.65

Table 4.2.: Power consumption of the failing processing platform for the individual HPS supply rails 139.1624 s after power-up.

the supplied power provided more information on the current state of the platform.

Initially, only the coarse-grained telemetry of the power system and power bus switches was available to monitor the boot process. However, during the boot phase the currents are changing quite rapidly. For a faster acquisition also the power manager telemetry was read directly via I<sup>2</sup>C and the values are shown in Tables 4.1 and 4.2. The power manager allows monitoring of all the individual supply voltages and currents, together with any occurring faults like over- or undervoltage conditions. In addition, it cyclically measures all internal supply voltages and currents with a 160 ms interval. Unfortunately, the high resolution mode of the odd channels prevents the acquisition of current statistics. The voltage statistics were acquired and showed no spikes or short-circuits on the supplies.

For comparison, a similar measurement was made with the identical spare unit on ground. Power consumption was monitored for the early boot phase, i.e. the first 3 s after the reset line is released. This is shown in Figure 4.1.

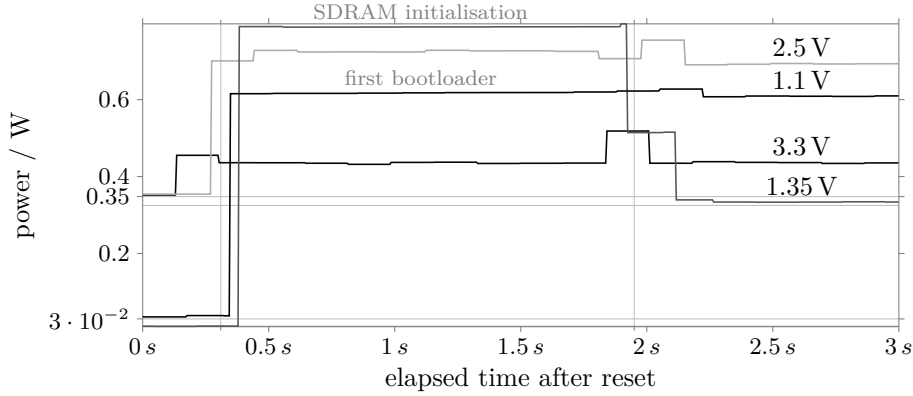


Figure 4.1.: Power consumption for the HPS supply rails of a spare processing platform in the early boot phase. The probable hand-over points between the three boot stages are marked with vertical grid lines. The horizontal grid lines correspond to the observed power levels of the processing platform’s fault state, derived from Tables 4.1 and 4.2.

The different power states can be assigned to the boot phases: Firstly, the bootROM configures the eMMC, then searches and loads the preloader stage from it. This is visible in the slightly, due to the access mode, increased current on the 3.3 V rail supplying it. Before handing over to the next boot stage, the Phase-Locked Loops (PLLs) are configured drawing more current from the 2.5 V supply. Secondly, the preloader takes over control and reconfigures the CPU, leading to the rise of the 1.1 V rail, and starts the SDRAM initialisation. The SDRAM is supplied with 1.35 V. After bringing it up, the next boot stage is searched for and loaded from the eMMC, which is again accompanied by a rising current on the 3.3 V supply. Thirdly, control is handed over to the next bootloader stage, which was just loaded. This stage waits for 2 s before it continues, resulting in a nearly constant power consumption until the end of the monitored window.

When comparing the observed currents on the spacecraft and the expected boot flow on the spare unit, the flow seems to be interrupted at a very early point. Figure 4.1 has additional grid lines at 350 mW and 3 mW to ease the comparison. The power levels of the supply rails matches rather precisely the

## *Chapter 4. Recovering SDRAM*

state before the bootROM hands over control to the next boot stage. Therefore, the cause of the processing platform failure could be related to a faulty eMMC memory. However, switching to the redundant QSPI-connected flash memory could eliminate this possibility, as the same behaviour could be observed.

In summary, the failure is definitely occurring before the SDRAM or the CPU are fully initialised, which would result in a constantly higher power consumption on the 1.35 V or 1.1 V rails. The definite cause can not be determined precisely, but the problem seems to happen in the bootROM stage or before it. Unfortunately, this phase cannot be influenced any further than by switching between the two non-volatile memories. The observed issues on the CAN bus suggest the presence of low-impedance pins that pull down the lines and force a dominant state on the CAN bus. This state occurs, for example, from power-up until the correct pin configuration is applied and the internal CAN controller is released from reset.

Consequently, the second processing platform was considered to be unusable and other possibilities for upcoming operations had to be considered. The obvious next choice was the redundant processing platform unit, but it had a known defect: Its SDRAM was failing randomly.

### 4.1. The Broken SDRAM

During checkout, the redundant processing platform showed instabilities in connection with the SDRAM. The behaviour was very unpredictable as it failed sometimes during runtime after several minutes to hours, but also in the early boot phase when the SDRAM is initialised. When failing in this early phase, the bootloader is unable, due to the uninitialised SDRAM, to load the second stage and continue the boot process. The failing SDRAM initialisation could be reproduced on ground with a disconnection of one of the address lines of one SDRAM DDR3 chip.

Due to this unstable behaviour also this redundant processing platform could not be considered as operational. However, as the FPGA part of the Cyclone V also has SDRAM (one 512 MB chip) connected to it, this could be used by



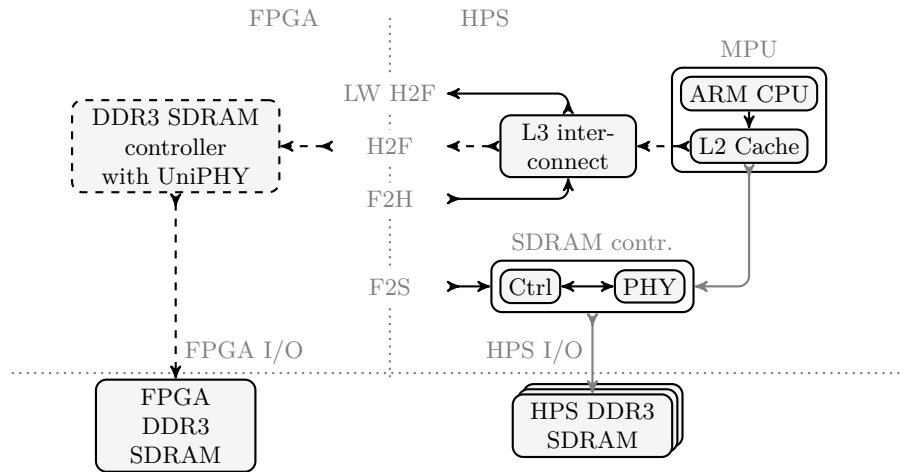


Figure 4.2.: Schematic diagram of the nominal and the rewired SDRAM structure loosely modelled after [34, p. 1-4]. The greyed path shows the nominal connection of the MPU to the HPS SDRAM. The dashed path is the newly implemented approach employing a soft-synthesised SDRAM controller. Therefore, the L2 cache is configured to use the L3 interconnect for all addresses. The L3 interconnect has a dedicated address range for the H2F bridge to access the synthesised SDRAM controller in the FPGA part.

the MPU in the HPS part as an alternative.<sup>1</sup> This setup is architecturally shown in Figure 4.2. In the MPU every memory access of one of the CPU cores is first handled by the L2 cache. The L2 cache has two ports: one is connected to the HPS SDRAM controller and the other one to the main L3 interconnect. It can be configured with a mask to direct certain accesses to the SDRAM port. For avoiding any (direct) access to the HPS SDRAM the mask has to be left unset, i.e. in reset state. Thus every memory access is handled by the L3 interconnect.

When seen from the MPU the next component on the route to the FPGA SDRAM is the L3 interconnect. It serves as the main switch for the memory

<sup>1</sup>This idea was originally proposed by members of the OPS-SAT Mission Control Team (MCT) at ESOC, i.e. Dominik Marszk and Vasundhara Shiradhonkar, and initial unfinished steps in this direction were implemented by them.

## *Chapter 4. Recovering SDRAM*

mappings of the integrated peripherals. Apart from switching OCRAM and bootROM into the visible memory space, which will be important later, it also handles access to the H2F bridge. This H2F bridge is where the soft-synthesised SDRAM controller in the FPGA portion exposes its (Avalon-MM) slave interface. Summarising, for the MPU to utilise the alternative SDRAM, it has to access the memory region assigned to the H2F bridge, the L2 cache has to be configured to route all accesses to the L3 interconnect, and these accesses have to be handled by a (soft-synthesised) SDRAM controller in the FPGA.

However, at the time the preloader needs to initialise the SDRAM for loading the next boot stage, the FPGA is not yet configured, as this is normally one of the tasks of this next stage. Consequently, neither the (soft-synthesised) SDRAM controller nor its associated memory are accessible. The preloader must therefore take over this task of configuring the FPGA. The revised boot strategy is depicted in Figure 4.3. Since the code of the preloader has to be quite size-optimised to fit into the OCRAM, the storage strategy had to be revised as well to facilitate a simple loading mechanism. Therefore, the FPGA image, the next bootloader stage and its environment are stored at fixed addresses in a raw hardware boot partition. The hardware partitions in eMMC are described in more detail in Section 3.3. This allows less complex code to load the boot data, while leaving the hardware user partition untouched.

In this scenario the preloader has only the OCRAM for buffering and program code available, but the FPGA image for this version of the Cyclone V is 6.7MB in size. Therefore, the FPGA image has to be loaded in chunks from the respective (hardware) boot partition of the eMMC memory and the FPGA configuration has to happen on-the-fly. For detecting read failures or image corruption a Cyclic Redundancy Check (CRC) is computed on-line during the load process. After the FPGA configuration the preloader activates the H2F bridge. With the now configured FPGA its SDRAM is initialised and accessible. These features were newly implemented in the U-Boot preloader code.

Then the usual boot flow continues with searching, loading and starting the second stage bootloader. The second stage configures and loads the

#### 4.1. The Broken SDRAM

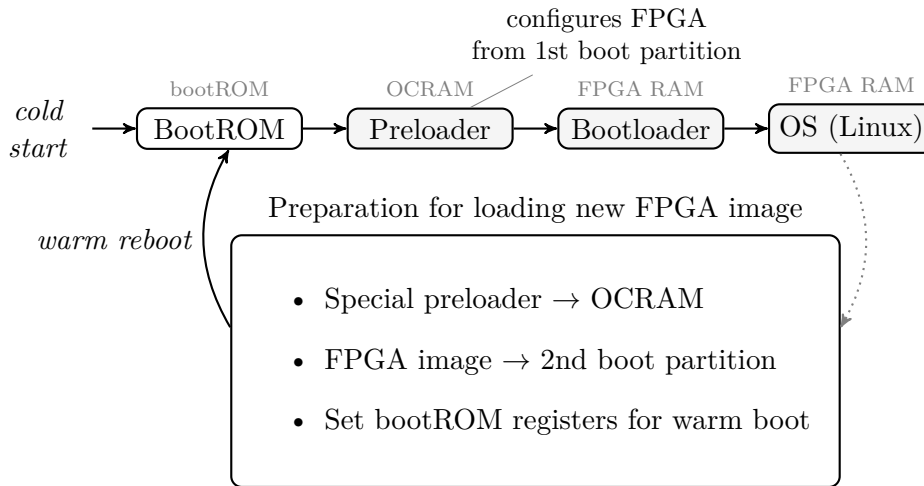


Figure 4.3.: After a cold start, the bootROM is mapped to address zero and executed by the ARM processor. The bootROM code loads the next boot stage (preloader) from non-volatile storage into OCRAM, and jumps there. The preloader initialises the main RAM to load the next stage there. In the present configuration, RAM is only accessible via the FPGA, and therefore the preloader configure the FPGA with an image from the first eMMC hardware boot partition. Then the second bootloader stage can be loaded and executed. For subsequent FPGA reconfigurations a dedicated preloader is loaded to OCRAM and the desired FPGA image is placed on the second eMMC boot partition. Dedicated registers instruct the bootROM to use the preloader in OCRAM after a warm boot.

## Chapter 4. Recovering SDRAM

necessary components for starting Linux. One particularly important configuration is the new SDRAM memory range starting at the H2F bridge address `0xc0000000`. This information is handed over via device tree to the Linux kernel.

Although this implementation mitigates the impact of the damaged SDRAM, this solution also has side effects. Due to the necessary use of SDRAM by the HPS, the FPGA configuration is effectively fixed. But as described in Section 3.1, FPGA experiments should also be performed. These require re-configuration. To make this possible in this scenario, the boot strategy mentioned above has been extended, as shown in the lower part of Figure 4.3. The desired FPGA image is stored in the *second* hardware boot partition. A preloader configured to use this image is loaded into the OCRM and the system manager registers are set to inform the bootROM that it is to be used on a warm reboot, similar to the warm boot strategy detailed in Section 3.3. A warm reboot is then initiated and the platform reboots with the new configuration. An additional requirement for any FPGA experiment is therefore to include the soft-synthesised SDRAM implementation in its design.

However, in spite of the new preloader features that have been implemented, the actual implementation of the new bootloader on an unstable system is a risk in itself. Nevertheless, the processing platform had to be booted in its original configuration and prepared for the update step by step, such as uploading the necessary images and utilities, while avoiding any stress to the system. Also fallback strategies, e.g. in case the new SDRAM could also be damaged, had to be considered and implemented. Therefore, the initial test was done with the warm boot approach described in Section 3.4. Only after the platform came up in the desired configuration and it was confirmed to be operational was the new bootloader persisted. Finally, on 24/02/2021 18:23:19 UTC the actual deployment was performed live in 7 seconds. After warm rebooting, the platform successfully came up in its new configuration roughly 2 minutes later. Despite this successful first boot, the Linux kernel had another issue, which was already discovered during the ground tests: Activating the second CPU core resulted in a system crash. The cause of this fault was identified as an access to an unmapped memory region. A suitable solution is described in the next section.

## 4.2. Reviving the Second Core

The initial implementation of the SDRAM rerouting prevented the activation of the second CPU core: As soon as the second core was activated, the processing platform eventually froze. Investigations showed a memory access violation of the second core. This was caused by the kernel's Symmetric Multiprocessing (SMP) bring-up code, which was designed for the usual memory mapping with the physical SDRAM being available starting at address zero.

The ARM CPU always starts fetching instructions from address zero as soon as it is released from reset. The L3 interconnect accommodates this with two mappings for address zero[34, p.191 5-7]: It can be switched to bootROM or to OCRAM. The default state is the bootROM setting which facilitates the boot media initialisation and loads the (first) bootloader stage from it. Since the start region of the SDRAM should also be accessible, the L2 cache (in the MPU) can be configured to access the HPS DDR3 SDRAM at address zero [34, pp.32 1-16].

The initial parts of the boot sequence are only running on the first CPU core. The second core is usually activated by the kernel. First, the kernel places a trampoline code at the CPU start address (zero). Then the core is released from reset. The trampoline code directs the core to the `secondary_startup` code of the Linux kernel and integrates itself to the scheduling.

The first issue with the original kernel code was the calculation of the `secondary_startup` section, which did not take into account the changed (shifted) physical address mappings, due to the changed SDRAM location. Another pitfall is that address zero can have two mappings depending on the L3 interconnect configuration. Instead of querying the current state of the mapping, a more universal procedure was developed. This procedure and its code is further described in Appendix A.

A kernel with these changes was uploaded to the processing platform on 15/03/2021, which restored it to a fully operational state and allowed the mission to continue. The noticeable degradations were less available memory, and a slightly slower access speed. In practice, the speed penalty was negligible, as the majority of accesses hits the L2 cache.

### 4.3. Conclusion

This case demonstrates the potential provided by a reconfigurable FPGA for recovering from a disaster scenario. One broken and one processing platform without a functioning SDRAM would have meant a severely limited experimental mission, but the reconfiguration capabilities were able to restore almost the full functionality. A critical prerequisite for the presented approach is a good connectivity of the FPGA, especially regarding the internal bridges to the HPS and to the outside, i.e. the redundant SDRAM. Another mitigation concept used is redundancy: the availability of a redundant SDRAM is another key element for the implementation.

Even though the platform could be brought back to operations, there are also some downsides. Due to the smaller size of the FPGA-attached SDRAM the ECC functionality was left disabled. Otherwise, the available memory would have been halved once more. There was also an impact on the upcoming FPGA experiments, as they now have to incorporate the SDRAM controller functionality in the design. One possible solution to simplify the reconfiguration strategy, while keeping a partially fixed configuration in place is a partial reconfiguration of the FPGA fabric. However, this function is not fully supported by the used version of the Cyclone V.

Also, existing functionality implemented in the FPGA had to be made compatible to the changed configuration. Affected components are the SDR and the SpaceWire cores. The latter one is expounded in Chapter 7, regarding its initial implementation and the necessary changes after the SDRAM swap.

Unfortunately, the successfully mitigated SDRAM failure was not the last defect in the mission. Later, the performance of the eMMC on the processing platform rapidly degraded. As a countermeasure the bootloader and the boot strategy were reworked, and thereby the above-mentioned FPGA configuration in the SPL incorporated as a default mechanism. This is described in the following chapter.

## Boot Strategy Revisited

The initial boot strategy in Section 3.3 evolved over the course of the mission. Especially, after recovering from the SDRAM issues of the first processing platform, the boot flow was effectively invalidated. The resulting boot strategy is described in detail at the end of Section 4.1. Another event causing a change of the boot strategy was the failing eMMC memory. As a result, the redundant NOR flash had to be updated, harmonising the two non-volatile boot media, and thereby improving the boot flow in terms of redundancy.

The whole boot flow relies heavily on boot data being available in non-volatile memory. However, for OPS-SAT’s environmental conditions and its partly non-radiation hardened electronic components this might not prove as reliable as expected as flash memories are especially susceptible to radiation. Since the beginning of 2022 file system corruptions were experienced on the eMMC. Initially, these faults could be solved by applying simple repairs like rebuilding the file system. However, the degradations became more severe over time. Subsequent investigations showed, that the eMMC was entering a failure state and became unresponsive. Consequently, an approach was developed to deal with an unreliable memory operationally. This was achieved by reducing the need for write accesses through keeping the non-volatile memory write-protected and redirecting the write operations to a volatile filesystem. This is schematically depicted in Figure 5.1 and described in more detail in [39, 40].

Considering that the proper functioning of the eMMC memory was hanging by a thread, the bootloader was extended to support execution from the

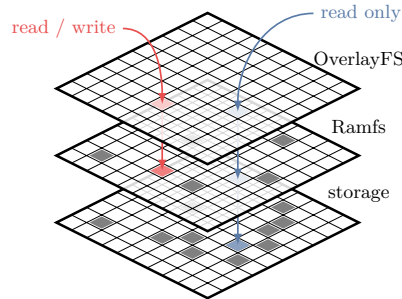


Figure 5.1.: Overlay filesystem (OverlayFS) schematic operation. A read-only access to a file targets the underlying non-volatile storage (storage layer). A write operation is forwarded transparently to the volatile ramfs filesystem (ramfs layer). Subsequent write or read operations to such changed or newly created files are solely handled by the overlayed volatile ramfs.

NOR flash and loading of the FPGA configuration from there as well. In addition, the layouts on the non-volatile boot media were harmonised to allow for a flawless interchange and thus adding redundancy for the boot data.

The new memory organisation for the NOR flash is depicted in Figure 5.2 and for the eMMC in Figure 5.3. When comparing these layouts, the eMMC hardware boot partition and the NOR flash have the exact same structure in the first 8 MB. At the start is four times an SPL followed by the full bootloader. Then the redundant bootloader environment (env, env2) and a boot image follows. The whole structure is aligned to the erase block sizes of both media, i.e. 64 KB and 4 MB (pSLC on eMMC). With this layout the bootloader environment or the bootloader image can be easily used from a boot stage loaded from another memory simply by switching the boot media. In addition, there is also a boot image containing the kernel, the platform DTB and a ramdisk, which will be described later.

The search strategy is determined by the boot select pins: the primary source is always the one selected by the pin configuration on boot and thus resembles the behaviour of the bootROM. If loading fails from this source the alternative one is used instead. In contrast to the previous implementation,



this allows to use the same bootloader on both memories, because the boot media selection is only influenced by an external setting.

In the first bootloader stage (SPL) the FPGA configuration has to be applied. Depending on the primary boot device it tries to locate it in NOR flash at two locations and in eMMC on both hardware boot partitions. After successful load of the configuration, which is protected with a CRC, the resulting CRC is stored in a system manager register. This allows to determine the currently used configuration. On the next run, i.e. a warm reboot, the CRC is used to locate an image with the same CRC. This mechanism can be reused to load a different image on warm reboot by modifying this register.

Finding and running a proper U-Boot is the next step of the SPL. It tries to find a U-Boot image with a correct checksum in all available memory locations, again with the same search strategy. Subsequently it loads it to RAM and hands over control to it.

The second stage (full) U-Boot will then run and load the environment variables from the primary boot device or, in case of an error, uses its built-in defaults. The environment is set-up to also follow the same prioritisation and tries to continue the boot process from the primary boot device and fallback to the secondary boot device, or a ramdisk if necessary.

The boot image loaded by the bootloader is a Flat Image Tree (FIT) containing a Linux kernel, a platform-specific DTB and a ramdisk, i.e. initramfs. The whole image integrity is assured with hashes and very compressed, as it occupies only 6 MB. The kernel, the DTB and a valid root filesystem allow the system to boot into Linux. However, in case the search for a valid root filesystem fails, the system will boot with the initramfs. The initramfs provides a fully volatile filesystem located in RAM with a bunch of utilities. Among them are all the tools necessary to allow communication with the ground, as well as for on-board diagnosis of the CAN and I<sup>2</sup>C bus. It also provides tools for filesystem check, creation and recovery. In addition, there is also the possibility to chain boot into a root filesystem on non-volatile media, with an automatic filesystem check in advance. Thereby, it is possible to use it in a default boot scenario, providing an additional safety check with fallback.

## Chapter 5. Boot Strategy Revisited

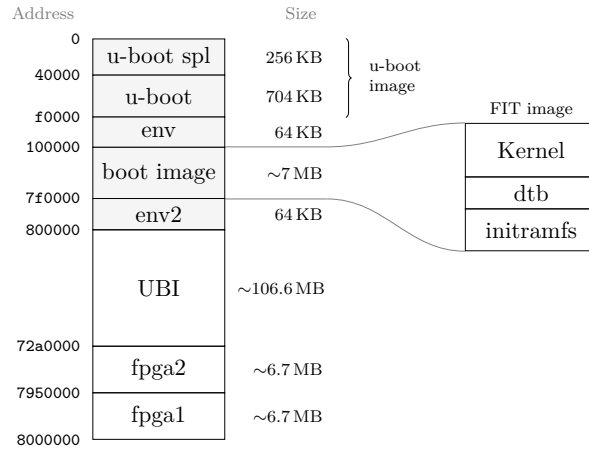


Figure 5.2.: Revised partitioning layout on QSPI-attached NOR flash. The layout is aligned to the erase block size of the NOR flash (64 KB). and the usable space was maximised, considering the case that the eMMC fails. The u-boot storage area is unsegmented in the first 1 MB. The redundant storage of the bootloader environment is located in different erase blocks. The heavily compressed boot image area contains in a FIT image the Kernel, its DTB, and an initramfs. The UBI area contains the file system and enables dynamic partitioning. The redundant FPGA image areas at the end are optimised for size (each image requires 64 B header + 7 007 204 B data in size).

In summary, the boot process could be improved to be more robust. An additional fallback option in the form of a ramdisk has been implemented. Furthermore, the modified boot data structure enables a smooth transition to backup storage areas during boot.

eMMC hardware boot partition			
Address	Block		Size
0	0	u-boot spl	256 KB
40000	512	u-boot	704 KB
f0000	1920	env	64 KB
100000	2048	boot image	7 MB
7f0000	16256	env2	64 KB
800000	16384	fpga1	8 MB
1000000	32768		

Figure 5.3.: Revised eMMC partitioning layout utilising the hardware boot partition. The block size is 512 B. Compared to Figure 5.2, the layout of the beginning of the eMMC hardware boot partition is identical. This allows to use the same bootloader binary for either memory. In addition, it is also aligned to the erase block size of the eMMC memory (4 MB in the boot partition).



# CHAPTER 6

## Testing Bypass Mode

Numerous test campaigns are conducted to ensure that all components, up to the fully assembled spacecraft, function as intended. These tests are particularly important for the communication interfaces, as they establish the connection to the ground. Comparable tests are carried out on the ground to ensure a fully operational connection to the satellite.

The CCSDS is standardising such space communications systems and these standards are also applied to OPS-SAT's S-band and X-band chain. The switching between these radios, the basic CCSDS-compliant operation and the satellite bus connection is handled by the CCSDS Engine protocol translator device.

The CCSDS Engine, by default, facilitates data transfer between the S-band radio and the CAN bus while adhering to CCSDS standards. To conduct communication related experiments beyond the standardised protocols, as per the mission requirements, the CCSDS Engine can be configured to bypass this protocol conversion. In this "bypass mode", the LVDS signals from the processing platform are translated to and from the respective electrical interface of the radio [55]. This allows for the implementation of any protocols from the coding layer upwards. However, this direct radio access also shifts radio commissioning effort to the testing mechanism or equipment.

In summary, the CCSDS Engine provides different data interfaces and a selection mechanism between radio subsystems. Particularly its bypass mode imposes additional challenges for efficient and effective testing. Dedicated test procedures have been developed to efficiently test this function, and

thereby the involved electrical interfaces. Before the launch, this was necessary for the system-level X-band radio test campaign, as presented in the next section. During the in-orbit commissioning, the bypass mode could be utilised to test the entire communication chain, including the ground segment, in accordance with the relevant CCSDS standards. This is presented in Section 6.2, and the procedures used, and their results, were published<sup>1</sup>. The present chapter contains material from this publication [38], which was adapted and extended where necessary, without explicitly citing it.

## 6.1. Pre-Launch Testing

For the system level radio test campaign of the X-band transmitter a dedicated test setup had to be implemented. As the X-band transmitter's maximum downlink capacity is 50 Mbit/s and the CCSDS Engine only supports rates of up to 24 Mbit/s, the data had to be generated at the maximum rate in a different way. Moreover, the system level test requires to generate the data on-board, because most of the internal connections are not accessible anymore after integration. Here, the FPGA on the processing platform was utilised to fill this gap.

There are a number of ways for testing the CCSDS Engine's LVDS port using an FPGA. Figure 6.1 shows examples of some of these possible test variants. For example, the generator side can range from low-level data generators to full (SpaceWire) protocol implementations. On the receiver side, in addition to the actual radios, the Electrical Ground Support Equipment (EGSE) can be used for verification, as well as other measurement equipment such as an oscilloscope.

Pseudo-random sequences are a frequently used method for testing communication equipment. The (pseudo-)randomness of the sequence allows to characterise the spectral properties of a transmitter, but at the same time provides a predictable sequence which allows to detect single bit errors. Therefore, the FPGA implementation uses the Avalon-ST Data Pattern Generator [26, Sec. 38.2] IP core, which allows to generate different types of test

---

<sup>1</sup>© 2022 IEEE. Reprinted, with permission, from [38]

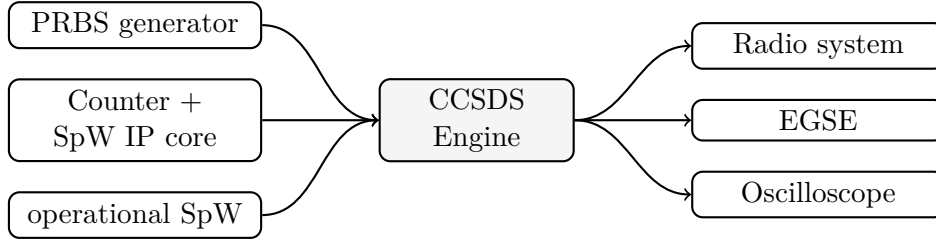


Figure 6.1.: Examples for test scenarios for ground and space application utilising the LVDS connection of the CCSDS Engine. The LVDS port of the CCSDS Engine can be used in bypass mode, e.g. with a Pseudo-Random Binary Sequence (PRBS) generator. The nominal SpaceWire mode of the LVDS link can be used with a conformance tester or similar hardware, including an IP core implemented on an FPGA platform. This IP core can be combined with a data, i.e. packet, generator. The output of the CCSDS Engine can either go directly to one of the radio interfaces or to its EGSE. For debugging purposes the output can also be measured, e.g. with an oscilloscope or logic analyser.

patterns, including PRBS sequences. The core can be activated on demand and the sequence is selectable at runtime.

In this test application, it is also important to be able to select different data rates. The test pattern generator is therefore clocked by a reconfigurable PLL [3]. This allows an adjustable output rate during operation. The feasible clock rates are limited by the timing constraints in the FPGA design. However, the targeted clock rate of 50 MHz can easily be achieved on the platform.

The radio tests were carried out in the anechoic chamber of the Institute of Microwave and Photonic Engineering at Graz University of Technology. This approach facilitates accurate measurements and avoids reflections. The measurement setup is depicted in Figure 6.2. For the test, the CCSDS Engine was switched to bypass mode and the FPGA was configured to generate a PRBS-23 sequence on its LVDS output lines at 50 Mbit/s. When the X-band radio is activated and detects a valid input clock signal, it starts modulating the input data as CCSDS-compliant OQPSK. The eventually

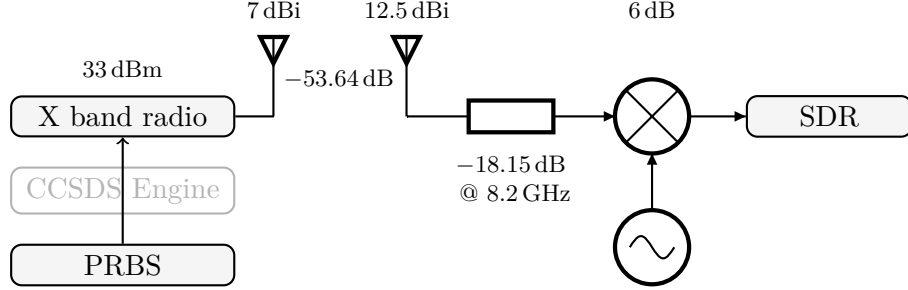


Figure 6.2.: Schematic setup for the X-band radio test in the anechoic chamber loosely modelled after [50]. A PRBS is sent by the FPGA on the processing platform to the CCSDS Engine in bypass mode. It forwards the unaltered stream to the X-band transmitter. The emitted X-band radio signal is received with a horn antenna, down-converted to an intermediate frequency at 3 GHz and recorded with an SDR.

emitted signal is captured with a horn antenna at the centre of the X-band antenna main lobe. The received signal is down-converted from 8.1775 GHz to a 3 GHz intermediate frequency. This intermediate frequency signal was recorded with an Ettus Research USRP X310.

The frequency spectrum of the recorded samples is plotted in Figure 6.3. The attenuation in the frequencies off the center are due to the filtering effect of the 160 MHz bandwidth limit of the SDR frontend. However, the visible part of the spectrum conforms to the Space Frequency Coordination Group (SFCG) recommendation for space-to-Earth links [24]. According to this recommendation the effective symbol rate is 100 MSym/s, because it is defined as the "baseband single line bit rate following error correcting coding" [24, p.2]. Although, the recorded data allows for further analysis, the anticipated outcome of the test focused on the analysis and conformance check of the spectral properties of the emitted signal. This could only be accomplished with the fully assembled spacecraft due to the availability of a reconfigurable FPGA on-board.

FPGAs are already widely used in various kinds of test applications. The test case presented shows that their integration into the device under test,



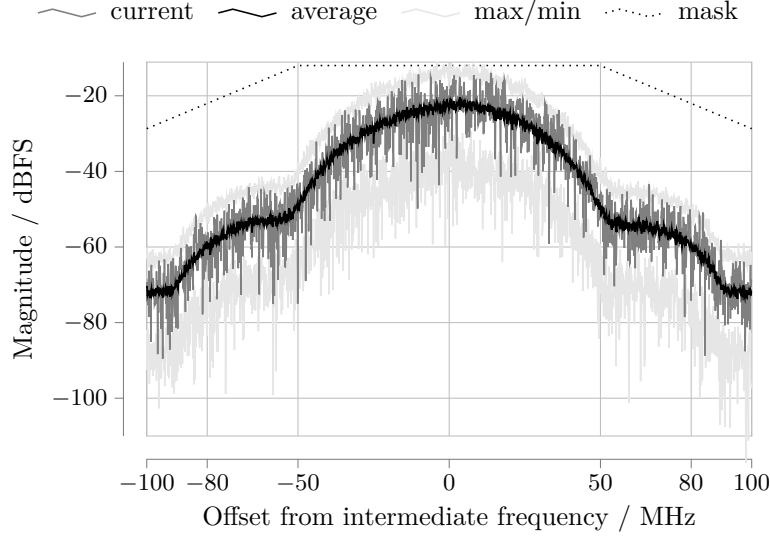


Figure 6.3.: Power spectral density with a resolution of 2048 points of the X-band transmitter emitting an OQPSK-modulated stream with 50 MSym/s. It was computed from the samples received with the USRP X310 SDR frontend. The emitted signal at 8.1775 GHz centre frequency was down-converted to an intermediate frequency at 3 GHz. The bandwidth limit of 160 MHz of the SDR frontend can be seen in the attenuation beyond  $\pm 80$  MHz. The spectrum conforms to the indicated spectrum mask as per the SFCG recommendation [24] for space-to-Earth links (symbol rate is 100 MSym/s according to this recommendation).

in a broader sense, expands the test possibilities immensely. Furthermore, this specific test scenario was not considered at the time of hardware design, demonstrating the operational flexibility of the fully assembled system. Exploiting this reconfiguration capability was also the basis for the following in-orbit test scenario.

## 6.2. In-Orbit Testing

The test approach used in the commissioning phase is based on the idea of a loop starting at the ground segment and covering all components of the communication chain up to the processing platform as depicted in Figure 6.4. This verifies the functionality of the entire chain, including the ground segment, in one step. The looped path starts at the ground modem, which modulates and emits a known binary sequence to the radio on the satellite. On the satellite the radio receives and demodulates the incoming data stream and sends it to the CCSDS Engine. However, neither the CCSDS Engine nor the radio modem have a loop-back functionality, so the path must be closed on the processing platform. Therefore, the CCSDS Engine is set to bypass mode and routes the radio lines directly to the processing platform. The processing platform closes the loop and connects the receive and transmit path, and thus the sent sequence arrives back at the ground segment. The design and results of this approach were published in 2022 [38] and the following sections contain content of that publication without citing it explicitly. The components involved in this loop and their individual settings are further discussed in the following paragraphs.

The primary **ground modem** for OPS-SAT is a Safran (formerly Zodiac) Data Systems Cortex-XL [57]. Here, it was set to a dedicated test mode. In this mode it transmits a predefined binary pattern repeatedly at a fixed uplink rate of 256 kbit/s. The transmitted pattern is shown on the right side of Listing 6.1. At the same time its downlink data stream operates at the same rate. The downlink stream is forwarded as usual to the ground data systems.

The **radio modem** is the first component to receive the emitted test pattern on the satellite. The S-band link can operate with downlink rates from

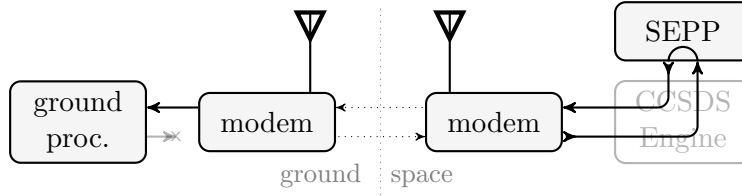


Figure 6.4.: Schematic loop connection over the ground and space segment up to the processing platform (SEPP). Disconnected components like uplink data from the ground data system or the bypassed CCSDS Engine are faintly visible in the background.

10 kbit/s to 1 Mbit/s. In uplink direction it is fixed to 256 kbit/s [31]. Therefore, uplink and downlink can operate at the same rate, which is the desired configuration for the anticipated loop-back scenario.

The **CCSDS Engine** processes the raw stream from the radio front-end. For the loop-back test the CCSDS Engine is switched to bypass mode with dedicated input pins on the CCSDS Engine. These pins are directly accessible on the processing platform and allow to change the link behaviour at run-time. In addition, the CCSDS Engine exposes the current mode of operation on output pins. Instead of transmitting according to the CCSDS Telecommand (TC) standard [70], as would be usual for the uplink, Telemetry (TM) [71] was defined as the uplinked binary sequence. This facilitates the verification of the received data with the unmodified receive part of the ground segment.

The predefined binary **test pattern** used was a pseudo-randomised TM frame for virtual channel 4 with an appropriate header and valid checksum as illustrated in Figure 6.5. It is prefixed with an ASM to form a valid CADU. The resulting binary pattern in its transmitted (right) and de-pseudo-randomised (left) form is listed in Listing 6.1. The frame contains three space packets (SPP) with distinct Application Process Identifiers (APIDs). Here, APID 256, 257 and 258, respectively, were chosen arbitrarily. The ground segment accepts the CADU when it arrives at the receiving end. However, if the satellite is not in radio bypass mode, the incoming TM frame would get rejected on-board, because a TC frame is expected.

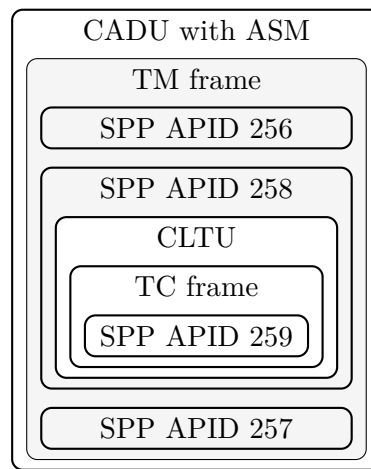


Figure 6.5.: Diagram of the protocol encapsulation in the Channel Access Data Unit (CADU). The CADU with the Attached Sync Marker (ASM) contains the pseudo-randomised TM frame. The de-pseudo-randomised Communications Link Transmission Unit (CLTU) is visible in white.

## 6.2. In-Orbit Testing

Listing 6.1: Derandomised TM frame on the left as it is visible after pre-processing on ground. The *TM frame header* and the *checksum* at the end are emphasised. Immediately after the header the first space packet follows. The space packet containing the **CLTU** starts at position 0x1f9. The ground modem repeatedly transmits this frame as a CADU as visible on the right hand side. After the *ASM* at the start, the pseudo-randomised TM frame begins. The contained **CLTU** at position 0x203, here visible in its clear form, is highlighted.

```

000      32a8000080000300000011      000 001acffc1dcde00ec01a0d73bc8e2c82
010 00c00001e348656c6c6f2120596f7520      ...
020 666f756e64206d657373616765202331
030 21000000000000000000000000000000
040 00000000000000000000000000000000
    *
1f0 00000000000000091141102c000017b00
200 00000014d82dea9b34707d9f4e906da7      200 66f558eb90232a013900c1116203c000
210 b66d863f3711a05d839f67654c84a8f2      210 012b4865a06c6c6f2120596f5c752066
220 a29fdf9ad23dec516992187bfd427b6c      220 6f756e6442206d6573736167a4652023
230 5fac9cb5b8fb9865457e7c14dfe31129      230 31210000960000000000000000fe000000
240 9bd563fdde3b026835c2f2384c4eb69e      240 00000000fe00000000000000fe000000
    *
360 aac7fa408804d0a40e6c6331f5a8f87f      360 00000000fe0000cf8b88125568c5c5c5
370 f3b7117e97000000006df41101c00000      370 c5c5c5c579619515f99da4968d84a66f
380 dc48656c6c6f2120596f7520666f756e      ...
390 64206d65737361676520233121000000
3a0 00000000000000000000000000000000
    *
450 000000000000000000000000b2c4eb64      450 396a5df730ca8afc82843c69097dcce

```

Therefore, the binary pattern was extended to accommodate for the non-bypass mode case, i.e. nominal setting, and for additional test coverage. The second space packet contains in its payload data section a pseudo-randomised sequence. As pseudo-randomisation is an XOR-operation with the generator sequence, it gets reverted by applying the same sequence. When applying the pattern to the TC sequence at this specific position, it will get effectively de-randomised and appears in its clear form on the right side of Listing 6.1. Although this approach circumvents the purpose of pseudo-randomisation, it lets the nominal communication chain on the satellite extract the TC frame inside the CLTU and then process the contained space packet with APID 259.

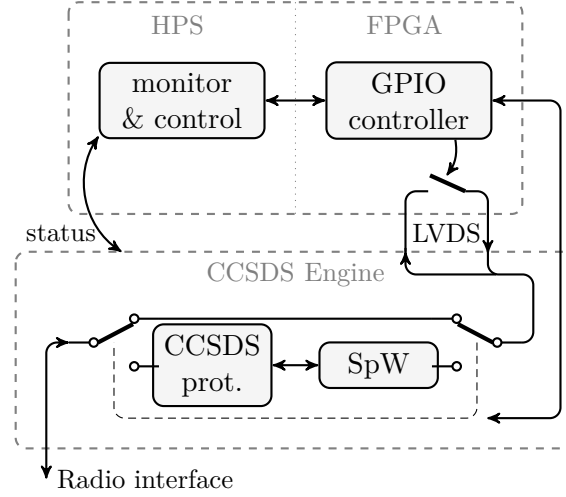


Figure 6.6.: Simplified schematic diagram of the configuration of the processing platform (top) and its interaction with the CCSDS Engine. The loop-back setting and the CCSDS Engine bypass mode are enabled and monitored with a dedicated control program on the processing platform. The nominal chain of the CCSDS Engine and the SpaceWire interface is also visible.

The APID 259 of the space packet was intentionally chosen to route it to the SpaceWire / LVDS interface of the processing platform. In the present configuration the physical LVDS interface of the SpaceWire link was looped back. Thus, any space packet sent to the SpaceWire interface in the forward direction arrived back in the downlink buffer and was then transmitted back to ground.

Taking both processing variants together, the test pattern can be processed regardless of the eventual mode of the CCSDS Engine. Furthermore, the CCSDS Engine outputs in both cases valid TM frames and allowed the ground data systems to process them.

The **processing platform** monitors and controls these dedicated pins of the CCSDS Engine for state control using a custom control program running on the ARM processor, a part of the Cyclone V. This setup is depicted in Figure 6.6. A synthesized Parallel Input/Output (PIO) controller IP core is

### 6.3. Control and Monitoring

used to access the pins connected to the FPGA portion [26, sec. 27]. The PIO controller is connected to the mode selection pins and a (synthesised) switch that activates the loop-back connection. This allows control of the loop-back connection from the processing platform and monitoring of the current status. In addition to pin monitoring, the control program polls the status of the CCSDS Engine via a separate bus. The radio status is also visible in this polled data. However, this entire approach requires in-flight reconfiguration of the FPGA. Since the test happens autonomously on the satellite, the reconfiguration has to be safe and revertible at any time. Therefore, the reconfiguration strategy described in Section 3.4 was employed.

### 6.3. Control and Monitoring

The first task for the control program is to use the method described in Section 3.4 to reconfigure the FPGA. In the initial run it checks to see if the FPGA is in the expected configuration. If not, it will prepare the reconfiguration and performs the warm reboot approach to put the FPGA into the desired state.

After the reboot the actual commissioning test can be executed. When the control program runs and detects the anticipated FPGA configuration it monitors the bitlock state of the radio via the CCSDS Engine. After 12 s, or when bitlock is acquired, it switches the CCSDS Engine to bypass mode and enables the loop-back configuration for 12 s. This configuration simply mirrors the incoming stream back to the ground. The pattern sent from the ground station will then appear identically at its receiver.

Starting again from the default configuration, it waits for 12 s for bitlock. When bitlock occurs it just enables LVDS loop-back connection for 12 s, now without bypass mode selected. The normal telecommand processing of the CCSDS Engine takes place and a valid space packet is forwarded to its respective interface. If a space packet for the SpaceWire interface is received it will be sent as usual. This is possible because the SpaceWire interface can synchronise with itself over the (LVDS) loop connection. The outgoing

Table 6.1.: Compiled data of the test performed on 08/10/2020

Start of run	Loop-back mode Time range	Nr.	Received frames Time range
05:02:33	05:02:46 - 05:02:58	0	
05:04:33	<b>05:04:35</b> - 05:04:48	334	05:04:37.024 - 05:04:48.756
05:05:33	<b>05:05:33</b> - 05:05:46	156	05:05:41.543 - 05:05:47.064
05:06:33	05:06:46 - 05:06:58	0	
05:08:33	<b>05:08:35</b> - 05:08:47	329	05:08:36.675 - 05:08:48.280
05:09:33	<b>05:09:35</b> - 05:09:47	312	05:09:37.274 - 05:09:48.253

The **highlighted** start times (in UTC) indicate the time of successful acquisition of bitlock by the radio front-end on the satellite. All runs produced a visible radio signal on ground. In case of no received frames the necessary signal level for successful reception could not be reached.

packet is thus mirrored back. It gets appropriately encapsulated as telemetry and is sent back to the ground station.

When the test run is finished, the captured data is collected and all previously applied settings are reverted to their original settings.

## 6.4. Results

After two pre-tests, a scheduled sequence of test runs took place in pass 4461 on 08/10/2020 as summarised in Table 6.1. The platform reconfiguration was started at 04:57:46 UTC. Four out of the six test runs were successful as the ground data systems could receive valid TM frames. In these runs the satellite's radio modem could acquire bitlock and the downlink signal was sufficiently strong for reception. The received TM frames and the contained space packets with APIDs 256, 258 and 257 showed that the bypass mode was successfully selected. Thus, the loop was correctly closed.

An additional analysis of the downlink signal power and its  $E_b/N_0$  on the ground modem showed level variations over all runs as depicted in Figure 6.7.



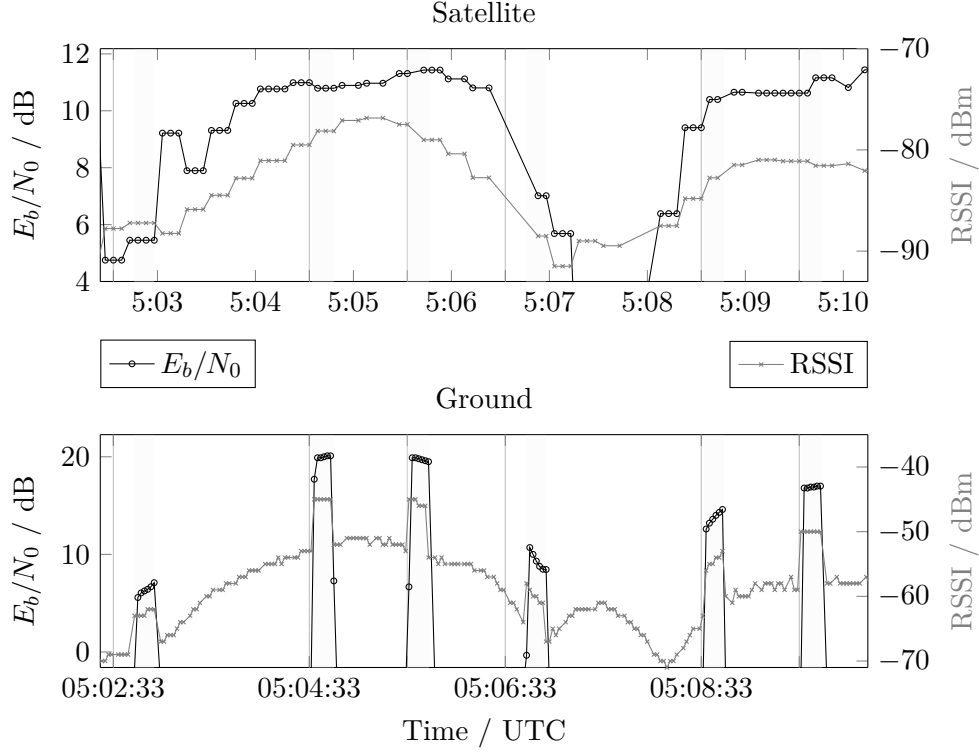


Figure 6.7.:  $E_b/N_0$  and Received Signal Strength Indicator (RSSI) on ground and on the satellite. The additional vertical tick lines denote the beginning of the test sequence on the satellite, where the control program starts to wait for acquisition of bitlock. When bitlock or a time-out occurs, the loop-back mode is activated. This is indicated by the shaded areas. In all six runs a visible radio signal could be observed on the ground modem (bottom graph). In the first and fourth run the downlink was enabled after time-out of 12s without bitlock on the uplink. This happened in both cases due to low uplink signal strength, as visible in the upper graph.

## *Chapter 6. Testing Bypass Mode*

For the two runs with no received frames (05:02, 05:06) the on-board monitoring by the control program revealed that the transmission was started without bitlock after 12 s. In these two runs the satellite's received signal level dropped below the minimum for successful demodulation. Taking this into account, together with the absence of bitlock on the radio and the correlation of received signal strength on ground and in space, it points to attitude variations of the satellite during the pass rather than a problem with the loop-back mechanism itself.

The test run on 05:05 had a decent signal strength and was running for the full duration as the radio signal analysis showed. This requires a stable uplink signal to retain the clock signal for the transmitter section of the radio. Thus, the only plausible explanation for the unsuccessful attempt is a demodulation or processing issue on ground. Although signal variations could be observed in every run, the signal was always clearly visible.

Unfortunately, the secondary objective of verifying the loop-back SpaceWire connection could not be entirely achieved in this specific run. The space packet with APID 259 was never observed and the time range where the mode was selected by the control program showed only a weak signal. This was tracked down to the CCSDS Engine operating at the nominal downlink rate of 1 Mbit/s, but since the ground modem was configured to 256 kbit/s, it could not demodulate the signal. Due to limited time during the commissioning campaign no further tests with an appropriate setting for the downlink rate could be performed. However, dedicated tests after the commissioning phase and the monitoring data of the control program verified that the CCSDS Engine could successfully establish a SpaceWire link with itself due to the loop connection. Although not all prepared test scenarios could be completed as envisioned, the primary objective of verifying the function of the communication chain components and the involved electrical interfaces, could be achieved.

## 6.5. Conclusion

The intended loop connection over the ground and space segment using the CCSDS Engine's bypass mode was successfully demonstrated as a feasible test approach. Although the nominal mode was omitted in this test, it was verified with subsequent commissioning tests. The general strategy and the binary sequence of the approach presented proved beneficial as an end-to-end test performed in a single step. Furthermore, the successful functional check of all the components and interfaces of OPS-SAT's communication chain up to the processing platform could be achieved in one pass of the satellite. The demonstrated flexibility also proved advantageous during the pre-launch ground tests. In this case, the fully assembled spacecraft could not cover all the desired tests without modification. The reconfigurability of the integrated FPGA was key to increasing the test coverage in the most representative way.

The presented approach also demonstrates the versatile and flexible platform of OPS-SAT. In particular this test scenario was only possible thanks to the in-flight reconfigurability of the processing platform and its integrated FPGA. In addition, it shows how increased hardware reconfigurability has the potential to massively extend the solution space for improved and better testing, and empowering innovation in flight.

The LVDS connection between the CCSDS Engine and the processing platform, as one of the tested interfaces in the loop, has many interesting features for the OPS-SAT mission. Firstly, different protocols placed on top of the physical radio layer can be implemented in bypass mode. This enables a whole range of protocol tests using software (or firmware) experiments that would otherwise be constrained by the CCSDS Engine protocols. Secondly, it provides the physical layer of a SpaceWire link between the CCSDS Engine to the processing platform. As the loop-back test of the CCSDS Engine's SpaceWire link was also successful, further use of it is also feasible. The following chapter describes the in-flight implementation of this SpaceWire link.



## Implementing SpaceWire

After a satellite launch, ideas about potential additions may arise, but then it is often too late to implement them. One potential field of particular interest for enhancements, as it is constantly evolving, is communication. It is a fundamental function of any satellite, regardless of its purpose, and there are ongoing improvements to communication protocols and methods. However, communication systems are often implemented as fixed building blocks, making it typically impossible to modify the communication chain in flight.

Even though OPS-SAT's communication chain has radio transceivers and transmitters with fixed modulation, they can operate in different configurations. The configuration switch and basic CCSDS-compatible operation is managed by the CCSDS Engine protocol translator device. In its default configuration the CCSDS Engine provides forwarding of data traffic between the radios and the main CAN bus. The CAN bus interface is per standard limited to 1 Mbit/s. Due to protocol overhead and inevitable segmentation overhead, the maximum net data rate is below 500 kbit/s and the maximum (space) packet size is limited to 256 B. These limitations are acceptable for configuration operations and standard telemetry retrieval, but for larger data exchange this presents a severe shortcoming. Therefore, the comparably weakest link of the communication chain is the connection to the processing platform.

These limitations were already considered in the design phase, and thus the CCSDS Engine provides also a direct LVDS connection to the processing platform. This LVDS connection can be used in a bypass mode to

## Chapter 7. Implementing SpaceWire

gain direct access to the radios, as employed in Chapter 6. However, to use LVDS concurrently with the CAN interface, buffering mechanisms and a protocol enabling flow-control is necessary. Therefore, a robust standard protocol for this link was planned prior to the launch, and, finally, SpaceWire was selected and integrated on the CCSDS Engine. The process and considerations leading to this decision are summarised in the next section.

This direct SpaceWire link from the processing platform to the CCSDS Engine could overcome the CAN bus limitations. The CCSDS Engine supports data rates of up to 10 Mbit/s on this link without particular limitations on the packet size. This makes it possible to take advantage of the high data rates that are available on both radios.

However, there was not enough time to integrate the link counterpart into the processing platform, so it was considered that its reconfigurability could be used to implement it in-flight. So a suitable controller was synthesised in the FPGA, tested and uploaded to the spacecraft. This controller also had to be integrated with the existing Linux system. A separate, but tightly coupled, software component has been developed to facilitate the seamless use of this interface.

The next section explains the evolution of the design to accommodate a suitable communication link and the reasoning behind the selection of SpaceWire.

### 7.1. Evolution of an Alternative Link

The initial proposal to connect the processing platform to the CCSDS Engine was a SPI connection. The processing platform would have been the slave component, but no corresponding driver was available in the Linux kernel. In addition, a suitable protocol would have been needed. For avoiding this additional development effort and for hardware simplicity, it was decided to use the existing LVDS connection with a dedicated flow control line.<sup>1</sup>

It was planned to use the LVDS interface to transmit a raw stream of space packets with clock and data pairs in each direction. To signal that the buffer

---

<sup>1</sup>The change to LVDS was approved by mail on 8.8.2017

## 7.2. *The SpaceWire Standard*

on the CCSDS Engine is full, a unidirectional clear-to-send flow control line was provided. This approach is generally known for serial interfaces such as RS232. However, the processing platform could not signal its currently available buffer space. This approach could have been extended by a second line in the opposite direction to signal the buffer status from the other end.

This design neither accounts for error detection nor correction on link-level. In addition, the current link state is unknown on both end-points. Taking these points into account and that errors are very common in harsh environments like space, it became clear that another protocol had to be implemented for this purpose.

Unfortunately, at this point in time major parts of the system design, including the CCSDS Engine and the processing platform, were already fixed. Moreover, the bus architecture and system-level design could no longer be changed. Thus, the issues had to be solved on protocol level and a survey on suitable protocols was done.

From that, SpaceWire emerged as a viable alternative to the original plan. It provides, among other features, fully bidirectional flow control, which allows buffer back-pressure on both ends of the link. In addition to that, SpaceWire also provides error detection on link-level with automatic link resynchronisation and link-state monitoring. Another notable feature are the high data rates of up to 400 Mbit/s, depending on the actual hardware implementation and LVDS interfaces.

The only hardware requirement for a SpaceWire implementation is a LVDS interface consisting of two differential pairs in each direction of the link. For OPS-SAT this necessity was already fulfilled in the hardware design. The only missing part was the logic to drive the interface.

## 7.2. The SpaceWire Standard

SpaceWire was driven by the need for high-speed on-board data-handling and processing, specifically on a spacecraft. To lower integration costs, improve compatibility, increase reliability and enable reuse of equipment across missions the SpaceWire standard was defined by European Cooperation for

## Chapter 7. Implementing SpaceWire

Space Standardization (ECSS). The first version of this standard [67] was published in 2003 by ESA's SpaceWire Working Group. It evolved from two pre-existing standards, i.e. [46] and [1]. SpaceWire has since then been used in various missions like Sentinel-1, BepiColombo, James Webb Space Telescope and Rosetta by ESA, NASA, JAXA and Roscosmos. [68]

ANSI/TIA/EIA-644 [1] is used as signal level. Due to the differential signalling it provides increased noise immunity against induced fields and low Electromagnetic Interference (EMI). Furthermore, ground loops are also avoided. SpaceWire borrows its data encoding and character level from data-strobe differential-ended (DS-DE) IEEE standard 1355-1995 [46]. It defines data to be transferred over two differential signal pairs in each direction, which makes it full-duplex. Due to the data-strobe encoding the signal carries the clock, avoiding the need for a global clock. Also the SpaceWire character level, flow-control mechanism, and the parity checks [67, p. 53] are based on the IEEE standard [46, pp. 38].

### 7.3. IP Core Selection

There are a large number of different IP cores that implement SpaceWire, with a wide variety of features. A list of the IP cores considered for the implementation on OPS-SAT is given in Table 7.1. Several factors were taken into account when deciding which of the candidates to select: Firstly, as so often, cost. Second, the implementation effort on the (different) hardware architectures, and third, the achievable interface speed. These criteria and the rationale behind them are explained in the following paragraphs.

Cost was the primary consideration as there was no budget for additional functions when the need arose. Therefore, the first three IP cores in the table were not considered, despite their stable and flight-proven code bases.

Another selection criterion was good support for different FPGA architectures. In particular, the IP core should support implementation with Intel/Altera toolchains for the processing platform and Microsemi/Actel on the CCSDS Engine, as it was planned to use the same on both endpoints to avoid compatibility problems later. This could be achieved either by specific



### 7.3. IP Core Selection

Table 7.1.: Listing of the IP core candidates considered for the SpaceWire implementation.

<b>Manufacturer</b>	<b>IP core</b>
Frontgrade Gaisler	grspw_codec, grspw2
Star Dundee	SpaceWire Interface IP
ESA	SpaceWire CODEC IP Core
OpenCores	SpaceWire light
OpenCores	SpaceWire
IRAP	Free SpaceWire IP core

support for these platforms or by flexibility in development. The availability of the source code was therefore also considered an important criterion, given the potential for future customisation and incompatibilities between the two implementations.

Another issue was the effective interface speed, as any SpaceWire compliant component must support at least the base rate of 10 MHz. However, this rate is already well above the maximum S-band radio capacity. As S-band is the primary communication channel, it was considered sufficient.

Thus, three IP cores were more closely considered. The "SpaceWire" core was ruled out, because it contains too many unneeded features and seemed to be not actively developed. The receive frontend of the IRAP "Free SpaceWire IP core" derives the clock signal from the input. Therefore, it requires asynchronous clock domains, which results in additional implementation and test efforts. In contrast, the "SpaceWire light" IP core is a fully synchronous design with oversampling frontends, which was expected to be less error-prone for implementation on different FPGA architectures. This led to the final decision for it.

SpaceWire light is a publicly available IP core [63] providing a SpaceWire compliant physical interface. Although it is targeted at Xilinx-based platforms [63, p.19], the non-optimised version also compiles well for the present Intel/Altera Cyclone V platform.

## 7.4. Implementation and Deployment

The most efficient way to implement the chosen IP core on OPS-SAT was conceived to use the same implementation on both end-points. This potentially avoids later incompatibilities and implementation efforts can be combined.

On the CCSDS Engine the streaming version (*spwstream*) was implemented because it provides a very similar interface to the one found on the IP cores which are handling up- / downlink encapsulation and decapsulation. Because of the hardware design and the internal main clock rates only the standard SpaceWire communication rate of 10 MHz was implemented.

On the processing platform side of the link the requirements were different: the packet data has to be transferred from and to the HPS where the Linux operating system is running. Therefore, a matching interface to the base version of the IP core was necessary for the data exchange between the FPGA and the HPS portion of the SoC. The optimum solution for an efficient transfer was a Direct Memory Access (DMA) controller, offloading the time-consuming task of data transfer, and an interface conversion to connect to the SpaceWire light core. The resulting setup is depicted in Figure 7.1.

This interface was implemented as an additional custom developed component<sup>2</sup>. It translates an Avalon streaming interface [6, ch.5], more specifically with packet interface [6, p.50], providing a seamless integration into Intel's synthesis framework, to the data interfaces of the SpaceWire IP core. Due to the framework compliance, a modular Scatter-Gather Direct Memory Access (mSGDMA) IP core could be used as DMA controller, which is part of this framework and provides a good performance in terms of speed and used area.

As a final step towards full integration into the operating system, a 'spw' kernel module<sup>3</sup> has been integrated as depicted in Figure 7.2. The module is based on the Altera Triple-Speed-Ethernet<sup>4</sup> driver implementation, which has very similar requirements. The module configures the link management

---

<sup>2</sup>The enhanced SpaceWire light IP core is publicly available [36].

<sup>3</sup>The code of the kernel module is publicly available [35].

<sup>4</sup>see [72, Version 4.9, `drivers/net/ethernet/altera/altera_tse.c`]

## 7.4. Implementation and Deployment

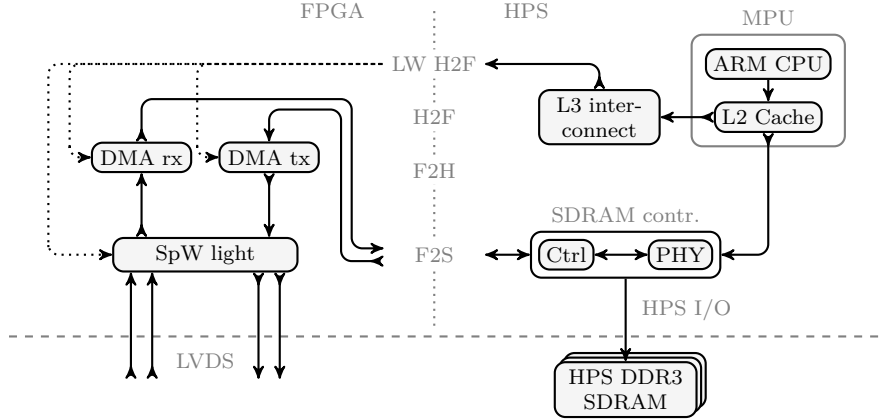


Figure 7.1.: Initial SpaceWire architecture on the processing platform. It used the HPS SDRAM for data exchange. The data transfer is executed by two DMA controllers, one for each direction, implemented in the FPGA part. The SpW light IP core forwards and converts the data to the physical LVDS interfaces.

of the SpaceWire IP core and initialises it appropriately. Furthermore, it manages the DMA controller and schedules the transfers. In terms of data transfer it exposes a network interface in the kernel, thus providing full integration into the Linux networking stack. As such it is usable by any user-space process. The SpaceWire link theoretically supports packet data of arbitrary length, as it only uses End-of-Packet (EOP) markers between packets. However, the buffers need to be limited and therefore the module supports a chosen but configurable Maximum Transmission Unit (MTU) of up to 8 KB.

As a result, the SpaceWire interface can be used like a normal network device, supporting all the usual operations, statistics and link state reporting. It can also be used by the higher layers of the Linux kernel network stack, which provides many existing protocols. This seamless integration into the network subsystem was also employed to capture the long-term statistics presented in Figure 7.4, as it could be queried by readily available monitoring software. Another positive aspect of this concept is the fact that the processing-intensive memory transfer is handled by the DMA control-

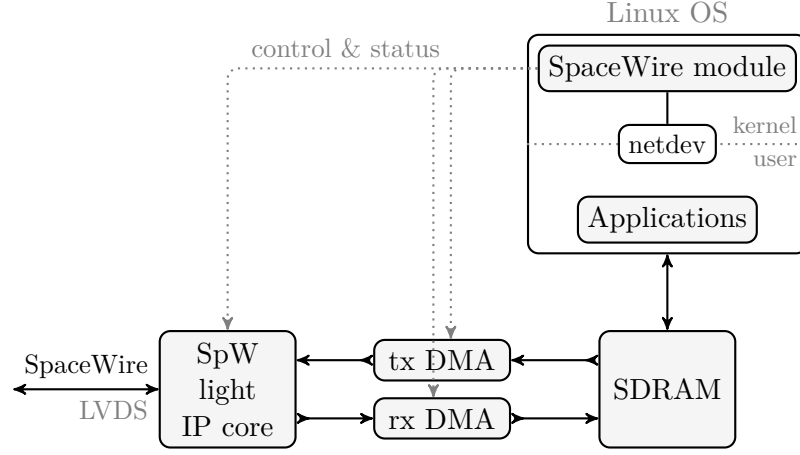


Figure 7.2.: High-level software architecture of the SpW implementation. The SpW light IP core and the DMA controllers for the data transfer are operated and monitored by a Linux kernel module. It exposes a network interface to user space applications. The data transfer utilises shared buffers in SDRAM.

ler, which enables very efficient processing in terms of CPU utilisation and time.

Unfortunately, not all applications running on-board the processing platform were able to access this network interface. Instead, some of them require a TCP socket serving a bidirectional stream of space packets. As a preliminary solution a user-space adapter application was developed. This adapter connects to the exposed SpaceWire network interface and transfers incoming and outgoing space packets from and to a (local) TCP socket.

Even though the first tests looked promising, an issue was discovered when testing the implementation on the engineering model: the CCSDS Engine cut off the last byte of every incoming packet. The Space Packets transferred over this link have a CRC-16 appended. Therefore, half of this CRC was missing at the receiving end, i.e. the processing platform. An operational workaround was therefore required and implemented: the kernel module calculates the CRC, compares it to the remaining byte, and adds the missing half if the check is successful. In case the check fails, it appends a zero byte.

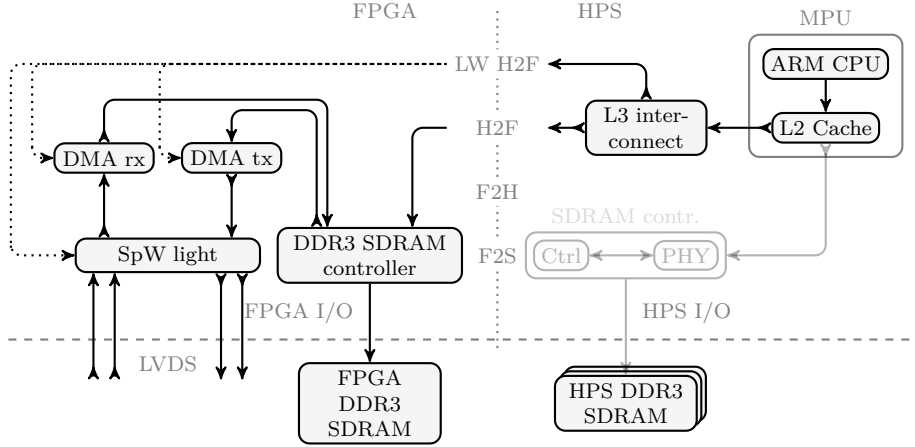


Figure 7.3.: Revised SpaceWire architecture on the processing platform with FPGA SDRAM as data exchange.

Thus, for space packets without a (valid) CRC at their tail, at least the correct packet length is ensured. For space packets with a (half-)valid CRC, it is augmented and the resulting packet is complete.

The kernel module can be loaded at any time to the running kernel. The FPGA configuration is applied as described in Section 3.4 and requires a platform warm reboot.

However, this initial implementation was invalidated with the swap to the FPGA SDRAM. Therefore, the initial design was adapted to fit to the changed requirements to the structure depicted in Figure 7.3. Instead of using the HPS SDRAM via the F2S bridge, the SDRAM is accessed directly in the FPGA fabric. The other components like the control of the DMA controller or the SpaceWire light core, are still operated in the same way as before.

## 7.5. Results

After successful testing on the engineering model, the kernel module, the additional applications and scripts, and the FPGA configuration were fi-

## Chapter 7. Implementing SpaceWire

nally uploaded to OPS-SAT on 30/11/2020. The first successful test was performed on 10/12/2020 04:32:34. End-to-end operation was confirmed by space packets generated on the processing platform and observed in the data systems on the ground. It was also possible to transmit a file over this link to verify the software implementation. The link was then put into operation, albeit with a longer interruption due to the SDRAM recovery.

As the kernel module exposes the link statistics on the network device, they can be conveniently acquired with readily available monitoring software<sup>5</sup>. Such acquired long-term statistics of the link, depicted in Figure 7.4, show that the maximum downlink rate (1 Mbit/s) is achieved for the majority of passes where a link could be established. The gaps with no or only limited downlink utilisation are due to, e.g. the S-band radio not activated, no contact to the ground station, or the CCSDS Engine or the processing platform not being operational. The downlink peaks of up to 3 Mbit/s are the result from tests with the X-band chain.

The average downlink rate never reaches the maximum downlink capacity due to several factors. Firstly, it is averaged over the entire pass duration, but (bi-directional) ground contact is only physically possible for much shorter periods, when the radio link between the ground station and the on-board transceiver is strong enough in both directions to establish communication. Secondly, the concurrent downlink over CAN bus is prioritised in the CCSDS Engine. Thirdly, the high-level link handshaking introduces an additional delay before the downlink can be fully utilised. This round-trip delay is in the order of at least 500 ms mainly due to processing delays on ground. Fourthly, the full downlink capacity is not always needed, similar to the uplink usage.

In summary, the overall performance of the introduced SpaceWire link is outperforming the concurrent CAN interface. It achieves downlink volumes of more than 70 MB per pass and uplinks of more than 9 MB. Although, many additional factors are affecting and limiting the achievable end-to-end performance, the added on-board link was a major step forward to drastically increase mission return.

---

<sup>5</sup>The 'collectd' [17] monitoring daemon was used.

## 7.5. Results

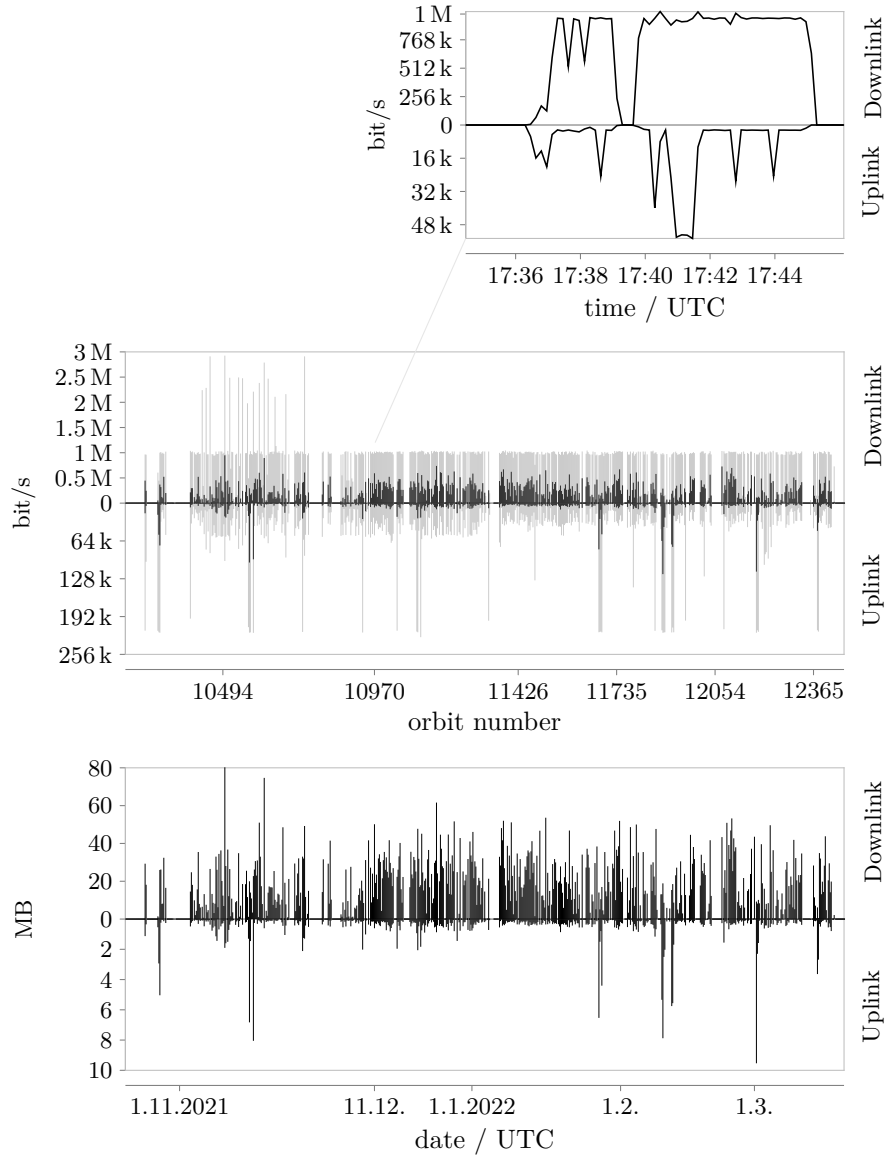


Figure 7.4.: Up- and downlink statistics of the SpaceWire interface covering approximately six months. The mean and maximum values are aggregated per pass and are shown in the upper part. One pass, orbit number 10970 on 11/12/2021, is plotted in the top region. The lower part depicts the total up- and downlink volumes.

## 7.6. Conclusion

The new on-board SpaceWire link enables high bandwidth communication directly between the processing platform and the CCSDS Engine. In addition, it provides flow-control, advanced error detection, and link state monitoring features. Combined with the accompanying software implementation in the Linux kernel it offers a seamless and transparent access to this additional bus.

As the SpaceWire connection has no restrictions in terms of packet size, IP-based communication also became possible. This was subsequently implemented in a CCSDS-compliant way on the ground and in the space system and triggered a whole wave of innovations thanks to the availability of standard networking software under Linux. In addition to secure shell access (SSH), it also enabled file transfer software such as "rsync" and facilitated remote access for experimental purposes.

The successful implementation of this new interface shows that it is feasible to add new functionality to the spacecraft in-flight. Although the implementation was planned and the CCSDS Engine had already implemented it, the approach of implementing and testing the other half of the link in orbit is unique. In addition, the implementation can be changed at any time, e.g. if problems are discovered with the current implementation, it can be updated, which will come in very handy in the following chapter.

At another scale, the whole implementation could have been avoided by using the CCSDS Engine in bypass mode, and thereby any rate supported by the X-band or S-band radio would be usable. However, in this scenario the concurrent use of the CAN bus, as required for the direct communication with the OBC, needs to be implemented on the processing platform. Although this is feasible, there are additional requirements and functionalities to be taken into account, than for the present approach.

Unfortunately, after months of smooth operation, the SpaceWire link showed instabilities. The CCSDS Engine and the processing platform reported errors on the link and on some occasions it could not be brought up again. The investigations, again utilising the flexibility provided by the FPGA, and the results are discussed in the following chapter.



## Debugging SpaceWire

The SpaceWire interface connecting the CCSDS Engine and the processing platform became increasingly unreliable. Although this connection is fault-tolerant, there were intermittent link drops where both ends reported the link as being down. The behaviour was not reproducible on ground and no correlations with external events could be found. The issue and the investigation approach were already published<sup>1</sup> and this chapter contains material from it [37], which was partly adapted and extended, without explicitly citing it.

For investigating these link instabilities several operational states were tried. As a first candidate for causing electromagnetic interference the magnetorquers were suspected. Therefore, the internal magnetorquers and the magnetorquers located on fine ADCS were turned off during passes. This seemed to have no influence on the link drops. Another component causing interference, which is also observable in magnetometer readings, is the UHF transceiver. This could be excluded by turning it off, but this also had no impact.

The investigation would have ended when no clear correlations could be found to determine the cause. On the ground, a logic analyser could have been used to gain more insight into the behaviour of the interface. However, it is possible to implement this function in an FPGA, such as the Signal Tap logic analyser included in Intel's FPGA development suite [48, Sec. 2]. In addition, it allows monitoring the FPGA implementation's inner workings.

---

<sup>1</sup>© 2023 IEEE. Reprinted, with permission, from [37]

The next subsection covers the framework, in which the Signal Tap logic analyser is embedded, in more detail.

However, remote execution of Signal Tap with an intermittent connection is not possible with the vendor's existing development suite. This was solved by developing a custom control application on the processing platform that performs the logic analysis autonomously. In addition, the current approach requires uploading and provisioning FPGA bitstreams on the satellite. Fortunately, reconfiguration of the processing platform is one of the core functions of the OPS-SAT mission. The deployment process, the implementation of autonomous operation and its successful execution is described in Section 8.2. Thanks to the captured traces, it was possible to investigate the physical behaviour of the interface. Selected results are presented in Section 8.4.

## 8.1. System-level Debugging Framework

With electronic devices getting increasingly complex also appropriate debugging tools are necessary. Due to higher integration with less accessible test points in assembled circuits, it is often nearly impossible to monitor the internal signals and states of a circuit or inject test data. Therefore, the Joint Test Action Group (JTAG) codified a standard [47] to address this need. A JTAG interface provides a debug port with data-in, data-out, mode-select and clock lines. Sometimes also an optional reset line is available.

This four-wire connection allows to span a serial *scan chain* through an assembled circuit consisting of the Test Access Ports (TAPs) inside the involved chips. Each TAP has a standardised state machine and provides access to the Boundary Scan-Cells (BSCs) of a chip. These scan-cells are positioned at the boundaries of each chip. Thereby, it enables read and write access to all externally accessible pins of the JTAG-enabled chips in a circuit. In a daisy-chain configuration it works like one large shift register containing all BSCs. Therefore, a good coverage, although at a slow access speed, can be achieved with this small amount of physical (inter-) connections.

This approach for an easily accessible test facility is extended by Intel's System-Level Debugging (SLD) infrastructure [76]. It abstracts the usual

### 8.1. System-level Debugging Framework

JTAG chain with a so-called virtual scan-chain. A virtual scan-chain contains only one node. The SLD Hub takes a role similar to a (virtual) TAP and arbitrates access to these scan-chains and the associated nodes. It allows to address each SLD node directly without the need to use the full JTAG scan-chain.

The SLD Hub arbiter operation is controlled with USER1 and USER0 instructions. The USER1 instruction expects the concatenated address and (virtual) instruction register value in the subsequent data register shift. The addressed node receives the instruction in its instruction register and eventually selects one of its virtual data registers. A following USER0 instruction combined with a data register shift allows to shift data into the previously selected node's data register.

However, this flexible and extensible debugging approach still requires physical access to the SLD Hub. In general this is accomplished via a JTAG connection to the FPGA where the SLD Hub is synthesised. To enable remote debugging, a bridging interface is available for SoC devices as the SLD Hub controller IP core and corresponding Linux integration. The `sld-hub` kernel module [13] provides a seamless access to the SLD Hub in the FPGA portion. Together with the `mmLink` application [12] it allows (networked) forwarding of JTAG transactions as depicted in Figure 8.1 and described in the application note [4].

For Intel FPGA systems the Signal Tap logic analyser IP core and frontend application is part of the provided development suite. Signal Tap allows to capture any signal in the FPGA fabric. Thereby it also provides access to external signals present on FPGA pins and, in case of a SoC, also rerouted pins from the HPS. Moreover, it provides features like flexible trigger conditions and segmented captures. The Signal Tap FPGA component uses the SLD framework and appears as an SLD node [48, Sec. 2].

A debugging functionality synthesised in the FPGA fabric unfolds its full potential with a controlling application on the host system. In an Intel FPGA development environment the System Console application serves this purpose. It handles the connection setup and arbitrates access to the nodes of an SLD-enabled debugging target, regardless of the physical connection, which can be physical JTAG but also a remote setup where JTAG transactions are forwarded over network [4]. The Signal Tap frontend application

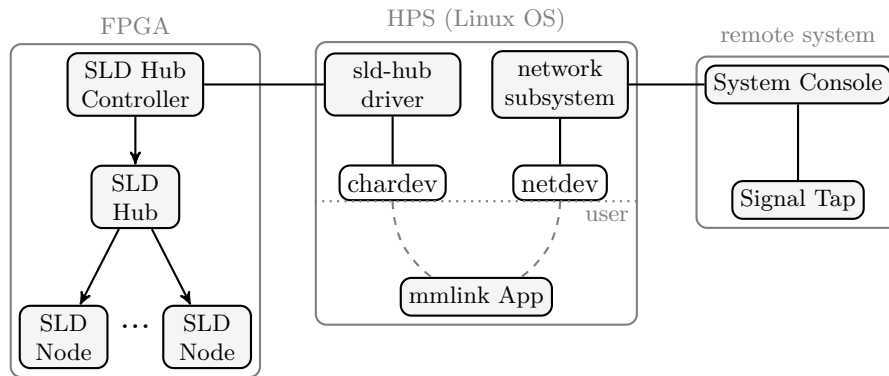


Figure 8.1.: System architecture of an SLD-enabled remote debugging system. Adapted and extended from [4, Fig. 2]. On the FPGA portion, the SLD Hub controller forwards JTAG instructions to the SLD Hub which orchestrates access to the connected SLD nodes. The nodes can be, e.g. Signal Tap (logic analyser), in-system sources and probes or even other JTAG controllers. The SLD Hub controller is accessed over an `sld-hub` kernel module from the HPS side. This exposes a character device to user space. Access to this character device is exposed to network by the `mmlink` application. Software like System Console (part of Quartus) can further distribute the network exposed interface as a virtual JTAG interface to other applications on a remote system, e.g. Signal Tap.

## 8.2. Integration on OPS-SAT

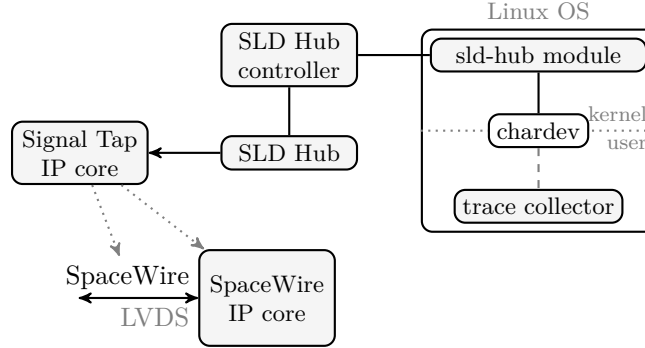


Figure 8.2.: System architecture for autonomously capturing SpaceWire operation. The SpaceWire core and its external connections are probed with a Signal Tap instance which is connected to the SLD Hub and the controller. The controller is accessible from Linux user space (on HPS side) via the `sld-hub` module. A custom trace collector application uses the exposed interface to set up the trigger conditions, monitor the state and eventually retrieve the traces from the attached Signal Tap instance.

uses the interface exposed by System Console to connect to the debugging target and allows interactive control.

These advanced debugging techniques also have limitations. One is the FPGA synthesis of the analysis tool. Since it is synthesised as part of the whole FPGA design also the same timing constraints apply to it. This limits, for example, the capturing speed to the rates supported by the FPGA. Another limitation is the direct and uninterrupted connection to a controlling remote system. Regarding the targeted scenario, this presents a major drawback and is thus addressed in the following subsection.

## 8.2. Integration on OPS-SAT

The mentioned Signal Tap logic analyser lacks one important property for the intended use: Autonomous operation is not possible, but it is a crucial requirement due to the intermittent access to the satellite. Therefore, the

trace collector was developed. It is a custom utility to operate Signal Tap over SLD Hub without requiring direct interaction. The resulting overall system architecture is visible in Figure 8.2. Since the trace collector application uses the `sld-hub` kernel module for accessing the JTAG chain, the module needs to be loaded as a prerequisite. The application performs the necessary steps to setup the trigger conditions and the trace capture process of an SLD-attached Signal Tap instance.

The trace collector program flow is depicted in Figure 8.3. It is split in three consecutive phases: Initialisation, setup and run phase.

The **initialisation** requires three steps. Firstly, the JTAG chain is scanned for available TAPs. Because the number of TAPs is unknown a priori, the whole chain is switched to bypass any input. Issuing the bypass instruction to the TAPs puts a single-bit register for each into the chain. In this configuration a single '1' is shifted through the chain and the number of shifts until the '1' appears at the output is counted. The number of necessary shifts equals the number of TAPs in the chain.

Secondly, an IDCODE instruction is sent to the chain. Every TAP connects its 32 bit IDCODE register to the chain and a shift of 32 bit for each discovered TAP from the previous step retrieves the IDCODE for all TAPs in the chain, i.e. the SLD Hub and its position in the chain is determined.

Thirdly, the SLD configuration is discovered. Most notably, the number of nodes  $N$  in the design, determining the number of address bits, and the length of the virtual instruction register  $m$  are acquired. These values apply to all following SLD instructions. The HUB\_INFO instruction [76, p. 33], which consists of all '0's, provides this information and it is sent to the SLD Hub, which is always present as SLD node 0. Therefore, a sufficiently long sequence of '0's is shifted into the SLD Hub TAP or rather the instruction register of the virtual JTAG chain. The virtual data register supplies the desired hub information containing the values  $N$  and  $m$ .

Since the SLD instruction structure is now known, the hub and all nodes can be enumerated and identified by using the SLD\_NODE\_INFO instruction [76, p.34]. This allows to identify the SLD node number of the Signal Tap instance, and hence the setup is now possible.

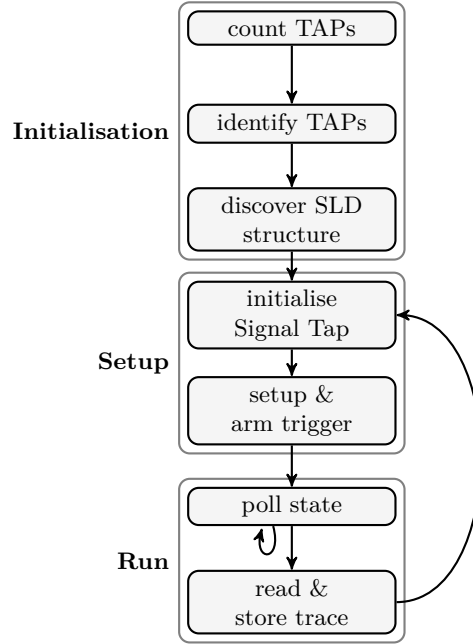


Figure 8.3.: Application flow of the trace collector. In its **initialisation** phase, the trace collector discovers the configuration of the JTAG chain and identifies all TAPs in it. When an SLD Hub instance is found, it enumerates and identifies the connected SLD nodes. For a Signal Tap SLD node the **setup** phase is entered. After the node is initialised, the trigger condition is setup and armed. In the **run** phase the trigger state is continuously monitored. As soon as it detects a stop of the acquisition, i.e. the trigger condition is met, the application retrieves the stored acquisition data, saves it and returns to the **setup** phase.

In the **setup** phase, the Signal Tap instance is initialised and configured with the desired trigger condition. Then the trigger is armed and the trace collector enters the run phase.

The **run** phase starts with continuously monitoring the acquisition state. On the FPGA side, the Signal Tap instance waits for a matching trigger condition, and as soon as it fires, the instance stores the result in a circular buffer and stops the acquisition. The trace collector detects the state change and retrieves the result from the buffer. It stores the captured data and returns to the setup phase. Thus, the trace collector development enables an autonomous logic analyser acquisition function. The deployment on the processing platform is presented in the following subsection.

### 8.3. Deployment

After successful tests on the OPS-SAT ground engineering model, the logic analyser was ready for deployment in-orbit. The deployment of an OPS-SAT FPGA experiment requires uploading of the experimental application, i.e. trace collector and the `slid-hub` kernel module, and the FPGA image. For applying the FPGA configuration the method described in Section 3.4, specifically the one with the SDRAM fix, was used.

Once the FPGA configuration is in place, the autonomous logic analysis can be started. The data-flow is depicted in Figure 8.4. The trace collector discovers the SLD structure including the Signal Tap instance. It sets up the trigger condition and arms the trigger. Afterwards, the state of the logic analysis is continuously monitored.

As soon as a trigger condition is met, the trace is captured by the Signal Tap instance and stored for later retrieval in FPGA memory. The trace collector application detects this, reads the trace data, stores it in a format suitable for later processing, and restarts the capturing loop.

At the next possible occasion the collected traces are downloaded from the satellite. On ground the trace data can be processed and analysed. One captured trace of the SpaceWire live operation is described in the following section.



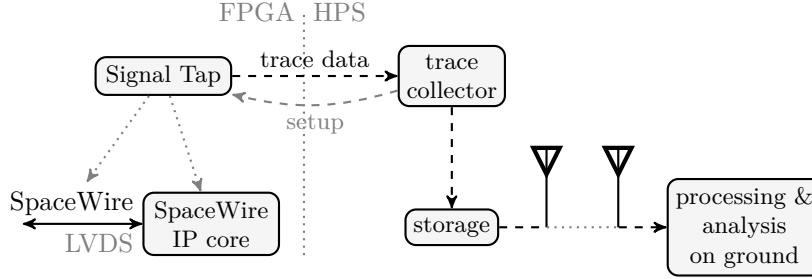


Figure 8.4.: Data-flow of the autonomous logic analyser function. The trace collector application sets up the triggers and arms them. The state of the Signal Tap logic analyser is continuously monitored and as soon as a trigger condition is met, the data is captured. The trace is retrieved and stored. At the next occasion the trace data can be downloaded from the satellite and processed further.

## 8.4. Measurement Results

Using the captured traces an in-depth analysis at a low level of the on-board SpaceWire communication link was possible. To achieve that, the SpaceWire signal lines and the FPGA internal behaviour were monitored. Because no obvious issues were noted in the implementation, only the outgoing and incoming SpaceWire signal lines were further analysed. These are depicted in Figure 8.5.

The SpaceWire standard [67] defines that if any end detects errors on the link like wrong parity bits, invalid control character combinations or silence, the link partner is considered disconnected or erroneous. In this case the link establishment is started and a link handshake is performed. In the handshake phase, one end tries to (re-)establish the link by sending NULLs. A NULL control code indicates an idle link and is sent if there is no payload data available. It consists of an Escape (ESC) character followed by a Flow-Control Token (FCT).

If the link partner receives a NULL character, it replies with a NULL to indicate its presence. Subsequently, it starts sending FCTs to signal that there is free space in its receive buffer. Each received FCT permits the

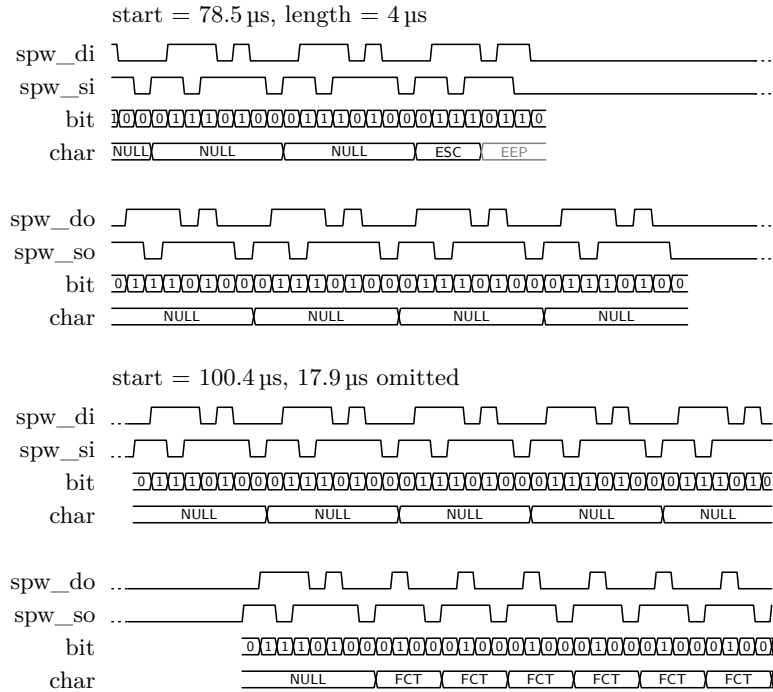


Figure 8.5.: Two selected sequences of a logic analyser trace of the SpaceWire signal lines on OPS-SAT. The trace was captured on 16/07/2021 at 00:37:14.312 UTC with a 100 MHz sampling rate and a total length of 163.84  $\mu$ s. A more extensive version of this trace is shown in Figure B.4 on page 113. In the top sequence the far end (spw\_di, spw\_si) sends an idle period (NULLs) and introduces an error by sending an invalid sequence of ESC followed by an EEP control character. Consequently, the far end starts its 12.5  $\mu$ s reconnect timer and the near end (spw\_do, spw\_so) stops transmission because of the error. In the lower part, the following reconnect sequence starts with an idle pattern (NULL) from the far end. The near end detects and acknowledges it with a NULL and FCTs. However, these seem to be unrecognised by the far end, because they should have been answered with FCTs in return.

#### 8.4. Measurement Results

receiving end to send eight N-Chars, the bytes of a packet. As soon as the initiating side detects the arriving NULL and FCTs, it replies with FCTs to signal the state of its own free buffer space. In case any end fails to emit NULLs or FCTs in the handshake phase, the link handshake has failed and the handshake sequence starts again after a timeout.

With this handshake mechanism an established SpaceWire link proves that both ends are able to communicate with each other. The exchange of FCTs guarantees that each end has enough buffer space available for an upcoming exchange of N-Chars.

In the trace on top of Figure 8.5 an already established link is visible. Both ends are emitting NULL characters to signal an idle link with no user data exchange. As can be seen in the centre of the top trace, the far end seems to detect an error since it stops transmitting NULLs. The near end detects this problem as an invalid sequence of ESC character followed by an Error End-of-Packet (EEP) marker. Thus, it also stops transmitting.

Consequently, the far end starts its reconnect timer to wait at least  $12.5\ \mu\text{s}$  before starting the next link handshake. The next link handshake – visible in the lower part of Figure 8.5 – starts with NULL characters emitted by the far end. The near end correctly detects this, because it replies with a NULL character followed by FCTs.

However, the far end seems to have either no buffer space available or is unable to successfully detect what was sent, because it does not reply with FCTs. Instead it keeps sending NULLs until the link establishment times out. In conclusion, it can be derived that the near end is operating correctly, but the outgoing direction of the link is not working as expected. This could be explained by a misbehaviour on the far end or any type of signal degradation on the lines in between.

One possible explanation for the signal degradation on the lines could be interference. Due to the dense integration of the satellite there are many potential sources nearby. For example, it could be caused by any of the six internal magnetorquers, one of the radios or just a mismatched data signal line close to the affected ones. Also a damaged switching regulator may produce interfering signals. However, the differential signalling on the physical interface should mostly alleviate these effects. Another possible

## *Chapter 8. Debugging SpaceWire*

issue could be a loose contact, caused by the vibration shocks during launch, or a broken wire. In both such cases a temperature change could toggle the connection state.

The erroneous behaviour may also be caused by the far end of the SpaceWire link. A temperature related effect in the receive pins or in the processing unit would also explain the observed behaviour. Another reason might be a drift of the common mode voltage beyond the positive or negative limit of the differential amplifier input at the far end.

The analysis is still ongoing and there are indications, that it could even be a combination of more than one of the previously mentioned causes. Even without determining the specific cause of the degradation, this type of analysis and the degree of visibility at the physical level would have been impossible without the measurement approach presented above.

### 8.5. Conclusion

The results above demonstrate the debugging capabilities of a reconfigurable FPGA in the field. Its availability, and the logic analyser implementation on OPS-SAT, provided a unique insight into the internal signals. Additionally, the design of the on-board logic analyser, its safe deployment, and successful execution on the OPS-SAT processing platform are presented.

Using the captured traces an in-depth analysis of the problematic SpaceWire communication link could be performed, that would have been impossible on a normal spacecraft. This provides a deep insight into the inner workings of the design running in the FPGA and all the external signals connected to it. These results highlight just one of the possibilities made possible by the inclusion of in-flight reconfigurable hardware, such as FPGAs, on space missions.

In this case the Signal Tap implementation was sufficient to monitor and trace the physical link. For future uses it can be extended to support complex trigger conditions or multiple consecutive captures. Furthermore, the same measurement principle could be implemented from scratch or with another IP core in a more customisable way, because Signal Tap is limited to the

### 8.5. Conclusion

vendor's supplied functionality. A potential extension beyond monitoring only could involve a flexible approach to injecting test vectors into the live system, even concurrently.

In summary, the processing platform was used to deploy a powerful and adaptable debugging facility in orbit. Furthermore, the methods utilised illustrate the unique flexibility provided by flying integrated FPGAs on spacecraft. Finally, the presented approach could be extended to allow future missions to implement more advanced debugging mechanisms. At another scale, the implementation demonstrates how increased hardware flexibility has the potential to massively extend the solution space for the monitoring, debugging and system evolution of any remote system.



## Final Remarks and Outlook

### 9.1. Summary and Lessons Learnt

Redundancy is a well-established concept for highly reliable systems. However, it is often implemented in the form of functional redundancy, i.e. two or more identical units are used. But redundancy in the connection between the units is also essential. An extreme case of both types of redundancy is found in FPGAs, which provide many identical units with a switching matrix for interconnection. In addition, the pin muxing of the SoC provides an additional switching layer. This was exploited during a space mission, as described in this thesis, for new implementations, but also to gain additional redundancy via the switching capabilities provided.

A defect on a satellite can almost exclusively be mitigated by (previously added) redundancy. However, the redundant units or components need to be connected to the location of the fault. By taking advantage of the reconfigurability and high degree of interconnectivity of the (OPS-SAT) spacecraft and its platform, a recovery could be performed for an otherwise lost (experimental) mission, as shown in Chapter 4. It exemplifies, how hardware, or rather its behaviour, could be changed on the fly.

Similarly, a more advanced communication interface to the radio system was integrated in the operating spacecraft as described in Chapter 7. Although, this function was foreseen, the actual implementation was done in flight. The mechanisms, and their evolution over time, for carrying out the FPGA reconfiguration in a safe and operationally feasible manner were presented in

## *Chapter 9. Final Remarks and Outlook*

Section 3.4. Furthermore, it establishes hardware reconfiguration as a (operational) strategy, allowing mission operators to react to unforeseen events with both hardware and software changes.

The thesis shows this flexibility being used to implement new test scenarios as presented in Chapter 6. There, tests could be performed on ground and in space, in a time-efficient minimal-invasive way and with a broader coverage. An extension to this is the debugging framework described in Chapter 8. The in-flight logic analyser implementation and its possible enhancements demonstrate the high potential to enable or ease debugging of distant or only intermittently reachable systems.

In all these cases, the reconfigurable computing platform allowed to change hardware behaviour and software on the fly. Thereby, new functionalities could be added, which enabled, for example, recovery from an otherwise lost mission or adding a dedicated hardware interface into the design. Furthermore, testing and debugging could be done in unforeseen scenarios.

All these operations were enabled by the satellite platform containing more features than necessary, but these were not necessarily used by default. Implementing such redundancy mitigates potential failures. The expectation of failures in any phase of the project and the variety of ways to mitigate them leads to a diverse functional redundancy, as opposed to just using multiple identical components or units.

Due to the low-cost character of the mission, there had to be compromises regarding the component selection and the test effort, while still assuring the reliability. Therefore, the spacecraft design incorporates safe-guards on multiple levels, allowing to entirely separate subsystems from each other. This partly alleviates the risk imposed by using less extensively tested COTS components or new developments. Nevertheless, core payloads vital for the mission, like communications or power subsystems, were extensively tested. For other tests, however, the time was sometimes quite limited. For example, the full functionality of the SpaceWire connection could not be tested and there was not enough time to implement the interface on the processing platform before the launch. This resulted in a lot of theories for its erroneous behaviour that needed to be falsified. On one hand, more ground tests and traces under varying external conditions could have been helpful to speed up the process. On the other hand, it shows that time could be saved



### 9.1. Summary and Lessons Learnt

by not executing such tests and rather using the reconfigurability provided to do it in flight. As the erroneous behaviour was never observed on the engineering model, additional tests would probably not have provided any further insights anyway.

Another spacecraft bus that would have also required more testing is I<sup>2</sup>C. Reflection artefacts were observed on board and on the ground, but they looked different. Due to the different physical arrangement of the spacecraft and the engineering model, they are probably caused by the long and branched signal lines. In addition, the bus configuration can be changed dynamically using the data bus switches. As a result, the I<sup>2</sup>C pull-up resistors are not always in the optimum position to avoid reflections in different configurations. Here, additional tests could have led to better optimisation of the connected controllers. One approach to mitigate the limited time for testing, could have been to include more automation. Even if it means more effort in the first run, it will pay off in recurring tests and also results in an increased visibility of the current system state.

Such increased visibility is also of particular interest in later operations. This can be achieved by capturing the hardware and software state on ground as much as possible, without the immediate need for an interpretation. Those captures could range from low-level bus measurements to mirroring of filesystems. In case of a problem on-board, this information can be used for reference and might help immensely to identify the cause of an observed behaviour. Specifically, knowledge and visibility of (low-level) hardware behaviour could have helped, for example, for falsifying the various SpaceWire interference theories or the observed eMMC issues. However, the actual eMMC behaviour could never, even with manufacturer support, be really understood. By using well known or open hardware probably the cause could be identified, and the use adapted accordingly. In addition, an identical component on ground could have been used to reproduce the behaviour, but, unfortunately, this chip is currently not available.

Similarly reproducing a behaviour and obtaining knowledge of system-level behaviour is possible with an engineering model on ground that is kept (as far as possible) in a representative state. This includes not only the configuration, but also the entire set of payloads and, if possible, also the simulation of external influences. Unfortunately this could only be partially

## *Chapter 9. Final Remarks and Outlook*

achieved for OPS-SAT for cost reasons, especially with regard to the lack of S-band/X-band radios for ground tests.

Increased visibility could also be achieved by incorporating the debugging function into the spacecraft, as demonstrated in Chapter 8. Going one step further, it could also be a dedicated debugging unit, maybe with an integrated FPGA, or a well connected multi-purpose device with routable inputs and outputs. To further expand the range of debugging capabilities beyond those of an FPGA-based logic analyser, an Analog-to-Digital Converter (ADC)/Digital-to-Analog Converter (DAC) could be implemented, with inputs/outputs that can be routed to different devices or buses. This approach allows for the monitoring or injection of analogue signals into the system. Recently, there are FPGAs offered by the leading manufacturers that have ADC/DAC functions already integrated. The result is something like a flexible oscilloscope or signal generator integrated into the spacecraft, opening a whole new range of analysis capability.

In this mission the FPGA on the processing platform is reconfigurable. But the approach could be extended and other (FPGA-based) payloads could allow reconfiguration in the field. Most prominently, if the CCSDS Engine would have been reconfigurable, the SpaceWire communication could have been checked from both ends. Hardware updates could also have been made to address some of the issues found in-orbit. Despite the risks, the possibility of in-field soft- or firmware changes for any payload is very valuable.

An unchangeable component is the bootROM of the Cyclone V SoC, providing the very first boot stage after power-up. In the experienced scenarios of a failing eMMC or the use of the warm boot method for temporary boot flow changes, a change or complete knowledge of the bootROM's internal flow could have eased the developments.

Retrospectively, significant effort was spent in the interconnection between the HPS and FPGA portion of the Cyclone V SoC. Although these integrated bridges provide a powerful and flexible connection, their use can be challenging. Firstly, it is related to the different timing requirements, i.e. a multitasking system communicating with hard real-time processes in the FPGA. Secondly, representative testing of such implementations, similar to multi-threaded processes, is often only possible in real-time, which complicates testing and development.

## 9.2. Conclusion

The SLD infrastructure and its use without the manufacturer's frontend software tools was another challenge. The infrastructure and its protocols are optimised for the provided software environment. In addition, the available documentation does not entirely describe it. Identifying the necessary steps and (efficiently) implementing the needed protocols for an autonomous operation in space was thus a time-consuming task.

## 9.2. Conclusion

This thesis describes some of the achievements made in the course of the OPS-SAT mission. This includes the recovery from a scenario that could have seriously limited the experimental mission. This recovery allowed the continued use of the processing platform. Subsequently, the memory layout and the bootloader were improved to increase resilience to future memory failures. Furthermore, the communication chain was enhanced with a custom IP core to provide higher bandwidth and direct access to the processing platform. Also, methods for debugging and testing have been implemented in-flight to enable the verification of the communication chain's correct function or the observation of the spacecraft's internal signals. Most of these implementations were deployed in orbit on the operating satellite.

These developments were mainly event- or application-driven, and an attempt to describe the specific implementations and what worked, has been made. However, they can also be interpreted collectively in looking at what went wrong and how it was possible to find solutions that were all characterised by a common theme, i.e. on-board hardware flexibility. Furthermore, these examples could be used and extended to ask what could be achieved in the future. In this sense, the decisive point is the amount and extent of possibilities, given that the hardware flexibility on board is well implemented. The increasing possibilities are providing more flexibility, either in terms of new implementations or for advanced recovery scenarios. All in all, it provides more freedom and a higher resilience, particularly to unforeseen events. In addition, this flexibility aids in the design of sophisticated test scenarios and can provide a high degree of visibility into the internal behaviour of the system.

## *Chapter 9. Final Remarks and Outlook*

The applications presented only represent specific examples, thus being only a subset of the whole solution space which reconfigurable computing can open up. Furthermore, the reconfiguration approach enables and encourages for more up to date hardware in spacecrafts to prepare for the unknown challenges ahead. In addition, not every function has to be perfect at the time of launch as reconfigurable computing can provide an additional time buffer because it allows to test and adapt in orbit. This has the advantage that the tests are executed under real conditions and may therefore be more efficient. The goal of fostering experimental usage of the OPS-SAT mission, can thus be generalised to any payload, resulting in an agile development of a spacecraft, and its components. This development approach provides a potentially strong driver for innovation in space. In general, this is not limited to space applications, but valuable for any difficult to access system.

For other applications or satellite missions, it is probably easiest to consider the reconfigurability and interconnectivity aspects of a system during the early design phases. Nevertheless, retaining or enhancing flexibility in configuration is a general strategy. This can be achieved through various means, such as using reprogrammable payloads, e.g. with a suitable bootloader, or connecting as many interfaces as possible. Also, the concepts and strategies for recovery, implementation, and debugging can be used, after adaptation to the specific target platform, for future developments. Furthermore, the adapted SpaceWire light core, plus Linux kernel support, may provide a starting point for upcoming missions for a performant on-board bus system. Additionally, any U-Boot/Linux-based system can employ the boot strategy, memory partitioning, and the approach for handling non-volatile memory failure by keeping it in a read-only state.

For the future it is noted that partial reconfiguration of the FPGA fabric could provide even more interesting opportunities. It enables the configuration of predefined partitions in a design at runtime, while leaving the remaining portions in place. In the OPS-SAT case, considering the operational necessity of always having certain portions in a design available e.g. the synthesised SDRAM controller for the SDRAM recovery, partial reconfiguration would have been a useful strategy. It would have allowed FPGA reconfiguration without requiring a disruptive reconfiguration or a reboot. The version of the Cyclone V used on OPS-SAT does not officially support partial reconfiguration, but in general it is supported by the Cyclone

## 9.2. Conclusion

V FPGA manager. Despite that, it could be worth a test to verify if the functionality is entirely or just partially unsupported. Subsequently, the Linux kernel support for partial reconfiguration of the Cyclone V also needs to be implemented, but there is already similar code for the Stratix-line available.

In summary, a reconfigurable computing platform provides many opportunities for future enhancements. This work only gives a brief and limited overview of the actual possibilities it offers. The range of improvements and solutions will most likely only be limited by the creativity, skill and will of the developers involved. In this sense, it is highly interesting to observe how reconfigurable computing will drive future developments, including but not limited to space applications.



## APPENDIX A

### Kernel SMP Bringup

The SMP bringup code of the Linux kernel was modified, with the more suitable Zynq platform implementation taken as example, to correctly translate to the physical addresses. The code is visible in Listing A.2. It puts the second CPU core into reset, writes a "trampoline code" to address zero, stores the physical secondary startup address in `cpu1startaddr` register of the system manager, and releases the reset. Then the core starts executing the code from address zero.

However, if address zero is mapped to either SDRAM or OCRM, there may not be any valid boot code present. Therefore, the trampoline code in Listing A.1 has been stored at address zero: It queries the system manager register for the (kernel) start address and jumps there. If the bringup code attempts to write the trampoline code to address zero and the bootROM is mapped to that location, the access will be ignored. The bootROM remains unaltered and performs the same address search, followed by a jump to the previously stored address.

Thus, in all cases, the second CPU core is instructed to jump to the start address defined by the kernel, allowing to register itself in the kernel.

## Appendix A. Kernel SMP Bringup

Listing A.1: Modified trampoline code in [72, Version 4.9, arch/arm/mach-socfpga/headsmpl.S]

```
.globl socfpga_cpulstart_addr      // global storage address,
                                   // filled by socfpga.c with system manager address

ENTRY(socfpga_secondary_trampoline)
ARM_BEQ(setend be)
ldr    r1, socfpga_cpulstart_addr // dereference -> system manager mpumodrst
ARM_BEQ(rev    r1, r1)
ldr    r0, [r1]                   // dereference -> Linux start address for second core
ARM_BEQ(rev    r0, r0)
bx     r0                         // jump there

.align
socfpga_cpulstart_addr:
.long  0
ENTRY(socfpga_secondary_trampoline_end)
```

Listing A.2: Modified SMP bringup code in [72, Version 4.9, Line 21, arch/arm/mach-socfpga/platsmp.c]

```
static int socfpga_boot_secondary(unsigned int cpu, struct task_struct *idle)
{
    int trampoline_size = &socfpga_secondary_trampoline_end - &socfpga_secondary_trampoline;

    if (socfpga_cpulstart_addr) {
        static u8 __iomem *zero;

        /* This will put CPU #1 into reset. */
        writel(RSTMGR_MPUMODRST_CPU1,
               rst_manager_base_addr + SOCFPGA_RSTMGR_MODMPURST);

        // if necessary, map memory of start address 0x0
        if ((__pa(PAGE_OFFSET)) {
            zero = ioremap(0, trampoline_size);
            if (!zero) {
                pr_warn("BOOTUP_jump_vectors_not_accessible\n");
                return -1;
            }
        } else {
            zero = (__force u8 __iomem *)PAGE_OFFSET;
        }

        // copy trampoline code (incl. socfpga_cpulstart_addr) to start address 0x0
        memcpy((__force void *) zero, &socfpga_secondary_trampoline, trampoline_size);

        // store physical start address in system manager register "cpulstartaddr"
        writel(virt_to_phys(secondary_startup),
               sys_manager_base_addr + (socfpga_cpulstart_addr & 0x000000ff));

        if ((__pa(PAGE_OFFSET))
            iounmap(zero);

        flush_cache_all();
        smp_wmb();
        outer_clean_range(0, trampoline_size);

        /* This will release CPU #1 out of reset. */
        writel(0, rst_manager_base_addr + SOCFPGA_RSTMGR_MODMPURST);
    }

    return 0;
}
```



# APPENDIX B

## Supplementary Figures

### B.1. Spare Unit's eMMC Erase Counters

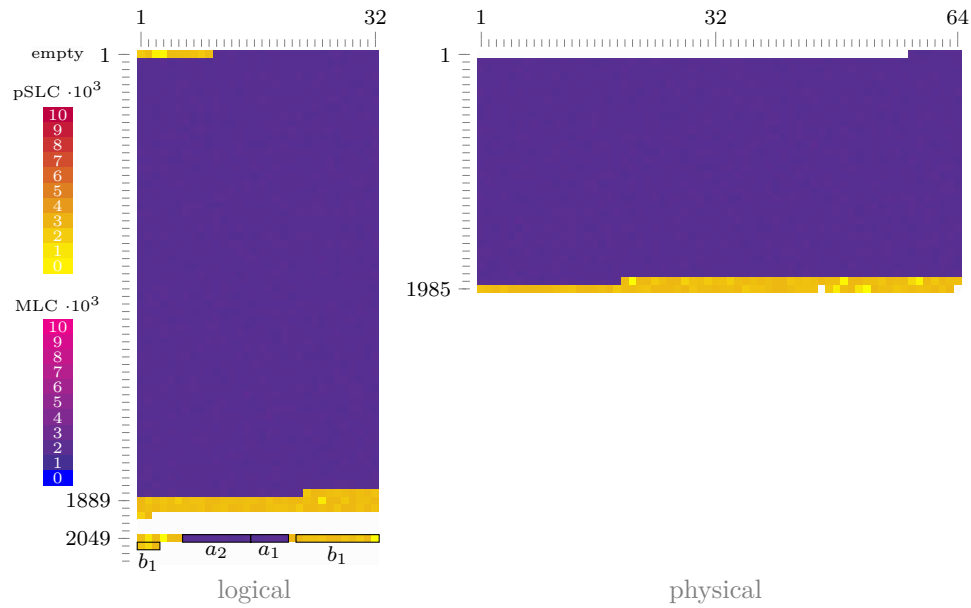


Figure B.1.: Erase counters of the Micron MTFC16GJDEC-4M eMMC on spare model ordered by logical (left) and physical (right) blocks. The data was captured on 09/01/2023 15:15:22.

## B.2. X-band Signal Analysis

In addition to the results in Chapter 6, the captured signal from the X-band transmitter is further analysed here. The in-phase and quadrature components are plotted after filtering and clock synchronisation in Figure B.2. The magnitude of the signal was normalised with an automatic gain control as a preprocessing step and has therefore no unit. For OQPSK, the quadrature component is shifted by half a symbol period, and this shift was compensated for the following diagrams. Effectively, the result is a QPSK modulation.

The signal was sampled with 200 MHz. As the symbol rate is 50 MSym/s, each symbol is sampled four times. Because (part of) the transition is also contained in it, the constellation diagram has at half the transition between two symbols additional point clouds, i.e. at  $(\pm 1/\sqrt{2}, 0)$ ,  $(0, \pm 1/\sqrt{2})$  and  $(0, 0)$ . However, the symbols themselves are clearly distinguishable. The same set of samples is also plotted as an eye diagram in Figure B.3.

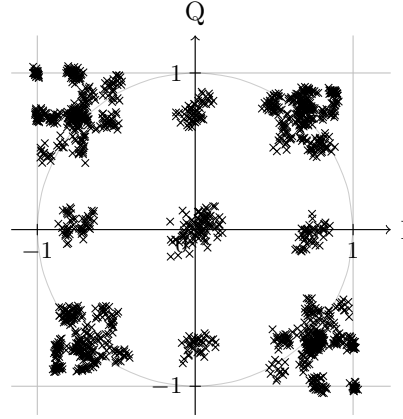


Figure B.2.: Complex samples of the captured signal from the X-band transmitter.

### B.2. X-band Signal Analysis

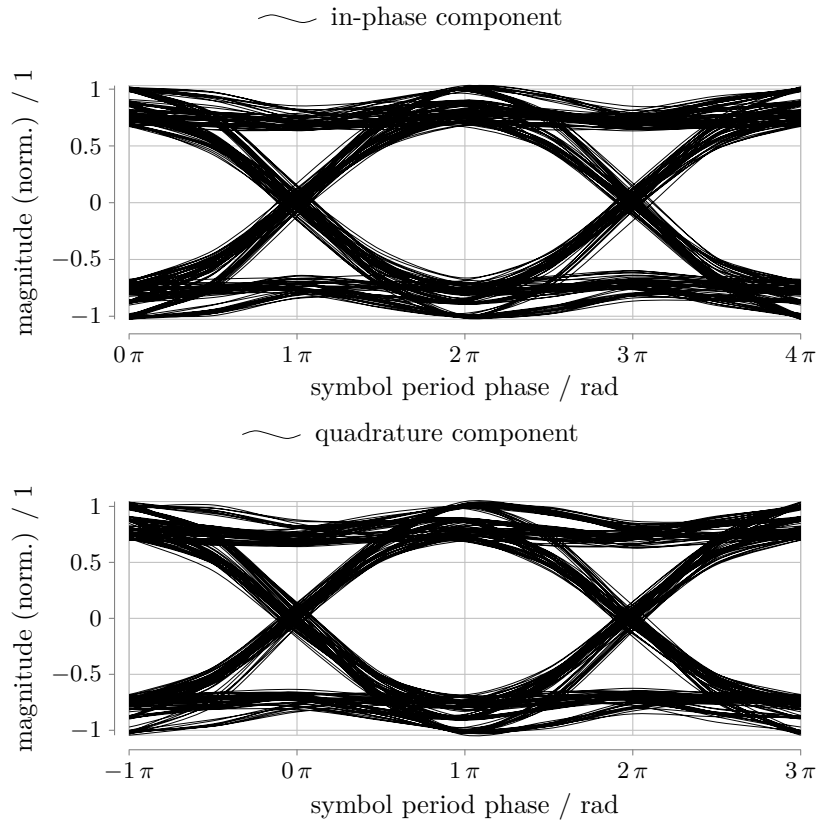


Figure B.3.: Eye diagram of the captured signal from the X-band transmitter. The in-phase and quadrature components are plotted after frequency synchronisation.

### B.3. Full SpaceWire Traces

The autonomous logic capturing mechanism presented in Chapter 8 allowed a detailed analysis of the SpaceWire interface. More than 1500 of such traces could be collected. In addition to the analysis in Chapter 8 two more traces are further analysed here.

The full trace of Figure 8.5 is shown in Figure B.4. The operational interface speed from the processing platform to the CCSDS Engine was by default lowered to 9.1 MHz, due to link drops when operating it with 10 MHz. This is visible in the NULL symbol, consisting of 8 B over 880 ns, resulting in a rate of  $1/(110 \times 10^{-9}) \text{ MHz} = 9.1 \text{ MHz}$ . The handshake phase follows the SpaceWire standard, as it has a 10 MHz clock rate, i.e. a NULL symbol transmitted in 800 ns.

As this initial trace revealed an issue with the reception on the far end, the implementation of the SpaceWire core was adapted and the rate was lowered. Figure B.5 shows the decreased rate of 5 MHz (1.6  $\mu\text{s}$  NULL period) for the handshake period and the running interface. However, for this instance a data character transmission resulted in a link drop, although there were enough FCT's exchanged in advance.

### B.3. Full SpaceWire Traces

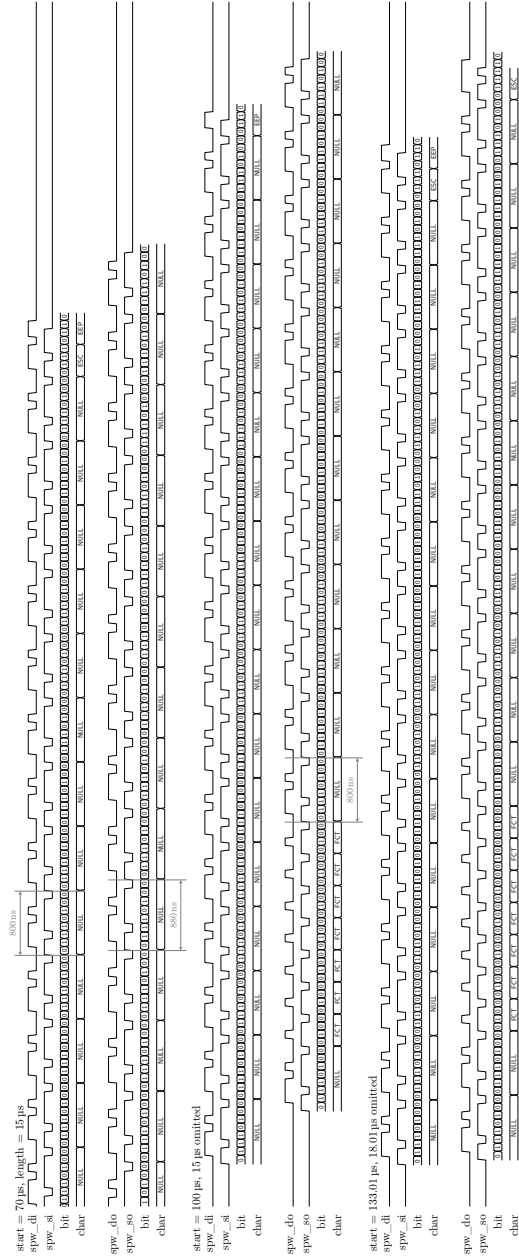


Figure B.4.: SpaceWire logic analyser trace on 16/07/2021 00:37:14 showing the full data of Figure 8.5.

## Appendix B. Supplementary Figures

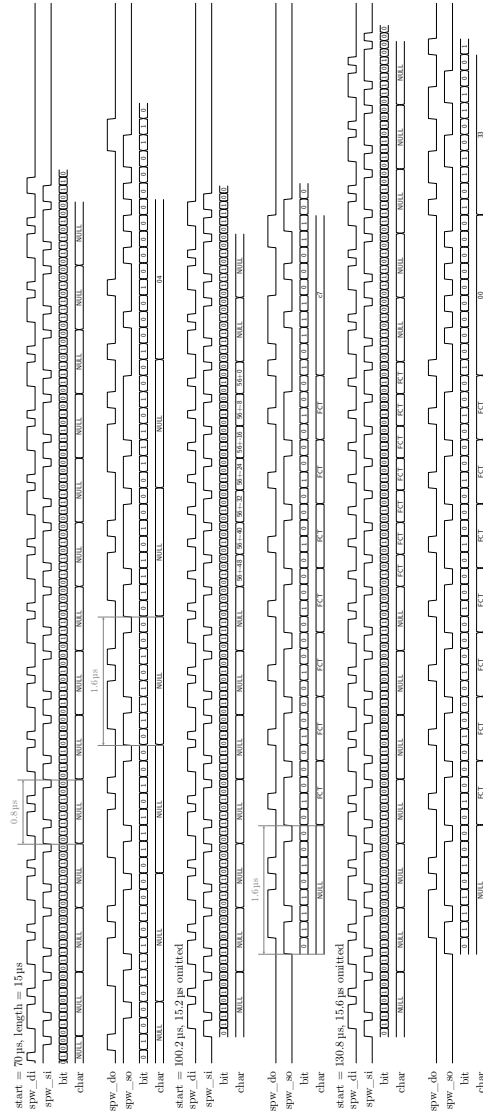


Figure B.5.: SpaceWire logic analyser trace on 29/10/2021 16:42:37

## List of Acronyms

ADC	Analog-to-Digital Converter.
ADCS	Attitude Determination and Control System.
APID	Application Process Identifier.
ASM	Attached Sync Marker.
BSC	Boundary Scan-Cell.
CADU	Channel Access Data Unit.
CAN	Controller Area Network.
CCSDS	Consultative Committee for Space Data Systems.
CFTP	Configurable Fault-Tolerant Processor.
CLTU	Communications Link Transmission Unit.
COTS	Commercial Off-The-Shelf.
CPU	Central Processing Unit.
CRC	Cyclic Redundancy Check.
DAC	Digital-to-Analog Converter.
DDR3 SDRAM	Double Data Rate 3 SDRAM.
DDR3L SDRAM	Double Data Rate 3 Low-power SDRAM.
DMA	Direct Memory Access.
DTB	Device Tree Binary.
ECC	Error Correction Code.
ECSS	European Cooperation for Space Standardization.
EEP	Error End-of-Packet.
EGSE	Electrical Ground Support Equipment.
EMI	Electromagnetic Interference.

*List of Acronyms*

eMMC	embedded MultiMediaCard.
EOP	End-of-Packet.
EPS	Electronic Power System.
ESA	European Space Agency.
ESC	Escape.
ESOC	European Space Operations Center.
F2H	FPGA-to-HPS.
F2S	FPGA-to-SDRAM.
FCT	Flow-Control Token.
FDIR	Fault Detection Isolation and Recovery.
FIT	Flat Image Tree.
FPGA	Field-Programmable Gate Array.
GNSS	Global Navigation Satellite System.
GPS	Global Positioning System.
H2F	HPS-to-FPGA.
HD	High Definition.
HPS	Hard Processing System.
IO	Input/Output.
IP	Intellectual Property.
JTAG	Joint Test Action Group.
LEO	Lower Earth Orbit.
LVDS	Low-Voltage Differential Signalling.
LWH2F	Lightweight HPS-to-FPGA.
MBR	Master Boot Record.
MCT	Mission Control Team.
MO	Mission Operations.
MPU	Microprocessor Unit.
mSGDMA	modular Scatter-Gather Direct Memory Access.
MTU	Maximum Transmission Unit.



*List of Acronyms*

NMF	NanoSat MO Framework.
NPI	Network Partnering Initiative.
OBC	On-Board Computer.
OCRAM	On-Chip Random Access Memory.
P/E	Program/Erase.
PCB	Printed Circuit Board.
PIO	Parallel Input/Output.
PLL	Phase-Locked Loop.
PRBS	Pseudo-Random Binary Sequence.
QSPI	Quad Serial Peripheral Interface.
RAM	Random-Access Memory.
RF	Radio Frequency.
RPMB	Replay Protected Memory Block.
RSSI	Received Signal Strength Indicator.
SD/MMC	Secure Digital/Multimedia Card.
SDR	Software-Defined Radio.
SDRAM	Synchronous Dynamic Random-Access Memory.
SEPP	Satellite's Experimental Processing Platform.
SFCG	Space Frequency Coordination Group.
SLD	System-Level Debugging.
SMP	Symmetric Multiprocessing.
SoC	System-on-Chip.
SPI	Serial Peripheral Interface.
SPL	Secondary Program Loader.
SRAM	Static Random-Access Memory.
TAP	Test Access Port.
TC	Telecommand.
TCP	Transmission Control Protocol.
TID	Total Ionising Dose.

*List of Acronyms*

TM	Telemetry.
UART	Universal Asynchronous Receiver-Transmitter.
UBI	Unsorted Block Images.
UHF	Ultra High Frequency.
UTC	Coordinated Universal Time.

## Contributions

Maximilian Henkel, Andreas Johann Hörmer, David Evans and Manuela Wenger. ‘In-Flight Logic Analysis on OPS-SAT’. 2023. [37]

Maximilian Henkel, Andreas Johann Hörmer, Manuel Kubicka and Reinhard Zeif. ‘Commissioning a Fully-Reconfigurable Communication Chain in a CCSDS-Compliant Way on OPS-SAT Satellite’. 2022. [38]

Maximilian Henkel et al. ‘Recovery From File System Corruption on the OPS-SAT-1 Experimental Processor’. 2023. [39]

Maximilian Henkel et al. ‘Mitigating and Recovering from Radiation Induced Faults in Non-Hardened Spacecraft Flash Memory’. 2024. [40]

Maximilian Henkel. *SpaceWire light driver module for Linux*. 2024. [35]

Maximilian Henkel. *SpaceWire light IP core with Avalon support*. 2024. [36]

David Evans et al. ‘Implementing the New CCSDS Housekeeping Data Compression Standard 124.0-B-1 (based on POCKET+) on OPS-SAT-1’. 2022. [28]

Andreas Johann Hörmer, Maximilian Henkel, Manuel Kubicka, Manuela Wenger and Otto Koudelka. ‘Analysis of Influences of External Components During Vibration Testing of CubeSats’. 2020. [43]

Andreas Johann Hörmer, Manuela Wenger and Maximilian Henkel. ‘Climate research using earth observation cubesats: the PRETTY satellite’. 2023. [44]

Andreas Johann Hörmer et al. ‘The PRETTY CubeSat system redundancy concept’. 2021. [45]

## Contributions

- Manuel Kubicka, Otto Koudelka, Reinhard Zeif, Maximilian Henkel and Andreas Johann Hörmer. ‘A simplified OPS-SAT thermal model to define thermal FDIR strategies’. 2018. [51]
- Manuel Kubicka, Reinhard Zeif, Maximilian Henkel and Andreas Johann Hörmer. ‘Thermal vacuum tests for the ESA’s OPS-SAT mission’. 2022. [53]
- Manuel Kubicka et al. ‘Thermal Vacuum Tests and Thermal Properties on ESA’s OPS-SAT mission’. 2020. [54]
- Dominik Marszk et al. *OPS-SAT in orbit - a technical rundown of this open experimentation platform*. 2021. [58]
- Reinhard Zeif, Andreas Johann Hörmer, Manuel Kubicka, Maximilian Henkel and Otto Koudelka. ‘A GPS Patch Antenna Array for the ESA PRETTY Nanosatellite Mission’. 2020. [79]
- Reinhard Zeif, Andreas Johann Hörmer, Manuel Kubicka, Maximilian Henkel and Otto Koudelka. ‘From OPS-SAT to PRETTY Mission: A Second Generation Software Defined Radio Transceiver for Passive Reflectometry’. 2020. [80]
- Reinhard Zeif et al. ‘The PRETTY Software Defined Radio System and its use as communication platform in space’. 2018. [82]
- Reinhard Zeif et al. ‘The redundancy and fail-safe concept of the OPS-SAT payload processing platform’. 2018. [83]

## Award

International SpaceOps Award for Outstanding Achievements, *17th International Conference on Space Operations, SpaceOps 2023*. Awarded to the OPS-SAT mission control team, as it „created and operates the world’s first mission dedicated to improving space operations, allowing rapid experimentation of mission critical processes on an open, flying laboratory”, on 10/03/2023 in Dubai, United Arab Emirates.

## Bibliography

- [1] Telecommunications Industry Association (TIA). *Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits*. Nov. 2000.
- [2] *1Gb, 3V, Multiple I/O Serial NOR Flash Memory*. rev. N 02/18 EN. Micron Technology. 2011.
- [3] *AN-661: Implementing Fractional PLL Reconfiguration with Altera PLL and Altera PLL Reconfig IP Cores*. 2019.10.14. Intel (Altera) Corporation. 101 Innovation Drive, San Jose, CA 95134, 2019. URL: <https://cdrdv2-public.intel.com/667020/an661-683640-667020.pdf> (visited on 15/03/2020).
- [4] *AN-693: Remote Hardware Debugging over TCP/IP for Altera SoC*. 2015.05.11. Intel (Altera) Corporation. 101 Innovation Drive, San Jose, CA 95134, 2015. URL: [https://cdrdv2-public.intel.com/723699/an\\_693-723698-723699.pdf](https://cdrdv2-public.intel.com/723699/an_693-723698-723699.pdf) (visited on 04/06/2023).
- [5] *AN-709: HPS SoC Boot Guide - Cyclone V SoC Development Kit*. 2015.03.26. Intel (Altera) Corporation. 101 Innovation Drive, San Jose, CA 95134, 2015. URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an7091.pdf> (visited on 07/06/2023).
- [6] *Avalon Interface Specifications*. 2022.01.24. Intel (Altera) Corporation. 101 Innovation Drive, San Jose, CA 95134, 2022. URL: [https://cdrdv2-public.intel.com/667068/mnl\\_avalon\\_spec-683091-667068.pdf](https://cdrdv2-public.intel.com/667068/mnl_avalon_spec-683091-667068.pdf) (visited on 13/11/2023).
- [7] Jayarao (Bob) Balaram, MiMi Aung and Matthew P. Golombek. ‘The Ingenuity Helicopter on the Perseverance Rover’. In: *Space Science Reviews* 217.4 (2021), p. 56. DOI: 10.1007/s11214-021-00815-w.

## Bibliography

- [8] Jayarao (Bob) Balaram et al. ‘Mars Helicopter Technology Demonstrator’. In: *Proceedings of the AIAA atmospheric flight mechanics conference*. 2018, p. 0023. DOI: 10.2514/6.2018-0023.
- [9] Michael Bergmann et al. ‘OPS-SAT Phase A/B1 - Mission Requirements Document’. unpublished. Dec. 2013.
- [10] Matthias Binder et al. ‘OPS-SAT Phase B2/C/D/E1 - Statement of Work’. unpublished. Sept. 2014.
- [11] Matthias Binder et al. ‘OPS-SAT Phase B2/C/D/E1 - System Design Report’. unpublished, reduced version publicly available. June 2019. URL: [https://hackopssat.cysat.eu/wp-content/uploads/2021/12/OPSSAT-SYS-SDR\\_v4.4\\_Excerpt\\_v1.0\\_PUBLIC.pdf](https://hackopssat.cysat.eu/wp-content/uploads/2021/12/OPSSAT-SYS-SDR_v4.4_Excerpt_v1.0_PUBLIC.pdf).
- [12] Altera Corporation (published by Brian Yates). *Source code repository of the mmlink application*. URL: [https://github.com/byates/altera\\_mmlink](https://github.com/byates/altera_mmlink) (visited on 08/10/2022).
- [13] Altera Corporation (published by Brian Yates). *Source code repository of the sld-hub kernel module*. URL: [https://github.com/byates/altera\\_sld\\_hub\\_driver](https://github.com/byates/altera_sld_hub_driver) (visited on 08/10/2022).
- [14] Michael Caffrey et al. ‘On-Orbit Flight Results from the Reconfigurable Cibola Flight Experiment Satellite (CFESat)’. In: *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. 2009, pp. 3–10. DOI: 10.1109/FCCM.2009.22.
- [15] Marti Farras Casas, Steve Parkes, Albert Ferrer Florit and Alberto Gonzalez Villafranca. ‘Testing SpaceFibre in Orbit: the OPS-SAT and NORBY Technology Demonstrators’. In: *2022 International SpaceWire & SpaceFibre Conference (ISC)*. 2022, pp. 1–3.
- [16] César Coelho, Otto Koudelka and Mario Merri. ‘NanoSat MO framework: When OBSW turns into apps’. In: *2017 IEEE Aerospace Conference*. 2017, pp. 1–8. DOI: 10.1109/AERO.2017.7943951.
- [17] *collectd – The system statistics collection daemon*. URL: <https://collectd.org> (visited on 16/11/2023).
- [18] *Cyclone V Device Overview*. 2018.05.07. Intel (Altera) Corporation. 101 Innovation Drive, San Jose, CA 95134, 2018. URL: [https://cdrdv2-public.intel.com/666729/cv\\_51001-683694-666729.pdf](https://cdrdv2-public.intel.com/666729/cv_51001-683694-666729.pdf) (visited on 27/03/2024).

- [19] Anwar S. Dawood, Stephen J. Visser and John A. Williams. ‘Reconfigurable FPGAs for real time image processing in space’. In: *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)*. Vol. 2. 2002, 845–848 vol.2. DOI: 10.1109/ICDSP.2002.1028222.
- [20] Anwar S. Dawood, John A. Williams and Stephen J. Visser. ‘Flexible real time signal filtering in space using reconfigurable logic’. In: *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)*. Vol. 2. 2002, 849–852 vol.2. DOI: 10.1109/ICDSP.2002.1028223.
- [21] Anwar S. Dawood, John A. Williams and Stephen J. Visser. ‘On-board satellite image compression using reconfigurable FPGAs’. In: *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.* 2002, pp. 306–310. DOI: 10.1109/FPT.2002.1188698.
- [22] *Cyclone V Device Handbook*. Vol. 1: *Device Interfaces and Integration*. 2023.10.18. 101 Innovation Drive, San Jose, CA 95134, 2023. URL: [https://cdrdv2-public.intel.com/666995/cv\\_5v2-683375-666995.pdf](https://cdrdv2-public.intel.com/666995/cv_5v2-683375-666995.pdf) (visited on 27/03/2024).
- [23] Dean A. Ebert, Charles A. Hulme, Herschel H. Loomis and Alan A. Ross. ‘Configurable Fault-Tolerant Processor (CFTP) for Space Based Applications’. In: *Proceedings of the AIAA/USU Conference on Small Satellites*. SSC03-XI-5. Aug. 2003. URL: <https://digitalcommons.usu.edu/smallsat/2003/A112003/79/>.
- [24] *Efficient Spectrum Utilisation for Space Research Service (Category A) and Earth Exploration-Satellite Service on Space-to-Earth Links*. Recommendation 21-2R5. Space Frequency Coordination Group (SFCG). July 2022.
- [25] *Embedded MultiMediaCard (eMMC) eMMC/Card Product Standard, High Capacity, including Reliable Write, Boot, and Sleep Modes. MMCA 4.3*. 2500 Wilson Boulevard, Arlington, Virginia, USA: JEDEC Solid State Technology Association, Nov. 2007.

## Bibliography

- [26] *Embedded Peripherals IP User Guide*. 2018.09.24. Intel (Altera) Corporation. 101 Innovation Drive, San Jose, CA 95134, 2018. URL: [https://cdrdv2-public.intel.com/704751/ug\\_embedded\\_ip-18-1-683130-704751.pdf](https://cdrdv2-public.intel.com/704751/ug_embedded_ip-18-1-683130-704751.pdf) (visited on 11/01/2023).
- [27] David J. Evans. ‘OPS-SAT: Operational Concept for ESA’s First Mission Dedicated to Operational Technology’. In: *SpaceOps 2016 Conference*. 2016. DOI: 10.2514/6.2016-2354.
- [28] David Evans et al. ‘Implementing the New CCSDS Housekeeping Data Compression Standard 124.0-B-1 (based on POCKET+) on OPS-SAT-1’. In: *Proceedings of the AIAA/USU Conference on Small Satellites*. Communications, SSC22-XII-03. Aug. 2022. URL: <https://digitalcommons.usu.edu/smallsat/2022/all2022/133/>.
- [29] Frédéric Férésin et al. ‘In space image processing using AI embedded on system on module: example of OPS-SAT cloud segmentation’. In: *Proceedings of the 2nd European Workshop on On-Board Data Processing (OBDP2021)*. Zenodo, June 2021. DOI: 10.5281/zenodo.5574960.
- [30] Agustin Fernández-León, André-Louis Pouponnot and Sandi Habinc. ‘ESA FPGA Task Force: Lessons Learned’. In: *Military and Aerospace Programmable Logic Device (MAPLD) International Conference*. 2002. URL: [http://microelectronics.esa.int/asic/FPGA\\_Task\\_Force\\_MAPLD2002.pdf](http://microelectronics.esa.int/asic/FPGA_Task_Force_MAPLD2002.pdf).
- [31] Miguel Angel Fernandez et al. ‘Game-changing radio communication architecture for Cube/Nano satellites’. In: *Proceedings of the AIAA/USU Conference on Small Satellites*. Aug. 2015. URL: <https://digitalcommons.usu.edu/smallsat/2015/all2015/94>.
- [32] Simone Fratini, Nicola Policella, Ricardo Silva and Joao Guerreiro. ‘On-board autonomy operations for OPS-SAT experiment’. In: *Applied Intelligence - The International Journal of Research on Intelligent Systems for Real Life Complex Problems* (2022), pp. 6970–6987. DOI: 10.1007/s10489-020-02158-5.
- [33] Raymond Galik. ‘Space Test Program-1 (STP-1) – First of its kind!’ In: *Proceedings of the AIAA/USU Conference on Small Satellites*. Technical Session I. 2004. URL: <https://digitalcommons.usu.edu/smallsat/2004/All2004/7>.



- [34] *Cyclone V Device Handbook*. Vol. 3: *Hard Processor System Technical Reference Manual*. 2015.11.02. 101 Innovation Drive, San Jose, CA 95134, 2015. URL: [https://cdrdv2-public.intel.com/705351/cv\\_5v4-17-1-683126-705351.pdf](https://cdrdv2-public.intel.com/705351/cv_5v4-17-1-683126-705351.pdf) (visited on 27/03/2024).
- [35] Maximilian Henkel. *SpaceWire light driver module for Linux*. 2024. DOI: 10.3217/8wdyv-a4p08.
- [36] Maximilian Henkel. *SpaceWire light IP core with Avalon support*. Version 0.3. 2024. DOI: 10.3217/gcr0r-pd431.
- [37] Maximilian Henkel, Andreas Johann Hörmer, David Evans and Manuela Wenger. ‘In-Flight Logic Analysis on OPS-SAT’. In: *Proceedings of the 17th International Conference on Telecommunications (ConTEL)*. 2023. DOI: 10.1109/ConTEL58387.2023.10199036.
- [38] Maximilian Henkel, Andreas Johann Hörmer, Manuel Kubicka and Reinhard Zeif. ‘Commissioning a Fully-Reconfigurable Communication Chain in a CCSDS-Compliant Way on OPS-SAT Satellite’. In: *Proceedings of the International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*. 2022. DOI: 10.1109/CoBCom55489.2022.9880813.
- [39] Maximilian Henkel et al. ‘Recovery From File System Corruption on the OPS-SAT-1 Experimental Processor’. In: *Proceedings of the AIAA/USU Conference on Small Satellites*. Weekend Poster Session 2. Aug. 2023. URL: <https://digitalcommons.usu.edu/smallsat/2023/all2023/263>.
- [40] Maximilian Henkel et al. ‘Mitigating and Recovering from Radiation Induced Faults in Non-Hardened Spacecraft Flash Memory’. In: *2024 IEEE Aerospace Conference*. Accepted for publication, forthcoming. 2024.
- [41] Alexander Hofmann, Robert Glein, Leo Frank, Rainer Wansch and Albert Heuberger. ‘Reconfigurable on-board processing for flexible satellite communication systems using FPGAs’. In: *2017 Topical Workshop on Internet of Space (TWIOS)*. 2017, pp. 1–4. DOI: 10.1109/TWIOS.2017.7869767.

## Bibliography

- [42] Alexander Hofmann, Rainer Wansch, Rob  rt Glein and Bernd Kollmannthaler. ‘An FPGA based on-board processor platform for space application’. In: *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2012, pp. 17–22. DOI: 10.1109/AHS.2012.6268653.
- [43] Andreas Johann H  rmer, Maximilian Henkel, Manuel Kubicka, Manuela Wenger and Otto Koudelka. ‘Analysis of Influences of External Components During Vibration Testing of CubeSats’. In: *Proceedings of the 70th International Astronautical Congress*. 2020. URL: <https://dl.iafastro.directory/event/IAC-2019/paper/51008/>.
- [44] Andreas Johann H  rmer, Manuela Wenger and Maximilian Henkel. ‘Climate research using earth observation cubesats: the PRETTY satellite’. In: *Proceedings of the Global Space Conference on Climate Change (GLOC 2023), Oslo, Norway*. May 2023. URL: <https://dl.iafastro.directory/event/GLOC-2023/paper/75112/>.
- [45] Andreas Johann H  rmer et al. ‘The PRETTY CubeSat system redundancy concept’. In: *Proceedings of the 72nd International Astronautical Congress, Dubai, UAE*. Oct. 2021. URL: <https://dl.iafastro.directory/event/IAC-2021/paper/65186/>.
- [46] *IEEE Standard for Heterogeneous InterConnect (HIC), (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)*. IEEE, 1996. DOI: 10.1109/IEEESTD.1996.81004.
- [47] *IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture*. IEEE, 2010. DOI: 10.1109/IEEESTD.2010.5412866.
- [48] *Intel Quartus Prime Pro Edition User Guide: Debug Tools*. 2022.12.12. Intel (Altera) Corporation. 2022. URL: <https://cdrdv2-public.intel.com/666789/ug-qpp-debug-683819-666789.pdf> (visited on 01/06/2023).
- [49] Shreeyam Kacker, Alex Meredith, Kerri Cahoy and Georges Labreche. ‘Machine learning image processing algorithms onboard OPS-SAT’. In: (Aug. 2022). URL: <https://digitalcommons.usu.edu/smallsat/2022/all2022/65/>.

- [50] Otto Koudelka, Reinhard Zeif, Andreas Johann Hörmer and Maximilian Henkel. ‘OPS-SAT Phase B2/C/D/E1 - System Level – RF Radio Test – Plan and Report’. unpublished. Oct. 2019.
- [51] Manuel Kubicka, Otto Koudelka, Reinhard Zeif, Maximilian Henkel and Andreas Johann Hörmer. ‘A simplified OPS-SAT thermal model to define thermal FDIR strategies’. In: *Proceedings of the 69th International Astronautical Congress*. Vol. 7. 2018, pp. 5035–5041. URL: <https://dl.iafastro.directory/event/IAC-2018/paper/47923/>.
- [52] Manuel Kubicka, Reinhard Zeif and Maximilian Henkel. ‘OPS-SAT Satellite Fact Sheet’. In: TU Graz - IKS - Poster Wall, Poster displayed at the Institute. Jan. 2022.
- [53] Manuel Kubicka, Reinhard Zeif, Maximilian Henkel and Andreas Johann Hörmer. ‘Thermal vacuum tests for the ESA’s OPS-SAT mission’. In: *Elektrotechnik und Informationstechnik* 139.1 (Feb. 2022), pp. 16–24. ISSN: 0932-383X. DOI: 10.1007/s00502-022-00990-w.
- [54] Manuel Kubicka et al. ‘Thermal Vacuum Tests and Thermal Properties on ESA’s OPS-SAT mission’. In: *Proceedings of the 3rd International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom 2020)*. Aug. 2020. DOI: 10.1109/CoBCom49975.2020.9174095.
- [55] Piotr Kuligowski and Michał Kuklewski. ‘OPS-SAT CCSDS Engine Interface Control Document (ICD)’. unpublished; kontakt@creotech.pl. Feb. 2019.
- [56] Georges Labrèche et al. ‘OPS-SAT Spacecraft Autonomy with TensorFlow Lite, Unsupervised Learning, and Online Machine Learning’. In: *2022 IEEE Aerospace Conference (AERO)*. 2022, pp. 1–17. DOI: 10.1109/AERO53065.2022.9843402.
- [57] André Løfaldli et al. ‘ESOC Ground Station Architecture for OPS-SAT’. In: *8th International Workshop on Tracking, Telemetry and Command Systems for Space Applications (TTC)*. 2019, pp. 1–3. DOI: 10.1109/TTC.2019.8895329.
- [58] Dominik Marszk et al. *OPS-SAT in orbit - a technical rundown of this open experimentation platform*. Paper presented at the Data Systems in Aerospace (DASIA) conference. Sept. 2021.

## Bibliography

- [59] *Mission Operations Monitor & Control Services*. 522.1-B-1. Recommendation for Space Data System Standards. Consultative Committee for Space Data Systems (CCSDS). Washington, DC, USA, Oct. 2017. URL: <https://public.ccsds.org/Pubs/522x1b1.pdf>.
- [60] *Mission Operations Services Concept*. ser. Green Book, No. 3. Informational Report. Washington, DC, USA: Consultative Committee for Space Data Systems (CCSDS), Dec. 2010. URL: <https://public.ccsds.org/Pubs/520x0g3.pdf>.
- [61] Tom Mladenov, David Evans and Vladimir Zelenevskiy. ‘Implementation of a GNU Radio-Based Search and Rescue Receiver on ESA’s OPS-SAT Space Lab’. In: *IEEE Aerospace and Electronic Systems Magazine* 37.5 (2022), pp. 4–12. DOI: 10.1109/MAES.2022.3143875.
- [62] Lorenzo Ortega, Roberto Camarero and Antoine Ressouche. ‘Lossless Image Compression on the EYESAT Nanosat’. In: *Proceedings of the 5th International Workshop on On-Board Payload Data Compression (OBPDC 2016)*. Sept. 2016. URL: [https://www.researchgate.net/publication/312197460\\_Lossless\\_Image\\_Compression\\_on\\_the\\_EYESAT\\_Nanosat](https://www.researchgate.net/publication/312197460_Lossless_Image_Compression_on_the_EYESAT_Nanosat).
- [63] Joris van Rantwijk. *SpaceWire Light*. Version 20130504. URL: [https://github.com/freecores/spacewire\\_light/blob/master/doc/Manual.pdf](https://github.com/freecores/spacewire_light/blob/master/doc/Manual.pdf) (visited on 27/03/2024).
- [64] Florian Rittner, Rob  rt Glein, Thomas Kolb and Benjamin Bernard. ‘Broadband FPGA payload processing in a harsh radiation environment’. In: *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2014, pp. 151–158. DOI: 10.1109/AHS.2014.6880171.
- [65] S. Salas et al. ‘ANGELS SmallSat: Demonstrator for new French product line’. In: *SpaceOps 2018 Conference*. AIAA, 2018. DOI: 10.2514/6.2018-2731.
- [66] *Space product assurance - Techniques for radiation effects mitigation in ASICs and FPGAs handbook*. ECSS-Q-HB-60-02A. ESA Requirements and Standards Division, ESTEC, P.O. Box 299, 2200 AG Noordwijk, The Netherlands, Sept. 2016.

- [67] *SpaceWire - Links, nodes, routers and networks*. ECSS-E-ST-50-12C. ECSS Secretariat - ESA-ESTEC Requirements and Standards Division. ESTEC, P.O. Box 299, 2200 AG Noordwijk, The Netherlands, July 2008.
- [68] *SpaceWire Handbook*. Suite E U 2, Bletchley Park, Milton Keynes, MK3 6EB, United Kingdom, 2013. URL: <http://spacewire.esa.int/WG/SpaceWire/SpW-WG-Mtg20-Proceedings/4Links-SpaceWire-Handbook-Draft-20130410.pdf> (visited on 24/03/2024).
- [69] Mindy Surratt, Herschel H. Loomis, Alan A. Ross and Russ Duren. ‘Challenges of Remote FPGA Configuration for Space Applications’. In: *2005 IEEE Aerospace Conference*. 2005, pp. 1–9. DOI: 10.1109/AERO.2005.1559549.
- [70] *TC Synchronization and Channel Coding*. 231.0-B-4. Recommendation for Space Data System Standards. Consultative Committee for Space Data Systems (CCSDS). Washington, DC, USA, July 2021. URL: <https://public.ccsds.org/Pubs/231x0b4e0.pdf>.
- [71] *TM Synchronization and Channel Coding*. 131.0-B-3. Recommendation for Space Data System Standards. Consultative Committee for Space Data Systems (CCSDS). Washington, DC, USA, Sept. 2017. URL: <https://public.ccsds.org/Pubs/131x0b3e1.pdf>.
- [72] Linus Torvalds, ed. *Linux (4.9) [Operating system]*. URL: <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git>.
- [73] *U-Boot mailing list discussion regarding the Cyclone V FPGA-to-SDRAM bridge configuration*. URL: <https://lists.denx.de/pipermail/u-boot/2020-February/399385.html> (visited on 16/10/2023).
- [74] *U-Boot source code*. URL: <https://source.denx.de/u-boot/u-boot/> (visited on 20/06/2023).
- [75] *UBIFS - UBI File-System*. URL: <http://www.linux-mtd.infradead.org/doc/ubifs.html> (visited on 20/06/2023).
- [76] *Virtual JTAG Intel FPGA IP Core User Guide*. 2021.08.12. Intel (Altera) Corporation. 2021. URL: [https://cdrdv2-public.intel.com/666577/ug\\_virtualjtag-683705-666577.pdf](https://cdrdv2-public.intel.com/666577/ug_virtualjtag-683705-666577.pdf) (visited on 15/03/2023).

## Bibliography

- [77] Stephen J. Visser, Anwar S. Dawood and John A. Williams. ‘FPGA based real-time adaptive filtering for space applications’. In: *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.* 2002, pp. 322–326. DOI: 10.1109/FPT.2002.1188702.
- [78] Christopher Wilson et al. ‘CSP Hybrid Space Computing for STP-H5/ISEM on ISS’. In: *Proceedings of the AIAA/USU Conference on Small Satellites*. Technical Session III: Next on the Pad. 2015. URL: <https://digitalcommons.usu.edu/smallsat/2015/all2015/21>.
- [79] Reinhard Zeif, Andreas Johann Hörmer, Manuel Kubicka, Maximilian Henkel and Otto Koudelka. ‘A GPS Patch Antenna Array for the ESA PRETTY Nanosatellite Mission’. In: *Proceedings of the 3rd International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom 2020)*. July 2020. DOI: 10.1109/CoBCom49975.2020.9173975.
- [80] Reinhard Zeif, Andreas Johann Hörmer, Manuel Kubicka, Maximilian Henkel and Otto Koudelka. ‘From OPS-SAT to PRETTY Mission: A Second Generation Software Defined Radio Transceiver for Passive Reflectometry’. In: *Proceedings of the 3rd International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom 2020)*. July 2020. DOI: 10.1109/CoBCom49975.2020.9174103.
- [81] Reinhard Zeif, Manuel Kubicka and Andreas Hörmer. ‘Development and application of an embedded computer system for CubeSats exemplified by the OPS-SAT space mission’. In: *e & i Elektrotechnik und Informationstechnik* (Feb. 2022). DOI: 10.1007/s00502-022-00991-9.
- [82] Reinhard Zeif et al. ‘The PRETTY Software Defined Radio System and its use as communication platform in space’. In: *Proceedings of the 69th International Astronautical Congress, Bremen, Germany*. Oct. 2018. URL: <https://dl.iafastro.directory/event/IAC-2018/paper/47499/>.
- [83] Reinhard Zeif et al. ‘The redundancy and fail-safe concept of the OPS-SAT payload processing platform’. In: *Proceedings of the 69th International Astronautical Congress*. 2018. URL: <https://dl.iafastro.directory/event/IAC-2018/paper/47503/>.

## *Bibliography*

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Graz University of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to [http://www.ieee.org/publications\\_standards/publications/rights/rights\\_link.html](http://www.ieee.org/publications_standards/publications/rights/rights_link.html) to learn how to obtain a License from RightsLink.