Tartu University

Faculty of Science and Technology

Institute of Technology

Melis Doğan

Integration of a High Speed Communications System into ESTCube-2

Master's thesis (30 EAP)
Robotics and Computer Engineering

Supervisors:

Kristo Allaje, MSc
Viljo Allik, MSc
Tõnis Eenmäe, MSc

Tartu 2024

Resümee/Abstract

Kiire sidesüsteemi integreerimine ESTCube-2 kuupsateliiti

ESTCube-2 oli Maa-lähedasel orbiidil töötav kuupsatelliidist demonstraator, mis startis 9. oktoobril 2023. Satelliidi üks peamisi teaduslikke laste oli Maa vaatluskaamerad, mis olid võimelised tootma mitme megabaidiseid pilte. Seetõttu vajas satelliit piltide edastamiseks spetsiaalset kiiret sidesüsteemi. Lõputöö alguses oli ESTCube'i meeskonna poolt selleks otstarbeks ostetud S-riba raadiosaatja.

See lõputöö esitleb IQ Spacecomi HISPICO saatja integreerimist ESTCube-2 platvormiga. See integratsioon hõlmas saatja juhtimiseks madaltaseme manussüsteemide draiverite loomist. Seadme draiverid kirjutati C-keeles, kasutades FreeRTOS-i, STM32L4 mikrokontrolleril. Lisaks loodi rakenduse tasemel loogika piltide vastuvõtmiseks kaamerast. Vastuvõetud pildid jagati RF-kaadriteks ja kaadritele lisati edasisuunas veaparandus (ingl. Forward Error Correction). See funktsioon on HISPICO-s olemas, kuid ESTCube-i meeskonnale ei antud piisavat ligipääsu signaalimuunduri dokumentatsioonile. Maajaama jaoks loodi LimeSDR Mini ja GNURadio Companion-i abil raadioside vastuvõtuahel. Sideahela testimiseks edastati 318 HISPICO kaadriga 76,8 kB suurune pilt ja see võeti edukalt vastu.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T320 Kosmosetehnoloogia

Märksõnad: kuupsatelliit, raadioside, S-riba

Integration of a High Speed Communications System into ESTCube-2

ESTCube-2 was a low Earth orbit CubeSat demonstrator that was launched on 9th of October, 2023. One of the main payloads of the satellite was the Earth observation payload, which was capable of producing images of tens of megabytes. Thus the spacecraft required a dedicated high speed communications system to downlink the images. At the beginning of the thesis, an S-band transmitter had been bought by the ESTCube team for this purpose.

This thesis presents the integration of the HISPICO transmitter from IQ Spacecom into the ESTCube-2 platform. This integration involved writing low-level embedded device drivers to control the transmitter. The device drivers were written in C, using FreeRTOS on an STM32L4 microcontroller. Furthermore, the application level logic was created for the reception of images from the imaging payload, dividing the received images into RF frames, and adding forward error correction to the frames - a functionality that exists on HISPICO, but the ESTCube team were not given access to. For the ground station, a reception pipeline using a LimeSDR Mini and GNURadio Companion was created. To test the chain of communication, an image of 76.8 kB was transmitted over 318 HISPICO frames and was successfully received.

CERCS: T120 Systems engineering, computer technology; T320 Space technology.

Keywords: CubeSat, radio communications, S-band

Contents

List of Figures

List of Tables

Acronyms

AWGN Additive White Gaussian Noise. A noise model used to simulate random noise that may occur during radio transmission. 40

BER Bit Error Rate. The rate of bit errors to the correctly received bits. 10, 40, 42

BPSK Binary Phase Shift Keying. A type of Phase Shift Keying (PSK) with only two phases, separated by 180°. 8, 10, 34, 37

CCSDS Consultative Committee for Space Data Systems. A multi-national committee creating standards for space data and communication systems. 13, 14, 31, 39, 40

COBS Consistent Overhead Byte Stuffing. A computationally inexpensive, highly predictable algorithm for packet framing with consistent overhead regardless of packet size. 29

COTS Commercial Off-The-Shelf. A ready-made product available for purchase and use. 14, 15, 21

CRC Cyclic Redundancy Check. A technique for error detection in data. 13, 25, 26, 38

DBPSK Differential Binary Phase Shift Keying. A type of Binary Phase Shift Keying (BPSK) with bits modulated by phase difference, as opposed to absolute phase. 34, 37

DQPSK Differential Quadrature Phase Shift Keying. A type of Quadrature Phase Shift Keying (QPSK) with bits modulated by phase difference, as opposed to absolute phase. 15, 34, 38

FEC Forward Error Correction. A technique for error detection and correction in transmitted data. 12–14, 18, 19, 25, 31, 32, 37–43

GFSK Gaussian Frequency Shift Keying. A type of Frequency Shift Keying with a Gaussian filter applied before modulation. 12, 14, 15

GMSK Gaussian Minimum Shift Keying. A type of Frequency Shift Keying with continuous phase, modulation index of 0.5i and a Gaussian filter applied before modulation. 12, 14

GS Ground Station. A radio station on Earth with capabilities for receiving/transmitting data from/to spacecrafts. 10, 11, 14, 16, 21, 26, 43

HAL Hardware Abstraction Layer. A software layer allowing convenient access and use of the hardware functionality. 22, 24, 26

HSCOM High-Speed Communications Subsystem. An additional communications subsystem that was added to ESTCube-2 for downlinking payload data. 10, 16, 20–24, 26–30, 32, 40, 42

ICP Internal Communication Protocol. A proprietary communication protocol for inter-subsystem communications, developed by the ESTCube team. 16, 28, 30, 40

MCU Microcontroller Unit. A small computing unit as a chip. 12, 15, 16, 20–24, 27, 29–31, 40, 42

OBC On-Board Computer. The central processing unit of a spacecraft. 12

PDU Protocol Data Unit. A special polymorphic data type specific to GNURadio consisting of a dictionary and a uniform vector. 37–39

PSK Phase Shift Keying. A modulation scheme where bits are encoded as the phases of the signal. 8–10

QPSK Quadrature Phase Shift Keying. A type of PSK with four phases separated by 90°. 8, 10, 13, 14, 34, 37

RF Radio Frequency. Electromagnetic waves with frequency in the 20 kHz to 300 GHz band. 10, 11, 34

SNR Signal-to-Noise Ratio. The ratio of received signal to the noise present in the channel, used for measuring signal quality. Measured in decibels (dB). 10, 40, 41

SPI Serial Peripheral Interface. A widely used protocol for synchronous communication over small distances, commonly used in embedded devices. 17, 18, 26–28, 42

UART Universal Asynchronous Receiver / Transmitter. A peripheral device that enables asynchronous serial communication. 17, 24, 25, 27, 28

UHF Ultra High Frequency. The RF band spanning from 300 MHz to 1 GHz. 10, 11, 15

VHF Very High Frequency. The RF band spanning from 30 MHz to 300 MHz. 10, 11

1 Introduction

CubeSats are a subset of nanosatellites, with a standard unit of ”U” determining their weight and size. One ”U” is defined as a 10 cm cube weighing up to 2 kilograms, with each CubeSat ranging between 1U to 12U in size. [1] The small size and standard dimensions of CubeSats enable quicker and cheaper development and more launch opportunities, important factors sought after by almost all student teams.

Generally, satellite communications use Radio Frequency (RF) within 30 MHz to 60 GHz to communicate with Earth or other satellites. Smaller satellites like CubeSats normally use the lower end of the frequency range, the Very High Frequency (VHF) and Ultra High Frequency (UHF) bands, which have been widely used and thus offer a higher technical maturity, and less attenuation due to obstacles and weather conditions. However, widespread usage also leads to crowding on these frequency ranges, as well as the lower bandwidth resulting in lower data rates. For missions requiring a higher data rate, higher frequency bands such as S or X-band are used. [2]

The choice of modulation and channel coding also has a drastic effect on the achievable data rate without affecting the satellites' size, weight, and energy consumption limitations. Modulation schemes such as PSK allow modulating multiple bits into one symbol - in comparison to BPSK which modulates one bit per symbol, QPSK uses the same bandwidth for twice the data rate by modulating two bits per symbol. Use of channel coding techniques allows trading the data rate for error detection and correction, achieving low Bit Error Rate (BER) at lower Signal-to-Noise Ratio (SNR)s. [3]

This master's thesis focuses on developing the High-Speed Communications Subystem (HSCOM) firmware and reception for ESTCube-2, a 3U CubeSat by the students of University of Tartu and the Estonian Student Satellite Foundation. The aim of this work is to interface HSCOM with the rest of the satellite, implement a high-level logic for receiving data from the payloads onboard, transmission of files and their reception at the Ground Station (GS), including demodulation and decoding of the packets. This thesis is structured to give the reader an overview of similar communication systems used in CubeSats in chapter 2, outlines previous work done by other team members in chapter 3, and the work done during the thesis in chapters 5, 6, 7 and 8.

2 High Speed S-Band Communications in CubeSats

The VHF and UHF bands are widely used for satellite telemetry and beacons due to their low complexity and signal attenuation due to obstacles and weather conditions. This reliability is useful in transmitting important data, such as the telemetry and control of a satellite. However, for missions producing large payload data, using the same transceivers to downlink this data as for command and telemetry can cause congestion on this communication link. Furthermore, due to the congestion on the VHF and UHF bands, acquiring a license is difficult - and the successfully acquired licenses are for very narrow bandwidths. In CubeSat missions with instruments and payloads that produce a large amount of data requiring a high data rate downlinking, a secondary transmitter working in higher RF bands is needed.

In this section, the reader is given a brief overview of other CubeSat missions that have used a secondary S-band communication system for transmitting large amounts of payload data.

2.1 Aalto-1

Aalto-1 is a 3U CubeSat created by the students at Aalto University. The work on the satellite began in 2010, and it was originally planned to be launched by a Falcon 9 rocket in 2015. [4] However, due to multiple delays caused by the rocket, it was postponed until the successful launch aboard an Indian Polar Satellite Launch Vehicle on June 23rd, 2017. [5] The Aalto-1 carried three different scientific payloads; a spectral imager, a radiation monitor, and a plasma brake [6].

Of these payloads, the spectral Earth observation imager produced spectral images with sizes up to 10 MB, with possibility of compressing the images - down to 3.7 Megabytes in one instance. Thus the satellite required a high data throughput link for transmitting the spectral images down to the GS. [8] [9]

For the high speed communication system on Aalto-1, the HISPICO from IQ Technologies was considered as a possible high speed transmitter candidate. However, a custom-built transmitter solution was preferred because it offered more freedom and educational value when designing

the subsystem. [9]

The custom S-band communication system on Aalto-1 is based on the TI CC2500 transceiver [10] controlled by the Texas Instruments MSP430F2274 Microcontroller Unit (MCU) [11] due to their capabilities as well as space heritage. [9] The transceiver offers multiple modulations, of which Gaussian Minimum Shift Keying (GMSK) was chosen, and operates on 2.4 GHz with a data rate of 500 kBaud. The On-Board Computer (OBC) of the Aalto-1 controls the functionality of the S-band subsystem, which is powered off until a command to transmit images is uplinked via the primary communication system. The full architecture of the S-band communications system can be seen in Fig. 2.1.

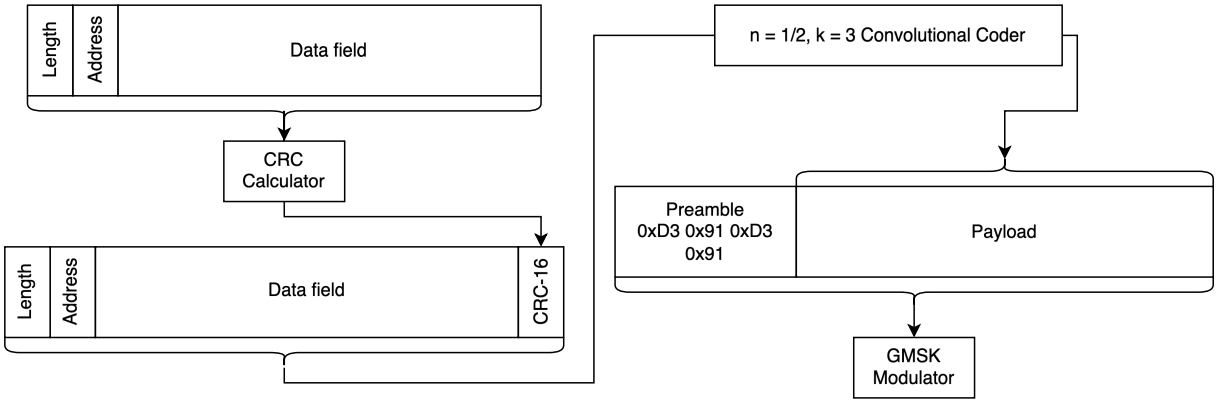

Figure 2.1: High level overview of the Aalto-1 S-band communications system.

The downlinked packet includes a preamble, followed by 256 byte payload, and supports optional Forward Error Correction (FEC) of rate $n = 1/2$ and constraint length $k = 3$. [12]

2.2 AMICal Sat

AMICal Sat is a 2U CubeSat developed by the Grenoble University Space Centre in France and the Moscow State University, Institute of Nuclear Physics in Russia. Launched on September 3rd, 2020, AMICal Sat carries a compact imager for capturing the Northern and Southern Lights. The imager uses the Onyx EV76C664 sensor [13], which produces 1408×1024 8-bit pixels, resulting in ∼1.4 MB files. These images are then compressed down to ∼27.9 kB, and downlinked. While the members of the project developed the payload itself, the satellite bus, including the communications subsystem, was purchased from SatRevolution, a private Polish company that specializes in nanosatellites. [14–17]

The communications subsystem contains an S-band transmitter based on a Nordic Semiconductor nRF24L01+ transceiver chip. Transmitting at 2.4 GHz with a 1 MBaud symbol rate using Gaussian Frequency Shift Keying (GFSK), the image data is encapsulated in a packet using the proprietary ShockBurst packet format, as seen in Fig. 2.2. [17]

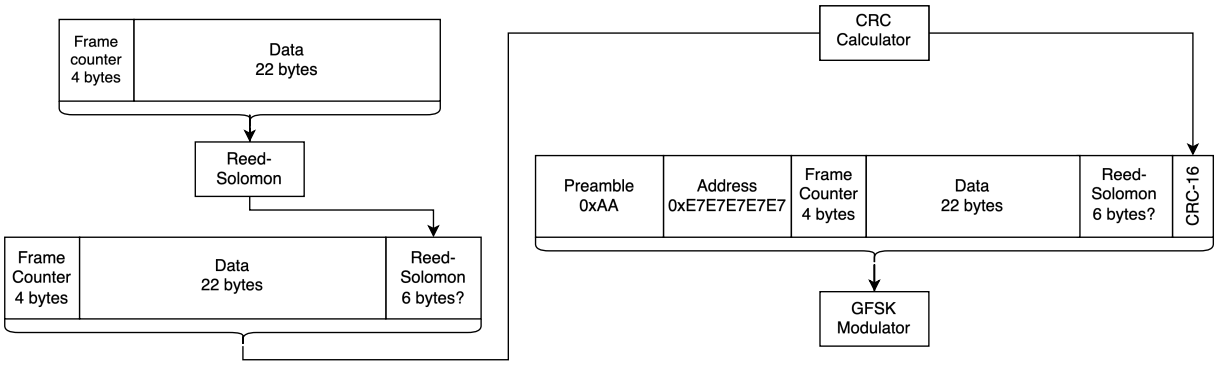

Figure 2.2: High level overview of the AMICal Sat S-band communications system.

The satellite uses 5 byte address field, 32 byte payload field, and 2 byte Cyclic Redundancy Check (CRC) field. Each image is sent with a 512 byte header, and the size of the image can differ depending on the compression applied. The contents of the 32 byte payload field depend on whether FEC is used:

- If no FEC is used, the payload field uses 2 bytes for the frame counter, and the remaining 30 bytes are used for the payload data.
- If FEC is used, the payload field uses 4 bytes for the frame counter, 22 bytes for the payload data, and 22 bytes for the Reed-Solomon parity bits. This is likely an error in the datasheet [16], as the total amount of payload data exceeds 32 bytes. The authors were contacted, but no response was received.

2.3 PICASSO

PICASSO is a 3U CubeSat mission initiated by the Belgian Institute for Space Aeronomy in partnership with VTT Technical Research Center of Finland, Centre Spatial de Liège of Belgium, and Clyde-Space Ltd. from the United Kingdom. PICASSO is administrated by the European Space Agency. Launched on September 3rd, 2020, PICASSO aims to demonstrate the capacity of small satellites within the context of high-cost and important science experiments. It carries a miniaturized hyperspectral imager to measure vertical profiles of ozone, and a Sweeping Langmuir Probe to measure electron density. The hyperspectral imager, similar to the one used in Aalto-1, is the main source of large payload data. [18]

The S-band communication system uses STX by Clyde-Space and offers 1 MBaud symbol rate using QPSK modulation at 2.2 - 2.45 GHz. The STX offers rate $n = 1/2$ convolutional encoding with a constraint length of $k = 7$, differential encoding, and scrambling using the IESS-308 standard. The STX does not offer any framing itself, and details of the frame structure are not available for public, however they are mentioned to be ”modified Consultative Committee for Space Data Systems (CCSDS) packets”. The STX works in two modes: the synchronization mode and the data mode. During the synchronization mode, the transmitter continuously trans-

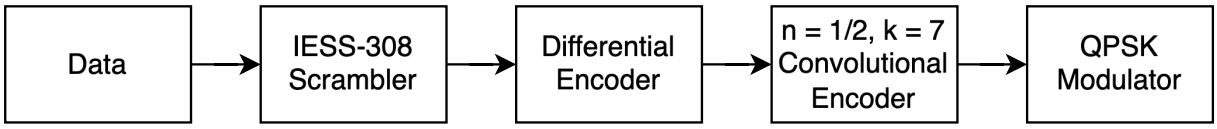

Figure 2.3: Possible high level overview of the PICASSO S-band communications system using an STX transmitter. The exact details of the framing are unknown.

mits the CCSDS 32 bit syncword $0x1ACFFC1D$. This mode can be used to synchronize the receiver at the GS before the packet transmission in data mode can begin. [18, 19]

2.4 Summary

The missions examined in this section have been chosen due to their variety in components and techniques, despite using similar center frequencies and their resulting baud rates.

For Aalto-1 S-band system, the HISPICO was considered for use as a Commercial Off-The-Shelf (COTS) product, but a system created in-house was preferred due to its educational value and freedom in building the system. This is a direct inverse of the ESTCube-2, as HISPICO was chosen after the planned bespoke system could not be created due to budgeting constraints, explained further in Section 3. AMICal Sat and PICASSO both use COTS products, and detailed implementation of the S-band systems of these missions are not available for public. [9, 17, 18]

All missions examined in this chapter use a form of FEC. Aalto-1 uses the FEC offered by the transceiver chip used in the system, whereas AMICal Sat implements their own. Both missions offer FEC optionally, and main configuration is transmission without FEC. [12, 17]

Satellite	**Transmitter**	**Frequency**	**Modulation**	**FEC**	**Baud Rate**
Aalto-1	CC2500	2.4 GHz	GMSK	Convolutional Code	0.5 MBaud
AMICalSat	nRF24L01+	2.4 GHz	GFSK	Reed-Solomon	1 MBaud
PICASSO	STX	2.2 - 2.45 GHz	QPSK	Convolutional Code	1 MBaud

Table 2.1: Comparison of the missions examined in this section.

3 ESTCube-2 High Speed Communication Subsystem

ESTCube-2 is a 3U CubeSat built by the Estonian Student Satellite Foundation and the students of University of Tartu. The satellite carries three main payloads on board;

- Two earth observation cameras, to observe the vegetation of the Earth.
- 15 anti-corrosion materials attached to the satellite's exterior, to be analysed for their reaction when exposed to atomic oxygen.
- Plasma brake to de-orbit ESTCube-2 in what is to be the first successful in-orbit demonstration.

For the operation of the satellite and downlinking of telemetry, the satellite carries a UHF transceiver for telemetry and telecommands, operating at 435.8 MHz GFSK, with a baud rate of 9600. However, the Earth observation cameras were predicted to produce a large amount of images, each at least 4 MB in size. Using the primary UHF communication system, a single image can be downloaded in 45 days, assuming 15 passes per day, with 8 minutes per pass, and including the primary communications subsystem protocol overhead. Due to this, a high-speed communication subsystem was added to the ESTCube-2 to downlink the images produced by the Earth observation cameras.

Initially, this subsystem was planned to be based on a C-band transmitter created by the students of the Ventspils University in Latvia, and a first mock-up was created. However, the funding for the project was cut, and no further developments were made. Due to time constraints, another bespoke solution could not be built by the ESTCube-2 team, and instead was replaced by a COTS S-band transmitter, the HISPICO from IQ Spacecom. [20] The HISPICO is an S-band radio transmitter, operating in 2.1 - 2.5 GHz with Differential Quadrature Phase Shift Keying (DQPSK) and offering user data rates of up to 1.3 Mbps. Due to its small form factor of less than 0.1U, the HISPICO could be placed into the slot that was intially allocated for the transmitter from Ventspils. A dedicated STM32L462CE MCU [21] was added to control the HISPICO transmitter, in addition to storing and parsing its periodic telemetry. The HISPICO will be explored in detail in section 3.1, while the patch antenna will be explored in section 3.2.

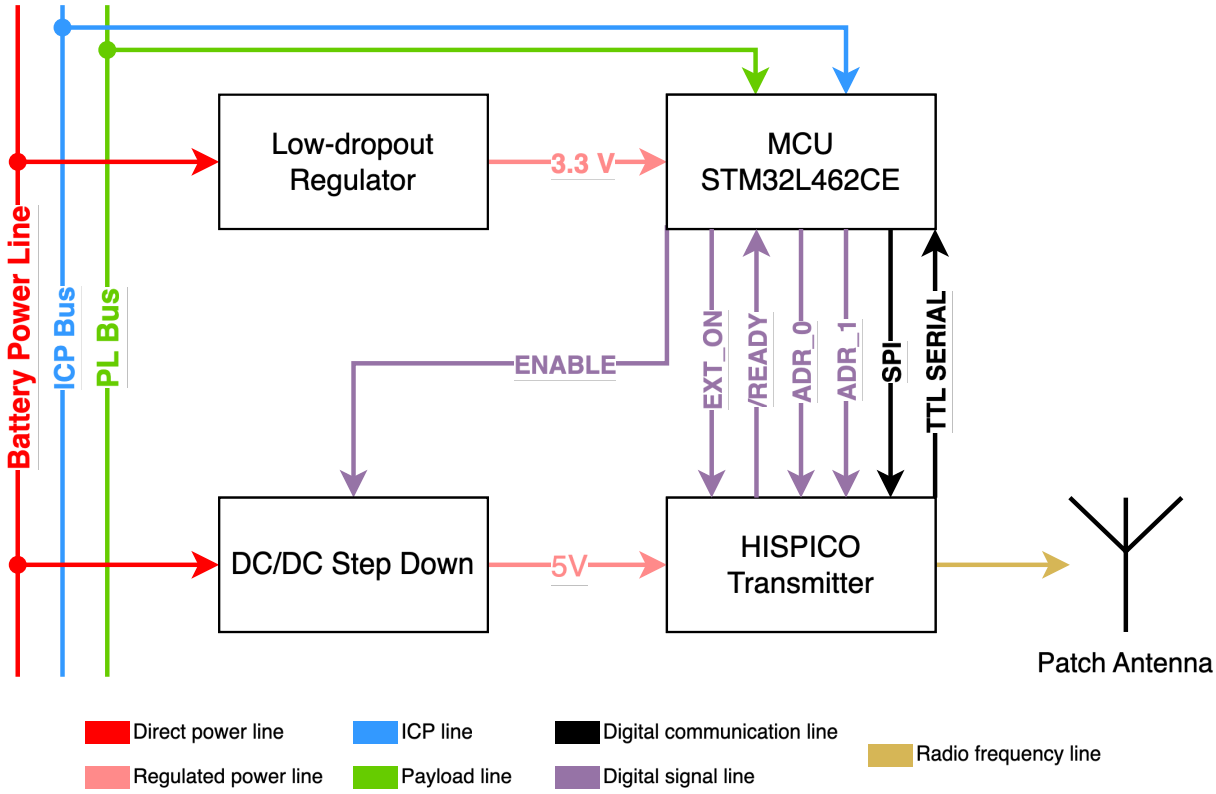

Figure 3.1: Block diagram of the main hardware components of the HSCOM on board ESTCube-2.

The HSCOM's main task is to transmit data received through the payload bus, which is connected to every payload aboard the satellite. Through the payload bus, the payloads can exchange data with the on-board computer and with HSCOM for transmitting their data down to the GS. The Internal Communication Protocol (ICP) bus is used to communicate with other subsystems in the satellite. This includes the transmission of housekeeping data that the COM subsystem periodically requests, and enables the possibility of transmitting ICP response and data packets through the HISPICO S-band communication link. A general overview of some of the hardware components and connections of the HSCOM subsystem can be seen in Fig. 3.1.

The MCU and the transmitter have different power requirements and thus are supplied by two different power lines. The MCU controls the supply of power to the transmitter through the $ENABLE$ line, allowing the satellite to deactivate the transmitter when not in use. After the transmitter is powered, the MCU can then turn on the transmitter by sending a logic high on the $ENABLE$ line. The exact sequence of communications required to turn on the transmitter will be covered in chapter 5.

After activation, the transmitter sends a telemetry block to the MCU over the serial interface every 5 seconds. The MCU can start forwarding data from the payload bus to the transmitter at any time. For forwarding the data to the transmitter, both the CLK and FRAME SYNC lines are

used in addition to the main Serial Peripheral Interface (SPI) line through which the data flows. The exact protocol used for this transmission will be covered in chapter 5. The transmitter uses a custom made patch antenna, covered in chapter 3.2, to transmit the data.

3.1 HISPICO Transmitter

The HISPICO is an S-band transmitter with a small form factor of less than 0.1U, transmitting at 2.4 GHz. It offers two interfaces for downlinking data; TTL serial over Universal Asynchronous Receiver / Transmitter (UART) with a maximum data rate of 17.1 Kbps, and SPI 3-wire with a maximum data rate of 1.3 Mbps. The HISPICO transmitter can be seen in Fig. 3.2.

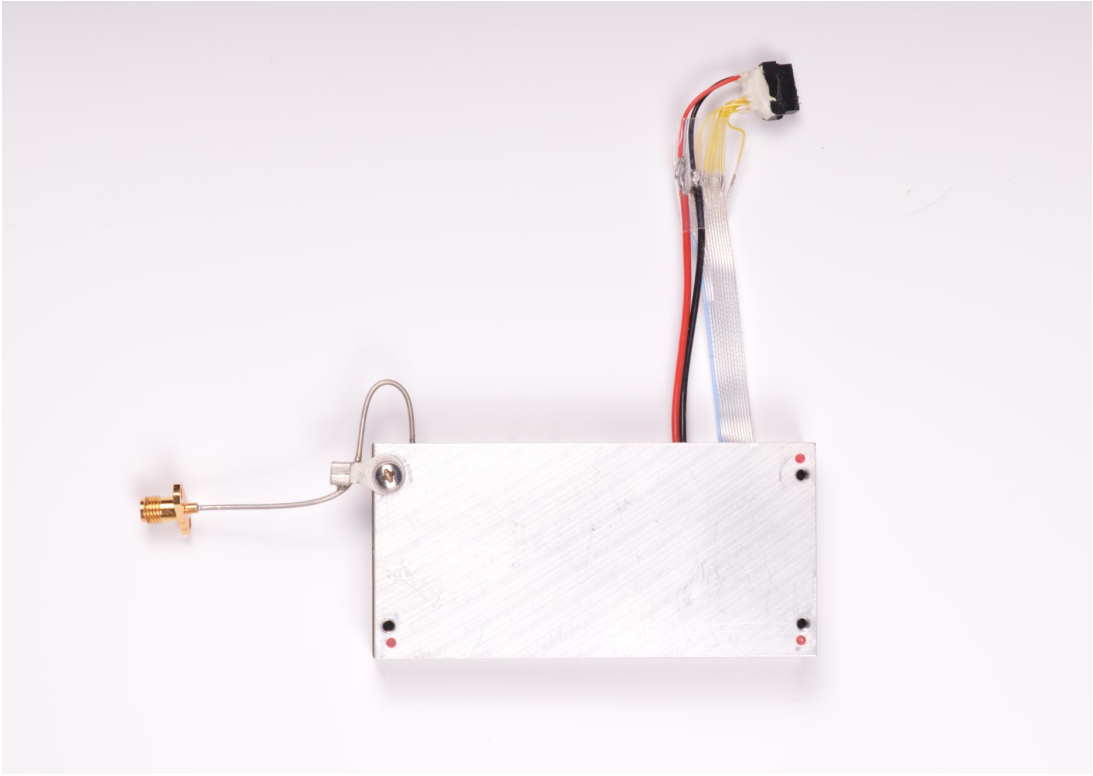

Figure 3.2: The HISPICO transmitter. The antenna cable of the engineering model of the HISPICO transmitter was replaced with a sturdier solution, as the original cable frayed during the development of the system.

The 3-wire SPI protocol used in HISPICO was initially incompatible with the STM32 processors used in ESTCube-2. Normally, the SPI protocol uses the CS signal to mark the beginning and the end of a transmission. The HISPICO 3-wire protocol instead used the CS to signal the start of every byte, toggling it on the same clock cycle as the first bit.

The ESTCube team consulted with the manufacturers of the transmitter regarding this issue, and it was suggested by the manufacturers to use the SPI TI mode [22] available on STM32 processors with a small modification to the transmitter firmware by the manufacturer to accept

the TI mode.

The comparison of all three SPI timings can be seen in Fig. 3.3.

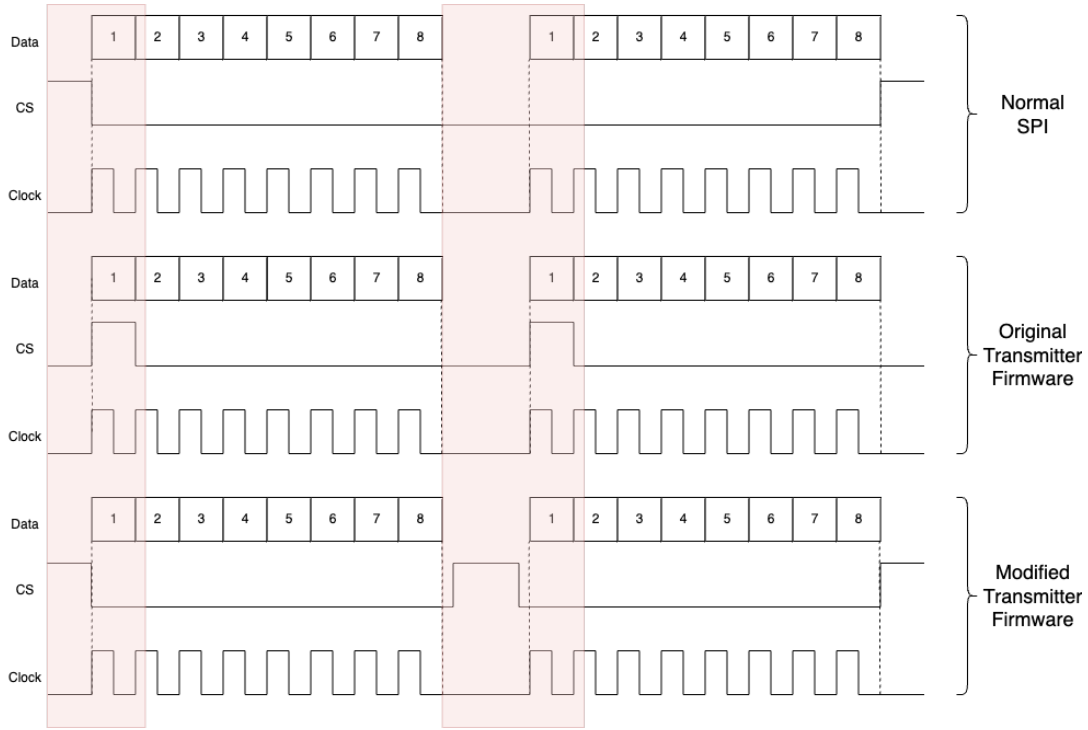

Figure 3.3: Comparison of the SPI protocols used in HISPICO. The difference in the Chip Select line is highlighted in red.

Furthermore, the transmitter can carry four different firmware images on board. A desired firmware image is chosen by setting the appropriate ADR pins HIGH or LOW prior to powering up the transmitter. The firmware images carried on board are all identical, with the exception of the FEC rates. FEC rates affect the ratio of useful data to FEC control bits used in the FEC algorithm, thus affecting the maximum amount of payload that can be transmitted with one packet. From a list of available FEC rates, 4 were chosen to be carried on board. These rates and the resulting maximum amount of data in one packet can be seen in Table 3.1.

When any FEC rate other than 1 is chosen, the transmitted data is passed through the FEC chip inside the transmitter. This chip also performs scrambling of the bits. Despites many requests by the ESTCube team, at the time of writing this thesis, the manufacturer has not shared any details of the scrambler in use, effectively barring the use of any hardware supported FEC algorithm. Thus, during this thesis only firmware slot three with no FEC was used.

ADR_0	ADR_1	Firmware	FEC Identifier	FEC Rate	Max. bytes per frame
LOW	LOW	0	1	0.489	250
LOW	HIGH	1	0	0.325	166
HIGH	LOW	2	6	0.223	115
HIGH	HIGH	3	5	1 (No FEC)	511

Table 3.1: Possible forward error correction schemes available onboard HISPICO and their respective firmware slots. The ”FEC Identifier” column is used to differentiate between the FEC rates by assigning them an identifier and is used in the HISPICO frame headers. See 6.1.

3.2 Patch Antenna

For S-band communications, different patch antenna prototypes were made and tested by the ESTCube team. The final iteration was made out of Rogers RO4003C-0600-1E-1E [23]. 114 x 152 mm panels with a thickness of 1.52 mm were used in testing and deemed appropriate for usage. The final antenna is a circularly polarized patch antenna with an ∼8dBi gain and a half-power beam width of ∼70°. [24]

Figure 3.4: The patch antenna used for HSCOM of ESTCube-2. [24]

3.3 High Speed Communication Breakout Board

The flight hardware of the HSCOM module resides on the battery management module aboard the satellite and thus was under constant development and testing by other team members. Due to a lack of flight model replicas, the ESTCube team created low cost modular development PCBs that contained all the elements of their respective flight hardware counterparts. In addition to being cheaper to produce, they also provided a form factor for easy access to probe signals and to debug in general. The breakout PCBs had interfaces that allowed them to be connected with the developers' PC, the flight hardware or other engineering models. By producing multiple cheap functional copies of the flight hardware, developers working on the same subsystem could carry on their work in parallel.

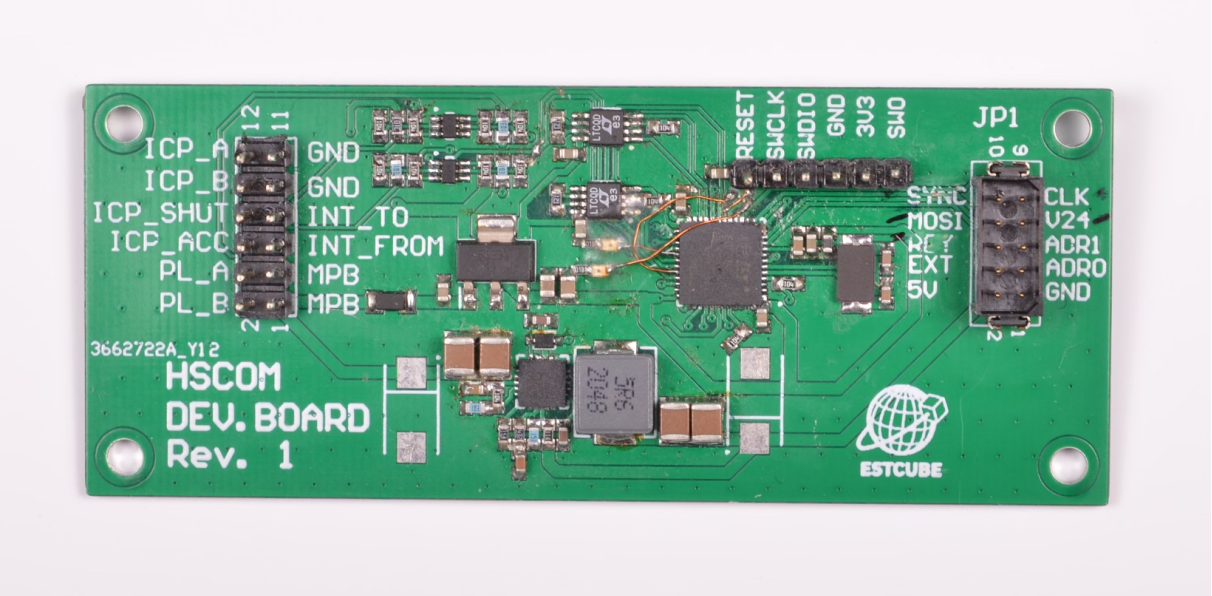

Figure 3.5: The HSCOM breakout board.

The work done in this thesis took place on the HSCOM breakout board (Fig. 3.5).

3.4 High Speed Communication Subsystem at the Start of the Thesis

Although the components required to build the subsystem were purchased, there had not been substantial work done to integrate the subsystem into the actual flight hardware. The HISPICO had electrically been powered on but had not been enabled, and its transmission capabilities had not been verified. The HSCOM hardware was checked for shorts and structural integrity, it had been added to the flight model physically but the subsystem had not been integrated to the rest of the satellite via software. No high-level code or drivers on the HSCOM MCU were written to interface with the transmitter.

4 Thesis Scope

In addition to the educational value of building a satellite, ESTCube-2 carries multiple payloads and scientific instruments on board. The high-speed communication subsytem plays a critical role in receiving the data from these payloads on Earth.

The goals of this thesis are to integrate the COTS HISPICO transmitter to the ESTCube-2 HSCOM subsystem and to build a receiver for receiving the HSCOM data at the GS of ESTCube-2. After analysing the system, the following tasks were agreed upon to achieve those goals:

- Create device drivers for interacting with the HISPICO transmitter on the HSCOM MCU.
- Create and implement transmitter daemon for the high level logic and operations of transmission required by ESTCube-2.
- Create a payload handler in HSCOM, with knowledge of all payloads connected to the ESTCube-2 payload bus, for recognizing and handling different packet types.
- Build the receiving pipeline in GNURadio for receiving, demodulating, decoding and deframing packets received from ESTCube-2 HSCOM.

5 Firmware Development

The HSCOM firmware is written for the STM32L462CE MCU in C, using FreeRTOS [25]. The ESTCube-2 team uses a Hardware Abstraction Layer (HAL) written in-house, as well as HAL drivers ensuring safe access to peripherals and preventing race conditions.

The firmware development of HSCOM consists of the device driver consisting of the logic required to operate the HISPICO, and the high-level transmission logic for correctly framing and transmitting the data.

5.1 Device Driver

The device driver enables safe and easy access to the device by presenting all of the device functionality in callable functions containing the necessary logic and safety precautions. If the device is ever replaced, only the modification of the device driver is necessary, ensuring the high-level application logic stays the same.

Operation of the HISPICO transmitter can be divided into three main functions: initialization, telemetry reception, and transmission.

5.1.1 Initialization and Power Up

The initialization of the HISPICO transmitter requires certain steps to be followed.

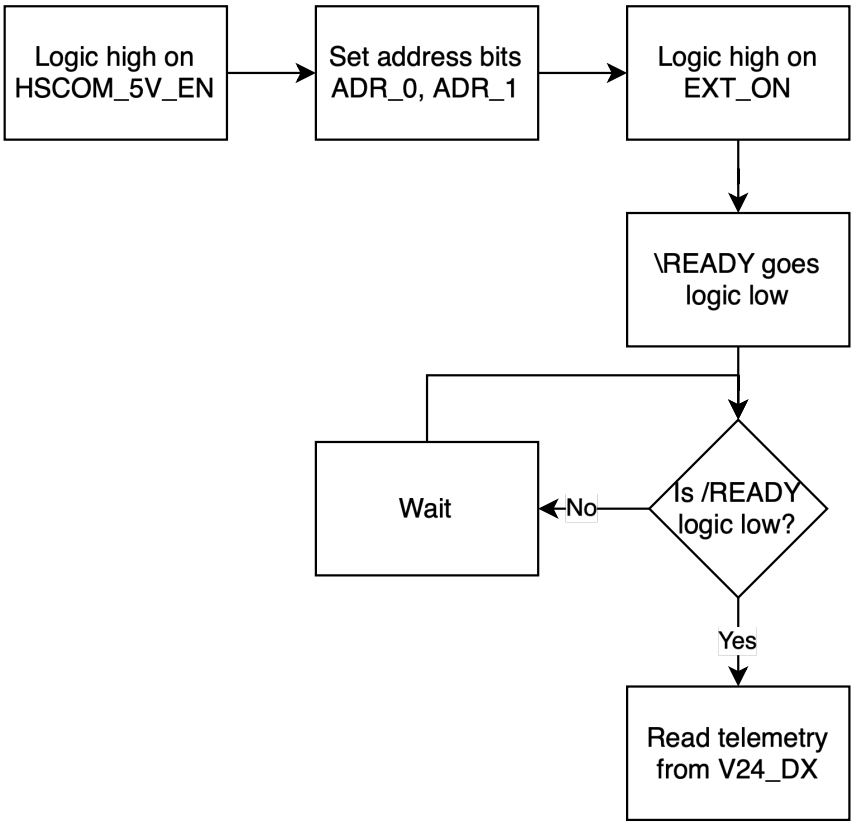

Figure 5.1: Flowchart of the HISPICO transmitter turning on sequence.

Firstly, the $ENABLE$ pin of the HSCOM must be set HIGH by the MCU to turn on the power rail supplying the transmitter. Secondly, the MCU has to choose the correct HISPICO firmware slot through the ADR_0 and ADR_1 pins. Possible values of the ADR pins and their corresponding firmware configurations can be seen in Table 3.1. The ADR pins must be set before the initialization of the transmitter occurs, and must not be modified during the initialization process.

After the ADR pins are set, the transmitter self-initialization can be started by setting the EXT_ON pin HIGH. The internal initialization process takes a variable amount of time, reported in the transmitter documentation to be between 500 ms to one second. For determining the exact time the initialization is complete, the $\overline{READY}$ pin can be used. The initialization sequence and the expected states of the EXT_ON and $\overline{READY}$ pins can be seen in Fig. 5.2.

The $\overline{READY}$ pin is set HIGH by the transmitter when the EXT_ON pin is set HIGH, and remains HIGH throughout the initialization process. Once initialization is complete, the $\overline{READY}$ pin is set to LOW. The device driver checks the $\overline{READY}$ pin until it is set to LOW, or a timeout occurs after one second. If a timeout has occurred, an exception is raised. If not, the initialization sequence is completed successfully.

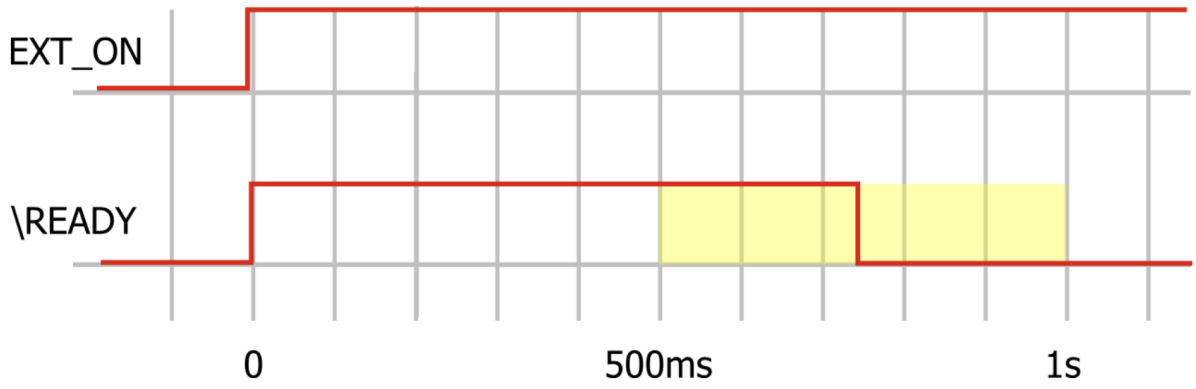

Figure 5.2: EXT_ON and $\overline{READY}$ signals during bootup. The time at which the $\overline{READY}$ signal might go low is highlighted in yellow.

5.1.2 Telemetry

The HISPICO transmitter outputs a telemetry block of 203 bytes every 5 seconds on the TTL serial interface over UART through the V24_DX pin. The UART configuration uses 8 data bits, no parity, and one stop bit, with a baud rate of 38400.

Due to an error present on the breakout board (Section 3.3) as well as the flight model of the HSCOM subsystem, the UART configuration deviates from the normal operation configuration. This error will be explained in more detail in Section 5.1.3. The solution to this error requires the UART RX pin, used to receive the HISPICO telemetry, to be disabled. To be able to receive the telemetry without the use of the RX pin, two registers in the UART configuration of the STM32L4 MCU must be changed. Setting the HDSEL bit (half-duplex communication, using only the TX pin) and SWAP bit (TX and RX pins are swapped) at the same time allows the MCU to disable the RX pin, while using the TX pin to receive the telemetry. The HAL as well as the UART drivers present in the ESTCube-2 firmware did not offer configuration settings for these bits and therefore had to be modified by the author to support the required configuration.

Telemetry Block Variable Name	Example	Explanation
firmware_version	01.03.00	Firmware version.
firmware_compilation_time	May 30 2011 17:44:59	Compilation date of firmware in use.
running_time	2597	Current running time in seconds.
firmware_crc	F2A3	CRC16 of the current firmware.
config_crc	1A1A	CRC16 of the current configuration data.
temperature	+039	Internal transmitter temperature in celsius.
received_commands	23	Numbers and repetition of received Telecommand bytes.
unknown_commands	11	Numbers and repetition of unknown Telecommand bytes.
last_commands	0	Numbers and repetition of last Telecommand bytes.
frequency	2425000	Configured transmission frequency in hertz.
txpo	208	Configured transmission power.
operating_mode	ON Normal 1.02	State, operating mode, FEC rate.
transmitted_data	0010000000	Transmitted data since power up..
data_interface	UART 115200	Operating mode of data interface.
crc	3EF0	CRC16 of telemetry block.

Table 5.1: Information contained within the telemetry block of HISPICO.

The device driver offers a function to wait for, receive, and format the telemetry block. The telemetry block starts with a preamble sequence of 4 bytes *0x1B 0x5B 0x32 0x4A*, allowing the firmware to subscribe to a character match interrupt on the serial line for the first character in the sequence. Upon the triggering of this interrupt, the device driver then reads the following characters of the preamble, checking whether the entire sequence is received. When the last character of the preamble is received, the driver then reads 199 bytes, as this is the remaining bytes expected in the telemetry block. The next expected character of the telemetry block after the preamble is always 'H', and this information is used to verify the expected telemetry block is received. Upon reception of the entire telemetry block, it is parsed and mapped to variables

within the telemetry struct. The variables, example values, and explanation of these variables can be found in 5.1. The driver then returns the filled telemetry struct. These values are then used to fill the housekeeping data of the HSCOM system. At the moment, the CRC fields of the telemetry block are only used in the housekeeping data, thus the validity of the data can be checked by the receiving GS.

5.1.3 Transmission

The HISPICO uses SPI with a maximum clock frequency of 1.06 MHz for downlinking data. As was previously mentioned in Section 3.1, the HISPICO firmware was custom-made for use with STM32 SPI protocol, and in particular, the TI mode. The custom-made SPI signal timings can be seen in 5.3. The implementation of the TI mode was not supported in the ESTCube-2 HAL and SPI driver, and the support for hardware controlled chip-select as well as controlling the register for activating the TI mode had to be implemented by the author.

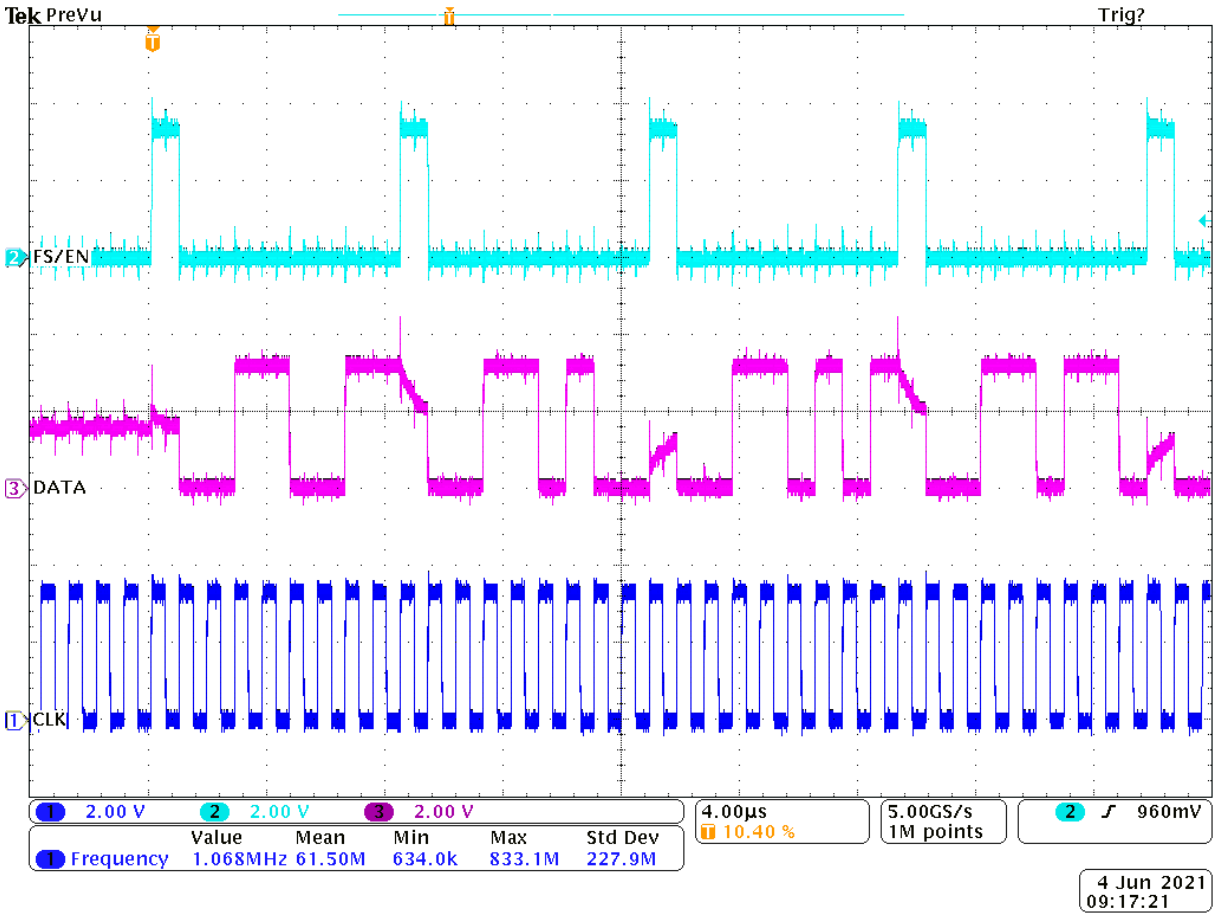

Figure 5.3: An oscilloscope screenshot received from the manufacturers of HISPICO, showing the modified timings of SPI lines that HISPICO accepts. Note that there are 9 clock pulses per byte, with the Chip Select pulse before the first bit of the byte.

Upon implementation of the TI mode, the data transmission was tested. The telemetry data

received from the HISPICO indicated that the amount of bytes transmitted by the transmitter did not match the data input by the MCU. However, the analysis of the data output from the MCU with a logic analyzer showed all bytes were being transferred to the transmitter correctly. During testing, a ground truth was not present, and it could not be established whether the errors existed on the transmitter, receiver, or both.

The manufacturers of HISPICO were contacted with regards to this issue, and an agreement was made to borrow a HISPICO receiver for development of the HSCOM subsystem. Upon testing of the HSCOM system with the HISPICO receiver, it could be seen that the majority of the received bits were ones. Further testing was done by connecting the SPI lines on board to an oscilloscope. The results showed that the DATA line had a HIGH-level voltage of 3.3 V as expected, but a LOW-level voltage of 1.46 V, which can be seen in Fig. 5.4.

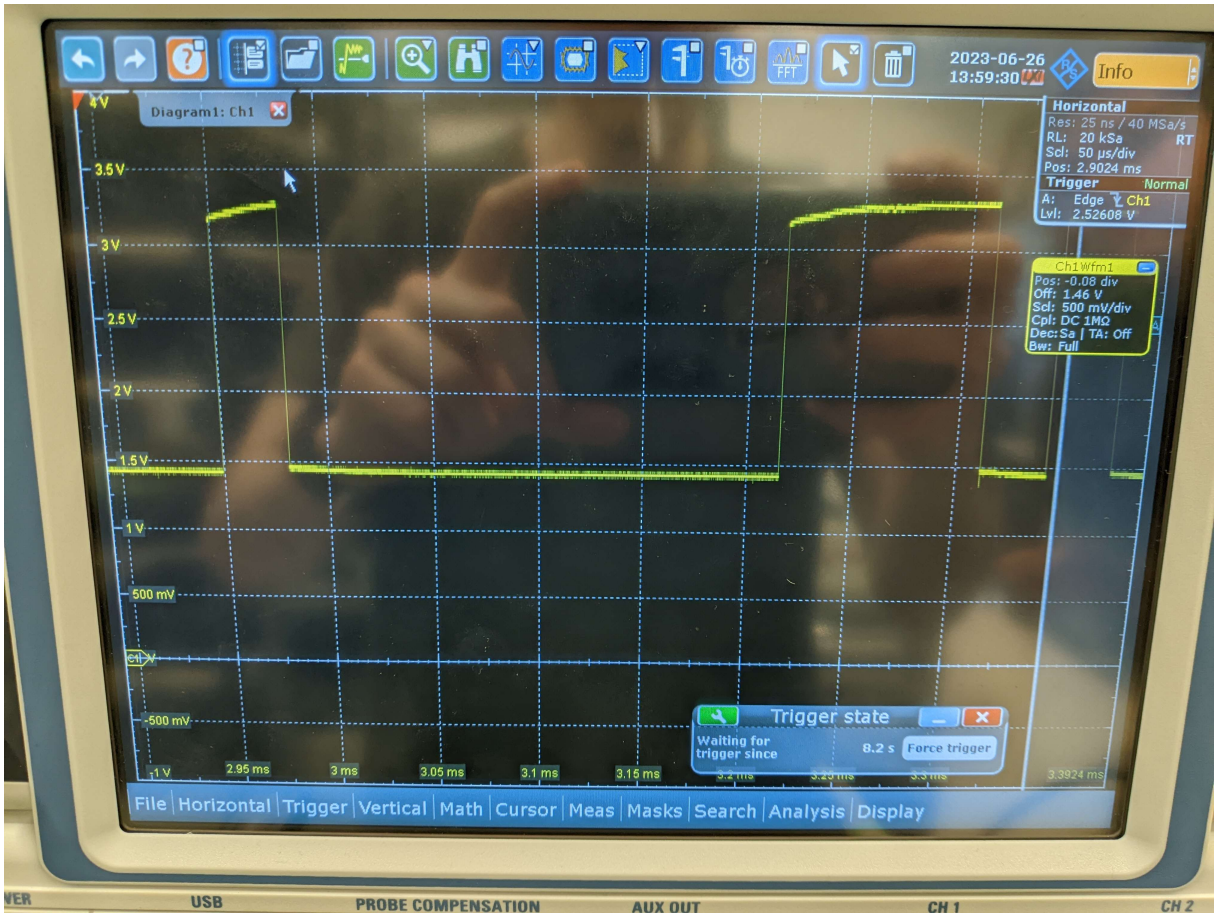

Figure 5.4: Inspecting the SPI MOSI line with an oscilloscope revealed the logic LOW signal never going below 1.45 V.

The datasheet for the HISPICO indicates that the HISPICO expects 2 - 3.3 V for HIGH-level voltage, and 0 - 0.8 V for LOW-level voltage. Further analysis revealed that two data lines on the HSCOM PCB, the UART TX, and the SPI MOSI, were erroneously connected together. At the time of the discovery of the bug, the flight model of the ESTCube-2 had already been

prepared for launch and could not be modified. Therefore, a solution in software was required. As mentioned in Section 5.1.2, disabling the UART TX pin by setting the HDSEL and SWAP registers allows the UART to transmit with the UART RX pin, and fixes the SPI MOSI voltage level error.

Furthermore, testing with the HISPICO receiver revealed the presence of missing bytes from the transmitted data. Logic analyzer results showed that the CS and CLOCK signal timings of the SPI TI mode, as recommended by the manufacturer, did not match the timings required by the HISPICO firmware (Fig. 5.3).

Despite the graph showing nine clock cycles per 8-bit byte in accordance with the SPI TI mode, the extra clock cycle is after the last bit of the byte, whereas for the TI mode the extra clock cycle happens before the first bit of the byte. This mistiming resulted in bytes being discarded.

Instead of using TI mode, the signal timings required by the HISPICO firmware can be matched by using the NSSP mode. The comparison of NSSP and TI modes can be seen in 5.5.

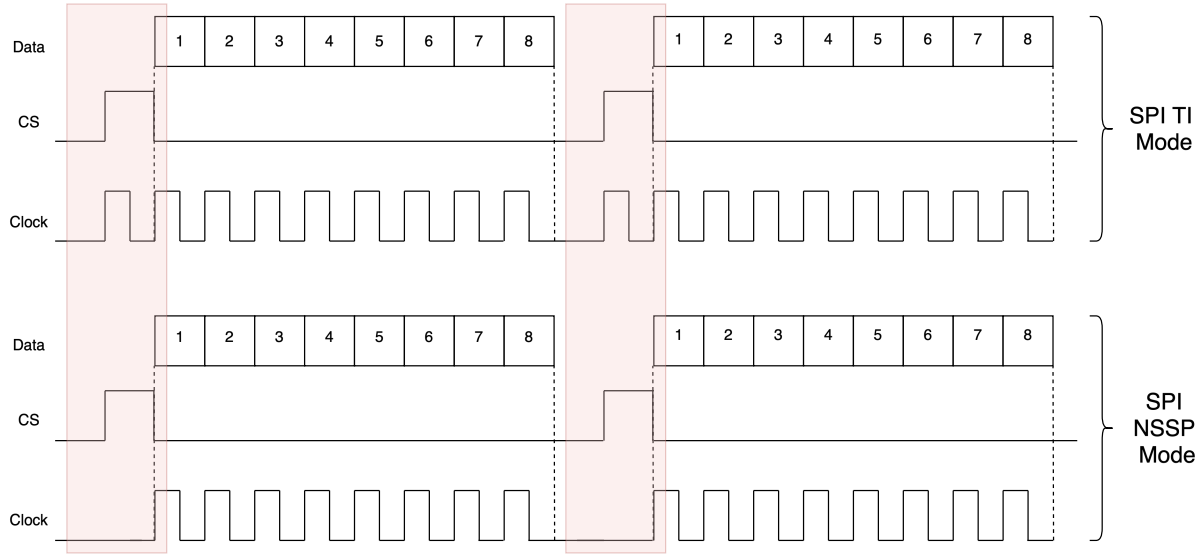

Figure 5.5: Comparison of SPI TI mode and SPI NSSP mode. The differences in Clock and Chip Select lines are highlighted in red.

5.2 High Level Transmission Logic

The HSCOM subsystem is connected to both the ICP and the payload buses, as seen in Fig. 3.5. Though the main purpose of the HSCOM subsystem is to transmit payload data, it can also transmit data received via an ICP packet as back-up. Any data received via the payload bus is first received by the HSCOM payload handler, ensuring the reception of full payload packets. The full logic of the payload handler is explained in Section 5.2.1. Both the payload packets through the payload handler, and the packets received through ICP are then sent to the transmission daemon, which offers forward error correcting capabilities, and then sent to

the HISPICO transmitter. The full logic of the transmitter daemon is explained in Section 5.2.2. The full flow of the high-level logic present in HSCOM subsystem can be seen in Fig. 5.6.

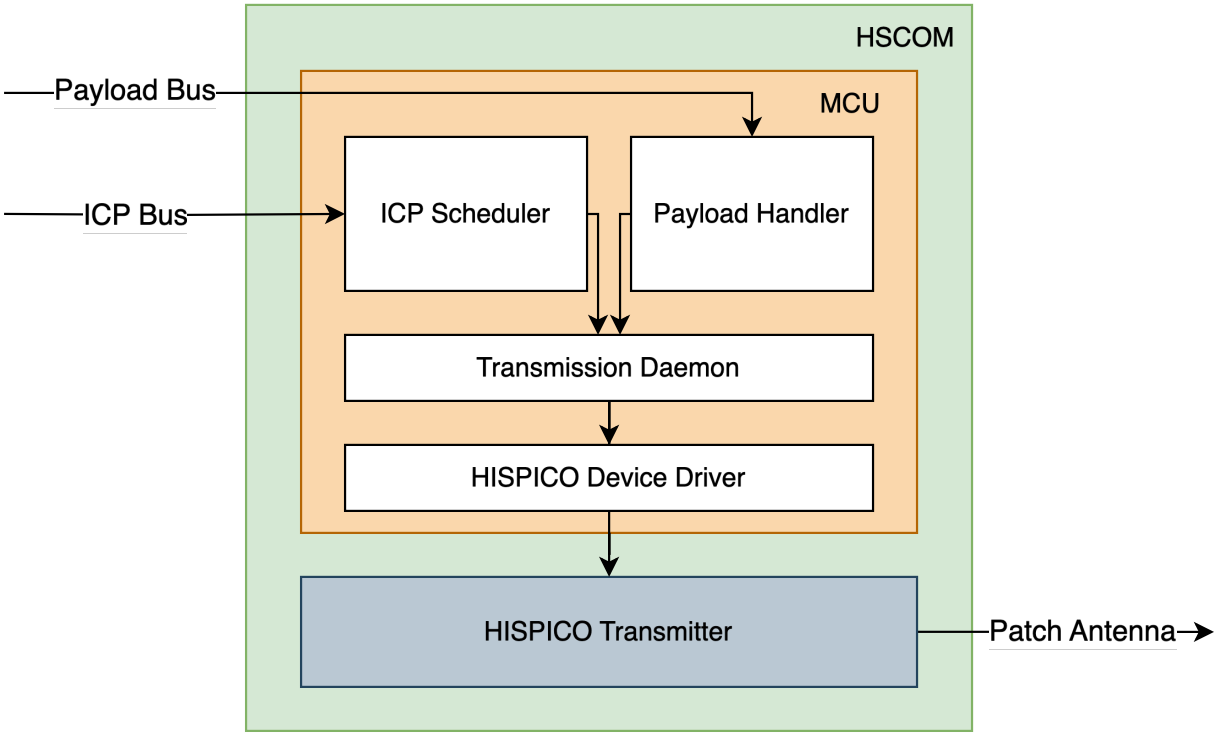

Figure 5.6: Block diagram of the high level transmission logic of ESTCube-2 HSCOM.

5.2.1 Payload Handler

The format of the payload data received may differ depending on the source payload. A payload handler was created to determine when a full packet of payload data is received. The payload handler has information about the payloads capable of transmitting data through the payload bus, and the format of the packets. During the work described in this thesis, only one payload, the CAM subsystem, was present and capable of using the payload bus. The CAM subsystem uses Consistent Overhead Byte Stuffing (COBS) packets, delimited with two zero bytes at the beginning, and one zero byte at the end.

The payload handler is initialised at power-up of the HSCOM MCU. The main task of the payload handler then begins waiting for the zero byte delimiters of the CAM packets. When there is a zero byte present on the payload bus, a character match interrupt occurs, resuming the main task of the payload handler. Once awoken, the payload handler also wakes the transmission daemon. The payload handler then receives the data present on the payload bus, and checks whether two zero bytes in succession are received. Upon receiving the double zeroes, the payload handler fills the internal buffer with the data from the payload bus until the end delimiter zero byte is received. When an entire packet has been received, the payload handler sends the buffer contents to the transmission daemon and starts waiting for another packet. The full flow of the payload handler can be seen in Fig. 5.7.

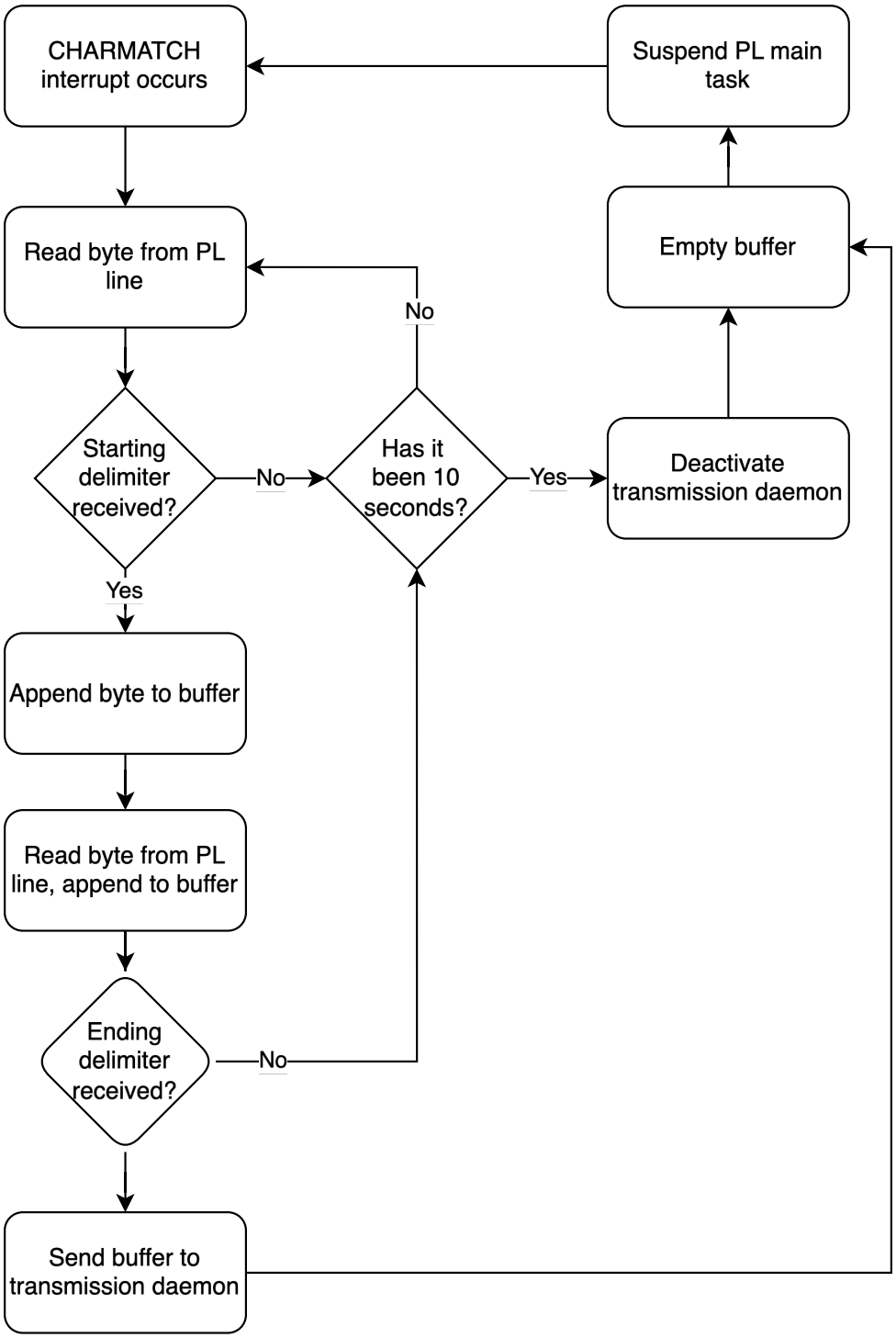

Figure 5.7: Flowchart of the high level payload handler logic of ESTCube-2 HSCOM.

5.2.2 Transmission Daemon

The transmission daemon is initialised at the power-up of the HSCOM MCU. Like the payload handler, the main task of the transmission daemon is then suspended. The transmission daemon offers a function for waking up the main task, powering up and turning on the transmitter. This function can be called from either the payload handler for downlinking of any payload data, or from an ICP command to the HSCOM for downlinking of any other data. The steps followed

for powering up the transmitter are explained in Section 5.1.1. After the transmitter power-up is complete, the input queue of the transmission daemon is read and stored in an internal buffer.

Due to the circumstances mentioned in Section 3.1, no hardware FEC can be used. To alleviate this problem, the transmission daemon offers the possibility of FEC encoding packets before transmission. If the FEC option is enabled, the transmission daemon then awaits for a set amount of bytes to be encoded before transmission. If no FEC is used, the transmission daemon waits until there is enough data to fill one HISPICO frame to full.

If no data is received after 10 seconds, the transmission daemon transmits any data remaining in its internal buffer, powers down the transmitter, and suspends its main task. The full flow of the transmission daemon logic can be seen in Fig. 5.8.

Forward Error Correction

Due to not having access to the FEC chip on the transmitter, two simple FEC algorithms were implemented in the firmware on the MCU. As the HISPICO already uses FEC for its frame headers, the same FEC algorithm was implemented for the payload for simplicity when decoding. The algorithm is a rate $n = 1/2$ convolutional code with a constraint length $k = 9$ and polynomials $g_0 = 753, g_1 = 561$. The rate of the convolutional code affects the data rate. For every bit of the payload data, two bits must be transmitted. Furthermore, the algorithm used in the header of the HISPICO frames use 16 tailbits. These bits are produced when $k - 1$ zero bits are input into the encoder after the payload data is encoded. This ensures that the encoder starts and ends in an all-zero state, simplifying the decoding process. Therefore, using the given convolutional code algorithm decreases the data rate by:

$$\frac{L}{n[L + (k - 1)]}$$

where the L is the data bits input into the encoder.

In addition to convolutional codes, another FEC algorithm was added to offer less of a data rate loss. Reed-Solomon code was implemented as 255 byte blocks, with 223 data bytes and 32 parity bytes, according to the CCSDS standard. [26] Phil Karn's FEC library [27] was used to implement the Reed-Solomon algorithm in the firmware. If there is a need to encode a block smaller than 223 bytes, Reed-Solomon offers the possibility of lowering the data rate by padding zero bytes to the end of the block until 223 bytes are reached, before the data is encoded. After encoding, the zero bytes are removed, and the resulting block is transmitted. The decoder on the receiving end must also append the same amount of zero bytes to the data before decoding.

For both FEC implementations, the decoder at the receiving side must know how many bytes were put into the encoder to be able to decode each block of data. This can be solved by

appending the data block size to the encoded block of data, with the caveat that the data block size information can not be encoded along with the data. Therefore any corruption, however small, would result in inability to decode the original data. Alternatively, the data block size can be separately encoded, adding the possibility to recover data size information in the event of corruption, but would lower the data rate further.

To solve this issue, the HSCOM firmware always encodes a fixed amount of data per block. When a block of data is received, it is divided into 223 byte sections for both FEC algorithms and transmitted. If the data is not divisible by 223, the last section with size that is less than 223 byte is either padded for Reed-Solomon, or encoded as is for Convolutional Code.

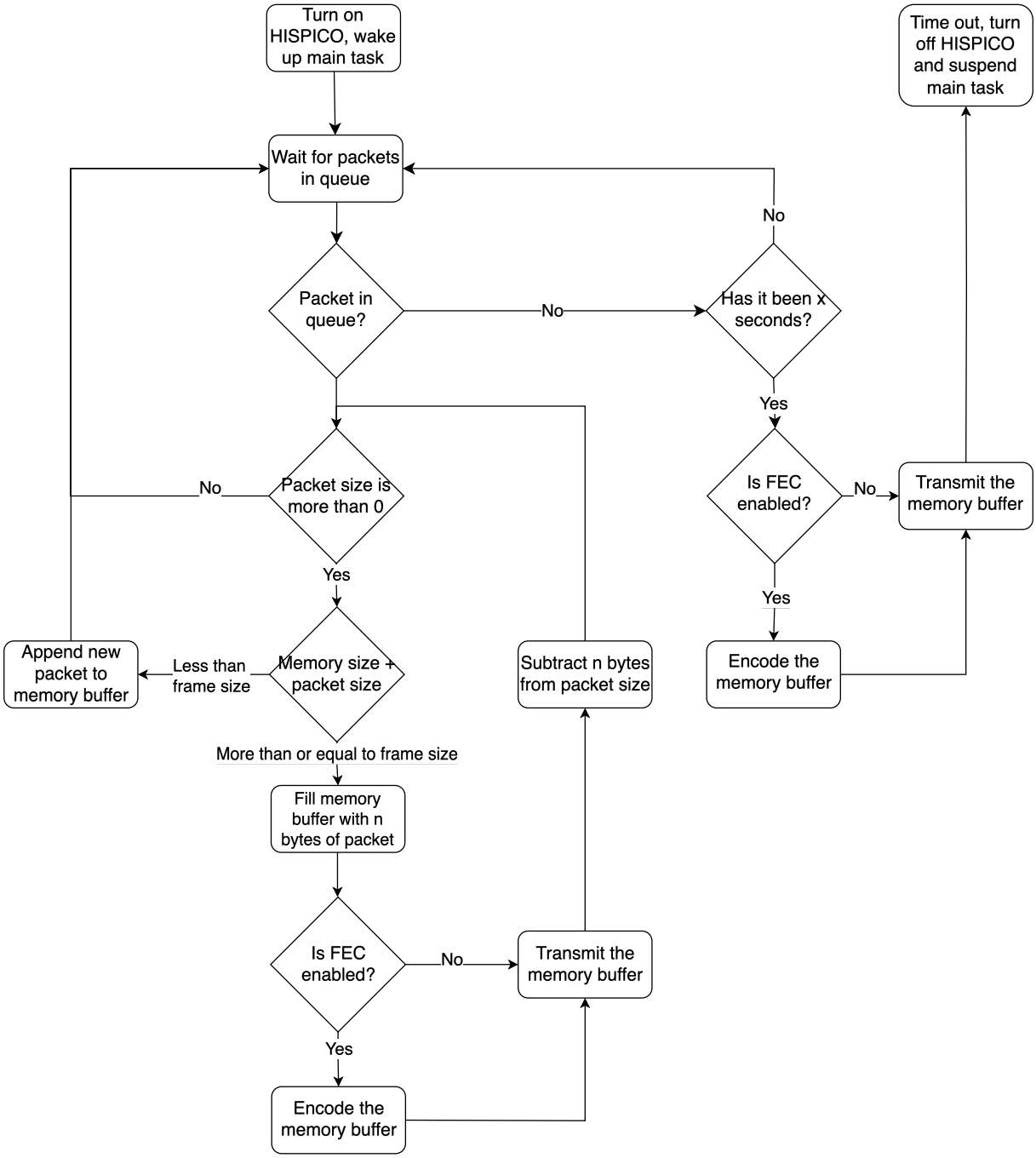

Figure 5.8: Flowchart of the high level transmission daemon logic of ESTCube-2 HSCOM.

6 Reception, Demodulation, Decoding

For development and testing of reception, a LimeSDR Mini with a 20 dB attenuator chain was connected to the antenna cable of the transmitter.

The HISPICO transmits at a constant symbol rate of 1.2 MHz. According to the datasheet supplied by the manufacturer, each frame transmitted is made of three sections:

- a preamble of 126 bits modulated with BPSK,
- a header of 66 bits modulated with Differential Binary Phase Shift Keying (DBPSK),
- the payload of 4096 bits modulated with DQPSK.

In a frame, the 1120, 1121, 2242 and 2243rd symbols are blank. Furthermore, every 2047th frame consists of completely blank symbols.

A GNURadio Companion flowchart was created using the `gr-satellites` library [28] and three custom blocks to demodulate, decode, and deframe the incoming packets. The flowchart can be used with a LimeSDR source block or raw recordings. The signal has an RF bandwidth of 1.3 MHz. Therefore, a sampling rate of at least 3 MHz is recommended to capture the entire signal. The Symbol Sync block was used to perform clock recovery, and a Costas Loop was used to perform carrier recovery.

Instead of demodulating the BPSK sections of the frame separately than the QPSK payload, thus increasing complexity and efficiency, a Costas Loop of 4th order followed by a QPSK demodulator was used, effectively decoding the BPSK parts as QPSK. This is possible due to both demodulation schemes using the phase of the received signal - a BPSK modulated signal only uses phases of 0° and 180°, while QPSK uses 0°, 90°, 180° and 270°. Therefore, a BPSK-modulated signal is a valid QPSK signal, as well.

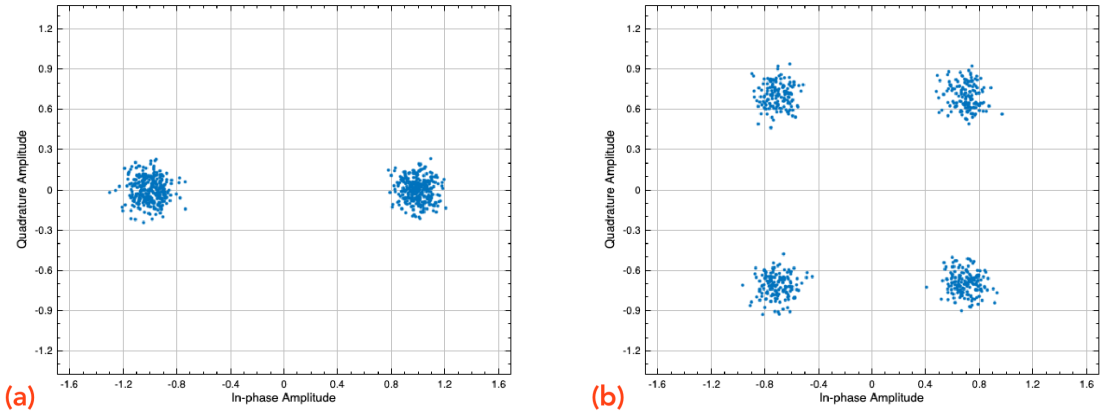

Figure 6.1: Constellation diagram of a; (a) BPSK modulated signal. The data points are divided between 0° and 180°. (b) QPSK modulated signal. The data points are divided between 0°, 90°, 180° and 270°.

The full flowchart for receiving, demodulating, decoding, and parsing packets can be found in Fig. 6.2.

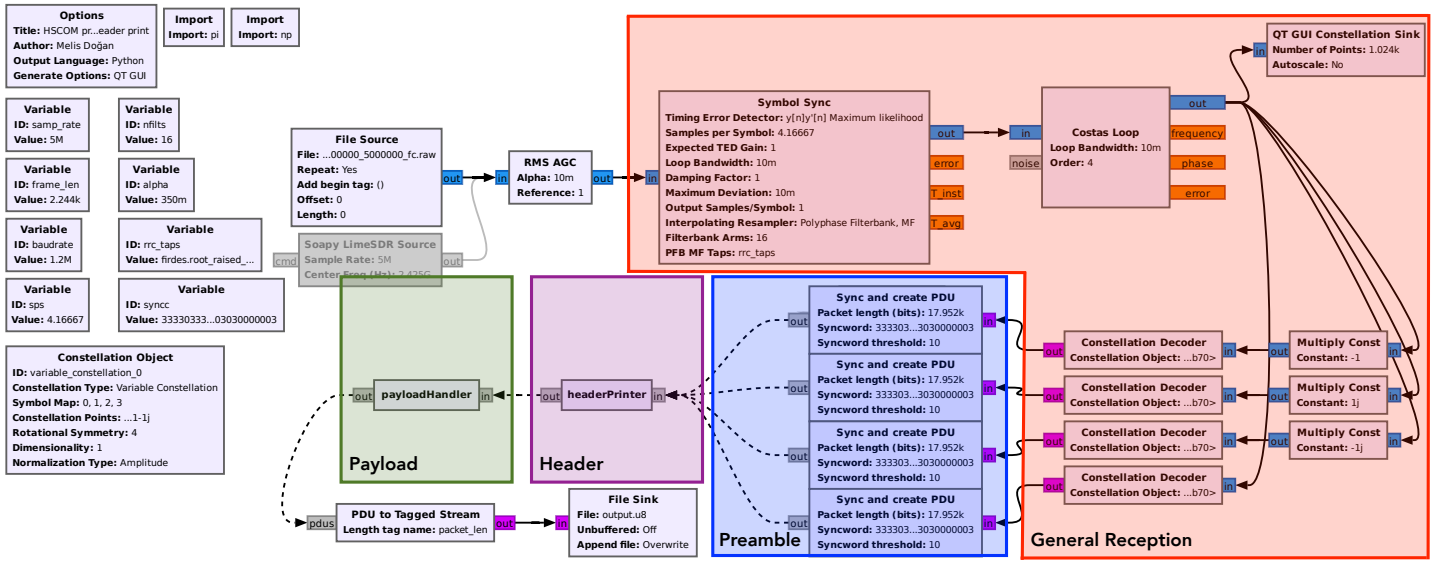

Figure 6.2: The GNURadio Companion Flowchart for Reception of ESTCube-2 HSCOM.

6.1 Preamble

The preamble is a 63-bit sequence, transmitted twice consecutively with a BPSK modulated wave. The hexadecimal sequence, 0x3F566ED271794610, is transmitted in little-endian bit order with its last bit discarded. This results in the sequence

`111101111001110101100001011100011011010010001001100101010000001`,

repeated twice at the start of each packet.

Due to QPSK demodulation instead of the intended BPSK, each bit is repeated. This is due to QPSK encoding two bits for each symbol - therefore 0° maps to the symbol 00, and 180° maps to the symbol 11. Packing these two bits into one symbol results in a sequence of 0 and 3 - therefore the sequence to sync becomes

`333033330033303033000030333000330330300300030033003030300000003`.

Furthermore, the BPSK modulation introduces a phase ambiguity of 180°. However, because it is demodulated by QPSK, which introduces a phase ambiguity of 90°, the preamble must be searched on all four possible shifted constellations. Only one of these branches results in a successful synchronizing of the sync word. Therefore, 4 parallel branches, all shifted by 90° from each other are used to sync to the preamble.

The ”Sync and Create Protocol Data Unit (PDU)” block from the gr-satellites is used to sync to the preamble. This block only takes syncwords of a maximum of 32 bits; therefore, it can only sync to half of the preamble. For this thesis, this block was not modified, and the second part of the preamble ”check” is made through the custom headerPrinter block. This block also handles the discarding of the second PDU received from the Sync and Create PDU block, which syncs to the second instance of the preamble in a frame.

6.2 Header

For demodulating, decoding and parsing the header, a custom block was created. This block takes in PDUs that were created by the Sync and Create PDU block.

The header is modulated with DBPSK - a phase difference of 0° equaling to the bit 0 and 180° equaling to the bit 1. This decoding is handled in the custom block.

The payload bits are also FEC coded using rate $n = 1/2$ convolutional coding, with the octal polynoms $g_0 = 753, g_1 = 561$ and a constraint length of $k = 9$. At the time of writing, there was no block available in the GNURadio Companion or any third party extension for decoding this coding scheme. The Python module `scikit-dsp-comm` [29] was used in the custom block to perform Viterbi decoding with the correct polynomials and the constraint length.

After decoding, the bits are then parsed according to Table 6.1.

Bits	Data
00 - 08	Number of valid bytes supplied by user.
09 - 11	Identifier for FEC used for this packet. See Table 3.1 for details.
12	Type of data in the next field. Can be 0 or 1.
13 - 20	Information field. If the bit 12 was 0, this field consists of the temperature of the transmitter. If the bit 12 was 1, this field consists of an incrementing frame counter.
21 - 24	4-bit CRC calculated with the polynom 0x1B.
25 - 32	Tailbits used for convolutional coding.

Table 6.1: Data contained within the header bits of a HISPICO frame.

This parsed data is printed to the console for a visual guide. The incrementing frame counter can be used to see if any packets were dropped or not received.

The number of valid bytes in each packet is then used to trim the payload. It must be noted that the number of valid bytes in the header do not include the empty symbols. Therefore, for a payload exceeding 232 bytes two additional empty symbols must be counted in the total size of the payload block. Another two empty symbols must be counted for payloads exceeding 511 bytes. The resulting payload after the trimming of dummy data is then passed into the next block.

6.3 Payload

The payload is modulated with DQPSK and always consists of 4096 bits. These bits are filled with the user data according to the FEC rate chosen and the clock rate of the communication. The remaining bits are filled with dummy data. The payload bits are extracted according to the information in the header and passed into the payload handler block.

The payload block includes 4 empty symbols; two in the middle of the frame at indexes 1120 and 1121, and towards the end of the frame at indexes 2242 and 2243. These empty symbols must be removed from the payload before the differential decoding occurs.

The differential decoding is done by iterating over the payload symbols and using the current symbol and the next symbol, which are transformed to their phase degree counterparts according to Table 6.2, then subtracted. The resulting phase difference is then transformed back to symbols, once again according to 6.2. For the first symbol, an initial pre-start symbol with a phase of 180°must be used to calculate the phase difference and the resulting initial symbol. After the decoding is complete, the symbols are packed into bytes and a PDU is created and output.

The PDU is then saved to a file by converting it back to a tagged stream and input into the file sink block. As the originally transmitted payload packets (or FEC packets encapsulating these payload packets, if FEC was enabled) might be divided between HISPICO frames, concatenating all the HISPICO frame payload sections ensures the rejoining of these divided packets.

Phase	Symbol
0°	00
90°	01
180°	11
270°	10

Table 6.2: The DQPSK constellation used in HISPICO.

Forward Error Correction

If no FEC is used, the reception ends in the previous step (6.3). If FEC is used, two additional blocks are used. The custom block for convolutional code decoding uses the same Viterbi decoder from `scikit-dsp-comm`. For Reed-Solomon decoding, since it uses the CCSDS standard, there is a block available for it in `gr-satellites`.

The input to these blocks must be a complete encoded packet. Therefore the output of payload handler can not be directly piped in, as each HISPICO frame might not have a full packet, and it might have been split between frames. It is also important to know which encoding algorithm was used. For a packet that is 223 bytes pre-encoding, the amount of bytes to decode differs for each of them - 448 for CC and 255 for RS. The output is piped to a file, in which the concatenation and joining of split CAM packets are handled. The full flowchart of the FEC decoding can be seen in Fig. 6.3.

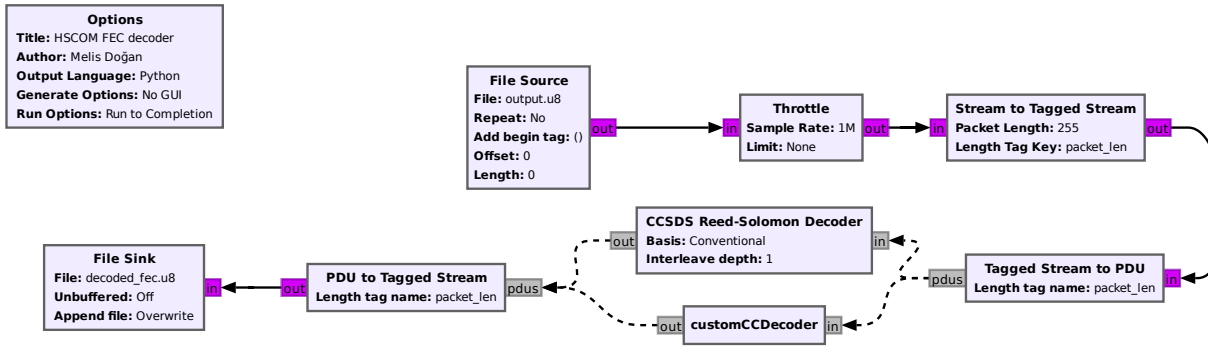

Figure 6.3: The GNURadio Companion Flowchart for decoding of FEC in ESTCube-2 HSCOM.

7 Testing

At the time of testing the HSCOM, no payloads were available to connect to the payload bus. The HSCOM firmware was used to simulate the payloads by re-mapping the ICP bus pins and connecting them to the payload bus pins. This way, the HSCOM MCU could send data to its own payload pins, which were then received and handled by the payload handler. It should be noted that for testing transmission through ICP commands, the ICP pins must be mapped to their original purpose.

The system was initially tested end-to-end by transmitting known sequences of bytes of varying lengths. Upon successful transmission and receiving of the bytes, the system was tested further by using different FEC and by varying the SNR levels. This was achieved by adding simulated Additive White Gaussian Noise (AWGN) to the recorded transmissions. The AWGN is a random noise model and can simulate random errors and noise sources that would normally occur during satellite communications.

The reception was modified for low SNR conditions. Low SNR leads to high BER, which causes the reception pipeline to miss preambles with high errors or produce faulty data from the frame headers, leading to the system extracting the incorrect amount of bytes of data from the payload. While this is expected behavior from the reception, for the purposes of testing, the preamble syncing and the extraction of payload bytes from the frame were done by a separate script to ensure even the frames with high BER were included in the final data file. For FEC testing, further corrections were made to the CCSDS Reed-Solomon decoder block from the gr-satellites library. Originally, the block did not output the packets it could not decode due to the amount of errors exceeding its maximum error correction capability. For fair analysis, the block was modified to output all packets.

The results of the testing, created with MATLAB's $berfit$ command, can be seen in Fig. 7.1

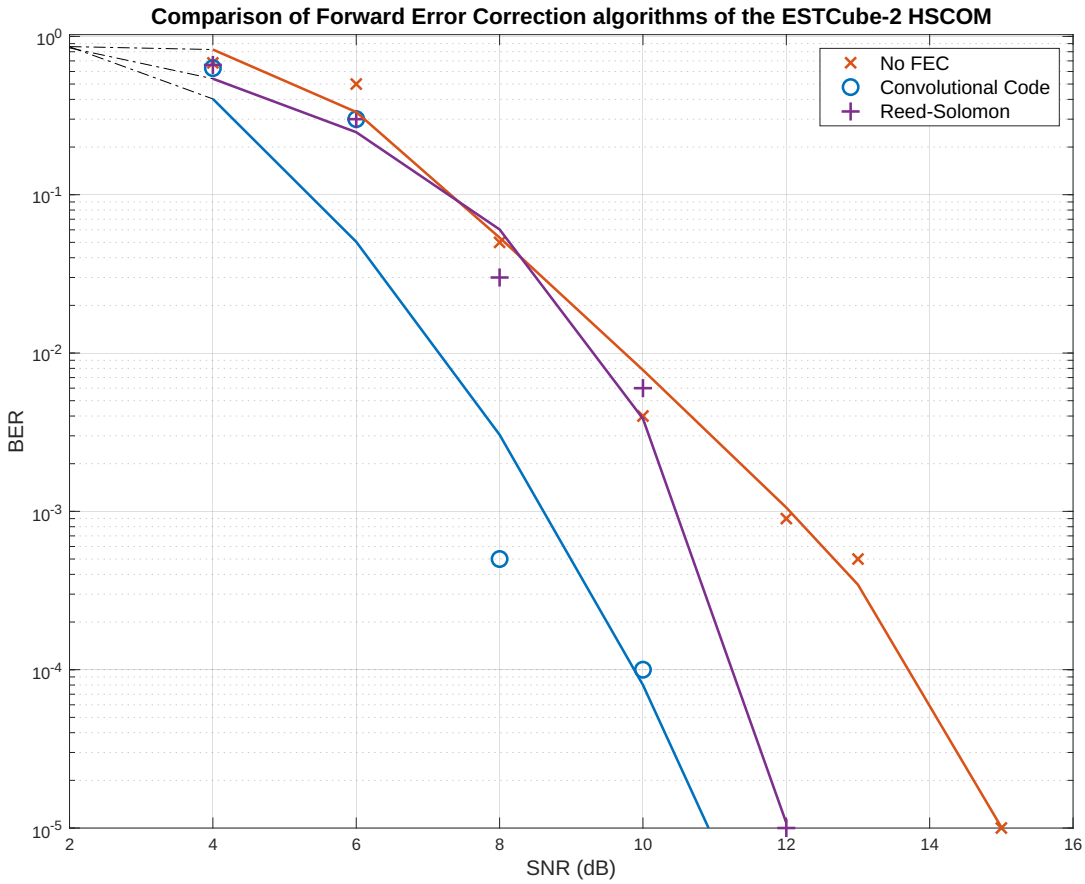

Figure 7.1: Comparison of Forward Error Correction algorithms of the ESTCube-2 HSCOM.

The transmission without using any FEC leads to errors in higher SNR levels than the FEC-coded transmissions. While Reed-Solomon initially performs better than transmissions without FEC, as the errors increase pre-decoding, its performance decreases. This is because Reed-Solomon codes perform better when correcting burst errors. It has the capability of correcting up to 16 bytes of errors with a block size of 255 bytes. The error correction capability of 16 bytes is regardless of how many bits are corrupted in a byte. Whether every bit is corrupted in a byte or a single one, it counts as one byte of error - in the best-case scenario, $16 * 8 = 128$ bits can be corrected. Therefore, when a burst error corrupts successive bits, they can be pooled into one single error in a byte. However, random errors are not successive and can cause a single bit error in 16 different bytes in a worst-case scenario, leading to a maximum of 16 bits being corrected. For this reason, while Reed-Solomon codes are useful in high SNR conditions due to their higher data rate compared to convolutional codes, they perform similarly to no-FEC in low SNR conditions.

In comparison, convolutional codes perform better than both the Reed-Solomon and the no-FEC transmissions, due to their strength in correcting random errors as opposed to burst errors. However, as a trade-off, they lower the data rate by more than half, and do not perform well for burst errors, which were not tested here.

There are options for increasing the error correcting capabilities of the system, such as interleaving the symbols to translate burst errors into random errors, or using concatenated codes by wrapping convolutional code in a Reed-Solomon code. However, the HISPICO already offers TURBO codes, which perform better than concatenated codes and offer better data rates. They are also on a dedicated hardware chip inside the transmitter, allowing the encoding to occur more efficiently than implementing it through software. Therefore, for the purposes of this thesis, any further implementations of FEC are outside of the scope.

In addition to BER testing, an image was stored on board the HSCOM MCU to test large image downlinking. The MCU had limited storage available for an image file, as it is not meant to store files in normal operation. An image with a size of 76.8 kB was stored on board and transmitted without any FEC, with an SPI clock rate of 1.06 MHz. The reception spanned over 318 HISPICO frames with 242 bytes per frame, taking a total of 0.6 seconds. This leads to a data rate of 1.02 Mbps. The manufacturers were contacted regarding the documentation of the HISPICO system, which notes that the SPI communications must not exceed a clock rate of 1.06 MHz. When using a clock rate of 1.06 MHz, the maximum data rate one can achieve is only 1.02 Mbps, which is below the advertised data rate of 1.6 Mbps. Increasing the clock rate of the SPI communication leads to the HISPICO frames reporting wrong user data bytes present in the payload, or dropping of bytes altogether. However, no response from the manufacturers was received at the time of writing.

8 Conclusion

The goal of this master's thesis was to integrate a high-speed transmitter into ESTCube-2. The work covered both the onboard firmware for controlling the transmitter, and the receiving pipeline for the GS. In total, the practical section of the work presented in this thesis took a year, partly due to delays in communicating with the manufacturer to receive necessary information for using HISPICO, such as the HISPICO frame specifications datasheet.

To achieve the goal, the following tasks were completed:

- Creation of onboard firmware for controlling the functionality of the transmitter
- Creation of onboard firmware for high-level logic of payload data transmission
- Implementation of a fix in firmware for an issue caused by the hardware
- Building of a receiving pipeline in GNURadio Companion for the GS
- Implementation of two FEC algorithms to replace the hardware FEC present within the transmitter, to which ESTCube-2 was not given the permission to access
- Testing of end-to-end image transmission

In-orbit testing and validation could not be performed, as following the launch in October 2023, ESTCube-2 did not successfully detach from its deployer and is assumed to be destroyed.

The work done in this thesis is considered for use in future satellites built by the Estonian Student Satellite Foundation. However, for future work, the author strongly suggests either gaining access to the functionality of HISPICO, which were restricted by the manufacturers by not sharing relevant information, or building another transmitter in-house to replace HISPICO.

Acknowledgements

I would like to extend my endless gratitude to;

- everything that makes me happy, including but not limited to fruits, drinks, sights, smells, video games, the wind, the sea, the sun
- my friends, classmates and coworkers
- Daniel Estévez, for his blog and his work
- my thesis advisors, for their unbelievably vast experience and wisdom
- my parents, for everything, and for calling me to ask if ESA would let them go to the moon if they sit real quiet in the spacecraft
- my sister, the graphic designer and the funniest person I know, I did not use any generative AI in this thesis to ensure your future job security
- Kristo Allaje, I do not think anybody can be a better coworker, boss, advisor, team leader, or most of all, friend, than you
- my partner, I love you

Bibliography

[1] CubeSat Design Specification Rev 14. 2022. https://www.cubesat.org/s/CDS-REV14_1-2022-02-09.pdf

[2] NASA, *State-of-the-Art of Small Spacecraft Technology, Communications*, https://www.nasa.gov/smallsat-institute/sst-soa/communications. Accessed May 20, 2024.

[3] A. Zedaan and T. Khattab, *A Critical Review of Baseband Architectures for CubeSats Communication Systems*, 2022, DOI:10.48550/arXiv.2201.09748.

[4] Aalto University, *Contract signed for Aalto-1 satellite launch.*, https://www.aalto.fi/en/news/contract-signed-for-aalto-1-satellite-launch. Accessed May 20, 2024.

[5] Aalto University, *Aalto-1 was launched into space a year ago – the Otaniemi ground station is already being prepared for the launch of the next satellites*, https://www.aalto.fi/en/news/aalto-1-was-launched-into-space-a-year-ago-the-otaniemi-ground-station-is-already-being. Accessed May 20, 2024.

[6] J. Praks, M.R. Mughal, R. Vainio, P. Janhunen, J. Envall, P. Oleynik, A. Näsilä, H. Leppinen, P. Niemelä, A. Slavinskis, J. Gieseler, P. Toivanen, T. Tikka, T. Peltola, A. Bosser, G. Schwarzkopf, N. Jovanovic, B. Riwanto, A. Kestilä, A. Punkkinen, R. Punkkinen, H.-P. Hedman, T. Säntti, J.-O. Lill, J.M.K. Slotte, H. Kettunen, A. Virtanen, *Aalto-1, multi-payload CubeSat: Design, integration and launch*, *Acta Astronautica*, 2021 DOI:https://doi.org/10.1016/j.actaastro.2020.11.042.

[7] I. Iakubivskyi, P. Janhunen, J. Praks, V. Allik, K. Bussov, B. Clayhills, J. Dalbins, T. Eenmäe, H. Ehrpais, J. Envall, S. Haslam, E. Ilbis, N. Jovanovic, E. Kilpua, J. Kivastik, J. Laks, P. Laufer, M. Merisalu, M. Meskanen, R. Märk, A. Nath, P. Niemelä, M. Noorma, M. R. Mughal, S. Nyman, M. Pajusalu, M. Palmroth, A. S. Paul, T. Peltola, M. Plans, J. Polkko, Q. S. Islam, A. Reinart, B. Riwanto, V. Sammelselg, J. Sate, I. Sünter, M. Tajmar, E. Tanskanen, H. Teras, P. Toivanen, R. Vainio, M. Väänänen, and A. Slavinskis, *Coulomb drag propulsion experiments of ESTCube-2 and FORESAIL-1*, *Acta Astronautica*, vol. 177, pp. 771–783, 2020. DOI:https://doi.org/10.1016/j.actaastro.2019.11.030.

[8] J. Praks, P. Niemela, A. Nasila, A. Kestila, N. Jovanovic, B. Riwanto, T. Tikka, H. Leppinen, R. Vainio, P. Janhunen, *Miniature spectral imager in-orbit demonstration results from Aalto-1 Nanosatellite Mission*, IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium, Jul. 2018. DOI:https://doi.org/10.1109/igarss.2018.8517658

[9] J. Jussila, *S-band transmitter for Aalto-1 nanosatellite*, Master's Thesis, Aalto University, 2013, URL:https://aaltodoc.aalto.fi/handle/123456789/10453.

[10] Texas Instruments, *MSP430F22x2, MSP430F22x4 Mixed Signal Microcontroller datasheet (Rev. G)*, SLAS504G, July 2006 [Revised Aug. 2012].

[11] Texas Instruments, *Low-Cost Low-Power 2.4 GHz RF Transceiver datasheet (Rev. C)*, SWRS040C, Jan. 2005 [Revised May 2008].

[12] Aalto University, *Radio Amateurs, S-Band Science Data Downlink*. https://wiki.aalto.fi/display/SuomiSAT/Radio+Amateurs. Accessed May 20, 2024.

[13] Texas Instruments, *EV76C664 Datasheet 1.3 Mpixels Monochrome and Sparse CMOS Image sensor*, Dec. 2016.

[14] M. Barthelemy, E. Robert, V. Kalegaev, V. Grennerat, T. Sequies, G. Bourdarot, E. Le Coarer, J.-J. Correia, and P. Rabou, *AMICal Sat: A sparse RGB imager on board a 2U CubeSat to study the Aurora*, *IEEE Journal on Miniaturization for Air and Space Systems*, vol. 3, no. 2, pp. 36–46, Jun. 2022. DOI:https://doi.org/10.1109/JMASS.2022.3187147.

[15] Université Grenoble alpes, *AMICal Sat Project*, https://www.csug.fr/projects/amical-sat-project/. Accessed May 20, 2024.

[16] AMICal Sat Downlink Technical Information. Rev 0.4. 2020. http://amicalsat.univ-grenoble-alpes.fr/assets/AmicalSat%20downlinks%20technicals%20informations%20v0.4-ba04019f31330c558c47f6618551195fdd10cb43273a4a9715fd8977feefaea1.pdf

[17] D. Estévez, "*Decoding AMICal Sat in-orbit images*", https://destevez.net/2020/10/decoding-amical-sat-in-orbit-images/. Accessed May 20, 2024.

[18] B. Mero, K. A. Quillien, M. McRobb, S. Chesi, R. Marshall, A. Gow, C. Clark, M. Anciaux, P. Cardoen, J. De Keyser, Ph. Demoulin, D. Fussen, D. Pieroux, S. Ranvier, *PICASSO: A State of the Art CubeSat*, *Proceedings of the 29th Annual AIAA/USU Conference on Small Satellites*, Logan, Utah, USA, August 8-13, 2015. http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=3179&context=smallsat

[19] D. Estévez, "*K2SAT S-band image receiver*", https://destevez.net/2018/07/k2sat-s-band-image-receiver/. Accessed May 20, 2024.

[20] IQ Spacecom, *Highly Integrated S-Band Transmitter for Pico and Nano Satellites*, https://confluence.tudengisatelliit.ut.ee:8433/display/ESTELLECOM/HSCOM?preview=/45974914/515 Accessed May 20, 2024.

[21] STMicroelectronics, *STM32L462CE*, https://www.st.com/en/microcontrollers-microprocessors/stm32l462ce.html. Accessed May 20, 2024.

[22] STMicroelectronics, *STM32F7 - SPI - Serial Peripheral Interface*, https://www.st.com/content/ccc/resource/training/technical/product_training/group0/3e/ee/cd/b7/84/4b/4 Accessed May 20, 2024.

[23] Rogers Corporation, *RO4000® Series High Frequency Circuit Materials*, https://www.rf-microwave.com/resources/products_attachments/5e38437d990bf.pdf. Accessed May 20, 2024.

[24] J. Dalbins, K. Allaje, H. Ehrpais, I. Iakubivskyi, E. Ilbis, P. Janhunen, J. Kivastik, M. Merisalu, M. Noorma, M. Pajusalu, I. Sünter, A. Tamm, H. Teras, P. Toivanen, B. Segret, A. Slavinskis, *Interplanetary student nanospacecraft: Development of the LEO demonstrator ESTCube-2*, Aerospace. 10 (2023) 503. DOI:https://doi.org/10.3390/aerospace10060503.

[25] Amazon Web Services, *The FreeRTOS™ Kernel*, https://www.freertos.org/RTOS.html. Accessed May 20, 2024.

[26] CCSDS, TM Synchronization and Channel Coding Blue Book 131.0-B-5, Sept. 2023. https://public.ccsds.org/Pubs/131x0b5.pdf

[27] P. Karn, *DSP and FEC Library*, http://www.ka9q.net/code/fec. Accessed May 20, 2024.

[28] D. Estévez, *gr-satellites*, https://github.com/daniestevez/gr-satellites. Accessed May 20, 2024.

[29] M. Wickert, *scikit-dsp-comm*, https://github.com/mwickert/scikit-dsp-comm. Accessed May 20, 2024.

Non-exclusive licence to reproduce thesis and make thesis public

I, Melis Doğan

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   "Integration of a High Speed Communications System into ESTCube-2"

   supervised by Kristo Allaje, Viljo Allik, and Tõnis Eenmäe.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Melis Doğan
20.05.2024