Project CubeCAT

High-Speed Data Path for a Laser Communication Terminal

Building a 100 Mbit/s Laser Communication Terminal for CubeSats

Alexander Matthijs van Leeuwen

Delft University of Technology

TU Delft

HYPERION TECHNOLOGIES

High-Speed Data Path for a Laser Communication Terminal

Building a 100 Mbit/s Laser Communication Terminal for CubeSats

by

Alexander Matthijs van Leeuwen

to obtain the degree of Master of Science
in Computer Engineering
at the Delft University of Technology,

to be defended publicly on Monday July 20, 2020 at 10:00.

Student number:	4351436	
Project duration:	February, 2019 – June, 2020	
Thesis committee:	Dr. ir. J. S. S. M. Wong,	TU Delft, supervisor
	Dr. ir. A. J. van Genderen,	TU Delft
	Dr. ir. C. J. M. Verhoeven,	TU Delft
	Dr. ir. S. Engelen	Hyperion Technologies, daily supervisor

This thesis is confidential and cannot be made public until July 20, 2022.

TU Delft Delft University of Technology

Abstract

In the recent years the satellite industry has progressed on the subject of optical communication for use in space. With recorded speeds over 5 Gb/s it has shown to be an alternative for radio communication. In the CubeSat market this technology is new, underdeveloped, and could lead to new missions that were not possible before. As such, TNO and Hyperion joined forces to create the CubeCAT LCT (Laser Communication Terminal). The core part of this LCT is the high-speed digital data path, which was not implemented. This thesis discusses the design, implementation, and verification of the high-speed data path of the CubeCAT LCT (Laser Communication Terminal) that has a targeted speed of 100 Mb/s, with a future upgrade path to 1 Gb/s.

The CubeCAT module consists of multiple modules, of which the DMU (Data Management Unit) hosts the high-speed data path. As the DMU was not implemented, a design for the DMU is proposed in this thesis. For this design multiple architectures, interconnects, and components where considered. The proposed design is based on an Hyperion CP400.85 microprocessor connected to a Lattice ECP5-5G FPGA, together with extra external memory and external storage.

Then, an implementation of the high-speed data path was made that is based on a QSPI link between the microprocessor and the FPGA. This implementation is based on streaming the data from the microprocessor to the FPGA, in which the data is encoded according to the TNO3k FEC (Forward Error Correction) scheme. After encoding, the data is outputted as an LVDS signal to the laser output.

The implementation of the high-speed data path was verified by simulation and on a development setup. This was done by first verifying all submodules, with focus on the QSPI link and the TNO3k encoder, and then as a whole system. All submodule tests were successful, with a note to the verification of the QSPI link. It was found that the development setup was limited to a SPI frequency of 41.50 MHz due to signal integrity issues.

During the system test it was found that the there was a lack of LVDS test material. As such the LVDS output was replaced by a UART output. With this output the whole system has been validated for a QSPI link speed of 119.2 Mb/s and an internal FPGA speed of 3.2 Gb/s. The system, with LVDS output, is estimated to consume 1 watts of power.

With the validation of the whole system, the high-speed data path of the CubeCAT LCT has been implemented. The design of the DMU allows for a later, 1 Gb/s upgrade of the high-speed data path.

Preface

With this thesis my time at Hyperion comes to an end. Almost one and a half years ago I started there, working with colleagues. Now when I look back I'll leave friends there.

My (almost) one and a half years at Hyperion were not always spent on working on this thesis project. In the meantime I've worked on two other projects, delaying the finalisation of this thesis by about three months. Besides, I've mingled in with the system administrators at Hyperion, as the setup was, in my opinion, in dire need of an update. This side-track helped me acquire some skills that I otherwise wouldn't have picked up. At other times I spent time helping colleagues with their code, as I couldn't resist their call for help.

With this big delay and unclear ending date I think I've stressed out my girlfriend Sofie a bit, as she hopes to buy a house, which is impossible on a thesis compensation. My sincere apologies for that and I thank her very much for the patience that you had.

Other people that I'd like to thank are my parents for helping me get where I am now. Then I'd like to thank my friends for their support and motivation, without them the thesis journey would have been significantly harder.

Additionally, I'd like to thank the people at Hyperion and the people on the TNO CubeCAT side for offering me the chance to gain experience my time at Hyperion. My supervisor, Steven, and I had some great discussions about computing technology, making me wonder if he really did study Aerospace Engineering. Thank you Steven for these discussions and for your guidance.

Lastly, I'd like to thank Stephan Wong, my professor, for guiding me in the thesis process especially when I was getting lost halfway through.

Sander van Leeuwen
Oegstgeest, July 2020

Table of Contents

List of Figures

List of Tables

Glossary

1U The standard unit (U) of a CubeSat is 10 cm x 10 cm x 10 cm. xv, 1, 2, 5–7, 12, 13, 16, 51

API A specification of a system interface, to be used by other systems. 28, 39

back-pressure The concept where a receiver needs the transmitter to pause since the receiver is still processing its data. 67

bit one binary element. iii, 1, 2, 5, 7, 9–13, 16, 17, 20–22, 24–26, 29, 32–34, 36–39, 44, 46–49, 51–53, 62–64

byte eight binary elements. xv, 20, 21, 25, 27, 28, 33–37, 44–46, 62–64

CCSDS is an international organization that is focused on the development of communication and data systems standards for spaceflight. Members include ESA, NASA and JAXA. 9, 10, 27, 61

constellation a group/ pack of satellites. 5

CubeCAT A Laser Communication Terminal being developed by Hyperion and TNO. iii, v, 1–3, 5–13, 15, 16, 24–26, 29, 51–53

CubeSat A satellite built around standardized size modules. See also 1U. iii, xv, 1, 2, 5–7, 11–13, 23, 51

GEO an orbit in which a satellite, relative to earth, stays fixed at on position, for example a TV broadcasting satellite. The distance relative to Earth is around 35700 km.. 6, 11

latch-up is an electrical event where a line is stuck at a (logic) level. In the case of a non-permanent latchup the solution is a reset (power cycle) of the circuitry. 23

LEO an orbit in which a satellite moves around Earth, for example the ISS (International Space Station). The distance relative to Earth is 2000 km or less.. 1, 6–8, 10–12, 17, 23, 26

NewSpace the change of the space sector toward more private companies that develop faster and provide cheaper access to spaceflight. 1

PCIe is a high-speed interconnect on computing systems. On version 2.0, with a x1 link the maximum data throughput is 500 MB/s. Compared to version 2.0, 3.0 supports a 984.6 MB/s maximum data throughput. 19, 20, 22–25, 65

PPM a signal modulation technique. 10, 64

RS-485 a serial protocol that can be used to communicate with multiple nodes over a bus topology. 7, 15, 16, 25, 27, 28

SEL an undesired, state change of the circuitry where the state remains. A power-cycle can fix non-permanent latchups. 6, 13, 51

SEU an undesired state change of circuitry due to a radioactive particle hitting the circuitry. 6, 13, 24, 25, 51

SmallSat A satellite with a total weight of (usually) less than 500 kg. 3, 5, 13, 51

SSD an SSD is a type of storage that is build around multiple parallel flash banks, providing high data rates and good random read write performance due to no mechanical limitations. 25

TID the total dose of radiation accumulated over time that the device can handle before stopping to function. 6, 13, 51

TNO Is a Dutch research organization focussing on applied science. iii, xv, 1, 2, 6, 7, 10, 38, 52, 53, 63

validation The process of checking a system where the primary question is ”Are we building the right system?”[1]. See also verification. xvi, 48, 49, 52

verification The process of checking a system where the primary question is ”Are we building the system right?”[1]. See also validation. xvi, 3, 41, 42, 44, 45, 49, 52

Verilog a Hardware Description Language. 65

Acronyms

ADC Analog to Digital Converter. 19

BER Bit Error Rate. 9, 61

CRC Cyclic Redundancy Check. 36, 37, 45, 53, 62, 63

DDR Double Data Rate. 20, 21, 24

DLR Deutsches Zentrum für Luft- und Raumfahrt. 10, 11

DMA Direct Memory Access. 23, 28, 39, 44, 51

DMU Data Management Unit. iii, 8, 9, 15, 16, 23, 25, 26, 28, 29, 31, 41, 48, 49, 51–53

EBI External Bus Interface. 19, 21, 23, 24, 26–29, 39, 51, 53

ECC Error-Correcting Codes. 6, 23, 25

eMMC embedded MultiMediaCard. 25, 29, 49, 51

ESA European Space Agency. 10, 11

FD-SOI Fully Depleted Silicon On Insulator. 23

FEC Forward Error Correction. iii, 2, 5, 7–10, 52, 61, 62, 67

FIFO First-In, First-Out. 33–35, 38, 48

FPGA Field Programmable Gate Array. iii, 3, 5, 17–29, 31, 32, 34, 35, 38, 39, 41, 44–49, 51–53, 64, 65

FSM Fine Steering Mirror. 8, 12

HDL Hardware Description Language. xvi, 31, 36, 41, 42, 65, 67

IP Intellectual Property. 31, 33, 34, 38, 65

ISS International Space Station. xv

JPL Jet Propulsion Laboratory. 6

LCT Laser Communication Terminal. iii, xv, 1, 2, 5, 7, 11–13, 23, 51, 53

LDPC Low-Density Parity Check. 10

LSE Lattice Synthesis Engine. 31

LVDS Low-Voltage Differential Signaling. iii, 9, 15–18, 20, 24, 25, 27, 29, 32, 38, 48, 49, 51, 52

MMC MultiMediaCard. 25

NASA National Aeronautics and Space Administration. 5, 6, 10, 12

NVMe Non-Volatile Memory Express. 22

OBC On-Board Computer. 24

OS Operating System. 18

OSI Open Systems Interconnection. 65

QSPI Quad SPI. iii, 19–21, 23–29, 31, 34, 35, 38, 39, 41, 42, 44, 46, 47, 49, 51–53

RAM Random Access Memory. 63, 64

REST Representational State Transfer. 28

RS Reed-Solomon. 36

SD Secure Digital. 19, 20, 24, 25

SDR Single Data Rate. 20

SerDes Serializer/Deserializer. 38

SoC System on a Chip. 24

SoM System on Module. 23, 24, 65

SPI Serial Peripheral Interface. iii, 19–21, 35, 44, 48, 53

UART Universal Asynchronous Receiver-Transmitter. iii, 28, 33, 41, 45, 48

UFS Universal Flash Storage. 25

USB Universal Serial Bus. 7, 15, 16, 19, 21–25, 27, 28

VHDL VHSIC-HDL (Very High Speed Integrated Circuit Hardware Description Language). 31, 37, 41, 42, 49, 52, 62, 65

1

Introduction

You grab your phone and take a picture of a famous bridge. With 4 taps the picture is sent to a friend and within an instant you have a reply with a picture of his cat. All this is made possible by the fast, land-based internet infrastructure. This is different for satellites, especially satellites in Low-Earth Orbit (LEO). Satellites in LEO fly over a coordinate every 90 to 120 minutes and can only communicate when a ground station is in 'sight'. Therefore, when a ground station is in sight it will only have a few minutes to send all stored data, as is illustrated by Figure 1.1.

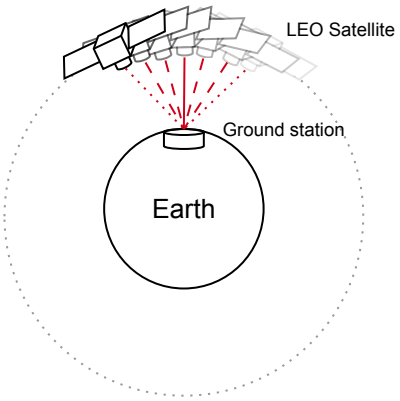


Figure 1.1: LEO orbit illustration

With the increasing scarcity of available radio-frequencies for satellite communication, the satellite link becomes a limiting factor for a mission. For example, a satellite could have plenty of storage and the best-in-class camera, but it can only take ten images per orbit because more images cannot be sent over the 19 kb/s link[2]. Compromises can be made by compressing or resizing the images, reducing them in quality and size making it possible to send more images over this link. Having more available bandwidth for the link would enable more and better quality images to be photographed and communicated.

The bandwidth problem has led the industry to develop high-bandwidth antennas for faster links. Some of the solutions include foldable satellite dishes, which can unfold in flight to save space during launch. These high-bandwidth antennas use different radio bands which can be harder to operate.

Another development to increase the bandwidth is using lasers for optical communication. Bigger satellites have already demonstrated high-speed links in space and to ground[3], proving it a viable technology. However, the CubeSats of the NewSpace industry lacks modules that support optical communication.

The Dutch research institute TNO started a project in collaboration with Hyperion Technologies (hereafter Hyperion) under the name of CubeCAT which envisions a 1 unit (1U) LCT (Laser Communication Terminal) for CubeSats. The envisioned CubeCAT module would feature a 1 Gb/s downlink and a 200 kb/s uplink that is operated from a satellite in Low-Earth Orbit (LEO) orbit.

During the project TNO is responsible for the optical, photonic, and the ground station parts, where Hyperion is responsible for the electronics. For the first iteration the goal is to achieve 100 Mb/s with hardware that is capable of 1 Gb/s. Later in the project this speed could be increased to 500 Mb/s with the final goal to meet the 1 Gb/s. As ultimate goal for the Laser Communication Terminal a speed of 10 Gb/s is envisioned.

1.1. Problem statement and Goals

One of the key features of the CubeCAT LCT is the high-speed digital data path which is responsible for the, first 100 Mb/s and later, 1 Gb/s throughput of the system. Despite the importance of this feature, this is still an open point that has not been implemented yet.

For the implementation of this high-speed digital data path, a design needs to be made with in mind the constraints of a CubeSat system, as well as the environment the module operates in, and the possibility of data corruption during transmission. The constraints manifest themselves in terms of power and size. Power in a CubeSat is limited due to the limited space for solar arrays and batteries in a satellite. The constrained size is due to the room required for the optics and other electronics in the 1U available volume. The environment CubeCAT will operate in is space, requiring hardware that can handle this environment. Furthermore, as data will be downlinked from space, it will be attenuated by Earths atmosphere. To reduce the possible corruption introduced by this attenuation, a FEC (Forward Error Correction) must be implemented.

As all the aforementioned items need to be addressed the following research question arises:

“ *How can the high-speed digital data path of the CubeCAT be implemented in such way that it provides a robust, upgradable, 100 Mb/s link, with constrained power and size?* ”

From this research question multiple goals can be devised. By achieving all goals the research question can be answered. The goals are as follows:

A Provide a design for the high-speed digital data path.

B Provide an implementation for the high-speed digital data path.

C Ensure robust communication by implementing a Forward Error Correction.

1.2. Methodology

Before being able to achieve the set goals, intermediate steps need to be performed. The following methodology is devised to address these intermediate steps, the goals and the research question.

- Explore where the CubeCAT module will be used as context for the design.
- Explore hardware in space as context for the design.
- Explore the CubeCAT project as context for all goals.
- Explore comparable work as context for the project.
- Detail the requisites of the high-speed digital data path for the design and implementation.
- Explore the design space.
- Propose a design for Goal **A**.
- Implement the Forward Error Correction.
- Verify the Forward Error Correction for Goal **C**.
- Implement the high-speed digital data path.
- Verify the whole high-speed digital data path for Goal **B**.
- Validate the results of the verification to the set goals.
- Conclude on the work done to answer the research question.
- Propose recommendations for future work on the high-speed digital data path.

1.3. Thesis Outline

The first part of Chapter 2 focusses on the SmallSat sector, followed by a discussion of the challenges of hardware in space. Following this discussion is a planar explanation of the CubeCAT project, its subsystems, and its coding scheme. Finally an overview of related work is included, with a conclusion given at the end. With the CubeCAT system explained, Chapter 3 zooms in on the design of the high-speed digital data path, by describing the design options, possible configurations, and the designs subsystem responsibilities. In Chapter 4 the design and implementation of the high-speed digital data path can be found. This is split up in a discussion about the FPGA implementation and the software implementation. Tests to verify the correct behaviour of the (sub-)systems are described in Chapter 5. Here, the power consumption of the system is also characterized. Lastly, in Chapter 6, a final conclusion is given for the thesis, with its main contributions, as well as future work.

2

Background

In this chapter information can be found that gives extra context to the market of the Laser Communication Terminal, the environment it operates in, the CubeCAT build-up, and its competitors. This information is later used as context for the design.

This chapter is organised as follows: in Section 2.1 the concept of SmallSats is explained. After that, the challenges of hardware in space are discussed in Section 2.2 after which the CubeCAT project is explained in Section 2.3, with special focus on the Forward Error Correction in Subsection 2.3.2. Following is the related work found in Section 2.4. Finally, a conclusion is given in Section 2.5.

2.1. SmallSats

Satellites like the Hubble telescope[4] use scientific equipment that can only fit in large satellites. The envisioned mission determines the size of the scientific equipment necessary. This also comes with extra mass, making them expensive to build and launch. When the scientific equipment is smaller in size a SmallSat can be used.

A SmallSat is a satellite with a total mass of (usually) less than 500 kg. Compared to bigger satellites SmallSats are cheaper since they require less space in the launch vehicle and less mass to get into orbit. Additionally, because they are smaller in size they are cheaper to design and produce. In most SmallSats commercial-off-the-shelf components are used, further reducing the costs. This enables larger production quantities of the satellites. The reduced total cost has the advantage that multiple satellites can be launched within the same budget. Multiple satellites improve redundancy in the mission, covering a ground target more frequently, or covering multiple ground targets. Multiple SmallSats can, for example, also be used to build constellations, for testing and qualifying hardware, for connectivity servicing (for example Hiber[5]), for earth observation and for student projects.

A sub-category of SmallSats is the CubeSat. It is a satellite based on modules of a standardized size, where 1 unit (1U) is a cube with edges of 10 cm. The standardized format allows for the creation of off-the-shelf components, making it easier, cheaper, and faster to build a satellite. Besides, more focus can be given to the payload, the core of the mission, as the rest of the components can be bought off-the-shelf.

With the CubeCAT Laser Communication Terminal a terminal of 1 unit (1U) is envisioned to enable SmallSat missions that require a lot of bandwidth, for example imaging missions. Compared to a traditional S-band transceiver[6] the downlink capability goes from 10 Mb/s to 1 Gb/s, resulting in a two decade gain. This gain in downlink capability it could enable missions that were otherwise not possible.

2.2. Hardware in space

In 1994 J.H. Trainor from NASA released a document[7] describing the effects of radiation in space. This document includes examples of missions encountering these kinds of problems. Trainor acknowledges in this document that the industry was making efforts to cope with the effects of radiation on hardware.

Since the publication, the industry hasn't stopped innovating. For example, Xilinx has radiation hardened FPGA families in its product portfolio that have been designed and tested with the help of

JPL and have even been independently tested by NASA[8]. Another example is BAE Systems with the RAD750 microprocessor[9]: a processor especially developed for use in space missions, such as the Mars 2020 Rover mission[10]. This hardware can deal with the challenges of the temperature and the effects of radiation in space. In the following subsections these challenges will be further detailed and mitigation options are discussed.

2.2.1. Thermal considerations

In space the temperatures can vary significantly. In a Low-Earth Orbit (LEO) the temperatures can change with 100 °C during one orbit on a 1U CubeSat[11]. This temperature change is smaller if it is measured at the circuitry. From [12] it can be seen that thermal cycling, for example from orbiting Earth, a chip can cause problems.

2.2.2. Ionization effects

On earth the atmosphere protects us from ionized particles coming from the galactic cosmic or the sun[13]. In space, these particles are not obstructed by the atmosphere and only affected by a magnetic field of a planet. The ionized particles can cause problems like Single Event Latchup (SEL), Single Event Upset (SEU), and Total Ionizing Dose (TID). These problems can be reduced/ mitigated as will be explained in the following subsubsections.

SEL

A Single Event Latchup (SEL) is caused by a particle hitting circuitry and inducing an electric charge. This can temporarily or permanently alter the function of circuitry by, for example, disabling the ability of a chip to change the state of one of its I/O pins[14]. If the problem is of temporary nature, then the problem can be solved by power-cycling the circuitry.

SEU

As with a SEL a Single Event Upset (SEU) is caused by a particle hitting circuitry and inducing an electric charge. The charge will change the state of, for example, a communication line, changing its binary value from a '0' to a '1'. The event is of temporary nature and will only induce a single error in the system, but will corrupt the communication that was ongoing on the communication line. Most noticeable are the upsets in memory, since the memory makes up large areas in chips. According to [15] multiple bit flips can happen per day.

TID

The Total Ionizing Dose (TID) closely relates to the doping of the semiconductors in a chip. Due to radiation the composition of the materials slightly changes over time[16]. This affects the doping of a semiconductor, changing its conductivity. The effects on a transistor (the fundamental building blocks of a chip) can be that the leak current increases, the transition speeds changes, or it stops functioning at all.

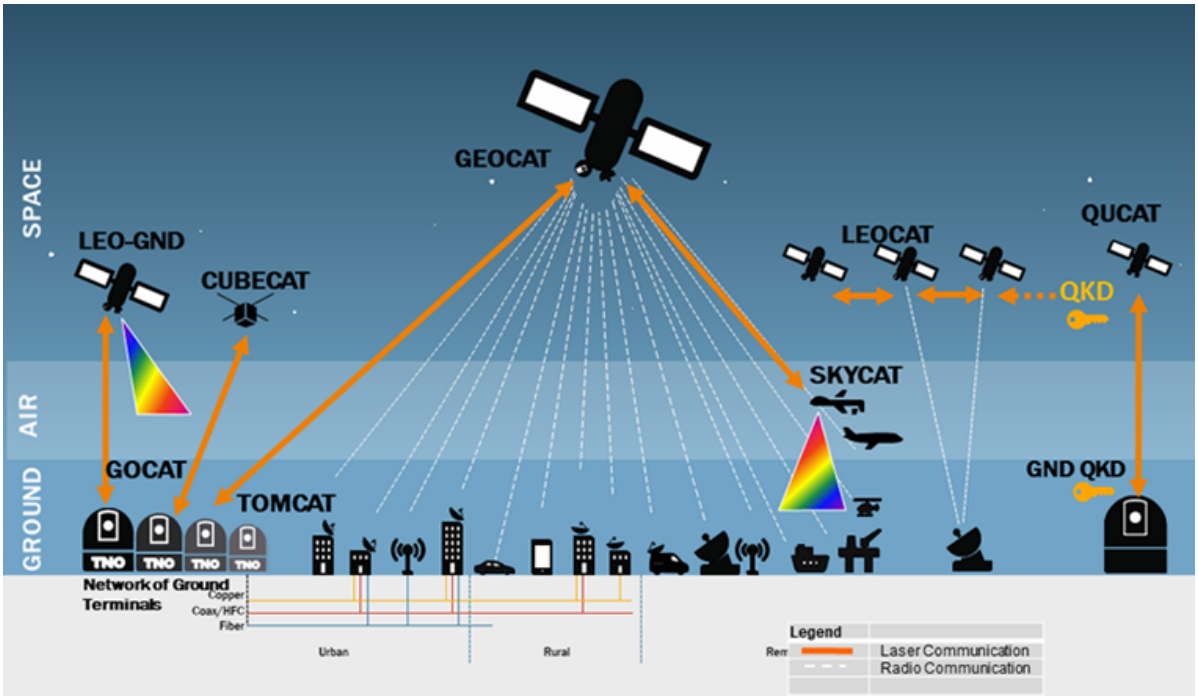
2.2.3. Mitigation

Thermal issues can be mitigated by installing heaters against the cold and radiators against the heat. These measures can reduce the difference in temperature. The effects of thermal cycling can be reduced by limiting the chip size.

Several options of mitigation exist for radiation. In circuitry blocks can be implemented redundantly, with a voting system such that one block can fail and the function of system as a whole isn't compromised. Another option is to shield the hardware from radiation[16], reducing the chance of particles actually hitting the circuitry. Data can be protected against corruption by radiation by using ECC (Error-Correcting Codes)[15], making it possible to repair data.

2.3. CubeCAT

CubeSat Communication Adaptive Terminal (CubeCAT) is part of a larger project of TNO where TNO envisions high-speed optical communication on a variety of vehicles, including airplanes, drones, ships, and satellites. The satellites can be split into GEO-satellites, LEO-ground-satellites, CubeSats, LEO-LEO-satellites (LEOCAT), and QUCAT (quantum key distribution). The TNO CAT systems are visualized in Figure 2.1.



Source: TNO website[17], TNO Insights Superfast internet using laser-satellite communications, ©TNO 2018

Figure 2.1: TNO CAT optical communication systems overview

CubeCAT is marketed as a 1 Gb/s downlink Laser Communication Terminal for use in CubeSats, requiring less than 15 watts of power and having an uplink channel capable of 200 kb/s. It is an Hyperion product that is the joint effort of TNO and Hyperion.

CubeCAT's intended usage orbit is LEO, to keep the distance between the terminal and the ground station minimal, allowing the 1 Gb/s in a 1U format with 15 watts power. To be able to support this downlink speed the CubeCAT has internal storage that acts as a buffer for the science data. During the transmission, data is read from the buffer at high speed, whilst in mission mode the science data can come in at a much lower rate via a RS-485 connection or a USB connection.

The optical link the Laser Communication Terminal has with the ground station is through Earth's atmosphere, meaning that the signal is attenuated by for example clouds and other atmospheric features. To reduce data corruption by attenuation the data that is downlinked is also encoded with a Forward Error Correction scheme.

In the Subsection 2.3.1 the CubeCAT components will be discussed in more detail. The Forward Error Correction codes discussed during the project are further detailed in Subsection 2.3.2.

2.3.1. System overview

The CubeCAT consists of multiple submodules, such as the controller for the *Fine Steering Mirror*, a *Quad Cell* detector, a *Laser driver* and a *Data Management Unit*. An abstract overview of the CubeCAT system with data flows is given in Figure 2.2. In the following subsubsections the submodules will be explained.

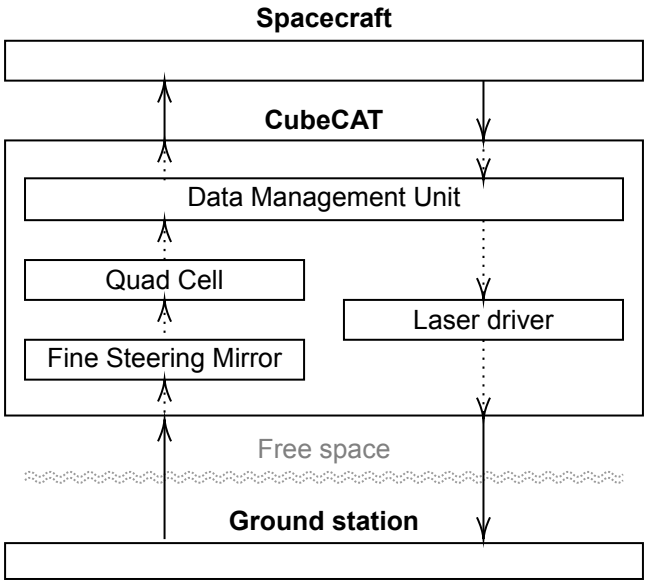


Figure 2.2: CubeCAT System overview

Data Management Unit

The DMU (Data Management Unit) provides an interface for the satellite to communicate with the CubeCAT, both for sending commands and for data. It stores the data received from the satellite on its internal storage bank and, when a link is established, downlinks the data at the required speed through the *Laser Driver* to the ground station. During the downlink, the data is also encoded by the DMU with a FEC (Forward Error Correction) code to reduce the data corruption during the transmission.

In Figure 2.2 it can be seen that the DMU has a path through the *Laser Driver* to the ground station. The path from the DMU to the *Laser Driver* is the end of the high-speed digital data path. Inside the DMU the high-speed digital data path starts. This is at the point where a file is loaded from the internal storage bank. As such the DMU is responsible for the high-speed digital data path.

The DMU has not been implemented, which means that to reach Goal **A**, a design needs to be made for the DMU. This design can then be used for Goal **B**; to implement the high-speed digital data path.

Quad Cell

A *Quad Cell* is a configuration of four photosensitive elements that can be used to detect the position of an incoming light beam. Based on the received power per photosensitive element the spot location, the centre of the incoming beam, can be calculated. Furthermore, the total received power can be calculated, which can be used as an indicator of how much interference the optical link has. Possibly, this interference indicator can be used to dynamically adjust the bitrate of the downlink, to prevent data corruption or to increase the bandwidth. Besides, the total received power over time can be used as an uplink channel to communicate with the satellite, completely removing the need for a radio link with the satellite.

Fine Steering Mirror

Ensuring the uplink and the downlink channels are aligned by only pointing the body of the satellite to the ground station is challenging. A satellite in LEO has an orbital velocity of more than 7 km/s, requiring very precise manoeuvres of the satellite to track the ground station that are not always possible.

To ease the stringent control of the satellite, a FSM (Fine Steering Mirror) is added to the CubeCAT. An FSM is a mirror attached to mechanical actuators to control the mirror's tip and tilt. It is used to aim the incoming laser beam on the *Quad Cell* and the outgoing beam on the ground station.

Laser driver

The *Laser Driver* is an electrical circuit designed to boost the low power data signal coming from the DMU to a signal that is able to drive the actual laser at the desired speed. In the CubeCAT system

a LVDS signal coming from the DMU is used to drive a laser at a maximum of 2.5 Gb/s for the first iteration.

2.3.2. Coding scheme

A is noted by Goal **C** a Forward Error Correction is needed to ensure robust communication. Different coding schemes can have different implications on the data rate, received BER (Bit Error Rate), and coding overhead. The decision for a coding scheme should fit with the intended application, otherwise the performance might be lower than is possible. In [18] can be read that choosing for one standard introduces a coding overhead of a factor 8.00, for a reparability of 25.00% of its input data (HPE-TC). If the link introduces little error then a coding scheme with an overhead of 3.78 and a reparability of 16.7% (HPE-TM) might be more suitable and doubles the achievable bandwidth for a link.

This section will briefly discuss three different coding schemes that were considered during the project, focussing on the reason it was considered, the design of the scheme, and if applicable, the reason it was changed in favour of another scheme.

CCSDS 131.0b3

During the early stages of the CubeCAT project in 2018 there where no standards for optical space communication. What was already available for space communication was the CCSDS 131.0b3 standard *TM Synchronization and Channel Coding*.

The standard was developed with radio communication in mind and contains a multitude of options for use in different scenarios as can be seen in Figure 2.3.

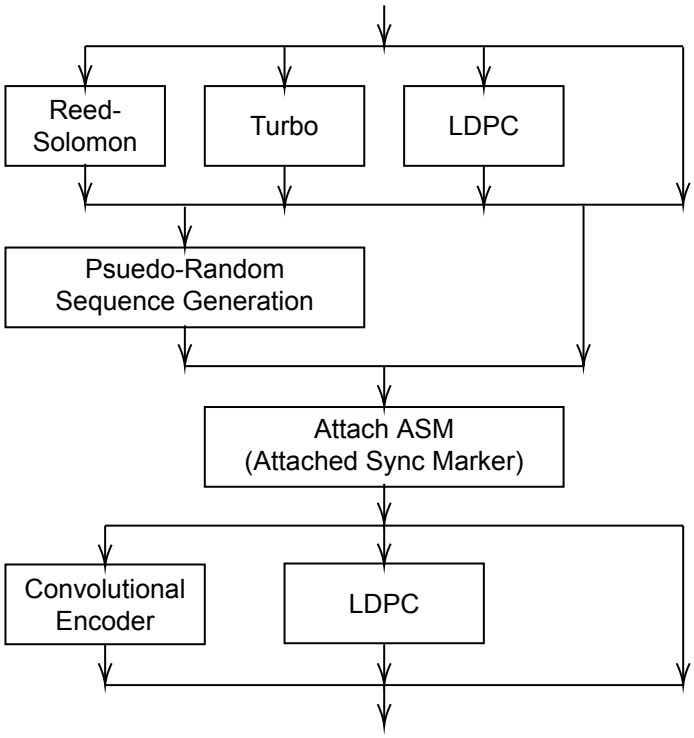


Figure 2.3: CCSDS 131.0b3 overview

Optical communication has extra challenges that are not covered by the 131 standard. With the discovery of the concept version of the 142 standard, the 131 standard was dropped in favour of the 142 standard.

CCSDS 142.0b1

First discovered as the concept version (CCSDS 142.0r1) the 142.0b1 standard was found to be more suitable for the challenge of optical communication. It is designed for a high-photon efficiency optical link and compared to the 131 standard adds a *channel interleaver* and a *PPM symbol mapper* as can be seen in Figure 2.4.

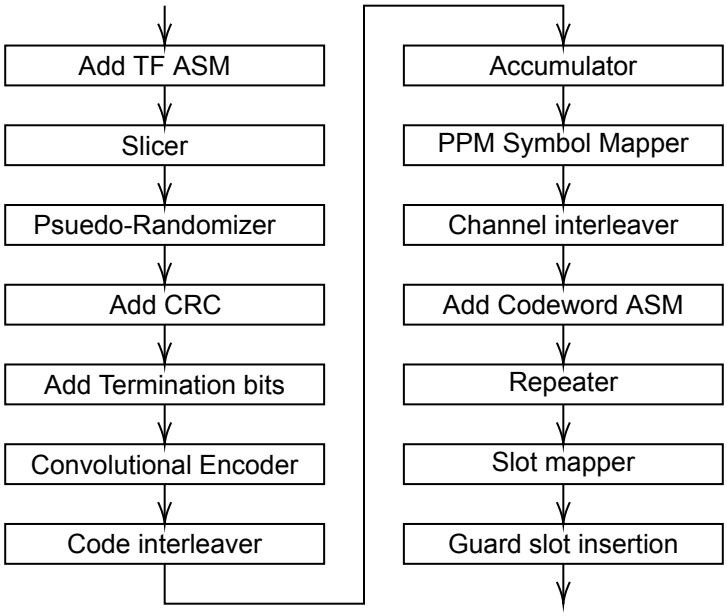


Figure 2.4: CCSDS 142.0b1 overview

The *channel interleaver* reorders the data stream in time, such that the data is communicated in a different order. This has as a consequence that a fade in the signal or a temporary loss of signal results in errors in multiple packets instead of the loss of packets. With the used error correction these erroneous packets can then be repaired.

A *PPM symbol mapper* places bits in transmission time slots, changing the detection method from '0' or '1' per instant to the time slot it was most likely received in.

The biggest problem of this standard was the lack of throughput. This is due to the fact that the standard has been designed with deep-space (long distance) communication in mind, whereas a CubeCAT terminal would be placed in LEO and communicate with a ground station (relatively short distance).

During discussions with TNO it became apparent another standard, the O3k standard, was in development. The O3k standard focusses on high-speed optical LEO links, matching exactly the purpose of CubeCAT.

The O3k standard was unfortunately, at the time of writing, only available in private communication in the working group. TNO has people in this working group and it was decided to move to self-made variant of the O3k standard; the TNO3k standard.

The efforts made on the 142 standard before the switch to the TNO3k standard can be found in Appendix A.

TNO3k

The CCSDS has a working group focussed on the development of the O3k standard. This standard focusses on 10 Gb/s (or divided by 2^N) optical LEO links, including LEO-to-ground links. With the development of the standard still in progress at the time of writing, the decision was made by TNO and Hyperion to make a variant on the standard with the CubeCAT system in mind: the TNO3k standard[19] (the name doesn't mention Hyperion due to the perfect match with the TNO name).

The base of the O3k standard, the Forward Error Correction, is the biggest discussion of the standard. Multiple parties, such as NASA, ESA and DLR each propose a different FEC scheme. These include Reed Solomon, Turbo coding, LDPC, and a variant on Turbo coding, with different parameters.

At the time of writing the TNO3k standard, the timeline for the first demonstration mission of CubeCAT was short. The amount of work that needed to be done, both at the CubeCAT side, as well as the ground station (the place of the decoder), was still significant. It was therefore decided to implement Reed-Solomon for Forward Error Correction due to its relative simple implementation for the decoder.

The functional structure of the encoder is quite similar to the 142 standard, without the PPM-modulation and repeater as can be seen in Figure 2.5.

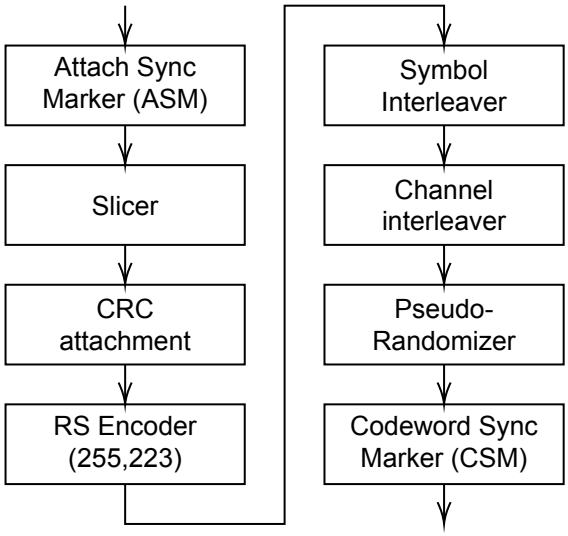


Figure 2.5: TNO3k overview

2.4. Related work

In the industry there are multiple initiatives for the development of a Laser Communication Terminal. These terminals can be split up based on their market as Non-CubeSat and CubeSat. This section lists those terminals in their respective market, but starts of with related work done on the CubeCAT.

2.4.1. CubeCAT work

On the 6th of July 2018 D.G. Bakker defended his thesis on the topic of *A Lasercom terminal for CubeSats*[20]. His thesis describes the architecture of the Laser Communication Terminal, with a focus on the quad cell receiver. His thesis was done at Hyperion, providing the fundamentals of the CubeCAT.

2.4.2. Non-CubeSat

CubeCAT is intended for the CubeSat market. However, on bigger satellites there is also work done on the subject of laser communication. A subset of this work is listed here.

AlphaSat TDP1

In 2007 ESA published an article [21] about the AlphaSat satellite in which the Inmarsat ”Geomobile” Mission was explained. The satellite has four technology demonstration platforms, of which one is a Low-Earth Orbit (LEO) to Geosynchronous Equatorial Orbit (GEO) LCT (Laser Communication Terminal). With this LCT ESA aimed to achieve high data throughput of up to 1.8 Gb/s[22], with a reduced latency [3]. This Laser Communication Terminal has been used as a basis for the *SpaceDataHighWay*. Compared to CubeCAT this module achieves a higher throughput and achieves it over a longer distance, since a LEO to GEO link is over 30000 km.

SpaceDataHighWay

In the ESA LCT article a collaboration between ESA, Airbus, TESAT-Spacecom and DLR was announced. This collaboration envisioned a *SpaceDataHighway* built upon laser communication to provide time-critical data at a low latency to Earth. Under the Copernicus project of ESA, satellites collect information about Earth. The Sentinels 1 and 2 developed under this project are equipped with an LCT that can make use of the *SpaceDataHighway*. The EDRS-series satellites relay the data coming from the Sentinels back to earth at data rates of 1.8 Gb/s[23]. With links over 40000 km[24] the *SpaceDataHighway* is focussing on a different application than CubeCAT. At this distance the throughput goes up to 1.8 Gb/s, meaning it is also faster than CubeCAT.

TerraSAR-X

In the same year DLR also demonstrated an inter-satellite optical link and a satellite to ground optical link [25] on the TerraSAR-X Earth Observation satellite. The module has mechanisms for fine pointing,

course pointing, and has a telescope. Furthermore, the module uses a base plate of 50 cm by 50 cm[26]. During a link of about 8 minutes a peak data rate of 5.6 Gb/s was achieved for a LEO to LEO link[26] using , on average, 35 watts of power. Compared to the CubeCAT it is faster, requires more power, and is significantly bigger.

10 Gb/s future mission

In 2012 NASA stated that for future Earth observation missions a downlink capability of 1 to 9 Gb/s is needed. H. Hemmati and J.M. Kovalik of the Jet Propulsion Laboratory of California Institute of Technology are designing and validating an LCT that is capable of 10 Gb/s[27]. This LCT is intended for use in LEO to ground links. Unfortunately it is unknown to the writer of this thesis what this technology developed into.

BiROS: OSIRISv2

The BiROS satellite is a satellite optimized for fire detection, flying at an altitude of around 500 km[28]. One of the modules used on the satellite is an LCT, the OSIRISv2[29]. This LCT is aimed to work at 1 Gb/s with an uplink channel of 1 Mb/s. The LCT also features a *Quad Cell* and uses a beacon for alignment.

In the latest update of the OSIRIS program the satellite has been launched, but the OSIRISv2 payload has not been tested yet[30]. The module works at 37 watts, which is more than double the amount of CubeCAT. In functionality it is comparable to CubeCAT, meaning that CubeCAT might compete with the OSIRISv2 for its reduced power requirement.

2.4.3. CubeSats

The intended market for the CubeCAT is the CubeSat market. Other than CubeCAT there are more LCTs in development. A subset is given here.

Nice^3 (The Nice Cube)

The Nice^3 is a project of the Université Côte d'Azur that aims to create an LCT, while keeping the satellite as a whole in a 1U CubeSat format. With this LCT a transmission rate of 1+ Kb/s)[31] is envisioned. Due to the size requirements of the Nice^3 satellite, it has less space for the LCT. This results in a limited transmission rate compared to CubeCAT.

Sinclair Interplanetary

Sinclair Interplanetary is working on a Laser Communication Terminal for full-duplex communication between satellites at a rate of 100 Mb/s, with a maximum distance of 250 km[32]. Operating the terminal is done at less than 10 W with a volume of the terminal that is less than 1U. Another Laser Communication Terminal Sinclair is working on is designed for downlinking data at a rate of 1 Gb/s at 1000 km[32]. This terminal is also designed to operate at less than 10 W and requires less than 1U volume. Compared to CubeCAT, Sinclair has a relative competitive 1 Gb/s terminal. Sinclairs terminal requires less space and power, but lacks any form of encoding, storage, or tracking capabilities.

MIT: CLICK B

The Click B[33] is a university Laser Communication Terminal that aims to provide full duplex 20+ Mb/s for links between 25 km to 580 km. With a peak power requirement of 40 W and a size of 1.5 U it can be used for CubeSats. Compared to CubeCAT it requires more power and space, to provide less bandwidth.

CubeLCT

Tesat-Spacecom has a Laser Communication Terminal that is capable of a 100 Mb/s downlink [34] and aims to achieve 1 Gb/s[35]. With only 0.3U it is designed to be able to communicate over a maximum distance of 1500 km. For this it requires at maximum 10 watts of power and uses a beacon and a FSM for alignment. As encoding the CCSDS 03k standard will be implemented. Compared to CubeCAT it lacks the ability to store data.

2.5. Conclusion

With a 1 unit (1U) Laser Communication Terminal, CubeCAT focusses on the CubeSat industry. This industry faces the challenge to protect the hardware against the effects of thermal differences, Single Event Latchup, Single Event Upset and Total Ionizing Dose. After two iterations of coding schemes the TNO3k standard will be used for the CubeCAT to ensure robust communication.

Multiple cases on bigger satellites that have already proved the optical laser communication technology are already on the market. In the SmallSat subset of the market Tesat-Spacecom has the *CubeLCT* that is capable of 100 Mb/s in a 0.3 U size and Sinclair is working on an 1U 1 Gb/s terminal. Other terminals are mostly university projects.

3

Data Management Unit

This chapter discusses the design of the DMU (Data Management Unit) to effectuate the high-speed digital data path. Here, multiple options for the design are discussed, as well as responsibilities, of the DMU. The options are used to select a design, effectively achieving Goal **A**.

Before reading this chapter the reader is expected to have read Section 2.2 about hardware in space and Section 2.3 about the CubeCAT project and the role of the DMU in the CubeCAT module.

This chapter is organised by first discussing the responsibilities of the DMU in Section 3.1. After that the design options for the DMU are discussed, differentiating between the architecture, interconnections and the components, which can be found in Section 3.2. In Section 3.3 three different designs are presented that were the options during the project. In Section 3.4 the subsystem responsibilities are further detailed, based on the selected design. A conclusion can be found in Section 3.5.

3.1. Responsibilities

This section discusses the responsibilities of the DMU by looking into the interfaces the DMU has in the CubeCAT system, followed by further detailing the tasks the DMU is responsible for. Lastly, these responsibilities are condensed into requirements. These requirements are then, later in this chapter, used to base the design choices on.

3.1.1. System interfaces

As was seen in Figure 2.2 the DMU is responsible for interfacing with the satellite. This includes both handling the data coming from the satellite as well as handling the commands coming from it. The data interface is either a RS-485 connection or an USB connection. A command interface is provided by re-using the data interface or a simple U(S)ART connection.

Furthermore, the *Quad Cell* interfaces with the DMU, providing sensor data meant for logging. This connection is a U(S)ART connection. A data path to the ground station is provided via the connection to the *Laser Driver*. The *Laser Driver* amplifies an incoming LVDS signal, meaning that the DMU should output an LVDS signal. The interfaces are summarized in Figure 3.1.

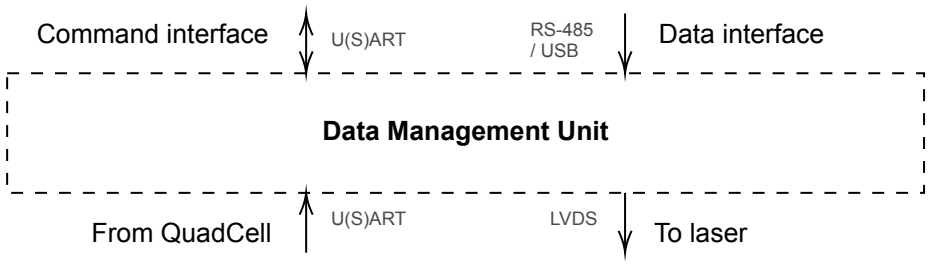


Figure 3.1: Data Management Unit interfaces

3.1.2. Tasks

In the CubeCAT the DMU has multiple tasks to ensure the correct operation of CubeCAT as a system. These tasks can be grouped in three main tasks, namely:

- **Data aggregation**: The DMU is responsible for storing the data from the data interface. It is also responsible for logging sensor data.
- **Data encoding**: To reduce data corruption during transmission the CubeCAT system should encode the data coming from the satellite. The DMU is responsible for encoding this data according to the TNO3k standard.
- **Data output**: Upon establishment of an optical link the CubeCAT system should start sending data to the ground station. The stored data should be sent to the *Laser Driver* in a non-interrupt data stream at a 100 Mb/s rate for the first iteration of CubeCAT. Later iterations/ updates should stream at a 500 Mb/s, 1 Gb/s or 10 Gb/s rate.

Both *Data encoding* and *Data outputting* are needed for the high-speed data path.

3.1.3. Requirements

The DMU needs to interface with the rest of the system of CubeCAT, as well as fulfil its own tasks within CubeCAT. Besides, it should fit with the rest of the CubeCAT in 1U and it should not draw too much power. As such, there are requirements for the design of the DMU. These are defined in Table 3.1.

Table 3.1: Data Management Unit requirements. The *Importance* column specifies the importance for the CubeCAT project.

Requirement	Importance	Description
Time	Medium	The DMU should be finished before September 2020.
Power	Medium	The DMU power should remain under 2 watts.
Size	High	The DMU PCB should fit in a 8 cm by 8 cm area.
RS-485	High	The DMU should be able to interface with RS-485.
USB	High	The DMU should be able to interface with USB.
Quad Cell interface	High	The DMU should be able to interface with the *Quad Cell* with U(S)ART.
Satellite interface	High	The DMU should be able to process U(S)ART commands from the satellite.
Storage	High	The DMU should be able to store data.
TNO3k encoding	High	The DMU should encode the data according to the TNO3k standard.
3 ms channel interleaver	Low	The TNO3k encoder should have a channel interleaver depth interleaving 3 ms of data.
100 ms channel interleaver	Low	The TNO3k encoder should have a channel interleaver depth interleaving 100 ms of data.
300 ms channel interleaver	Low	The TNO3k encoder should have a channel interleaver depth interleaving 300 ms of data.
100 Mb/s speed	High	The DMU should be able to output data at a steady 100 Mb/s.
500 Mb/s speed	Medium-High	The DMU should be able to output data at a steady 500 Mb/s.
1 Gb/s speed	Medium	The DMU should be able to output data at a steady 1 Gb/s.
10 Gb/s speed	Future	The DMU should be able to output data at a steady 10 Gb/s.
Hardware	High	The DMU hardware should be capable of meeting Requirement ‘1 Gb/s speed‘ and Requirement ‘3 ms channel interleaver ‘.
LVDS	High	The DMU output should be LVDS.

3.2. Design options

This section discusses the design options for the DMU. It starts by discussing the different possible architectures. This is followed by discussing the possible interconnections needed to support some

architectures. Lastly, components are listed that would support the required functionality as well as to be able to survive in LEO.

3.2.1. Architecture

With different architectures come different advantages and disadvantages. For example, an architecture could allow for a more flexible swap of components during the project when issues are found with that component. Another architecture would require to redo all the work with that component, causing a set-back in time. With Requirement ‘Time‘ in mind, it is thus nessacary to explore different architectures.

An important element in all architectures is the use of extra memory to support Requirement ‘3 ms channel interleaver ‘ due to Requirement ‘Hardware‘. Calculated from Requirement ‘3 ms channel interleaver ‘ and Requirement ‘1 Gb/s speed‘ a memory of at least 300 Mbit is needed. To support Requirement ‘100 ms channel interleaver‘ and Requirement ‘300 ms channel interleaver‘ at least respectively 1 Gbit or 3 Gbit is needed.

In this subsection the following architectures will be explored. First a microcontroller with an LVDS interface will be discussed, followed by a discussion of a microcontroller with an FPGA. Then a microprocessor with an FPGA is explored. Lastly, an architecture with only an FPGA is discussed.

Microcontroller with LVDS interface

Interfacing with the satellite can be handled by a microcontroller. They allow for deterministic program flows, at relatively low power. Regarding I/O microcontrollers are flexible and can emulate digital interfaces they don’t have peripherals for. For the LVDS interface a converter is needed, or special microcontrollers need to be used. The memory for the *Channel Interleaver* is connected to the microcontroller, as well as the storage.

On the other hand, micro controllers are limited in speed, with, for example, simple ARM M0 cores working at (not limited to) 48 MHz and more advanced M7 cores working at (not limited to) 480 MHz, with instruction- and data cache and 64-bit floating point support. Operation at 1 Gb/s of the laser will be challenging with the I/O handling overhead, the encoding, storage handling and incoming data from the QuadCell.

An overview of this architecture can be seen in Figure 3.2.

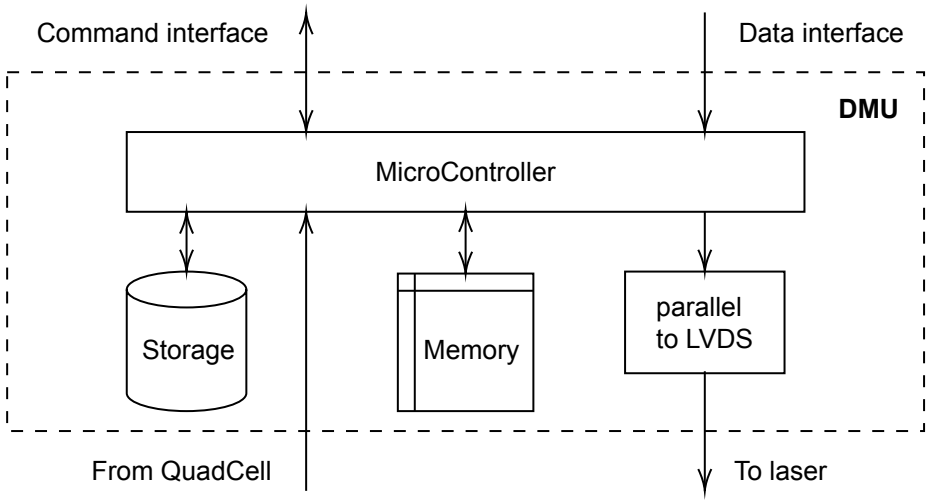


Figure 3.2: Microcontroller LVDS bridge interface architecture

Microcontroller with FPGA

As with the *Microcontroller with LVDS interface*, the microcontroller can implement the interfacing with the satellite and the QuadCell. Interfacing with the laser is handled by the LVDS driver of an FPGA, as this interface is quite common on FPGAs. Optionally, the data interface can be connected directly to the FPGA, reducing overhead in the microcontroller. With this option the data can also be stored directly by the FPGA, because it is connected with the storage. Encoding the data can be done efficiently at the FPGA, reducing the load on the microcontroller as well. This requires the FPGA to also interface with the memory to support the *Channel Interleaver*.

A downside of this configuration is the effort needed on the FPGA side to implement all drivers and logic in-between the drivers. The storage driver will need to interface with the memory, deal with delays and needs to be able to recover state between power cycles. This can be offloaded by the microcontroller, which commands for data movements and loads the state of the storage. Another disadvantage of using an FPGA is the increased power consumption. It does however allow for faster operation.

An overview of the architecture can be seen in Figure 3.3.

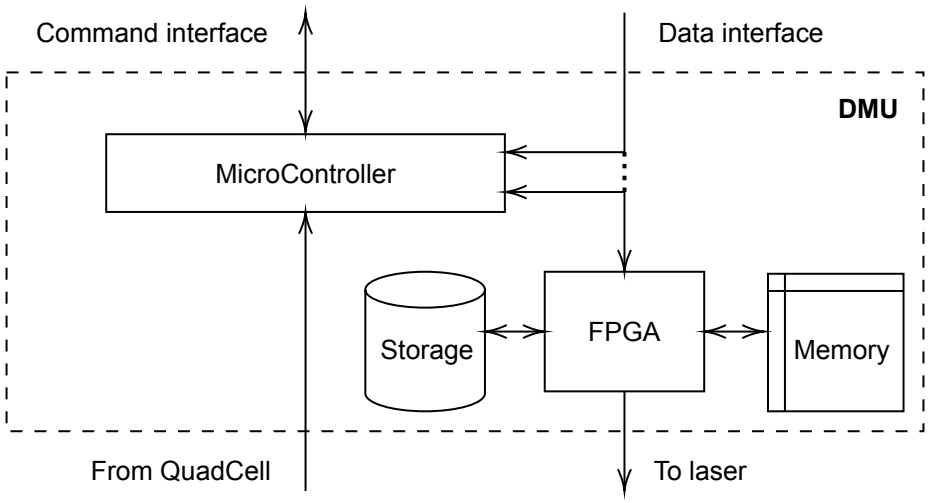


Figure 3.3: Microcontroller with an FPGA architecture

Microprocessor with FPGA

Replacing the microcontroller with a microprocessor enables the use of Linux, instead of writing bare-metal software. Drivers for the processors peripherals are written and tested, shifting the focus to data handling rather than implementation. Besides, peripherals for high-speed data storage are available and their drivers are ready to use, making attaching or swapping storage only a physical effort. With the increased possibilities of a microprocessor the power consumption can increase, that is, if those possibilities are used.

The FPGA will then still do the TNO3k encoding of the data and drive the LVDS signal. This means that the effort on the FPGA part is the same as with the microcontroller.

The downside of this configuration is that the use of an OS like Linux obstructs (near) real-time handling of commands, leading to potential problems that are timing related. Using an real-time OS would improve this, but would require more focus on the drivers and the program flow.

An overview of the architecture can be seen in Figure 3.4.

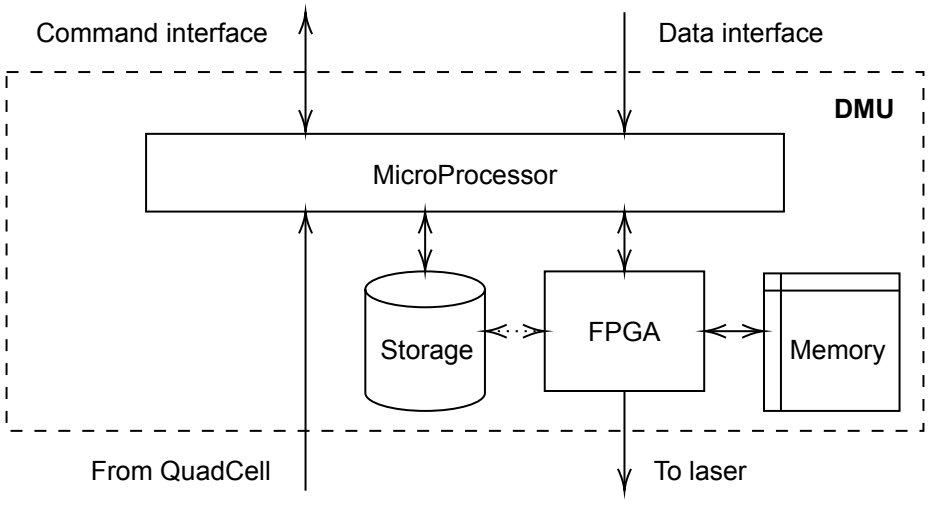


Figure 3.4: Microprocessor with an FPGA architecture

Only FPGA

Another architecture that can be used only using an FPGA that interfaces directly with the satellite. This solution can be the most efficient implementation in terms of speed and responsiveness, as all operations are in hardware. However, this architecture comes at the highest development effort, as both the command interface and the *Quad Cell* interface need to be handled by the FPGA.

An overview of the architecture can be seen in Figure 3.5.

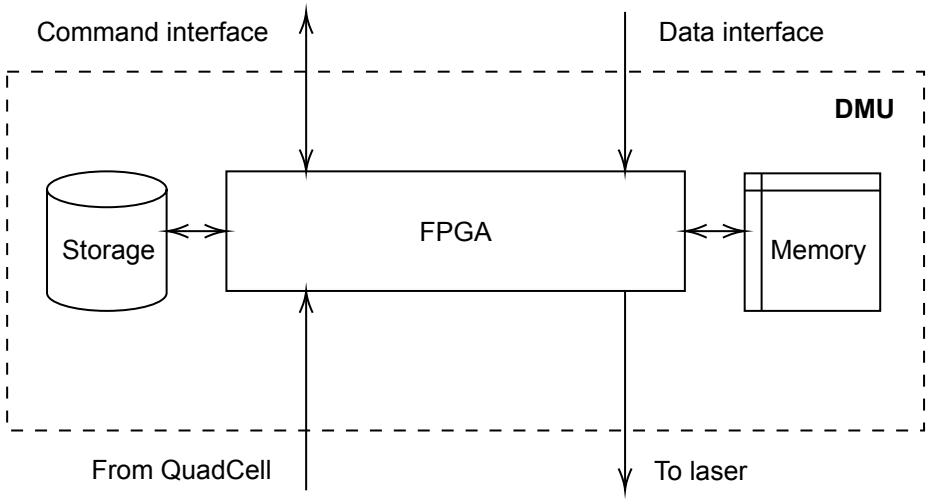


Figure 3.5: FPGA only architecture

Architecture decision matrix

Based on the architectures discussed before a choice table has been made. This table gives an overview of the implications on *power*, *speed*, *design effort*, *size* and *expandability* of the different architectures relative to each other. The overview can be seen in Table 3.2.

Table 3.2: DMU architecture choice matrix

Parameter	**Microcontroller with LVDS controller**	**Microcontroller with FPGA**	**Microprocessor with FPGA**	**FPGA only**
Power	Low	Medium	Medium-High	Medium
Speed	Low	Medium	High	High
Design effort	Low	High	Medium	High
Size	Low	Medium	High	Medium
Expandability	Low	Medium	High	Medium

From this table can be seen that the choice of architecture depends on multiple factors; there is no one solution that fits them all. Choosing an architecture depends on the power available, desired speed, maximum size allowed, expandability desired and how much time is available to design and implement the architecture.

3.2.2. Interconnection

The *Microcontroller with FPGA* and the *Microprocessor with FPGA* architectures require communication with an FPGA. This communication can be realized by different protocols, resulting in different speeds and implementation complexity.

In this subsection the following protocols will be discussed: SPI, SD, QSPI, EBI, USB and PCIe. Focus will be put on the relative speed, but also on the design effort for Requirement 'Time', upgradability for Requirement 'Hardware' and the amount of traces for Requirement 'Size'.

SPI

SPI is used for many interfaces, such as in an accelerometer, ADC, gyroscope or flash memory for which the industry has developed the SPI-NOR standard. The SPI protocol is duplex and uses four

traces: one clock line (SCLK), one chip select (NCS), one line to the slave and one line from the slave. This can be seen in Figure 3.6 as well.

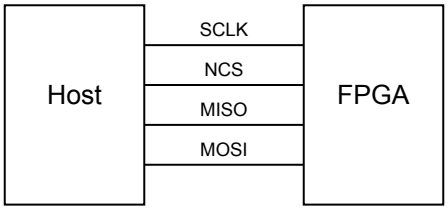


Figure 3.6: SPI connection

With the data rate directly related to the clock speed of SCLK, the data rate can get as high as the clock speed can. This clock rate usually starts around 1 MHz and can go up to 250 MHz on for example a Raspberry Pi 3, as this maximum is defined by the peripheral bus. On boards that have higher peripheral bus speeds, the speed can be increased even further to 400 MHz. With this in mind Requirement ‘100 Mb/s speed‘ can be met with SPI, but Requirement ‘500 Mb/s speed‘ will be hard as transmission line effects start to get into play at higher frequencies. Switching from SDR to DDR for SPI doubles the data throughput, but will still not reach the 1 Gb/s for Requirement ‘1 Gb/s speed‘, but 500 Mb/s for Requirement ‘500 Mb/s speed‘ should be reachable.

As SPI is a common standard its design complexity is low; a SPI peripheral can be found on many devices. Since the only way to upgrade the speeds is to increase the clock frequency, it is not quite upgradable.

SD

SD is a common storage interface that can be found on phones, mp3 players, dashcams and many other types of devices. With the increasing demand for faster SD-cards, the SD Association has developed multiple iterations on the standard. The original standard starts with a bus speed of 100 Mb/s and with the latest iteration (8.00) the speeds goes up to 3940 Mbyte/s using PCIe[36]. To enable these fast bus speeds LVDS pins were added to SD interfaces. Ignoring these extra pins for Requirement ‘Hardware‘, the SD interface can go up to 104 Mbyte/s.

The layout of an SD interface without LVDS pins is as follows. There is a clock line to sync the data on (CLK). Then there is a line to signal commands to the slave and for data there are four lines available. A visualisation of an SD connection can be seen in Figure 3.7. An SD interface with LVDS pins would add two differential pairs.

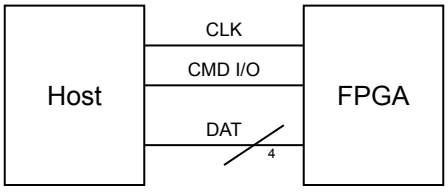


Figure 3.7: SD connection

A lot of microprocessor/microcontroller manufacturers include an SD peripheral in their products, easing the connection to this kind of storage. For an FPGA there are open source designs available such as [37]. The problem with the SD standard for upgradability is that its bus protocol changes for increased speeds, meaning that per iteration a significant design effort has to be made.

QSPI

Based on the SPI standard, QSPI extends the amount of lines with two extra generic IO lines and repurposes the MISO and MOSI lines to generic IO lines, totalling to four (Quad) IO lines. With these extra lines and as basis the SPI standard the bus throughput is 4 times as big. For reference, using the Raspberry Pi 3 SPI clock speed a bus throughput of 1 Gb/s is possible on SDR. The QSPI connection is visualized in Figure 3.8.

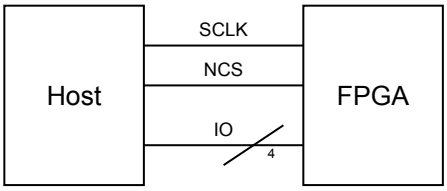


Figure 3.8: QSPI connection

The QSPI interface is mostly used as an extension of the SPI-NOR standard. QSPI peripherals used in microcontrollers/microprocessors might implement the command structure of the SPI-NOR standard, requiring the FPGA implementation to also work according to this command structure. This increases the design complexity, but with the basis of SPI it is still a reasonable effort.

Compared to SPI it inherits the same issues for the upgradability. Only the clock frequency can be increased if DDR is already implemented.

EBI

An EBI (External Bus Interface) can be implemented in a microcontroller/microprocessor to expose the internal peripheral interconnection and utilize the available infrastructure in the chip to increase the possible bandwidth with devices connected on this bus. It consists of a clock line (MCK), multiple addressing lines (A), multiple data lines (D) and five control lines. These control lines include a chip select (ME), a read enable (RE), a write enable (WE), a lower byte enable (LBE) and a high byte enable (HBE). These lines are visualized in Figure 3.9. If needed, multiple address lines can be ignored as the data handled by the FPGA will not work with addresses.

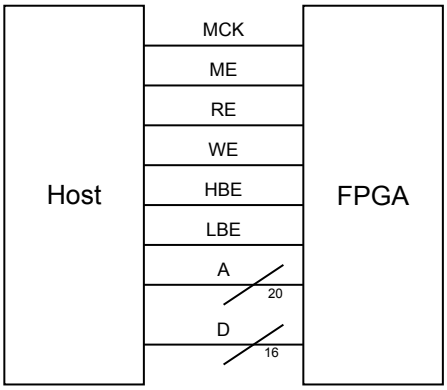


Figure 3.9: EBI connection

Due to the amount of data lines it is possible to send multiple bytes per clock. With 16 data lines working at 100 MHz a data throughput of 1.6 Gb/s is possible, meeting Requirement ‘1 Gb/s speed‘. Implementing this interface is, however, harder due to the fact that it is less popular and thus the available examples are scarce. Besides, with 16 lanes for data extra care needs to be taken to ensure all signals are in phase. The upgradability of this interface has the same issues as SPI. As the amount of lanes are set, the speed can only be further increased by increasing the clock speed.

USB

USB (Universal Serial Bus) can be found anywhere, from phones to tablets, from e-readers to speakers and from external hard disks to microphones. Depending on the version, different amount of data lanes are available. To guarantee backwards compatibility the newest version (at the date of writing 4.0) still has the D+ and D- pair for data transfer at a maximum of 480 Mb/s. With the newer standards an extra connection is made with 4 extra high speed differential pairs and two lanes for auxiliary usage. This can be seen in Figure 3.10.

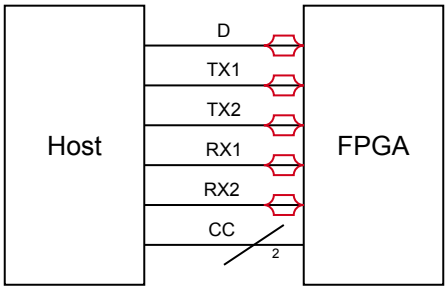


Figure 3.10: USB connection

Using the newest specification a link speed of 40 Gb/s[38] can be achieved. Additionally, the specification has alternative modes for use of PCIe over the USB link[39].

On the integration side USB is a significant task. With all the link modes, packet types and transfer types to support the highest speeds, it not a simple task to implement USB on the FPGA. The up-side is however the possibility to connect all lines needed for the highest protocol version and then start implementation to support version 2.0 first, only using the D+/D- pair. Upgrading is then ensured by implementing or reusing a driver and message handler that supports 3.0 (5 Gb/s), after which upgrading to USB 3.2 (10 Gb/s), USB 3.2 (20 Gb/s) or USB 4.0 (40 Gb/s) is still a possibility.

PCIe

PCIe can be found in almost every PC that is currently in use. It can be used to connect add-in cards in the PC to expand the abilities of the PC. Common add-in cards are network and graphics cards, with a new trend towards NVMe SSDs for fast storage. With version 2 of the protocol a maximum of 4 Gb/s can be reached on a x1 link. Version 3 allows for a maximum of 7.88 Gb/s on a x1 link. Increasing the link bandwidth can be done by either upgrading in PCIe version or increasing the number of links. A link consists of 2 differential data lines and one differential clock line, resulting in a total of 6 traces needed, which is visualized in Figure 3.11.

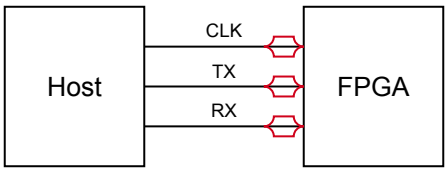


Figure 3.11: PCIe connection

Part of the PCIe specification is the way data is handled. PCIe is a packet based protocol, making that the actual data throughput is lower. Besides, on the FPGA side the packets need to be processed and handled. On a microprocessor a driver needs to be made to be compatible with the functionality of the FPGA and the FPGA needs to implement the whole PCIe stack, having quite the complexity.

Interconnection decision matrix

The 6 described interconnection options have their respective qualities. Table 3.3 shows an overview of the interconnections with the implications on *speed*, *design effort*, *upgradability* and *traces*.

In this table the *design effort* implies the expected amount of effort needed to implement this interface with respect to Requirement 'Time'. *Speed target* describes which speed requirement can be met with this interconnection and *traces* describes the amount of traces needed on a PCB to support the interconnection. The *upgradability* implies the ease of significantly increasing the transfer speed.

Table 3.3: DMU interconnection choice matrix. N denotes the link width.

Parameter	Design effort	Speed target	Traces	Upgradability
SPI	Low	100 Mb/s - 500 Mb/s	4	Low
SD	Medium	100 Mb/s - 1 Gb/s	6 + 4	Medium
QSPI	Low	100 Mb/s - 1 Gb/s	6	Low
EBI	Medium	1 Gb/s	22 + 0-20	Low
USB	High	100 Mb/s - 10 Gb/s	2 + 10	High
PCIe	High	1 Gb/s - 10 Gb/s	2 + 4 * N	High

3.2.3. Components

Knowing what is available in the market helps making a design that can be made in reality. As such in this subsection multiple components are explored to help decide on the build-up of the DMU. The listed components are a considered subset of all possible components on the market. The components are categorized in *Microcontroller*, *Microprocessor*, *Integrated microprocessor with FPGA*, *FPGA* and *Storage*.

Microcontroller

Multiple variants of microcontrollers are available on the market, with a clear distinction between the instruction set they use. The listed microcontrollers are expected to be able to survive LEO for a nominal CubeSat missions duration.

ATSAMV70 With a 300 MHz ARM Cortex-M7 processor the ATSAMV70[40] should be able to handle data at a high rate. With integrated DMA, USB, EBI and QSPI peripherals the chip should be able to meet Requirement ‘100 Mb/s speed‘.

STM32H745 The STM32H745XI[41] is a dual core design with a Cortex-M7 at 480 MHz and a Cortex-M4 at 240 MHz. It has multiple peripheral, including DMA, QSPI, SDMMC and USB. This should make the chip able to meet Requirement ‘100 Mb/s speed‘.

Microprocessor

The *microprocessor* provides an interface between the satellite and the LCT. Besides, it should interface with the FPGA and the Quad Cell and it should be able to store data on the storage.

i.MX8M The NXP i.MX 8M[42] features 4 ARM Cortex-A53 microprocessors and 1 ARM Cortex-M4 microcontroller, 2 DMA controllers that can be used for efficient data transfer, 3 watch dog timers, 2 PCIe 2.0 peripherals (x1 each), 2 USB 3.0 connections, 1 QSPI and 2 SDIO3.0/MMC5.0 controllers for data storage. All *i.MX8* series processors are produced on an FD-SOI (Fully Depleted Silicon On Insulator) process, which is better resistant to latch-ups[43]. These features make the i.MX 8M a good fit for the DMU, meeting Requirement ‘100 Mb/s speed‘, Requirement ‘500 Mb/s speed‘, Requirement ‘1 Gb/s speed‘ and Requirement ‘Hardware‘.

Due to the size constraint Requirement ‘Size‘ the SolidRun SoM i.MX8M[44] was found to be a good fit. With its small size and rich peripheral set multiple options are available for using and connecting the module.

i.MX8X The NXP i.MX 8X[45] is a processor that is marketed as being *Safety certifiable and efficient performance*. With the focus on safety, the RAM can be protected against corruption by use of ECC. Besides, that it uses newer ARM Cortex-A35 cores[46] that, compared to the ARM Cortex-A53, are more focused on energy efficiency[47]. As an integrated microcontroller an additional Cortex-M4F is implemented.

On the I/O side of the processor there is support for 1 PCIe 3.0 x1 connection, 1 USB 3.0 connection, 2 Quad SPI (or 1 Octal SPI) and 2 SDIO3.0/eMMC5.1 controllers for data storage. As mentioned before, this processor is also produced on the FD-SOI process, making it better resistant to latch-ups. This makes it fit for Requirement ‘100 Mb/s speed‘, Requirement ‘1 Gb/s speed‘ and Requirement ‘Hardware‘.

As the i.MX 8X is, as of date, still in preproduction the available modules are scarce. One relatively small SoM (System on Module) is the TQMa8Xx[48] from TQ-Group. The SoM has all important connections available, making it a good fit.

CP400.85 The Hyperion CP400.85[49] is a SoM designed by Hyperion which is to be used as OBC (On-Board Computer) on satellites. By design it is build with components that are space-grade and it has already flown before. This makes the CP400.85 a validated platform for the CubeCAT mission.

It features an ARMv7-A capable processor running at 500 MHz and is capable of interfacing with storage with QSPI and SD. The QSPI is able to communicate at a maximum of 133 MHz or 532 Mb/s, where the SD interface is capable of 120 MHz at 8 bits wide for a total of 960 Mb/s. Further more an EBI is exposed and USB 2.0 is available for use. It should easily meet Requirement '100 Mb/s speed' and Requirement '500 Mb/s speed'. After optimization Requirement '1 Gb/s speed' is reachable.

Integrated microprocessor with FPGA

Integrating the processor with the FPGA has the advantage of not needing a interconnection, since the FPGA is memory mapped. This reduces the design effort.

Zync UltraScale+ CG Following this philosophy Xilinx has integrated 2 Cortex-A53 cores, 2 Cortex-R5 cores and an FPGA into one chip, the Zync UltraScale+ CG[50]. For user application USB 3.0 is available and for storage QSPI, SD/eMMC, SATA and PCIe 2.0 are available.

With ECC on its microprocessors, microcontrollers and RAM the chip is also protected to data corruption during execution. It does, however, not provide information on radiation tolerance.

The Zync UltraScale+ CG has 16.3 Gb/s transceivers starting from the ZU4CG model and upwards. This transceiver covers all desired speeds for the connection to the *Laser Driver*, making it a viable solution for all speed requirements.

However, from earlier experience of Hyperion the Zync platform seemed to be power hungry and unfavoured in space application. It is thus unfavoured in the application of CubeCAT.

PolarFire SoC Based on the original PolarFire FPGA (see later) MicroSemi designed a SoC with the PolarFire FPGA connected to a hardened microprocessor system. This microprocessor system consists of multiple RISC-V[51] microprocessors and a RISC-V microcontroller. For storage it can interface with MMC5.1 and QSPI. For user application 1 USB 2.0 is available.

Unfortunately the SoC is still in early access making it at the moment of writing not a viable option.

FPGA

The *FPGA* is responsible for providing the laser with data and thus should be able to interface with LVDS to the laser. Other than that it should be capable of encoding the data stream according to the TNO3k standard.

PolarFire The MicroSemi PolarFire[52] is a space-grade FPGA[53] with transceivers that are capable of doing 12.7 Gb/s making it capable of reaching Requirement '10 Gb/s speed'. Besides, it has an integrated PCIe endpoint, as well as an integrated block for DDR memory.

ECP5-5G Providing support for PCIe, DDR memory and SEU mitigation the Lattice ECP5-5G[54] is an FPGA that is suitable for use in the CubeCAT. Given the requirements Requirement '100 Mb/s speed' and Requirement '1 Gb/s speed' the ECP5-5G fits the need. Besides, its relative small size helps with Requirement 'Size', and the relative low power helps with Requirement 'Power'.

Storage

The *Bulk storage* is responsible for storing science data from the satellite during an orbit. When an optical link is established with the ground station the science data is loaded and streamed through the system to the laser. The stream should be continuous, requiring the bulk storage to have sufficient read speeds.

eMMC Originally the MMC standard was a standard for swappable cards like an SD card. Due to the need for embedded chips the eMMC standard was introduced for integrated use of MMC. With a data transfer speed of 400 MB/s (3.2 Gb/s)[55][56] maximum transfer speed in HS400 mode eMMC is suitable for use in CubeCAT.

PCIe SSD As mentioned in Figure 3.2.2 the PCIe standard allows for very fast transfer speeds. Connecting a SSD to this bus provided storage with transfer speeds of 15000 MB/s (120 Gb/s)[57], provided PCIe 4.0 x16. Using PCIe 2.0 x1, this type of storage can saturate the PCIe bus, providing a maximum data transfer speed of 500 MB/s. This makes PCIe SSDs an option for the CubeCAT.

QSPI QSPI (Quad SPI) can be used to interface with NOR flash devices[58] or interface with SD cards (in normal SPI mode). Although it can achieve good speeds capacity is limited for the NOR flash case or speed is limited for the SD card use-case. Connecting multiple flash devices with different chip selects can improve total capacity. This makes this option not viable for CubeCAT.

UFS UFS (Universal Flash Storage) is a standard developed to replace eMMC and SD, providing a faster interface and increased reliability. Improved task queuing also improves performance in time-critical applications[59]. With data transfer rates of a maximum of 23.2 Gb/s[60] the standard also fits the CubeCAT use case and can even meet the performance needed for Requirement ‘10 Gb/s speed‘.

3.3. Data Management Unit build-up

From the design options mentioned in Section 3.2 different choices can be made to support the 100 Mb/s of Requirement ‘100 Mb/s speed‘, the 500 Mb/s of Requirement ‘500 Mb/s speed‘, the 1 Gb/s of Requirement ‘1 Gb/s speed‘ and the 10 Gb/s of Requirement ‘10 Gb/s speed‘. This section discusses 3 designs that were discussed during the design of the DMU.

All designs were based on the *microprocessor with FPGA* architecture. This was due to the fact that using a microprocessor would allow the use of Linux and re-use drivers that are already developed and tested. Given Requirement ‘Time‘ this architecture would allow more time to be spent on the FPGA, whilst allowing the whole architecture to go up to 10 Gb/s. Where an architecture with a microcontroller would have troubles with the read speed of the storage, the microprocessor architecture allows commodity hardware that is already capable of reaching this speed. For storage an eMMC chip would be used, allowing speed up to 400 MB/s, which would be fast enough to meet Requirement ‘1 Gb/s speed‘ and fulfilling Requirement ‘Storage‘.

All considered designs use the ECP5-5G as FPGA. With its small size, build-in SEU mitigation and low power it matched the best with Requirement ‘Size‘ and Requirement ‘Power‘. Additionally, it has 5 Gb/s LVDS transceivers, meeting Requirement ‘1 Gb/s speed‘ and Requirement ‘LVDS‘.

To enable the channel interleaver to interleave multiple milliseconds of data an external memory is used. This memory is in all designs a 8 Gbit LPDDR3 chip, directly connected to the ECP5-5G.

For the satellite interface all designs have an available USB port to meet Requirement ‘USB‘. Besides, this port or any serial port can be used to provide the RS-485 port for Requirement ‘RS-485‘ and Requirement ‘Satellite interface‘ as well by means of a converter. Furthermore, all designs have available U(S)ART ports for Requirement ‘Quad Cell interface‘.

3.3.1. HummingBoard Pulse with ECP5-5G via PCIe

The first design was based on the HummingBoard Pulse[61]. This board would be connected to the ECP5-5G via PCIe. The HummingBoard was chosen because of the fast processor and the inherent radiation resistance of the process it was build with. Additionally, it has a PCIe connector, allowing the use of a fast and scalable interconnection.

During later iterations the ECP5-5G FPGA could be replaced by an FPGA that would have 10+ Gb/s transceivers and could work with the same driver. Unfortunately, during the development of this solution it was found that the development board had no connection routed to its mPCIe slot, making it unable to test with.

At that time, with the early availability of the NXP i.MX 8X chip it was decided to move to that chip for its availability of ECC on its RAM. The development that was done on the HummingBoard can be read in Appendix B.

3.3.2. TQMa8Xx with ECP5-5G via PCIe

The i.MX 8X is a processor that is designed with safety and reliability in mind. This fits better with the LEO environment the CubeCAT will be in and thus this processor was favoured over the i.MX 8M of the HummingBoard. The peripherals are also better suited for the intended architecture, allowing a faster connection to the FPGA.

At the time the switch was made, the timeline became more stringent and a back-up solution was sought to reduce the risk of the first demonstration mission. This risk reduction was found by also adding the CP400.85 on the PCB as well, with a connection to the FPGA via QSPI and, if the PCB allowed, the EBI.

After a few weeks this whole solution was dropped in favour of only using the CP400.85 due to height problems in the 3D design.

3.3.3. CP400.85 with ECP5-5G via QSPI and EBI

The reason to choose the CP400.85 came down to three things. One the hardware has flight heritage, which is important in the space industry. Second, the size of the module eases the PCB design and third, Hyperion has experience with this board.

With a connection via QSPI the system would be able to meet Requirement ‘100 Mb/s speed‘ and Requirement ‘500 Mb/s speed‘. This connection is also available on the development board of the CP400.85, allowing the start of development for this interface first. On an integrated version with the FPGA the EBI would be available. This interface allows a speed over 1 Gb/s between the processor and the FPGA, but the read speed of the storage should be able to keep up with this rate. With a theoretical maximum of 960 Mb/s, this is quite troublesome. Luckily the output rate of the DMU should reach 1 Gb/s, meaning that before the encoding the incoming rate needs to be around $\frac{223}{255} = 0.87$ times the output rate. This would mean that this design meets Requirement ‘Hardware‘ and thus qualifies as a suitable design, effectively achieving Goal **A**.

The design build-up is visualized in Figure 3.12.

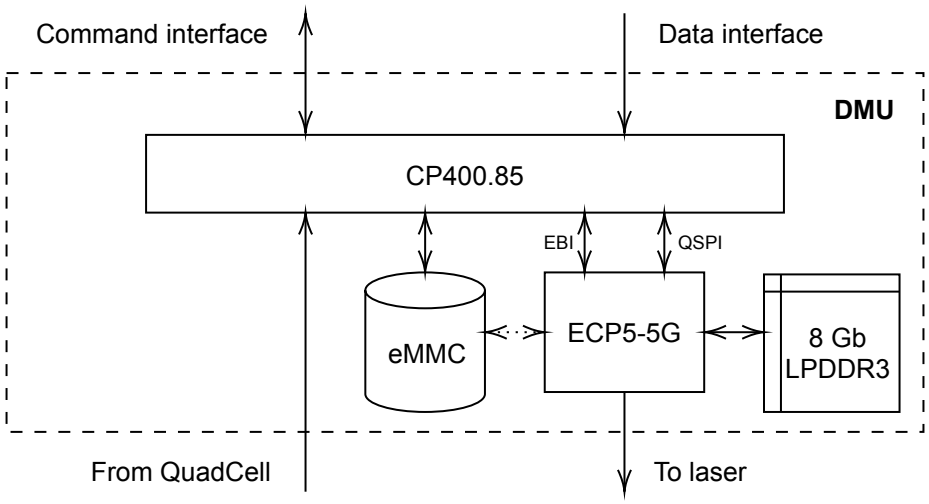


Figure 3.12: DMU build-up with the CP400.85 and the ECP5-5G.

3.4. Subsystem responsibilities

With the design explained in Section 3.3, the subsystems and their responsibilities will be further detailed. The subsystems consist of three parts: the processor, the FPGA and the storage. With the storage only responsible for safely storing the data, only the FPGA, and the processor will be further discussed. As the processor responsibilities are executed by software, software is used to refer to these responsibilities in this report.

Shared responsibilities between the FPGA and the processor are the power requirement (Requirement ‘Power‘), the deadline (Requirement ‘Time‘), and meeting the speed target.

3.4.1. FPGA responsibilities

The first reason to use an FPGA is the availability of LVDS transceivers on it, to meet Requirement 'LVDS'. Besides, the FPGA is responsible for the encoding of the data stream according to the TNO3k standard. Part of the standard is the channel interleaver, which requires external memory for which the hardware is available later.

To interface with the processor the FPGA is responsible for providing a QSPI and later, when the hardware is available, an EBI interface. These responsibilities are summarized in Table 3.4.

Table 3.4: FPGA responsibilities. The *Importance* column specifies the importance for the CubeCAT project.

Key	Importance	Description
TNO3k	High	The FPGA should encode the data according to the TNO3k standard.
3 ms channel interleaver	Low	The TNO3k encoder should have a channel interleaver depth interleaving 3 ms of data.
100 ms channel interleaver	Low	The TNO3k encoder should have a channel interleaver depth interleaving 100 ms of data.
300 ms channel interleaver	Low	The TNO3k encoder should have a channel interleaver depth interleaving 300 ms of data.
LVDS	High	The FPGA output should be LVDS.
QSPI	High	The FPGA should provide a QSPI interface.
EBI	Low	The FPGA should provide an EBI interface.

3.4.2. Processor/ Software responsibilities

Compared to the FPGA the software has to focus more on interfacing rather than data manipulation. It has to interface with the *Quad Cell* (Requirement 'Quad Cell interface'), the satellite (Requirement 'Satellite interface'), and the FPGA. These interfaces are provided using USB (Requirement 'USB') and RS-485 (Requirement 'RS-485'), and are needed to handle incoming data and incoming commands. The interfaces to the FPGA that need to be provided are QSPI and EBI. These responsibilities are summarized in Table 3.5.

Table 3.5: Software responsibilities. The *Importance* column specifies the importance for the CubeCAT project.

Key	Importance	Description
RS-485	High	The DMU should be able to interface with RS-485.
USB	High	The DMU should be able to interface with USB.
Quad Cell interface	High	The DMU should be able to interface with the *Quad Cell* with U(S)ART.
Satellite interface	High	The DMU should be able to process U(S)ART commands from the satellite.
QSPI	High	The software should drive the QSPI interface.
EBI	Low	The software should drive the EBI interface.

With these responsibilities not everything is covered for the software. Some details surface when the software is further specified. As such software tasks have been defined, namely: *packetization*, *satellite command interface*, *satellite data interface*, *quad cell interface*, and *FPGA interface*. In the following subsubsections these tasks are, in order, further discussed. These tasks are specified to run on Linux/Debian.

Packetization

The data sent to the processor is of unknown size and thus a mechanism needs to be provided that ensures the data size is a multiple of 2048 bytes. Compared to the OSI model[62] the CCSDS has split the *Data Link Layer* in two: a *Data Link Protocol Sublayer* and the *Synchronization and Channel Coding Sublayer*. Packetization is done in the *Data Link Protocol Sublayer* and the TNO3k standard describes only the *Synchronization and Channel Coding Sublayer*, meaning that a mechanism for packetization

needs to be provided by the software. The method for packetization is not yet known and as such this topic needs to be revisited once it is decided upon.

The packetization can happen at two locations: in the *FPGA interface* or at the moment the data is stored. If the data is packetized at the *FPGA interface* a run time overhead is added, possibly reducing the throughput. If the data is packetized at the moment the data is stored the amount of data stored is increased.

Satellite command interface

The physical interface for the satellite is either RS-485 or USB as dictated by Software responsibility 'RS-485' and Software responsibility 'USB'. The RS-485 interface can be provided by using a serial interface of the processor. Furthermore the USB port can provide a serial channel, meaning that both interfaces are accessed as tty-ports in Linux.

Code has to be written that uses a tty-port and from there parses the commands. Internally Hyperion has developed the Hyperion Protocol[63] that defines the structure of the commands and handling of commands. Once a set of desired commands has been made these need to be implemented for Software responsibility 'Satellite interface'. As such, the task of the *Satellite interface* needs to revisited once these commands are known.

Satellite data interface

The data sent to the DMU is sent via a different (virtual) route than the commands. When the data arrives via the RS-485 interface a command has to be made that stores the data. When the USB port is used the CP400.85 can act as a gadget device, providing a mass storage point or an ethernet connection. All data sent through this connection needs to be stored.

Depending on the desires of the satellite, different entry points need to be implemented. For example, for the ethernet connection a REST API can be made where data gets posted to. The data is then, if needed, packetized and stored.

Quad cell interface

The *Quad Cell* provides uplink data and sensor data. This data is communicated over a UART link and, as the *Quad Cell* is made by Hyperion, the Hyperion Protocol is used to transfer data to the processor. Once the set of available data is defined a set of commands can be made that retrieves this data from the *Quad Cell*. These commands then need to be implemented for Software responsibility 'Quad Cell interface'. As such the task of the *Quad cell interface* needs to revisited once the commands are known.

FPGA interface

The *FPGA interface* consists of two tasks. One task is communicating with the FPGA and the other task is to send the stored data to the FPGA when an optical link is established. These tasks will be further discussed in the following paragraphs.

FPGA communication FPGA communication is needed for the high-speed data path in the DMU. To interface with the FPGA two interfaces have been defined: QSPI according to Software responsibility 'QSPI' and EBI according to Software responsibility 'EBI'. The data streamed to the FPGA is expected to be a multiple of 2048 bytes, due to the TNO3k standard. Both interfaces are capable of this and thus no extra work needs to be done to correct this.

Then a way needs to be provided to Linux user space to send data to the FPGA. The most common way to represent a device in Linux is to map it to either a character or a block device. With a character device being byte-oriented and a block device meant to transfer blocks of data.

To minimize the involvement of the actual processor during transmission DMA can be used. Minimizing the involvement of the processor reduces stutter in the transmission and thus improves the average throughput. For this to work the peripheral needs a DMA implementation, as well as the source of the data.

Sending data When an optical link is established the data needs to be loaded from the storage and send to the FPGA. This can be implemented by means of a simple BASH script that executes a `dd` with the data. Another option is to write a program that provides this functionality.

3.5. Conclusion

The DMU is responsible for hosting the high-speed digital data path, handling commands, and processing data from the satellite. Inside the CubeCAT system it also interfaces with the *Quad Cell* and the laser driver. All this is needed to enable the main tasks of the DMU: data aggregation, data encoding, and data output.

Enabling these tasks can be done by a multitude of architectures, components, and connection. The finalised design uses the Hyperion CP400.85 as microprocessor, the Lattice ECP5-5G as FPGA and eMMC for storage, effectively achieving Goal **A**. In the first iteration the FPGA is connected via QSPI to enable 100 Mb/s and 500 Mb/s output speeds. When the first design is produced the EBI can be used to get to an 1 Gb/s output speed. In this design the FPGA is responsible for interfacing with the processor, encoding the data stream, and providing an LVDS output to the *Laser Driver*. The software is responsible for packetization, processing satellite commands, handling satellite data, communicating with the *Quad Cell* and interfacing with the FPGA.

4

Data Path Implementation

This chapter discusses the implementation of the high-speed data path on the Data Management Unit for Goal **B**. The DMU consists of two subsystems: the FPGA and the software.

For the FPGA an architecture has been devised that is designed to fulfil its responsibilities. A discussion is given about the role each sub-block plays in this architecture, with extra attention to the TNO3k encoder for Goal **C**. The responsibilities set for the software are converted into tasks that detail the working of the software. From these tasks the implementation of the QSPI driver is further discussed as this driver is required for the high-speed data path.

Both the discussions of the FPGA and the software are based on the system architecture of Section 3.3. The responsibilities for the FPGA and the software are set out in respectively Subsection 3.4.1 and Subsection 3.4.2. It is recommended to have read these sections before reading this chapter.

This chapter is structured as follows: first, the setup and tools used for the implementation on the FPGA and software can be read in Section 4.1. This is followed by the architecture and implementation of the FPGA in Section 4.2. Part of this architecture is the QSPI peripheral that can be found in Subsection 4.2.3 and the encoder that can be found in Subsection 4.2.8. In Section 4.3 a discussion of the software implementation for the high-speed data path can be found. Lastly, in Section 4.4 a conclusion is given, referring back to the set responsibilities for the FPGA and the software.

4.1. Tools

Since the FPGA chip is produced by Lattice, the *Lattice Diamond* toolchain is used for synthesising the HDL code. This toolchain has two synthesizers: the integrated LSE (Lattice Synthesis Engine) and the bundled *Synplify Pro*. To be able to use the latest IP, *Synplify Pro* has to be used. For simulation Lattice bundles *Aldec Active-HDL* with *Lattice Diamond*.

For editing the code, *Lattice Diamond* has an integrated text editor. This editor is limited to syntax highlighting, which is limited compared to editors for programming languages, e.g. *Eclipse*[64], *IntelliJ*[65], *Visual Studio Code*[66].

As such, another editor has been used: *Visual Studio Code*. It has been used together with the *VHDL Formatter*[67] extension for code linting/ formatting and the *VHDL LS*[68] extension for snippet suggestions, syntax highlighting and parsing, code suggestions, and declaration resolving. *Visual Studio Code* has also been used to develop the software for the CP400.85, making use of the *C/C++* extension.

All HDL code is written in VHDL, as this is the HDL language learned by the author, as well as the companies HDL language. The version of VHDL used is VHDL-2002. For the software the Linux kernel source tree has been used. As such, the C version used is C90.

4.2. FPGA Architecture and Implementation

The FPGA is responsible for encoding the data stream according to the TNO3k standard. Furthermore, it is also responsible for driving an LVDS signal to the output driver. The data stream comes from the processor, for which the FPGA needs to implement an I/O peripheral. These blocks are summarized in Figure 4.1.

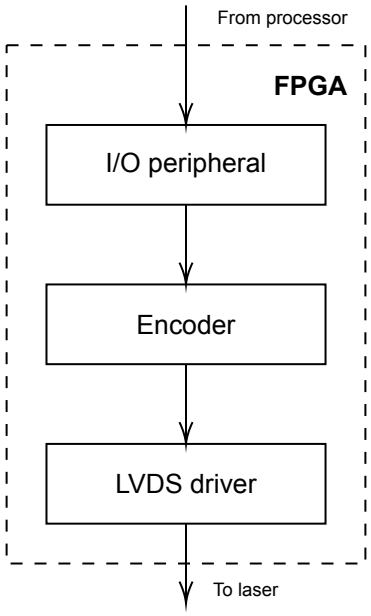


Figure 4.1: FPGA Functional Block overview

Based on the functional overview of Figure 4.1 an architecture has been designed. This architecture of the FPGA is designed to enable the core responsibilities devised in Subsection 3.4.1, but also to enable flexibility in replacing components and debuggability. This is done by integrating a separate control path besides the main data path. In the main data path multiple MUXes are used to change the data flow, providing a way for direct data output without encoding and a way for providing a data source with known sequences.

All interfaces for the main data flow are 64 bit wide words and adhere to the Avalon ST standard (see Section C.2). This width has been chosen due to the width of the Reed-Solomon Interleaved block of Section 4.2.8. This has the advantage that it reaches Requirement '10 Gb/s speed' with a 156.25+ MHz clock on the main data path.

An overview of the architecture can be found in Figure 4.2. The blocks from the architecture are further discussed in the following subsections.

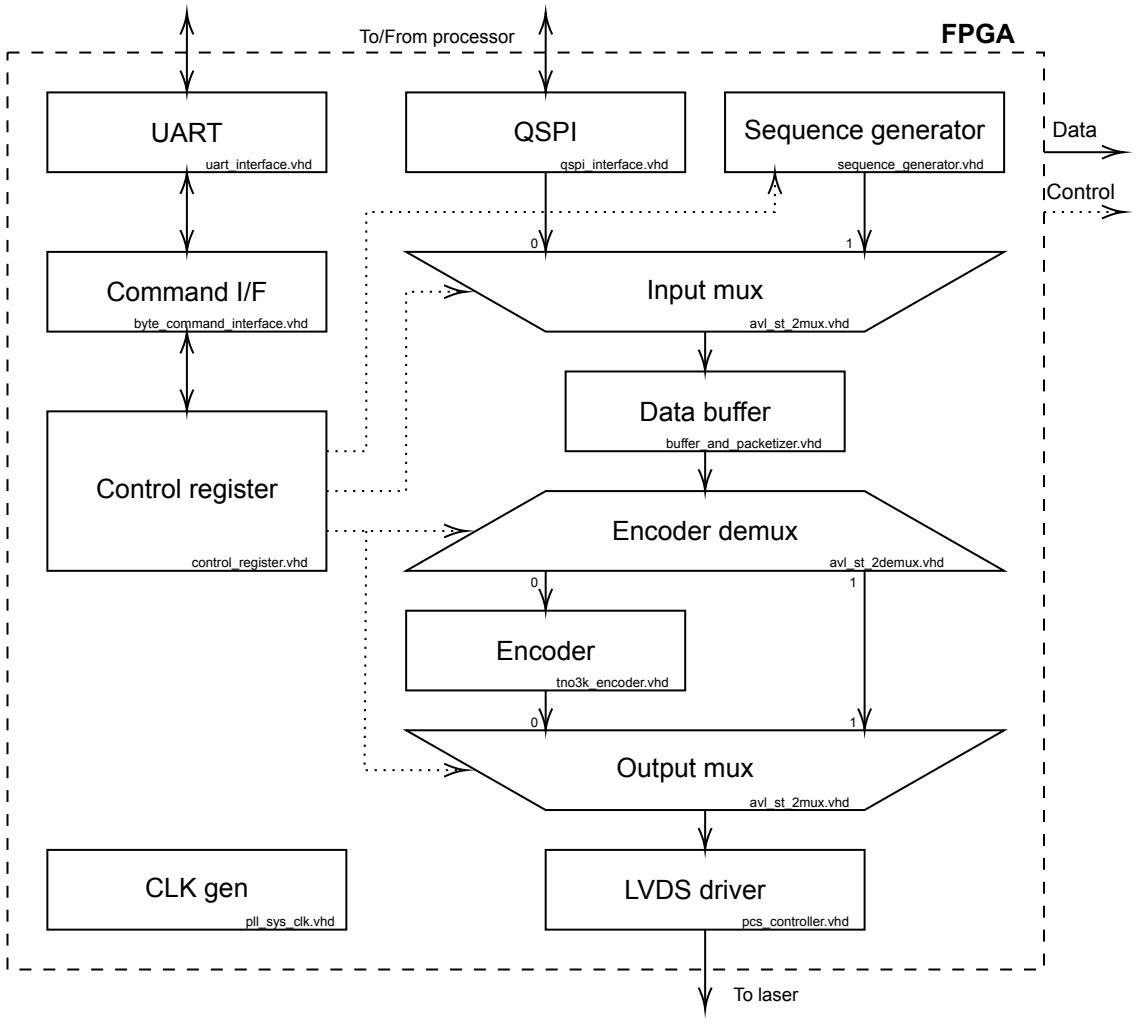


Figure 4.2: FPGA architecture overview

4.2.1. Control interface

To support multiple data paths, and sources, control signals are needed to control the data paths. A control interface is implemented to change the settings and can be further extended if needed. This control interface can be accessed via a UART interface, which is connected to a command interface. The command interface handles the data and if needed updates the control register. These blocks will be further discussed as follows.

UART

The UART is based on the open source reference design from Lattice[69]. This design consists of multiple components to have a fully configurable NS16450 compatible UART. As we only need the receiver and the transmitter only these components are used.

The UART operates in a different clock domain than the main data path of the system. As such a dual clock FIFO is used to decouple the system interface with the UART operation. The internal baud clock is derived from a clock divider (counter) based on the system clock. This baud clock cannot be directly generated from an IP block as this clock is too slow for the available IP blocks.

Command interface

With the physical interface handled by the UART the commands need to be interpreted. The *Command interface* has been implemented as a byte oriented command interface, such that the UART can be replaced by any byte oriented physical interface.

To initiate communication the *Command interface* expects the character 'w' (write) or 'r' (read) for command. The write command must be followed by four bytes to form a 32 bit word, which is the width

of the current *Control register*. After the expected number of bytes is received an update is sent to the *Control register*. If the read command is received the current value of the *Control register* is sent back to the requester.

Control register

The base of the *Control interface* is the *Control register*. It is based around a register, which in the current implementation is 32 bits wide. Lines controlling for example the input mux are directly connected to this register.

A mechanism is provided to read and write the register value. Bit modifications need to be done by a read-modify-write operation.

4.2.2. Clock generator

To provide the system with a clock at a desired frequency a *clock generator* is used. This IP block is connected to an external crystal of a known frequency and creates (if needed) multiple output frequencies. A *clock generator* is needed because the QSPI peripheral doesn't have a constant clock or an oscillator, thus an external clock source is needed.

4.2.3. QSPI peripheral

The *QSPI peripheral* is required for FPGA responsibility 'QSPI'. As the CP400.85 has a specialized SPI-NOR QSPI controller and Linux already having drivers for SPI-NOR memory the SPI-NOR standard is used as a basis. Here the FPGA acts as a slave device. As later became apparent during the development of the specialized QSPI driver (Section 4.3), this was not fully needed. This insight, which will be discussed later, has not been applied and thus the peripheral has been structured to support the SPI-NOR standard.

From the system perspective, it interfaces with the QSPI interface. This block decouples the system clock domain from the QSPI clock domain. Connected to the QSPI interface is the QSPI controller, which handles: the SPI-NOR commands, issues commands for the correct pin settings to the PHY, and the data. A PHY block has been made to ensure the correct pin settings and sampling settings are applied, avoiding short circuits.

In Figure 4.3 an overview can be seen of the QSPI architecture. These blocks will be further discussed in the following subsubsections.

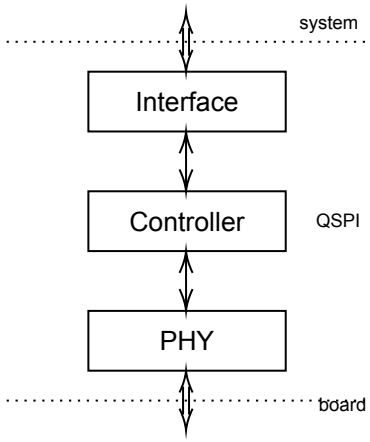


Figure 4.3: QSPI architecture overview

Interface

As mentioned before the *Interface* decouples the two clock domains. This is implemented by means a dual clock FIFO. Extra logic is used to comply with the Avalon standard for the system interface.

Controller

Commands received on the byte interface of the PHY are parsed by the *Controller*. Following the SPI-NOR standard, only one command can be given during the selection of a chip. A SPI-NOR command

consists of 5 different possible phases: command, address, dummy, read, and write. Depending on the command sent during the command phase different phases may follow.

The SPI-NOR command handling is implemented by resetting the controller when the FPGA is deselected, meaning that on selection of the FPGA the controller will always expect to receive a command. Then, when a full command has been received, the PHY is updated with the settings for this command and the controller switches to the relevant control flow.

During the selected control flow the PHY is then further updated if needed, for example during a *read data* command. This command uses the address phase to select the read address and the read phase to read the data, starting from the selected read address.

When a command is finished the controller is put in an idle state, until reset. This can happen when there is no more data left to be read from the controller, but the QSPI master keeps reading. In theory this idle state should never be triggered. If it happens, due to radiation for example, the *Controller* has predefined behaviour.

The *Controller* has been implemented to partially mimic the behaviour of a Windbond w25q32jv spiflash device[70], such that it later can be recognised by the QSPI driver. This also implies that only SPI mode 0 and 3 were implemented.

PHY

The core task of the PHY is handling the I/O pins. This is done to prevent short circuits. Besides, it is also used to signal when a new valid byte is received on the interface, irrelevant of the amount of I/O pins used for communication.

The pin control is implemented by a process that triggers on both edges of the QSPI clock line. The rising edge is used to sample data and the falling edge is used for the control flow. This control flow includes keeping track of, and signalling valid bytes (read or write). This process is restarted when the chip is deselected.

Handling the pins is done by a different process that knows the state of the PHY (read, write, idle) and correctly drives or samples the requested number of pins.

4.2.4. Sequence generator

In the FPGA the *Sequence generator* can be used to generate a sequence which is known to the receiver. This can be useful for debugging or channel estimation. It generates transfer frames of 2048 bytes as input for the *Encoder*. This size is defined by the TNO3k standard. Implemented in sequences are a binary counter per transfer frame and a black-white pattern.

4.2.5. Input mux

The *Input mux* is used to select the data source for the main data flow. This can either be the *QSPI* peripheral or the *Sequence Generator*.

4.2.6. Data buffer

The *Data buffer* is used to smoothen out the bursty nature of the QSPI peripheral. Besides, it packetizes the data by adding *start of packet* and *end of packet* signals to the data. This is done per transfer frame of 2048 bytes as required by the TNO3k standard. For this the *Data buffer* assumes the data received is sent per multiple of 2048 bytes. It is implemented by using a large FIFO and extra logic for the interfaces and packetization. When the FIFO is empty an *end of stream* signal is raised on the output interface. The packetization is done by making use of a counter.

4.2.7. Encoder demux

The *Encoder demux* is connected to the same selection signal as the *Output mux*. With this deMUX the *Data buffer* is either connected to the *Encoder* or via the *Output mux* to the *LVDS driver*.

4.2.8. Encoder

The core of the design is the *Encoder*. This block encodes an incoming data stream according to the TNO3k standard for FPGA responsibility 'TNO3k'. The implementation of the *Encoder* does not strictly follow the block division devised in the standard. This was done because of the possibility to reuse and combine blocks. However, the functionality described by the standard is followed.

As input to the *Encoder* data packets of 2048 bytes is defined by the TNO3k standard. When the end of the data packets stream is signalled on the *Encoder* and the *ASM* is done processing data, then and *end of stream* signal is toggled for the *Slicer*.

An overview of the implemented blocks can be seen in Figure 4.4. These blocks will be further discussed in this subsection.

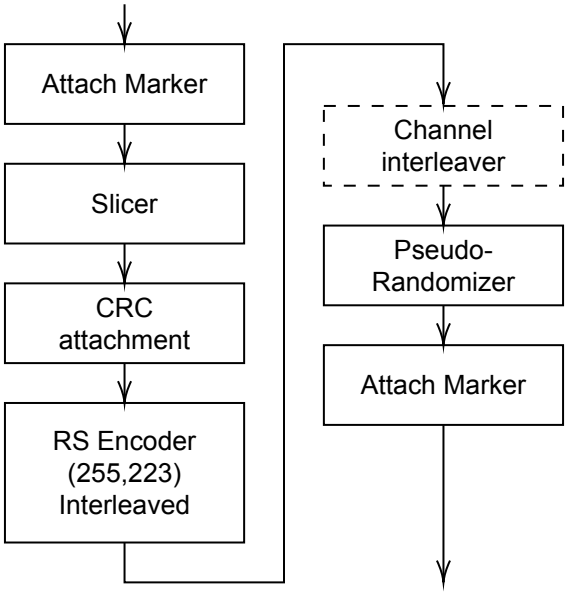


Figure 4.4: TNO3k encoder implementation overview

Attach (Sync) Marker

To synchronize the decoder a marker is added to the data stream. As the ASM and the CSM block in the standard both add the same marker at the same location in the data stream (prepend `0x1ACFFC1D`) a common HDL block has been implemented.

This implementation is based on the detection of the *start of packet* flag defined in the Avalon standard. When this flag is detected the marker is added to the stream. As the marker is only 32 bits long and the transfer frames are 2048 bytes long (for the ASM), the placement of this marker is not fixed. It needs to be placed in the first or second 32 bits of the output word. This is done by storing the other 32 bits of the input word. When two markers have passed, the stored data is flushed. This implementation doesn't affect the behaviour of the CSM.

Slicer

A *slicer* is used to finalise the data stream by appending zeroes to the stream if the stream ends, which is signalled by an *end of stream* signal. The amount of zeroes added is based on how many data bytes are needed for a complete Reed-Solomon encoding.

The Reed-Solomon encoding needs 223 bytes. As the *CRC attachment* adds 32 bits to the stream, only 219 bytes are needed per RS-data block. With a symbol interleaver depth of 8, 1752 bytes are needed for a complete Reed-Solomon encoding. Thus, the *slicer* appends, if needed, zeroes until a block of 1752 bytes have passed.

CRC Attachment

To check the integrity of the data block received a CRC is appended to each RS-data block. This CRC is a CRC-32 that is calculated per RS-data block of 219 bytes, meaning that for 8 RS-data blocks the added CRC shifts one position per block.

To implement the *CRC attachment* a state machine was defined. This state machine starts with feeding the input data to the CRC and counting how many bytes have passed. Input data is directly passed on to the output. When there are less than 8 bytes (input word) left to CRC, it switches to a state where the remainder of the needed bytes are fed to the CRC. If there are 0 remaining bytes left in either the start state or the remainder state, the state switches to a state where the calculated CRC

is put on the output and the CRC is resetted to its initial value. The process of calculating can then restart if there are no remaining bytes left on the input. When there are remaining bytes left another state is entered. This state calculates the CRC over the remainder of the input and once it has done that, it will go back to the start state. This state machine has been visualized in Figure 4.5.

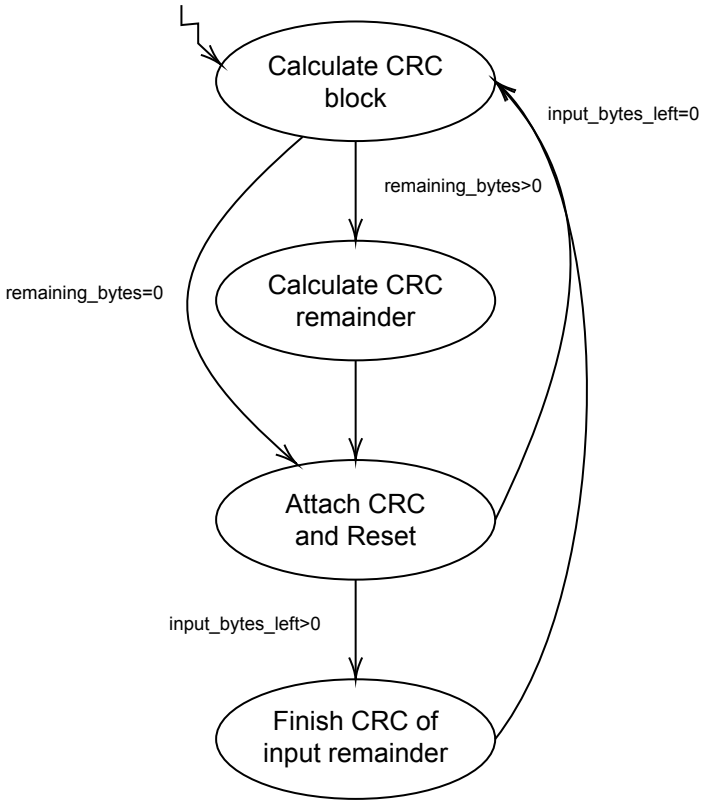


Figure 4.5: CRC implementation state diagram

For the actual CRC calculation a VHDL function has been generated using the online tool from easics[71].

Reed-Solomon Interleaved

The base of the TNO3k encoding is the Reed-Solomon(223,255) encoder. It encodes the incoming bytes in such a way that later, in the decoder of the ground station, the data that is received can be repaired, if needed. The description of the Reed-Solomon variant used can be found in [72, section 4]. In the TNO3k standard the encoder is followed by the symbol interleaver. In [72, section 4.4.1] a method is described where the encoder and the symbol interleaver are 'merged' into one functional block. This method is used to implement both the Reed-Solomon encoder and the symbol interleaver, hence the name of this section *Reed-Solomon Interleaved*.

The symbol interleaver part of the merge is realised by repeating the encoder blocks for N times the interleaver depth. As the interleaver depth has been chosen to be 8, 8 Reed-Solomon encoders are instantiated, with each one byte of an input word. The reason a 64-bit data path is implemented is because of this interleaver depth.

For the Reed-Solomon encoder two states can be defined: one where the data is passed through to the output and one where the check symbols are outputted. In the check symbols output state the Reed-Solomon encoder blocks any further input.

To calculate the check symbols each input byte is modulo-2 added with a generator polynomial. This can be implemented based on linear-feedback shift registers, as explained in [73]. As the standard defines the data should be represented in a dual basis symbol representation to get the functionality of a Berlekamp Reed-Solomon encoder. This conversion is called tracing. As can be read in [72, Annex F], the input data has to be inverse traced, then put through the built 'conventional' Reed-Solomon

encoder and then traced back. This tracing can be implemented by a matrix multiplication or by using a generator polynomial[74].

Due to the nearing deadline of Requirement 'Time' and, at the time, the amount of work left to be done, this block was implemented by TNO and shared with Hyperion. This also had the advantage of validating the Reed-Solomon encoder against the Reed-Solomon decoder, as TNO is responsible for that.

Channel interleaver (not implemented))

The *Channel interleaver* is used to spread blocks of data over longer periods of time to enhance the performance of the link during a fade and is required for FPGA responsibility '3 ms channel interleaver', FPGA responsibility '100 ms channel interleaver' and FPGA responsibility '300 ms channel interleaver'. This interleaving of the data is defined in the same way as the CCSDS 142 standard, by means of multiple, in depth increasing, queues and two selection arms. These arms select a queue in increasing manner modulo the amount of queues.

To implement this a significant amount of data needs to be stored to interleave 3 ms for FPGA responsibility '3 ms channel interleaver'. That is why external memory is needed to support this feature of the TNO3k standard. Due to time limitations, lack of hardware and limited relevance (only effective against fades), TNO and Hyperion agreed that the *Channel interleaver* is left to be implemented at a later stage.

Pseudo-Randomizer

The *Pseudo-Randomizer* is used to avoid long-runs that potentially unlock the receiver. Besides, it is also used to average the bit-entropy to 0.5. Preferably, the output would be pure random noise. But, to ensure the modulation is known for the receiver a known sequence is modulated on top of the data.

With the *Pseudo-Randomizer* of the TNO3k standard based on the CCSDS 142 standard the same generator polynomial is used. The output of this generator is then fed into a modulo-2 adder with as other input a bitstream coming from the *Channel interleaver*. With each bit the generator is also updated.

To increase parallelism this operation is expanded to a byte operation, which is then applied byte per byte on the input word of the *Pseudo-Randomizer*. Due to the nearing deadline of Requirement 'Time' and, at the time, the amount of work left to be done, this block was implemented by TNO and shared with Hyperion.

4.2.9. Output mux

The *Output mux* is connected to the same selection signal as the *Encoder demux*. With this MUX either the *Encoder* output can be connected to the *LVDS driver* or output from the *Data buffer*.

4.2.10. LVDS driver

As the block responsible for driving an LVDS signal to the *laser driver* (FPGA responsibility 'LVDS') the block controls one of the high speed transceivers on the FPGA. This transceiver is directly connected to a *PCS* IP block, which incorporates all functionality needed for a SerDes. The output pin is directly connected to this IP block, requiring to interface with it to provide the LVDS signal.

Data is fed to the transceiver in a different clock domain, as well as a different width. The *LVDS driver* uses a dual clock FIFO, with a 64 bit input size and a 16 bit output size to decouple the clock domains and change the data width. Connected to the transceiver side of the FIFO is extra logic to interface with the transceiver IP block.

4.3. Software Implementation

The operation of the CP400.85 is defined by the software that is run by it. With the CP400.85 responsible for the satellite interfaces, the *Quad Cell* interface and the data interfaces to the FPGA the CP400.85 needs to handle multiple tasks at once. This section discusses how the software is implemented to support the high-speed data path. For this the *QSPI Driver* is implemented.

The *QSPI Driver* needs to provide communication with the FPGA. This is done by building a driver stack that both controls the QSPI peripheral of the processor and provides an interface to user space. For the implementation a transfer speed of 100 Mb/s has been targeted to meet Requirement '100 Mb/s

speed‘. Four iterations of this driver have been made. First, the implementation fully based on the *SPI-NOR* framework will be discussed in Subsection 4.3.1. This is followed by a discussion in Subsection 4.3.2 about an iteration where the *SPI-MEM* controller has been customized. In Subsection 4.3.3 an iteration is discussed where the QSPI driver was upgraded with DMA. Lastly, in Subsection 4.3.4 an integrated driver is discussed.

4.3.1. SPI-NOR

With the FPGA acting as a SPI-NOR device the SPI-NOR[75] framework can be reused. This SPI-NOR framework is built on top of a generic SPI-NOR controller (m25p80), the SPI-MEM framework and then a SPI controller driver.

Already available in the Linux kernel was the QSPI controller driver that interfaces with the SPI-MEM framework, requiring only effort to make sure the m25p80 driver recognises the FPGA. This effort has been made on the FPGA, with it mimicking the Winbond w25q32jv spiflash device[70].

With this implementation the FPGA presents itself as an MTD device, which allows limited flexibility considering direct byte access. As such this implementation was dropped.

4.3.2. Custom SPI-MEM controller

Building further on the SPI-NOR stack in Linux the m25p80 driver was edited to remove the MTD bindings and implement a character device that allowed byte access. With a specific entry in the Device Tree the FPGA didn’t require a detection routine and this routine was thus removed from the driver.

After implementation it was found that the throughput that was possibly by this driver stack was too low and as such, this implementation was iterated upon.

4.3.3. QSPI driver DMA

Reusing the *Custom SPI-MEM controller* the actual QSPI driver was modified to support DMA. The advantage of DMA is that the processor involvement is limited to starting and finishing a transmission, rather than sending bytes individually.

The QSPI peripheral has a DMA channel for receiving and transmitting. Providing the data that needs to be sent is done from a DMA-able buffer. This buffer is allocated per transaction and the data is safely copied to it from user space.

Once this implementation was made the speed was sufficient, but having an API overhead by the SPI-MEM framework it was decided to create an integrated driver.

4.3.4. Integrated driver

The *Integrated driver* combines the *QSPI driver DMA* and the character device implementation from the *Custom SPI-MEM controller* into one driver. This removes the complete overhead of any framework in between the layers. As will be discussed in Subsection 5.4.1, this implementation was found to be satisfactory for Requirement ‘100 Mb/s speed‘.

4.4. Conclusion

The architecture of the FPGA was designed with the set responsibilities in mind. With the QSPI peripheral, the *Encoder* and the *LVDS driver* FPGA responsibility ‘TNO3k‘, FPGA responsibility ‘QSPI‘ and FPGA responsibility ‘LVDS‘ are met. All high importance responsibilities have been fulfilled and with a data path of 64 bits wide the FPGA was designed with the 10 Gb/s target of Requirement ‘10 Gb/s speed‘ in mind. The implementation of the channel interleaver and the implementation of the EBI are still left to be done, missing responsibilities FPGA responsibility ‘3 ms channel interleaver‘, FPGA responsibility ‘100 ms channel interleaver‘, FPGA responsibility ‘300 ms channel interleaver‘ and FPGA responsibility ‘EBI‘.

To support the high-speed data path a QSPI driver is needed. With the implementation of the QSPI driver Software responsibility ‘QSPI‘ is fulfilled. The QSPI driver is implemented as a customized Linux driver that uses DMA for transactions. Together with the FPGA implementation an implementation for the high-speed digital data path has been made.

5

Testing and validation

This chapter discusses the testing and validation of the implemented parts of the DMU for the high-speed data path. It discusses the verification techniques and results, which are then matched to the set requirements for the DMU. This is done to verify Goal **B** and Goal **C**.

For this chapter it is expected from the reader to have read Section 3.1. Additionally, it is advised for the reader to have read Chapter 4.

This chapter is structured in the following way: it starts with discussing the available tools and the used setup in Section 5.1. This is followed by further discussing the testing and validation of the FPGA parts, focussing on the encoder and the *QSPI peripheral*. In Section 5.3 the test of the *QSPI driver* can be read. An integrated system test can be found in Section 5.4, which is split up in the *QSPI communication* and the whole system. After this the system power consumption is discussed in Section 5.5, followed by the conclusion that is found in Section 5.6.

5.1. Tools and setup

To verify the QSPI communication two different logic analysers have been used. The first one is an Open Bench Logic Sniffer[76] and the second one is a Saleae Logic Pro 16[77].

Verification of the HDL modules has been done using standard VHDL testbenches. For the TNO3k encoder implementation cocotb[78] has been used, together with GHDL[79] and GTKWave[80].

To capture the debug/ output data from the FPGA an Arduino Due[81] is used. The debug information produced by the FPGA is sent via UART.

The ECP5-5G FPGA board used is the ECP5-5G Evaluation Board. This board is connected to a iADCS400.OBC-DB-V0.6 board (CP400.85 development board) that hosts the CP400.85. To analyse the QSPI signals the logic analysers can be connected on the CP400.85 development board, creating a T-connection. The Arduino Due is connected to the FPGA. This setup is shown in Figure 5.1.

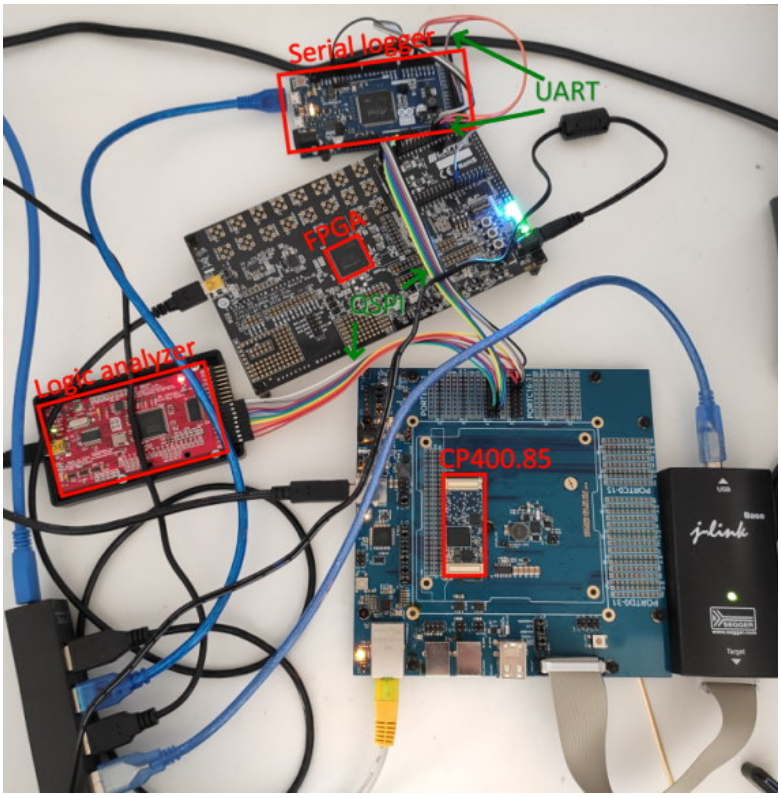


Figure 5.1: Test setup.

5.2. FPGA

As mentioned before, the HDL modules are verified using VHDL testbenches. Two HDL modules require extra attention: the *Encoder* and the QSPI peripheral. They will be further discussed in the following subsections.

5.2.1. Encoder

As mentioned before the *Encoder* has been verified using cocotb. This was done by making a model for each individual component that has been implemented for the *Encoder* (see Figure 4.4 for an overview). For each model a test was written that sends data through both the model and the VHDL module and compares the output. This test has been executed multiple times, with different parameters for data size, data content, backpressure signal modulations and input data valid signal modulations. A waveform of the *Attach Marker* test can be seen in Figure 5.2. It highlights the start and stop of a test, the valid modulation and the backpressure signal.

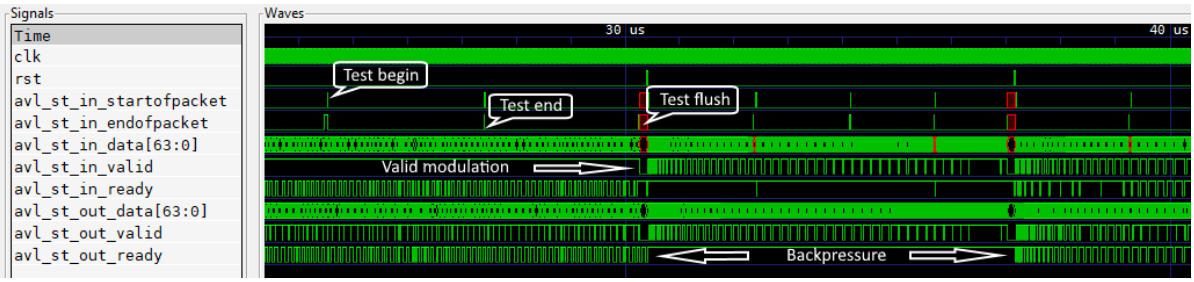


Figure 5.2: Attach Marker cocotb result in GTKWave.

After all tests of the individual components were successful, a model was by combining the other models. This model was then also tested with the afore mentioned parameters, verifying the encoder as a whole. All tests have been executed successfully, meaning that the encoder behaves according

to the TNO3k standard, achieving Goal **C**. The output of the cocotb test can be seen in the following listing.

Listing 5.1: cocotb TNO3k encoder test results

```
**************************************************************************************
** TEST                            PASS/FAIL  SIM TIME(NS)  REAL TIME(S)  RATIO(NS/S) **
**************************************************************************************
** test_tno3k_encoder.run_test_001    PASS         5340.00          1.98      2700.33  **
** test_tno3k_encoder.run_test_002    PASS         7055.00          2.22      3183.28  **
** test_tno3k_encoder.run_test_003    PASS         6180.00          2.18      2835.21  **
** test_tno3k_encoder.run_test_004    PASS        10445.00          2.78      3757.95  **
** test_tno3k_encoder.run_test_005    PASS         6305.00          2.20      2870.75  **
** test_tno3k_encoder.run_test_006    PASS         7060.00          2.27      3106.24  **
** test_tno3k_encoder.run_test_007    PASS         6635.00          2.32      2863.13  **
** test_tno3k_encoder.run_test_008    PASS        10495.00          2.91      3601.64  **
** test_tno3k_encoder.run_test_009    PASS         5545.00          2.10      2646.12  **
** test_tno3k_encoder.run_test_010    PASS         7035.00          2.39      2940.42  **
** test_tno3k_encoder.run_test_011    PASS         6225.00          2.21      2818.18  **
** test_tno3k_encoder.run_test_012    PASS        10555.00          2.73      3859.68  **
** test_tno3k_encoder.run_test_013    PASS         7595.00          2.28      3329.40  **
** test_tno3k_encoder.run_test_014    PASS         7700.00          2.29      3363.24  **
** test_tno3k_encoder.run_test_015    PASS         7835.00          2.36      3322.81  **
** test_tno3k_encoder.run_test_016    PASS        10580.00          2.72      3888.69  **
** test_tno3k_encoder.run_test_017    PASS        18180.00          7.92      2294.70  **
** test_tno3k_encoder.run_test_018    PASS        24715.00          9.89      2499.56  **
** test_tno3k_encoder.run_test_019    PASS        21370.00          8.51      2512.02  **
** test_tno3k_encoder.run_test_020    PASS        38980.00         11.03      3532.61  **
** test_tno3k_encoder.run_test_021    PASS        21690.00          8.80      2465.83  **
** test_tno3k_encoder.run_test_022    PASS        25890.00          9.10      2844.04  **
** test_tno3k_encoder.run_test_023    PASS        23145.00          9.25      2502.20  **
** test_tno3k_encoder.run_test_024    PASS        39010.00         11.82      3299.39  **
** test_tno3k_encoder.run_test_025    PASS        19115.00          8.03      2379.77  **
** test_tno3k_encoder.run_test_026    PASS        24835.00          8.81      2819.69  **
** test_tno3k_encoder.run_test_027    PASS        21505.00          8.46      2542.27  **
** test_tno3k_encoder.run_test_028    PASS        38995.00         10.73      3635.66  **
** test_tno3k_encoder.run_test_029    PASS        29000.00         10.41      2784.49  **
** test_tno3k_encoder.run_test_030    PASS        30320.00          9.91      3058.39  **
** test_tno3k_encoder.run_test_031    PASS        29450.00          9.25      3184.90  **
** test_tno3k_encoder.run_test_032    PASS        38970.00         11.22      3474.69  **
** test_tno3k_encoder.run_test_033    PASS         5340.00          2.02      2648.39  **
** test_tno3k_encoder.run_test_034    PASS         7055.00          2.26      3117.28  **
** test_tno3k_encoder.run_test_035    PASS         6160.00          2.24      2746.94  **
** test_tno3k_encoder.run_test_036    PASS        10440.00          2.80      3734.03  **
** test_tno3k_encoder.run_test_037    PASS         6305.00          2.21      2857.66  **
** test_tno3k_encoder.run_test_038    PASS         7060.00          2.30      3073.52  **
** test_tno3k_encoder.run_test_039    PASS         6720.00          3.35      2007.30  **
** test_tno3k_encoder.run_test_040    PASS        10595.00          2.77      3823.76  **
** test_tno3k_encoder.run_test_041    PASS         5550.00          2.08      2669.53  **
** test_tno3k_encoder.run_test_042    PASS         7075.00          2.32      3048.96  **
** test_tno3k_encoder.run_test_043    PASS         6205.00          2.19      2832.27  **
** test_tno3k_encoder.run_test_044    PASS        10515.00          2.74      3839.49  **
** test_tno3k_encoder.run_test_045    PASS         7540.00          2.63      2872.23  **
** test_tno3k_encoder.run_test_046    PASS         7775.00          2.88      2702.13  **
** test_tno3k_encoder.run_test_047    PASS         7795.00          2.43      3213.62  **
** test_tno3k_encoder.run_test_048    PASS        10550.00          2.96      3568.63  **
** test_tno3k_encoder.run_test_049    PASS        18180.00          8.54      2127.80  **
** test_tno3k_encoder.run_test_050    PASS        24715.00          9.51      2599.13  **
** test_tno3k_encoder.run_test_051    PASS        21495.00          9.19      2338.33  **
** test_tno3k_encoder.run_test_052    PASS        39040.00         13.13      2974.43  **
** test_tno3k_encoder.run_test_053    PASS        21690.00          9.27      2339.69  **
** test_tno3k_encoder.run_test_054    PASS        25890.00         10.05      2577.12  **
** test_tno3k_encoder.run_test_055    PASS        23070.00          9.45      2441.84  **
** test_tno3k_encoder.run_test_056    PASS        39005.00         11.45      3405.09  **
** test_tno3k_encoder.run_test_057    PASS        19110.00         11.86      1611.77  **
** test_tno3k_encoder.run_test_058    PASS        24850.00          9.77      2544.66  **
** test_tno3k_encoder.run_test_059    PASS        21480.00          9.38      2289.55  **
** test_tno3k_encoder.run_test_060    PASS        38890.00         11.32      3434.62  **
** test_tno3k_encoder.run_test_061    PASS        29160.00         10.46      2788.59  **
** test_tno3k_encoder.run_test_062    PASS        30145.00         11.44      2634.47  **
** test_tno3k_encoder.run_test_063    PASS        29770.00         10.54      2824.76  **
** test_tno3k_encoder.run_test_064    PASS        38845.00         11.21      3466.74  **
**************************************************************************************

******************************************************************************
**                            ERRORS : 0                                    **
******************************************************************************
**                         SIM TIME : 1135770.00 NS                         **
**                        REAL TIME : 398.27 S                              **
**                  SIM / REAL TIME : 2851.79 NS/S                          **
******************************************************************************
```

5.2.2. QSPI peripheral

To verify the *QSPI peripheral* two types of testbench procedures were made, one for writing data on a variable amount of lines and one for reading data on a variable amount of lines. These procedures were then combined into the command structure of SPI-NOR commands. This command structure was then used to imitate the line behaviour of a Windbond w25q32jv spiflash device[70].

Read identifier information was verified against the known value, verifying the *PHY* and the *Controller* for correctly interpreting the data, processing the command, and sending back data. Short circuits on the line were inspected visually. When writing data to the *QSPI peripheral* the output data at the *Interface* was checked against each sent value, also verifying the *Interface*. These tests were executed successfully.

5.3. Software

To reach Requirement '100 Mb/s speed' the software needs to be able to send data at this rate to the FPGA. For this the driver throughput needs to be high enough and, as such, the *QSPI* driver speed is verified.

5.3.1. QSPI driver

To verify if the *QSPI driver* can reach the target speed of 100 Mb/s, the throughput needs to be measured. Using `dd if=/dev/zero of=/dev/fpga_tx bs=32M` as data move command, data will be 'generated' and sent to the driver. Once `dd` is finished it will report the systems throughput. A parameter to this test is the SPI clock frequency. Increasing this parameter value should increase the throughput. Since driver is tested for throughput the SPI clock frequency is maximized. The test results and a discussion of the results will follow.

Results

In Table 5.1 shows the test results for the different driver implementations. It also adds a column for the ratio between the reported throughput and the theoretical maximum based on the SPI frequency.

Table 5.1: QSPI driver test results. Ratio as calculated from the reported `dd` speed / theoretical bandwidth. The theoretical bandwidth is 4 bits * SPI speed.

Test name	**SPI speed (MHz)**	**`dd` speed (MB/s)**	**ratio**
SPI-NOR	166.00	8.2	0.10
Custom SPI-MEM controller	166.00	8.4	0.10
QSPI driver DMA	166.00	34.7	0.42
Integrated driver	166.00	35.8	0.43

Discussion

From the table it can be seen that using DMA improved the data throughput significantly. By cutting out the processors involvement the speed improved by a factor 4. Besides, it can be seen that the required speed, the 100 Mb/s for Requirement '100 Mb/s speed' can be reached. For higher speeds, such as for Requirement '500 Mb/s speed' extra effort has to be made.

In the ratio column a factor lower than 0.5 can be found, meaning that the data can be moved more efficiently. This can by explained by the fact that currently the data is received from user space and then sent to the peripheral. By already starting to send data to the peripheral after the first bytes have been received, the copy from user space can be overlapped with the peripheral send. This concept of latency-hiding hasn't been implemented as the target speed of 100 Mb/s had been reached.

5.4. System

Connecting all individual parts together can cause unexpected behaviour. To prevent this from happening the system was tested as a whole. This was first was needed to verify the QSPI communication and then the final implementation. These discussions follow in the subsections.

5.4.1. QSPI communication

As core communication between the FPGA and the microprocessor the *QSPI communication* needs to be tested. This subsection describes how the test was executed, the test results and it gives a discussion of the test results.

Method

To verify the *QSPI communication*, the CP400.85 and the ECP5-5G were connected, with the *QSPI driver* and the *QSPI peripheral* ready. As input to the driver a file was generated with a binary counter from 0 to `0x1FFFFF`, resulting in a 6 MiB file (3 bytes per count). This file was then sent through the driver with `dd if=/tmp/binary_counter.bin of=/dev/fpga_tx bs=32M`.

To verify the communication a stream validator was implemented that calculates a CRC over all passed bytes per transaction (one *chip select* cycle). This CRC is based on the same characteristics (ANSI X3.66) as the Linux tool `crc32`, such that a comparison can be made. The calculated CRC was then sent to an UART peripheral which sends out this data and is captured by the Arduino Due. Together with the pre-calculated value of the CRC tool the integrity of the stream could be checked.

An overview of the test setup can be seen in Figure 5.3

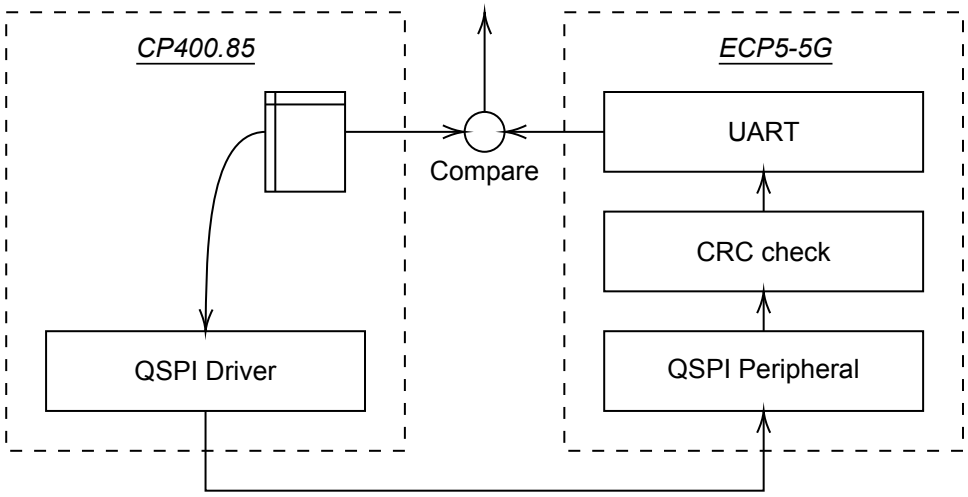


Figure 5.3: QSPI communication test setup overview.

Results

For the FPGA the synthesis tool reports a maximum frequency of 89.670 MHz for the *QSPI* peripheral, with longest delay in the *QSPI controller* of 10.090 ns. The first run of the test resulted in Table 5.2. During this test it was found that the used cables in this test caused problems. We replaced these cables after the first test. A new run of the test with replaced cables can be found in Table 5.3.

Table 5.2: QSPI interface test results. Ratio as calculated from the reported `dd` speed / theoretical bandwidth. The theoretical bandwidth is 4 bits * SPI speed.

SPI speed (MHz)	`dd` speed (MB/s)	corrupted	ratio
83.00	23.0	yes	0.55
55.33	17.9	yes	0.65
41.50	14.9	yes	0.72
33.20	12.6	yes	0.79
27.67	10.9	rarely	0.82
23.71	9.7	no	0.82
20.75	8.7	no	0.84
15.09	6.6	no	0.87
10.38	4.7	no	0.91
5.03	2.4	no	0.95
1.00	0.495	no	0.99

Table 5.3: QSPI interface test results with improved cables. Ratio as calculated from the reported `dd` speed / theoretical bandwidth. The theoretical bandwidth is 4 bits * SPI speed.

SPI speed (MHz)	`dd` speed (MB/s)	corrupted	ratio
83.00	23.0	yes	0.55
55.33	18.0	yes	0.65
41.50	14.9	no	0.72
33.20	12.6	no	0.79
27.67	10.9	no	0.82
23.71	9.7	no	0.82
20.75	8.7	no	0.84

Discussion

From the synthesis results it can be seen that the QSPI peripheral can run at 358.68 Mb/s, which is fast enough for Requirement '100 Mb/s speed', but not for Requirement '500 Mb/s speed'. In the initial run it can be seen that at maximum 10.9 MB/s was reached and that the communication became unstable at that point. To improve the QSPI signal integrity between the FPGA and the microprocessor the wires in-between the boards have been replaced by shorter, higher quality wires. The signal integrity was suspected to be an issue as the used cables were rather long, the traces of the development boards unmatched, and the cables themselves where of rather bad quality.

With the new setup higher speeds up to 119.2 Mb/s were recorded, suggesting that the improved cables worked. After 41.50 MHz the data still got corrupted. Using a logic analyser the lines were analysed, resulting in three electrical problems that were found.

The first problem that was found was (measured) noise on the line. This can be seen in the chip select line (NCS) of Figure 5.4 and the I/O lines. The I/O lines should have shown a harmonica pattern due to the binary counter, but the trace showed none of this.

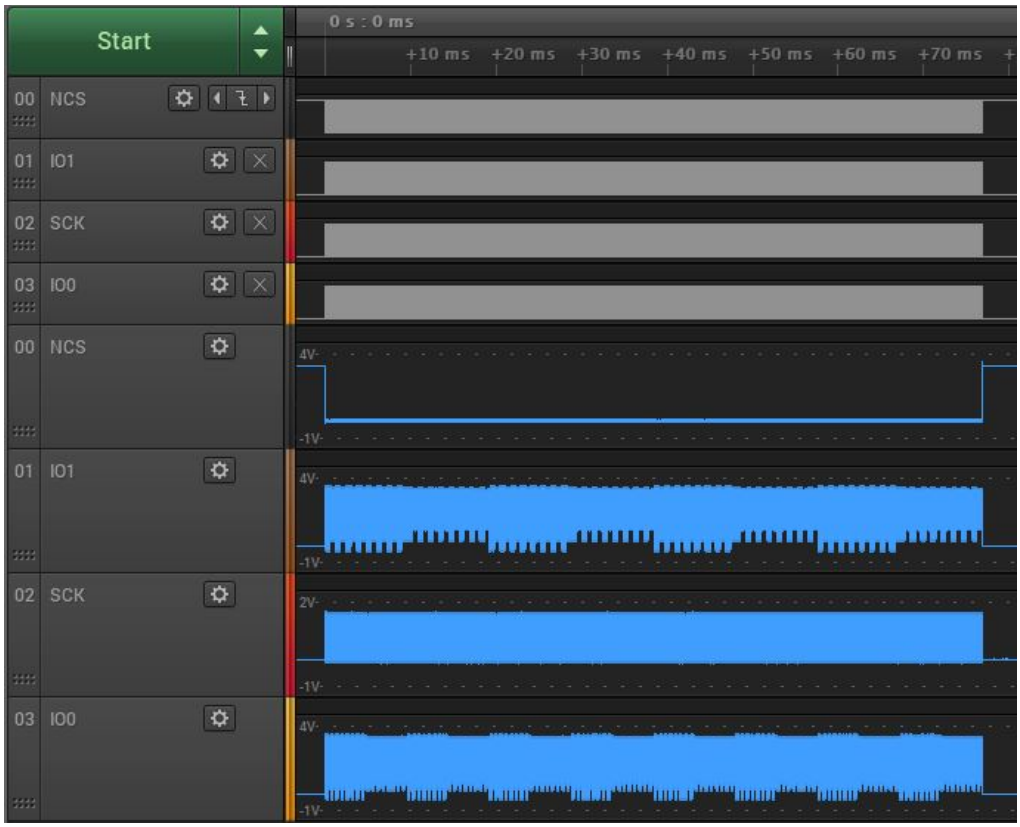


Figure 5.4: QSPI interface test logic analyser trace of a 83 MHz transmission, showing noise on the lines.

The second problem that was found was coupling. In Figure 5.5 this can be seen by looking at the

chip select line (NCS), which shows the same pattern as the clock line (CSK).

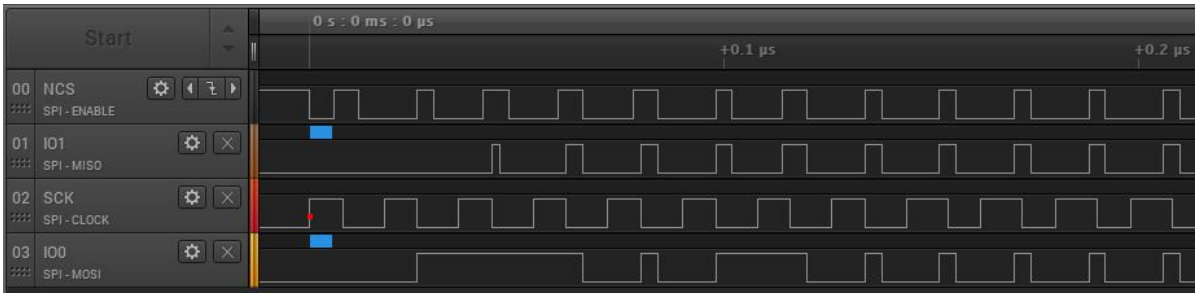


Figure 5.5: QSPI interface test logic analyser trace of a 55 MHz transmission, with coupling occurring.

The third problem that was found was a phase shift that occurred. In Figure 5.6 it can be seen that on IO0 the data is received out of phase with any edge of the clock line.

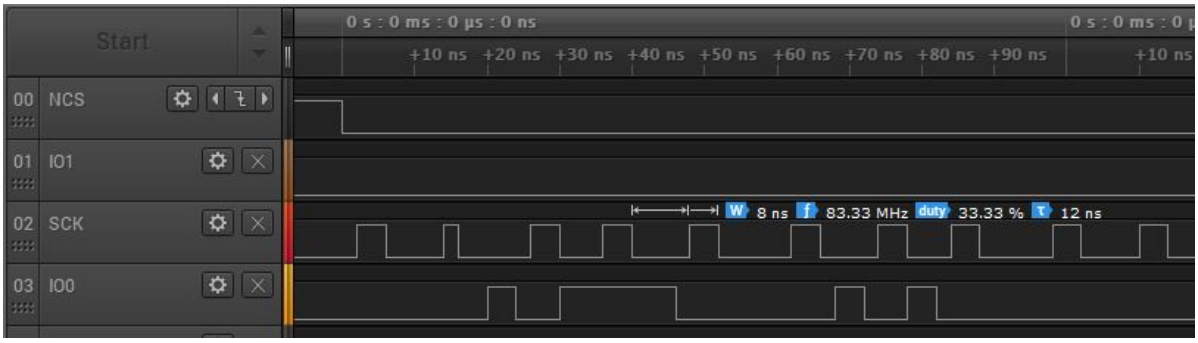


Figure 5.6: QSPI interface test logic analyser trace of a 83 MHz transmission, showing a significant phase shift.

As the found problems were electrical and the target speed of 100 Mb/s was reached, these problem were not further investigated. With an integrated design, on one PCB, with matched lines this problem should be fixed. The test was thus successful.

5.4.2. End-to-end

As final verification of all implemented parts the whole system was tested. This subsection describes how the test was executed, the test result and it gives a discussion of the test result.

Method

With the built models in cocotb an expected output can be generated. Comparing this expected output against the received output from the FPGA verifies the system. To realise this test the CP400.85 was connected to the ECP5-5G with QSPI at a frequency of 41.50 MHz. On the CP400.85 a known file was loaded, which also had been put into the model. Sending this file to the FPGA, the FPGA should output the exact same data as the model. This setup has been visualised in Figure 5.7.

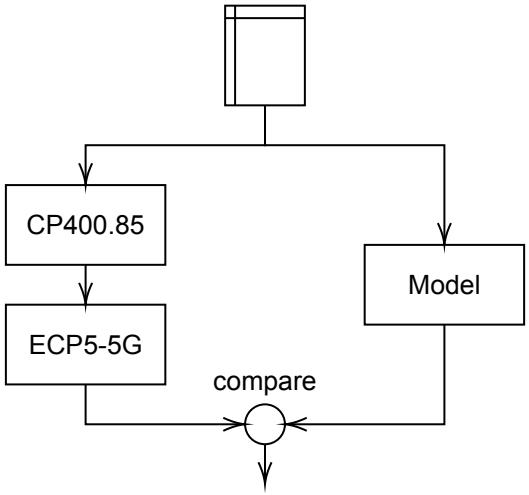


Figure 5.7: End-to-end system test overview.

With the lack of available LVDS test material the *LVDS driver* was replaced by an UART channel. This bottlenecks the output rate of the FPGA to a 115200 baud rate. With the system clock running at 50 MHz the data still processed at a rate of 3.2 Gb/s, meaning that this test partially validates the system.

Result

In the data path of the FPGA a maximum frequency of 60.724 MHz can be used as reported by the synthesis tool, with the longest delay path in the *CRC attachment* of 16.585 ns.

With all data received the output was compared to the expected output. Using a `diff` tool it was found that there was no difference.

Discussion

The test result shows that the system behaves as expected. This means that the built system is validated for all components except the LVDS output required by Requirement 'LVDS'. With the SPI frequency set at 41.50 MHz, a link speed of 119.2 Mb/s between the microprocessor and the FPGA was reached. In the FPGA a throughput of 3.2 Gb/s was calculated with the system clock at 50 MHz. With the maximum reported frequency this would be 3.88 Gb/s. This means that Requirement '100 Mb/s speed' has been reached for the system. For the FPGA Requirement '500 Mb/s speed' and Requirement '1 Gb/s speed' are also met, providing an implementation that can be used for later upgrades. As validated by the model, the system encodes the data according to the TNO3k standard, also fulfilling Requirement 'TNO3k encoding'. This partially fulfils Goal **B**, as the LVDS output isn't validated.

In synthesis and simulation the LVDS peripheral has shown that it is a relative simple peripheral. Furthermore, both the UART and the LVDS peripherals have a lower output speed than the internal 3.2 Gb/s of the FPGA. This means that the decoupling FIFOs provide the same functionality: slowing down the stream. On the input an issue can arise that the input data isn't provided fast enough and thus the stream isn't continuous. However, the input speed is higher than the 100 Mb/s output speed. To cope with stutters the *Data buffer* was implemented. It can thus be assumed that there won't be major issues in changing the output peripheral and it can thus be assumed that Goal **B** is achieved.

5.5. Power

As for Requirement 'Power' the DMU power should remain under the 2 watts. To test whether the currently implemented DMU stays within this bound the power has been calculated. The method, results and discussion will follow.

5.5.1. Method

Measuring the used power of the current implementation requires the measurement of the power used by the CP400.85 and the ECP5-5G during operation. The used CP400.85 development board has a built-in power measurement, which measures the power used by the whole development board. During

operation these numbers need to be noted. A base distribution without further power optimization has been used on the CP400.85. The QSPI communication was set to 41.5 MHz. For the ECP5-5G the tools lacked to measure the actual power used by the chip. Instead the estimated power consumption from the Diamond toolchain was noted. This was done for the end-to-end design, with the LVDS driver at 1 Gb/s and a system clock of 50 MHz. The ambient temperature was set to 25 °C.

5.5.2. Results

The measured current drawn for the CP400.85 was 89 mA in idle and 103-121 mA during operation. This was measured at a 4.95 V system input, maximizing the power to 0.60 watts drawn by the CP400.85.

For the ECP5-5G the total is expected to be 0.18 watts.

5.5.3. Discussion

Looking at the combination of the power numbers it is still within the 2 watts required by Requirement ‘Power‘. However, this is without taking into account the power used by the external memory to the FPGA. According to [82] the LPDDR3 power consumption is around 7 mW for a Dhrystone[83] test and around 38 mW for MP3 audio playback. Closer to the use case is a file transfer in which [84] shows a power consumption of around 224 mW for DDR3 memory.

Adding the worst case values together the estimated power consumption is around 1 watts. This is within the required 2 watts, meaning Requirement ‘Power‘ is met. Note that the storage power consumption is included in the CP400.85 measurement. According to Samsung the power consumption of eMMC can be around 0.5 watts[85].

5.6. Conclusion

All implemented subsystems have been verified using different methods. The VHDL modules have been tested using VHDL testbenches and the TNO3k encoder parts have been tested using cocotb. On the software side the QSPI driver has been tested for its throughput target speed. All these tests were executed successfully fulfilling Requirement ‘TNO3k encoding‘ and achieving Goal **C**.

After all these test were executed, the system integration was tested by physically connecting the CP400.85 and the ECP5-5G. A test was executed to verify the QSPI communication and a final system test was executed to validate the whole system. Both these test were executed successfully with Requirement ‘100 Mb/s speed‘, Requirement ‘500 Mb/s speed‘, and Requirement ‘1 Gb/s speed‘ met for the 3.2 Gb/s core of the FPGA. For the software only Requirement ‘100 Mb/s speed‘ has been met due to the recorded maximum of 119.2 Mb/s for the QSPI link.

With the system test verification, for Requirement ‘LVDS‘ of the DMU was missed due to the lack of proper LVDS testing material, meaning that Goal **B** has been partially met. However, it is assumed that testing with LVDS doesn’t change the results and thus it can be assumed that Goal **B** is achieved.

The expected power of the currently implemented system has been calculated to be around 1 watts. This estimation is based on measurements of the CP400.85, estimates for the ECP5-5G and estimates for the LPDDR3. The 1 watts of power is within the specification of Requirement ‘Power‘.

6 Conclusion and Future Work

This chapter shows the conclusion of the work done in this thesis. Besides, it highlights the main contributions done to the CubeCAT project and concludes with a summary of the possible future work.

It starts with a summary of all conclusions in Section 6.1. This is followed by a final conclusion for this thesis in Section 6.2. The main contributions of this thesis can be read in Section 6.3 and lastly, the future work is listed in Section 6.4.

6.1. Summary

With a 1 unit (1U) Laser Communication Terminal, CubeCAT focusses on the CubeSat industry. This industry faces the challenge to protect the hardware against the effects of thermal differences, Single Event Latchup, Single Event Upset and Total Ionizing Dose. After two iterations of coding schemes the TNO3k standard will be used for the CubeCAT to ensure robust communication.

Multiple cases on bigger satellites that have already proved the optical laser communication technology are already on the market. In the SmallSat subset of the market Tesat-Spacecom has the *CubeLCT* that is capable of 100 Mb/s in a 0.3 U size and Sinclair is working on an 1U 1 Gb/s terminal. Other terminals are mostly university projects.

The DMU is responsible for hosting the high-speed digital data path, handling commands, and processing data from the satellite. Inside the CubeCAT system it also interfaces with the *Quad Cell* and the laser driver. All this is needed to enable the main tasks of the DMU: data aggregation, data encoding, and data output.

Enabling these tasks can be done by a multitude of architectures, components, and connection. The finalised design uses the Hyperion CP400.85 as microprocessor, the Lattice ECP5-5G as FPGA and eMMC for storage, effectively achieving Goal **A**. In the first iteration the FPGA is connected via QSPI to enable 100 Mb/s and 500 Mb/s output speeds. When the first design is produced the EBI can be used to get to an 1 Gb/s output speed. In this design the FPGA is responsible for interfacing with the processor, encoding the data stream, and providing an LVDS output to the *Laser Driver*. The software is responsible for packetization, processing satellite commands, handling satellite data, communicating with the *Quad Cell* and interfacing with the FPGA.

The architecture of the FPGA was designed with the set responsibilities in mind. With the QSPI peripheral, the *Encoder* and the *LVDS driver* FPGA responsibility ‘TNO3k‘, FPGA responsibility ‘QSPI‘ and FPGA responsibility ‘LVDS‘ are met. All high importance responsibilities have been fulfilled and with a data path of 64 bits wide the FPGA was designed with the 10 Gb/s target of Requirement ‘10 Gb/s speed‘ in mind. The implementation of the channel interleaver and the implementation of the EBI are still left to be done, missing responsibilities FPGA responsibility ‘3 ms channel interleaver‘, FPGA responsibility ‘100 ms channel interleaver‘, FPGA responsibility ‘300 ms channel interleaver‘ and FPGA responsibility ‘EBI‘.

To support the high-speed data path a QSPI driver is needed. With the implementation of the QSPI driver Software responsibility ‘QSPI‘ is fulfilled. The QSPI driver is implemented as a customized Linux driver that uses DMA for transactions. Together with the FPGA implementation an implementation for the high-speed digital data path has been made.

All implemented subsystems have been verified using different methods. The VHDL modules have been tested using VHDL testbenches and the TNO3k encoder parts have been tested using cocotb. On the software side the QSPI driver has been tested for its throughput target speed. All these tests were executed successfully fulfilling Requirement 'TNO3k encoding' and achieving Goal **C**.

After all these test were executed, the system integration was tested by physically connecting the CP400.85 and the ECP5-5G. A test was executed to verify the QSPI communication and a final system test was executed to validate the whole system. Both these test were executed successfully with Requirement '100 Mb/s speed', Requirement '500 Mb/s speed', and Requirement '1 Gb/s speed' met for the 3.2 Gb/s core of the FPGA. For the software only Requirement '100 Mb/s speed' has been met due to the recorded maximum of 119.2 Mb/s for the QSPI link.

With the system test verification, for Requirement 'LVDS' of the DMU was missed due to the lack of proper LVDS testing material, meaning that Goal **B** has been partially met. However, it is assumed that testing with LVDS doesn't change the results and thus it can be assumed that Goal **B** is achieved.

The expected power of the currently implemented system has been calculated to be around 1 watts. This estimation is based on measurements of the CP400.85, estimates for the ECP5-5G and estimates for the LPDDR3. The 1 watts of power is within the specification of Requirement 'Power'.

6.2. Conclusion

With the design of the DMU the goal of providing a design for the high-speed digital data path has been achieved. Furthermore, the goal for an implementation of the high-speed digital data path has been partially achieved by the validation of the data path, excluding the LVDS driver of the DMU. Included in the data path is the TNO3k encoder, which ensures robust communication by use of a Forward Error Correction scheme. This means all goals have been achieved.

The research question was formulated as: " *How can the high-speed digital data path of the CubeCAT be implemented in such way that it provides a robust, upgradable, 100 Mb/s link, with constrained power and size?* "

As final conclusion to this thesis we can answer the question as follows. With the combination of an ECP5-5G FPGA and the CP400.85 microprocessor a basis for the high-speed digital data path of the CubeCAT can be made. This solution stays within the set power and size constraints. Together with a TNO3k encoder in the data path a robust, upgradable, 100 Mb/s link can be ensured. This is how the high-speed digital data path of the CubeCAT can be implemented, such that it provides a robust, upgradable, 100 Mb/s link, with constrained power and size.

6.3. Main contributions

In this thesis an effort was made to implement the high-speed digital data path of the CubeCAT module. The contributions that are made are listed here, along with a short description of the contribution.

- **High-speed data path:** The main contribution of this thesis is the high-speed digital data path for the CubeCAT module. A version has been implemented, and validated, that supports an 100 Mb/s throughput. Besides, a design has been made that supports speeds up to 1 Gb/s.

- **TNO3k encoder:** To ensure interoperability with a TNO ground station the TNO3k FEC scheme was implemented, resulting in the *TNO3k encoder*. This encoder has been implemented and verified. However, it misses the *Channel Interleaver* as external memory wasn't available and the development time was expected to be too long.

- **QSPI peripheral:** To interface with the CP400.85 a QSPI peripheral has been made in VHDL. This peripheral has been used for the high-speed data path and, at the time of writing, is being reused for another project.

- **Custom QSPI driver:** For the CP400.85 a custom QSPI driver has been made. This driver has been used for the high-speed data path and, at the time of writing, is being reused for another project.

6.4. Future work

For the high-speed data path and the DMU several items remain to be (further) implemented. This can be split up in work on the FPGA and work on the software. In the following subsections these items will be further discussed.

6.4.1. Work on the FPGA

The work left to be done on the FPGA is listed in this subsection. The items are discussed in the following, separate subsubsections.

QSPI interface

The baseline speed of 100 Mb/s has been reached, but 500 Mb/s is possible with the QSPI peripheral of the CP400.85. For this however, the integrated board needs to be available as the current setup isn't able to reach the required 125 Mb/s SPI clock frequency. The FPGA QSPI peripheral needs to be revised to support this speed, as the maximum with the current implementation is 358.68 Mb/s.

External Bus Interface

The *CP400.85* has an External Bus Interface available for use when it is fully integrated. When the integrated board is available, an EBI interface should be implemented on the FPGA to support higher speeds.

Channel interleaver

During discussions with TNO it was decided to leave the *channel interleaver* to be implemented. On further work on the DMU, especially the *TNO3k encoder*, this interleaver should be implemented using the available external LPDDR3 memory.

Increase throughput

On a later system with an FPGA with 10 Gb/s transceivers, the currently implemented system needs to be updated to support a minimal of 156.25 MHz for the system clock, to reach the 10 Gb/s target. A point of improvement is the implementation of the CRC module in the TNO3k encoder. Due to the non-ideal alignment of the bytes per packet, this module has the longest delay path.

6.4.2. Work on the Software

The work left to be done on the software is listed in this subsection. The items are discussed in following, separate subsubsections.

QSPI Driver

On removal of the loadable module (`rmmod`) the kernel errors on problems with the peripheral clock being already unprepared. On reload of the module the module crashes, indicating that the clean-up procedure needs improvement. It is unknown what data isn't properly cleaned.

Another factor that can be improved in the QSPI driver is the way data is handled. Currently everything is sequential, which could be improved to a version that hides the transfer latency. It is expected that this would significantly improve the speed.

External Bus Interface

When the integrated version of the designed system is available the EBI interface needs a driver. This is needed to support the 1 Gb/s goals of the CubeCAT.

Satellite interface

Interfacing with the Data Management Unit is not yet implemented. To be able to store data in the DMU a software interface should be implemented.

Sensor data collection

The Laser Communication Terminal contains several monitoring sensors in, for example, the *Quad Cell*. This data needs to be collected from the available interfaces and stored in the DMU.

Packetization

As mentioned in Section 3.4.2 there is no defined method of packetization. Defining a method of packetization is needed for a smooth transmission of multiple files, as well as having an indication which packets need to be retransmitted.

Bibliography

[1] The difference between Verification and Validation. [Online]. Available: https://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/

[2] UHF Transceiver II CubeSat Communication. [Online]. Available: https://www.endurosat.com/cubesat-store/cubesat-communication-modules/uhf-transceiver-ii/

[3] European data relay satellite system (EDRS) overview. ESA. [Online]. Available: https://artes.esa.int/edrs/overview

[4] Telescope Quick Facts. [Online]. Available: http://hubblesite.org/home/quick-facts/telescope-quick-facts

[5] Hiber - Low cost, Low power IoT connectivity. [Online]. Available: https://hiber.global/index.html

[6] CubeSat S-band Transceiver. [Online]. Available: https://www.isispace.nl/product/s-band-transceiver/

[7] J. H. Trainor, "Instrument and spacecraft faults associated with nuclear radiation in space," vol. 14, no. 10, pp. 685–693. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0273117794905274

[8] M. Berg, "Xilinx Virtex-5QV (V5QV) Independent SEU Data." [Online]. Available: https://nepp.nasa.gov/workshops/etw2014/talks/Thur/1030%20-%202014-561-Berg-Final-Web-Pres-ETW-FPGA-TN16271_v5.pdf

[9] BAE Systems, "RAD750® family of radiation-hardened products." [Online]. Available: https://www.baesystems.com/en-us/download-en-us/20190103202640/1434555668211.pdf

[10] mars.nasa.gov. Rover Brains. [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/brains/

[11] S.-J. Kang and H.-U. Oh. On-Orbit Thermal Design and Validation of 1 U Standardized CubeSat of STEP Cube Lab. [Online]. Available: https://www.hindawi.com/journals/ijae/2016/4213189/

[12] J. Lau, "Solder joint reliability of flip chip and plastic ball grid array assemblies under thermal, mechanical, and vibrational conditions," vol. 19, no. 4, pp. 728–735.

[13] L. W. Townsend, J. Wilson, and J. E. Nealy, "Space radiation shielding strategies and requirements for deep space missions."

[14] T. Page and J. Benedetto, "Extreme latchup susceptibility in modern commercial-off-the-shelf (COTS) monolithic 1M and 4M CMOS static random-access memory (SRAM) devices," in *IEEE Radiation Effects Data Workshop, 2005.*, pp. 1–7.

[15] C. Underwood, "The single-event-effect behaviour of commercial-off-the-shelf memory devices. A decade in low-Earth orbit," in *RADECS 97. Fourth European Conference on Radiation and Its Effects on Components and Systems (Cat. No.97TH8294)*, pp. 251–258.

[16] M. Sajid, N. G. Chechenin, F. Sill Torres, M. Nabeel Hanif, U. A. Gulzari, S. Arslan, and E. U. Khan, "Analysis of Total Ionizing Dose effects for highly scaled CMOS devices in Low Earth Orbit," vol. 428, pp. 30–37. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168583X18303215

[17] Superfast internet using laser-satellite communications | TNO. [Online]. Available: https://www.tno.nl/en/tno-insights/articles/superfast-internet-using-laser-satellite-communications/

[18] Sander van Leeuwen, “Comparing two CCSDS specifications for optical space communication,” in *Literature Review*. Delft University of Technology/ Faculty of Electrical Engineering, Mathematics and Computer Science, p. 8.

[19] C.W. Korevaar and H.J.W. Medenblik, “ICD Physical & Link Layer - OGS MODEM & CUBECAT.”

[20] D. Bakker, “A lasercom terminal for CubeSats,” web. [Online]. Available: http://resolver.tudelft.nl/uuid:663b14ad-6e42-4f71-86aa-880a1c534aee

[21] ESA and Inmarsat sign innovative Alphasat satellite contract. ESA. [Online]. Available: https://www.esa.int/About_Us/Business_with_ESA/ESA_and_Inmarsat_sign_innovative_Alphasat_satellite_contract

[22] Alphasat’s laser terminal on target. [Online]. Available: http://www.esa.int/Applications/Telecommunications_Integrated_Applications/Alphasat/Alphasat_s_laser_terminal_on_target

[23] D. C. Troendle, C. Rochow, P. Martin-Pimentel, H. Zech, F. F. Heine, H. Kaempfner, M. Motzigemba, U. Sterr, R. Meyer, M. Lutzer, and S. D. Philipp-May, “Optical LEO-GEO data relays: From demonstrator to commercial application,” in *32nd AIAA International Communications Satellite Systems Conference*. American Institute of Aeronautics and Astronautics. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2014-4439

[24] Laser communication terminals - Tesat Spacecom. [Online]. Available: https://web.archive.org/web/20160911143720/http://www.tesat.de/en/divisions/laser-products/laser-communication-terminals

[25] DLR, “DLR communicates with TerraSAR-X Earth Observation satellite via laser beam.” [Online]. Available: https://www.dlr.de/en/desktopdefault.aspx/tabid-78/7420_read-14120/

[26] B. Smutny, H. Kaempfner, G. Muehlnikel, U. Sterr, B. Wandernoth, F. Heine, U. Hildebrand, D. Dallmann, M. Reinhardt, A. Freier, R. Lange, K. Boehmer, T. Feldhaus, J. Mueller, A. Weichert, P. Greulich, S. Seel, R. Meyer, and R. Czichy, “5.6 Gbps optical intersatellite communication link,” H. Hemmati, Ed., p. 719906. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.812209

[27] H. Hemmati and J. M. Kovalik, “10 Gb/s Lasercom Terminal for Satellites,” p. 7. [Online]. Available: http://icsos2012.nict.go.jp/pdf/1569591919.pdf

[28] BIROS (FireBird 2). [Online]. Available: https://space.skyrocket.de/doc_sdat/biros.htm

[29] C. Schmidt, M. Brechtelsbauer, F. Rein, and C. Fuchs, “OSIRIS Payload for DLR’s BiROS Satellite,” p. 7.

[30] C. Fuchs and C. Schmidt, “Update on DLR’s OSIRIS program,” in *International Conference on Space Optics — ICSO 2018*, vol. 11180. International Society for Optics and Photonics, p. 111800I. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11180/111800I/Update-on-DLRs-OSIRIS-program/10.1117/12.2535937.short

[31] F. Millour, S. Ottogalli, M. Maamri, A. Stibbe, F. Ferrero, L. Rolland, S. Rebeyrolle, A. Marcotto, K. Agabi, M. Beaulieu, M. Benabdesselam, J.-B. Caillau, F. Cauneau, L. Deneire, F. Mady, D. Mary, A. Mémin, G. Metris, J.-B. Pomet, O. Preis, R. Staraj, E. A. Lachgar, D. Baltazar, B. Gao, M. Deroo, B. Gieudes, and M. Jiang, “The Nice Cube (Nice3) nanosatellite project,” p. 13.

[32] Opticalcomms - www. [Online]. Available: http://www.sinclairinterplanetary.com/opticalcomms

[33] L. Yenchesky, O. Cierny, P. Grenfell, W. Kammerer, P. Periera, T. Sevigny, and K. Cahoy, “Optomechanical Design and Analysis for Nanosatellite Laser Communications.” [Online]. Available: https://digitalcommons.usu.edu/smallsat/2019/all2019/161

[34] Laser Products. [Online]. Available: https://tesat.de/en/products/laser-products

[35] Satsearch - The global marketplace for the space industry. [Online]. Available: https://satsearch.co/products/tesat-spacecom-cube-lct-100m

[36] Bus Speed (Default Speed/High Speed/UHS/SD Express) - SD Association. [Online]. Available: https://www.sdcard.org/developers/overview/bus_speed/index.html

[37] Overview :: Sd card controller :: OpenCores. [Online]. Available: https://opencores.org/projects/sdcard_mass_storage_controller

[38] B. Saunders, "USB Type-C® System Overview," p. 33.

[39] G. Yedidia, "USB4TM Configuration Layer, USB3 Tunneling, DP Tunneling and PCIe Tunneling," p. 97.

[40] ATSAMV71Q19 - 32-bit SAM Microcontrollers. [Online]. Available: https://www.microchip.com/wwwproducts/en/ATSAMV71Q19

[41] STM32H745XI. [Online]. Available: https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32h7-series/stm32h745-755/stm32h745xi.html

[42] I.MX 8M Applications Processor | Arm® Cortex®-A53, Cortex-M4 | 4K display resolution | NXP. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i.mx-applications-processors/i.mx-8-processors/i.mx-8m-family-armcortex-a53-cortex-m4-audio-voice-video:i.MX8M

[43] NXP, "I.MX 8X Family of Applications Processors." [Online]. Available: https://www.nxp.com/docs/en/fact-sheet/IMX8XFAMFS.pdf

[44] I.MX8 SOM | The New IoT Multimedia Powerhouse. [Online]. Available: https://www.solid-run.com/nxp-family/imx8-som/

[45] I.MX 8X Applications Processors| Arm® Cortex®-A35, Cortex-M4; | NXP. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i.mx-applications-processors/i.mx-8-processors/i.mx-8x-family-arm-cortex-a35-3d-graphics-4k-video-dsp-error-correcting-code-on-ddr:i.MX8X

[46] "List of ARM microarchitectures." [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_ARM_microarchitectures&oldid=942478644

[47] A. Frumusanu. ARM Announces New Cortex-A35 CPU - Ultra-High Efficiency For Wearables & More. [Online]. Available: https://www.anandtech.com/show/9769/arm-announces-cortex-a35

[48] Embedded Module TQMa8Xx. [Online]. Available: https://www.tq-group.com/en/products/tq-embedded/arm-architecture/tqma8xx/

[49] Cp400.85. [Online]. Available: https://hyperiontechnologies.nl/products/cp400-85-processing-platform/

[50] Zynq UltraScale+ MPSoC. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html#productAdvantages

[51] RISC-V Foundation | Instruction Set Architecture (ISA). [Online]. Available: https://riscv.org/

[52] PolarFire FPGAs | Microsemi. [Online]. Available: https://www.microsemi.com/product-directory/fpgas/3854-polarfire-fpgas

[53] Single Event Effects (SEE) | Microsemi. [Online]. Available: https://www.microsemi.com/product-directory/reliability/4883-see#test-results

[54] ECP5 / ECP5-5G - Lattice Semiconductor. [Online]. Available: https://www.latticesemi.com/en/Products/FPGAandCPLD/ECP5

[55] J. K. DeBoy, "eMMC Architecture and Operation." [Online]. Available: http://cmosedu.com/videos/s17/ecg721/eMMC_Architecture_and_Operation.pdf

[56] eMMC 5.1 Device. [Online]. Available: https://www.arasan.com/products/emmc51/emmc51-device/

[57] Gigabyte. AORUS Gen4 AIC SSD 8TB Elevates Storage Performance. [Online]. Available: https://www.aorus.com/news-detail.php?i=2271

[58] 1 Gbit (128 Mbyte) S79FL01GS Dual-Quad SPI NOR Flash Memory Datasheet. [Online]. Available: https://www.cypress.com/documentation/datasheets/1-gbit-128-mbyte-s79fl01gs-dual-quad-spi-nor-flash-memory-datasheet

[59] Scott Jacobson and Harish Verma, "UFS Tutorial." [Online]. Available: https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130814_T1_Jacobson.pdf

[60] JEDEC Publishes Universal Flash Storage (UFS & UFSHCI) Version 3.0 and UFS Card Extension Version 1.1 | JEDEC. [Online]. Available: https://www.jedec.org/news/pressreleases/jedec-publishes-universal-flash-storage-ufs-ufshci-version-30-and-ufs-card

[61] HummingBoard Pulse. [Online]. Available: https://developer.solid-run.com/knowledge-base/hummingboard-pulse-getting-started/

[62] ISO/IEC and ISO/IEC, "Information technology - open systems interconnection - basic reference model: The basic model," ISO/IEC. [Online]. Available: https://www.iso.org/standard/20269.html

[63] Floris Rouwen, "Hyperion Protocol."

[64] E. F. Inc. The Platform for Open Innovation and Collaboration | The Eclipse Foundation. [Online]. Available: https://www.eclipse.org/

[65] IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains. [Online]. Available: https://www.jetbrains.com/idea/

[66] Visual Studio Code - Code Editing. Redefined. [Online]. Available: https://code.visualstudio.com/

[67] Vinrobot, "Vinrobot/Vscode-Vhdl-Formatter." [Online]. Available: https://github.com/Vinrobot/vscode-vhdl-formatter

[68] H. Bohlin, "Bochlin/Rust_hdl_vscode." [Online]. Available: https://github.com/Bochlin/rust_hdl_vscode

[69] UART - Lattice Semiconductor. [Online]. Available: https://www.latticesemi.com/en/Products/DesignSoftwareAndIP/IntellectualProperty/ReferenceDesigns/ReferenceDesign03/UART

[70] W25Q32JV Datasheet. [Online]. Available: https://www.winbond.com/resource-files/w25q32jv%20spi%20revc%2008302016.pdf

[71] CRC. [Online]. Available: https://www.easics.com/webtools/crctool

[72] CCSDS, *CCSDS 131.0-B-3 - TM Synchronization and Channel Coding*, ser. Blue Book. CCSDS Secretariat, National Aeronautics and Space Administration. [Online]. Available: https://public.ccsds.org/Pubs/131x0b3e1.pdf

[73] D. S. K. Patra, "Professor & Head Department of Electronics & Communication Engineering National Institute of Technology, Rourkela," p. 55.

[74] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and Their Applications*. John Wiley & Sons.

[75] Spi-mem: Bringing some consistency to the SPI memory ecosystem - Bootlin's blog. [Online]. Available: https://bootlin.com/blog/spi-mem-bringing-some-consistency-to-the-spi-memory-ecosystem/

[76] Open Bench Logic Sniffer - DP. [Online]. Available: http://dangerousprototypes.com/docs/Open_Bench_Logic_Sniffer

[77] Logic Analyzers from Saleae - #1 with Professional Engineers. [Online]. Available: https://www.saleae.com/

[78] Introduction — cocotb 1.3.1 documentation. [Online]. Available: https://docs.cocotb.org/en/v1.3.1/introduction.html

[79] GHDL Main/Home Page. [Online]. Available: http://ghdl.free.fr/

[80] GTKWave. [Online]. Available: http://gtkwave.sourceforge.net/

[81] Arduino Due | Arduino Official Store. [Online]. Available: https://store.arduino.cc/arduino-due

[82] "I.MX 6SLL Power Consumption Measurement," p. 14.

[83] ARM Information Center. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0273a/ar01s01.html

[84] "I.MX 6 DualLite Power Consumption Measurement," p. 42.

[85] eMMC - The Best Mobile Storage | Samsung Semiconductor. [Online]. Available: https://www.samsung.com/semiconductor/estorage/emmc/

[86] CCSDS, *CCSDS 142.0-B-1 - Optical Communications Coding and Synchronization*, ser. Blue Book. CCSDS Secretariat, National Aeronautics and Space Administration. [Online]. Available: https://public.ccsds.org/Pubs/142x0b1.pdf

[87] CRC calculation. [Online]. Available: http://www.zorc.breitbandkatze.de/crc.html

[88] Sunshine's Homepage - Online CRC Calculator Javascript. [Online]. Available: http://www.sunshine2k.de/coding/javascript/crc/crc_js.html

[89] J. Corber, Alessandro Rubini, and Greg Kroah-Hartman, *Linux Device Drivers, Third Edition*, 3rd ed. O'Reilly Media, Inc. [Online]. Available: https://lwn.net/Kernel/LDD3/

[90] Java.Util.Stream (Java Platform SE 8). [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html

[91] Intel, "Avalon® Interface Specifications," p. 70. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

A

CCSDS 142x0b1

The encoder is responsible for encoding the data with a FEC (Forward Error Correction) scheme to reduce the BER (Bit Error Rate) of the optical link. It reduces the amount of total data corruption introduced by attenuation in the link. This is done by adding extra data to the data stream that makes it possible to repair corrupted data.

The first design of the encoder was based on the CCSDS 142.0-B-1 standard[86]. This appendix will discuss the design of the 142 standard in Section A.1. In Section A.2 the implementation of the 142 standard can be found.

A.1. Design

As standard for the CCSDS encoder the CCSDS 142.0-B-1 standard is followed for the encoding of data. The standard describes multiple modifications to the data to improve the error recovery in the ground station. An overview of blocks that modify the data can be seen in Figure A.1.

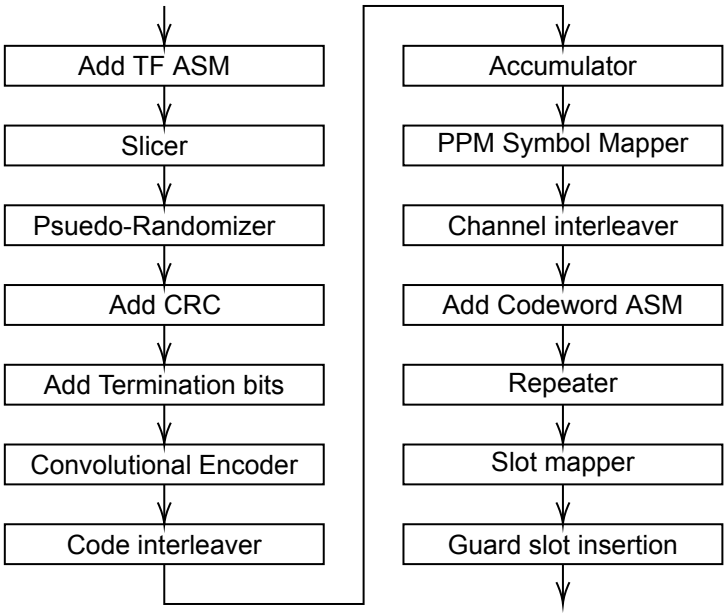


Figure A.1: CCSDS 142.0-B-1 overview

In the first block a marker is attached to an incoming frame for synchronization on the receiver side. Then, the outputted frame with marker is sliced into pieces of a predetermined size. This size is dependent on the code rate of the *Convolutional Encoder*. On transmission closure the slicer will fill the stream with zeroes to fill a slice.

To increase the randomness of the data a *Psuedo-Randomizer* is used. The effect of this process is ensured activity on the output, preventing long runs without changes of data.

Then, another aid for the receiver is added: a CRC. This helps to detect corruption of a data frame after receival. To reset the *Convolutional Encoder* to a known state a functional block is added. This block adds two termination bits at the end of a slice to set the *Convolutional Encoder* to a known state. The *Convolutional Encoder* provides the needed Forward Error Correction for the data stream. With puncturing the output of the *Convolutional Encoder*, the code rate can be changed.

A *Code interleaver* is then used to spread sequential errors during transmission to different locations in the data block, improving error recovery. The *Code interleaver* is followed by an *Accumulator* which changes the detection method in the data to edge detection.

Next is the *PPM Symbol Mapper* which reinterprets the data to an N-b unsigned value to be later used in the *Slot mapper*. The PPM symbols are then interleaved, mixing multiple data blocks such that in the event of a fade errors occur in multiple data blocks and not losing one block.

Per predefined amount of PPM-symbols another marker is added in the *Add Codeword ASM*. The data is then per PPM-symbol repeated, where the amount depends on the setting.

The *Slot mapper* has multiple send slot, depending on the value range of the PPM-symbol. A send slot is a numbered period in time where a bit may exclusively be '1' during that period range. The mapper interprets a PPM-symbol as the unsigned value *v* and toggles slot number *v*. Per slot range, the *Guard slot insertion* adds a 1/4th period of idle time and then transfers its binary data to the output.

A.2. Implementation

A VHDL implementation of the 142 has been made. The blocks have been tested, but where never properly verified, or tested with backpressure. Besides, the first (and only) implementation of the 142 standard was built to closely resemble the standard, such that it can be used to verify other implementations. This had the effect that the built modules had little parallelism.

In the following subsections discussions can be found on the implemented blocks.

A.2.1. Add TF ASM

The sync marker is a predefined sequence, `0x1ACFFC1D`, that is appended to the data stream after N_{TFsize} bytes, where N_{TFsize} is a configurable parameter.

It is implemented by a two-state machine. The first state, s_DATA, passes the data to the output and counts how many bytes have passed. After N_{TFsize} bytes have passed the state machine switches to the second state s_ATTACH. This state outputs the predefined sequence to the output.

A.2.2. Byte-to-bit stream

The natural output of the *Add TF ASM* is bytes and the natural input of the *Slicer* is bits. To follow to the standard as closely as possible, the bytes-stream is converted to a bit-stream.

A.2.3. Slicer

The slicer block is responsible for slicing incoming data blocks to a predefined, configurable size. In the process of slicing it also adds bit positions for the later blocks *Add CRC* and *Add Termination bits*.

A three-state machine is used for the implementation of the *Slicer*, as well as a counter how many bits have been outputted. The first state s_DATA passes the data inputted to the output.

After the predefined amount of bits, the second state s_CRC is activated. This state blocks the input and inserts dummy bits into the stream, which are given a value in the *Add CRC* block.

When 32 bits have passed the last state is activated: the $s\_TERMINATION$ state. This state adds another two dummy bits, for the *Add Termination bits* block. After two bits the state is switched back to the s_DATA state.

The *Slicer* has additional outputs to signal what can of data is being outputted. This has been done to avoid a counter in the *Add CRC* and *Add Termination bits* blocks. These signals are synchronized with the data outputted by the intermediate blocks until the intended block has been reached.

A.2.4. Psuedo-Randomizer

A *Psuedo-Randomizer* block has been implemented by generating a psuedo-random sequence generator. Per clock tick the sequence is updated. The data that is clocked to the output is the inputted

data xor'ed with the last element of the random sequence.

A.2.5. Add CRC

After synchronization to the correct data-time domain, the state output of the *Slicer* is used to determine what should be done. If it is in the s_DATA state, a CRC is calculated over the data inputted. If the *Slicer* is in the s_CRC state, the calculated CRC is outputted to the stream. Otherwise the stored CRC is zeroed.

The operation has been verified using a known hex-sequence. Using this known hex-sequence an expected output was calculated, which was then compared against the modules output. The expected output was calculated using [87] and [88] with the following paramters: CRC order 32, CRC polynomial `0x20044009`, Initial value direct `0xFFFFFFFF`, Final XOR value `0x00000000`, no reverse data bytes, no reverse CRC result before Final XOR and data sequence `0xDEADBEEFABADBABE0000000FF1CE`..

A.2.6. Add Termination bits

During the s_DATA and the s_CRC states of the *Slicer* the *Add Termination bits* block passes the data inputted to the output. During the $s\_TERMINATION$ state, the outputted data is set to a termination value.

A.2.7. Convolutional Encoder

The *Convolutional Encoder* consists of two parts, namely: the encoder and a mechanism to puncture the data, where the puncturing is configurable by a system parameter. Implementing the encoder is done by storing data in a 'delay'-buffer. A xor'ed combination of the input and the delay elements is then used to create three output signals. These signals are then punctured by using a counter to decide which bit should be outputted.

To support all values of the system parameter with a continuous flow, an implementation has been made by using a double encoder. This implementation first buffers a bit and then uses two bits in the encoder to output six signals at a time. These signals are then used in the puncturing process. A counter keeps track of which signal to output. This counter is updated according to the system parameter. The two-input encoder can be seen in Figure A.2.

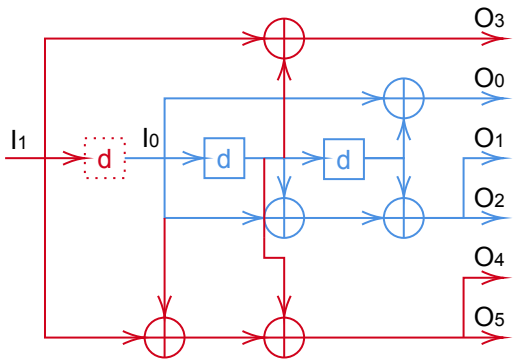


Figure A.2: Convolutional encoder. In blue the one-input version. The red extends the one-input version to a two output version, where the dotted delay element is implicitly the second input.

A.2.8. Code interleaver

The *Code interleaver* rearranges the input data. For this to work it has to be buffered. Two RAM banks have been used to buffer two full slices, to be able to support a continuous stream of data. The first one will accept new input data, where the second one will be read out according to the predefined polynomial. Once the first one is finished filling data and the second one has been read out, the banks will switch in function.

A.2.9. Accumulator

An accumulator has been implemented to support all blocks of the standard. After a discussion with TNO, it had been disabled. The functionality is simply put, store the current data input, and use it for the next clock cycle to xor it with the then current input data, to output that result.

A.2.10. PPM Symbol Mapper

The *PPM Symbol Mapper* changes the interpretation of the data. It actually does no mutations on the data and is essentially an empty block. A Pulse-Position Modulation (PPM) symbol interprets $log_2 M$ bits as one PPM-M symbol.

A.2.11. Channel interleaver

The *Channel interleaver* increases the time over which one full slice of data is outputted. This is done by implementing $N - 1$ rows, consisting of buffer elements of size B. The amount of buffer elements per N_{th} row is N_{th}.

The parameters B and N are must be chosen such that $B * N = S = \frac{15120}{log_2 M}$, where M is the parameter for the PPM size. As the PPM is an interpretation of the data, $B * N = 15120$ bits $\rightarrow \frac{15120}{B} = N$ can be used to determine the bit-size of the buffers. These buffers map to RAM blocks, where N doesn't map directly to an FPGA element.

With high N, the data is spread over a larger time, which is preferable for prevention of unrecoverable data corruption due to fading. To keep support for all possible PPM options, a B is needed that has a integer division with all $log_2 M$ and maximizes N. This resulted the following set $(B, N) \in [(15120, 1); (7560, 2); (5040, 3); (2520, 6); (1680, 9); (840, 18)]$. This resulted in $(B, N) = (840, 18)$ to be chosen for the design.

The design generates per row the needed amount of RAM blocks of 840 bits. To select the current row a counter is used that counts how many PPM symbols ($\frac{bits}{log_2 M}$) have passed. It loops at N PPM symbols.

To ease the implementation of the following blocks, the output data is aggregated into bytes before being outputted. However, the byte is only filled for $log_2 M$ bits. This keeps the interface the same, but support all M. It could be seen as if PPM symbols are outputted.

A.2.12. Add Codeword ASM

The *Add Codeword ASM* prepends a PPM sequence to the PPM data stream. It first halts incoming data before it allows the data to pass through. Three different sequences are possible, depending on whether M is 4, 8, or 16+. After the sequence has been outputted, the data inputted is directly outputted, without modifications.

A.2.13. Repeater

The *Repeater* repeats the incoming PPM symbol r times, where r is a configuration parameter. A new PPM symbol is accepted after r repetitions and the amount of repetitions is tracked by use of a counter.

A.2.14. Slot mapper

The *Slot mapper* takes the PPM symbol as input, interprets it as an unsigned number and converts it to one positional bit in a bit-vector. This bit-vector represents a time-slot. As output interface the maximum size of the bit-vector is used, namely $M_{max} = 256$ bits.

A.2.15. Guard slot insertion

The *Guard slot insertion* takes the 256 bits vector and maps it to a vector with 'silent' slots already in it. Where it is mapped depends on the set PPM M parameter. A guard slot is inserted of size $M/4$, making the total time slots grow by 25%, thus the maximum size increases to 320, thus the vector with 'silent' slots is of size 320.

The mapped vector is then streamed per 5 bits. This is done because the smallest PPM, PPM-4, also needs to be supported, which scales up to 5 bits. Other PPM sizes are multiples of 5 bits. Input data is halted until all data is streamed.

B

HummingBoard

Initially a design based on PCIe communication between the microprocessor and the FPGA was proposed. The HummingBoard would have been the solution for the microprocessor, connected to an ECP5-5G FPGA. Unfortunately the HummingBoard was replaced by the CP400.85 due to missing PCIe lanes. When this was found out, the deadline of the demonstration launch was too close, requiring a different solution than the HummingBoard.

This appendix discusses the implemented parts of the PCIe interconnect using the HummingBoard.

B.1. PCIe

To meet Requirement '10 Gb/s speed' a fast interconnect is needed between the FPGA and the microprocessor. This was envisioned to be PCIe, as both the ECP5-5G and the HummingBoard support the standard. When needed, one of both could be swapped out with minimal effort, potentially upgrading the interconnect speed or transceivers.

In the following subsections first the FPGA implementation will be discussed. This is followed by a discussion on the Linux driver. Finally, the reason is given why this solution didn't work out.

B.1.1. FPGA PCIe core

Lattice provides an IP block for the PCIe peripheral. This IP block implements OSI (Open Systems Interconnection)[62] layer 1 and layer 2. Layer 3 is exposed to the user, which should implement all control on this level. Lattice has provided example code for the PCIe IP block, but unfortunately this is written in Verilog. As the Hyperion company HDL language is VHDL an effort has been made to convert the Verilog code to VHDL, to be reused in the actual design.

In its basic form it should be detected by the PCIe root controller. Before implementation of the actual packet handling this was first to be confirmed.

B.1.2. Linux PCIe driver

For detection of the PCIe device a Linux driver was written. Before this was written a book on Linux Device Drivers[89] was read to learn about Linux drivers. Then, based on the kernels documentation and multiple internet posts, a driver was that should be able to detect and claim a PCIe device, in this case the FPGA.

B.1.3. Traces

When the first try was done on trying to get the FPGA detected by the HummingBoard it didn't work. The FPGA wasn't recognised and the FPGA showed that no reference clock was provided. During the investigation of this problem it was found that due to the placed Wi-Fi modem on the SoM the PCIe traces weren't connected to the mPCIe slot. After a discussion with Hyperion it was found that this wasn't pointed out when the order for the HummingBoard was placed at SolidRun. Unfortunately, the time was getting too short to reinvestigate this matter and thus a different solution was needed leaving this implementation unfinished.

C

Stream mechanism code

Data flowing through a system can get expanded. If this happens, the input of this stream may need to pause the input until the expanded data is streamed out, before continuing. This concept is called .

Other than that, a mechanism needs to be provided to know when data is valid, otherwise invalid data can be marked as valid.

To ensure this correct operation a handshake has been defined on the interfaces of the HDL blocks. First a self-made handshake will be discussed, followed by the Intel Avalon handshake.

C.1. Initial stream mechanism

As can be seen from Figure 2.3, Figure 2.4 and Figure 2.5 the data flow of the FEC schemes is linear; there is no data branching. In addition to that in most blocks the data is expanded, but with sequential operations.

To ensure a constant data flow, different clocks per expanding block can be used to mitigated the synchronization issue. However, two problems can occur with this concept. Firstly, in a more general concept the division of the clock does not always translate to a synthesizable clock. The second problem is the amount of clock synchronization needed between blocks can cause an overhead that could possibly lead to bugs.

Another solution for a constant data flow is concept of streaming. It is a concept used in different programming languages, for example Java[90]. The concept of streams boils down to waiting for the next element in line for it to be ready to accept new data. Besides, the next element is only notified of data if there is new/ valid data.

Extracted from the concept of streaming are the following signals:

- **data_in** The data from the previous element to be processed by the current element.
- **enable** Notification of the previous element that there is new/ valid data.
- **hold** An input for the current element to signal if the next element is ready for new data. If not it should halt all current operations until the next element is ready.
- **ready** A signal to notify the previous element that it is ready to accept new data.
- **valid** A signal to notify the next element that the current outputted data is valid and should be processed.
- **data_out** The data from the current element to be processed by the next element.

The receiving side of this interface defines the following signals: *ready*, *data_in* and *enable*. The *ready* signal is used to signal the back-pressure and *enable* is used to clock in *data_in*. Output interfaces define the following signals: *hold*, *valid* and *data_out*. The *hold* signal is used as input for the back-pressure-mechanism. The other signals are used for valid data communication.

An example of this mechanism can be seen in the following listing.

Listing C.1: VHDL streaming mechanism

```
1  -- ----------------------------------------------------------------------
2  --
3  -- Title:   Stream mechanism demonstrator
4  -- Author:  Sander van Leeuwen <s.vanleeuwen@hyperiontechnologies.nl>
5  -- Date:    2020-01-30
6  --
7  -- ----------------------------------------------------------------------
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 entity stream_mechanism is
14     port (
15         -- System inputs
16         clk      : in std_logic;
17         rst      : in std_logic;
18         data_in  : in integer;
19         enable   : in std_logic;
20         hold     : in std_logic;
21
22         -- Outputs
23         ready    : out std_logic;
24         valid    : out std_logic;
25         data_out : out integer
26
27     );
28 end entity stream_mechanism;
29
30 architecture direct_hold of stream_mechanism is
31     signal next_ready    : std_logic;
32     signal next_valid    : std_logic;
33     signal next_data_out : integer;
34
35 begin
36
37     ready    <= not hold;
38     streaming : process (all)
39     begin
40         if rst then
41             valid    <= '0';
42             data_out <= 0;
43         elsif rising_edge(clk) then
44             if hold then
45                 valid    <= valid;
46                 data_out <= data_out;
47             else
48                 valid    <= next_valid;
49                 data_out <= next_data_out;
50             end if;
51         end if;
52     end process;
53
54     data_processing : process (all)
55     begin
56         if enable then
57             next_valid    <= '1';
58             next_data_out <= data_in;
59         else
60             next_valid    <= '0';
```

```
61              next_data_out <= data_out;
62          end if;
63      end process;
64
65  end architecture;
```

C.2. Avalon Streaming interface

Intel has written the Avalon Interface [91] specification, in which common interfaces are defined and characterized. It describes, for example, how a peripheral can be connected to a memory bus. In this specification the streaming mechanism is also discussed, including examples and parameters. Following this standard has the advantage of documented operation of the streaming mechanism, making it easier to interpret for people that are new to the code. Besides, it also describes packetized streaming and what is needed for it. As the Avalon specification is a common standard, this standard has been used in the design.