Downlink System Design for the STRATHcube Satellite Mission

04 April 2025

Supervisor: Louise Crockett
Department of Electronic and Electrical Engineering
University of Strathclyde, Glasgow

Daniel Stebbings
reg. 202118874

I hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Louise Crockett.

Abstract

Efficient downlink of recorded telemetry is a critical challenge in CubeSat missions, constrained by power limitations, bandwidth restrictions, and dynamic channel conditions imposed during a ground station pass. Further, many CubeSats use the Ultra High Frequency (UHF) amateur band which introduces the further issue of in-band interference which cannot be accounted for in advance.

To maximise throughput under these conditions, Adaptive Coding and Modulation (ACM) can be used. Digital Video Broadcasting – Satellite Second Generation (DVB-S2) is one such ACM system, with near Shannon limit performance and a modular standard allowing it to be matched to the use case. Despite this, there are no space rated UHF DVB-S2 transmitters currently on the market; this necessitates its implementation on a Software Defined Radio (SDR). While integrating an SDR can be costly and complex for many missions, the design of STRATHcube already includes one, minimising additional expenses.

STRATHcube is a student led 2U CubeSat mission at the University of Strathclyde, the primary payload being a technology demonstrator of a Passive Bistatic Radar (PBR) using an SDR and communicating in the UHF band. Consequently, there will already be a powerful System on Chip (SoC) Field Programmable Gate Array (FPGA) based SDR included on the mission. This allows for a DVB-S2 transmitter to be implemented as a piggy-back on the primary payload using spare resources.

This dissertation outlines the downlink system design for STRATHcube. Performance and link analysis was conducted, analysing communication windows over the course of the mission and expected theoretical performance. A high level architecture of the system was created and the design implemented in hardware using MathWorks HDL Coder. Work also began on a Generic Stream Encapsulation (GSE) implementation using C++.

Initial performance analysis was conducted in comparison to a baseline system which uses Constant Coding and Modulation (CCM) and optimises for maximum availability and showed a significant uplift in data throughput over the course of the mission. Additionally, resource analysis of the target FPGA SoC and the synthesised design show that the implementation is feasible in hardware.

Acknowledgements

Thanks are given to the sponsors of the STRATHcube, without whom this project would not be possible: The University of Strathclyde Alumni Fund, the Institute of Mechanical Engineers, the Royal Aeronautical Society, the University of Strathclyde Mechanical and Aerospace Engineering Department, and the University of Strathclyde Aerospace Centre of Excellence.

Contents

List of Figures

Definitions

ACM: Adaptive Coding and Modulation
ADC: Analog to Digital Converter
APSK: Amplitude and Phase Shift Keying
AWGN: Additive White Gaussian Noise
BBFRAME: Baseband Frame
BBHEADER: Base Band Header
BCH: Bose-Chaudhuri-Hocquenghem
BER: Bit Error Rate
BPSK: Binary Phase Shift Keying
CAN: Controller Area Network
CCDD: Core Flight System (CFS) Command and Data Dictionary
CCM: Constant Coding and Modulation
CCSDS: Consultative Committee for Space Data Systems
CNR: Carrier to Noise Ratio
COTS: Commercial Off-The-Shelf
CRC-8: 8 bit Cyclic Redundancy Check
DAC: Digital to Analog Converter
DFL: Data Field Length
DVB-S2: Digital Video Broadcasting - Satellite - Second Generation
EDHPC: European Data Handling & Data Processing Conference
EIRP: Effective Isotropic Radiated Power
ESA: European Space Agency
ETSI: European Telecommunications Standards Institute
FE: Front End
FEC: Forward Error Correction
FECFRAME: Forward Error Correction Frame
FIFO: First In First Out
FMC: FPGA Mezzanine Card
FPGA: Field Programmable Gate Array
FYS: Fly Your Satellite
GSE: Generic Stream Encapsulation
I2C: Inter-Integrated Circuit
IP: Internet Protocol
IPC: Inter-Process Communication
IQ: In-phase and Quadrature
ISM: Industrial, Scientific, and Medical
ISS: International Space Station
ITU: International Telecommunication Union
LDPC: Low-Density Parity-Check
LSB: Least Significant Bit
LVDS: Low Voltage Differential Signalling
MATYPE: Mode Adaption Type
MODCOD: MODulation and CODing rate
MSB: Most Significant Bit
OBC: On Board Computer
PBR: Passive Bistatic Radar
PDR: Preliminary Design Review
PDU: Protocol Data Unit
PL: Programmable Logic
PLFRAME: Physical Layer Frame
PS: Processing System
RAM: Random Access Memory
RF: Radio Frequency
RTL: Register Transfer Level
SDR: Software Defined Radio
SNR: Signal to Noise Ratio
SPI: Serial Peripheral Interface
STAC: Spacecraft Tracking & Command Station
SoC: System on Chip
TBC: To Be Confirmed
TLE: Two-Line Element set
TS: Transport Stream
TT&C: Telemetry, Tracking and Command
UART: Universal Asynchronous Receiver-Transmitter
UHF: Ultra High Frequency
UPL: User Packet Length
VCM: Variable Coding and Modulation
VHF: Very High Frequency

VL-SNR: Very Low Signal-to-Noise Ratio
XTCE: XML Telemetric and Command Exchange

1 Introduction

1.1 CubeSat Communications

A typical ground station pass is depicted in Figure 1, showing some of the sources of attenuation and interference. These factors are more impactful in the Ultra High Frequency (UHF) amateur satellite allocation, as this band is shared between several satellites, and is adjacent to a license free Industrial, Scientific, and Medical (ISM) allocation used by devices such as car keys. Due to the small size of CubeSats, high power transmission and large high gain antennas are infeasible which means that the received signal power at the ground station is relatively low and more vulnerable to the effects of interference.

To mitigate these issues, many large satellites requiring high data throughput modify their modulation and coding settings over time, in a process called Variable Coding and Modulation (VCM), where this is pre-planned, or Adaptive Coding and Modulation (ACM), where this is based on real time measurements of channel conditions. This allows much higher throughput for a given bandwidth, which is particularly relevant in the constrained amateur allocations.

Although ACM/VCM has a high performance improvement, there are few examples of CubeSats using these systems in the UHF band, due to the high complexity required and the fact that high rate communications are typically conducted in the S band or higher, as higher bandwidths are available. For this reason, any implementation for STRATHcube must be developed using a Software Defined Radio (SDR) which increases cost and time requirements.

One system for high rate satellite data transmission is Digital Video Broadcasting - Satellite - Second Generation (DVB-S2). Originally designed for the transmission of video data, it has since been expanded to support generic data streams. This system is typically used on high throughput satellites operating on bandwidths of multiple megahertz at frequencies in the tens of gigahertz, for which ACM and VCM are particularly important due to atmospheric effects. For this reason, the standard includes support for Constant Coding and Modulation (CCM), VCM and ACM. Additionally, it has a highly modular structure allowing it to be adapted for a given application.

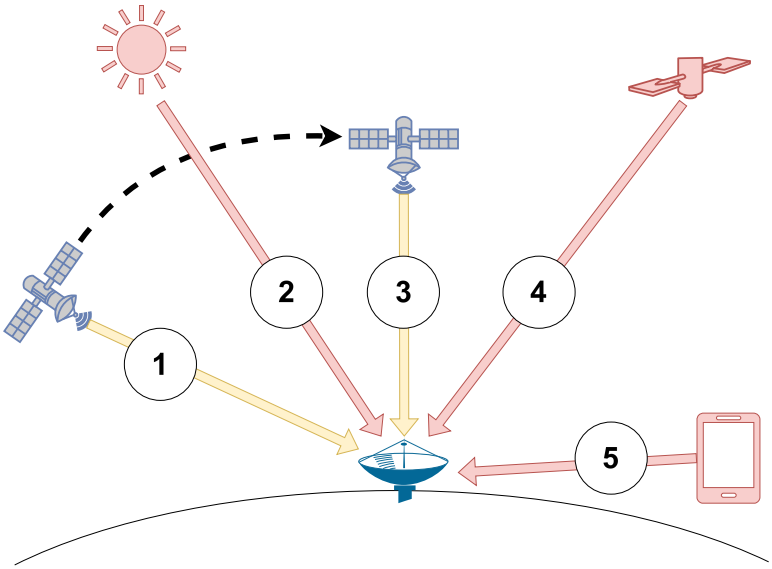


Figure 1: Factors affecting a ground station pass. At (1) the satellite is at a low elevation and, due to its circular orbit, will be at its longest range and the highest attenuation from the atmosphere. When the satellite is directly overhead, (3), the distance is at minimum and the signal strength at maximum. There are also various sources of interference, such as from the Sun and galactic noise, (2), other satellites in the same band, (4), or terrestrial devices, (5).

1.2 STRATHcube

1.2.1 Mission Overview

STRATHcube is a student-led 2U CubeSat mission in development at the University of Strathclyde as part of the European Space Agency's European Space Agency (ESA) Fly Your Satellite! and Design Booster programs. The mission addresses critical aspects of space sustainability through its two payloads.

The primary payload involves the demonstration of a Passive Bistatic Radar (PBR) system for in-orbit space debris detection. This payload operates by measuring the signal power of Iridium satellite transmissions in order to detect the attenuation caused by passing debris. The acquired data requires ground-based processing, necessitating reliable high-speed downlink capabilities. A key operational constraint arises from the substantial volume of data generated during each observation pass. While onboard data compression mitigates this to some extent, the downlink speed remains the limiting factor governing the payload's operational time.

The secondary payload focuses on characterising aerothermal effects during atmospheric re-entry, specifically investigating the conditions leading to solar panel fragmentation. This payload becomes operational in the hours before the demise of the satellite and relies on the Iridium satellite network for data transmission.

However, this secondary communication channel operates at significantly lower data rates compared to the primary UHF system and incurs additional operational costs.

A render of the satellite is shown in Figure 2, although this has a slightly outdated model.

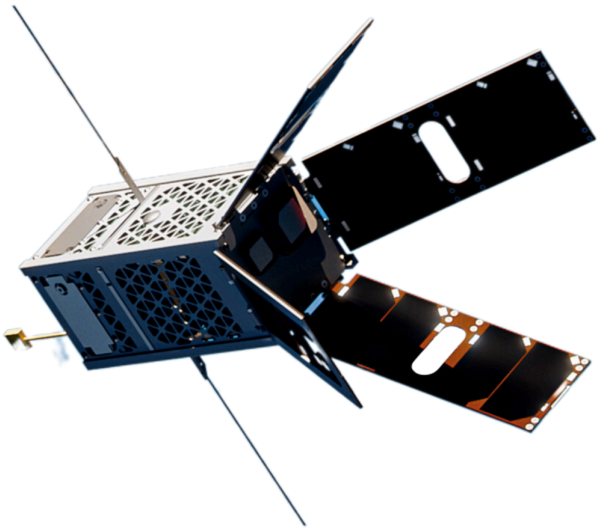


Figure 2: Render of STRATHcube in re-entry configuration. The dipole antenna for UHF communications is deployed in the center.

1.2.2 Communication Architecture

The primary communication system will operate in the amateur space service UHF allocation which spans 3 MHz, from 435 to 438 MHz. The ground station implementation is currently under investigation, with options including the use of the currently non-operational Spacecraft Tracking & Command Station (STAC) or the use of existing ground stations elsewhere. Both uplink and downlink communications will use the onboard TOTEM SDR.

The secondary communication system will communicate through the Iridium satellite constellation, as re-entry may occur outside of the range of any ground station. Additionally, ionisation effects create a communications black out zone that prevents communication from the spacecraft directly to the ground.

1.2.3 Mission Phases

The mission timeline comprises four distinct operational phases. The Early Operations and Commissioning Phase begins immediately following deployment from the International Space Station (ISS) at an initial altitude of approximately 415 kilometers. This critical 10-day period involves antenna deployment, system activation, and establishment of communications with the ground station.

The primary phase occupies the majority of the mission, during which the satellite alternates between PBR measurement collection and dedicated downlink periods using the primary UHF communication system. This phase continues until the satellite reaches the altitude trigger of 170 kilometers, expected to occur around 180 days post-deployment and initiating the Transition Phase.

During the Transition Phase, the satellite undergoes reconfiguration to prepare for atmospheric re-entry. Finally, the Secondary Phase encompasses the re-entry process itself, where the secondary payload actively records sensor data and transmits available measurements through the Iridium network until communication becomes impossible and the demise of the satellite.

These phases are shown graphically in Figure 3 [1, fig. 3].

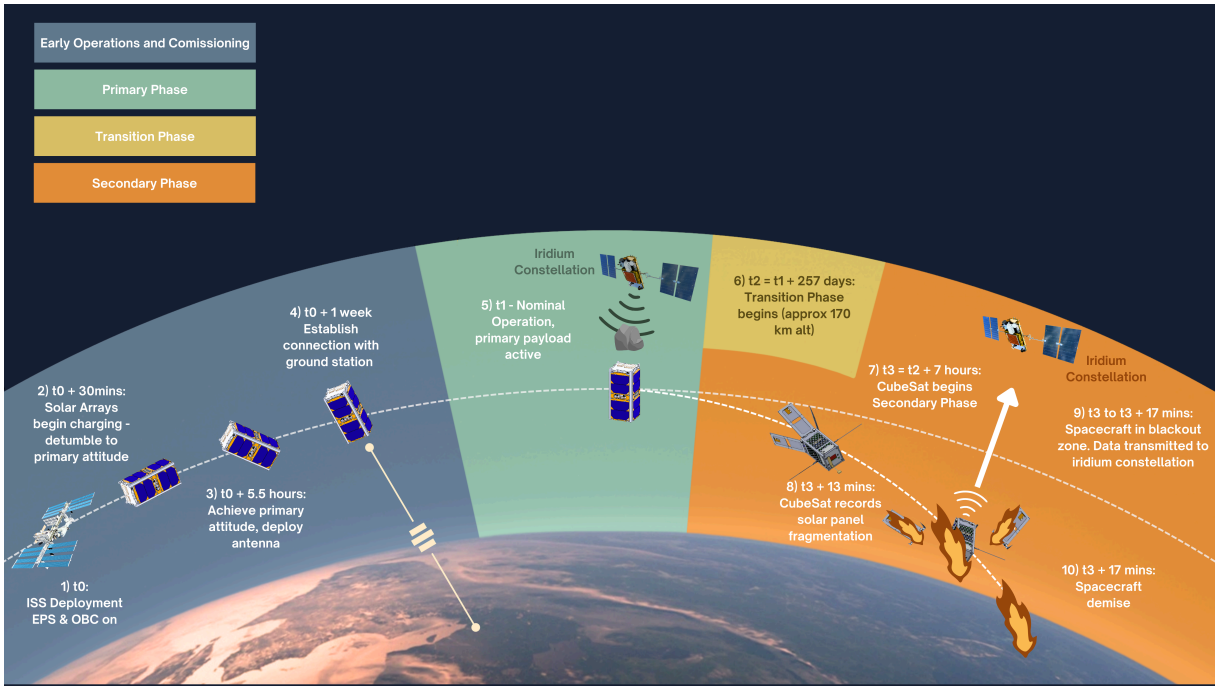


Figure 3: STRATHcube mission phases.

1.3 Objectives

The project aimed to advance the design of the downlink communications system for STRATHcube while actively contributing to design reviews. Key objectives included:

1. Developing a detailed system design for downlink communications.
2. Building an engineering model of the downlink system using development boards for validation.

Given the complexity of the STRATHcube mission, the project scope was deliberately constrained to prevent “scope-creep”. The focus was solely on downlink communications, deferring uplink system design for future work. Further, implementation on hardware was reserved as an optional stretch goal.

To maintain flexibility for later development, the implementation was designed to be modular, preventing unnecessary constraints on subsequent development. This approach ensured that progress could be made efficiently while laying a strong foundation for future enhancements.

2 Literature Review

2.1 STRATHcube

Sporadic work has been completed for the downlink communications system in the previous years of the project.

In [2] a tradeoff study was conducted of the communications system. It identified the UHF amateur allocation in 435 - 438 MHz as suitable for uplink and downlink. A preliminary link budget was conducted using information about the STAC ground station and a radio module was selected.

The first radio selected was the GAUSS Low Power UHF Radio which can use various frequency shift keying modes and convolutional codes for error correction. This has a stated max capacity of 250kbps, but per the datasheet this rating is not currently supported leaving the actual maximum rate of 100kbps. Additionally, the datasheet also states that Reed Solomon coding is supported, whilst also stating that it is not *currently* supported in a footnote. Additionally, the coding rates for which the datarate values are calculated are not available. [3]

In [4] another tradeoff study was conducted for the primary payload system, ultimately selecting the TOTEM SDR from Alén Space. [5] Alén Space also manufacture a UHF Front End (FE) compatible with the TOTEM SDR, [6] for this reason it was decided to consolidate both the primary telecommunications and primary payload onto the TOTEM SDR in order to reduce component count and project cost. The full specification and operation of the TOTEM SDR is defined in Section 2.2.

UHF communications will be transmitted using the ISIS deployable antenna system [7] configured with a single deployable dipole antenna. The exact specifications are selectable by the user, being rated for 10 MHz of usable bandwidth at design frequency and a VSWR < 1.9:1.

A member of the team completed a research internship analysing the downlink in summer 2024. [8] This analysis compared multiple types of modulation and FEC, ultimately recommending DVB-S2 for the final system. It also analysed the maximum bandwidth for the link to close, i.e. to have a positive link margin, arriving at a figure of 158.5 kHz. The resulting link budget is shown in Table 1. With the ground station undefined, many of the performance figures were based on the link budget created by AcubeSAT, [9] an open source satellite mission. The final link margin number is effectively the Carrier to Noise Ratio (CNR) above the margin requirement, so the actual margin would be 12 dB.

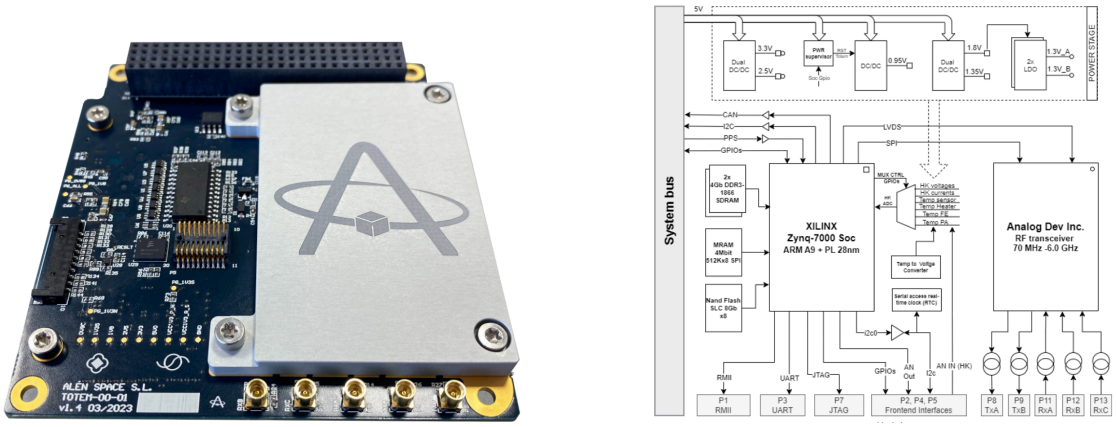
Table 1: Summer 2024 Link Budget

Spacecraft		
Transmit Power	1.76	dBW
Transmitter Gain	0	dBi
Transmitter Line Losses	−1.93	dB
Transmitter Antenna Pointing Loss	−3	dB
Effective Isotropic Radiated Power (EIRP)	−3.17	dBW
Link		
Frequency	435	MHz
Bandwidth (Max Achievable)	158.5	kHz
Path Length (409 km Altitude, 15° Elevation)	1196.622	km
Free Space Path Loss	−146.777	dB
Other Losses (Atmospheric, Polarisation, ...)	−4.6	dB
Ground Station		
Received Incident Power	−154.61	dBW
Receiver Gain	12	dB
Receiver Line Losses	−2.4	dB
Receiver System Noise Temperature	249.3	K
Receiver Sensitivity	−11.97	dB-K
Implementation Loss	−2	dB
Received CN0R	62.03	dB
Received CNR	12.025	dB
Required CNR	−10	dB
Link Margin	2.025	dB

The ISIS space antenna VSWR and Table 1 contributed to the link analysis in Section 3.

2.2 TOTEM SDR

The TOTEM SDR consists of an AMD Zynq 7020 System on Chip (SoC) Field Programmable Gate Array (FPGA) connected to an AD9364 RF Transceiver via Low Voltage Differential Signalling (LVDS), as shown in Figure 4b. [5] Compatible frontend modules can be mounted on the motherboard using the connector in the center of the board, which can be seen in Figure 4a.



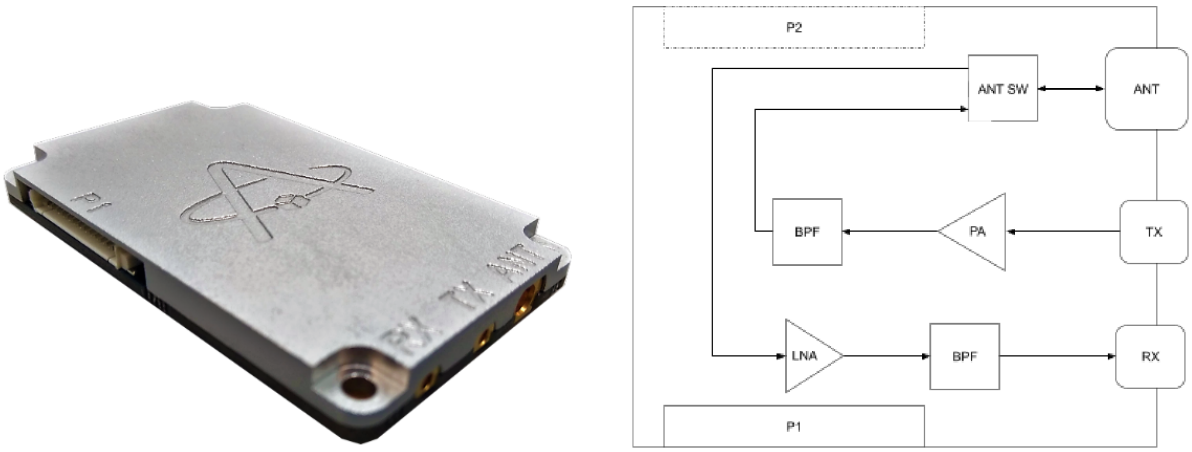
(a) Motherboard (b) Block Diagram

Figure 4: TOTEM SDR

The Zynq 7020 has dual ARM Cortex-A9 processor cores that form the Processing System (PS) and FPGA fabric that forms the Programmable Logic (PL). This allows improved performance, as serial operations such as network interfacing can be accomplished on the PS and highly parallel operations such as signal processing can be accomplished on the PL. The Zynq

The AD9364 is a high performance Radio Frequency (RF) transceiver with separate ports for receive and transmit. It can operate from 70 MHz to 6 GHz and has a bandwidth of up to 56 MHz with a 12 bit Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC). A Serial Peripheral Interface (SPI) bus is used for control signals and a high speed LVDS interface for In-phase and Quadrature (IQ) data. This device is part of the larger family of AD936x transceivers form Analog Devices. The chips in this family differ in port count, bandwidth, and operating frequency, however the control interfaces are very similar across devices. For this reason, many software drivers and FPGA Internet Protocol (IP) blocks work with multiple devices in the family. [10]

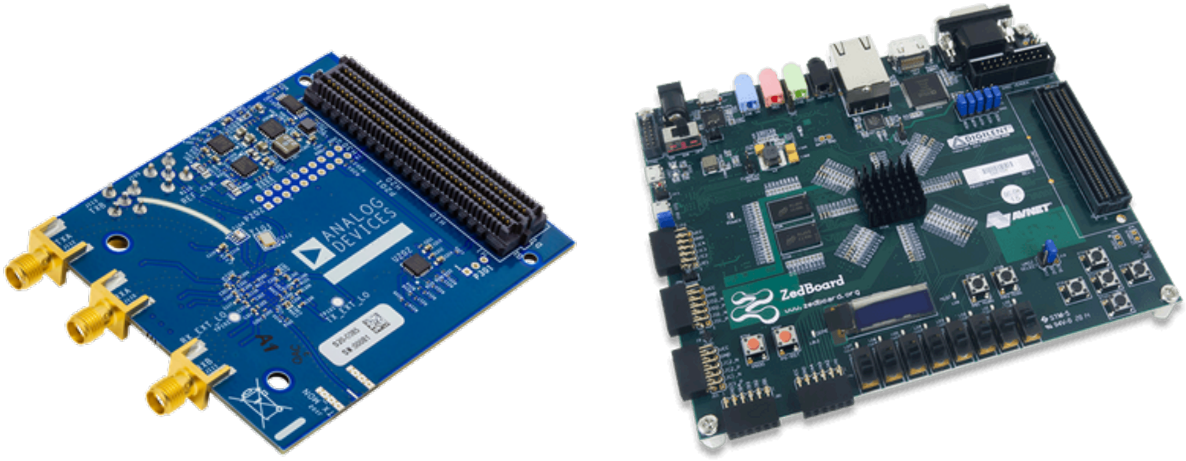
The frontend module is called FrontendUHF, and includes a transmit / receive switch, bandpass filtering and amplification for both transmit and recieve. It has a frequency range of 430 to 440 MHz and a typical power output of 30dBm. Figure 5 depicts the module and a block diagram of its operation. [6]



(a) FrontendUHF module (b) FrontendUHF Block Diagram

Figure 5: FrontendUHF

As the TOTEM SDR could not be purchased in time for the project, implementation was targeted towards development boards. The Digilent Zedboard is a Zynq 7020 development board with much less external Random Access Memory (RAM) than the TOTEM, however it has the same FPGA resources. The Zedboard also includes a FPGA Mezzanine Card (FMC)connector, allowing the connection of high speed peripherals. [11] This can be used to connect an FMCOMMS development board from Analog Devices, one of which is available for the AD9364 used in the TOTEM. As the control interfaces are the same, another FMCOMMS board from the AD936x family can be used for functional testing without changing the design. [12], [13]



(a) FMCOMMS4 (b) Zedboard

Figure 6: Development Boards

The interface of the TOTEM UHF described here contributed to the system design in Section 4, and the selected development boards used as the target for implementation in Section 5.

2.3 STAC Ground Station

The decision for which ground station will be used for communications has not been finalised, however one of the most likely options is the STAC ground station. It was first created in 2008 and resides on the roof of the James Weir building at the University of Strathlycde. After a fire in the building, the project was abandoned in 2012. In 2015/2016, a master's group renovated the STAC and were successful in receiving telemetry from the Strand-1 CubeSat. [14] The station has since been out of use and has not been maintained, as such, the current status of the components is unknown.

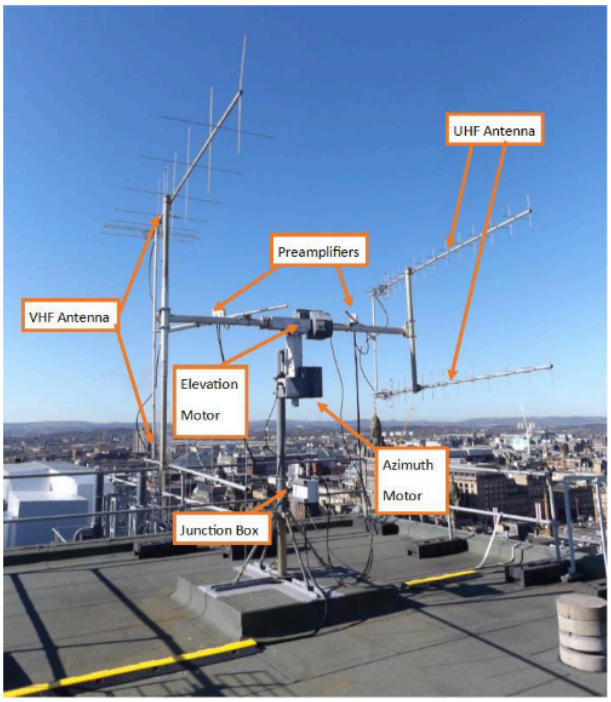


Figure 7: STAC After 2016 Renovation

The STAC has dual Very High Frequency (VHF) and UHF antennas and includes pre-amplifiers suitable for both receive and transmit. The UHF antennas are the 436CP30 from M2 Antenna Systems and the UHF pre-amplifier is the SP-7000 from SSB. The details of which are shown in Table 2 and Table 3 respectively. [15], [16].

Table 2: SP7000 Specification

Frequency Range	430-440 MHz
Noise Figure	0.9 dB
Maximum Gain	20 dB

Table 3: 436CP0 Antenna Specification

Frequency Range	432-440 MHz
Gain	15.50 dBic
Beamwidth	30 °
Max VSWR	1.6:1

As part of the 2015/2016 renovation, a link budget was created for reception of communications from the Strand-1 satellite. Table 4 shows the original numbers, with the "Unit" column subsequently added for context. [14]

Table 4: STAC Strand-1 UHF Downlink Link Budget

Parameter	20 Degrees	90 Degrees	Unit
Frequency	4.40E+08	4.40E+08	Hz
Speed	3.00E+08	3.00E+08	m/s
Wavelength	2.06	2.06	m
Transmitted Power	0.5	0.5	W
Transmitter Gain	1	1	dB
Line Loss	−3	−3	dB
EIRP	−1.5	−1.5	dBW
Path Length	3253000	811000	m
Space Loss	−136.5	−123.5	dB
Antenna Pointing Loss	−0.7	−0.7	dB
Polarization Loss	−0.3	−0.3	dB
Receiver Gain	10.2	10.2	dB
System Noise Temperature	19.03089987	19.03089987	K
Data Rate	9600	9600	bps
Boltzmann's Constant	228.6	228.6	dB
Bit Energy to Noise Ratio	16.88	28.95	dB
Miscellaneous	−25	−25	dB
Required Energy to Noise Ratio	10	10	dB
Margin	6.88	18.95	dB

The link budget here was used to update the link budget described in [8], contributing to the analysis in Section 3.

2.4 Standards

2.4.1 DVB-S2 Overview

The input to a DVB-S2 system is one or more datastreams. These can be transport streams, for transmission of video, or generic streams which are either packetised or continuous. A packetised stream has a constant packet size termed the User Packet Length (UPL). Variable packet sizes are sent via a continuous generic stream. These are depicted in Figure 8.

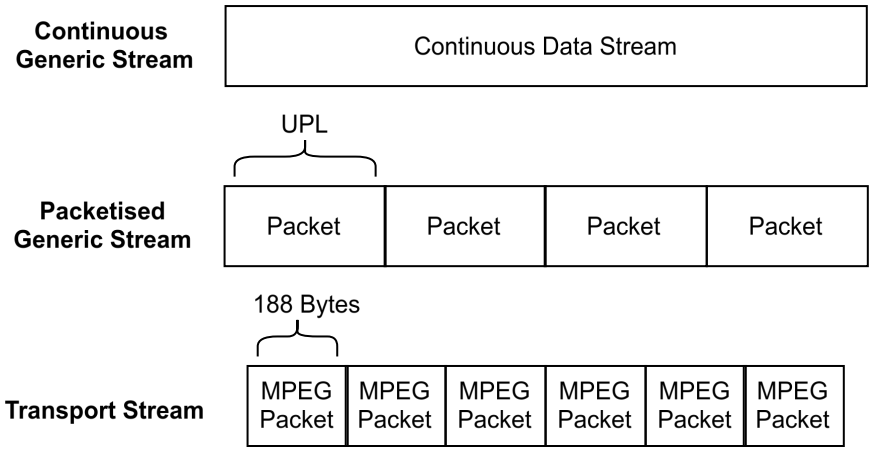


Figure 8: DVB-S2 Input Stream Types.

The input streams are sliced and merged to create a Baseband Frame (BBFRAME). The Structure of which is shown in Figure 10a. A Base Band Header (BBHEADER) is also applied, which contains various signalling fields providing information about the transmitted frame. The structure is shown in Figure 9. Contained within the Mode Adaption Type (MATYPE) field is the MODulation and CODing rate (MODCOD) field, which indicates the settings being used on the frame.

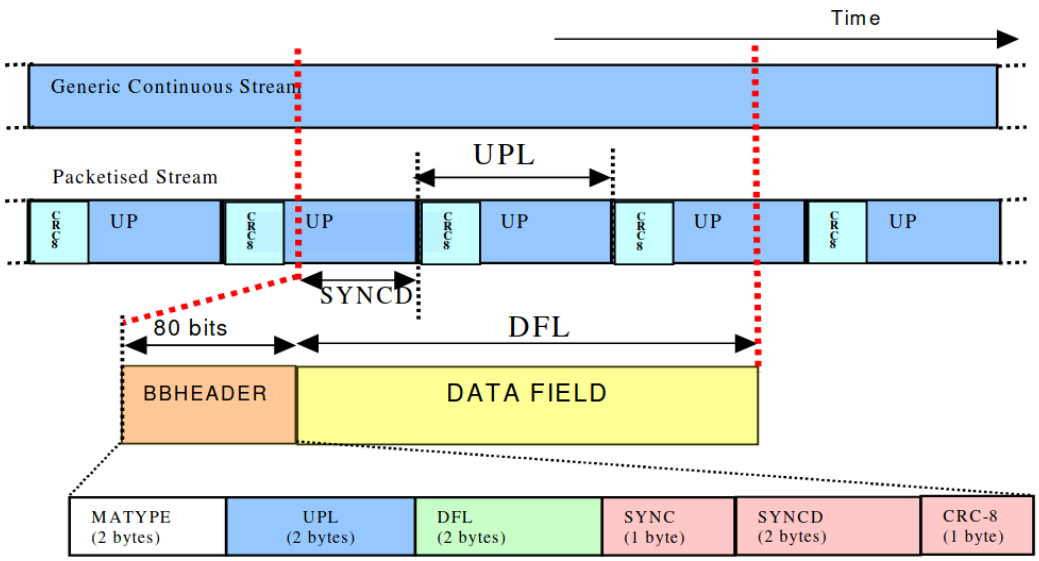


Figure 9: DVB-S2 BBHEADER Structure

DVB-S2 utilises two stages of FEC, the first code is Bose-Chaudhuri-Hocquenghem (BCH) followed by Low-Density Parity-Check (LDPC). This combination results in extremely strong error correction performance, allowing near Shannon limit performance. [17] The output of the FEC subsystem is a Forward Error Correction Frame (FECFRAME). This can either be short, 16200 bits, or normal, 64800 bits. This is shown in Figure 10b. The information content of the FECFRAME is determined by the coding rate, which can be from 1/4 to 9/10.

The FECFRAME is then mapped to symbols and modulated to become a Physical Layer Frame (PLFRAME). Pilots can optionally be inserted here to aid in decoding. Dummy

frames are PLFRAMEs that carry no information. DVB-S2 used Amplitude and Phase Shift Keying (APSK) modulation, configurable between QPSK to 8PSK, 16PSK and 32PSK.

The final block in the system is a root-raised-cosine filter for pulse shaping. This can have a roll-off factor of 0.35, 0.25 or 0.20. The value of which is indicated in the MATYPE field.

Mapping a packet into a dynamically changing frame is difficult. The Generic Stream Encapsulation (GSE) protocol introduces an intermediate layer to handle fragmentation of network packets, referred to as Protocol Data Unit (PDU)s, into one or more GSE packets. [18] The operation of the protocol is discussed further in Section 4.2.2.

ETSI released an extension to the original DVB-S2 standard called DVB-S2X, this included several features to improve efficiency for high throughput satellites, such as bonding multiple transponders, beam hopping, and high order modulations up to 256APSK. Particularly relevant, are the new features for Very Low Signal-to-Noise Ratio (VL-SNR), which include new PLFRAME structures, Binary Phase Shift Keying (BPSK) modulation, and lower coding rates down to 1/5. Features were also added to improve the efficiency of GSE packet transmission.

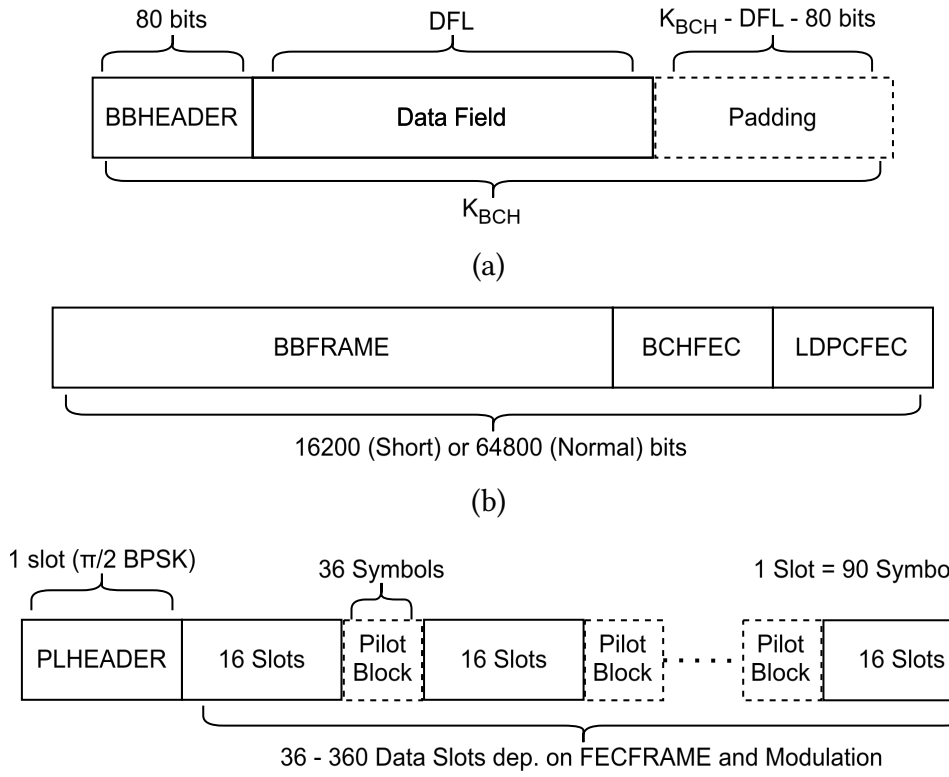


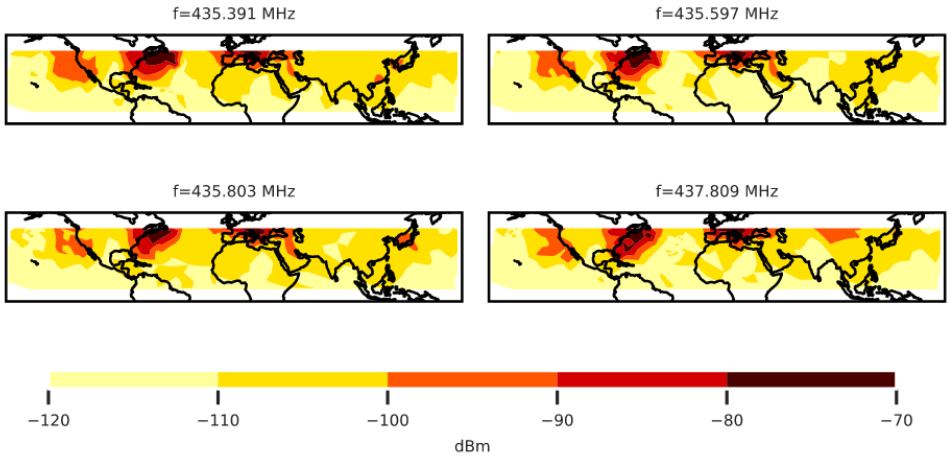
Figure 10: DVB-S2 Frame Structures. (a) Shows a BBFRAME, where K_{BCH} is the input data size to the FEC system. (b) Shows a FECFRAME before scrambling. (c) Shows a PLFRAME, where the header is transmitted using $\frac{\pi}{2}$ BPSK modulation and data transmitted in groups of 16 slots, followed by an optional 36 symbol pilot block.

In [19], an SDR-based communication system operating in the 915 MHz UHF band and utilizing ACM is described. The system employed modulation techniques similar to DVB-S2 but implemented a different coding scheme. Their analysis demonstrated nearly double

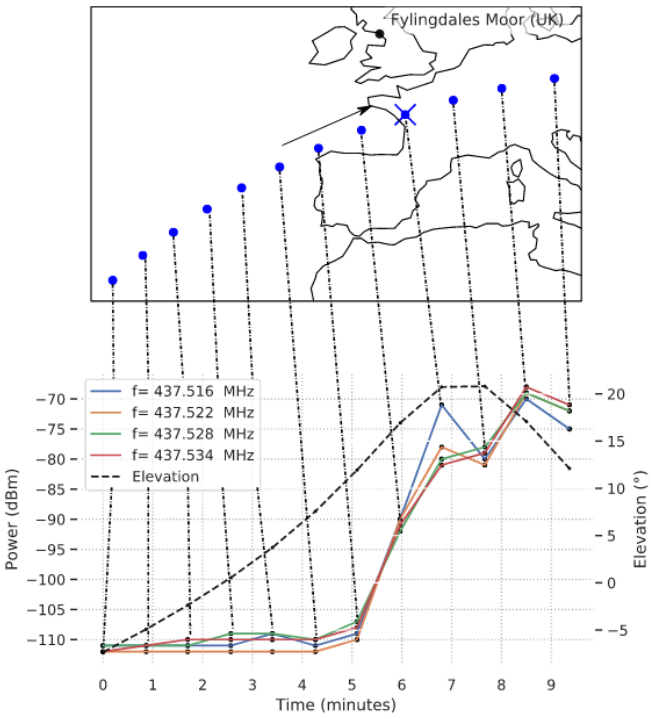
the throughput compared to CCM, confirming the feasibility and significant performance advantages of ACM systems for CubeSats.

2.5 Interference

Interference is particularly prevalent in the UHF amateur band. In [20] interference power in the UHF amateur space allocation was measured using the Serpens satellite. Sustained interference power of over −70dBm was measured, with particularly high levels measured over Europe, as shown in Figure 11a and Figure 11b. This issue is more relevant for uplink communications, as ground stations can use highly directional antennas to reduce the impact of terrestrial interference, however during low elevation passes this will impact the downlink communications as well.



(a) Interference power heatmap.



(b) Interference power over time during a pass over Europe.

Figure 11: Interference Analysis

To reduce the impact of this interference, advanced filtering could be used. In [21] the effect of narrowband continuous wave interference was investigated for DVB-S2 QPSK communications and an adaptive notch filter designed. Figure 12 [21, Fig. 27] shows the effect on Bit Error Rate (BER) of a Jamming to Signal Ratio (JSR) of just −6dB. Without filtering, and at low Eb/N0 values, the recovery is significantly degraded however the filter is able to improve performance to close to that of the theoretical peak. The paper also notes that the filter could reduce performance in certain scenarios.

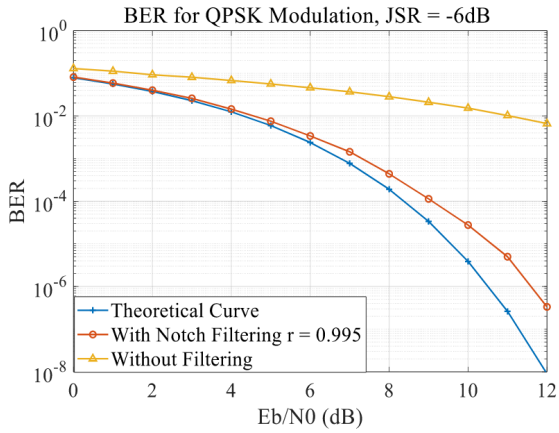


Figure 12: Impact of interference on DVB-S2 QPSK signal reception.

The issue of interference was identified as a gap in the analysis described in Section 3, and will form the basis of future analysis of the communications system. It will be particularly important during the design of the uplink communications system.

3 ACM Analysis

The link budget described in Table 1 was reassessed and updated to identify parameters that vary over time and to reflect the impact of ACM. It has also been updated to reflect the information obtained regarding the STAC ground station, shown in Table 4, although these numbers will require update as the condition of the ground station has likely degraded. This analysis contributed to the ESA Baseline Design Review as part of the Link and Data Analysis Report.

3.1 System Definitions

Table 5: Static Parameters

Name	Value	Source
System		
Frequency (f)	437 MHz	Centre of UHF Amateur Satellite Service Allocation
Bandwidth (B)	150 kHz	
STRATHCube		
Transmit Power (P_{Tx})	1W	TOTEM UHF FE Limit, [6]
Cable Losses (L_{Cable})	0.116 dB	20cm RG-188/AU, AcubeSAT

Name	Value	Source
VSWR	1.9 : 1	ISIS Antenna Datasheet [7]
Antenna Reflection Loss ($L_{\text{Reflection}}$)	0.44 dB	Equation 1, AcubeSAT
Connector Losses ($L_{\text{Connector}}$)	0.2 dB	4 Connectors @ 0.05dB, AcubeSAT
Switch Losses (L_{Switch})	0.5 dB	AcubeSAT, included specutalively
Total Line Losses (L_{Line})	1.26 dB	Equation 2
Transmit Antenna Gain (G_{Tx})	0 dBi	ISIS Antenna Datasheet [7]
EIRP	**0.50 dBW**	Equation 3
Ground Station		
Receive Antenna Gain (G_{Rx})	**14.15 dBi**	STAC UHF Antenna [14]
LNA Gain (G_{LNA})	20 dB	SP-7000 Datasheet, STAC [15]
Line Losses	2.39 dB	AcubeSAT
Ground Station Noise Temperature		
Reference (T_{Ref})	290K	Standard Value, [22, Sec. 5.5.2]
Antenna / Sky (T_{Ant})	154K	AcubeSAT
Feedline (T_{Feed})	290K	AcubeSAT
LNA Noise Figure (N_{LNA})	0.9 dB	SP-7000 Datasheet [15], STAC
LNA (T_{LNA})	66.8K	Equation 5
Frontend Noise Figure	8 dB	USRP B210 Maximum [23]
Frontend (T_{FE})	1539.8K	Equation 5
Cable Loss (L_{Cable})	1.023 dB	AcubeSAT
Transmission Line Coefficient (α)	0.6331	AcubeSAT
Receiver Noise Temperature (T_{Rx})	**282.7K**	Equation 4
Atmospheric Path Losses		
Scintillation ($L_{\text{Scint,dB}}$)	0.16 dB	AcubeSAT
Rain Fade ($L_{\text{Rain,dB}}$)	0 dB	Assumed as negligible in UHF
Ionospheric ($L_{\text{Ion,dB}}$)	0.4 dB	AcubeSAT
Polarisation ($L_{\text{Pol,dB}}$)	3 dB	
AEL	**3.56 dB**	Equation 6

Table 6: Dynamic Parameters

Name	Adverse	Nominal	Favourable	Source(s)
Elevation	10°	20°	40°	Section 3.2.3
Atmospheric Absorption (L_{Atm})	2.1 dB	1.1 dB	0.4 dB	AcubeSAT

Name	Adverse	Nominal	Favourable	Source(s)
Slant Range	Calculated / Scenario			Equation 7
FSPL	Calculated / Scenario			Equation 9
CNR	Calculated / Scenario			Equation 10
CNR Margin	Calculated / Scenario			Equation 11

3.2 Methodology

3.2.1 Link Budget

Reflection loss can be calculated as follows: [9]

$$\begin{aligned} L_{\text{Reflection,W}} &= P_{\text{Tx,W}} \times \frac{(\text{VSWR}-1)^2}{(\text{VSWR}+1)^2}[\text{W}] \\ L_{\text{Reflection,dB}} &= 10 \times \log_{10}\left(\frac{P_{\text{Tx,W}} - L_{\text{Reflection,W}}}{P_{\text{Tx,W}}}\right)[\text{dB}] \end{aligned} \tag{1}$$

The Transmitter total line losses are calculated as follows:

$$L_{\text{Line,dB}} = L_{\text{Cable,dB}} + L_{\text{Reflection,dB}} + L_{\text{Connectors,dB}} + L_{\text{Switch,dB}}[\text{dB}] \tag{2}$$

EIRP:

$$\text{EIRP}_{\text{dB}} = 10 \times \log_{10}(P_{\text{Tx,W}}) - L_{\text{Line,dB}} + G_{\text{Tx,dB}}[\text{dBW}] \tag{3}$$

Receiver noise temperature: [9]

$$T_{\text{Rx}} = \alpha \times T_{\text{Ant}} + (1-\alpha) \times T_{\text{Feed}} + \frac{T_{\text{FE}} \times L_{\text{Cable}}}{G_{\text{LNA}}}[K] \tag{4}$$

Noise figure can be converted to noise temperature as follows: [22, eq. 5.28]

$$T = T_{\text{Ref}}\left(10^{\frac{F_{\text{dB}}}{10}} - 1\right) \tag{5}$$

Atmospheric Effect Loss and Atmospheric Path Loss:

$$\text{AEL}_{\text{dB}} = L_{\text{Scint,dB}} + L_{\text{Rain,dB}} + L_{\text{Ion,dB}} + L_{\text{Pol,dB}}[\text{dB}] \tag{6}$$

$$\text{APL}_{\text{dB}} = \text{AEL}_{\text{dB}} + L_{\text{Atm,dB}}[\text{dB}] \tag{7}$$

Slant Range: [24]

$$\begin{gathered}\mathrm{SR} = \sqrt{R^2 + (R+H)^2 - 2 \times R \times (R+H) \times \cos(\theta)} \\ \text{Where :} \\ R = \text{Earth Radius} + \text{Ground Station Altitude} \\ H = \text{Satellite Altitude} - \text{Ground Station Altitude} \\ \alpha = \text{Satellite Elevation Angle} \\ \theta = \arccos\left(\frac{R - R\sin^2(\alpha) + \sin(\alpha)\sqrt{(R\sin(\alpha))^2 + H^2 + 2RH}}{R+H}\right)\end{gathered} \tag{8}$$

Free Space Path Loss: [22, eq. 5.10]

$$\mathrm{FSPL} = 20\log_{10}\left(\frac{4\pi d f}{c}\right)[\mathrm{dB}] \tag{9}$$

Received CNR:

$$\mathrm{CNR}_{\mathrm{Rx}} = \mathrm{EIRP} - \mathrm{FSPL} - \mathrm{APL} + G_{\mathrm{Rx}} + 228.6 - 10\log_{10}(T_{\mathrm{Rx}}) - 10\log_{10}(B)[\mathrm{dB}] \tag{10}$$

Link Margin:

$$\mathrm{Margin} = \mathrm{CNR}_{\mathrm{Rx}} - \mathrm{CNR}_{\mathrm{Required}}[\mathrm{dB}] \tag{11}$$

3.2.2 Required CNR by Modulation and Coding Rate

Values for spectral efficiency and minimum $\frac{E_s}{N_0}$ are provided in [25, Tab. 13]. Those values are derived from simulation of a DVB-S2 system with 50 LDPC decoding iterations, 64,800 bit FECFRAME, no pilots, perfect carrier and synchronization recovery, no phase noise and an Additive White Gaussian Noise (AWGN) channel. The system shall include pilots to improve decoding performance, the spectral efficiency of which was obtained from [26, Tab. 3-1]. As such, the actual minimum $\frac{E_s}{N_0}$ will be lower, adding further margin under AWGN conditions.

Interference, demodulation and phase noise effects are still to be analysed, which will reduce the calculated link margin. For this reason a margin target of 10dB was selected.

Additionally, it was assumed that ideal matched filtering is in place and thus that the total system spectral efficiency is equal to that of the theoretical value for the modulation and coding rate. With that assumption, Equation 12 and Equation 13 were used to create Table 7.

$$\frac{E_s}{N_0} = \frac{C}{N}[\mathrm{dB}] \tag{12}$$

$$\mathrm{Capacity} = \eta \times B[\mathrm{bps}] \tag{13}$$

Table 7: DVB-S2 Modulation and Coding Rate Requirements, 150kHz Bandwidth,

Bits / Symbol	Modulation	Coding Rate	Spectral Efficiency (bps/Hz)	Data Rate (bps)	Minimum CNR (dB)	Minimum CNR w. 10dB Margin (dB)
4	QPSK	1/4	0.4786	75860	-2.35	7.65
4	QPSK	1/3	0.6408	101600	-1.24	8.76
4	QPSK	2/5	0.7706	122100	-0.3	9.7
4	QPSK	1/2	0.9653	153000	1	11
4	QPSK	3/5	1.16	183900	2.23	12.23
4	QPSK	2/3	1.291	204600	3.1	13.1
4	QPSK	3/4	1.452	230200	4.03	14.03
4	QPSK	4/5	1.549	245600	4.68	14.68
4	QPSK	5/6	1.615	256000	5.18	15.18
4	QPSK	8/9	1.724	273300	6.2	16.2
4	QPSK	9/10	1.746	276700	6.42	16.42
8	8PSK	3/5	1.74	275700	5.5	15.5
8	8PSK	2/3	1.936	306800	6.62	16.62
8	8PSK	3/4	2.178	345100	7.91	17.91
8	8PSK	5/6	2.422	383900	9.35	19.35
8	8PSK	8/9	2.586	409900	10.69	20.69
8	8PSK	9/10	2.618	415000	10.98	20.98
16	16APSK	2/3	2.575	408100	8.97	18.97
16	16APSK	3/4	2.896	459100	10.21	20.21
16	16APSK	4/5	3.091	489800	11.03	21.03
16	16APSK	5/6	3.222	510700	11.61	21.61
16	16APSK	8/9	3.44	545200	12.89	22.89
16	16APSK	9/10	3.483	552000	13.13	23.13
32	32APSK	3/4	3.623	574300	12.73	22.73
32	32APSK	4/5	3.866	612800	13.64	23.64
32	32APSK	5/6	4.031	638900	14.28	24.28
32	32APSK	8/9	4.303	682000	15.69	25.69
32	32APSK	9/10	4.357	690600	16.05	26.05

3.2.3 Adaptive Coding and Modulation Analysis

An initial proof of concept investigation was conducted to determine if implementation of ACM is worth the increased complexity. This investigation assumed a fixed orbit altitude of 409km and used ISS Two-Line Element set (TLE) data to propagate this for the full mission length of 180 days. Orbit propagation was conducted using the MATLAB Satellite Communications Toolbox [27] Satellite Scenario and Access objects.

Using the system parameters defined in Section 3.1, the equations defined in Section 3.2.1, and the required CNR values defined in Table 7, the maximum achievable data rate was precomputed for each elevation value from 0 to 90° in 5° increments for both 170km and 409km altitude orbits. The resulting rates are shown in Figure 13.

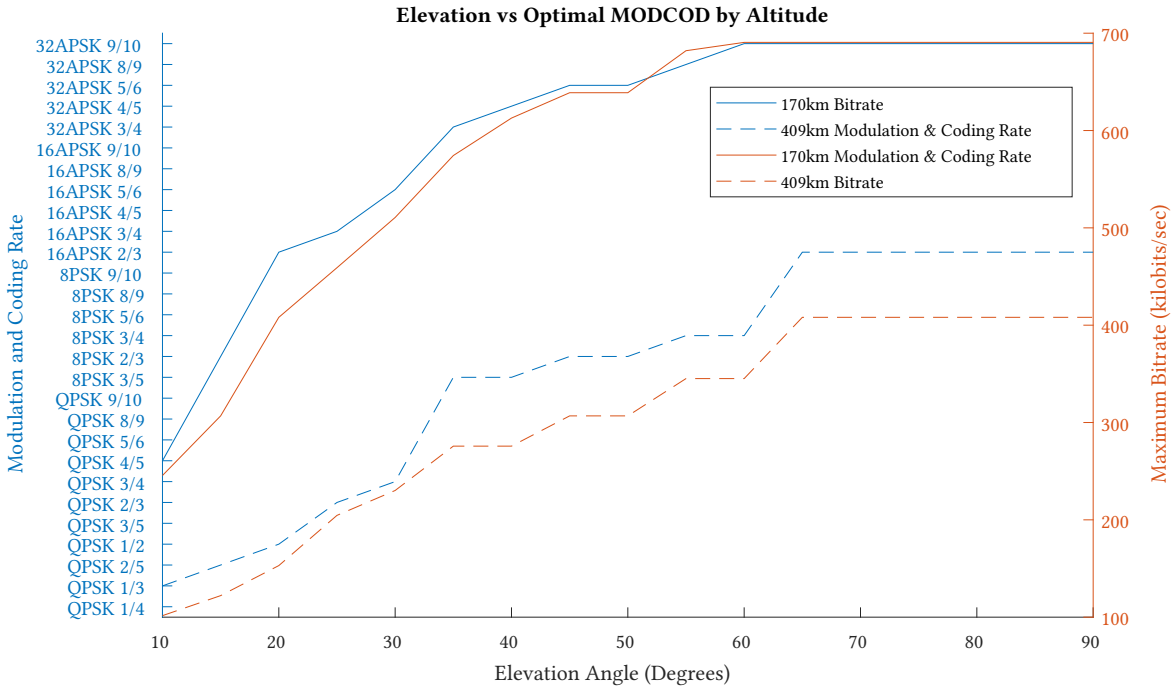


Figure 13: Elevation vs Throughput

To improve performance, the simulation is run with two timesteps. First, a coarse search is run with a timestep of 100 seconds and the accessIntervals() function used to find the starting time of each ground station pass. Then, a fine search using a timestep of 1 second is used to find the elevation of the satellite over time during each pass.

The simulated elevation angles were then binned with values rounded down to the nearest multiple of 5. A histogram of the binned elevation angles versus the time spent within each elevation bin was then created. This was then multiplied by the corresponding values in Figure 13 to obtain an estimate of the total data downlinked over the course of the mission using ACM. For CCM it was assumed that the capacity value corresponding to 10° elevation was used for the entire mission. The resulting plots are shown in Figure 14.

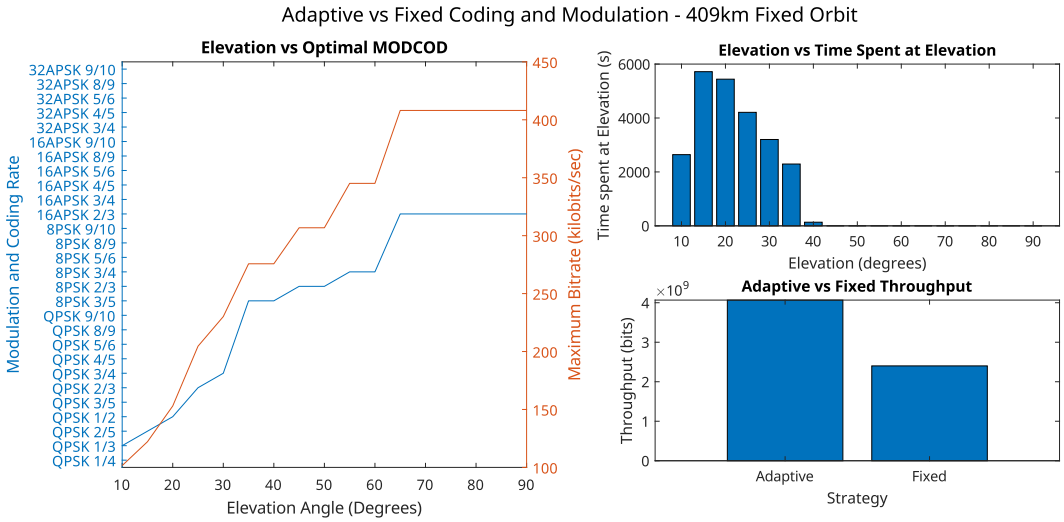


Figure 14: ACM Performance Analysis

A MATLAB function was created to automate this process, called datarate(). This is used to calculate the highest datarate possible given a set link budget and the satellite's elevation relative to the ground station and altitude above the Earth. This has been integrated into the Mission Analysis Tool, an internal mission analysis framework for STRATHcube. The beginning of the function definition is shown in Listing 1, and the full code is shown in Appendix A.1.1

```
function [rates] = datarate(elevation,altitude,bandwidth,margin,verbose,
link_parameters)
%DATARATE Calculates datarate for each elevation and altitude pair
% Returns:
%   rates:              Datarates for each      bits/sec, array
size
%                       elevation & altitude    length(elevation)
%
% Parameters:
%   elevation:          Satellite Elevation     degrees,    can be array
%   altitude:           Satellite Altitude      meters,     can be array
(must be same length as elevation)
%   bandwidth:          Link Bandwidth          Hz          Optional
%   margin:             Link Margin Required    dB          Optional
%   link_parameters:    Static link values      struct      Optional
%
% Requires modcod_to_CNR.mat to be in same directory. This contains
% spectral efficiency values for each MODCOD
```

Listing 1: Excerpt from datarate.

3.3 Scenarios

Current analysis has been restricted to link budgets based on elevation relative to ground station and satellite altitude, as the impact of FSPL is dominant over most other factors. Additionally, the impact of component degradation is yet to be investigated and requires determination. The scenarios were selected such that they represent the best and worst case for FSPL using the STAC ground station over the expected lifetime. As discussed in

Section 1.2.3, the satellite is at its highest altitude at release from the International Space Station. The primary downlink communications are no longer used during the secondary phase, so the final relevant altitude is 170km when the transition phase begins.

3.3.1 Commisioning Phase

STRATHcube shortly after deployment from the international space station.

Table 8: STRATHcube Commissioning Phase Budget

Name	Adverse	Nominal	Favourable
Altitude	409km		
Slant Range	1463km	1001km	611km
FSPL	148.5 dB	145.2 dB	140.9 dB
CNR	10.9 dB	14.1 dB	19.1 dB
Highest Achievable MODCOD w. 10dB Margin	QPSK 2/5	8PSK 3/5	16PSK 3/4
CNR Required	−0.3 dB	4.0 dB	9.0 dB
Bitrate	115.6 kbps	217.8 kbps	386.2 kbps

3.3.2 Transition Phase

STRATHcube at end of primary phase.

Table 9: STRATHcube Transition Phase Budget

Name	Adverse	Nominal	Favourable
Altitude	170km		
Slant Range	743km	456km	260km
FSPL	142.7 dB	138.4 dB	133.5 dB
CNR	16.1 dB	20.4 dB	26.0 dB
Highest Achievable MODCOD w. 10dB Margin	8PSK 2/3	16APSK 3/4	32APSK 9/10
CNR Required	6.62 dB	10.21 dB	16.05 dB
Bitrate	290.4 kbps	434.5 kbps	653.5 kbps

3.4 Areas for Further Investigation

Thus far the link budget has been concerned with proving viability and performance limits for development of the transmitter system. As such, there has been comparatively less investigation of the ground station receiver system. There has also been a large reliance on numbers derived from the AcubeSAT link budget due to this.

There are multiple key challenges for implementing a DVB-S2 receiver, the largest issue being interference, which can be prevalent on the planned UHF band, see Section 2.5. Further challenges include carrier synchronisation and phase correction.

Additionally, the bandwidth calculations have been made with the assumption of perfect filtering with zero roll-off, which is not representative of a practical DVB-S2 system. This is discussed further, and practical datarates established in Section 5.1.2.1.

4 System Design

4.1 Systems Engineering

In order to work effectively in a large project structure, a review of system requirements was conducted with close collaboration with system engineers on the team.

This contributed to a complete overhaul of all Telemetry, Tracking and Command (TT&C) requirements to be more relevant to actual operations, for submission as part of the ESA Baseline Design Review. The relevant requirements are shown in Table 10. The figures marked To Be Confirmed (TBC) are indicative only, pending full analysis of the mission parameters. PPL refers to the Primary Payload, the PBR. TM refers to telemetry data, including data about the spacecraft state, such as battery charge levels. GS refers to the Ground Station, (network) used to indicate that there may be several.

Table 10: Relevant system and TT&C requirements for downlink communications.

Code	Description
SYS-082	The CubeSat shall downlink PPL data according to a packet prioritisation system.
SYS-083	The CubeSat shall downlink TM data according to a packet prioritisation system.
TTC-013	The primary communication system shall be capable of transmitting PPL data to the GS with a minimum data rate of 5 kbps (TBC).
TTC-014	The primary communication system shall be capable of transmitting TM data to the GS with a minimum data rate of 5 kbps (TBC).
TTC-016	The primary communication system shall downlink packetised data according to a packet prioritisation system.
TTC-019	The primary communications system shall be capable of transmitting packetised data to the ground station (network) at a capacity in the range of 5-50 kbps (TBC).
TTC-021	During ground communication windows, the primary communications system shall have a minimum link margin of 3dB.

The data handling requirements had not yet been substantially explored at time of writing. The most relevant requirements that had been created are shown in Table 11. OBC refers to the On-Board Computer, used for data processing and control of the spacecraft. OBSW refers to On-Board SoftWare, this refers to any code running on the satellite, whether it is on the SDR, OBC, or elsewhere. The listed protocols are Controller Area Network (CAN), Universal Asynchronous Receiver-Transmitter (UART), and Inter-Integrated Circuit (I2C).

Table 11: Relevant data handling requirements.

Code	Description
ODH-039	The OBC shall support standardised protocols such as CAN/UART/I2C.
ODH-040	The OBSW shall produce PPL data packets that include timestamp, I/Q, position, and attitude.

From these requirements, rough design objectives were derived for the downlink communication system:

1. The system shall support packet input from all interfaces present on the TOTEM SDR
2. The system should maximise datarate while conforming to BER requirements.
   a. The system shall support modification of modulation and coding rate through ACM or VCM.
   b. Efficiency should be prioritised. The design should use minimal padding and seek to reduce overheads.
3. The system should be reliable
   a. The system shall be designed for testability.
      i. Large pieces of logic shall be broken up into smaller systems that can be unit tested.
      ii. Assertions shall be used regularly throughout code and block designs.
   b. The system shall be tested thoroughly using simulation to verify operation.
4. The downlink subsystem and PBR system should fit on the PL at the same time.
   a. The downlink system should use no more than 50% of any one resource type.
5. The downlink subsystem shall be capable of transmitting packets that are over 64kB.

Requirement 4 follows on from requirement 3 of reliability. If the FPGA has to be reconfigured for every downlink pass, this introduces a large amount of complexity. If this process fails, communication will have to fall back to the much slower secondary communication system. There is also the effect of radiation to consider, if the stored bitstream file is corrupted its recovery could be very difficult.

Requirement 5 is derived from the PBR packet size requirements. This is discussed further in Section 4.2.2.

4.2 Packet Handling

4.2.1 Quality of Service

The packets onboard STRATHcube can be grouped into two distinct types. The first is telemetry, those that are required for the continued functioning of the spacecraft, such as information regarding battery state or the orientation of the craft. The second being payload packets, such as the measured IQ signals during an observation period.

Of the two types, telemetry packets are more important and make up a much smaller proportion of total data sent. These telemetry packets are to be compressed using the POCKET or POCKET+ algorithm. This may occur on the SDR, but is more likely to be done in advance on the OBC.

To maximise datarates, it is possible to group packets into classes, and apply rules for BER seperately for each class. For instance, the primary payload data may be transmitted with a BER target of $10^{-}5$, and the telemetry data with a target of $10^{-}9$. The exact targets should be subject of future analysis, as retransmission of primary payload data could be extremely costly due to large block sizes.

4.2.2 Fragmentation

The FAPEC algorithm was one of the options identified for compression of the PBR data, which outputs data in "chunks". These chunks typically have a fixed size, ranging from 4kB to 384 MB [28]. If the compressed data is corrupted, only those chunks containing errors need to be corrected, therefore, the selection of chunk size provides a logical minimum packet size. The maximum Data Field Length (DFL) for DVB-S2 is 58,112 bits, or 7264 Bytes. This is only possible when using the highest modulation and coding rates, which are not expected to be feasible for much of the mission. For this reason, it was deemed likely that the PBR data packets would require fragmentation across multiple frames.

GSE will be used for this purpose, as it is specifically designed for DVB-S2 transmission. A GSE packet has the following structure:

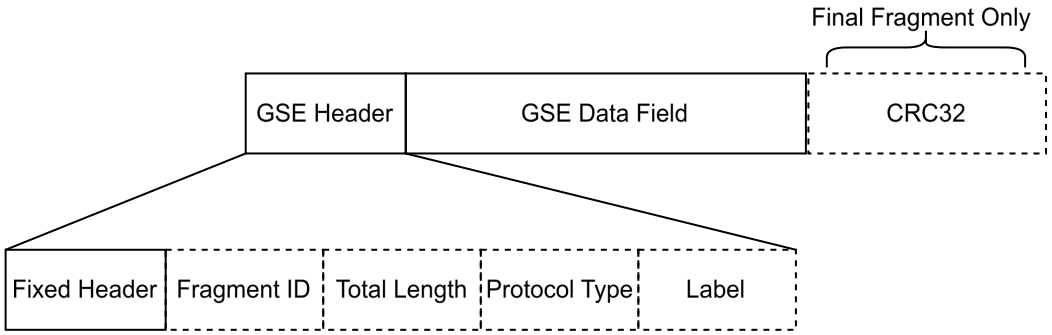


Figure 15: GSE packet structure. The header is composed of two sections, the first is fixed, and the second has content determined by those fixed flags. This is followed by the data field, with the final part being a CRC32 checksum to detect reconstruction errors.

Bits	0	1	2:3	4:15
Field	Start	End	Label Type	GSE Length

(a) Fixed

Bits	1 B	2 B	2 B	3 or 6 B
Field	Fragment ID	Total Length	Protocol Type	Label

(b) Variable

Figure 16: GSE header fields. The fixed header is 16 bits long, while the variable header can be from 0 to 11 Bytes.

There are four key parts of the header for this application.

1. **Label Type**: Indicates the length or existence of the label field. Can also indicate if a label is to repeated from the previous GSE packet.
2. **Label**: Typically used to indicate addresses for networking. Optional.
3. **Total Length**: Total length of PDU before fragmentation in bits. Maximum of 65536 Bytes.
4. **GSE Length**: Length of GSE packet in bits. Maximum of 4096 Bytes.

For this application, the label field is unnecessary, as the source and destination of packets is always known. This also means that the label type field could be omitted, saving two further bits per GSE packet at the expense of standard compliance.

The GSE standard also defines GSE-Lite which is optimised for constrained systems, however it only supports a max PDU size of 1800 Bytes, making it unsuitable for this application.

4.2.3 Packet Definition

As discussed in Section 4.1, the data handling systems on the satellite were underdeveloped at time of writing, it was decided that the final system should accept packet schemas in order to handle future updates to packet structure with minimal refactoring. The XML Telemetric and Command Exchange (XTCE) format was identified as suitable, as it is a Consultative Committee for Space Data Systems (CCSDS) standard that has been used for several missions.

NASA has created an open source program called Core Flight System (CFS) Command and Data Dictionary (CCDD) which can be used for the creation and management of XTCE packet schemas. Additionally, it can export packets to C headers, which could be useful for future packet handling work.

4.3 Transmitter

It was decided to handle merging of packet streams in the processing system, and to implement a single stream transmitter using the programmable logic. This is due to the limited memory available in the FPGA fabric, which would otherwise be consumed by large buffers required for handling multiple streams simultaneously. It was also decided that the transmitter should support both continuous and generic stream transmission, as the GSE design architecture had not been finalised, meaning that variable packet sizes may be necessary. The resulting architecture with optional blocks removed is shown in Figure 17.

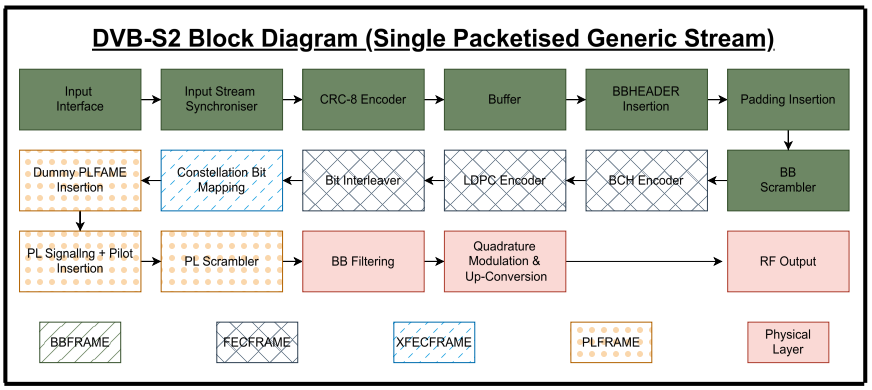


Figure 17: Transmitter Block Diagram

4.4 Full System

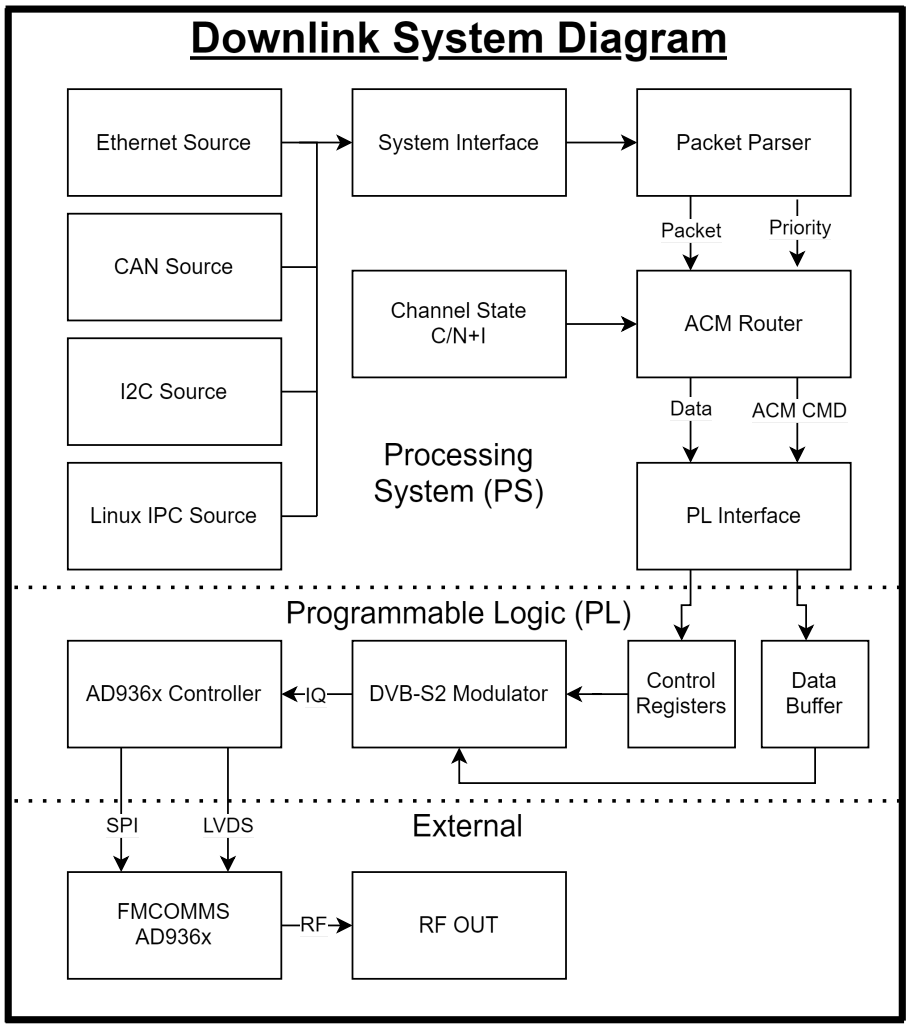


Figure 18: System Diagram

Due to the limitation of scope for this project, the particulars of the channel measurement has not been investigated, there are two main requirements for the system: Accuracy and frequency of updates.

As identified in Section 3, the main varying factor is the distance from the ground station to the satellite which can be computed in advance, giving VCM. Interference has not been analysed thusfar, but is expected to be a significant and varying factor that cannot be predicted, requiring the implementation of ACM. The presented architecture allowed either to be implemented without redesign.

Another important factor for the system is “in the blind” communications, i.e. downlink data transmission without any uplink communications from the ground station. Under these conditions, there is no return channel to facilitate ACM operation. Additionally, prescheduled VCM operation requires orbit position information that may not be present. For this reason, the lowest MODCOD of QPSK 1/4 shall be used, as this maximises availaibility.

5 Implementation

5.1 Transmitter

5.1.1 Tool Selection

There were multiple options for implementation of the DVB-S2 subsystem. Table 12 shows the tradeoff analysis conducted for the five most relevant options. MATLAB HDL Coder [29] was selected due to the Simulink integration reducing the complexity of verification and the Hardware Software Co-Design features discussed further in Section 5.1.3.1.

Table 12: Implementation Tool Tradeoff

Method	**Type**	**Advantage**	**Disadvantage**
GNU Radio software implementation	Software	Simplicity of implementation	Slower processing speed
COTS IP Core	Hardware	Pre-tested, guaranteed reliability	Expensive, could complicate integration with PBR system
Vitis Model Composer	Hardware	Simplicity of implementation, Can be fully verified using MatLab simulation	Would have to implement interface with transceiver block manually, taking more development time.
Scratch implementation in VHDL or Verilog	Hardware	Highest resource efficiency.	Will require much more development time, as well as significantly increased testing time
MATLAB HDL Coder	Hardware	Supports AD936x transceiver as a target. Can be fully verified using MatLab simulation	Poorer resource efficiency

5.1.2 DVB-S2 Modulation

5.1.2.1 Bandwidth and Symbol Rate

The block design has a sample rate dependent on the symbol rate of the transmitter. To limit the clock domains on the design, this was used to define the sample rate for the entire system.

The single sided bandwidth of a modulated DVB-S2 signal is a function of the symbol rate, R_s, and the roll-off factor, α, of the root raised cosine filter as shown in Equation 14. This can be rearranged to find the highest symbol rate for a given bandwidth, as shown in Equation 15.

$$\begin{aligned} B_{\text{single}} &= f_N(1+\alpha) \\ \text{Where } f_N &= \frac{R_s}{2} \end{aligned} \tag{14}$$

$$R_s = \frac{B_{\text{double}}}{1+\alpha} \qquad (15)$$
$$\text{Where } B_{\text{double}} = 2B_{\text{single}}$$

The standard allows for a roll-off factor of 0.35, 0.25 or 0.20. [25] For the target bandwidth of 150 kHz, this gives the theoretical maximum symbol rates shown in Table 13.

Table 13: Symbol rate vs roll-off factor

α	R_S (Symbols / sec)
0.35	111,111
0.25	120,000
0.20	125,000

This resulted in a tradeoff between maximum symbol rate and the roll-off factor, which was linked to the complexity of the filter. A pulse shaping filter with more weights was required to achieve the higher symbol rates, which increased the overall resource requirements of the design.

With this filter in place, the expected bitrate was calculated for each roll-off factor using the spectral efficiency values from Table 7. The result is shown in Figure 19. Note that the resulting datarates were lower than in Section 3 as the usable bandwidth had reduced.

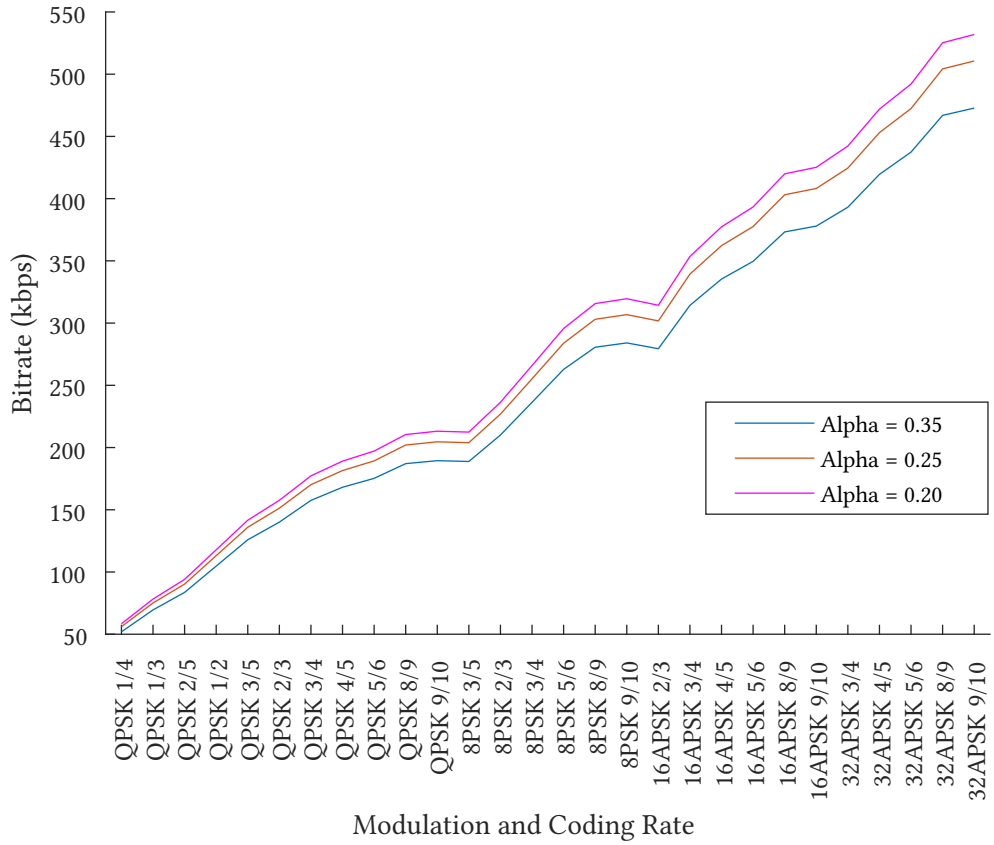


Figure 19: Bitrate achievable by modulation, coding rate, and roll-off factor (Alpha).

For most of the settings, all three have relatively similar performance. As the modulation and coding rates get higher, the differences become more apparent. Overall, the difference between $\alpha = 0.25$ and $\alpha = 0.20$ is minimal, but $\alpha = 0.35$ consistently performs significantly worse than the other two.

It was determined in Section 3 that the majority of time during a pass will be using the lower modulation and coding rate settings, so it was decided to use $\alpha = 0.25$ to conserve resources. Through simulation, it was determined that a symbol rate of 125,000 Symbols / second using this rolloff was achievable while staying within the International Telecommunication Union (ITU) spectrum mask, discussed further in Figure 32.

5.1.2.2 Transmitter block

The core of the design is built upon a DVB-S2 HDL coder example implementation created by Mathworks. [30] This block implemented all of the necessary logic for the modulation of a bitstream into a valid DVB-S2 signal.

Table 14: DVB-S2 Transmitter Ports

Port	Direction	Type	Purpose
Sideband Signals			
pkTVALIDIn	Input	Boolean	Indicates that input data is valid
pktBitsIn	Input	Boolean Stream	Information for transmission
pktStartIn	Input	Boolean	Indicates packet start, not used for continuous stream
pktEndIn	Input	Boolean	Indicates packet end, not used for continuous stream
frameStartIn	Input	Boolean	Indicates frame start
frameEndIn	Input	Boolean	Indicates frame end
Control Signals			
TSorGS	Input	ufix2	Indicates if transport stream, continuous or packetised generic stream
DFL	Input	uint16	Data Field Length, can be from 0 to K_{BCH}
UPL	Input	uint16	User Packet Length, 0 for continuous stream, up to K_{BCH}
SYNC	Input	uint8	Used in stream synchronisation
MODCOD	Input	ufix5	MODulation and CODing rate selection
FECFRAME	Input	boolean	FECFRAME length selection. 1 for
Output			
dataOut	Output	ufix18en16 (c)	IQ output data. 18 bit complex fixed point data with 16 fraction bits
validOut	Output	boolean	Indicates that IQ data is valid
flag	Output	ufix2	Flag bits to indicate when a dummy frame is sent, and when a real frame is sent.
nextFrame	Output	boolean	Asserted when transmitter is ready for next frame. Is reset when frameStart is asserted

5.1.3 PL System Integration

5.1.3.1 Hardware-Software Co-Design

The transmission subsystem required an interface with both with the packet handling software on the PS and the external transceiver chip. It was decided to use the Hardware-Software Co-Design features of HDL coder to accomplish this.

The Co-Design workflow started with the creation of an IP block using HDL coder with a specific interface. This design was then automatically integrated into a predefined “reference design” in Vivado and could be synthesised and implemented on hardware. Additionally, software drivers could be generated to handle the PS-PL interface.

This workflow massively simplified the PS-PL interfacing, which was identified as one of the most challenging parts of an SoC FPGA design. However, it came with some drawbacks. The

first is that the design is limited in that it must conform to the reference design. If there is not a reference design for the target platform this must be implemented manually, a non-trivial process. Additionally, any other software running on the PS would have to integrate with the generated C code, an area that appeared to have little documentation or examples.

A Co-Design example project [31] was used as a template to ensure that all necessary interfaces were generated in order to comply with the requirements of the reference design. This example project had support for both receive and transmit, which would allow both the PBR and downlink communications systems to be implemented on the same design. The top level interface is shown in Figure 20.

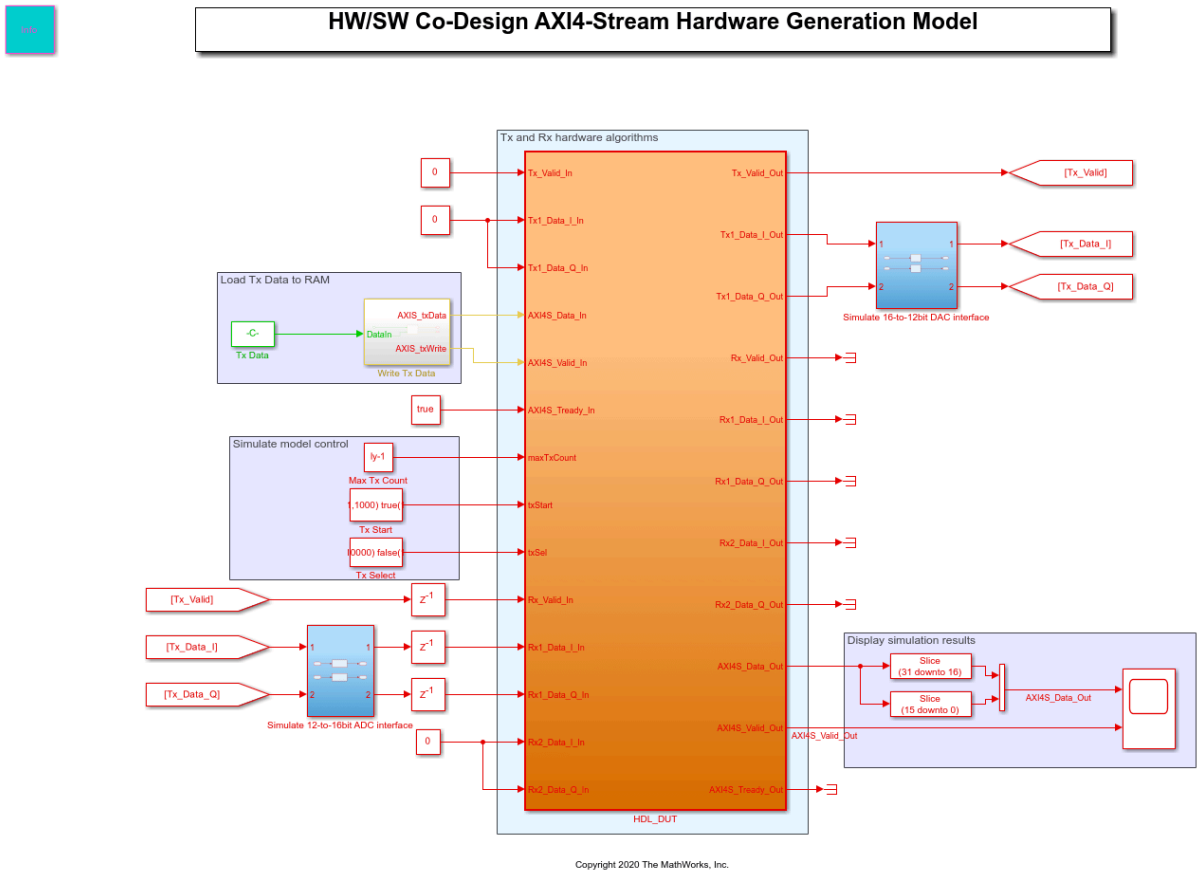


Figure 20: HW/SW Co-Design Chirp Example. The logic to be implemented on the PL is defined within the orange block. External blocks are used to verify operation.

Data is transferred from the PS via one of two methods. AXI-Lite interfaces are used for control signals, as they have lower throughput, this was used for all control signals, such as DFL, UPL, etc. An AXI-Stream was used to transfer packet data, as it has a much higher throughput.

The reference design included an AXI-Stream for the transfer of data from the PS to the design, and separate I and Q ports to connect to the transceiver. Additional control signals could be implemented using AXI-Lite and the design included further ports for implementation of the receiving logic.

An AXI-Stream interface can be implemented with as few as three signals, TDATA, TVALID and TREADY but there are several optional signals defined in the standard, the most relevant being TLAST and TUSER. TLAST is used to indicate the final AXI packet in a stream, while

TUSER can be used to carry any arbitrary user defined data, commonly being information to control the transfer such as flags. These signals are summarised in Table 15.

Table 15: Relevant AXI-Stream Signals

Signal	Type	Purpose
TDATA	uint32	Data to be transferred from master to slave
TVALID	boolean	Indication from master to slave that data is valid and can be read
TREADY	boolean	Indication from slave to master that it is ready for new data
TLAST	boolean	Indication from master to slave that the current packet is the last in the current stream (optional)
TUSER	User Defined	Encodes extra information about the transfer (optional)

TDATA is listed as 32 bits here although by the standard this can be any integer number of bytes. This is due to the fact that the physical PS-PL interface on the Zynq 7020 is 32 bits wide, meaning wider interfaces require more complexity. Additionally, the PS connects to the PL through AXI-MM memory mapped interfaces, with additional PL logic required to interface with a stream. This means that the creation of TUSER signalling synchronised with the AXI-Stream is non-trivial.

An AXI-Stream transfer occurs on the rising edge of the clock, when both TREADY is asserted by the slave, and TVALID by the master. This process is shown in Figure 21 for a minimal stream, where ACLK is the clock controlling the interface.

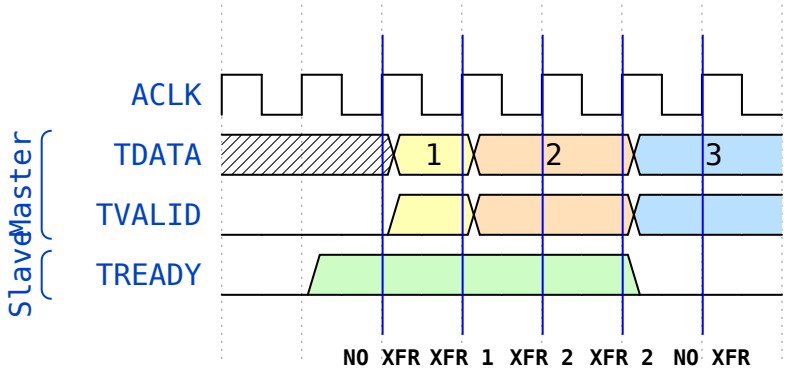


Figure 21: Minimal AXI-Stream timing diagram. A transfer (XFR) occurs on the active edge of the clock when both TVALID and TREADY are high. TDATA can change after a transfer occurs, but must be held once TVALID is asserted until the next transfer.

5.1.3.2 DVB-S2 Transmitter integration

The selected template architecture suited the requirements for the downlink transmission system well. Control signals could be implemented using the AXI-Lite interface, as they can be asserted at any point before the next frame has started. The packet and sideband data could then be transferred using the AXI-Stream, as this required much higher datarates and guaranteed alignment.

For the reasons mentioned previously, it was determined that the implementation of the sideband signals using the TUSER features of the AXI-Stream would overcomplicate implementation as the interface is determined by the reference design. These signals would instead be transferred within the 32 bit TDATA stream.

The remaining 28 bits could then be used to carry the packet data. No DFL size was evenly divisible by 28, giving two avenues for implementation. The first option was to implement a signalling system within the TDATA field allowing for a variable amount of packet bits within each TDATA field. The second was to find the largest number that factored all possible DFL sizes and accept the lower performance.

The maximum DFL sizes corresponding to all valid modulation and coding rates, as shown in Table 16, were factorised and the largest common factor found to be 8, giving a maximum of 1 byte per AXI packet without additional signalling.

AXI-Streams can transfer data much faster than the resultant symbols could be transmitted, so it was decided to accept the performance loss of using a subset of the TDATA field as it greatly reduced complexity. The performance impact was minimal, as each packet needed to be serialised over multiple clock cycles anyway. The resultant TDATA structure is shown in Figure 22.

Table 16: Possible DFL Sizes (bits)

Normal FECFRAME	15928, 21328, 25648, 32128, 38608, 42960, 48328, 51568, 53760, 57392, 58112
Short FECFRAME	2992, 5152, 6232, 6952, 9472, 10552, 11632, 12352, 13072, 14152

Bits	0	1	2	3	4:11	12:31
Signal	Packet Start	Packet End	Frame Start	Frame End	Data	Unused

Figure 22: Designed TDATA structure.

The AXI packets were transferred as uint32 words, which required serialisation for interface with the DVB-S2 transmitter block. This introduced multiple rates for the interface, for which a First In First Out (FIFO) buffer was implemented as it could be used to manage the AXI-Stream interface logic.

The implementation of the TREADY handling is very important for an AXI-Stream slave and is a common source of issues within an AXI based design. The DVB-S2 transmitter block asserts nextFrame to indicate it is ready for data and deasserts it once frameStart has been asserted, however the AXI-Stream required TREADY to be high for the entire frame. This resulted in the circuit shown in Figure 23, with the intended timings shown in Figure 24.

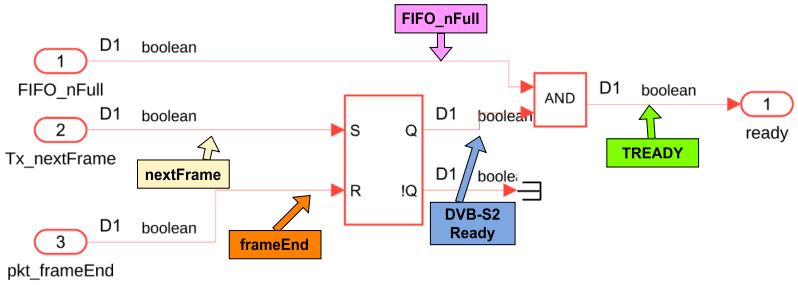


Figure 23: AXI-Stream TREADY generation logic. nextFrame is connected to the Set input of an SR flip flop, and frame end to the Reset. The output of this is then ANDed with FIFO nFull.

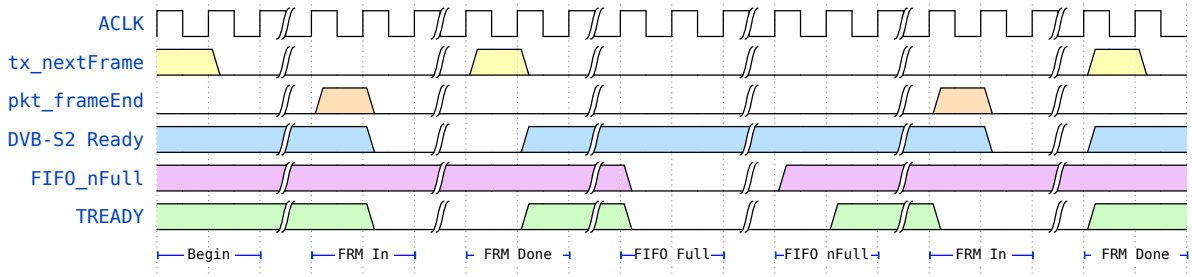


Figure 24: AXI-Stream TREADY timing. The DVB-S2 transceiver block asserts nextFrame for a single clock. This signal is extended until a packet signalling the end of a frame is received to create DVB-S2 Ready. TREADY is only asserted when both the transceiver block is ready and the FIFO is not full.

The next stage of the input stream handling is the FIFO buffer. Data is pushed onto the FIFO when TREADY and TVALID are asserted, in compliance with an AXI transaction. There are three conditions to be met to pop data off the FIFO. First, TREADY must be asserted, indicating that the transmitter is ready for new data. Second, the packet serialiser block must be ready for a new packet. Third, the FIFO must not be empty. There is a further delay circuit to ensure only one packet is output at a time. The resulting blocks can be seen in Figure 25, and intended timing in Figure 26.

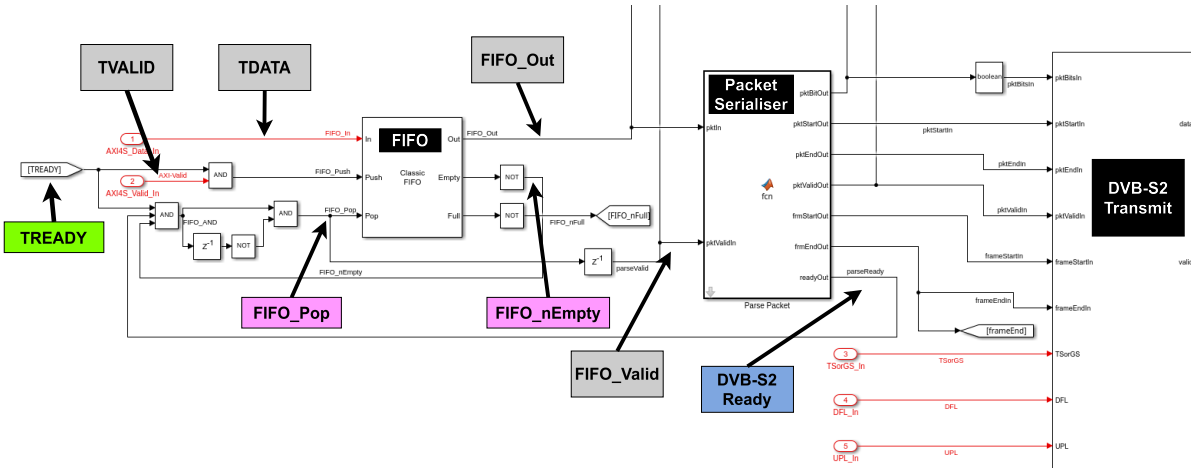


Figure 25: Annotated Packet Input Circuit

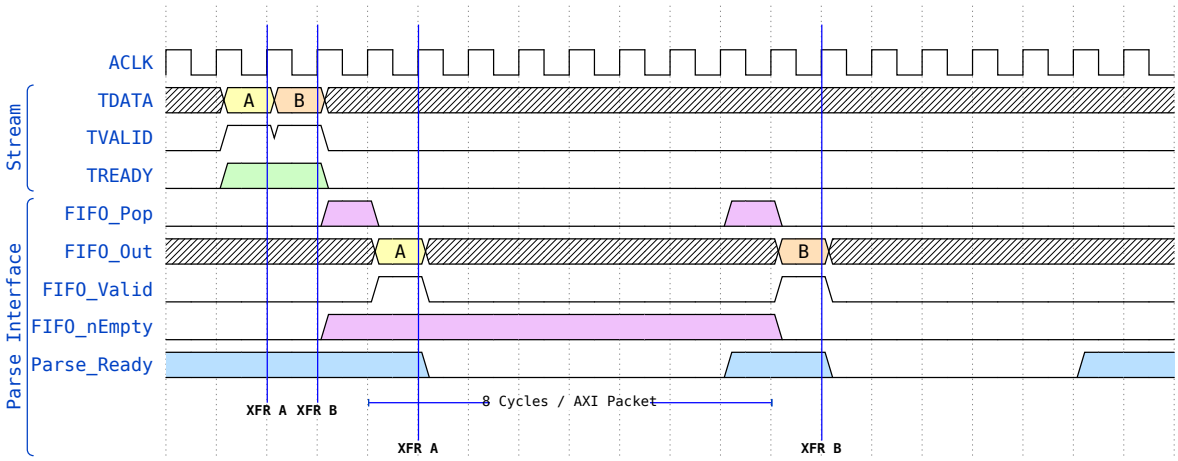


Figure 26: FIFO stream and parser interfacing timing. The input of the FIFO is used to handle the AXI-Stream, with TVALID driving "FIFO_Push". If TREADY and Parse_Ready are asserted, and the FIFO is not empty, then FIFO_Pop is asserted and a packet output on the next clock.

The serialisation block, Parse Packet, was implemented through MATLAB code wrapped in a function block. It takes two inputs, pktIn and pkTVALIDin, and is configurable via a mask. The ports and parameters are defined in Table 17. The block serialises the data as shown in Figure 27. The DVB-S2 standard defines that the data is sent Most Significant Bit (MSB) first, however the option to send data Least Significant Bit (LSB) first was included for completeness.

Table 17: Parse Packet Interface

Name	Type	Purpose
Inputs		
pktIn	uint32	AXI Packet to be serialised
pkTVALIDIn	boolean	Indicates if packet input is valid
Outputs		
readyOut	Boolean	Indicates that the block is ready for new data
pktBitOut	boolean	Output data stream
pktStartOut	boolean	Synchronised sideband signal outputs
pktEndOut	boolean	
pkTVALIDOut	boolean	
frmStartOut	boolean	
frmEndOut	boolean	
Configuration		
LSB First	boolean	If false, data is output MSB first, if true data, is output LSB first.
Frame End Position	unsigned int	Bit position of signals within AXI packet.
Frame Start Position	unsigned int	
Packet End Position	unsigned int	
Packet Start Position	unsigned int	
Payload Position	unsigned int	Bit position of LSB of payload data byte

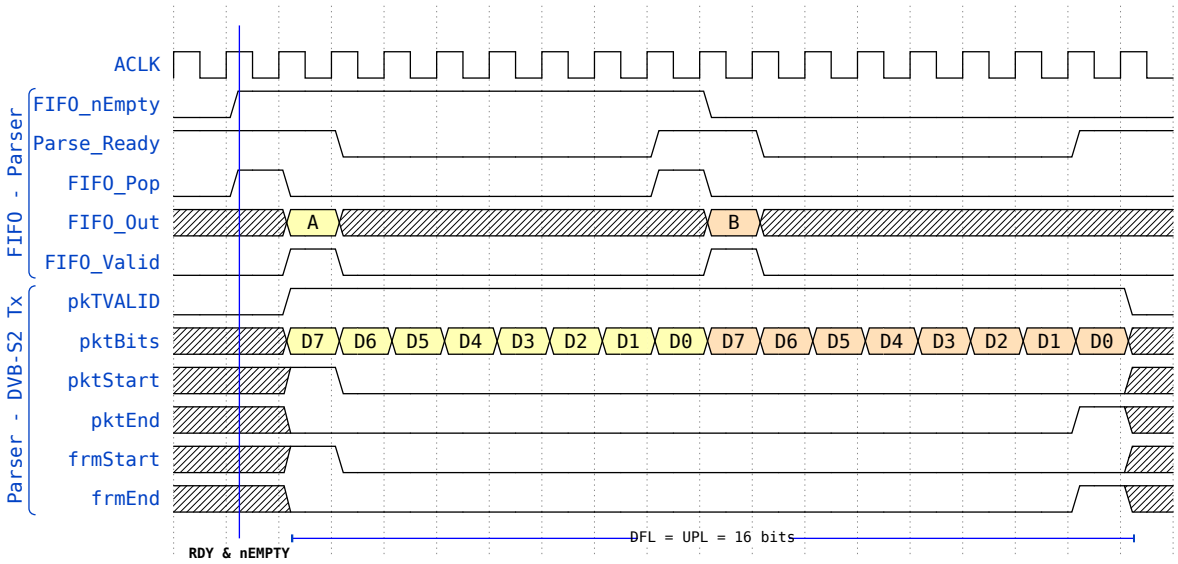


Figure 27: Parser timing. When Parse_Ready is asserted and the FIFO is not empty, the FIFO outputs a packet on the next cycle. The Parser processes the packet when FIFO_Valid is asserted, outputting the MSB of the packet data field along with start flags in the same cycle. Over the next six cycles, the packet data field is output sequentially, ending with the LSB and end flags on the 8th cycle. Parse_Ready is reasserted on the 8th cycle, ensuring no gaps between packets.

The next stage of the design handled the interface between the DVB-S2 Transmitter block and the DAC interface.

Both I and Q outputs are in the form of signed 16 bit words, where only the most significant 12 bits are used and the remaining 4 bits ignored. The DVB-S2 Transmitter block outputs signed fixed point 18 bit complex words with 16 fractional bits, which is more precision than required. The block was tested with all modulation constellations and the maximum magnitude of the output signal was found to be less than 1. This indicated that the bit depth could be safely reduced. To ensure that the full 12 bits of accuracy were maintained only the final block was modified, the root-raised-cosine pulse shaping filter.

With the modification, the output data type was signed 12 bit words with 11 fractional bits. These are then scaled and cast to a signed 16 bit word for output to the DAC. The resulting blocks are shown in Figure 28. The scaling factor is indicated as En11, meaning $\times 2^{-}11$ and E4 meaning $\times 2^{4}$.

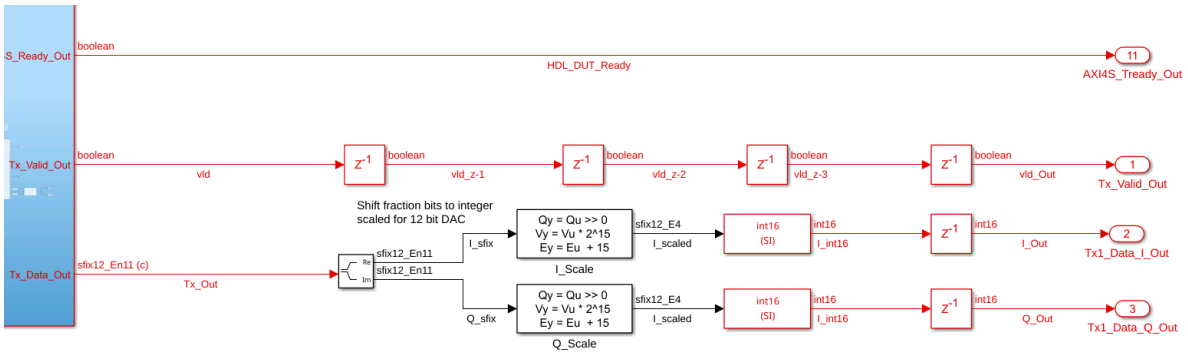


Figure 28: Blocks for scaling of IQ data for transceiver interface

5.2 HDL Generation

Once the PL design was ready, HDL code could be generated. The device target was set as “Zedboard and FMCOMMS2/3/4” and the “Receive and transmit path” reference design selected.[1] The target interface was configured according to Table 18. Most signals were automatically assigned to the interface of the reference design, however the control signals required manual assignment.

The FMCOMMS2 and FMCOMMS3 are based on the AD9361 and AD9363 respectively. These have two channel transmit and recieve, wheras the FMCOMMS4 with its AD9364 only has single channel receive. The controller block is common across all three devices, however in the case of the FMCOMMS4, these extra channel interfaces are nonfunctional. For this reason, there are two receive (Rx) channels defined within the block design, with the second channel tied to “0”.

All of the interface ports for the design are shown in Table 18.

[1]Vivado expected a version of the Zedboard board file that was not present. This required the manual modification of an existing board file before the design could be generated.

Port Name	Port Type	Data Type	Target Platform Interface
PS-PL Transmit			
AXI4S_Data_In	Input	uint32	AXI4-Stream Read Slave
AXI4S_Valid_In	Input	boolean	AXI4-Stream Read Slave
AXI4S_TREADY_Out	Output	boolean	AXI4-Stream Read Slave Ready (optional)
DVB-S2 Control Signals			
TSorGS_In	Input	ufix2	AXI4-Lite
DFL_In	Input	uint16	AXI4-Lite
UPL_In	Input	uint16	AXI4-Lite
SYNC_In	Input	uint8	AXI4-Lite
MODCOD_In	Input	ufix5	AXI4-Lite
FECFRAME_In	Input	boolean	AXI4-Lite
Transceiver Transmit			
Tx_Valid_In	Input	boolean	DMA Tx Valid In
Tx1_Data_I_In	Input	int16	DMA Tx I1 In [0:15]
Tx1_Data_Q_In	Input	int16	DMA Tx Q1 In [0:15]
Tx_Valid_Out	Output	boolean	Baseband Tx Valid Out
Tx1_Data_I_Out	Output	int16	Baseband Tx I1 Out [0:15]
Tx1_Data_Q_Out	Output	int16	Baseband Tx Q1 Out [0:15]
Transceiver Receive			
Rx_Valid_In	Input	boolean	Baseband Rx Valid In
Rx1_Data_I_In	Input	int16	Baseband Rx I1 In [0:15]
Rx1_Data_Q_In	Input	int16	Baseband Rx Q1 In [0:15]
Rx_Valid_Out	Output	boolean	DMA Rx Valid Out
Rx1_Data_I_Out	Output	int16	DMA Rx I1 Out [0:15]
Rx1_Data_Q_Out	Output	int16	DMA Rx Q1 Out [0:15]
Rx2_Data_I_Out	Output	int16	DMA Rx I2 Out [0:15]
Rx2_Data_Q_Out	Output	int16	DMA Rx Q2 Out [0:15]
PL-PS Receive			
AXI4S_TREADY_In	Input	boolean	AXI4-Stream Write Master
AXI4S_Data_Out	Output	uint32	AXI4-Stream Write Master
AXI4S_Valid_Out	Output	boolean	AXI4-Stream Write Master Valid

5.3 Packet Handling

5.3.1 PS-PL Interface

HDL Coder generated the software interface shown in Figure 29, however due to time constraints, this was not tested.

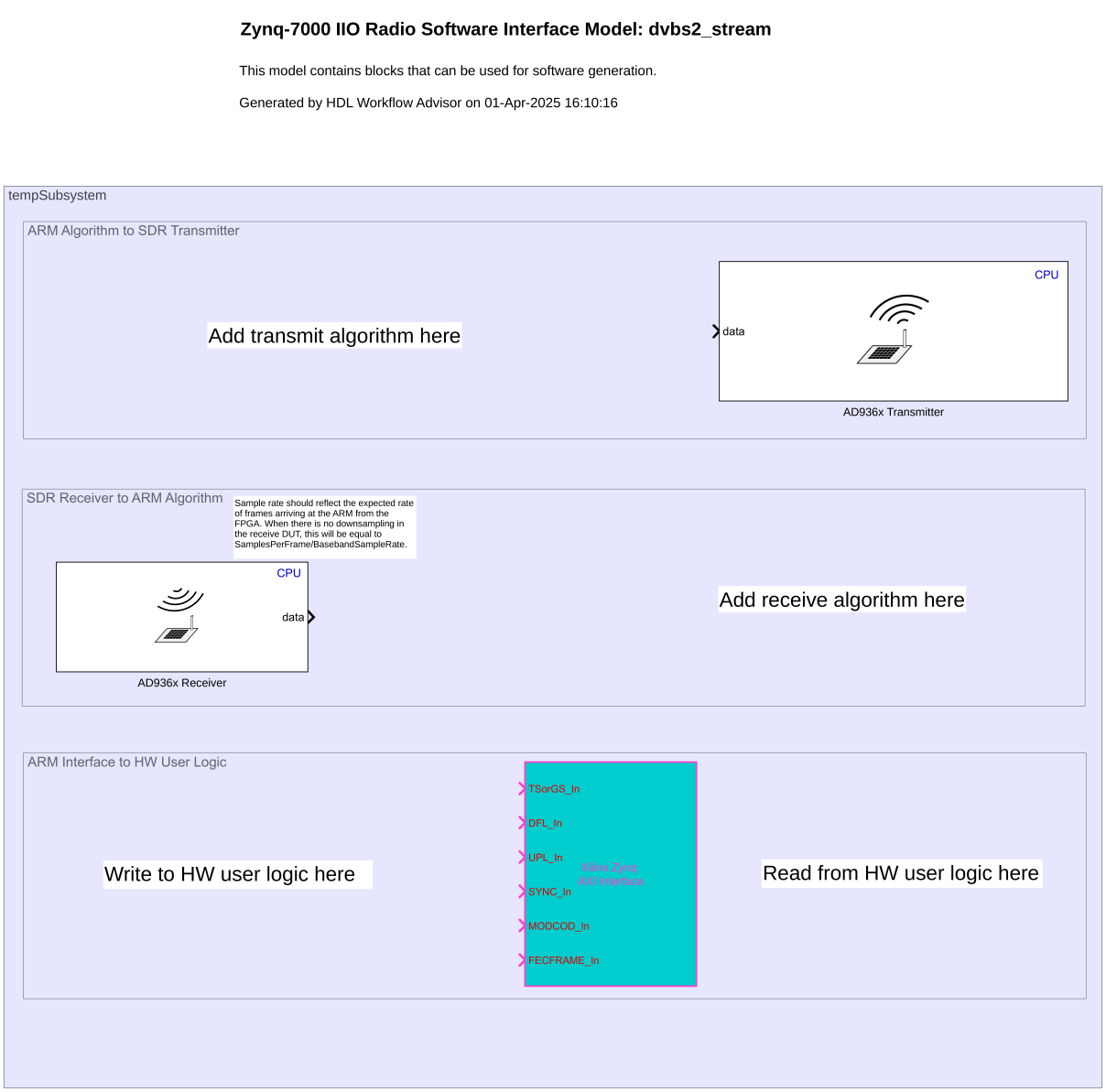


Figure 29: Generated software interface. Packets are transmitted by sending data into the "data" port of the AD936x Transmitter block. This block can also be used to configure the radio, including center frequency.

5.3.2 Test Data Generation

To verify the proper functioning of the transceiver, synthetic packet data was required. Additionally, all sideband signals and control signals would need to be generated appropriately to ensure all inputs were valid.

The test data encoded DVB-S2 frame and packet, as well as AXI packet, counters into the data field to ease debugging of the resulting waveforms. Only 8 bits were available, so these were assigned accordingly: A single frame may contain hundreds of small packets and thousands of AXI packets. These counters would rollover and were zero indexed. The resulting packet structure is shown in Figure 30, and the flow for generation in Figure 31.

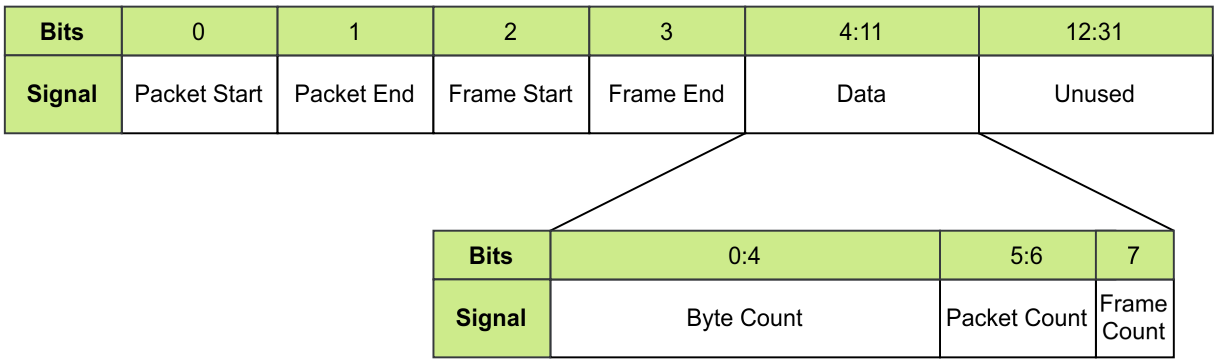
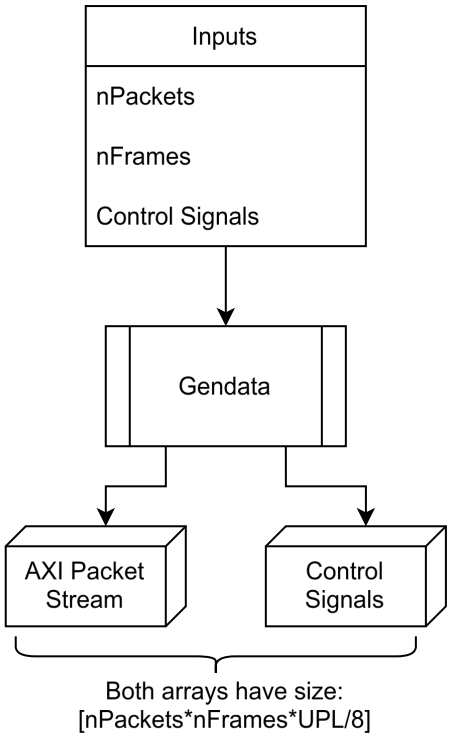
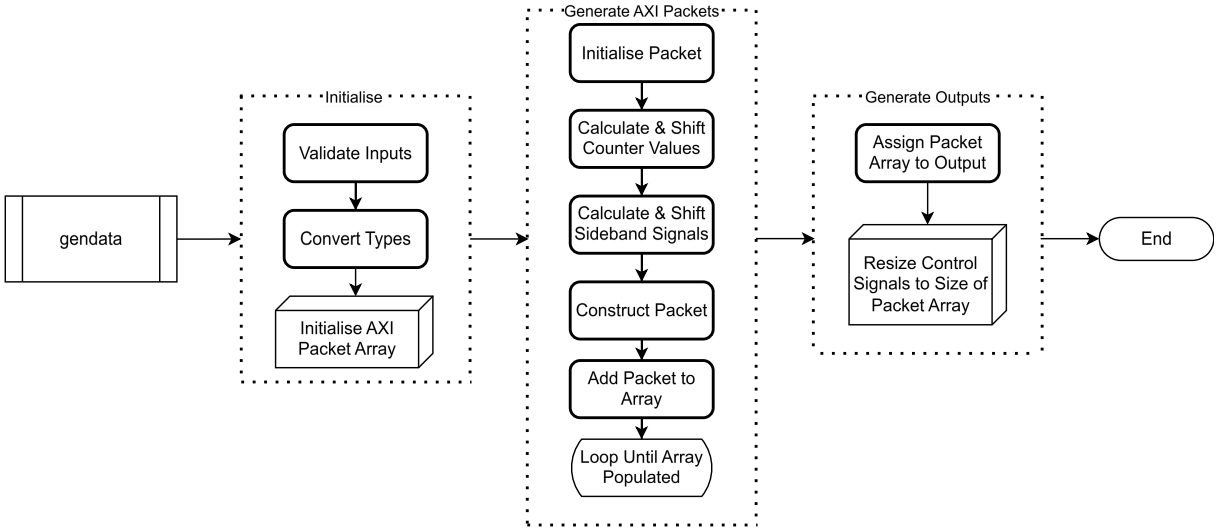


Figure 30: Test Packet Structure



(a) High Level Flow



(b) Low Level Flow

Figure 31: Data generation function structure. (a) Shows the inputs and outputs of the function, where both the AXI packet stream and control signals have the same size. (b) Shows the functionality of the gendata function.

Another function called parse_axipkt was developed to verify that gendata was generating data as expected. This used bit shift operations to parse the generated 32 bit data word into a formatted struct.

5.3.3 GSE

Initial work on the GSE implementation has been completed. C++ was selected for this purpose and various libraries for serialisation were investigated, with the cereal library [32] being considered as candidate. It supported the creation of binary archive files with embedded

flags to make them portable across computer architectures. Ultimately, the conclusion was reached that serialisation will have to be implemented manually, as the libraries found increased overheads to improve compatibility. For this application the exact packet structures are known, as schema definition files will be created ahead of time and loaded onto the satellite.

Listing 2 shows the class definition for a GSE Packet. The fixed header specifies the bit alignmment of the members in order to byte align it properly. std::optional was chosen to handle the missing members of the variable header. The serialise function shall return a byte array that can be passed to the PS-PL interface.

```
class GSE_Packet {
public:
    struct FixedHeader {
        bool start    : 1;
        bool end      : 1;
        int labelType : 2;
        int GSElength : 12;
        };
    struct VariableHeader {
        std::optional<char>     fragID;
        std::optional<uint16_t> totalLength;
        std::optional<uint16_t> protocolType;
        std::optional<char[3]>  label;
    };
private:
    FixedHeader fh;
    std::optional<VariableHeader> vh;
    Packet* pdu;

public:
    // constructor
    GSE_Packet(FixedHeader fh, VariableHeader vh, Packet* pdu);
    std::vector<uint8_t> serialise();
};
```

Listing 2: GSE_Packet Class Definition

During the investigation of packet structure, it decided that all packets shall include both a type and version field. The type field shall occur first such that buffers can be allocated appropriately, and the version field included to ensure that packet structure can be updated without breaking backwards compatibility. For the final system, these fields could be reduced in size, as it is not expected for the packet definitions to change during the mission. However they should not be omitted in order to allow the possibility of updates in the future. Listing 3 shows a possible implementation of a packet class, with the cereal library used for serialisation.

```
class Packet {
    uint8_t type;
    uint8_t version;
    timestamp time;
    virtual cereal::PortableBinaryOutputArchive serialise();
};
```

Listing 3: Packet Interface Definition

Listing 4 shows a possible function definition for fragmentation.

```
std::vector<GSE_Packet> fragment(Packet* pdu);
```

Listing 4: Packet Fragmentation Function Structure

An example primary payload packet is shown in Listing 5. It inherits the members from Packet and creates a definition for serialise().

```
struct PPL_pkt_64k : Packet{
    double standard_deviation;  // Standard deviation of data
    double codebook[16];        // Quantization levels
    uint32_t data[2048];        // compressed PPL data

    cereal::PortableBinaryOutputArchive serialise();
};
```

Listing 5: Primary Payload Packet Definition

6 Implementation and Simulation Results

6.1 Simulation

The system was succesfully simulated and the output waveform measured using a spectrogram block. This was compared against the ITU out-of-band spectrum mask for the amateur and amateur-satellite service, as shown in Figure 32. The green indicates that the waveform succesfully complies. A MATLAB function called spectMask() was created to generate the mask coefficients, the code for which is shown in Listing 6. The mask is defined in terms of necessary bandwidth, which was set to 150 kHz, and transmit power, which was set to 1 W.

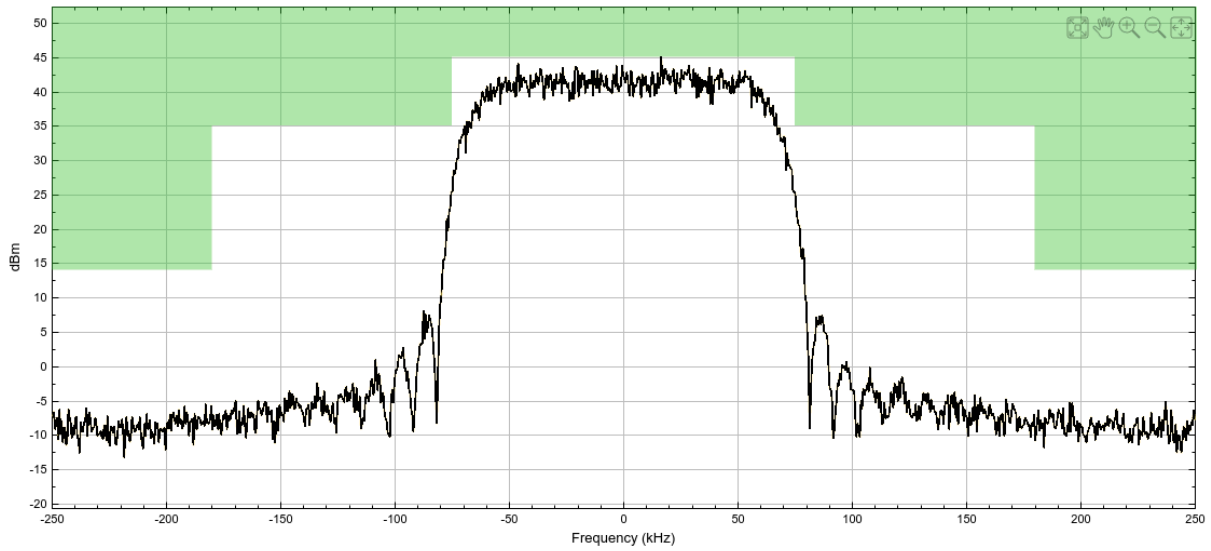


Figure 32: Spectrum at output to DAC with ITU out of band emission spectra mask. Note that the power scale does not reflect the true output power, as the DAC only uses the upper 12 bits.

```
function [mask] = spectMask(Bn, Ptx)
%SPECTMASK Creates mask limit boundaries based on ITU-R SM.1541-7 Fig 43
%   Bn   : Necessary bandwidth (Hz)
%   Ptx  : Transmit power (W)
%   Mask : all dBc
arguments
    Bn  (1,1) = 150e3;
    Ptx (1,1) = 1;
end
    bound_50  = -10;
    bound_120 = -1*(31 + 10*log10(Ptx));
    bound_225 = -1*(38 + 10*log10(Ptx));
    posmask = [0,             0;  ...
               Bn/2,          0;  ...
               Bn/2,   bound_50;  ...
               Bn*1.2, bound_50;  ...
               Bn*1.2, bound_120; ...
               Bn*2.25, bound_120; ...
               Bn*2.25, bound_225; ...
               ];
    negmask = flip([posmask(:,1).*-1, posmask(:,2)],1);
    mask = [negmask;posmask];
end
```

Listing 6: Primary Payload Packet Definition

Late in testing it was found that the transmitter was not functioning correctly with the generated packet inputs. The nextFrame signal did not get reasserted after the first frame was transmitted. Additionally, the flag output did not indicate that valid frames were being transmitted. The waveform observed is shown in Figure 33, and the circuit for the flag signal generation shown in Figure 34.

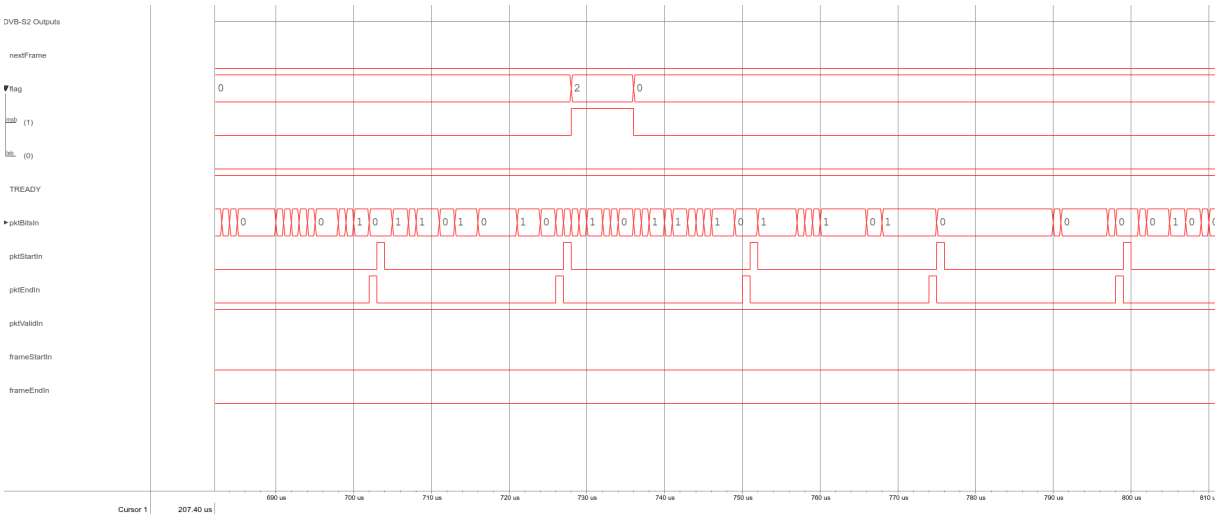


Figure 33: Logic analyser circuit of DVB-S2 inputs and outputs. The input data is being sent, however only the MSB of flag is being set, and never the LSB.

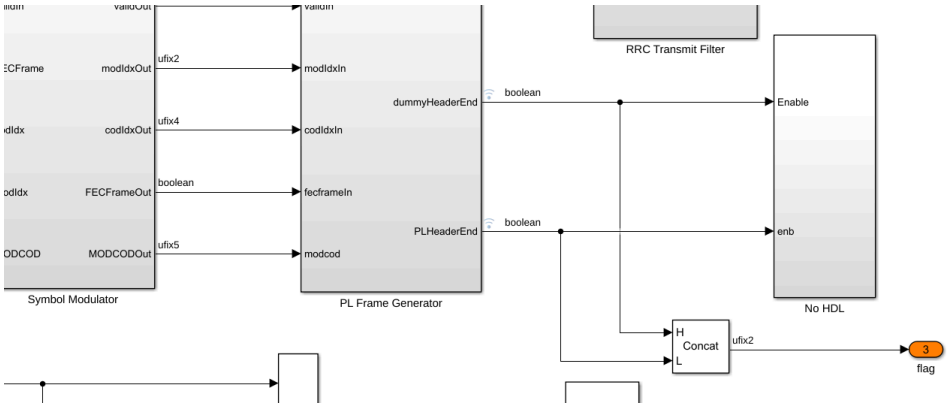


Figure 34: Flag signal generation. The MSB of the flag signal indicates the end of a dummy header, and the LSB the end of a valid PL Header carryig data.

Assertion logic was added to verify that the serialisation of packet data was functioning correctly. The circuit is shown in Figure 35. By modifying the circuit, it was confirmed that packet data was being serialised correctly, however the transmitter was still not functional. The resulting waveform can be seen in Figure 36.

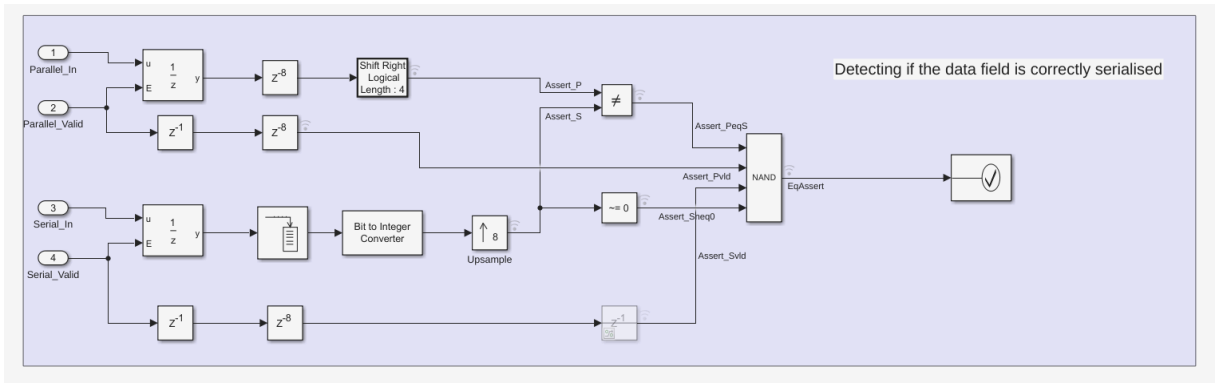


Figure 35: Serialisation verification circuit. The AXI packets are input into Parallel_In, and bit shifted to isolate the payload byte. At the same time, the output of Packet Parser is connected to Serial_In. This is upsampled to match the rates of the two signals with zero padding. The assertion triggers if the parallel and serial are not equal, while both inputs are valid.

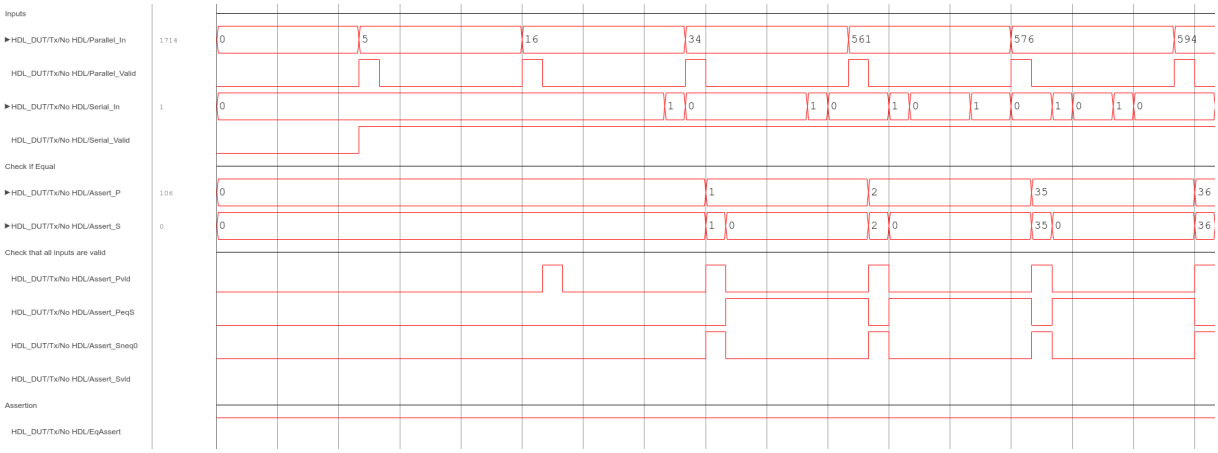


Figure 36: Logic analyser waveform of assertion circuit. The serialised data, Assert_S, can be seen to be identical to the parallel data, Assert_P with zeros inserted. The signals are only not equal during the zero padding, from Assert_Sneq0.

The serialisation circuit was also analysed using a logic analyser, resulting in Figure 37, and found to be functioning correctly.

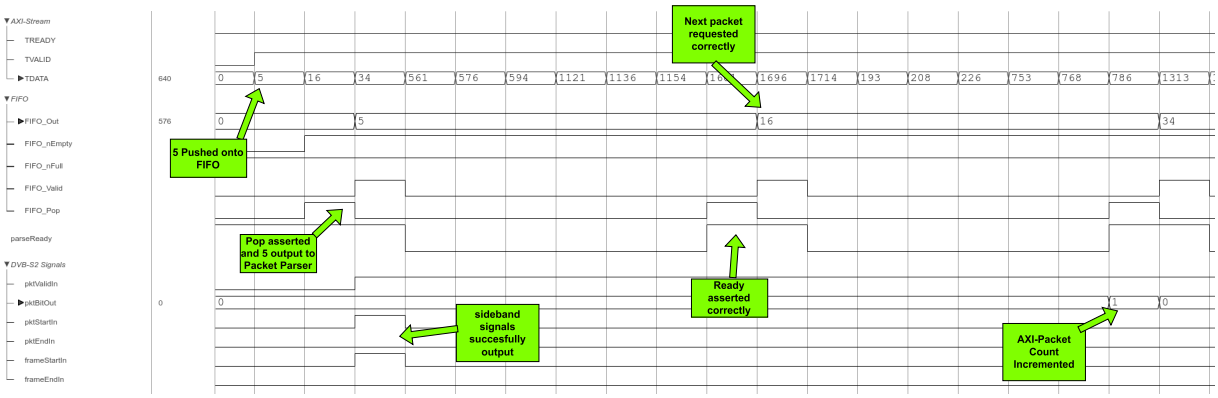


Figure 37: AXI-Stream Interface Logic Analyser Waveform

Unfortunately, the issue with the transmitter interface was not able to be corrected before the end of the project, however the design was still able to be exported to Vivado to test feasibility.

6.2 Code Generation & Hardware Implementation

The HDL Coder design was exported to Vivado. The generated block diagram is shown in Figure 38. This design was succesfully implemented and synthesised. The resource report showed that the design fit onto the chip, however there was very little Block RAM left over. The usage by subsystem and resource type is shown in Figure 39. The placed and routed design on the chip is shown in Figure 40. The design also succesfully passed timing, with the slack shown Table 19.

Table 19: Timing Analysis Results

Worst Negative Slack	Worst Hold Slack	Worst Pulse Width Slack
0.126 ns	0.016 ns	0.264

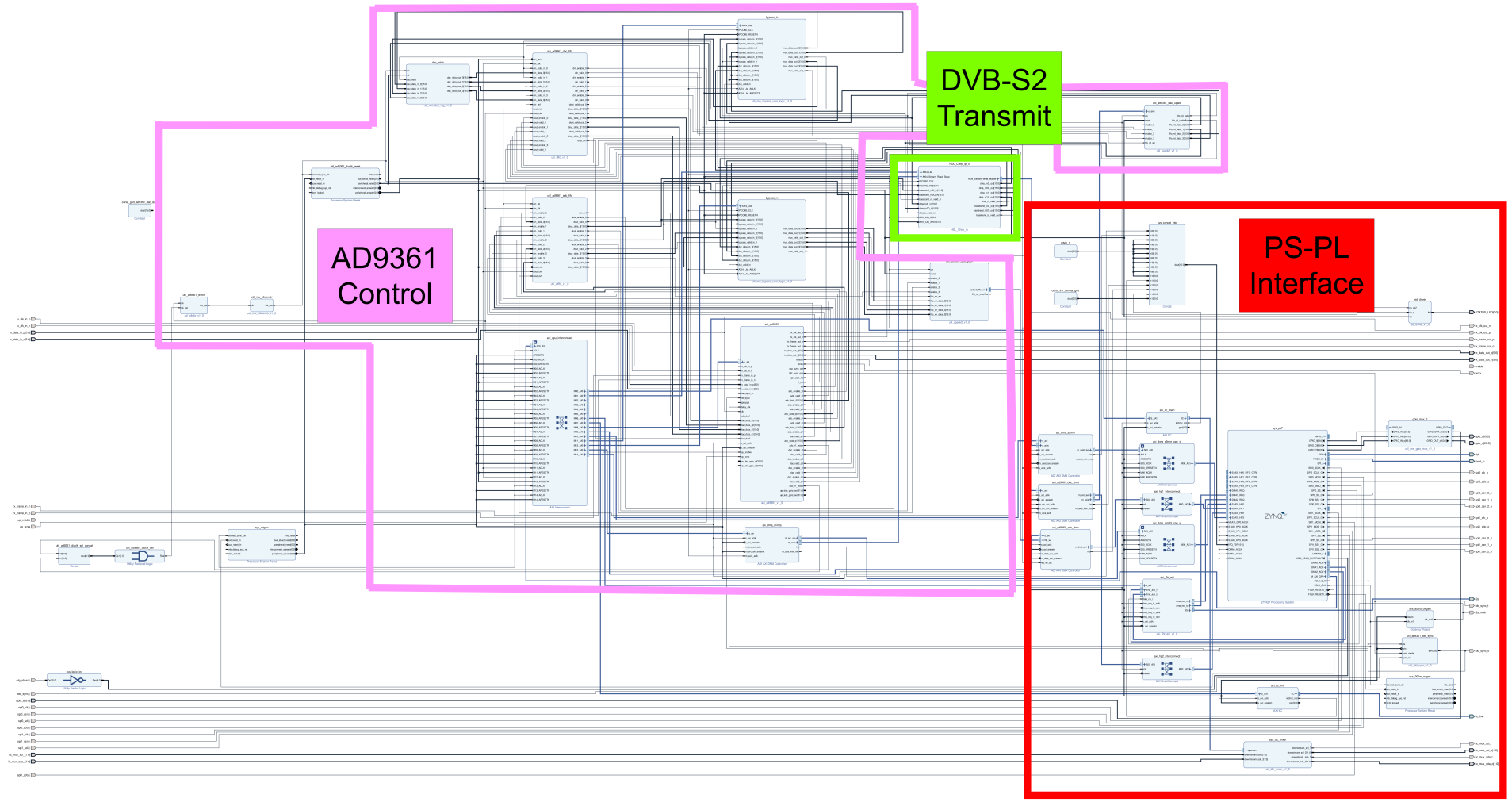


Figure 38: Generated Vivado IP Block Diagram. Most blocks are used for interfacing with the transceiver, or the PS.

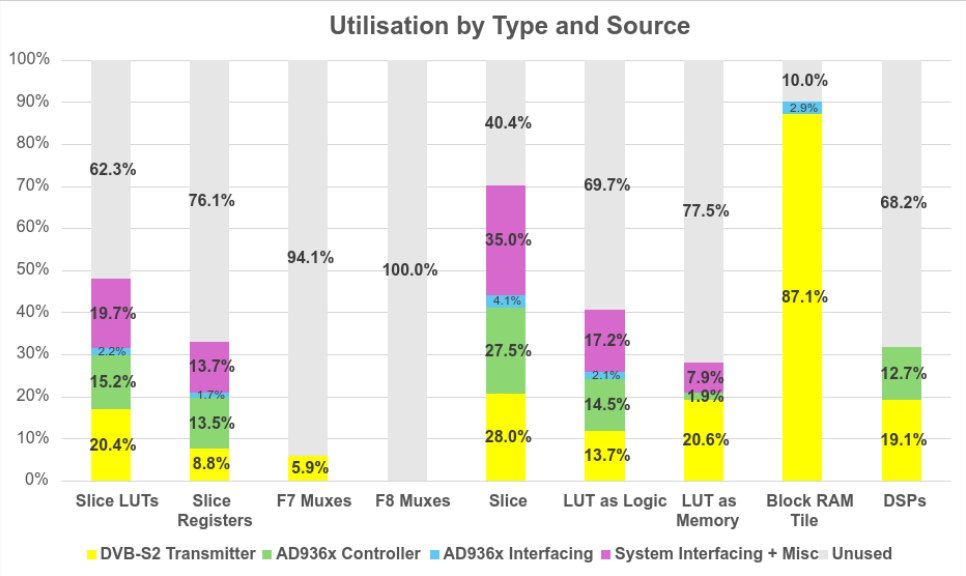


Figure 39: Utilisation of implemented design by subsystem and type. The DVB-S2 transmitter block requires relatively little resources, with the exception of Block RAM, of which it uses 87.1%.

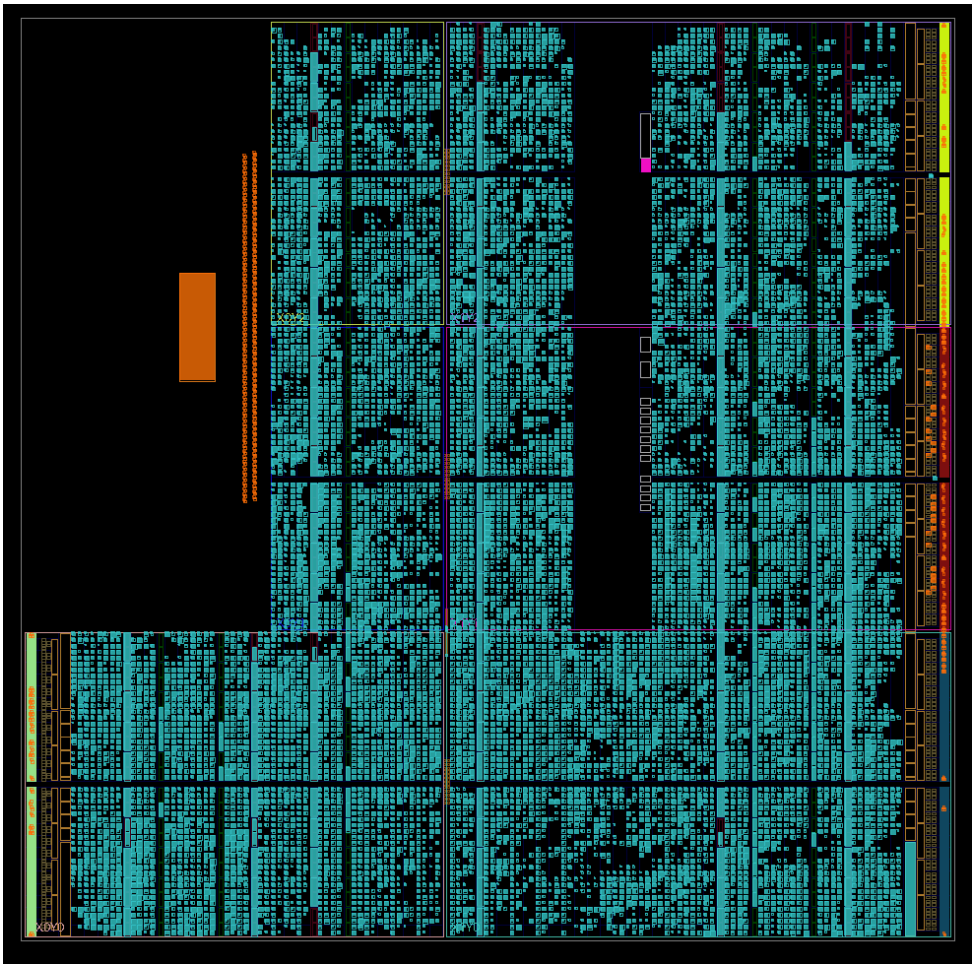


Figure 40: Implemented design on Zynq 7020 SoC FPGA. Areas coloured in blue are FPGA resources used by the design.

7 Discussion

The transmitter complies to the spectral mask for all tested inputs. This also held true for the DVB-S2 transmitter block when tested in the original example code, ensuring that the observed interfacing issues did not have an effect.

The logic analyser waveform of the AXI-Stream interfacing logic was found to match extremely well to the intended timing diagram, as the measured waveform matches the timing shown in Figure 27 clock for clock. Further, the implemented assertion proved that the data field was serialised correctly, as it did not trigger during the simulation. This indicates that further investigation of the original DVB-S2 HDL Coder example design to ensure that the designed system is appropriate.

The resource usage of the implemented design was found to comply with the goal of no more than 50% usage on any one resource, with the exception of Block RAM. High RAM usage is to be expected for a HDL Coder design, as extra buffering is used between blocks. In order to integrate the PBR design with the downlink implementation, this may require further optimisation.

The design was found to meet timing, however this was by fractions of a nanosecond in all cases. The clock period was around 15 ns, so this is an extremely small margin. The clock tolerance settings were left at default, so further analysis will be required to determine if this margin is large enough. The synthesis settings could also be modified to allow the solver to run for more iterations to potentially find a better place and route solution.

With reference to Figure 40, it is clear that the design is extremely spread out over the chip, meaning long chains of logic, which likely impacted timing closure. Therefore, optimisation of the Block RAM usage might also improve the timing of the design, allowing higher clock rates to be used.

7.1 Further Work

The work on the downlink communications has several avenues for further development. The first priority, is to fix the interfacing issues with the transmitter block. Another avenue for work is testing the design in hardware using development boards. This is expected to take some time to complete, as the AXI packet handling code has not been fully verified to ensure compliance to the protocol. The packet handling systems could also be worked upon, with the investigation of the ACM routing being of particular importance.

Once the system is operational, the addition of DVB-S2X features could be of interest. The VL-SNR features could improve system performance considerably at low elevations. The impact of interference is also yet to be analysed in the context of the STRATHcube mission, which may necessitate the link budget to be reassessed.

As part of the ESA Baseline Design Review, several work packages were created that are relevant to the future of this project. They include interference analysis of the ground station site and creation of a receiver for the downlink communications. These are detailed in Appendix B.

An extended abstract regarding the work completed so far has been created and submitted to the European Data Handling & Data Processing Conference for consideration, as shown

in Appendix B. If accepted, further development on hardware implementation and packet handling will be conducted as part of that work.

8 Conclusion

The development of the downlink communications system has now been significantly furthered over the course of this project. A detailed review of CubeSat communications and the context of the STRATHcube mission was conducted to ensure that the design was feasible, and to identify design requirements. One of the main requirements identified was the maximisation of data transmitted over the mission, to allow the thorough testing of the primary payload.

ACM was analysed with this context to identify the possible performance uplift over the course of the mission. MATLAB code was created for this analysis and integrated into internal tools. The investigation found that ACM offered significant benefits and that it was worth the extra system complexity.

A system architecture was then defined, with all key blocks for packet handling detailed and relevant tools and standards identified. An initial implementation of the transmitter system was created, with further work required on interfacing for the design to be fully functional. Initial code was developed for a GSE implementation, with further work required.

9 Bibliography

[1] Thomas McIlwraith, Daniel Betoin, Cameron Fergus, Freya Groves, Aidan Bennis, Hannah Douglas, Sami Rahman, Ethan Dyer, Louis Galasso, "STRATHcube CubeSat Project File." STRATHcube, Mar. 2025.

[2] Lewis Creed, Ciarán Jenkins, Sam Kirk, Douglas McGarrity &, Gary Stewart, and Julie Graham, "STRATHcube: A CubeSat that Encourages the Sustainable Usage of Space." 2021.

[3] GAUSS Srl, "GAUSS Low Power UHF Radio Datasheet." 2018.

[4] Pablo Alvarez-Icaza, Matthew Robert Norrie, Niamh O'Neill-Berest, Evie Clark, and Maria Nepheli Kardassi, "Advancing STRATHcube Towards Consolidated Design." Mar. 2024.

[5] Alén Space, "TOTEM Motherboard Datasheet." Accessed: Dec. 11, 2024. [Online]. Available: https://alen.space/products/totem-sdr/

[6] Alén Space, "FrontendUHF Datasheet." Alén Space, Apr. 2021.

[7] ISISPACE, "CubeSat Antenna System for 1U/3U - ISISPACE." Accessed: Feb. 28, 2025. [Online]. Available: https://www.isispace.nl/product/cubesat-antenna-system-1u-3u/

[8] Preben Rasamanickam, "STRATHcube Downlink Link Budget Background." Aug. 2024.

[9] E. Amarantidou, "AcubeSAT UHF Link Budget Analysis." Accessed: Mar. 12, 2025. [Online]. Available: https://gitlab.com/acubesat/comms/budgets/link-budget-analysis/-/blob/master/CDR%20budgets/UHF_link.xlsx?ref_type=heads

[10] Analog Devices, "AD9364 Datasheet." Accessed: Apr. 03, 2025. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad9364.pdf

[11] "ZedBoard Product Brief." Accessed: Dec. 11, 2024. [Online]. Available: https://www.avnet.com/wps/wcm/connect/onesite/a43adf00-158c-4614-b2c7-4a7f35b53f25/FY23_800_ZedBoard_Product_Brief_r2.pdf?MOD=AJPERES&CACHEID=ROOTWORKSPACE.Z18_NA5A1I41L0ICD0ABNDMDDG0000-a43adf00-158c-4614-b2c7-4a7f35b53f25-oj5IUo.&srsltid=AfmBOorWAay42Ouz38BFG7Q_qeN0xz9oDGqfjOMLLDh7Vcjr2rLG2in7

[12] "AD-FMCOMMS4-EBZ Evaluation Board | Analog Devices." Accessed: Mar. 24, 2025. [Online]. Available: https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcomms4-ebz.html

[13] "AD-FMCOMMS1-EBZ User Guide [Analog Devices Wiki]." Accessed: Dec. 11, 2024. [Online]. Available: https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms1-ebz

[14] Calum Clarke, Daniel Aitken, Craig Sutherland, and Hannah Wright, "Strathclyde Spacecraft Tracking & Command Station." 2016.

[15] SSB, "SP-7000 Datasheet." Accessed: Mar. 14, 2025. [Online]. Available: https://www.ssb.de/shop/download/EN-1050_SP-7000.pdf

[16] M2 Antenna Systems Inc, “436CP30 Antenna Datasheet.” Accessed: Mar. 21, 2025. [Online]. Available: https://www.m2inc.com/content/PDF%20MANUALS/70CMANTS/436CP30MAN04-W.pdf

[17] M. Eroz, F.-W. Sun, and L.-N. Lee, “DVB-S2 low density parity check codes with near Shannon limit performance,” *International Journal of Satellite Communications and Networking*, vol. 22, no. 3, pp. 269–279, 2004, doi: 10.1002/sat.787.

[18] ETSI, “TS 102 606-1 - V1.2.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE); Part 1: Protocol.” Jul. 2014.

[19] E. Grayver, A. Chin, J. Hsu, S. Stanev, D. Kun, and A. Parower, “Software defined radio for small satellites,” in *2015 IEEE Aerospace Conference*, Mar. 2015, pp. 1–9. doi: 10.1109/AERO.2015.7118901.

[20] G. Quintana-Diaz, D. Nodar-López, A. González Muíño, F. Aguado Agelet, C. Cappelletti, and T. Ekman, “Detection of radio interference in the UHF amateur radio band with the Serpens satellite,” *Advances in Space Research*, vol. 69, no. 2, pp. 1159–1169, Jan. 2022, doi: 10.1016/j.asr.2021.10.017.

[21] M. H. Same, G. Gandubert, P. Ivanov, R. Landry, and G. Gleeton, “Effects of Interference and Mitigation Using Notch Filter for the DVB-S2 Standard,” *Telecom*, vol. 1, no. 3, pp. 242–265, Dec. 2020, doi: 10.3390/telecom1030017.

[22] B. Sklar, *Digital communications: fundamentals and applications*, 2. ed., 17. print. Upper Saddle River, NJ: Prentice Hall PTR, 2009.

[23] Ettus Research, “B210 Spec Sheet.” Accessed: Mar. 14, 2025. [Online]. Available: https://www.ettus.com/wp-content/uploads/2019/01/b200-b210_spec_sheet.pdf

[24] MathWorks, “MATLAB slantRangeCircularOrbit Documentation.” Accessed: Mar. 14, 2025. [Online]. Available: https://uk.mathworks.com/help/satcom/ref/slantrangecircularorbit.html

[25] ETSI, “EN 302 307-1 - V1.4.1 - Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1: DVB-S2.” Jul. 2014.

[26] CCSDS, “CCSDS Protocols over DVB-S2—Summary of Definition, Implementation, and Performance.” Jun. 2023.

[27] MathWorks, “Satellite Communications Toolbox.” Accessed: Dec. 13, 2024. [Online]. Available: https://uk.mathworks.com/products/satellite-communications.html

[28] J. P. I. De Mora, Aniol Martí, R. Iudica, D. Evans, and V. Zelenevskiy, “Image and radio-frequency data compression for OPS-SAT using FAPEC,” Sep. 2022, doi: 10.5281/ZENODO.7244826.

[29] T. M. Inc, “HDL Coder (R2024b).” [Online]. Available: https://www.mathworks.com/

[30] MathWorks, “DVB-S2 HDL Transmitter.” Accessed: Dec. 11, 2024. [Online]. Available: https://uk.mathworks.com/help/satcom/ug/dvbs2-hdl-transmitter.html

[31] T. M. Inc, “HW/SW Co-Design with AXI4-Stream Using Analog Devices AD9361/AD9364.” [Online]. Available: https://uk.mathworks.com/help/soc/ug/hwsw-co-design-with-axi4-stream-using-analog-devices-ad9361-ad9364.html

[32] USC iLab, “USCiLab/cereal.” Accessed: Apr. 04, 2025. [Online]. Available: https://github.com/USCiLab/cereal

Appendix A: Code Listings

A.1. ACM Analysis

A.1.1 datarate.m

```
function [rates] = datarate(elevation,altitude,bandwidth,margin,verbose,
link_parameters)
%DATARATE Calculates datarate for each elevation and altitude pair
% Returns:
%   rates:              Datarates for each      bits/sec, array
size
%                       elevation & altitude    length(elevation)
%
% Parameters:
%   elevation:          Satellite Elevation     degrees,    can be array
%   altitude:           Satellite Altitude      meters,     can be array
(must be same length as elevation)
%   bandwidth:          Link Bandwidth          Hz          Optional
%   margin:             Link Margin Required    dB          Optional
%   link_parameters:    Static link values      struct      Optional
%
% Requires modcod_to_CNR.mat to be in same directory. This contains
% spectral efficiency values for each MODCOD

arguments
    elevation   (1,:)
    altitude    (1,:)
    bandwidth   (1,1) = 158.5*1000
    margin      (1,1) = 10
    verbose     (1,1) = false
    link_parameters struct = struct( ...
                                "Frequency",437e6, ...
% Hz
                                "GS_Altitude", 41, ...
% m
                                "EIRP", 0, ...
% dBW
                                "Receiver_Gain",14.15-2.39, ...
% dB
"Receiver_Noise_Temperature",282.6592, ...  % K
                                "Atmo_Effect_Path_Loss",3.56 ...
% dB
                                );


end
    % Validate Inputs
    if length(elevation) ~= length(altitude)
        error("Elevation and Altitude must have same length")
    end

    % Atmospheric losses
    APL = zeros(1,length(elevation));
    for elevation_it = 1:length(elevation)
```

```
        % Acubesat absorption values
        if elevation(elevation_it) < 2.5 || elevation(elevation_it) > 180
- 2.5
            Atmo_Absorption_Loss = 10.2;
        elseif elevation(elevation_it) < 5 || elevation(elevation_it) >
180 - 5
            Atmo_Absorption_Loss = 4.6;
        elseif elevation(elevation_it) < 10 || elevation(elevation_it) >
180 - 10
            Atmo_Absorption_Loss = 2.1;
        elseif elevation(elevation_it) < 30 || elevation(elevation_it) >
180 - 30
            Atmo_Absorption_Loss = 1.1;
        elseif elevation(elevation_it) < 45 || elevation(elevation_it) >
180 - 45
            Atmo_Absorption_Loss = 0.4;
        else
            Atmo_Absorption_Loss = 0.0;
        end

        APL(elevation_it) = Atmo_Absorption_Loss +
link_parameters.Atmo_Effect_Path_Loss;


    end
    slant_ranges = zeros(1,length(altitude));
    for slant_it = 1:length(altitude)
        slant_ranges(slant_it) =
slantRangeCircularOrbit(elevation(slant_it),  ...

altitude(slant_it),   ...

link_parameters.GS_Altitude ...
                                                              );
    end
    free_space_path_loss =
20*log10(4*pi*slant_ranges*link_parameters.Frequency/299792458); % dB
    CNR =  link_parameters.EIRP - free_space_path_loss - APL + ...
            link_parameters.Receiver_Gain + 228.6 -
10*log10(link_parameters.Receiver_Noise_Temperature) ...
            - 10*log10(bandwidth);

    persistent modcodvalues
    if isempty(modcodvalues)
        modcodvalues = load("modcod_to_CNR.mat","mdvals").mdvals;
    end
    modcodvalues.("datarateбps") = modcodvalues.("SpectralEfficiency") *
bandwidth;


    modcodvalues.("CNR_min") = modcodvalues.("EbN0_min") +
10*log10(modcodvalues.("datarateбps")) - 10*log10(bandwidth);

    rates = zeros(1,length(CNR));
    for cnr_it = 1:length(CNR)
```

```
            modcod_row = modcodvalues(find(modcodvalues.CNR_min <
(CNR(cnr_it) - margin),1,"last"),:);
            if verbose
                modcod_row
            end
            % Check if link closes
            if height(modcod_row) == 0
                rates(cnr_it) = 0;
            else
                rates(cnr_it) = modcod_row.datarateBps;
            end

        end
        if verbose
            slant_ranges
            free_space_path_loss
            CNR
        end
end
```

A.1.2 ACM_Analysis.m

```
% Analysing the effect of Active Coding and Modulation (ACM) for DVB-S2
% By Daniel Stebbings
% Based upon "dynamic_link_schemes_analysis" by Preben

%% Imports

load("modcod_to_CNR.mat")
% Ideal CNR calculated from values for normal FECFRAME, Pilotless given
by:
% Table 13 EN 302 307-1 - V1.4.1
%
% Spectral efficiency from normal FECFRAME, w. pilots given by:
% Table 3-1 CCSDS 130.12-G-2
%
% This means that these values for throughput are lower than the maximum
% possible.

%% Satellite Parameters
% ESA FYS! constrains antenna power output to 1.5W. ISIS Antenna max
power
% output is 2W
p_tx = 10*log10(1); % dBW
% Estimate of line losses in satellite telecom system. Based on numbers
% from AMSAT Link Budget Calculator
line_loss_tx = 1.26; % dB
% ISIS antenna spec
gain_antenna_tx = 0; % dBi
% assumption as the ISIS antenna's beamwidth is unknown
antenna_pointing_loss_tx = 0; % dB

EIRP = p_tx - line_loss_tx + gain_antenna_tx - antenna_pointing_loss_tx;

%% Downlink transmission configuration
```

```
bandwidth = 158.5*1000; % 158.5kHz BW defined by link budget investigation
freq = 437e6; % Hz

%% Ground Station Parameters
gain_antenna_rx = 14.15; % dB STAC
gain_lna_rx = 20; % dB STAC
line_loss_rx = 0.2852; % dB AcubeSAT
other_line_losses_rx = 2.1; % dB AcubeSAT
% System noise temperature
temperature_antenna_rx = 154; % K AcubeSAT
% reference temperature, usually 290 K
temperature_feedline_rx = 290; % K Sklar
temperature_lna_rx = 66.8; % K STAC
temperature_frontend_rx = 1539.8; % K STAC
cable_loss = 1.023; % AcubeSAT
coeff_transmission_line = 0.6331; % AcubeSAT

temperature_system_noise_rx = coeff_transmission_line*temperature_antenna_rx + (1-coeff_transmission_line)*temperature_feedline_rx + (temperature_frontend_rx*cable_loss)/gain_lna_rx;

%% Path Losses
altitude_sat = [170e3,409e3]; % highest and lowest, m
% Approximate altitude of James Weir
altitude_gs = 41; % m
lowest_elevation = 10;
elevation_angles = lowest_elevation:5:90; % deg

% direct distance between GS and satellite
slant_ranges = zeros(1,length(altitude_sat));
for alt_it = 1:length(altitude_sat)
    for elev_it = 1:length(elevation_angles)
        slant_ranges(alt_it,elev_it) = slantRangeCircularOrbit(elevation_angles(elev_it), altitude_sat(alt_it), altitude_gs);
    end
end

% FSPL dependent on elevation and altitude
free_space_path_loss = 20*log10(4*pi*slant_ranges*freq/299792458); % dB

% Other Path losses weakly dependent, take as constant
ionospheric_space_loss = 0.4; % dB
atmospheric_space_loss = 1.1; % dB
rain_space_loss = 0; % dB
scintillation_space_loss = 0.16; % dB
polarisation_space_loss = 3; % dB
path_loss = ionospheric_space_loss + atmospheric_space_loss + rain_space_loss + scintillation_space_loss + polarisation_space_loss;



%% System Performance
CNR   = EIRP - free_space_path_loss - path_loss + gain_antenna_rx + 228.6 - 10*log10(temperature_system_noise_rx) - 10*log10(bandwidth) - 2.9;
```

```
Margin_Requirement = 10; %dB

%% Optimal MODCOD from CNR by Elevation and Altitude

% Creating a table by flattening each value array into columns
elevation_col = repmat(elevation_angles,[1,length(altitude_sat)])';
alt_col = repelem(altitude_sat,
length(elevation_angles)*ones(length(altitude_sat),1))';
CNR_col = reshape(CNR',[],1);

% Finding optimal modulation and coding from CNR
opt_modcod_col = repelem("",length(CNR_col));
opt_bitrate = zeros(length(CNR_col),1);
for cnr_it = 1:length(CNR_col)
    % Find fastest modulation with minimum CNR below current CNR w.
margin
    modcod_row = mdvals(find(mdvals.CNR_min < (CNR_col(cnr_it) -
Margin_Requirement),1,"last"),:);
    opt_modcod_col(cnr_it) = strcat(string(modcod_row.Modulation)," ",
string(modcod_row.CodingRate));
    opt_bitrate(cnr_it) = modcod_row.dataratebps;
    %[CNR_col(cnr_it),modcod_row.CNR_min] % Debug
end

% Change type for plotting
modcod_cats = strcat(string(mdvals.Modulation),"
",string(mdvals.CodingRate));
opt_modcod_col = categorical(opt_modcod_col,modcod_cats);


opt_modcod_tab = table( ...

elevation_col,alt_col,CNR_col,opt_modcod_col',opt_bitrate, ...
            'VariableNames',
["Elevation","Altitude","CNR_dB","Optimal_MODCOD","Optimal_Bitrate_bps"] ...
            );

%% Analysing MODCOD Benefit with ISS orbit over 1 month

% Setup Satellite Scenario with iss
simlength = 30; % days
sc_start = datetime(2024,11,12,12,0,0); % 12:00, 12/11/2024
sc_stop  = sc_start + hours(24*simlength);     %
sc_coarse_sample = 120; % seconds / sample
sc_fine_sample   = 1;   % seconds / sample
sc_coarse   = satelliteScenario(sc_start,sc_stop,sc_coarse_sample);
sc_fine     = satelliteScenario(sc_start,sc_stop,sc_fine_sample);
sat_coarse  = satellite(sc_coarse,  "iss_tle.txt");
sat_fine    = satellite(sc_fine,    "iss_tle.txt");

% Setup James Weir Ground Station
gs_name = "James Weir Ground Station";
gs_lat  = 55.86194444;
gs_long = 4.24527778;
gs_coarse =
groundStation(sc_coarse,"Name",gs_name,"Altitude",altitude_gs,"Latitude",gs_lat,"Lon
```

```
gs_fine   = groundStation(sc_fine,"Name",gs_name,"Altitude",altitude_gs,"Latitude",gs_lat,"Longi

% Use coarse sample to quickly find passes
ac_coarse     = access(sat_coarse,gs_coarse);
passes_coarse = accessIntervals(ac_coarse);

% Use fine sampling to get accurate elevations per pass
pass_elevations = zeros( ... % Initialise dimensions
                        height(passes_coarse), ...
                        max(passes_coarse.Duration), ...
                        "uint16" ...
                        );

for pass_it = 1:height(passes_coarse)
    pass = passes_coarse(pass_it,:);
    for time_it = 1:pass.Duration
        [~,pass_elevation_val,~] = aer(gs_fine,sat_fine,[pass.StartTime + seconds(time_it -1)]);
        pass_elevations(pass_it,time_it) = uint16(pass_elevation_val);
    end
end


%% Analysing Predicted Pass Data
% Bin into 5 degree angle bins
elevation_times = histcounts(pass_elevations(:),[lowest_elevation:5:95]);

% Time in each modcod at high altitude

time_modcod_tab = opt_modcod_tab(opt_modcod_tab.Altitude == altitude_sat(end),:);
time_modcod_tab.("Elevation_Time") = elevation_times';
time_modcod_tab.("Bits_Sent") = time_modcod_tab.Optimal_Bitrate_bps .* time_modcod_tab.Elevation_Time;

bits_sent_ACM = sum(time_modcod_tab.Bits_Sent);
bits_sent_fixed = time_modcod_tab(time_modcod_tab.Elevation == lowest_elevation,"Optimal_Bitrate_bps").Optimal_Bitrate_bps .* sum(time_modcod_tab.Elevation_Time);

%% Analysing MODCOD Benefit with IST orbit
% pass_elevations(time(s),altitude (km), pass_flag (bool), elevation (degrees),
ist_pass_data = load("IST_Sim_data.mat").outputs;
ist_altmin = 170;  % Minimum altitude to simulate
ist_timestep = 10; % seconds
% Find passes where flag is 1 and altitude > minimum
ist_passes = ist_pass_data(ist_pass_data(:,3)==1 & ist_pass_data(:,2)>ist_altmin,:);
ist_pass_times      = ist_passes(:,1) - ist_passes(1,1); % seconds
ist_pass_altitudes  = ist_passes(:,2)*1000;     % in meters
ist_pass_elevations = ist_passes(:,4);          % degrees
% Bin into 5 degree angle bins
ist_pass_elevations_binned = floor(ist_pass_elevations/5)*5;
% Get capacity for each timestep
ist_pass_capacities = datarate(ist_pass_elevations,ist_pass_altitudes);
```

```
% Time in each modcod at high altitude

ist_time_modcod_tab = opt_modcod_tab(opt_modcod_tab.Altitude ==
altitude_sat(end),:);
ist_time_modcod_tab.("Elevation_Time") = elevation_times';
ist_time_modcod_tab.("Bits_Sent") =
time_modcod_tab.Optimal_Bitrate_bps .* time_modcod_tab.Elevation_Time;

ist_bits_sent_ACM = sum(ist_time_modcod_tab.Bits_Sent);
bits_sent_fixed = time_modcod_tab(time_modcod_tab.Elevation ==
lowest_elevation,"Optimal_Bitrate_bps").Optimal_Bitrate_bps .*
sum(time_modcod_tab.Elevation_Time);



%% Plot Elevation vs MODCOD by Altitude
figure
hold on
for alt_it = 1:length(altitude_sat)
    CNR_vals = opt_modcod_tab(opt_modcod_tab.Altitude ==
altitude_sat(alt_it),:);

    yyaxis left
    plot(CNR_vals.Elevation,CNR_vals.Optimal_MODCOD, ...
        "DisplayName",sprintf('%.3g',altitude_sat(alt_it),'m'));

    yyaxis right
    plot(CNR_vals.Elevation,CNR_vals.Optimal_Bitrate_bps/1e3, ...
        "DisplayName",sprintf('%.3g',altitude_sat(alt_it),'m'));
end

legend
xlabel("Elevation Angle (Degrees)")
yyaxis left
ylabel("Modulation and Coding Rate")
yyaxis right
ylabel("Maximum Bitrate (kilobits/sec)")
title("Elevation vs Optimal MODCOD by Altitude")
hold off

%% ACM vs Single Coding / Modulation Rate
% Plotting Time at each elevation
% Also plot Datathroughput of each strategy
figure
hold on
sgtitle("Adaptive vs Fixed Coding and Modulation - 409km Fixed Orbit")

% Elevation to MODCOD and bitrate
subplot(2,2,[1,3])
for alt_it = 1:length(altitude_sat)
    CNR_vals = opt_modcod_tab(opt_modcod_tab.Altitude ==
altitude_sat(alt_it),:);

    yyaxis left
    plot(CNR_vals.Elevation,CNR_vals.Optimal_MODCOD, ...
        "DisplayName",sprintf('%.3g',altitude_sat(alt_it),'m'));

    yyaxis right
```

```
    plot(CNR_vals.Elevation,CNR_vals.Optimal_Bitrate_bps/1e3, ...
        "DisplayName",sprintf('%.3g',altitude_sat(alt_it),'m'));
end

xlabel("Elevation Angle (Degrees)")
yyaxis left
ylabel("Modulation and Coding Rate")
yyaxis right
ylabel("Maximum Bitrate (kilobits/sec)")
title("Elevation vs Optimal MODCOD")



subplot(2,2,2)
bar(time_modcod_tab.Elevation',time_modcod_tab.Elevation_Time')
title("Elevation vs Time Spent at Elevation")
xlabel("Elevation (degrees)")
ylabel("Time spent at Elevation (s)")

subplot(2,2,4)
bar(["Adaptive","Fixed"],[bits_sent_ACM,bits_sent_fixed])
title("Adaptive vs Fixed Throughput")
xlabel("Strategy")
ylabel("Throughput (bits)")


hold off

figure
hold on
for alt_it = 2:length(altitude_sat)
    CNR_vals = opt_modcod_tab(opt_modcod_tab.Altitude ==
altitude_sat(alt_it),:);

    yyaxis left
    plot(CNR_vals.Elevation,CNR_vals.Optimal_MODCOD, ...
        "DisplayName",sprintf('%.3g',altitude_sat(alt_it),'m'));

    yyaxis right
    plot(CNR_vals.Elevation,CNR_vals.Optimal_Bitrate_bps/1e3, ...
        "DisplayName",sprintf('%.3g',altitude_sat(alt_it),'m'));
end

xlabel("Elevation Angle (Degrees)")
yyaxis left
ylabel("Modulation and Coding Rate")
yyaxis right
ylabel("Maximum Bitrate (kilobits/sec)")
title("Elevation vs Optimal MODCOD")
hold off

figure
hold on
bar(time_modcod_tab.Elevation',time_modcod_tab.Elevation_Time')
title("Elevation vs Time Spent at Elevation")
xlabel("Elevation (degrees)")
ylabel("Time spent at Elevation (s)")
hold off

```

```
figure
hold on
bar(["Adaptive","Fixed"],[bits_sent_ACM,bits_sent_fixed])
title("Adaptive vs Fixed Throughput")
xlabel("Strategy")
ylabel("Throughput (bits)")
hold off
```

A.2. Transmitter Implementation

A.2.1 DFL_Adressing.mlx

```
%Intro
%As we're using an AXI-Stream to access the DVB-S2 transmitter, data will be input in 32b words. For ease of implementation, 4b will be reserved for control words leaving a 28b data field. The DFL of a packet can be of size Kbch - 80. What is the optimal data size to fill all possible DFL?
% https://uk.mathworks.com/matlabcentral/answers/1870657-is-there-a-function-to-find-the-paired-factors-of-an-integer
function D = principaldivisors(N)
  % Compute the principal divisors of the scalar variable N, so only those divisors < N.
  % Note if N is prime, then this function returns ONLY 1, as the only factor less than N.
  F = factor(N);
  D = 1;
  if numel(F) > 1
    % N is composite, so it has more than one divisor less than N
    % if N were prime, then 1 and N are the only factors.
    for Fi = F
      D = unique([D,D*Fi]);
    end
    D(D == N) = [];
  end
end
function pairs = allfactorpairs2(N)
  D = principaldivisors(N)';
  D(D > sqrt(N)) = [];
  pairs = [D,N./D];
end
normal FECFRAME

K_normal = [16008, 21408, 25728, 32208, 38688, 43040, 48408, 51648, 53840, 57472, 58192]-80;
normal_factors = allfactorpairs2(K_normal(1));
normal_factors = normal_factors(normal_factors(:,1)<29);


% Find all factors in first column < 29
% Take intersect with current set of factors

for i= 2:length(K_normal)
    ifacts = allfactorpairs2(K_normal(i));
    ifacts = ifacts(ifacts(:,1) < 33);
    normal_factors = intersect(normal_factors,ifacts);
```

```
end
normal_factors

Short FECFRAME
K_short = [3072,5232,6312,7032,9552,10632,11712,12432,13152,14232] - 80;
short_factors = allfactorpairs2(K_short(1));
short_factors = short_factors(short_factors(:,1)<29);


% Find all factors in first column < 29i) = mod(K_normal(i),24)
% Take intersect with current set of factors

for i= 2:length(K_short)
    ifacts = allfactorpairs2(K_short(i));
    ifacts = ifacts(ifacts(:,1) < 33);
    short_factors = intersect(short_factors,ifacts);
end
short_factors

fprintf("The highest factor was: %d bits \n",max(short_factors))


Checking all long FECFrames at best short:
mods = zeros(1, length(K_normal));
for i = 1:length(K_normal)
    mods(i) = mod(K_normal(i),24);
end
mods(mods > 0)
```

A.2.2 rolloff_demonstration.mlx

```
%Investigation of impact of roll-off factor
%The final block in the DVB-S2 transmission system is a Root-Raised-Cosine Filter for pulse shaping. This has the effect of increasing the total bandwidth of the transmission based on the roll-off factor. Therefore, the maximum symbol rate can be calculated for the target bandwidth for each value of roll-off in the standard.
modtable = readtable("Rs2Rb.csv");
Bandwidth =150000
alphas = [0.35;0.25;0.20];
alpha = 0.35
Rs = 150e3 / (1 + alpha)
bitsPerSymbol = 5
ratesCells = unique(modtable.CodingRate);
rateVals = double(string(ratesCells).split("/"));
rates = rateVals(:,1) ./ rateVals(:,2)

codingRate    = rates(3)
bitrate = Rs*bitsPerSymbol*double(codingRate)
bitrate_kb = bitrate/1024

fprintf("Bitrate = %2f kbps for: \n" + ...
            "   Alpha = %2f  \n" + ...
            "   Bandwidth = %2f  \n" + ...
            "   BitsPerSymbol = %2f  \n" + ...
            "   Coding Rate = %2f",bitrate_kb, alpha, Bandwidth, bitsPerSymbol, codingRate)
```

```
%Plotting resulting bitrates
modcods = string(modtable.Modulation) + " " +
string(modtable.CodingRate);
modcod_Cat = categorical(modcods);
modcod_Cat = reordercats(modcod_Cat,modcods)
figure
hold on
%alpha 0.35
plot(modcod_Cat,modtable.Datarate_kbps____0_35_);
plot(modcod_Cat,modtable.Datarate_kbps____0_25_);
plot(modcod_Cat,modtable.Datarate_kbps____0_20_,"Color","magenta");

xlabel("Modulation and Coding Rate");
ylabel("Bitrate (kbps)")
legend(["Alpha = 0.35", "Alpha = 0.25", "Alpha = 0.20"], "Position",
[0.6543 0.3621 0.2507, 0.1335])


hold off
```

A.2.3 gendata.m

```
function
[data_Out,TSorGS_Out,DFL_Out,UPL_Out,SYNC_Out,MODCOD_Out,FECFRAME_Out] =
gendata(nFrames,nPackets, ...

TSorGS_In,DFL_In,UPL_In,SYNC_In,MODCOD_In,FECFRAME_In, ...

frm_bits, pkt_bits, pl_cnt_bits, ...

frameEnd_pos, frameStart_pos, pktEnd_pos, pktStart_pos)
%GENDATA Generates test data samples for the dvb-s2 stream block
% Outputs data bits and control signals padded accordingly
% Data packets contains counters for frame, packet, and byte, as well as
% start and end markers.
% If UPL = 0, then generic stream. Fr
% ---------------------------
% Bit Maps
% data_Out:
%   31 downto 12:   0 / padding
%   11 downto  4:   Data Byte
%   3:              frameEndIn
%   2:              frameStartIn
%   1:              pktEndIn
%   0:              pktStartIn
% Data Byte:
%   7                   downto (7-frm_bits): Frame Count
%   (7-frame_bits-1)    downto (byte_bits) : Packet Count
%   (payload_bits - 1)    downto 0:            Data payload Count
%
% ---------------------------
% Outputs all of length data_Out
% TODO: Multiple control settings / function run

    arguments
        % Packet Generation
```

```
30          nFrames         (1,1) {mustBeNonnegative,mustBeInteger}
% Frames to generate
31          nPackets        (1,1) {mustBeNonnegative,mustBeInteger}
% Packets to generate / frame
32
33          % Control Signals
            TSorGS_In       (1,1)
34  {mustBeNonnegative,mustBeInteger,mustBeLessThan(TSorGS_In,    4)} %
fixdt(0,2,0)
            DFL_In          (1,1)
35  {mustBeNonnegative,mustBeInteger,mustBeLessThan(DFL_In,   65536)} %
uint16
            UPL_In          (1,1)
36  {mustBeNonnegative,mustBeInteger,mustBeLessThan(UPL_In,   65536)} %
uint16
            SYNC_In         (1,1)
37  {mustBeNonnegative,mustBeInteger,mustBeLessThan(SYNC_In,    256)} % uint8
            MODCOD_In       (1,1)
38  {mustBeNonnegative,mustBeInteger,mustBeLessThan(MODCOD_In,   32)} %
fixdt(0,5,0)
            FECFRAME_In     (1,1)
39  {mustBeNonnegative,mustBeInteger,mustBeLessThan(FECFRAME_In,  2)} %
logical
40
41          % Marker bit settings
42          frm_bits        (1,1) = 1;
43          pkt_bits        (1,1) = 2;
44          pl_cnt_bits     (1,1) = 5;
45
46          % Control Signal bit positions
47          frameEnd_pos    (1,1) = 3;
48          frameStart_pos  (1,1) = 2;
49          pktEnd_pos      (1,1) = 1;
50          pktStart_pos    (1,1) = 0;
51      end
52      % Validate Inputs
53      assert((UPL_In == 0) | (UPL_In > 8),       "Packet size must be
larger than 8 if packetised stream");
54      assert(UPL_In <= DFL_In,                    "Packet size larger than
Datafield");
55      assert(nPackets*UPL_In==DFL_In,            "Packets / Frame larger
than Datafield");
56      assert(frm_bits+pkt_bits+pl_cnt_bits == 8, "bit fields in data byte
must add to 8 bits");
57      assert(frm_bits < 8 && pkt_bits < 8 && pl_cnt_bits < 8, "bit fields
must not individually be 8 bits")
58
59      % Fixing Types: Will use at end
60      TSorGS_In_t   = fi(TSorGS_In,0,2,0);
61      DFL_In_t      = uint16(DFL_In);
62      UPL_In_t      = uint16(UPL_In);
63      SYNC_In_t     = uint8(SYNC_In);
64      MODCOD_In_t   = fi(MODCOD_In,0,5,0);
65      FECFRAME_In_t = logical(FECFRAME_In);
66
```

```
    % Generate nFrames each with nPackets, each containing UPL[bits] / 8 bytes
    axiDataArray = zeros(1,nFrames.*nPackets.*UPL_In./8,'uint32');

    frame_it    = uint32(0);
    packet_it   = uint32(0);
    payload_it  = uint32(0);

    for axi_it = 0:length(axiDataArray)-1
        axi_pkt = uint32(0);
        payload = uint32(0);
        flags   = uint32(0);
        UPL_it  = UPL_In; % TODO: change for UPL_In matrix
        DFL_it  = DFL_In; % TODO: change for DFL_In matrix

        % Creating payload byte
        % 0b[00...][byte_it] -> 0b[0...][byte_bits]
        payload = bitor(payload,mod(payload_it,2^pl_cnt_bits));
        % 0b[000...][packet_it] -> 0b[00...][pkt_bits][byte_bits]
        payload = bitor(payload,bitshift( ...
                            mod(packet_it,2^pkt_bits), ... % Modulo to remain length pkt_bits
                            pl_cnt_bits) ...
                        );
        % 0b[0...][frame_it] -> 0b[0...][frm_bits][pkt_bits][byte_bits]
        payload = bitor(payload,bitshift(...
                            mod(frame_it,2^frm_bits), ...
                            pl_cnt_bits+pkt_bits) ...
                            );
        % payload is of the form: 0b[0...][frm count][pkt count][byte count]
        assert(payload<256, "Payload byte should be a byte! payload_it: %d , payload val: %d",payload_it,payload)

        % Creating control flag bits
        bits_sent = (axi_it+1)*8;
        packet_start = uint32(mod(bits_sent-8,  UPL_it)  == 0);
        packet_end   = uint32(mod(bits_sent,    UPL_it)  == 0);
        frame_start  = uint32(mod(bits_sent-8,  DFL_it)  == 0);
        frame_end    = uint32(mod(bits_sent,    DFL_it)  == 0);
        assert(packet_start<2 && packet_end < 2 && frame_start < 2 && frame_end<2,"Flags should be 1 bit");

        % Creating control flag field
        flags = bitor(flags, bitshift( ...
                                packet_start, ...
                                pktStart_pos ...
                                ) ...
                        );
        flags = bitor(flags, bitshift( ...
                                packet_end, ...
                                pktEnd_pos ...
                                ) ...
                        );
        flags = bitor(flags, bitshift( ...
                                frame_start, ...
                                frameStart_pos ...
```

```
                                    ) ...
                    );
        flags = bitor(flags, bitshift( ...
                                    frame_end, ...
                                    frameEnd_pos ...
                                    ) ...
                    );

        % Create axi packet
        % 0b[0...][payload byte][flags]
        payload_startpos = 4; % Flags always 4 bits
        axi_pkt = bitor(flags, bitshift( ...
                                        payload, ...
                                        payload_startpos ...
                                        ) ...
                    );
        % Place in array
        axiDataArray(axi_it+1) = axi_pkt;

        % Update counters
        if frame_end
            frame_it = frame_it + 1;
        end
        if packet_end
            packet_it = packet_it + 1;
        end
        payload_it = payload_it + 1;
    end % Axi Stream loop


% Assign outputs
data_Out     = axiDataArray;
% Match length of control signals
TSorGS_Out   = repelem(TSorGS_In_t,   length(axiDataArray));
DFL_Out      = repelem(DFL_In_t,      length(axiDataArray));
UPL_Out      = repelem(UPL_In_t,      length(axiDataArray));
SYNC_Out     = repelem(SYNC_In_t,     length(axiDataArray));
MODCOD_Out   = repelem(MODCOD_In_t,   length(axiDataArray));
FECFRAME_Out = repelem(FECFRAME_In_t, length(axiDataArray));


end % Function
```

A.2.4 parse_axipkt.m

```
function [parsed_pkt] = parse_axipkt(axipkt_in, ...
                                    frm_bits, pkt_bits, pl_cnt_bits, ...
                                    frameEnd_pos, frameStart_pos,
pktEnd_pos, pktStart_pos)
%PARSE_AXIPKT Parses control fields and payload data generated by
gendata.m
%
arguments
    axipkt_in (1,:) uint32 % 32bit axi TData

```

```
    % Marker bit settings
    frm_bits        (1,1) = 1;
    pkt_bits        (1,1) = 2;
    pl_cnt_bits    (1,1) = 5;

    % Control Signal bit positions
    frameEnd_pos    (1,1) = 3;
    frameStart_pos  (1,1) = 2;
    pktEnd_pos      (1,1) = 1;
    pktStart_pos    (1,1) = 0;
end
    pkt_struct = struct( ...
                    "Frame_Count",  uint32(0),...
                    "Packet_Count", uint32(0),...
                    "Byte_Count",   uint32(0),...
                    ...
                    "Frame_End",    logical(0),...
                    "Frame_Start",  logical(0),...
                    "Packet_End",   logical(0),...
                    "Packet_Start", logical(0)...
                        );
    pkt_struct_arr = repelem(pkt_struct,length(axipkt_in));

    % Bit Masks
    frameEnd_mask   = 2^frameEnd_pos;
    frameStart_mask = 2^frameStart_pos;
    pktEnd_mask     = 2^pktEnd_pos;
    pktStart_mask   = 2^pktStart_pos;

    frm_mask        = bitshift(...
                        (2^frm_bits)-1,...
                        pkt_bits + pl_cnt_bits...
                        );
    pkt_mask        = bitshift(...
                        (2^pkt_bits)-1,...
                        pl_cnt_bits...
                        );
    pl_cnt_mask       = (2^5)-1;

    payload_mask    = bitshift( ...
                        (2^8)-1,...
                        4 ...
                        );


    for pkt_it = 1:length(axipkt_in)
        % Setup loop variables
        axipkt = axipkt_in(pkt_it);
        pkt_struct_temp = pkt_struct;

        % Get payload byte
        payload_startpos = 4; % Flags always 4 bits
        payload = bitshift(...
                    bitand(axipkt,  payload_mask),...
                    payload_startpos*-1 ...
                );
        % Get payload byte counts
```

```
        pkt_struct_temp.("Frame_Count") = bitshift(...
                                bitand(payload,frm_mask),...
                                -1*(pkt_bits+pl_cnt_bits)...
                                );

        pkt_struct_temp.("Packet_Count") = bitshift(...
                                bitand(payload,pkt_mask),...
                                -1*(pl_cnt_bits)...
                                );
        pkt_struct_temp.("Byte_Count") = bitand(payload,pl_cnt_mask);

        % Flags
        pkt_struct_temp.("Frame_End") = logical(bitshift(...
                            bitand(axipkt,  frameEnd_mask),...
                            frameEnd_pos*-1 ...
                            ));
        pkt_struct_temp.("Frame_Start") = logical(bitshift(...
                            bitand(axipkt,  frameStart_mask),...
                            frameStart_pos*-1 ...
                            ));
        pkt_struct_temp.("Packet_End") = logical(bitshift(...
                            bitand(axipkt,  pktEnd_mask),...
                            pktEnd_pos*-1 ...
                            ));
        pkt_struct_temp.("Packet_Start") = logical(bitshift(...
                            bitand(axipkt,  pktStart_mask),...
                            pktStart_pos*-1 ...
                            ));

        % Store result
        pkt_struct_arr(pkt_it) = pkt_struct_temp;
    end
    parsed_pkt = pkt_struct_arr;
end
```

A.2.5 spectMask.m

```
function [mask] = spectMask(Bn, Ptx)
%SPECTMASK Creates mask limit boundaries based on ITU-R SM.1541-7 Fig 43
%   Bn   : Necessary bandwidth (Hz)
%   Ptx  : Transmit power (W)
%   Mask : all dBc
arguments
    Bn  (1,1) = 150e3;
    Ptx (1,1) = 1;
end
    bound_50  = -10;
    bound_120 = -1*(31 + 10*log10(Ptx));
    bound_225 = -1*(38 + 10*log10(Ptx));
    posmask = [0,             0;  ...
               Bn/2,          0;  ...
               Bn/2,   bound_50;  ...
               Bn*1.2, bound_50;  ...
               Bn*1.2, bound_120; ...
               Bn*2.25, bound_120; ...
               Bn*2.25, bound_225; ...
```

```
                ];
    negmask = flip([posmask(:,1).*-1, posmask(:,2)],1);
    mask = [negmask;posmask];
end
```

A.3. Packet Handling

A.3.1 Packet.hpp

```
/* Definition of Packet Class
*/

#include <cereal/archives/portable_binary.hpp>

#include <stdint.h>
#include <assert.h>
#include <climits>
#include <optional>
#include <vector>

static_assert(sizeof(double) * CHAR_BIT == 64, "64-bit double is
assumed.");
typedef uint64_t timestamp;

class GSE_Packet {
public:
    struct FixedHeader {
        bool start    : 1;
        bool end      : 1;
        int labelType : 2;
        int GSElength : 12;
        };
    struct VariableHeader {
        std::optional<char>     fragID;
        std::optional<uint16_t> totalLength;
        std::optional<uint16_t> protocolType;
        std::optional<char[3]>  label;
    };
private:
    FixedHeader fh;
    std::optional<VariableHeader> vh;
    Packet* pdu;

public:
    // constructor
    GSE_Packet(FixedHeader fh, VariableHeader vh, Packet* pdu);
    std::vector<uint8_t> serialise();
};


class Packet {
    uint8_t type;
    uint8_t version;
    timestamp time;
    virtual cereal::PortableBinaryOutputArchive serialise();
};

```

```
48 std::vector<GSE_Packet> fragment(Packet* pdu);
49
50
51
52 struct PPL_pkt_64k : Packet{
53     double standard_deviation;  // Standard deviation of data
54     double codebook[16];        // Quantization levels
55     uint32_t data[2048];        // compressed PPL data
56
57     cereal::PortableBinaryOutputArchive serialise();
58 };
```

Appendix B: Work Packages

B.1. TTC

Code	Domain	Title	Responsible	Duration	Deliverables	Prerequisites	Description	Required Resources/Facilities	Predecessor WPs	Successor WPs	Completeness
SD.TTC.DES.01.00	TTC	Creation and Testing of Downlink Communications System	RF Specialist	1 Year 4 months	Downlink communications system design	STRATHcube mission documentation	Evaluation of requirements, implementation on engineering model, integration with PPL system and performance testing	Mission documentation, Target SDR documentation, MATLAB, Simulink, Vivado, Python, C++, Static analysis tools, FPGA Development board, transceiver development board	N/A	N/A	50%
SD.TTC.DES.01.01	TTC	Assess STRATHcube Communication System Work to Date	RF Specialist	2 weeks	Literature review of STRATHcube communication system	STRATHcube mission documentation	Evaluate current state of communication system to identify progress, gaps and requirements	Mission documentation	N/A	SD.TTC.DES.01.04	100%
SD.TTC.DES.01.02	TTC	Review DVB-S2 Standard and Relevant Communication Standards	RF Specialist	2 weeks	Literature review of relevant standards	None	Analyse DVB-S2 system, determine suitability to mission, identify suitable architecture	Standards Documents	N/A	SD.TTC.DES.01.06	100%
SD.TTC.DES.01.03	TTC	Research DVB-S2 SDR Implementations	RF Specialist	1 month	Literature review of previous implementations	None	Research previous implementations of DVB-S2 transmitters on SDR	Research Papers, Open Source Repositories	N/A	SD.TTC.DES.01.06	100%
SD.TTC.DES.01.04	TTC	Functional Requirement Generation for Downlink Communication System	RF Specialist, Systems Engineer	1 week	Functional Requirements	SD.TTC.DES.01.01, SD.TTC.DES.01.02, SD.TTC.DES.01.03	Collaborate with Systems Engineers to develop functional requirements for system development.	N/A	N/A	SD.TTC.DES.01.06	100%
SD.TTC.DES.01.05	TTC	Review of Target SDR Hardware	RF Specialist	1 week	FPGA resource requirements, identification of suitable development boards for engineering model	None	Investigate available documentation regarding the target SDR platform to evaluate FPGA resources and design requirements.	Target SDR Documentation	N/A	SD.TTC.DES.01.06	100%
SD.TTC.DES.01.06	TTC	Downlink Communications Architecture Development	RF Specialist	3 months	High level end to end system design document for downlink transmitter	SD.TTC.DES.01.02, SD.TTC.DES.01.03, SD.TTC.DES.01.04	Identification of all blocks required to implement downlink communication system. Identification of	N/A	SD.TTC.DES.01.02, SD.TTC.DES.01.03, SD.TTC.DES.01.04	SD.TTC.DES.01.07, SD.TTC.DES.01.09	100%

Code	Domain	Title	Responsible	Duration	Deliverables	Prerequisites	Description	Required Resources/Facilities	Predecessor WPs	Successor WPs	Completeness
							suitable resources to aid implementation.				
SD.TTC.DES.01.07	TTC	Implement DVB-S2 Transmitter on Engineering Model (Programmable Logic)	RF Specialist	3 months	DVB-S2 Transmitter FPGA Implementation	SD.TTC.DES.01.06	Creation of FPGA design to implement DVB-S2 Transmission System	MATLAB, Simulink, Vivado	SD.TTC.DES.01.06	SD.TTC.DES.01.8	50%
SD.TTC.DES.01.08	TTC	Validation of DVB-S2 Transmitter on Engineering Model (Programmable Logic)	RF Specialist	1 month	Validated DVB-S2 Transmitter Design	SD.TTC.DES.01.07	Validation of implemented DVB-S2 transmitter to ensure adherence to standard and suitability.	MATLAB, Simulink, Vivado	SD.TTC.DES.01.07	SD.TTC.DES.01.11	0%
SD.TTC.DES.01.09	TTC	Implementation of Packet Handling on Engineering Model (Processing System)	RF Specialist	1 month	Packet Handling System Software Implementation	SD.TTC.DES.01.06	Implementation of the packet handling system in software	Python, C++	SD.TTC.DES.01.06, SD.TTC.DES.01.08	SD.TTC.DES.01.10	0%
SD.TTC.DES.01.10	TTC	Validation of Packet Handling on Engineering Model (Processing System)	RF Specialist	1 month	Validated Packet Handling Software	SD.TTC.DES.01.09	Validation of packet handling system in software.	Python, C++, Static Analysis Tools	SD.TTC.DES.01.09	SD.TTC.DES.01.11	0%
SD.TTC.DES.01.11	TTC	Downlink Engineering Model End to End Testing	RF Specialist	1 month	Downlink Engineering Model Test Report	SD.TTC.DES.01.06, SD.TTC.DES.01.08	Validation of entire system in hardware.	Appropriate FPGA development board, Appropriate transceiver development board	SD.TTC.DES.01.06, SD.TTC.DES.01.08	SD.TTC.DES.01.12	0%
SD.TTC.DES.01.12	TTC	Combination of PPL and Primary Downlink on Engineering Model	RF Specialist, PBR Specialist	1 month	Engineering model of combined PPL and downlink communications system	SD.TTC.DES.01.11, SD.PPL.PRF.03.06	Combine PPL and Downlink FPGA design	Appropriate FPGA development board, Appropriate transceiver development board	SD.TTC.DES.01.11, SD.PPL.PRF.03.06	SD.TTC.DES.01.13	0%
SD.TTC.DES.01.13	TTC	Analyse FPGA Resource Usage of PPL System and Downlink System on Engineering Model	RF Specialist, PBR Specialist	2 weeks	Resource Usage Report of combined design	SD.TTC.DES.01.12	Ensure that both designs can fit on target FPGA. Optimising as necessary.	Appropriate FPGA development board, Appropriate transceiver development board	SD.TTC.DES.01.12	SD.TTC.DES.01.14	0%
SD.TTC.DES.01.14	TTC	End to End Testing of Combined PPL and Downlink System on Engineering Model	RF Specialist, PBR Specialist	1 month	Test report of combined design	SD.TTC.DES.01.13	Ensure both systems work as expected when combined	Appropriate FPGA development board, Appropriate transceiver development board	SD.TTC.DES.01.13	SD.TTC.DES.01.15	0%
SD.TTC.DES.01.15	TTC	End to End Testing of Combined PPL and Downlink System on Target SDR	RF Specialist, PBR Specialist	3 months	Downlink System Test Report	SD.TTC.DES.01.14	Ensure design works as expected on target SDR.	Target SDR	SD.TTC.DES.01.14	SD.TTC.DES.01.00	0%
SD.TTC.ANL.01.00	GS	Receiver Analysis and Modelling	RF Specialist	3 months	Receiver Analysis Results and requirements	Ground station site selection	Modelling of receive channel for satellite	MATLAB, STK, Simulink	GD.GS.TRD.09.00	SD.TTC.DES.02.00	0%

Code	Domain	Title	Responsible	Duration	Deliverables	Prerequisites	Description	Required Resources/Facilities	Predecessor WPs	Successor WPs	Completeness
							and subsequent receiver requirement				
SD.TTC.ANL.01.01	GS	Channel Research	RF Specialist	1 month	Interference	Ground station site selection	Identification of relevant literature and model parameters including interference	MATLAB, STK, Simulink	GD.GS.TRD.09.00	SD.TTC.ANL.01.02	0%
SD.TTC.ANL.01.02	GS	Channel Model Creation	RF Specialist	1 month	Channel Analysis Results, Link Budget	Ground station site selection	Modelling expected channel with interference for satellite receiver over course of mission. Creation of link budget	MATLAB, STK, Simulink	SD.TTC.ANL.01.01	SD.TTC.ANL.01.03	0%
SD.TTC.ANL.01.03	GS	Receiver Requirement Generation	RF Specialist	1 month	Receiver functional and performance requirements	SD.TTC.ANL.01.02			SD.TTC.ANL.01.02	SD.TTC.ANL.01.00	
SD.TTC.DES.02.00	GS	Primary Uplink Receiver Design & Implementation	RF Specialist	6 months	Primary uplink receiver system	Ground station site interference analysis, ground station hardware selection	Design and implement receiver system for primary uplink communications with appropriate filtering to improve SINR.	MATLAB, Simulink, GNU Radio, Vivado	SD.TTC.TRD.02.00	None	0%
SD.TTC.DES.02.01	GS	Receiver Implementation Research	RF Specialist	1 month	Primary uplimk receiver system architecture, identification of open source resources for development		Research into requirements for receiver, previous implementations	Research papers, STRATHcube documents	N/A	SD.TTC.DES.02.02	0%
SD.TTC.DES.02.02	GS	Receiver Interference Filter Design	RF Specialist	3 months	Interference Filter Design	SD.TTC.DES.02.01, SD.TTC.ANL.01.00	Design of anti interference filter based on literature and previous analysis	Interference Report, Research Paper	SD.TTC.DES.02.01, SD.TTC.ANL.01.00	SD.TTC.DES.02.03	0%
SD.TTC.DES.02.03	GS	Receiver Implementation	RF Specialist	1 month	Primary uplink receiver design	SD.TTC.DES.02.02	Implementation of primary uplink receiver	GNU Radio, MATLAB, Simulink, C++	SD.TTC.DES.02.02	SD.TTC.DES.02.04	0%
SD.TTC.DES.02.04	GS	Receiver Testing on Engineering Model	RF Specialist	1 month	Tested primary uplink receiver design	SD.TTC.DES.02.03, SD.TTC.DES.01.11	Testing of design with synthesised data	Tx capable SDR	SD.TTC.DES.02.03, SD.TTC.DES.01.11	SD.TTC.DES.02.00	0%
SD.TTC.DES.02.04	GS	Receiver Testing on Target SDR	RF Specialist	1 month	Tested primary uplink receiver design	SD.TTC.DES.02.03, SD.TTC.DES.01.11	Testing of design with synthesised data	Tx capable SDR	SD.TTC.DES.02.03, SD.TTC.DES.01.11	SD.TTC.DES.02.00	0%
SD.TTC.TRD.02.00	GS	Uplink System Configuration	RF Specialist	2 months	Decision for Uplink System Design	SD.TTC.ANL.01.00	Perform tradeoff of options for uplink system	MATLAB, Simulink	SD.TTC.ANL.01.00	SD.TTC.DES.02.00	0%

Code	Domain	Title	Responsible	Duration	Deliverables	Prerequisites	Description	Required Resources/Facilities	Predecessor WPs	Successor WPs	Completeness
SD.TTC.TRD.02.01	GS	Tradeoff of Modulation, Coding Methods and Rates	RF Specialist	2 weeks	Tradeoff Analysis of uplink modulation, coding method and rates.	SD.TTC.ANL.01.00	Identification of suitable methods, tradeoff report outlining expected performance of each	Channel analysis results	SD.TTC.ANL.01.00	SD.TTC.TRD.02.02	0%
SD.TTC.TRD.02.02	GS	Recommendation of Uplink System Configuration	RF Specialist	2 weeks	Final report including methodology, results, and recommendations	SD.TTC.TRD.02.01	Compile all findings and select suitable option.	Tradeoff Analysis Result	SD.TTC.TRD.02.01	SD.TTC.TRD.02.00	0%

B.2. Ground Station

Code	Domain	Title	Responsible	Duration	Deliverables	Prerequisites	Description	Required Resources/ Facilities	Predecessor WPs	Successor WPs	Completeness
GD.GS.ANL.01.01	GS	Study Design	RF Specialist	1 week	Plan for interference analysis	Ground station site selection	Identify required equipment, measurements to be made, and required study length	Ground station site selection	GD.GS.TRD.09.00	GD.GS.ANL.01.01	0%
GD.GS.ANL.01.02	GS	Interference Measurements	RF Specialist	3 weeks	Interference Analysis Results	GD.GS.ANL.01.01	Measurement and analysis of interference	Calibrated spectrum analysis equipment	GD.GS.ANL.01.01	GD.GS.ANL.01.00	0%
GD.GS.DES.01.00	GS	Primary Downlink Receiver Design & Implementation	RF Specialist	6 months	Primary downlink receiver system	Ground station site interference analysis, ground station hardware selection	Design and implement receiver system for primary downlink communications with appropriate filtering to improve SINR.	MATLAB, Simulink, GNU Radio, Vivado	GD.GS.ANL.01.00	None	0%
GD.GS.DES.01.01	GS	Receiver Implementation Research	RF Specialist	1 month	Primary downlink receiver system architecture, identification of open source resources for development		Research into requirements for receiver, previous implementations	Research papers, STRATHcube documents	N/A	GD.GS.DES.01.02	0%
GD.GS.DES.01.02	GS	Receiver Interference Filter Design	RF Specialist	3 months	Interference Filter Design	GD.GS.DES.01.01, GD.GS.ANL.01.00	Design of anti interference filter based on literature and previous analysis	Interference Report, Research Paper	GD.GS.DES.01.01, GD.GS.ANL.01.00	GD.GS.DES.01.03	0%
GD.GS.DES.01.03	GS	Receiver Implementation	RF Specialist	1 month	Primary downlink receiver design	GD.GS.DES.01.02	Creation of primary downlink receiver	GNU Radio, MATLAB, Simulink, C++	GD.GS.DES.01.02	GD.GS.DES.01.04	0%
GD.GS.DES.01.04	GS	Receiver Testing	RF Specialist	1 month	Tested primary downlink receiver design	GD.GS.DES.01.03, SD.TTC.DES.01.11	Testing of design with synthesised data and downlink engineering model	Downlink engineering model, Tx capable SDR	GD.GS.DES.01.03, SD.TTC.DES.01.11	GD.GS.DES.01.00	0%
GD.GS.DES.02.00	GS	Primary Uplink Transmitter Design & Implementation	RF Specialist	6 months	Primary uplink transmitter system	SD.TTC.TRD.02.00	Design and implement transmitter system for primary uplink communications with appropriate filtering to improve SINR.	MATLAB, Simulink, GNU Radio, Vivado	SD.TTC.TRD.02.00	None	0%
GD.GS.DES.02.01	GS	Transmitter Implementation Research	RF Specialist	1 month	Primary uplink transmitter system architecture, identification of open source resources for development	SD.TTC.TRD.02.00	Research into requirements for transmitter, previous implementations	Research papers, STRATHcube documents	SD.TTC.TRD.02.00	GD.GS.DES.02.02	0%
GD.GS.DES.02.02	GS	Transmitter Implementation	RF Specialist	1 month	Primary uplink transmitter design	GD.GS.DES.02.01	Creation of primary uplink transmitter	GNU Radio, MATLAB, Simulink, C++	GD.GS.DES.02.01	GD.GS.DES.02.03	0%

Code	Domain	Title	Responsible	Duration	Deliverables	Prerequisites	Description	Required Resources/ Facilities	Predecessor WPs	Successor WPs	Completeness
GD.GS.DES.02.03	GS	Transmitter Testing	RF Specialist	1 month	Tested primary uplink transmitter design	GD.GS.DES.02.02	Testing of design with synthesised data and uplink engineering model	Tx capable SDR, Rx capable SDR	GD.GS.DES.02.02	GD.GS.DES.02.00	0%

Appendix C: Extended Abstract Submitted to European Data Handling & Data Processing Conference

Analysis and Implementation of DVB-S2 in the UHF Band for STRATHcube Downlink Communications

***Abstract*—This paper outlines the downlink system design for STRATHcube. Performance and link analyses were conducted, analysing communication windows over the course of the mission and expected theoretical performance. The design was implemented in hardware using MathWorks HDL Coder and C++ code developed for packet handling in software. Software is being created for a basic ACM router and fragmentation using Generic Stream Encapsulation (GSE).**

Initial performance analysis was conducted in comparison to a typical system which uses Constant Coding and Modulation (CCM) and optimises for maximum availability and shows a significant uplift in data throughput over the course of the mission. Additionally, resource analysis of the target FPGA SoC and the implemented design, as well as timing analysis show that the system will be implementable in hardware.

***Index Terms*—CubeSat, Communications, DVB-S2, Software Defined Radio**

I. Introduction

A. CubeSat Communications

Efficient downlink of recorded telemetry is a critical challenge in CubeSat missions, constrained by power limitations, bandwidth restrictions, and dynamic channel conditions imposed during a ground station pass. Further, many CubeSats use the Ultra High Frequency (UHF) amateur band which introduces the further issue of in-band interference which cannot be accounted for in advance.

To maximise throughput under these conditions, Adaptive Coding and Modulation (ACM) must be used. Digital Video Broadcasting – Satellite Second Generation (DVB-S2) is one such ACM system, with near Shannon limit performance and a modular standard allowing it to be matched to the use case. Despite this, there are no space rated UHF DVB-S2 transmitters currently on the market which necessitates its implementation on a Software Defined Radio (SDR). While integrating an SDR can be costly and complex for many missions, the design of STRATHcube already includes one, minimising additional expenses.

In [1] a design for an SDR based communication system in the 915 MHz UHF band leveraging ACM is presented. The system used similar modulation to DVB-S2, with a different coding method. Their analysis showed an almost doubling in throughput compared to CCM. This proved that ACM systems for CubeSats are feasible and offer large performance benefits.

Thanks are given to the sponsors of the STRATHcube, without whom this project would not be possible: The University of Strathclyde Alumni Fund, the Institute of Mechanical Engineers, the Royal Aeronautical Society, the University of Strathclyde Mechanical and Aerospace Engineering Department, and the University of Strathclyde Aerospace Centre of Excellence.

B. STRATHcube Mission

Given the constraints and trade-offs involved in telemetry downlink design, particularly in small satellite missions, emerging CubeSat projects increasingly explore novel approaches to balance performance, complexity, and cost.

STRATHcube is a 2U CubeSat developed at the University of Strathclyde which is part of the *ESA Fly Your Satellite! Design Booster* programme. It has two payloads, each targeting a key area of space sustainability.

The primary payload is a technology demonstrator of a Passive Bistatic Radar (PBR) using an SDR and communicating in the UHF band. Consequently, there will already be a powerful System on Chip (SoC) Field Programmable Gate Array (FPGA) based SDR included on the mission allowing a DVB-S2 transmitter to be implemented as a "piggy-back" on the primary payload using spare resources.

The secondary payload aims to measure the aerothermal effects leading to solar panel fragmentation by recording and transmitting sensor data during re-entry. Due to this, direct communications with a ground station is infeasible and are instead conducted using the Iridium communications network.

C. Mission Phases

The mission is structured into distinct phases, each imposing unique constraints and opportunities on the communication architecture.

- **Early Operations & Commissioning (EOC) (10 days):** Deployment from International Space Station (ISS) at 415 km, system activation
- **Primary Operations Phase (180 days):** Alternates PBR measurements with UHF downlinks until 170 km altitude
- **Transition Phase:** Reconfiguration for re-entry
- **Secondary Phase:** Re-entry data collection and downlink via Iridium

UHF communications take place during the EOC phase, Primary Operations Phase and Transition phase, driving the need for a robust communication system.

II. Objectives

This work focused on STRATHcube's downlink system design, with objectives to:

- Develop detailed downlink system design

- Build and validate an engineering model using development boards

The scope was limited to downlink design (excluding uplink), with hardware implementation as an optional goal. Due to this, the implementation was designed for modularity in order to preserve flexibility for future development.

III. Adaptive Coding and Modulation Analysis

Proof-of-concept analysis was conducted to assess whether the performance gains of Adaptive Coding and Modulation (ACM) justified its implementation complexity. The initial analysis assumed a fixed 409 km orbit altitude, propagated over the 180-day mission duration using ISS TLE data through MATLAB's Satellite Communications Toolbox. [2] As the altitude of the satellite lowers over time, the slant range lowers and throughput increases, therefore the analysis provides a rough lower bound on data throughput for the mission.

The methodology comprised three key steps:

1) **Data Rate Precomputation**: Maximum achievable data rates were calculated for discrete 5 °elevation increments (0 °– 90 °) using link budget calculations and required carrier-to-noise ratios from [3], as shown in Fig. 1.
2) **Two-Phase Simulation**:
   - Coarse orbital analysis (100 s timestep) identified ground station access windows via Toolbox `accessIntervals()` function
   - Fine-grained propagation (1 s timestep) generated precise elevation-angle time histories. A histogram of which is depicted in Fig. 2.
3) **Performance Analysis**: Time-at-elevation statistics (binned at 5 °resolution) were combined with precomputed rate tables to estimate total ACM throughput. The CCM baseline scenario assumed conservative operation at 10 °elevation capacity throughout all passes.

The ACM system achieved over 4 Gb of data transfer over the course of the mission, whereas the CCM strategy achieved only 2.4 Gb. These results suggest a 69% increase in throughput over the course of the mission, indicating that the performance uplift is worth the increased complexity requirements. Additionally, if the ground station is located at a lower latitude, higher order modulations could be used and the performance further increased relative to CCM.

It is worth noting that interference was not considered in this analysis and is a major factor in planned UHF band, further research will be required to identify the impact of this.

IV. System Design and Implementation

Following the ACM analysis, it was decided to implement a DVB-S2 compliant system with ACM capabilities. As flight hardware could not be sourced for this investigation, the target platform was a combination of a Digilent Ze dBoard Zynq 7020 development board attached to a FMCOMMS AD936x series development board. As the interface of the AD936x series transceivers are the same, the system could be verified with a different transceiver in the same family.

TABLE I
STRATHCUBE COMMISSIONING PHASE LINK BUDGET

Name	Adverse	Nominal	Favourable
Altitude		409 km	
Elevation	10 °	20 °	40 °
Slant Range	1463 km	1001 km	611 km
FSPL	148.5 dB	145.2 dB	140.9 dB
CNR	10.9 dB	14.1 dB	19.1 dB
Highest Achievable MODCOD w. 10 dB Margin	QPSK 2/5	8PSK 3/5	16PSK 3/4
MODCOD CNR Required	-0.3 dB	4.0 dB	9.0 dB
Bitrate	115.6 kbps	217.8 kbps	386.2 kbps

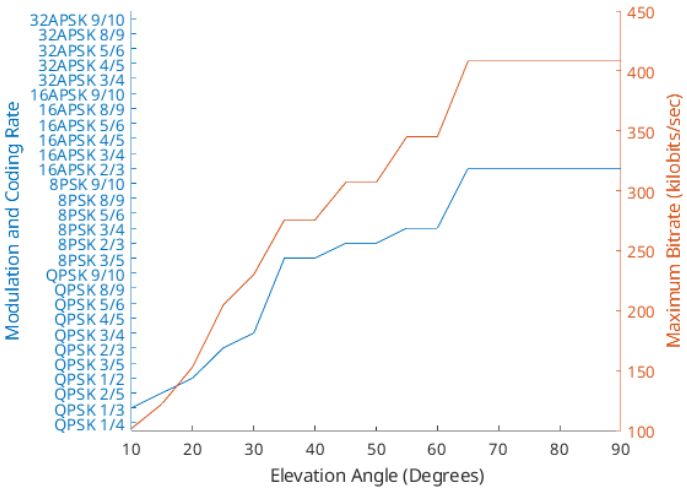


Fig. 1. Highest throughput by elevation angle. The highest achievable modulation and coding rate is shown in blue, with the corresponding throughput in orange.

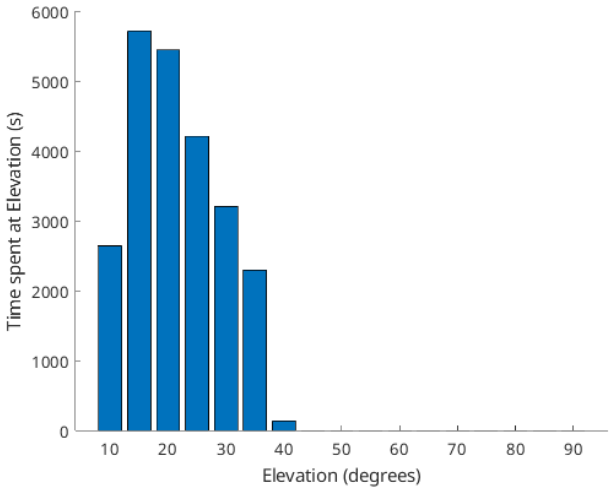


Fig. 2. Histogram of time spent at each simulated pass elevation, binned into 5 °increments. The majority of time is spent in the range of 15 to 25 °.

The designed system had a modular structure, allowing unit testing of each component and facilitating changes in the future as the satellite is developed further.

1) **Processing System**:
   - **System Interface**: Aggregate packets from all external input sources.
   - **Packet Parser**: Parse the input packets to determine their priority level.
   - **ACM Router**: Assign packets into buffers according to Quality of Service (QoS) requirements and encapsulation using Generic Stream Encapsulation (GSE) for efficient framing.
   - **PS-PL Interface**: Break up packets for transfer using an Advanced eXtensible Interface (AXI) and manage control signals for the DVB-S2 Transmitter system.
2) **Programmable Logic**:
   - **DVB-S2 Interface**: Manage interface from DVB-S2 transmitter block to PS AXI-Stream interface.
   - **DVB-S2 Transmitter**: Manages framing, coding, modulation and filtering at baseband.
   - **AD936x Controller**: Manages interface with transceiver.
3) **External Hardware**:
   - **AD936x Transceiver**: Upconversion and transmission of signal.

The FPGA implementation was accomplished using MathWorks HDL Coder [4] due to ease of implementation and ease of testing. Further, using the *Hardware Software Codesign* [5] methodology, both the transceiver interface and PS - PL interface bindings could be automatically generated.

A reference DVB-S2 HDL Coder implementation by MathWorks [6] was selected to reduce technical risk, as it was pretested, allowing faster development of the rest of the system. Further logic was added around this block to manage the AXI-Stream interface and to convert the output for the 12 bit DAC interface. The transmitter portion is shown in Fig. 3.

The implementation of packet handling is yet to be completed, although relevant libraries have been identified and work begun on a C++ implementation of GSE. Further, the packet handling system shall be designed to work with XTCE schemas for packet definition to reduce the difficulty of modification as the satellite design is updated.

V. Simulation & Implementation Results

MATLAB code was created to generate synthetic AXI packets to test the system in Simulink. The resulting output spectrum was then compared against the ITU out of band emissions mask for the amateur and amateur satellite service [7], as shown in Fig. 4.

The system was then exported and an IP block design generated using Vivado. The resource usage was then analysed by subsystem and resource type as shown in Fig. 5. Following this, a timing report was generated, finding a worst negative slack of 0.126 ns, a worst hold slack of 0.016 ns and a worst pulse width slack of 0.264 ns.

VI. Discussion

The simulated output spectrum successfully passes the ITU spectral mask, meaning that it complies with emissions requirements. This requires further testing in hardware to verify that the implemented design aligns with simulation results.

For most resource types, there is enough below 50% usage, leaving plenty for the PBR implementation, however the transmitter design uses almost all of the Block RAM available on the device, which would severely restrict the PBR subsystem. The Block RAM usage of the final system will require optimisation in the future, as well as analysis of the PBR implementation to identify the required resources.

The timing report showed that the system is implementable in hardware, however the margins are quite slim. This may be related to how spread out the implemented design was on the silicon, as so many Block RAMs were required. An optimisation of resource requirements could therefore improve the timing slack available.

VII. Conclusion

The benefits of ACM for the mission has been analysed using orbital simulation and link budget analysis, indicating that a substantial uplift in throughput could be possible. With this in mind, the system architecture was designed, with all key blocks identified for packet handling and transmission using DVB-S2. A transmitter was created building upon a MathWorks reference implementation using MathWorks HDL Coder. This hardware implementation was then synthesised using Vivado and found to meet both resource and timing requirements.

The packet handling systems are still to be implemented, although work has begun on GSE.

References

[1] E. Grayver, A. Chin, J. Hsu, S. Stanev, D. Kun and A. Parower, "Software defined radio for small satellites," 2015 IEEE Aerospace Conference, Big Sky, MT, USA, 2015, pp. 1-9, doi: 10.1109/AERO.2015.7118901.

[2] MathWorks, Satellite Communications Toolbox (R2024b). (2024). The MathWorks Inc., Natick, Massachusetts, United States. [Online]. Available: https://www.mathworks.com

[3] ETSI, EN 302 307-1 - V1.4.1 - Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broa dBand satellite applications; Part 1: DVB-S2, EN 302 307-1, Jul. 2014.

[4] MathWorks, HDL Coder (R2024b). (2024). The MathWorks Inc., Natick, Massachusetts, United States. [Online]. Available: https://www.mathworks.com

[5] MathWorks, 'Hardware-Software Co-Design Workflow for SoC Platforms', The MathWorks Inc., Accessed: Mar. 31, 2025. [Online]. Available: https://uk.mathworks.com/help/hdlcoder/ug/hardware-software-codesign-workflow-for-soc-platforms.html

[6] MathWorks, DVB-S2 HDL Transmitter. (2024). MATLAB. The MathWorks Inc. Accessed: Dec. 11, 2024. [Online]. Available: https://uk.mathworks.com/help/satcom/ug/dvbs2-hdl-transmitter.html

[7] ITU, Unwanted emissions in the out-of-band domain, SM.1541-7, Sep. 2024.

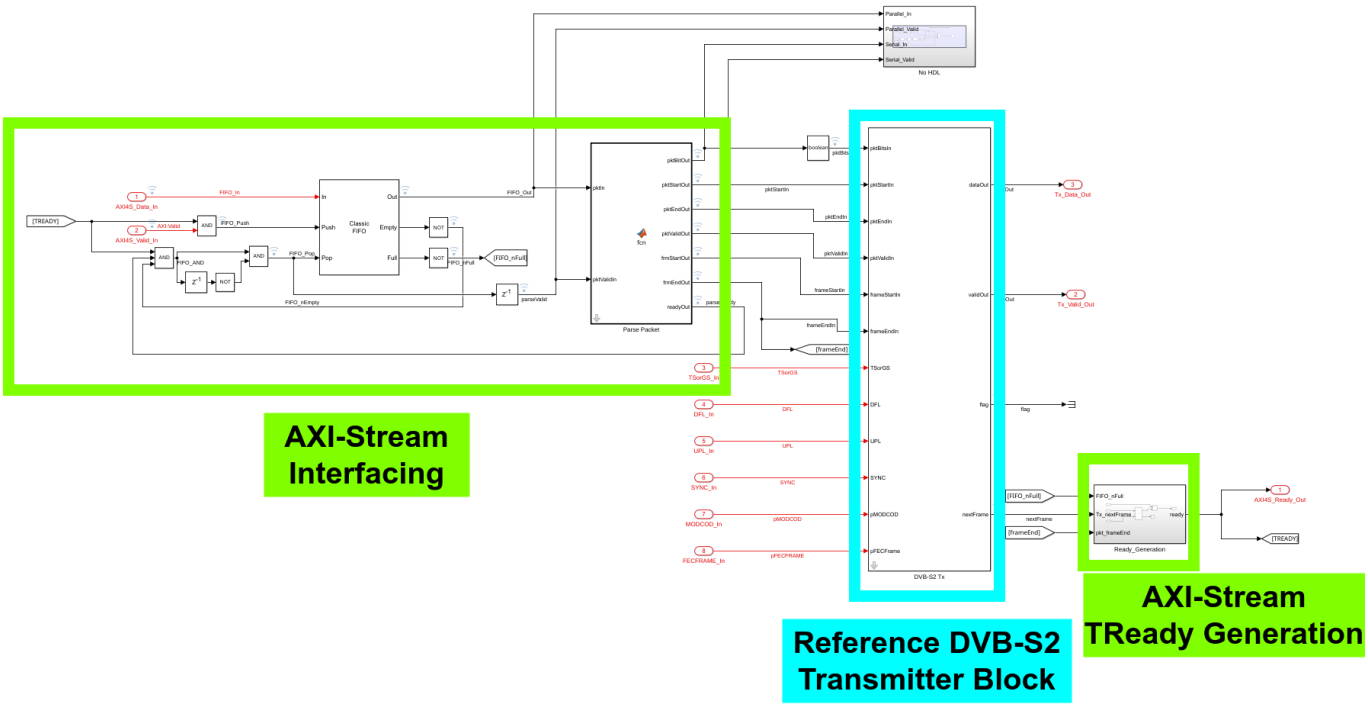


Fig. 3. MathWorks HDL Coder transmitter blocks. Highlighted in green are the blocks used for the AXI-Stream interface. In Cyan is the reference DVB-S2 transmitter block created by MathWorks

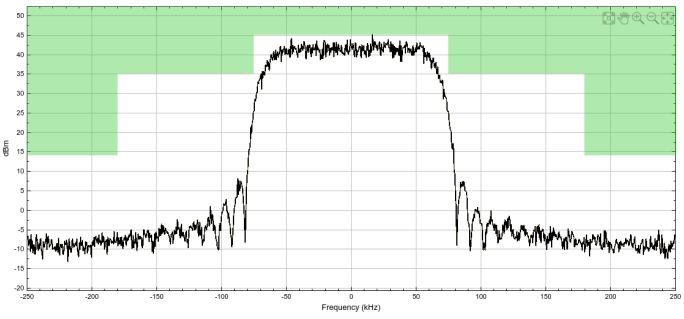


Fig. 4. Simulated output spectrum with ITU spectral mask. The mask is shown as the coloured section at the top, with green indicating that the spectrum is passing. Note that the power axis is not representative of the actual output power, as this has been scaled for interfacing with the DAC.

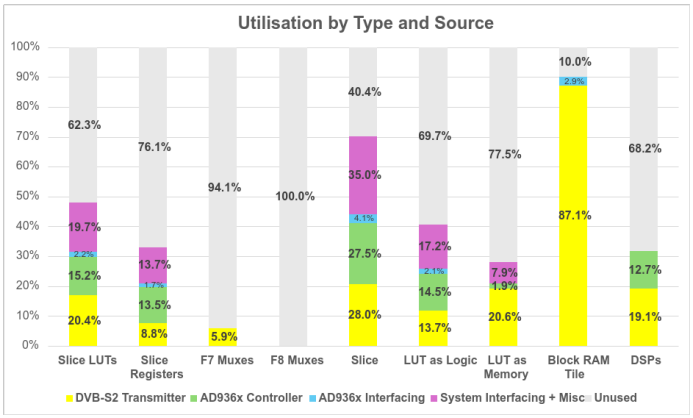


Fig. 5. Implemented design utilisation by resource type and subsystem. Most resource types are below 50% usage, with the exception of Block RAM, of which only 10% is free.