

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

Carlos Augusto Porto Freitas

PLATAFORMA DE TESTES COM ARQUITETURA REUTILIZÁVEL PARA SUBSISTEMAS DE CUBESATS

Florianópolis

2026

Carlos Augusto Porto Freitas

PLATAFORMA DE TESTES COM ARQUITETURA REUTILIZÁVEL PARA SUBSISTEMAS DE CUBESATS

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Eletrônica do Centro Tecnológica da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador: Eduardo Augusto Bezerra

Coorientador: João Cláudio Elsen Barcellos

Florianópolis

2026

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Freitas, Carlos Augusto Porto
Plataforma de testes com arquitetura reutilizável para subsistemas de CubeSats / Carlos Augusto Porto Freitas ; orientador, Eduardo Augusto Bezerra, coorientador, João Cláudio Elsen Barcellos, 2026.
86 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Eletrônica, Florianópolis, 2026.

Inclui referências.

1. Engenharia Eletrônica. 2. FlatSat. 3. CubeSats. 4. testes. 5. verificação e validação. I. Bezerra, Eduardo Augusto. II. Barcellos, João Cláudio Elsen. III. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. IV. Título.

Carlos Augusto Porto Freitas

PLATAFORMA DE TESTES COM ARQUITETURA REUTILIZÁVEL PARA SUBSISTEMAS DE CUBESATS

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Eletrônica" e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Eletrônica.

Florianópolis, 25 de fevereiro de 2026.



Documento assinado digitalmente
Fernando Rangel de Sousa
Data: 03/03/2026 09:58:06-0300
CPF: ***.649.114-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Fernando Rangel de Sousa, Dr.
Coordenador do Curso de Graduação em
Engenharia Eletrônica



Documento assinado digitalmente
Eduardo Augusto Bezerra
Data: 25/02/2026 15:30:13-0300
CPF: ***.851.577-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Augusto Bezerra, Ph.D.
Orientador
Universidade Federal de Santa Catarina - UFSC



Documento assinado digitalmente
Joao Claudio Elsen Barcellos
Data: 25/02/2026 15:33:52-0300
CPF: ***.320.029-**
Verifique as assinaturas em https://v.ufsc.br

Eng. João Cláudio Elsen Barcellos, Me.
Coorientador
Universidade Federal de Santa Catarina - UFSC

Banca de avaliação:



Documento assinado digitalmente
Eduardo Augusto Bezerra
Data: 25/02/2026 15:30:32-0300
CPF: ***.851.577-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Augusto Bezerra, Ph.D.
Orientador
Universidade Federal de Santa Catarina - UFSC



Documento assinado digitalmente
HECTOR PETTENGHI ROLDAN
Data: 25/02/2026 16:35:12-0300
CPF: ***.054.601-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Héctor Pettenghi Roldán, Dr.
Avaliador
Universidade Federal de Santa Catarina - UFSC



Documento assinado digitalmente
ELCO JOAO DOS SANTOS JUNIOR
Data: 25/02/2026 16:11:33-0300
CPF: ***.534.089-**
Verifique as assinaturas em https://v.ufsc.br

Eng. Elço João dos Santos Junior, Dr.
Avaliador
SENAI/SC

Dedico este trabalho à minha vó, Satira.

AGRADECIMENTOS

Agradeço à minha mãe e maior inspiração, Maria Braulia, por seu carinho, incentivo e apoio incondicionais. Sem a sua presença constante não teria sido possível seguir o caminho que escolhi.

Registro ainda minha gratidão à memória de minha vó, Satira, e de meu pai, Carlos, por acreditarem em mim e nos meus sonhos.

Agradeço ao meu padastro, Emil, ao meu tio, Luiz, à minha tia, Laurita e aos meus primos, André e Lucas, por todo o apoio e carinho durante minha vida.

Agradeço ao meu orientador, Eduardo Bezerra, pela sua orientação, pelo apoio, pelas oportunidades e pela confiança depositada em mim.

Agradeço ao meu coorientador e grande amigo, João Barcellos, pela sua orientação, dedicação e paciência. Sem sua ajuda este trabalho não seria possível.

Agradeço aos meus colegas e amigos do SpaceLab e ISI-SE, que estiveram ao meu lado durante os momentos felizes, tristes, fáceis ou difíceis; a presença de vocês nesses momentos foi fundamental.

Agradeço, por fim, ao CNPq e à Agência Espacial Brasileira pelo incentivo ao estudo e a pesquisa científica, tornando esse trabalho possível.

RESUMO

A adoção de satélites de pequeno porte, em especial os CubeSats, cresceu significativamente nos últimos anos. Contudo, aspectos relacionados a confiabilidade desses satélites são levantados na literatura, em particular, como a falta de rigor na campanha de Verificação e Validação (V&V) desses satélites impacta negativamente a taxa de sucesso de suas missões. Nesse contexto, este trabalho apresenta a pesquisa e o desenvolvimento de uma arquitetura reutilizável para a execução de procedimentos de teste e de V&V em subsistemas de CubeSats, com o objetivo de aprimorar campanhas de V&V e aumentar a confiabilidade desses sistemas. Inicialmente, são discutidos aspectos de confiabilidade aplicados a CubeSats, bem como o conceito de FlatSat e sua relevância em campanhas de V&V. Em seguida, a plataforma FlatSat2.0, desenvolvida pelo SpaceLab para realização de testes com subsistemas de CubeSats, é apresentada como base para o desenvolvimento da arquitetura proposta. A metodologia empregada para validação da plataforma e desenvolvimento da arquitetura é descrita, seguida da apresentação dos testes iniciais realizados na FlatSat2.0, cujos resultados foram divulgados no Latin American CubeSat Workshop 2024. O desenvolvimento da arquitetura contempla a descrição de hardware, a construção de um sistema Linux customizado e a implementação de serviços de software, os quais são executados em um processador embarcado na plataforma. A solução proposta é open source e disponibilizada para uso em futuras missões. Por fim, um estudo de caso com subsistemas reais de CubeSats demonstra a aplicabilidade da arquitetura, validando tanto a solução desenvolvida quanto a plataforma FlatSat2.0, viabilizando sua utilização em missões em andamento do SpaceLab, a exemplo da Constelação Catarina.

Palavras-chaves: FlatSat, CubeSats, testes, verificação e validação.

ABSTRACT

The adoption of small satellites, particularly CubeSats, has grown significantly in recent years. However, reliability-related issues have been highlighted in the literature, especially regarding how the lack of rigor in Verification and Validation (V&V) campaigns negatively impacts mission success rates. In this context, this work presents the research and development of a reusable architecture for the execution of test and V&V procedures in CubeSat subsystems, aiming to improve V&V campaigns and increase system reliability. Initially, reliability aspects applied to CubeSats are discussed, as well as the concept of FlatSat and its relevance in V&V campaigns. Then, the FlatSat2.0 platform, developed by SpaceLab for testing CubeSat subsystems, is introduced as the foundation for the development of the proposed architecture. The methodology employed for platform validation and architecture development is described, followed by the presentation of the initial tests performed on FlatSat2.0, whose results were presented at the Latin American CubeSat Workshop 2024. The architecture development encompasses hardware description design, the construction of a customized Linux system, and the implementation of software services executed on an embedded processor within the platform. The proposed solution is open source and made available for use in future missions. Finally, a case study involving real CubeSat subsystems demonstrates the applicability of the architecture, validating both the developed solution and the FlatSat2.0 platform, thereby enabling its use in ongoing SpaceLab missions, such as the Catarina Constellation.

Keynotes: FlatSat, CubeSats, testing, verification and validation.

LISTA DE ILUSTRAÇÕES

Figura 1 – Número de nanossatélites lançados 20
Figura 2 – Exemplos de tamanhos de CubeSats e suas classes 21
Figura 3 – Falhas e problemas técnicos observados em nanossatélites 22
Figura 4 – Exemplo de filosofia de desenvolvimento e teste para CubeSats 25
Figura 5 – Exemplos para cada classificação FlatSats 27
Figura 6 – Plataforma FlatSat2.0 29
Figura 7 – Metodologia para o desenvolvimento e validação da FlatSat2 33
Figura 8 – Fluxo do projeto de descrição de hardware 38
Figura 9 – *Block Design* para o projeto base 40
Figura 10 – Exemplo de uso da infraestrutura para comunicação com os serviços 45
Figura 11 – Diagrama de blocos do serviço *read-sensors* 47
Figura 12 – *Block Design* para o projeto usado no estudo de caso 53
Figura 13 – Diagrama de blocos das aplicações e configuração do Zynq 54
Figura 14 – Configuração do ambiente de testes para a primeira campanha de V&V . . 55
Figura 15 – Telemetria contendo leituras de tensão das baterias 56
Figura 16 – Leituras de temperatura obtidas do sensor TMP112 e do EDC 56
Figura 17 – Leituras de corrente, tensão e potência obtidas no primeiro teste; incluindo leituras dos sensores embarcados e telemetria 57
Figura 18 – Configuração do ambiente de testes para a segunda campanha de V&V . . 58
Figura 19 – Simulação da órbita do satélite Horyu-4; mostrando a ativação do aquecedor, eventos de ativação do EDC e períodos de operação simultânea 60
Figura 20 – Consumo de potência do sistema durante a emulação do cenário de missão 60
Figura 21 – Comportamento do sistema de potência durante o cenário emulado; incluindo leituras dos barramentos regulados e o perfil de descarga da bateria 61
Figura 22 – Leituras de temperatura realizadas durante a emulação de cenário de missão; incluindo medições dos termopares e telemetria do EPS2.0 62
Figura 23 – *Dashboard* do *Grafana* com as leitura dos sensores durante o teste 63

LISTA DE TABELAS

Tabela 1 – Classificação para satélites baseado na sua massa 19
Tabela 2 – Comparação entre FlatSats e suas funcionalidades 28
Tabela 3 – Uso de recursos do PL pelo design base 40
Tabela 4 – Comandos do serviço *read-sensors* expostos via interface ZMQ 47
Tabela 5 – Tarefas do *flight software* da missão GOLDS-UFSC emuladas pela MicroZed 50
Tabela 6 – Uso de recursos do PL pelo design usado no estudo de caso 52

LISTA DE ABREVIATURAS E SIGLAS

AIT	*Assembly, Integration, and Testing*
AXI	*Advanced eXtensible Interface*
BD	*Block Design*
BRAM	*Block Random Access Memory*
BSP	*Board Support Package*
CAN	*Controller Area Network*
CDS	*CubeSat Design Specification*
CLPD	*Complex Programmable Logic Device*
COTS	*Commercial-off-The-Shelf*
DDR	*Double Data Rate*
DFU	*Device Firmware Update*
DoA	*Dead on Arrival*
DSP	*Digital Signal Processing*
ECSS	*European Cooperation for Space Standardization*
EDC	*Environmental Data Collector*
EGSE	*Electrical Ground Support Equipment*
EMIO	*Extended Multipurpose I/O*
EMC	Compatibilidade Eletromagnética
EMI	Interferência Eletromagnética
EPS	*Electrical Power System*
EPS2	*Electrical Power System 2.0*
EQM	Modelo de Engenharia e Qualificação
ESA	*European Space Agency*
FM	Modelo de Voo
FPGA	*Field-Programmable Gate Array*

GPIO	*General Purpose I/O*
IIO	*Industrial I/O*
IIOD	*IIO Daemon*
INPE	Instituto Nacional de Pesquisas Espaciais
IP	Propriedade Intelectual
I²C	*Inter Integrated Circuit*
JSON	*JavaScript Object Notation*
MIO	*Multipurpose I/O*
NASA	*National Aeronautics and Space Administration*
OBC	*On-Board Computer*
OBDH	*On-Board Computer and Data Handling*
OBDH2	*On-Board Computer and Data Handling 2.0*
OS	Sistema Operacional
HIL	*Hardware-in-the-Loop*
PCB	*Printed Circuit Board*
PCD	Plataforma de Coleta de Dados
PFM	Modelo *Protoflight*
PL	*Programmable Logic*
PMM	Plataforma Multi-Missão
PS	*Processing System*
PUB/SUB	*Publish-Subscribe*
RTD	*Resistance Temperature Detector*
SoC	*System-on-Chip*
SoM	*System-on-Module*
SOA	*Service-Oriented Architecture*
SPI	*Serial Peripheral Interface*
SSH	*Secure Shell*
TLE	*Two-line Element*
TTC	*Telemetry, Tracking and Command*
TTC2	*Telemetry, Tracking and Command 2.0*
UART	*Universal Asynchronous Receiver / Transmitter*

UFSC	Universidade Federal de Santa Catarina
UHF	*Ultra High Frequency*
USB	*Universal Serial Bus*
V&V	Verificação e Validação
XSA	*Xilinx Support Archive*
ZMQ	*ZeroMQ*

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVOS 17
1.1.1 Objetivo Geral 17
1.1.2 Objetivos Específicos 18
1.2 ORGANIZAÇÃO DO TEXTO 18

2 FUNDAMENTAÇÃO E TRABALHOS RELACIONADOS 19
2.1 CONFIABILIDADE EM CUBESATS 21
2.2 FLATSATS 25
2.2.1 Revisão da Literatura 25
2.2.2 Plataforma FlatSat2.0 28
2.2.2.1 MicroZed 29
2.2.2.2 Sensoriamento 30
2.2.2.3 Injeção de Falhas 31
2.3 LINUX EMBARCADO 31

3 METODOLOGIA 33
3.1 TESTES PRELIMINARES 34
3.2 DESENVOLVIMENTO DA PLATAFORMA DE SOFTWARE 34
3.3 ESTUDO DE CASO 35

4 DESENVOLVIMENTO 36
4.1 VALIDAÇÃO INICIAL 36
4.2 DESCRIÇÃO DE HARDWARE - ZYNQ PL 37
4.3 ECOSSISTEMA DE SOFTWARE - ZYNQ PS 40
4.4 SERVIÇOS E CONECTIVIDADE 43
4.4.1 Protocolo de Comunicação com Serviços 45
4.4.2 Sistema de Monitoramento 46
4.4.3 Acesso Remoto e Interface de Rede 48
4.4.4 Emulador de Subsistemas 49

5 ESTUDO DE CASO - FLORIPASAT2 51
5.1 PLATAFORMA FLORIPASAT2 51
5.2 MUDANÇAS NO DESIGN PARA A PLATAFORMA 52
5.3 EMULAÇÃO DE SUBSISTEMAS E COLETA DE TELEMETRIA 54
5.3.1 Descrição do Teste 54

5.3.2 Resultados e Discussão 55
5.4 EMULAÇÃO DE CENÁRIOS DE MISSÃO 58
5.4.1 Descrição do Teste 58
5.4.2 Resultados e Discussão 59

6 CONCLUSÃO 64

REFERÊNCIAS 65

APÊNDICE A ARTIGO APRESENTADO EM: *LATIN AMERICAN CUBESAT WORKSHOP 2024* 74

1 INTRODUÇÃO

Desde o lançamento bem sucedido do primeiro satélite, *Sputnik 1* (1), em 1957, milhares de satélites foram lançados nas décadas seguintes (2). Tradicionalmente, os satélites produzidos eram consideravelmente grandes e complexos, normalmente contendo múltiplos instrumentos e podendo totalizar toneladas de massa (3), a exemplo do *Envisat Earth Observation* desenvolvido pela *European Space Agency* (ESA), o qual continha 10 instrumentos para monitoramento terreste e pesava mais de 8 toneladas (4).

Contudo, o alto custo e complexidade do desenvolvimento, operação e lançamento de satélites de grande porte fizeram com que essa indústria permanecesse, por muito tempo, restrita a grandes instituições, normalmente financiadas por governos de países industrializados, como a *National Aeronautics and Space Administration* (NASA) e a ESA, entre outras (3). Além disso, à medida que aumentam a complexidade e as dimensões desses sistemas, a infraestrutura necessária para testes e validações torna-se proporcionalmente mais escassa e onerosa. Esse cenário é exemplificado no desenvolvimento do *Envisat*, no qual diversas adaptações nos equipamentos de teste foram necessárias para acomodar o satélite, além das correções implementadas em seus próprios subsistemas (5).

Desse modo, o uso de satélites de menor porte, com componentes mais acessíveis e menor tempo de desenvolvimento tornou-se mais popular. Um marco importante dessa tendência foi o surgimento do padrão CubeSat, concebido em 1999 por Jordi Puig-Suari e Bob Twiggs nas universidades de *Stanford* e *California Polytechnic State University* (6). Esse padrão é definido através da *CubeSat Design Specification* (CDS) (7), a qual define, entre outros parâmetros, o formato de um CubeSat em cubos de aresta com dimensão de 10 cm. Um único cubo que atende essa especificação corresponde à uma unidade básica de um CubeSat, denominada de 1U. A combinação de várias unidades básicas possibilita a concepção de satélites de maior porte, com dimensões que variam de 1U até 27U (3).

A padronização do formato CubeSat, por meio da CDS, diminuiu o custo com a estrutura necessária para o lançamento de cada satélite (8). Além disso, CubeSats tendem a utilizar somente componentes *Commercial-off-The-Shelf* (COTS) — projetados para o mercado geral e disponíveis comercialmente em larga escala — em substituição a componentes de classe espacial e componentes desenvolvidos especificamente para cada missão, como ocorre tradicionalmente em satélites de grande porte. Essas características possibilitam, portanto,

o desenvolvimento de satélites com consumo significativamente menor de tempo e recursos. Consequentemente, seu desenvolvimento torna-se viável a instituições que, historicamente, não tinham acesso ao setor espacial, incluindo universidades e *startups*.

Dessa forma, a adoção de nanosatélites — em especial os CubeSats — apresentou um crescimento significativo nos últimos anos. Conforme apresentado em (9), até 30 de abril de 2025 foram lançados, ao todo, 2730 CubeSats, dos quais — entre aqueles ainda em órbita — 812 permanecem operacionais e 208 encontram-se não operacionais. Contudo, segundo a análise feita em (10), dentre todos os nanosatélites lançados, pelo menos 677 apresentaram problemas ou falhas. As falhas observadas foram organizadas em categorias, sendo alguns exemplos: problemas com o lançamento, falhas desconhecidas e falha no contato inicial. Além disso, entre as falhas conhecidas, aquelas relacionadas à comunicação, ao sistema de potência e à atitude do satélite estão entre as mais recorrentes.

Em linha com os dados da análise, a literatura destaca de forma recorrente a confiabilidade desses satélites, discutindo falhas frequentes e técnicas de mitigação associadas. Os autores de (11) e (12), por exemplo, analisam que as missões estudantis — conduzidas por universidades e comumente realizadas, contribuindo para a formação de estudantes e fomentando o desenvolvimento tecnológico e científico (3, 13) — não tem altas taxas de sucesso, ainda de acordo com os autores de (12), isso se deve, principalmente, pela falta de rigor na campanha de testes, verificação e validação.

Para mitigar esses problemas, pode-se empregar uma extensa campanha de Verificação e Validação (V&V), que utiliza diversos métodos para validar o sistema diante de cenários prováveis e atípicos, observando seu comportamento de modo a identificar vulnerabilidades (14). Uma opção válida para a campanha de V&V é o uso de uma FlatSat, que consiste em uma estratégia para integração dos módulos do satélite, onde os subsistemas são montados sobre uma bancada de trabalho (15), facilitando o acesso aos subsistemas quando se comparado ao satélite integrado. O termo FlatSat também é frequentemente utilizado para se referir a uma implementação específica desse conceito, sendo usado como sinônimo de plataforma de integração e/ou de testes. A adoção desse tipo de plataforma pode ser visto em diversos trabalhos, a exemplo de (12, 16, 17, 18), os quais relatam seu uso durante a campanha de V&V de suas respectivas missões.

Um mapeamento sistemático sobre essas plataformas foi conduzido em (15), explorando diversas plataformas em um esforço de classificá-las de acordo com sua estrutura e interface com os subsistemas. Nessa revisão foi observado que muitas dessas plataformas incluem funcionalidades integradas além de simplesmente conectar os subsistemas. Contudo, percebe-se também que poucas FlatSats incluem capacidade de realizar testes automatizados ou reconfiguração. No geral, essas funcionalidades são implementadas através de *hardware* externo na forma de um *Electrical Ground Support Equipment* (EGSE), como nos casos do ISTSat-1 e EIRSAT-1, apresentados em (12) e (17) respectivamente, que se utilizaram EGSEs para realizar

testes como monitoramento do consumo e simulação de comandos da estação terrestre.

Nesse contexto, dado o histórico da utilização de FlatSats no SpaceLab (19, 18), Laboratório de Pesquisa em Sistemas Espaciais da Universidade Federal de Santa Catarina (UFSC), uma plataforma híbrida foi desenvolvida em (20), baseada na consolidação das informações provenientes do mapeamento sistemático (15). A plataforma conta com sensores, sistemas de injeção de falhas e processador embarcado além da integração dos subsistemas. O processador embarcado — um *System-on-Module* (SoM) que integra um *System-on-Chip* (SoC) composto por processadores *ARM* e lógica programável — é uma adição de grande relevância para a plataforma, visto que isso possibilita a implementação de testes automatizados, emulação de subsistemas, entre outras funcionalidades que, normalmente, precisariam ser implementatadas através de *hardware* externo.

Contudo, apesar do potencial da plataforma proposta por (20), ainda não há *software* implementado para o processador embarcado, efetivamente limitando o uso da FlatSat para apenas a conexão entre os subsistemas do satélite. Apesar dessa integração eliminar problemas práticos causados por conexões manuais entre os módulos por cabos ou *jumpers*, problemas esses que aconteceram em (16) e (19), isso implica em todo e qualquer teste que necessita se comunicar, controlar ou monitorar o sistema requerer equipamento externo para tal. Adicionalmente, apesar de tentativas anteriores (21), a FlatSat ainda não foi validada de modo que demonstre sua funcionalidade plena, não tendo sido realizados ensaios com subsistemas reais de nanosátelites.

Desse modo, este trabalho consiste na implementação de ferramentas de *software* reutilizáveis e extensíveis que possibilitem a realização de testes automatizados, monitoramento dos sensores embarcados e emulação de subsistemas para múltiplas missões. Ademais, um estudo de caso utilizando subsistemas reais é realizado, validando ambos a plataforma e as ferramentas desenvolvidas, permitindo, portanto, seu uso em missões futuras, a exemplo da Constelação Catarina (22).

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo geral do trabalho consiste na pesquisa e desenvolvimento de uma arquitetura reutilizável para execução de procedimentos de teste e V&V em módulos de CubeSats, implementada na plataforma desenvolvida no SpaceLab, que, para fins deste trabalho, será referida como FlatSat2.0.

1.1.2 Objetivos Específicos

- Desenvolvimento de imagem de sistema operacional e descrição de *hardware* customizados para a plataforma;
- Definição de meios para extensão e customização das ferramentas;
- Elaboração de aplicações para monitoramento de sensores e emulação de subsistemas;
- Validar a plataforma por meio de ensaios experimentais com subsistemas reais de satélite, incluindo estudo de caso com emulação de cenários de missão.

1.2 ORGANIZAÇÃO DO TEXTO

O restante deste trabalho é organizado da seguinte forma: no Capítulo 2 uma revisão de literatura sobre os temas de confiabilidade em CubeSats e FlatSats, bem como a fundamentação teórica para o trabalho são apresentadas. Subsequentemente, o Capítulo 3 descreve a metodologia utilizada na validação da plataforma, assim como no desenvolvimento da arquitetura proposta. Depois, no Capítulo 4, o desenvolvimento da arquitetura em todos os seus detalhes é apresentado. No Capítulo 5, um estudo de caso que busca validar a arquitetura e a plataforma com subsistemas reais é discutido. Finalmente, no Capítulo 6, as conclusões e considerações finais são apresentadas.

2 FUNDAMENTAÇÃO E TRABALHOS RELACIONADOS

A classificação de satélites normalmente é realizada através de sua massa, contudo não há consenso ou padronização formal sobre a nomenclatura e classe de satélites. Dessa forma, instituições, agências e pesquisadores adotaram classificações consideravelmente diferentes ao longo dos anos. Considerando isso, em uma tentativa de propor uma classificação unificada, os autores de (23) apresentam e discutem algumas dessas diferentes classificações, destacando as variações entre elas. A Tabela 1 apresenta uma nomenclatura comumente utilizada, inclusive em banco em dados conhecidos como (9). Nessa classificação satélites de pequeno porte são definidos como satélites de massa menor que 500 kg, sendo que essa classe é subdividida em sub-classes, apresentadas na Tabela 1.

Tabela 1: Classificação para satélites baseado na sua massa

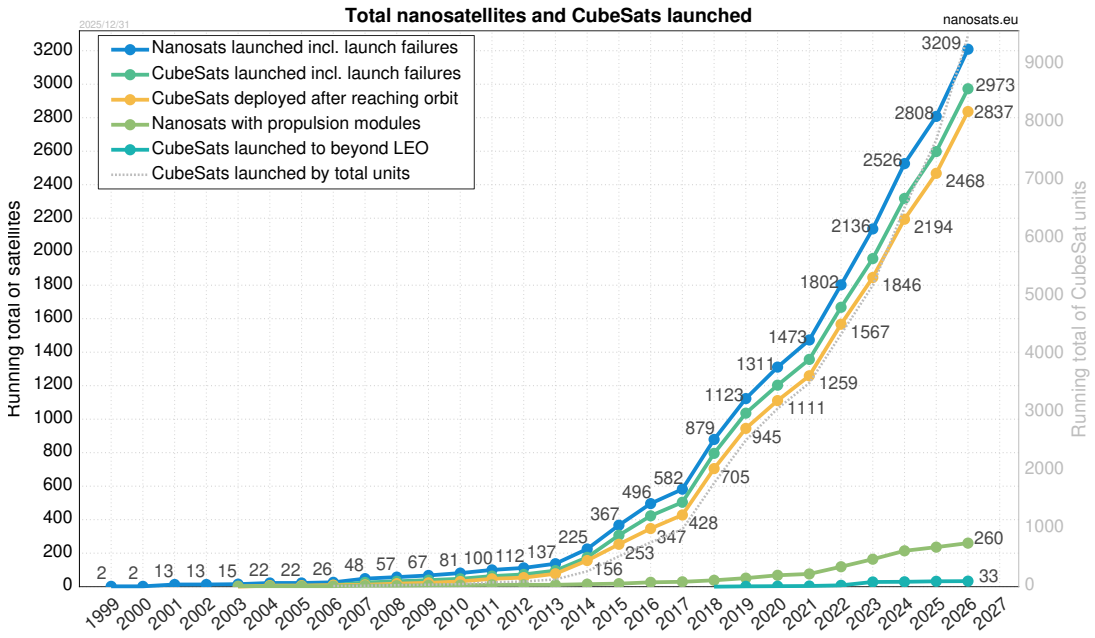
Classe do satélite	**Massa do satélite**
Satélites de grande porte	$\geq 1000\,\text{kg}$
Satélites de médio porte	$\geq 500\,\text{kg}$ e $< 1000\,\text{kg}$
Minissatélites	$\geq 100\,\text{kg}$ e $< 500\,\text{kg}$
Microssatélite	$\geq 10\,\text{kg}$ e $< 100\,\text{kg}$
Nanossatélite	$\geq 1\,\text{kg}$ e $< 10\,\text{kg}$
Picossatélites	$\geq 100\,\text{g}$ e $< 1\,\text{kg}$
Femtossatélites	$< 100\,\text{g}$

Fonte: Adaptado de (9) e (23).

A utilização de satélites de pequeno porte remonta ao início da exploração espacial. Um exemplo desse fato é o primeiro satélite artificial, o *Sputnik-1*, lançado com massa de aproximadamente 84 kg (1), o qual seria, portanto, classificado como um microssatélite segundo as classes atualmente adotadas. Contudo, a fim de se reduzir o custo com lançamento por carga útil ou instrumento, satélites de médio e grande porte se tornaram progressivamente mais comuns até atingirem seu ápice de uso no início dos anos 2000 (3). Os satélites de pequeno

porte passaram a ser utilizados de forma mais frequente a partir do final da década de 1990, impulsionados pela evolução tecnológica, bem como pelo aumento da complexidade e dos custos associados ao desenvolvimento, lançamento e operação de satélites de maior porte. Ademais, alguns autores como (3) e (24) atribuem parte desse movimento a concepção de padrões para desenvolvimento de satélites, a exemplo do padrão CubeSat. A evolução dos lançamentos de nanossatélites pode ser vista na Figura 1.

Figura 1 – Número de nanossatélites lançados

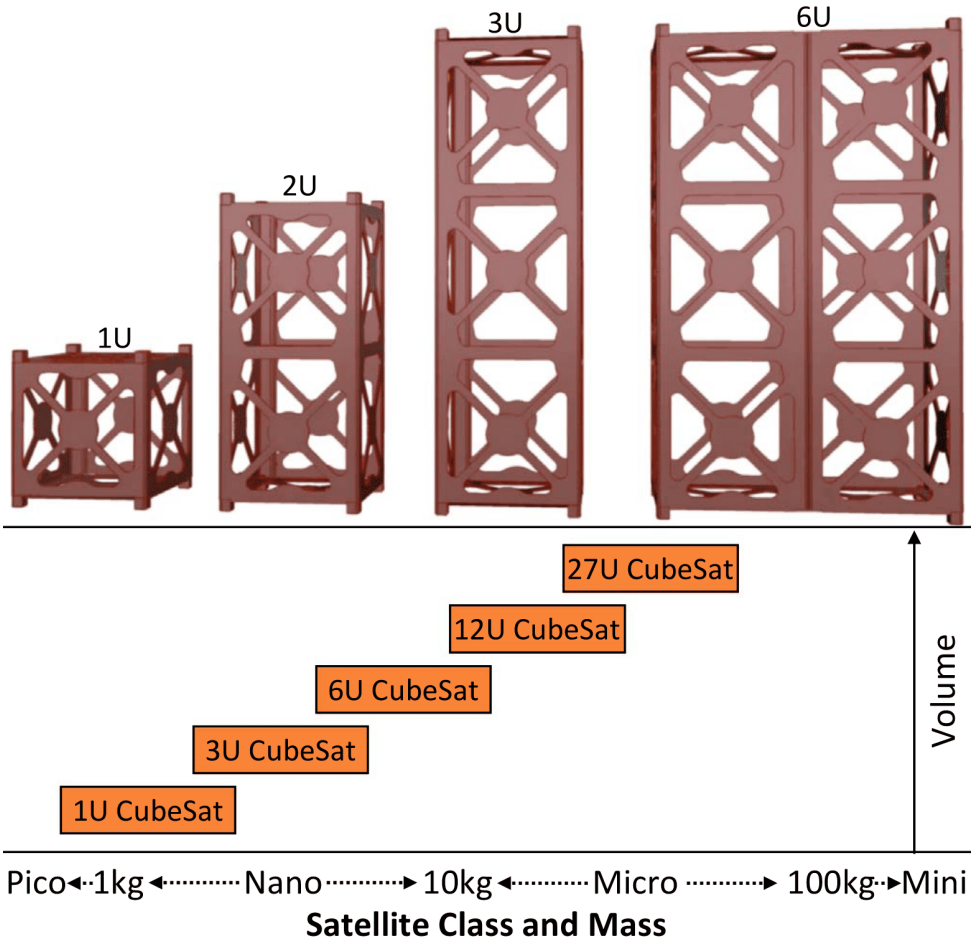


Fonte: Obtido de (9).

O padrão CubeSat, em particular, tem participação expressiva nos recentes lançamentos de satélites de pequeno porte. Através da redução dos custos relacionados com seu lançamento e facilidade de desenvolvimento, milhares desses satélites foram lançados até o presente momento (10). Concebido em 1999 nas universidades de *Stanford* e *California Polytechnic State University* (6), o padrão descreve, através da CDS (7), satélites baseados em uma unidade base, denominada 1U, a qual tem formato cúbico de arestas de 10 cm e massa de até 2 kg. Além disso, a especificação permite a junção de múltiplas dessas unidades para a concepção de satélites maiores. A Figura 2 demonstra alguns exemplos de tamanhos de CubeSats, bem como suas respectivas classes de satélite.

CubeSats foram utilizados inicialmente para validação de componentes em órbita, experimentos científicos e formação de estudantes, sendo majoritariamente desenvolvidos em universidades (3, 11). Entretanto, com o amadurecimento tecnológico e a ampla adoção dos CubeSats, sua aplicação estendeu-se a soluções mais ambiciosas, incluindo experimentos científicos complexos e aplicações comerciais, como as desenvolvidas pela *Planet*, que oferece comercialmente dados ópticos destinados à observação da Terra, sendo utilizada, por exemplo, em monitoramento florestal (25).

Figura 2 – Exemplos de tamanhos de CubeSats e suas classes



Fonte: Obtido de (3, pg. 60).

2.1 CONFIABILIDADE EM CUBESATS

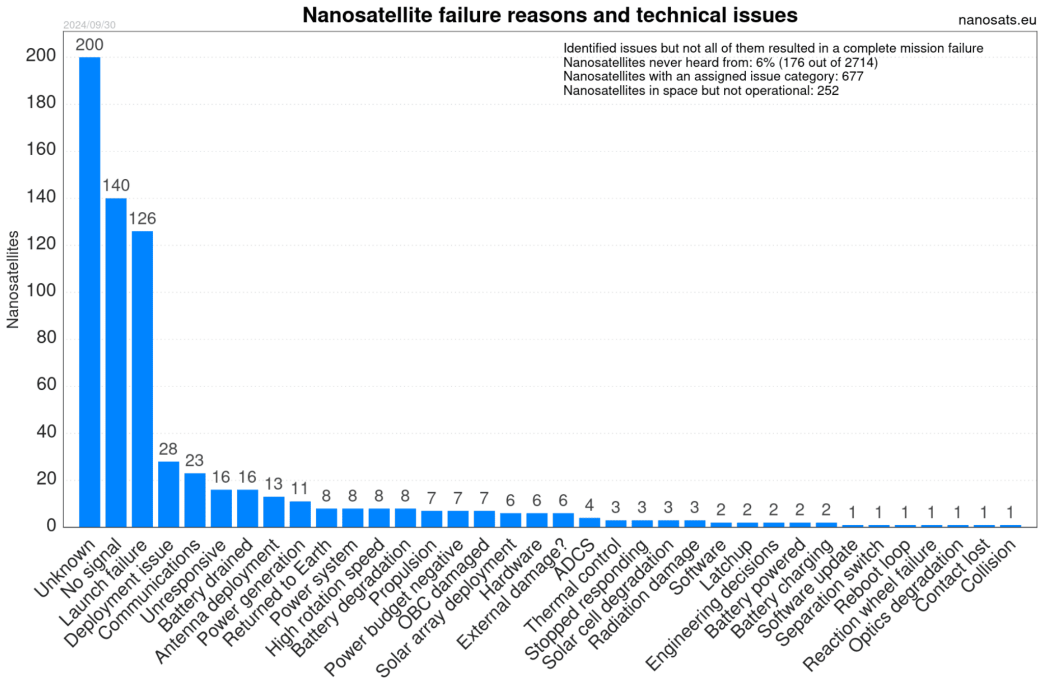
Contudo, apesar crescente popularidade desses satélites, discussões sobre a confiabilidade e sucesso de suas missões são comuns na literatura. Os autores de (11), por exemplo, reportam que apenas 48% dos satélites que foram lançados com sucesso até o ano de 2010 completaram suas missões. Adicionalmente, ainda segundo os autores, mais da metade dos satélites foram desenvolvidos por universidades e grande parte das missões tinham como objetivo demonstração tecnológica. Os autores de (12) associam a alta taxa de erro à inexperiência dos times de estudantes, bem como o foco educacional, culminando em menos esforço no controle de qualidade da solução final. Entretanto, os autores também comentam que a taxa de falhas dos satélites lançados em 2019 foi de 10%, uma melhora significativa desde 2010, a qual é atribuída a uma preocupação maior com a campanha de V&V.

Em consonância com os trabalhos anteriores, a pesquisa elaborada em (26) evidencia que missões elaboradas em universidades, até 2016, tem taxas elevadas de falhas no contato inicial — também nomeado de *Dead on Arrival* (DoA) — ou falhas nas etapas iniciais da

missão, totalizando quase 50% dos lançamentos bem sucedidos. Os autores argumentam que isso ocorre devido a falta de tempo e/ou recursos para testes a nível de sistema, reforçando as conclusões de trabalhos anteriores.

Em (10) — um estudo mais recente — informações de pesquisas e bancos dados foram compiladas a fim de se traçar um perfil da confiabilidade de nanossatélites ao longo dos anos. Dentre as discussões apresentadas, a análise do motivo das falhas é de particular interesse, visto que pode ser utilizado como guia para novas missões. A Figura 3 mostra um histograma com a ocorrência de cada tipo de falha. Percebe-se pela figura que, ao se retirar falhas desconhecidas, falhas no lançamento e DoA, problemas relacionados à comunicação e ao sistema de potência são mais recorrentes. O autor também aponta que apesar de ser comum atribuir falhas a falta ou insuficiência de testes, muitos dos problemas registrados poderiam ser solucionados ou mitigados com um design de *hardware* e *software* mais robusto.

Figura 3 – Falhas e problemas técnicos observados em nanossatélites



Fonte: Obtido de (10, pg. 11).

Nesse contexto, os autores de (14) modelam a confiabilidade de CubeSats ao longo do tempo, comparando duas estratégias para aumento de confiabilidade: i) melhoria dos procedimentos de testes e ii) redundância de subsistemas. Os modelos e simulações foram alimentados com o banco de dados que contém descrições de falhas em CubeSats apresentado em (27). A análise da simulação demonstra que, se mutuamente exclusivos, alocar recursos na melhoria dos procedimentos de teste produz resultados melhores para a maioria das missões.

Além disso, ao não implementar redundância, há o benefício de menor complexidade do satélite e mais espaço físico para a carga útil. Ademais, a pesquisa indica que a adição de

redundância torna-se mais vantajoso apenas para missões de duração maior que 10 anos, algo considerado atípico dado o uso atual de CubeSats. Naturalmente, utilizar ambas as estratégias resulta no satélite com a maior confiabilidade e, caso seja factível, os autores recomendam o uso de ambas.

Em síntese, percebe-se que a literatura associa fortemente falhas nesse tipo de missão com a campanha de V&V realizada durante o desenvolvimento, sendo pauta recorrente nos trabalhos que abordam o tema. Logo, faz-se necessário descrever brevemente o processo e as etapas de uma campanha de V&V.

Tradicionalmente, projetos de satélites de maior porte utilizam normas e padrões específicos, em particular, as normas da *European Cooperation for Space Standardization* (ECSS) são comumente utilizadas. Similarmente, no desenvolvimento e availação de CubeSats tipicamente também se adota os padrões da ECSS, complementando as definições da CDS. Dentre os diversos padrões empregados, no contexto de V&V destacam-se ECSS-E-ST-10C (28) para engenharia de sistemas, ECSS-E-ST-10-03C e ECSS-E-HB-10-03A (29, 30) para integração e testes, e, por fim, os padrões ECSS-E-ST-10-02C e ECSS-E-HB-10-02A (31, 32) para verificação. Adicionalmente, o padrão GSFC-STD-7000B (33) concebido pela NASA é frequentemente adotado para a especificação de testes ambientais, bem como para a especificação dos padrões de Compatibilidade Eletromagnética (EMC) e Interferência Eletromagnética (EMI).

Os padrões da ECSS organizam o processo de desenvolvimento das missões em diversas etapas e revisões de projeto, sendo que, a fim de reduzir complexidade, projetos de CubeSats tendem a utilizar apenas parte dos padrões e revisões (34, 35, 36). No início do projeto, requisitos mínimos para execução da missão são delimitados, os quais servem de guia para a escolha ou desenvolvimento dos subsistemas, além de estabelecer o escopo da campanha de V&V. Idealmente, os requisitos originais devem ser mantidos até o final da missão, contudo, no contexto de CubeSats, devido ao processo de desenvolvimento acelerado e a possível inexperiência das equipes, os requisitos podem ser revistos ao longo do projeto, algo que aconteceu em (36).

Uma vez que os requisitos mínimos foram elaborados, começa-se o projeto e desenvolvimento do satélite propriamente dito. Durante esse processo, a campanha de V&V é realizada de diversas formas, refletindo o nível de maturidade do projeto. No começo, por exemplo, testes funcionais a nível de subsistema são realizados, os quais verificam se tanto *hardware* quanto o *software* dos subsistemas implementam as funcionalidades esperadas.

Essa abordagem é frequentemente utilizada, principalmente quando há desenvolvimento próprio de subsistemas, podendo ser vista em (12, 17, 36, 37, 38). Além disso, a depender do projeto, testes ambientais como testes de vibração, exposição a radiação e ciclagem térmica em vácuo podem ser realizados nesse estágio. Os autores de (37) e (38) demonstram que, para o desenvolvimento e verificação da carga útil de suas respectivas missões, testes ambientais a

nível de subsistema foram feitos.

Em seguida, testes de integração são normalmente realizados, buscando validar as interfaces entre subsistemas, eliminar possíveis interferências e avaliar o comportamento do sistema como todo. Esses testes frequentemente utilizam configurações de *Hardware-in-the-Loop* (HIL), bem como empregam intensivamente EGSEs para emular sensores, dados orbitais, falhas, entre outras variáveis ambientais e de operação (38). Nesse contexto, a equipe de (36) conseguiu identificar e corrigir diversos problemas de interface durante os testes de integração feitos com o auxílio de uma FlatSat.

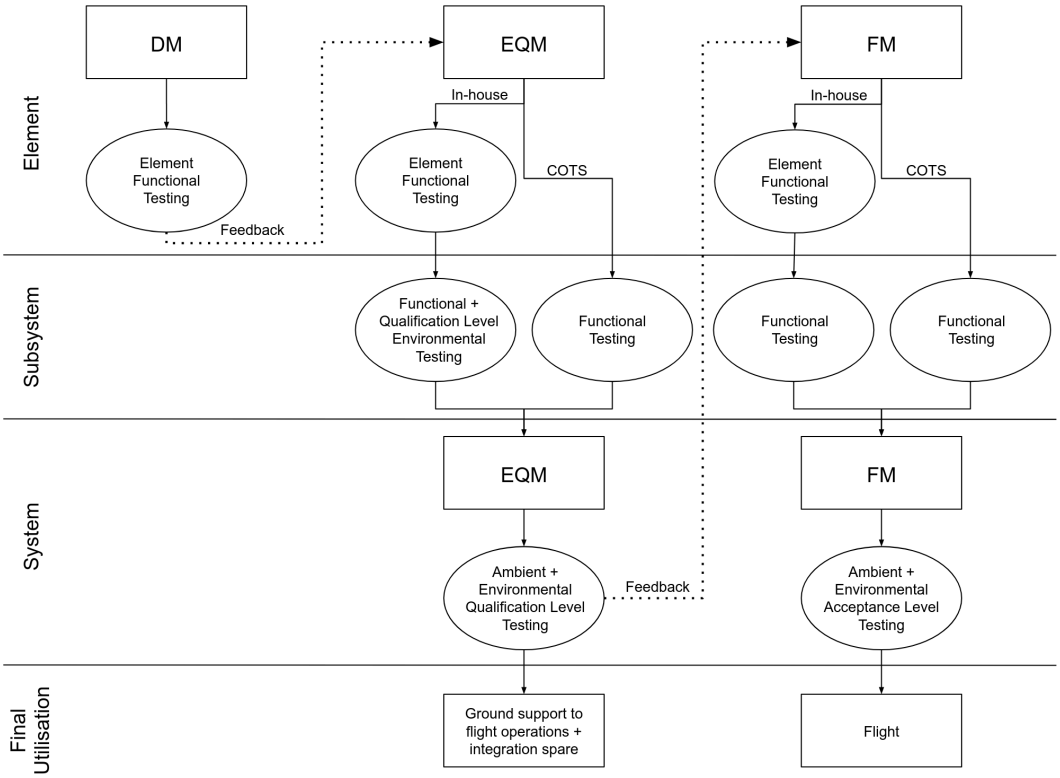
Adicionalmente, testes de cadeia completa (*end-to-end tests*) — os quais emulam cenários reais de operação, bem como verificam a funcionalidade de todos os modos de operação do satélite — são comuns. Em (39), esse tipo de teste permitiu a identificação e correção de um falha critíca na cadeia de comunicação do IDEASSAT, causada por problemas no casamento de impedância entre o transceptor de *Ultra High Frequency* (UHF) e a antena. Esse descasamento causou a ruptura da trilha de transmissão, o que essencialmente compromete a capacidade de transmissão do satélite.

Por fim, o satélite é integrado seguindo o plano de *Assembly, Integration, and Testing* (AIT) e é submetido aos testes funcionais e ambientais necessários para garantir que o sistema completo cumpra integralmente os requisitos de missão. Essa campanha de testes é comumente realizada de duas formas distintas, escolhidas através diversos fatores como herança de voo dos subsistemas, experiência da equipe, nível de risco aceitável, entre outros (17, 38).

A primeira abordagem utiliza dois modelos do satélite, o Modelo de Engenharia e Qualificação (EQM) e o Modelo de Voo (FM), sendo que o EQM é submetido a níveis de qualificação — os quais submetem o sistema a condições mais extremas do que as esperadas em operação, de modo a verificar seu funcionamento — já o FM é submetido a níveis de aceitação, os quais são mais brandos que os de qualificação evitando impor estresse ao satélite que irá, de fato, ser lançado. Por outro lado, a segunda abordagem utiliza apenas um modelo, nomeado de Modelo *Protoflight* (PFM), que é submetido a testes mais rigorosos que os do FM, correndo o risco de estressar o satélite ou atrasar o cronograma com um retrabalho inesperado (17, 38).

A escolha sobre qual abordagem é a ideal depende da missão, visto que a concepção de um EQM tende a diminuir o risco da missão e permite uma campanha de qualificação e V&V mais completa, contudo essa filosofia também aumenta os custos e o cronograma da missão quando comparada ao uso de um PFM (17). Essa abordagem é frequentemente utilizada em CubeSats que tem módulos de desenvolvimento próprio e/ou sem herança de voo, alguns exemplos que a seguem são (17, 36, 38). A Figura 4 demonstra a filosofia de desenvolvimento do EIRSAT-1 (17), a qual serve para exemplificar o ciclo de desenvolvimento e V&V para missões desse tipo.

Figura 4 – Exemplo de filosofia de desenvolvimento e teste para CubeSats



Fonte: Obtido de (17, pg. 5).

2.2 FLATSATS

Essa seção apresenta uma breve revisão da literatura acerca de FlatSats, incluindo suas possibilidades de implementação e seu uso em campanhas de V&V. Além disso, na sequência, é apresentada a plataforma FlatSat2.0 — desenvolvida no SpaceLab —, permitindo uma comparação direta com as plataformas discutidas na revisão.

2.2.1 Revisão da Literatura

Ao pesquisar sobre campanhas de V&V para missões de CubeSats percebeu-se que muitas delas utilizaram FlatSats para auxiliar no processo, alguns exemplos de missões em que essas plataformas foram usadas são (12, 16, 17, 36, 38, 39). De fato, isso não é algo novo, satélites de médio e grande porte também utilizam-se dessas plataformas para avaliar sua funcionalidade durante o desenvolvimento (15), alguns exemplos são o *Lunar Reconnaissance Orbiter* (40), concebido pela NASA, e o satélite *Hope* desenvolvido pelo *Mohammed Bin Rashid Space Centre* (41). No Brasil, os autores de (42) adotaram uma FlatSat para testar um subsistema de comunicações desenvolvido para a Plataforma Multi-Missão (PMM) do Instituto Nacional de Pesquisas Espaciais (INPE).

Entretanto, apesar de sua ampla adoção e os benefícios ao desenvolvimento das missões apresentadas, tem-se pouca literatura acerca dessas plataformas por si só, sendo comum serem apresentadas como uma ferramenta utilizada, mas sem qualquer detalhe de implementação ou funcionamento. A fim de suprir essa lacuna na literatura, um mapeamento sistemático foi elaborado em (15). A pesquisa em uma tentativa de caracterizar as plataformas encontradas na literatura, teve como foco listar suas características principais e os testes realizados, categorizar as plataformas quanto a sua estrutura e tentar observar o seu uso ao longo dos anos.

Os autores do mapeamento sistemático, após considerarem as interfaces entre os subsistemas implementadas pelas plataformas, bem com a estrutura das próprias FlatSats, categorizaram as plataformas encontradas em quatro classes (Figura 5):

1) *Raw*: Plataformas onde os subsistemas são conectados utilizando chicotes elétricos e cabos, além disso elas podem ser montadas diretamente em cima de bancadas de testes. Alguns exemplos são de missões que as utilizaram são: Prox-1 (16, 43) e a FASTRAC (44, 45, 46);

2) *Bridge*: Plataformas que substituem todas as conexões via cabo por uma *Printed Circuit Board* (PCB), essa placa apenas conecta os subsistemas sem fornecer uma base para eles. Exemplos de missões que usaram esse tipo de FlatSat são: MOVE-II (47, 48, 49) e MIST (50, 51);

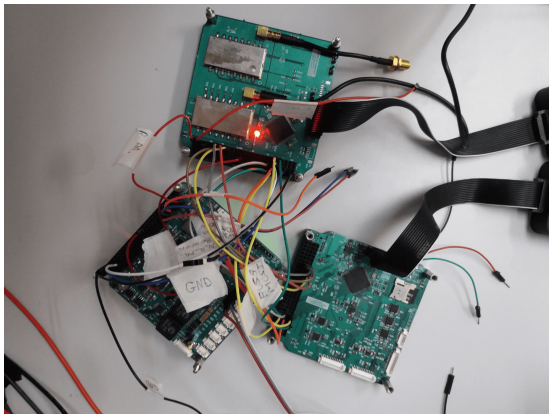
3) *Dock*: Plataformas que também utilizam uma PCB, semelhante as do tipo *bridge*, contudo elas oferecem uma base para conexão e integração dos subsistemas. Exemplos desse tipo de plataforma podem ser visto nas missões: Aalto-1 (36) e EIRSAT-1 (17);

4) *Modular*: Plataformas que utilizam PCBs modulares para cada subsistema e, quando conectados, permitem a interface entre subsistemas. Um exemplo desse tipo de plataforma é utilizado na missão ISTSat-1 (12).

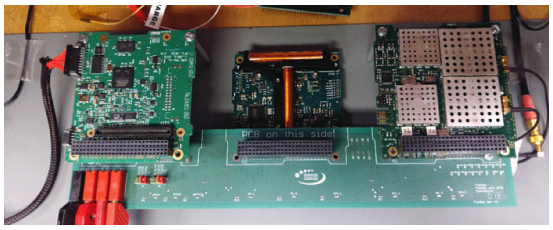
Dentre as plataformas apresentadas algumas se destacam por sua construção e suas funcionalidades embarcadas. A plataforma usada na missão MOVE-II (47, 48, 49), em particular, tem um *layout* de PCB simples e pode emular corrosão nas trilhas dos barramentos, efetivamente introduzindo injeção de falhas no barramento principal do satélite. Outra plataforma interessante é usada no projeto VST104 (52), a qual, notávelmente, permite o monitoramento dos barramentos de potência através de resistores *shunt*, sendo a única amostra de plataforma que embarca formas de sensoriamento na revisão conduzida em (15).

Adicionalmente, uma plataforma notável, apresentada em (53), propôs a ideia de se utilizar um *Complex Programmable Logic Device* (CLPD) a fim de se conceber um barramento reconfigurável. Essa abordagem permite que a plataforma seja utilizada entre múltiplas missões com poucas ou nenhuma alteração de *hardware*, tornando a FlatSat altamente reutilizável.

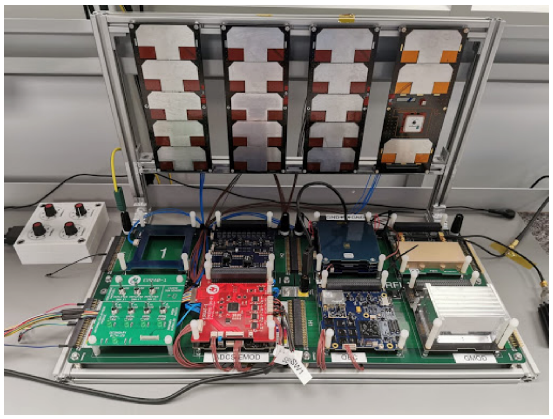
Figura 5 – Exemplos para cada classificação FlatSats



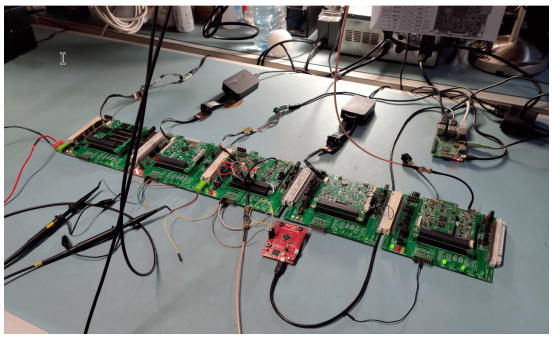
(a) *Raw*: FlatSat do FloripaSat-1



(b) *Bridge*: FlatSat da missão MIST



(c) *Dock*: FlatSat do EIRSAT-1



(d) *Modular*: FlatSat do ISTSat-1

Fonte: (a) Obtido de (20, pg. 28); (b) Obtido de (50, pg. 5); (c) Obtido de (17, pg. 7); (d) Obtido de (12, pg. 6).

Além disso, os autores de (15) fizeram uma análise sobre quais tipos de testes foram conduzidos utilizando essas plataformas. Em geral, as FlatSats foram utilizadas majoritamente para testes funcionais ou emulação de cenários de missão, o que está de acordo com o que foi visto na Seção 2.1. Ademais, o uso de técnicas de HIL — as quais podem aumentar a confiabilidade de missões enquanto reduzem tempo de desenvolvimento (54) — são frequemente usadas.

Ademais, algumas equipes utilizam FlatSats em testes não convencionais, evidenciando a versatilidade dessas plataformas. No desenvolvimento do ISTSat-1 (12), por exemplo, a FlatSat foi utilizada em testes de EMC/EMI, visto que aumentaria a exposição dos subsistemas à radiação eletromagnética. Além disso, essa plataforma utilizou-se de um EGSE para a realização de testes integração de forma contínua, nesses testes a EGSE foi usada para acessar e controlar variáveis de dados, enviar comandos, configurar subsistemas e registrar eventos ocorridos. Adicionalmente, essa EGSE realiza verificações para evitar erros de regressão, ou seja, a cada nova alteração no software de um subsistema um teste automático é feito, reportando se houve

algum erro novo decorrente da atualização.

Por fim, essas plataformas, em alguns casos, também foram utilizados em contexto de testes ambientais e térmicos. Na missão UWE-3 (55) uma FlatSat foi empregada para avaliar a dissipação de calor no subsistema de potência do satélite (*Electrical Power System* (EPS)). Outro exemplo dessa aplicação é a missão SOAREX-8 (56), a qual utilizou uma plataforma para analisar o comportamento do sistema em um câmara de termo-vácuo. A Tabela 2 compara algumas plataformas consideradas relevantes de acordo com suas funcionalides embarcadas.

Tabela 2: Comparação entre FlatSats e suas funcionalidades

Plataforma	Inclui sensoriamento	Possui reconfiguração	Injeção de falhas embarcada	Inclui processador embarcado	Ref.
ISTSat	✗	✗	✗	✗	(12)
VST104	✓	✗	✗	✗	(52)
MOVE-II	✗	✗	✓	✗	(47, 48, 49)
BIRDS-3	✗	✓	✗	✗	(53)
MIST	✗	✗	✗	✗	(50, 51)

Fonte: Adaptada de (15).

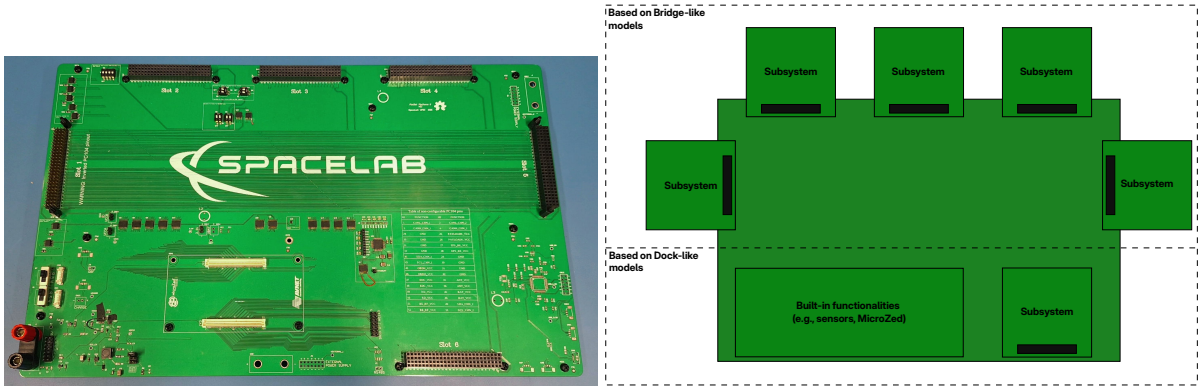
2.2.2 Plataforma FlatSat2.0

Contudo, o mapeamento sistemático feito em (15) evidencia diversas limitações nas plataformas analisadas, as quais, em sua maioria, apenas interfaceiam eletricamente os subsistemas, com pouca ou nenhuma funcionalidade embarcada; o que é evidenciado na Tabela 2. Dessa forma, tarefas mais complexas como injeção de falhas, emulação de subsistemas e sensoriamento tornam-se dependentes de equipamento externo — tais como osciloscópios, analisadores lógicos e computadores — o que eleva a complexidade e reduz a reusabilidade do ambiente de testes.

Devido a experiência positiva com FlatSats nas plataformas anteriores do SpaceLab (18, 57), bem como ao conhecimento obtido na concepção do mapeamento sistemático, o autor de (20) desenvolveu uma nova plataforma, buscando abordar as limitações observadas em outras plataformas. Para tal, a FlatSat inclue, entre outras funcionalidades, sensores embutidos, injeção de falhas e um SoM com lógica reprogramável, ampliando assim a capacidade da plataforma em executar tarefas complexas e auxiliar missões futuras do SpaceLab. A Figura 6 mostra um diagrama representando a estrutura da plataforma, bem como uma foto da mesma.

Adicionalmente, percebe-se que a plataforma consegue integrar até 6 subsistemas ao mesmo tempo, dos quais cinco tem ambos os lados expostos, isso sendo possível dado o uso da abordagem do tipo *bridge*, a qual é descrita em (15). Além disso, nota-se que a plataforma

Figura 6 – Plataforma FlatSat2.0



(a) Foto da plataforma (b) Diagrama da plataforma

Fonte: (a) Tirada pelo Autor (2026); (b) Adaptado de (20).

utiliza conectores do tipo PC/104 para a integração de subsistemas, os quais são utilizados frequentemente como o barramento principal de CubeSats, sendo presente tanto em módulos comerciais como em módulos feitos em universidades.

As subseções seguintes detalham as funcionalidades integradas à plataforma, destacando sua utilidade na realização de testes.

2.2.2.1 MicroZed

A fim de possibilitar a realização testes automatizados, bem como controlar e monitorar o restante das funcionalidades embarcadas da plataforma, um SoM — que atua como um processador embarcado — foi integrado à FlatSat2.0. O SoM escolhido foi a MicroZed (58), que incorpora um SoC da familia Zynq-7000 da AMD-Xilinx, memórias, interfaces *ethernet*, *Universal Serial Bus* (USB) e cartão SD, bem como mais de uma centena de pinos I/O reprogramáveis. O SoC utilizado é composto por dois domínios: *Processing System* (PS) e *Programmable Logic* (PL), sendo que o primeiro engloba dois processadores ARM A9 e seus periféricos, já o segundo é constituído por recursos de lógica programável equivalentes aos de um *Field-Programmable Gate Array* (FPGA).

A FlatSat2.0 se aproveita da reprogramabilidade do SoC para viabilizar o uso da plataforma para multiplas missões e conjuntos de subsistemas. Os pinos de I/O reprogramáveis foram conectados ao barramento principal do satélite, tendo interface com os subsistemas através dos conectores PC/104. Adicionalmente, a fim de abordar a possibilidade de diferentes níveis de tensão serem utilizados entre missões, tradutores de tensão bidirecionais foram adicionados entre a MicroZed e o PC/104.

Dada a versatilidade do SoM, diversos procedimentos que auxiliam a campanha de V&V podem ser realizados, alguns exemplos são: emulação de subsistemas e da estação de

solo, injeção de falhas e testes do tipo HIL. Além disso, a MicroZed auxilia o monitoramento do sistema em testes de longa duração, permitindo testes autônomos que podem durar horas ou dias. Nesse contexto, todos os dados coletados (telemetria, leituras dos sensores, etc.) podem ser armazenados, junto a uma marcação temporal, em um banco de dados local, garantindo um registro estruturado e persistente, facilitando a análise posterior aos testes e a integração com ferramentas padrão de ciência de dados.

Ademais, acesso remoto a plataforma pode ser implementado com a interface de rede, utilizando-se de ferramentas populares como, por exemplo, *Secure Shell* (SSH), permitindo aos operadores supervisionar e controlar os procedimentos em tempo real de qualquer lugar.

2.2.2.2 Sensoriamento

A plataforma conta com 7 sensores de corrente embarcados, os quais medem o consumo individual dos canais de potência expostos através do barramento principal. O sensor utilizado foi o INA219 (59), o qual é capaz de fornecer medições de corrente, tensão e potência diretamente à MicroZed por meio de uma interface *Inter Integrated Circuit* (I^2C). Os sensores possibilitam a avaliação do consumo de energia de cada subsistema, bem como do sistema como um todo sem depender de telemetria proveniente dos módulos. Além disso, em configurações mais complexas, eles poderiam ser usados para estimar a eficiência do EPS.

Ademais, a FlatSat2.0 inclui dois sensores de temperatura distintos, sendo eles o TMP112 (60) e o LTC2983 (61), os quais são, respectivamente, um sensor de temperatura ambiente e um sensor de temperatura baseado em termopares. Ambos os sensores são controlados através da MicroZed, sendo que o TMP112 compartilha o barramento I^2C com os sensores de corrente e o LTC2983 utiliza uma interface *Serial Peripheral Interface* (SPI) dedicada. O sensor de temperatura ambiente pode ser utilizado como referência em testes ambientais, nos quais o sistema é submetido a mudanças de temperatura. Por outro lado, o LTC2983, através do uso de termopares, pode ser empregado para monitorar mudanças de temperatura em áreas sensíveis do sistema, como amplificadores de potência, conversores ou aquecedores.

Com a adição desses sensores a plataforma oferece uma capacidade de obtenção de dados superior as FlatSats que apenas conectam eletricamente os subsistemas, as quais dependem de equipamento externo ou de telemetria dos próprios subsistemas para a coletar as mesmas informações. Além disso, a abordagem tradicional do uso de equipamento externo também pode ser utilizada em conjunto com os sensores embutidos, ampliando a capacidade de monitoramento da plataforma.

2.2.2.3 Injeção de Falhas

A FlatSat2.0 também permite a injeção de falhas no barramento principal do satélite, sendo possível realizar isso atráves tanto de *hardware* como de *software*. Para falhas geradas por *hardware*, emprega-se *switches* a fim de emular falhas físicas no barramento *Controller Area Network* (CAN). As falhas implementáveis são: i) curto circuito entre as linhas do barramento, ii) curto circuito entre as linhas do barramento e o terra, iii) ruptura de uma das linhas do barramento e iv) perda ou adição do resistor de terminação.

Adicionalmente, falhas geradas por software podem ser implementadas através do uso da MicroZed. Nesse sentido, diversas opções tornam-se disponíveis, a exemplo da injeção de dados corrompidos nos barramentos de comunicação ou forçar determinadas linhas dos barramentos para um estado fixo. Além disso, também é possível realizar testes de estresse e caracterização dos barramentos de comunicação dos subsistemas, de modo similar ao que foi feito por meio de equipamento externo em (12).

2.3 LINUX EMBARCADO

Esta seção aborda o conceito de Linux embarcado, bem como seu uso em nanossatélites, servindo de contextualização para sua menção ao longo do trabalho. Contudo, detalhes de implementação ou funcionalidades específicas sobre os sistemas Linux são apresentadas em conjunto com sua utilização no trabalho. Diante disso, em seguida, tem-se uma apresentação sobre esse tema.

O termo Linux, apesar de ser comumente associado a um Sistema Operacional (OS), se refere, na verdade, a kernel de um OS concebida por Linus Torvalds em 1991. A kernel, ao longo dos anos, incluiu suporte a uma grande variedade de arquiteturas diferentes, bem como possui um rico ecossistema de ferramentas de desenvolvimento, implementações de protocolos de comunicação, *drivers*, entre outras ferramentas, tornando-a atrativa para o uso em sistemas embarcados (62). Além disso, o Linux é utilizado em uma vasta gama de dispositivos, dos processadores embarcados mais simples aos supercomputadores e servidores de alto desempenho, sendo que, quando usado em sistemas embarcados, emprega-se frequentemente o nome de Linux embarcado.

Para seu uso em sistemas embarcados, existem diversas distribuições — conjunto de aplicações e ferramentas que, ao combinadas com a kernel, definem um OS Linux — comerciais disponíveis, muitas vezes derivadas de distribuições usadas em servidores ou computadores de uso pessoal; contudo, a construção de distribuições customizadas também é uma abordagem utilizada (62). Nesse contexto, o uso de ferramentas como Buildroot (63) e o Yocto Project (64) para a construção de um sistema Linux customizado é comum, as quais são adaptadas e

otimizadas para a aplicação para qual serão utilizadas.

No contexto da indústria aeroespacial, o autor da revisão feita em (62) demonstra que a adoção de sistemas baseados em Linux acontece a pelo menos uma década. De fato, seu uso, segundo as informações compiladas pelo autor, pode trazer diversos benefícios, como suporte para múltiplas plataformas e protocolos, ferramentas de desenvolvimento padronizados, maior comunidade de desenvolvedores e o enorme ecossistema de *software* disponível. Esses benefícios tornam a adoção de Linux em sistemas aeroespaciais, em espacial aqueles que empregam componentes COTS, algo atrativo.

Em CubeSats, múltiplas missões utilizam distribuições de Linux embarcado em seus subsistemas. Na missão Aalto-1 (36), por exemplo, a equipe desenvolveu um computador de bordo — subsitema frequentemente nomeado, em inglês, como *On-Board Computer* (OBC) — redundante que utiliza Linux para implementar os serviços e operações da missão, se aproveitando do ecossistema de *software* (65). Outro exemplo pode ser visto na missão HYPSO-1 (38), onde, devido a restrições relacionadas a uma câmera COTS se utilizou Linux embarcado em sua carga útil (66).

Nesse contexto, também impulsionados pela adoção do Linux, a classe de SoCs que incorporam lógica programável as capacidades tradicionais de SoCs, ganham relevância. Nesse sentido, uma das vantagens desses SoCs é que blocos de Propriedade Intelectual (IP) e lógica customizada podem ser utilizados para implementar funcionalidades de tempo real, algo que o Linux não suporta nativamente (67).

No contexto de CubeSats, esses SoCs são frequentemente utilizados devido sua flexibilidade e capacidade de processamento, sendo empregados tanto em OBCs, como detalhado em (68), como em cargas úteis a exemplo da carga útil apresentada em (66). Esses módulos muitas vezes empregam Linux, tornando-se mais uma evidência de seu uso nesse tipo de satélites.

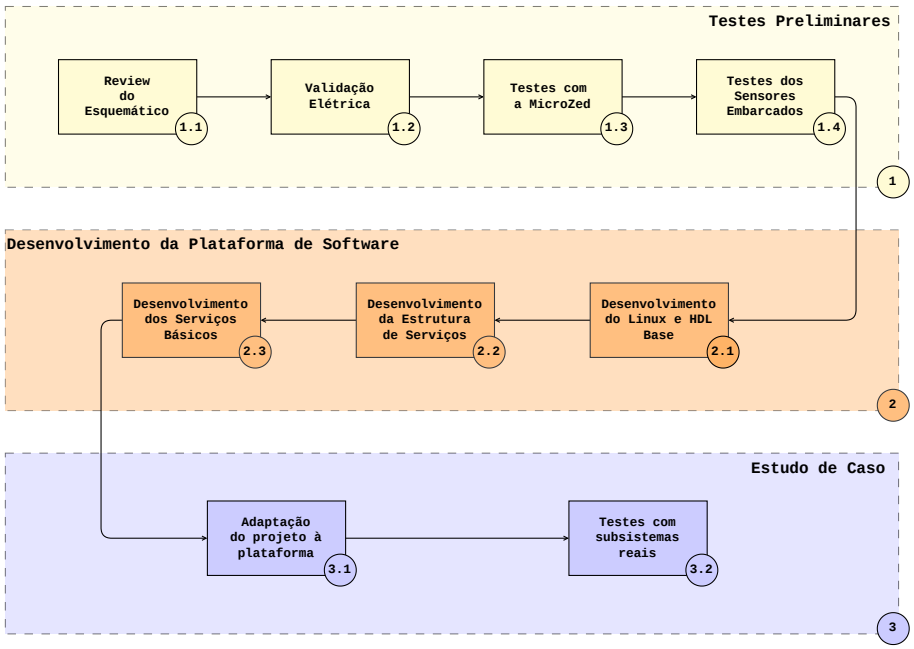
Diante do que foi exposto, percebe-se que a utilização de Linux para sistemas embarcados é uma prática comum, inclusive na indústria aeroespacial. Sendo empregado a fim de facilitar o desenvolvimento e teste de *software*, se aproveitando de seu amplo ecossitema de ferramentas de desenvolvimento, além de auxiliar na reutilização de componentes de *software* em arquiteturas de *hardware* distintas.

3 METODOLOGIA

Embora a concepção e a fabricação da plataforma proposta em (20) terem sido realizadas há algum tempo, ainda não há uma validação completa de suas funcionalidades, apesar dos esforços realizados em (21). Portanto, faz-se necessário uma validação mínima da plataforma antes de qualquer teste ou procedimento com subsistemas reais, dado a possibilidade de danificar os subsistemas. Adicionalmente, essa validação também é utilizada para verificar se os circuitos e funcionalidades implementados funcionam corretamente.

Desse modo, o processo de desenvolvimento e validação da FlatSat2.0 pode ser dividido em três grandes partes: i) Testes Preliminares, ii) Desenvolvimento da Plataforma de *Software* e iii) Estudo de Caso. O diagrama visto na Figura 7 demonstra o processo realizado, bem como as atividades empregadas em cada parte dele.

Figura 7 – Metodologia para o desenvolvimento e validação da FlatSat2



Fonte: Elaborado pelo Autor (2026).

As subseções a seguir detalham os passos realizados em cada etapa do processo de desenvolvimento e validação da plataforma.

3.1 TESTES PRELIMINARES

O prodecimento de teste e verificação adotado na etapa inicial se baseia nos padrões ECSS-E-ST-10-03C (29) e ECSS-E-ST-10-02C (31) definidos pela ECSS; em particular, os seguintes metódos de verificação foram utilizados ao longo do processo: Revisão de Design, Inspeção e Teste. Na sequência, o detalhamento das atividades a serem realizadas nessa etapa é apresentado.

As atividades se iniciam com uma revisão do esquemático da plataforma, a fim de garantir que não há nenhum circuito ou conexão equivocadas na implementação da plataforma. Além disso, essa etapa serve também para verificar que a MicroZed consegue realizar as atribuições esperadas pelo roteamento. Isso torna-se relevante devido ao fato que o SoM expõe alguns pinos do PS — denominados de *Multipurpose I/O* (MIO) — os quais não tem a mesma configurabilidade daqueles conectados ao PL (69).

Em seguida, caso não seja detectada nenhuma falha crítica no esquemático, segue-se com a validação elétrica da plataforma, a qual cobre testes de continuidade, inspeção visual das trilhas e da solda dos componentes da PCB, entre outras atividades. Ademais, quaisquer ajustes, adaptações ou retrabalhos menores que forem necessários — detectados nessa etapa ou na anterior — são realizados nesse momento.

Após essa etapa, uma breve validação da MicroZed é realizada, tendo como objetivo verificar se o SoM está funcional, bem como obter um primeiro contato com o *hardware* e as ferramentas de desenvolvimento. Nesse sentido, um projeto básico para o PL e uma pequena distribuição Linux são preparados utilizando o *Board Support Package* (BSP) — pacote de arquivos que descrevem os componentes e configurações de *hardware* de uma placa — disponibilizado pela fabricante da MicroZed.

Por fim, os sensores embarcados são testados através do uso da MicroZed, responsável por configurar e ler suas medições. Além disso, sensores externos podem ser utilizados para comparação e validação. Nesse ponto, a integração de subsistemas de CubeSats na plataforma já é possível e, portanto, pode-se realizar um primeiro teste preliminar com eles, a fim de monitorar o comportamento das funcionalidades da FlatSat com esses módulos presentes.

3.2 DESENVOLVIMENTO DA PLATAFORMA DE SOFTWARE

Com o *hardware* da plataforma agora validado, segue-se com o projeto e desenvolvimento da plataforma de *software* e descrição de *hardware* utilizada como base para as funcionalidades implementadas na MicroZed. Esse projeto deve fornecer aplicações, bibliotecas e blocos IP mínimos para que os componentes do SoM e aqueles embutidos na FlatSat2.0 funcionem

perfeitamente. Além disso, o projeto deve ser desenvolvido tendo em vista reutilização, extensão e personalização, uma vez que a FlatSat deve ser capaz de suportar múltiplas missões.

Em sequência, uma composição de bibliotecas, protocolos e ferramentas são desenvolvidas para facilitar a concepção, controle e comunicação entre serviços de *software*, os quais implementam funcionalidades específicas — a exemplo de monitoramento dos sensores embarcados — promovendo uma arquitetura de *software* modular e flexível (66, 70). A arquitetura do tipo *Service-Oriented Architecture* (SOA), quando combinada com sistemas de mensagens, possibilita a adição ou remoção de serviços/componentes com pouca ou nenhuma alteração no sistema (70), característica desejável para a FlatSat2.0, dado sua natureza multimissão.

Por último, implementa-se os primeiros serviços básicos da plataforma, a exemplo dos serviços de controle e leitura dos sensores embarcados e emulação de subsistemas, obtendo-se, finalmente, a capacidade de realizar testes automatizados de longa duração.

3.3 ESTUDO DE CASO

A fim de concluir o processo de validação, realiza-se um estudo de caso integrando uma plataforma de CubeSat (o termo *bus* também é comumente utilizado) com a FlatSat, a fim de se avaliar a capacidade dela em conduzir testes de longa duração, coletar dados e/ou telemetria dos subsistemas, emular módulos e auxiliar na campanha de V&V de missões.

O *bus* escolhido para o estudo de caso foi o FloripaSat-2 (18), plataforma de código aberto desenvolvida pelo SpaceLab utilizando as lições aprendidas e herança de voo da plataforma FloripaSat-1 (19, 57, 71). Sua escolha deu-se devido ao fácil acesso aos módulos e ao *flight software* dos subsistemas, facilitando a emulação de subsitemas, além de aumentar a fidelidade da emulação.

O estudo de caso consiste na realização de dois testes de longa duração, sendo o primeiro utilizado para validar a coleta de dados dos sensores embarcados e telemetria dos subsistemas de forma simultânea; o segundo consiste na emulação de um cenário de missão, em particular, na emissão de comandos baseados nas condições orbitais do satélite, as quais foram simuladas com um propagador de órbita embarcado. Além disso, no segundo procedimento o monitoramento dos sensores e telemetria também são realizados, com a adição da medição de temperatura em locais específicos através da interface para termopares do LTC2983.

Os testes realizados no estudo de caso não só validam as funcionalidades embarcadas na FlatSat, como também os serviços e ferramentas desenvolvidos ao longo do trabalho. Finalmente, após essa etapa, pode-se declarar que a plataforma FlatSat2 está testada e validada, sendo possível emprega-lá em missões atuais e futuras do SpaceLab.

4 DESENVOLVIMENTO

Esse capítulo descreve o processo de desenvolvimento da infraestrutura de *firmware* para a MicroZed, servindo como base para o desenvolvimento de aplicações e testes automatizados mais complexos. Nesse sentido, de acordo com o método ilustrado na Figura 7, as atividades que compõem as etapas de 1 e 2 — testes preliminares e desenvolvimento da plataforma de software, respectivamente — são realizadas.

A fim de compartilhar o progresso do desenvolvimento e validação da plataforma FlatSat2.0, um artigo que descreve os testes preliminares foi escrito, o qual foi apresentado no *Latin American Cubesat Workshop 2024*, sendo disponibilizado no Apêndice A. O artigo cobre as atividades da primeira etapa da metodologia e, portanto, recomenda-se lê-lo para se ter acesso aos detalhes da execução dessa etapa, visto que a seguir tem-se apenas uma síntese dos resultados obtidos.

4.1 VALIDAÇÃO INICIAL

Inicialmente, uma revisão do esquemático e roteamento da plataforma foi conduzida. Nessa revisão, foram encontrados alguns problemas relacionados ao roteamento dos pinos de MIO do SoM e ao circuito de potência da plataforma. Em particular, foram descobertas inconsistências entre os protocolos de comunicação esperados pela FlatSat2.0 e os pinos de MIO do PS disponíveis para os controladores dessas funcionalidades. Esse descasamento impossibilita o uso dos controladores do PS para esses protocolos, contudo, foi possível cotornar esse problema através de um roteamento alternativo utilizando o PL.

Quanto ao circuito de potência, dois erros foram identificados, sendo problemas relacionados ao circuito de alimentação da MicroZed, especificamente com os sinais *POWER_EN* e *VCCIO_EN*, os quais tem a funcionalidade de habilitar os conversores de potência da MicroZed e sinalizar que os bancos de I/O do SoM estão energizados, respectivamente.

Ambos os problemas encontrados no circuito foram corrigidos com um pequeno retrabalho na PCB, realizado durante a etapa de validação elétrica. Ainda nessa etapa, diversos curtos circuitos foram encontrados nos componentes da placa durante o teste de continuidade, sendo derivados da solda manual dos componentes. Diante disso, foi necessário a remoção do

excesso do fluxo de solda e a substituição de alguns tradutores de tensão da placa.

Após a resolução desses problemas, os testes restantes foram concluídos sem maiores ocorrências. Tanto a validação dos sensores embarcados — com exceção do LTC2983, pois os termopares usados com ele não estavam disponíveis durante a realização dos testes — como da MicroZed foram alcançados, inclusive, foi possível utilizar subsistemas reais em um teste posterior com os sensores, sendo uma experiência relevante para o estudo de caso conduzido posteriormente.

4.2 DESCRIÇÃO DE HARDWARE - ZYNQ PL

Finalizada a fase dos testes preliminares, segue-se com o processo de desenvolvimento, em particular com a descrição de *hardware* que é sintetizada e implementada no PL do Zynq. Para projetos que envolvam descrição de *hardware* de FPGAs e SoCs da AMD Xilinx, o *Vivado Design Suite*, ferramenta proprietária da empresa, deve ser utilizada. O *Vivado* incorpora funcionalidades que permitem configuração, síntese lógica, implementação e depuração da lógica programável.

O fluxo de desenvolvimento de projetos com o *Vivado* tem duas abordagens principais: *Project Mode* e *Non-Project Mode* (72). No *Project Mode* o *Vivado* constrói um diretório em disco para o projeto, gerenciando de forma automatizada os arquivos e o estado do projeto, além disso esse modo pode ser executado através tanto da interface gráfica como através do uso de scripts *tcl* (72). Por outro lado, o *Non-Project Mode* permite que o usuário tenha total controle do fluxo de síntese do design através de comandos *tcl*, além disso o projeto é construído em memória e qualquer relatório ou estado intermediário deve ser preservado manualmente. Ademais, esse modo pode fazer uso parcial da interface gráfica do *Vivado* (73).

Para este trabalho, optou-se pelo uso do *Project Mode* devido a sua simplicidade, gerenciamento automático de projeto e sua representação em disco. Todavia, adota-se o uso de scripts *tcl* por sua reprodutibilidade, extensibilidade e fácil integração com ferramentas de controle de versão, a exemplo do *Git*. A interface gráfica, apesar de ser intuitiva e conveniente, depende demasiadamente do gerenciamento automático do *Vivado*, o qual conflita com as ferramentas de controle de versão, em especial quanto ao controle de código fonte.

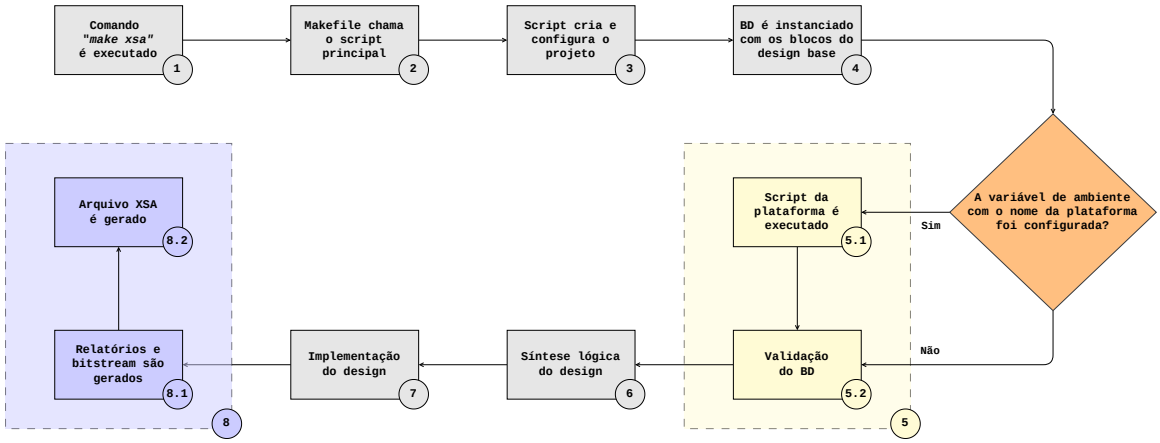
Diante disso, o projeto foi organizado em uma coleção de scripts *tcl* que constroem o projeto, rodam o processo de síntese e implementação, reportam relatórios, geram o *bitstream* e, por fim, exportam um arquivo *Xilinx Support Archive* (XSA) que descreve o *hardware* final. O fluxo de síntese utiliza intensamente o ambiente de *Block Design* (BD), o qual permite a integração de blocos IP através de suas interfaces, efetivamente estruturando um diagrama de blocos do sistema. O fluxo completo é apresentado na Figura 8, enquanto a organização do projeto pode ser visto na Listagem 1.

Listagem 1 – Organização de diretórios do projeto do PL

```
hdl/
|-- Makefile
|-- README.rst
|-- scripts
|   |-- bd.tcl
|   |-- designs
|   |-- gen_sdt.tcl
|   |-- main.tcl
|   `-- zynq_ps.tcl
|-- VERSION
`-- xdc
    |-- bitstream_compression_enable.xdc
    |-- flatsat2.xdc
    `-- template
        `-- mz7020_som_base.xdc
```

Tendo em vista a natureza multimissão e a necessidade de se adaptar a múltiplos subsistemas, os scripts preveem variações do design através de uma variável de ambiente que controla a execução de scripts específicos para cada plataforma. Esses scripts são organizados no subdiretório "*scripts/designs*", onde cada variação de design tem seu próprio diretório com um arquivo base denominado "*platform.tcl*", o qual é incluído no fluxo de síntese pelos scripts principais.

Figura 8 – Fluxo do projeto de descrição de hardware



Fonte: Elaborado pelo Autor (2026).

Desse modo, adicionar um novo design ou suporte para novos subsistemas torna-se um processo simples e definido, sendo necessário a confecção de scripts que adicionem a funcionalidade necessária e então chamá-los através do "*platform.tcl*". As configurações e

restrições de projeto também podem ser adicionadas da mesma forma, sendo integradas através dos scripts.

Evidencia-se, portanto, a flexibilidade do uso de scripts *tcl* para o gerenciamento de projetos no *Vivado*, os quais provém o suporte necessário para simplificar o desenvolvimento da descrição de *hardware* da plataforma.

A MicroZed, por ser um SoM, impõe algumas restrições ao projeto de descrição de *hardware*, a interface com as memórias *Double Data Rate* (DDR) e a configuração de pinos MIO são exemplos. No geral, essas configurações se limitam, essencialmente, a configuração do PS, a qual também é realizada através do *Vivado*. Por isso, a fim de garantir que as configurações estão coerentes, os valores adotados foram retirados diretamente do BSP do fabricante.

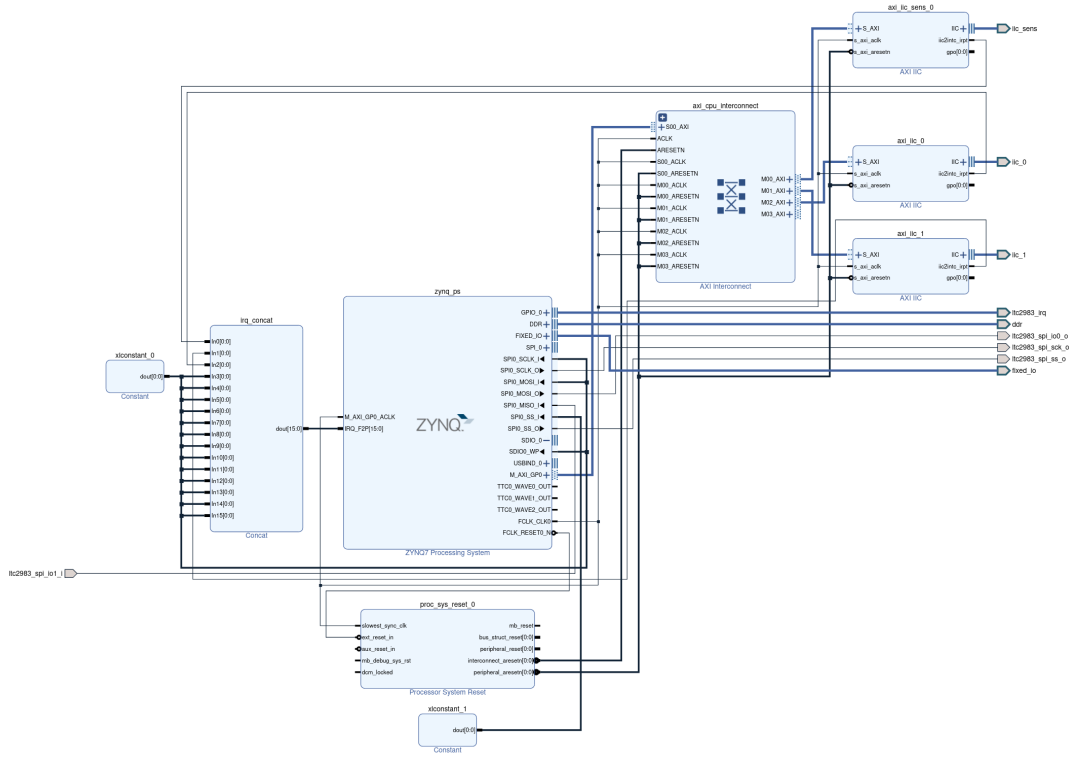
Em sequência, buscou-se implementar as adições mínimas necessárias para que todas as funcionalidades embarcadas da FlatSat2.0 fossem contempladas. Conforme comentado na subseção anterior, almeja-se utilizar o ambiente BD o tanto quanto possível, a fim de facilitar a integração dos componentes do sistema e se aproveitar do amplo catálogo de blocos IP da Xilinx.

A plataforma, inicialmente, necessitava implementar, por meio do PL, uma interface SPI e uma interface I^2C destinadas a comunicação com os sensores. Contudo, devido aos problemas de roteamento descritos anteriormente, duas novas interfaces I^2C se tornaram necessárias. As três interfaces foram sintetizadas utilizando o IP *AXI_IIC*, bloco oferecido pela Xilinx que implementa o protocolo I^2C e expõe seus registradores ao processador através de uma interface *Advanced eXtensible Interface* (AXI). Após a adição desses blocos, precisou-se configurar seus pinos de saída para que sejam mapeados corretamente na plataforma, sendo que um dos blocos faz interface com os sensores e os outros dois são roteados para o PC/104.

Por outro lado, a interface SPI, destinada para comunicação com o LTC2983, foi implementada utilizando o controlador de SPI do PS, roteando seus pinos pelo PL. Para que isso seja possível, o Zynq foi configurado para que essa interface fosse exposta via *Extended Multipurpose I/O* (EMIO), o qual, essencialmente, permite que periféricos do PS sejam utilizados em pinos de I/O conectados ao PL (69).

A Figura 9 mostra o diagrama de blocos do projeto base, enquanto a Tabela 3 descreve a utilização de recursos do PL. Ao se observar as informações da tabela percebe-se que o projeto consome pouquissímos recursos do SoC, inclusive nenhuma *Block Random Access Memory* (BRAM) ou blocos de *Digital Signal Processing* (DSP) foram usados. Diante disso, conclui-se que ainda há muitas possibilidades a serem exploradas para o PL da MicroZed, sendo que a implementação de novas interfaces de comunicação, IPs customizados ou até mesmo um analisador lógico são apenas alguns dos inúmeros exemplos práticos que se beneficiariam do uso de recursos do PL.

Figura 9 – *Block Design* para o projeto base



Fonte: Elaborado pelo Autor usando o *Vivado IDE* (2026).

Tabela 3: Uso de recursos do PL pelo design base

Recurso	**Utilização**	**Disponível**	**Porcentagem de Uso**
LUT	2776	53200	5,22%
LUTRAM	150	17400	0,86%
FF	3172	106400	2,98%
IO	11	125	8,80%
BUFG	1	32	3,13%

Fonte: Elaborada pelo Autor (2026).

4.3 ECOSSISTEMA DE SOFTWARE - ZYNQ PS

Para o *software* que executa nos processadores ARM do PS, após uma pequena análise, adotou-se uma distribuição Linux customizada. A escolha tem racional semelhante ao que é apresentado na Seção 2.3. Em síntese, foi uma escolha baseada no ecossistema, nas ferramentas e na praticidade do uso do Linux.

Além disso, para esse caso em particular, as ferramentas disponibilizadas pela Xilinx

facilitam consideravelmente a integração com as funcionalidades de FPGA. Nesse contexto, manter uma solução *bare metal*, com sistema operacional de tempo real ou até mesmo uma distribuição Linux comercial seria um pesadelo no cenário em que o *hardware* pode mudar completamente, inclusive em tempo de execução.

Diante disso, optou-se pelo *Petalinux* (74), ferramenta da AMD Xilinx que extende o Yocto Project (64) com scripts, utilitários e configurações a fim de facilitar o desenvolvimento de sistemas Linux para SoCs da Xilinx. Um recurso notável dessa ferramenta é gerar automaticamente os arquivos necessários para integrar a descrição de *hardware* e, portanto, os blocos IP, com o Linux a partir do arquivo XSA gerado no *Vivado*.

Adicionalmente, a fabricante da MicroZed fornece seu BSP especificamente para o *Petalinux*, facilitando ainda mais a customização do Linux quando se compara com as outras soluções mencionadas. Além disso, o BSP contém os metadados necessários para a configuração dos periféricos incluídos na PCB do SoM. Consequentemente, apenas as adaptações quanto aos sensores e periféricos da própria FlatSat2.0 são requeridas.

Nesse contexto, utilizou-se a *layer* padrão gerada pelo *Petalinux*, denominada de *meta-user*, para organizar os metadados relevantes na adaptação da imagem do Linux à plataforma. Por outro lado, o BSP incluí, além de configurações específicas para o *Petalinux*, uma *layer* — nomeada de *meta-avnet* — que contém os metadados relacionados à MicroZed. Diante desse cenário, para uma melhor compreensão do projeto, faz-se necessário descrever brevemente o conceito de *layer* no Yocto Project, bem como as vantagens desse modelo de organização.

Uma *layer* é um repositório que contém um conjunto de instruções logicamente relacionadas que informam ao sistema de construção (*build system*) *Open Embedded* quais ações devem ser feitas (75). Além disso, as *layers* permitem sobrescrever instruções e configurações previamente definidas, possibilitando a adaptação de *layers* de fabricantes e/ou da comunidade às necessidades específicas do projeto. Ademais, a separação lógica das informações em diferentes repositórios — como BSP e interface gráfica — favorece a modularidade, simplifica a reutilização e facilita futuras customizações e manutenções do sistema (75).

Portanto, através desse modelo, o projeto pode ser customizado e expandido por meio da adição de novas *layers*, sendo o metódo recomendado para incluir suporte a novas plataformas ao projeto. Essas *layers* podem ser mantidas junto aos metadados do projeto e, quando houver a necessidade, podem ser habilitadas na configuração do *build system*.

A organização atual do projeto, incluindo as suas *layers*, scripts auxiliares, entre outros arquivos e diretórios podem ser vistos na Listagem 2.

Listagem 2 – Organização de diretórios do projeto do *Petalinux*

```
1 emb-linux/
2 |-- Dockerfile
3 |-- hw-design
```

```
|   `-- flatsat2.xsa
|-- project-spec
|   |-- attributes
|   |-- configs
|   |-- hw-description
|   |-- meta-avnet
|   `-- meta-user
|-- README.md
|-- scripts
|   |-- build-image.sh
|   |-- dfu-flash.sh
|   |-- run-qemu.sh
|   `-- sync-time-usb.sh
`-- utils
    `-- 99-dfu.rules
```

Como comentado na Seção 4.3, as principais adições ao projeto são referentes ao suporte do *hardware* embarcado na FlatSat2.0. Os sensores embutidos, por exemplo, tem *drivers* disponíveis na kernel, contudo eles não são habilitados por padrão, sendo necessário criar uma *receita* — arquivo que descreve os metadados de um único componente de *software* (75) — para sobrescrever a configuração original da kernel. Os *drivers* selecionados fazem parte do subsistema de *Industrial I/O* (IIO) da kernel, o qual provê suporte para dispositivos que realizam algum tipo de conversão analógico-digital ou digital-analógico, incluindo sensores de temperatura, giroscópios, entre outros dispositivos (76).

Após habilitar os *drivers*, torna-se necessário informar a kernel que os sensores estão presentes nos barramentos de I^2C e SPI, para tal, a *device-tree* é frequemente utilizada. A *device-tree* é uma estrutura de dados e linguagem utilizada para descrever *hardware*, sendo estruturada como uma árvore de nó nomeados, onde cada nó pode conter diversas propriedades (77). No contexto do projeto, o *Petalinux* popula a *device-tree* automaticamente com os nós referentes ao projeto do PL e, portanto, para adicionar os sensores basta criar os nós que os descrevem dentro do nó do controlador de seus respectivos barramentos. A Listagem 3 mostra a configuração da *device-tree* para alguns dos sensores do barramento I^2C.

Listagem 3 – Device-tree dos sensores presentes no barramento I^2C

```
/include/ "system-conf.dtsi"
/include/ "avnet-bsp.dtsi"

&axi_iic_sens_0 {
    status = "okay";
    clock-frequency = <100000>;
    #address-cells = <1>;
```

```
8       #size-cells = <0>;
9
10      ina219@40 {
11          compatible = "ti,ina219";
12          reg = <0x40>;
13          shunt-resistor = <47000>;
14          label = "h1-45_46";
15      };
16
17      ina219@46 {
18          compatible = "ti,ina219";
19          reg = <0x46>;
20          shunt-resistor = <47000>;
21          label = "h2-27_28";
22      };
23
24      tmp112@48 {
25          compatible = "ti,tmp112";
26          reg = <0x48>;
27          #thermal-sensor-cells = <1>;
28          label = "ambient-temp";
29      };
30  };
```

Adicionalmente, algumas configurações e utilitários foram incluídos na distribuição Linux a fim de facilitar o processo de desenvolvimento ou prover alguma depedência comum a serviços futuros. Pode-se citar como exemplo a configuração do *Device Firmware Update* (DFU), o qual permite a atualização de *firmware* por USB através de comandos no *bootloader*. Além disso, tem-se também o suporte para *device-tree overlays* — os quais buscam modificar a *device-tree* em uso, alterando, em tempo de execução, os componentes de *hardware* vistos pela kernel.

Dito isso, ao fim dessa etapa do desenvolvimento, obtém-se um sistema Linux e um projeto de descrição de *hardware* funcionais, servindo como base para a implementação dos procedimentos e testes complexos idealizados na concepção da FlatSat2.0.

4.4 SERVIÇOS E CONECTIVIDADE

Para organização e estrutura das ferramentas e aplicações de *software* adotou-se a arquitetura SOA, dada sua modularidade e suporte para um processo de desenvolvimento incremental (70). Nesse contexto, as aplicações são implementadas de forma independente provendo um serviço ou funcionalidade específica, a exemplo de controlar os sensores embarcados

ou uma interface de comunicação. Além disso, essas aplicações, doravante denominadas também de serviços, também podem integerir entre si ou com o usuário através de protocolos de comunicação bem definidos.

Contudo, para que a implementação e desenvolvimento dessa estrutura seja padronizada e escalável, um conjunto de ferramentas torna-se necessário para interação e controle desses serviços. Um exemplo básico dessa infraestrutura é uma forma padronizada de iniciar, reiniciar, pausar e parar os serviços. Ademais, uma forma de descrever os serviços, suas dependências e sua ativação também torna-se útil. Por fim, a padronização da comunicação entre serviços, também chamado de *software bus* (68), facilita a reutilização de código e ferramentas, escalabilidade do sistema e facilita o desenvolvimento.

Para o controle dos serviços e sua execução pode-se utilizar o *init system systemd* (78). O *systemd* é um *init system* e gerenciador de serviços utilizado em sistemas Linux, responsável pela inicialização do OS, gerenciamento de serviços, montagem de sistemas de arquivos e controle de dependências entre unidades de execução. Ele foi escolhido em detrimento de outras opções devido a quantidade de funcionalidades disponíveis, bem como a familiaridade com seu funcionamento.

Por outro lado, a solução quanto à comunicação entre serviços foi a utilização do modelo *Publish-Subscribe* (PUB/SUB), em particular, a implementação da biblioteca de mensagens *ZeroMQ* (ZMQ) (79). Nesse modelo, um *subscriber* pode se inscrever em um ou mais tópicos, expressando interesse em um determinado evento, e, subsequentemente, passa a receber eventos, gerados por *publishers*, que correspondem com os tópicos previamente registrados (80).

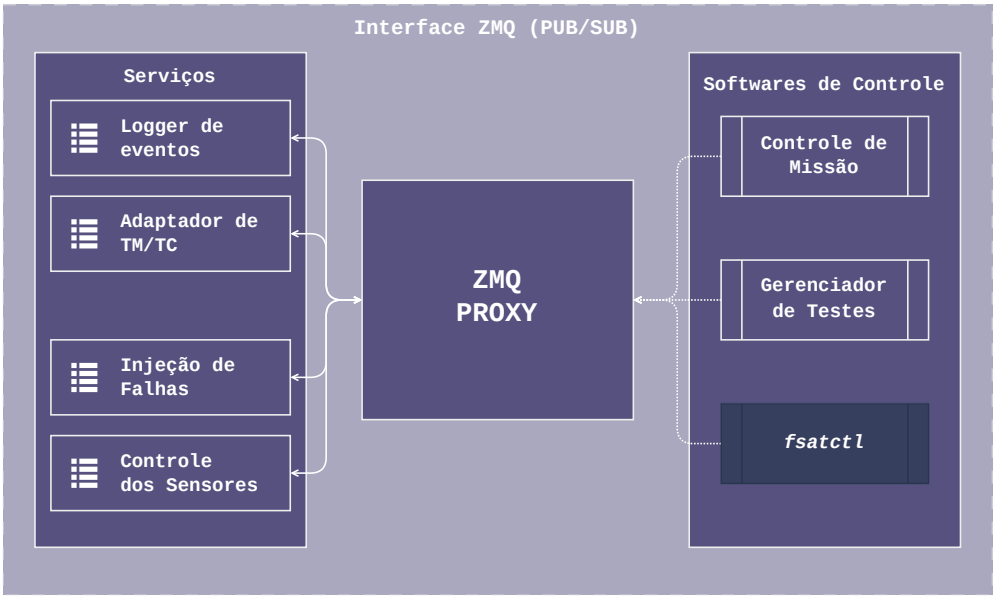
Nesse contexto, o modelo promove um desacoplamento entre os produtores e consumidores das mensagens, visto que os *publishers* publicam mensagens ou eventos sem saber quantos *subscribers* estão conectados ao barramento, de modo similar, *subscribers* não necessitam de informações quantos aos *publishers* para receber os eventos publicados (80). Essa característica permite que componentes sejam adicionados, removidos ou substituídos frequemente sem que o sistema como todo seja afetado (70, 80).

O desacoplamento mencionado é alcançado, normalmente, através de um *software bus*, nesse caso também nomeado de *event service*, o qual propaga os eventos ou mensagens publicados a todos àqueles que estão subscritos neles (80). Nesse sentido, esse componente também pode ser considerado como um *proxy* (80) e, no contexto da FlatSat2.0, a implementação utilizada é o *ZMQ Proxy*.

Esse *proxy* permite que os serviços da plataforma se inscrevam, de forma independente, em tópicos relevantes, expondo uma interface de controle e notificação. Nesse contexto, *softwares* de controle de missão ou gerenciadores de testes podem interagir e controlar os serviços por meio do *proxy*. Essa solução permite, também, o acesso ao barramento através da rede, possibilitando a interação remota entre os *softwares* de controle com as aplicações da

FlatSat2.0. A Figura 10 ilustra um exemplo de como essa infraestrutura pode ser utilizada.

Figura 10 – Exemplo de uso da infraestrutura para comunicação com os serviços



Fonte: Elaborado pelo Autor (2026).

4.4.1 Protocolo de Comunicação com Serviços

A fim de facilitar o reuso e desenvolvimento de *software*, um protocolo de mensagens simplificado foi concebido. Esse protocolo requer que os serviços conectados ao *proxy* se subscrevam em pelo menos dois tópicos: o primeiro sendo o seu próprio nome; o segundo é o tópico "*disc*" usado para descoberta dinâmica dos serviços. Além disso, o protocolo não impede aos serviços que se inscrevam em outros tópicos.

As mensagens trocadas pelos *publishers* e os serviços utilizam a implementação de mensagem de múltiplas partes do ZMQ, a qual possibilita o envio de múltiplas mensagens de uma só vez, garantindo que será entregue aos *subscribers* ou todas as partes ou nenhuma parte. O protocolo utiliza a seguinte composição de mensagens: i) o tópico, ii) um cabeçalho iii) o contéudo propriamente dito.

Atualmente, por padrão, dois tipos de mensagens podem ser enviados, sendo eles: descoberta e comando; o primeiro utiliza o tópico "disc" e funciona como um *broadcast* para os serviços, requisitando informações como versão da aplicação, nome e comandos disponíveis. Por outro lado, a mensagem do tipo comando utiliza o tópico com o nome do serviço e envia um comando diretamente para ele; o conteúdo do comando, incluindo seus argumentos, utiliza *JavaScript Object Notation* (JSON) como formatação.

Nesse contexto, uma biblioteca que implementa esse protocolo, nomeada de *fsatutils*, foi escrita em C++. Dessa forma, serviços podem utilizar a biblioteca diretamente para implementar a interface ZMQ. Além disso, uma aplicação de linha de comando chamada *fsatctl* foi desenvolvida de modo a implementar a funcionalidade de um *publisher*. Essa aplicação permite que, através do console serial da MicroZed ou de SSH, seja possível enviar comandos aos serviços. A *fsatctl* pode ser utilizada para depuração e controle dos serviços diretamente da linha de comando, sem a necessidade do uso de um *software* de controle completo, facilitando assim a interação com os serviços. A Listagem 4 mostra um exemplo de seu uso.

Listagem 4 – Exemplo do envio de um comando ao serviço *read-sensors* através da aplicação *fsatctl*

```
1 fsatctl command -s read-sensors set_measurement_period \
2 --arg=period_ms:100
```

4.4.2 Sistema de Monitoramento

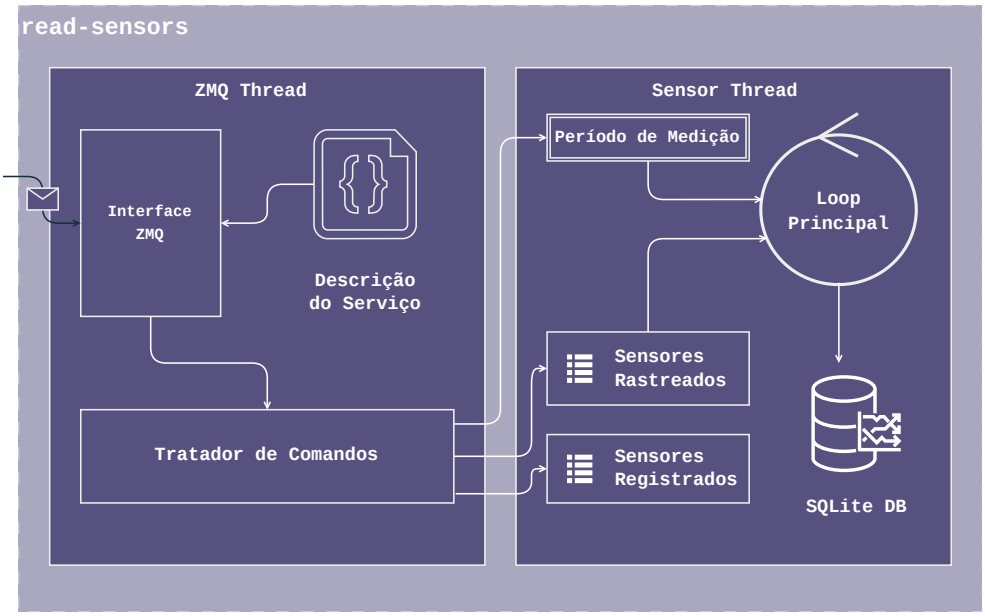
O primeiro serviço implementado foi o serviço nomeado de *read-sensors*, cuja função consiste no controle e monitoramento dos sensores embarcados na plataforma. Para fazer interface com o sensores, esse serviço utiliza a biblioteca *libiio*, a qual foi criada para se controlar sensores do subsitema IIO de forma programática. O serviço, por meio da *libiio*, oferece suporte aos seguintes dispositivos INA219, TMP112, LTC2983 e o XADC, este último incorpora um sensor interno ao SoC que monitora sua temperatura e tensões de alimentação (81).

Esse serviço foi programado em C++ e permite que múltiplos sensores de um único tipo sejam instanciados, sendo diferenciados pelo seu nome, o qual é descrito na propriedade *label* na *device-tree*. Além disso, para sensores que contém canais múltiplos, a exemplo do LTC2983 e XADC, canais individuais podem ser monitorados. Dito isso, os sensores podem ter dois estados dentro da aplicação: registrado e rastreado; esses estados correspondem, respectivamente, a sensores que o serviço registrou e a sensores que estão sendo monitorados ativamente. Um sensor rastreado necessariamente precisa ser registrado primeiro e, para sensores com funcionamento baseado em canais, canais podem ser rastreados desde que mapeados a um sensor registrado.

A aplicação monitora os sensores com o auxílio de duas listas internas, uma com os sensores registrados e outra com aqueles que estão sendo rastreados. As listas utilizam polimorfismo para suportar as implementações diferentes de cada tipo de dispositivo. Desse modo, para o suporte de novos sensores basta a confecção de uma nova implementação que herda da classe base.

O serviço é estruturado em dois blocos: o primeiro responsável pelo laço que monitora os sensores; o segundo implementa a interface ZMQ descrita na Seção 4.4. A tarefa que obtém

Figura 11 – Diagrama de blocos do serviço *read-sensors*



Fonte: Elaborado pelo Autor (2026).

as medições dos sensores é, essencialmente, um laço que executa periodicamente, lendo as medições de todos os sensores e/ou canais rastreados e, em seguida armazena as leituras em um banco de dados SQLite. Por outro lado, a interface ZMQ foi desenvolvida utilizando a biblioteca *fsatutils* e permite que aplicações externas controlem algumas funcionalidades do serviço. A Figura 11 apresenta a estrutura interna do serviço, além disso uma lista dos comandos disponíveis e suas descrições pode ser vista na Tabela 4.

Tabela 4: Comandos do serviço *read-sensors* expostos via interface ZMQ

Comando	**Argumentos**	**Descrição**
Register	Nome do sensor; Classe de dispositivo	Instancia, configure e registra o sensor com o nome e classe provenientes dos argumentos do comando
Unregister	Nome do sensor	Remove o sensor com esse nome da lista de registrados
Track	Nome do sensor; Nome do Canal	Adiciona o sensor ou canal à lista de sensores rastreados
Untrack	Nome do sensor; Nome do Canal	Remove o sensor ou canal da lista de sensores rastreados

(Continua na próxima página)

Tabela 4: Comandos do serviço *read-sensors* expostos via interface ZMQ (continuação)

Comando	**Argumentos**	**Descrição**
Set Measurement Period	Periodo em milissegundos	Define o período da tarefa que lê as medições dos sensores rastreados

Fonte: Elaborada pelo Autor (2026).

Em termos de implementação, a aplicação também suporta acesso remoto aos sensores através do *network context* da *libiio*. Nesse configuração, a aplicação se conecta a um servidor embarcado, nomeado de *IIO Daemon* (IIOD), que encapsula as operações de acesso ao *hardware* através de uma comunicação via rede. Isso permite que a aplicação seja utilizada de forma transparente, como se estivesse localmente conectada aos sensores. Um exemplo de uso dessa funcionalidade é apresentado na seção a seguir.

4.4.3 Acesso Remoto e Interface de Rede

A MicroZed dispõe de uma interface *ethernet* que, somada as ferramentas e aplicações disponíveis no sistema Linux, permite o SoM se conectar a internet da mesma forma que um servidor ou computador pessoal. Desse modo, o acesso a plataforma pode ser realizado remotamente, possibilitando o monitoramento ou condução de testes de qualquer lugar. Esse recurso é particularmente interessante durante o processo de desenvolvimento, visto que novas funcionalidades podem ser desenvolvidas e testadas remotamente, sendo utilizado, com o suporte de equipamento externo, nas plataformas anteriores do SpaceLab (20).

Outro procedimento do acesso remoto que se pode utilizar é a realização de avaliações ou testes de longa duração, os quais podem durar dias ou até mesmo semanas. Nesse contexto, nem sempre é possível ter acesso as instalações de teste ou dispor de pessoal disponível durante todo o procedimento. Assim como no desenvolvimento remoto, esse procedimento foi realizado diversas vezes durante as campanhas de V&V da plataforma FloripaSat-2 (20, 18).

Ademais, além do controle dos testes através do barramento de serviços feito com ZMQ — descrito na Seção 4.4 — pode-se empregar o IIOD para controle e leitura dos sensores por interface de rede. Uma demonstração desse recurso foi realizada utilizando o *software* de *dashboards Grafana* (82). Nessa demonstração um *dashboard* capaz de apresentar os dados dos sensores embarcados na plataforma foi contruído. Os dados utilizados no *dashboard* são obtidos, em tempo real, diretamente do banco de dados mantido pela aplicação *read-sensors*. Ambos o *Grafana* e a aplicação dos sensores são executadas em um computador de desenvolvimento, conectadas à FlatSat2.0 através da rede. Um exemplo do *dashboard* é apresentado no estudo de caso do Capítulo 5.

Tendo sido apresentados os benefícios dessa funcionalidade, cabe comentar brevemente sobre as adições e configurações aplicadas ao sistema Linux para implementá-la. Nesse sentido, para fins de comunicação e depuração, foi empregado o *framework libcomposite* do subsistema *USB Gadget* da kernel, o qual permite criar uma interface de rede virtual sobre a conexão USB, permitindo a troca de dados utilizando protocolos de rede convencionais. Essa adição é particularmente útil quando não há acesso a uma interface *ethernet* ou quando se necessita de uma conexão ponto a ponto com um computador de desenvolvimento. Além disso, utilizou-se o *systemd-networkd* para a configuração das interfaces de rede e, portanto, foi necessário criar arquivos de configuração para cada interface.

4.4.4 Emulador de Subsistemas

Finalmente, esta seção aborda a capacidade e o processo de se emular subsistemas através da MicroZed, além disso um exemplo concreto de emulação é apresentado. Nesse sentido, descreve-se, inicialmente, as funcionalidades que permitem a emulação de uma vasta gama de subsistemas e interfaces de comunicação; na sequência apresenta-se a estrutura de uma aplicação que emula um OBC, elaborada para o estudo de caso do Capítulo 5.

A plataforma FlatSat2.0 foi estruturada de modo a conectar o SoM ao maior número de interfaces do barramento principal quanto possível. Nesse contexto, quase todos os pinos do PC/104 — com os canais de potência sendo uma das exceções — foram roteados para a MicroZed através de tradutores de tensão bidirecionais. Em particular, muitos desses pinos foram conectados ao PL e, portanto, podem ter várias atribuições.

Por isso, a MicroZed consegue emular essencialmente todas as interfaces de comunição mais simples, a exemplo de I^2C, SPI e *Universal Asynchronous Receiver / Transmitter* (UART). Protocolos mais complexos, que utilizam de *transceivers* como CAN e RS485 também são suportados, mas apenas em alguns conectores/pinos específicos. Dessa forma, através do uso do PL e blocos IP, pode-se implementar grande parte das interfaces presentes nos subsistemas de CubeSats, o que se torna extremamente relevante no contexto de V&V, em particular, em testes de integração.

Em termos de implementação de *software*, pode-se aproveitar os *drivers* e bibliotecas de Linux disponíveis para adaptar as rotinas de baixo nível dos subsistemas. Desse modo, o *flight software*, com pequenas alterações, pode ser testado em paralelo com o desenvolvimento do *hardware*. Além disso, esse procedimento pode ser utilizado para o teste isolado de um subsistema, no qual apenas o dispositivo em teste é integrado a FlatSat2.0 e todos os outros módulos são emulados na MicroZed, buscando verificar a interoperabilidade desse subsistema.

Para exemplificar esse procedimento, bem como para seu uso no estudo de caso, implementou-se um serviço que emula parcialmente as rotinas do *flight software* do *On-Board*

Computer and Data Handling 2.0 (OBDH2), subsistema que atua como OBC da plataforma FloripaSat-2. Nesse cenário, baseado no *flight software* para a missão GOLDS-UFSC, o serviço implementa as tarefas de controle e coleta de telemetria dos subsistemas de *Telemetry, Tracking and Command* (TTC) e EPS. Além disso, tarefas que implementam um propagador de órbita, controle dos aquecedores da bateria e coleta de dados de uma carga útil também foram implementadas. Mais detalhes sobre os subsistemas e configurações utilizadas nesse estudo de caso podem ser vistas no Capítulo 5.

Os dados coletados — seja telemetria, dados da carga útil ou posição — são armazenados em bancos de dados SQLite dedicados a cada tarefa, permitindo a subsequente análise dos eventos e resultados do teste. A Tabela 5 descreve as rotinas implementadas, provendo descrição e o período de execução de cada tarefa.

Tabela 5: Tarefas do *flight software* da missão GOLDS-UFSC emuladas pela MicroZed

Tarefa	**Período de execução**	**Descrição**
Telemetria do TTC	60 segundos	Coleta telemetria do subsistema TTC
Telemetria do EPS	60 segundos	Coleta telemetria do subsistema EPS
Determinação de Posição	60 segundos	Calcula uma estimativa da posição simulada do satélite, bem como estima se o satélite estaria em eclipse
Controle da carga útil	60 segundos	Controla o estado da carga útil (ligada/desligada), coleta telemetria e dados do receptor de rádio
Controle dos aquecedores da bateria[a]	10 segundos	Controla a ativação dos aquecedores da bateria baseado se o satélite estaria em eclipse ou não

[a] Tarefa adaptada para o estudo de caso, detalhes sobre a adaptação estão no Capítulo 5.

Fonte: Elaborada pelo Autor (2026).

Convém salientar que esse serviço não implementa uma interface para o barramento ZMQ, visto que o objetivo da aplicação é emular as rotinas autônomas do OBDH2 e, portanto, não se faz necessário implementar comandos via interface ZMQ.

5 ESTUDO DE CASO - FLORIPASAT2

Por fim, o último passo da campanha de V&V da FlatSat2.0 — o que inclui a validação das aplicações de *software* desenvolvidas neste trabalho — é realizado através de um estudo de caso com subsistemas reais. Nesse contexto, as seções a seguir apresentam os subsistemas utilizados e, em seguida, apresentam a descrição e resultados dos testes realizados durante a campanha.

5.1 PLATAFORMA FLORIPASAT2

O FloripaSat-1 — um satélite 1U e o primeiro desenvolvido pelo SpaceLab — foi lançado em dezembro de 2019 e contava com uma plataforma de subsistemas de CubeSat desenvolvidade no próprio laboratório (19, 57, 71). Essa plataforma foi concebida para ser de propósito geral, ou seja, ela deve ser flexível o bastante para ser utilizada em missões com diferentes cargas úteis e requisitos (18).

Nesse contexto, dado o sucesso da missão do nanossatélite FloripaSat-1, uma nova plataforma *open source*, nomeada de FloripaSat-2, foi concebida baseada na experiência obtida com o satélite (18). Esse *bus* representa um avanço significativo em comparação ao anterior, oferecendo agora suporte para CubeSats 2U, bem como melhorias gerais. Além disso, a plataforma também se aproveita da herança de voo obtida com o FloripaSat-1, elevando a probabilidade do sucesso das missões (18).

A plataforma é composta, essencialmente, por subsistemas de um módulo de serviço de CubeSat, incluindo EPS, TTC e *On-Board Computer and Data Handling* (OBDH); este último também pode ser nomeado de OBC. Esses subsistemas são nomeados, respectivamente, de *Electrical Power System 2.0* (EPS2) (83), *Telemetry, Tracking and Command 2.0* (TTC2) (84) e OBDH2 (85). Esses módulos fornecem as capacidades mínimas necessárias — a exceção do controle de atitude — para que um CubeSat possa executar sua missão, o que inclui gerenciamento e coleta de energia, comunicação via rádio e gerenciamento de comandos e dados (18).

Algumas missões em execução utilizam esse *bus*, a exemplo das missões GOLDS-UFSC, Catarina-A1 e Catarina-A3. Além disso, os satélites FloripaSat-2A, FloripaSat-2B e Aldebaran-1

também utilizaram essa plataforma, contudo eles foram perdidos devido a falhas em seus respectivos lançamentos.

Tendo sido apresentada, cabe discutir a motivação de seu uso neste estudo de caso. Nesse contexto, pode-se ressaltar que, por ser um desenvolvimento próprio do SpaceLab, teve-se fácil acesso ao subsitemas, o que não seria o caso para plataformas comerciais devido ao custo associado. Ademais, um extenso histórico de testes faz-se presente, incluindo testes em bancada e testes ambientais; além disso, devido a sua adotação em múltiplas missões, o funcionamento e peculiaridades dos subsistemas são amplamente documentados.

Por fim, cabe comentar que, devido a natureza *open source*, pode-se ter livre acesso aos esquemáticos, *layout* e *flight software* de todos os módulos; este último, em particular, foi imprescindível para a emulação do OBDH2 comentada na Subseção 4.4.4.

5.2 MUDANÇAS NO DESIGN PARA A PLATAFORMA

Para que a implementação da emulação do subsistema OBDH2 fosse possível, precisou-se adaptar o design base do sistema Linux da FlatSat2.0. Essa seção comenta brevemente as modificações ao projeto, evidenciando seu impacto no consumo de recursos do PL, bem como na configuração da imagem do OS.

No caso do projeto de descrição de *hardware*, a adição de dois novo blocos IP foi necessária, sendo o primeiro um bloco *AXI_GPIO* foi utilizado para emular os *General Purpose I/O*s (GPIOs) do OBDH2 que controlam a ativação da carga útil; a segunda segunda adição foi um bloco *AXI_UART16550* para emular a interface de comunicação serial com a carga útil (85). Ademais, o segundo controlador de SPI do PS foi habilitado e configurado para emular a interface de comunicação do OBDH2 com o TTC2.

Essas adições foram implementadas utilizando a organização de scripts específicos para plataforma descrita na Seção 4.2. O BD final pode ser visto na Figura 12, sendo evidente as mudanças quando comparado com a Figura 9. Além disso, a utilização de recursos para esse design pode ser vista na Tabela 6, sendo levemente maior que a do projeto base (Tabela 3), mas ainda sub-utiliza o SoC.

Tabela 6: Uso de recursos do PL pelo design usado no estudo de caso

Recurso	**Utilização**	**Disponível**	**Porcentagem de Uso**
LUT	4007	53200	7,53%
LUTRAM	255	17400	1,47%
FF	4561	106400	4,29%

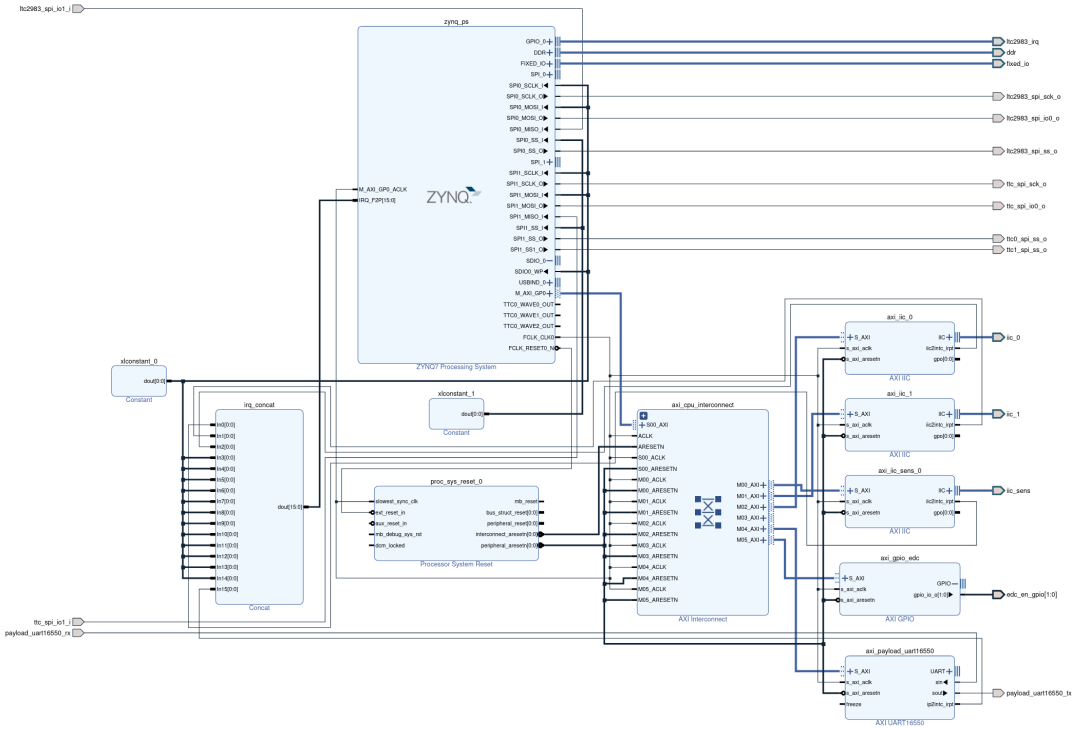
(Continua na próxima página)

Tabela 6: Uso de recursos do PL pelo design usado no estudo de caso (continuação)

Recurso	**Utilização**	**Disponível**	**Porcentagem de Uso**
IO	20	125	16,00%
BUFG	1	32	3,13%

Fonte: Elaborada pelo Autor (2026).

Figura 12 – *Block Design* para o projeto usado no estudo de caso

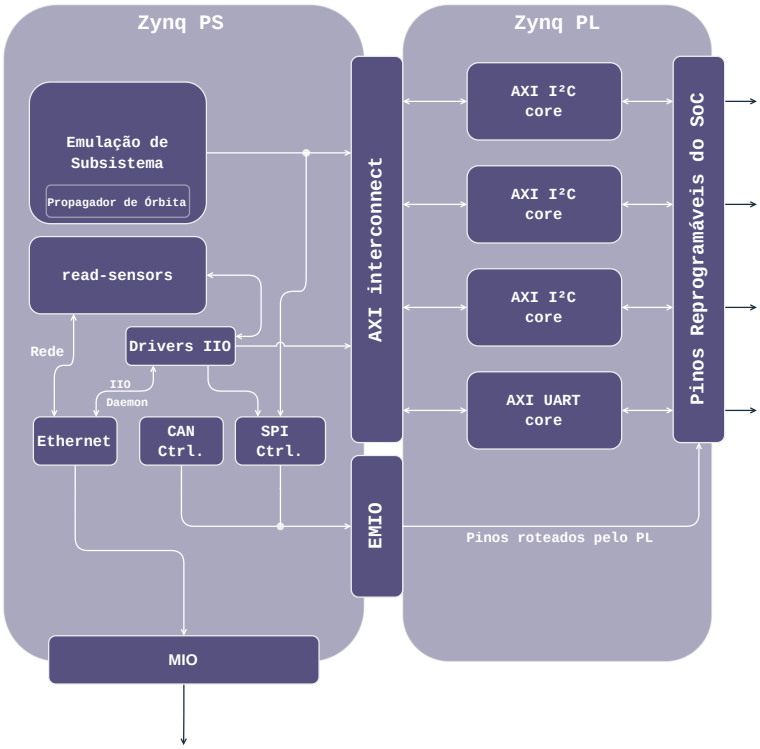


Fonte: Elaborado pelo Autor usando o *Vivado IDE* (2026).

O projeto do sistema Linux, por outro lado, precisou de apenas algumas alterações na kernel e *device-tree*, além de, claro, o serviço que emula o *obdh2*. Na kernel foram habilitados os *drivers* para a interface serial, implementada pelo IP *AXI_UART16650*, e o *spidev*, o qual permite o controle de interfaces SPI em aplicações de espaço de usuário. O *driver spidev* possibilitou, através de alterações mínimas, o reuso da parte do *flight software* referente a comunicação entre OBDH2 e TTC2.

Nesse contexto, a Figura 13 ilustra as aplicações e descrição de *hardware* implementados no Zynq, evidenciando as conexões lógicas entre os diferentes blocos.

Figura 13 – Diagrama de blocos das aplicações e configuração do Zynq



Fonte: Elaborado pelo Autor (2026).

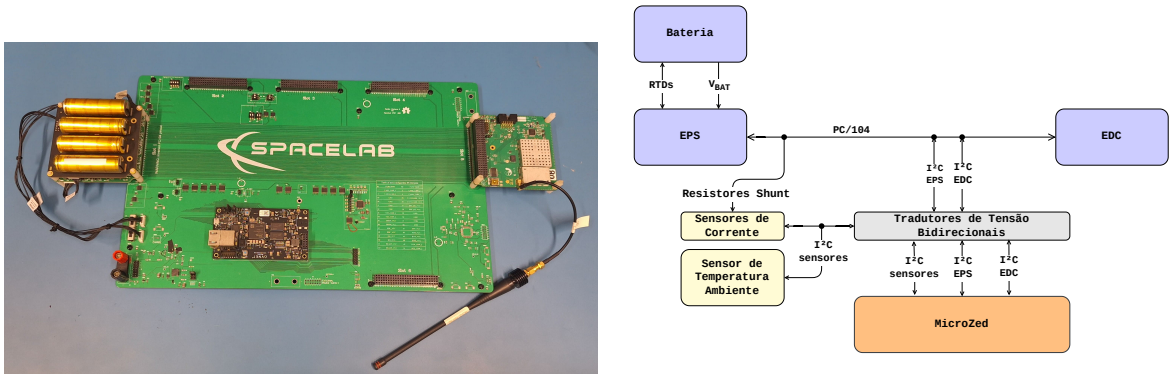
5.3 EMULAÇÃO DE SUBSISTEMAS E COLETA DE TELEMETRIA

O primeiro teste teve como objetivo validar a capacidade da FlatSat2.0 de emular subsistemas ausentes e, ao mesmo tempo, monitorar os sensores integrados, bem como unificar a telemetria com dados de sensores integrados em um único fluxo de armazenamento de dados. Para este fim, o ambiente de testes exibido na Figura 14 foi montado.

5.3.1 Descrição do Teste

Como ilustrado na Figura 14a, uma versão simplificada da plataforma FloripaSat-2, utilizada nas missões GOLDS-UFSC e Constelação Catarina (22), foi implementada. Em particular, os subsistemas EPS2 e o *Environmental Data Collector* (EDC) foram utilizados fisicamente, com o OBDH2 sendo emulado através da MicroZed. O EDC, apresentado em (86), é a principal carga útil de ambas as missões, sendo utilizado como receptor e concentrador de dados provenientes de Plataformas de Coleta de Dados (PCDs) instaladas no território brasileiro. A escolha do EPS2 se deu pela extensa telemetria disponível, bem como seu uso

Figura 14 – Configuração do ambiente de testes para a primeira campanha de V&V



(a) Foto do ambiente de testes (b) Diagrama de blocos

Fonte: (a) Tirada pelo Autor (2026); (b) Elaborado pelo Autor (2026).

para alimentação do EDC, o qual, além de ser uma carga útil conhecida e validada, representa uma carga representativa ao que será visto em voo.

Além disso, como descrito na Figura 14b, a MicroZed emite requisições de telemetria para ambos os módulos através de barramentos I^2C dedicados, assim como lê as medições dos sensores de temperatura ambiente e corrente (TMP112 e INA219) também através de I^2C. Essa configuração permitiu a aquisição contínua de telemetria dos subsistemas juntamente com as medições dos sensores integrados da plataforma, possibilitando a comparação e análise dos dados coletados.

5.3.2 Resultados e Discussão

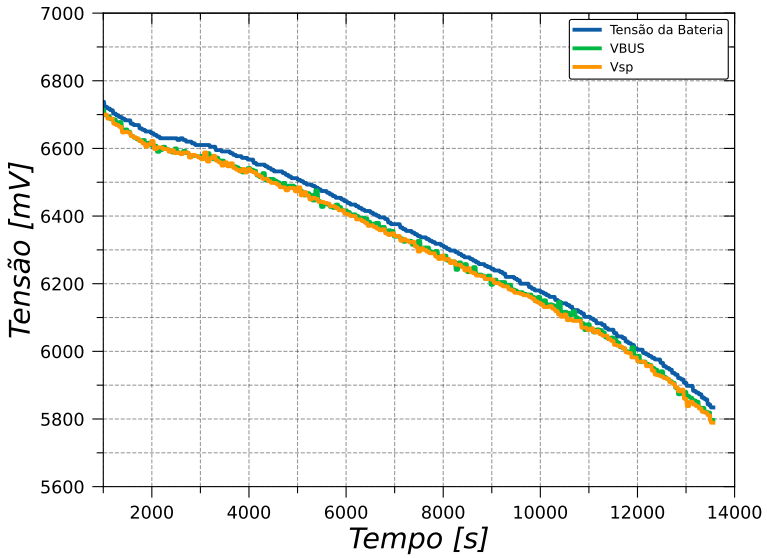
Durante o teste, o OBDH2 foi emulado pela MicroZed enquanto o EPS2 e o EDC foram simultaneamente monitorados pelos sensores da FlatSat2.0, bem como por suas telemetrias. A comunicação nos barramentos I^2C permaneceram sem erros durante toda a duração do teste.

Para ilustrar essa comunicação, a Figura 15 mostra as leituras do nível de tensão das baterias durante toda a duração do testes (aproximadamente 3,75 h ou 13 500 s), as quais foram obtidas através da telemetria do EPS2. A figura compara uma leitura obtida diretamente da bateria com o auxílio de um monitor de baterias integrado ao EPS2 (curva azul) com duas leituras indiretas obtidas após componentes passivos do circuito de potência, os quais introduzem uma pequena queda na tensão.

Percebe-se também que o teste começou com uma bateria parcialmente descarregada, com tensão inicial perto de 6,8 V. Além disso, o perfiu de descarga observado foi condizente com testes previamente realizados.

Ademais, medições elétricas e térmicas foram continuamente realizadas através do

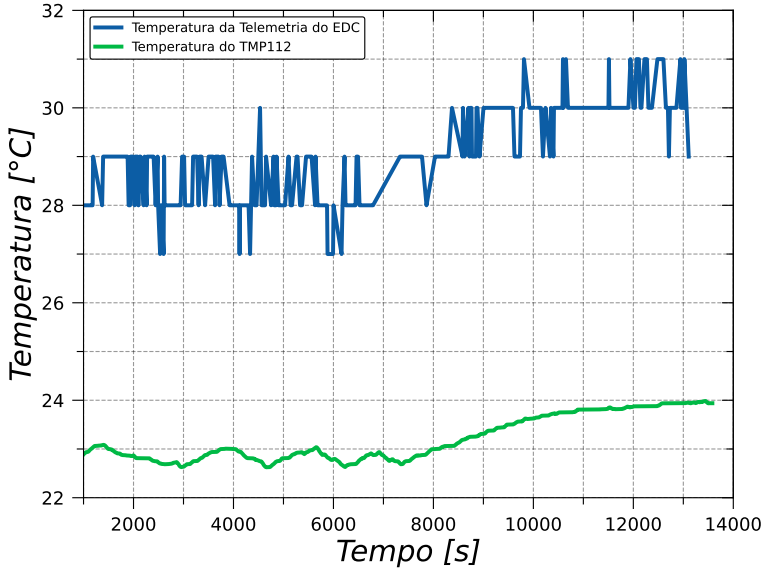
Figura 15 – Telemetria contendo leituras de tensão das baterias



Fonte: Elaborado pelo Autor (2026).

sensores embarcados. Nesse sentido, a Figura 16 compara as leituras do TMP112 com a telemetria do EDC. O sensor da FlatSat2.0, forneceu uma descrição mais detalhada das condições do laboratório durante o teste, quando comparado com os dados obtidos da telemetria do EDC.

Figura 16 – Leituras de temperatura obtidas do sensor TMP112 e do EDC



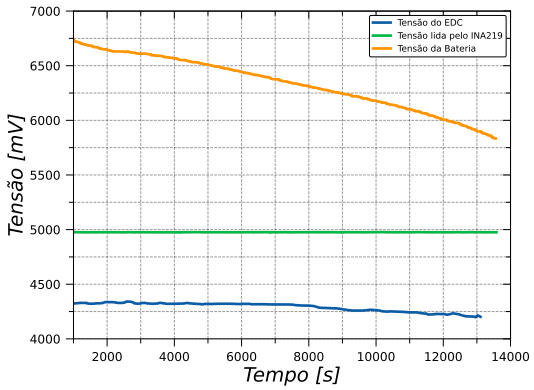
Fonte: Elaborado pelo Autor (2026).

Por exemplo, nas leituras do TMP112 foi possível perceber oscilações na temperatura ambiente — sendo atribuídas ao sistema de ar-condicionado do laboratório — seguidas por aumento da temperatura até 24 °C, ocasionado por um ajuste ao ponto de referência do ar-

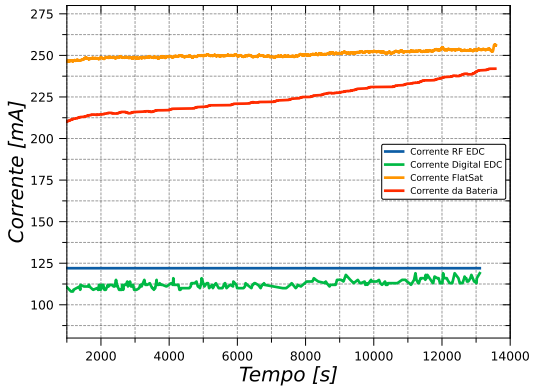
condicionado. Ao se observar apenas as leituras do EDC, não se pode discernir essas pequenas variações observadas com o TMP112. Apesar disso, é possível sim observar o gradual aumento da temperatura devido ao ajuste; começando, aproxidamente, em 7500 s.

Além disso, uma diferença de aproxidamente 6 °C pode ser vista entre as leituras do EDC e do TMP112: Após uma análise e inspeção da PCB, essa discrepância foi atribuída ao posicionamento do sensor na PCB, o qual possivelmente foi posicionado perto de fontes de calor. Contudo, uma avaliação mais conclusiva não foi possível, visto que o design do EDC é fechado, não se tendo acesso aos arquivos de seu projeto.

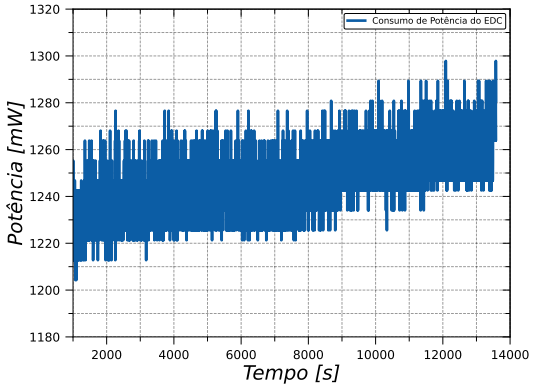
Figura 17 – Leituras de corrente, tensão e potência obtidas no primeiro teste; incluindo leituras dos sensores embarcados e telemetria



(a) Leituras de tensão



(b) Leituras de corrente



(c) Consumo de potência do EDC

Fonte: Elaborado pelo Autor (2026).

Outro exemplo de informações coletadas pode ser visto na Figura 17, a qual apresenta medições de corrente, tensão e potência obtidas através dos sensores INA219 e da telemetria dos subsistemas. Pelas leituras obtidas, em especial as do EDC, pode-se chegar a conclusão errônea que há algum problema com o barramento de 5 V da carga útil. Contudo ao se observar as leituras obtidas do sensor INA219 dedicado à esse canal, observa-se que a tensão se manteve estável em algo próximo de 4,97 V por toda duração do teste.

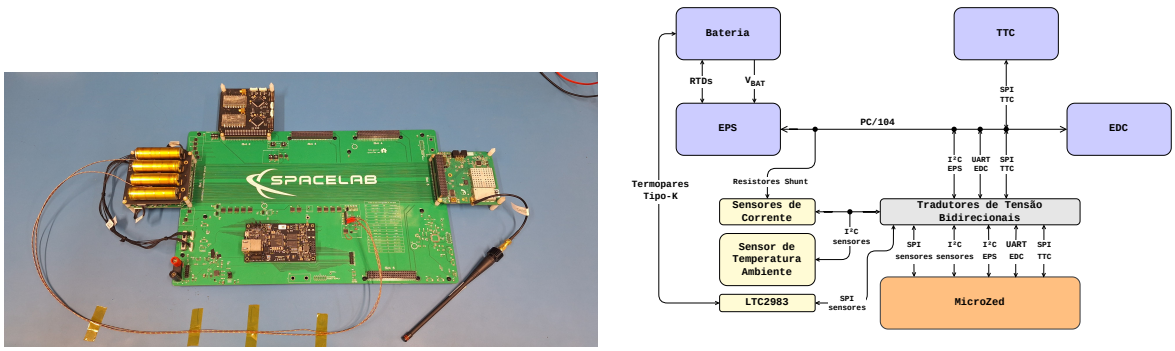
Novamente, essa discrepância pode ser atribuída a detalhes de implementação de *hardware* ou *firmware* do EDC. Nesse sentido, o fato de que não houveram quaisquer problemas perceptíveis com a operação do subsistema — mesmo diante de uma alimentação consideravelmente menor que a nominal — corrobora com a hipótese de que houve algum erro na leitura do EDC.

Adicionalmente, as mesmas leituras mostram um aumento no consumo de potência do EDC ao longo do teste, comportamento evidenciado na Figura 17b e Figura 17c. Essa tendência se alinha, quase que perfeitamente, com o aumento da temperatura registrado na Figura 16. Entretanto, esse comportamento não é discernível apenas com a telemetria do EDC, o qual fragmenta as leituras de corrente em sua telemetria, além do problema com as leituras de tensão; sendo necessário a análise das leituras dos sensores embarcados, as quais deixam essa tendência explicíta.

5.4 EMULAÇÃO DE CENÁRIOS DE MISSÃO

Para o segundo teste, seu objetivo era validar a capacidade da FlatSat2.0 de funcionar de modo semelhante a um EGSE, explorando sua capacidade de lidar com simulações complexas — a exemplo de cenários de missão — e múltiplos serviços simultâneos. Para esse fim, o ambiente de testes exibido na Figura 18 foi montado.

Figura 18 – Configuração do ambiente de testes para a segunda campanha de V&V



(a) Foto do ambiente de testes (b) Diagrama de blocos

Fonte: (a) Tirada pelo Autor (2026); (b) Elaborado pelo Autor (2026).

5.4.1 Descrição do Teste

Como pode ser observado na Figura 18a, a configuração do segundo teste manteve os subsistemas EPS2 e EDC e adicionou o módulo TTC2, o qual serve como rádio principal do *bus* FloripaSat-2. Além disso, termopares do tipo K foram conectados ao sensor LTC2983,

empregando um transistor como junção fria. Esses termopares foram instalados ao lado dos aquecedores de bateria a fim de prover medições de temperatura em locais críticos da placa de bateria.

Adicionalmente, com o objetivo de simular um cenário de missão em órbita, um modelo propagador de órbita baseado no SGP4 (87) — utilizado também no *flight software* do OBDH2 — foi implementado na MicroZed. Esse modelo gera estimativas em tempo real do estado da órbita do CubeSat, incluindo informações sobre passagens pelo território brasileiro e períodos de eclipse do satélite. Baseado nessas predições, dois tipos de comandos foram gerados pelo SoM: i) os aquecedores foram habilitados durante períodos de eclipse e desabilitados com a presença de luz solar; e ii) o EDC foi habilitado apenas durante passagens do satélite pelo Brasil.

O primeiro tipo de comando tem o objetivo de emular a ativação automática dos aquecedores de bateria pelo EPS2, contudo — como não foi possível injetar valores aos sensores do tipo *Resistance Temperature Detector* (RTD) que controlam esse procedimento — adaptou-se o *flight software* emulado para conduzir isso forçadamente, algo que não aconteceria em voo. Por outro lado, a ativação do EDC apenas em passagens pelo Brasil é o comportamento esperado para ambas as missões comentadas, sendo inclusive requisito de projeto. Portanto, tem-se nessa configuração, um ambiente de teste autônomo que emula um cenário de missão de forma realista; possibilitando a análise do comportamento dos subsistemas e do *flight software* ao cenário emulado.

Ademais, a MicroZed coleta telemetria de todos os subsistemas conectados através das interfaces I^2C, SPI e UART, como pode ser visto na Figura 18b. Isso permite a comparação direta entre as leituras dos RTDs presentes na placa de baterias e os termopares utilizados pelo sensor LTC2983.

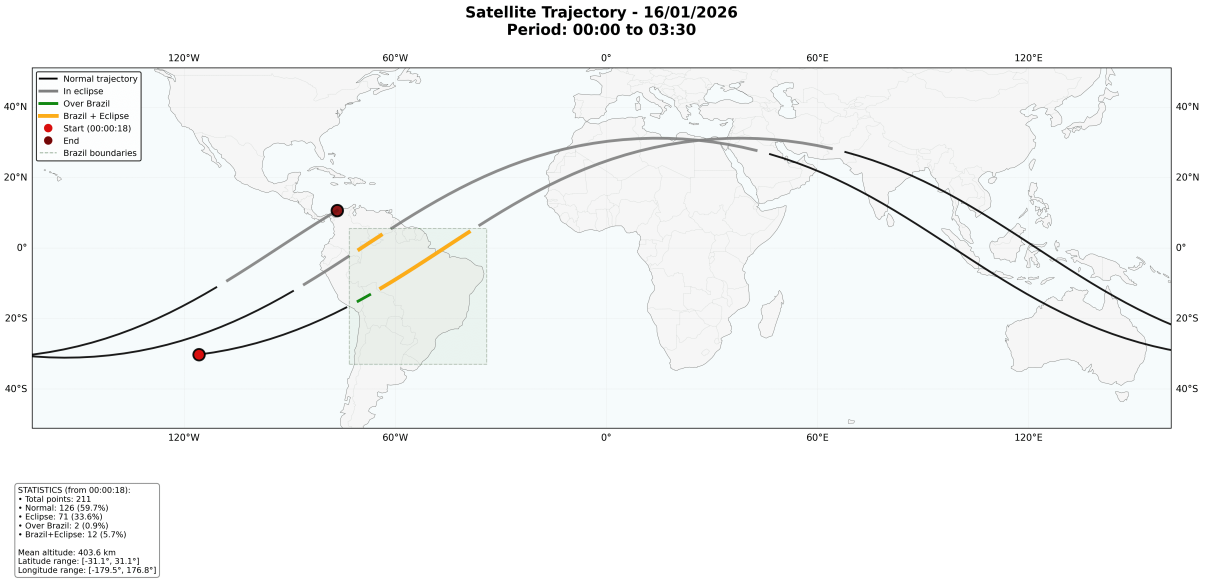
5.4.2 Resultados e Discussão

Durante o segundo teste, uma simulação de órbita foi executada na MicroZed, baseada nos elementos orbitais provenientes da *Two-line Element* (TLE) do satélite HORYU-4. Nesse sentido, a TLE do HORYU-4 foi escolhida pois seus parâmetros orbitais, em particular sua inclinação, são próximos do que é esperado para a missão GOLDS-UFSC, tornando-a adequada para a emulação de cenários de missão representativos.

A fim de demonstrar o comportamento da órbita simulada, uma representação gráfica pode ser vista na Figura 19. Os dados e eventos usados para a construção dessa ilustração foram armazenados durante a execução do teste em um banco de dados SQLite, isso sendo feito em paralelo com a coleta de telemetria e leitura dos sensores embarcados. O teste teve uma duração de 3,5 h, cobrindo duas revoluções orbitais completas, com uma terceira começando

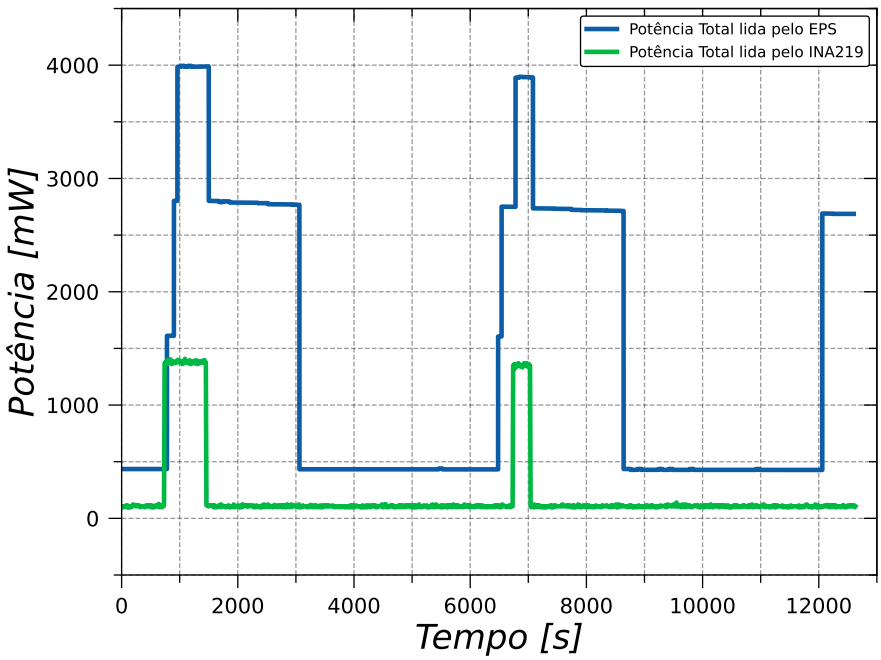
perto do fim do procedimento.

Figura 19 – Simulação da órbita do satélite Horyu-4; mostrando a ativação do aquecedor, eventos de ativação do EDC e períodos de operação simultânea



Fonte: Elaborado pelo Autor (2026).

Figura 20 – Consumo de potência do sistema durante a emulação do cenário de missão



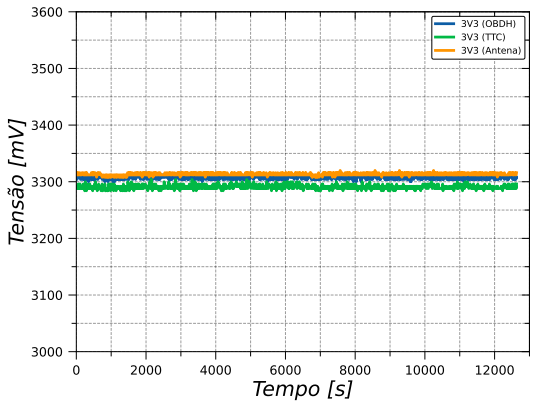
Fonte: Elaborado pelo Autor (2026).

Ao se observar a Figura 19, percebe-se uma ativação do EDC no início da primeira revolução, sendo ligado sozinho inicialmente e, na sequência, os aquecedores da bateria são habilitados também. Esse comportamento também é visto no consumo de potência total do

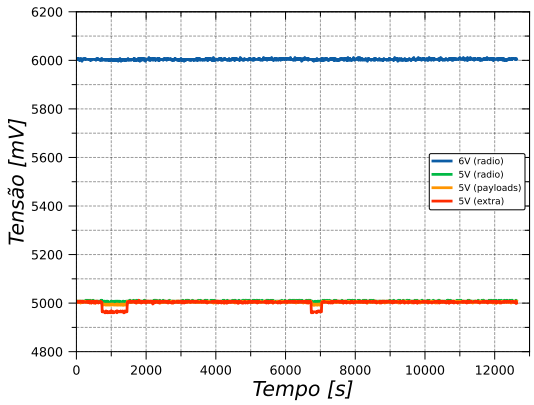
sistema (Figura 20), onde um primeiro aumento do consumo total (curva azul) é observado, sendo esse associado à ativação do EDC (curva verde). Após isso, dois aumentos de consumo podem ser vistos, os quais correspondem à ativação dos aquecedores, resultando em pico um no consumo de, aproximadamente, 4 W. Logo em seguida, o satélite sai do território brasileiro — resultando no desligamento do EDC — enquanto os aquecedores permanecem ligados até o fim do período de eclipse.

No geral, a correspondência entre os dados obtidos da simulação orbital e da telemetria dos subsistemas indica que a lógica de ativação do EDC foi emulada de forma consistente e correta durante o procedimento de teste. Além disso, mais informações podem ser obtidas da telemetria e leituras do sensores, em particular, as leituras dos barramentos de potência e tensão da bateria (Figura 21) revelam comportamentos interessantes.

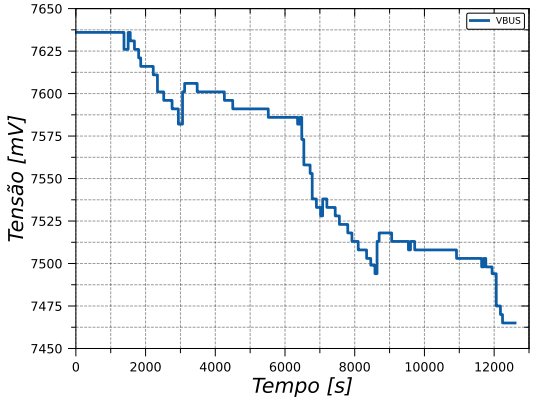
Figura 21 – Comportamento do sistema de potência durante o cenário emulado; incluindo leituras dos barramentos regulados e o perfil de descarga da bateria



(a) Leituras do barramento de 3,3 V



(b) Leituras dos barramentos de 5 V e 6 V



(c) Leituras da tensão da bateria

Fonte: Elaborado pelo Autor (2026).

Um exemplo disso é o perfil de descarga da bateria, observado na Figura 21c, o qual é consideravelmente mais representativo de um cenário real de operação do que o obtido no teste anterior (Figura 17a). Anteriormente, a tensão da bateria decaiu de maneira constante —

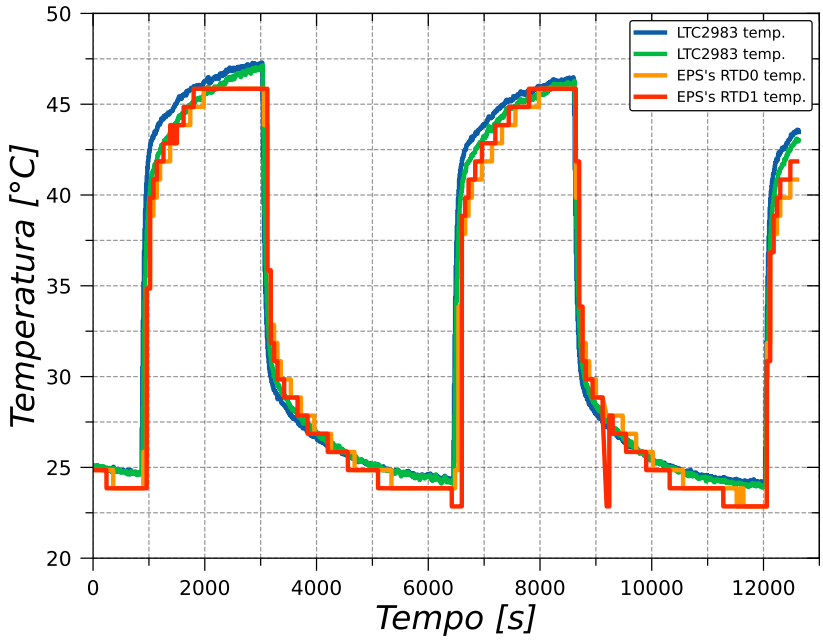
começando em 6,8 V e decaindo até 5,75 V — ao longo do primeiro teste.

Por outro lado, durante o segundo o teste, a queda total de tensão não excedeu 0,3 V. Além disso, quedas maiores são observadas em períodos com cargas mais elevadas, em particular a ativação dos aquecedores e/ou o EDC, o que é o comportamento esperado. Essa característica é consistente com os testes de qualificação executados durante a campanha V&V da missão GOLDS-UFSC e, portanto, torna-se uma evidência que a FlatSat2.0 pode ser usada para emular cenários de missão representativos.

Sobre os barramentos de potência, observa-se uma operação estável durante todo o procedimento, não havendo nenhuma evidência de comportamento anômalo nas linhas de 3,3 V, 5 V e 6 V. Nesse contexto, uma pequena queda de tensão pode ser vista no barramento de 5 V nos instantes que o EDC permanece ligado, coincidindo com o que é observado na Figura 20. Esse comportamento é atribuído ao aumento súbito da carga no barramento durante a ativação da carga útil e é consistente com o perfil esperado para o EPS2.

Ademais, a adição dos termopares ao sensor LTC2983 possibilitou medições de temperatura que não eram anteriormente possíveis em testes de bancada (Figura 22). As leituras de temperatura perto aos aquecedores atingiram, aproximadamente, 47 °C, com um desvio menor que 2 °C das medições dos RTDs da placa de baterias. Esse alinhamento entre as medidas valida a telemetria do EPS2 relacionada ao controle térmico das baterias. Acima de tudo, esse procedimento demonstra a viabilidade do monitoramento de componentes críticos, a exemplo as bateriais ou amplificadores de potência, os quais podem não ter sensoriamento dedicado.

Figura 22 – Leituras de temperatura realizadas durante a emulação de cenário de missão; incluindo medições dos termopares e telemetria do EPS2.0

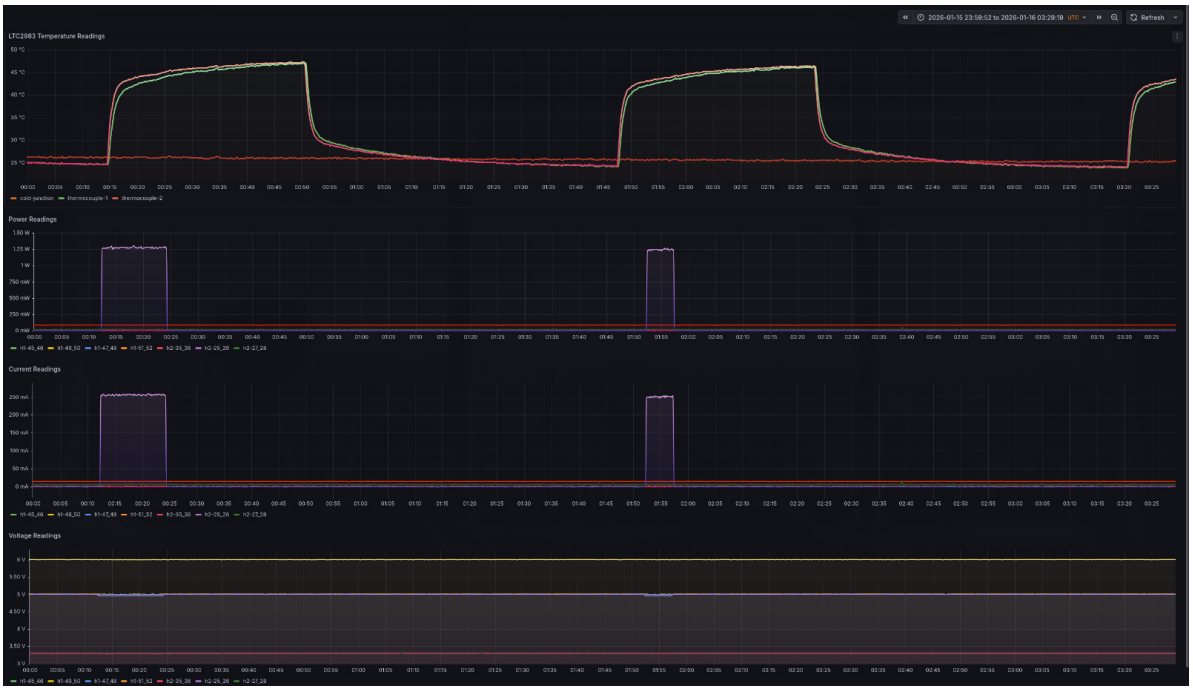


Fonte: Elaborado pelo Autor (2026).

Além disso, o uso de memória e dos processadores foram monitorados ao longo do teste. As leituras obtidas mostram que o uso de memória atingiu, no máximo, 11,6% do total disponível durante o teste; por outro lado, no caso dos processadores, não se ultrapassou 5% de uso ao longo do teste. Esses resultados indicam que, mesmo com os serviços já implementados, há recursos computacionais de sobra para a implementação e uso de novos serviços.

Por fim, foi possível monitorar, em tempo real, as leituras do sensores embutidos na FlatSat2.0 através da interface de rede. Nesse sentido, foi utilizado o *software Grafana* para apresentar os dados em um banco em um formato de *dashboard*. A Figura 23 mostra as leituras dos sensores durante o segundo o teste.

Figura 23 – *Dashboard* do *Grafana* com as leitura dos sensores durante o teste



Fonte: Elaborado pelo Autor (2026).

6 CONCLUSÃO

Neste trabalho, se conduziu uma pesquisa e desenvolvimento de uma arquitetura reutilizável para execução de procedimentos de testes e V&V em subsistemas de CubeSats, visando a melhoria da campanha de V&V e o aumento da confiabilidade desses sistemas. É apresentado uma breve revisão sobre confiabilidade no contexto de CubeSats, assim como uma revisão sobre FlatSats e seu uso na campanha de V&V dessa classe de satélites. Além disso, a plataforma FlatSat2.0 foi apresentada, a qual serviu como base para o desenvolvimento e implementação da arquitetura proposta neste trabalho.

Ademais, a metodologia utilizada para a validação da plataforma, bem como para o desenvolvimento da arquitetura foi apresentada. Em seguida, os testes realizados para a validação inicial da FlatSat2.0 são mostrados, os quais — a fim de compartilhar o progresso do uso dessa FlatSat — foram compilados em um artigo apresentado no *Latin American Cubesat Workshop 2024*.

Depois, o desenvolvimento da arquitetura em si foi descrito, mostrando as etapas e procedimentos envolvidos. Esse desenvolvimento engloba o projeto de descrição de *hardware* e a construção de um sistema Linux customizado, os quais são empregados no processador embarcado da plataforma. Adicionalmente, os serviços de *software* implementados são detalhados. Cabe ressaltar que a arquitetura implementada neste trabalho é *open source* e é disponibilizada gratuitamente para seu uso em outras missões.

Por fim, um estudo de caso foi realizado com objetivo de demonstrar o uso da arquitetura proposta com subsistemas reais de CubeSats. Essa etapa não só valida a arquitetura, como também valida a plataforma propriamente dita, viabilizando seu uso nas missões em andamento do SpaceLab.

Também é de interesse comentar que um artigo científico que apresenta a plataforma FlatSat2.0 e seu uso em campanhas de V&V para CubeSats — baseado neste trabalho e no que é apresentado em (20) — está sendo escrito. Além disso, trabalhos futuros abordando procedimentos mais complexos com a arquitetura proposta, a exemplo da avaliação completa de um EPS, também foram planejados.

REFERÊNCIAS

1 KUZNETSOV, V. D.; SINELNIKOV, V. M.; ALPERT, S. N. Yakov alpert: Sputnik-1 and the first satellite ionospheric experiment. **Advances in Space Research**, Elsevier, v. 55, n. 12, p. 2833–2839, 2015. Citado 2 vezes nas páginas 15 e 19.

2 Orbit.ing-now. **Orbit.ing-now**. 2025. Online. Acesso em: 15 nov. 2025. Disponível em: <https://orbit.ing-now.com/>. Citado na página 15.

3 POGHOSYAN, A.; GOLKAR, A. Cubesat evolution: Analyzing cubesat capabilities for conducting science missions. **Progress in Aerospace Sciences**, Elsevier, v. 88, p. 59–83, 2017. Citado 5 vezes nas páginas 15, 16, 19, 20 e 21.

4 European Space Agency. **Envisat**. 2025. Online. Acesso em: 15 nov. 2025. Disponível em: <https://www.esa.int/Applications/Observing_the_Earth/Envisat>. Citado na página 15.

5 DUBOCK, P. A.; SPOTO, F.; SIMPSON, J.; SPENCER, D.; SCHüTTE, E.; SONTAG, H. The envisat satellite and its integration. **ESA Bulletin**, v. 106, p. 26–45, 2001. Acesso em: 15 nov. 2025. Citado na página 15.

6 PUIG-SUARI, J.; TURNER, C.; TWIGGS, R. J. Cubesat: The development and launch support infrastructure for eighteen different satellite customers on one launch. In: **AIAA/USU Conference on Small Satellites**. [S.l.: s.n.], 2001. Citado 2 vezes nas páginas 15 e 20.

7 CAL POLY. **CP-CDS-R14.1**: Cubesat design specification. San Luis Obispo, USA, 2022. 34 p. Citado 2 vezes nas páginas 15 e 20.

8 NASA. **CubeSat101: Basic Concepts and Processes for First-Time CubeSat Developers**. S.l., 2017. Acesso em: 30 dez. 2025. Disponível em: <https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf>. Citado na página 15.

9 Kulu, Erik. **Nanosatellite & CubeSat Database**. 2025. Online. Acesso em: 30 dez. 2025. Disponível em: <https://www.nanosats.eu/>. Citado 3 vezes nas páginas 16, 19 e 20.

10 KULU, E. Cubesats & nanosatellites – 2024 statistics, forecast and reliability. In: **Proceedings of the 75th International Astronautical Congress (IAC 2024)**. Milan, Italy: International Astronautical Federation (IAF), 2024. (IAC, B4.6A.13), p. –. Citado 3 vezes nas páginas 16, 20 e 22.

11 BOUWMEESTER, J.; GUO, J. Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology. **Acta Astronautica**, Elsevier, v. 67, n. 7, p. 854–862, 2010. Citado 3 vezes nas páginas 16, 20 e 21.

12 MONTEIRO, J. P.; ROCHA, R. M.; SILVA, A.; AFONSO, R.; RAMOS, N. Integration and verification approach of istsat-1 cubesat. **Aerospace**, MDPI, v. 6, n. 12, 2019. Citado 8 vezes nas páginas 16, 21, 23, 25, 26, 27, 28 e 31.

13 MILLAN, R. M.; STEIGER, R. von; ARIEL, M.; BARTALEV, S.; BORGEAUD, M.; CAMPAGNOLA, S.; CASTILLO-ROGEZ, J. C.; FLÉRON, R.; GASS, V.; GREGORIO, A.; KLUMPAR, D. M.; LAL, B.; MACDONALD, M.; PARK, J. U.; RAO, V. S.; SCHILLING, K.; STEPHENS, G.; TITLE, A. M.; WU, J. Small satellites for space science: A cospar scientific roadmap. **Advances in Space Research**, Elsevier, v. 64, n. 8, p. 1466–1517, 2019. Citado na página 16.

14 BOUWMEESTER, J.; MENICUCCI, A.; GILL, E. Improving cubesat reliability: Subsystem redundancy or improved testing? **Reliability Engineering & System Safety**, v. 220, p. 108288, 2022. ISSN 0951-8320. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0951832021007584>. Citado 2 vezes nas páginas 16 e 22.

15 BARCELLOS, J. C. E.; SPENGLER, A. W.; SEMAN, L. O.; SILVA, R. D. C. e; ROLDÁN, H. P.; BEZERRA, E. A. Flatsat platforms for small satellites: A systematic mapping and classification. **IEEE Journal on Miniaturization for Air and Space Systems**, IEEE, v. 4, n. 2, p. 186–198, 2023. Citado 6 vezes nas páginas 16, 17, 25, 26, 27 e 28.

16 GEBARA, C.; SPENCER, D. Verification and validation methods for the prox-1 mission. In: AIAA/USU CONFERENCE ON SMALL SATELLITES, 30., 2016, Logan, USA. **Proceedings...** Logan, USA: AIAA/USU, 2016. p. 8. Disponível em: <https://digitalcommons.usu.edu/smallsat/2016/TS8StudentComp/3/>. Citado 4 vezes nas páginas 16, 17, 25 e 26.

17 WALSH, S.; MURPHY, D.; DOYLE, M.; REILLY, J.; THOMPSON, J.; DUNWOODY, R.; ERKAL, J.; FINNERAN, G.; FONTANESI, G.; MANGAN, J.; MARSHALL, F.; SALMON, L.; FAOITE, D. de; HANLON, L.; MARTIN-CARRILLO, A.; MCKEOWN, D.; O'CONNOR, W.; ULIYANOV, A.; WALL, R.; MCBREEN, S. Development of the eirsat-1 cubesat through functional verification of the engineering qualification model. **Aerospace**, MDPI, v. 8, n. 9, 2021. Citado 6 vezes nas páginas 16, 23, 24, 25, 26 e 27.

18 MARCELINO, G. M.; MATTOS, A. M. P. de; BARCELLOS, J. C. E.; RIBEIRO, B. F.; SEMAN, L. O.; FILHO, E. M.; BEZERRA, E. A. Floripasat-2: An open-source platform for cubesats. **IEEE Embedded Systems Letters**, IEEE, v. 16, n. 1, p. 77–80, 2024. Citado 6 vezes nas páginas 16, 17, 28, 35, 48 e 51.

19 MARCELINO, G. M.; FILHO, E. M.; VEGA-MARTINEZ, S.; MATTOS, A. M. P. de; SEMAN, L. O.; SLONGO, L. K.; BEZERRA, E. A. Qualification and validation test methodology of the open-source cubesat floripasat-i. **Journal of Systems Engineering and Electronics**, BIAI, v. 31, n. 6, p. 1230–1244, 2020. Citado 3 vezes nas páginas 17, 35 e 51.

20 BARCELLOS, J. C. E. Projeto e desenvolvimento de uma plataforma de flatsat para testes de subsistemas de cubesats. **Universidade Federal de Santa Catarina**, 2022. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Citado 7 vezes nas páginas 17, 27, 28, 29, 33, 48 e 64.

21 MARTINS, R. H. **Proposição de testes para FlatSat da Missão FloripaSat utilizando dispositivo programável**. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica), 2022. Disponível em: <https://repositorio.ufsc.br/handle/123456789/243151>. Citado 2 vezes nas páginas 17 e 33.

22 Agência Espacial Brasileira. **Constelação Catarina**. 2023. Online. Acesso em: 05 jan. 2026. Disponível em: <https://www.gov.br/aeb/pt-br/acoes-e-programas/desenvolvimento-tecnologico/constelacao-catarina>. Citado 2 vezes nas páginas 17 e 54.

23 SILVA, R. C. B. A. da; JÚNIOR, A. L. X. A unified satellite taxonomy proposal based on mass and size. **Advances in Aerospace Science and Technology**, SCIRP, v. 4, n. 4, p. 57–73, 2019. Citado na página 19.

24 DEPASQUALE, J.; CHARANIA, A. C.; KANAMAYA, H.; MATSUDA, S. Analysis of the earth-to-orbit launch market for nano and microsatellites. In: AIAA SPACE 2010 CONFERENCE & EXPOSITION, 22., 2010, Anaheim, USA. **Proceedings...** Anaheim, USA: AIAA, 2010. Disponível em: <:https://arc.aiaa.org/doi/abs/10.2514/6.2010-8602>. Citado na página 20.

25 SHIELDS, S. G.; COOPS, N. C.; ACHIM, A.; HAMELIN, R. C.; MULVERHILL, C. A review of planetscope cubesats for forest monitoring. **Science of Remote Sensing**, v. 12, p. 100314, 2025. ISSN 2666-0172. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666017225001208>. Citado na página 20.

26 SWARTWOUT, M.; JAYNE, C. University-class spacecraft by the numbers: Success, failure, debris. (but mostly success.). In: AIAA/UTAH STATE UNIVERSITY. **Proceedings of the 30th AIAA/USU Conference on Small Satellites**. Logan, UT, USA, 2016. Acesso em: 28 jan. 2026. Disponível em: <https://digitalcommons.usu.edu/smallsat/2016/TS13Education/1/>. Citado na página 21.

27 LANGER, M.; BOUWMEESTER, J. Reliability of cubesats - statistical data, developers' beliefs and the way forward. In: AIAA/USU CONFERENCE ON SMALL SATELLITES, 30., 2016, Logan, USA. **Proceedings...** Logan, USA: AIAA/USU, 2016. p. 12. Disponível em: <https://digitalcommons.usu.edu/smallsat/2016/TS10AdvTech2/4/>. Citado na página 22.

28 EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-E-ST-10C**: System engineering general requirements. Noordwijk, The Netherlands, 2017. 116 p. Citado na página 23.

29 EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-E-ST-10-03C**: Testing. Noordwijk, The Netherlands, 2022. 146 p. Citado 2 vezes nas páginas 23 e 34.

30 EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-E-HB-10-03A**: Testing guidelines. Noordwijk, The Netherlands, 2022. 268 p. Citado na página 23.

31 EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-E-ST-10-02C**: Verification. Noordwijk, The Netherlands, 2018. 51 p. Citado 2 vezes nas páginas 23 e 34.

32 EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-E-HB-10-02A**: Verification guidelines. Noordwijk, The Netherlands, 2010. 96 p. Citado na página 23.

33 NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. **GSFC-STD-7000B**: General environmental verification standard (gevs) for gsfc flight programs and projects. Greenbelt, USA, 2021. 337 p. Citado na página 23.

34 BATISTA, C. L. G.; CORSETTI, A.; MATTIELLO-FRANCISCO, F. Using Fault Injection on the Nanosatellite Subsystems Integration Testing. arXiv, 2021. ArXiv:2102.11776 [cs]. Disponível em: <http://arxiv.org/abs/2102.11776>. Citado na página 23.

35 BORBA, R. d. A. **Plano de testes para módulos de gerenciamento de energia de cubesats**. 111 p. BSc. thesis — Universidade Federal de Santa Catarina (UFSC), Florianópolis, Brazil, 2024. Disponível em: <https://repositorio.ufsc.br/handle/123456789/254727>. Citado na página 23.

36 PRAKS, J.; MUGHAL, M. R.; VAINIO, R.; JANHUNEN, P.; ENVALL, J.; OLEYNIK, P.; NäSILä, A.; LEPPINEN, H.; NIEMELä, P.; SLAVINSKIS, A.; GIESELER, J.; TOIVANEN, P.; TIKKA, T.; PELTOLA, T.; BOSSER, A.; SCHWARZKOPF, G.; JOVANOVIC, N.; RIWANTO, B.; KESTILä, A.; PUNKKINEN, A.; PUNKKINEN, R.; HEDMAN, H.-P.; SäNTTI, T.; LILL, J.-O.; SLOTTE, J.; KETTUNEN, H.; VIRTANEN, A. Aalto-1, multi-payload cubesat: Design, integration and launch. **Acta Astronautica**, v. 187, p. 370–383, 2021. ISSN 0094-5765. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0094576520307189>. Citado 5 vezes nas páginas 23, 24, 25, 26 e 32.

37 AZAMI, M. H. B.; ORGER, N. C.; SCHULZ, V. H.; OSHIRO, T.; ALARCON, J. R. C.; MASKEY, A.; NAKAYAMA, K.; FUKUDA, Y.; KOJIMA, K.; YAMAUCHI, T.; MASUI, H.; CHO, M.; KITSUNE Team Members. Design and environmental testing of imaging payload for a 6u cubesat at low earth orbit: Kitsune mission. **Frontiers in Space Technologies**, Volume 3 - 2022, 2022. ISSN 2673-5075. Disponível em: <https://www.frontiersin.org/journals/space-technologies/articles/10.3389/frspt.2022.1000219>. Citado na página 23.

38 PRENTICE, E. F.; HONORé-LIVERMORE, E.; BAKKEN, S.; HENRIKSEN, M. B.; BIRKELAND, R.; HJERTENæS, M.; GJERSVIK, A.; JOHANSEN, T. A.; AGUADO-AGELET, F.; NAVARRO-MEDINA, F. Pre-launch assembly, integration, and testing strategy of a hyperspectral imaging cubesat, hypso-1. **Remote Sensing**, v. 14, n. 18, 2022. ISSN 2072-4292. Disponível em: <https://www.mdpi.com/2072-4292/14/18/4584>. Citado 4 vezes nas páginas 23, 24, 25 e 32.

39 CHIU, Y.-C.; CHANG, L. C.; CHAO, C.-K.; TAI, T.-Y.; CHENG, K.-L.; LIU, H.-T.; TSAI-LIN, R.; LIAO, C.-T.; LUO, W.-H.; CHIU, G.-P.; HOU, K.-J.; WANG, R.-Y.; GACAL, G. F.; LIN, P.-A.; DENDUONGHATAI, S.; YU, T.-R.; LIU, J.-Y.; CHANDRAN, A.; ATHREYAS, K. B. N.; HARI, P.; VARGHESE, J. J.; MEFTAH, M. Lessons learned from ideassat: Design, testing, on orbit operations, and anomaly analysis of a first university cubesat intended for ionospheric science. **Aerospace**, v. 9, n. 2, 2022. ISSN 2226-4310. Disponível em: <https://www.mdpi.com/2226-4310/9/2/110>. Citado 2 vezes nas páginas 24 e 25.

40 HALVERSON, J.; CALHOUN, P.; HSU, O.; DONGMO, J.; BESSER, R.; ELLIS, B.; HART, R. D.; TEDLA, Y.; ROSNEY, S.; SNELL, S. T. Testing of the Lunar Reconnaissance Orbiter Attitude Control System Re-Design Without a Gyro. In: **Proceedings of the 42nd Annual AAS Guidance and Control Conference**. Breckenridge, CO, EUA: [s.n.], 2019. Disponível em: <https://ntrs.nasa.gov/citations/20190002613>. Citado na página 25.

41 ALSUWAIDI, A. Emirates Mars Mission Flight Simulator: FlatSat Overview. In: **Proceedings of the IEEE Aerospace Conference**. Big Sky, MT, USA: [s.n.], 2020. p. 1–11. Citado na página 25.

42 BEZERRA, E. A.; AZEVEDO, L. R.; SILVA, E. M. da. Dependability issues of inpe's esa/ccsds telecommand/telemetry subsystem. In: **2011 Fifth Latin-American Symposium on Dependable Computing Workshops**. [S.l.: s.n.], 2011. p. 58–63. Citado na página 25.

43 SPENCER, D. A.; CHAIT, S. B.; SCHULTE, P. Z.; OKSENIUK, K. J.; VETO, M. Prox-1 University-Class Mission to Demonstrate Automated Proximity Operations. **Journal of Spacecraft and Rockets**, v. 53, p. 847–863, 2016. Citado na página 26.

44 SMITH, A.; MUÑOZ, S.; HAGEN, E.; JOHNSON, G.; LIGHTSEY, G. The FASTRAC Satellites: Software Implementation and Testing. In: **Proceedings of the 22nd Annual AIAA/USU Small Satellite Conference**. Logan, UT, USA: [s.n.], 2008. Disponível em: <https://digitalcommons.usu.edu/smallsat/2008/all2008/81/>. Citado na página 26.

45 MUÑOZ, S.; GREENBAUM, J.; CAMPBELL, T.; HOLT, G.; LIGHTSEY, E. G. The FASTRAC Experience: A Student Run Nanosatellite Program. In: **Proceedings of the 24th Annual AIAA/USU Small Satellite Conference**. Logan, UT, USA: [s.n.], 2010. Disponível em: <https://digitalcommons.usu.edu/smallsat/2010/all2010/76/>. Citado na página 26.

46 MUÑOZ, S.; GREENBAUM, J.; LIGHTSEY, G.; CAMPBELL, T.; STEWART, S.; HOLT, G. The FASTRAC Mission: Operations Summary and Preliminary Experiment Results. In: **Proceedings of the 25th Annual AIAA/USU Small Satellite Conference**. Logan, UT, USA: [s.n.], 2011. Disponível em: <https://digitalcommons.usu.edu/smallsat/2011/all2011/24/>. Citado na página 26.

47 LANGER, M. **Reliability Assessment and Reliability Prediction of CubeSats Through System Level Testing and Reliability Growth Modelling**. Tese (Ph.D thesis) — Technical University of Munich, Munich, BY, Germany, 2018. Citado 2 vezes nas páginas 26 e 28.

48 SCHUMMER, F. **Reliability Assessment of Small Spacecraft Before Launch**. Dissertação (Master's thesis) — Technical University of Munich, Munich, BY, Germany, 2017. Citado 2 vezes nas páginas 26 e 28.

49 KIESBYE, J.; MESSMANN, D.; PREISINGER, M.; REINA, G.; NAGY, D.; SCHUMMER, F.; MOSTAD, M.; KALE, T.; LANGER, M. Hardware-In-The-Loop and Software-In-The-Loop Testing of the MOVE-II CubeSat. **Aerospace**, v. 6, 2019. ISSN 2226-4310. Citado 2 vezes nas páginas 26 e 28.

50 GRUBER, M. **Functional Testing of the MIST Satellite**. Dissertação (Master's thesis) — Royal Institute of Technology, Stockholm, Sweden, 2018. Citado 3 vezes nas páginas 26, 27 e 28.

51 BARRA, S. **Functional Testing of the Electrical Power System of the MIST Satellite**. 89 p. BSc. thesis — KTH Royal Institute of Technology, Stockholm, Sweden, 2019. Disponível em: <https://s3vi.ndc.nasa.gov/ssri-kb/static/resources/FULLTEXT01%20(2).pdf>. Citado 2 vezes nas páginas 26 e 28.

52 FILIP, G. **VST104-Foxtrot**. [S.l.]: GitHub, 2020. <https://github.com/visionspacetec/VST104-Foxtrot>. Citado 2 vezes nas páginas 26 e 28.

53 TUMENJARGAL, T.; KIM, S.; MASUI, H.; CHO, M. CubeSat Bus Interface With Complex Programmable Logic Device. **Acta Astronautica**, v. 160, p. 331–342, 2019. ISSN 0094-5765. Citado 2 vezes nas páginas 26 e 28.

54 CORPINO, S.; STESINA, F. Verification of a CubeSat via Hardware-In-The-Loop Simulation. **IEEE Transactions on Aerospace and Electronic Systems**, v. 50, p. 2807–2818, 2014. ISSN 1557-9603. Citado na página 27.

55 BUSCH, S.; BANGERT, P.; DOMBROVSKI, S.; SCHILLING, K. UWE-3, In-Orbit Performance and Lessons Learned of a Modular and Flexible Satellite Bus for Future Pico-Satellite Formations. **Acta Astronautica**, v. 117, p. 73–89, 2015. ISSN 0094-5765. Citado na página 28.

56 STONE, T.; MURBACH, M.; ALEMA, R.; GILSTRAP, R. SOAREX-8 Suborbital Experiments 2015 - A New Paradigm for Small Spacecraft Communication. In: **Proceedings of the IEEE Aerospace Conference**. Big Sky, MT, USA: [s.n.], 2015. p. 1–7. Citado na página 28.

57 MARCELINO, G. M.; VEGA-MARTINEZ, S.; SEMAN, L. O.; SLONGO, L. K.; BEZERRA, E. A. A critical embedded system challenge: The floripasat-1 mission. **IEEE Latin America Transactions**, IEEE, v. 18, n. 2, p. 249–256, 2020. Citado 3 vezes nas páginas 28, 35 e 51.

58 MICROZED. **MicroZed Development Board**. [S.l.], 2025. Acesso em: 29 jan. 2026. Disponível em: <https://www.tria-technologies.com/product/microzed/>. Citado na página 29.

59 INA219. **INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface (Rev. G)**. [S.l.], 2015. Acesso em: 29 jan. 2026. Disponível em: <https://www.ti.com/lit/ds/symlink/ina219.pdf>. Citado na página 30.

60 TMP112. **TMP112x High-Accuracy, Low-Power, Digital Temperature Sensors With SMBus and Two-Wire Serial Interface in SOT563 (Rev. I)**. [S.l.], 2018. Acesso em: 29 jan. 2026. Disponível em: <https://www.ti.com/lit/ds/symlink/tmp112.pdf>. Citado na página 30.

61 LTC2983. **Multi-Sensor High Accuracy Digital Temperature Measurement System (Rev. C)**. [S.l.], 2014. Acesso em: 29 jan. 2026. Disponível em: <https://www.analog.com/media/en/technical-documentation/data-sheets/2983fc.pdf>. Citado na página 30.

62 LEPPINEN, H. Current use of linux in spacecraft flight software. **IEEE Aerospace and Electronic Systems Magazine**, v. 32, n. 10, p. 4–13, 2017. Citado 2 vezes nas páginas 31 e 32.

63 Buildroot Project. **Buildroot**. 2025. Acesso em: 02 fev. 2026. Disponível em: <https://buildroot.org/>. Citado na página 31.

64 The Yocto Project. **The Yocto Project**. 2026. Acesso em: 02 fev. 2026. Disponível em: <https://www.yoctoproject.org/>. Citado 2 vezes nas páginas 31 e 41.

65 LEPPINEN, H.; NIEMELä, P.; SILVA, N.; SANMARK, H.; FORSTéN, H.; YANES, A.; MODRZEWSKI, R.; KESTILä, A.; PRAKS, J. Developing a linux-based nanosatellite on-board computer: Flight results from the aalto-1 mission. **IEEE Aerospace and Electronic Systems Magazine**, v. 34, n. 1, p. 4–14, 2019. Citado na página 32.

66 BAKKEN, S.; HONORé-LIVERMORE, E.; BIRKELAND, R.; ORLANDIć, M.; PRENTICE, E. F.; GARRETT, J. L.; LANGER, D. D.; HASKINS, C.; JOHANSEN, T. A. Software development and integration of a hyperspectral imaging payload for hypso-1. In: **2022 IEEE/SICE International Symposium on System Integration (SII)**. [S.l.: s.n.], 2022. p. 183–189. Citado 2 vezes nas páginas 32 e 35.

67 GALARRAGA, M.; LEFEBVRE, C.-A.; PEREZ-CERROLAZA, J.; PASCUAL, J. A. Linux for safety-critical systems: A survey. **Journal of Systems Architecture**, v. 169, p. 103598, 2025. ISSN 1383-7621. Disponível em: <https://www.sciencedirect.com/science/article/pii/S138376212500270X>. Citado na página 32.

68 CRATERE, A.; GAGLIARDI, L.; SANCA, G. A.; GOLMAR, F.; DELL'OLIO, F. On-board computer for cubesats: State-of-the-art and future trends. **IEEE Access**, v. 12, p. 99537–99569, 2024. Citado 2 vezes nas páginas 32 e 44.

69 Xilinx. **Zynq-7000 SoC Overview**. [S.l.], 2018. Datasheet DS190. Acesso em: 02 fev. 2026. Disponível em: <https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview>. Citado 2 vezes nas páginas 34 e 39.

70 GONZALEZ, C. E.; ROJAS, C. J.; BERGEL, A.; DIAZ, M. A. An architecture-tracking approach to evaluate a modular and extensible flight software for cubesat nanosatellites. **IEEE Access**, v. 7, p. 126409–126429, 2019. Citado 3 vezes nas páginas 35, 43 e 44.

71 MARCELINO, G. M.; Morsch Filho, E.; MARTINEZ, S. V.; SEMAN, L. O.; BEZERRA, E. A. In-orbit preliminary results from the open-source educational nanosatellite floripasat-i. **Acta Astronautica**, v. 188, p. 64–80, 2021. ISSN 0094-5765. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0094576521003489>. Citado 2 vezes nas páginas 35 e 51.

72 Advanced Micro Devices, Inc. **Vivado Design Suite User Guide: Design Flows Overview**. [S.l.], 2025. Acesso em: 04 fev. 2026. Disponível em: <https://docs.amd.com/r/en-US/ug892-vivado-design-flows-overview/>. Citado na página 37.

73 Xilinx, Inc. **Vivado Design Suite User Guide: Using the Vivado IDE**. [S.l.], 2020. Acesso em: 04 fev. 2026. Disponível em: <https://www.xilinx.com/support/documents/sw_manuals/xilinx2020_1/ug893-vivado-ide.pdf>. Citado na página 37.

74 Advanced Micro Devices, Inc. **PetaLinux Tools Documentation: Reference Guide**. [S.l.], 2025. Acesso em: 04 fev. 2026. Disponível em: <https://docs.amd.com/r/en-US/ug1144-petalinux-tools-reference-guide>. Citado na página 41.

75 Yocto Project. **Yocto Project Documentation**. 2026. Documentação oficial do Yocto Project. Acesso em: 04 fev. 2026. Disponível em: <https://docs.yoctoproject.org/>. Citado 2 vezes nas páginas 41 e 42.

76 Intel Corporation. **Industrial I/O Subsystem Documentation**. 2026. Documentação oficial do subsistema Industrial I/O (IIO) do kernel Linux. Acesso em: 04 fev. 2026. Disponível em: <https://www.kernel.org/doc/html/latest/driver-api/iio/index.html>. Citado na página 42.

77 LIKELY, G. **Device Tree Usage Model**. 2026. Documentação oficial do kernel Linux. Acesso em: 06 fev. 2026. Disponível em: <https://www.kernel.org/doc/html/latest/devicetree/usage-model.html>. Citado na página 42.

78 The systemd Project. **systemd: System and Service Manager**. 2025. Acesso em: 6 fev. 2026. Disponível em: <https://systemd.io/>. Citado na página 44.

79 ZeroMQ Project. **ZeroMQ: An open-source universal messaging library**. 2025. Acesso em: 6 fev. 2026. Disponível em: <https://zeromq.org/>. Citado na página 44.

80 EUGSTER, P. T.; FELBER, P. A.; GUERRAOUI, R.; KERMARREC, A.-M. The many faces of publish/subscribe. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 35, n. 2, p. 114–131, jun. 2003. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/857076.857078>. Citado na página 44.

81 Advanced Micro Devices, Inc. **7-Series FPGAs and Zynq-7000 SoC XADC: Dual 12-Bit 1 MSPS Analog-to-Digital Converter**. [S.l.], 2022. Acesso em: 7 fev. 2026. Disponível em: <https://docs.amd.com/r/en-US/ug480_7Series_XADC/7-Series-FPGAs-and-Zynq-7000-SoC-XADC-Dual-12-Bit-1-MSPS-Analog-to-Digital-Converter-User-Guide Citado na página 46.

82 Grafana Labs. **Grafana: Open Source Analytics and Monitoring Platform**. 2026. Acesso em: 6 fev. 2026. Disponível em: <https://grafana.com/oss/grafana/>. Citado na página 48.

83 SpaceLab. **EPS2 User Manual**. Github, 2023. Acesso em: 6 fev. 2026. Disponível em: <https://github.com/spacelab-ufsc/eps2/releases/download/v0.4/slb-eps2-doc-v0.4.pdf>. Citado na página 51.

84 SpaceLab. **TTC2 User Manual**. Github, 2025. Acesso em: 6 fev. 2026. Disponível em: <https://github.com/spacelab-ufsc/ttc2/releases/download/v1.0/slb-ttc2-doc-v1.0.pdf>. Citado na página 51.

85 SpaceLab. **OBDH2 User Manual**. Github, 2022. Acesso em: 6 fev. 2026. Disponível em: <https://github.com/spacelab-ufsc/obdh2/blob/master/doc/user_manual/slb-obdh2-doc-v0.10.pdf>. Citado 2 vezes nas páginas 51 e 52.

86 QUEIROZ, K.; DIAS, S.; DUARTE, J.; CARVALHO, M. Uma soluÇÃo para o sistema brasileiro de coleta de dados ambientais baseada em nanossatÉlites. **HOLOS**, v. 7, p. 132–142, 12 2018. Citado na página 54.

87 VALLADO, D.; CRAWFORD, P. Sgp4 orbit determination. In: **AIAA/AAS Astrodynamics Specialist Conference and Exhibit**. [s.n.], 2008. Disponível em: <https://arc.aiaa.org/doi/abs/10.2514/6.2008-6770>. Citado na página 59.

Apêndices

APÊNDICE A – ARTIGO APRESENTADO EM: *LATIN AMERICAN CUBESAT WORKSHOP 2024*

Nas próximas páginas, o artigo apresentado na conferência *Latin American Cubesat Workshop 2024* é disponibilizado.

LATIN AMERICAN CUBESAT WORKSHOP
SALVADOR, BR
OCTOBER 2024

Progress Report: FlatSat Platform for GOLDS-UFSC and Future Missions

Carlos Augusto Porto Freitas[1], Eduardo Dutra Bitencourt[1], João Cláudio Elsen Barcellos*[1], Gabriel Mariano Marcelino†[1], and Eduardo Augusto Bezerra‡[1]

[1]Department of Electrical and Electronics engineering, Federal University of Santa Catarina

Abstract

CubeSats are small satellites, known as "nanosatellites," that are used for Earth observation, scientific research, and remote sensing applications. Despite their growing popularity, CubeSat missions usually suffer from a significantly high failure rate. To address the issues encountered in CubeSats, a more rigorous verification and validation (V&V) process of the system can be adopted, using a FlatSat. A FlatSat is an electrical model of the satellite with the subsystems mounted side by side on a workbench. This approach is gaining popularity in small satellite applications due to its ability to integrate and test systems at a level that would otherwise be complex and error-prone. An example of a FlatSat platform is the one developed at SpaceLab/UFSC for testing the FloripaSat family of satellite platforms. This FlatSat was designed to be highly configurable, capable of testing and monitoring CubeSat subsystems, reading operation data, and injecting faults into communication buses. However, the FlatSat platform itself has not yet been fully tested, and no full satellite mission has been developed using it. This paper presents the initial exploration of the platform, starting with basic electrical testing and progressing to reading real data from a CubeSat subsystem. The tests are divided into three parts: (i) first, electrical and power tests are performed to validate the physical structure of the FlatSat board; (ii) then, the MicroZed is attached for initial design validation and on board sensor tests; (iii) finally, satellite's subsystems are connected to conduct temperature and current reading tests, as well as interface with a subsystem simulation application running on MicroZed. To carry out these tests, embedded Linux and the MicroZed's programmable logic are used, providing a highly

*Orcid: https://orcid.org/0000-0002-2550-8674
†Orcid: https://orcid.org/0000-0001-5204-2040
‡Orcid: https://orcid.org/0000-0002-2191-6064

reconfigurable and versatile platform for testing and data acquisition. This configuration allows for future expansion of FlatSat functionality, including more complex applications than those initially envisioned. The first part of the tests aims to create a validated and secure environment for implementing the MicroZed and satellite modules, as well as conducting their tests. These tests are designed to simulate CubeSat operational conditions, and the results aim to validate the basic use of FlatSat, paving the way for future applications.

1 Introduction

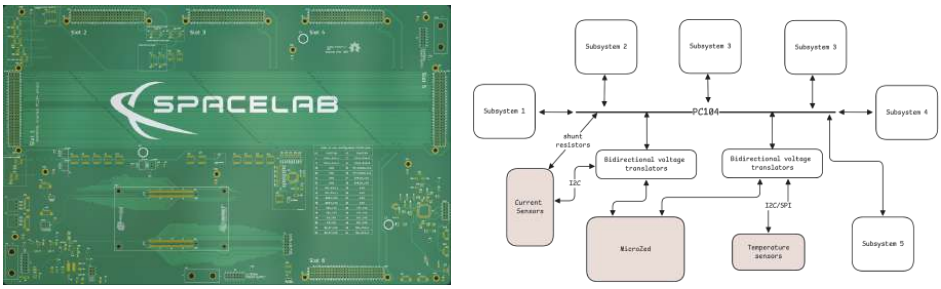
The space industry's small satellite development has grown explosively over the years, in particular CubeSats, which are nanosatellites, with lower cost and complexity, played a important role in it [1]. Because of these characteristics, CubeSats have shown great importance in the area of technology demonstration and in-orbit validation [1]. In addition, these satellites are widely used in student-driver missions, contributing to student training while fostering technological and scientific development [2].

Even tought these satellites can be developed relatively fast and inexpensively, numerous issues can arise both during development and throughout their mission. For instance, launch-related mechanical failures, software bugs, and radiation-induced faults are just a few of the potential complications. To mitigate these, a extensive Verification and Validation (V&V) campaign can be used [3], which employs various methods to validate the system against likely and unlikely scenarios, observing its behavior in a way to indentify vulnerabilities.

One valid option for the V&V campaign is the usage of a FlatSat, which is a eletrical model of the satellite, where the satellite's subsystems are assembled and connected side by side over a workbench [4]. Such platforms, according to [4], have grown in popularity among small satellite applications. FlatSats provide a broad view of the system operability, easing the V&V process from both the hardware and software perpective, they allow the early detection of some of the problems mentioned previously, as seen in [5, **?**]. For instance, software bugs and system-level integration testing can be quite complex and error prone without proper testing instrumentation, especially if the team lacks experience, as it happened in FloripaSat-1 [6] V&V campaign.

During Latin American CubeSat Workshop (LACW) 2022, a FlatSat was proposed at [7] with the aim of aiding the development of satellites at the SpaceLab laboratory at the Federal University of Santa Catarina. The project support up to six subsystems, including service modules and payloads, interfacing with a MicroZed [8] and build-in circuits for sensing, monitoring and controlling the system. The aim of the MicroZed is to make it possible to simulate the inputs and outputs of satellite modules in order to observe the response of a dispositive under test. Figure 1 describes how the project is architectured.

In order to utilize the FlatSat implementation proposed by [7] to its full potential, with similar funcionality to an Electrical Ground Support Equipment (EGSE), there is a need to validate the platform itself. This first step aims to create a validated and secure enviroment to actually conduct tests and is composed of the basic eletrical validation of the Printed Circuit Board (PCB), which includes checking for design and fabrication flaws, checking the power circuitry operation and evaluate the built-in



(a) FlatSat2 Platform (b) Architecture diagram

Figure 1: FlatSat Architecture

sensors of the board.

Following the platform validation, the plan is to attach a satellite subsystem to collect real operation data, specifically ambient temperature and current measurements. Naturally, a subsystem will need to communicate with other subsystems, which will be simulated by the MicroZed. The MicroZed will also collect data from the onboard sensors and store it to further processing and analysis. Additionally, subsystem data obtained from the simulated communication the MicroZed will also be presented.

As said previously the platform has yet to be tested and, currently, there is no application developed for it, thus conducting this initial exploration is necessary to validate the basic funtionality of the proposed FlatSat. This study paves the way for more complex applications, including EGSE-like operation, futhermore, it also allows the platform usage for testing GOLDS-UFSC and future missions, enhancing their V&V campaigns.

The remaining of this work is organized as follows. In section 2, the methodology used to validate and collect data of the FlatSat is described. The process of testing, validating and collectiong operational data of the platform are desbribed in detail in section 3. Additionally, obtained results are also discussed in section 3. Further discussions and directions are shown in section 4. Finally, in section 5, conclusions are presented.

2 Methodology

Taking in account the aforementioned objectives, a test flowchart becomes necessary, mainly to ensure result validity for each test and to provide an overall test suite organization. In addition, it also helps to avoid possible unrelated sources of error interfering with the results. The flowchart can be seen at Figure 2. The tests follow the SpaceLab's methodology for initial subsystem testing, which can be seen at [0, 0], additionaly they also follow the European Cooperation for Space Standardization (ECSS) testing standard [0] when applicable.

2.1 Electrical validation

The electrical validation process of the PCB is essential for identifying manufacturing and

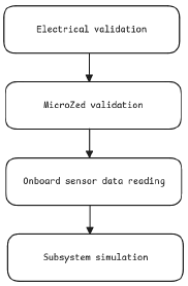


Figure 2: Tests flowchart

assembly errors, such as short circuits, open circuits, poor soldering and missing components. It also helps uncover design flaws that could lead to permanent damage to connected modules or cause malfunctions. Initially, considering the need to guarantee this safety, continuity and connection inspection tests were carried out in order to validate the design and manufacture of the board [0]. To do this, all the connections on the board were tested using a multimeter, along with the design schematic (in order to check that the connection is consistent with what was planned). After this first stage, it was possible to carry out power and signal tests along the board, i.e. to ensure that the power supply and the signal applied to the FlatSat are able to reach all the expected points [0]. To carry out this experiment, stimuli was applied via the power supply bench and then the voltages at the expected points were measured using a multimeter and an oscilloscope.

2.2 MicroZed validation

As mentioned earlier, the FlatSat has a MicroZed embedded in its design, this addition extends quite a lot the functionality of the platform. However, the System on Module (SOM) has to be tested before the integration with the board, as to eliminate any possibility of hardware failures unrelated to the FlatSat. Testing it also involves booting the SOM with a embedded linux operating system as well as the Zynq's Programmable Logic (PL) design. Those were used for all operations of the MicroZed, such as simulating subsystems or collection sensor data.

Choosing Linux as its operating system was done mainly for future use, as Linux provides a highly customizable system with a very large ecosystem of libraries, device drivers and packages. By using it, development time is reduced, especially as application complexity grows towards a EGSE-like functionality.

Next, the MicroZed was integrated into the platform, and its operation reevaluated to identify any errors introduced by the FlatSat design. Once any present errors are addressed, the test suite can progress to collecting data from the onboard sensors. This next step intends to only verify the basic functionality of the sensors and its communication with the MicroZed, since a more thorough test is done later in a real operation setting, with satellite subsystems integrated to the FlatSat.

2.3 Functional tests with real subsystems

Following the test suite, the GOLDS-UFSC mission's Electrical Power System (EPS), named as EPS 2.0, was connected to the FlatSat as a representative subsystem. Being part of the FloripaSat-2 platform, proposed in [0], this subsystem has been thoroughly tested in the previous FlatSat version and during integration tests with the Engineering Qualification

Model (EQM) of GOLDS-UFSC satellite.

As a well known module, the usage of it as a testing reference facilitates the validation of the FlatSat, since its usual behavior is already documentated.

After the EPS validation step, it was possible to test the sensors in a real setting. In this sense, the Environmental Data Collector (EDC), proposed in [0], was used as a payload. As is the case with EPS, EDC has been thorougly tested and has a well known behavior, since it is the main payload for the GOLDS-UFSC mission [0] and was also tested with the EQM.

The test aims to collect readings from the built-in sensors and verify its accuracy and feasibility. In it, there was two measurements being collected concurrently, one being ambient temperature readings by the TMP112 [0] sensor and the other one being EDC current consumption readings using the INA219 [0]. Measurements of the EDC current were also made using a Keithley DMM6500 digital multimeter [0] to serve as a reference.

Lastly, the test suite's final test aims to collect real operation data from the EPS, by simulating the FloripaSat-2 platform OnBoard Data Handling (OBDH) module [0] through the MicroZed. During simulation, the SOM gathered the available telemetry data obtained by mimicking OBDH commands in the EPS's Inter Integrated Communication (I2C) interface.

The OBDH simulation was running simultaneously with the measurements from the previous test, so that the EPS telemetry data could be compared with the platform's built-in sensors data. Perfoming both tests concurrently served to assess MicroZed's ability to handle multiple applications running simultaneously, which will be used extensibly in future works.

Furthermore, having access to the satellite communication busses allow the FlatSat's usage for communication and fault injection analysis as done in [0], as well as to do a performance characterization of the bus like the one made in [5]. Therefore, the proposed test also serves to evaluate the feasibility of conducting these intricate tests and analysis.

3 Implementation & Results

In order to test a system, in this case the FlatSat platform, it is imperative to build a sequence of operations that are consistent with the system and the related objective. For this article, the test flow described at Figure 2 was followed.

3.1 Electrical Tests

With a view to electrical validation and the safety of the other devices, the first step was to ensure that there were no short circuits within the power systems and then to investigate the signal connections on the board. To carry out both steps, an inspection was carried out using a multimeter and the original schematic of the board. As a result, it was possible to find several short circuits in the voltage translators, as well as one between one of the input voltages and ground. These problems were caused during the soldering stage of the FlatSat components, making it necessary to remove excess solder flux and replace some of the voltage translators.

After ensuring that Flatsat was electrically coherent with what had been developed, it was

necessary to check the routing on the board, so that it would be possible to understand and analyze how it worked. An inconsistency was found in relation to the board's purpose: the pinouts related to the communication protocols were not consistent with those found on Zynq's available peripheral controllers for the Processing System (PS), making it impossible to use the board for these protocols via PS. In order to continue testing the rest of the board, access via PL was used.

Finally, with regard to the system's power system, there was two problems in the circuit, one relating to the Power_Enable power input, making it necessary to use an external source to supply 3.3V directly. While the other one was a misdirection of the Zynq's IO banks enable pin, which is an output signal generated by the SOM, however, the FlatSat circuitry considered this signal to be an input, therefore, to avoid any issues, this part of the power circuitry was not populated.

3.2 MicroZed Setup

After making sure that the platform's power circuitry and the SOM itself work as intended the next step is to create a custom embedded linux image and hardware design. For that, it was used Xilinx's Petalinux and Vivado Design Suite tools. Using Vivado, a simple hardware design was made, which basically routes the relevant peripherals, like the I2C controllers, to the sensors and subsystems interfaces.

For the Progamming System (PS) side of Zynq, a custom embedded linux was made, which includes essential libraries, device drivers for the FlatSat sensors, custom services and user applications. The image includes a sensor reading service, which provide a simple interface to enable specific sensor readings, manage log files, etc. Futhermore, it also have a application for OBDH simulation that communicates with EPS, requesting its telemetry periodically.

With the SOM configured and working individually, it was attached to the FlatSat aiming to test the software applications, also serving as a preliminary test for the onboard sensors. To assess the INA219 functionality and the sensor reading service, resistors and one light emitting diode (LED) were connected to the power busses of the platform, which were powered by a digital power analyzer. Additional resistors were added in 30 second intervals, as to change the current drawn by the LED and test the responsiveness of the current sensor.

The results came in as expected, the service was able to read data from the onboard sensors, which includes temperature measurements from the TMP112 and current, bus voltage, shunt voltage and power measurements from the INA219. At this stage, it was discovered an issue regarding current readings, which had a greater inacurracy as the drawn current gets smaller, particulary when the current is less than 10 milliamperes (mA). This happens mainly due to the device driver resolution, which only provides measurements that are integer multiples of milliamperes, therefore changes on current of less then 1 mA will most likely be unnotice. More discussions about measurement adequacy are made at subsection 3.3.

3.3 Functional Tests

As per the test suite, a test setup was made to allow the execution of functional tests with real subsystems, these will involve mainly the EDC and EPS subsystems, both having well documented behaviors, which aids the valida-

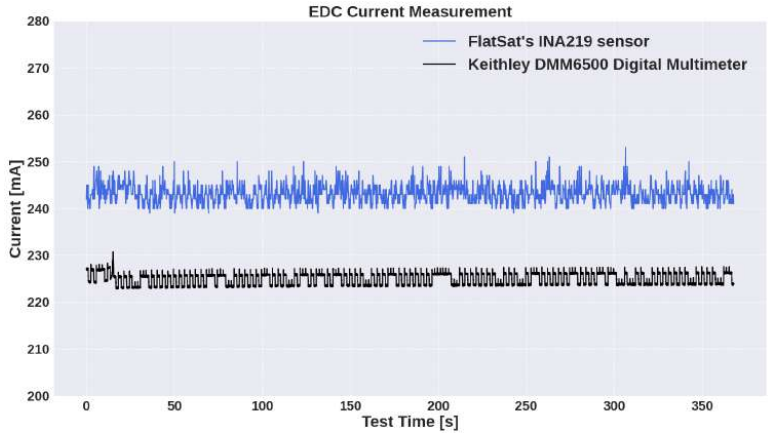


Figure 3: EDC current test readings.

tion process of the FlatSat. A diagram describing the setup can be seen at Figure 4, for simplicity the diagram does not include the power connections besides the 5V payload bus, which is produced by EPS.

Both EPS and EDC were powered by a lithium-ion battery, being the same as the one used in the GOLDS-UFSC mission satellite. As for the MicroZed and the onboard sensors, they were powered by a digital power analyzer connected to FlatSat's power circuitry, which had to be adapted due to the problems described at subsection 3.1.

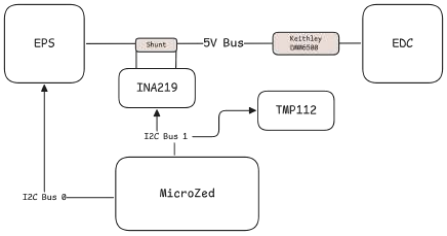
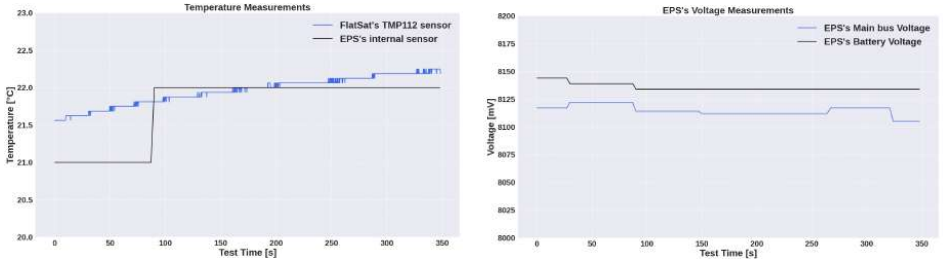


Figure 4: Test setup diagram.

As stated in section 2 this setup is used for two tests to run simultaneously, where the first one is composed of current measurements to monitor EDC power consumption, taken in 250 millisecond intervals by both the INA219 and DMM6500, and the second consists of reading EPS telemetry and TMP112 data every three seconds.

The EDC consumption results can be seen at Figure 3. The comsumption itself is in line with EDC's datasheet [0] and the one observed during tests of GOLDS-UFSC mission. By looking at the chart it becomes clear that the INA219 readings, represented by the blue curve, have quite a bit of deviation from DMM6500's reference readings, with the largest sample to sample deviation being of, approximately, 27 mA, the lowest 13 mA and the average 18 mA, which represent 10.81%, 5.37% and 7.51% of relative error, respectively.

This difference can be explained by driver details and overall precision difference between the devices. As discussed previously, the



(a) Temperature readings. (b) EPS voltage readings.

Figure 5: Subsystem simulation readings.

INA219 driver used, which is the linux kernel device driver for INA2xx devices [0], can only provide integer multiples of milliamperes, moreover, it also sets a default calibration configuration on the sensor, those two factors can explain the deviation seen at Figure 3.

To mitigate this, a custom device driver can be made to provide a runtime calibration configuration and microampere reading resolution. However, that would be quite time consuming and would require a specific analysis to determine whether it is justified. After all, if a highly precise reading is imperative to the test, using the DMM6500 or a similiar device directly would be a prefered approuch.

Concurrently to the EDC consumption test, the MicroZed had a running application that simulates OBDH data requests to EPS and was collecting ambient temperature readings. The application simulates OBDH data requests using it's own firmware drivers and tasks [0] developed to communicate with EPS, minor modifications had to be done at the low level interfaces due to OBDH running FreeRTOS as an operating system, as opposed to the MicroZed running embedded linux. The data requests were composed of battery voltage, main bus voltage and temperature, all acessible in the EPS parameter table [0] through its I2C interface.

A graphic containing both the ambient temperature from TMP112 and EPS microcontroller temperature measurements can be seen at Figure 5a. The purpose of the comparison is to quantify the difference of temperature experienced by EPS and the actual ambient temperature. This difference is relevant not only to understand the heat produced by the board, but also to verify the behavior of the subsystem, since it takes in to account the current temperature to enable specific modules. By looking at the Figure 5a, it becomes clear that the exposed temperature paremeter is a integer value, nevertheless, it still allows for a rough estimate of its behavior, which, in this case, resembles very much the ambient temperature readings from TMP112.

As for the voltage paremeters from Figure 5b, both measurements are obtained directly from EPS and represent the FlatSat's capability to track the system's parameters and facilitate behavioral analysis of all available data, which can be used to perform more sophisticated tests, as described in subsection 2.3. The readings itself are not that interesting, since the test time was about 6

minutes and, for a good understanding of the battery behavior over time, much more time would be needed, moreover, charge measurements would be better for that type of test as opposed to voltage ones. An interesting result taken from the Figure 5b is to observe the power loss due to the EPS circuitry, which manifests in the voltage drop between battery voltage and main bus voltage.

4 Discussions and Directions

This article developed a test suite in order to do a follow up of the work presented at LACW 2022 [7], in particular, it aims to perform the initial validation of the proposed FlatSat.

As the test suite was carried out, the validation importance became apparent, specially the electrical validation, which, if had not been done, would result in several functional issues and could cause permanent damage to the platform components. In addition, the schematic reassessment was also critical to the tests, since it unveiled routing issues that would hinder MicroZed's ability to communicate with the subsystems.

For the validation's software side, the decision to start with embedded linux has facilited considerably the application and service development, it also eased testing them. Furthermore, the flexibily that a linux based operating system brings has allowed the applications and services to be integrated in a much simpler way, which will be crucial to the platform as new tests and services get added.

Moreover, effort on developing these new features are already being made. Examples of them are: adding a internal database to MicroZed, as well as a cloud based one, to organize test data, extending the simulation application to replicate more subsystems, adding support to other sensors, MicroZed remote access through Secure Shell (SSH), among other software applications. Extending hardware capabilities of the platform through daughter boards is also planned, with a major example being a board to interface other connector types with the PC104.

5 Conclusion

This article presented the beginning of the V&V process of the FlatSat platform proposed by [7]. In addition, applications and services were developed to utilize the platform's builit-in features.

A overall design revision was made, including a schematic review, in which both problems and improvement points were found. For instance, the MicroZed routing and FlatSat's power circuitry will have to be corrected in the next version of the platform.

Moreover, an effort to test and validate the platform's onboard capabilites, specially the MicroZed and sensor integration, was carried. It covered both the hardware and software side, starting with electrical tests of the platform, passing to basic software applications embedded in the SOM, to, finally, collecting real operation data from both onboard sensors and real CubeSat subsystems.

From obtained results, it was possible to see that the addition of onboard sensors and the MicroZed provides much more data retriving capabilities to the platform, which eliminate the need to rely on the subsystem's sensors to monitor the whole system's behavior. Even thought the sensors accuracy have some

caveats, like in the INA219 case, it, nevertheless, can be useful for estimates and overall monitoring. In addition, MicroZed's presence provide a wide range of possibilities, including test automation and subsystem simulations, which will be explored in order to enhance V&V campaigns for future missions.

Finally, lessons learned and future directions were discussed. As addressed throughout this work, the FlatSat has a great potential for aiding V&V campaigns, however, there still much effort to be done to realize that potential, for that, possible additional features to be developed were presented. Furthermore, the project of the platform is open source and can be found at: `https://github.com/spacelab-ufsc/flatsat-platform2`.

Acknowledgements

The authors want to to acknowledge that this work was supported by CNPq/Brazil (National Council for Scientific and Technological Development - `http://www.cnpq.br`), and by AEB (Brazilian Space Agency - `http://www.aeb.gov.br`).

References

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

[2] R. M. Millan, R. von Steiger, M. Ariel, S. Bartalev, M. Borgeaud, S. Campagnola, J. C. Castillo-Rogez, R. Fléron, V. Gass, A. Gregorio, D. M. Klumpar, B. Lal, M. Macdonald, J. U. Park, V. Sambasiva Rao, K. Schilling, G. Stephens, A. M. Title, and J. Wu, "Small Satellites for Space Science: A Cospar Scientific Roadmap," *Advances in Space Research*, vol. 64, pp. 1466–1517, 2019.

[3] J. Bouwmeester, A. Menicucci, and E. Gill, "Improving CubeSat reliability: Subsystem redundancy or improved testing?" *Reliability Engineering & System Safety*, vol. 220, p. 108288, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832021007584

[4] J. C. E. Barcellos, A. W. Spengler, L. O. Seman, R. D. C. e Silva, H. P. Roldán, and E. A. Bezerra, "Flatsat platforms for small satellites: A systematic mapping and classification," *IEEE Journal on Miniaturization for Air and Space Systems*, 2023.

[5] J. P. Monteiro, R. M. Rocha, A. Silva, R. Afonso, and N. Ramos, "Integration and Verification Approach Of ISTSat-1 CubeSat," *Aerospace*, vol. 6, p. 131, 2019.

[6] M. G. Mariano, F. E. Morsch, M. S. Vega, D. M. A. M. Pio, S. L. Oriel, S. L. Kessler, and B. E. Augusto, "Qualification and Validation Test Methodology of the Open-Source CubeSat FloripaSat-I," *Journal of Systems Engineering and Electronics*, vol. 31, pp. 1230–1244, 2020.

[7] J. C. E. Barcellos, B. F. Ribeiro, A. W. Spengler, L. O. Seman, and E. A. Bezerra, "Development of a flatsat platform

for golds-ufsc and future missions," in *Companion Book of the IAA Latin American CubeSat Workshop*, 2022.

[8] MicroZed, *FY23*$_8 00_M icroZed_P roduct_B rief_r 2$, *Avnet*, 2022, *available in:* .

SpaceLab, "OBDH2 User Manual," `https://github.com/spacelab-ufsc/obdh2/blob/master/doc/user_manual/slb-obdh2-doc-v0.10.pdf`, 2022.

——, "EPS2 User Manual," `https://github.com/spacelab-ufsc/eps2/blob/master/doc/slb-eps2-doc-v0.4.pdf`, 2023.

ECSS Secretariat, "ECSS-E-ST-10-03C – Testing," ECSS-ESTEC, Technical Report TBD, May 2022.

G. M. Marcelino, A. M. P. de Mattos, J. C. E. Barcellos, B. F. Ribeiro, L. O. Seman, E. Morsch Filho, and E. A. Bezerra, "Floripasat-2: An open-source platform for cubesats," *IEEE Embedded Systems Letters*, 2023.

K. Queiroz, S. Dias, J. Duarte, and M. Carvalho, "Uma soluÇÃo para o sistema brasileiro de coleta de dados ambientais baseada em nanossatÉlites," *HOLOS*, vol. 7, pp. 132–142, 12 2018.

TMP112, *TMP112x High-Accuracy, Low-Power, Digital Temperature Sensors With SMBus and Two-Wire Serial Interface in SOT563 (Rev. I)*, Texas Instruments, 2018, available in: `https://www.ti.com/lit/ds/symlink/tmp112.pdf`.

INA219, *INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface (Rev. G)*, Texas Instruments, 2015, available in: `https://www.ti.com/lit/ds/symlink/ina219.pdf`.

DMM6500, *DMM6500 6½-Digit Bench/System Digital Multimeter*, Keithley, 2022, available in: `https://download.tek.com/datasheet/1KW-61315-1_DMM6500_Datasheet_050924.pdf`.

C. A. Conceicao, F. Mattiello-Francisco, and C. L. Batista, "Dependability verification of nanosatellite embedded software supported by a reusable test system," in *2016 Seventh Latin-American Symposium on Dependable Computing (LADC)*, 2016, pp. 157–163.

E. D. Team, "doc," `https://gitlab.com/edc_team/doc`, 2022.

L. Felten, *Kernel driver ina2xx*, 2024, available in: `https://docs.kernel.org/hwmon/ina2xx.html`.

SpaceLab, "obdh2," `https://github.com/spacelab-ufsc/obdh2`, 2024.