UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Émerson da Trindade Lemos

ANÁLISE E REFORMULAÇÃO DO ARCABOUÇO DE INTERCONEXÃO DO FLORIPASAT VISANDO MITIGAÇÃO DE FALHAS NAS CAMADAS FÍSICA E DE ENLACE DE DADOS.

Florianópolis
2021

Émerson da Trindade Lemos

ANÁLISE E REFORMULAÇÃO DO ARCABOUÇO DE INTERCONEXÃO DO FLORIPASAT VISANDO MITIGAÇÃO DE FALHAS NAS CAMADAS FÍSICA E DE ENLACE DE DADOS.

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica para a obtenção do Grau de Mestre em Engenharia Elétrica. Orientador: Prof. Ph.D. Eduardo Augusto Bezerra.

Florianópolis

2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Lemos, Émerson
ANÁLISE E REFORMULAÇÃO DO ARCABOUÇO DE INTERCONEXÃO DO FLORIPASAT VISANDO MITIGAÇÃO DE FALHAS NAS CAMADAS FÍSICA E DE ENLACE DE DADOS. / Émerson Lemos ; orientador, Eduardo Bezerra, 2021.
118 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, 2021.

Inclui referências.

1. Engenharia Elétrica. 2. Nanossatélite. 3. Confiabilidade. 4. Comunicação embarcada. 5. Protocolo CAN. I. Bezerra, Eduardo. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

Émerson da Trindade Lemos

ANÁLISE E REFORMULAÇÃO DO ARCABOUÇO DE INTERCONEXÃO DO FLORIPASAT VISANDO MITIGAÇÃO DE FALHAS NAS CAMADAS FÍSICA E DE ENLACE DE DADOS.

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

_____

Prof. Fabian Luis Vargas, Dr.

Pontifícia Universidade Católica do Rio Grande do Sul.

_____

Prof. Roberto de Matos, Dr.

Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de "Mestre em Engenharia Elétrica" pelo Programa de Pós-Graduação em Engenharia Elétrica.

Documento assinado digitalmente
Telles Brunelli Lazzarin
Data: 20/10/2021 19:23:25-0300
CPF: 023.400.719-25
Verifique as assinaturas em https://v.ufsc.br

_____

Prof. Telles Brunelli Lazzarin, Dr.

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Universidade Federal de Santa Catarina.

Documento assinado digitalmente
Eduardo Augusto Bezerra
Data: 11/10/2021 03:43:35-0300
CPF: 830.851.577-00
Verifique as assinaturas em https://v.ufsc.br

_____

Prof. Eduardo Augusto Bezerra, PhD.

Orientador.

Universidade Federal de Santa Catarina.

Florianópolis, 2021

Este trabalho é dedicado aos meus pais, Itamar e Fátima, por serem as pessoas mais importantes em minha vida e fundamentais para a conclusão da minha pós-graduação.

AGRADECIMENTOS

A minha namorada Franciele Menezes de Oliveira, por todo o suporte oferecido a mim durante estes anos.

Ao meu orientador professor Doutor Eduardo Augusto Bezerra, por todos os ensinamentos passados a mim ao decorrer da pós-graduação e toda a orientação durante o desenvolvimento desta dissertação.

Ao meu irmão Anderson da Trindade Lemos, por ser uma referência em minha vida.

A todos os professores do curso por todos os ensinamentos passados a mim.

A todos que de alguma forma estiveram comigo e ajudaram-me a alcançar este objetivo.

“O caminho mais certeiro para o sucesso é sempre tentar apenas uma vez mais”.

Thomas A. Edison.

RESUMO

Inicialmente, nanossatélites, satélites com massa entre 1 e 10 kg, eram utilizados principalmente em atividades de ensino na área espacial e para validação de tecnologias em órbita. Nos últimos anos, com o crescimento da área, esse tipo de satélite passou a ser utilizado também para aplicações científicas e comerciais. Como consequência, além do custo reduzido, estes sistemas também devem apresentar uma alta taxa de confiabilidade durante a missão espacial. Desta forma, a confiabilidade na transferência das informações de um satélite é imprescindível e o protocolo de comunicação embarcado de um nanossatélite deve ser tão confiável quanto os demais sistemas. Nesta dissertação é discutido o estudo realizado em protocolos de comunicação utilizados nos sistemas embarcados de nanossatélites. Durante o estudo foi avaliada a melhor maneira de realizar a transferência das mensagens entre os subsistemas da plataforma FloripaSat, visando a utilização para sistemas embarcados baseados em FPGA e baseados em microcontroladores. Para isto, foram utilizadas como ponto de partida as informações obtidas a partir da missão FloripaSat-1. Para validação dos resultados, foram realizadas análises matemáticas e análise de um modelo simulado do barramento, que foram comparadas com as informações de um modelo de barramento prático, baseado nos três módulos principais do FloripaSat, o OBDH, o EPS e o TT&C. Como resultado final do estudo, foi definido e implementado o protocolo CAN para utilização na plataforma FloripaSat, visando melhoria nos aspectos de confiabilidade na transferência de dados entre os subsistemas do satélite.

Palavras-Chave: nanossatélites; confiabilidade; comunicação embarcada; FPGA; microcontroladores; FloripaSat; protocolo CAN;

ABSTRACT

Initially, nanosatellites, satellites with mass between 1 and 10 kg, were used mainly in educational activities in the space area and for validation of technologies in orbit. In recent years, with the growth of the area, this type of satellite has also been used for scientific and commercial applications. As a result, in addition to the reduced cost, these systems must also have a high reliability during a space mission. Thus, reliability in the information transference from a satellite is essential and the on-board communication protocol of a nanosatellite must be as reliable as other systems. This dissertation discusses the study carried out on communication protocols used in embedded nanosatellite systems. During the study, the best way to carry out the transfer of messages between FloripaSat subsystems was evaluated, aiming at their use for embedded systems based on FPGA and based on microcontrollers. For this, the information obtained from the FloripaSat-1 mission was used as a starting point. To validate the results, mathematical analyzes and analysis of a simulated bus were performed, which were compared with information from a practical bus model, based on the three main FloripaSat modules, OBDH, EPS and TT&C. As a final result of the study, the CAN protocol was defined and implemented for use on the FloripaSat platform, aiming at improving the reliability aspects of data transfer between the satellite subsystems.

Keywords: nanosatellites; reliability; on-board communication; FPGA; microcontrollers; FloripaSat; CAN protocol;

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ACK - Acknowledgement

AMBA – Advanced microcontroller bus architecture

APP - Advanced Payload Processor

Cal Poly – California Polytechnic State University

CAN – Controller Area Network

CAN-SU – Controller Area Network – Spacecraft Usage

CAPL - Communication Access Programming Language

COTS – Commercial off-the-shelf

CRC – Cyclic Redundancy Check

CSMA/CD – Carrier Sense Multiple Access with Collision Detection

DLC – Data Length Code

ECC – Error Correction Code

ECU – Electronic Control Unit

EOF – End Of Frame

EPS – Electric Power System

ESA – European Space Agency

FPGA – Field Programmable Gate-array

FRAM – Ferroelectric Random Access Memory

GEO – Geostationary Earth Orbit

GNSS - Global Navigation Satellite System

GPS – Global Positioning System

GSE – Grupo de Sistemas Espaciais

I2C – Inter-Integrated Circuit

IDE – Identifier Extension

IFS – Inter Frame Space

INPE – Instituto Nacional de Pesquisas Espaciais

Km – Quilômetro

LEO – Low Earth Orbit

Li-Íon – Íons de Lítio

LLC – Logical Link Control

LON – Local Operating Network

MAC – Medium Access Control

Mbps – Megabits por Segundo.

MCU – Microcontroller Unit

MEO – Medium Earth Orbit

NVP – n-Version programming

NZR – Non-Return to Zero

OBC – On-Board Computer

OBDH – On-Board Data Handling

RadHard – Radiation Hardened

REC – Reception Error Counter

RTR – Remote Transmission Request

SEE – Single Event Effects

SEL – Single Event Latchup

SEU – Single Event Upset

SOF – Start Of Frame

SPI – Serial Peripheral Interface

SQA – Software Quality Assurance

SRAM – Static Random Access Memory

SRR – Substitute Remote Request

SSTL – Survey Satellite Tecnology Ltd

TEC – Transmission Error Counter

TID – Total Ionizing Dose

TMR – Triple Modular Redundancy

TT&C – Telemetry, Tracking and Command

UART - Universal Asynchronous Receiver Transmitter

USART – Universal Synchronous Asynchronous Receiver Transmitter

UFRGS - Universidade Federal do Rio Grande do Sul

UFSC – Universidade Federal de Santa Catarina

UFSM – Universidade Federal de Santa Maria

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas o uso de satélites se tornou imprescindível no nosso cotidiano, muitas vezes sendo utilizada de forma imperceptível, desde pequenos dispositivos pessoais, como GPS (*Global Positioning System* ou Sistemas de Posicionamento Global, em português), smartphones, sinais de televisão e Internet até o uso industrial e governamental, como satélites para estudos meteorológicos, sensoriamento dos recursos terrestres e para uso militar (MAINI; AGRAWAL, 2011).

Há, também, um crescimento no uso de satélites para comunicação móvel, observação da Terra, monitoramento do tráfego e do meio ambiente, principalmente por meio de satélites que atuam na órbita terrestre baixa (*Low Earth Orbit - LEO*) (JANSCHEK; BRAUNE, 2000).

Com o desenvolvimento de pequenos satélites (satélites com massa inferior a 500 Kg), atualmente, é possível que muitas empresas de pequeno e médio porte, universidades e instituições de ensino em geral possam atuar na área de desenvolvimento de sistemas espaciais. Visto que o principal empecilho para o desenvolvimento de grandes satélites é seu alto custo, foram criados incentivos para o desenvolvimento de satélites menores, como o padrão CubeSat, criado pela California Polytechnic State University (Cal Poly) em 1999 com o intuito de baratear o custo de desenvolvimento e lançamento de satélites e incentivar o avanço da indústria tornando-a mais acessível (NASA, 2017) (The CubeSat Program, 2020).

CubeSats são satélites que se enquadram na categoria nanossatélites e são desenvolvidos em universidades e instituições de ensino por possuírem algumas características que fazem com que sejam mais interessantes do que grandes satélites. Entre as principais vantagens no desenvolvimento de pequenos satélites estão menor tempo de desenvolvimento, padronização de tamanhos e, principalmente, o uso de componentes comerciais de prateleira (*Commercial of the shelf – COTS)* (NASA, 2017), o que torna mais fácil a transferência de informações e de tecnologias entre os grupos de desenvolvedores e resultam em um custo reduzido de lançamento e de projeto.

Tendo em vista a redução do custo de desenvolvimento de satélites cada vez mais são utilizados componentes COTS, que além do custo também reduzem o número de componentes utilizados e, consequentemente, a massa do satélite. (JANSCHEK; BRAUNE, 2000) (BLANCO; D'ERRICO; CONTICCHIO, 2006). Porém componentes COTS, normalmente, não possuem uma confiabilidade elevada e nem tolerância a falhas no nível necessário para a sua utilização em ambiente espacial, por este motivo, em alguns casos, é necessário desenvolver sistemas redundantes (KAHE, 2018).

Um exemplo de COTS utilizado na indústria aeroespacial são os FPGAs (F*ield Programmable Gate Array),* dispositivos lógicos programáveis que podem ser configurados conforme a necessidade do projeto, tais dispositivos costumam ser mais rápidos que microcontroladores, uma vez que utilizam a estrutura de hardware ao invés de serem baseados em software, como é o caso dos microcontroladores (JAYARATHNE; JAYANANDA, 2013).

Os FPGAs apresentam algumas características que os tornam uma opção interessante, por exemplo, a possibilidade de reuso do hardware para mais de uma função, modificação parcial ou total do projeto e a adaptação do hardware conforme novos requisitos da missão (BRAVHAR et al., 2018), devido a todas essas características os FPGAs são o candidato ideal para aumento da confiabilidade de uma missão. No trabalho de Martins et. al. (2018) foi demonstrada uma técnica de reconfiguração parcial dinâmica (DPR, do inglês *Dynamic Partial Reconfiguration)* para mitigar falhas permanentes, pois, uma parte do sistema que apresenta alguma falha permanente pode ser reconfigurada em outra região do FPGA, evitando a falha completa do sistema.

Em relação a infraestrutura de comunicação nos sistemas embarcados em CubeSats, diversos tipos de interface são utilizados. As mais comuns são I2C, SPI, UART e CAN. Entre essas opções, a interface CAN (Controller Area Network) apresenta bons indicadores quando levados em conta questões como confiabilidade, praticidade, facilidade e o custo de implementação do sistema, sendo cada vez mais utilizado por grupos desenvolvedores de nanossatélites, como pode ser visto no trabalho de Bouwmeester, Langer e Gill (2016). A empresa Cobham Gaisler desenvolveu um IP core de CAN de código aberto para utilização em FPGAs (COBHAM, 2020) contudo, uma análise deve ser feita para verificar a viabilidade de seu uso em comparação a um microcontrolador com controladores CAN já implementado em seus periféricos.

Assim, a confiabilidade apresentada por cada uma dessas opções é um fator importante que deve ser analisado, principalmente, tendo em vista a utilização em ambiente espacial, visto que, a transferência de dados precisa ser realizada da forma mais confiável possível.

No trabalho de George e Wilson (2018) são discutidos conceitos de computadores de bordo para CubeSats, com relação a tolerância a falhas e computação híbrida. A computação híbrida utiliza componentes mistos, como FPGAs e microcontroladores ou componentes tolerantes à radiação (RadHard, do inglês, *Radiation Hardened*) e dispositivos comerciais de alto nível.

Desta forma, nesta dissertação será avaliada a atual arquitetura utilizada na missão FloripaSat-1 e serão levantadas características de desempenho dos barramentos utilizados,

atualmente, na plataforma FloripaSat com base em consumo de energia, velocidade de processamento, velocidade de comunicação, entre outros. Também, serão levantadas as características dos processadores utilizados e será feita uma análise do melhor modelo para o sistema de comunicação embarcada, visando a utilização em microprocessadores e FPGAs para implementação de uma interface de comunicação mais confiável.

Para a validação do modelo proposto será realizada uma análise da confiabilidade do barramento escolhido por meio de uma análise de tempo real do envio das mensagens da missão FloripaSat-1, bem como, a análise da probabilidade de ocorrência de erros e a interferência destes erros no tempo de resposta da transmissão das mensagens. As análises realizadas serão comparadas com um modelo reformulado do barramento de intercomunicação do FloripaSat-1 e com um modelo prático desenvolvido para emular a transferência de dados entre os principais módulos da plataforma FloripaSat.

1.1 OBJETIVO GERAL

Este trabalho tem como objetivo principal realizar um estudo para incrementar a confiabilidade dos sistemas embarcados de um CubeSat. O incremento da confiabilidade de um CubeSat pode ser alcançado de diversas formas, assim, visando contribuir com outros trabalhos desenvolvidos pelo grupo de pesquisa do SpaceLab, este trabalho tem seu foco no aumento da confiabilidade na transferência de informações entre os sistemas de um CubeSat por meio da mitigação de falhas nas camadas física e de enlace de dados.

O estudo será desenvolvido com base na análise dos métodos de mitigação de falhas em um projeto de um CubeSat, tendo como foco a utilização de um modelo de transferência de dados confiável para melhoria e substituição dos protocolos de comunicação utilizados, atualmente, no FloripaSat. Além da substituição dos barramentos que apresentaram problemas durante o projeto da plataforma FloripaSat, este trabalho tem como foco o envio das mensagens entre os subsistemas de maneira rápida e segura.

1.2 OBJETIVOS ESPECÍFICOS

Visando uma melhor estruturação para o desenvolvimento do trabalho, foram levantados os seguintes objetivos específicos:

- Levantamento do estado da arte, com o intuito de saber em que nível se encontram os estudos realizados, atualmente, sobre o tema.
- Análise das características dos protocolos atuais do FloripaSat, levando em conta os problemas encontrados durante seu projeto e desenvolvimento, com o objetivo de tornar a transferência das informações entre os sistemas da plataforma FloripaSat mais confiável.
- Levantamento das características de desempenho do novo protocolo, como velocidade de processamento, velocidade de transmissão de dados, quantidade de informações transferidas, entre outros.
- Definição de um novo modelo de barramento para ser utilizado na plataforma FloripaSat, visando as novas missões as quais utilizarão a plataforma.
- Desenvolvimento de um trabalho que seja capaz de orientar e auxiliar novos grupos de desenvolvimento de satélites e que contribua com o meio acadêmico com relação à utilização do protocolo CAN.

1.3 MOTIVAÇÃO E JUSTIFICATIVA

Com a constante evolução da plataforma FloripaSat, a busca por melhorias na confiabilidade das informações obtidas em ambiente espacial, bem como, uma maior longevidade do sistema e, consequentemente, das missões futuras, vem tornando-se cada vez mais importantes no desenvolvimento da plataforma.

Para isto, a equipe do SpaceLab vem trabalhando intensamente na melhoria dos processos e procedimentos de testes e validação, como também no uso de componentes mais resistentes e apropriados para o ambiente espacial. Tais procedimentos são necessários, uma vez que a plataforma FloripaSat será utilizada em importantes projetos e missões futuras, entre elas, as missões FloripsaSat-2, Aldebaran-1 e a Constelação Catarina.

Portanto, a necessidade de substituição e melhoria dos barramentos de comunicação levaram a busca de um novo protocolo que fosse confiável, robusto, com um bom índice de tolerância à falhas e fosse uma solução viável para ser implementada em um CubeSat, por este motivo o protocolo CAN foi sugerido, em conjunto com os integrantes do SpaceLab, como uma alternativa.

Também, no trabalho de Miranda et. al. (2020), uma das questões levantadas sobre a escolha de não usar o protocolo CAN no satélite QUETZAL – I foi à falta de experiência com

o protocolo, motivando ainda mais o desenvolvimento desta pesquisa para demonstração das características e cálculos necessários para utilização do protocolo CAN.

O protocolo CAN pode ser uma boa alternativa para melhoria dos protocolos de comunicação atuais do FloripaSat, pois apresenta características como baixo custo, alta confiabilidade, ampla utilização no mercado, uso em diversos componentes e microcontroladores comerciais, IP cores desenvolvidos para FPGAs, entre outros (PLUMMER; ROOS; STAGNARO, 2003).

1.4 ORGANIZAÇÃO DO DOCUMENTO

No capítulo 2 é descrito o referencial teórico, onde são demonstrados os conceitos levantados neste trabalho, como nanossatélites, CubeSats, FPGAs, componentes COTS, confiabilidade, protocolo CAN e camadas ISO/OSI. No capítulo 3 o protocolo CAN é descrito com mais detalhes, onde são descritas as características do CAN relevantes para este trabalho. No capítulo 4 é realizado o levantamento dos trabalhos relacionados e estado da arte, para demonstrar o nível atual da pesquisa sobre o tema proposto, enquanto no capítulo 5 é demonstrada a metodologia matemática, bem como, o modelo sugerido de envio de mensagens para realização da análise do modelo. No capítulo 6 são demonstrados os estudos de caso para aplicação do CAN em FPGAs e microcontroladores, com base nas missões GOMX-5 e FloripaSat-I, respectivamente e finalmente, no capítulo 7, as conclusões sobre o trabalho realizado.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão descritos conceitos básicos para contextualizar o leitor com o assunto abordado ao decorrer desta dissertação, com a intenção de facilitar o entendimento de conceitos como satélites, componentes COTS, FPGAs, MCU (*Microcontroller unit)* para uso espacial, camadas ISO/OSI, confiabilidade de um sistema e o protocolo CAN.

2.1 SATÉLITE

Qualquer corpo que se move em volta de um outro corpo celestial é considerado um satélite, estes objetos podem ser considerados satélites naturais ou artificiais. Satélites artificiais são objetos desenvolvidos por humanos e normalmente lançados na órbita terrestre. Estes objetos podem ser lançados na órbita da Terra, da Lua ou de qualquer outro corpo celestial e a trajetória que ele realiza é que define sua órbita (AGRAWAL; MAINI, 2011; SOUZA, 2007).

A padronização da classificação de satélites segundo sua massa pode variar de acordo com a entidade ou agência a qual a missão está relacionada, sendo a mais utilizada, principalmente para desenvolvimento de CubeSats, demonstrada por Konecny (2004):

- Grandes satélites: com massa superior à 1000 kg;
- Satélites médios: com massa entre 500 e 1000 kg;
- Minissatélites: com massa entre 100 e 500 kg;
- Microssatélites: com massa entre 10 e 100 kg;
- Nanossatélites: com massa entre 1 e 10 kg;
- Picossatélite: com massa entre 0,1 e 1 kg;
- Femtossatélite: com massa menor que 100 g.

2.1.1 CubeSat

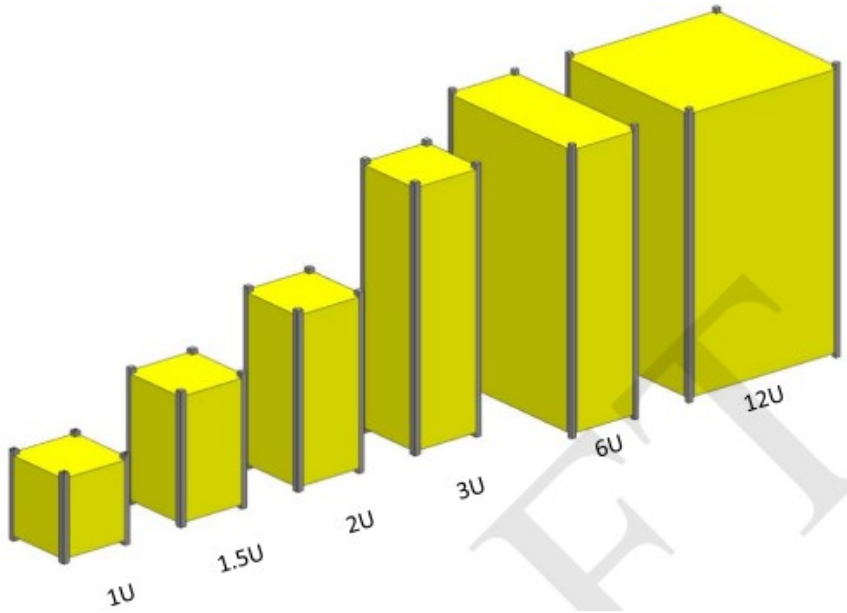
Com o avanço da indústria espacial, a busca por tornar o preço de desenvolvimento de satélites mais acessível e tornar a indústria mais atrativa para novos pesquisadores, levou a criação de um programa para desenvolvimento de pequenos satélites: O CubeSat.

Criado pela Cal Poly em 1999 o CubeSat é um programa que busca padronizar o desenvolvimento de pequenos satélites, estes devem seguir as diretrizes do programa com relação ao seu peso, tamanho e formato (NASA, 2017) (The CubeSat Program, 2020).

A padronização dos CubeSats busca reduzir o custo e o tempo de desenvolvimento, bem como, tornar o acesso ao espaço mais abrangente (CUBESAT, 2014). As características padronizadas de um CubeSat tornam o seu desenvolvimento mais simples, uma vez que a indústria desenvolve dispositivos padronizados e estes são utilizados por diversos grupos de desenvolvimento, barateando o custo e possibilitando a comprovação da eficácia de um dispositivo ao ser utilizado em diferentes missões e projetos.

Os CubeSats podem ser desenvolvidos em diversos tamanhos, de acordo com as características de cada categoria, todos devem ser feitos em formato cúbico e respeitando o tamanho máximo que é definido por "Unidades"[1] e podem ser encontradas nos tamanhos 1U, 1.5U, 2U, 3U, 6U e 12U (MABROUK, 2017), como demonstrado na figura 1.

Figura 1 – Unidades de medida padronizadas para CubeSats



Fonte: The CubeSat Program, 2020.

[1] Padrão utilizado para determinar o tamanho de um CubeSat, sendo que cada 1U possui tamanho 10cm x 10cm x 10cm e peso máximo de 2 Kg.

2.1.2 O FloripaSat

A plataforma FloripaSat foi desenvolvida no SpaceLab por estudantes da graduação e pós-graduação da Universidade Federal de Santa Catarina, com a intenção de desenvolver todos os sistemas de um nanossatélite, capaz de ser adaptado e reutilizado em mais de uma missão. Os principais módulos do FloripaSat são o OBDH (*On Board Data Handling*), o EPS (*Electric Power System*) e o TT&C (*Telemetry, Tracking and Command*).

O OBDH é o sistema responsável por realizar o gerenciamento dos dados armazenando-os em uma memória não volátil para posterior envio para a estação terrestre. O OBDH também decodifica as informações recebidas pelo TT&C e realiza as ações necessárias, bem como, faz o sincronismo das informações de todos os demais sistemas.

O EPS tem a função de captar, armazenar e distribuir a energia para os demais sistemas do satélite. O sistema de captação de energia é realizado pela conversão de energia solar que é armazenada em duas baterias de Lítio. O EPS é o sistema responsável por definir o modo de funcionamento do satélite de acordo com seu nível de energia.

O TT&C é responsável pela comunicação do satélite com a estação terrestre e é divido em dois subsistemas, o Beacon, responsável por enviar o ID do satélite e algumas informações básicas e o Rádio Principal, o qual recebe comandos da estação terrestre e retorna as informações solicitadas. Os comandos recebidos pelo TT&C são direcionados para o OBDH que realiza o desempacotamento dos dados e executa as operações solicitadas.

A primeira vez que a plataforma foi utilizada foi na missão FloripaSat-1, missão que tinha como objetivos principais, realizar a validação em órbita dos módulos do FloripaSat, fornecer serviços de transmissão de rádio amador e realizar testes de novos componentes em ambiente espacial, além de servir como uma missão educacional. (FLORIPASAT, 2020).

2.2 CONFIABILIDADE DE UM SISTEMA

De acordo com Laprie, Randell e Landwehr (2004), o conceito de confiabilidade pode ser definido como a capacidade de evitar a ocorrência de defeitos que sejam mais frequentes e mais graves do que o aceitável, enquanto a ABNT (Associação Brasileira de Normas Técnicas), por meio da NBR-5462, define confiabilidade como a capacidade de um item desempenhar uma função requerida sob condições especificadas, durante um dado intervalo de tempo. (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 1994).

2.2.1 Terminologia.

A seção 2.2.1 foi dedicada à definição dos termos utilizados em (LAPRIE; RANDELL; LANDWEHR, 2004) e que foram utilizados também ao longo da seção 2.2:

Sistema: É uma aplicação que interage com outros sistemas, os quais podem ser hardware, software, humanos ou até mesmo o mundo físico;

Ambiente: São todos os elementos que interagem com o sistema em questão;

Limite do sistema: É a fronteira entre o sistema e seu ambiente;

Função: É o que um sistema se destina a fazer;

Comportamento: É como o sistema implementa as suas funções;

Componentes: Compõem a estrutura, são considerados sistemas e sua recursividade acaba quando são considerados atômicos;

Serviço: É o comportamento do provedor, visto pelo usuário;

Provedor: Sistema que fornece um serviço;

Usuário: É um outro sistema, o qual "recebe" o serviço do provedor;

Defeito: É quando o serviço entregue ocorre de modo diferente do seu funcionamento correto.

Erro: É definido como o desvio no funcionamento do sistema e que faz com que ele sofra um defeito;

Falha: É a causa (concreta ou hipotética) de um erro;

Restauração do serviço: Transição entre o funcionamento incorreto para o funcionamento correto do sistema;

2.2.2 Problemas que afetam a confiabilidade de um sistema.

Os problemas que podem afetar a confiabilidade de um sistema podem ocorrer em qualquer período do seu ciclo de vida. Ameaças à confiabilidade do sistema como falha, erro e defeito, bem como, suas causas e consequências, serão apresentadas com maior detalhe nesta seção.

2.2.2.1 Falha.

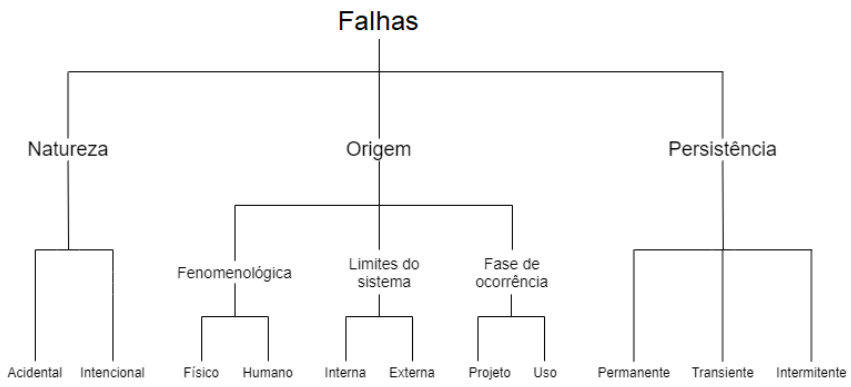
Em Laprie (1992), falhas são diferenciadas em três classes, as quais são classificadas de acordo com sua natureza, origem e persistência.

Com relação a natureza das falhas, elas podem ser classificadas como falha acidental, quando a causa da falha é acidental ou casual e falha intencional, quando a falha é causada de modo deliberado.

Para classificar as falhas de acordo com sua origem é possível subdividi-las em outros três grupos, causas fenomenológicas, limites do sistema e fase de ocorrência. As causas fenomenológicas podem ser físicas ou por erro humano, causas físicas são causadas por um fenômeno físico adverso, enquanto falhas por erro humano ocorrem devido alguma imperfeição do sistema causada pela manipulação humana. As falhas classificadas com relação aos limites do sistema podem ser internas, falhas inerentes ao sistema (problemas de fabricação, degradação dos componentes, entre outros), ou externas, causadas por interferência do meio (perturbações eletromagnéticas, radiação, temperatura, vibração, entre outros). Finalmente, as falhas classificadas com relação à fase de ocorrência, podem ocorrer durante o projeto (neste caso, sendo considerado tanto o período de projeto quanto o período de manutenção) ou podem ocorrer durante o uso.

Falhas temporais podem ser classificadas com relação à sua persistência, sendo classificadas como falhas permanentes, que são falhas as quais o sistema não consegue se recuperar por conta própria, falhas transientes, normalmente associadas a alguma condição passageira, durando apenas algum período de tempo e falhas intermitentes, que são falhas que ocorrem de tempo em tempo e, normalmente, por uma curta duração. A classificação das falhas pode ser vista na figura 2.

Figura 2 – Classificação das falhas.



Fonte: Próprio Autor.

O uso de componentes desenvolvidos e disponibilizados para aquisição em um período de tempo muito próximo ao do desenvolvimento de um projeto também pode ser um problema, pois podem apresentar falhas (conhecidas ou desconhecidas) e, em alguns casos, não ter documentação e nem reportes de erros suficiente para que funcione de modo satisfatório em um projeto.

Para evitar muitos destes problemas, ao longo dos anos foram desenvolvidos métodos para mitigar as falhas, classificados em quatro grupos principais (LAPRIE; RANDELL; LANDWEHR, 2004):

- Prevenção à falha: Como prevenir a ocorrência ou introdução de uma falha no sistema;
- Tolerância à falha: Como manter a prestação do serviço correta, mesmo com a ocorrência de falhas;
- Remoção de falhas: Como reduzir o número ou a severidade de falhas;
- Previsão de falhas: Como estimar o número de falhas, incidências futuras e suas consequências;

2.2.2.2 Erro.

Nos trabalhos de Laprie (1992) e de Laprie, Randell e Landwehr (2004), erros são definidos como os responsáveis por levar um sistema a sofrer um defeito. O que define se um erro, realmente, irá gerar um defeito depende de dois fatores principais, a composição do sistema e as atividades do sistema.

A composição do sistema refere-se principalmente ao seu nível de redundância, que no caso pode ser redundância intencional, com o objetivo de mitigar falhas ou redundância não intencional, que acaba tendo o mesmo efeito que a redundância intencional. Por outro lado, as atividades do sistema podem ser responsáveis por sobrescrever um erro antes que ele se torne um defeito.

2.2.2.3 Defeito.

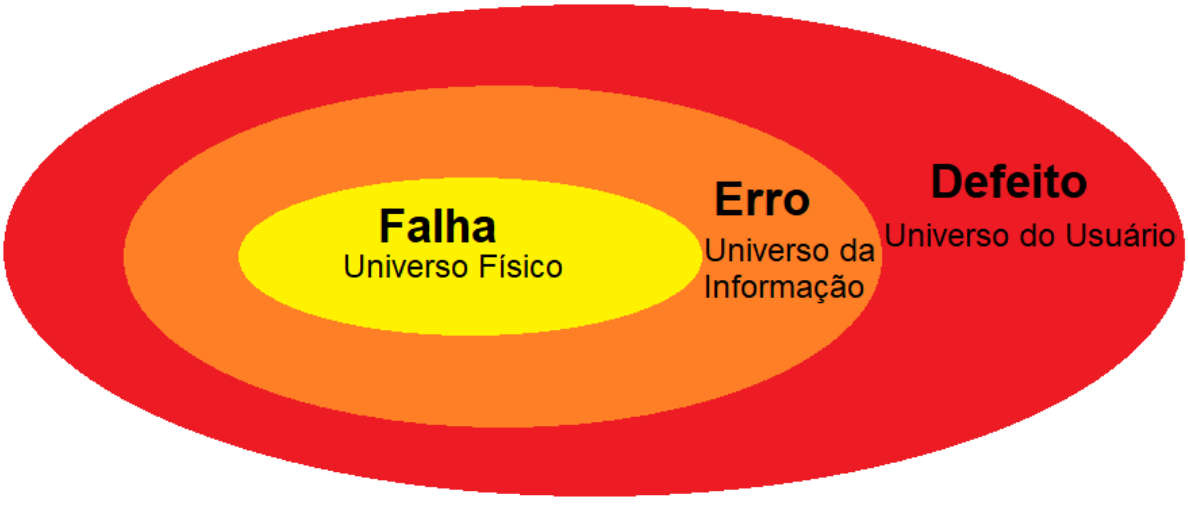
Defeitos são eventos que ocorrem e que afetam o funcionamento de um sistema, levando-o a fornecer resultados incorretos, os defeitos são classificados conforme seu domínio, detecção, consistência e consequências ao ambiente do usuário.

Com relação ao domínio, um defeito pode ser de conteúdo, quando o conteúdo da informação entregue não condiz com o previsto para o sistema, ou de tempo, quando o tempo de entrega das informações ocorre de modo divergente ao definido pelas funções do sistema. A detecção de um defeito ocorre por meio da sinalização, para o usuário, que aquele sistema não entregou a informação ou entregou de forma errada. Normalmente, estas sinalizações são feitas por mecanismos de detecção que monitoram o sistema para avisar caso o sistema pare de funcionar como planejado.

Por outro lado, um defeito é considerado consistente quando todos os usuários do sistema o percebem da mesma maneira, todavia quando nem todos os sistemas percebem o defeito ou percebem de maneiras diferentes, estes são definidos como inconsistentes. E por fim, as consequências de um defeito podem ser classificadas como defeitos menores, onde as consequências do defeito se assemelham (na questão de custo para o sistema) aos benefícios de gerados pela entrega correta das informações, e defeitos catastróficos, quando a consequência do mal funcionamento do sistema pode gerar um custo muito maior que os benefícios gerados pela entrega do serviço de forma correta (LAPRIE; RANDELL; LANDWEHR, 2004).

Defeitos são gerados por erros e fazem com que o sistema funcione de forma divergente a sua proposta ou não funcione de forma alguma. Um defeito pode ativar uma falha existente no sistema que estava dormente, falhas também podem ser ativadas por algum fator externo ou por alguma rotina interna. As falhas ativas podem gerar erros que podem ser propagados pelo sistema até o momento em que viram um defeito, formando assim, uma “cadeia de ameaças”, na figura 3 é demonstrada a relação entre falha, erro e defeito.

Figura 3 – Definição de falha, erro e defeito.



Fonte: Próprio Autor.

2.2.3 Prevenção à falha.

Para prevenção de falhas, não é levado em conta apenas as falhas, mas também qualquer anormalidade (funcionamento divergente do esperado) no comportamento do sistema ou dos seus componentes. Para isto, parte do processo de prevenção à falha é o diagnóstico preciso do sistema. Primeiramente, uma análise sobre possíveis comportamentos anormais deve ser feita, para posterior diagnóstico da causa e da localização da falha e, por fim, devem ser geradas as sugestões para remoção das falhas (BORDASCH; GÖHNER, 2013).

As constantes melhorias nos processos de desenvolvimento, tanto de software quanto de hardware, visam à redução da injeção de falhas no sistema, estas melhorias baseiam-se principalmente no registro de falhas e na eliminação da causa das falhas por meio da alteração no seu processo de desenvolvimento (LAPRIE; RANDELL; LANDWEHR, 2004).

2.2.4 Tolerância à falha.

Segundo Laprie, Randell e Landwehr (2004), a tolerância a falhas é realizada mediante a detecção de erros e correção do sistema com o objetivo de evitar defeitos no sistema. A tolerância a falha se diferencia da manutenção por não necessitar de nenhum agente externo ao sistema. Desta maneira, a detecção de erro, o gerenciamento de erro e o gerenciamento de falhas trabalham de forma conjunta na tolerância a falhas, com técnicas como *Rollback* [2] e *Rollforward*[3].

As técnicas de tolerância de falhas são fortemente relacionadas com a suposição de falha, desta forma, uma falha só será tolerada se houver na fase de desenvolvimento a suposição da possibilidade desta falha ocorrer. Sendo assim, uma técnica de tolerância à falhas muito utilizada é a redundância, assumindo que uma falha pode ocorrer e para isto o sistema necessita de um outro caminho para as informações serem transferidas.

---

[2] *Rollback:* Técnica utilizada para voltar a um estado anterior ao da ocorrência da falha, onde o sistema estava funcionando de forma correta.
[3] *Rollforward:* Técnica utilizada para que o sistema refaça suas atividades e sobrescreva os resultados para garantir a sua consistência.

2.2.5 Remoção de falhas.

A remoção das falhas na fase de desenvolvimento do sistema ocorre em três etapas: verificação, diagnóstico e correção. Na etapa de verificação são constatados os pontos onde o sistema não atende as especificações para então, ser feito o diagnóstico da origem e o motivo da falha e posteriormente realizar as correções necessárias.

As verificações em um sistema podem ser estáticas ou dinâmicas, na verificação estática as análises são feitas sem, necessariamente, colocar o sistema para executar, podendo ser realizada por meio de análises do fluxo de dados, fluxo do programa (software), busca por vulnerabilidades ou até com um modelo baseado no sistema real, já na verificação dinâmica o sistema é posto para funcionar em execução parcial ou total para ser testado.

A remoção de falhas detectadas durante a fase de uso deve ocorrer durante as manutenções, que podem ser corretivas ou preventivas. Na manutenção corretiva são removidas as falhas sinalizadas por intermédio do envio do aviso de um erro, enquanto na manutenção preventiva as falhas são removidas do sistema antes mesmo de gerarem um erro (LAPRIE, 1992).

2.2.6 Previsão de falhas.

A previsão de falhas ocorre com a avaliação do comportamento do sistema e pode ser tanto qualitativa, onde são identificadas e classificadas as combinações de eventos que podem levar o sistema a sofrer um defeito, ou quantitativa, que avalia probabilisticamente a capacidade de o sistema atingir seu objetivo.

As duas principais formas de análise probabilística, são o modelamento e a testagem, estas formas são complementares, uma vez que, o modelamento precisa de validação por meio de uma base de dados que pode ser obtida mediante a realização de testes. O modelamento é baseado em duas fases, a primeira por meio da construção de um modelo baseado nas características funcionais dos componentes e dos processos do sistema e a segunda por meio do processamento do modelo para obter os parâmetros de confiabilidade e de falhas (LAPRIE; RANDELL; LANDWEHR, 2004).

2.3 PRÁTICAS PARA INCREMENTO DE CONFIABILIDADE EM NANOSSATÉLITES

Nesta seção, serão expostas as práticas mais comuns para incremento na confiabilidade de um nanossatélites, levando em conta as restrições que este modelo de satélite possui, com relação à massa, volume, processamento de dados, consumo de energia e custo. Assim, serão descritas algumas práticas adotadas por grupos desenvolvedores de nanossatélites, incluindo o SpaceLab, o qual busca aproveitar da melhor forma possível os recursos disponibilizados.

Requisitos de tolerância a falhas sempre são um desafio no desenvolvimento de sistemas, pois sempre que um método de tolerância a falha é introduzido no sistema, há um custo em hardware, software, área, energia ou de orçamento.

Mamoutova e Antonov (2015), no seu trabalho, tratam sobre o projeto confiável da eletrônica de um nanossatélite, como mencionado pelos autores, um nanossatélite pode ser considerado como uma aplicação crítica de dados. Com isto, em caso de um defeito em um nanossatélite não haveriam, necessariamente, grandes danos a humanidade ou a indústria, porém poderiam ser perdidos dados valiosos da missão que seriam transmitidos para a terra. Desta maneira, o nível de confiabilidade de um nanossatélite está fortemente relacionado com o propósito da missão para qual ele foi projetado.

Em sistemas críticos, como em nanossatélites, deve-se utilizar de forma consciente componentes de hardware e métodos de incremento da tolerância a falha em software, que tragam vantagens e aumentem a confiabilidade do projeto. Muitas vezes, o incremento na confiabilidade pode acarretar no aumento do consumo energético, tempo de processamento, área utilizada e do orçamento do projeto, recursos que são limitados quando se trata do desenvolvimento de um nanossatélite. Assim, vale ressaltar que o um grupo desenvolvedor de nanossatélites deve aproveitar ao máximo os recursos disponibilizados no projeto, unindo robustez de hardware e software.

2.3.1 Software

Como demonstrado no trabalho de Swartwout (2018), problemas de software não costumam ser os grandes causadores de defeitos em CubeSats, estando relacionado, principalmente, ao fato do software ser uma parte fundamental no desenvolvimento de um sistema de processamento de dados para ambiente espacial. Por este motivo, diversas técnicas são utilizadas para incremento da confiabilidade do sistema, entre as técnicas utilizadas para aumentar a confiabilidade do software podem ser citados códigos de correção de erro,

redundância de tarefas, múltiplas versões de programas, algoritmos para análise e correção de erros, entre outras.

Em complemento a análise do software em Yusong et al. (2018) são demonstrados conceitos de Garantia de Qualidade de Software (SQA, do inglês Software Quality Assurance), onde são demonstradas as etapas seguidas no desenvolvimento de um software de qualidade, como demonstrada na tabela 1.

Tabela 1 - Etapas de desenvolvimento de um software de qualidade.

	I - Projeto	II – Análise dos Requisitos	III - Design	IV – Desenvolvimento do Software	V – Teste de Software	VI – Finalização do Projeto	VII – Entrega do Produto
Trabalho principal	Aprovação do Projeto e Revisão dos requisitos do produto.	Planejamento do projeto, análise e revisão dos requisitos.	Esboço do projeto e projeto detalhado	Desenvolvimento do código fonte e documentação.	Testes de Software	Testes do projeto e revisão	Entrega do produto
Atividades SQA	Definir equipe de SQA	Fazer o planejamento de SQA	Realizar de SQA, revisão do projeto, teste de software, reportar erros de SQA.				Fazer análise estática do projeto

Fonte: Adaptado de Yusong et al., 2018.

Por fim, no trabalho de Kulyagin, Tsarev e Kovalev (2015) é demonstrada a técnica de tolerância a falha denominada multi-versão de programa ou NVP (do inglês, *n-version programming*), onde duas ou mais versões de um programa são desenvolvidas com funcionalidades equivalentes e com as mesmas especificações e restrições. Neste trabalho foram demonstrados os resultados obtidos para os 9 sistemas considerados críticos, como pode ser visto no quadro 1, onde é demonstrado o incremento no custo do sistema, bem como, na confiabilidade em comparação com um programa único.

Quadro 1 – Incremento do custo Vs. Confiabilidade NVP.

Módulo	Custo		Confiabilidade	
	Programa único	NVP	Programa único	NVP
4	14	22	0,9689	0,99842 (+2,95%)
5	15	29	0,9869	0,99998 (+1,31%)
16	15	29	0,9963	0,99997 (+3,67%)
17	8	14	0,9227	0,99394 (+7,12%)
18	23	51	0,9853	0,99999 (+1,47%)
19	16	23	0,9867	0,99956 (+1,29%)
48	39	72	0,9692	0,99851 (+2,93%)
51	62	170	0,9862	0,99998 (+1,38%)
52	66	131	0,9964	0,99995 (+3,55%)

Fonte: Adaptado de Kulyagin, Tsarev e Kovalev, 2015.

2.3.2 Hardware

Como grande parte do mau funcionamento de um nanossatélite provém dos componentes eletrônicos, muitas vezes afetados pela radiação, o foco principal da análise será para estes componentes como memórias, microprocessadores e FPGAs, bem como, nos barramentos que fazem a intercomunicação entre os sistemas.

2.3.2.1 Componentes tolerantes à radiação.

Uma das melhores opções para incrementar a confiabilidade de sistema utilizado em aplicações espaciais é a utilização exclusiva de componentes tolerantes à radiação. Porém, devido ao fato de componentes RadHard apresentarem alto custo, menor poder de processamento e maior consumo energético em comparação com componentes comerciais, esta opção torna-se inviável, sendo necessário utilizar outros métodos para incremento da confiabilidade.

Desta forma, a utilização destes componentes deve ser limitada apenas para situações onde os pré-requisitos da missão tornem o seu uso indispensável. Para contornar este problema, frequentemente são utilizados componentes COTS associados a algum outro método para aumentar a sua confiabilidade, como a redundância dos componentes, aplicada principalmente em sistemas essenciais, como os processadores e computadores de bordo do satélite.

2.3.2.2 Redundância de componentes.

A redundância de componentes é uma prática muito utilizada em projetos que requerem uma confiabilidade maior no processamento, transferência e armazenamento dos dados, principalmente quando falamos de nanossatélites, devido a utilização de componentes que, normalmente, não são de uso específico para ambiente espacial e radioativo. Para contornar os problemas que estes componentes podem apresentar, são utilizados componentes redundantes, além de utilizar, preferencialmente, componentes com histórico de sucesso na utilização em pequenos satélites e CubeSats.

Porém, a redundância de componentes afeta diretamente o consumo energético e a área de utilização, uma vez que dois ou mais componentes utilizam energia e espaço no satélite para realizar a mesma tarefa.

2.3.2.3 Multiprocessadores

Um modo de aumentar a confiabilidade no processamento das informações é a utilização de mais de um processador para realizar o tratamento das informações. Este modelo pode ser utilizado de diversas formas, pois os sistemas podem trabalhar em paralelo de forma redundante, podem trabalhar em paralelo de forma que cada processador seja responsável por uma parte do gerenciamento das informações ou podem trabalhar de modo que um dos processadores seja responsável por validar as informações do processador principal (*Lockstep*). Para este último caso, atualmente, vem tornando-se cada vez mais comum o uso de microprocessadores utilizados na indústria automobilística, como no caso do MCU TMS570 que possui dois processadores trabalhando em *Lockstep*.

Em casos de utilização de computação distribuída, ou seja, quando ambos os processadores funcionam em paralelo, porém dividem outros recursos como memórias ou barramentos de comunicação, deve haver um bom gerenciamento destes recursos, utilizando semáforos, unidades de gerenciamento de memória, gerenciamento de mensagens, entre outros. A utilização destes recursos pode acarretar em um incremento na complexidade do software e consequentemente o consumo energético e o processamento das informações.

2.3.2.4 Sistemas híbridos

No trabalho de George e Wilson (2018), a definição de sistemas híbridos é utilizada tanto para sistemas compostos por microcontrolador e FPGA, quanto para sistemas compostos por processadores RadHard e COTS. Este modelo de computação híbrida vem ganhando espaço na indústria de nanossatélites por permitir usufruir das vantagens de cada sistema, ao mesmo tempo que reduz os efeitos negativos de utilizar apenas uma destas tecnologias. Enquanto componentes COTS possuem baixo custo, baixo consumo energético e melhor performance, componentes RadHard dão uma maior segurança para o sistema.

No SpaceLab, estão sendo desenvolvidas pesquisas sobre sistemas híbridos com FPGA tolerante à radiação e microprocessador embarcado, como é o caso da Payload-X, desenvolvida para ser lançada na missão FloripaSat-1. No projeto da Payload-X foi utilizada a FPGA NX-Medium, desenvolvida pela NanoXplore, com uma FPGA tolerante à radiação e com microprocessador ARM Cortex-R5. Atualmente, uma nova versão está sendo desenvolvida (Payload-XL) para ser lançada na missão GOMX-5, com previsão de lançamento para 2022.

2.3.2.5 Memórias

No desenvolvimento de um nanossatélite confiável, também é necessário projetar um sistema confiável de armazenamento de dados. O sistema de armazenamento será o responsável por armazenar as informações transmitidas entre os sistemas de um satélite, bem como, armazenar informações importantes para o funcionamento do satélite. Um exemplo de dados importantes que deve ser armazenado é o Bit Stream, enviado da estação terrestre para o satélite, caso este possua uma plataforma reconfigurável, como no caso da Payload-X e Payload-XL.

Nos trabalhos de Gupta e Shahi (2016) e Khaled e Zhang (2017) é feito um levantamento sobre o projeto da arquitetura de memória para nanossatélites, onde os autores buscam uma solução capaz de balancear consumo energético, custo, confiabilidade e capacidade de processamento de dados. Para isto, os autores realizaram um estudo baseado na união de memórias Flash e Memórias de Acesso Aleatório Estática (SRAM do inglês, *Static Random Access Memory*), pois por mais que memórias SRAM sejam mais rápidas, algumas características das memórias Flash tornam seu uso uma opção bastante interessante. Entre as características da memória Flash estão armazenamento não volátil, menor consumo energético, menor consumo em área e menor custo. Porém, o uso de memória Flash requer que sejam desenvolvidos métodos para melhorar sua performance, principalmente, em relação a sua

velocidade de leitura e escrita, por isto a memória SRAM é utilizada como um buffer temporário para posterior escrita na memória Flash.

Atualmente, grupos desenvolvedores de nanossatélites estão estudando o uso de FPGAs baseadas em memória SRAM, como pode ser visto nos trabalhos de Benevenuti et al. (2019) e Shashidhara, Jadhav e Kim (2020).

O trabalho de Benevenuti et al. (2019) foi desenvolvido no Brasil, em uma parceria entre Universidade Federal de Santa Maria (UFSM), Universidade Federal do Rio Grande do Sul (UFRGS) e Instituto Nacional de Pesquisas Espaciais (INPE), denominada missão NanosatC-BR2. Em ambos os trabalhos foram utilizados o método de Redundância Modular Tripla (TMR, do inglês *Triple Modular Redundancy*) para aumentar a confiabilidade do sistema. Na Redundância Modular Tripla, três componentes realizam a mesma tarefa e no final se a informação for igual em pelo menos dois, ela é validada como uma informação confiável.

2.3.3 Comunicação entre subsistemas

A intercomunicação dos sistemas de um nanossatélite é uma parte importante no seu projeto, visto que as restrições de área tornam inviável a utilização de barramentos complexos, enquanto a restrição no orçamento faz com que seja necessário utilizar barramentos de comunicação comuns aos diversos componentes que integram o satélite. Estas características tornam a escolha do barramento limitada, principalmente, aos barramentos mais comuns que são integrados aos processadores e periféricos (I2C, UART, SPI, CAN, etc.).

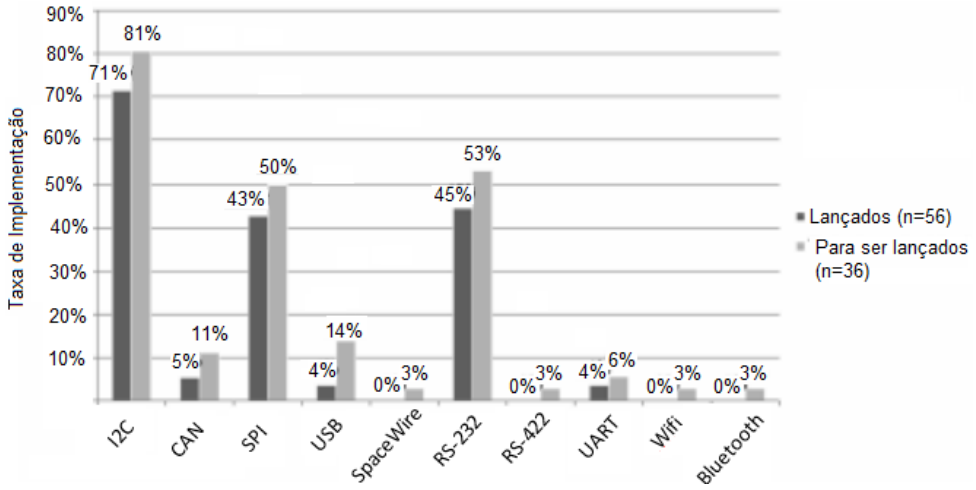
Bouwmeester, Langer e Gill (2016), fazem um estudo baseado em 60 satélites lançados e 44 que ainda seriam lançados para avaliar as interfaces dos barramentos utilizados, neste trabalho eles concluíram que o barramento I2C, amplamente utilizado na intercomunicação dos sistemas de nanossatélites, é responsável por diversos defeitos catastróficos no satélite e por uma grande quantidade de travamento no barramento [4]. Este trabalho corrobora com a observação feita no desenvolvimento do FloripaSat-1, uma vez que o barramento I2C gerou diversos problemas na comunicação do satélite, sendo o travamento do barramento o principal deles. Para mitigar o problema de travamento do barramento I2C foi necessário reformular

[4] O travamento do barramento I2C ocorre quando um nó não responde o mestre mantendo o barramento em um estado “ocupado” impedindo que o mestre inicie uma nova transmissão.

todos os barramentos que utilizavam I2C para que estes ficassem o mais simples e com o menor número de escravos possível.

Neste trabalho, também é possível verificar que entre os protocolos de comunicação que já vinham sendo utilizadas nos satélites lançados, o CAN e USB foram os que tiveram o maior crescimento relativo, como pode ser visto na figura 4.

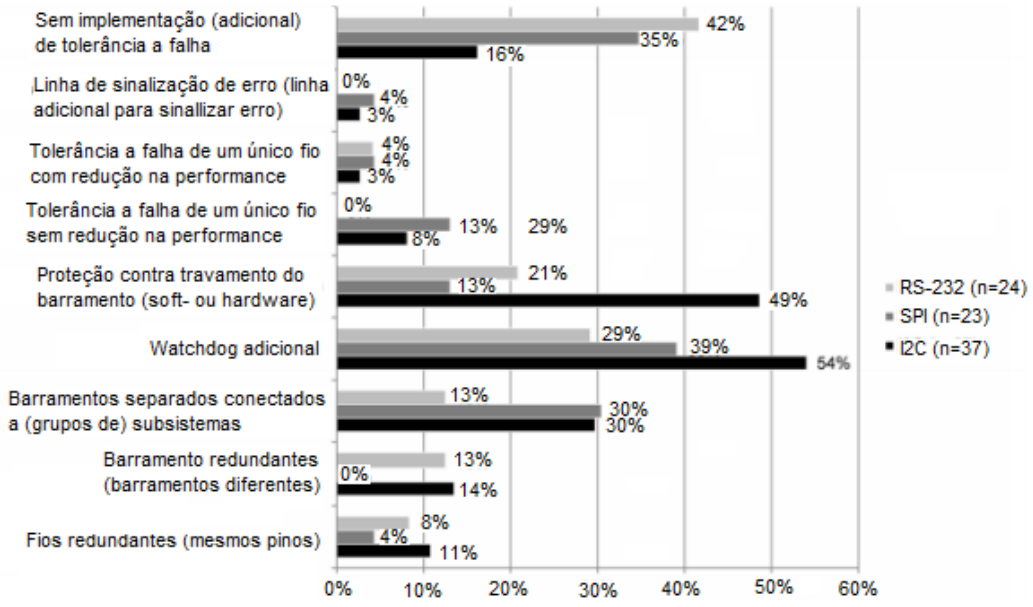
Figura 4 – Barramentos utilizados nos satélites analisados.



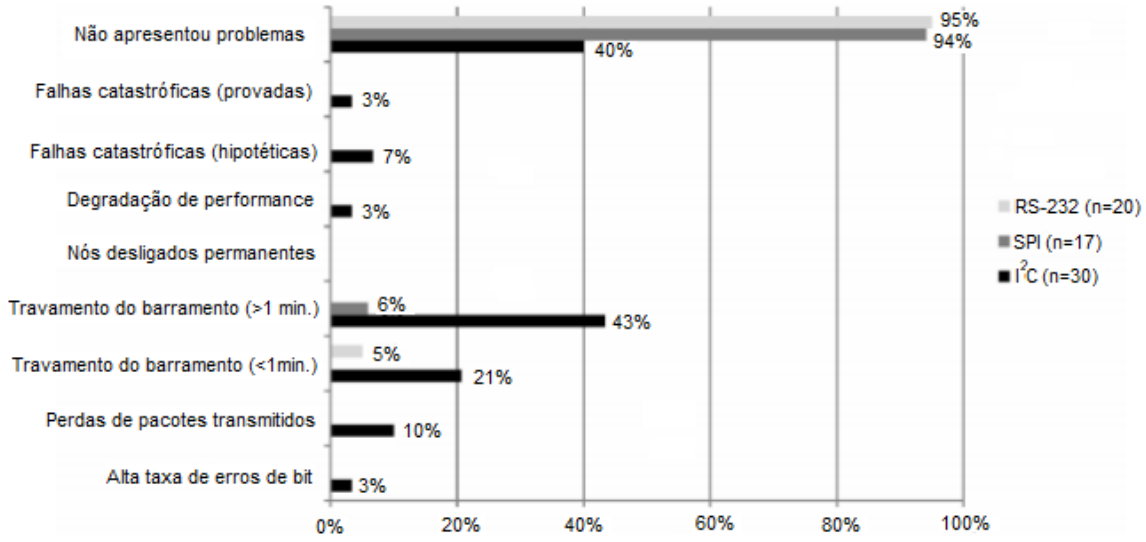
Fonte: Bouwmeester, Langer e Gill, 2016.

Nas figuras 5.a e 5.b é possível verificar que o I2C foi o barramento que mais precisou de métodos de tolerância a falha e foi o que mais apresentou problemas durante as missões.

Figura 5 – Comparativo entre protocolos RS232, I2C e SPI.



(a)



(b)

Fonte: Adaptado de Bouwmeester, Langer e Gill, 2016.

No trabalho de Kimm e Jarrell (2014) o protocolo CAN foi escolhido como parte do projeto de um barramento tolerante à falha, devido suas características de baixo custo para o sistema em relação a consumo, espaço, energia e processamento, uma vez que não há necessidade de realizar grandes modificações nos sistemas de intercomunicação do satélite.

Com o crescimento do uso de componentes desenvolvidos, primariamente, para a indústria automobilísticas no projeto de satélites, o barramento CAN ganhou destaque entre estas tecnologias, pois além do custo da implementação ser pequeno, ele reduz a necessidade de gerenciamento de erros no barramento pelo processador, por possuir arbitragem não destrutiva, entre outras características.

Visto que já existem estudos sendo desenvolvidos pela equipe do SpaceLab para incremento na confiabilidade no processamento e no armazenamento de dados mediante a utilização de componentes híbridos RadHard e do uso de memórias FRAM (Ferroelectric RAM), buscou-se uma alternativa para incremento da confiabilidade na transferência das informações pelos barramentos de interconexão dos sistemas embarcados do FloripaSat.

Com os estudos demonstrados nesta seção foi definido o protocolo CAN como forma principal de transferência de dados e para substituição do barramento I2C. Assim, a escolha de incrementar a confiabilidade por intermédio da substituição do arcabouço de intercomunicação do satélite, deu-se por ser uma das opções que geraria o menor custo ao sistema, ao mesmo tempo que busca sanar um problema encontrado no desenvolvimento da plataforma FloripaSat.

2.4 PROTOCOLO CAN.

O protocolo CAN foi desenvolvido pela empresa Robert Bosch GmbH e apresentado no ano de 1983, o CAN foi desenvolvido, inicialmente, para a indústria automobilística com o intuito de melhorar a comunicação entre os sistemas eletrônicos devido ao crescente número de Unidades Eletrônicas de Controle (ECU – *Electronic Control Unit)* inseridas nos veículos (BURJE; KARANDE; JAGADALE, 2014). Por ser um protocolo confiável, de baixo custo, fácil implementação e atingir uma velocidade de transmissão de dados de até 1 Mbps, além de possuir um sistema de gerenciamento de colisão não destrutivo, muitas outras áreas começaram a utilizá-lo e, atualmente, é amplamente utilizado em áreas como robótica, automação, área médica e muitos estudos mostram que é um bom protocolo para utilização na área espacial também.

Um diferencial do CAN é a simplicidade do protocolo, pois ele é composto apenas por duas camadas do modelo ISO/OSI, a camada de Enlace de dados e a camada Física. Na camada de Enlace de dados há uma divisão em duas subcamadas, o Controle de Enlace Lógico (LLC – *Logical Link Control)* e o Controle de Acesso ao Meio (MAC – *Medium Access Control)*. Na figura 6 são listadas algumas características de cada camada.

Figura 6 – Camadas do protocolo CAN

Fonte: Adaptado de BOSCH, 1991.

Para uma melhor compreensão sobre o protocolo CAN, no capítulo 3 serão detalhadas todas as características do protocolo e seu modo de funcionamento.

2.5 MODELO ISO/OSI.

Em 1977 foi criado, pela Organização Internacional de Padronização (ISO, do inglês *International Organization for Standardization*), um subcomitê (SC16) para Interconexões de Sistemas Abertos (OSI, *Open Systems Interconnection*, em inglês) visto a necessidade de padronização das redes de informática. O SC16 então criou um modelo de arquitetura em camadas, sendo composto por sete camadas (camada de aplicação, camada de apresentação, camada de sessão, camada de transporte, camada de rede, camada de enlace e camada física) que posteriormente foi adotado pelo comitê técnico em processamento de dados (TC97) (ZIMMERMANN, 1980).

A separação, em camadas, de uma arquitetura de transferência de dados complexa, torna a sua utilização mais simples, visto que assim, as mudanças em uma camada não alteram o sistema como um todo, contanto que ela utilize os mesmos serviços da camada abaixo dela e forneça os mesmos serviços para a camadas acima (KUROSE; ROSS, 2012).

Camada de Aplicação: É a camada mais alta da arquitetura e tem como objetivo servir diretamente a aplicação, todas as demais camadas servem apenas para dar suporte à camada de aplicação. A aplicação pode ser a fonte ou o destino dos dados que serão transferidos pela rede. Parte do processo da aplicação é composto pela camada de aplicação, onde é feito o encapsulamento da informação a ser transmitida ou desencapsulamento da informação recebida. A informação que é transferida dentro da camada de aplicação é denominada mensagem (ZIMMERMANN, 1980) (KUROSE; ROSS, 2012).

Camada de Apresentação: É responsável por auxiliar a camada de aplicação fornecendo os serviços necessários para que ela interprete o significado dos dados trocados (ZIMMERMANN, 1980). É por intermédio da camada de apresentação que duas aplicações diferentes podem se comunicar, nesta camada é feita a "tradução" das informações.

Camada de Sessão: É responsável por dar suporte as entidades de apresentação, vinculando ou desvinculando duas entidades de apresentação e realizando o controle, delimitação e sincronismo na troca de informações (ZIMMERMANN, 1980).

Camada de Transporte: É na camada de transporte que é realizado o controle de qualidade das mensagens que serão entregues a camada de rede, também é na camada de transporte que é feita a correção de erros e controle de fluxo, garantindo a entrega da mensagem da camada de aplicação para seu destino e seus dados são denominados segmentos (KUROSE; ROSS, 2012).

Camada de Rede: A camada de rede fornece os meios para a troca de dados entre duas entidades de transporte, nesta camada é feito o roteamento e comutação da informação, que aqui é denominada pacote (ZIMMERMANN, 1980) (KUROSE; ROSS, 2012).

Camada de Enlace de dados: Nesta camada que é realizado o controle de fluxo da transmissão dos dados, detecção e correção de erros do nível físico. (ZIMMERMANN, 1980) (KUROSE; ROSS, 2012). No protocolo CAN é nesta camada que são definidos os filtros para recepção de mensagens, as notificações de sobrecarga e o gerenciamento da recuperação das mensagens, bem como, o encapsulamento/desencapsulamento das mensagens, codificação e decodificação com base no *bit stuffing*, gerenciamento de acesso ao meio, detecção e sinalização de erros, entre outros.

Camada Física: Nesta camada que é realizada a transferências dos bits de um módulo para outro, sendo assim, é na camada física que são definidas as características mecânicas, elétricas e funcionais para estabelecer e manter a conexão entre os módulos (ZIMMERMANN, 1980) (KUROSE; ROSS, 2012). No protocolo CAN esta camada é responsável por realizar a codificação e decodificação dos bits, o *bit timing* e a sincronização entre os módulos, também é nesta camada que são definidas as características do *transceiver* CAN.

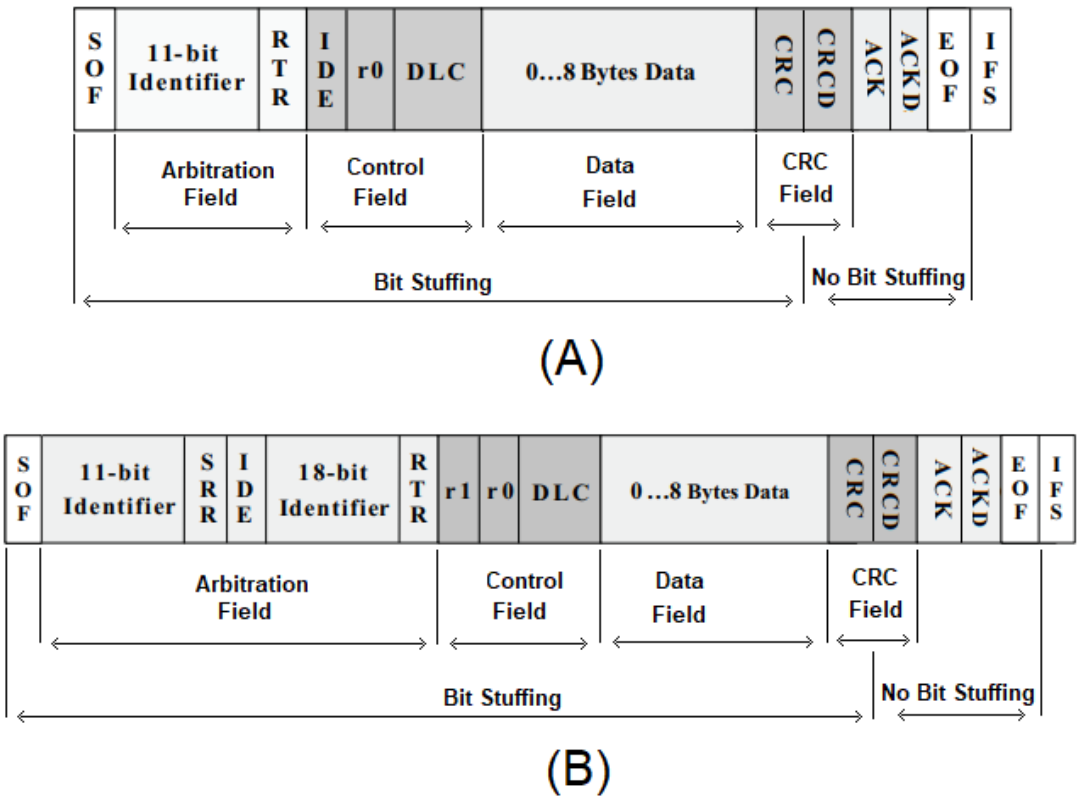
3 CARACTERÍSTICAS DO CAN

Neste capítulo serão detalhadas as principais características do CAN, como o formato dos quadros CAN, os tipos de quadros enviados, forma de envio no barramento, detecção de erro, entre outros.

3.1 FORMATOS DOS QUADROS CAN

No protocolo CAN existem dois formatos de quadros diferentes, o quadro padrão que contém 11 bits para identificação da mensagem e o quadro estendido com 29 bits de identificação. Os quadros CAN são compostos por quatro diferentes campos, campo de arbitragem, campo de controle, campo de dados e campo de CRC (Cyclic Redundancy Check), (BOSCH, 1991). Na figura 7 são demonstrados o quadro padrão (a) e o quadro estendido (b), seguidos de uma breve explicação sobre sua formatação.

Figura 7 – Formato dos quadros CAN.



Fonte: Adaptado de Texas Instruments, 2002.

3.1.1 Campo de Arbitragem

O campo de Arbitragem (ou *Arbitration field,* em inglês) é a parte localizada logo após o bit SOF (*Start of Frame* ou Início do Quadro, em português) e abrange desde os bits de identificação até o RTR (*Remote Transmission Request* ou Requerimento de Transmissão Remota, em português). Para o formato de quadro padrão, o campo de arbitragem é composto pelo bit SOF, o primeiro bit (dominante) a ser enviado no barramento, que é responsável por iniciar a comunicação e sincronizar todos os nós. Logo após o SOF são enviados os 11 bits de identificação do nó transmissor e o RTR (*Remote Transmission Request* ou Requerimento de Transmissão Remota, em português). No caso do nó transmissor estar solicitando um dado de outro nó, o RTR deve ser composto por um bit dominante e neste caso os 11 bits de identificação devem ser referentes à identificação do nó que deve enviar as informações solicitadas.

Para o quadro estendido, após os 11 bits mais significativos de identificação o quadro é composto pelo SRR (*Substitute Remote Request* ou Substituto do Requerimento Remoto, em português), responsável por substituir o RTR e manter o mesmo formato do quadro. No quadro estendido, o IDE (Identifier Extension ou Identificador de extensão, em português) fica dentro do campo de Arbitragem e pode ser composto por um bit dominante (no caso do quadro ser padrão) ou por um bit recessivo (no caso do quadro ser estendido) e, por fim, os 18 bits menos significativos da identificação do nó e o RTR.

3.1.2 Campo de Controle

O Campo de controle (ou *Control field,* em inglês) é composto pelo DLC (Data Length Code ou Código do comprimento dos Dados, em português), composto por quatro bits e que deve informar quantos bytes de dados serão enviados no quadro, também fazem parte do campo de controle do quadro estendido os bits reserva, r0 e r1 enquanto o quadro padrão é composto pelo bit reserva r0 e o IDE.

Os bits r0 e r1, são reservados no protocolo CAN para possível utilização em aplicações futuras, porém alguns controladores CAN dão maior liberdade para os desenvolvedores para utilizar estes bits caso necessitem em alguma aplicação.

3.1.3 Campo de Dados

O campo de dados (ou *Data field*, em inglês) é composto por até 64 bits (8 bytes), conforme definido na DLC e nele são adicionadas as informações que devem ser transmitidas para um ou mais nós do barramento. O campo de dados pode estar vazio no caso do nó transmissor estar solicitando uma informação de outro nó da rede, desta maneira, os bits de identificação do quadro devem ser referentes a identificação do nó que deve enviar as informações solicitadas.

3.1.4 Campo de CRC e ACK

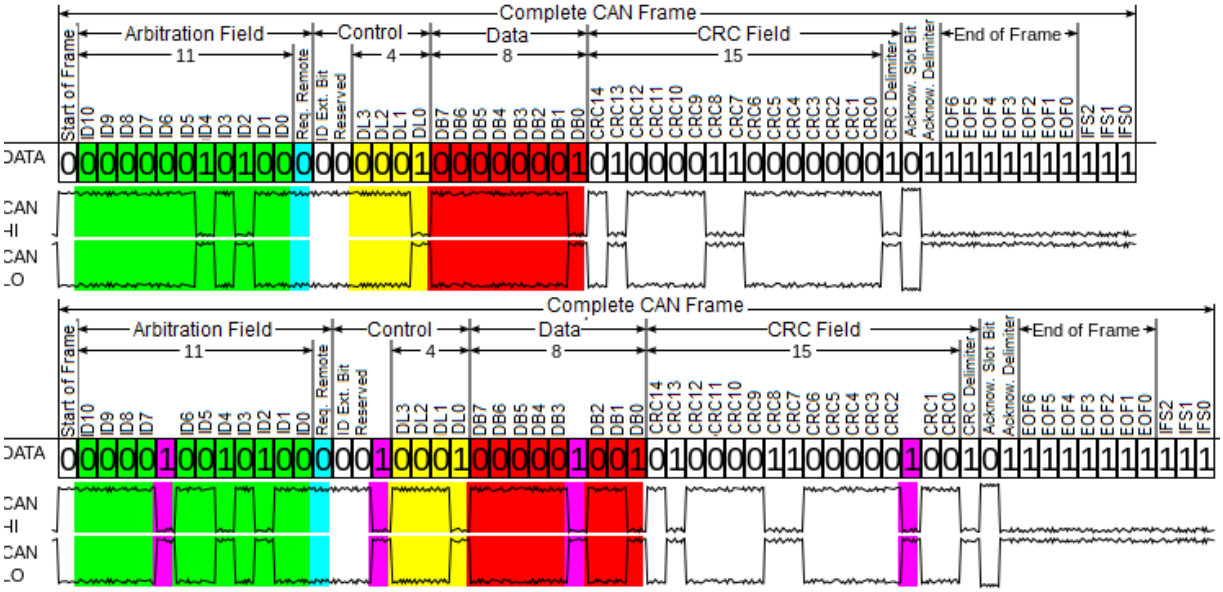
O Campo de CRC (ou *CRC field*, em inglês), é responsável por realizar a checagem da informação contida em todo quadro, no caso do protocolo CAN é utilizado um CRC polinomial de 15 bits ($X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + X^{0}$), após os 15 bits do CRC é adicionado o bit delimitador de CRC, bit recessivo de valor fixo.

O cálculo de CRC é realizado após o Bit Stuffing e se o receptor verifique que o CRC confere com os bits recebidos, ele deve enviar um bit de ACK (Acknowledgment) seguido por um bit delimitador de ACK, que também possui um valor fixo (recessivo). Ao final do quadro são enviados 7 bits recessivos de Fim do Quadro (*End Of Frame*, EOF) e mais 3 bits recessivos (*Inter Frame Space*, IFS) que tem a função de dar um tempo entre as mensagens para que sejam "carregadas" no buffer. Os bits de valores fixo, como o SOF, CRCD, ACKD e EOF, tem uma função importante para checagem de erros.

3.1.5 Bit Stuffing

Pelo fato do protocolo CAN ser NRZ (*non-return to zero*) é necessário o bit stuffing para assegurar a confiabilidade do barramento e manter o seu sincronismo. O bit stuffing insere um bit de valor oposto a cada 5 bits consecutivos de mesmo valor, estes bits devem ser removidos do quadro pelo receptor. O bit stuffing acontece do primeiro bit (SOF) até o final do CRC, evitando assim que os bits de valor fixo (CRCD, ACKD e EOF) e o bit de ACK sejam sobrescritos, todos estes componentes fazem com que o protocolo CAN funcione de forma confiável. Na figura 8 é demonstrado o funcionamento do bit stuffing.

Figura 8 – Funcionamento do Bit Stuffing.



Fonte: https://en.wikipedia.org/wiki/CAN_bus.

3.2 TIPOS DE QUADRO DO PROTOCOLO CAN

O protocolo CAN possui quatro tipos diferentes de quadros, quadro de dados, quadro remoto, quadro de erro e quadro de sobrecarga.

O quadro de dados é o mais comum na comunicação e é o responsável por grande parte da transferência de dados no barramento, seguido pelo quadro remoto, estes quadros seguem os modelos demonstrados na figura 7. A principal diferença entre estes dois quadros é que no quadro remoto os bits de identificação são compostos pelo identificador do nó que deve enviar as informações solicitadas e não a identificação do nó que está utilizando o barramento, no quadro remoto também não vai nenhuma informação nos bytes de dados (TEXAS, 2016).

O quadro de erro sinaliza que algum nó não recebeu a informação de forma correta, este quadro não segue o padrão dos quadros de dados e remoto. Um quadro de erro é composto por 6 bits consecutivos de mesmo valor e “quebra” a regra do *Bit Stuffing*. Quando um nó envia um quadro de erro todos os demais nós também sinalizam o erro e o nó que estava enviando as informações deve reenviar a mensagem.

O quadro de sobrecarga é similar ao quadro erro e é enviado por um nó para sinalizar que ele não está conseguindo armazenar as informações a tempo, assim o nó transmissor aguarda por mais um tempo antes de enviar a próxima mensagem.

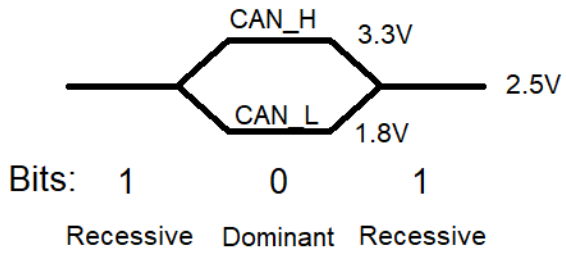
3.3 BARRAMENTO

Nesta seção serão demonstradas algumas características do barramento CAN que tornam o protocolo mais confiável.

3.3.1 Transmissão diferencial

No protocolo CAN os bits de nível lógico alto (um) são chamados de recessivos e bits com nível lógico baixo (zero) são chamados dominantes. No barramento CAN a transferência de dados ocorre de modo diferencial por meio de dois fios V_{CAN_H} e V_{CAN_L} e para transmissão de um bit recessivo tanto o V_{CAN_H} quanto o V_{CAN_L} ficam com a mesma tensão (aproximadamente 2,5V), enquanto para transmitir um bit dominante a tensão do V_{CAN_H} sobe para aproximadamente 3,3V e a tensão do V_{CAN_L} baixa para aproximadamente 1,8V, criando uma diferença de potencial de aproximadamente 1,5V. Desta forma, um bit dominante consegue sobrescrever um bit recessivo. Na figura 9 é demonstrado o funcionamento da transmissão diferencial no barramento CAN.

Figura 9 – Modo de transferência de dados no protocolo CAN.

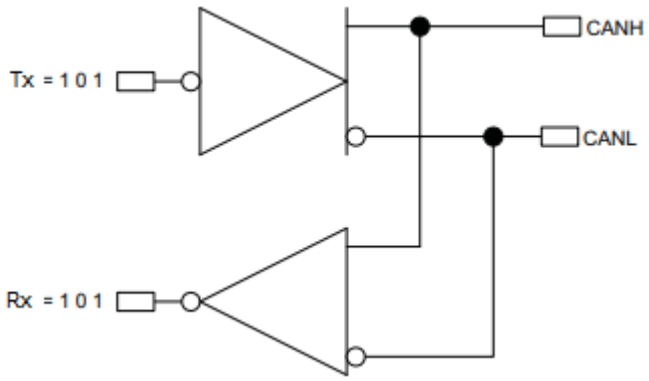


Fonte: Próprio autor.

3.3.2 Arbitragem

O acesso ao meio no protocolo é uma variação do CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*), por isso é considerado multi-master, pois todos os nós têm as mesmas condições para iniciar sua transmissão tão logo o barramento esteja livre, sendo definido quem deve transmitir apenas pela sua prioridade. Todos os nós leem o barramento durante qualquer transmissão, isto permite que um nó detecte se outro nó com prioridade mais alta está transmitindo. Uma representação do funcionamento da escrita e leitura do barramento é demonstrada na figura 10.

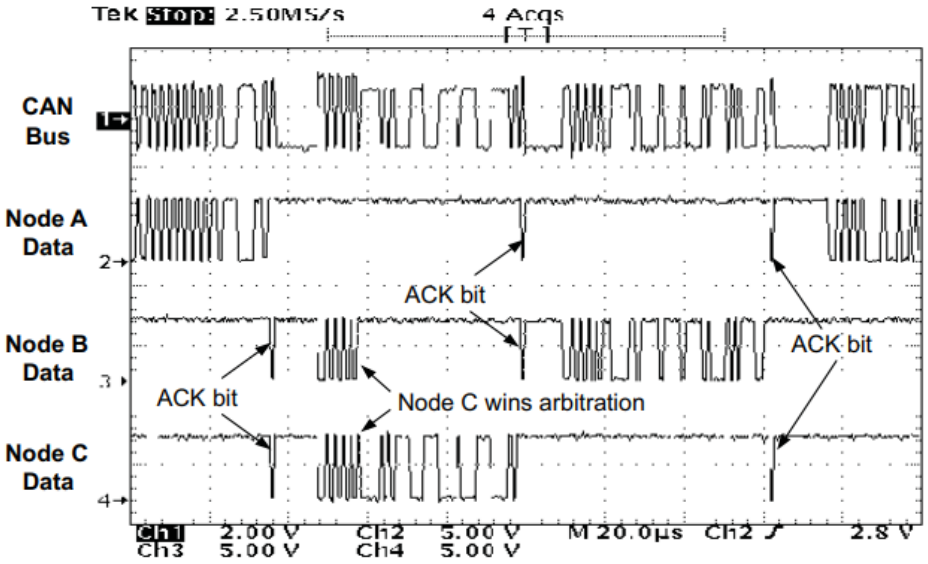
Figura 10 – Escrita e leitura simultânea no barramento CAN.



Fonte: Texas Instruments, 2002.

A prioridade de uma mensagem é definida na parte de identificação do quadro, sendo assim, um nó com maior prioridade é o nó com menor número binário. Devido ao modelo de transmissão utilizado no protocolo CAN não há perda de informação quando dois nós estão transmitindo e disputando o barramento ao mesmo tempo. No momento que um dos nós apresentar um bit dominante (zero), enquanto outro apresentar um bit recessivo (um), o bit dominante sobrescreverá o bit recessivo, assim o nó de menor prioridade lerá um bit divergente do que ele enviou e irá parar a sua transmissão, permitindo que o nó de maior prioridade continue sua transmissão sem perder seus dados. Uma demonstração da arbitragem do barramento pode ser vista na figura 11.

Figura 11 – Arbitragem do barramento CAN.



Fonte: Texas Instruments, 2002.

3.4 DETECÇÃO DE ERRO

O protocolo CAN possui uma alta confiabilidade, principalmente, devido a sua capacidade de verificação de erro, por meio do uso de bits com formatos fixos (SOF, CRCD e ACKD), bit stuffing, cálculo de CRC e envio de Acknowledgement. A capacidade que todos os nós têm de ler o barramento ao mesmo tempo que enviam suas mensagens, tornam a detecção de erros uma das principais características do protocolo e fazem com que o CAN seja um protocolo bastante confiável. Existem 5 tipos diferentes de erros que podem ser detectados no CAN, sendo dois classificados como erros de bit e três classificados como erro de mensagem (TEXAS, 2016).

Os erros de mensagem classificados em:

- Erro de CRC – O erro de CRC acontece quando o cálculo do CRC feito pelo transmissor não confere com o cálculo feito pelo receptor
- Erro de acknowledgment – O erro de acknowledgment ocorre quando o transmissor não recebe um bit dominante no ACK.
- Erro de formato – O erro de formato acontece quando um bit que tem formato fixo tem um valor diferente do que deveria (Ex. SOF, CRCD, ACKD e EOF).

Os erros de bit são:

- Erro de bit – O erro de bit acontece quando o transmissor lê no barramento um bit diferente do que ele enviou (exceto durante a arbitragem do barramento).
- Erro de Stuff – Este erro ocorre quando mais de cinco bits do mesmo valor são lidos no barramento.

Ao detectar algum dos erros mencionados, os nós que detectaram o erro enviam um quadro de erro. Os tipos de quadros de erro no protocolo CAN podem ser definidos como ativo ou passivo e o tipo de quadro de erro que será enviado é definido pelo sistema de contagens de erro do protocolo. Controladores CAN possuem dois contadores de erro, Contador de Erro de Transmissão (*Transmission Error Counter*, TEC) e Contador de Erro de Recepção (*Reception Error Counter*, REC). Enquanto ambos os contadores estão com suas contagens abaixo de 127 erros o controlador funciona em modo erro ativo, isso significa que no caso de detectar algum erro ele enviará um quadro de erro com 6 bits **dominantes** consecutivos, já se o TEC ou o REC

estiverem acima de 128 (desde que TEC menor que 255), ele funciona em modo passivo, assim o quadro de erro será composto por 6 bits **recessivos** consecutivos e no caso de TEC maior de 255 o controlador entre em modo *bus off,* neste modo o nó é desconectado do barramento e fica impossibilitado de enviar ou receber informações do barramento, devendo ser reiniciado por software ou hardware. (GAUJAL; NAVET, 2005). No quadro 2 é apresentado um resumo dos 3 estados de erro do protocolo CAN.

Quadro 2 – Tipos de erro do protocolo CAN.

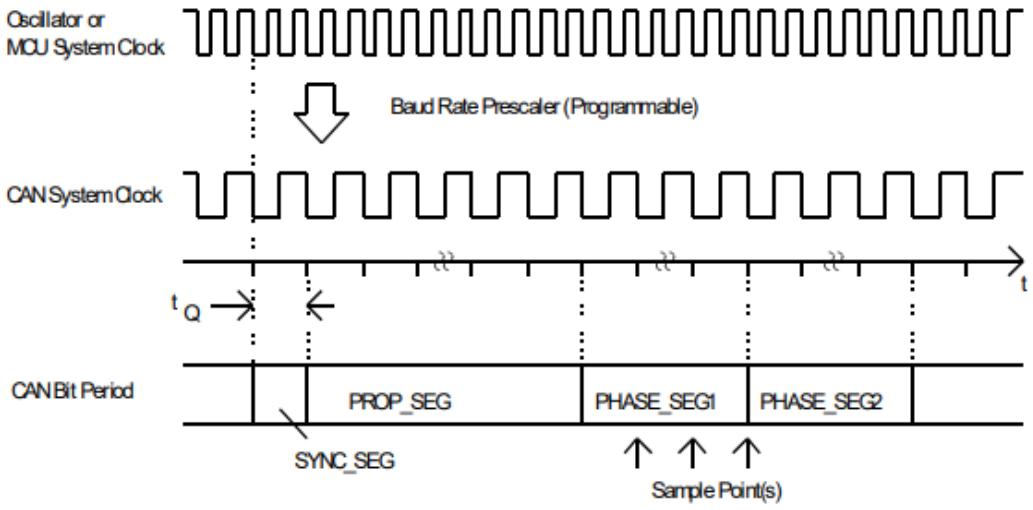
Erro Ativo	REC e TEC menores que 127
Erro Passivo	REC maior que 127 ou TEC entre 127 e 255
Bus off	TEC maior que 255

Fonte: Próprio Autor

3.5 BIT TIMING

O tempo necessário para enviar um bit no protocolo CAN é definido como Bit Timing e é separado em quatro segmentos SYNC_SEG, PROP_SEG, PHASE_SEG1 e PHASE_SEG2, cada um destes segmentos é derivado de um tempo base, denominado Time Quantum (t_Q) que é o mesmo que um ciclo de Clock gerado pelo controlador CAN (Clock_CAN), o Clock do CAN é obtido por meio do Clock fundamental (oscilador ou Clock do sistema), como pode ser visto na figura 12.

Figura 12 – Descrição do tempo de um bit no protocolo CAN.



Fonte: Texas Instruments, 2002.

O número de Time Quantum necessário para cada segmento é variável e deve ser calculado com base no comprimento da rede e a velocidade da transmissão dos dados, com exceção do SYNC_SEG que é fixo em um Time Quantum. Os valores de Time Quantum para PROP_SEG, PHASE_SEG1 e PHASE_SEG2 variam de 1 a 8 Time Quantum e o ponto de amostragem ocorre entre PHASE_SEG1 e PHASE_SEG2. (Freescale Semiconductors, 2004).

- SYNC_SEG: O segmento de sincronização inicia-se no momento da transição de um nível lógico para outro e é o responsável por sincronizar os nós do barramento com relação ao nó transmissor.
- PROP_SEG: O segmento de propagação deve ser maior que duas vezes o tempo de propagação do barramento e existe para atrasar a amostragem do bit. Assim, caso mais de um nó esteja transmitindo no momento da arbitragem do barramento, ambos os nós podem ler as informações recebidas dentro do mesmo período o qual estão transmitindo, evitando erros.
- PHASE_SEG1 e PHASE_SEG2: Os segmentos de fase são responsáveis por corrigir qualquer erro de fase que possa ocorrer durante a transferência de informações no barramento.

4 TRABALHOS RELACIONADOS E ESTADO DA ARTE

Este capítulo detalhará a pesquisa realizada com o objetivo de validar a importância do trabalho proposto, avaliando o estado da arte sobre a utilização de CAN em CubeSats. Para isto, foram analisadas produções bibliográficas as quais demonstrassem a necessidade de protocolos de comunicação mais confiáveis e apresentassem o que está sendo desenvolvido com relação à utilização do protocolo CAN. A pesquisa tem seu foco na aplicação do protocolo na comunicação entre os módulos dos sistemas de controle da plataforma FloripaSat e para a missão de lançamento do GOMX-5, em 2022 (missão explicada com maiores detalhes no apêndice A).

As características do protocolo CAN, como sua velocidade de transmissão de dados de até 1Mbit/s, sua capacidade de gerenciamento de colisão e definição de prioridade de forma não destrutiva, sua imunidade à interferência eletromagnética, detecção e sinalização de erros, entre outros, fazem com que este protocolo seja utilizado, amplamente, em diversas áreas além da automobilística, como aviação, automação, robótica e também na indústria de satélites.

Em Reges e Santos (2008) foi demonstrada a criação da subcamada de controle de acesso ao meio (MAC, *Medium Access Control,* em inglês) em uma FPGA. O sistema foi dividido em quatro partes, uma responsável pelo envio das informações para a rede (CAN TX), uma responsável pelo recebimento das informações da rede (CAN RX), uma responsável pelo cálculo do CRC, que realiza a detecção de erros nos bits de dados e outra responsável pela verificação da mensagem, para informar a existência de stuff bits. Nos testes realizados e demonstrados pelos autores, foi verificado o funcionamento correto do protocolo seguindo o modelo CAN, também foi feita uma comparação da FPGA utilizada pelos autores com as características da HurriCANe da ESA (*European Space Agency*), onde foi possível verificar que o módulo de recepção desenvolvido pelos autores utiliza menos recursos da FPGA.

Com foco no desenvolvimento de um sistema para converter dados no formato serial do padrão RS-232 para o protocolo CAN, Jayarathne e Jayananda (2014) desenvolveram um *buffer* de armazenamento e envio (*store and forward buffer)* que recebe as informações advindas do padrão RS-232, realiza o empacotamento das informações no formato CAN e envia para o barramento. Para este projeto, devido a diferença de velocidade dos protocolos, foram utilizados pulsos de *clock* diferentes para cada um dos sistemas, sendo assim, após o recebimento das informações vindas pela RS-232 ele deve sincronizar o *clock* para realizar o envio das informações para o barramento CAN. Porém, como este trabalho não tinha a intenção de desenvolver todo o protocolo CAN não foram implementadas funções de recebimento de dados

para o CAN, então, não é possível realizar verificações de erro e nem de sobrecarga no barramento, funções que são intrínsecas do protocolo.

Para aplicação em satélites o protocolo CAN vem sendo utilizado faz alguns anos, uma demonstração da sua utilização nesta área pode ser vista no trabalho de Khurram e Zaidi (2005), onde o protocolo CAN foi utilizado em um satélite de órbita terrestre baixa e foi implementado na telemetria, telecomando e intercomunicação do satélite, sendo que no barramento CAN estão conectados quatro nós.

No projeto, são utilizados dois barramentos CAN para haver uma redundância nas informações e dar maior robustez ao projeto. Neste caso, foram considerados dois modelos, em um dos modelos os nós são conectados ao barramento por meio de um relé e no caso do nó primário ficar mais de cinco minutos sem receber informações de telemetria o sistema considera que houve uma falha no barramento e a comunicação passa a ser realizada pelo barramento secundário. No segundo modelo as informações são enviadas nos dois barramentos ao mesmo tempo, fazendo com que o nó tenha a capacidade de verificar se a informação em ambos é igual. Como os autores observaram, as missões espaciais que utilizam componentes COTS vem sendo bem-sucedidas, tornando o protocolo CAN uma alternativa extremamente interessante para estas aplicações.

Os autores Janschek e Braune (2000), realizaram um estudo para validar a utilização do protocolo CAN em satélites LEO em comparação com os protocolos Profibus DP e LON (Local Operating Network). Neste trabalho, eles tem como objetivo utilizar a rede em uma missão de telecomunicação e observação da terra, consequentemente a rede de comunicação do satélite deve ser confiável e robusta para lidar com dados complexos do rádio e o gerenciamento dos dados da câmera de observação da Terra.

Dentre os aspectos que foram avaliados na escolha do protocolo CAN para esta aplicação, estavam sua padronização e disponibilidade, sua velocidade, sua forma de acesso ao barramento (multi-master), detecção e gerenciamento de erro, entre outros. Também foi avaliado o comportamento do barramento em tempo real e sua robustez em aplicações espaciais, o que levou os autores a concluírem que o protocolo CAN pode ser uma ótima rede para ser utilizada em satélites LEO.

Blanco, D'errico e Conticchio (2006) realizaram um estudo para avaliar qual o melhor protocolo para utilização no controle e gerenciamento de baterias Li-Íon em satélites LEO, realizando uma comparação mais aprofundada entre os protocolos de comunicação demonstrados na tabela 2, onde é possível visualizar as diferenças mais relevantes entre os modelos.

Tabela 2 – Comparativo entre diferentes protocolos de comunicação.

Protocolo	**CAN**	**FIP**	**INTERBUS**	**PROFIBUS**	**SERCOS**	**MIL-1553**
Velocidade (Kb/s)	1000	2500	500	500	2000	1000
Nº de nós / repetidores	30 / 2048	64 / 256	8 / 256	32 / 127	254 / --	31 / --
Nº mestres	30	1	1	127	1	1 por ciclo
Broadcast	Sim	Sim	Não	Apenas mestre	Apenas mestre	Apenas mestre
Detecção de erro	Sim	Não	Não	Não	Não	Não

Fonte: Adaptado de Blanco, D'Errico e Conticchio (2006).

Levando em conta as características demonstradas na tabela 2 verifica-se que o protocolo CAN por mais que não seja o mais veloz, possui algumas vantagens, sendo elas: broadcast, múltiplos mestres e detecção de erros, o que o torna o protocolo preferido para a aplicação.

Uma aplicação mais completa, onde foram desenvolvidas tanto a camada de enlace quanto a camada física, para utilização em ambientes com radiação pode ser vista no trabalho de Wielandt et al. (2012). Para isto, os autores desenvolveram um sistema a partir de um componente RadHard (tolerante à radiação) utilizado para comunicação RS-485, adaptando-o para que ele funcionasse como transceiver CAN RadHard, garantindo a sua aplicabilidade em órbita baixa. Para o controlador CAN foi utilizado uma FPGA RadHard e o projeto baseou-se em alguns IP cores, sendo escolhido para o projeto o GRCAN IP Core da Gaisler, que foi escolhido pelos autores por ter o protocolo AMBA (*advanced microcontroller bus architecture),* possuir uma interface AHB e acesso direto à memória.

No trabalho, foram realizados diversos testes para validar o funcionamento da FPGA e a robustez e confiabilidade do hardware, desta forma foi possível verificar o funcionamento correto da leitura e escrita dos registradores do *CAN IP Core* e da recepção de dados. Finalmente, foram realizados testes de hardware, divididos em três categorias, o primeiro para verificar o comportamento funcional do controlador CAN, o segundo onde são injetados erros no barramento para verificar o comportamento do controlador durante situações anormais e por fim testes de performance. Todos os testes de hardware obtiveram sucesso.

A BRAVE (NX-Medium, NX-Large e NX-Ultra), nova tecnologia desenvolvida pela NanoXplore, é uma FPGA tolerante à radiação que, atualmente, vem sendo estudada para ser utilizada em aplicações espaciais. Alguns trabalhos como o realizado por Bravharl et. al. (2018)

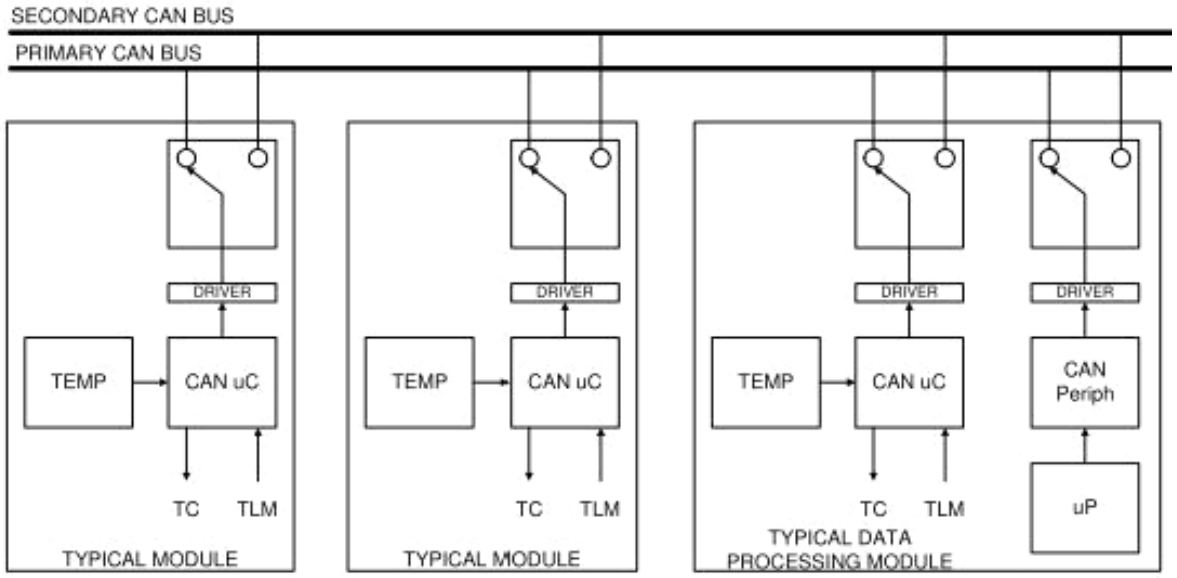
demonstram as vantagens desta nova tecnologia, como a utilização do protocolo SpaceWire, um protocolo que também é tolerante à radiação e é frequentemente utilizado em aplicações espaciais, pois reduz o tempo de reconfiguração e envio do bit stream.

Outro trabalho que corrobora com o a validação da FPGA BRAVE foi realizado por Maragos et. al. (2018), onde foram avaliadas as características da FPGA BRAVE e dos programas disponibilizados para a configuração do Hardware, o estudo levou em conta as etapas de Síntese, Roteamento e Geração do Bit Stream. Tais levantamentos foram realizados para benchmarks simples (circuitos combinacionais e sequenciais, operações aritméticas e buffers), médios (controladores CAN, controladores VGA, etc.) e benchmarks de alta performance (filtro de resposta ao impulso finita, etc.), sendo este último o mais relevante no trabalho publicado, demonstrando que a NG-Medium é uma FPGA eficiente para o desenvolvimento de circuitos de alta performance tanto na utilização dos recursos disponíveis quanto na taxa de transferência e consumo de energia.

Para validação da FPGA BRAVE no espaço, Gouveia Jr et. al. (2020) estão desenvolvendo a Payload-XL constituída de uma NX-Large e fará parte da missão GOMX-5 que tem seu lançamento previsto para 2022. Neste projeto, está sendo criado uma plataforma capaz de integrar os principais sistemas de um satélite, como o Computador de Bordo (OBC, On-Board Computer, em inglês), a Telemetria e Telecomando (TT&C) e o Sistema elétrico de Potência (EPS). O sistema também possui a capacidade de ser reconfigurado diversas vezes, com base no envio do Bit Stream pela estação terrestre.

Um estudo realizado pelos autores Woodroffe e Madle (2004) demonstra outro modelo para o uso do protocolo CAN em naves espaciais (CAN spacecraft usage – CAN-SU). No modelo desenvolvido existem dois barramentos (barramento principal e barramento secundário), assim, no caso de o barramento principal parar de funcionar por 5 minutos é considerado que houve uma falha e o secundário entra em funcionamento. Módulos de tratamento de dados são ligados tanto ao controlador CAN (responsável pela telemetria quando o processador principal está desligado) quanto ao CAN periférico (ligado ao processador principal), enquanto módulos de menor importância são ligados apenas ao controlador CAN da telemetria, como pode ser visto na figura 13.

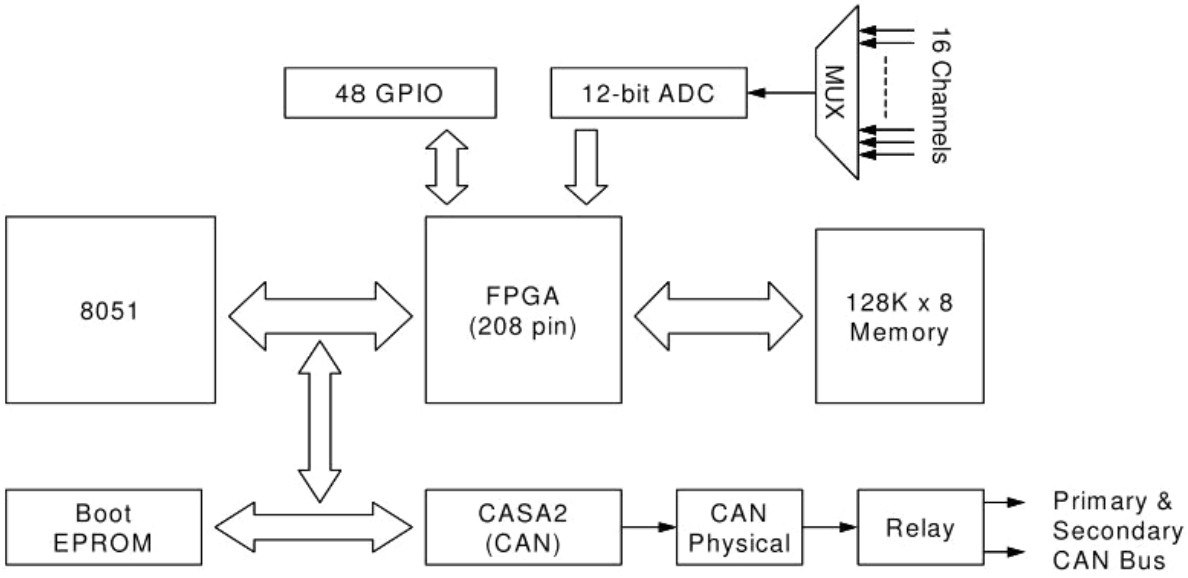
Figura 13 – Arquitetura do barramento CAN da SSTL.



Fonte: Woodroffe e Madle (2004).

No trabalho são demonstradas as modificações realizadas em cima do protocolo CAN, mas principalmente, é realizado um estudo sobre o uso de componentes COTS em missões anteriores e demonstrado um sistema criado para utilização do protocolo CAN com componentes RadHard, como o microcontrolador 8051, eprom, memória, ADC e FPGA, como mostra a figura 14.

Figura 14 – Modelo desenvolvido pela SSTL



Fonte: Woodroffe e Madle (2004).

Os autores, então, demonstram em um quadro comparativo, como pode ser visto no quadro 3, o impacto gerado para migração do projeto com componentes COTS para componentes RadHard.

Quadro 3 – Comparação entre CAN RadHard para COTS.

	COTS	RadHard
Tamanho	1 in²	>6 in²
Massa	<5g	>50g
Consumo energético	<0.75W(máx.), 0.20W(nom.)	Estimado 1W
Dose Total	<10Krads	100Krads
Proteção SEE	Não	Tolerante à SEU, imune à Latch-up
Custo dos componentes	<$10	>$15000
Restrições de Importação	Não	ITAR

Fonte: Adaptado de Woodroffe e Madle (2004).

Portanto, os autores concluem que devido à sua larga experiência na utilização de COTS, um sistema desenvolvido com tolerância à falha e módulos redundantes é uma opção viável para utilização em missões LEO, não apresentando nenhuma falha no barramento CAN ou em seus componentes.

Uma possibilidade demonstrada no trabalho de Gebelein, Gudge e Kebschull (2016) é utilizar um microcontrolador COTS com multiprocessadores executando em Lockstep, amplamente utilizados na indústria automotiva, para trabalhar em ambientes com radiação ionizante. O modelo estudado foi o Hércules TMS570, que além de possuir dois processadores em Lockstep, também possui Código de Correção de Erros (ECC, do inglês *Error Correction Code*), capaz de corrigir erros de bit único (SBU, do inglês *Single Bit Upset*). Neste trabalho, foi observado que todos os eventos de SBU injetados no MCU foram corrigidos com sucesso, além do microcontrolador apresentar um resultado satisfatório quando testado com impacto total da radiação a noventa graus.

Uma abordagem mais específica é vista em Leppinen et. al. (2014), onde o MCU TMS570 foi utilizado no desenvolvimento do OBDH de um nanossatélite, o qual foi comparado com outros OBC's tradicionais. Neste trabalho, também foi analisada a tolerância à falha devido ao Lockstep e comparada com outros métodos de tolerância a falha. Logo, a conclusão que os autores chegaram foi que o MCU Hercules tem desempenho intermediário entre MCU's

simples, baseado em ARM-7 e ARM-9 e que não possuem características de controle de erro necessitando de sistemas externos complexos para evitar falhas, e MCU's mais robustos para uso espacial baseado em processadores LEON, com alta tolerância a ambientes com radiação. Desta maneira, é possível dizer que o MCU Hercules é uma boa solução para missões de curta duração como no caso de nanossatélites, pois por tratar-se de um componente COTS, possui custo reduzido e um bom nível de tolerância à falha.

Conforme demonstrado neste capítulo e visto a crescente pesquisa nesta área, este trabalho busca desenvolver uma forma de comunicação baseada no protocolo CAN trazendo as características mais adequadas para utilização em nanossatélites, para isto, foi desenvolvida a tabela 3, onde o protocolo CAN foi comparado com os protocolos atualmente utilizados no FloripaSat-1.

Tabela 3 – Comparação entre o CAN e os protocolos atuais do FloripaSat.

Protocolo	**CAN**	**UART**	**SPI**	**I2C**
Velocidade (Kb/s)	1000	115,2	2000	400
Método	Assíncrono	Assíncrono	Síncrono	Síncrono
Nº de nós	30	--	--	1024
Nº de mestres	30	1	1	1
Broadcast	Sim	Sim	Sim	Não
Detecção de erro	Sim (vide seção 3.4)	Bit de paridade e Stop bit	Não	Não

Fonte: Próprio Autor

Com a experiência adquirida durante o desenvolvimento do FloripaSat-1, foi verificado que o protocolo I2C não é o protocolo ideal para esta comunicação devido sua necessidade de resposta na comunicação mestre-escravo, assim quando um escravo apresenta algum problema, todo o barramento pode ficar inutilizável. Neste sentido, o protocolo CAN apresenta-se como uma solução interessante, uma vez que não depende de uma resposta de seus escravos e estes podem ser adicionados e removidos do barramento sem grande necessidade de modificações. Características como acesso ao barramento livre para todos, prioridade não destrutiva, robustez a interferência eletromagnética, detecção e correção de erro, entre outras, tornam o protocolo CAN, uma opção vantajosa para utilização na plataforma FloripaSat, visando a melhoria da confiabilidade em missões futuras.

No quadro 4 foi realizada uma comparação entre as implementações realizadas nos trabalhos estudados, com a ideia inicial de implementação no FloripaSat. Informações que não foram encontradas na literatura foram completadas com "**".

Entre os componentes comparados estão o microcontrolador, o controlador CAN, o *transceiver* CAN, a versão do protocolo CAN utilizada, o modo de comunicação entre microcontrolador e controlador CAN e a capacidade de detectar e corrigir erros.

O microcontrolador é responsável por toda as atividades do sistema, bem como, o gerenciamento das mensagens. O controlador CAN, durante a transmissão das mensagens, é responsável por receber a mensagem do microcontrolador, codificar o quadro no formato CAN e enviar para o barramento, enquanto durante a recepção das mensagens o controlador CAN é responsável por realizar o filtro das mensagens, decodificá-las e enviá-las para o microcontrolador. O transceiver CAN é o componente responsável por receber as informações na forma digital (0V e 5V) e convertê-las para tensão diferencial e a versão do protocolo CAN define as características e especificações do protocolo, sendo a versão 2.0B a mais utilizada por ser compatível com quadros no formato padrão e estendido.

Quadro 4 – Comparação das implementações apresentadas nos trabalhos estudados.

	BLANCO, C. D. V.; D'ERRICO, M.; CONTICCHIO, A.	**JAYARATHNE, D. G. N.; JAYANANDA, M. K.**	**KHURRAM, M.; ZAIDI, S. M. Y.**	**REGES, J. E.; SANTOS, E. J. P.**	**WIELANDT, S. et al.**	**WOODROFFE, A, M.; MADLE, P.**
Microcontrolador(uC)	PIC 18F258	**	i386EX-EngineL	**	**	8051
Controlador CAN	Integrado ao uC	FPGA	MCP2515	FPGA (apenas camada MAC)	FPGA (Gaisler IP core)	FPGA
Transceiver CAN	MCP2551	**	MCP2551	**	Baseado no RS-485	Físico (COTS)
Versão CAN	2.0B	**	2.0B	**	2.0B	CAN-SU
Comunicação entre controlador CAN e uC	Interno	UART	UART	**	PCI bus	**
Detecção e correção de erro	Sim	Não	Sim	Sim	Sim	Sim

Fonte: Próprio Autor.

5 METODOLOGIA

Neste capítulo, será descrita a metodologia utilizada para avaliar a confiabilidade do barramento CAN, realizada por intermédio da análise matemática do pior tempo de resposta para envio das mensagens no barramento.

5.1 CONFIABILIDADE DO BARRAMENTO

A confiabilidade do barramento é um dos principais fatores que deve ser levado em conta no momento da tomada de decisão de qual protocolo utilizar em um satélite. Uma forma de avaliar a confiabilidade de um barramento é avaliando a sua capacidade de entregar as mensagens dentro do tempo determinado para aquela aplicação, desta maneira, a análise do pior tempo de resposta é um indicador essencial no momento da escolha do protocolo.

Nos trabalhos de Tindell e Burns (1994), Tindell, Hansson e Wellings (1994) e Tindell, Burns e Wellings (1995) foram demonstrados os modelos de análise temporal baseados em modelos de análise de tarefas de tempo real em processadores. Assim, o pior caso do tempo de resposta, denominado R_m, é definido como o tempo gasto desde a inserção da mensagem na fila até o recebimento completo da mensagem pelo receptor. O cálculo de R_m é demonstrado na equação 1.

$$R_m = J_m + w_m + C_m \tag{1}$$

Onde, J_m é o Jitter da mensagem, ou seja, a variação do tempo em que uma determinada mensagem pode levar para ser enfileirada, normalmente sendo o tempo de execução da tarefa de envio da mensagem para a fila. w_m, representa o pior caso de atraso de enfileiramento, referente ao tempo que uma mensagem fica aguardando na fila, enquanto o termo C_m é o tempo que um quadro leva para ser enviado no barramento.

Para o cálculo de C_m, deve ser levado em conta o tamanho do quadro, considerando o seu pior caso, consequentemente o cálculo leva em conta o pior caso para Bit Stuffing, como demonstrado na equação 2, modificada em Davis et. al. (2007) para abranger situações onde o próprio stuff bit é contabilizado para adição do stuff bit seguinte, sendo necessários apenas 4 bits de informação de mesmo valor consecutivo.

$$C_m = 8Nb + Sb + 13 + \left\lfloor \frac{Sb + 8Nb - 1}{4} \right\rfloor \tau_{bit} \quad (2)$$

Onde, Nb é o número de bytes de dados que serão enviados no quadro, Sb é o número de bits onde pode ocorrer bit stuffing (sendo 34 para o formato padrão, com 11 bits de identificação e 54 para o formato estendido, com 29 bits de identificação) e 13 é o número de bits que não ocorre bit stuffing. τ_{bit} é o tempo necessário para que um bit seja transmitido no barramento.

O delay da fila é dividido em duas partes, tempo Bloqueado (B_m) e Interferência (I_m). Tempo Bloqueado é o maior tempo em que a mensagem pode ficar esperando uma mensagem de menor prioridade terminar seu envio e a Interferência é o maior tempo que uma mensagem tem que aguardar devido mensagens de maior prioridade entrarem na fila e enviarem seus dados. Matematicamente B_m pode ser representado como:

$$B_m = Max\ (C_k) \quad \forall k \in lp(m) \quad (3)$$

Onde $lp(m)$ é o conjunto de todas as mensagens de prioridade mais baixa que m, assim a mensagem que tiver o maior tempo de envio entre as mensagens de menor prioridade que m, será considerada o pior caso para o tempo de bloqueio. Para o cálculo de Interferência é definida a equação 4.

$$I_m = \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m + J_k + \tau_{bit}}{T_k} \right\rceil C_k \quad (4)$$

Onde $hp(m)$ é o conjunto de todas as mensagens de prioridade mais alta que m e T_k é o período da mensagem de prioridade k. Consequentemente, a equação de w_m é representada como:

$$w_m = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m + J_k + \tau_{bit}}{T_k} \right\rceil C_k \quad (5)$$

Como a equação 5 não pode ser modificada de modo que seja possível deixar o w_m em evidência, é necessário realizar o cálculo de forma recursiva, como demonstrado na equação 6.

$$w_m^{n+1} = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \quad (6)$$

Para a primeira iteração w_m^0, um valor apropriado é o próprio tempo de Bloqueio (B_m). As iterações devem ocorrer até $w_m^{n+1} = w_m^n$ ou até $J_m + w_m^{n+1} + C_m > Dm$, sendo Dm o Deadline da mensagem e neste caso a mensagem não cumpre os requisitos temporais e não é possível inserir a mensagem na fila.

Assim sendo, alguns requisitos são indispensáveis para que os cálculos demonstrados acima, possam ser utilizados de forma correta, sendo alguns deles:

- As mensagens devem ser periódicas ou esporádicas com um período mínimo entre suas ocorrências e com período conhecido (T_m).
- As mensagens só poderão ser inseridas na fila se o pior tempo de resposta for inferior ao deadline ($R_m \leq D_m$).
- O pior tempo de resposta deve ser inferior ao período da mensagem menos o *jitter* ($R_m \leq T_m - J_m$).

5.1.1 Análise de probabilidade de erro

Um problema que deve ser levado em conta quando falamos de análise do barramento CAN é a possibilidade de algum quadro ser corrompido, assim gerando um overhead no sistema pela necessidade de retransmissão do quadro.

Para uma análise mais precisa e menos pessimista sobre o barramento CAN, os autores Broster Burns e Rodriguez-navas (2002) fizeram uma análise do pior caso de tempo de resposta para o caso de ocorrer um erro no intervalo de tempo em que uma mensagem está na fila aguardando para ser enviada, incrementando o tempo de enfileiramento da mensagem.

Para estes cálculos a equação 1 foi modificada para compreender os casos em que o tempo de espera da mensagem na fila seja alterado devido a erros no barramento.

$$R_m = J_m + t_m \quad (7)$$

$$t_m = B_m + C_m + I_m(t_m) + E_m(t_m) \quad (7.1)$$

$$t_m^{n-1} = B_m + C_m + I_m(t_m^n) + E_m(t_m^n) \quad (7.2)$$

Com este novo arranjo, o cálculo do tempo sobre Interferência sofre uma pequena alteração, uma vez que para a interferência não é levado em conta o tempo de envio da mensagem (C_m), tornando o cálculo de I_m como demonstrado na equação 8.

$$I_m = \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m - C_m + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{8}$$

O Cálculo de $E_m(t)$ é feito considerando o pior caso, ou seja, caso o erro aconteça ao final do envio do maior quadro de prioridade mais alta que m e o quadro de erro enviado para sinalizar a falha é o maior possível ($E_{max} = 31$bits). Assim, o pior caso de erro é dado pela equação 8.

$$M_m = E_{max}.\tau_{bit} + Max\ (C_k) \quad \forall k \in hep(m) \tag{9}$$

Onde, $hep(m)$ é o conjunto de todas as mensagens de prioridade maior ou igual m, uma vez que o erro pode acontecer desde as mensagens de maior prioridade até a o final da transmissão da própria mensagem de prioridade m. Com isto, E_m é calculado com base na quantia k de erros que podem acontecer em determinado tempo. A probabilidade de ocorrência de k erros em um tempo t, denominada P(k,t) é dada pela distribuição de Poisson $F \sim Po(\lambda)$, demonstrada na equação 10.

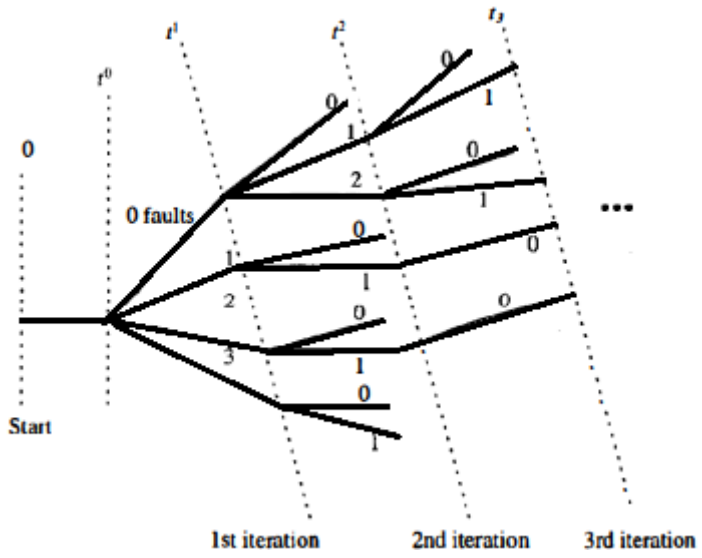
$$P\ (k,t) = \frac{e^{-\lambda t}(\lambda t)^m}{m!} \tag{10}$$

Assim a função do overhead, $E_m(t)$, é dada por uma distribuição aleatória:

$$E_m(t) = k.M_m\ com\ probabilidade\ P(F = m) \tag{11}$$

Resolvendo a equação 7.2 iterativamente é gerada uma árvore de probabilidade, como demonstrado na figura 15.

Figura 15 – Árvore de probabilidades.



Fonte: Broster Burns e Rodriguez-navas., 2002.

Porém este modelo pode apresentar diversos problemas, como o tamanho da árvore que pode tornar-se extremamente complexa com o crescimento dos "ramos", bem como, erros por "podar" ramos.

Em Broster Burns e Rodriguez-navas (2005) estes problemas foram corrigidos. Para uma melhor compreensão pode-se reescrever a equação 7.1 em relação ao pior caso de tempo de resposta para k erros:

$$t_{m|k} = B_m + C_m + I_m\left(t_{m|k}\right) + E_m(t_{m|k}) \tag{12}$$

Assim, para cada pior caso para o tempo de resposta ($R_{m|k}$) são calculadas suas probabilidades, como demonstrados a seguir:

- Probabilidade de $R_{m|0}$

Uma vez que para acontecer o tempo de resposta $R_{m|0}$ obrigatoriamente não podem ocorrer erros no barramento, a probabilidade de acontecer o tempo de resposta para zero erros é a mesma probabilidade de ocorrer zero erros.

$$P\left(R_{m|0}\right) = P\left(0, R_{m|0}\right) \tag{13}$$

- Probabilidade de $R_{m|1}$

A probabilidade de acontecer o tempo de resposta $R_{m|1}$ é calculada como a probabilidade de acontecer um erro durante o tempo $R_{m|1}(P\ (1, R_{m|1}))$ menos a probabilidade deste erro acontecer após o tempo $R_{m|0}$ (pois não havendo erros até $R_{m|0}$, o envio estaria completo neste período), assim sendo a probabilidade $P\ (R_{m|1})$ é dada por:

$$P\ (R_{m|1}) = P\ (1, R_{m|1}) - \ P\ (R_{m|0}) P(0, R_{m|1} - R_{m|0}) \tag{14}$$

- Probabilidade de $R_{m|2}$

A probabilidade de acontecer o tempo de resposta $R_{m|2}$ é calculada como a probabilidade de acontecer um erro durante o tempo $R_{m|2}(P\ (2, R_{m|2}))$ menos a probabilidade não ocorrer erros até o tempo $R_{m|2}$ e a probabilidade de acontecer apenas um erro até o tempo $R_{m|1}$, assim sendo a probabilidade $P\ (R_{m|2})$ é dada por:

$$\begin{aligned} P\ (R_{m|2}) = P\ (2, R_{m|2}) - \ P\ (R_{m|1}) P(1, R_{m|2} - R_{m|1}) \\ - \ P\ (R_{m|0}) P(2, R_{m|2} - R_{m|0}) \end{aligned} \tag{15}$$

Então, o cálculo da probabilidade de k erros durante um tempo $R_{m|k}$ pode escrito como demonstrado na equação 16.

$$P\ (R_{m|k}) = P\ (k, R_{m|k}) - \sum_{j=0}^{k-1} P\ (R_{m|j})\ P\ (k - j, R_{m|k} - R_{m|j}) \tag{16}$$

5.2 ENVIO DE MENSAGENS

O envio das mensagens de forma rápida e segura é a principal motivação deste trabalho, sendo assim, as características como quantidade de bytes de uma mensagem, periodicidade da mensagem, pior caso de tempo de resposta, são fundamentais nesta análise. Nesta seção serão demonstradas características que foram definidas para o envio das mensagens em tempo hábil com o menor custo para o barramento.

As mensagens que devem ser transmitidas no barramento, em alguns casos, são maiores que os 8 bytes de informação que cada quadro CAN pode carregar, deste modo, uma solução é dividir uma mensagem em diversos quadros, porém esta solução necessita de um modelo de organização dos quadros para que os mesmos possam ser reorganizados ao chegarem no destino.

Para evitar perder um byte de dados e para realizar o gerenciamento e organização dos quadros de uma mensagem com mais de 64 bits (8 bytes), uma solução utilizada foi fazer o gerenciamento direto no campo de arbitragem, sendo assim os 11 bits de identificação podem ser separados em dois grupos, os bits de identificação da origem da mensagem (Idm) e os bits de identificação do quadro (Idq). Deste modo, a arbitragem é realizada durante o envio de Idm e os bits Idq servem apenas para que o receptor consiga reorganizar os quadros. Para que o sistema funcione, é necessário que tanto o número de mensagens de origens diferentes quanto o número de quadros da maior mensagem possam ser representados dentro dos 11 bits de identificação. Sendo assim, foi desenvolvida a equação 17 para esta verificação.

$$\lceil \log_2 Msg \rceil + \lceil \log_2 Q \rceil \leq 11 \tag{17}$$

Uma alternativa sugerida para enviar as mensagens que necessitam mais de um quadro, foi a possibilidade de modificar a prioridade de uma mensagem após enviar seu primeiro quadro para que não sofra interferência até a conclusão de seu envio. No trabalho de Bartolini, Lipari e Almeida (2007) uma proposta similar é apresentada, onde o cálculo para pior caso para uma mensagem dividida em diversos quadros é feito utilizando como tempo de envio da mensagem ($C_{m_{msg}}$), o somatório do tempo de envio de todos os quadros (C_{m_q}) da mensagem.

$$C_{m_{msg}} = \sum C_{m_q} \tag{18}$$

A solução proposta foi definir o primeiro bit de identificação como indicador de fragmentação, ou seja, o primeiro quadro disputa o barramento de modo igual aos demais, porém após ganhar o barramento os quadros seguintes (da mesma mensagem) configuram o primeiro bit de identificação para ter uma prioridade mais alta, assim sendo, não é possível interromper a mensagem durante seu envio. Um exemplo com 3 mensagens é demonstrado no quadro 5.

Quadro 5 – Modelo proposto para envio das mensagens em burst.

	Identificação (Binário)			Identificação (Hexadecimal)
	Prioridade	Idm	Idq	
Msg1.1	1	0001	000000	0x440
Msg2.1	1	0010	000000	0x480
Msg2.2	0	0010	000001	0x81
Msg2.3	0	0010	000010	0x82
Msg3.1	1	0011	000000	0x4C0

Fonte: Próprio Autor.

Como demonstrado no quadro 5 após o primeiro quadro da mensagem 2 disputar o barramento com as demais mensagens, ele altera seus valores do bit de prioridade, impossibilitando a interrupção do envio de seus quadros até finalizar a comunicação. Esta possibilidade vai ser testada e comparada com o envio simples no estudo de caso para verificar qual o melhor modo para o FloripaSat.

6 REFORMULAÇÃO DAS INTERCONEXÕES PARA SISTEMAS BASEADOS EM FPGAS E EM MICROCONTROLADORES

Neste capítulo, serão demonstradas as alterações realizadas em dois sistemas, um baseado em FPGA e outro baseado em microcontrolador. Para o sistema baseado em FPGA foi utilizada a missão GOMX-5 como ponto de partida para a análise, enquanto para o sistema baseado em microcontrolador a análise foi realizada com base na plataforma FloripaSat e com as informações obtidas durante a missão FloripaSat-1.

6.1 BARRAMENTO CAN PARA FPGA (PAYLOAD-XL)

Para justificar o estudo da aplicabilidade do barramento CAN em FPGA, este trabalho buscou a implementação do sistema baseando-se em sistemas reais e que apresentassem um motivo significativo para tal. Assim, para implementação do barramento CAN em FPGA foi utilizada a missão GOMX-5 como ponto de partida, uma vez que, o SpaceLab será responsável por projetar uma das cargas úteis da missão em um consórcio com 3 empresas europeias. A carga útil desenvolvida pelo SpaceLab (Payload-XL) tem como objetivo validar, em órbita, as novas FPGAs tolerantes à radiação da empresa NanoXplore.

Com base na Payload-XL, a nova proposta para este sistema busca simplificá-lo inserindo o controlador CAN na FPGA por meio da portabilidade de um IPCore de código aberto, desenvolvido para outras tecnologias, para funcionar na NX-Large. Com isso, a Payload-XL poderá se conectar diretamente com o computador de bordo do satélite GOMX-5. O IPCore utilizado foi o CAN_OC (IP Core de código aberto), criado pela Cobham Gaisler, que por sua vez utiliza um IP Core CAN da OpenCores.

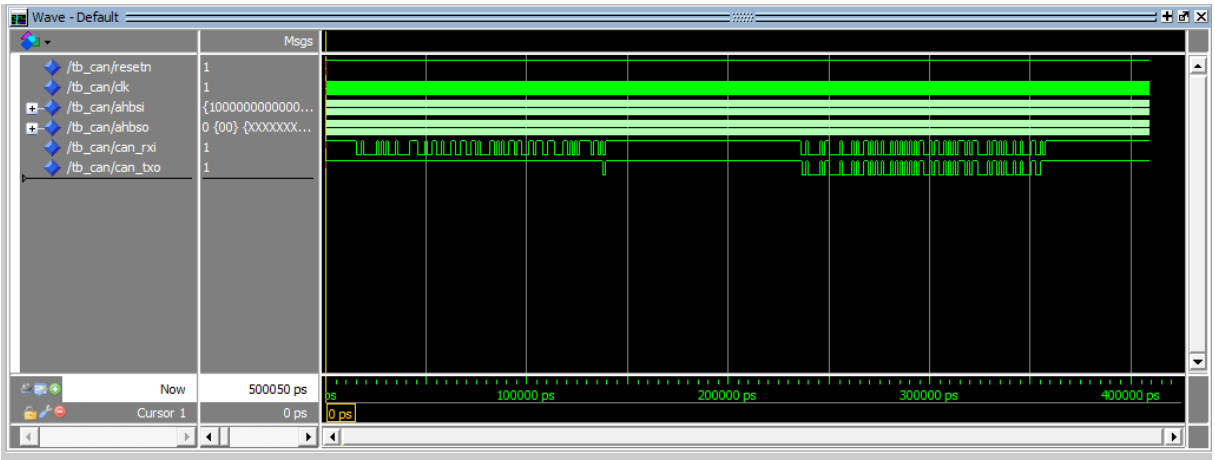
Maiores detalhes sobre a missão GOMX-5, a Payload-XL desenvolvida no SpaceLab e o IP Core CAN utilizado podem ser encontrados no Apêndice A.

6.1.1 Síntese e Simulação

Inicialmente, para realizar a síntese e validar o IP Core do controlador CAN, foi utilizado o programa Quartus, bem como, a memória da altera a qual está entre as tecnologias aceitas pelo IP Core. Foi utilizado, inicialmente, o Quartus devido à maior familiaridade do autor com o programa. Após alguns ajustes que foram necessários para a utilização do controlador CAN,

foram iniciadas as simulações no programa ModelSim para validar o comportamento do controlador CAN. Na figura16 é possível ver o funcionamento do controlador tanto no envio quanto no recebimento das mensagens CAN.

Figura 16 – Simulação de envio e recebimento de mensagens no ModelSim.



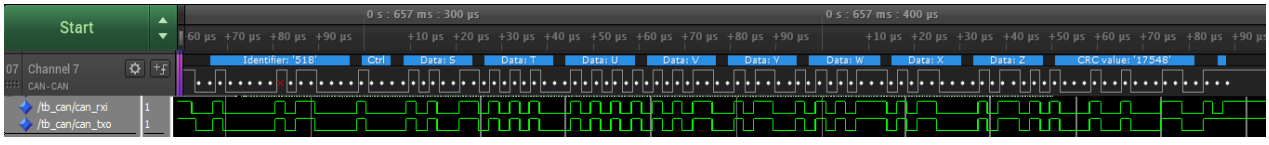
Fonte: Próprio Autor.

Para esta simulação os valores das mensagens recebidas (can_rxi) foram inseridas de acordo com as mensagens obtidas com o analisador lógico utilizado no estudo de caso do FloripaSat-1 e as informações a serem enviadas (can_txo) também foram baseadas nos dados transmitidos na missão FloripaSat-1.

Com a simulação da transferência de dados foi possível observar o funcionamento correto no envio e recebimento das mensagens, assim na figura 16 é possível ver que durante o recebimento da mensagem pelo can_rxi, o can_txo envia o bit de “ack” o qual também é lido pelo can_rxi. Durante o envio da mensagem, o can_txo envia toda a sua mensagem com exceção do bit de “ack”, que deve ser enviado pelos outros nós do barramento para sinalizar o recebimento correto da mensagem, esta informação é recebida pelo can_rxi.

Na figura 17 as informações enviadas pelo can_txo foram comparadas com as informações obtidas no barramento criado para validação da simulação e apresentado na seção 6.3.3, para verificar se estavam corretos em relação ao envio e sincronismo das mensagens.

Figura 17 – Comparação entre a simulação e o resultado esperado.



Fonte: Próprio Autor.

Para testar o funcionamento da memória FIFO, baseada na RAM da altera, foi simulado o recebimento de duas mensagens consecutivas e em seguida foram solicitados os dados da memória. Com o funcionamento correto de todo o controlador CAN, foi iniciado o porte para BRAVE.

Para realizar o porte para a BRAVE foi utilizado o programa NXmap da NanoXplore, onde foi feita a síntese do VHDL com a memória RAM própria da NX, disponibilizada junto com as bibliotecas do NXmap e foram obtidos os seguintes resultados, demonstrados na tabela 4.

Tabela 4 – Resultado da síntese no NXmap.

	Utilizado/Total	Porcentagem
4-LUT	1552/129024	2%
DFF	615/129024	1%
XLUT	0/8064	0%
Carry	181/32256	1%
File Block	0/672	0%
Domain Clock	0/672	0%
Buffer	0	-
Switch	0/1344	0%
Signal Processor	0/384	0%
Block	1/192	1%
WFG	3/40	8%
PLL	0/4	0%

Fonte: Próprio autor.

Porém, não foi possível simular o sistema portado para a BRAVE, pois o mesmo estava apresentando algumas falhas. Posteriormente foi verificado que os problemas encontrados tinham relação com o modo que o programa NXmap organiza os "*latches*", não havendo tempo hábil para reorganização de todo o IPCore. Porém, mesmo não sendo implementado na plataforma BRAVE foi possível validar a utilização do controlador CAN, por meio de simulações realizadas no Quartus, bem como, introduzir o uso do controlador CAN na plataforma.

Uma análise mais completa e que complementa as informações deste capítulo pode ser vista nas seções 6.2.3 e 6.2.5 onde as informações de envio das mensagens são baseadas nas mensagens advindas da missão FloripaSat-1.

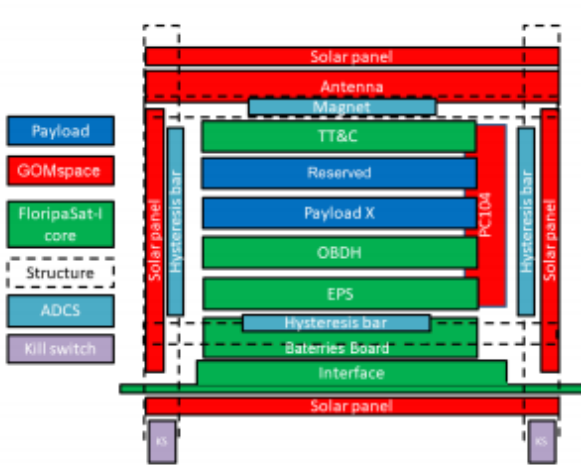
6.2 BARRAMENTO CAN PARA MICROCONTROLADOR (FLORIPASAT)

Neste capítulo, será detalhado o modelo de implementação sugerido para o FloripaSat, com base nos conceitos apresentados no capítulo 5.

6.2.1 A Plataforma FloripaSat

A plataforma FloripaSat foi utilizada a primeira vez no desenvolvimento da missão FloripaSat-1, primeiro nanossatélite desenvolvido na Universidade Federal de Santa Catarina (UFSC) pelo Grupo de Sistemas Espaciais (GSE) com a supervisão do professor Eduardo Augusto Bezerra. Este projeto tem o objetivo de capacitar alunos da graduação e pós-graduação para trabalharem com sistemas de aplicações espaciais. Para este primeiro satélite foram utilizados os três módulos da plataforma (EPS, OBDH e TT&C), desenvolvidos para realizar o controle do satélite e dois módulos de carga útil (PAYLOAD-X e PAYLOAD-RUSH), como pode ser visto na figura 18.

Figura 18 – Módulos FloripaSat-1.

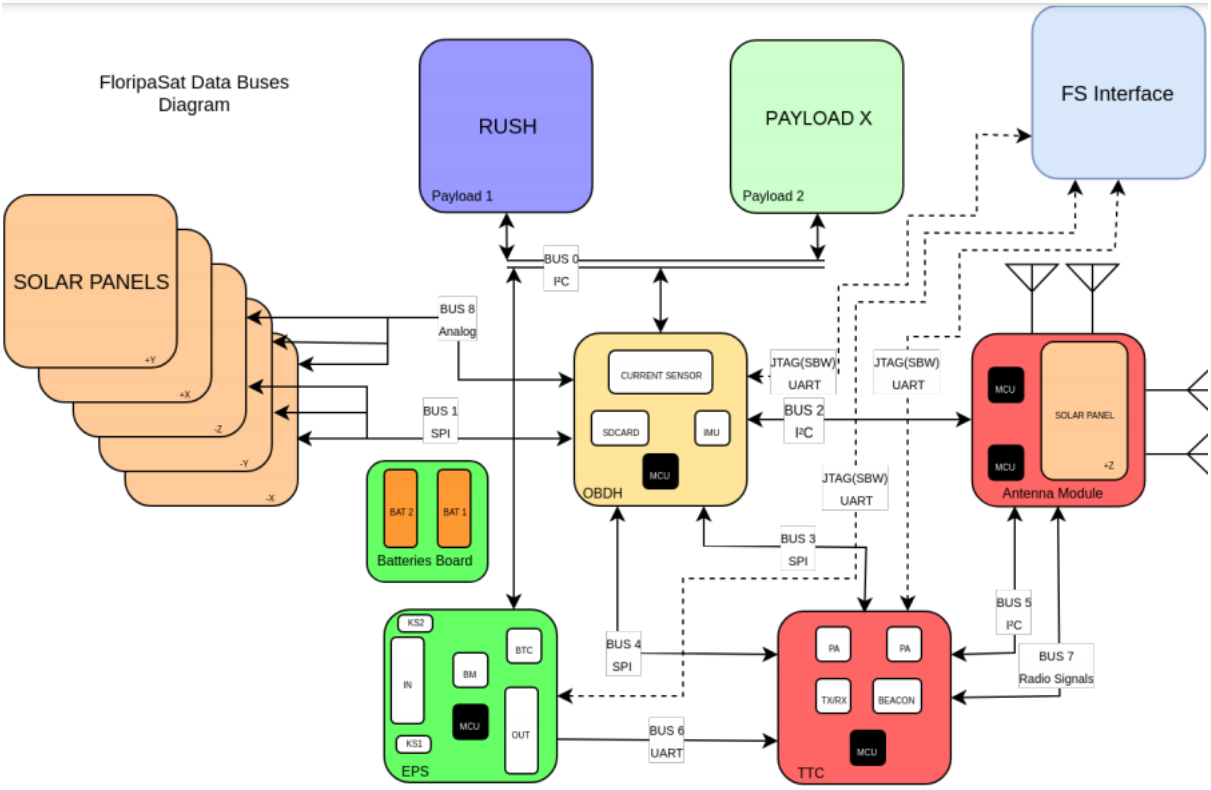


Fonte: FloripaSat, 2020

Uma parte fundamental do projeto, para que o satélite possa funcionar de forma satisfatória, é o desenvolvimento de um sistema de comunicação embarcada robusto, para que a troca de informações entre os módulos ocorra de forma segura e eficaz. No FloripaSat-1 foram utilizados diversos protocolos de comunicação, entre eles, *Serial Peripheral Interface* (SPI),

Universal Asynchronous/Synchronous Receiver/Transmitter (UART/USART), I2C e 1-Wire, onde ao total foram 9 barramentos individuais, como pode ser visto na figura 19.

Figura 19 – Barramentos FloripaSat-1.



Fonte: FloripaSat, 2020

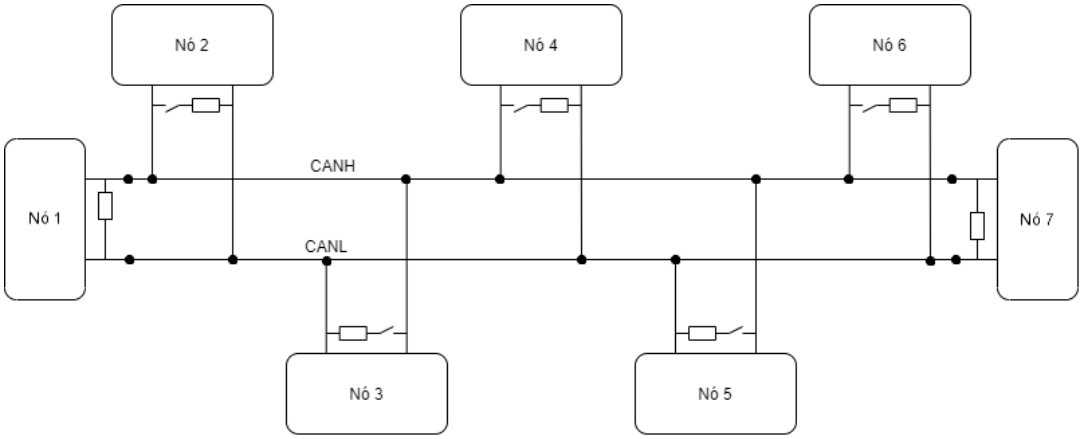
- Bus 0 – I2C principal: Comunicação entre OBDH (mestre), EPS e Payloads, com velocidade de 100 Kbps.
- Bus 1 – SPI: Comunicação entre OBDH e painéis solares (sensores de Temperatura), com velocidade de 8 Mbps.
- Bus 2 – I2C: Comunicação entre OBDH (mestre) e MCU 1 da antena, com velocidade de 100 Kbps.
- Bus 3 – SPI: Comunicação entre OBDH e TT&C, com velocidade de 2 Kbps.
- Bus 4 – SPI: Comunicação entre OBDH e rádio principal, com velocidade de 8 Mbps.
- Bus 5 – I2C: Comunicação entre MCU do Beacon e MCU 2 da antena, com velocidade de 100 Kbps.
- Bus 6 – UART: Comunicação entre MCU do EPS e Beacon, com velocidade de 9600 bps.

- Bus 7 – Rádio: Comunicação entre o rádio e a antena.
- Bus 8 – Analógico: Leitura dos fotodiodos dos painéis solares.

6.2.2 Modelo de hardware proposto

Para o projeto do barramento CAN foram utilizados conceitos vistos nos trabalhos de Woodroffe e Madle (2004) e Wielandt et al (2012), onde foram utilizados dois barramentos CAN para aumentar a confiabilidade de sistema de comunicação, como demonstrado na figura 13. Para que o sistema de comunicação apresente um maior grau de escalabilidade, ou seja, a possibilidade de inclusão e exclusão de nós no barramento seja feita de modo simples e sem causar grandes alterações no barramento, sugere-se o uso de um resistor de 120 ohms com uma chave logo após a saída do *transceiver* CAN, isso é necessário devido à obrigatoriedade deste resistor nas extremidades do barramento. O modelo é apresentado na figura 20.

Figura 20 – Modelo sugerido para facilitar a inserção de nós.



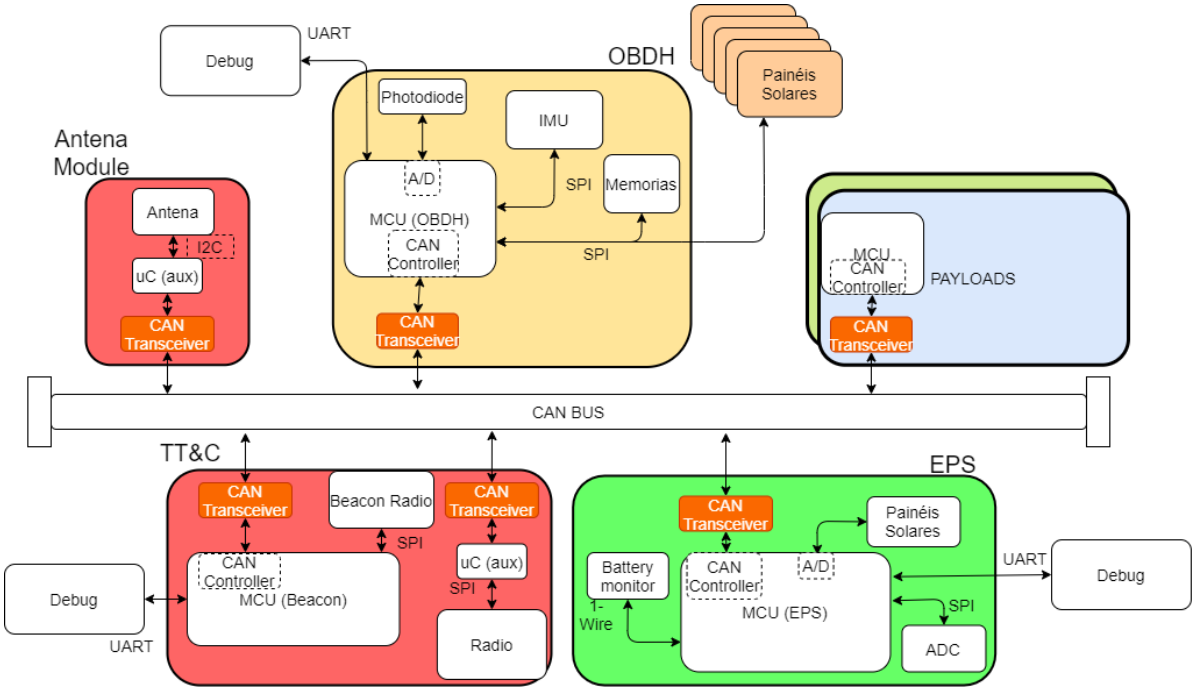
Fonte: Próprio Autor

Para o modelo proposto, sugere-se a utilização do MCU Hercules da Texas Instruments, o modelo TMS570, por ser um MCU já usado em missões com nanossatélite anteriormente e possuir a interface de comunicação CAN com controlador CAN interno, como demonstram Leppinen et. al. (2014), além de já ter sido utilizado em testes com radiação ionizante como mostra Gebelein, Gudge e Kebschull (2016). Estes fatores associados com o fato do microcontrolador possuir dois processadores funcionando em Lockstep e possuir ECC, o tornam uma opção viável para ser utilizado na nova versão proposta da plataforma FloripaSat.

No modelo proposto, a comunicação entre cada módulo é realizada por meio do barramento CAN, porém a comunicação entre os sensores e periféricos, bem como, a

comunicação entre componentes que estão na mesma placa de circuito impresso, são realizadas pelo modo de comunicação que mais convier para o sistema (podendo ser utilizado o protocolo CAN, se necessário). Esta abordagem foi escolhida para otimizar o uso do hardware, uma vez que cada dispositivo com CAN necessita de um *transceiver* CAN, para este trabalho foram mantidos os barramentos de comunicação já utilizados entre os periféricos do FloripaSat-1, como demonstrado na figura 21.

Figura 21 – Modelo sugerido para a disposição dos nós no barramento CAN.



Fonte: Próprio Autor

6.2.3 Mensagens

Para o modelo proposto foi levantado o fluxo de mensagens do FloripaSat-1 e com todas as mensagens foi possível analisar as características do barramento CAN com relação a utilização, probabilidade de erro, entre outros.

As mensagens foram separadas por prioridade, assim sendo, a mensagem para liberar a antena tem a maior prioridade, uma vez que não é uma mensagem recorrente, logo em seguida as mensagens com os dados do satélite, as mensagens recebidas pela antena e, por fim, as mensagens das cargas úteis. Desta maneira, as mensagens foram separadas em quadros e calculados seus tempos de envio, conforme demonstrado na equação 2 e no quadro 6.

As mensagens foram definidas a partir da sua origem, a informação que ela transporta e o número do quadro, como demonstrado a seguir:

- OAx: mensagem da OBDH para ejetar a antena;
- TAx: mensagem da TT&C para ejetar a antena;
- OCDx: mensagem da OBDH com os dados completos do FloripaSat (OBDH Complete Data);
- EBDx: mensagem da EPS com os dados básicos do FloripaSat (EPS Basic Data);
- OBDx: mensagem da OBDH com os dados básicos do FloripaSat (OBDH Basic Data);
- ECDx: mensagem da EPS com os dados completos do FloripaSat (EPS Complete Data);
- RADx: mensagem advindas do rádio;
- OPRx: mensagem da OBDH para a Payload-R;
- OPXx: mensagem da OBDH para a Payload-X;

Quadro 6 – Mensagens no barramento CAN do FloripaSat.

ID msg	Nº bytes	C_m (Eq.2)	Idm (bits 0-3)	Idq (bits 4-10)	Id
OA0	1	65 us	0x00	0x00	0x000
TA0	1	65 us	0x01	0x00	0x080
OCD0	8	135 us	0x02	0x00	0x100
OCD1	8	135 us	0x02	0x01	0x101
...	...	...	...	...	...
OCD61	7	125 us	0x02	0x3D	0x13D
EBD0	8	135 us	0x04	0x00	0x200
EBD1	8	135 us	0x04	0x01	0x201
...	...	...	...	...	...
EBD3	7	125 us	0x04	0x03	0x203
OBD0	8	135 us	0x05	0x00	0x280
OBD1	8	135 us	0x05	0x01	0x281
...	...	...	...	...	...

OBD6	2	75 us	0x05	0x06	0x286
ECD0	8	135 us	0x07	0x00	0x380
ECD1	8	135 us	0x07	0x01	0x381
...	...	...	...	...	...
ECD8	5	105 us	0x07	0x08	0x388
RAD0	8	135 us	0x0B	0x00	0x580
RAD1	8	135 us	0x0B	0x01	0x581
...	...	...	...	...	...
RAD3	4	95 us	0x0B	0x02	0x583
OPR0	8	135 us	0x0C	0x00	0x600
OPR1	8	135 us	0x0C	0x01	0x601
...	...	...	...	...	...
OPR4	8	135 us	0x0C	0x04	0x604
OPX0	8	135 us	0x0D	0x00	0x680
OPX1	8	135 us	0x0D	0x01	0x681
...	...	...	...	...	...
OPX24	1	65 us	0x0D	0x18	0x698

Fonte: Próprio Autor.

Com o levantamento das mensagens transmitidas no barramento CAN foi feita a análise probabilística para o pior caso de tempo de resposta demonstrada na seção 5.1.1 para as mensagens, para realizar a análise matemática foi desenvolvido um script no Matlab que será detalhado na seção 6.2.4.

6.2.4 Modelo calculado no MatLab

Devido à complexidade dos cálculos introduzidos na seção 5.1.1, principalmente, quando levado em conta a probabilidade de erro em cada quadro enviado, foi utilizada a ferramenta MatLab para auxiliar na análise. Sendo assim foi desenvolvido um script, que está listado na figura 22.

Com o script demonstrado na figura 22, foram obtidos os conjuntos de valores para o pior caso e suas probabilidades para todos os quadros seguindo o modelo apresentado no quadro 7.

Figura 22 – Script para o MatLab.

```
Inserir: velocidade do barramento (vel)
Inserir: número de mensagens (Nmsg)

For (1:Nmsg)
        Inserir: Período da mensagem (Tm)
        Inserir: Número de bytes da mensagem (Nb)
        Inserir: Jitter da mensagem (Jm)
        Calcular: número de quadros necessários (Nq)
        For (1:Nq)
                Calcular: número de bytes no quadro (Nbq)
                Calcular: Tempo de envio (Cm)
                Save {Nmsg, Nq, Tm, Nbq, Jm, Cm}
        End
End

For (1:total de quadros)
        Calcular: Tempo Bloqueado (Bm)
End

For (1:total de quadros)
        While (Rm<Tm-Jm ou Probabilidade erro < E*)
                Calcular: Tempo em fila (wm) // wm = Bm + Im
                Calcular: Rm = Cm + Bm + Im + Ji
                Calcular: Probabilidade de erro (Prob)
        End
        Save   {Rm,Prob}    p/ 0 erros
               {Rm,Prob}    p/ 1 erro
               ---          ---
End

For (1:total de quadros)
        Utilização do barramento (U)
End

* Sendo E um valor da probabilidade tão pequeno que pode ser
desconsiderado (definido pelo desenvolvedor de acordo com as
restrições do seu projeto)
```

Fonte: Próprio Autor

Quadro 7 – Conjunto de informações obtidas com base nos cálculos do MatLab.

Mensagem	Quadro	Período	Nº Bytes	Jitter	Tempo de envio (Cm)	Pior caso (Rm)	Probabil.
Com 1 erro							
Com 2 erros							
Com 3 erros							
Com 4 erros							
Com 5 erros							

Fonte: Próprio Autor

Os cálculos completos para todas as mensagens podem ser encontrados no Apêndice B.

6.2.5 Modelo simulado

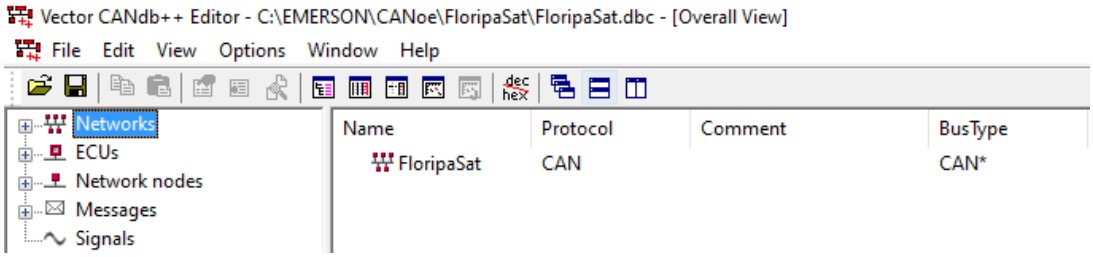
Para validação dos cálculos apresentados no capítulo 5, buscou-se um programa que possibilitasse simular a rede CAN e configurar os nós conforme necessário, sendo assim, foi escolhido o CANoe da Vector, empresa alemã com foco em desenvolvimento de sistemas para utilização em automóveis.

O simulador é composto da união de quatro diferentes programas, o CANdb++, Panel, CAPL e o CANoe.

6.2.5.1 CANdb++

O CANdb++ é o programa responsável por definir toda a base de dados da simulação e onde serão configuradas todas as características principais da rede. Nele são configuradas as informações, como quantos nós terá a rede, quais os nós do sistema, quantas mensagens cada nó irá enviar e receber, quais os tipos de mensagens serão enviadas, entre outras. As partes principais do CANdb++ são: Networks, ECU (*Electronic Control Unit,* ou Unidade de controle eletrônico, em português), Network Nodes, message e signals, como mostra a figura 23.

Figura 23 – CANdb++.



Fonte: Próprio Autor

Para um melhor entendimento do funcionamento do programa, será explicado cada um dos seus componentes.

6.2.5.1.1 Signals

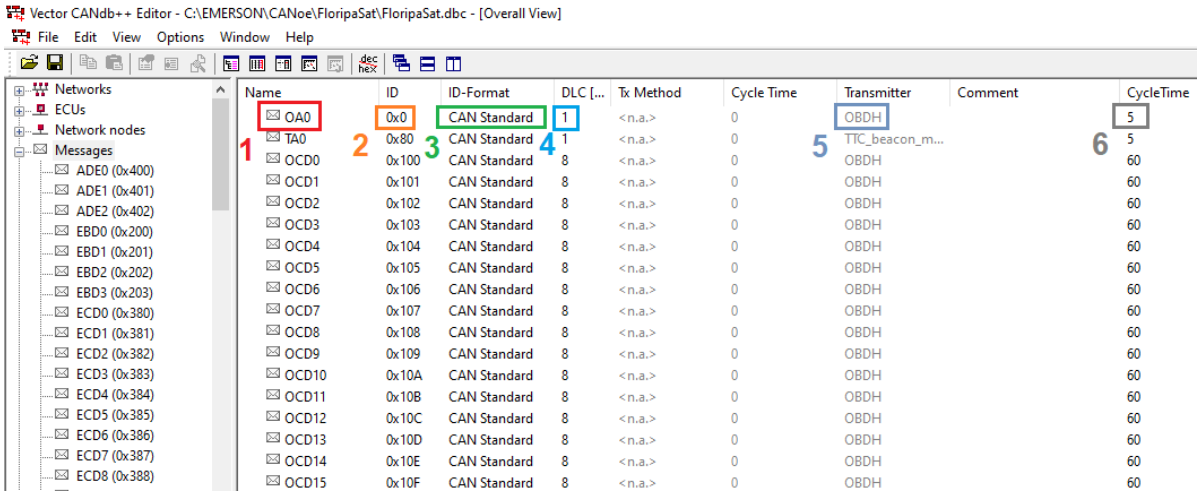
Os sinais são informações provenientes dos sensores ou das tarefas, estes sinais serão a carga útil transportada em uma mensagem. Neste exemplo não foram utilizados sinais, uma vez

que para a análise estatística do barramento a informação transportada não possui grande importância, sendo necessário apenas que o quadro possua o tamanho desejado.

6.2.5.1.2 Messages

As mensagens, como mencionado anteriormente, são responsáveis por transportar as informações entre os nós do barramento, nas mensagens são configuradas as informações básicas de cada quadro, como demonstrado na figura 24.

Figura 24 – Mensagens.



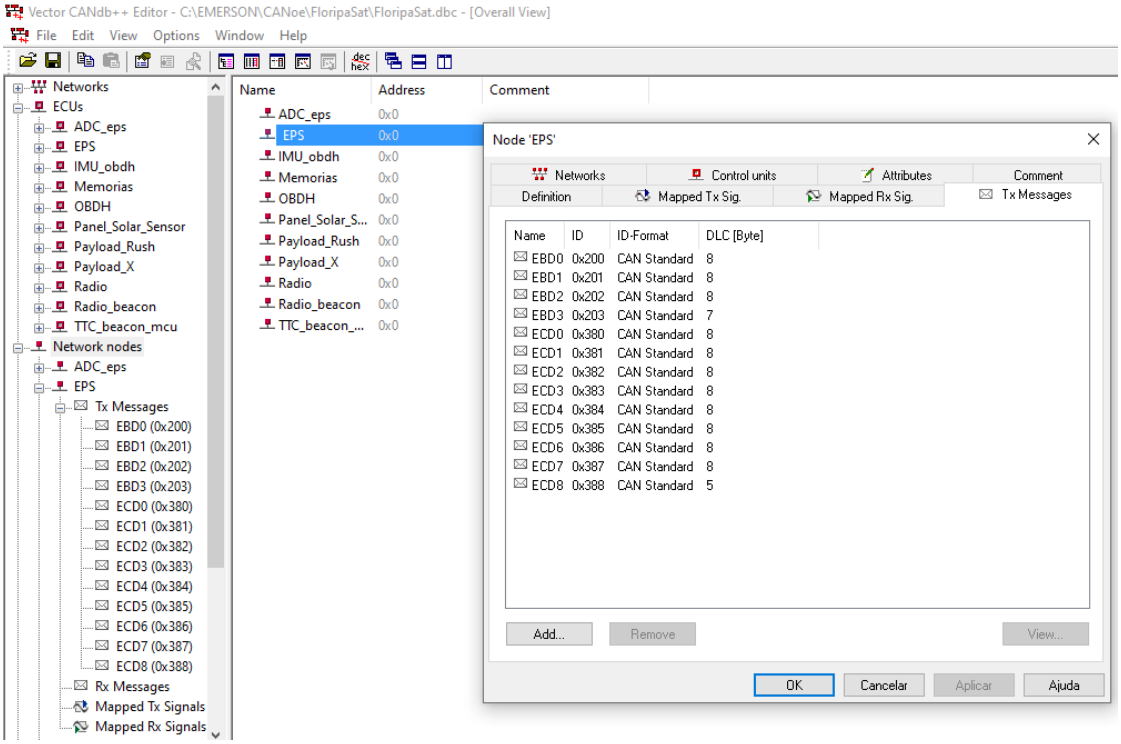
Fonte: Próprio Autor

Nesta parte são configuradas as mensagens com relação ao seu nome (1), identificação/prioridade (2), formato do quadro CAN (3), quantidade de bytes que serão enviados (4), origem da mensagem / nó transmissor (5) e o período (6).

6.2.5.1.3 Network Nodes e ECUs

Os nós da rede e as Unidades de Controle Eletrônicas foram unificadas em uma seção, pois uma é parte componente da outra, neste caso, a ECU é a unidade de processamento, enquanto o nó da rede é responsável por realizar a interface entre a ECU e o barramento CAN, sendo assim, ao criar um nó de rede, automaticamente uma ECU é criada. Nos nós de rede são definidas as mensagens que o nó enviará e receberá e no ECU são inseridos os nós, como é possível ver na figura 25.

Figura 25 – Nós da rede e ECUs.

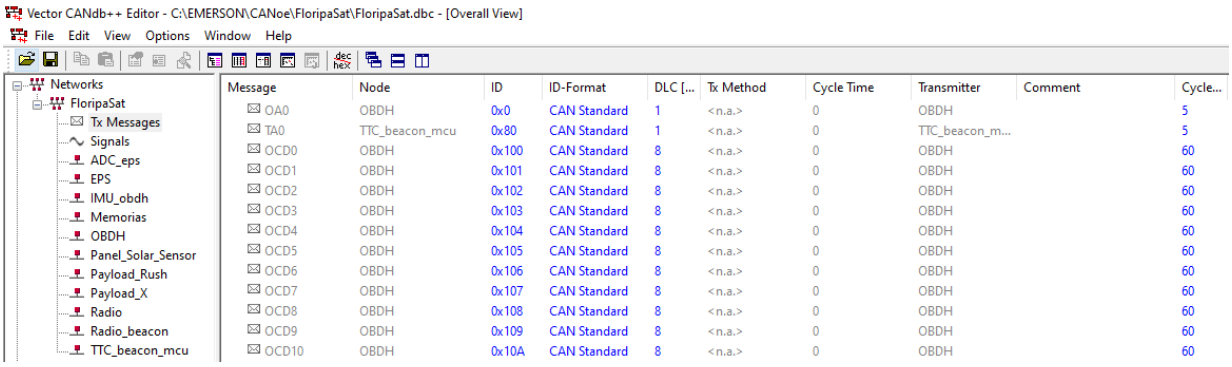


Fonte: Próprio Autor

6.2.5.1.4 Networks

A network é a base para a configuração de todo o sistema, todas as configurações realizadas anteriormente ficam salvas em uma network e a partir destas informações são desenvolvidas as demais configurações que serão unificadas no CANoe. Na figura 26 é demonstrado como ficou a configuração final da rede.

Figura 26 – Network.

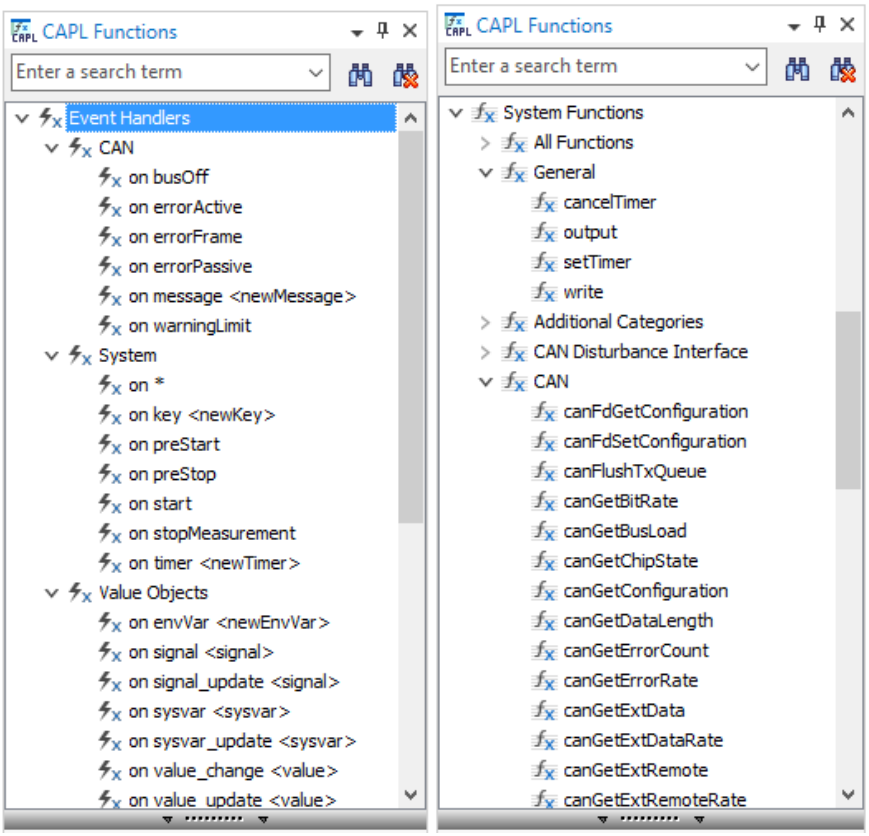


Fonte: Próprio Autor

6.2.5.2 CAPL Browser

O CAPL (*Communication Access Programming Language*) é uma linguagem alto nível similar a linguagem C e C++. Por meio do CAPL Browser é possível programar cada nó da rede CAN baseado em diversos eventos e funções pré-existentes, como pode ser visto na figura 27.

Figura 27 – CAPL Functions.



Fonte: Próprio Autor

Um modelo de programa desenvolvido no CAPL Browser é demonstrado na figura 28, onde inicialmente são definidas as variáveis, no caso, as mensagens que serão transmitidas. Posteriormente, é definido o modo de envio, nesta aplicação é um timer periódico com período de 10 segundos e a cada vez que o timer é acionado ele reinicia sua contagem e envia as mensagens.

Figura 28 – CAPL Program.

```
/*@!Encoding:1252*/
includes
{

}

variables
{
  message MB0 msg23; // a cada 10s
  message MB1 msg24;
  message MB2 msg25;
  message MB3 msg26;
  message MB4 msg27;
  message MB5 msg28;
  message MB6 msg29;
  message MB7 msg30;

  Timer myTimer10;
}

on start
{
  myTimer10.set(msg23.CycleTime);
}

on timer myTimer10
{
  myTimer10.set(msg23.CycleTime);
  output(msg23);
  output(msg24);
  output(msg25);
  output(msg26);
  output(msg27);
  output(msg28);
  output(msg29);
  output(msg30);
}

```

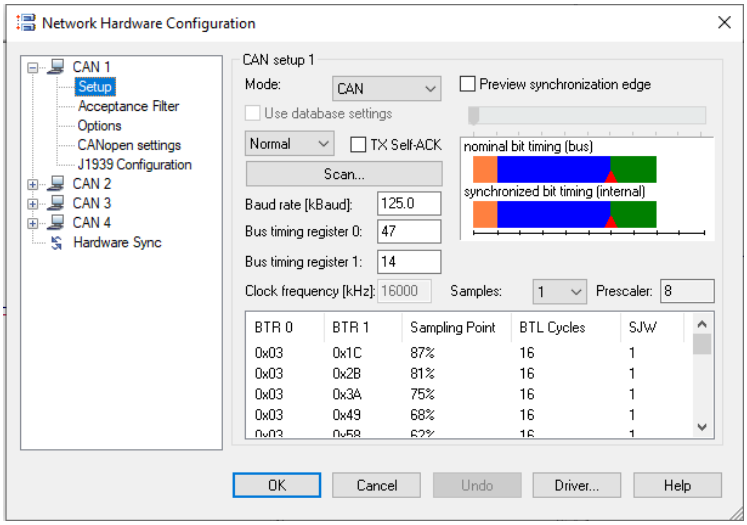
Fonte: Próprio Autor

6.2.5.3 CANoe

O CANoe é o componente principal para a simulação do barramento CAN, é neste programa que são unificadas todas as configurações realizadas anteriormente e também é realizada a configuração das características do barramento, como velocidade, topologia, disposição dos nós, entre outros. Também é no CANoe que é feita a simulação, onde é possível verificar os parâmetros da rede e todas as suas características de funcionamento.

Inicialmente, definem-se características básicas do barramento, como velocidade e bit timing, como pode ser visto na figura 29.

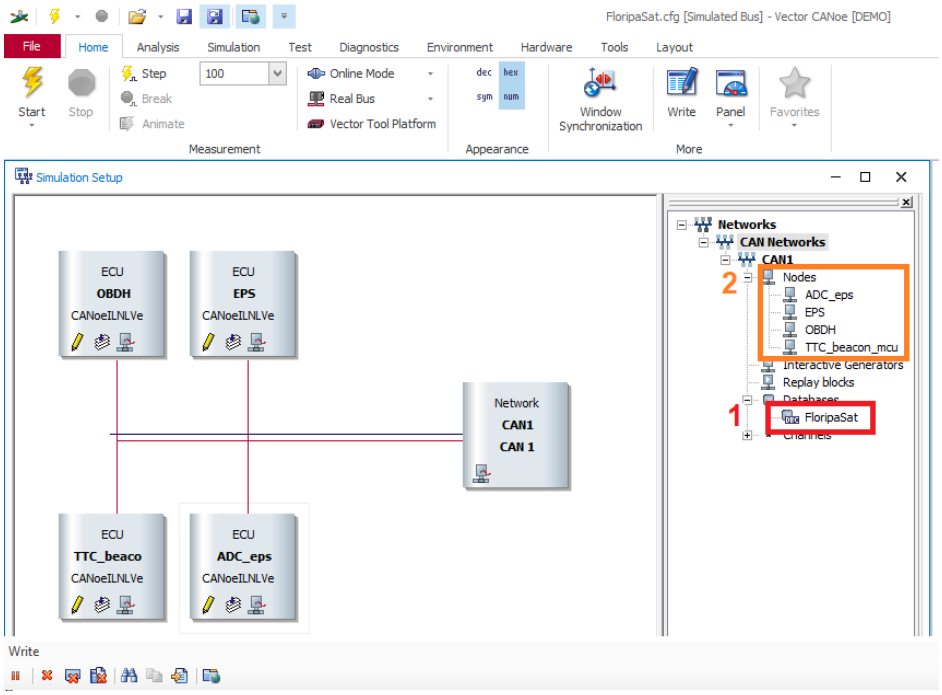
Figura 29 – Configuração CANoe.



Fonte: Próprio Autor

Após as configurações básicas do simulador, são inseridos na rede os componentes já desenvolvidos, para isto é necessário associar a base de dados (CANdb++) ao CANoe (1), bem como, os nós (2) que devem ser utilizados, demonstrado na figura 30.

Figura 30 – Interface CANoe.



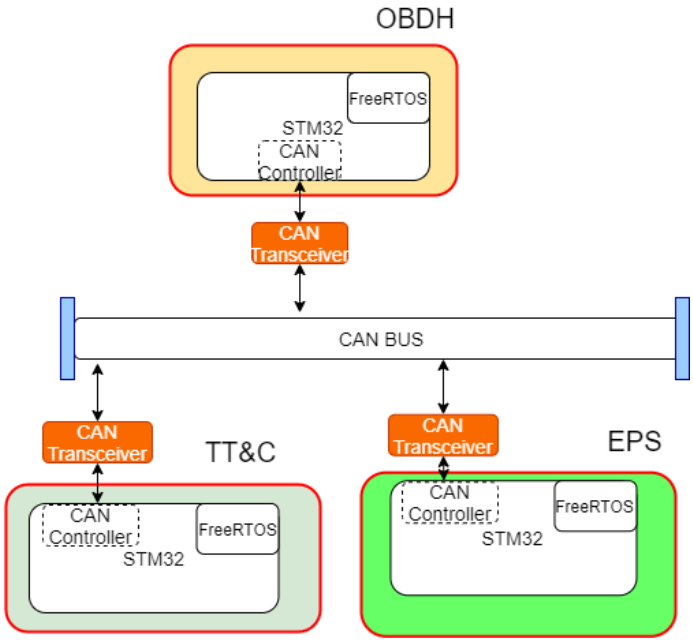
Fonte: Próprio Autor

Como esta é uma versão Demo do programa é possível utilizar apenas 4 nós em um barramento, sendo assim, todas as mensagens advindas dos sensores e Payloads foram unificadas no ADC_eps, não afetando o funcionamento do barramento para determinação dos parâmetros desejados.

6.2.6 Modelo Prático

Para verificar o funcionamento do barramento simulado foi criada uma pequena rede CAN com microcontroladores para verificar o envio das mensagens. Para este sistema, foram utilizados três MCU's STM32F103C8T6 para representar os principais módulos do FloripaSat (OBDH, EPS e TT&C) e utilizado o sistema operacional de tempo real, FreeRTOS, para gerenciamento das tarefas, como mostrado na figura 31.

Figura 31 – Modelo prático.



Fonte: Próprio Autor

No modelo mostrado na figura 31, os MCU's (os três retângulos vermelho) da OBDH (superior), EPS (inferior direito) e TT&C (inferior esquerdo), são conectados aos transceivers CAN (retângulos cor de laranja) que tem a função de converter os níveis lógicos dos microcontroladores (Tx e Rx) para níveis diferenciais (CAN_H e CAN_L). Com o sistema funcionando foi possível, com o auxílio de um analisador lógico, confirmar o funcionamento correto do modelo simulado, bem como, validar o protocolo CAN.

6.3 RESULTADOS

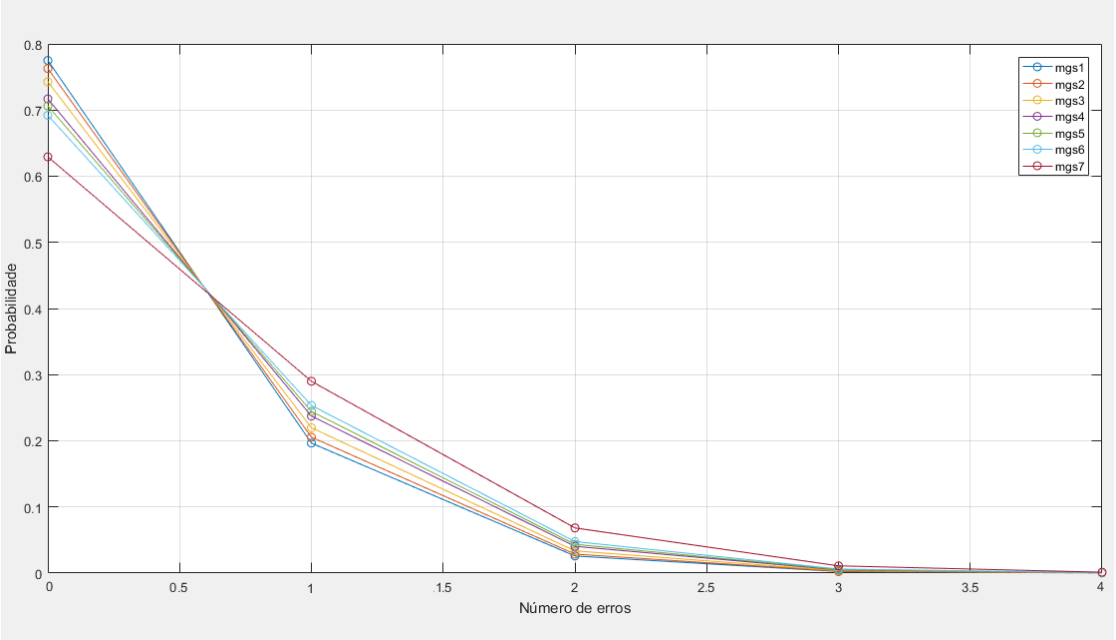
Nesta seção serão demonstradas as análises realizadas e resultados obtidos para o FloripaSat-1, com base nos conceitos matemáticos, modelo simulado e no modelo prático.

6.3.1 Análise matemática

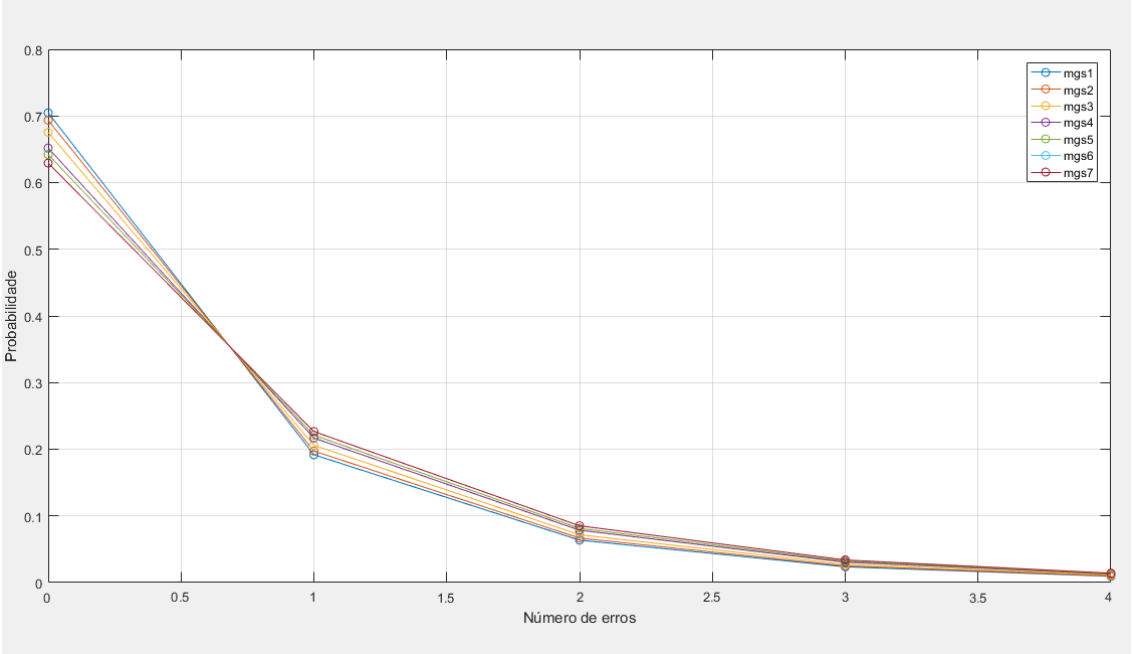
Os cálculos foram realizados para ambos os modelos de envio, o primeiro demonstrado no quadro 6, o qual será denominado **envio simples**, pois pode sofrer interferência entre o envio de um quadro e seu subsequente (caso de uma mensagem de maior prioridade entre na fila) e o segundo, denominado **envio em burst**, onde uma mensagem deve ser enviada completamente para que outra possa assumir o barramento, como sugerido no quadro 5. Para esta validação os cálculos foram baseados em um barramento com velocidade de 1 MHz, velocidade máxima do barramento CAN.

Na figura 32.a e 32.b, são mostrados os gráficos da probabilidade de erro em relação ao número de ocorrências para a forma de envio simples e em burst, respectivamente. Enquanto nas figuras 33.a e 33.b são demonstrados os piores tempos de resposta para cada mensagem em relação ao número de ocorrência de erros.

Figura 32 – Gráficos da probabilidade de erros das mensagens CAN [1MHz].



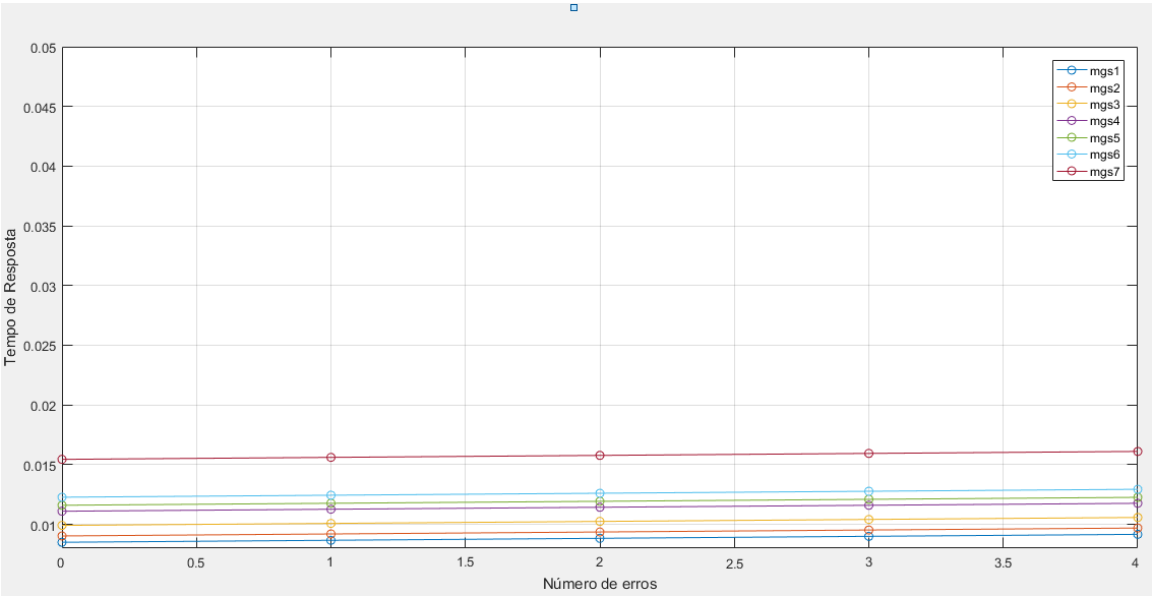
a)



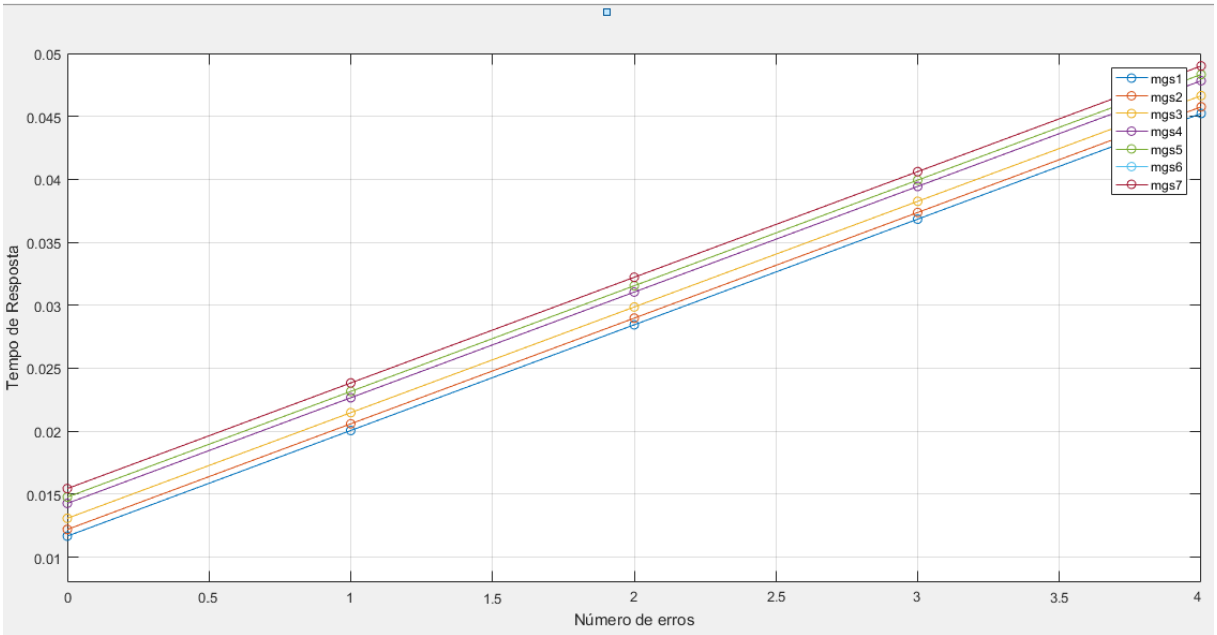
b)

Fonte: Próprio Autor

Figura 33 – Gráficos do pior tempo de resposta das mensagens CAN [1MHz].



a)



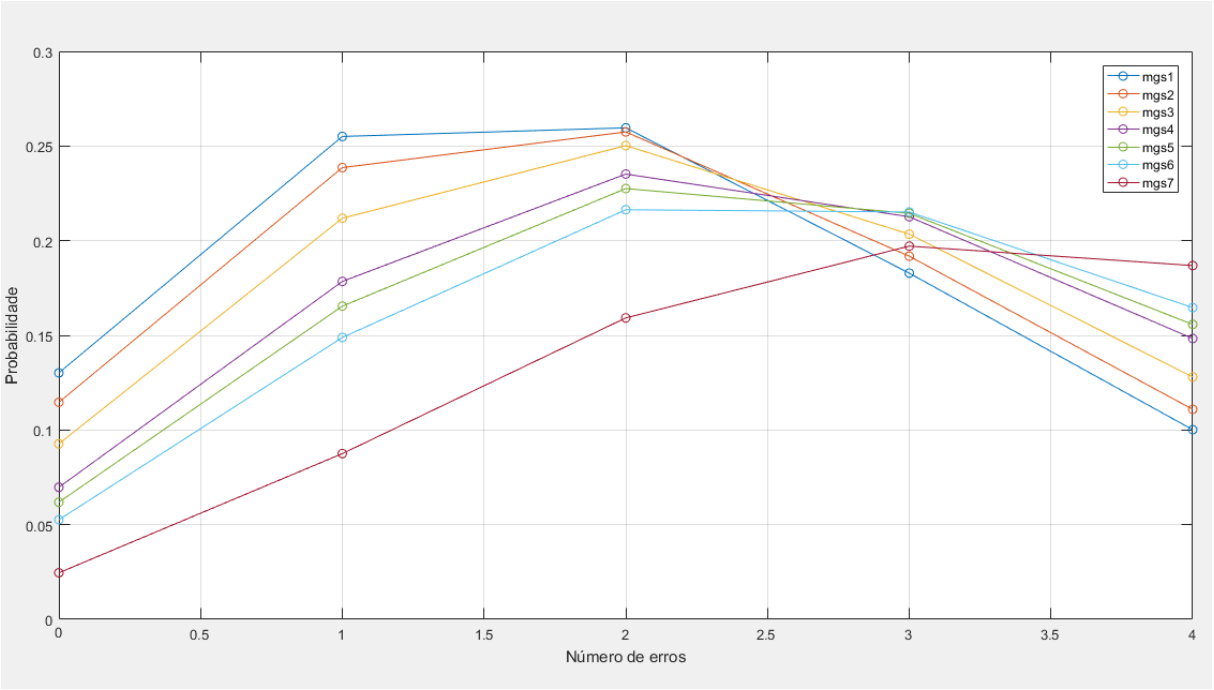
b)

Fonte: Próprio Autor

Após a análise realizada, fica claro que a melhor opção, neste caso, é realizar o envio simples, pois nos gráficos 32.a e 32.b é possível verificar que no envio simples há uma maior probabilidade de não ocorrerem erros e uma probabilidade mais baixa de que ocorram mais de dois erros. Também é possível verificar que o pior tempo de resposta no caso do envio em burst sofre um aumento considerável, uma vez que mensagens de mais alta prioridade podem ficar aguardando a conclusão do envio de mensagens de prioridade menor. Assim, os cálculos a seguir são baseados apenas no modelo de envio simples.

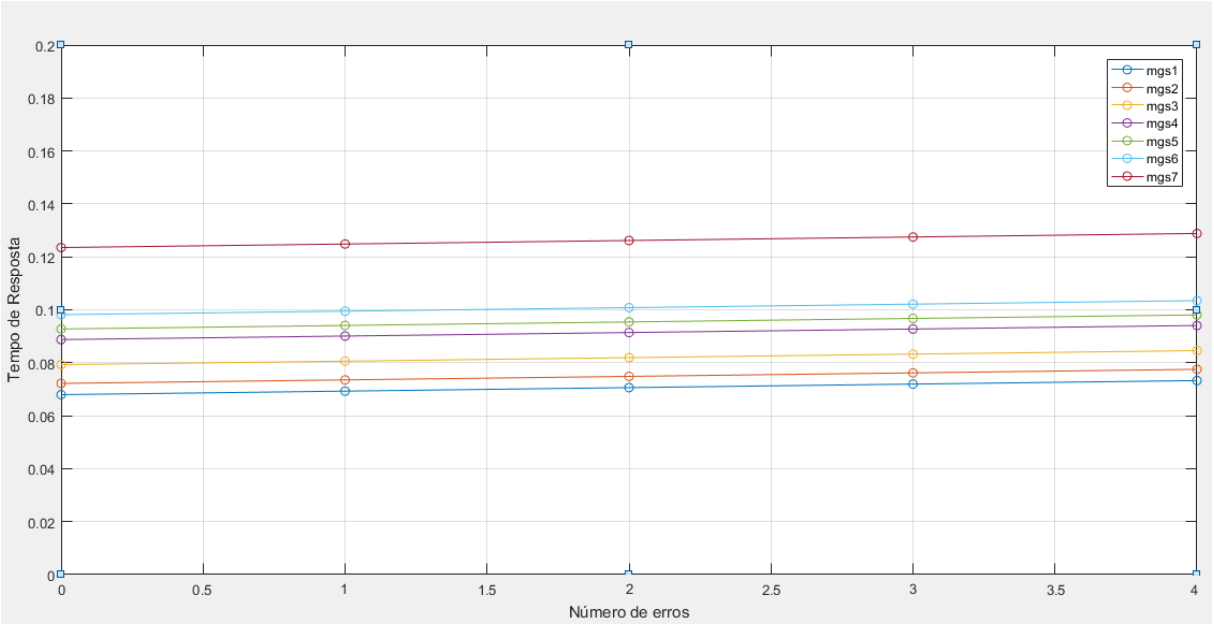
Foram realizados testes para a velocidade de 125 KHz para verificar sua viabilidade e para comparação com o programa CANoe. Com os cálculos realizados para velocidade de 125 KHz foram obtidos os gráficos de probabilidade de erro e de pior tempo de resposta mostrados nas figuras 34 e 35, respectivamente.

Figura 34 – Gráficos da probabilidade de erros das mensagens CAN [125 KHz].



Fonte: Próprio Autor

Figura 35 – Gráficos do pior tempo de resposta das mensagens CAN [125 KHz].



Fonte: Próprio Autor

Com a conclusão dos cálculos realizados no MatLab é possível iniciar a análise e comparação dos resultados simulados no CANoe. Um cálculo importante para ser comparado é a utilização do barramento (Busload), dado pelo somatório da razão do tempo máximo de transmissão pelo período de cada quadro. Demonstrado na equação 17.

$$U = \sum \frac{C_m}{T_m} \quad (19)$$

Com os cálculos foram obtidos valores da utilização do barramento para 1MHz e para 125 KHz:

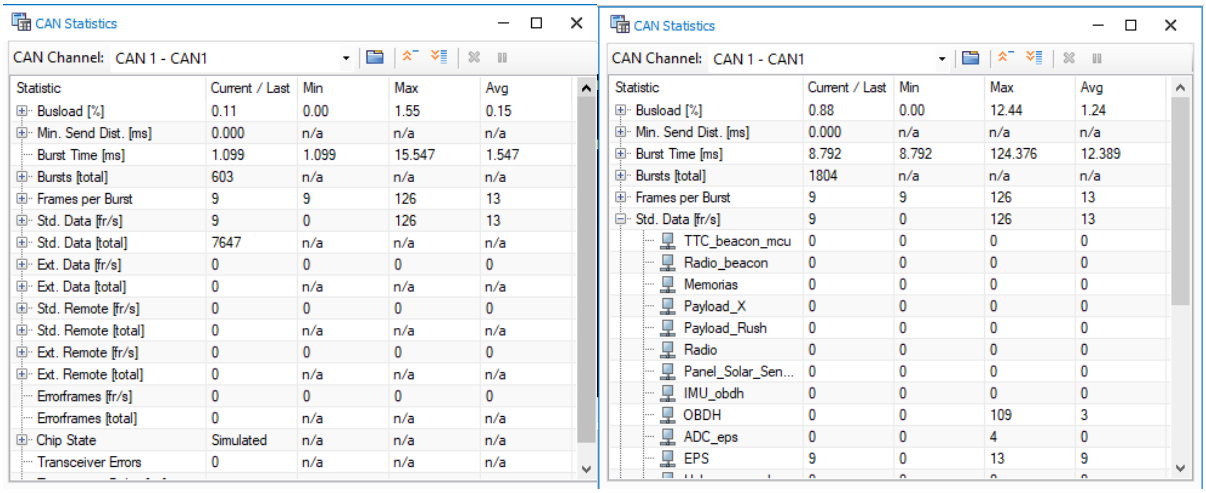
$$U_{[1MHz]} = 0{,}16\ \%$$

$$U_{[125KHz]} = 1{,}31\ \%$$

6.3.2 Análise do modelo simulado

Os valores obtidos na seção anterior podem ser confirmados por meio do programa CANoe, para isto o modelo desenvolvido foi simulado por 10 minutos, para que as mensagens que ocorrem com o maior período (1 minuto) sejam transferidas no barramento no mínimo dez vezes. Com a simulação, foram obtidos valores de utilização do barramento de 0,15% e 1,24%, como pode ser visto nas figuras 36.a e 36.b, juntamente com diversas características do barramento, como tempo de envio em sequência (Burst Time), número de quadros enviados em sequência (Frames per Burst), entre outras. Todas estas características podem ser expandidas para ver cada nó separadamente, como feito na figura 36.b para os dados enviados no formato padrão (Std. Data).

Figura 36 – CAN Statistics.

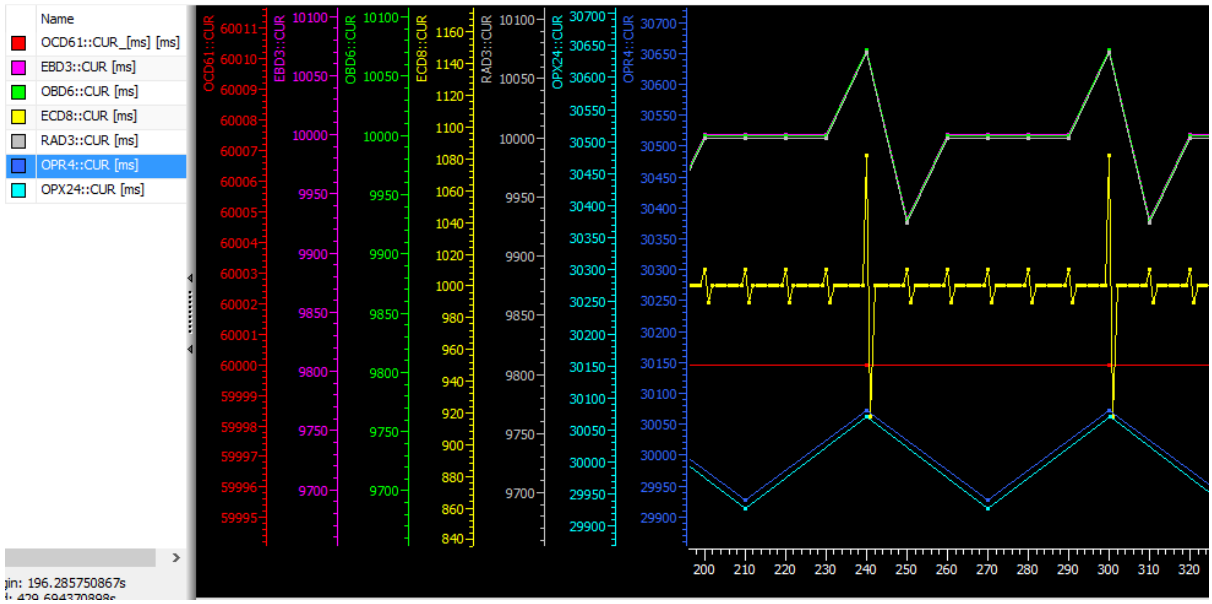


a) b)

Fonte: Próprio Autor

Outra análise bastante relevante para fazer é a interferência que cada mensagem pode sofrer ao ser solicitado o envio ao mesmo tempo que outra mensagem de prioridade mais alta. Tal análise pode ser feita com diferentes ferramentas disponibilizadas pelo programa, tais como, *Graphics, Trace* ou *Data*, neste caso sendo utilizado a ferramenta de gráfico para uma melhor compreensão do envio das mensagens no tempo, como mostra a figura 37.

Figura 37 – Gráfico da interferência sobre cada mensagem.



Fonte: Próprio Autor

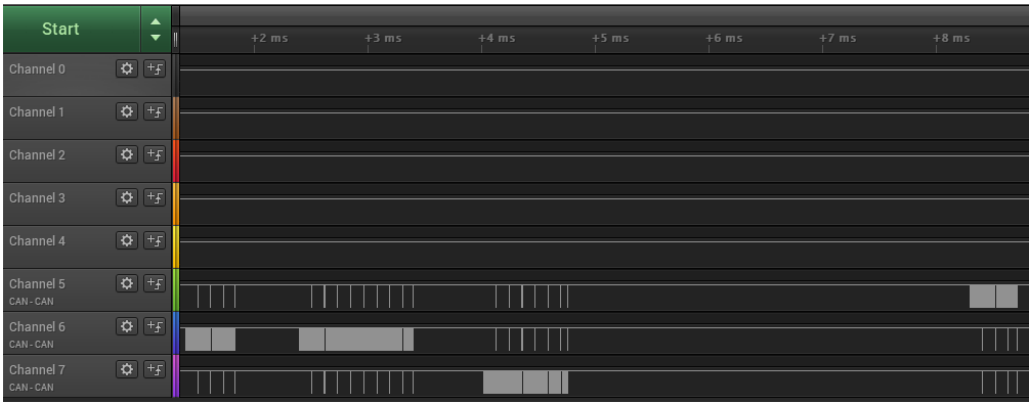
Neste gráfico é possível verificar a interferência que cada mensagem gera nas mensagens subsequentes, sendo assim, as mensagens de maior prioridade OCD (Vermelho) não sofrem nenhuma alteração no seu tempo de envio, por sua vez, todas as demais mensagens sofrem um atraso a cada 60 segundos quando são transmitidas as mensagens OCD. As mensagens ECD (Amarelo), que são enviadas a cada um segundo, também sofrem um pequeno atraso a cada 10 segundos com o envio das mensagens EBD e OCD [Roxo e Verde].

6.3.3 Análise do modelo prático

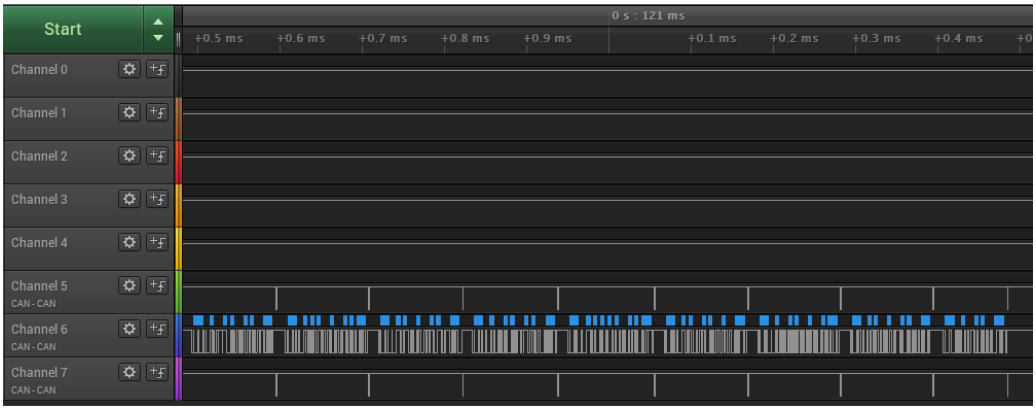
Com o modelo prático é possível validar as informações obtidas matematicamente e simuladas. Com o analisador digital foi analisado o barramento físico e todas as mensagens transmitidas em tempo real.

Nas figuras 38.a, 38.b e 38.c são apresentadas as informações obtidas no barramento CAN.

Figura 38 – Analisador lógico.



(a)



(b)

(c)

Na figura 38.a é possível ver cada módulo enviando suas mensagens, enquanto os demais módulos enviam o bit de ACK, aproximando um pouco (figura 38.b) é possível analisar o tempo que foi necessário para o envio de uma mensagem inteira, bem como, na figura 38.c tem-se o tempo necessário para o envio de um quadro e todas as suas características, como o identificador, os oito bytes de dados e o valor do CRC.

Com os resultados obtidos nas diferentes análises feitas para a nova configuração sugerida para a plataforma FloripaSat baseada em microcontroladores, foi possível obter um bom nível de confiabilidade no barramento. Também foram obtidos bons níveis de utilização do barramento e tempo de entrega das mensagens, desta forma, a utilização do protocolo CAN em um barramento único (com barramento redundante funcionando apenas em caso de falha do barramento principal), não acarretaria em perda de informação, nem comprometeria a entrega das informações no quesito temporal. Assim é possível validar a solução apresentada como uma alternativa para aumentar a robustez da plataforma FloripaSat, incrementando sua confiabilidade.

7 CONCLUSÕES

Neste trabalho, foi feito um estudo sobre a utilização do protocolo CAN na comunicação de CubeSats, levando em conta todos os requisitos de um barramento de comunicação de um satélite e analisando diversos trabalhos já realizados para buscar a melhor solução possível para implementação nos nanossatélites desenvolvidos pelo SpaceLab. Diversos fatores foram levados em conta para cada decisão de projeto, tais como, quantidade de mensagens que serão transportadas, tempo de envio de cada mensagem, periodicidade das mensagens, número de nós no barramento, entre outros.

Com a análise de todas as características, foram escolhidos os melhores modelos de transmissão para atender os requisitos de forma satisfatória. O projeto baseou-se em uma análise matemática sólida, por meio dos scripts desenvolvidos e buscando tornar o sistema o mais previsível possível dentro das características intrínsecas do protocolo CAN, para avaliar a confiabilidade do barramento e reduzir as falhas.

Assim, nos dois estudos de caso, foi demonstrado o sistema e suas características de modo prático, inicialmente utilizando um simulador e posteriormente desenvolvendo o sistema voltado para utilização em FPGAs e em microcontroladores. Um comparativo entre os trabalhos estudados e as propostas demonstradas neste trabalho são mostrados no quadro 8.

Quadro 8 – Comparativo entre as implementações apresentadas nos trabalhos estudados e as propostas do autor.

	BLANCO, C. D. V.; D'ERRICO, M.; CONTICCHIO, A.	JAYARATHNE, D. G. N.; JAYANANDA, M. K.	KHURRAM, M.; ZAIDI, S. M. Y.	REGES, J. E.; SANTOS, E. J. P.	WIELANDT, S. et al.	WOODROFFE, A, M.; MADLE, P.	LEMOS, É. T. (FPGA/MCU)
Microcontrolador(uC)	PIC 18F258	**	i386EX-EngineL	**	**	8051	interno/TMS570
Controlador CAN	Integrado ao uC	FPGA	MCP2515	FPGA (apenas camada MAC)	FPGA (Gaisler IP core)	FPGA	CAN_OC/interno
Transceiver CAN	MCP2551	**	MCP2551	**	Baseado no RS-485	Físico (COTS)	Físico (COTS)
Versão CAN	2.0B	**	2.0B	**	2.0B	CAN-SU	2.0B
Comunicação entre controlador CAN e uC	Interno	UART	UART	**	PCI bus	**	AMBA AHB/interno
Detecção e correção de erro	Sim	Não	Sim	Sim	Sim	Sim	Sim

Fonte: Próprio Autor

Observando todos os pontos desenvolvidos ao decorrer deste trabalho é possível verificar que os objetivos propostos foram atingidos, deixando uma análise relevante sobre a aplicação do protocolo CAN em nanossatélites. Vale ressaltar as contribuições mais importantes, como análise da confiabilidade do barramento, desenvolvimento de scripts para facilitar os cálculos, sugestão de modelo para o barramento e modelo para envio das mensagens, visando o melhor aproveitamento do barramento.

Este trabalho também atingiu o objetivo de ser uma fonte de informação para novos grupos de pesquisa e desenvolvimento de nanossatélite, que buscam informações sobre utilização do protocolo CAN como meio de comunicação principal para seus sistemas. Sendo assim, esta pesquisa se justifica pela crescente pesquisa na área de pequenos satélites e pela necessidade de protocolos confiáveis e simples para comunicação entre os subsistemas de um nanossatélite, além da necessidade de um maior conhecimento sobre o protocolo CAN em todas as etapas, desde o cálculo de viabilidade até o momento da implementação na plataforma que mais convier para o projeto.

7.1 TRABALHOS FUTUROS

Este trabalho teve como objetivo ser um ponto de partida para a análise da utilização do protocolo CAN em nanossatélites, sendo assim, diferentes possibilidades podem ser implementadas em trabalhos futuros, como:

- Implementar um protocolo responsável por fazer o gerenciamento e encapsulamento das mensagens em camadas acima das disponibilizadas pelo CAN (ex. protocolo CSP) e analisar o nível de influência no tempo de resposta do sistema;
- Corrigir o código VHDL do IPCore CAN, visando à utilização na BRAVE;
- Implementar o CAN como forma exclusiva de comunicação do satélite e analisar o quão significativo será o aumento do custo em processamento e em hardware;
- Implementar sistemas tolerantes à falhas nas demais camadas do sistema ISO/OSI;
- Realizar testes de confiabilidade com inserção de erros, utilizando a placa FlatSat, desenvolvida no SpaceLab.

- Buscar uma base de dados robusta sobre satélites e suas falhas, para analisar quão significativas podem ser as falhas nos barramentos de dados.

REFERÊNCIAS

AGRAWAL, V.; MAINI, A. K. **Satellite Technology - Pricnciples and Applications**. Second ed. United Kindom: John Wiley an Sons, LTD., 2011.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 5462: **Confiabilidade e mantenabilidade: Terminologia.** Rio de Janeiro, p. 3. 1994.

BARTOLINI, C; LIPARI, G; ALMEIDA, L; **Using Priority Inheritance Techniques To Override The Size Limit Of Can Messages.** VII IFAC International Conference of Fieldbuses and Networks in Industrial and Embedded Systems (FET). 2007.

BENEVENUTI, F. **Experimental Applications on SRAM-based FPGA for the NanosatC-BR2 Scientific Mission**. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). 2019.

BLANCO, C. D. V.; D'ERRICO, M.; CONTICCHIO, A. **A distributed design architecture for Li-ion battery: Integration of COTS ICs, µCs and CAN bus.** IEEE Aerospace Conference Proceedings, 2006.

BORDASCH, M; GÖHNER, P. **Fault Prevention in Industrial Automation Systems by means of a functional model and a hybrid abnormity identification concept.** 39th Annual Conference of the IEEE Industrial Electronics Society. Pág. 2845 – 2850. 2013

BOSCH, R. **CAN Specification Version 2.0, Parts A and B**, Set. 1991.

BOUWMEESTER, J.; LANGER, M; GILL, E. **Survey on the implementation and reliability of CubeSat electrical bus interfaces.** CEAS Space Journal. 2016.

BRAVHAR, K. et al. **BRAVE NG-MEDIUM FPGA reconfiguration through SpaceWire: Example use case and performance analysis**. 2018 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2018, p. 135–141, 2018.

BROSTER, I.; BURNS, A.; RODRIGUEZ-NAVAS, G; **Probabilistic analysis of CAN with faults**. Proceedings of the 23rd Real-time Systems Symposium. Austin, Texas. Pág. 269–278, 2002.

BROSTER, I.; BURNS, A.; RODRÍGUEZ-NAVAS, G; **Timing analysis of real-time communication under electromagnetic interference.** Real-Time Systems 30**,** pág. 55–81, 2005.

BURJE, P. R.; KARANDE, K. J.; JAGADALE, A. B. **Embedded on-board diagnostics system using CAN network.** Proceedings - 2014 IEEE Global Conference on Wireless Computing and Networking, GCWCN 2014, pág. 31–35, 2014.

COBHAM; **GRLIB IP Core User's Manual.** Versão 2020.2. Jun. 2020.

CUBESAT. **CubeSat Design Specification.** Rev. 13. California. 2014.

DAVIS, R. I; BURNS A; BRIL R. J; LUKKIEN J. J; **Controller area network (CAN)**

schedulability analysis: refuted, revisited and revised, Real-Time Systems, vol. 35, no. 3, pp. 239–272, 2007.

FLORIPASAT. **FloripaSat-1.** Disponível em: < https://floripasat.ufsc.br/pt/home-br/>. Acesso em: 05 mar. 2020.

GAUJAL, B; NAVET, N; **Fault Confinement Mechanisms on CAN: Analysis and Improvements.** IEEE Transactions On Vehicular Technology, v. 54, No. 3, 2005.

GEBELEIN, J.; GUDGE, P.; KEBSCHULL, U. **Preparation of COTS TMS570 MCU for use in ionizing radiation environments.** Proceedings of the European Conference on Radiation and its Effects on Components and Systems, RADECS, V. 2016-Set, pág. 1–4, 2017.

GEORGE, A. D; WILSON, C. M. **Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites**. Proceedings of the IEEE | Vol. 106, No. 3, Mar, 2018.

GOUVEIA JR, K, R; et al. **Payload XL: a new fault-tolerant reconfigurable computing platform for CubeSats.** IV IAA Latin American CubeSat Workshop, pág. 81-91, 2020.

GUPTA, N.; SHAHI, B. **Memory Architecture Design for Nano Satellites.** IEEE Aerospace Conference. Big Sky, MT, USA. 2016.

ISMAIL, E. et al. **The choice of measures reliability of the software for space applications**. 2nd International Conference on Electrical, Communication and Computer Engineering (ICECCE), Istanbul, Turkey, June 2020.

JANSCHEK, K.; BRAUNE, A. **Application of industrial CAN bus technology for LEO-satellites**. Acta Astronautica, v. 46, n. 2, pág. 313–317, 2000.

JAYARATHNE, D. G. N.; JAYANANDA, M. K. **Development of a Field programmable GATE array based computer to controller area network store and forward buffer**. 2014 7th International Conference on Information and Automation for Sustainability, 2014.

JAYARATHNE, N.; JAYANANDA, M. K. **Development of a field programmable gate array based Controller Area Network sniffer.** 2013 IEEE 8th International Conference on Industrial and Information Systems, ICIIS 2013 - Conference Proceedings, pág. 610–615, 2013.

KAHE, G. **Triple-Triple Redundant Reliable Onboard Computer Based on Multicore Microcontrollers**. International Journal of Reliability, Risk and Safety: Theory and Application, v. 1, n. 1, pág. 17–24, 2018.

KHALED, A; ZHANG, Q. **An Energy Aware Mass Memory unit for small satellites using Hybrid Architecture.** IEEE International Conference on Computational Science and Engineering (CSE) e IEEE International Conference on Embedded and Ubiquitous Computing (EUC). 2017.

KHURRAM, M.; ZAIDI, S. M. Y. **CAN as a spacecraft communication bus in LEO satellite mission.** RAST 2005 - Proceedings of 2nd International Conference on Recent Advances in Space Technologies, pág. 432–437, 2005.

KIMM, H.; JARRELL, M. **Controller Area Network For Fault Tolerant Small Satellite System Design.** IEEE 23rd International Symposium on Industrial Electronics (ISIE). Istanbul, Turkey. 2014.

KONECNY, G. **Small Satellites—A Tool for Earth Observation?** Proceedings of 20th ISPRS Congress, pág. 580-582. 2004.

KULYAGIN, V. A., TSAREV R. Y., KOVALEV I. V.; **N-version Design of Fault-Tolerant Control Software for Communications Satellite System**. International Siberian Conference on Control and Communications (SIBCON). 2015.

KUROSE, J; ROSS, K. **Computer Networking: A Top-Down Approach.** 6th Edition. New Jersey. Pearson. Mar, 2012.

LAPRIE, J. C. **Dependability: Basic Concepts and Terminology,** ed. Springer-Verlag, 1992.

LAPRIE, J. C; RANDELL, B; LANDWEHR, C. **Basic Concepts and Taxonomy of Dependable and Secure Computing**. IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, V. 1, No. 1, 2004.

LEPPINEN, H; et. al. **On-Board Data Handling For Ambitious Nanosatellite Missions Using Automotive-Grade Lockstep Microcontrollers**, The 4S Symposium, 2014.

MABROUK E. **What are SmallSats and CubeSats?** NASA**.** 2017. Disponível em: <https://www.nasa.gov/content/what-are-smallsats-and-cubesats> Acesso em: 02/07/2020.

MAINI, A. K.; AGRAWAL, V. **Satellite technology: principles and applications.** United Kingdom: John Wiley & Sons, 2011.

MAMOUTOVA, O.V.; ANTONOV, A. A. **On Design For Reliability Of Electronics In Nanosatellite**. 1st Symposium on Space Educational Activities. Padova. 2015.

MARAGOS, K; et al. E**valuation Methodology and Reconfiguration Tests on the New European NG-MEDIUM FPGA**. Conference on Adaptive Hardware and Systems (AHS). pág. 127-134, 2018.

MARTINS, V; et al. **A dynamic partial reconfiguration designflow for permanent faultsmitigation in FPGAs.** Microelectronics Reliability. pág 50 – 63. 2018.

MIRANDA, E, J; et al. **Quetzal-1 Cubesat's Command And Data Handling - Design And Development Considerations.** IV IAA Latin American CubeSat Workshop, pág. 291-299, 2020.

NANDA, K. G.; RAJ, N. B. **Portable embedded data display and control unit using CAN bus.** Procedia Engineering, v. 38, pág. 791–798, 2012.

NASA. **CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers**. 2017.

PLUMMER, C.; ROOS, P.; STAGNARO, L. **CAN Bus as a Spacecraft Onboard Bus.** European Space Agency, (Special Publication) ESA SP, n. 532, pág. 473–483, 2003.

REGES, J. E.; SANTOS, E. J. P. **A VHDL CAN module for smart sensors**. 4th Southern Conference on Programmable Logic, pág. 179–182, 2008.
SHASHIDHARA, B.; JADHAV S.; KIM, Y. S. **Reconfigurable Fault Tolerant Processor on a SRAM based FPGA.** IEEE International Conference on Electro Information Technology (EIT). Chicago, IL, USA. 2020.

SOUZA, P. N. **Satélites e plataformas digitais**. INPE, p. 68, 2007.

SWARTWOUT, M. **The First One Hundred CubeSats: A Statistical Look**. JoSS, Vol. 2, No. 2, pp. 213-233, 2013.

TEXAS INSTRUMENTS; **Introduction to the Controller Area Network (CAN).** Application Report – 2002. Revised 2016.

THE CUBESAT PROGRAM. **CubeSat Design Specification.** California Polytechnic State University. Rev. 14. 2020.

TINDELL K.W; HANSSON H; WELLINGS A.J**; Analysing real-time communications: Controller Area Network (CAN).** Proceedings 15th real-time systems symposium (RTSS'94). IEEE Computer Society Press, pág 259–263, 1994.

TINDELL, K.W; BURNS A; **Guaranteeing message latencies on Controller Area Network (CAN)**. Proceedings of 1st international CAN conference, pág 1–11, 1994.

TINDELL, K.W; BURNS, A; WELLINGS, A.J; **Calculating Controller area network (CAN) message response times.** Control Engineering Practice, pág 1163–1169, 1995.

WIELANDT, S. et al. **Integration of a CAN bus in an Onboard Computer for Space Applications**. 11th International Conference on Development and Application Systems, pág. 56–59, 2012.

WOODROFFE, A, M.; MADLE, P. **Application and experience of can as a low cost obdh bus system.** MAPLD, 2004.

YUSONG, C. et al. **Research on Software Failure Analysis and Quality Management Model.** 8º IEEE International Conference on Software Quality, Reliability and Security Companion. Pg 94-99. 2018

ZIMMERMANN; H. OSI Reference Model - **The ISO Model of Architecture for Open Systems Interconnection.** IEEE Transactions On Communications, Vol 28, Nº 4. 1980.

APÊNDICE A – Missão GOMX-5

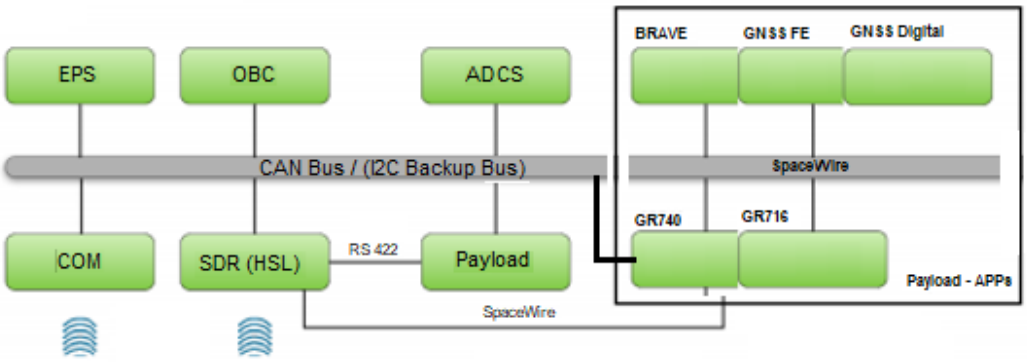
Neste apêndice, será descrito o modelo desenvolvido para utilização com a missão proposta pela empresa GOMSpace, no qual o SpaceLab possui uma carga útil a bordo.

A.1 - GOMX-5

O satélite da missão GOMX-5, com lançamento previsto para 2022, possui dimensões 12U e é composto por, além dos sistemas de controle principais (EPS, OBC, TT&C/COM, etc.), mais algumas cargas úteis que foram desenvolvidas por diferentes organizações parceiras da ESA. O objetivo destas cargas úteis é demonstrar as tecnologias utilizadas atualmente e validar novas tecnologias para diminuir os riscos em constelações de missões futuras.

O SpaceLab formou um consórcio com três empresas europeias que junto desenvolveram uma carga útil para o GOMX-5, as APPs (*Advanced Payload Processor*, em inglês). Na figura A1 são mostradas as cinco placas que formam a carga útil, circuladas pelo retângulo no lado direito da figura. A comunicação entre o APPS e a plataforma de serviço do GOMX-5 é implementada por intermédio do barramento CAN na parte central da figura. A placa GR740 funciona como um gateway CAN, sendo a única que possui conexão com o CAN. A conexão entre as demais placas da carga útil é realizada por meio de interfaces SpaceWire.

Figura A1 – Sistemas de controle e APPs da GOMX-5.



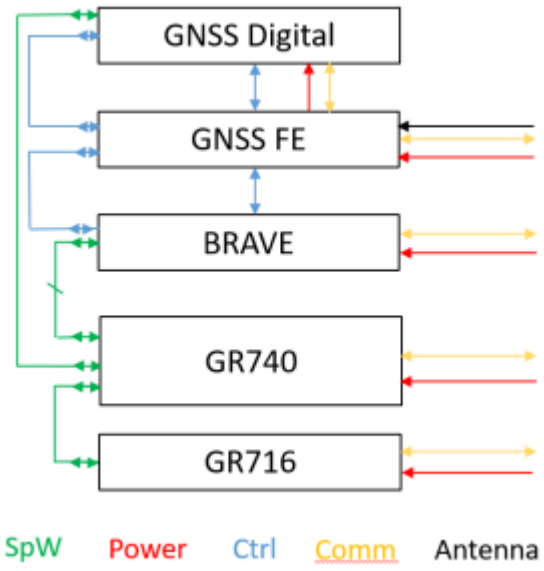
Fonte: Missão GOMX-5.

As placas desenvolvidas para o Sistema de Navegação Global por Satélite (GNSS, *Global Navigation Satellite System*, em inglês), tem seu foco em software e buscam captar os sinais da

terra com o mínimo de hardware possível, utilizando eletrônicos Front-End comerciais e enviando suas informações para os processadores GR740 e para a BRAVE. No projeto das placas GR740 e GR716, o objetivo é validar o microprocessador LEON4FT (na GR740) e o microcontrolador LEON3FT (na GR716), em ambiente radioativo, para demonstrar as características de tolerância a radiação, bem como, os modelos de correção de erro. Por fim, a BRAVE, desenvolvida no SpaceLab, tem como objetivo validar a FPGA BRAVE NX-Large da NanoXplore com intuito de mitigar falhas e também realizar a reconfiguração da FPGA em órbita.

Atualmente, a principal forma de comunicação entre os módulos é SpaceWire, como demonstrado na figura A2. Todos os módulos comunicam com a GR740, responsável por fazer a comunicação entre as Payloads e os sistemas de controle por meio de um barramento CAN.

Figura A2 – Comunicação entre Payloads GOMX-5.



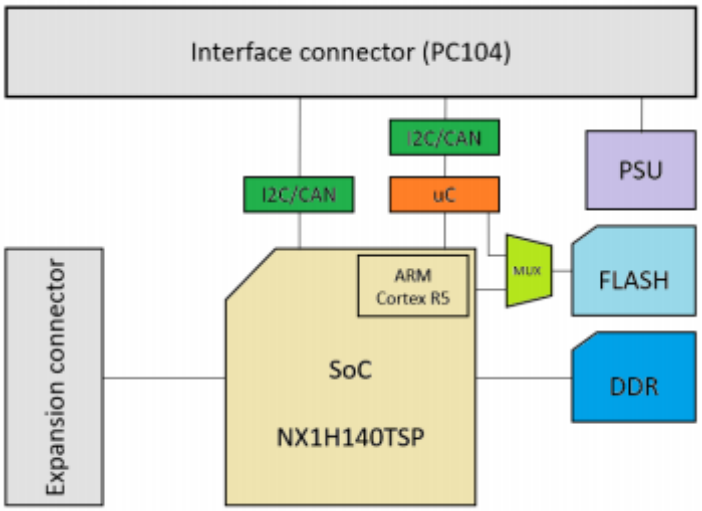
Fonte: Missão GOMX-5.

A.2 PAYLOAD-XL

A Payload-XL é uma evolução de um projeto anterior, o qual baseava-se na reconfiguração, em órbita, da FPGA NX-Medium da NanoXplore. O projeto tem o seu foco no envio de um novo Bit Stream pela estação terrestre e também na validação das características de tolerância à radiação da FPGA. O novo modelo denominado Payload-XL utiliza a NX-Large, uma FPGA que possui mais lógica reconfigurável que o FPGA NX-Medium, além de possuir também um processador ARM Cortex-R5 integrado.

Além da capacidade de ser reconfigurável como sua antecessora, a Payload-XL, está sendo desenvolvida para comportar todos os sistemas principais de um satélite (EPS, OBC, TT&C, etc.) em uma única placa. Na figura A3 é demonstrado o modelo de hardware proposto para o projeto.

Figura A3 – Diagrama de blocos da Payload –X.

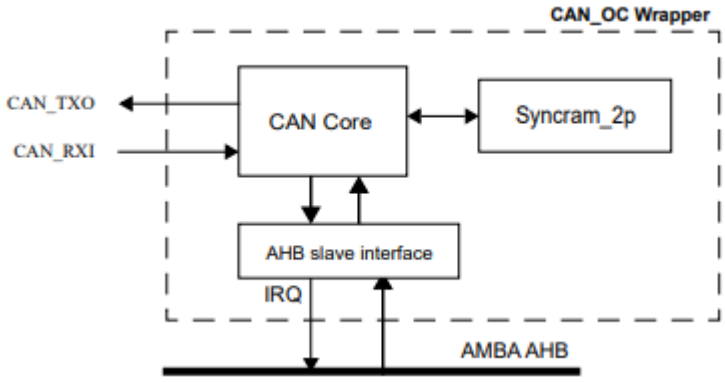


Fonte: Projeto Payload-XL.

A.3 IPCORE CAN

O IPCore escolhido para este projeto foi desenvolvido pela Cobham Gaisler denominado CAN_OC, que por sua vez utiliza um CAN Core da OpenCores, uma comunidade de desenvolvimento de IPCores de código aberto. O CAN_OC é baseado no controlador CAN SJA1000 da Philips e foi escolhido por suportar ambas as versões CAN 2.0A e CAN 2.0B, além de possuir interface para *advanced microcontroller bus architecture* (AMBA) *advanced high performance bus* (AHB), denominado AHB slave interface e um IPcore capaz de portar diferentes modelos de memórias RAM para funcionar como a memória FIFO (*First-In First-Out*) do controlador, denominado Syncram_2p, como demonstra a figura A4.

Figura A4 – CAN_OC Gaisler.



Fonte: COBHAM, 2020.

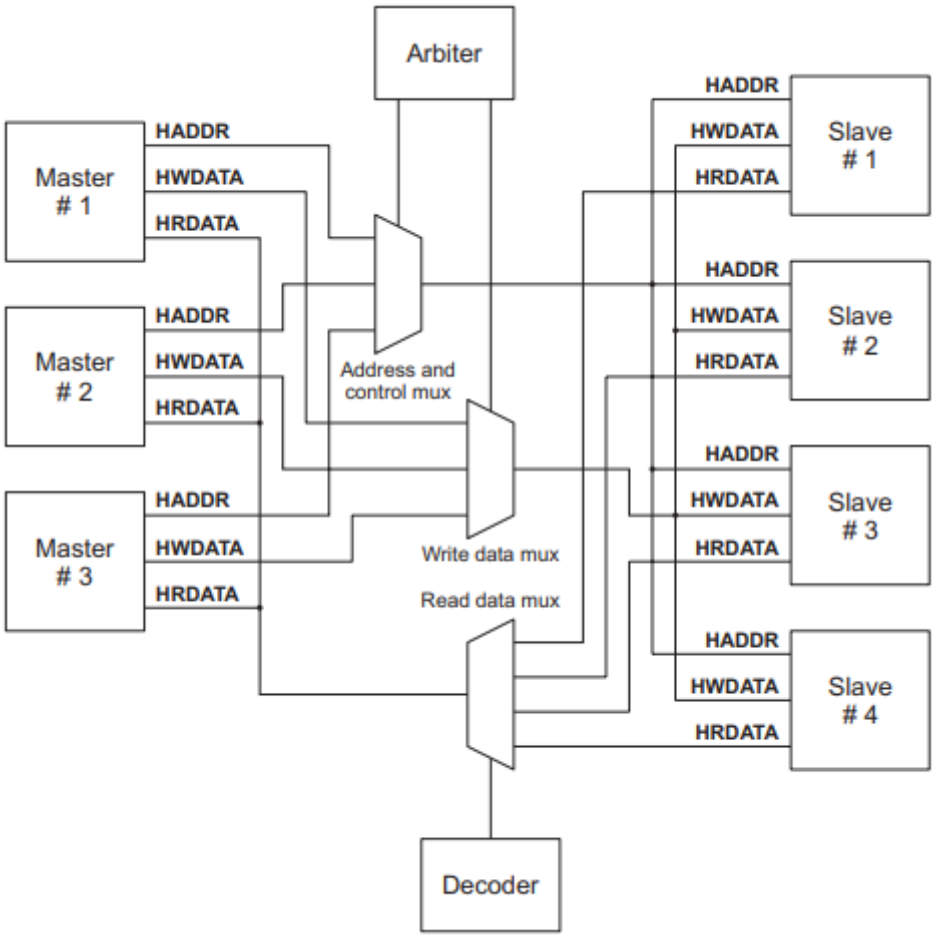
A.3.1 - AMBA AHB

AHB é um modo de transferência de informações de alta performance, de acordo com as especificações técnicas disponibilizadas pela ARM (1999), AHB possui diversas características como:

- Transferência em Burst
- Transações com divisão
- Operação em uma única borda de Clock
- Configurações de barramento de dados mais ampla (até 128 bits)

O protocolo AMBA AHB é projetado com um multiplexador central, assim todos os mestres de barramento conduzem os sinais de endereço e controle indicando a transferência que desejam realizar e o árbitro determina qual mestre terá suas informações enviadas para todos os escravos. O AMBA AHB também necessita de um decodificador central que tem a função de selecionar a informação advinda do escravo envolvido na transferência de dados. Este esquema é mostrado na figura A5.

Figura A5 – Esquema AMBA AHB.



Fonte: ARM, 1999.

A.3.2 - Syncram_2P

O Syncram_2p é um IPCore em VHDL de uma memória RAM de duas portas com uma porta de escrita e uma porta de leitura e com barramento de endereço e de dados separado, desenvolvido pela Gaisler com o intuito de tornar o seu projeto adaptável a diferentes tecnologias. As memórias suportadas são demonstradas no quadro A1.

Quadro A1 – Tecnologias de memórias RAM aceitas pela Syncram_2p.

Tech name	Technology	RAM cell	abit range	dbit range
Inferred	Behavioural description	Tool dependent	unlimited	unlimited
altera	All Altera devices	altsyncram	umlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2, virtex4, virtex5, spartan3, spartan6, virtex7, kintex7, artix7, zynq7000, kintexu	Xilinx Virtex2/4/5/6, Spartan3/3a/3e/6, 7-Series, Ultras-cale	RAMB16_Sn	2 - 14	unlimited
axcel / axdsp	Actel AX, RTAX and RTAX-DSP	RAM64K36	2 - 12	unlimited
proasic	Actel Proasic	RAM256x9SST	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9, ram512x18	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
memartisan	Artisan ASIC RAM	rf2_256x32m4 rf2_512x32m4	8 - 9	32
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram	2 - 12	unlimited
easic45	eASIC 45 nm Nextreme2	bRAM, rFile	unlimited	unlimited
igloo2 / smartfusion2	Microsemi IGLOO2 / SmartFusion2	RAM1K18	2 - 14	unlimited
rtg4	Microsemi RTG4	RAM1K18_RT	2 - 16	unlimited

Fonte: GRLIB, 2020.

APÊNDICE B – Quadros CAN FloripaSat

Mensagem.Quadro	Periodo	Bytes	Jitter	Tempo de envio	Tempo Bloqueado	Ri	Prob
1.1	60	8	0.000000	0.000135	0.000135	0.000270	0.991932717
	1 erro					0.000436	0.007994742
	2 erros					0.000602	0.000071834
	3 erros					0.000768	0.000000700
	4 erros					0.000934	0.000000007
1.2	60	8	0.000000	0.000135	0.000135	0.000405	0.987923513
	1 erro					0.000571	0.011943643
	2 erros					0.000737	0.000131381
	3 erros					0.000903	0.000001446
	4 erros					0.001069	0.000000016
1.3	60	8	0.000000	0.000135	0.000135	0.000540	0.983930514
	1 erro					0.000706	0.015860492
	2 erros					0.000872	0.000206425
	3 erros					0.001038	0.000002538
	4 erros					0.001204	0.000000031
1.4	60	8	0.000000	0.000135	0.000135	0.000675	0.979953654
	1 erro					0.000841	0.019745484
	2 erros					0.001007	0.000296774
	3 erros					0.001173	0.000004035
	4 erros					0.001339	0.000000053
1.5	60	8	0.000000	0.000135	0.000135	0.000810	0.975992868
	1 erro					0.000976	0.023598811
	2 erros					0.001142	0.000402239
	3 erros					0.001308	0.000005996
	4 erros					0.001474	0.000000084
1.6	60	8	0.000000	0.000135	0.000135	0.000945	0.972048090
	1 erro					0.001111	0.027420668
	2 erros					0.001277	0.000522634
	3 erros					0.001443	0.000008480
	4 erros					0.001609	0.000000127
1.7	60	8	0.000000	0.000135	0.000135	0.001080	0.968119257
	1 erro					0.001246	0.031211244
	2 erros					0.001412	0.000657770
	3 erros					0.001578	0.000011542
	4 erros					0.001744	0.000000183
1.8	60	8	0.000000	0.000135	0.000135	0.001215	0.964206303
	1 erro					0.001381	0.034970731
	2 erros					0.001547	0.000807465
	3 erros					0.001713	0.000015240
	4 erros					0.001879	0.000000257

1.9	60	8	0.000000	0.000135	0.000135	0.001350	0.960309165
	1 erro					0.001516	0.038699318
	2 erros					0.001682	0.000971533
	3 erros					0.001848	0.000019628
	4 erros					0.002014	0.000000350
1.10	60	8	0.000000	0.000135	0.000135	0.001485	0.956427778
	1 erro					0.001651	0.042397193
	2 erros					0.001817	0.001149795
	3 erros					0.001983	0.000024760
	4 erros					0.002149	0.000000466
1.11	60	8	0.000000	0.000135	0.000135	0.001620	0.952562078
	1 erro					0.001786	0.046064543
	2 erros					0.001952	0.001342070
	3 erros					0.002118	0.000030689
	4 erros					0.002284	0.000000608
1.12	60	8	0.000000	0.000135	0.000135	0.001755	0.948712004
	1 erro					0.001921	0.049701556
	2 erros					0.002087	0.001548178
	3 erros					0.002253	0.000037468
	4 erros					0.002419	0.000000780
1.13	60	8	0.000000	0.000135	0.000135	0.001890	0.944877490
	1 erro					0.002056	0.053308416
	2 erros					0.002222	0.001767943
	3 erros					0.002388	0.000045147
	4 erros					0.002554	0.000000984
1.14	60	8	0.000000	0.000135	0.000135	0.002025	0.941058475
	1 erro					0.002191	0.056885307
	2 erros					0.002357	0.002001189
	3 erros					0.002523	0.000053777
	4 erros					0.002689	0.000001226
1.15	60	8	0.000000	0.000135	0.000135	0.002160	0.937254896
	1 erro					0.002326	0.060432413
	2 erros					0.002492	0.002247742
	3 erros					0.002658	0.000063408
	4 erros					0.002824	0.000001508
1.16	60	8	0.000000	0.000135	0.000135	0.002295	0.933466690
	1 erro					0.002461	0.063949917
	2 erros					0.002627	0.002507428
	3 erros					0.002793	0.000074088
	4 erros					0.002959	0.000001836
1.17	60	8	0.000000	0.000135	0.000135	0.002430	0.929693795
	1 erro					0.002596	0.067437999
	2 erros					0.002762	0.002780077
	3 erros					0.002928	0.000085864
	4 erros					0.003094	0.000002214

1.18	60	8	0.000000	0.000135	0.000135	0.002565	0.925936149
	1 erro					0.002731	0.070896840
	2 erros					0.002897	0.003065518
	3 erros					0.003063	0.000098784
	4 erros					0.003229	0.000002645
1.19	60	8	0.000000	0.000135	0.000135	0.002700	0.922193691
	1 erro					0.002866	0.074326619
	2 erros					0.003032	0.003363582
	3 erros					0.003198	0.000112893
	4 erros					0.003364	0.000003136
1.20	60	8	0.000000	0.000135	0.000135	0.002835	0.918466360
	1 erro					0.003001	0.077727515
	2 erros					0.003167	0.003674103
	3 erros					0.003333	0.000128236
	4 erros					0.003499	0.000003690
1.21	60	8	0.000000	0.000135	0.000135	0.002970	0.914754094
	1 erro					0.003136	0.081099706
	2 erros					0.003302	0.003996914
	3 erros					0.003468	0.000144858
	4 erros					0.003634	0.000004314
1.22	60	8	0.000000	0.000135	0.000135	0.003105	0.911056832
	1 erro					0.003271	0.084443367
	2 erros					0.003437	0.004331851
	3 erros					0.003603	0.000162802
	4 erros					0.003769	0.000005011
1.23	60	8	0.000000	0.000135	0.000135	0.003240	0.907374513
	1 erro					0.003406	0.087758674
	2 erros					0.003572	0.004678751
	3 erros					0.003738	0.000182110
	4 erros					0.003904	0.000005787
1.24	60	8	0.000000	0.000135	0.000135	0.003375	0.903707078
	1 erro					0.003541	0.091045803
	2 erros					0.003707	0.005037453
	3 erros					0.003873	0.000202825
	4 erros					0.004039	0.000006649
1.25	60	8	0.000000	0.000135	0.000135	0.003510	0.900054466
	1 erro					0.003676	0.094304925
	2 erros					0.003842	0.005407795
	3 erros					0.004008	0.000224986
	4 erros					0.004174	0.000007601
1.26	60	8	0.000000	0.000135	0.000135	0.003645	0.896416617
	1 erro					0.003811	0.097536215
	2 erros					0.003977	0.005789619
	3 erros					0.004143	0.000248635
	4 erros					0.004309	0.000008649

1.27	60	8	0.000000	0.000135	0.000135	0.003780	0.892793471
	1 erro					0.003946	0.100739844
	2 erros					0.004112	0.006182767
	3 erros					0.004278	0.000273810
	4 erros					0.004444	0.000009799
1.28	60	8	0.000000	0.000135	0.000135	0.003915	0.889184970
	1 erro					0.004081	0.103915982
	2 erros					0.004247	0.006587082
	3 erros					0.004413	0.000300549
	4 erros					0.004579	0.000011058
1.29	60	8	0.000000	0.000135	0.000135	0.004050	0.885591053
	1 erro					0.004216	0.107064800
	2 erros					0.004382	0.007002410
	3 erros					0.004548	0.000328892
	4 erros					0.004714	0.000012430
1.30	60	8	0.000000	0.000135	0.000135	0.004185	0.882011663
	1 erro					0.004351	0.110186467
	2 erros					0.004517	0.007428597
	3 erros					0.004683	0.000358874
	4 erros					0.004849	0.000013924
1.31	60	8	0.000000	0.000135	0.000135	0.004320	0.878446739
	1 erro					0.004486	0.113281150
	2 erros					0.004652	0.007865491
	3 erros					0.004818	0.000390532
	4 erros					0.004984	0.000015544
1.32	60	8	0.000000	0.000135	0.000135	0.004455	0.874896225
	1 erro					0.004621	0.116349017
	2 erros					0.004787	0.008312939
	3 erros					0.004953	0.000423902
	4 erros					0.005119	0.000017297
1.33	60	8	0.000000	0.000135	0.000135	0.004590	0.871360061
	1 erro					0.004756	0.119390234
	2 erros					0.004922	0.008770793
	3 erros					0.005088	0.000459017
	4 erros					0.005254	0.000019191
1.34	60	8	0.000000	0.000135	0.000135	0.004725	0.867838189
	1 erro					0.004891	0.122404966
	2 erros					0.005057	0.009238904
	3 erros					0.005223	0.000495912
	4 erros					0.005389	0.000021232
1.35	60	8	0.000000	0.000135	0.000135	0.004860	0.864330552
	1 erro					0.005026	0.125393378
	2 erros					0.005192	0.009717124
	3 erros					0.005358	0.000534621
	4 erros					0.005524	0.000023426

1.36	60	8	0.000000	0.000135	0.000135	0.004995	0.860837092
	1 erro					0.005161	0.128355633
	2 erros					0.005327	0.010205308
	3 erros					0.005493	0.000575175
	4 erros					0.005659	0.000025781
1.37	60	8	0.000000	0.000135	0.000135	0.005130	0.857357753
	1 erro					0.005296	0.131291894
	2 erros					0.005462	0.010703309
	3 erros					0.005628	0.000617606
	4 erros					0.005794	0.000028303
1.38	60	8	0.000000	0.000135	0.000135	0.005265	0.853892476
	1 erro					0.005431	0.134202322
	2 erros					0.005597	0.011210986
	3 erros					0.005763	0.000661947
	4 erros					0.005929	0.000031000
1.39	60	8	0.000000	0.000135	0.000135	0.005400	0.850441205
	1 erro					0.005566	0.137087079
	2 erros					0.005732	0.011728195
	3 erros					0.005898	0.000708226
	4 erros					0.006064	0.000033880
1.40	60	8	0.000000	0.000135	0.000135	0.005535	0.847003883
	1 erro					0.005701	0.139946324
	2 erros					0.005867	0.012254795
	3 erros					0.006033	0.000756474
	4 erros					0.006199	0.000036948
1.41	60	8	0.000000	0.000135	0.000135	0.005670	0.843580454
	1 erro					0.005836	0.142780216
	2 erros					0.006002	0.012790647
	3 erros					0.006168	0.000806719
	4 erros					0.006334	0.000040213
1.42	60	8	0.000000	0.000135	0.000135	0.005805	0.840170863
	1 erro					0.005971	0.145588915
	2 erros					0.006137	0.013335611
	3 erros					0.006303	0.000858991
	4 erros					0.006469	0.000043682
1.43	60	8	0.000000	0.000135	0.000135	0.005940	0.836775052
	1 erro					0.006106	0.148372576
	2 erros					0.006272	0.013889549
	3 erros					0.006438	0.000913316
	4 erros					0.006604	0.000047363
1.44	60	8	0.000000	0.000135	0.000135	0.006075	0.833392966
	1 erro					0.006241	0.151131357
	2 erros					0.006407	0.014452327
	3 erros					0.006573	0.000969723
	4 erros					0.006739	0.000051263

1.45	60	8	0.000000	0.000135	0.000135	0.006210	0.830024550
	1 erro					0.006376	0.153865413
	2 erros					0.006542	0.015023808
	3 erros					0.006708	0.001028236
	4 erros					0.006874	0.000055390
1.46	60	8	0.000000	0.000135	0.000135	0.006345	0.826669749
	1 erro					0.006511	0.156574899
	2 erros					0.006677	0.015603858
	3 erros					0.006843	0.001088882
	4 erros					0.007009	0.000059752
1.47	60	8	0.000000	0.000135	0.000135	0.006480	0.823328507
	1 erro					0.006646	0.159259969
	2 erros					0.006812	0.016192345
	3 erros					0.006978	0.001151686
	4 erros					0.007144	0.000064357
1.48	60	8	0.000000	0.000135	0.000135	0.006615	0.820000770
	1 erro					0.006781	0.161920776
	2 erros					0.006947	0.016789136
	3 erros					0.007113	0.001216673
	4 erros					0.007279	0.000069212
1.49	60	8	0.000000	0.000135	0.000135	0.006750	0.816686483
	1 erro					0.006916	0.164557473
	2 erros					0.007082	0.017394102
	3 erros					0.007248	0.001283866
	4 erros					0.007414	0.000074326
1.50	60	8	0.000000	0.000135	0.000135	0.006885	0.813385591
	1 erro					0.007051	0.167170210
	2 erros					0.007217	0.018007112
	3 erros					0.007383	0.001353288
	4 erros					0.007549	0.000079706
1.51	60	8	0.000000	0.000135	0.000135	0.007020	0.810098041
	1 erro					0.007186	0.169759138
	2 erros					0.007352	0.018628039
	3 erros					0.007518	0.001424963
	4 erros					0.007684	0.000085361
1.52	60	8	0.000000	0.000135	0.000135	0.007155	0.806823779
	1 erro					0.007321	0.172324408
	2 erros					0.007487	0.019256755
	3 erros					0.007653	0.001498911
	4 erros					0.007819	0.000091299
1.53	60	8	0.000000	0.000135	0.000135	0.007290	0.803562751
	1 erro					0.007456	0.174866168
	2 erros					0.007622	0.019893134
	3 erros					0.007788	0.001575155
	4 erros					0.007954	0.000097527

1.54	60	8	0.000000	0.000135	0.000135	0.007425	0.800314903
	1 erro					0.007591	0.177384566
	2 erros					0.007757	0.020537052
	3 erros					0.007923	0.001653716
	4 erros					0.008089	0.000104055
1.55	60	8	0.000000	0.000135	0.000135	0.007560	0.797080182
	1 erro					0.007726	0.179879750
	2 erros					0.007892	0.021188383
	3 erros					0.008058	0.001734613
	4 erros					0.008224	0.000110891
1.56	60	8	0.000000	0.000135	0.000135	0.007695	0.793858536
	1 erro					0.007861	0.182351866
	2 erros					0.008027	0.021847007
	3 erros					0.008193	0.001817866
	4 erros					0.008359	0.000118042
1.57	60	8	0.000000	0.000135	0.000135	0.007830	0.790649911
	1 erro					0.007996	0.184801059
	2 erros					0.008162	0.022512800
	3 erros					0.008328	0.001903494
	4 erros					0.008494	0.000125518
1.58	60	8	0.000000	0.000135	0.000135	0.007965	0.787454254
	1 erro					0.008131	0.187227476
	2 erros					0.008297	0.023185643
	3 erros					0.008463	0.001991517
	4 erros					0.008629	0.000133326
1.59	60	8	0.000000	0.000135	0.000135	0.008100	0.784271514
	1 erro					0.008266	0.189631259
	2 erros					0.008432	0.023865415
	3 erros					0.008598	0.002081951
	4 erros					0.008764	0.000141475
1.60	60	8	0.000000	0.000135	0.000135	0.008235	0.781101637
	1 erro					0.008401	0.192012552
	2 erros					0.008567	0.024551999
	3 erros					0.008733	0.002174815
	4 erros					0.008899	0.000149974
1.61	60	8	0.000000	0.000135	0.000135	0.008370	0.777944573
	1 erro					0.008536	0.194371498
	2 erros					0.008702	0.025245277
	3 erros					0.008868	0.002270124
	4 erros					0.009034	0.000158832
1.62	60	7	0.000000	0.000125	0.000135	0.008495	0.775032744
	1 erro					0.008661	0.196535905
	2 erros					0.008827	0.025893067
	3 erros					0.008993	0.002360569
	4 erros					0.009159	0.000167359

2.1	10	8	0.000000	0.000135	0.000135	0.008630	0.771900209
	1 erro					0.008796	0.198852210
	2 erros					0.008962	0.026598910
	3 erros					0.009128	0.002460636
	4 erros					0.009294	0.000176930
2.2	10	8	0.000000	0.000135	0.000135	0.008765	0.768780335
	1 erro					0.008931	0.201146581
	2 erros					0.009097	0.027311108
	3 erros					0.009263	0.002563195
	4 erros					0.009429	0.000186885
2.3	10	8	0.000000	0.000135	0.000135	0.008900	0.765673071
	1 erro					0.009066	0.203419156
	2 erros					0.009232	0.028029549
	3 erros					0.009398	0.002668262
	4 erros					0.009564	0.000197231
2.4	10	7	0.000000	0.000125	0.000135	0.009025	0.762807174
	1 erro					0.009191	0.205504080
	2 erros					0.009357	0.028700241
	3 erros					0.009523	0.002767794
	4 erros					0.009689	0.000207167
3.1	10	8	0.000000	0.000135	0.000135	0.009160	0.759724053
	1 erro					0.009326	0.207735069
	2 erros					0.009492	0.029430390
	3 erros					0.009658	0.002877729
	4 erros					0.009824	0.000218293
3.2	10	8	0.000000	0.000135	0.000135	0.009295	0.756653393
	1 erro					0.009461	0.209944666
	2 erros					0.009627	0.030166456
	3 erros					0.009793	0.002990212
	4 erros					0.009959	0.000229835
3.3	10	8	0.000000	0.000135	0.000135	0.009430	0.753595144
	1 erro					0.009596	0.212133009
	2 erros					0.009762	0.030908329
	3 erros					0.009928	0.003105256
	4 erros					0.010094	0.000241803
3.4	10	8	0.000000	0.000135	0.000135	0.009565	0.750549255
	1 erro					0.009731	0.214300232
	2 erros					0.009897	0.031655901
	3 erros					0.010063	0.003222874
	4 erros					0.010229	0.000254206
3.5	10	8	0.000000	0.000135	0.000135	0.009700	0.747515678
	1 erro					0.009866	0.216446470
	2 erros					0.010032	0.032409065
	3 erros					0.010198	0.003343076
	4 erros					0.010364	0.000267053

3.6	10	8	0.000000	0.000135	0.000135	0.009835	0.744494362
	1 erro					0.010001	0.218571859
	2 erros					0.010167	0.033167714
	3 erros					0.010333	0.003465874
	4 erros					0.010499	0.000280351
3.7	10	2	0.000000	0.000075	0.000135	0.009910	0.742821133
	1 erro					0.010076	0.219743670
	2 erros					0.010242	0.033591517
	3 erros					0.010408	0.003535221
	4 erros					0.010574	0.000287938
4.1	1	8	0.000000	0.000135	0.000135	0.010045	0.739818791
	1 erro					0.010211	0.221836890
	2 erros					0.010377	0.034358489
	3 erros					0.010543	0.003662080
	4 erros					0.010709	0.000301957
4.2	1	8	0.000000	0.000135	0.000135	0.010180	0.736828584
	1 erro					0.010346	0.223909599
	2 erros					0.010512	0.035130678
	3 erros					0.010678	0.003791563
	4 erros					0.010844	0.000316450
4.3	1	8	0.000000	0.000135	0.000135	0.010315	0.733850463
	1 erro					0.010481	0.225961929
	2 erros					0.010647	0.035907982
	3 erros					0.010813	0.003923678
	4 erros					0.010979	0.000331427
4.4	1	8	0.000000	0.000135	0.000135	0.010450	0.730884379
	1 erro					0.010616	0.227994011
	2 erros					0.010782	0.036690298
	3 erros					0.010948	0.004058436
	4 erros					0.011114	0.000346896
4.5	1	8	0.000000	0.000135	0.000135	0.010585	0.727930283
	1 erro					0.010751	0.230005975
	2 erros					0.010917	0.037477525
	3 erros					0.011083	0.004195845
	4 erros					0.011249	0.000362865
4.6	1	8	0.000000	0.000135	0.000135	0.010720	0.724988128
	1 erro					0.010886	0.231997950
	2 erros					0.011052	0.038269562
	3 erros					0.011218	0.004335914
	4 erros					0.011384	0.000379344
4.7	1	8	0.000000	0.000135	0.000135	0.010855	0.722057863
	1 erro					0.011021	0.233970066
	2 erros					0.011187	0.039066311
	3 erros					0.011353	0.004478651
	4 erros					0.011519	0.000396341

4.8	1	8	0.000000	0.000135	0.000135	0.010990	0.719139443
	1 erro					0.011156	0.235922450
	2 erros					0.011322	0.039867674
	3 erros					0.011488	0.004624064
	4 erros					0.011654	0.000413865
4.9	1	5	0.000000	0.000105	0.000135	0.011095	0.716877718
	1 erro					0.011261	0.237427411
	2 erros					0.011427	0.040494082
	3 erros					0.011593	0.004739016
	4 erros					0.011759	0.000427864
5.1	10	8	0.000000	0.000135	0.000135	0.011230	0.713980234
	1 erro					0.011396	0.239345030
	2 erros					0.011562	0.041303405
	3 erros					0.011728	0.004889202
	4 erros					0.011894	0.000446347
5.2	10	8	0.000000	0.000135	0.000135	0.011365	0.711094462
	1 erro					0.011531	0.241243269
	2 erros					0.011697	0.042117072
	3 erros					0.011863	0.005042083
	4 erros					0.012029	0.000465380
5.3	10	8	0.000000	0.000135	0.000135	0.011500	0.708220353
	1 erro					0.011666	0.243122253
	2 erros					0.011832	0.042934988
	3 erros					0.011998	0.005197663
	4 erros					0.012164	0.000484974
5.4	10	4	0.000000	0.000095	0.000135	0.011595	0.706204799
	1 erro					0.011761	0.244433027
	2 erros					0.011927	0.043513055
	3 erros					0.012093	0.005308767
	4 erros					0.012259	0.000499102
6.1	30	8	0.000000	0.000135	0.000135	0.011730	0.703350454
	1 erro					0.011896	0.246279494
	2 erros					0.012062	0.044337995
	3 erros					0.012228	0.005468961
	4 erros					0.012394	0.000519669
6.2	30	8	0.000000	0.000135	0.000135	0.011865	0.700507645
	1 erro					0.012031	0.248107041
	2 erros					0.012197	0.045166931
	3 erros					0.012363	0.005631869
	4 erros					0.012529	0.000540820
6.3	30	8	0.000000	0.000135	0.000135	0.012000	0.697676326
	1 erro					0.012166	0.249915793
	2 erros					0.012332	0.045999773
	3 erros					0.012498	0.005797496
	4 erros					0.012664	0.000562562

6.4	30	8	0.000000	0.000135	0.000135	0.012135	0.694856451
	1 erro					0.012301	0.251705869
	2 erros					0.012467	0.046836429
	3 erros					0.012633	0.005965846
	4 erros					0.012799	0.000584905
6.5	30	8	0.000000	0.000135	0.000135	0.012270	0.692047973
	1 erro					0.012436	0.253477393
	2 erros					0.012602	0.047676809
	3 erros					0.012768	0.006136923
	4 erros					0.012934	0.000607858
7.1	30	8	0.000000	0.000135	0.000135	0.012405	0.689250847
	1 erro					0.012571	0.255230485
	2 erros					0.012737	0.048520824
	3 erros					0.012903	0.006310730
	4 erros					0.013069	0.000631428
7.2	30	8	0.000000	0.000135	0.000135	0.012540	0.686465026
	1 erro					0.012706	0.256965264
	2 erros					0.012872	0.049368385
	3 erros					0.013038	0.006487271
	4 erros					0.013204	0.000655624
7.3	30	8	0.000000	0.000135	0.000135	0.012675	0.683690465
	1 erro					0.012841	0.258681851
	2 erros					0.013007	0.050219406
	3 erros					0.013173	0.006666548
	4 erros					0.013339	0.000680456
7.4	30	8	0.000000	0.000135	0.000135	0.012810	0.680927118
	1 erro					0.012976	0.260380363
	2 erros					0.013142	0.051073799
	3 erros					0.013308	0.006848564
	4 erros					0.013474	0.000705931
7.5	30	8	0.000000	0.000135	0.000135	0.012945	0.678174940
	1 erro					0.013111	0.262060920
	2 erros					0.013277	0.051931479
	3 erros					0.013443	0.007033319
	4 erros					0.013609	0.000732057
7.6	30	8	0.000000	0.000135	0.000135	0.013080	0.675433886
	1 erro					0.013246	0.263723638
	2 erros					0.013412	0.052792360
	3 erros					0.013578	0.007220817
	4 erros					0.013744	0.000758844
7.7	30	8	0.000000	0.000135	0.000135	0.013215	0.672703911
	1 erro					0.013381	0.265368634
	2 erros					0.013547	0.053656358
	3 erros					0.013713	0.007411058
	4 erros					0.013879	0.000786300

7.8	30	8	0.000000	0.000135	0.000135	0.013350	0.669984970
	1 erro					0.013516	0.266996024
	2 erros					0.013682	0.054523390
	3 erros					0.013848	0.007604043
	4 erros					0.014014	0.000814433
7.9	30	8	0.000000	0.000135	0.000135	0.013485	0.667277018
	1 erro					0.013651	0.268605924
	2 erros					0.013817	0.055393374
	3 erros					0.013983	0.007799773
	4 erros					0.014149	0.000843251
7.10	30	8	0.000000	0.000135	0.000135	0.013620	0.664580011
	1 erro					0.013786	0.270198448
	2 erros					0.013952	0.056266227
	3 erros					0.014118	0.007998248
	4 erros					0.014284	0.000872763
7.11	30	8	0.000000	0.000135	0.000135	0.013755	0.661893905
	1 erro					0.013921	0.271773711
	2 erros					0.014087	0.057141868
	3 erros					0.014253	0.008199468
	4 erros					0.014419	0.000902978
7.12	30	8	0.000000	0.000135	0.000135	0.013890	0.659218656
	1 erro					0.014056	0.273331826
	2 erros					0.014222	0.058020217
	3 erros					0.014388	0.008403433
	4 erros					0.014554	0.000933902
7.13	30	8	0.000000	0.000135	0.000135	0.014025	0.656554219
	1 erro					0.014191	0.274872907
	2 erros					0.014357	0.058901195
	3 erros					0.014523	0.008610142
	4 erros					0.014689	0.000965546
7.14	30	8	0.000000	0.000135	0.000135	0.014160	0.653900552
	1 erro					0.014326	0.276397064
	2 erros					0.014492	0.059784723
	3 erros					0.014658	0.008819593
	4 erros					0.014824	0.000997916
7.15	30	8	0.000000	0.000135	0.000135	0.014295	0.651257610
	1 erro					0.014461	0.277904410
	2 erros					0.014627	0.060670723
	3 erros					0.014793	0.009031786
	4 erros					0.014959	0.001031020
7.16	30	8	0.000000	0.000135	0.000135	0.014430	0.648625351
	1 erro					0.014596	0.279395057
	2 erros					0.014762	0.061559118
	3 erros					0.014928	0.009246719
	4 erros					0.015094	0.001064868

7.17	30	8	0.000000	0.000135	0.000135	0.014565	0.646003731
	1 erro					0.014731	0.280869113
	2 erros					0.014897	0.062449832
	3 erros					0.015063	0.009464390
	4 erros					0.015229	0.001099467
7.18	30	8	0.000000	0.000135	0.000135	0.014700	0.643392707
	1 erro					0.014866	0.282326690
	2 erros					0.015032	0.063342788
	3 erros					0.015198	0.009684797
	4 erros					0.015364	0.001134825
7.19	30	8	0.000000	0.000135	0.000135	0.014835	0.640792236
	1 erro					0.015001	0.283767896
	2 erros					0.015167	0.064237912
	3 erros					0.015333	0.009907937
	4 erros					0.015499	0.001170950
7.20	30	8	0.000000	0.000135	0.000135	0.014970	0.638202275
	1 erro					0.015136	0.285192839
	2 erros					0.015302	0.065135130
	3 erros					0.015468	0.010133807
	4 erros					0.015634	0.001207850
7.21	30	8	0.000000	0.000135	0.000135	0.015105	0.635622783
	1 erro					0.015271	0.286601628
	2 erros					0.015437	0.066034369
	3 erros					0.015603	0.010362405
	4 erros					0.015769	0.001245533
7.22	30	8	0.000000	0.000135	0.000135	0.015240	0.633053717
	1 erro					0.015406	0.287994370
	2 erros					0.015572	0.066935555
	3 erros					0.015738	0.010593727
	4 erros					0.015904	0.001284007
7.23	30	8	0.000000	0.000135	0.000135	0.015375	0.630495034
	1 erro					0.015541	0.289371172
	2 erros					0.015707	0.067838616
	3 erros					0.015873	0.010827769
	4 erros					0.016039	0.001323279
7.24	30	8	0.000000	0.000135	0.000065	0.015440	0.629266767
	1 erro					0.015606	0.290028422
	2 erros					0.015772	0.068274071
	3 erros					0.015938	0.010941425
	4 erros					0.016104	0.001342474
7.25	30	1	0.000000	0.000065	0.000000	0.015440	0.629266767
	1 erro					0.015606	0.290028422
	2 erros					0.015772	0.068274071
	3 erros					0.015938	0.010941425
	4 erros					0.016104	0.001342474