ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

DEVELOPMENT/ TESTING OF SOFTWARE FOR A CUBESAT FOR HIGH RESOLUTION EARTH OBSERVATION IN A LOW EARTH ORBIT

M.Sc. THESIS

Mehreen Azam

Department of Aeronautical and Astronautical Engineering

Aeronautical and Astronautical Engineering Programme

JUNE 2024

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

DEVELOPMENT/TESTING OF SOFTWARE FOR A CUBESAT FOR HIGH RESOLUTION EARTH OBSERVATION IN A LOW EARTH ORBIT

M.Sc. THESIS

Mehreen Azam
(511221120)

Department of Aeronautical and Astronautical Engineering

Aeronautical and Astronautical Engineering Programme

Thesis Advisor: Prof. Dr. Alim Rüstem ASLAN

HAZIRAN 2024

ISTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ALÇAK DÜNYA YÖRÜNGESİNDE YÜKSEK ÇÖZÜNÜRLÜKLÜ DÜNYA GÖZLEMİNE YÖNELİK BİR CUBESAT YAZILIMININ GELİŞTİRİLMESİ/TEST EDİLMESİ

YÜKSEK LİSANS TEZİ

MEHREEN AZAM
(511221120)

Uçak ve Uzay Mühendisliği Ana Bilim Dalı

Uçak ve Uzay Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Alim Rüstem ASLAN

HAZIRAN 2024

Mehreen Azam, a M.Sc. student of İTU Graduate School student ID 511221120, successfully defended the thesis/dissertation entitled "DEVELOPMENT/TESTING OF SOFTWARE FOR A CUBESAT FOR HIGH RESOLUTION EARTH OBSERVATION IN A LOW EARTH ORBIT", which she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Alim Rüstem Aslan** Istanbul Technical University	.....
Jury Members :	**Prof. Dr. Tuncay Younus Erkec** Turkish National Defence University	.....
	Assist. Prof. Dr. Cuma Yarım Istanbul Technical University	.....

Date of Submission : 24 May 2024
Date of Defense : 24 June 2024

To my family and friends,

FOREWORD

Thanks to my advisor Prof. Dr. Alim Rüstem Aslan for his support. Special thanks to my husband Naveed Naseer for his unwavering assistance.

Thank you to all my Space Systems Design and Testing Laboratory colleagues for their invaluable contributions to this project. Special thanks to Eng.Onur Oztekin, Eng. Kaan Sarica, Eng. Kaan Gokturk and Designer Rumeysa Ustuncan.

24 June 2024

Mehreen Azam
Astronautical Engineer

TABLE OF CONTENTS

Page

ABBREVIATIONS

ADCS **:** Attitude Determination and Control System
CRC **:** Cyclic Redundancy Check
CoTS **:** Commercially off the Shelf
dTDI **:** Digital Time Delay Integration
EPS **:** Electrical Power System
EQM **:** Engineering Qualification Model
FSW **:** Flight Software
GPIO : General Purpose Input/ Output
GNSS **:** Global Navigation Satellite System
ICD : Interface Control Document
IDE **:** Integrated Development Environment
JTAG **:** Joint Test Action Group
Ksps **:** Kilo samples per second
LEO **:** Low Earth Orbit
LSB **:** Least Significant Bit
NVM : Non-Volatile Memory
OBC **:** On Board Computer
PCB **:** Printed Circuit Board
RAM **:** Random Access Memory
ROM **:** Read Only Memory
RF **:** Radio Frequency
RTOS **:** Real Time Operating System
SCL **:** Serial Clock Line
SDA **:** Serial Data
SPI **:** Serial Peripheral Interface
SSDTL **:** Space Systems Design and Test Laboratory
TC : Timer Counter
TDI **:** Time Delay Integration
TT&C **:** Telemetry Tracking and Command
UART **:** Universal Asynchronous Receiver Transmitter

UTC **:** Unix Time Counter

UHF **:** Ultra High Frequency

WDT **:** Watch Dog Timer

LIST OF TABLES

Page

LIST OF FIGURES

Page

DEVELOPMENT/ TESTING OF SOFTWARE FOR A CUBESAT FOR HIGH-RESOLUTION EARTH OBSERVATION IN A LOW EARTH ORBIT

SUMMARY

CubeSats, ranging from 1U to 27U, are small satellites many nations pursue for academic and commercial purposes. The success of their missions depends greatly on the design of their software architecture. Beyond merely achieving functionality and optimal performance, the software must also be resilient to faults and shielded from the effects of radiation, potential failures, and errors. As CubeSats accommodates more advanced subsystems, developers worldwide are exploring agile development methods. Consequently, software development must prioritize three essential factors: Modularization, refactoring, and generalization.

This study aims to describe the design, implementation, and testing of software modules of a 16U CubeSat, focusing on its onboard computer (OBC) software. A comprehensive software platform has been developed featuring a flexible architecture capable of supporting a multispectral payload and other subsystems. Multiple studies were done to familiarize the current work with experience from past projects, coding standards, and rules. Three fundamental requirements were derived to ensure software development quality: Concurrent documentation, version control for efficient tracking, and Debug tools support.

The mission software has been developed using the Free RTOS Real-Time Operating System for real-time scheduling functionality, inter-task communication, timing, and synchronization. SEU/SEL management is considered for relevant subsystems. The development environment of choice was the Eclipse IDE, with code crafted in the C language. The code architecture is structured around creating libraries for individual subsystems, which serve as building blocks for developing higher-level applications specific to each subsystem. Followed by creating subsystem managers and various operating modes (Initialization, idle, Payload operation mode, etc.) ensuring reliable operation. Finally, a mode manager is implemented which acts like a state machine handling decision-making and switching between operating modes. Additional peripherals like packet routing, housekeeping, timekeeping, data logging, and even power management have been designed to match the mission profile in these modes.

Following code development, the subsequent phase involves testing the code on actual hardware. The chosen OBC hardware has 03 interfaces; I2C for housekeeping/telemetry, JTAG for programming and debugging, and UART for development and testing. Testing of the developed code is in process for various subsystems. As future work, implementation of developmental changes is an ongoing process to ensure robustness and reliability.

ALÇAK DÜNYA YÖRÜNGESİNDE YÜKSEK ÇÖZÜNÜRLÜKLÜ DÜNYA GÖZLEMİNE YÖNELİK BİR CUBESAT YAZILIMI GELİŞTİRİLMESİ/TEST EDİLMESİ

ÖZET

Küp uydular, birçok ülke tarafından akademik ve ticari amaçlarla geliştirilen, boyutları 1U'dan 27U'ya kadar değişen, küçük uydulardır. Görevlerinin başarısı büyük ölçüde yazılım mimarisinin tasarımına bağlıdır. İşlevsellik ve optimum performans elde etmenin ötesinde, yazılım hatalarına karşı dayanıklı olmalı, radyasyona olası etkilerine ve arızalara karşı korunaklı olmalıdır. Küp uydular daha gelişmiş alt sistemleri barındırdıkça, dünya çapındaki geliştiriciler atik geliştirme yöntemlerini araştırmaktadır. Sonuç olarak, yazılım geliştirme üç temel faktöre öncelik vermelidir: Modülerleştirme, yeniden düzenleme ve genelleştirme.

Bu çalışmanın amacı, 16U boyutundaki bir küp uydunun yazılım modüllerinin tasarımını, uygulanmasını ve test edilmesini, esas olarak Yerleşik Bilgisayar (OBC) yazılımına odaklanarak açıklamaktır. Küp uydular, alçak dünya yörüngesinden Dünya yüzeyinin yüksek çözünürlüklü görüntülerini yakalamayı amaçlar. Görüntüleyici, 1,5 m GSD'ye sahip çok spektrumlu bir çizgi tarayıcıdır. Görüntüleyici, küp uydular görevindeki tek yüktür; geri kalan tüm donanım, görüntüleme işlemine yardımcı olmak için birleştirilmiştir. Yer istasyonuna görüntü aktarımında yüksek veri hızlı X-bant verici kullanılmış, TTNC için ise S-bant alıcı-verici devreye alınmıştır. Bu RF iletişim modüllerinin dışında üç adet pil paketli EPS alt sistemi kullanılmıştır. Gerekli elektrik enerjisini toplamak ve pil paketlerinde depolamak için hem gövdeye monte hem de açılabilir güneş panelleri bulunmaktadır. Uyduyu yönlendirmek için güneş sensörü, yıldız izleyici gibi çoklu sensörlere ve reaksiyon tekerlekleri gibi aktüatörlere sahip bir ADCS alt sistemi kullanılmıştır. ADCS alt sistemi, yönlü görüntüler için uyduyu yönlendirebilecek ve Dünya'ya görüntü verisi iletimi için yer istasyonunu takip edebilecektir. Son olarak küp uydular, yönlü iletişim için güneş panellerini ve antenleri yerleştirmek için kullanılacak özel olarak geliştirilmiş bir çevre kartına sahiptir.

Yukarıda belirtilen tüm alt sistemlerin ve görev yüklerinin entegrasyonunu ve düzenli çalışmasını sağlamak için esnek bir mimariye sahip kapsamlı bir yazılım platformu geliştirilmiştir. Geçmiş projelerden edinilen deneyimlere, kodlama standartlarına ve kurallara aşina olmak için çok sayıda çalışma yapılmıştır. Gereksinim analizinin taslağının hazırlanması, tasarım ve geliştirme, doğrulama ve doğrulama ve dağıtım sonrası bakımı içeren ünlü NASA Uçuş Yazılımı (FSW) geliştirme yaşam döngüsü takip edilmiştir.

Görev gereksinimlerinin taslağının hazırlanmasına kapsamlı bir değerlendirme ve özel zaman ayrılmış ve bu gereksinimler iki ana sınıfa ayrılmıştır; fonksiyonel ve fonksiyonel olmayan gereksinimler. İşlevsel olmayan kategoride, yazılım geliştirmenin kalitesini sağlamak için sonuçlandırılan üç temel gereksinim; eş zamanlı

dokümantasyon, etkin takip için sürüm kontrolü ve yazılımın yeniden kullanılabilirliği olarak belirlenmiştir. İşlevsel kategoride, geliştirme ortamı, görüntü ve veri indirme, yönlendirme ve kararlılık ve güç üretimi gereksinimlerinin her biri ayrı ayrı ve kapsamlı bir şekilde tanımlanmıştır.

Görev yazılımı, gerçek zamanlı planlama işlevselliği, görevler arası iletişim, zamanlama ve senkronizasyon için RTOS Gerçek Zamanlı İşletim Sistemi kullanılarak geliştirilmiştir. İlgili alt sistemler için SEU/SEL yönetimi dikkate alınır. Tercih edilen geliştirme ortamı, kodun C dilinde hazırlanmış olduğu Eclipse IDE'ydi. OBC'ye paylaşılan kaynaklar için rekabet edecek birden fazla alt sistem eklendiğinden, semaforlar, muteksler, kuyruk yapıları vb. gibi RTOS modülleri uygulamaya konmuştur.

Seçilen OBC'nin birden fazla arayüzü var; bunlar ön hazırlık/telemetri için I2C, programlama ve hata ayıklama için JTAG ve geliştirme ve test için UART. İlk adımlardan biri, tüm alt sistemin bağlantıları için bir Donanım Soyutlama Katmanı (HAL) tanımlamaktır. Bu HAL çoğunlukla alt sistemlerin çoğu ile OBC arasındaki I2C bağlantısından oluşur, ancak OBC yükünü azaltmak ve görüntü verilerinin X bandı üzerinden doğrudan iletimi için kamera ile X bandı arasında yüksek hızlı uzay kablosu kullanılmıştır.

Kod mimarisi, her alt sisteme özel üst düzey uygulamalar geliştirmek için yapı taşları görevi gören, bağımsız alt sistemler için kitaplıklar oluşturma etrafında yapılandırılmıştır. Bu alt sistemler, Alt Sistem Yöneticileri adı verilen RTOS iş parçacıkları tarafından kontrol edilmektedir. Bu yöneticiler sistemin mevcut durumunu kontrol eder ve belirli bir çalışma modunun ihtiyacına göre sistemi uydu için çalışır durumda tutar. Alt sistem yöneticilerinin oluşturulmasının ardından uydunun yedi çalışma modu nihai hale getirilmiştir, bu modlar; Fırlatma ve Erken Operasyonlar (LEOP), Bekleme, Şarj, Tehlike, Kalibrasyon, Kamera ve Görev Yükü çalışma modu. Bu çalışma modları FSW'nin çalışma bloğunu oluşturur. Uydu bu çalışma modlarından birine girer ve Mod Yöneticisi adı verilen bir modül tarafından kontrol edilir. Mod yöneticisi, karar almayı ve çalışma modları arasında geçiş yapmayı sağlayan bir durum makinesi gibi davranır. Görev Kontrolcüsü adı verilen bir modül tarafından desteklenir. Yer istasyonu kullanıcısı bir görüntü yakalama görevi veya görüntü verisi indirme görevi gerçekleştirmek istediğinde, görev kontrolcüsü programlanmış görevleri ayarlar ve uydunun belirli bir çalışma moduna girmesine izin verir. Bahsedilen görev kontrolcüsü aşamalı olarak tasarlanmıştır; bu aşamalar görev hazırlığı, görevin yürütülmesi ve görevin tamamlanmasıdır. Her aşamaya ilişkin zamanlamalar kullanıcı tarafından Yer istasyonundan iletilir ve alt sistemlerin belirlenen görevleri, ilgili alt sistem yöneticisi kullanılarak çalışma modu tarafından gerçekleştirilir.

Yedi adet uydu çalışma modu ve bunların geliştirilmesinin yanı sıra, FSW mimarisinin önemli bir parçası olan ek çevre birimleri de bulunmaktadır; bunlar arasında paket yönlendirme, ön hazırlık ve telemetri kaydetme ve iletim, zaman işleyişi ve izleme uygulaması, veri kaydı ve dosya sistemi operasyonları gibi modüller bulunur. Tüm bu ilgili modüller, bu modlardaki görev profiline uyacak şekilde dikkatlice tasarlanmıştır.

Küp uydu iletişim mimarisi bir başka karmaşık zorluktur. TTNC mimarisinin tamamı, paket mimarisi (CCSDS), OBC ve S-band üzerinden paket iletimi ve yer istasyonu

yazılımından gelen veri paketlerini ayrıştırmak için AX.25 çerçeveleme konularında derinlemesine bilgi sahibi olmayı gerektirmiştir. Mimari, iletişim yöneticisi tarafından kontrol edilen iletim, alım ve iletişim hizmet bloklarından oluşur.

Kod geliştirmeden sonraki aşama, kodun gerçek donanım üzerinde test edilmesini içerir. Bu aşama için sağlam bir test çerçevesi uygulanmıştır. Test üç ana aşamaya ayrılmıştır; her bir alt sistemin birim testi, entegre testi ve otonom modda, gerçek yörünge senaryosuymuş gibi, komple donanım kullanılarak yapılacak son test.

Görüntüleyicinin birim testinde, ilk kontrolleri gerçekleştirmek ve işlevsellik ile servis elverişliliğini değerlendirmek için görüntüleyici şirketi tarafından sağlanan Yer Destek Ekipmanı (GSE) ve Python komut dosyaları kullanılmıştır. Diğer alt sistemler için, başarılı I2C iletişimini görselleştirmek amacıyla PC-104 konnektör pinleri ve mantık analizörü kullanılmıştır. Bu ilk birim testlerinin ardından OBC, EPS, S-bant alıcı-verici, X-bant verici, görüntüleyici vb. alt sistemler entegre edilmiştir.

Kritik çalışma modları; kamera ve görev yükü veri iletimi tamamen yer istasyonu yazılımı ile gerçekleştirilmiş ve test edilmiştir. Ayrıca yer istasyonuna giden ve yer istasyonundan gelen birçok TTNC paketi uygulamaya konulmuş ve doğrulanmıştır. Artık odak noktası, ADCS kodunun uygulanması ve büyük ölçüde ADCS alt sistemine dayanan çalışma modlarının çoğunun başarılı bir şekilde uygulanmasıdır. Gelişimsel değişikliklerin uygulanması sağlamlık ve güvenilirliği sağlamak için gelecek yönelik çalışma olarak devam eden bir süreçtir.

1. INTRODUCTION

1.1 Purpose of Thesis

The purpose of this thesis is to explain the requirements and architecture of a 16U CubeSat Flight Software (FSW), followed by a detailed discussion of its design procedure, implementation, testing, and verification

The first chapter includes an introduction to CubeSat and some basic information on Flight software for space missions. The Second Chapter discusses specifically the work-in-progress i.e., the development of a 16U CubeSat with high-resolution imager. Details have been given in this chapter of all the hardware in use and intended operations. The third chapter completely focuses on the FSW architecture of said 16U CubeSat, thoroughly discussing the design and implementation. The fourth chapter describes the testing completed up till now with the results and the ongoing work. The final chapter gives the conclusion of this thesis with a brief description of its impact on future studies/projects.

1.2 CubeSats

A CubeSat is a class of satellites that adopt a standard size and form factor defined as ‘U’. 1U is a 10cm cube with a mass of up to 2kg [1]. The size of a CubeSat can be up to 27U [2]. Due to this relatively large range, CubeSats can either be a nano (1–10kg) or a micro (10-100kg) satellite.[3] A CubeSat, just like other satellites, includes an onboard computer (OBC), electronic power system (EPS), and communication equipment (transmitter and receivers) at various radio frequency (RF) bands. Also depending on the mission, a CubeSat could include an attitude determination and control system (ADCS), propulsion system, etc. These subsystems are there to support the payload of the satellite. CubeSats are designed to reduce cost and development time, and increase accessibility to space [1]. Till 2024 around 2600+ CubeSats have

been launched. Although CubeSats are usually launched into low Earth orbits (LEO), recently they have been deployed successfully into lunar and Martian orbits.

1.2.1 CubeSat research at İTÜ

CubeSat research at İstanbul Technical University (ITU) is done by the Space Systems Design and Testing Laboratory (SSDTL). It is one of the world's first makers of a CubeSat, and first in Türkiye. Since its establishment in 2007, 07 CubeSats have been designed and launched. The Flight Software explained in this document is being designed for a 16U CubeSat, currently under development in USTTL. The Engineering Qualification Model for the under-development 16U CubeSat can be seen in Figure 1.1.

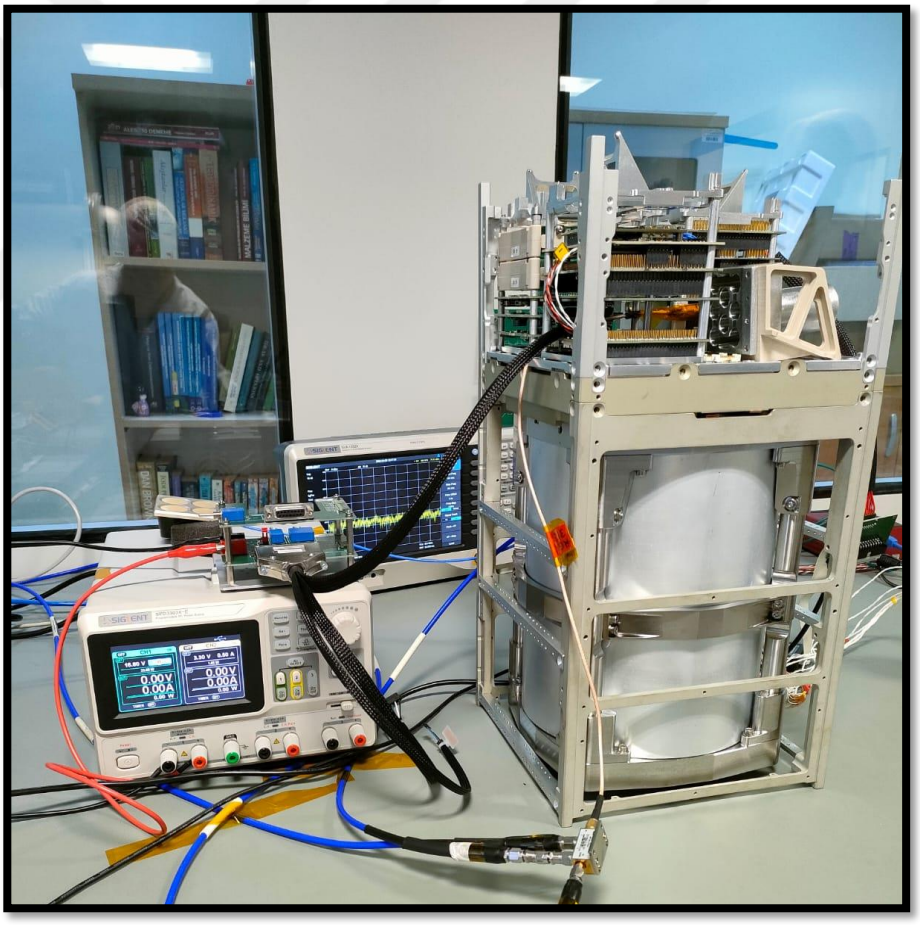

Figure 1.1 Image of 16U CubeSat EQ Model

1.3 Satellite flight software information

Satellite flight software is the basics of modern space mission design. It plays a crucial role in the successful operation of satellite missions. Hence, the software is responsible for various essential functions, including attitude and orbit control, data handling, and communication management [2]. By ensuring precise orientation and positioning, the flight software allows the satellite to maintain its intended orbit and point its instruments accurately toward Earth. Additionally, it manages the collection [3], processing, storage, and transmission of data, ensuring that the satellite's valuable observations are accurately sent to the ground station.

1.3.1 Historical missions

Several historical missions over the past decades have demonstrated the critical role of flight software in achieving required mission goals. A table of such historical missions with critical flight software is depicted in Table 1.1. Among these, one of the most notable examples is the Hubble Space Telescope, launched in 1990. Since its launch, Hubble's flight software has undergone multiple upgrades over the years, thus, allowing it to adapt to new scientific objectives and improve its capabilities [3]. The software can manage complex tasks such as precise pointing and scheduling of observations which has subsequently led to numerous groundbreaking discoveries in astronomy.

Table 1.1 Historic Satellite Missions with Critical FSW.

Mission	Launch Date	Primary Objective	Key Contributions of Flight Software
Hubble Space Telescope	24-04-1990	High-resolution astronomical observations	Precision pointing, autonomous scheduling, adaptable software
Mars Reconnaissance Orbiter	12-08-2005	High-resolution imaging	Imaging, data transmission, autonomous orbit adjustments
Landsat 8	11-02-2013	Earth observation and monitoring	Efficient data handling, orbit control, operational adjustments

Another exemplary mission is the Mars Reconnaissance Orbiter (MRO), which has been orbiting Mars since 2006 [4]. The MRO's flight software has been designed to handle high-resolution imaging and data transmission back to Earth. The highly sophisticated software has enabled the orbiter to perform detailed scans of the Martian surface, identifying potential landing sites for future missions and studying the planet's climate and geology. Thus it may be safely said that the reliability and efficiency of the MRO's flight software have been pivotal in its success, making it one of the most productive missions in the history of Mars exploration.

1.3.2 Modular design

All the historic, modern, and ongoing missions have one thing in common; a modular design. This modular flight software design is a key approach to enhancing satellite operations' reliability, maintainability, and flexibility. In this concept, the software is divided into distinct, interchangeable modules, each responsible for specific functions [5]. This separation allows for easier development, testing, and debugging, as well as the ability to update or replace individual modules without affecting the entire system.

Some key features of Modular Design are shown in Fig 1.1 below and discussed ahead.

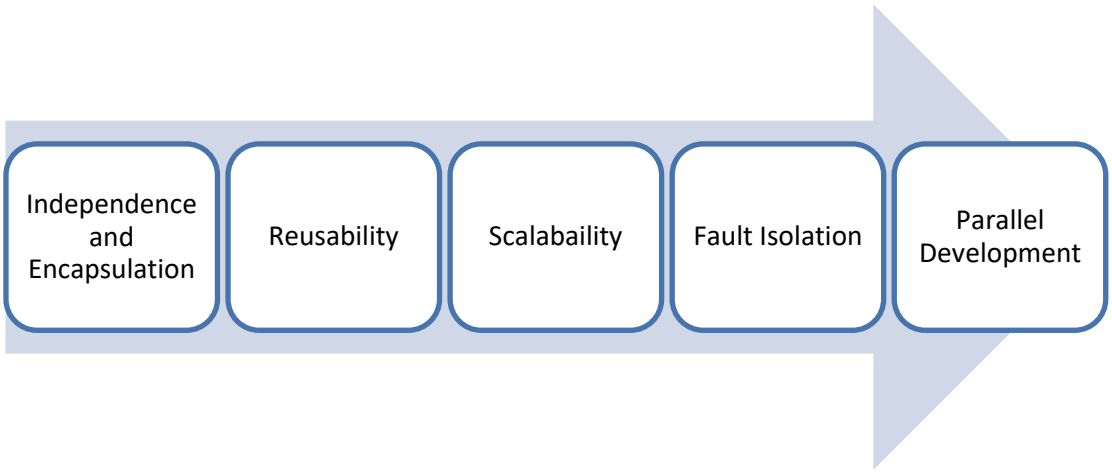

Figure 1.2 Software Modular Design.

- **Independence and Encapsulation:** Each module operates independently, with well-defined interfaces for communication with other modules. This

encapsulation ensures that changes within one module do not propagate errors to other parts of the system.

- **Reusability:** Modules may be reused across different missions, reducing development time and costs. Proven modules from previous missions can be adapted and integrated into new projects with minimal changes.
- **Scalability:** Modular design allows for the system to be scaled easily by adding or removing modules as needed. This is particularly useful for missions with evolving requirements.
- **Fault Isolation**: By isolating functions into separate modules, any fault or failure can be contained within a single module, preventing it from affecting the entire system. This enhances the fault tolerance of the satellite.
- **Parallel Development**: Different teams can work on different modules simultaneously, speeding up the development process. This also allows for specialized expertise to be applied to specific parts of the software.

1.3.3 FSW development lifecycle

The development lifecycle of flight software is a detailed and tedious process that ensures the software is reliable, efficient, and capable of meeting the rigorous demands of space missions. This lifecycle involves several key stages, each critical to the success of the overall system. The primary stages in the flight software development lifecycle are Requirements Analysis, Design and Implementation, Testing and Integration, Deployment, and finally Maintenance [6]. These are discussed further below and shown in Fig 1.3.

1.3.3.1 Requirement's analysis

It is the initial stage where the foundation of the software is laid out. This stage involves collaborating closely with stakeholders to gather and define both functional and non-functional requirements. The goal is to create a comprehensive [7] document that outlines the software's expected functionality, performance criteria, safety protocols, and interface requirements. This document serves as a critical reference

throughout the development process, ensuring that all team members have a clear and consistent understanding of what the software must achieve.

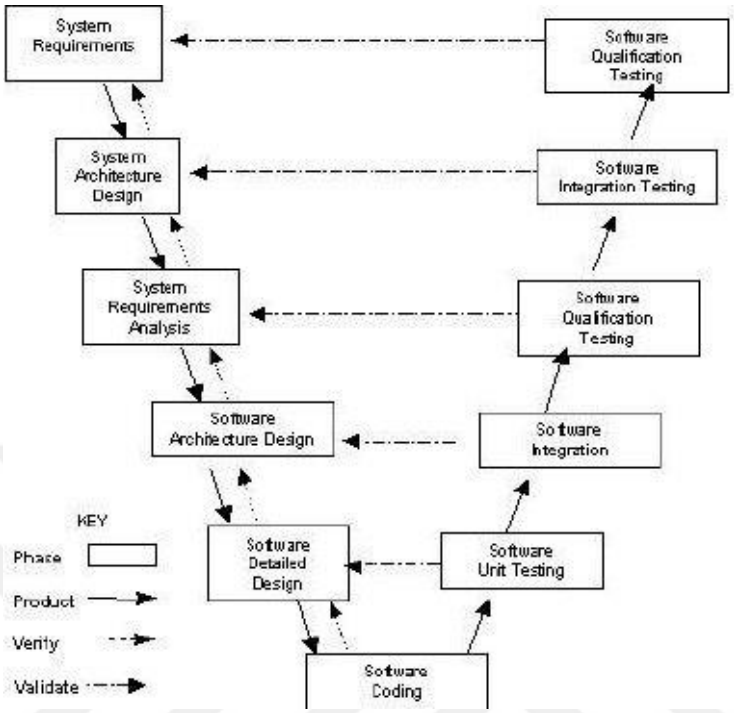

Figure 1.3 FSW Development Lifecycle.

1.3.3.2 Design and implementation

This stage is split into two sub-stages: high-level design and detailed design. The high-level design focuses on creating architecture diagrams and defining the modules and components along with their interactions. [8] It emphasizes independence and encapsulation to ensure a modular and maintainable system. The detailed design delves into the specifics of each module, outlining the internal structures, data flows, and algorithms. This detailed planning ensures that the software design supports reusability and scalability, which are crucial for accommodating future missions and updates. The implementation/ coding phase follows, where developers implement the designed

modules, adhering to coding standards and best practices. Regular code reviews are conducted to identify and rectify issues early, ensuring a robust and reliable codebase.

1.3.3.3 Testing and integration

This phase ensures that the software functions correctly and meets all specified requirements. Unit testing is performed on individual modules to verify their functionality in isolation. Integration testing follows, where modules are combined and tested as a group to ensure they work together seamlessly. System testing is conducted in an environment that simulates the actual operational conditions of the satellite, verifying the software's overall performance and reliability. This stage is critical for identifying and addressing any issues that could impact the mission [9].

1.3.3.4 Verification and validation

Verification and validation (V&V) are critical processes in the development of flight software, ensuring that it meets requirements and performs reliably under the extreme conditions of space. These processes are essential to guarantee the safety, reliability, and functionality of the software, which in turn ensures the success of the satellite mission [6]. The process of verification involves checking that the software is built correctly according to the design specifications. Key steps in verification include requirements review, code reviews and inspections, unit testing, integration testing, and simulation testing [9].

1.3.3.5 Reliability and fault tolerance

Reliability and fault tolerance are critical attributes of flight software, ensuring that satellite missions can withstand the harsh and unpredictable environment of space. Reliability refers to the software's ability to perform its required functions under specified conditions for a designated period, while fault tolerance is the software's capacity to continue operating correctly in the event of hardware or software failures.To achieve high reliability, flight software undergoes rigorous testing and validation processes to detect and rectify errors before launch. [9] These processes include extensive unit testing, integration testing, and system testing under simulated space conditions. Additionally, redundancy is often built into the software architecture,

allowing multiple pathways for critical functions, so that if one component fails, others can take over seamlessly. Fault tolerance is mainly achieved through techniques such as error detection correction and watchdog timers.

1.3.3.6 Deployment

This involves preparing the software for installation on the satellite, including finalizing documentation, configuration, and any necessary training for operators. The deployment must be meticulously planned to minimize risks and ensure a smooth transition to operational status. The software is then uploaded to the satellite, where it becomes fully operational.

1.3.3.7 Maintenance

This stage begins once the software is deployed. This stage involves ongoing monitoring, updating, and optimizing the software to address any issues that arise during the satellite's mission. Maintenance ensures that the software continues to perform effectively throughout its operational life, incorporating any necessary updates or enhancements to meet evolving mission needs.

1.3.4 Innovations and trends

Recent advancements in satellite flight software (FSW) are focused on enhancing autonomy, thus creating more dependency on artificial intelligence, and improving resilience and efficiency to meet the evolving demands of modern space missions.

One of the most significant trends is the increased autonomy of satellite operations. Modern flight software is designed to perform complex decision-making processes independently, thus, reducing the need for constant ground control intervention. This autonomy is crucial for deep-space missions where communication delays can hinder real-time control [10]. Autonomous satellites can manage their orbits, conduct scientific observations, and even handle minor system malfunctions without human intervention.

Additionally, Artificial intelligence (AI) and machine learning (ML) are also making significant contributions to satellite flight software missions. AI algorithms are being

integrated to optimize tasks such as image recognition, anomaly detection, and predictive maintenance. For example, AI can analyze vast amounts of data collected by satellites to identify patterns and anomalies that might be missed by human operators. This capability enhances the efficiency and effectiveness of satellite missions, enabling more accurate and timely data analysis.

Another innovation is the adoption of software-defined satellites, which allow for greater flexibility and adaptability. Software-defined satellites can reconfigure their functions and operations through software updates, enabling them to adapt to new missions or changing conditions without the need for physical modifications. This adaptability extends the lifespan of satellites and maximizes their utility.

2. BACKGROUND

2.1 16U CubeSat Mission

The ongoing project discussed in this study is a 16U-sized CubeSat that is being developed by a collaboration between Air University (AU), Pakistan, and Space Systems Design and Test Laboratory of Istanbul Technical University (SSDTL-ITU). The main mission of the satellite is to take high-resolution images of Earth's surface. Hence, all the subsystems have been selected to aid the mission objective. The design of the satellite is complete. However, the manufacturing, testing, and programming of the satellite is underway. The main payload for this CubeSat is a multi-spectral line scan imager. Apart from the imager, the satellite houses an EPS with 03 battery packs, an S-band transceiver, an X-Band transmitter, OBC, an interface board, a real-time clock, an ADCS system with a star tracker, GPS, and solar panels which can be seen in CAD model shown below in Fig 2.1. Currently, the satellite's Engineering Qualification Model is being built and the Flight Model is expected to be launched in the final quarter of 2024.

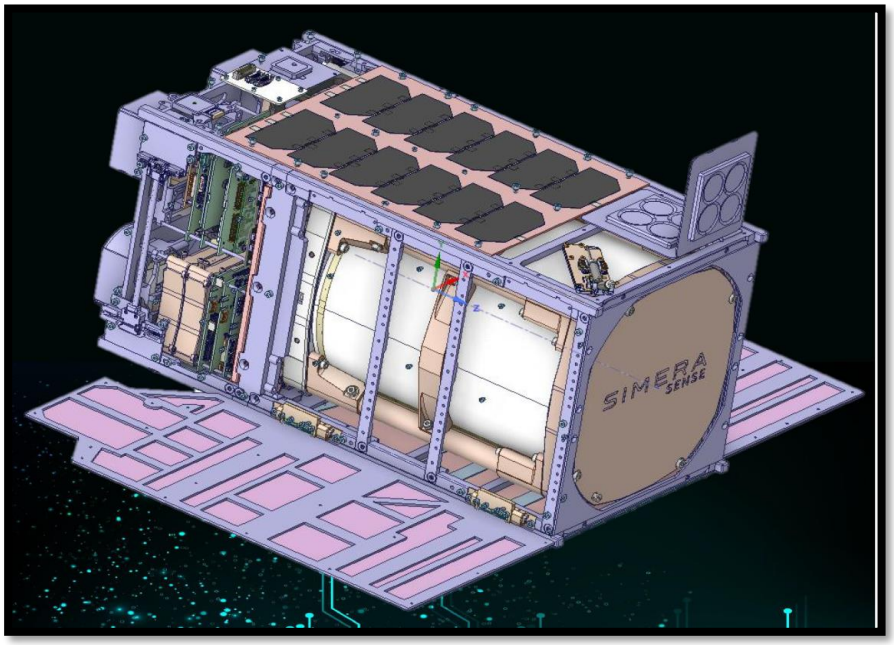

Figure 2.1 CAD Model of 16U CubeSat.

2.1.1 OBC selection

Out of all the hardware, the most important subsystem was the selection of OBC as it will house the complete software and act as a command and handling system for all other subsystems. The selected OBC is from ISISpace. It is built around a 32-bit ARM9-based processor operating at 400 Mhz. The system has 32MB SDRAM for data and 1MB NOR flash for code storage. During boot-up, the code is copied from the NOR Flash to the RAM for faster execution. For mass non-volatile storage, OBC is integrated with 2x16GB SD cards and also has high endurance 256kB FRAM for critical data storage.

The FRAM is a non-volatile storage unit that is overall more robust than Flash or EEPROM memories. It can sustain trillions of writes and retain data for more than 151 years at 65 °C, as opposed to a 20-year lifetime of typical Flash memories. It has fast access and does not require page-erase cycles. Additionally, the memory cells of the FRAM are resistant to single-event upsets (SEU). Consequently, it is the best choice for storing critical flight parameters. The amount of memory and the protection mechanisms eliminate the need to add additional memory device that will complicate the system.

The SD-Cards used in the OBC are high-quality industrial SD-Cards with Single Level Cell (SLC) Flash memory to improve reliability. Figure 2.2 shows the ISISpace On-Board Computer.

Other features of the OBC are listed below;

- 2 redundant real-time clocks
- External watchdog and power supervisor
- Temperature, current, and voltage measurements with over-current protection
- 1 Up to 400kbit/s I²C peripheral
- 1 SPI supporting bit rates up to 10Mbit/s.
- 2-level configurable UARTs

- 1 10-bit ADC for general-purpose usage
- JTAG and UART interface for programming and debugging

Figure 2.2 ISISpace On-Board Computer.

The working schematic of OBC is shown in Fig 2.3.

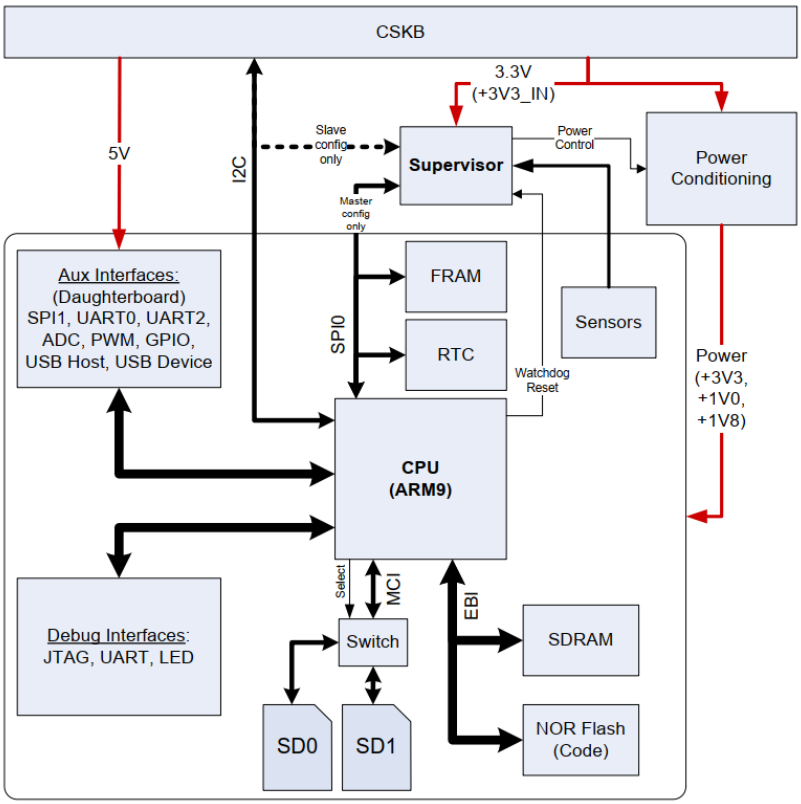

Figure 2.3 ISISpace OnBoard Schematic Diagram.

2.1.1.1 OBC interfaces

The OBC (iOBC) came with a software development kit (SDK) which includes several tools to use OBC and create interfaces between hardware and software. The hardware abstraction layer tool in SDK includes the following drivers provided by ISIS: I²C, SPI, UART, ADC, PWM, GPIO, LED, FRAM, timing, watchdog reset, and supervisor interface. In addition, SDK also includes libraries from ATMEL and FAT32 file systems. Two communication interfaces have been used on the satellite to collect housekeeping data and send commands to all subsystems.

- `An I²C line has been designed as a bus and distributed between each subsystem. This line handles all low-speed configuration and telemetry between subsystems and OBC. Each subsystem on the I²C bus includes a high impedance I²C buffer on the bus side. This enables a level of I²C bus redundancy. The OBC of the satellite acts as a master in the I²C bus where every other subsystem acts as a slave. The nominal voltage of the I²C bus is +3.3V.

- A high-speed LVDS line is placed between the S-Band transmitter and OBC to send telemetry data to the ground station.

Additionally, OBC has a JTAG interface for programming and debugging. It has a dedicated debug UART interface as well, and both are currently in use for development. The drivers for bus interfaces (I2C, SPI, UART) use DMA for efficient transfer and employ Free RTOS queues for flexibility and maintaining atomicity of transactions.

2.1.2 Main payload

The Payload to be used in this 16U CubeSat is Simera Sense MultiScape 200 CIS multispectral push-broom imager. The payload is designed to provide a GSD of 1.5m/px resolution at an orbital height of 500km and is sized to fit in 12U volume with a mass of around 12kg. Additionally, it is designed to resist the associated space environment and was received after thorough testing. The Fig 2.4 shows the part breakdown of the imager [11].

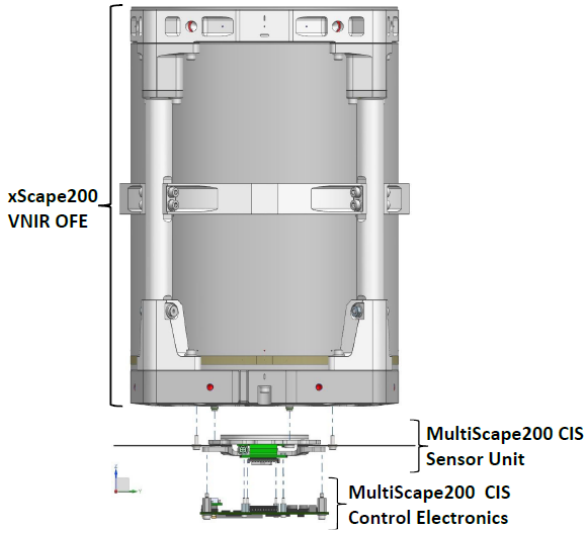

Figure 2.4 Parts of Imager Camera Payload.

The detailed specifications of the imager are mentioned in Figure 2.5.

Optics	
Focal Length	1067 mm ±1 mm
Aperture	190 mm
Full Field of View (FFOV)	1.6° (across-track); 1.2° (along-track)
Imaging	
Configuration	Line-scan (push broom)
Sensor Technology	CMOS
Across Track	9344 pixels
Pixel Size	3.2 µm
Pixel Depth	12-bit
Spectral Bands	7
dTDI Stages	Up to 32 per band
Line Rate	Up to 8000 Hz
Signal to Noise Ratio	36 for Band 2 using 8 dTDI stages (1)(3)
On-Board Electronics	
Storage Capacity	1Tbit (128GByte) EDAC protected NAND Flash
Continuous Strip Length	Up to 500 km (1)(2)
Image Processing	Binning, Thumbnails
Image Compression	CCSDS 122.0-B-2 Lossy/Lossless (optional) (4)
Control Interface Options	I2C, SPI SpaceWire (ECSS-E-ST-50-12C) (optional) RS-422, RS-485 (optional) CAN 2.0B (optional)
Data Interface Options	LVDS SpaceWire (ECSS-E-ST-50-12C) (optional) USART (optional)
Physical Interface Options	CSKB Compatible PC-104 connector Wirable high-speed connector
Power Supply	5 V DC ± 150 mV
Power Consumption	4 W when idle or during readout 8.75 W during imaging
Mechanical	
Mass	12.1 kg ± 2%
Dimensions	216 x 216 x 304 mm
Environmental	
Operating Temperature Range	-10 to 50 °C
Survivable Temperature Range	-20 to 60 °C
Sun-facing duration	Sun can be within FFOV for up to 3 minutes

Figure 2.5 Table of Payload Specifications.

2.1.3 Payload data transmission

After performing imaging operations, the imagery data is planned to be transmitted to the Ground Station via an X-Band Transmitter. X-Band is part of the electromagnetic spectrum's microwave band ranging from 8 to 12 GHz. The high operating frequency of the band enables the communication system to operate at higher bit rates and bandwidths than lower frequency bands. Therefore, to meet the satellite's high data rate transfer requirement, an X-Band transmitter has been used to provide a high data rate downlink channel from the satellite to the ground station. For achieving this functionality, Cubecom's X-Band Transmitter (XTX) have been used. The X Band Transmitter is shown in Figure 2.6 [12].

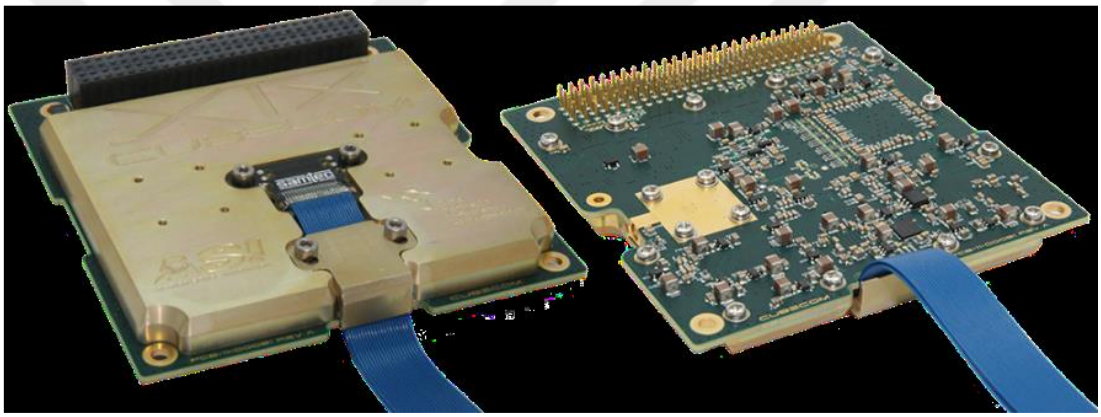

Figure 2.6 Cube Comm X Band Transmitter.

The selected RF transmitter module operates in X-Band frequency, ranging from 8.0GHz to 8.4GHz with an option of in-flight configurable frequency. Additionally, useful transmission data rates up to 80 Mbps are achievable with configurable 100kbps steps. Additionally, the transmitter supports RF output power from 27dBm to 33dBm with 1 dBm steps adjustable configuration.

As the primary connector for the satellite is the PC/104 header, it is also supported by this transmitter module. This PC-104 connector supplies power and facilitates various I/O functions such as I2C, CAN, and control I/Os. To prevent dead logic, the I/Os through the PC/104 connector is isolated from the FPGA using protection buffers when the board is powered off.

The power amplifier is powered by battery voltage, and the rest of the board is configured to operate from a 5V bus available from the satellite power supply subsystem. Additionally, the XTX can be powered down externally through an XTX enable line on the PC/104 header. The transmitter supports 6v to 30v input supply voltage levels. The safe temperature range for the XTX is -25°C to 61°C.

In the XTX, options for selecting various MODCODs are available, hence the XTX supports QPSK, 8PSK, and 16APSK modulation schemes as per the DVB-S2 standard. Apart from modulation, the symbol rate and the center frequency of operation can be configured in Flight.

The electrical characteristics of XTX are shown in Figure 2.7.

Parameter	Note	Value (typical)	Unit
Power			
Total Power Consumption			
Off Mode Power		40	mW
Standby Mode Power	All RF logic OFF	1.5	W
Config Mode Power	PA Enable OFF	5.7	W
Transmit Mode Power (27dBm)	PA Enabled	10.9	W
Transmit Mode Power (30dBm)	PA Enabled	12.4	W
Transmit Mode Power (33dBm)	PA Enabled	14.5	W
Battery VBAT pin (5V Bus power (6W) must be added to this pin if single VBAT supply is used on option sheet)			
Nominal Voltage		7.2	V
Current	PA Enabled	1230	mA
Current	XTX Disabled	70	µA
5V Bus pin (if 5V bus was selected)			
Nominal Voltage		5	V
Current	XTX Enabled	1200	mA
Current	XTX Disabled	50	µA
Power Amplifier (internal after VBAT SMPS)			
Nominal Voltage		20	V
Current (typical)	27dBm output power	190	mA
	30dBm output power	250	mA

Figure 2.7 Electrical Characteristics of XTX.

There are two different communication interfaces to command the transmitter and send transmit data. A low-speed I²C bus has been used to command and configure the transmitter. However, for the payload data, an LVDS communication layer have been implemented between the transmitter and imager. This implementation has removed

the OBC interfacing between the imager and transmitter thus, in turn reducing the OBC's processing load significantly.

A compatible patch antenna from CubeComm has been incorporated with the transmitter. The selected antenna is a surface mount patch antenna with a small size and mass with good directionality. It is shown in Figure 2.8.

Figure 2.8 CubeComm X-Band Patch Antenna.

2.1.4 TT&C

TT&C stands for Telemetry, TeleCommand, and Communication which is the core of the communication system of satellite missions. Most LEO orbits make use of either UHF, VHF, or S-band frequency range for TT&C. Thus, for this 16U cubeSat, S-band TXS and RXS sets have been employed for successful TT&C.

S-Band is part of the electromagnetic spectrum's microwave band ranging from 2 to 4 GHz. To exchange commands and telemetry data between the ground station and the satellite, two different communication links have been implemented on the satellite: One downlink channel to download telemetry data and one uplink channel to send commands to the satellite. Both channels will operate at the S-Band frequencies. For these operations, ISISpace S-Band Transmitter (TXS) and ISISpace S-Band Receiver (RXS) are being used together, each handling one channel. Both subsystems are PC104

compatible and are stacked together to be used as a single transceiver system. The safe temperature range for the TXS is -40°C to 70°C, and for RXS is -20°C to 60°C [13].

A single patch antenna of ISISpace supports both the transmitter and receiver. It is shown in Figure 2.9. The antennae are fed through an RF splitter connector into the TXS.

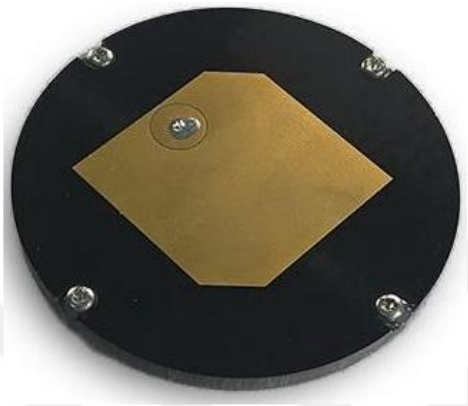

Figure 2.9 ISISpace S-band Patch Antenna.

To eliminate communication loss when the satellite is not controlled and tumbling, the system will be operated with 2 of the same antenna. These antennas are placed on opposite sides of the satellite to create an omnidirectional radiation pattern. A beacon-like operation is also integrated into the S-band transmitter.

2.1.4.1 S-Band transmitter

The ISISpace S Band TXS is a high data rate transmitter. The system incorporates CCSDS-compliant coding and forward error correction schemes which makes it more compatible with many demodulators. Up to 4.3 Mbps useful information data rate is achievable with the system. Both operating frequency and RF output power levels are configurable where the frequency is selectable with 1kHz steps and RF power is adjustable between 27 to 33 dBm [13].

The transmitter can be supplied with voltage levels between 7V to 20V which is within the range of the satellite's battery voltage range. Therefore, the system is being

supplied the power directly from the battery voltage bus. Moreover, 1 watt (30dBm) of RF output power is being used during operation which consumes about 10 watts from system supply input.

The transmitter includes two different command and data interfaces. For commanding and configuring the transmitter a low-speed I²C is being used at 400 kHz rate. Subsequently, for the housekeeping data, the system is integrated with LVDS-level SPI-type communication. The S-band Transmitter is shown in Fig 2.10.

Figure 2.10 ISISpace S-band Transmitter.

2.1.4.2 S-band receiver

The ISISpace S-Band Receiver (RXS) is being used on the satellite side for the uplink channel. It is shown in Figure 2.11. The selected receiver operates at a frequency which is configurable in-flight with 1kHz steps with a fixed data rate of 9600 bps and incorporates one of the FSK (G3RUH) or GMSK modulations [14].

The receiver supply voltage range is between 7V to 20V. Since the receiver is operable within battery voltage levels, the battery is used to supply to the receiver directly over the battery voltage bus. The average power consumption of the system is defined as

0.9 Watts which creates a good advantage in terms of the electrical power budget of the satellite.

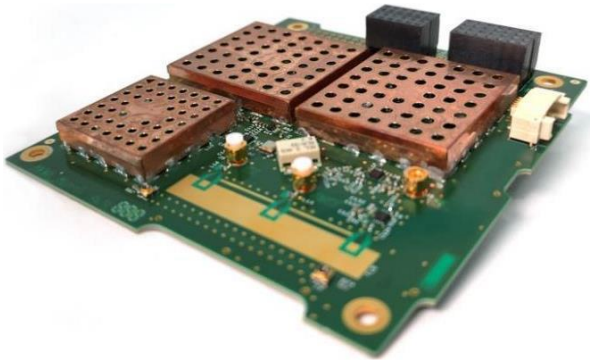

Figure 2.11 ISISpace S-Band Receiver.

The Receiver board is designed to receive frames in AX.25 format. This makes the receiver very easy to operate with a good range of modulators on the ground station side. AX.25 protocol is designed for use by amateur radio operators and stands for "*Link Access Protocol For Amateur Packet Radio".* All command and data handling is being operated over a single I²C interface as mentioned before. Moreover, to ensure good communication, a deployable antenna system is planned to maintain the antenna nadir direction in parallel with the imager nadir direction.

2.1.5 Orientation and stability

As mentioned before, the main purpose of this 16U Cubesat project is imaging for Earth observation. Therefore, the success of this satellite mission depends on the ADCS since the camera payload of the satellite requires accurate pointing and target tracking. In addition to that, ADCS is responsible for the proper pointing of S-Band and X-Band communication modules, sun pointing for power generation, and thermal management. Moreover, to acquire good quality of the payload data, the satellite shall have as little jitter and smear values as possible during the payload operation which is also the responsibility of ADCS. Thus after careful consideration, ISISpace ADCS

was selected to be put in use with all other subsystems. The purchased ADCS bundle is shown in Fig 2.12 and discussed below in detail:

- ADCS Computer: This computer performs the calculations for optimal attitude and orbit control. The software embedded inside the ADCS computer includes a robust 3-axis controller which facilitates tracking of inertial or rotating target reference frames. It runs a Cortex-M7 ARM microprocessor with a double-precision floating-point arithmetic unit for high-precision models and filters for attitude estimation at a suitably high frequency. It includes 2 sub-units:

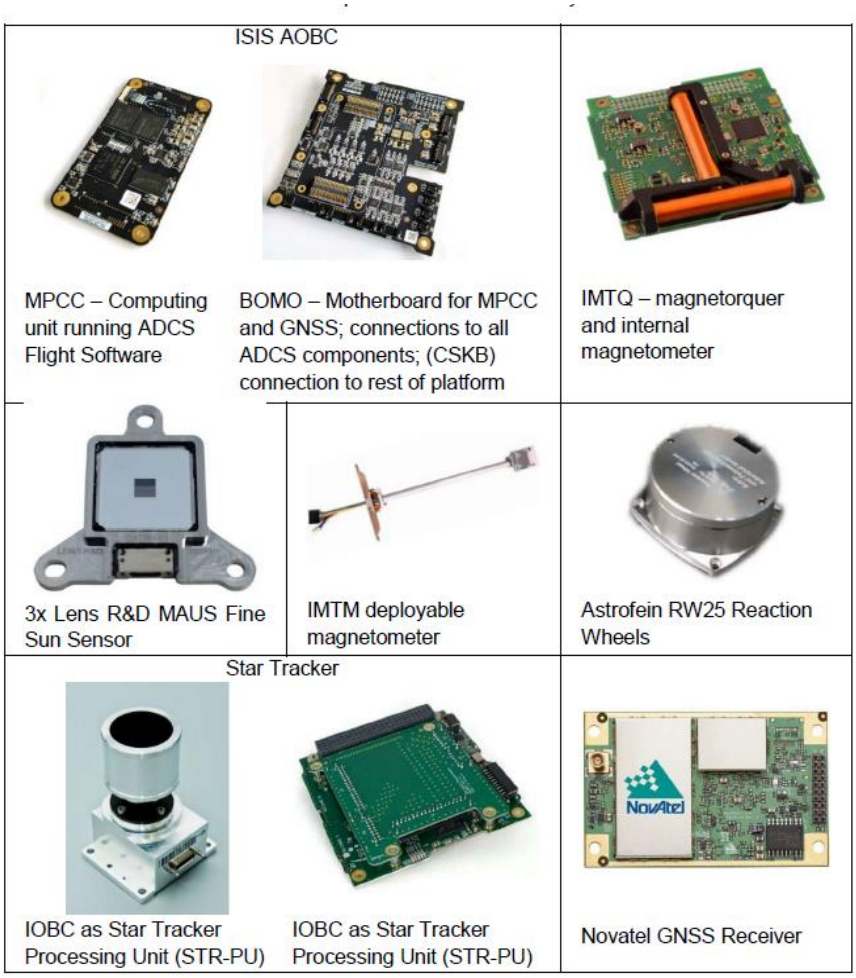

Figure 2.12 ISISpace ADCS Bundle.

MPCC and BOMO. They are two separate processors for ADCS that are stacked together.

- Star Tracker with its Computer: A star tracker that also includes an iOBC (ISISpace OBC) to run only star tracker software for easy operation and interfacing.

- Magnetorquer subsystem (iMTQ) Board: This is a PCB-based 3-axis magnetic actuation and control system for CubeSats, designed as a stand-alone detumbling system. It includes 2 magnetorquer rods and 1 air core magnetorquer. Also, it contains an onboard three-axis magnetometer for magnetic field measurement, current sensors for each torque, temperature telemetry of actuators, and includes a detumbling algorithm.

Thus, to meet all the mission requirements, the ADCS was selected to be fully-fledged including 3 fine sun sensors, 2 magnetometers, a star tracker, GNSS, 2 magnetometers, and 4 reaction wheels, The data connections for the ADCS subsystems are shown in Fig 2.13.

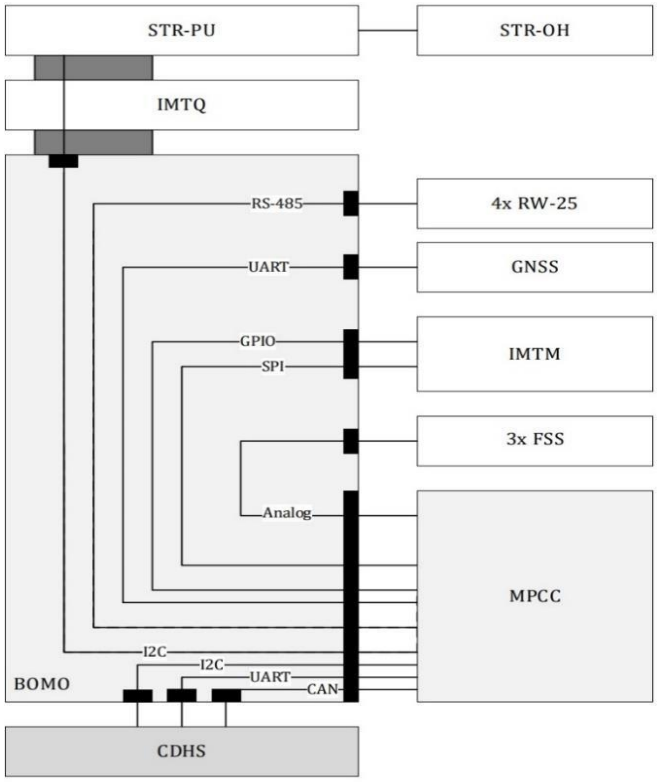

Figure 2.13 ADCS Bundle Schematic Diagram.

2.1.6 Power generation and requirement

An electrical power system (EPS) is a satellite system that handles voltage regulation, power distribution, and energy harvesting tasks. In general, an EPS system consists of several subsystems such as battery packs, battery chargers, solar arrays, and several protection mechanisms. Some of the main purposes of the EPS system are;

- Converting solar energy to electrical energy with maximum efficiency.
- Controlling charge and de-charge for efficiency and maximum lifetime.
- Storing harvested solar energy in battery packs.
- Protecting both satellite and battery from unexpected electrical events (overcurrent, under and over voltage, etc.)
- Measuring and calculating battery parameters like state of charge (SoC), state of health (SoH), current and voltage, and temperature, and reporting these parameters (telemetry) to the CubeSat computer.

Similar features were also required for the 16U CubeSat. A trade-off analysis was performed to determine which EPS system will be used for the satellite. Several parameters were taken into account for trade-off analysis including switchable power busses, supporting several battery chargers due to the high energy harvesting requirement of the satellite, and supporting high-capacity battery packs. After the analysis, the ISISpace Modular EPS v2 product was selected for use shown in Fig 2.14. The selected EPS, ISISpace Modular EPS v2, consists of multiple independent subsystems.

The functional diagram for the EPS is shown in Fig 2.15

- Power Distribution Unit (PDU)
- Power Battery Unit (PBU)
- Power Battery Pack (PBP)
- Power Conditioning Unit (PCU)

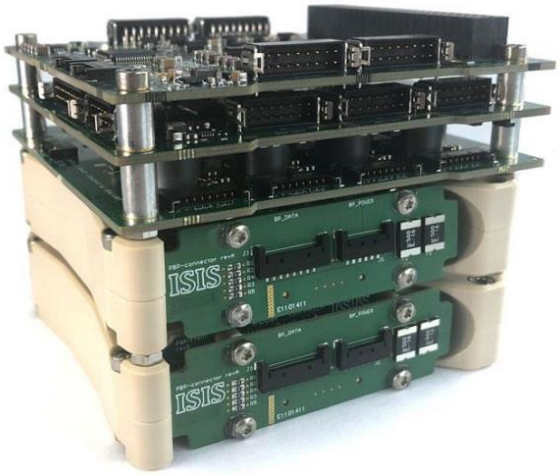

Figure 2.14 ISISpace Modular EPS.

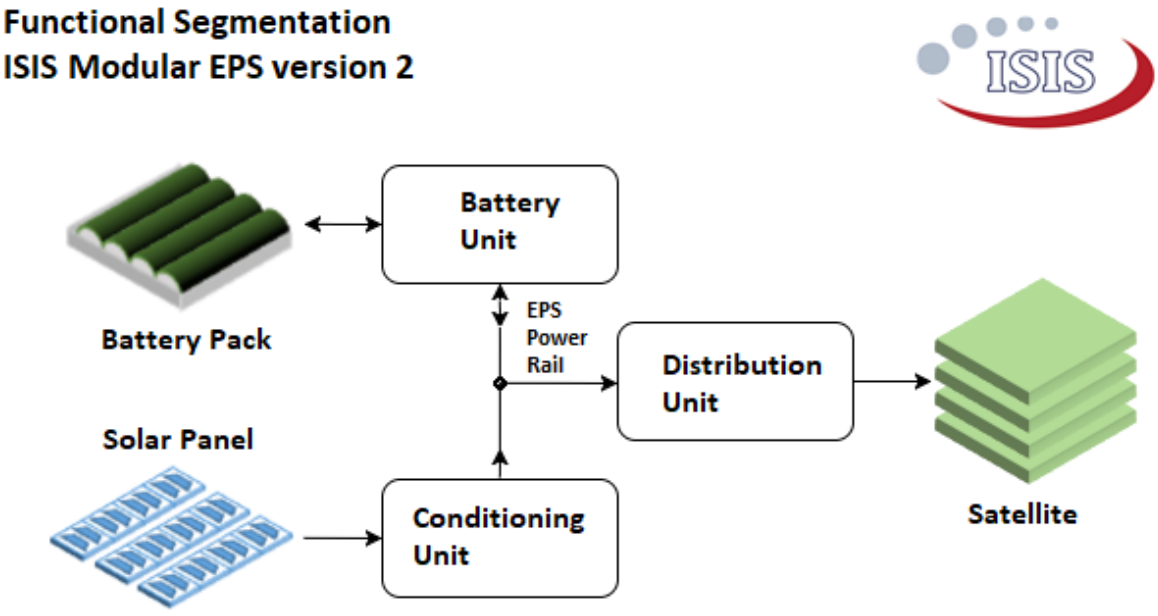

Figure 2.15 EPS Schematic Diagram.

.

The PDU is used to distribute power to the satellite in various voltages and output channels. There are 3 different voltage levels are required within the satellite's subsystems: unregulated battery voltage, +5V, and +3.3V. A single power distribution unit can provide all required voltage levels. All output channels of the PDU include overcurrent protection, backflow protection, and slew rate controls and for each output channel current, voltage, and power levels are measured and can be read from the onboard computer.

The power battery unit protects the attached battery packs and collects voltage, current, and temperature telemetry. The system also includes an embedded battery heater controller which is connected to the battery heaters and temperature sensors. The electrical power budget calculations for the satellite show that at least 81Wh of battery capacity is required to always keep the satellite operational. Considering this value, 3 battery packs are decided to be used on satellite. All these separate battery packs are connected to the PBU unit in a parallel configuration to reach the required energy capacity.

Solar energy will be used as a continuous energy source for the satellite. Harvesting will be provided over solar cells and the Power Conditioning Unit (PCU) will condition the harvested energy and will charge the battery packs. A single PCU unit has 4 different solar string channels, and each channel on the PCU subsystem is equipped with a maximum power point tracker (MPPT) to maximize the harvested energy under different illumination and temperature conditions. Each channel can be attached to a solar panel string with different configurations. Limits of allowable solar string configuration are determined by the applicable voltage and current limits of the PCU. To meet the energy harvesting requirements for the satellite, the satellite will include deployable solar panels to increase the solar array surface and, therefore the harvested energy.

Each of these subsystems, excluding battery packs, includes a pin-compatible PC104 connector and is stackable on top of each other. Battery packs are connected to the power battery unit (PBU) over a connection harness. Each unit includes both I^2C and UART peripherals for command and data handling. Due to its bus-type connection

capability, the I²C communication line will be used for each subsystem. The I²C line is +3.3V capable and supports up to 400 kHz data rates.

2.1.7 Subsystem's availability

The project started in Feb 2022, however, the purchase orders started around Sept 2022-Dec, 2022. After necessary payments and multiple back-to-back meetings with the OEM, vendors, and customs authorities the equipment arrived in the last quarter of 2023. Before the arrival of hardware, necessary documentation had been received which helped in drafting the initial designs for both hardware and software.

3. SOFTWARE ARCHITECTURE

Section 3 discusses detailed software architecture, design goals, and design limitations such as software coding standards, programming architecture, mission management, and satellite operating modes.

3.1 Design Goals

The following design goals are set for the system shown in Fig 3.1:

Figure 3.1 Design Goals for 16U CubeSat.

The design of our flight software is guided by several critical goals aimed at ensuring both current mission success and future adaptability.

A primary objective is to achieve strong re-usability for future satellite missions. By developing modular and scalable software, we are focused to enable its application across different missions which we foresee in the future. This will help us to perform minimal modifications and reduce our development time and costs significantly.

Secondly, Documenting the code thoroughly and implementing robust version control practices is another key goal. This ensures that every change is tracked, facilitating easier debugging, maintenance, and collaboration among development teams.

Additionally, the software architecture is designed to support comprehensive debug tools. This enables effective identification and resolution of issues, both during development and post-deployment, enhancing the overall reliability of the satellite operations. Moreover, we have chosen an open-source real-time operating system (RTOS) as the foundation for our software. This RTOS provides essential features such as task scheduling, which allows for efficient management of multiple processes; task notifications, which enable inter-process communication and synchronization; queueing with First-In-First-Out (FIFO) mechanisms to manage data flow efficiently; and static memory allocation, which enhances predictability and reduces fragmentation, critical for the constrained environment of space.

Finally, our design emphasizes efficient power management, reliable data processing, and robust fault tolerance. These aspects are crucial for the sustainability and success of long-term missions.

3.2 Design Considerations and Limits

The majority of CubeSats follow similar specifications and design rules for hardware, however for software the design considerations are dependent on mission goals and objectives. The greater the functionality, the more complex yet advanced software architecture is required [6] [15]. However, several design considerations and limitations were there for this project as well. These are given in Table 3.1 below.

Table 3.1 Design Considerations and Limitations.

Considerations	Limitations
Computational Constraints	Hardware Limitations
Environmental Resilience	Communication Constraints
Real-Time Performance	Development/Testing Constraints
Safety Critical Operations	Regulatory/Standardization Requirements

3.3 Literature Review and Initial Design Draft

After some discussion on possible design limitations and considerations, the next step was to carefully go through the documentation (Interface and Control documents) provided for each subsystem by their OEM. The focus in these documents was particularly on the following items

- Request/Command Transactions via possible interfaces (in our case I2C)
- Control Interface API and possible implementation
- Bootloader Commands
- Data Interface protocols such as interface formats, data packet structures, etc
- All power requirements
- Necessary cabling and wiring information

Hence, each documentation was carefully reviewed and a detailed working flowchart was made. The flowchart contained information on possible connections, to and fro wiring information, stacking scheme, chosen communication protocol, and corresponding ports.

After a careful literature review of all purchased hardware, the next step was to finalize the selection of a real-time operating system (RTOS) followed by a coding language. Free RTOS was considered the best option owing to its robust library that uses preempting, mutexes, and semaphores to enable the development of a real-time application. Details of Free RTOS are discussed further in the next section.

The board support package for the OBC was written in C language, which is a good choice for embedded systems programming as it allows direct hardware access and can be used to develop efficient programs.

In real-time applications where guaranteed interaction response times are required, a master-slave architecture is recommended. Therefore, communication between the subsystems is achieved via the I2C bus. The OBC is used as the master and all other subsystems are slaves that are operated by commands from the OBC. Like other hardware peripherals, a mutex is used to lock the I2C bus if it's in use by a subsystem, so the bus is guaranteed to be used sequentially.

To achieve portability and re-usability, abstraction for all peripherals (I2C, SPI, UART, PDMA) was done. Bare-metal drivers for each subsystem were developed so that only the abstract peripheral functions can be replaced according to the hardware (e.g., I2C read/write) and the whole driver can be used in a different system. Separating both the OS and hardware layers also helps with increasing maintainability. The layers of the software development are shown below in Fig 3.2.

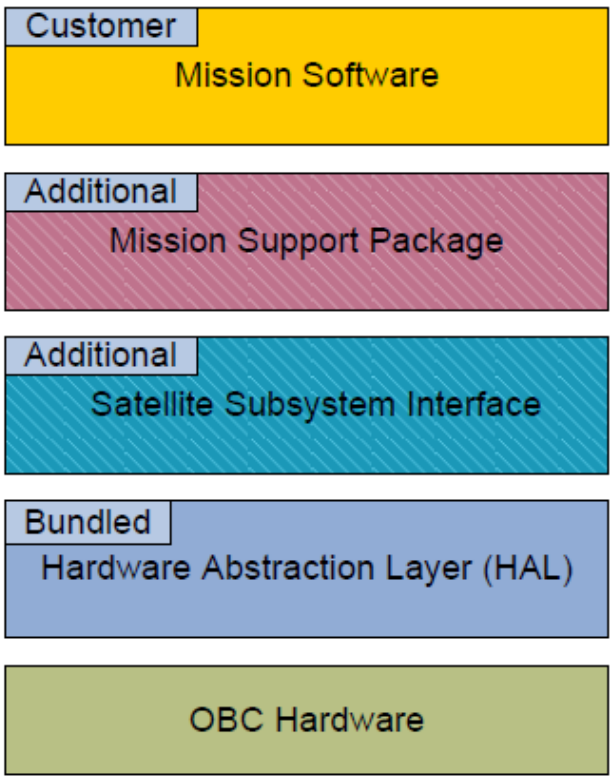

Figure 3.2 Flight Software Layers.

3.3.1 Free RTOS

As mentioned earlier, FreeRTOS Real-Time Operating System has been used for developing mission software. ISISpace's HAL libraries and ARM's CMSIS library have been used for hardware abstraction between core and peripheral (SPI, I2C, etc.) functions. These hardware abstraction libraries have been used with appropriate protection methods (mutex, semaphore, etc.) from concurrent access by RTOS tasks [15].

There are several reasons why FreeRTOS was chosen. Firstly, it is a class of RTOS that is designed to be small enough to run on a microcontroller. Secondly, Traditional real-time schedulers, such as the scheduler used in FreeRTOS, achieve determinism by allowing the user to assign a priority to each thread of execution. The scheduler then uses the priority to know which functionality, inter-task communication, timing, and synchronization primitives to perform. This means it is more accurately described as a real-time kernel or executive. Additional functionality, such as a command

console interface, or networking stacks, can then be included with add-on components thread of execution to run next. In FreeRTOS, a thread of execution is called a task.

Finally, FreeRTOS is licensed under an open-source MIT license, which makes it completely free to use unless additional backup is needed. Some additional reasons to choose FreeRTOS are listed below.

- Reliable and supported
- Compatible with the ISIS OBC, includes support packages
- Minimal ROM, RAM, and processing overhead.
- Rich in features with an active developer community and a well-established user base
- Truly free of charge
- Commercial support is available if required
- Scalable and easy to use
- Flight Heritage
- Software team experience

3.3.2 Subsystem's libraries

After discussing the basics of the initial design, in this section, we discuss the interface libraries given by OEMs along with the purchased hardware and the one prepared

using the available documentation. The list of subsystems and their OEMs is mentioned in Table 3.2.

Table 3.2 Subsystems and OEMs.

Subsystems	OEM
Payload (Imager)	Simera Sense, SA
OBC	ISISpace, Netherlands
EPS (PBU, PDU, PCU, Battery Packs)	ISISpace, Netherlands
ADCS Bundle	ISISpace, Netherlands
S-Band Transceiver with antennae	ISISpace, Netherlands
X Band with antenna	CubeComm

Maximum subsystems were procured as a bundle from ISISpace. Hence, the Interface control documents, user manuals, and software dependencies were quite similar and the subsystems could be stacked on each other using a PC 104 connecter and controlled and commanded via an I2C bus.

For OBC, a quick start guide was provided along with the datasheet. Both documents provided in depth detail of available memory, power, data and electrical interfaces, development environment, and flight software layers, Moreover, several OBC libraries were given alongside. These can be classified into two main categories

Satellite Subsystem Interface. This library allows easy implementation of interfaces between OBC to a variety of subsystems that can be used on a nanosatellite bus.

Mission Support Package. This library provides additional building blocks needed for robust and efficient OBC flight software such as FRAM parameters addition and data logging.

After OBC, our next aim was to set up EPS so necessary bus voltages were made sure to be available for onward satellite subsystems stacking. Necessary libraries were given for all three parts (PBU, PCU, and PDU). Using the libraries, multiple low-level global and local functions were created. Some of these functions with details are mentioned in Table 3.3 above. After OBC, a similar exercise was carried out for S-band Tx, and Rx subsystems. Table 3.4 shows some low-level application functions for the S-Band Transmitter.

Table 3.3 EPS Low Level Functions.

Subsystems	OEM
EPS_init	EPS initialize function. Initializes EPS (PBU, PDU & PCU) and collects the initial status of the system
EPS_set_outputChannelState	Sets output channel to the desired state
EPS_GetSystemStatus	Collects and populates general system status fields
EPS_GetBatteryStatus	Collects and populates Battery Status Data
EPS_Get_BusVoltagesChannelData	Collects and populates Bus status data(Bus voltages and channel data)

Table 3.4 S Band TX Low Level Functions

Subsystems	OEM
SBandTx_Init	This function initializes the S-band transmitter module
SbandTx_SendDataOverI2CInterface	Sends data over I2C interface
SbandTx_Get_SystemStatus	This function collects and populates general system status fields
SbandTx_Get_HousekeepingData	This function Retrieve housekeeping telemetry from the supervisor
SbandTx_Set_Freq	Set Tx Frequency levels

After EPS and S-band, again a similar exercise was carried out for the X band, as the X band was not from ISISpace no libraries were provided rather library had to be made from scratch. Following Table 3.5 shows some of the application layer functions created for the operation of the X band.

Table 3.5 X Band Application functions

Subsystems	OEM
Xband_SetAllConfig	Setting all configurations of X band transmitter
Xband_StartTransmit	To start transmission of X band
Xband_StopTransmit	To stop transmission of X band
Xband_SetTestPattern	This function sets a Test pattern for functional check
Xband_Get_SpacewireStatus	Tell the status of XTX and Camera link

3.3.3 Payload library

After subsystem libraries, one of the key software requirements was to align the imager with the software design. The software for the camera has been designed in a way that if the imager is replaced, the software can work along after slight adjustments. The basic hierarchy of the camera software is given below.

- Camera-specific libraries
- Camera low-level applications
- Camera management
- Camera modes (Image capture or data readout)

Camera-specific library files have been made using Command and Request identifiers mentioned in the Interface Control Document of the payload. They are further described below in detail.

Command identifiers. These transactions are governed by command IDs and comprise of Session commands, Imaging Commands, and System Commands. Additional commands about encryption, data compression, and the SpaceWire interface were also included for completely defining the functionality of the payload.

Requests identifiers. These request transactions are also governed by Request IDs and make-up functions that deal with the state/ status of the imager, telemetry collection, and diagnostics.

The functions created against each Command type and Request identifiers are mentioned in Figure 3.3 below. Figure 3.4 shows some Low-Level application functions for the camera.

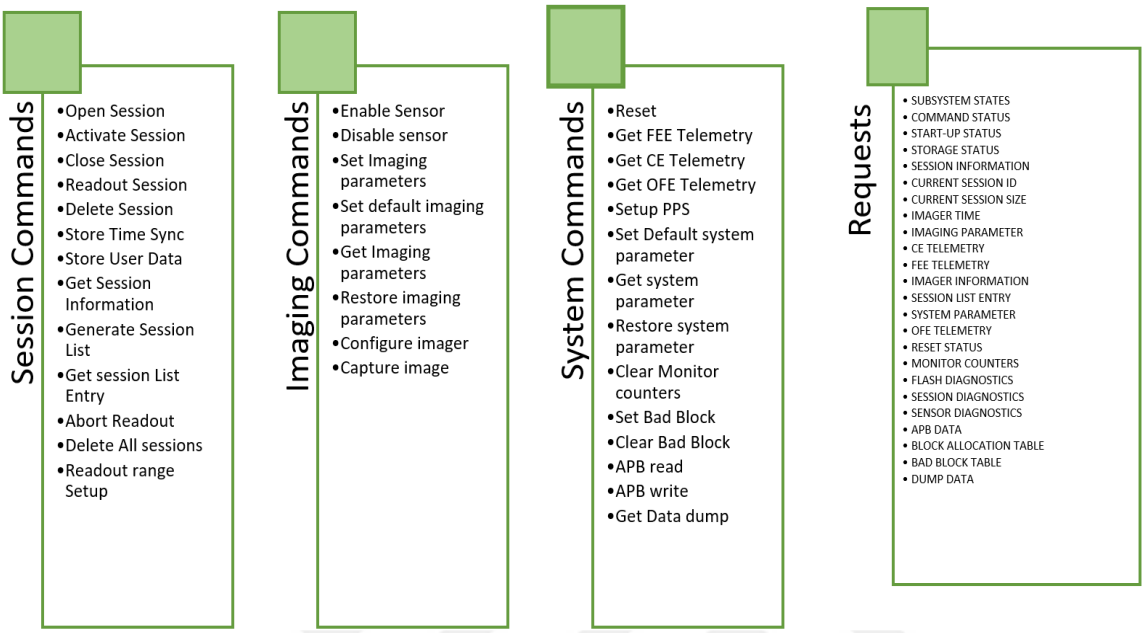

Figure 3.3 Payload Command and Request Functions.

S No	Global Functions	Use of Function
(a)	CamPayload_Init	This function initialize payload and load default parameters using required functions
(b)	CamPayload_GetStatus	Collects latest issued command status and subsystem states of imager
(c)	CamPayload_GetStoredSessionsList	Reads all stored sessions IDs and information
(d)	CamPayload_InitializeCapture	Update imaging parameters, configure imager, initialize session,
(e)	CamPayload_StartCapture	Enables sensors and start capture
(f)	CamPayload_EndCapture	End the capture and disables sensor
(g)	CamPayload_AddImageMetadata	Add user data to active session of the capture
(h)	CamPayload_ReadOut_Setup	Carries out setup for readout by checking parameters, setting sessions and readout filters.

Figure 3.4 Low-Level Camera Applications.

3.4 Flight Software Design

This section will describe in complete detail the Flight software design. Managing complexity was the most important technical topic in software development, and once one understands that then all other technical goals in software are secondary to this.

The software design as a system consists of 2 main parts: Management and Operations. All software blocks are to be implemented to support these two parts. The general software structure can be seen in Figure 3.5.

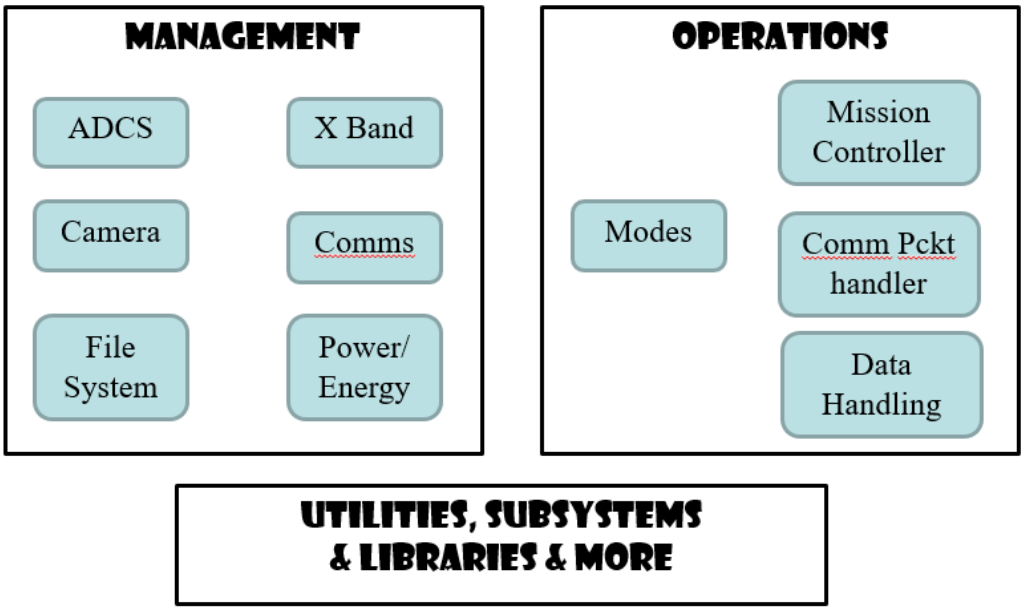

Figure 3.5 Main Software Blocks.

3.4.1 Management block

The management part of the software is responsible for general operations and maintenance of the system. Management blocks keep the system running and operational for operations blocks to use. Management blocks keep the systems running based on the configuration of that system. For example, the power and energy handling require continuous access to the EPS subsystem therefore that block (and related

subsystem(s)) are kept operational at all times while blocks like camera management are only run when access to related systems is required.

3.4.2 Operation block

Operation blocks are upper-level blocks that manages specific tasks of the satellite such as handling commands, controlling satellite operation modes, and executing missions. Execution of operation blocks relies on and depends on the management of the satellite. When there is an operation to run such as image capture with the camera, the operation block commands related management block or blocks to initialize required systems required to run the current operation.

All blocks under management and operations have their own resources and software modules but can share subsystems. One example for this is camera-related blocks where the camera management block use camera modules to initialize the camera and read telemetry and camera mode operation uses camera modules to initialize capture and run the image sensor. To distribute access between blocks more fairly, operating systems objects such as semaphores are used.

3.4.3 Utilities and tools

Blocks depend on several subblocks to run their operation properly. Those blocks are grouped under another block. Tools such as counters, loggers, file system access are used by management and operations blocks to ease the processes running under those blocks.

3.5 Subsystems Manager

As mentioned earlier all subsystems are being controlled by a low-level manager called a Subsystem Manager. These managers are a part of the management block, which handles the turning on and off, saving telemetries and ensuring the availability of the subsystems to be used for a planned mission.

3.5.1 Camera manager

The Camera Manager will keep the Camera turned ON till the finalization of the imaging and during the Readout part. The thread is initialized when the camera is initialized, then the camera state is continuously checked. Following are the states of Camera Manager.

Cam_State_Powereddown: The Camera is powered down. The next state is BOOT state if the camera is successfully powered on**.**

Cam_State_Boot: The camera is allowed to boot and after successful booting it will transit to INIT state.

Cam_State_Init: Initializing the Camera and transitioning to IDLE state.

Cam_State_Idle: The camera is now properly initialized and in Idle mode collecting all the telemetries. It will now transition to either ERROR or POWEROFF state depending upon the next operation.

Cam_State_Turnoff: The Camera Manager structure is reset except for the counters. Passes to the Cam_State_Powereddown state when it is done.

Cam_State_Error: An error has occurred and the Camera Manager is not working properly.

3.5.2 X-band manager

For the software part of the X-Band, an operating system thread, the Xband manager, keeps the transmitter enabled, just like the Camera Manager. The X-Band Manager

thread is initialized when the X-Band is powered up. Then the state is set to idle and waits for the transmit commands handled under the operation manager.

The state machine task for X-Band is given below in Fig 3.6**.**

XBand_State_Powereddown: The X-Band is powered down and no telemetry is available. Passes to XBand_Manager_Init state if the TurnOn flag is set.

XBand_State_Rollback: The board is powered but the initialization hasn't started.

XBand_State_Init: Initializing the X-Band and collecting all the data to store in the XBand_T structure. Starts to read and log the necessary telemetry data.

XBand_State_Idle: The X-Band is ready to transmit data. Configuration changes are made and transmits data if the operation manager sends a transmit command. X-Band stays in this state until the Turn Off flag is set and then passes into the XBand_State_Turnoff state.

XBand_State_Turnoff: The X-Band Manager structure is reset except the counters. XTX_EN pin is disabled. Passes to the XBand_State_Powereddown state when it is done.

XBand_State_Error: An error has occurred and the X-Band Manager is not working properly.

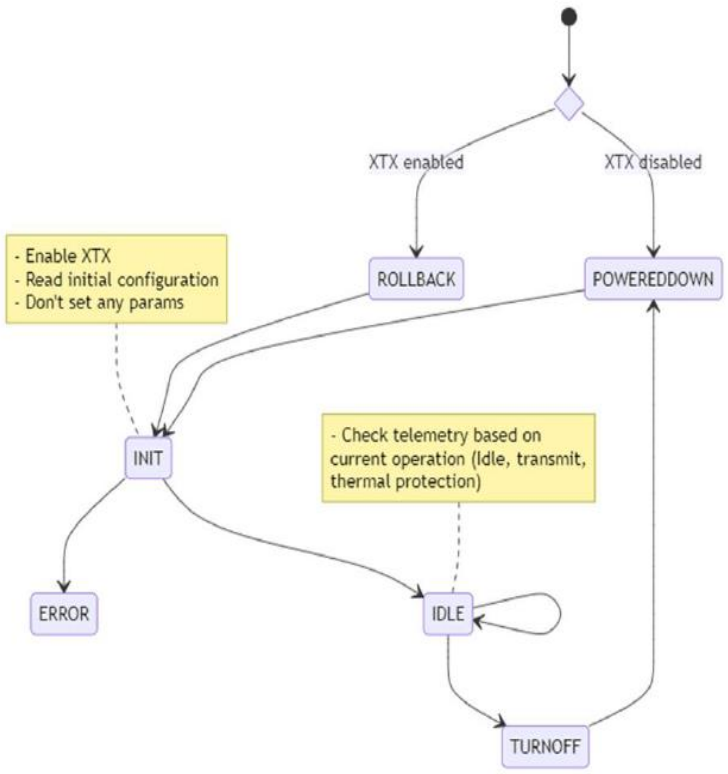

Figure 3.6 X-Band Manager.

3.5.3 Comm manager

The software communication interactions in the S-band communication system involve various blocks, protocols, and interfaces to facilitate efficient data packet handling and communication between the satellite and the ground station. Below is a detailed summary of each block and its role in the overall communication scenario.

A basic flowchart explaining the blocks is shown in Fig 3.7 below.

Communication manager. This independent block, implemented as OS tasks, is responsible for managing and handling communication-related operations. It interacts with a single Command Handler block to process incoming commands and manage communication tasks effectively.

Command handler. The Command Handler block receives commands from the Communication Manager and initiates the packet transmission process. It interfaces with the Packet Porter block to transfer the command data for further processing.

Comm packet porter. The Packet Porter block acts as interface between the Command Handler and other key blocks. It routes the packets to different blocks based on their destination, such as the Packet Parser, Packet Builder, S-band Transmitter, S-band Receiver, or Debug Port Handler. The command handler sends the built packet to the Packet Porter module. The Packet Porter then sends the packet to the appropriate communication interface (I2C, LVDS, or UART) for transmission. The Packet Porter module also receives incoming packets from these communication interfaces to be forwarded to the command handler for further processing. Packet Porter also will be used to change or read the configuration of S-band subsystems.

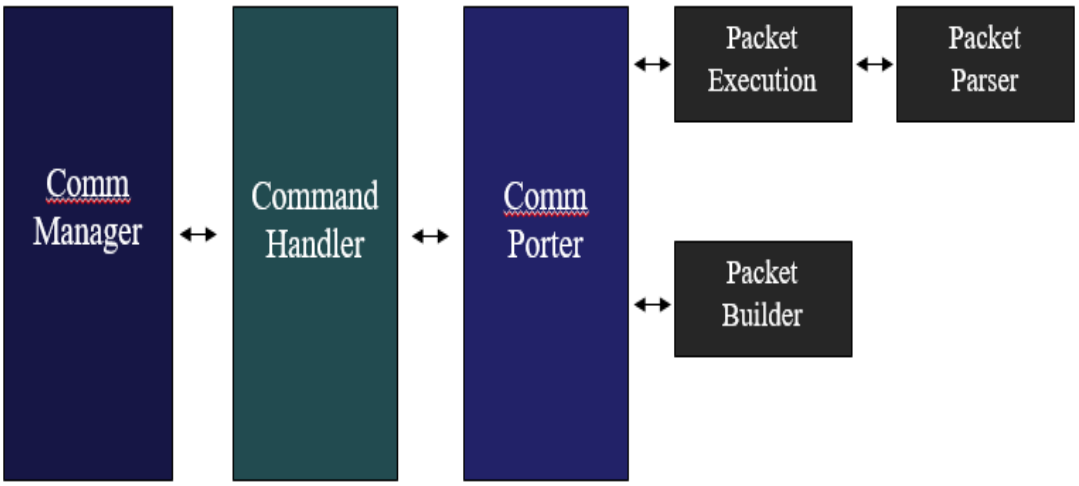

Figure 3.7 Communication Manager.

Packet parser. Packet Parser module by the Command handler for parsing relevant information such as the command type, destination subsystem or sub-module, and any associated parameters or data. The AX.25 protocol includes an error-checking mechanism such as Frame checksum sequence (FCS) for corrupted packets. The packet parser will compare the received packet's FCS value with the calculated FCS and if they match, the packet is considered to have been received successfully and is handled by the Command handler to carry out the intended operation for the relevant

subsystem/module. If there is a mismatch in the FCS value, the packet will be discarded and a retransmission request for the packet can be generated.

Packet builder. The Packet Builder block is responsible for constructing packets for transmission. It receives data from the Packet List block and AX.25 Frame Controller block to build the appropriate packets for transmission. The Packet Builder module builds packets for both the data part and the ax.25 frame part that will be transmitted to the ground station. The output of the builder function is a string representing an AX.25 frame. Next, the command handler takes the packets and forwards it to packet porter to be transmitted either via UART or S-band.

AX.25 frame controller. The AX.25 Frame Controller block manages the AX.25 protocol operations, including frame formatting, synchronization, and checksum error control. It interacts with the Packet Parser and Packet Builder blocks to process incoming frames and prepare frames for transmission. The Frame Controller also interfaces with the Checksum Control block to ensure data integrity.

S-band transmitter. The S-band Transmitter block handles the transmission of S-band communication signals. It interfaces with the LVDS block and I2C block to transmit data over the S-band frequency. The transmitter block receives data from the Packet Builder and sends it for transmission.

S-band receiver. The S-band Receiver block is responsible for receiving S-band communication signals. It interfaces with the I2C block to receive data. The received data is then processed further for decoding and handling by other blocks.

3.5.4 Power manager

Power management is a very critical aspect of not only a satellite but all electronic systems. Managing harvested and consumed energy plays a critical role in managing satellite operations and keep the battery health at maximum. Several parameters play critical roles in power management such as battery voltage and instantaneous power drawn from the battery, overall battery health, and power generation capacity. Based on these parameters when there is not enough energy, the execution of a mission can

be postponed or when energy levels become low, the satellite can be put into lower energy-consuming modes to reduce power drawn from the batteries.

The telemetry line going to the EPS enables OBC to read energy-related data and take required actions based on the state of the batteries. Several battery voltage levels are implemented in the system for these actions to be taken. Even though it is not possible to directly extract the state of the battery from the battery voltage, it is possible to decide mode changes based on the battery voltage.

A separate task runs parallel in OBC to handle power management operations. This task continuously communicates with the EPS subsystem to collect battery and power generation units' statuses and acts based on the telemetry collected from these systems.

3.6 Mode Manager and Mission Controller

The satellite has currently 07 operating modes, in which it will be switching in its lifetime. For this a software module has been designed that enables easy transition between these operating modes; it is called Mode Manager. The two most important modes of the satellite are Camera mode and Readout Data mode which are dedicated to the execution of the main missions of the satellite, thus, when the time to execute the mission is reached, the mode needs to be transitioned to the required state which is handled by the Mode manager.

With the mode manager, there is another software module called Mode Controller which handles mission-related controls. The mode controller holds several parameters to indicate when the mission starts, what is the total amount of time that the mission needs to prepare for the mission, and what is the maximum amount of time that the mission can late start before mission leaves the window to complete shown in Fig 3.8. All these parameters are loaded to the mission controller via ground station commands during mission scheduling. These two modules; Mode manager and Mode controller work in close sync with each other to execute the missions successfully.

In the mission controller module, only mission scheduling parameters are kept. Whereas the Mission-specific configurations are kept in a config file under the file

system. The name of that config file is also provided to the mission controller during mission scheduling. and the mission controller feeds this file to the mode manager as well.

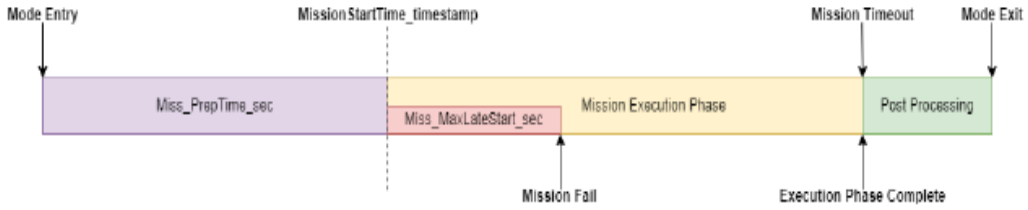

Figure 3.8 Mission Phase Diagram.

3.7 Operating Modes

As mentioned earlier, several operation modes will be implemented on the application software for proper management of all satellite subsystems. The mode of operation will be decided based on several states, statuses, and requests (either from the ground station or subsystem). The satellite is planned to be operated on the following modes shown in Fig 3.9.

Apart from the operating modes, there is a Start-up phase. It is to be executed at every OBC reset. While this mode is only entered when OBC initialization is required, it handles all initial and critical operations of the satellite. This is the default mode that is entered at the beginning of the code execution and runs until “Mode Manager Task” is started and next mode is selected by mode manager. In this mode, all OBC hardware peripherals are initialized such as communications (I2C, SPI), timers, debug tools and operating systems configured and started. After the operating system starts main task starts which starts the mode manager task. After the mode manager task starts, the next mode is entered and the startup mode is exited and not to be entered again before OBC reset.

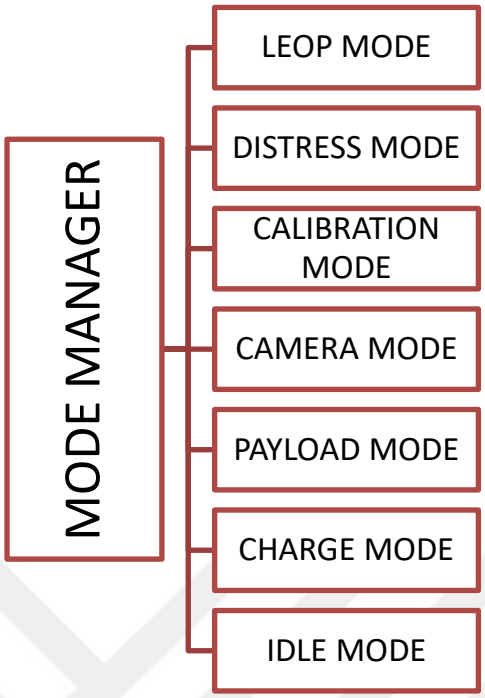

Figure 3.9 Operating Modes.

3.7.1 LEOP mode

LEOP (Launch and early operations) mode is to be run only once during the lifetime of the satellite when in orbit. This is the first operation mode the satellite will use after launch. After its first boot, it will wait for 90 minutes and start transmitting a periodic beacon. If the battery is depleted or the satellite is reset due to any other reason during this phase, enough orbits will be given to charge the batteries. Afterward, the antennae are deployed and the ADCS begins calibration and operation to detumble the satellite. This operation is expected to take a couple of days, and the ADCS can be commanded manually from the ground station to ensure everything goes as intended. The LEOP mode ends when a command to end it, is received from the ground, or when a timeout of 3 weeks is reached.

3.7.2 Camera mode

Camera mode is the mode where the camera payload is operated to capture images. For this mode to operate properly, the configuration of the mode, parameters such as image capture settings, operation length, etc. should be provided when scheduling this mission. These are sent by Ground station otherwise default parameters are used to complete the mission.

The state machine deals with the Operation Mode of the camera. Several operation modes will be implemented for the proper working of the camera. The transitions between these modes are decided based on the previous state, system status, and request. Some of the conditioning used for transitioning states inside the camera operating mode are shown in Figure 3.10 below.

Criteria	Next State
If the camera is initializing	Wait on same state
If camera is initialized	Go to Init capture state
If capturing is in progress	Go to Capture in progress
If readout is in progress	Go to error state

Figure 3.10 Camera Status Checks.

Brief details of the Transition state are mentioned below.

Check Cam State. This state checks the camera status and decides the future mode selection.

Initialize Capture. This state requires the camera to be initialized and further request ADCS orientation. It further goes to the IDLE state if there is no error otherwise it will go to the ERROR state.

Idle State. The camera will remain in an IDLE state when it is waiting for the capture time to start, it will get the time stamps from Time Keeping files and Mission Controller files and will only move to START CAPTURE State when the Current Time>Capture Start Time.

Start Capture state. From the IDLE state, the camera will jump into this state. The sensor and Mission timeout Counter get enabled in this state. When the capture begins without any error the camera goes to the CAPTUREINPROGRESS state.

Capture in progress state. Till the capturing is in progress, the camera will remain in this state and will finally go to the END CAPTURE state.

End capture state. The sensor gets disabled in this state and if the capture ends without errors, then the camera goes to the FINALIZE CAPTURE State otherwise to the ERROR state.

Finalize capture state. This state is only reached when capture ends without error and allows saving image data and auxiliary data. Finally, the camera will go to the END OPERATION state.

End Operation state. This state means the end of all operations and changes the mission status in the Mission Controller.

Error state. This state is achieved in the case of error in any of the above states.

3.7.3 Payload Mode/Data Readout

This mode will be used to download payload data from the camera through an X-band transmitter. This mode will be the highest power and resource-consuming mode since every subsystem in the satellite will be running. Pointing capabilities of the satellite and joint operation of the satellite and ground station are very important during this mode. Payload data mode will be implemented as a mission in which case all the configuration of the mode should be send to the satellite beforehand.

This mode will also be operated in several states like the camera mode: ModeInit, idle, transmit, thermal protection, and inactivity. Xband and camera managers will be

commanded to prepare the systems for operation in this state. Following are some brief details of the states of Payload Data mode.

Mode intialization. This state checks the initialization state of the X-Band Manger and Camera Manger and if okay allows us to jump further to the IDLE state.

Idle state. In this state, three main checks are done, first, it is checked if X Band parameters are set or not otherwise the UPDATE PARAMETERS STATE is jumped into. Secondly, the mission is checked and if successfully updated then the TRANSMIT STATE is jumped into. Finally, if none of the above conditions are met then the mode changes to EXIT MODE.

Transmit state. The X Band takes data from the Camera and Read Out is done and data is transmitted to the Ground Station. After successful transmission, the X-band state machine jumps back to the IDLE state. After the transmission is done, the state machine also checks the temperature levels of the X band, and if the temperature is above the threshold level, then the THERMAL PROTECTION mode is entered.

Update parameters. This state is jumped from the IDLE state, in which parameters like frequency, symbol rate, and power levels are set.
Thermal protection state. The X-band state machine remains in this mode until the temperature levels go below the threshold values.

Mode exit. After the transmission is completed, then this mode is entered into and the X-band is successfully disabled.

3.7.4 Distress mode

Distress mode will be used to handle and clear fatal errors and unexpected conditions/situations. This operating mode will be entered into when the battery voltages go below a certain threshold value. Subsequently, to get out of this mode, the battery needs to be charged hence the ADCS will point the solar panels toward the sun's direction to harvest the maximum possible electrical energy to get the satellite out of distress mode. Apart from EPS voltage values, other entry and exit conditions

of this mode are to be decided after all subsystem implementations are completed especially the ADCS subsystem.

3.7.5 Calibration mode

Calibration mode will be used to recalibrate the satellite during normal operation when the need arises. The conditions for entry/exit of this mode are yet to be determined, as this mode is specifically related to the ADCS subsystem whose implementation is still underway.

3.7.6 Charge mode

Charge mode is to be used to point the solar panels to the sun directly and charge the battery when no other operation is required to run on the satellite. This mode is rest very similar to IDLE mode.

3.7.7 Idle mode

This is the default mode that the satellite will use. The ADCS is used to achieve nadir pointing. Beacon is operated and telemetry data is transmitted at regular intervals. Three periodic checks are performed in this mode. Firstly, the NVM data is read to check to see if a mission is scheduled from the ground station. If there's one in 15 minutes, the camera or Data Readout task gets operational and the idle task is suspended. Secondly, the battery voltages are read to check if they are under a configurable threshold. If that is not the case, then the satellite will switch to the Safe Mode. Finally, the temperatures of the imager will be monitored to keep them within the range stated by OEM for optimum imagery, otherwise, heaters will be operated till the desired temperatures are achieved.

4. TESTING AND VERIFICATION

Testing of every CubeSat involves a comprehensive approach, including both unit testing and integrated testing of all subsystems to ensure reliable and efficient operation [16]. A basic list of the tests to be performed/intended for this CubeSat mission is discussed below in Table 4.1.

Table 4.1 Software Testing List.

Test	Output
Demand telemetry packets from all subsystems for a determined time interval. Check the time tag of the received data.	The received telemetry packet should belong to the predetermined time interval
Safe mode transition from other modes based on voltage level.	Operation mode should be changed when battery voltage decreases below 13.8V, and returns to the previous mode when it increases to 14.2V.
Hard reset satellite sometime after start-up.	The satellite should detect for how long it was powered down, and deduce it from the amount of time it should wait.
Schedule multiple camera and X-band Readout missions from the GS.	Beacon should operate in configured time intervals and should transmit correct data.
Receive beacon message from ground station	The missions should start at the desired time and use the correct parameters.

4.1 Test Setup and Debugging Environment

The First step of testing the developed code was to generate a comprehensive test environment in Eclipse using the debugging tools available in the OBC hardware shown in Fig 4.1. The developed environment setup constitutes the debugger attached to the OBC to be programmed at any stage. To support this functionality, an umbilical connector was designed and integrated into the top panel of the satellite. This connector serves a dual purpose:

- To provide external power to the satellite
- To connect the debugger.

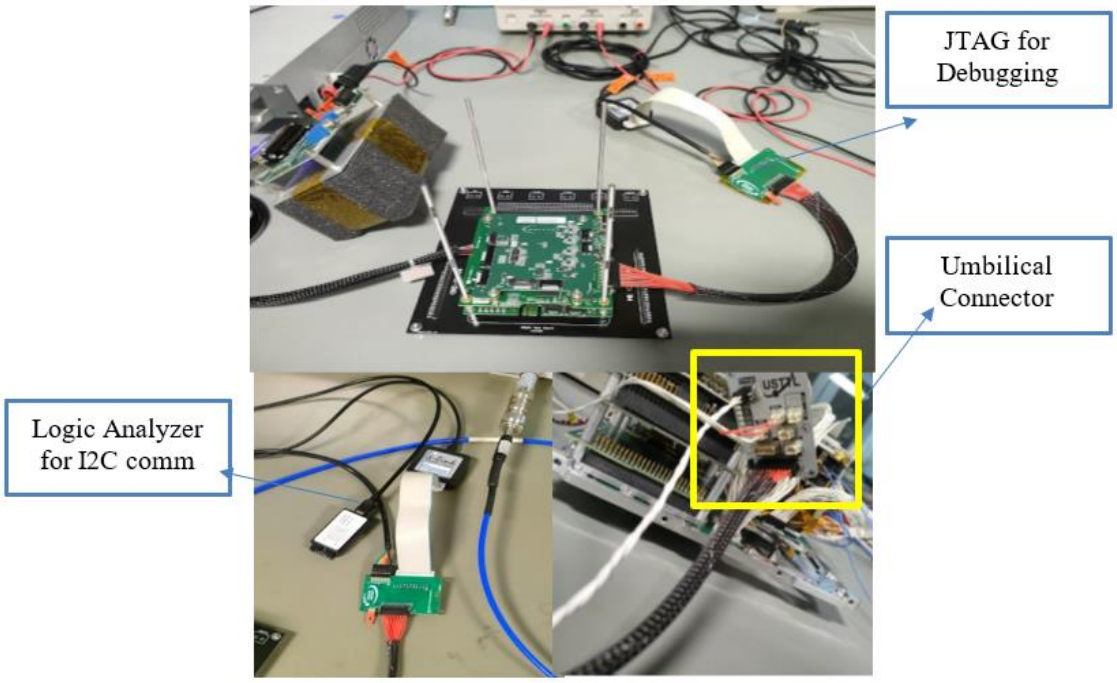

Figure 4.1 Test Setup.

This debugging mode enabled us to monitor the program flow and address any bugs or errors. When the DEBUG macro is defined in the software, debug statements are

transmitted through this UART line and displayed on the PC, allowing for real-time debugging and diagnostics**.**

4.2 Unit Testing of Subsystems

During the unit testing phase, individual components, and subsystems, such as the power system, communication modules, and attitude determination and control system (ADCS), are rigorously evaluated in isolation. Each component is subjected to various test scenarios to validate its functionality, performance, and resilience to potential faults. Some of the unit testing is described below.

4.2.1 EPS

The testing of the EPS (Electrical Power System) subsystem of a satellite involves a meticulous process where the Discharge Unit (DU), Charging Unit (CU), and Power Bus Unit (PBU) are stacked and interconnected for comprehensive evaluation. As this system is the foremost unit upon which all other subsystems are dependent for power supply hence its detailed testing is deemed crucial for simulating the actual operational setup of the satellite.

During testing, the system is rigorously assessed for housekeeping values shown in Fig 4.2, which include monitoring parameters such as voltage, current, and temperature to ensure the health and functionality of the power system. Additionally, bus voltages are closely monitored to verify that they are within acceptable ranges, ensuring that all subsystems receive stable power. The battery pack connections are also tested to confirm robust and reliable connectivity, which is essential for the continuous power supply during the satellite’s mission. This thorough testing ensures

that the EPS can manage power distribution effectively and maintain the satellite's operations under all expected conditions.

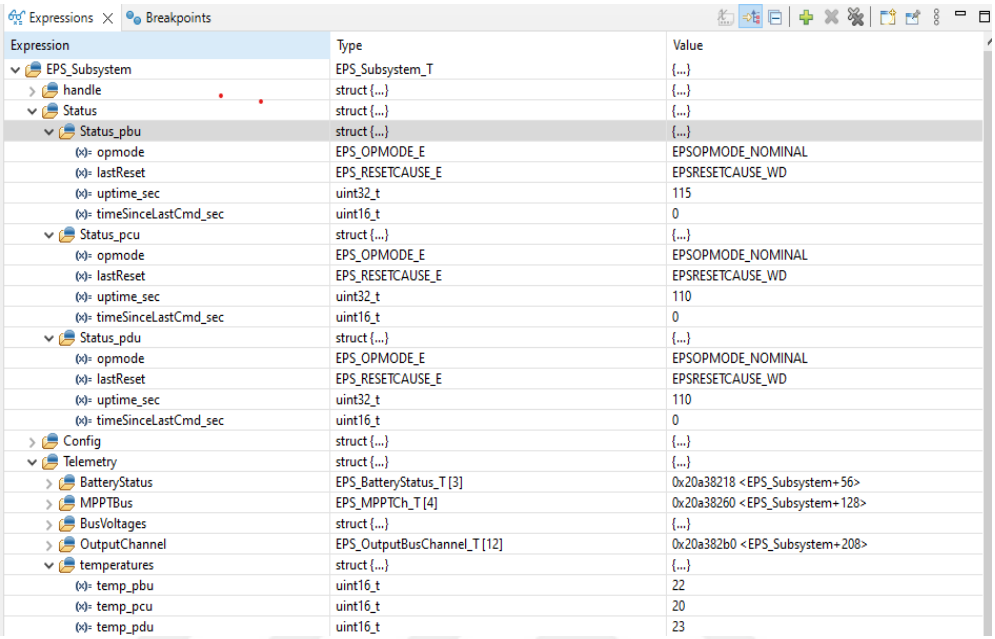

Figure 4.2 EPS HouseKeeping Values.

4.2.2 Imager

The Imager came with ground testing equipment called EGSE with Python scripts for testing the camera and its functions shown in Fig 4.3. Multiple line scans were taken with different configurations. Additionally, features like compression, encryption, and PPS delays were tested using EGSE to validate the performance of imager. The test setup is shown in Fig 4.3. These line scans/snapshots helped determine different functionalities and features of the imager and helped the image processing team of

Pakistan draft an effective calibration procedure for the imagery that will be received after launch.

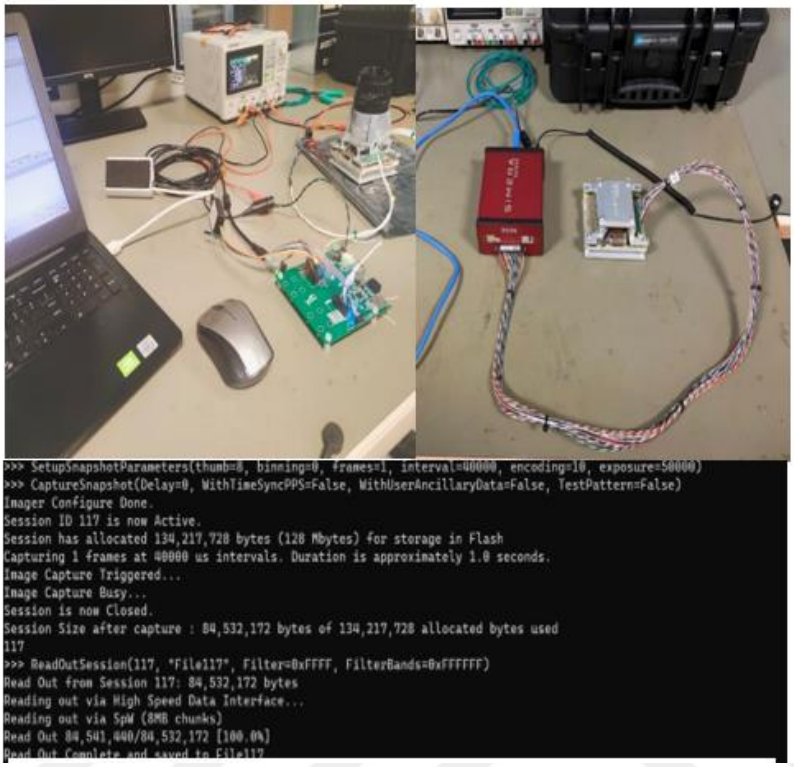

Figure 4.3 Camera Test Setup.

4.2.3 Communication subsystems

After the successful integration of EPS and OBC and flashing the code on it, the next step was to stack the S-band Comm set on top of it using PC-104 connections and check for I2C communication. The same was eventually done for X-band as well. For this logic analyzer was put into use. The housekeeping values were thoroughly checked and it was made sure that the equipment was working fine. As evident from Fig 4.4 and 4.5 below, the values were mostly related to Status, telemetry, and Configuration setup, which were then matched with the received documentation.

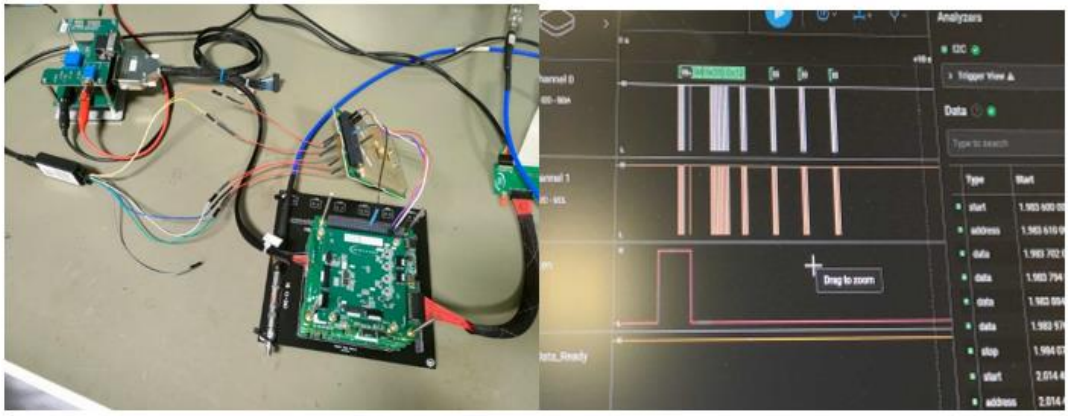

Figure 4.4 XTX I2C Communication.

Expressions | Breakpoints

Expression	Type	Value
SBANDTX_Subsystem	SBANDTX_Subsystem_T	{...}
handle	void *	0x20a384e0 <SBANDTX_Handle>
Flags	struct {...}	{...}
Status	struct {...}	{...}
SupervisorCurrMode	uint8_t	1 '\001'
SupervisorVersion	uint16_t	0
state	SBANDTX_STATE_E	SBANDTX_STATE_STANDBY
status	SBANDTX_TRANSMIT_STATUS_E	SBANDTX_TRANSMIT_OFF
LastReset	SBANDTX_RESETCAUSE_E	SBANDTX_RESETCAUSE_POR
MSSPowerTime_sec	uint32_t	19
MSSVersion	uint32_t	16777216
MSSUpTime_sec	uint32_t	5
SupervisorUpTime_sec	uint32_t	19
mode	SBANDTX_OUTPUT_MODE_E	SBANDTX_OUTPUT_MODE_EXTERNAL
scid	uint16_t	0
EngineStatus	uint32_t	310809665
FabricVersion	uint32_t	0
CmdStatus	SBANDTX_COMMAND_STATUS_CODE_E	SBANDTX_CODE_RESULT_OK
DataFrame	struct {...}	{...}
Configuration	struct {...}	{...}
Frequency_Hz	uint32_t	2245000
DataRate_bps	uint32_t	0
TemperatureThreshold_C	uint8_t	0 '\0'
TemperatureProtection_EN	uint8_t	0 '\0'
AttenuationCtrl	SBANDTX_ATTENUATION_E	SBANDTX_ATTENUATION_0_dB
BitRateCtrl	SBANDTX_BITRATE_E	SBANDTX_BITRATE_SIXTEENTH
Modulation	SBANDTX_MODULATION_E	SBANDTX_MODSCHEME_OQPSK
LUT	SBANDTX_LUT_E	SBANDTX_LUT_0_35
Telemetry	struct {...}	{...}
VoltageSerialDAC_mV	uint16_t	2700
ModulatorOutPower_W	uint8_t	1 '\001'
ReflPower_mW	uint16_t	42654
ForwPower_mW	uint16_t	54864
Volt3v3_mV	uint16_t	3311
Volt3v3sw_mV	uint16_t	3296
Volt5v_mV	uint16_t	0
VoltBat_mV	uint16_t	16390
Cur3v3_mA	uint16_t	26373
Cur3v3sw_mA	uint16_t	375
Cur5v_mA	uint16_t	80
CurBat_mA	uint16_t	85
VoltCtl_mV	uint16_t	0
TempDriver_C	uint16_t	22
TempPD_C	uint16_t	0
TempPA_C	uint16_t	22

Figure 4.5 S band Housekeeping Values.

4.2.4 ADCS

The ADCS OBC was connected to the stack containing CubeSat OBC and EPS and was powered on. Subsequently, all the purchased sensors and actuators were connected one after another and tested to check the functionality of the purchased ADCS software along with the hardware. The ADCS sensors were extremely sensitive to the external environment hence necessary precautionary measures were taken to check especially the Fine Sun Sensor, Start Tracker, and Reaction wheels. The test setup is shown in Fig 4.6.

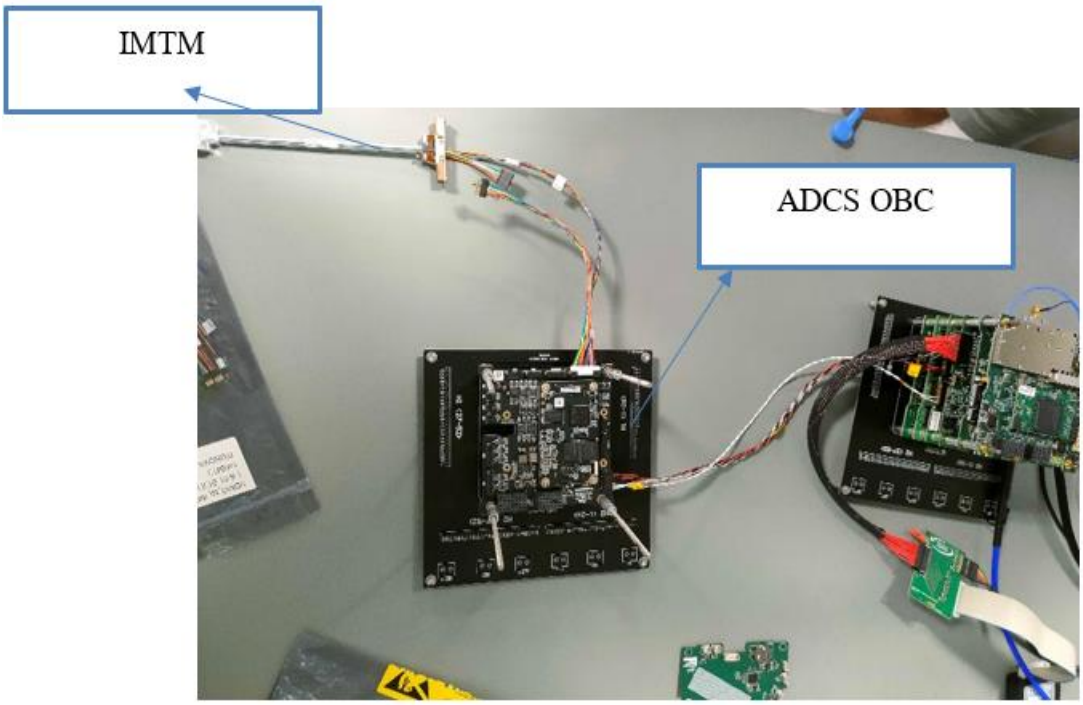

Figure 4.6 ADCS Testing with IMTM.

4.3 Integrated testing

After verifying all received units, we jumped toward integrated testing. By then all subsystems managers except ADCS had been completed and were ready to be tested. Moreover, the three main operating modes i.e. Camera, Readout, and Distress were also partially done. That meant that after checking individual low-level applications and system managers, we may also be able to use Mode Manager to see if its switching

effectively between operating modes and appropriately instructing the relevant subsystem managers to turn on/off as per requirement.

4.3.1 EPS, OBC, and S-band transceiver

In this testing, S-band TXS and RXS were stacked on top of complete EPS and OBC shown in Fig 4.7. Furthermore, RF connections were done using ISISpace splitter as both the RXS and TXs antenna is to be connected directly with the Transmitter. For visualizing the transmitted signal, the spectrum analyzer was connected and for visualizing the received signal Teledyne Modem (QubeFlex) was put into use. S-band was found to be successfully transmitting a test pattern with the desired modulation as it was visible on the spectrum diagram of QubeFlex.

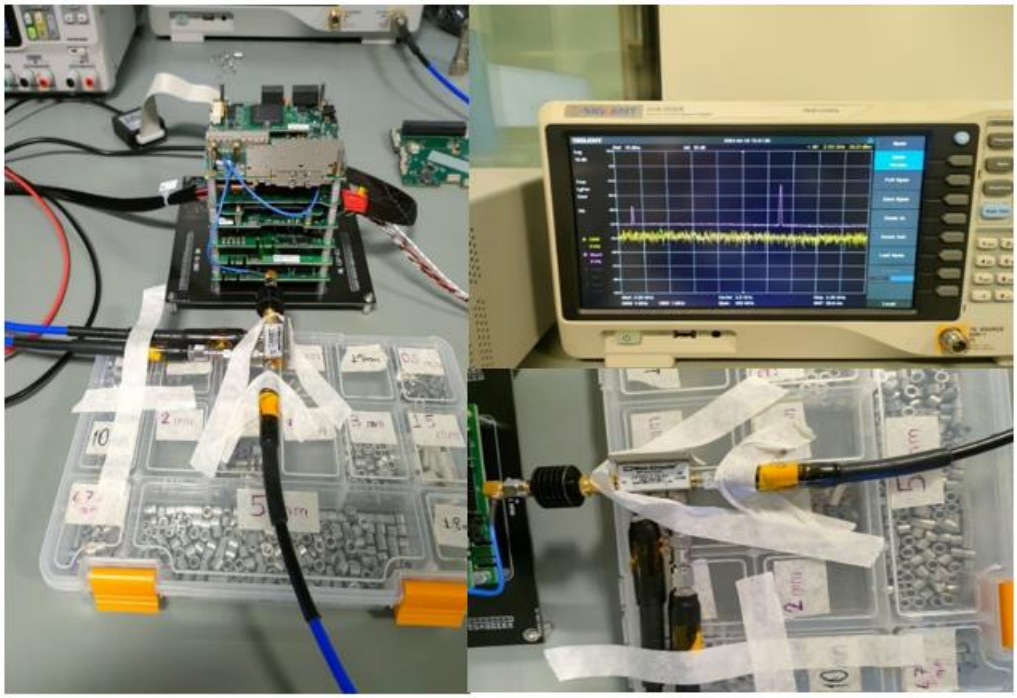

Figure 4.7 S-Band Integrated Testing.

4.3.2 EPS, OBC, and X band transmitter

In this testing, the X band Transmitter (XTX) was stacked on top of complete EPS and OBC. Furthermore, RF connections were made and the X-band Transmitter was directly connected to Teledyne QFlex 400. For visualizing the received signal. It was found that XTX was successfully transmitting a test pattern with desired modulation as it was visible on the spectrum diagram and constellation diagram in Qflex 400.

As the X band Transmitter is transmitting at a high power level, a heat sink was used during testing to keep the temperatures within operational range. A downconverted is used alongside the Qflex -400 modem as the X band frequency range visibility was not available in either the spectrum analyzer or Qflex-400 mode. In Fig 4.9 below, we can see the constellation diagrams on Qflex-400 for changing modulations. The

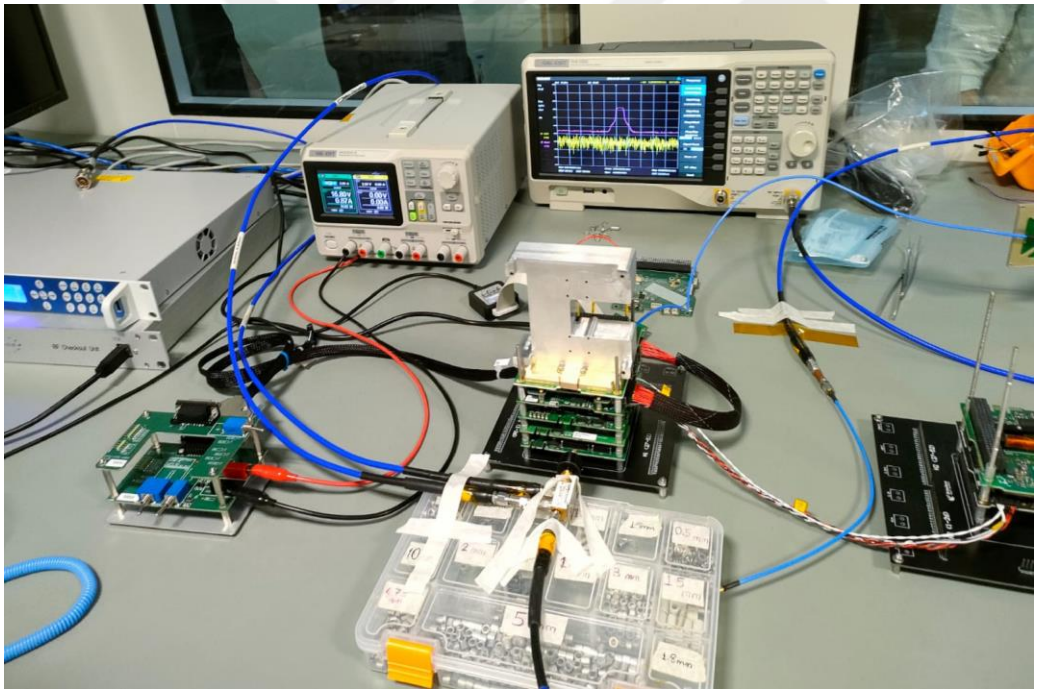

Figure 4.8 X-Band Integrated Testing.

frequency and other parameters have been removed from the image due to confidentiality reasons.

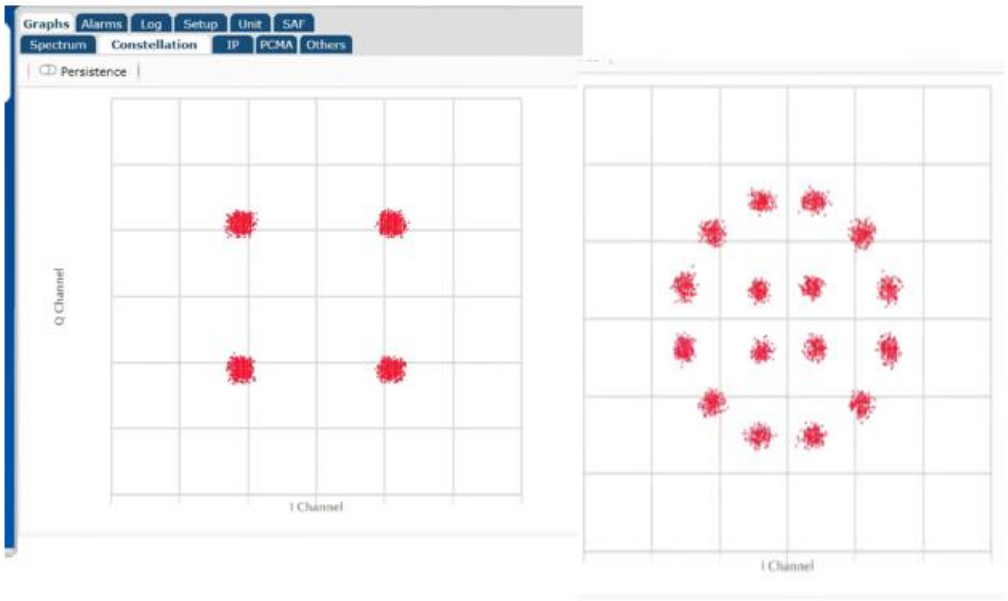

Figure 4.9 X-Band Modulation Diagram.

4.3.3 EPS, OBC and camera (camera operating mode)

As the camera had been tested earlier, the key operation in this part was to schedule an imagery mission, which will be coming in the form of a telecommand packet from the Ground Station(GS) with necessary parameters like Mission Start Time, Mission Preparation Time, and imager configurations. As the mission packet was generated by the GS, it was sent via the uplink box to the satellite. The Comm Manager receives the packet, parses the information, and builds the response. In this case, the response is

adding the desired mission with configurations to the mission controller which will perform the following steps

- Checks the type of Misson, it should be Capture
- Checks the mission parameters in case they are invalid
- Adds the Mission to Mission List and also saves it in the Non-Volatile memory

The next step is handled by Mode Manager, as the mode manager will be in any state e.g. IDLE. It keeps on checking the entry conditions set for each mode. When the Entry conditions for OPMODE_Camera are set because a mission is scheduled for imaging, the mode manager will be shifted to Camera Mode and the Camera mode will initialize its thread where it will ask camera states and use necessary camera applications functions to perform the desired mission.

4.3.4 EPS, OBC, and X-band (data readout operating mode)

After successfully performing the camera mission. The next step was to check the data readout mission. This operating mode as defined earlier is Data readout mode and makes use of the camera and X-Band both. The data being emitted by the camera does not go to OBC, rather goes directly to X-Band via a SpaceWire link and is transmitted via X-Band to Ground station at a high data rate. Like the camera mission, these mission parameters along with readout parameters for the camera and X-Band are sent

by GS by the user. When received by the satellite, the. The Comm Manager parses the information and builds the response.

Again like the camera mode, the response in this case is also adding the desired mission with configurations to the mission controller which will perform the following steps

- Checks the type of Misson, it should be Readout
- Checks the mission parameters in case they are invalid
- Adds the Mission to Mission List and also saves it in the Non-Volatile memory

The next step is similarly handled by Mode Manager, as the mode manager will be in any state e.g. IDLE or camera. It will make sure that the previous state is over and simultaneously keeps on checking the entry conditions set for this readout mode. When the Entry conditions are met, the specific thread is initialized which makes use of the X band manager and camera manager to transmit imaging data with the necessary imager and user ancillary dat via X band at the desired frequency, power level, symbol rate, and modulation scheme. After successful transmission both the X band and Camera are turned off to save on-board power and the satellite may then shift to Idle or charging state depending upon the conditions.

4.4 EQM Assembly

After successful Unit Testing and Integrated testing with transitioning in between different operating modes, the structural team put together all 03 stacks i.e Communication, ADCS, and EPS together on top of the Interface Board (IF). This interface board allowed bus voltages and UART connections to be done directly to the

Umbilical connector instead of single wires going from one subsystem to another. The EQM assembly has been completed and the testing is underway.

4.4.1 End-to-End testing with GS

Although in our integrated testing, we were able to verify the transmitter functionality accurately, one of the most important tests was to send a Telecommand from the Ground Station and receive it on the satellite end and subsequently, the reply should send telemetry back to the Ground Station. The Ground Station Software is also under development jointly by the AU and ITU teams. An ISISpace RF Uplink Box has been put into use for sending uplink Requests to satellites. Although Lab testing had been done, we opted to test it with the Ground Station Antenna itself.

The EQM model (OBC, EPS, XTX, and S-band Transceiver with antennae) was taken under a secure environment and helical casing to the antenna site for testing shown in Fig 4.10. The RF uplink box was connected to the Ground Station antenna and commands were transmitted through the antenna, those were picked by our EQM antenna and subsequently, the response packets were related and transmitted, which

were again received successfully by the Ground Station antenna and were visible on RF modem in the form of spectrum and modulation diagram.

This testing was quite fruitful as it helped us assess our current link budget conditions, our Ground Station antenna validity, and the validity of our developed software for communication packet exchange.

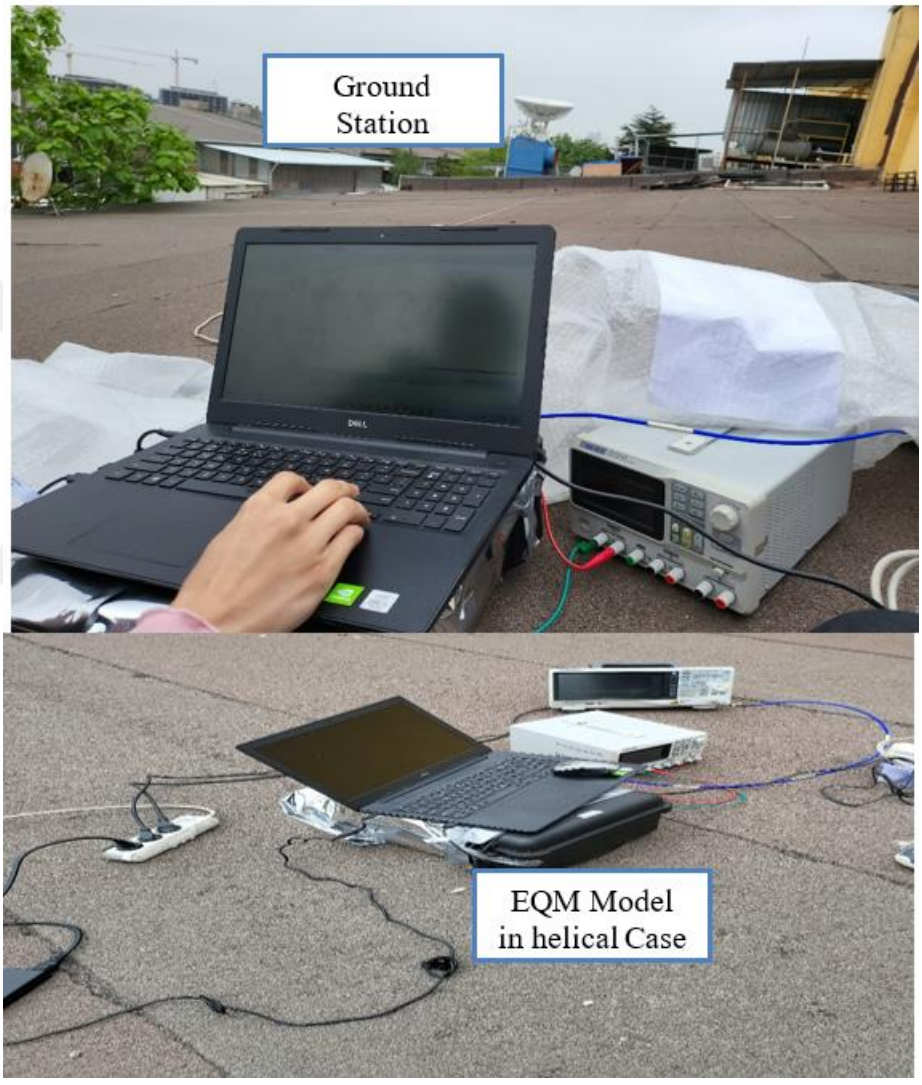

Figure 4.10 Test setup with GS antenna.

4.5 Result and Ongoing Tests

In the above discussion, some parts of testing have been shared, where actually in each testing multiple parts of code get tested and several issues/faults were removed on the way. A detailed list of tests done and pending are shown below in Table 4.2.

Table 4.2 Results and Ongoing Tests

Test	Status	Remarks
Unit Testing of each subsystem	Completed successfully	-
OBC Functionality and additional peripherals such as RTC, Counters, Watchdog	Completed successfully	
S-band communication (both uplink and downlink)-packet exchange and periodic beacon	Completed successfully	The number of packets and packet contents are still under revision
Camera Mission Scheduling and capturing images	Completed successfully	
Readout Mission scheduling and data output through X Band	Completed successfully	
Receiving telemetry from each subsystem and saving using the File system	Completed successfully	
Receiving telemetry from each subsystem and saving using the File system	Completed successfully	
Distress Mode-transitioning of the voltage levels is low and shifting to charge mode	Completed successfully	More elaborate conditions will be added
ADCS subsystem, related operating modes, and operations concerning space environment and scenario	Partially completed	Work in progress
Bootloader for in-space calibration and updates	Partially completed	

5. CONCLUSIONS AND RECOMMENDATIONS

This thesis described the architecture, development, and testing of the onboard flight software of a 16U CubeSat. The whole development process started about 2 years back, and more than 31,000 lines of code have been written on top of the OBC drivers, FreeRTOS, and file system source codes. A lot of subsystem codes, operating modes codes, and utility software have been developed. Testing of the mentioned modules is still in process. The satellite will be launched in the fourth quarter of 2024 and the reliability of the software will be proven then.

5.1 Practical Application of This Study

This study has several practical applications and benefits as it is an Earth observation satellite dedicated to serving the United Nations Satellite Development Goals of 2030. Similar satellite missions are foreseeable in the future hence this study will help in the development of software for such missions. The payload may vary depending on the mission objective such as real-time environmental monitoring, Agricultural applications, Urban planning and development, and Disaster Management. However, the overall architecture and associated utilities would be quite similar to this mission architecture.

Apart from serving the mission, the development of satellite software contributed to the educational training of both Pakistan and Turkiye team members. This design and

development was a learning tool that promoted STEM education and fostered a next generation of space scientists and engineers.

References

[1] C. P. S. University, «Cubesat Design Specification Rev 14.1,» 02 2022. [Çevrimiçi]. Available: https://www.cubesat.org/cubesatinfo.

[2] L. T. LUMBWE, «Development of an onboard computer (OBC) for a CubeSat,» Faculty of Engineering, Cape Peninsula University of Technology, Cape Town, 2013.

[3] D. Bolles, «Hubble Space Telescope,» [Çevrimiçi]. Available: https://science.nasa.gov/mission/hubble/.

[4] D. Bolles, «Mars Reconnaissance Orbiter,» [Çevrimiçi]. Available: https://science.nasa.gov/mission/mars-reconnaissance-orbiter/.

[5] Sommervile, Software Engineering, International Computer Science Series, Addison Wesley, 2004.

[6] Koopman, Better Embedded System Software, Drumnadrochit Education, 2010.

[7] A. Burak, «Your 2024 Guide to Writing a Software Requirements Specification – SRS Document,» Relevant Software, 2024. [Çevrimiçi]. Available: https://relevant.software/blog/software-requirements-specification-srs-document/. [Erişildi: May 2024].

[8] A. W. Burns, Real-Time Systems and Programming Languages, Pearson Education Limited, 2009.

[9] J. R. L. S. S. Stephen A., «Verification Validation and Certification Challenges for Adaptive Flight-Critical Control System Software,» p. 9, August 2004.

[10] B. TECH, «Smallsats by the Numbers 2022,» 2022. [Çevrimiçi]. Available: https://brycetech.com/reports. [Erişildi: 14 Feb 2024].

[11] S. Sense, «MultiScape200 Product,» Simera Sense, 2020. [Çevrimiçi]. Available: https://www.simera-sense.com/products/xscape200/multiscape200-cis/. [Erişildi: 2024].

[12] CubeComm, «X-BAND TRANSMITTER DVBS2 Encoding User Manual,» 2022.

[13] ISISpace, «TXS Interface Control Document,» ISISpace, 2022.

[14] ISISpace, «Interface Control Document RXS [S-Band Receiver],» ISISpace, 2022.

[15] R. B. -a.-. T. F. Team, «Mastering the FreeRTOS™ Real Time Kernel,» p. 326, 2023.

[16] C. Cappelletti, «CubeSat assembly, integration, testing and verification,» %1 içinde *CubeSat HandBook*, 2020.

[17] S. I. L. a. S. Maria, «SIL Chameleon 12U to 27U OMSR Flexible Satellite Bus Bus to Payload ICD,» 2020, pp. 1-24.

[18] E. Mabrouk, «What are SmallSats and CubeSats?,» 07 06 2017. [Çevrimiçi]. Available: https://www.nasa.gov/content/what-are-smallsats-and-cubesats . [Erişildi: 12 01 2024].

[19] A. N. Nikicio, «Software Development for Galassia CubeSat-Design, Implementation and In-Orbit Validation,» *Research Gate*, p. 9, 2017.

[20] T. O. Moxnes, «A common software framework for a CubeSat with multiple Payloads,» Norwegian University of Science and Technology, 2021.

[21] B. K. K. S. A. R. A. Emirhan Eser Gül, «ON-BOARD SOFTWARE DEVELOPMENT FOR A 3U CUBESAT,» p. 7, 2022.

[22] K. Mars, «About Space Radiation,» 2017. [Çevrimiçi]. Available: https://www.nasa.gov/hrp/elements/radiation/about. [Erişildi: Mar 2024].

CURRICULUM VITAE

Name Surname : Mehreen Azam

EDUCATION :

- **B.Sc. :** 2018, National University of Sciences and Technology (NUST), Faculty of Aeronautics, Avionics Engineering, Islamabad Pakistan

PROFESSIONAL EXPERIENCE AND REWARDS:

- Istanbul Technical University - Spacecraft Systems Design and Test Laboratory
  - Communication and Software Engineer 2022- Present

- Defense Setup
  - Airborne Avionics System Maintenance Officer, 2020-2022
  - Communication and Networking Officer 2019
  - R&D EW Officer 2018

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Mehreen Azam,** 2024: Development/Testing of Software for a CubeSat for High-Resolution Earth Observation in a Low Earth Orbit, 3rd International Graduate Research Symposium, IGRS'24 May 09, 2024 Istanbul, Turkiye.