TRABALHO DE CONCLUSÃO DE CURSO

Projeto de Hardware e Desenvolvimento de Software para Computador de Bordo para CubeSats

Eduardo Vaz Fagundes Rech

Brasília, Novembro de 2022

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

TRABALHO DE CONCLUSÃO DE CURSO

Projeto de Hardware e Desenvolvimento de Software para Computador de Bordo para CubeSats

Eduardo Vaz Fagundes Rech

Trabalho de conclusão de curso submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro em Engenharia de Computação

Banca Examinadora

Prof. Daniel Chaves Café, FT/UnB _____
Orientador

Prof. Adson Ferreira da Rocha Banca 1, ENE/UnB _____
Examinador Interno

Prof. Gilmar Silva Beserra Banca 2, FGA/UnB _____
Examinador interno

FICHA CATALOGRÁFICA

RECH, EDUARDO VAZ FAGUNDES
Projeto de Hardware e Desenvolvimento de Software paraComputador de Bordo para CubeSats [Distrito Federal] 2022.
xvi, 30 p., 210 x 297 mm (ENE/FT/UnB, Engenheiro, Engenharia Elétrica, 2022).
Trabalho de conclusão de curso - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica
1. Trabalho de Graduação - Universidade de Brasilia
2. Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

RECH, E. V. F. (2022). *Projeto de Hardware e Desenvolvimento de Software paraComputador de Bordo para CubeSats*. Trabalho de conclusão de curso, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 30 p.

Eduardo Vaz Fagundes Rech
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

RESUMO

Este trabalho trata da concepção e implementação de um computador de bordo (OBC) para um Cubesat, capaz de delegar tarefas a outros subsistemas, assim como agendar e executar os comandos recebidos da estação terrestre, implementado no laboratório LODESTAR da Universidade de Brasília (UnB).

O algoritmo B-dot, que atualmente está sendo utilizado no laboratório, exige operações em ponto flutuante em sua implementação, que sobrecarregam o OBC atual. Este algoritmo desempenha um papel importante no sistema, pois realiza os cálculos do controle de atitude necessários para o funcionamento correto do satélite. Uma das finalidades deste projeto 'e a migração para um microcontrolador com módulo dedicado a essas operações.

A concepção do projeto teórico contempla a categorização dos pequenos satélites e seus principais subsistemas, além das escolhas feitas para a seleção do microcontrolador e dos componentes utilizados no projeto. Para o desenvolvimento do software, foi utilizado o FreeRTOS como kernel do sistema operacional e foi apresentada a organização do "planner"com alguns exemplos de teste de como cada padrão de tarefa deve se comportar.

Foi definido uma organização do protocolo de comandos para o satélite, os comandos são de classe para leitura e escrita de memória e acesso aos periféricos e subsistemas dos satélites. Foram testados os comandos *BUS_INJECTION*, *MEMORY_READ_WRITE*, *TELEMETRY*, *PING* e *ECHO* com tamanho variando entre 5 a 28 bytes e as respostas esperadas foram de fato obtidas.

Ao final os testes ocorreram conforme o esperado, ou seja, o OBC foi capaz de receber, validar e executar ações conforme as mensagens recebidas. Alguns testes, porém, não obtiveram resposta do sistema, indicando uma falha na comunicação. O projeto, portanto, deverá ter continuidade visando melhorias e soluções às possíveis falhas.

ABSTRACT

This work deals with the design and implementation of an onboard computer (OBC) for a Cubesat, capable of delegating tasks to other subsystems, as well as scheduling and executing the commands received from the ground station, implemented in the LODESTAR laboratory of the University of Brasília (UnB) .

The B-dot algorithm, which is currently being used in the laboratory, requires floating-point operations in its implementation, which overload the current OBC. This algorithm plays an important role in the system, as it performs the necessary attitude control calculations for the correct

functioning of the satellite. One of the purposes of this project is the migration to a microcontroller with a module dedicated to these operations.

The conception of the theoretical project includes the categorization of small satellites and their main subsystems, in addition to the choices made for the selection of the microcontroller and the components used in the project. For the development of the software, FreeRTOS was used as the operating system kernel and the organization of the "planner"was presented with some test examples of how each task pattern should behave.

An organization of the command protocol for the satellite was defined, the commands are of a class for reading and writing memory and access to peripherals and satellite subsystems. The commands *BUS_INJECTION*, *MEMORY_READ_WRITE*, *TELEMETRY*, hspace0.01cm *PING* and *ECHO* with sizes ranging from 5 to 28 bytes and the expected responses were actually obtained.

At the end, the tests occurred as expected, that is, the OBC was able to receive, validate and execute actions according to the messages received. Some tests, however, did not obtain a response from the system, indicating a failure in communication. The project, therefore, must be continued, aiming at improvements and solutions to possible failures.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 UNIVERSO DOS SATÉLITES

Satélite artificial é qualquer corpo feito pelo ser humano e colocado em órbita ao redor da Terra ou de qualquer outro corpo celeste.

A história dos satélites artificiais começou no século XX, no período conhecido como a "Corrida Espacial"entre os Estados Unidos e a União Soviética, no âmbito da Guerra Fria. Em 4 de outubro de 1957, os soviéticos lançaram o primeiro satélite artificial da Terra, o Sputnik I, e em 3 de novembro de 1957, o Sputnik II. Em 31 de janeiro de 1958, os Estados Unidos lançaram seu primeiro satélite, o Explorer 1. O primeiro satélite brasileiro, denominado "Satélite de Coleta de Dados"(SCD-1), foi lançado em 1993.

Na atualidade, os satélites artificiais são desenvolvidos por meio de sistemas tecnológicos de ponta e desempenham importantes funções, colaborando com o avanço científico em diversas áreas do conhecimento, tais como comunicações, navegações, geológicas, climáticas, militares, dentre outras[1].

Esses equipamentos são levados até a altura desejada a bordo de um ônibus espacial ou acoplados a um foguete. Atingida essa altura, o satélite é acelerado para que atinja a velocidade suficiente para permanecer em órbita, ocupando posições ao redor da Terra onde não existe atrito com o ar. Isso garante que não haja perda de energia cinética, mantendo o satélite seu movimento por inércia.

Até hoje milhares de satélites foram lançados ao espaço mas a maioria encontra-se desativada. Quando ocorrem falhas no lançamento ou no próprio satélite, partes deles podem ficar orbitando o planeta por tempo indeterminado, formando o lixo espacial [2].

1.2 EXPLORAÇÃO ESPACIAL

Exploração espacial é o conjunto de esforços feitos pelo homem com o objetivo de estudar o espaço e seus astros. Para isso, recorre-se da tecnologia disponível, como naves espaciais, satélites artificiais ou sondas espaciais, e, muitas vezes também do próprio homem.

Nas últimas décadas, a exploração espacial modificou a vida moderna tanto pelas descobertas científicas que surgiram como resultado dessa exploração quanto pela criação de inúmeros novos mercados. Hoje, por exemplo é praticamente impossível falar em comunicação a longas distâncias sem de certa forma citar um componente ligado ao espaço.

Como mostra a Figura 1.1, a maior parte dos satélites é usada para a comunicação comercial,

com destaque o uso para observação da Terra, o qual inclui aplicações para a agricultura, detecção de mudanças na superfície, meteorologia e levantamento de recursos[3][4].

Fonte: Satellite Industry Association

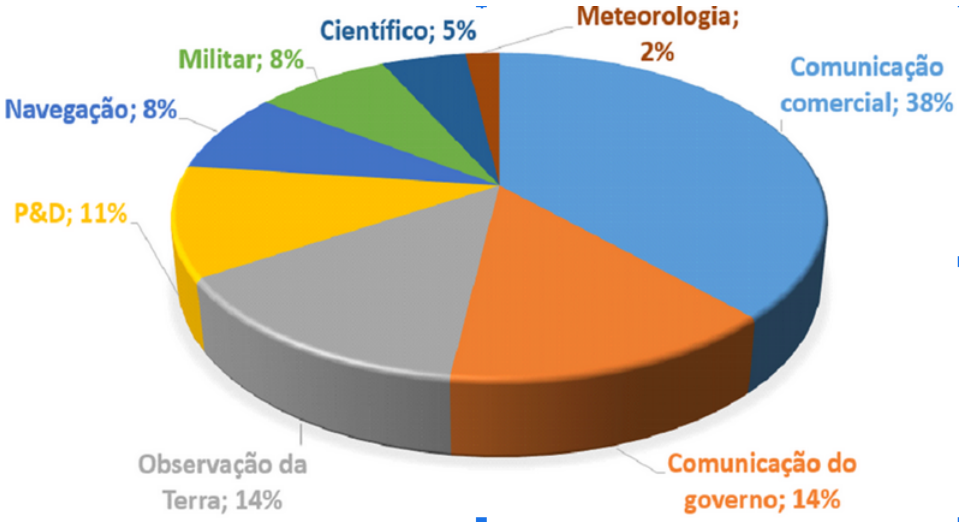

Figura 1.1: Gráfico Distribuição das Funções dos Satélite Comerciais

De acordo com relatório do grupo Tauri (*Salellite Industry Association*), o número de satélites em órbita aumentou aproximadamente 40% nos últimos cinco anos, sendo que a maior contribuição adveio de pequenos satélites em órbitas terrestres baixas (altitudes abaixo de 2000 quilômetros)[4]. No período de 2016 a 2022 esperava-se uma explosão no crescimento da indústria de satélites muito pequenos (com massa abaixo de 10kg).

1.3 CONTEXTUALIZAÇÃO DO PROBLEMA

Com o desenvolvimento de sistemas tecnológicos de ponta, os satélites artificiais têm desempenhado cada vez mais importantes funções, colaborando com o avanço científico em diversas áreas do conhecimento.

Com isso, a ideia principal deste projeto é a concepção de um computador de bordo capaz de gerenciar o completo funcionamento de um satélite, assim como trazer a tecnologia in-house.

No laboratório LODESTAR tem-se a iniciativa de construir uma Gaiola de Helmholtz que é um simulador capaz de promover uma interação eletromagnética controlável utilizado para fazer simulações de controle de posição e atitude. O algoritmo utilizado para realizar este controle é chamado de algoritmo B-dot.

O algoritmo B-dot desempenha um papel importante no sistema, pois ele utiliza a variação do campo magnético em um pequeno período de tempo para determinar o torque de controle a ser empregado no corpo. Por conveniência, a implementação deste algoritmo que atualmente

está sendo utilizada no laboratório exige operações em ponto flutuante que sobrecarregam o OBC atual. Uma das finalidades deste projeto é a migração para um microcontrolador com um módulo dedicado a essas operações[5][6][7].

2 FUNDAMENTAÇÃO TEÓRICA

2.1 CLASSIFICAÇÃO DE SATÉLITES

Atualmente, cerca de 3000 satélites artificiais estão em funcionamento em todo o planeta Terra, podendo ser classificados como[2]:

- **Exploração:** utilizados para realizar pesquisas sobre o Universo e o Sistema Solar;
- **Observação:** usados para criação de mapas e observações do meio ambiente terrestre;
- **Comunicação:** usados para os meios de comunicação e telecomunicações;
- **Navegação:** usados por diversas embarcações, substituindo a bússola;
- **Meteorologia:** usados para monitorar o tempo e o clima no planeta Terra;
- **Militar:** usados para estratégia militar, ou seja, para observar outros territórios, sendo também denominados de "satélites espiões".

As empresas produtoras de satélites utilizam uma classificação para o tipo de satélite baseado em sua massa total. Essa padronização ocorreu no intuito de promover uma escala de produção e tornar mais barato o desenvolvimento do produto no mercado. Assim sendo, os satélites variam de tamanho e custo, a depender do uso a que se destinam, indo de satélites extremamente pequenos até satélites do tamanho da ISS (International Space Station).

Para caracterizar os satélites, a NASA faz definições sobre a massa[8]:

- Satélites Grades: >1000 kg (ou >1t);
- Satélites Médios: Pesam entre 500 até 1000 kg (ou 0,5t até 1t);
- Satélites Pequenos: Pesam menos do que 500 kg (<0,5t).

Com relação aos satélites pequenos, existe uma subclassificação[8]:

- Minissatélites: Pesam entre 100 até 500 kg;
- Microssatélites: Pesam entre 10 até 100 kg;
- Nanosatélites: Pesam entre 1 até 10 kg;
- Picosatélites: Pesam entre 0,1 até 1 kg;
- Femtosatélites: Pesam entre 10 até 100 g;

- Attosatélites: Pesam entre 1 até 10 g;
- Zeptosatélites: Pesam entre 0,1 até 1 g.

Os satélites mais usados, atualmente, são os Cubesats. O peso deles depende do volume e da massa. Geralmente, eles pesam de 0,2 kg até 40 kg. O volume varia de 0,25 U até 27 U, no qual U é definido como um CubeSat com 1,3 kg e um cubo com 10 cm de aresta[8].

O comum é a não utilização de órbitas abaixo de 500 km, pois abaixo dessa altitude a atmosfera é suficientemente densa para causar danos à plataforma orbital, reduzindo sua vida operacional. Acima dela, aparece a dificuldade de manutenção do satélite e ocorre a saída do campo magnético da Terra, deixando o equipamento sujeito a altas doses de radiação. Portanto, divididas em faixas de altitude, as principais órbitas são[8]:

- Órbita baixa (LEO): de 500 km até 1000 km de altitude;
- Órbita Média (MEO): de 5000 km até 15000 km de altitude;
- Órbita Geossíncronas (GSO): acima de 20000 km de altitude (mais comum 36000 km). Essa órbita acompanha a rotação da Terra.

Outro ponto importante é a inclinação da órbita que será varrida pelo satélite. Por definição, a inclinação é o ângulo entre a linha do equador e o plano de órbita seguido pelo satélite. Além disso, e de acordo com os conceitos de Mecânica Orbital propostos por Chobotov, existem três tipos de cobertura orbital[8]:

- Cobertura Global - envolve a cobertura completa da Terra;
- Zona de Cobertura - faixa de observação entre dois valores de latitude;
- Cobertura Regional - cobertura para uma região específica.

Na classificação de nanosatélites, os mais utilizados atualmente são os CubeSats, que sera objeto de estudo deste trabalho.

2.2 CUBESATS

O primeiro desenho de CubeSats foi proposto em 1999 com o intuito de permitir que estudantes conseguissem projetar, construir, testar e operar um satélite com capacidades similares às do primeiro satélite. Os primeiros CubeSats foram lançados em Julho de 2003 por um foguete Rokot russo, e cerca de 75 CubeSats foram colocados em órbita até 2012. Em dezembro de 2019, o FloripaSat-1, CubeSat brasileiro oriundo de um projeto da Universidade Federal de Santa Catarina (UFSC), foi lançado pela China e entrou em operação[9].

A especificação dos CubeSats compreende vários objetivos de alto nível. A simplificação da infraestrutura de satélites torna possível projetar e produzir satélites funcionais a um baixo custo. O acoplamento de interfaces do lançador e da carga útil descarta a necessidade de uma enorme quantidade de trabalho que é necessária para acomodar os satélites aos lançadores no sistema convencional. Esse sistema também permite a substituição de cargas úteis de forma rápida, e a utilização de "janelas de lançamento"de forma otimizada.

O formato padrão de 10x10x10 cm do CubeSat é conhecido como "uma unidade"ou "CubeSat 1U". O CubeSat é um sistema escalável ao longo dos eixos através de incrementos de "1U", permitindo a criação de CubeSats "2U"(20x10x10 cm) e "3U"(30x10x10 cm) que já foram construídos e lançados. Mais recentemente, plataformas maiores de CubeSats vem sendo propostas, chegando até a "12U"(24x24x36 cm), estendendo a capacidade dos CubeSats além das aplicações acadêmicas e de validação de tecnologias, atingindo objetivos mais complexos nas áreas de ciência e defesa[10].

Fonte:(MABROUK, 2017, pág.1).

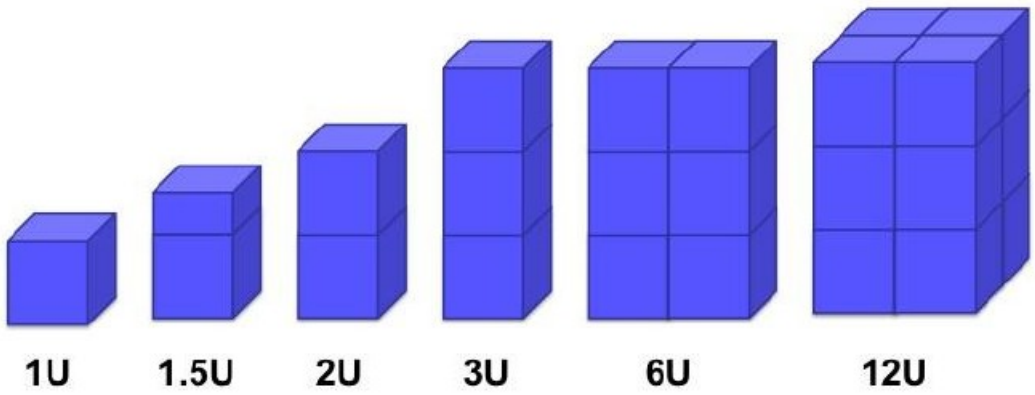

Figura 2.1: Escalas Padrão de CubeSats

Como os CubeSats medem 10x10 cm (independente do comprimento), todos eles podem ser colocados em órbita usando o mesmo sistema de liberação. Os CubeSats são em geral lançados e liberados usando um dispositivo mecânico chamado *Poly-PicoSatellite Orbital Deployer* (P-POD), também projetado e construído pela *Cal Poly*. Os P-PODs foram responsáveis pela liberação de mais de 90% de todos os CubeSats lançados até o momento (incluindo lançamentos malsucedidos), e 100% de todos os CubeSats lançados desde 2006.

Os CubeSats formam um meio independente e de custo competitivo para colocar cargas úteis em órbita. A maioria dos CubeSats carregam um ou dois instrumentos científicos como parte de sua missão primária.

Estrutura dos CubeSats

O CubeSat é construído a partir de quatro tipos específicos de liga de alumínio para garantir que eles tenham o mesmo coeficiente de expansão térmica que o veículo de lançamento. Os satélites também são revestidos com uma camada protetora de óxido ao longo de qualquer superfície que entre em contato com o veículo de lançamento, para impedir que eles sejam soldados a frio no local por estresse extremo[9].

Figura 2.2: Visão Geral das Camadas de CubeSat

Geralmente, CubeSats carregam um computador de bordo para realizar pesquisas, além de fornecer controle de atitude, propulsores e comunicações. Então outras unidades de processamento especializadas são incluídas para garantir que o computador de bordo não fique sobrecarregado por vários fluxos de dados, mas todas estas unidades de processamento devem ser capazes de interagir com ele. Dessa forma, o computador de bordo é responsável por delegar tarefas a outros computadores - como controle de atitude, cálculos para manobras orbitais e tarefas de agendamento. Ainda, assim, o computador principal pode ser usado para tarefas relacionadas à carga, como processamento de imagens, análise de dados e compactação de dados[9].

Controle de Atitude é o controle da orientação do veículo em relação a um sistema inercial de referência. Para sua realização, são necessários sensores para definir a atitude atual do veículo, atuadores para aplicar o torque necessário para reorientá-lo à atitude desejada e algoritmos para comandar tais atuadores[11].

Os componentes miniaturizados fornecem controle de atitude, geralmente composto de rodas de reação, magnetorquers, propulsores, rastreadores de estrelas, sensores do Sol e da Terra, sensores de taxa angular, receptores e antenas de GPS. Muitos desses sistemas são frequentemente usados em conjunto para compensar deficiências e fornecer níveis de redundância. Os sensores Sol são usados para fornecer apontamentos direcionais, enquanto a detecção da Terra e seu horizonte é essencial para a realização de estudos atmosféricos e da Terra. Os sensores Sol também são úteis para garantir que o CubeSat possa maximizar seu acesso à energia solar, que é o principal meio de alimentação de suas baterias - onde os painéis solares são incorporados ao revestimento externo dos satélites[11].

Enquanto isso, a propulsão pode ocorrer de várias formas, todas envolvendo propulsores miniaturizados, fornecendo pequenas quantidades de impulso específico. Os satélites também estão sujeitos ao aquecimento radioativo do Sol, da Terra e da luz solar refletida, sem mencionar o calor

gerado por seus componentes[11].

Os CubeSats contam com muitos métodos diferentes de propulsão, os mais comuns incluem gás frio, propulsão química, elétrica e velas solares. Um propulsor a gás frio depende de gás inerte (como nitrogênio) que é armazenado em um tanque e liberado através de um bico para gerar empuxo. Trata-se de um sistema muito seguro, já que a maioria dos gases frios não é volátil nem corrosivo. No entanto, eles têm desempenho limitado e não podem realizar manobras de alto impulso. Por isso, eles geralmente são usados em sistemas de controle de atitude, e não como propulsores principais.

Os sistemas de propulsão química dependem de reações químicas para produzir gás de alta pressão e alta temperatura, que é então direcionado através de um bico para criar empuxo. Eles podem ser líquidos, sólidos ou híbridos e geralmente se resumem à combinação de produtos químicos combinados com catalisadores ou oxidantes. Esses propulsores são simples (e, portanto, podem ser facilmente miniaturizados), possuem baixos requisitos de energia e são muito confiáveis.

A propulsão elétrica depende da energia elétrica para acelerar as partículas carregadas a altas velocidades. Este método é benéfico, pois combina alto impulso específico com alta eficiência, e os componentes podem ser facilmente miniaturizados. Uma desvantagem é que eles requerem energia adicional, o que significa células solares maiores, baterias maiores e sistemas de energia mais complexos.

As velas solares têm como benefício não requerer propulsor. Podem ser dimensionadas para as próprias dimensões do CubeSat, e a pequena massa do satélite resulta em maior aceleração para uma determinada área de vela solar. No entanto, as velas solares ainda precisam ser muito grandes em comparação com o satélite, o que torna a complexidade mecânica uma fonte adicional de falha potencial. No momento, poucos CubeSats empregaram uma vela solar, mas continua sendo uma área de potencial desenvolvimento, pois é o único método que não precisa de propulsor ou envolve materiais perigosos.

Como tal, os CubeSats também vêm com camadas de isolamento e aquecedores para garantir que seus componentes não excedam suas faixas de temperatura e que o excesso de calor possa ser dissipado. Os sensores de temperatura são frequentemente incluídos para monitorar aumentos ou quedas perigosas de temperatura. Para comunicações, os CubeSats podem contar com antenas que funcionam nas bandas VHF, UHF ou L-, S-, C- e X, que podem ser antenas helicoidais, dipolo ou monopolar, embora modelos mais sofisticados estejam sendo desenvolvidos[9].

2.3 PROJETO DE PLACA IMPRESSA

Pontes de terminais são barras de fenolite ou outro material isolante que contem pequenos encaixes metálicos onde componentes podem ser soldados e eletricamente conectados.

As Placas de Circuito Impresso (do inglês *Printed Circuit Board* - PCB) foram criadas para substituir estas pontes de terminais. Eles mecanicamente suportam e eletricamente conectam componentes eletrônicos usando trilhas, *pads* e outros, gravados em folhas de cobre laminado em um substrato não condutor[12].

CAMADAS DE METALIZAÇÃO

PCBs consistem em uma ou mais placas formadas por camadas de materiais plásticos e fibrosos (como fenolite, fibra de vidro ou fibra e filme de poliéster, entre outros polímeros) que conta com finas películas de substancias metálicas (cobre, prata, ouro ou níquel). Essas películas formam as "trilhas"ou "pistas"que são responsáveis pela condução da corrente elétrica para os componentes eletrônicos, viabilizando seu funcionamento e, consequentemente, do sistema formado na PCB.[12].

PROJETO DE ATERRAMENTO

O Projeto de Aterramento de uma Placa de Circuito Impresso é uma área ou camada de folha de cobre conectada ao ponto de terra do circuito. Ele serve como o caminho de retorno para a corrente de vários componentes distintos. Em PCBs com uma única camada, sua maior parte, toda a área não ocupada por traços do circuito é aterrada. Em PCBs multicamadas, normalmente, uma camada separada é utilizada como referência do circuito para facilitar o layout e aterramento dos componentes eletrônicos utilizados[12].

CROSSTALK

Crosstalk é o fenômeno em que um sinal transmitido em um circuito ou canal de sistema de transmissão cria um efeito indesejado em outro. É normalmente causado por capacitâncias, indutâncias ou conexão condutiva com um circuito ou canal. Este fenômeno é um problema significativo em cabeamento estruturado, circuitos integrados, comunicação sem fio e outros tipos de sistemas de comunicação[13].

PADRÃO PCB PARA CUBESATS

As PCBs feitas para CubeSats tem que seguir um tamanho, padrão na furação e nos conectores específicos, para se adequar às especificações do CubeSat. Um modelo base pronto se encontra disponível em um repositório no GitHub[14], o que contribuiu na adequação do design da PCB

proposta neste projeto.

2.4 MICROCONTROLADORES

Microcontrolador é um pequeno computador em um único circuito integrado o qual contém um núcleo de processador, memória e periféricos programáveis de entrada e saída. A memória de programação pode ser *NOR flash* ou *NAND flash* a qual, muitas vezes, é incluída no *chip*. O seu consumo de energia é relativamente baixo, normalmente na casa dos miliwatts, e possui habilidade para entrar em modo espera (ou baixo consumo). Quando entram nesse modo, o consumo de energia destes microcontroladores pode chegar na casa dos nanowatts, tornando-os ideaits para aplicações onde a exigência de baixo consumo de energia é um fator decisivo para o sucesso do projeto.

De forma diferente da programação para microprocessadores, que, em geral, contam com um sistema operacional e uma BIOS, o programador de microcontroladores cria todo programa que será executado pelo sistema ou pode utilizar um sistema operacional próprio para microcontroladores chamado de RTOS.

No OBC atual, o microcontrolador utilizado é uma MSP430 que possui um processador 16-bits de até 25 MHz, 128 KB de memória Flash e 8 KB de RAM, porém não possui uma Unidade de Ponto Flutuante (FPU)[15]. Neste projeto, o microcontrolador proposto é uma STM32 que possui um processador ARM-Cortex-M4 com FPU de 32-bits de até 80 MHz, com 256 KB de memória Flash, 64 KB de RAM, além de todos os componentes necessários para o projeto[16].

2.5 INTERFACES DE COMUNICAÇÃO

PC/104

O PC/104 é muito comum em CubeSats, porém ele surgiu como um padrão de barramento e formato para computadores e outros módulos. Ele existe desde 1987 e é considerado uma tecnologia obsoleta. É composto por 104 conectores macho-fêmea dispostos em quatro fileiras para empilhar multiplacas e manter uma comunicação entre elas via estes conectores, além de possuir um design que proporciona uma forte fixação entre as placas.

O PC/104 também é composto por dimensões dos conectores, da placa e dos furos de encaixe. Isto é conveniente para o CubeSat, pois especifica um tamanho próximo ao exigido, e permite que seus módulos sejam empilhados para aproveitar melhor o espaço de um cubo ou paralelepípedo. Além disso, ele garante uma boa conexão eletromecânica entre as placas[17].

SPI

Serial Peripheral Interface (SPI) é um protocolo que permite a comunicação do microcontrolador com diversos outros componentes. É uma especificação da interface de comunicação série síncrona usada para comunicação de curta distância, principalmente em sistemas embarcados. Os dispositivos SPI comunicam entre si em modo *full duplex* usando arquitetura "mestre-escravo"com um único mestre. Em modo "escravo", o microcontrolador comporta-se como um componente da rede, recebendo o sinal de *clock*. Em modo "mestre", o microcontrolador gera um sinal de *clock* e deve ter um pino de entrada/saída para habilitação de cada periférico[18].

I²C

O Circuito Inter-Integrado (I²C) é uma interface de comunicação serial multi-controlada *single-ended* inventada em 1982 pela Philips Semiconductors usada para conectar periféricos de baixa velocidade a um sistema embarcado. Esta interface utiliza duas linhas bidirecionais, uma para dados e a outra para *clock*, com resistores de *pull-up*[19].

UART

UART se refere a *Universal Asynchronous Receiver Transmitter* que é um padrão para comunicação de dados de forma serial. Dois fios são utilizados para transmitir dados, um em cada direção, em regime *full-duplex*. Para isso, cada dispositivo deve ter seu *clock* e velocidades de transmissão e recepção iguais[20].

2.6 FREERTOS

FreeRTOS é um *kernel* de sistema operacional de tempo real para microcontroladores e sistemas embarcados. Feito para ser simples e pequeno, o FreeRTOS foi escrito em C para facilitar seu porte e manutenção, mas também conta com algumas instruções em assembly necessárias na maioria das rotinas de escalonamento, que são dependentes da arquitetura do processador.

FreeRTOS fornece suporte para várias tarefas com prioridades, mutexes, semáforos, temporizadores de software e gerência de memória[21].

2.7 TRABALHOS ANTERIORES

Trabalhos relacionados com este projeto foram objetos de leitura sendo utilizados como referência.

Eduardo Lemos conceitualizou a ideia do planner. O objetivo era utilizar esse programa para

o sistema operacional do satélite conseguir registrar e agendar novas tarefas a serem executadas, bem como identificar e checar se houve algum erro durante a transmissão[22]. Esta ideia foi adotada neste trabalho com o mesmo proposito, detalhado na Metodologia.

Em seu trabalho, Ana Carolina Cabral analisou a performance e eficiência da energia de microcontroladores para aplicações de determinação e controle de atitude, assim como uma revisão dos algoritmos utilizados em missões espaciais[23]. Deste trabalho, foi registrado que a STM32 é uma opção viável, pelo seu baixo consumo, para o aprimoramento necessário para do projeto.

O trabalho do Artur Hugo Pereira remeteu à ideia para injeção de código como uma ferramenta para atualização e registro de tarefas em pleno voo feito para missões de satélites[24]. As ideias apresentadas nesse trabalho serviram apenas como conhecimento tendo em vista que serão colocadas em pratica em versões futuras deste projeto.

Vitor Lasserré apresentou seu método para atualização sem fio de sistemas de nanossatélites em pleno voo, modificando seu código fonte e alterando suas funcionalidades para o satélite poder realizar novas tarefas ou consertar possíveis erros tanto de hardware quanto de software[25]. As ideias apresentadas nesse trabalho serviram apenas como conhecimento tendo em vista que serão colocadas em pratica em versões futuras deste projeto.

3 PROJETO

3.1 REQUISITOS DO PROJETO

Para o desenvolvimento de qualquer projeto, a definição dos requisitos é essencial já que estes definirão as expectativas para o funcionamento das partes do projeto. Com base nas especificações do padrão CubeSat e nas especificações gerais da missão, os requisitos estabelecidos são definidos nas tabelas 3.1, 3.2, 3.3, 3.4 e 3.5:

Tabela 3.1: Requisitos de Hardware

1.	*Data Handling and Housekeeping*
2.	Possuir sistema de proteção contra travamentos
3.	Componentes devem suportar a faixa de temperatura das órbitas baixas
4.	O OBC deve possuir interfaces condizentes com cada subsistema do satélite
5.	Concepção Versátil
6.	Proteção contra falhas
7.	Armazenamento não volátil

Tabela 3.2: Requisitos de Software

1.	Controlar os subsistemas do CubeSat
2.	Realizar um log de eventos do sistema
3.	Ter um controle de referência temporal (precisão de 500 ms)
4.	Realizar o pacote de telemetria e enviar para o subsistema TT&C durante a transmissão
5.	Identificar e executar os comandos recebidos pela estação terrestre
6.	Alternar entre os modos de operação de acordo com a potência na bateria
7.	Deve possuir um sistema anti travamento

Tabela 3.3: Requisitos do Sistema para Mitigar Falhas

1.	Proteção Metálica (EMI Shielding)
2.	Proteção contra Corrente Excessiva (Overcurrent Protection)
3.	Código Corretor
4.	Watchdog Externo

Tabela 3.4: Requisitos de Interfaces Seriais

1.	Interface UART para comunicação com o rádio
2.	Interface UART para Debug
3.	Interface I²C para comunicação com o EPS
4.	Interface SPI para comunicação com o Magnetorquer

Tabela 3.5: Requisitos das Características do Microcontrolador

1.	Ponto Flutuante
2.	Baixo Consumo
3.	Conversores A/D
4.	GPIO
5.	Interfaces Seriais (UART, I²C, SPI)
6.	PWM
7.	Frequência do Clock Elevada
8.	Faixa de Temperatura (-80ºC até 53ºC)

3.2 ESCOLHAS FEITAS

Com base nos requisitos elencados na seção 3.1 acima e o objeto geral do projeto, optou-se pela escolha dos seguintes dispositivos:

- Barramento PC/104 devido a sua versatilidade, com vários barramentos de dados para escolher e a liberdade de alocar sinais personalizados em pinos de uso geral;
- Watchdog Externo TPL5010 motivado pela sua temperatura de operação, baixa dissipação de potência e tempo de operação ajustável;
- Sensor de Corrente ACS712 por causa da sua temperatura de operação, largura de banda, baixa dissipação de potência e níveis de tensão e corrente adequados ao padrão CubeSat;
- Memória Flash W25Q32JV pela sua capacidade de 32 Mbit, capacidade de leitura contínua de até 64 bytes, baixa dissipação de potência, Interface SPI para comunicação e sua temperatura de operação;
- Microcontrolador STM32L431RCT6 levando em consideração sua tensão de operação adequada ao projeto, CPU de 80 MHz, 256 kbytes de memória flash interna, 2 interfaces I²C, 5 interfaces UART, 3 interfaces SPI, 2 watchdogs independentes e, principalmente, um processador ARM-CORTEX-M4 com uma Unidade de Ponto Flutuante (FPU).

Na secção 3.3, sera demonstrada a conexão realizada entre estes dispositivos e o esquemático geral do projeto.

3.3 ESQUEMÁTICO DO PROJETO

A Figura 3.1 mostra como os pinos do microcontrolador foram selecionados e conectados a seus periféricos e ao barramento de comunicação, seguindo as especificações das interfaces suportadas e seu uso geral.

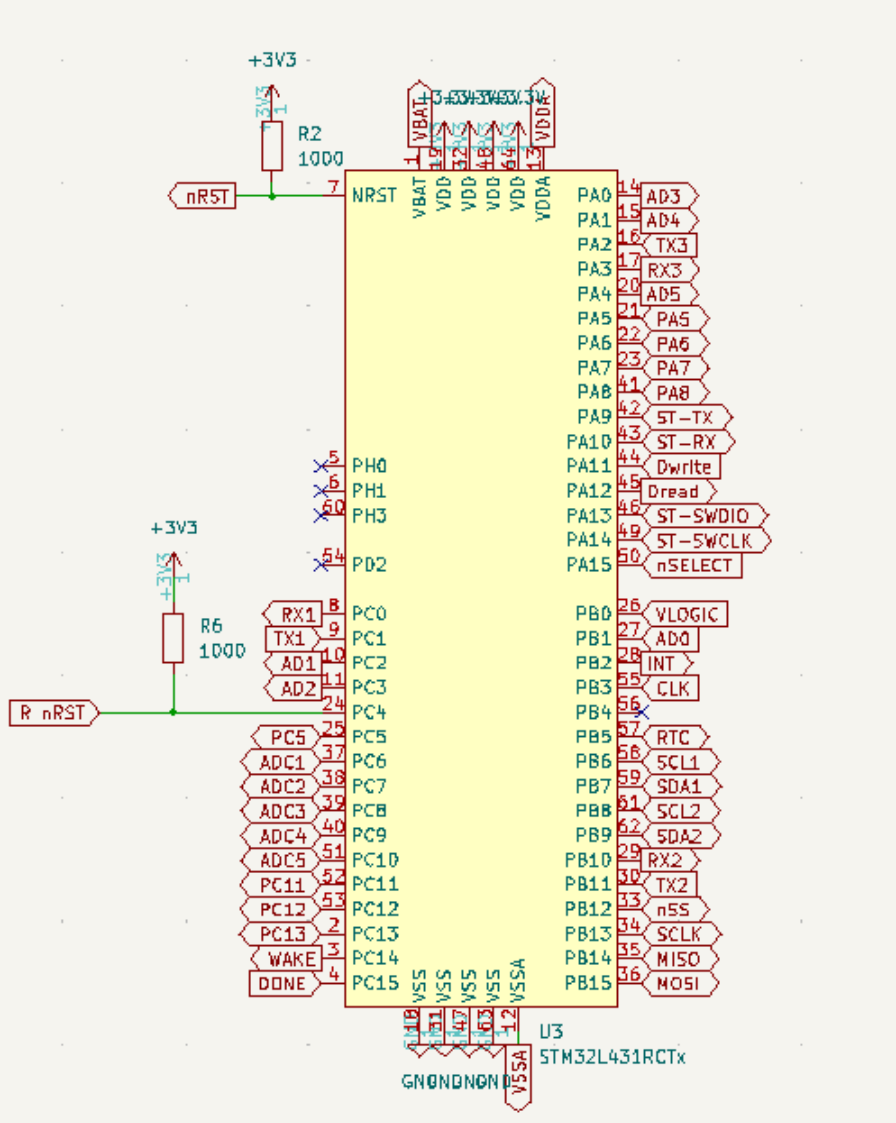

Figura 3.1: Esquemático Eletrônico Geral do OBC

A Figura 3.2 detalha o esquemático eletrônico do barramento PC/104 e como foi realizada as conexões entre seus pinos e o microcontrolador.

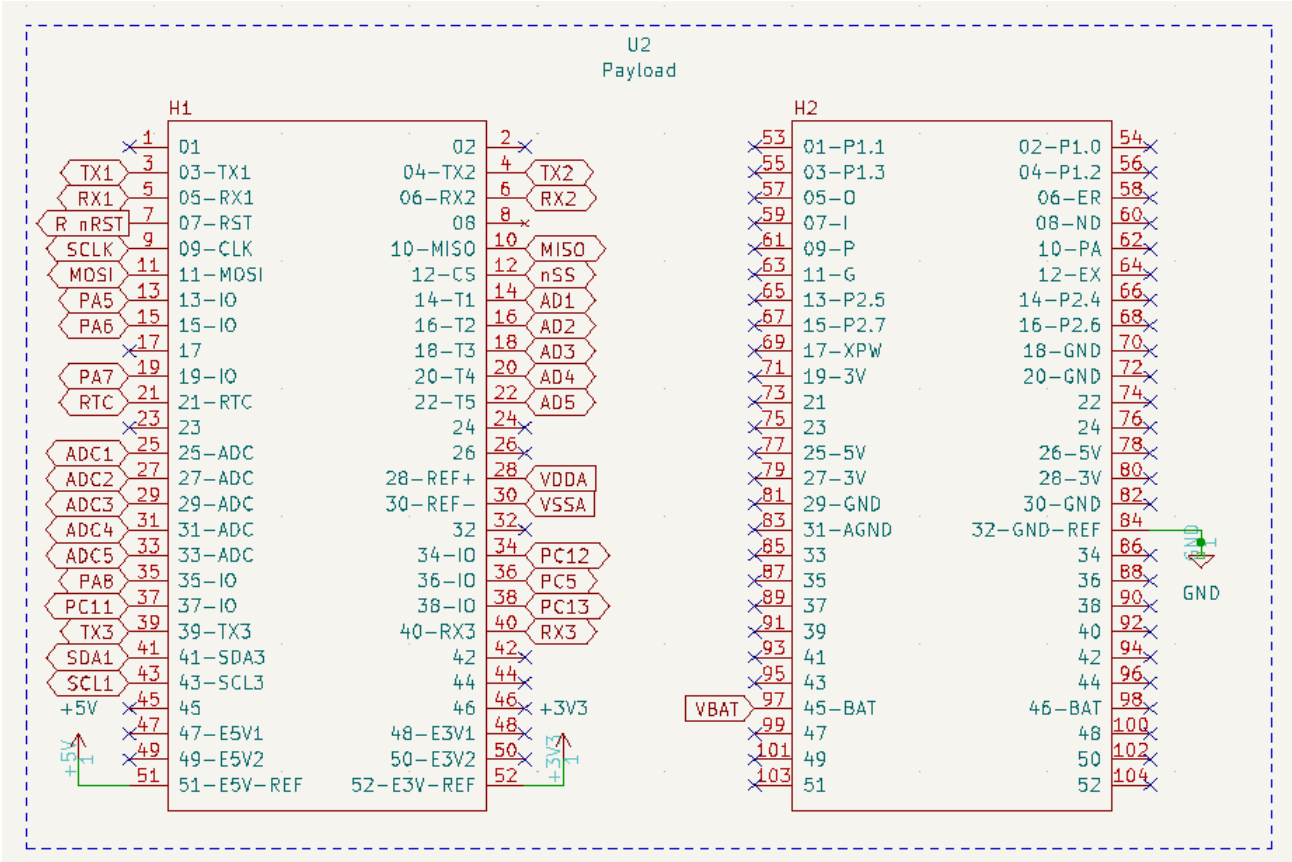

Figura 3.2: Esquemático Eletrônico Geral do PC/104

A Figura 3.3 contêm o esquemático eletrônico dos periféricos utilizados neste projeto e como foram conectados para servir seu propósito ao computador de bordo.

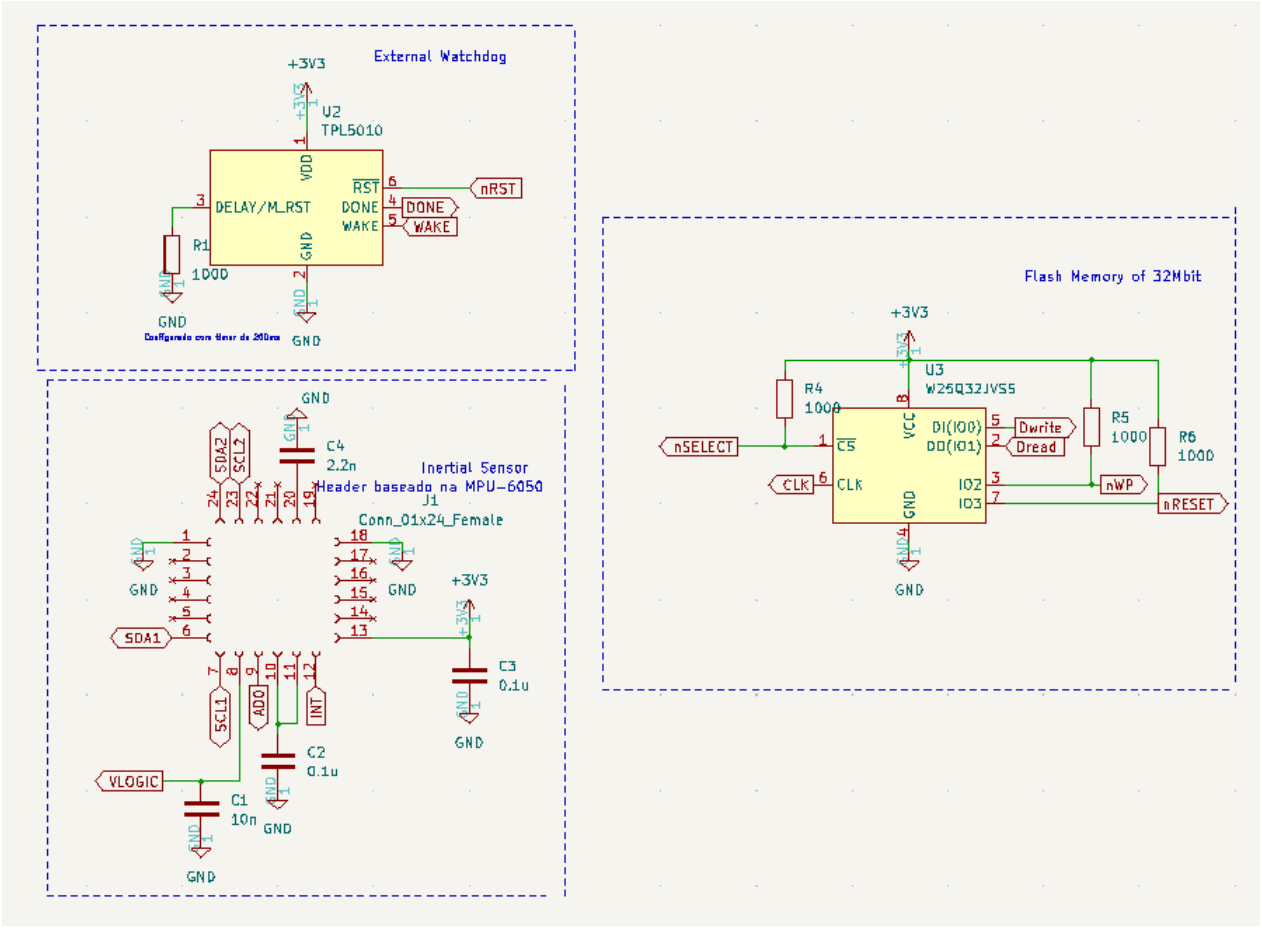

Figura 3.3: Esquemático Eletrônico das Componentes do OBC

4 METODOLOGIA

4.1 ESTRUTURA DO PLANNER

A ideia principal do *planner* é criar uma tarefa especial que executa em plano de fundo, responsável por registrar todas as futuras tarefas após o lançamento. A tarefa terá que lidar com um formato específico composto de um identificador, um byte de *checksum* e alguns bytes de configuração da tarefa que será registrada, como é demonstrado na Figura 4.1.

Fonte: *Orchestrating an Experimental Satellite*

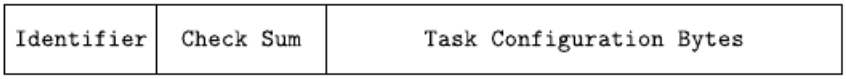

Figura 4.1: Formato Geral de um comando

4.2 DESCRIÇÃO DOS TESTES

Para testar o software desenvolvido, um *script* em python foi criado para mandar mensagens pela a interface UART do microcontrolador, via um cabo USB, tentando simular e avaliar o comportamento do satélite. Primeiramente, foi implementado um sistema de *echo*, cuja funcionalidade é mandar uma mensagem para o microcontrolador e esperar que o mesmo responda com a mesma mensagem, para testar se a interface do microcontrolador funciona e é capaz de receber e enviar dados.

Em seguida, foi testado se o OBC era capaz de reconhecer e responder uma mensagem de acordo com os identificadores de instrução definidos na Especificação de Comandos do Satélite Alfacrux, como demonstrado na Figura 4.2.

```
BUS_INJECTION       = 0x01,
MEMORY_READ_WRITE   = 0x02,
TELEMETRY           = 0x03,
CODE_INJECTION      = 0x04,
RUN_INJECTED        = 0x05,
FIRMWARE_UPDATE     = 0x06,
MAGNETORQUER        = 0x07,
PING_WATCHDOGS      = 0x08,
RADIO_OFF           = 0x09,
RADIO_ON            = 0x0A,
PING                = 0x0E,
ECHO                = 0x0F
```

Figura 4.2: Identificadores dos comandos do Satélite Alfacrux

Logo após, verificou-se a capacidade do OBC em ler e escrever dados através de seus barramentos I²C e UART, utilizando o identificador de instrução *BUS_INJECTION* e o formato de mensagem byte a byte <identificador>, <barramento>, <parâmetro>, <ação>, <tamanho_de_leitura_ou_escrita> <dados_de_escrita>.

4.2.1 Exemplos Práticos

Nesta seção serão descritos alguns exemplos de mensagens que podem ser enviadas e o comportamento esperado do sistema.

- Para a mensagem "0x01 0xFE 0x11 0x3F 0x11 0x00 0x03 0x61 0x62 0x63" é esperado que o comando seja validado e a mensagem "abc" seja escrita e lida pelo barramento I²C_1, comunicando-se com o "escravo"de endereço 0x3F;
- Para a mensagem "0x01 0xFE 0x32 0x01 0x11 0x00 0x05 0x74 0x65 0x73 0x74 0x65" é esperado que o comando seja validado e a mensagem "teste" seja escrita e lida pelo barramento UART_2 com baudrate de 19200;
- Para a mensagem "0xFF 0x00 0xAA 0xBC 0xCA" é esperado que seja retornada a mensagem "NOT FOUND", pois o identificador de comando da mensagem não está definido no projeto.
- Para a mensagem "0x01 0xFF 0xAB 0xBE 0xCE" é esperado que seja retornada a mensagem "CHECKSUM ERROR", pois a mensagem possui um identificador válido, mas o byte de verificação (checksum) da mensagem indica que houve algum erro durante a transmissão.
- Para a mensagem "0x02 0xFD 0xAC 0xCB 0xFF" é esperado que seja retornada a mensagem "MEMORY", indicando que o sistema foi capaz de validar o comando e identificar que seu tipo é MEMORY_READ_WRITE.
- Para a mensagem "0x03 0xFC 0xFA 0xEC 0xDD" é esperado que seja retornada a mensagem "TELEMETRY", indicando que o sistema foi capaz de validar o comando e identificar que seu tipo é TELEMETRY.
- Para a mensagem "0x0E 0xF1 0xAB 0xBB 0xEE" é esperado que seja retornada a mensagem "V", indicando que o sistema foi capaz de validar o comando, identificar o seu tipo e retornar uma mensagem simples para indicar que o sistema está funcionando.
- Para a mensagem "0x0F 0xF0 0xAA 0xBB 0xCC" é esperado que seja retornada a mensagem "0xAA 0xBB 0xCC", indicando que o sistema foi capaz de validar o comando, identificar o seu tipo e devolver o corpo da mensagem, criando um eco.

5 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos, tanto do desenvolvimento do hardware quanto do software.

As Figuras 5.1 5.2 mostram a renderização 3D da PCB e a lista de materiais utilizados para sua confecção. Seguindo o modelo para CubeSat, a PCB possui ranhuras para passar os cabos entre as camadas do CubeSat como é demonstrado na Figura 2.1.

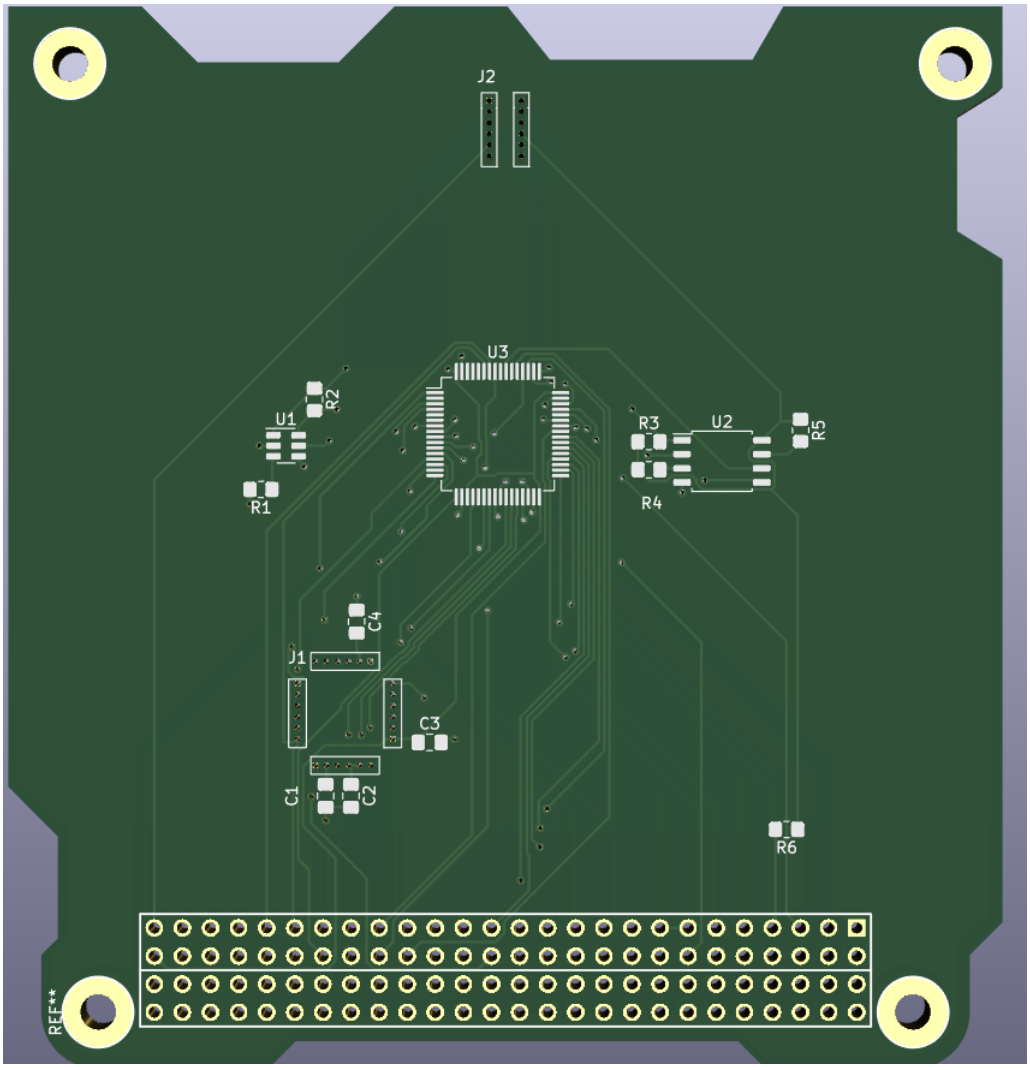

Figura 5.1: Renderização 3D da PCB

Id	Designator	Package	Quantity	Designation
1	REF**	PC104-Standard	1	PC104-Standard
2	C4	C_0805_2012Metric_Pad1.18x1.45mm_HandSolder	1	2.2n
3	R6,R1,R4,R	R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	6	1000
4	U1	SOT-23-6	1	TPL5010
5	J1	PinHeader_1x24_P2.54mm_Vertical	1	MPU-6050_Header
6	C3,C2	C_0805_2012Metric_Pad1.18x1.45mm_HandSolder	2	0.1u
7	C1	C_0805_2012Metric_Pad1.18x1.45mm_HandSolder	1	10n
8	U2	SOIC-8_5.23x5.23mm_P1.27mm	1	W25Q32JVSS
9	U3	LQFP-64_10x10mm_P0.5mm	1	STM32L431RCTx
10	J2	PinHeader_1x12_P1.00mm_Vertical	1	PinHeader_1x12_P1.00mm_Vertical

Figura 5.2: Lista de Materiais utilizado na PCB

Os testes de software foram feitos utilizando um *script* em Python para simular a transmissão e recepção de mensagens, o pseudo-código indicado na Figura 5.3 foi utilizado como modelo para o desenvolvimento deste *script*.

```
funcao inicio()
{
    string mensagem = "0x01 0xFE 0xAE 0xCB 0xFC"
    serial ser
    ser.abrir()        // Inicia a comunicacao via UART

    escreva("Mensagem Enviada ", mensagem, "\n")

    para cada byte em mensagem {
        ser.escreva(byte)
    }

    escreva("Mensagem Recebida ")
    enquanto ser.recebendoBytes() {
        escreva(ser.ler(), " ")
    }
    escreva("\n")
    ser.fechar()
}
```

Figura 5.3: Pseudo-código Modelo

A Figura 5.4 mostra os testes de *BUS_INJECTION* realizados no escravo de endereço 0x3F pelo barramento I2C_1, selecionados pela sequência de bytes “0x01 0xFE 0x11 0x3F” das mensagens enviadas. Primeiro é testado a ação *Write* com o byte “0x01”, enviando os dados “0x00 0x05 0x54 0x45 0x53 0x54 0x45”. Logo em seguida, é realizado o teste de *Read*, solicitando ao microcontrolador os dados enviados no teste anterior com os bytes “0x10 0x00 0x05 0x00”

e esperasse que a resposta seja a mensagem “T E S T E”. Por último, a ação *Write then Read* é feita, com a mensagem “0x00 0x0F 0x00 0x0F 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F” enviada e esperando a mensagem “a b c d e f g h i j k l m n o” como resposta. Pode-se verificar que as mensagens foram enviadas com sucesso e o microcontrolador foi capaz de responder de acordo.

```
Mensagem enviada: 0x01 0xFE 0x11 0x3F 0x01 0x00 0x05 0x54 0x45 0x53 0x54 0x45
BUS_INJECTION
I2C_1
Write
Mensagem enviada: 0x01 0xFE 0x11 0x3F 0x10 0x00 0x05 0x00
BUS_INJECTION
I2C_1
Read
T E S T E
Mensagem enviada: 0x01 0xFE 0x11 0x3F 0x11 0x00 0x0F 0x00 0x0F 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6A 0x6
B 0x6C 0x6D 0x6E 0x6F
BUS_INJECTION
I2C_1
Write then Read
a b c d e f g h i j k l m n o
```

Figura 5.4: Resultados dos Testes do Barramento I²C_1

A Figura 5.5 mostra os testes de *BUS_INJECTION* realizados no escravo de endereço 0x27 pelo barramento I2C_2, selecionados pela sequência de bytes “0x01 0xFE 0x12 0x27” das mensagens enviadas. Primeiro é testado a ação *Write* com o byte “0x01”, enviando os dados “0x00 0x05 0x74 0x65 0x73 0x74 0x65”. Logo em seguida, é realizado o teste de *Read*, solicitando ao microcontrolador os dados enviados no teste anterior com os bytes “0x10 0x00 0x05 0x00” e esperasse que a resposta seja a mensagem “t e s t e”. Por último, a ação *Write then Read* é feita, com a mensagem “0x11 0x00 0x0B 0x00 0x0B 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7A” enviada e esperando a mensagem “p q r s t u v w x y z” como resposta. Pode-se verificar que as mensagens foram enviadas com sucesso e o microcontrolador foi capaz de responder de acordo.

```
Mensagem enviada: 0x01 0xFE 0x12 0x27 0x01 0x00 0x05 0x74 0x65 0x73 0x74 0x65
BUS_INJECTION
I2C_2
Write
Mensagem enviada: 0x01 0xFE 0x12 0x27 0x10 0x00 0x05 0x00
BUS_INJECTION
I2C_2
Read
t e s t e
Mensagem enviada: 0x01 0xFE 0x12 0x27 0x11 0x00 0x0B 0x00 0x0B 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7
A
BUS_INJECTION
I2C_2
Write then Read
p q r s t u v w x y z
```

Figura 5.5: Resultados dos Testes do Barramento I²C_2

A Figura 5.6 mostra os testes de *BUS_INJECTION* realizados pelo barramento UART_0, selecionados pela sequência de bytes “0x01 0xFE 0x30” das mensagens enviadas. Primeiro é testado a ação *Write* com baudrate de 9600 selecionado pelos bytes “0x00 0x01”, enviando os

dados “0x00 0x07 0x65 0x63 0x68 0x6F 0x69 0x6E 0x67”. Logo em seguida, é realizado o teste de *Read* com baudrate de 19200, solicitando ao microcontrolador os dados enviados no teste anterior com os bytes “0x01 0x10 0x00 0x07 0x00” e esperasse que a resposta seja a mensagem “e c h o i n g”. Por último, a ação *Write then Read* é feita com baudrate de 38400, a mensagem “0x02 0x11 0x00 0x13 0x00 0x13 0x74 0x65 0x73 0x74 0x20 0x72 0x65 0x73 0x75 0x6C 0x74 0x20 0x63 0x6F 0x72 0x72 0x65 0x63 0x74” enviada e esperando a mensagem “t e s t r e s u l t c o r r e c t” como resposta. Pode-se verificar que as mensagens foram enviadas com sucesso e o microcontrolador foi capaz de responder de acordo.

```
Mensagem enviada: 0x01 0xFE 0x30 0x00 0x01 0x00 0x07 0x65 0x63 0x68 0x6F 0x69 0x6E 0x67
BUS_INJECTION
UART_0
Baudrate: 9600
Write
Mensagem enviada: 0x01 0xFE 0x30 0x01 0x10 0x00 0x07 0x00
BUS_INJECTION
UART_0
Baudrate: 19200
Read
e c h o i n g
Mensagem enviada: 0x01 0xFE 0x30 0x02 0x11 0x00 0x13 0x00 0x13 0x74 0x65 0x73 0x74 0x20 0x72 0x65 0x73 0x75 0x6C 0x7
4 0x20 0x63 0x6F 0x72 0x72 0x65 0x63 0x74
BUS_INJECTION
UART_0
Baudrate: 38400
Write then Read
t e s t   r e s u l t   c o r r e c t
```

Figura 5.6: Resultados dos Testes do Barramento UART_0

A Figura 5.7 mostra os testes de *BUS_INJECTION* realizados pelo barramento UART_1, selecionados pela sequência de bytes “0x01 0xFE 0x31” das mensagens enviadas. Primeiro é testado a ação *Write* com baudrate de 57600 selecionado pelos bytes “0x03 0x01”, enviando os dados “0x00 0x07 0x45 0x43 0x48 0x4F 0x49 0x4E 0x47”. Logo em seguida, é realizado o teste de *Read* com baudrate de 115200, solicitando ao microcontrolador os dados enviados no teste anterior com os bytes “0x04 0x10 0x00 0x07 0x00” e esperasse que a resposta seja a mensagem “E C H O I N G”. Por último, a ação *Write then Read* é feita com baudrate de 230400, a mensagem “0x05 0x11 0x00 0x13 0x00 0x13 0x54 0x45 0x53 0x54 0x20 0x52 0x45 0x53 0x55 0x4C 0x54 0x20 0x43 0x4F 0x52 0x52 0x45 0x43 0x54” enviada e esperando a mensagem “T E S T R E S U L T C O R R E C T” como resposta. Pode-se verificar que as mensagens foram enviadas com sucesso e o microcontrolador foi capaz de responder de acordo.

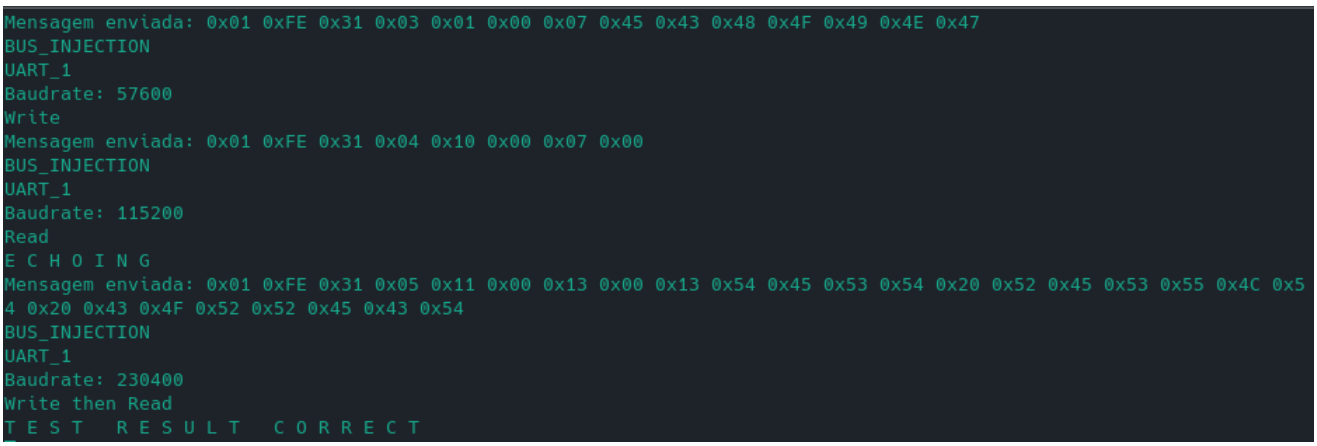

Figura 5.7: Resultados dos Testes do Barramento UART_1

A Figura 5.8 mostra os testes de *BUS_INJECTION* realizados pelo barramento UART_2, selecionados pela sequência de bytes “0x01 0xFE 0x32” das mensagens enviadas. Primeiro é testado a ação *Write* com baudrate de 460800 selecionado pelos bytes “0x06 0x01”, enviando os dados “0x00 0x07 0x65 0x63 0x6F 0x61 0x6E 0x64 0x6F”. Logo em seguida, é realizado o teste de *Read* com baudrate de 9600, solicitando ao microcontrolador os dados enviados no teste anterior com os bytes “0x00 0x10 0x00 0x07 0x00” e esperasse que a resposta seja a mensagem “e c o a n d o”. Por último, a ação *Write then Read* é feita com baudrate de 19200, a mensagem “0x01 0x11 0x00 0x12 0x00 0x12 0x72 0x65 0x73 0x75 0x6C 0x74 0x61 0x64 0x6F 0x20 0x64 0x6F 0x20 0x74 0x65 0x73 0x74 0x65” enviada e esperando a mensagem “r e s u l t a d o d o t e s t e” como resposta. Pode-se verificar que as mensagens foram enviadas com sucesso e o microcontrolador foi capaz de responder de acordo.

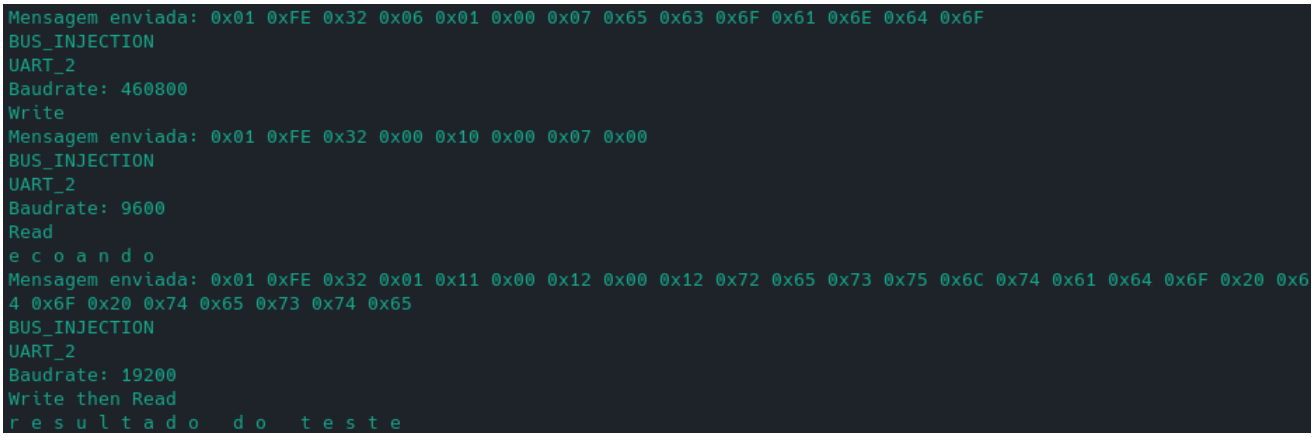

Figura 5.8: Resultados dos Testes do Barramento UART_2

A Figura 5.9 mostra os testes de *BUS_INJECTION* realizados pelo barramento UART_3, selecionados pela sequência de bytes “0x01 0xFE 0x33” das mensagens enviadas. Primeiro é testado a ação *Write* com baudrate de 38400 selecionado pelos bytes “0x02 0x01”, enviando os dados “0x00 0x07 0x45 0x43 0x4F 0x41 0x4E 0x44 0x4F”. Logo em seguida, é realizado o

teste de *Read* com baudrate de 57600, solicitando ao microcontrolador os dados enviados no teste anterior com os bytes “0x03 0x10 0x00 0x07 0x00” e esperasse que a resposta seja a mensagem “E C O A N D O”. Por último, a ação *Write then Read* é feita com baudrate de 115200, a mensagem “0x04 0x11 0x00 0x12 0x00 0x12 0x52 0x45 0x53 0x55 0x4C 0x54 0x41 0x44 0x4F 0x20 0x44 0x4F 0x20 0x54 0x45 0x53 0x54 0x45” enviada e esperando a mensagem “R E S U L T A D O D O T E S T E” como resposta. Pode-se verificar que as mensagens foram enviadas com sucesso e o microcontrolador foi capaz de responder de acordo.

```
Mensagem enviada: 0x01 0xFE 0x33 0x02 0x01 0x00 0x07 0x45 0x43 0x4F 0x41 0x4E 0x44 0x4F
BUS_INJECTION
UART_3
Baudrate: 38400
Write
Mensagem enviada: 0x01 0xFE 0x33 0x03 0x10 0x00 0x07 0x00
BUS_INJECTION
UART_3
Baudrate: 57600
Read
E C O A N D O
Mensagem enviada: 0x01 0xFE 0x33 0x04 0x11 0x00 0x12 0x00 0x12 0x52 0x45 0x53 0x55 0x4C 0x54 0x41 0x44 0x4F 0x20 0x4
4 0x4F 0x20 0x54 0x45 0x53 0x54 0x45
BUS_INJECTION
UART_3
Baudrate: 115200
Write then Read
R E S U L T A D O   D O   T E S T E
```

Figura 5.9: Resultados dos Testes do Barramento UART_3

A Figura 5.10 mostra o teste de *COMMAND_NOT_FOUND* realizado ao tentar selecionar um comando não definido no sistema. Conforme a Figura 4.2, não há um comando com identificador “0xFF”, logo a mensagem “0xFF 0x00 0xAA 0xBC 0xCA” foi enviada, esperando que o sistema respondesse com “*NOT FOUND*”.

```
Mensagem enviada 0xff 0x0 0xaa 0xbc 0xca
Mensagem recebida NOT FOUND
```

Figura 5.10: Resultado do Teste para Comando Não Encontrado

A Figura 5.11 mostra o teste de *CHECKSUM_ERROR* realizado ao tentar enviar um byte de verificação não correspondente ao comando selecionado. Com a mensagem “0x01 0xFF 0xAB 0xBE 0xCE”, espera-se que o sistema responda com “CHECKSUM ERROR”, pois o identificador "0x01" requer "0xFE" como byte verificador e não "0xFF" conforme foi enviado.

```
Mensagem enviada 0x1 0xff 0xab 0xbe 0xce
Mensagem recebida CHECKSUM ERROR
```

Figura 5.11: Resultado do Teste de Erro no Checksum

A Figura 5.12 mostra o teste de *MEMORY_READ_WRITE* que, por enquanto, verifica so-

mente se o sistema é capaz de identificar o comando e testar se o byte de verificação está correto. Enviando a mensagem “0x02 0xFD 0xAC 0xCB 0xFF”, esperasse que o sistema responda com “*MEMORY*”.

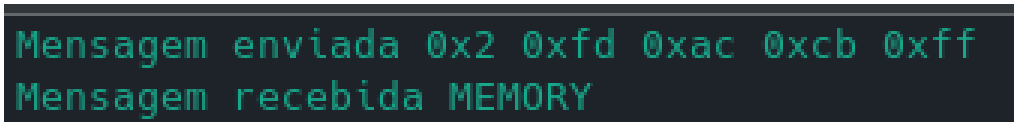

Figura 5.12: Resultado do Teste de Escrita e Leitura de Memória

A Figura 5.13 mostra o teste de *TELEMETRY* que, por enquanto, verifica somente se o sistema é capaz de identificar o comando e testar se o byte de verificação está correto. Enviando a mensagem “0x03 0xFC 0xFA 0xEC 0xDD”, esperasse que o sistema responda com “*TELEMETRY*”.

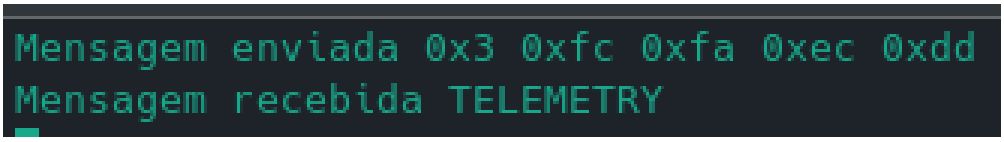

Figura 5.13: Resultado do Teste de Telemetria

A Figura 5.14 mostra o teste de *PING* que serve para verificar se o sistema está ativo e é capaz de receber comandos. Enviando a mensagem “0x0E 0xF1 0xAB 0xBB 0xEE”, esperasse que o sistema responda com “V”.

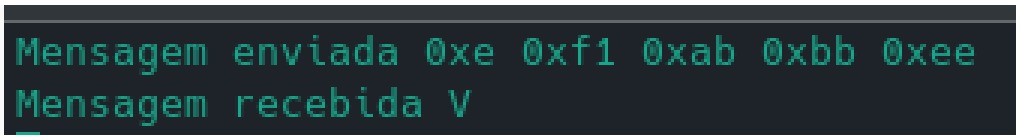

Figura 5.14: Resultado do Teste de Ping

A Figura 5.15 mostra o teste de *ECHO* que serve para verificar se o sistema é capaz de receber e retransmitir uma mensagem. Enviando a mensagem “0x0F 0xF0 0xAA 0xBB 0xCC”, esperasse que o sistema responda com “0xAA 0xBB 0xCC”.

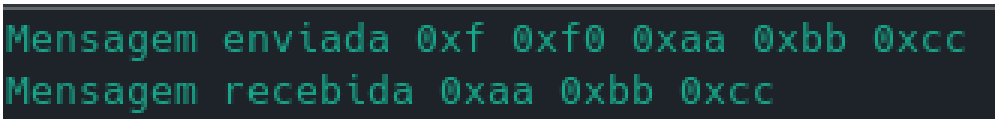

Figura 5.15: Resultado do Teste de Echo

Como pode-se observar, os testes ocorreram conforme o esperado, ou seja, o OBC foi capaz de receber, validar e executar ações conforme as mensagens recebidas.

Porém, devido a problemas na implementação das instâncias do Serial da API STM32duino

utilizada neste projeto, ocorreram alguns testes sem resposta do sistema, indicando uma falha na comunicação.

6 CONCLUSÃO

Este trabalho apresentou o projeto do Computador de Bordo para uma futura missão CubeSat da Universidade de Brasília. Inicialmente, foi realizada uma conceituação básica do tema seguido da categorização dos pequenos satélites e seus principais subsistemas. Logo em seguida, foi apresentado os requisitos do projeto e as escolhas feitas para a seleção do microcontrolador e os componentes que foram utilizados no projeto, como o *watchdog* externo, memória *flash*, sensor de corrente, etc.

Para o desenvolvimento de software foi utilizado a IDE Arduino Sketch, as APIs FreeRTOS como kernel base do sistema operacional do OBC e STM32duino para a compatibilidade do Arduino Sketch com os microcontroladores STM32. Continuou-se com a metodologia que apresentou a organização do *planner* e alguns exemplos de teste de como cada identificador das futuras tarefas que serão registradas devem se comportar.

Por fim, o capítulo de resultados consistiu em apresentar o projeto da PCB com uma lista de componentes utilizados em sua confecção, e os resultados dos testes dos comandos que o sistema deve reconhecer.

Os objetivos gerais propostos nas Tabelas 3.1, 3.2, 3.3, 3.4 e 3.5 foram alcançados. A ideia, contudo, é dar continuidade ao projeto, fazendo testes na PCB impressa, corrigindo o problema do Serial citado anteriormente, e melhorando o sistema de forma geral.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] W. website, “Historia dos Satélites Artificiais, https://pt.wikipedia.org/wiki/Satélite_artificial (acessado em 18/11/2022).”

[2] U. N. website, “Office for Outer Space Affairs, Online Index of Objects Launched into Outer Space, https://www.unoosa.org/oosa/osoindex/search-ng.jspx (acessado em 18/11/2022).”

[3] A. E. Brasileira, “Benefícios da Exploração Espacial, https://www.gov.br/aeb/pt-br/programa-espacial-brasileiro/aplicacoes-espaciais/beneficios-da-exploracao-espacial (acessado em 18/11/2022).”

[4] S. I. Association, “Satellite Industry Report, https://sia.org/news/resources/state-of-the-satellite-industry-report (acessado em 18/11/2022).”

[5] K. Borschiov, “Generic on-board-computer hardware and software development for nano-satellite applications,” vol. 1, 2012.

[6] L. T. Lumbwe, “Development of an onboard computer (obc) for a cubesat,” vol. 1, 2013.

[7] E. Razzaghi, “Design and qualification of on-board computer for aalto-1 cubesat,” vol. 1, 2012.

[8] A. L. X. Jr., “Em direçao a uma taxonomia unificada para satelites com base em massa e tamanho,” vol. 1, 2019.

[9] C. P. S. U. website, “Cal Poly Cubesat Laboratory (CPCL), https://cubesat.org (acessado em 18/11/2022).”

[10] A. Johnstone, “Cubesat design specification (1u -12u) rev 14,” vol. 1, 2020.

[11] R. A. Medeiros, “Implementaçap de um controle de atitude em um satélite utilizando rodas de reaçao em 3 eixos,” vol. 1, 2017.

[12] C. Hamilton, “A guide to printed circuit board design,” vol. 1, 1984.

[13] S. A. Pignari, “Characterisation of crosstalk in terms of mean value and standard deviation,” vol. 1, 2003.

[14] A. Becerra, E. Obreque, and M. Vidal, “Payload PC104, https://github.com/spel-uchile/Payload-PC104 (acessado em 18/11/2022).”

[15] T. Instruments, “Msp430f552x datasheet,” vol. 1, 2009.

[16] S. Microeletronics, “Stm32l431xx datasheet,” vol. 1, 2016.

[17] P. E. Consortium, “Pc/104 specification,” vol. 1, 2008.

[18] P. Dawound, “6 serial peripheral interface (spi),” vol. 1, 2020.

[19] K. Hemmanur, “Inter-integrated circuit (i2c),” vol. 1, 2009.

[20] R. Ranjan, “Design and implementation of high-speed universal asynchronous receiver and transmitter,” vol. 1, 2020.

[21] R. Barry, “Mastering the freertos real time kernel,” vol. 1, 2016.

[22] E. L. Rocha, “Orchestrating an experimental satellite,” vol. 1, 2022.

[23] A. C. C. P. de Melo, “A benchmark for assessing nanosatellite on-board computer power-efficiency and performance,” vol. 1, 2019.

[24] A. H. C. Pereira, “Code injection as a tool for satellite missions,” vol. 1, 2018.

[25] V. L. N. Coelho, “Over-the-air update in nanosatellite missions,” vol. 1, 2022.