ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

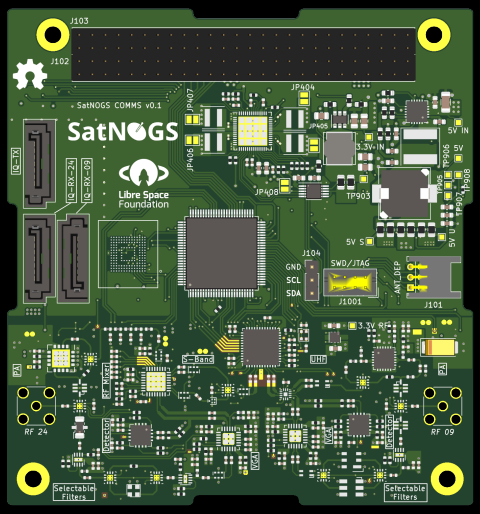
CENTER FOR PROJECT-BASED LEARNING

DEPARTMENT OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Spring Semester 2023

Implementation of FEC on FPGA for a dual-band satellite transceiver

Semester Thesis



Lionnus Kesting
lkesting@student.ethz.ch

March 2023

Supervisors: Dr. Christian Vogt, christian.vogt@pbl.ee.ethz.ch
Nicolas Schärer, nicolas.schaerer@pbl.ee.ethz.ch

Professor: Prof. Dr. Luca Benini, lbenini@iis.ee.ethz.ch

Acknowledgements

I would like to take this opportunity to express my gratitude to the individuals and teams who have played a role in the successful completion of this project. Their support, guidance, and contributions have been invaluable, and I am truly grateful for their efforts.

First and foremost, I would like to thank my supervisors, Christian Vogt and Nicolas Schärer. Their valuable feedback and continuous support throughout the project have been instrumental in shaping its outcome.

I would also like to extend my gratitude to my team members at Akademische Raumfahrt Initiative Schweiz (ARIS). The project they set up provided an interesting and engaging platform for my research. Their collective efforts and dedication have enabled the realization of this project, and I am grateful for the opportunity to be part of such a motivated team.

Within ARIS, I would like to especially thank Lars Horvath for proposing this exact project that happened to align perfectly with my interests. His insight has played a crucial role in driving the project forward, and I am delighted to have had the chance to work on this exciting endeavor.

Lastly, I want to express my appreciation to the developers of the SatNOGS COMMS board for the creation of an exciting open-source satellite communication system, and for the work they already did to develop one of the needed IP cores.

Cover Image Source: NASA

Abstract

The SAGE CubeSat project tackles the critical task of establishing a dependable communication link between space and Earth. It prioritizes adherence to the Consultative Committee for Space Data Systems (CCSDS) standards, a key requirement for uniform space hardware communication. This work centers around the SatNOGS COMMS, an unfinished open-source dual-band transceiver, with an emphasis on signal processing according to CCSDS standards.

This thesis implements a full transmission chain on a Field Programmable Gate Array (FPGA) and sets up communication with the Microcontroller Unit (MCU) over SPI. The proposed system is trialed using FPGA and MCU development boards, comparing the results with a simulation of a CCSDS-compliant transmitter in MATLAB.

Despite requiring more time and resources for a fully functional SatNOGS COMMS transceiver that complies with CCSDS standards, this work provides significant initial steps. It establishes the communication between the FPGA and MCU and implements CCSDS-compliant forward error correction together with the other critical modules for the reliable transmission of data.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix B.

Lionnus Kesting,
Zurich, March 2023

Contents

List of Figures

List of Tables

List of Acronyms

ARIS . . . . . Akademische Raumfahrt Initiative Schweiz
ARQ . . . . . Automatic Repeat Request
ASM . . . . . Attached Synchronisation Marker
AT86IP . . . . . LSF AT86RF215 IP
AT86RF215 . . . Atmel AT86RF215 IC
AXI . . . . . Advanced eXtensible Interface

BCH . . . . . Bose-Chaudhuri-Hocquenghem
BER . . . . . Bit Error Rate
BPSK . . . . . Binary Phase Shift Keying

CADU . . . . . Channel Access Data Unit
CAN . . . . . Controller Area Network
CCSDS . . . . . Consultative Committee for Space Data Systems
CCSDS TF . . . CCSDS Transfer Frame
CLTU . . . . . Command Link Transmission Unit
CRC . . . . . Cyclic Redundancy Check

DDR . . . . . Double Data Rate
DGIER . . . . . SPI Global Interrupt Enable Register
DSP . . . . . Digital Signal Processing

FEC . . . . . .Forward Error Correction

FF . . . . . .Flip-Flop

FIFO . . . . . .First In First Out

FPGA . . . . . .Field Programmable Gate Array

FSM . . . . . .Finite State Machine

GMSK . . . . . .Gaussian Minimum Shift Keying

HAL . . . . . .Hardware Abstraction Layer

IC . . . . . .Integrated Circuit

ILA . . . . . .Internal Logic Analyzer

IP . . . . . .Intellectual Property

IPIER . . . . . .SPI Interrupt Enable Register

IQ . . . . . .In-Phase, Quadrature

KPI . . . . . .Key Performance Indicators

LDPC . . . . . .Low-Density Parity-Check

LSF . . . . . .Libre Space Foundation

LUT . . . . . .Look-up Table

LVDS . . . . . .Low Voltage Differential Signalling

MCU . . . . . .Microcontroller Unit

OBC . . . . . .On Board Computer

OQPSK . . . . . .Offset Quadrature Phase Shift Keying

OSI . . . . . .Open Systems Interconnection

PL . . . . . .Programmable Logic

PLOPs . . . . . .Physical Layer Operations Procedures

QPSK . . . . . .Quadrature Phase Shift Keying

RS . . . . . Reed-Solomon

SAGE . . . . . Swiss Artificial Gravity Experiment
SDR . . . . . Software Defined Radio
SPI . . . . . Serial Peripheral Interface
SPI IP . . . . . AXI Quad SPI (v3.2) IP Core
SPICR . . . . . SPI Control Register
SPIDDR . . . . SPI Data Receive Register
SPIDTR . . . . SPI Data Transmit Register
SPISR . . . . . SPI Status Register
SS . . . . . SPI Slave Select

TC . . . . . Telecommand
TM . . . . . Telemetry

UHF . . . . . Ultra High Frequency

Chapter 1

Introduction

The development of CubeSats revolutionized the space research domain by democratizing access to space for a larger range of researchers. A CubeSat is a type of miniaturized satellite for space research that is made up of multiples of 10×10×10 cm cubic units. These compact, cost-efficient satellites have broadened the scope for conducting space-based experiments, thereby engaging a wider community, including academic institutions, burgeoning companies, and student-led teams [1].

In this context, a student-led organization, known as Akademische Raumfahrt Initiative Schweiz (ARIS), has multiple space-related projects ongoing. A sub-team within ARIS is Swiss Artificial Gravity Experiment (SAGE), whose members are working on developing a 3U CubeSat. The primary objective of this team is to establish a microgravity research platform in orbit. One of the critical challenges faced by SAGE involves establishing a reliable communication link that allows for the transmission of commands to the spacecraft and the reception of payload data from the spacecraft.

To address this challenge, the team has adopted a partially developed dual-band transceiver, which is an open-source project of Libre Space Foundation (LSF) [2]. The necessary hardware, including the PCB with Field Programmable Gate Array (FPGA) and Microcontroller Unit (MCU), has been developed by the SatNOGS team and will be provided by SAGE. The transceiver hardware serves as the foundation for establishing communication capabilities for the CubeSat, but additional measures are required to enhance its reliability and its adherence to certain protocols.

The primary objective of this project is to set up the full transmission chain on the Field Programmable Gate Array (FPGA) and to set up communication with the Microcontroller Unit (MCU), responsible for packaging the telemetry data and relaying the received command to the On Board Computer (OBC). By adhering to the Consultative Committee for Space Data Systems (CCSDS) standards for Telemetry (TM) and Telecommand (TC), specifically, CCSDS 401.0-B-32 [3], CCSDS 132.0-B3 [4], CCSDS

131.0-B-4 [5], CCSDS 231.0-B-4 [6], and CCSDS 231.0-B-4 [7], the project ensures compatibility and interoperability with existing space communication systems and with the SatNOGS ground stations operated around the globe.

1.1. Objective

Since the hardware of the SatNOGS COMMS board has already been developed, the focus is on implementing the required functionality through the generation of In-Phase, Quadrature (IQ) signals. These signals are essential for the integrated transceiver IC on the SatNOGS COMMS board, allowing for transmission which is compliant with CCSDS protocols. Therefore, the objective of this project is to develop the necessary code for the FPGA to enable signal processing, including error correction, to facilitate the reliable transmission of data from the CubeSat.

1.2. Research Questions

1. What are the requirements for satellite communication, according to the CCSDS recommended standards?
2. Which FEC techniques are suitable for satellite communication?
3. How can this system be implemented on an FPGA?

1.3. Outline

This report is organized into five main sections. Following an initial introduction to the project, the second a focus is put on the theory associated with the system's implementation. Here, the topics outlined in the relevant Consultative Committee for Space Data Systems (CCSDS) Recommended Standards are outlined. Here, the Forward Error Correction (FEC) codes that have been suggested are accentuated. Further, the hardware employed for this implementation is presented, including the microcontroller unit (MCU), Field Programmable Gate Array (FPGA), and Integrated Circuit (IC) transceiver. As a concluding point in the theoretical chapter, an introduction to the Advanced Extensible Interface 4 (AXI4) and Serial Peripheral Interface (SPI) that interconnect the subsystems is given.

In the third segment, related work will be discussed. Previous implementations and similar projects will be showcased, identifying potential shortcomings.

The fourth section details the implementation process. Here, the proposed system architecture is presented and justified. After this, the selected Intellectual Property (IP) cores are explored and the integration into the system is analyzed in-depth. The section finished with the proposal of five self-developed modules capable of implementing the transmission (TX) chain of the system.

The fifth chapter is dedicated to discussing the outcomes of this project. In this section, the experimental setups are detailed to authenticate the implementation. Subsequently, in the last section, the results are assessed and the limitations are identified. It concludes with an overview of the remaining tasks necessary for the full implementation of the dual-band satellite transceiver system.

Chapter 2

Theory

To implementing the discussed SatNOGS COMMS communication system, the requirements have to be known or set first. This entails deciding on the hardware to be employed and the protocols to be followed. This section discusses the applicable CCSDS Recommended Standards, with a particular emphasis on the Forward Error Correction (FEC) methods contained therein. We will also delve into the selected hardware, including the FPGA, MCU, and transceiver IC, and subsequently introduce and analyze the interface protocols between subsystems.

The recommendations differentiate between TM andTC. TC refers to data sent from Earth to the spacecraft, while TM signifies data sent from the spacecraft to Earth, as illustrated in Fig. 2.1.

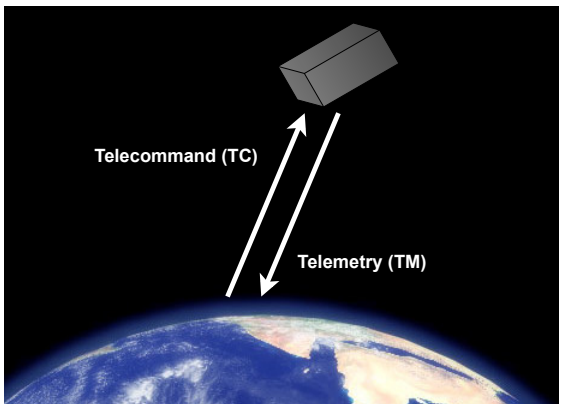


Figure 2.1.: Illustration of Telemetry (TM) and Telecommand (TC). Background image adapted from NASA.

2.1. Consultative Committee for Space Data Systems Recommended Standards

The communication of the CubeSat should follow the CCSDS Recommended Standards, specifically [3, 5, 4, 7, 6]. The standards [5, 7] cover the Synchronization and Channel Coding for TM and TC, respectively, standards [4, 6] cover the Space Data Link Protocol also for TM and TC, respectively. Lastly, [3] covers the RF specifications such as frequency utilization, power limitations, modulation methods, and operational procedures. The standards listed above will be discussed in more detail.

2.1.1. Space Data Link

In the CCSDS Recommended Standards, a customized Open Systems Interconnection (OSI) model is defined [5, 7], as seen in Fig. **??**.

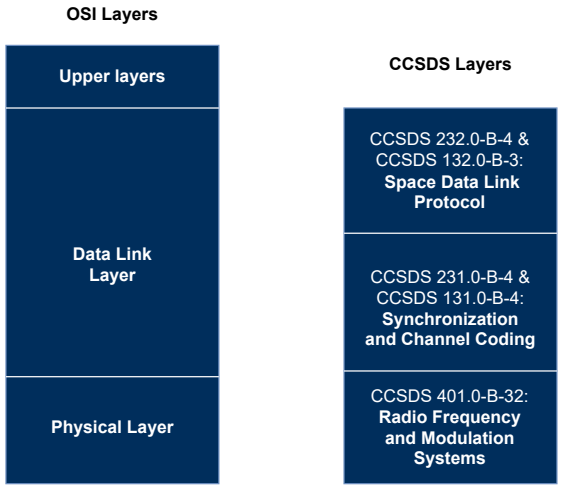


Figure 2.2.: Illustration between the analogy of OSI layers and the CCSDS layers.

For the CCSDS case, the OSI Data Link Layer is replaced by two separate layers, the upper layer is the Space Data Link Protocol, and the lower layer is the Synchronization and Channel Coding protocol. Since the Space Data Link specifies how the packets should be handled this is more relevant for the development of the MCU code, whereas the Synchronization and Channel Coding covers the error correction, which is more relevant for the FPGA code.
The Space Data Link standards also define the structure of a CCSDS Transfer Frame (CCSDS TF). A CCSDS TF consists of a header, the actual data, and optionally, a frame error control field. The structure of the whole frame is shown in Table 2.1. Here it is seen that the header has a length of 5 bytes. The length of the actual data field is variable and is indicated in the CCSDS TF header. The length includes both the header and the

optional frame error control field. The maximum length of a CCSDS TF is determined by either the used FEC code or the number of bits specified in the header field. With 10 bits allocated for the frame length, the maximum length of a CCSDS TF is 1023 bytes. This leaves a maximum of 1023-5=1017 bytes of actual data per CCSDS TF. The exact dimension can be made variable in the HDL code such that the length can be set by other constraints arising in later development stages of the SAGE project.

Table 2.1.: CCSDS Transfer Frame

Field	**Bits**
Transfer Frame Version Number	2
Bypass Flag	1
Control Command Flag	1
Reserved Spare	2
Spacecraft Identifier	10
Virtual Channel Identifier	6
Frame Length	10
Frame Sequence Number	8
Actual data	variable
(Optional) Frame Error Control	16

2.1.2. Synchronization and Channel Coding

The CCSDS 231.0-B-4 [7] and 131.0-B-3 [5] Recommend Standards specify the lower part of the OSI Data Link Layer. Firstly, it specifies for both TM and TC separately what the allowed FEC codes are, and it defines ranges or specific values of managed parameters such as coding rates, code block sizes, and generator polynomial values. Furthermore, it specifies how to synchronize the CCSDS TF for each FEC code. Lastly, it specifies the length of the CCSDS TF since this value can be constrained depending on the type of FEC used.

The way the CCSDS TFs are transmitted is different for TM and TC, which use Command Link Transmission Unit (CLTU) and Channel Access Data Unit (CADU), respectively. The CLTU is a data packet consisting of the encoded codewords containing command data, it provides synchronization and an indication where the data starts. The length of a CLTU depends on the used FEC. For BCH it consists of a 16-bit start sequence, a 64-bit codeword containing the CCSDS TF, and a 64-bit tail sequence. For Low-Density Parity-Check (LDPC) this consists of a 64-bit start sequence, a 128 or 256-bit codeword, and an optional 128-bit tail sequence. The CADU is similar but consists of an Attached Synchronisation Marker (ASM) together with a CCSDS TF for convolutional coding, and of the ASM and the code word/ code block for the other types of coding

specified for TM. Here, the ASM is used as a way of performing frame synchronization. This will be discussed first.

Attached Synchronisation Marker

The ASM is a specific string of 32 bits that serves as a header for a CADU and serves the purpose of frame synchronization [5]. The ASM is placed in front of every CCSDS TF or code block, depending on the type of FEC used. For convolutional encoding, the ASM is encoded as well. In case of a concatenated code, the ASM shall be encoded by the inner code, but not by the outer code. Specifically, for a uncoded, convolutional encoding, Reed-Solomon (RS), concatenated encoding, 7/8 LDPC coding, the ASM code consists of the 32-bit hexadecimal code

$$\text{ASM} = \text{0x1ACFFC1D}. \tag{2.1}$$

Lastly, to ensure that the transmitted signal can be synchronized reliably at the receiving end, enough bit transitions have to take place. In [5] this is implemented by integrating a pseudo-randomizer, or scrambler, module. This method will later be explained in more detail.

2.1.3. Earth Stations and Spacecraft: Radio Frequency and Modulation

Modulation

The modulation methods in the CCSDS Recommended Standard [3] depend on both the distance at which the satellite will be and on the direction of communication (space-to-earth/TM, or earth-to-space/TC). A 'Category A' spacecraft is recognized as a spacecraft with an orbit altitude of less than 6000 km, which is the case for the SAGECubeSat. Therefore, only the recommendations for a 'Category A' spacecraft are considered here. The recommendations of the modulation are shown in Tab. 2.2.

Here, it is noted that the BPSK, QPSK, and OQPSK refer to baseband-filtered modulation techniques. The specific modulation technique can be chosen differently for TM and TC, however, it is noted that for both links the Binary Phase Shift Keying (BPSK) modulation can be used. Therefore, this modulation technique will be implemented first.

2.2. Forward Error Correction

To enable a reliable communication link in space, some type of error mitigation is essential. Two main techniques can be used, Automatic Repeat Request (ARQ) or FEC [8].

Table 2.2.: Modulation techniques for the CCSDS Recommended Standard [3]

Mode	**Modulation**	**Data rate**
Telemetry	GMSK	<10 coded MSymbol/s
	BPSK	<2 coded MSymbol/s
	QPSK	<2 coded MSymbol/s
	OQPSK	<10 coded MSymbol/s
Telecommand	PSK	Low data rate (<8 kSymbol/s)
	PCM/PM/bi-phase-L	Medium data rate (8-256 kSymbol/s)
	BPSK	High data rate (256k-2048MSymbol/s)

For ARQ the receiving system can detect errors using parity bits, and it will request for retransmission of a data block when an error is detected. For FEC, the receiving system can also detect errors, but additionally, it has the means to deduce where the error is exactly, and thus how to correct the received data to form the original message. Since the earth-space communication link introduces a notable transmission delay, a system with ARQ is infeasible since it would introduce long idle periods to wait for a response [8]. Therefore, FEC is used for this communication system, the details of which will be discussed now.
A FEC code can be classified into two classes:

- Block code: It maps the input of k bits to an output of n bits. This yields a (n, k) block code with $n > k$. A block coder is a memoryless device [8].
- Convolutional code: The coder accepts k input bits and produces n output bits. The output depends on the last K inputs, which is called the constraint length. The last K inputs are stored in the internal register which therefore contains kK bits. The output is generated using the kK stored bits using some logic defined by the specific convolutional code definition.

To select a protocol, the performance of them has to be evaluated and compared. This is done using their Key Performance Indicators (KPI). For this analysis, the chosen KPI are:

- Code rate, r: Code rate is the ratio of the number of input bits to the number of output bits after encoding, so $r = k/n$. A higher code rate means more information can be transmitted per unit of time [8].
- Coding gain: Coding gain is the reduction of the energy-per-bit/noise-density ratio (E_b/N_o) when the FEC is used [8].
- Resource Utilization: Based on previous research, an estimate is made of how much FPGA resources are required to implement this type of FEC. This determines if it is feasible to implement on an FPGA.

- FEC class: Block or convolutional code.

In the CCSDS Standards [5, 7], various FEC codes are proposed. Since requirements are different for TM and TC, the proposed codes also differ. The computing power at the ground station is not a major constraint. However, it is limited on the satellite due to space and power constraints. Therefore, the requirement is that both the encoding for TM and decoding for TC requires less computational effort. Additionally, due to the higher data volume of TM, the satellite needs to encode more data, preferably at a higher data rate. The proposed codes will be discussed in detail.

For TM the options in [5] are convolutional encoding, Reed-Solomon coding, concatenated coding, Turbo coding, and LDPC coding. For TC the options consist of BCH and LDPC. The working of these codes, the restraints CCSDS put on the usage of them, and some sample implementations are covered next.

2.2.1. Convolutional Encoding

Figure 2.3 illustrates the functioning principle of the CCSDS convolutional encoder. In this specific implementation, the number of input bits, k, is one per clock cycle, resulting in two output bits, n, per clock cycle. Consequently, this exemplifies a code rate, r, of $1/2$. While other code rates such as $2/3$, $2/4$, $5/6$, and $7/8$ are permitted using punctured convolutional codes, they offer less robust error correction capabilities [5]. Additionally, the constraint length is set to be 7, resulting from the six single-bit delays(memory elements, M), as $M = K - 1$ [9]. The logic between the $n = 2$ outputs and the $K = 7$ register states is denoted by two connection vectors $G_1 = 1111001$ and $G_2 = 1011011$.

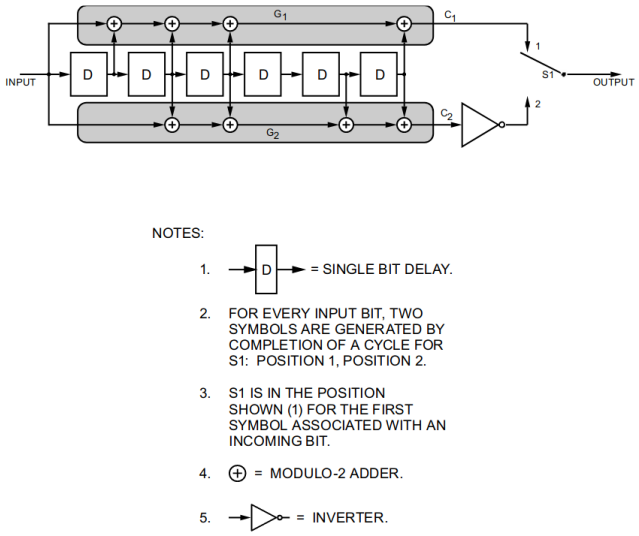


Figure 2.3.: Flowchart of the CCSDS convolutional encoder, figure adapted from [5].

2.2.2. Reed-Solomon Encoding

The second FEC for TM is the Reed-Solomon (RS) coding, a member of the block codes class. RS is an error correction technique that excels in mitigating errors in burst communication channels prone to bursts of errors [5]. An RS(n, k) code possesses k inputs and n outputs and appends $2EI = n - k$ bits that facilitate error detection and correction. Here, E is the number of errors the code can correct per block and I is the interleaving depth. Per [5], the codeword is fixed to 255 bytes, and since E is fixed to 8 or 16, the possible options are either RS$(255, 223)$ or RS$(255, 239)$, respectively. So, this limits the maximum code block length to

$$L_{max} = nI = 255I\text{bytes}. \tag{2.2}$$

The interleaving depth, I, is fixed to either 1, 2, 3, 4, 5, or 8. The generator polynomial is defined as

$$F(x) = x^8 + x^7 + x^2 + x + 1. \tag{2.3}$$

An implementation of the RS encoder can be seen in Fig. 2.4. Here, the data is fed sequentially, and the parity bits are given as the values shown as q_0 through q_31 [10, 11].

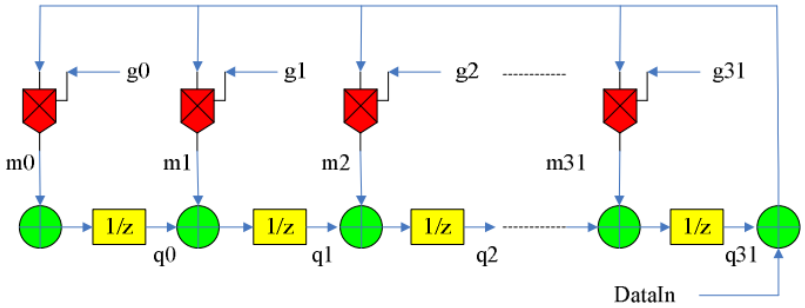


Figure 2.4.: Flowchart of a Reed-Solomon encoder, figure adapted from [10].

2.2.3. Concatenated Encoding

The third FEC for TM is the concatenated coding, which is an approach where two FEC codes are combined to yield an improved performance, without adding much more complexity. The scheme involves an inner and an outer code, where [5] defines that the outer code shall be a RS code and the inner code shall be a convolutional code.

2.2.4. Turbo Encoding

The fourth TM FEC technique is turbo coding, which employs two parallel recursive convolutional codes. Here, the input to an encoder is a permuted version of the input

data to another encoder. Turbo codes yield promising results close to the Shannon limits [9]. However, for decoding a more complex iterative decoding technique has to be deployed, and additionally, a pseudo-randomizer needs to be integrated since it does not guarantee sufficient bit transitions [5]. Moreover, in [5] it is defined that the convolution codes shall both have 16 states, and the nominal code rates, r, shall be $1/2$, $1/3$, $1/4$, or $1/6$. The input of a Turbo encoder is k bits wide, which is the information block length. This length can be selected from 223 times 1, 2, 3, 4, or 5 bytes. These lengths correspond to RS with corresponding interleaving depth, I, of 1, 2, 3, 4, and 5. Longer block lengths achieve the highest gain but a higher latency [5]. The corresponding code block is calculated from the information block length and the nominal coding rate as $n = (k + 4)/r$ [5]. The flowchart given in [5] can be seen in Fig. 2.5. The connection vectors are defined to describe the logic, just as for the convolutional encoder. The difference here is the addition of the backward connection vector, G_0, which implements the recursive behavior of the code. Lastly, the exact locations where the permutations have to be done are defined in [5].

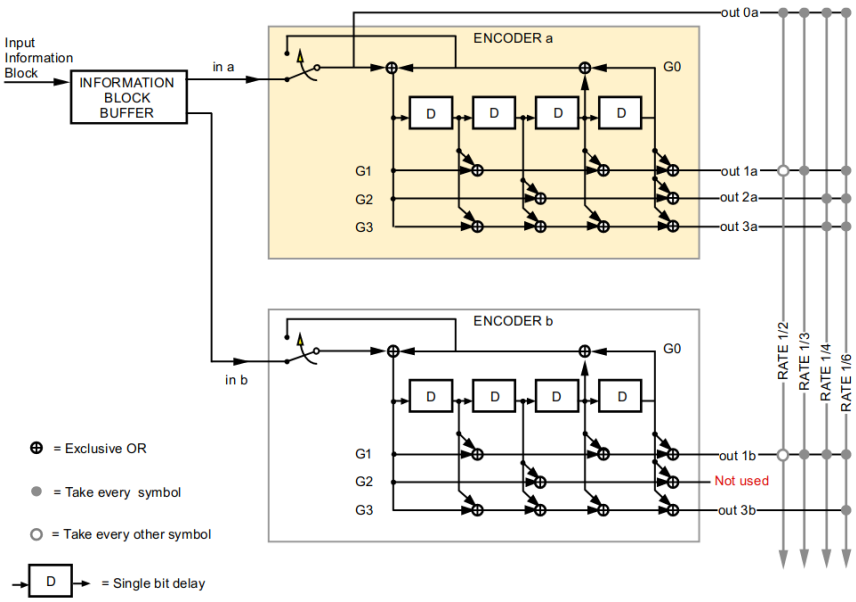


Figure 2.5.: Flowchart of the CCSDS Turbo encoder, figure adapted from [5].

2.2.5. Low-Density Parity-Check (LDPC) Encoding

LDPC coding is an error correction method that leverages sparse parity-check matrices for encoding and decoding data [5]. As a member of the block code class, LDPC operates on extensive codewords, often composed of hundreds or thousands of bits. The LDPC code is defined for both TM, and TC. Moreover, this FEC strategy can offer more significant code gains than concatenated coding systems [5]. The LDPC specified in [5] corresponds to an

$(n, k) = (8176, 7136)$ code, yielding a code rate of $r = 223/255$ and thereby matching the length of an RS$(255, 223), I = 4$ code. An implementation of an LDPC implementation on FPGA can be seen in Fig. 2.6. It is immediately noticed that this is a much more complex structure than the other types of FEC. However, this specific implementation utilizes multiple convolutional encoders in parallel to boost the throughput. However, in [12] it is also seen that the resource utilization is in the range of 500-3000 Flip-Flops and 2500-8000 Look-up Table (LUT)s.

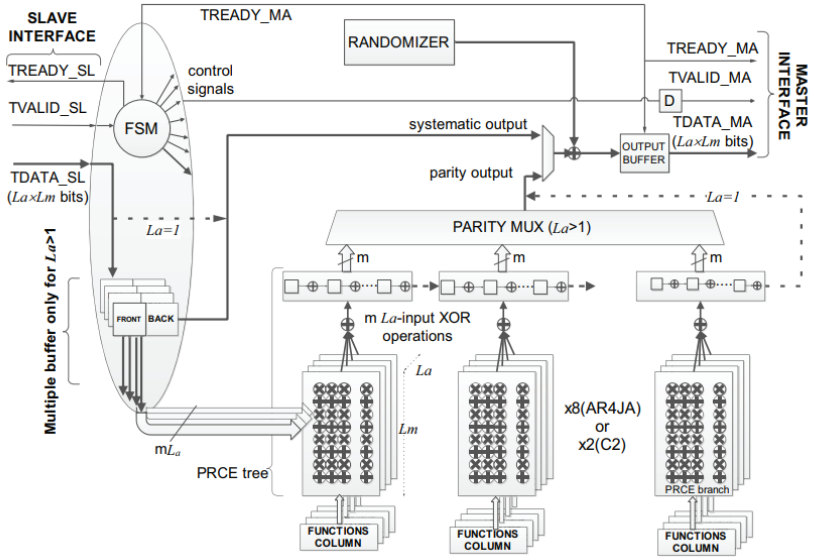


Figure 2.6.: Implementation of an LDPC encoder, figure adapted from [12].

2.2.6. BCH Encoding

The last FEC code is only specified for TC. The BCH is classified as a block code, the parameters of which are fixed by [7]. The codeword consists of 64 bits, of which 56 are information bits, 7 are parity bits and there is one appended filler bit. This can be seen in Tab. 2.3. Therefore, the specified code is the $(n, k) = (63, 56)$ code, yielding a code rate of $7/8$. Specified in [7] is the polynomial

$$g(x) = x^7 + x^6 + x^2 + 1, \tag{2.4}$$

which determines the connections in the encoder. The implementation of [7] can be seen in Fig. 2.7. Moreover, an FPGA implementation of the CCSDS BCH code is seen in [13].

To identify the most appropriate FEC schemes for implementation, a study was performed highlighting KPI for each FEC to facilitate a qualitative comparison. Since the

Table 2.3.: BCH Codeword Structure

Bit Position	[63:7]	[7:1]	[0:0]
Content	Information	Parity	Filler

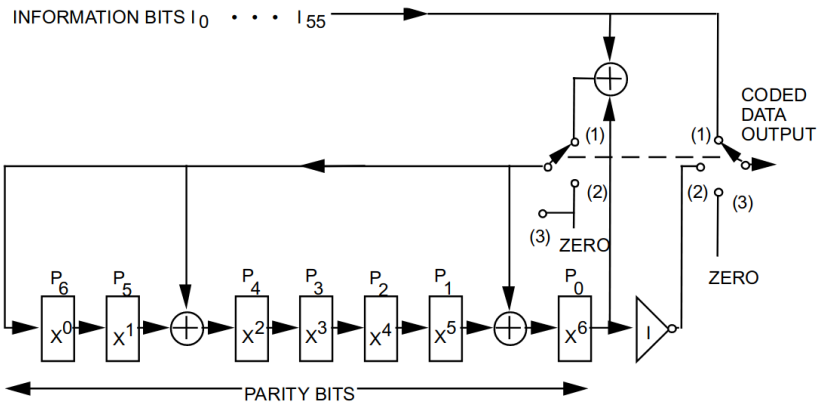


Figure 2.7.: Flowchart of the CCSDS BCH encoder, figure adapted from [7].

range of FEC codes available was broad, the KPI considered for this analysis are coding gain and resource utilization. Other KPI such as throughput, latency, and power will be considered for the final implemented system. Next to these performance indicators, some properties that are defined in [5, 7] are shown. These include the link type for which the code is recommended (TM or TC), the allowed code rates, and the type of the code (block code (B) or a convolutional code (C)).

Table 2.4.: Comparison of FEC codes described by CCSDS. Resource utilization for encoder structures.

FEC Type	**Mode**	**Code Rate**	**Coding gain**	**RU**	**Type**
Convolutional	TM	1/2	M [14]	2 LUT, 6 FF [15]	C
Reed-Solomon	TM	223/255= 7/8	M [16]	48 LUT [17]	B
Turbo	TM	1/2-1/6	H [18]	51 LUT, 41 FF [19]	B
LDPC	TM/TC	1/2-7/8	H [18]	500-2500 LUT, 2500-8000 FF [12, 20]	B
BCH	TC	7/8	M [18]	40 LUT, 8 FF [13]	B

The findings of the comparison are depicted in Table 2.4. Please note that the coding gain values are represented as low (L), medium (M), and high (H) to provide a comparative assessment of the FEC protocols. This is an estimate since the exact coding gains are highly dependent on the specific parameters of the code, for example, the code word length for LDPC and Turbo codes [18], or the chosen error correction capability for RS [5].

Due to its efficient resource utilization and satisfactory coding gain, the convolutional

encoding will be implemented on the TM link in this thesis. Having now chosen an FEC technique, the hardware used in this thesis is discussed next.

2.3. Hardware

As mentioned previously, the hardware is already provided and consists of the SatNOGS COMMS board. The system design document [21] goes over the general system design, and contains the overview seen in Fig. 2.8.

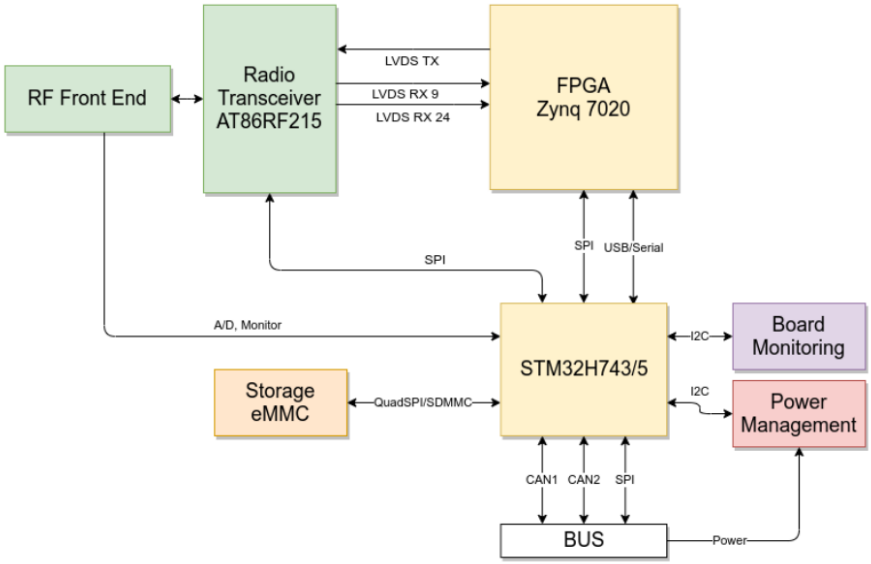


Figure 2.8.: Flowchart of the SatNOGS COMMS system, figure adapted from [21].

Here, it is seen that the system consists of an STM32H7 MCU which receives and transmits data to the OBC via Controller Area Network (CAN) and has access to storage. Moreover, it performs some monitoring and management of the whole communication module. The MCU is also responsible for transmitting and receiving the data that is being exchanged over the wireless link, this is done via the SPI connection to the FPGA. Here, the FPGA is responsible for the processing of this data, generating the transmitting IQ signals, or retrieving the data bits from the received IQ signals. The FPGA additionally has a Low Voltage Differential Signalling (LVDS) interface with the transceiver IC, the Atmel AT86RF215 IC (AT86RF215). In the following section, the three separate hardware parts relevant to this project, the FPGA, the MCU, and the transceiver IC, will be discussed in more detail.

2.3.1. Field Programmable Gate Array

The FPGA that is used by the SatNOGS COMMS board is the Zynq 7020 [22]. This specific FPGA is mounted on top of the ALINX AC7020 core board [23], a space-efficient module that integrates both the FPGA and separate 1 GB DRAM chips and a 32 MB QSPI Flash. Specification for the FPGA and the board can be seen in Tab. 2.5. Here it is noted that the FPGA has two separate ARM™Cortex™-A9 cores, which can be used for the Zynq Processing System, next to the Programmable Logic [22]. Here, it is seen that the number of available LUTs is 53,200, and 106,400 Flip-Flop (FF)s are available. For this thesis, only the Programmable Logic (PL) will be used and the synthesis, implementation, and generation of the final bitstream are done with Vivado 2020.2 [24]. Moreover, the debugging interface based on the Internal Logic Analyzer (ILA) IP cores is used to debug the hardware whilst running the communication system. The block design in Vivado Design Suite will be used to easily prototype with the generated modules. It is also noted, that for testing the PYNQ-Z2 development will be used, which features almost the same FPGA as the ALINX AC7020 board, albeit with a slightly slower speed grade.

Table 2.5.: FPGA Module Specifications [22, 23]

Parameter	**Value**
FPGA	XC7Z020-2CLG400I
Processors	Dual-core ARM™Cortex™-A9
PS Terminal	1GB DDR3, 32-bit
Flash Memory	32MB
Logic Cells	85,000
Look-Up Tables (LUTs)	53,200
CLB Flip-Flops	106,400
DSP Slices	220
Block RAM	4.9Mb
IO pins	200
ADC	2x 12-bit at 1 MSample/s
Speed Grade	-2, Industrial Grade
Max. Clock Freq.	766 MHz
Working Temperature	-40 C to 85 C
Size	35 × 42 mm

2.3.2. Microcontroller

The MCU that is used by the SatNOGS COMMS board is the STM32H743. A Nucleo-144 development board [25] with the same MCU as on the SatNOGS is used to interface with the FPGA development board for initial testing. The MCU is capable of establishing SPI

communication with the FPGA and establishing UART communication to the terminal on a PC. The MCU is programmed using the STM32CubeIDE [26].

2.3.3. Transceiver IC

To convert the IQ output signals of the FPGA into a carrier-modulated signal, a separate transceiver IC is used on the SatNOGS COMMS board, the Atmel AT86RF215 Integrated Circuit (IC) [27]. This IC is a fully integrated radio transceiver that consists of two separate radios, the S-Band 2.4 GHz radio (RF24) and the UHF-Band Sub-1 GHz radio (RF09). Next to this, it has two integrated baseband cores, capable of performing modulation and error correction. However, the supported error correction techniques do not satisfy the CCSDS Recommended Standards and are thus insufficient. Instead, the IQ interfaces of the separate radios are accessed by the already implemented IP core from LSF, as given in their repository [28]. LSF developed two IP cores, one for receiving data and one for transmitting data. The TX Core has an AXI4-Stream Slave interface, and the RX Core has an AXI4-Stream Master interface. The I and Q data consist both of 13-bit signed 2's complement samples. This data is sent in 32-bit words, according to the word format shown in Tab. 2.6. Here, it is seen that additional synchronization bits are sent to distinguish I and Q data. Additionally, the LSB of the 32-bit word is a control bit (`CTRL`) that can utilize the embedded control capability of the transceiver. This control bit is, therefore, only present for the transmitted data, for the received data this bit will equal 0.

Table 2.6.: AT86RF215 I/Q Data Word Format

Bits	[31:30]	[29:17]	[16:16]	[15:14]	[13:1]	[0:0]
Name	`I_SYNC`	`I_DATA`	-	`Q_SYNC`	`Q_DATA`	`CTRL`
Data	0b10	I Data (13 bits)	0	0b01	Q Data (13 bits)	1/0

To send the data to the transceiver IC, a 64 MHz Double Data Rate (DDR) clock has to be used for a LVDS signal. Moreover, the data can be sent at different intervals, as seen in Fig. 2.9. For the highest accuracy, the samples can be sent at a rate of 4 Msample/s. At a data rate of 1 Mbps for QPSK Modulation and 1/2 convolutional encoding, this yields 4 samples per symbol.

2.4. Interfaces

To connect all the subsystems on the SatNOGS COMMS board, multiple communication protocols are used. Most notably, the Serial Peripheral Interface (SPI) communication between the MCU and the FPGA, and the AXI4 protocol for communication between different IP cores in the FPGA. The communication between the FPGA and the transceiver

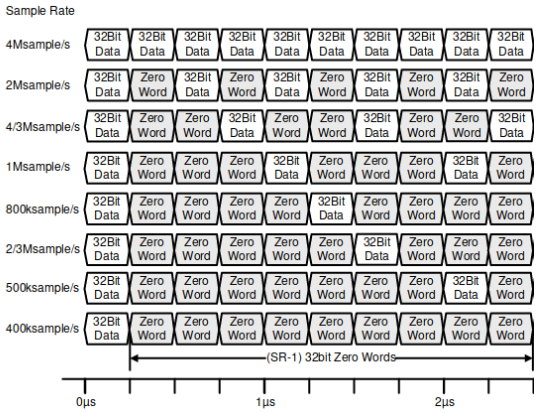


Figure 2.9.: Overview of the possible intervals of new data to the AT86RF215. Figure adapted from [27].

IC is done via LVDS. LVDS is not discussed here since this is implemented already by the developers of the SatNOGS COMMS board, LSF in their IP core [28].

2.4.1. Serial Peripheral Interface

SPI communication is a synchronous, full-duplex serial communication protocol commonly used for short-range communication between microcontrollers, sensors, and peripheral devices. It involves a master-slave architecture where one device (master) initiates and controls communication with one or more devices (slaves) by providing a clock signal. The particular implementation described here is set by Motorola [29]. The communication for standard SPI occurs over the following four lines:

- SCLK (Serial Clock): The clock signal generated by the master device.
- MOSI (Master Output, Slave Input): Carries the data from the master to the slave during transmission.
- MISO (Master Input, Slave Output): Carries the data from the slave to the master during transmission.
- SPI Slave Select (SS): This line is active-low and is used to select the specific slave device with which the master wants to communicate. Each slave has its own separate SS line, and the master thus as multiple SS outputs.

To complete a bi-directional transfer the following steps are involved:

1. The master pulls the SS line of the desired slave device low, which indicates the start of the communication.

2. Simultaneously, the master generates clock pulses on the SCLK line. This happens only during transmission and thus dictates whether a transfer is happening.
3. The master and slave devices publish data simultaneously on the MOSI and MISO lines, respectively.
4. The number of clock pulses determines the length of the data transfer. The length of an SPI transaction for the STM32H723 can be 8, 16, or 32 bits.
5. After the data transfer the master pulls the SS line high again and stops sending a clock signal, indicating the end of the communication.

SPI communication supports different configurations, such as clock polarity, clock phase, and data order, which have to be equal for the master and slaves. Overall, SPI is a straightforward communication protocol suitable for bi-directional data transfer between the MCU and the FPGA.

2.4.2. Advanced eXtensible Interface

The Advanced eXtensible Interface (AXI) protocol is widely used for interconnecting components within an FPGA or a System-on-Chip (SoC) design. The protocol knows various versions, the ones used here are the AXI4-Lite [30] and AXI4-Stream [31]. To implement these protocols, it is important to first understand them.
The AXI4-Lite protocol is a simplified version of the full AXI4 protocol lacking burst-access capability. Since the burst-access capability is not needed for our purposes the AXI4-Lite protocol will be implemented.

With the AXI4-Lite protocol, communication is established between a single master and a single slave device. It is a memory-mapped protocol, so the master initiates read and write transactions to access the slave's registers. Establishing communication over an AXI4-Lite bus can be separated into two distinct phases: Read Data Transaction and Write Data Transaction. As seen in [32], the read/write transactions are executed as follows:

1. Read Data Transaction:
    a) The master asserts the `ARVALID` (Address Read Valid) signal along with the address on `ARADDR`. It can also already assert the `RREADY` (Read Ready) signal to indicate it is ready to receive the data.
    b) The slave asserts the `ARREADY` (Address Read Ready) signal to acknowledge it received the address.
    c) After the address exchange, the slave accesses the requested register and places the data on the `RDATA` (Read Data) line. The slave asserts the `RVALID` (Read Valid) signal to indicate valid data.

d) The slave waits for the master to assert `RREADY` before deasserting `RVALID`.

2. Write Data Transaction:

a) The master asserts the `AWVALID` (Address Write Valid) signal along with the address on `AWADDR`. It will also already assert `BREADY` (Response Ready) signal, to indicate it is ready to receive a response.

b) The slave asserts the `AWREADY` (Address Write Ready) signal to acknowledge it received the address.

c) After the address phase, the master places the data to be written on the `WDATA` (Write Data) line, and the master asserts the `WVALID` (Write Valid) signal to indicate valid data.

d) The slave receives the data at the moment the `WVALID` and `WREADY` handshake occurs.

e) The master waits for the slave to assert `WREADY` before deasserting `WVALID`.

f) The slave asserts the `BVALID` (Write Response Valid) signal and puts a status message on the `BRESP` (Write Response) signal, indicating a successful or failed data exchange.

In the end, AXI4-Lite is a well-suited protocol for on-chip communication between the different modules. A lot of IP cores in the Vivado Design Suite use the AXI4 communication protocol, and therefore it makes sense to implement this in a custom module as well.

Now, understanding the context of the project, the related work can be explored.

Chapter 3

Related Work

For this project, the hardware is already available and is developed and produced by a team at LSF. The SatNOGS COMSS board is entirely open-source, and all the hardware and software files can be found at [33]. In this repository, the FPGA folder was found to be empty at the time of this thesis. Another repository [28] contains an IP core implementing the conversion between an AXI4-Stream and the needed LVDS protocol to communicate with the transceiver IC.

As a matter of fact, there exists a similar project in which an open-source communication module has already been released [34], the UPSat. The UPSat is a CubeSat launched in 2017 by LSF and is the first open-sourced hardware and software satellite [35]. The goal of the SatNOGS COMMS project is, however, the implementation of an open-source communication module featuring integration with the SatNOGS ground station network, adherence to the CCSDS protocols, with data rate of up to 1 Mbps for the S-band radio [36] The UPSat communication module is found to utilize a single IC for the communication, the Texas Instruments CC1120 [37]. This chip is only designed for Ultra High Frequency (UHF) with a limited data rate of 200 kbps [37] and limited modulation options and no FEC, thus not compliant with CCSDS standards. Therefore, this open-source design does not yet satisfy the goals of the SatNOGS COMMS.

Regarding the implementation of the communication system according to the CCSDS standards, there is another team working on this, ACubeSat. Specifically, the ACubeSat team has worked on implementing LDPC FEC for their CubeSat [38]. However, they are implementing this on an MCU instead of an FPGA. Here, it is noted that the ACubeSat team was having difficulties with the high computational effort for an MCU to perform LDPC encoding and concluded LDPC decoding was even unfeasible on the CubeSat. Here, the use of an on-board FPGA can relieve the MCU from all computationally heavy signal processing, enabling the implementation of the most sophisticated FEC available within the CCSDS Standards. An implementation of the LDPC encoder for an FPGA has

been investigated in [12]. This would be a viable approach, but the resource utilization was in a much higher range than for the RS [17], convolutional encoding [15] or BCH encoding [13], which is a few tens of LUTs, compared to hundreds or thousands of LUTs for LDPC [12].

The CCSDS 131.0-B-4 [5] has been implemented in MATLAB [39]. This can be used as a verification of the final communication system, where the IQ signals measured at the output of the FPGA can be directly compared against the IQ signals created by this MATLAB package. Moreover, this can be utilized to do a qualitative comparison of the FEC protocols specified in the CCSDS standards in terms of ber.

Chapter 4

Implementation

For the realization of the communication system according to CCDS Recommended Standards, an FPGA implementation is selected to have full freedom over creating the IQ waveforms. This means there is complete freedom for implementing the FEC, modulation, filtering, and synchronization techniques. An overview of the system can be seen in Fig. 4.1. This overview includes the MCU which receives the data from the OBC and the interface with the transceiver IC. The development of the system is divided into two parts, the implemented modules on the FPGA, and the software written for the MCU.

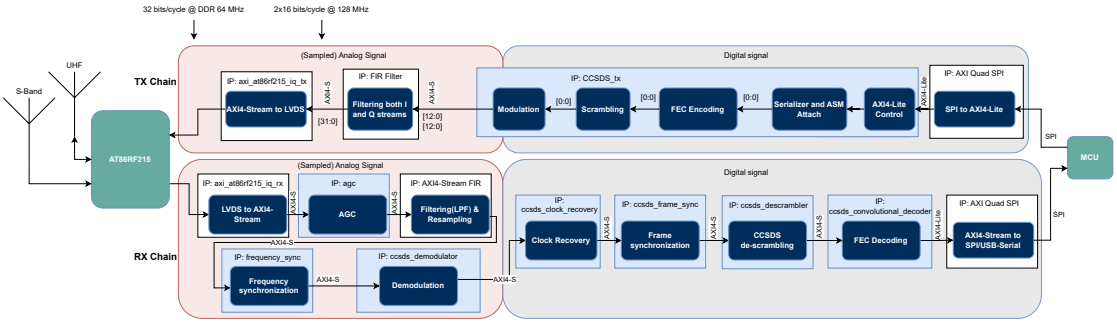


Figure 4.1.: Flowchart of the processing chain of the data through the FPGA.

4.1. Developed HDL Modules/IP Core

In order to streamline the development process of various modules such as the convolutional encoder, scrambler, and modulator, a direct chain connection has been established between them. To implement the SPI communication on the FPGA end, the AXI Quad SPI (v3.2) IP Core (SPI IP) module [40] was selected. This specific module demands an

AXI4(-Lite) interface, while the LSF AT86RF215 IP (AT86IP) RX and TX IP cores [28] necessitate an AXI4-Stream protocol. In response to these prerequisites, an IP core has been developed to manage the data handling process conform CCSDS Standards, and to transmit data at the required rate.

4.1.1. CCSDS TX Chain IP - `ccsds_tx_ip_v1_0.sv`

To facilitate the system's interfacing with the SPI interface of the MCU and the LVDS interface of the AT86IP, certain IP cores can be used which implement these communication protocols. The previously discussed LSF IP core implements the interface with the AT86RF215 and is thus chosen as output. When it comes to SPI communication, a variety of options exist. As an initial choice, the SPI IP has been selected due to its pre-existing integration into Vivado, which indicates it's a thoroughly tested IP core. One potential drawback is its proprietary nature, as an open-source solution would be more fitting with the rest of the SatNOGS COMMS system. A more detailed discussion on the specific characteristics of the AT86IP and SPI IP will follow to understand the implementation of this control module.

LSF AT86RF215 IP Core

In Fig. 4.2 the block design view of the SPI IP can be seen. Since the AT86RF215 requires a DDR data signal clocked at 64 MHz, this set the clock frequency for the last part of the system. To avoid clock-domain crossings, a frequency of 128 MHz can be used for the whole system.

Xilinx AXI QUAD SPI IP Core

The IP core used for SPI is the AT86IP from Xilinx, the block design of which is seen in Fig. 4.3.

The block can be directly connected to the SPI pins of the FPGA, and interfaces inside the FPGA via AXI4-Lite as a slave. The module contains multiple registers, most importantly the SPI Control Register (SPICR), SPI Status Register (SPISR), SPI Data Receive Register (SPIDDR), and SPI Data Transmit Register (SPIDTR). For the SPIDDR and SPIDTR, the full register can be used for the data that is being sent, which is 32 bits per SPI transaction, or a part for an 8-bit SPI transaction. The module is configurable to contain a First In First Out (FIFO) for both SPIDDR and SPIDTR. The function of the SPICR and SPISR bits can be seen in Tab. 4.1 and Tab. 4.2.

From the SPICR bits it can be seen that various settings can be changed regarding the SPI protocol, e.g. the SPI transactions can be enabled or disabled, and the SPI mode

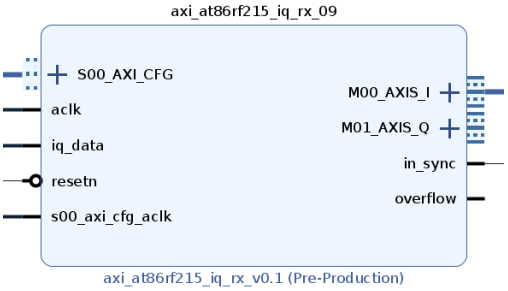


Figure 4.2.: Overview of the AT86IP block design.

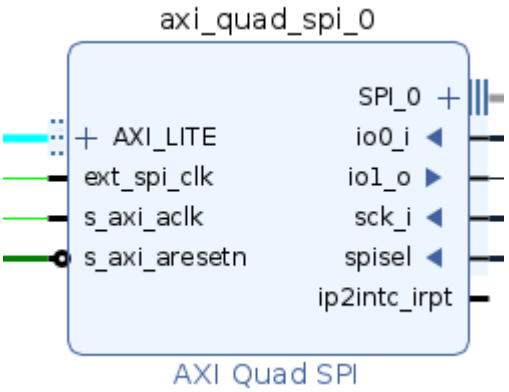


Figure 4.3.: Overview of the SPI IP block design configured as an SPI slave with the `spisel`, or SPI select, input.

Table 4.1.: SPI Control Register

Bit	Description
31:10	Reserved
9	MSB/LSB select
8	Master Trans. Inhibit
7	Manual SS Enable
6	RX FIFO Reset
5	TX FIFO Reset
4:3	CPHA/CPOL Config.
2	Master/Slave select
1	SPI System enable
0	Loopback enable

Table 4.2.: SPI Status Register

Bit	Description
31:11	Reserved
10:6	Error indicators
5	‘spisel‘ indicator
4	Mode-fault error
3	TX Full
2	TX Empty
1	RX Full
0	RX Empty

can be switched between master and slave. The bits in the SPICR can be changed during operation. Next to these settings, there are more customization options that are only adjustable at compile time. This includes the SPI mode (standard, dual, or quad), the number of transaction bits (8, 16, or 32 bits), and the frequency ratio. The SPI clock frequency is determined by the IP core input `ext_spi_clk`, and the frequency ratio (division ratio and multiplication) settings. Lastly, the depth of the FIFOs can be chosen as either 0 (no FIFOs), 16, or 256.
The setting that were chosen for the SPI IP are

- SPI Mode: Slave, since MCU dictates data transfer.
- SPI Type: Standard; relatively slow data rate is required and reduces complexity.

- FIFO depth: 16; makes sure no data is missed but limits used resources.
- Frequency: 4 MHz; covers the maximum data rate but avoids unnecessarily fast clock with a higher error probability.

To utilize the SPI IP as an SPI slave, the following steps need to be performed using the AXI4-Lite interface:

1. Write SPICR with desired settings
2. Enable SPI system by asserting `spisel`, either with the SS line or internally
3. Optionally enable interrupts with certain settings by writing to SPI Global Interrupt Enable Register (DGIER) and SPI Interrupt Enable Register (IPIER).
4. Write SPIDTR if data needs to be transmitted
5. Read SPISR bit 0 to check if new SPI data is received(i.e. if RX FIFO is non-empty)
6. Read SPIDDR for new data if available

To implement these actions, a custom Finite State Machine (FSM) has been developed which is seen in Fig. 4.4.

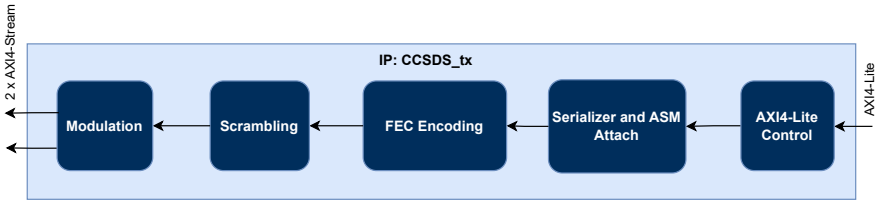


Figure 4.4.: Overview of the encapsulating CCSDS TX IP module, instantiating all the submodules and providing the AXI4 interfaces.

Implementation of the FSM Control module

The input is the AXI4-Lite interface with the SPI IP, and the output is an AXI4-Stream interface to either the AT86IP directly or through the FIR IP [41]. In between the internal modules, there is a single-bit data connection, and a second signal is connected to propagate `cycles_per_bit`, the number of cycles each bit should be present to the modulator core. It is already evident here that there are very low requirements in terms of throughput in this part of the chain. A stricter requirement emerges only after the modulation of the signal as it is filtered. The proposed FSM implementation of the central control module can be seen in Fig. 4.5. This implements the steps needed for performing a transaction with the SPI IP as explained above and it additionally makes the data available to the serializer module which will handle sending out the bits through the TX chain.

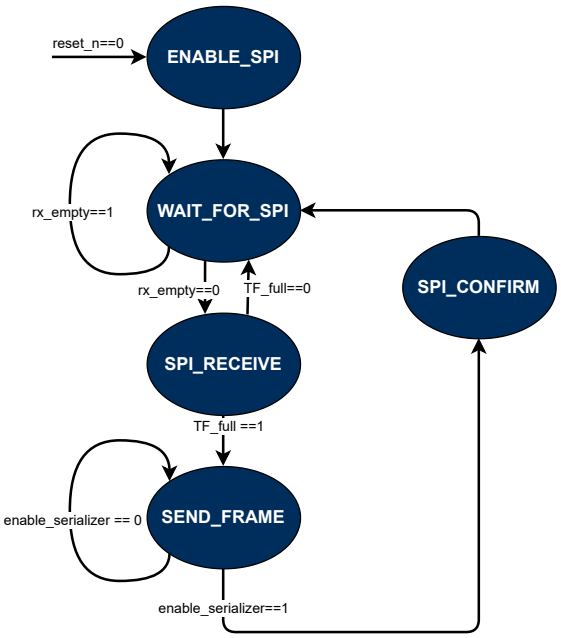
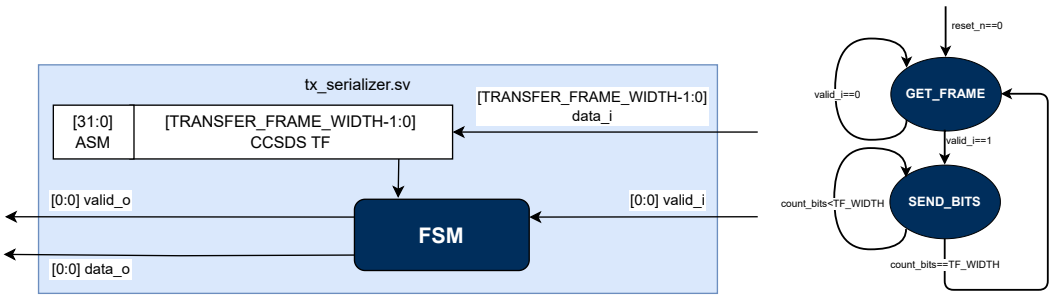


Figure 4.5.: Flowchart of the FSM logic used for accepting the CCSDS TF.

4.1.2. Serializer - `tx_serializer.sv`

Once the full CCSDS TF is available inside the registers of the FPGA the data can be processed according to the CCSDS Recommended Standards. The full CCSDS TF is completely sampled inside the FPGA to ensure the reliable transmission of a full CADU, independent of faults in the SPI communication. Moreover, a Cyclic Redundancy Check (CRC) could be performed on the whole CCSDS TF after receiving it in the FPGA with this approach. However, for reducing latency a FIFO could be implemented with the downside of increased risk of transmission of invalid CCSDS TFs. The size of the CCSDS TF is set by the parameter `TRANSFER_FRAME_WIDTH` and is fixed at compile time. This is because this parameter is fixed for a specific mission. For testing purposes it is set to 16 bytes initially, however, this can be up to 1023 bytes. This is a parameter that can later be determined by other members of SAGE depending on their requirements. The serializer module is instantiated inside the `ccsds_tx_ip_v1_0.sv`. Once the CCSDS TX Chain IP is finished reading the data that was communicated over SPI, the `enable_serializer_o` wire is asserted for a single clock cycle and the data is available on the `frame_o` wire. At this moment the serializer reads in the frame and will start sending out the data bit-by-bit over multiple clock cycles, set by the `cycles_per_bit` line. The number of bits for this signal line was initially set to 32 bits but could later be adjusted to 16 bits. The latter 16-bit for a 64 MHz clock frequency, results in a maximum coded data rate of 64 Mbps, and a minimum coded data rate of around 1 kbps, which should be sufficiently low. See Fig. 4.6(a) for the flowchart of this module, and Fig. 4.6(b) for the FSM of this

module.



(a) Flowchart (b) FSM logic

Figure 4.6.: The implementation of the TX serializer module.

4.1.3. Forward Error Correction - `ccsds_convolutional_encoder.sv`

The module that has an important role in ensuring reliable communication in space is the FEC module. For the convolutional encoder, the input is a single bit, and the output is either two single-bit lines or a single output line with twice the data rate for the rate 1/2 encoder. To accommodate an easier implementation of the different modulation techniques, the convolutional encoder was implemented using a single output bit line, and a second output `cycles_per_bit_o`, indicating the cycles per bit that the subsequent modules have to use. The logic of the convolutional encoder itself is, however, the same as already seen in Fig. 2.3.

4.1.4. Scrambler - `ccsds_scrambler.sv`

The scrambler ensures sufficient bit transitions in the data transmitted over the physical channel to enable reliable synchronization at the receiver. The scrambler implementation follows the guidelines provided in [5], depicted in Fig. 4.7. This method utilizes a 255-bit random pseudo-random sequence, which is applied using an XOR operation on the code block, codeword, or CCSDS TF. For convolutional encoding, scrambling is applied before encoding, while for other FEC schemes used in both TM and TC, scrambling is applied after FEC. The CCSDS standards specify different polynomials for generating the random sequence for TM and TC as follows

$$h_{\mathrm{TM}}(x) = x^8 + x^7 + x^5 + x^3 + 1 \quad \text{and} \quad h_{\mathrm{TC}}(x) = x^8 + x^6 + x^4 + x^3 + x^2 + x + 1. \tag{4.1}$$

Changing the HDL scrambler module to accommodate both versions simply involves adjusting the parameters. Furthermore, [5] notes that the descrambling process at the receiver can be performed either through another bitwise XOR operation for 'hard bits'

or by inverting the 'soft bits'. It is noteworthy that this module has not been tested in the final system yet, however, it has been verified independently using its automated testbench.

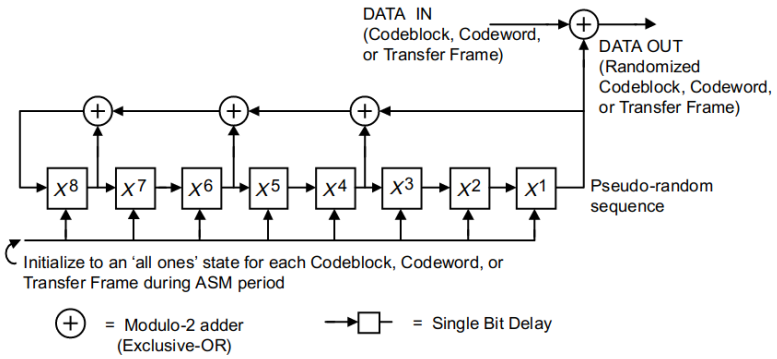


Figure 4.7.: Flowchart of the CCSDS scrambler module, figure adapted from [5].

4.1.5. Modulation - `ccsds_modulator.sv`

An overview of the implementation of the modulator module can be seen in Fig. 4.8. The input is the stream of data bits together with a valid signal. Moreover, a `cycles_per_bit` signal is present to ensure correct buffering of the input bits. The third input is the control signal, which can be utilized to select a certain modulation technique. One of the control bits is used for the delay unit needed for Offset Quadrature Phase Shift Keying (OQPSK) modulation, which delays the Q signal by half the symbol time. The outputs are 2 13-bit IQ signals, together with an accompanying valid signal. For the implemented BPSK system, the values are scaled between the maximum, 4095, and the minimum, -4095.

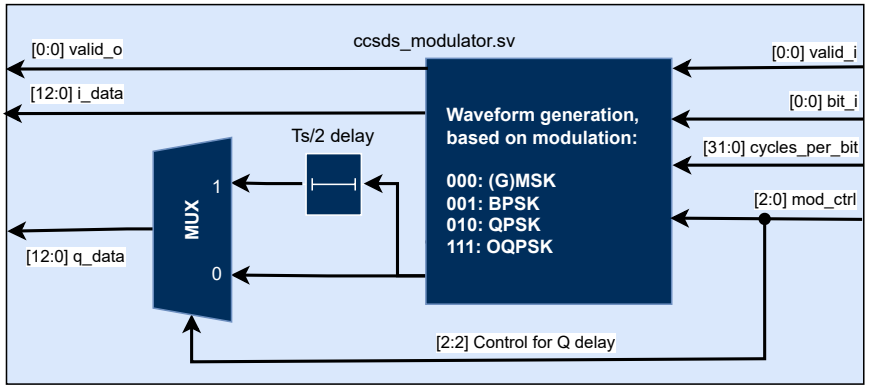


Figure 4.8.: Flowchart of the modulator module.

4.2. MCU Software

Besides the FPGA, also the MCU needs to be programmed. This is done in the C programming language using the STM32CubeIDE [26]. The SatNOGS COMMS board will be controlled by the discussed STM32H723ZG with Zephyr RTOS [42] to schedule all the tasks for controlling the board and processing the data. For the SPI communication, it has been chosen to make the MCU the SPI master, since the MCU also needs to communicate with different devices over SPI. The first implemented system on the MCU is a terminal interface to the MCU over UART, with which two-way communication can be established using the SPI connection between the MCU and FPGA. Firstly, the SPI communication was set using the STM32 configuration tool. Here, the SPI frame format was set to Motorola, the frame size to 8 bits with MSB first, the data rate was set to 4 Mbpps, the SPI clock polarity to low and the clock phase to 1 edge, all corresponding to the settings in the SPI IP. For this code, the Hardware Abstraction Layer (HAL) functions of STM32 have been used, specifically the `HAL_SPI_TransmitReceive`. In the code, firstly characters are read over the terminal with UART communication between the PC and the STM32 development board by using the `scanf` function. These characters are then converted to an array of `uint8` numbers which can be passed into `HAL_SPI_TransmitReceive`. When calling this latter function, another variable can be passed in which the received SPI data is stored. This received data is also printed in the terminal. Moreover, in this initial implementation, the ASM was automatically added to the CADU that was sent to the FPGA. This could be used by the FPGA to distinguish new CADUs and to avoid errors in the SPI communication.

With the implementation described in detail, the test setups used to validate the system are presented and the results are shown in the following section.

Chapter 5

Results

The following section provides an overview of the various experimental setups employed for validating the system on a functional level. The tests defined here are tests to show the functionality of the system but are by no means the final tests for a system going to space. For this, post-implementation simulations have to be done, incorporating the specific placement, and temperature and voltage variations. It is worth mentioning that all the necessary code to reproduce these experiments can be accessed on the designated GitLab repository [43]. An explanation of the structure of this repository can be found in Appendix C.

5.1. Testbench setup

The initial phase in testing all the written HDL modules involves creating test benches. Each HDL module that is developed has accompanying testbench files. These testbench files generate clock and reset signals and simulate different scenarios in which the circuit can operate. Consequently, tests have been developed for each module, encompassing typical use cases as well as less common but possible situations, known as edge cases.

5.1.1. CCSDS TX Chain IP

Methods

The first module to test, is the overall control module that accepts the SPI frames and puts them together in a single CADU. In this test 8-bit SPI frames are used, and a small 16 bit CADU is used. This can easily be changed to the bigger vales of 1023 bytes, but for the testbench it is easier if it consists of less bits. However, also tests with larger lengths

have been performed and were shown to pass as well. Since this module interfaces over AXI4-Lite as a master, the testbench will generate responses as a AXI4-Lite slave. The goal of this testbench will be to guide the module through all possible states by giving specific responses to read requests corresponding with the expected values of the SPI IP. This means reading and writing to specific offsets with specific values to enable or disable certain functions. The exact values can be seen in the testbench waveform output.

Results

The output of the testbench can be seen in Fig. 5.1. It is seen that the test follows contains all the states shown in the flowchart shown in Fig. 4.5, shown by the `current_state` in the waveform. Multiple AXI4-Lite transactions can be seen, by checking the `m00_axi*` signals. Lastly, it is seen that the `serializer_done_i` signal is asserted after two successful data receives, indicating the start of the serializer. This is accompanied by the `frame` register signal that shows the data that was indeed simulated to be in the SPIDDR. In the test it performs the following steps:

- 0-80 ns: Perform AXI4-Lite write requests to SPICR(0x44a00060), IPIER (0x44a00028), and DGIER (0x44a0001c), and finally the SPIDTR (0x44a00068).
- 80-100 ns: Here, two reads of the SPISR (0x44a00064) are performed, but returned with a value where the LSB is non-zero, meaning no new SPI data is available.
- 100-155 ns: Now, the SPISR returns a zero LSB, meaning new data arrived over SPI. Consequently, the SPIDDR (0x44a0006c) is read out with values 0x4f. This is stored in the `frame[15:0]` variable. After this, the FSM goes back to `WAIT_FOR_SPI` and performs another cycle of reading out SPISR and SPIDDR till the frame is full, which is for this case already after two readouts.
- 155-180 ns: The serializer is enabled and the bits can be send through the chain. Here, parallelization is not happening yet.
- 180-205 ns: The FSM writes the SPICR to reset the SPI TX and RX FIFOs.
- 205-215ns: The FSM writes a certain confirmation of a succesfull transfer to SPIDTR (0x44a00068).
- $>$215 ns: The FSM went back into `WAIT_FOR_SPI` to start another transmission cycle.

In this module the latency is quite substantial since first the whole CADU is read in, before processing happens. This makes the latency a function of the length of the CADU. Moreover, the latency depend on the rate at which the AXI4-Lite transactions happen.

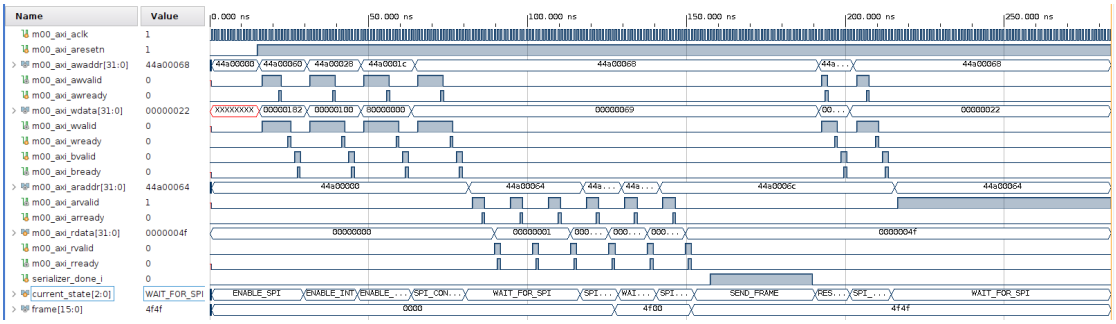


Figure 5.1.: Vivado simulation waveform of the CCSDS TX Chain module. Only the serializer module has been connected outputting a short 16-bit sequence, corresponding to the 2 SPI transactions.

5.1.2. Serializer

Methods

The testbench of the serializer module comprises a test where the `valid_i` signal is asserted, together with a known register value on the `frame_i`. The register is in this testbench set to a small value of only 5 bytes. The output is then automatically compared against the expected output data, and it is checked whether the number of cycles per bit is correct. The number of cycles per bit that are tested are 1 and 3.

Results

The output of the testbench can be seen in Fig. 5.2. It is here verified that the system can correctly send the data stored in the register bit-by-bit for both a single clock cycle per bit and also for 3 cycles per bit. Longer values have been checked manually and also pass. Moreover, this shows the module is capable of signaling whether the output currently contains data by asserting and deasserting the `valid_o` output. Since the serializer sends out the data stream one clock cycle after receiving the frame, the latency of this module is one clock cycle.

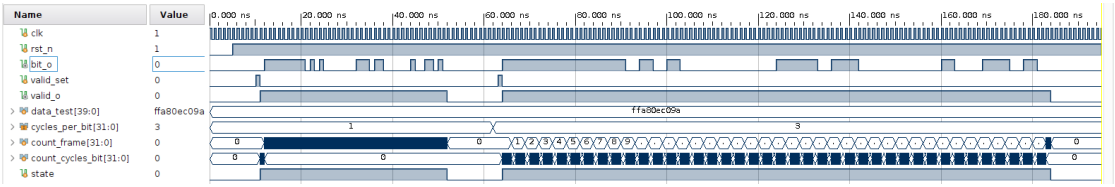


Figure 5.2.: Vivado simulation waveform of the testbench for the serializer module.

5.1.3. Convolutional Encoder

Methods

The testbench of the convolutional encoder comprises a test in which a certain data sequence is fed into the encoder, of which the output is known. The output bits are sampled at every clock cycle and automatically checked against the expected output. Moreover, since the convolutional encoder should be able to adapt the error correction to the final data rate, the output bits should not change for `cycles_per_bit` amount of cycles. Therefore, also different values of this parameter are tested, and it is checked whether the output is stable during these cycles. In the testbench, two separate tests are defined. The first one is a short output, which is calculated by hand using the implementation shown in [5]. This yields the binary sequences:

- Input: 1110111
- Output: 10001111101001

However, to test longer sequences this method is not feasible. Therefore, the MATLAB function `convenc` has been used to generate a much longer test output as 'Golden Model'. An arbitrarily chosen 128-bit input sequence has been convolutionally encoded, and afterwards, every second bit has been inverted, as per [5]. This yields the following input and output pair:

- Input: 3FA598708734FE00BA349826400EDFE6
- Output: 58C19230026A0FDF0A7FD293FB19CF95B7E505E3CE1A37919852558FA82B69FA

The MATLAB code used to generate this can also be found in the repository [43].

Results

The waveform output can be seen in Fig. 5.3. Here three test cases can be seen. First, a 7-bit input with 4 cycles per bit, resulting in a 14-bit output sequence with 2 cycles per bit. Secondly, a 7-bit input sequence with 6 cycles per bit is seen, resulting in a 14-bit sequence of 3 cycles per bit. Finally, a 128-bit input sequence with 4 cycles per input bit is seen, resulting in 256 output bits of each 2 cycles per output. In the test bench, the output signal is sampled and compared against the values calculated in MATLAB. These automated test cases all pass successfully, thus showing that the output is a correct error-corrected signal and that the system is capable of functioning at different data rates. Since the data rate is changed by the encoder, a normal latency value can not be given. However, the output is changed based on the input value one cycle after the input changes, effectively yielding a one clock cycle latency.

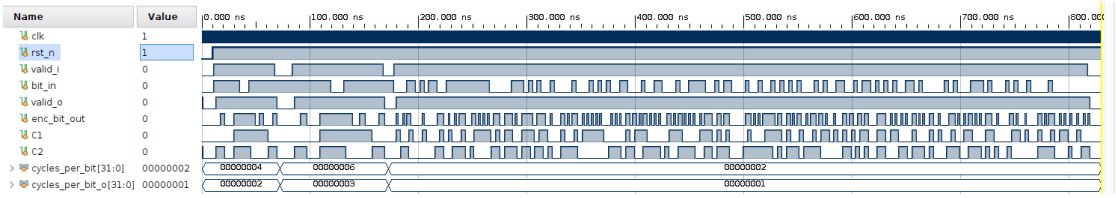


Figure 5.3.: Convolutional encoder testbench output for 3 test cases. First a 7-bit input with 2 cycles per output bit, second a 7-bit input with 3 cycles per output bit, and finally a 128-bit input with 2 cycles per output bit.

5.1.4. Modulator

Methods

The modulator module has been tested using two test cases with differing `cycles_per_bit` values. This is done to ensure the module is capable of operating with different values, and also to ensure the module behaves as expected after deasserting the `valid_o` signal. This output is automatically checked in the testbench, but also a visual inspection was done.

Results

The output of the testbench can be seen in Fig. 5.4. It is seen that the modulated signal `i_data_o` follows the input `bit_i`, however, in this case, the signal ranges between 4095 and -4095. These are the expected maximum and minimum values of the 13-bit 2's complement modulated output. Moreover, the `valid_o` signal is asserted and deasserted as expected based on the presence of a valid output signal. As seen in the testbench output, the latency of this module is equivalent to the value of `cycles_per_bit`. Noted is that this can be optimized by reducing it to one clock cycle for every value of `cycles_per_bit`, at the expense of a slightly more complex implementation. This can be changed if latency requirements require this.

5.2. FPGA and MCU Development Boards Setup

5.2.1. Method

The second method to test the running system is using the development boards of the FPGA and MCU. For this, the PYNQ-Z2 and Nucleo144 development boards have been used. An initial test consists of a test set in which a certain string of 40 bits saved on the FPGA can be sent, this can be seen in Fig. 5.5. The output has been manually verified using an oscilloscope connected to the outputs of the two FPGA pins to match

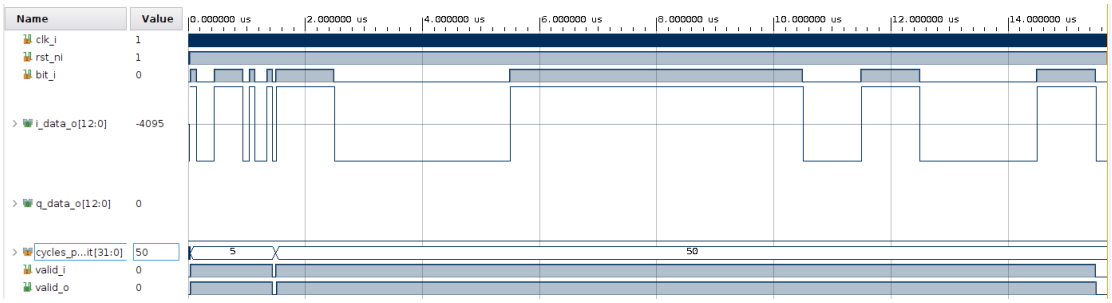


Figure 5.4.: Modulator testbench output for two test cases. First, a sequence with a `cycles_per_bit` of 5 is modulated, and secondly, a sequence with `cycles_per_bit` of 50 is modulated.

the expected output with this older version of the convolutional encoder with the valid signals and implementation of different data rates. This initial test was done at a clock frequency of 64 MHz.

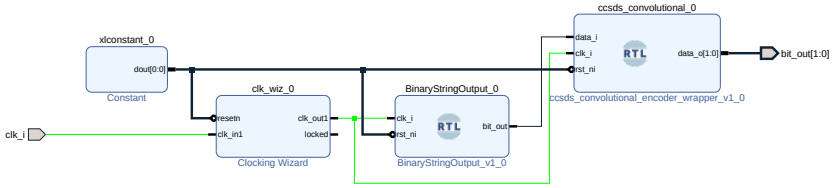
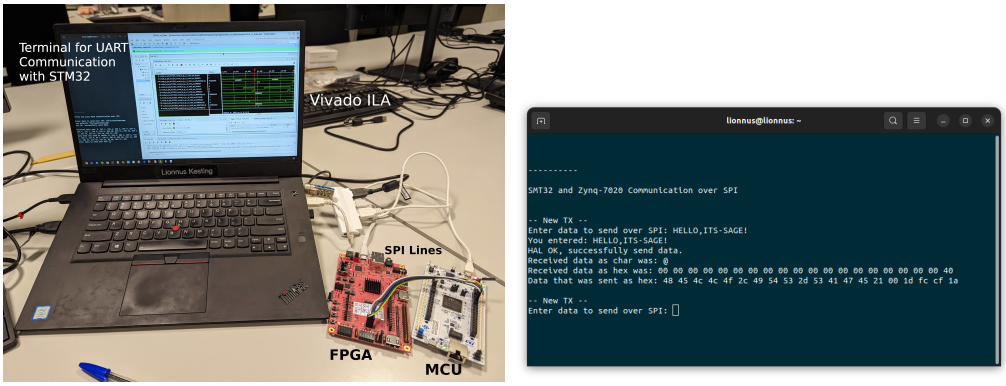


Figure 5.5.: Block design for the test setup to test the output of a convolutionally encoded random bit sequence.

A second test is defined where data can be received and sent using a UART connection between the STM32 and a terminal on the PC using the `screen` command on Linux. A picture of this test setup can be seen in Fig. 5.6(a), and the terminal interface is seen in Fig. 5.6(b).

This test setup is aimed toward testing the full TX chain, including the SPI communication and the chain of HDL modules. To validate the system, via UART the STM32 can communicate in the terminal, and this can be used to verify SPI frames send by the FPGA. Secondly, the error-corrected signal can be output on one of the pins of the FPGA. This output can be sampled by an oscilloscope and compared to the MATLAB CCSDS Simulation setup described in [39]. Next, this can be utilized to investigate whether this waveform would satisfy the requirements for the CCSDS protocol. The block design utilized for this test can be seen in Fig. 5.7. Since the output of the modulator are two 13-bit signals, it was not feasible to output this on FPGA pins and sample this with an oscilloscope. Therefore, the output of the convolutional encoder is sampled, and the output of the modulator is only verified in the Vivado Debugger with the System ILA. This therefore still verifies the physical signals on the FPGA. The oscilloscope was set



(a) Test setup (b) Terminal interface

Figure 5.6.: Test setup for the initial test with a terminal interface with the MCU to send and receive commands from and to the FPGA over SPI.

up to trigger at an edge of the `valid_o` signal of the convolutional encoder module. The `data` signal was then sampled and is compared to the MATLAB simulation using the same FEC, transfer frame size, utilization of the ASM, and using the same modulation. The block design was implemented with a 100 MHz clock, and a SPI clock of 32 MHz, but prescaled by 4 to yield a 4 Mbps SPI transaction. These are the same settings as set in the STM32 configuration.

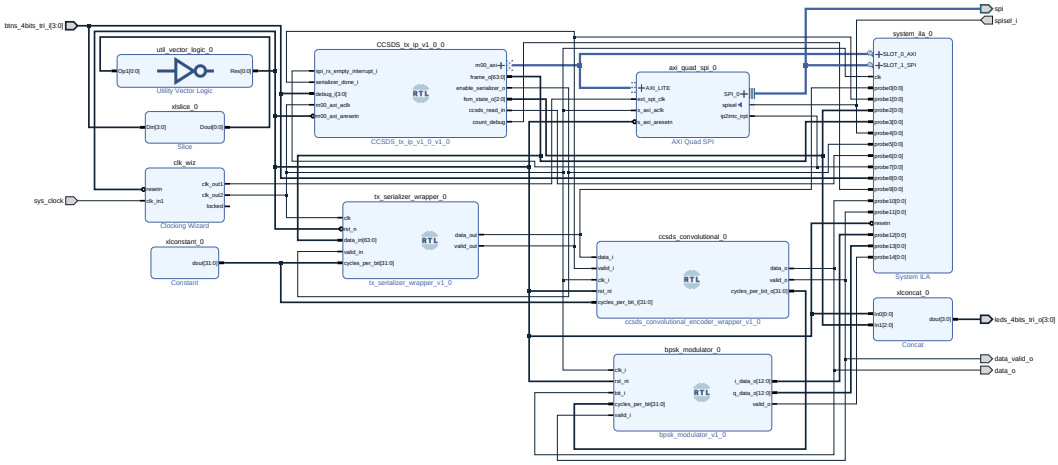


Figure 5.7.: Block design for the test setup to test SPI communication between the MCU and FPGA, as well as testing the serializer module.

5.2.2. Results

In Fig. 5.8 the output of the Vivado debugger window can be seen and comprised of the following:

- 0-4800 ns: A lot of AXI4-Lite transactions are happening, all subsequently reading out SPICR to check for new SPI data, and then SPIDDR to read out the actual data and store it in the `CCSDS_tx_ip_v1_0_0_frame_o` register.
- >4800 ns: The signal is transfered to the serializer module in a single clock cycle, and the output is subsequently error corrected and moduled. THe modulated signals are `bpsk_modulator_0_i_data_o` and `bpsk_modulator_0_q_data_o`. Since BPSK modulation is chosen, only the I value changes between -4095 and 4095, and the Q values stays zero and is therfore not shown in the output. Moreover, an accompanying valid signal is shown for the modulated signal, `bpsk_modulator_0_i_valid_o`.

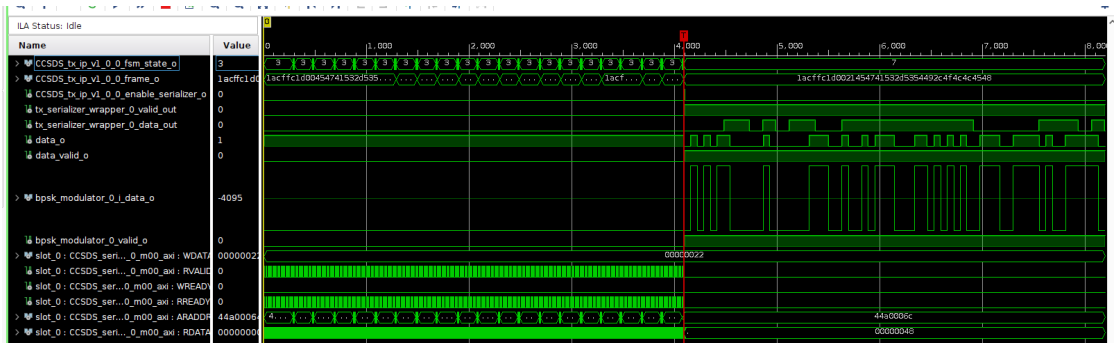


Figure 5.8.: Output waveform in the Vivado Debugger for the test setup shown in Fig. 5.6. Shown are the serializer output, convolutional encoder output, and the BPSK modulated signal.

The output of the convolutional encoder is connected to one of the FPGA output pins, and subsequently sampled with an oscilloscope. This output is shown in Fig. 5.9. Here, additionally the MATLAB simulation waveform is show and can be compared with the output pin of the FPGA. Noted should be that output of the FPGA pins is the output of the convolutional encoder instead of the modulated values needed for the transceiver IC, so the signal varies between a logical high (3.3 V) and a logical low (0 V). However, the internal modulated signal seen in the Vivado Debugger has been confirmed to match the BPSK modulated signal, as already seen in Fig. 5.8. The MATLAB simulation and this test where performed at a data rate of approximately 1.6 Mbps, resulting from a clock frequency of 100 MHz and 64 `cycles_per_bit`.

The resource utilization can be seen in Tab. 5.1. In the bottom rows, the total utilization is shown in both absolute numbers, and as a percentage of the available resources on this specific FPGA. It is seen that the total utilization is only a small percentage of the available resources on the FPGA. However, it should be noted that this is only for one of

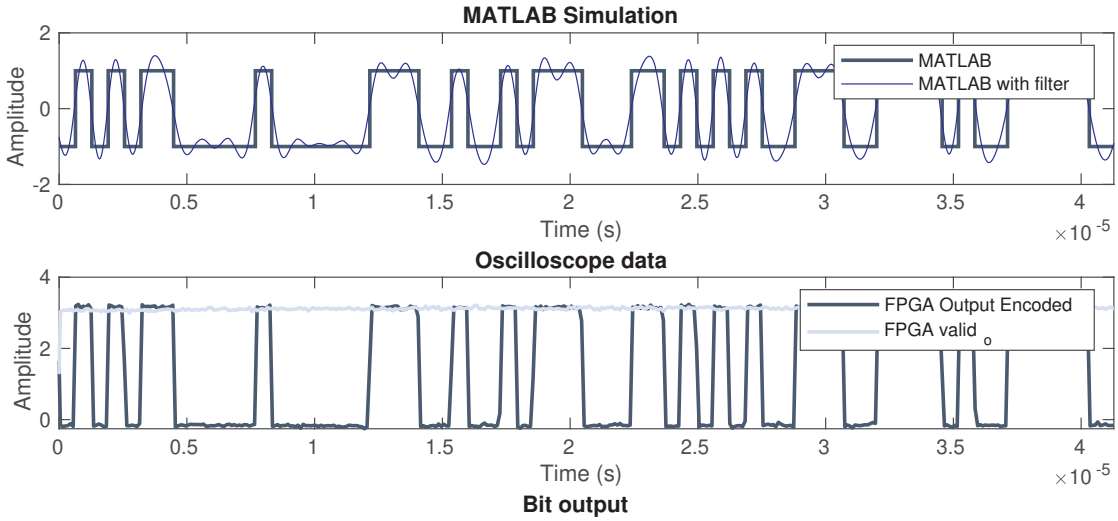


Figure 5.9.: Measured output waveform for the test setup shown in Fig. 5.6. Shown are the simulated CCSDS-compliant waveform generated in MATLAB, the measured output of the convolutional encoder, and the expected output bits generated with MATLAB.

the two TX chains (one chain for UHF and a separate one for S-Band). Additionally, the RX chain needs to be implemented, which is anticipated to be more resource-heavy.

Table 5.1.: Power and resource utilization

Component	**Power (W)**	**#/LUTs**	**#/Registers**	**#/Slices**
Clock wizard	0.124	0	0	0
AXI QUAD SPI IP	0.003	381	582	181
Control module	0.001	131	210	75
Serializer	0.001	78	203	58
Convolutional encoder	<0.001	68	40	27
BPSK Modulator	<0.001	52	35	27
Total	0.128	710 (1.33%)	1,070 (1.01%)	368 (2.77%)
Available	-	53,200	106,400	13,300

Chapter 6

Discussion

During the course of this thesis, a functional tranmission chain has been implemented on FPGA. This includes the transmisison of abritrary data over SPI from a MCU to the FPGA, storing this data in a transfer frame, and error correcting these bits. These bits can then be send at a variable data rate and can be modulated yielding two 13-bit I and Q data signals as output. Even though this is an elaborate initial setup, some things still need to be added to yield a fully functional transmission chain that can be implemented on the SatNOGS COMMS hardware.

The first thing that needs to be finished is the AXI4-Stream interface at the output of the `CCSDS_TX_IP`. A first implementation can already be found in the repository [43], but tests still need to validate this module. The AXI4-Stream enables the connection to the Xilinx FIR Compiler to filter the square pulse output of the modulator with a raised-cosine filter, as also seen in the MATLAB simulation. After this, the output of the system also needs to be an AXI4-Stream to interface with the AT86IP. A second critical addition is the implementation of the RX chain. This can be implemented by keeping the proposed system overview shown in Fig. 4.1 in mind.

Moreover, some performance improvements can be made. Firstly, a lot of linear shift registers are used in this project, an improvement to this would be the use of a parallel implementation of the convolutional encoder and scrambler [44]. However, these improvements are not strictly necessary since the processing speed for the TX chain is limited by the physical RF channel, and not by the computational speed of the FPGA.

Another performance improvement is pipelining of the SPI communication module. At the moment a single CADU is fetched and sent bit-by-bit through the chain. However, a new CADU could already be fetched in the meantime such that there is a continuous output stream.

Moreover, the SPI communication is currently not protected against any errors. A possible solution for this would be using a CRC for either every SPI frame or for a whole CADU [45].

As discussed before, to utilize this design on a satellite way more extensive tests have to be performed. These include post-implementation simulations, simulation of different voltages and simulations at different temperatures. Moreover, a more extensive timing analysis needs to be performed.

Chapter 7

Conclusion and Future Work

The present thesis has made a substantial contribution toward the successful development and implementation of a transmission chain on a Field Programmable Gate Array (FPGA). It has successfully achieved the transfer of arbitrary data over SPI from a MCU to an FPGA. The project further involved storing this data in a transfer frame and conducting forward error correction on these bits. The bits could then be transmitted at a variable data rate and modulated to produce two 13-bit I and Q data signals as output using BPSK modulation.

However, to construct a fully functional transmission chain that can be implemented on the SatNOGS COMMS hardware, further development is needed. This includes the adaptation of the AXI4-Stream interface at the output of the modulation module and the implementation of the RX chain. Performance improvements such as using parallel implementation of the convolutional encoder and scrambler, pipelining of the SPI communication module, and error protection for the SPI communication also require attention.

In conclusion, this project has paved the way for the advancement of the SAGE CubeSat mission, contributing to the success of satellite communication through the development of effective transmission chains. It has laid the groundwork for future research and development of the FGPA implementation for the SatNOGS COMMS board, gearing towards the successful control of the CubeSat and transmission of research data from space utilizing a worldwide network of SatNOGS ground stations.

Appendix A

Task Description

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CENTER FOR PROJECT-BASED LEARNING

Task Description for a Semester Project on

Implementation of FEC on FPGA for a dual-band satellite transceiver

at the Department of Information Technology and Electrical Engineering

for

Kesting Lionnus
lkesting@student.ethz.ch

Advisors:	Nicolas Schärer, nicolas.schaerer@pbl.ee.ethz.ch Dr. Christian Vogt, christian.vogt@pbl.ee.ethz.ch
Professor:	Prof. Dr. Luca Benini, lbenini@iis.ee.ethz.ch
Handout Date:	14.03.2023
Due Date:	23.06.2023

1 Project Goals

The SAGE CubeSat team develops a 3U CubeSat to demonstrate a microgravity platform in orbit. For that a reliable communication link is required in order to send commands to the spacecraft and receive the payload data from the biological payload. The hardware for the dual-band transceiver is based on an open source project[1] from librespace and will be provided by ARIS (see Figure 1). The challenge is to implement a protocol that complies with the CCSDS[2] standard and implements fast FEC (Forward Error Correction) on the Xilinx FPGA.

The goal of this project will be to implement FEC on the FPGA and write software on the microcontroller to have a reliable protocol for receiving commands and sending telemetry data. This shall be tested and validated using a SDR and a simple GNU Radio flowgraph. The hardware for that (PCB with FPGA and microcontroller) already exists and will be provided to the student by ARIS. There also exists some code examples[3] and drivers from librespace on how to communicate with the FPGA and RF transceiver.

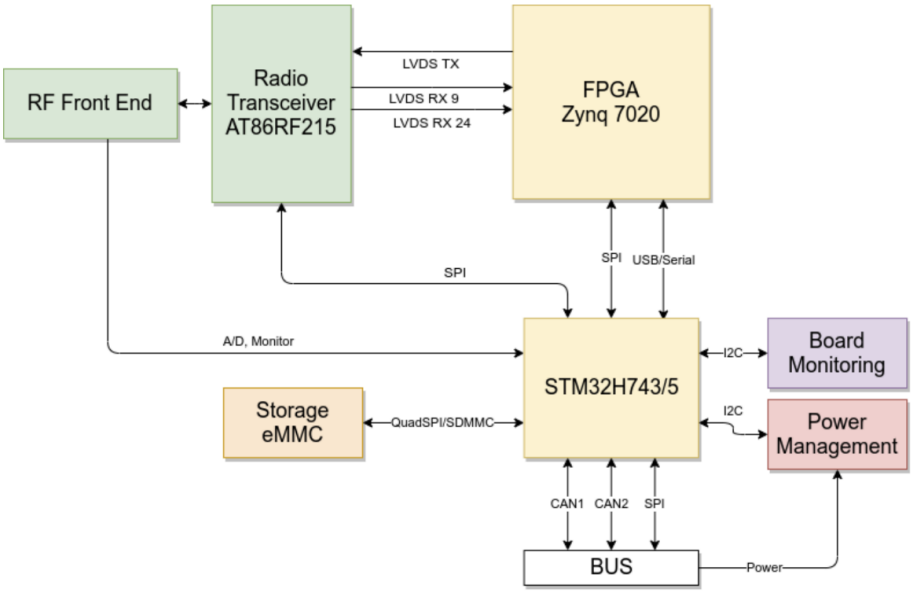


Figure 1: System architecture of the SatNOGS COMMS module

[1] https://libre.space/projects/satnogs-comms/
[2] https://public.ccsds.org/default.aspx
[3] https://gitlab.com/librespacefoundation/satnogs-comms

2 Tasks

The project will be split into three phases, as described below:

Phase 1 (Week 1-4)

1. Investigate the state-of-the-art of CCSDS protocols and used FEC codes
2. Study and get used to the hardware and the tools to program, i.e. microcontroller and FPGA programming, communication, debugging.
3. Read into the existing material from the CubeSat mission and documentation from librespace.
4. Identify and choose possible communication protocols and FEC codes
5. Preliminary setup for testing and debugging the FPGA, STM32 microcontroller and the system (Vivado, STM Cube IDE, GNU radio)
6. Defining key metrics to evaluate the system on (KPI)

Phase 2 (weeks 5-10)

1. Implementing at least 2 FEC codes on the Xilinx ZYNQ-7020 FPGA and compare them in terms of KPI, latency, resources on FPGA, max specs, power consumption
2. Implementing the needed software for the protocol on the STM32H743 ARM Cortex-M7 Microcontroller (for later communication over CAN bus with the On Board Computer (OBC))
3. Test, evaluation, and characterization of the throughput and bit error rate (BER) on a simulated gaussian channel (see Figure 2 for Reference)
4. Optimization (if any) for a reliable transmission (defined as BER $< 10^{-9}$)

Phase 3 (week 11-14)

1. Finalizing code for FPGA and Microcontroller
2. Testing and evaluating performance of proposed system
3. Documentation: Report & presentation

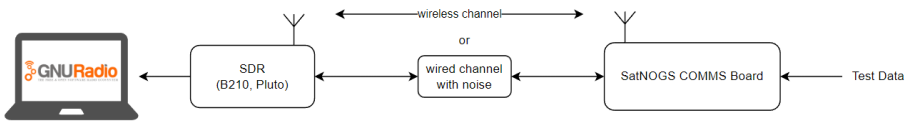


Figure 2: Test Setup with stored test data on the STM32

Milestones

The following milestones need to be reached during the thesis:

- Proposing communication protocols and their KPIs, following the CCSDS and state of the art FEC codes
- Software for FPGA and Microcontroller implementing proposed protocols
- Evaluation and comparison of performance with throughput, BER, power consumption and resources on FPGA
- Final design and test
- Final report and presentation

3 Project Organization

During the thesis, students will gain experience in the independent solution of a technical-scientific problem by applying the acquired specialist and social skills. The grade is based on the following: Student effort; thoroughness and learning curve; achieving qualitative and quantitative results with a scientific approach; supporting practical findings with theoretical background and literature investigations; final presentation and report; documentation and reproducibility. All theses include an oral presentation, a written report and are graded. The report and presentation need to have publication grade quality to achieve a good grade. Students are graded based on the official ITET grading form[4]. For students of IIS (Prof. Benini) a special grading scheme exists, please contact your supervisor for details there. Before starting, the project must be registered in myStudies and all required documents need to be handed in for archiving by PBL.

3.1 Laboratory Rules

The students agree to follow the lab rules set by PBL staff, for detail please contact us. The most important points are:

- All ETH safety regulations need to be followed[5], in addition to ones given by PBL staff
- No device in the lab is used without introduction by your supervisor or PBL staff
- No device leaves the lab without being offically borrowed, this is done by PBL staff and needs your Legi.
- Any damage to devices or tools needs to be reported immediately to PBL staff.
- The Lab-desk is clean and free for others after you finished your task, or when you take longer breaks. All tools are correctly sorted into their drawers/cupboards when you leave.

3.2 Weekly Report

There will be a weekly report/meeting held between the student and the assistants. The exact time and location of these meetings will be determined within the first week of the project in order to fit the students and the assistants schedule. These meetings will be used to evaluate the status and document the progress of the project (required to be done by the student). Beside these regular meetings, additional

[4] https://ethz.ch/content/dam/ethz/special-interest/itet/department/Studies/Forms/Grading%20Form.xlsx

[5] https://ethz.ch/staffnet/en/service/safety-security-health-environment/sicherheit-in-laboren-und-werkstaetten/laborsicherheit.html

meetings can be organized to address urgent issues as well. The weekly report, along with all other relevant documents (source code, datasheets, papers, etc), should be uploaded to a clouding service, such as Polybox and shared with the assistants.

3.3 Project Plan

Within the first month of the project, you will be asked to prepare a project plan. This plan should identify the tasks to be performed during the project and sets deadlines for those tasks. The prepared plan will be a topic of discussion of the first week's meeting between you and your advisers. Note that the project plan should be updated constantly depending on the project's status.

3.4 Final Report and paper

PDF copies of the final report written in English are to be turned in. Basic references will be provided by the supervisors by mail and at the meetings during the whole project, but the students are expected to add a considerable amount of their own literature research to the project ("state of the art").

3.5 Final Presentation

There will be a presentation (15 min presentation and 5 min Q&A for BT/ST and 20 min presentation and 10 min Q&A for MT) at the end of this project in order to present your results to a wider audience. The exact date will be determined towards the end of the work.

References:

Will be provided by the supervisors by mail and at the meetings during the whole project.

Place and Date _____ Signature Student _____

Appendix B

Declaration of Originality

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

IMPLEMENTATION OF FEC ON FPGA FOR A DUAL-BAND SATELLITE TRANSCEIVER

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
KESTING	LIONNUS

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 15.03.2023	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Appendix C

GIT Repository Structure

In the git repository, all the (System) Verilog modules will be structured in the following way:

```
ccsds_sample/
├── sourcecode/
│   ├── ccsds_sample.sv
│   └── tb/
│       └── ccsds_sample_tb.sv
└── vivado/
    ├── ccsds_sample.xpr (Not guaranteed)
    ├── constraints/
    │   └── pynq_z2.xdc
    └── scripts/
        └── *.tcl
```

To reliably create all the Vivado project files, Vivado version 2020.2 has to be used. As seen in the project structure in the previous section, certain Tool Command Language (tcl) scripts are provided to recreate the project. To do this please do the following:

1. Open Vivado 2020.2
2. In the tcl command window change directory to the ‘vivado‘ directory, e.g. ‘cd /satnogs-fpga/ccsds_sample/vivado‘.
3. From here, run the script to setup the vivado project, e.g. ‘source scripts/init_project.tcl‘

> Note: The ‘.xpr‘ file and/or other Vivado project files may or may not already be included. However, the Vivado project can always be generated by using the procedure described above.

Glossary

ARIS The Akademische Raumfahrt Initiative Schweiz is a student-led organization in Switzerland working on various projects connected with space research, such as, for example, the CubeSat (SAGE)..

LSF The Libre Space Foundation is an organization creating open-source space technology. This includes a network of ground stations, rockets, and satellites. Moreover, they developed the SatNOGS COMMS hardware used for this project..

Bibliography

[1] A. Poghosyan and A. Golkar, "CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions," *Progress in Aerospace Sciences*, vol. 88, pp. 59–83, Jan. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0376042116300951

[2] "Librespacefoundation Webpage." [Online]. Available: https://libre.space/

[3] CCSDS, "Radio Frequency and Modulation Systems—Part 1: Earth Stations and Spacecraft," 2021. [Online]. Available: https://public.ccsds.org/Pubs/401x0b32.pdf

[4] ——, "TM Space Data Link Protocol," 2021. [Online]. Available: https://public.ccsds.org/Pubs/132x0b3.pdf

[5] ——, "TM Synchronization and Channel Coding," 2022. [Online]. Available: https://public.ccsds.org/Pubs/131x0b4.pdf

[6] ——, "TC Space Data Link Protocol," 2021. [Online]. Available: https://public.ccsds.org/Pubs/232x0b4.pdf

[7] ——, "TC Synchronization and Channel Coding," 2021. [Online]. Available: https://public.ccsds.org/Pubs/231x0b4e0.pdf

[8] L. W. Couch, *Digital and analog communication systems*, 8th ed. Boston, Mass. München: Pearson, 2013.

[9] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC '93 - IEEE International Conference on Communications*, vol. 2, May 1993, pp. 1064–1070 vol.2.

[10] Y. Liu, Y. Guan, J. Zhang, G. Wang, and Y. Zhang, "Reed-Solomon Codes for Satellite Communications," in *2009 IITA International Conference on Control, Automation and Systems Engineering (case 2009)*, Jul. 2009, pp. 246–249.

[11] H. Saidi, M. Turki, Z. Marrakchi, A. Obeid, and M. Abid, "Implementation of Reed Solomon Encoder on Low-Latency Embedded FPGA in Flexible SoC based on ARM Processor," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, Jun. 2020, pp. 1347–1352.

[12] D. Theodoropoulos, N. Kranitis, and A. Paschalis, "An efficient LDPC encoder architecture for space applications," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Jul. 2016, pp. 149–154.

[13] S. Arunkumar and T. Kalaivani, "FPGA implementation of CCSDS BCH (63, 56) for satellite communication," in *2012 IEEE International Conference on Electronics Design, Systems and Applications (ICEDSA)*, Nov. 2012, pp. 248–253.

[14] S. Dhaliwal, N. Singh, and G. Kaur, "Performance analysis of convolutional code over different code rates and constraint length in wireless communication," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Feb. 2017, pp. 464–468.

[15] M. A. Rawoof, U. Ch., D. N. Kumar, D. K. Basha, and N. Madhur, "Verilog based efficient convolution encoder and viterbi decoder," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 8, no. 1, p. 75, Feb. 2019. [Online]. Available: http://ijres.iaescore.com/index.php/IJRES/article/view/17887

[16] V. Lakkundi and M. Kasal, "COMPARATIVE ANALYSIS OF FORWARD ERROR CORRECTING CODES FOR AMSAT PHASE 3-E SHORT MESSAGE APPLICATION," *Citeseer*, 2005.

[17] G. M. Almeida, E. A. Bezerra, L. V. Cargnini, R. D. R. Fagundes, and D. G. Mesquita, "A Reed-Solomon Algorithm for FPGA Area Optimization in Space Applications," in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, Aug. 2007, pp. 243–249.

[18] R. Leoraj and J. A. V. Selvi, "Comparative performance analysis of forward error correcting codes for Free Space Optical communication," in *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*, Feb. 2016, pp. 1–6.

[19] M. A. Fleah and Q. F. Al-Doori, "Design and Implementation of Turbo encoder/ decoder using FPGA," in *2019 First International Conference of Computer and Applied Sciences (CAS)*, Dec. 2019, pp. 46–51.

[20] R. Wang, W. Chen, and C. Han, "Low-complexity encoder implementation for LDPC codes in CCSDS standard," *IEICE Electronics Express*, vol. 18, no. 9, pp. 20 210 128–20 210 128, May 2021. [Online]. Available: https: //www.jstage.jst.go.jp/article/elex/18/9/18_18.20210128/_article

[21] librespacefoundation, "SatNOGS System Design." [Online]. Available: https://gitlab.com/librespacefoundation/satnogs-comms/satnogs-comms-design-doc/-/jobs/artifacts/master/raw/build/system-design.pdf?job=docs

[22] Xilinx, "Zynq-7000 SoC Data Sheet: Overview (DS190)," 2018. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview

[23] Alinx, "AC7Z020 MoC Datasheet," Jun. 2023. [Online]. Available: https://alinx.com/public/upload/file/AC7Z020_UG.pdf

[24] Xilinx, "Vivado Design Suite - HLx Editions 2020.2." [Online]. Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

[25] STMicroelectronics, "STM32H7 Nucleo-144 User Manual," Jun. 2023. [Online]. Available: https://www.st.com/resource/en/user_manual/um2407-stm32h7-nucleo144-boards-mb1364-stmicroelectronics.pdf

[26] ——, "STM32CubeIDE." [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html

[27] Atmel, "Atmel AT86RF215 Datasheet." [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42415-WIRELESS-AT86RF215_Datasheet.pdf

[28] librespacefoundation, "LSF IP Core Repository." [Online]. Available: https://gitlab.com/librespacefoundation/fpga-cores

[29] Motorola, "SPI Protocol." [Online]. Available: https://www.nxp.com/files-static/microcontrollers/doc/ref_manual/S12SPIV4.pdf

[30] ARM, "AMBA 4 AXI Protocol Specification." [Online]. Available: https://developer.arm.com/documentation/ihi0022/latest/

[31] ——, "AMBA 4 AXI4-Stream Protocol Specification," 2010. [Online]. Available: https://documentation-service.arm.com/static/642583d7314e245d086bc8c9?token=

[32] "AXI4-Lite Interface." [Online]. Available: https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081

[33] librespacefoundation, "SatNOGS COMMS Gitlab Repository." [Online]. Available: https://gitlab.com/librespacefoundation/satnogs-comms

[34] U. o. P. LSF, "UPSat COMMS," Mar. 2016. [Online]. Available: https://upsat.gr/?page_id=22

[35] ——, "UPSat Open-Source Satellite," May 2017. [Online]. Available: https://upsat.gr/

[36] LSF, "SatNOGS COMMS Specifications." [Online]. Available: https://satnogs.org/wp-content/uploads/sites/2/2022/08/SatNOGS-COMMS-Product-flyer-1.pdf

[37] "CC1120 data sheet, product information and support | TI.com." [Online]. Available: https://www.ti.com/product/CC1120

[38] K. Kanavouras and E. Karakosta-Amarantidou, "LDPC code overview and testing ACUBESAT." [Online]. Available: https://helit.org/mm/docList/public/AcubeSAT-COM-ET-002#db-document

[39] MathWorks, "End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections," Mar. 2023. [Online]. Available: https://ch.mathworks.com/help/satcom/ug/end-to-end-ccsds-telecommand-simulation-with-rf-impairments-and-corrections.html

[40] Xilinx, "AXI Quad SPI v3.2 LogiCORE IP Product Guide." [Online]. Available: https://docs.xilinx.com/r/en-US/pg153-axi-quad-spi

[41] ——, "FIR Compiler LogiCORE IP Product Guide." [Online]. Available: https://docs.xilinx.com/r/en-US/pg149-fir-compiler

[42] "Zephyr RTOS," Jun. 2023. [Online]. Available: https://github.com/zephyrproject-rtos/zephyr

[43] L. Kesting, "SatNOGS FPGA Repository," Jun. 2023. [Online]. Available: https://git.ee.ethz.ch/pbl/fs2023/lionnus-kesting/satnogs-fpga

[44] L. Kekely, J. Cabal, and J. Kořenek, "Effective FPGA Architecture for General CRC," in *Architecture of Computing Systems – ARCS 2019*, ser. Lecture Notes in Computer Science, M. Schoeberl, C. Hochberger, S. Uhrig, J. Brehm, and T. Pionteck, Eds. Cham: Springer International Publishing, 2019, pp. 211–223.

[45] C. Borrelli, "IEEE 802.3 Cyclic Redundancy Check," 2001.