



UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA
MESTRADO ACADÊMICO EM ENGENHARIA DE TELEINFORMÁTICA

DAVID FREITAS MOURA MOTA

**OPENOBC: UMA ARQUITETURA DE UM COMPUTADOR DE BORDO OPEN
SOURCE E BAIXO CUSTO PARA O PADRÃO CUBESAT**

FORTALEZA

2017

DAVID FREITAS MOURA MOTA

OPENOBC: UMA ARQUITETURA DE UM COMPUTADOR DE BORDO OPEN SOURCE E
BAIXO CUSTO PARA O PADRÃO CUBESAT

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia de Teleinformática do Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia de Teleinformática. Área de Concentração: Sinais e Sistemas

Orientador: Prof. Dr. João Cesar Moura Mota

Co-Orientador: Prof. Dr. Jarbas Aryel Nunes da Silveira

FORTALEZA

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F936o Freitas Moura Mota, David.
Openobc: uma arquitetura de um computador de bordo open source e de baixo custo para o padrão Cubesat / David Freitas Moura Mota. – 2017.
75 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2017.

Orientação: Prof. Dr. João Cesar Moura Mota.

Coorientação: Prof. Dr. Jarbas Aryel Nunes da Silveira.

1. CubeSat. 2. Computador de Bordo. 3. Arquitetura. 4. Hardware. 5. Processador. I. Título.

CDD 621.38

DAVID FREITAS MOURA MOTA

OPENOBC: UMA ARQUITETURA DE UM COMPUTADOR DE BORDO OPEN SOURCE E
BAIXO CUSTO PARA O PADRÃO CUBESAT

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia de Teleinformática do Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia de Teleinformática. Área de Concentração: Sinais e Sistemas

Aprovada em: 27/01/2017

BANCA EXAMINADORA

Prof. Dr. João Cesar Moura Mota (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo César Cortez
Universidade Federal do Ceará (UFC)

Prof. Dr. Fabian Luís Vargas
Pontifícia Universidade Católica do Rio Grande do
Sul (PUCRS)

Prof. Dr. Antonio Macílio Pereira de Lucena
Instituto de Nacional de Pesquisa Espaciais (INPE)

A todos que, de alguma maneira, contribuíram com este trabalho. Muito obrigado, sem vocês nada disso estaria acontecendo.

AGRADECIMENTOS

Primeiramente, agradeço a Deus pela benção que é minha vida, pelas oportunidades, pela saúde e por todos os acontecimentos que me levaram até aqui. Muito obrigado.

Ao meu pai, que é o meu grande incentivador, tutor, conselheiro, orientador, amigo e pai. Sem suas palavras jamais chegaria até aqui. O senhor é o maior exemplo em todos os aspectos da vida, que um filho poderia ter.

A minha querida mãe, dona de uma garra sobre-humana, que me faz acreditar que a luta pela vida é tão importante que as vezes é necessário abdicar até das coisas que mais gostamos.

A minha linda amada, que faz os meus dias mais felizes, que me compreende e está sempre disposta a me mostrar que a vida não é só lógica, mas também sentimento. Eu te amo e você sabe.

Aos meus irmãos, primos e sobrinhos, a força da família é essencial para o meu avanço.

Ao professor Jarbas pelos conselhos e orientação neste trabalho.

Aos meus amigos júlio César e Davyd pelo empenho e dedicação que fizeram com que o desenvolvimento deste projeto tenha sido bem executado.

Ao professor Lima e ao Instituto de Educação Física e Esportes da UFC pela liberação de meio período da minha jornada de trabalho para que eu dedicasse esse tempo a obtenção do título de mestre.

Ao meu filho que está por vir, seja bem vindo. Papai te ama.

E à Agência Espacial Brasileira (AEB) pelo financiamento deste projeto.

“Temo o dia em que a tecnologia se sobreponha à humanidade. Então o mundo terá uma geração de idiotas.”

(Albert Einstein)

RESUMO

A definição do padrão CubeSat impulsionou a pesquisa e o desenvolvimento de nano e pico satélites por parte das universidades. Além da padronização das dimensões físicas, o CubeSat também especifica um protocolo de comunicação entre os subsistemas do satélite, o que torna possível reduzir o tempo e o custo do projeto através do uso de componentes COTS (do inglês *Comercial off-the-shelf*). Embora o custo seja um requisito importante para o desenvolvimento de pequenos satélites, a sua redução não deve decorrer da diminuição da confiabilidade do sistema e sim da escolha de arquiteturas adequadas. Focando em um componente específico, como o OBC (do inglês *On Board Computer*), fica claro que as soluções disponíveis atendem principalmente ao requisito de baixo custo, mas falham em oferecer alta tolerância à falhas. Alguns modelos de OBC não oferecem nenhum mecanismo de detecção ou correção de falhas, enquanto outros oferecem apenas uma cobertura parcial dos pontos de falha ao introduzirem lógica para correção de erros nas memórias externas. Neste trabalho, será proposto uma arquitetura *open source* de baixo custo e alta confiabilidade para um computador de bordo compatível com o padrão CubeSat. A arquitetura proposta utiliza o processador TMS570LS0432 do fabricante Texas Instruments, o qual possui: núcleo ARM Cortex-R4 em duas CPUs; detecção e correção de falhas em suas memórias RAM e ROM internas; *hardware* BIST (Auto-teste interno de fábrica) tanto na CPU quanto na memória RAM; e outras características de segurança como o monitoramento do *clock* e da tensão de alimentação. Uma memória Flash externa foi utilizada para armazenamento de código e dados. Foram disponibilizadas duas interfaces I2C para a comunicação com os subsistemas existentes em um CubeSat, sendo uma exclusiva para comunicação com o *Transponder* e outra comum para os demais. A arquitetura é complementada por uma interface UART para diagnóstico e depuração, sinais PWM para acionamento das bobinas de torque e entradas ADC para medição da intensidade da luz solar nas faces do satélite. Estão previstos ainda um cartão MicroSD para armazenamentos de dados e uma interface CAN para tráfego de informações transmitidas em tempo real, garantindo assim um controle rígido de erros e a recepção de mensagens. O computador de bordo foi eletronicamente testado e está apto à integração com os demais subsistemas.

Palavras-chave: CubeSat. Computador de Bordo. Arquitetura. *Hardware*. Processador.

ABSTRACT

The definition of the CubeSat standard boosted the research and development of the pico and nano satellites inside of the universities. Besides the standardization of the physical dimensions, the CubeSat specifies a communication protocol between the sub-systems of the satellite, what can make possible the shortening of the time and cost of the project using COTS (Comercial off-the-shelf) components. Although cost is an important requirement for the development of small satellites, the cost reduction must come from the correct architecture choice, not from the loss of system reliability. Focusing in a specific component, like the OBC (On Board Computer), becomes clear that the currents solutions meets the low cost requirements but fails in providing high failure tolerance. Some OBC models do not provide a mechanism of detection and correction of failures, while others cover only partially the failure points with a way to correct errors in external memories. At this job will be proposed an open source architecture with low cost and high reliability for an on board computer with compatibility with the CubeSat standard. The proposal architecture uses a TMS570LS0432 processor from Texas Instruments, which have dual ARM Cortex-R4 core, detection and failure correction in RAM and internal ROM, hardware BIST (built-in self test) at the CPU and RAM memory and others securities characters like clock monitoring and power supply voltage. An external flash memory was used for storage of code and data. Two I2C interfaces for communication between the sub systems, one had been used exclusively for transponder communication and the other one for the other systems. The architecture is complemented with an UART interface for diagnosis and debugging, PWM signals for activate the torque coils and ADC inputs for light measuring in the satellite. A MicroSD card reader has been planned for data storage and a CAN bus for data traffic in real time, ensuring a rigid control of failures and messages receiving. The OBC was electronically tested and is able to be embedded with the others sub systems.

Keywords: CubeSat. On Board Computer. Architecture. Hardware. Processor

LISTA DE ILUSTRAÇÕES

Figura 1 – Nano-satélite no Padrão CubeSat	15
Figura 2 – Arquitetura do NanoMind A712C	23
Figura 3 – Arquitetura do CubeComputer V3	25
Figura 4 – Arquitetura do CubeComputer V3	26
Figura 5 – Arquitetura do OpenOBC	28
Figura 6 – Arquitetura interna do TMS570LS0432	30
Figura 7 – Lista de sinais existentes nos conectores utilizados no padrão CubeSat . . .	31
Figura 8 – Distribuição dos módulos de <i>software</i> em camadas e sua distribuição sobre o <i>hardware</i>	34
Figura 9 – Implementação dos dois núcleos do processador	37
Figura 10 – Tensões de alimentação do OpenOBC	41
Figura 11 – Circuito projetado para a geração do sinal nRST	41
Figura 12 – Circuito projetado para a geração do sinal nPORRST	42
Figura 13 – Sinais de nRST e de clock	43
Figura 14 – A IDE CCS rodando um código para demonstração da validação da comuni- cação JTAG e gravação do processador	44
Figura 15 – Barramento SPI com três escravos	44
Figura 16 – Transação SPI transmitindo/recebendo duas palavras de dados, cada uma com 8 bits.	45
Figura 17 – Esquema elétrico da ponte SPI/I2C com seus sinais de comunicação	48
Figura 18 – Distribuição dos espaços das memórias de resultados a partir da definição dos registradores ADBNDCR e ADBNDEND	52
Figura 19 – Circuito de acionamento dos <i>magnetorquers</i> através de sinais PWM	53
Figura 20 – Circuito de comunicação da interface CAN	55
Figura 21 – Placa de Circuito Impresso do OpenOBC com suas dimensões	61

LISTA DE TABELAS

Tabela 1 – Comparação entre os processadores	26
Tabela 2 – <i>Device-drivers</i> do projeto OpenOBC	35
Tabela 3 – Mapeamento dos módulos SPI do processador nos dispositivos escravos do projeto OpenOBC	46
Tabela 4 – Relação de testes executados na memória Flash do projeto OpenOBC	47
Tabela 5 – Interrupção básicas relacionadas aos eventos de leitura das conversões e ao acesso à memória de resultados	52
Tabela 6 – Tabela de Configuração do FreeRTOS para o OpenOBC	60

LISTA DE ABREVIATURAS E SIGLAS

COTS	Comercial Off-The-Shelf
OBC	On Board Computer
PCI	Placa de Circuito Impresso
LEO	Low Earth Orbit
TID	Total Ionizing Dose
SEE	Single-Event Effects
CAN	Controller Area Network
I2C	Inter-Integrated Circuit
ESL	Electronic Systems Laboratory
SEU	Single-Event Upsets
SEL	Single-Event Latch-up
RTC	Real Time Clock
ADC	Analog-Digital Converter
UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
UFSC	Universidade Federal de Santa Catarina
CPU	Central Processing Unit
IMU	Inertial Measurement Units
DAC	Digital-Analogic Converter
USCI	Universal Serial Communication Interfaces
BIST	Built-in Self Test
PWM	Pulse Width Modulation
OTP	On Time Programmable
HAL	Hardware Abstraction Layer
RTOS	Real-Time Operating System
API	Application Programming Interface
ECC	Error-Correcting Code
OCP	Over Current Protection
OVP	Over Voltage Protection
TSD	Thermal Shutdown

BOM	Bill of Materials
GPL	General Public License
JTAG	Joint Test Access Group
IDE	Integrated Development Environment
CCS	Code Composer Studio
MISO	Master In Slave Out
MOSI	Master Out Slave In
CLK	Clock
CS	Chip Select
SS	Slave Select
GPIO	General Purpose IN/OUT
LDM	Load Multiple
HET	High-End Timer
DLC	Data Length Code

SUMÁRIO

1	INTRODUÇÃO	15
1.1	Objetivos	17
1.2	Organização do Trabalho	17
1.3	Produção Intelectual	18
2	FUNDAMENTOS TEÓRICOS	19
2.1	A Origem do Padrão CubeSat	19
2.2	Computador de Bordo	21
2.3	OBCs existentes	23
2.3.1	<i>GomSpace NanoMind A712C</i>	23
2.3.2	<i>ESL CubeComputer V3</i>	24
2.3.3	<i>FloripaSat OBDH</i>	24
2.3.4	<i>Tabela Comparativa</i>	25
3	DESENVOLVIMENTO DO OPENOBC	27
3.1	Arquitetura do OpenOBC	27
3.2	O processador TMS570LS0432	29
3.3	Interfaces do OpenOBC	30
3.4	Unidades de Armazenamento do OpenOBC	32
3.5	Fontes de Alimentação do OpenOBC	33
3.6	<i>Software e device-drivers</i> do OpenOBC	33
3.7	Características de Tolerância a Falhas	36
3.8	Requisito de Baixo Custo	37
3.9	OpenOBC: um projeto <i>Open Source</i>	38
4	RESULTADOS DO OPENOBC	40
4.1	Validação da fonte de alimentação e Sinais vitais do processador	40
4.2	Comunicação JTAG	42
4.3	Comunicação SPI	43
4.4	Comunicação I2C	47
4.5	Conversor Analógico Digital	50
4.6	Sinais PWM	53
4.7	Comunicação CAN	54

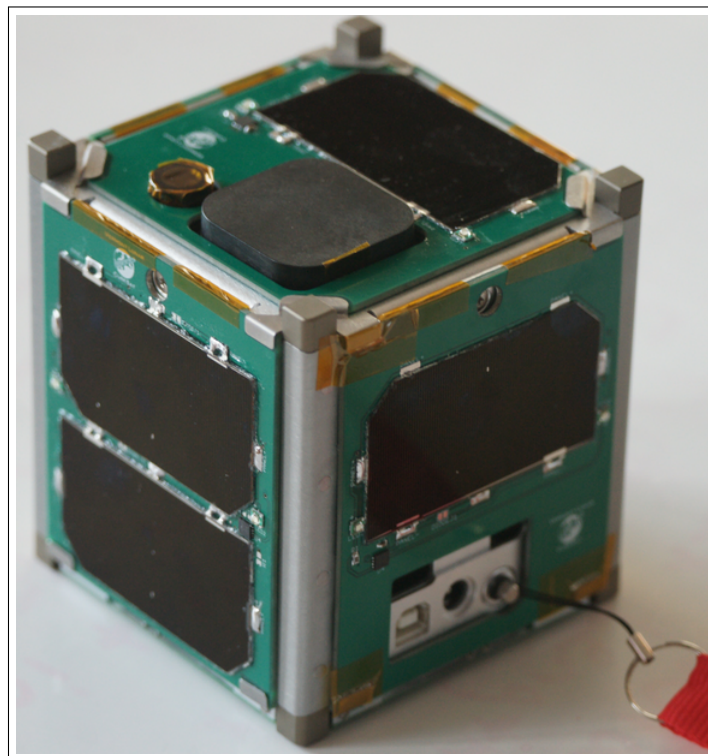
4.8	<i>Porting do FreeRTOS</i>	55
4.8.1	<i>Modo de Operação</i>	56
4.8.2	<i>Gerenciamento de Recursos</i>	57
4.8.3	<i>Funções de Callback</i>	57
4.8.4	<i>Gerenciamento de memória</i>	58
4.8.5	<i>Coleta e análise de dados</i>	59
4.8.6	<i>Software Timer</i>	59
4.8.7	<i>Configuração do FreeRTOS</i>	59
4.9	Placa de Circuito Impresso	59
5	CONCLUSÕES E TRABALHOS FUTUROS	62
	REFERÊNCIAS	63
	APÊNDICES	65
	APÊNDICE A – Esquemas Elétricos	65
	APÊNDICE B – Lista de Materiais	73

1 INTRODUÇÃO

Nas últimas décadas o interesse em aplicações espaciais tem crescido bastante, mas o alto custo das missões tornava proibitivo o desenvolvimento de missões por parte das universidades. Com isso, as aplicações de pico satélites e nano satélites se apresentavam como uma solução com um custo bem mais acessível, permitindo o rápido desenvolvimento deste campo de pesquisa.

Mesmo utilizando-se de pico e nano satélites, o tempo médio para finalizar um projeto era muito alto, uma vez que a cada iteração era necessário desenvolver todos os módulos da aplicação. Em 1999 foi proposto pela Universidade Politécnica do Estado da Califórnia e pela Universidade de Stanford o padrão CubeSat para nano-satélites como uma resposta para diminuir o tempo de desenvolvimento de missões espaciais. Desde então, o CubeSat foi adotado por centenas de organizações em todo o mundo e os desenvolvedores desse padrão incluem não apenas as universidades e instituições de ensino, mas também as empresas privadas e organizações governamentais, pelo fato de ser mais acessível e frequente as oportunidades de lançamentos para o espaço (CUBESAT, 2016). Um modelo desse padrão é mostrado na Figura 1.

Figura 1 – Nano-satélite no Padrão CubeSat



Fonte – Nasa (2013).

O padrão CubeSat trouxe duas grandes vantagens para a pesquisa aeroespacial:

- A primeira é o tamanho padrão de 1U (10x10x10 cm), que permite fácil encaixe do nano satélite no veículo de lançamento junto de outros grandes satélites em estações orbitais, através de um adaptador de lançamento padrão;
- A segunda é a possibilidade do uso de vários componentes COTS (do inglês *Comercial Off-The-Shelf*) para montar um sistema CubeSat completo, uma vez que o padrão especifica, além do tamanho físico, um protocolo de comunicação que permite que todas as partes do sistema sejam adquiridas de fabricantes independentes e montadas.

Com essas duas características citadas acima, os pesquisadores podem se concentrar na carga útil que o satélite levará.

Como dito anteriormente, um satélite CubeSat é formado por alguns subsistemas no formato modular, nos quais se comunicam entre si através de protocolos padronizados. Há diversos subsistemas que podem fazer parte do conjunto completo, porém alguns desses subsistemas são de grande importância para o funcionamento do satélite e outros variam com o tipo de missão que o mesmo será destinado. São exemplos de subsistemas o computador de bordo, o módulo de energia, o *transponder*, etc.

Após uma pesquisa no mercado brasileiro, constatou-se que não possuímos um subsistema de computador de bordo (OBC, do inglês *On Board Computer*) como oferta nacional de componentes COTS para o padrão CubeSat. Neste sentido, este projeto tem como objetivo projetar e desenvolver protótipos de um computador de bordo que atenda as características necessárias para o padrão CubeSat. Além disso, fornecer ao final do projeto um componente COTS nacionalizado e que poderá ser prontamente integrado em qualquer plataforma que atenda este padrão.

Mas apenas a criação de uma oferta nacional não é considerada um incentivo suficiente para substituir outras plataformas que já provaram sua eficiência em missões prévias. Será pleiteada a criação de um módulo com preço de custo inferior aos módulos internacionais (GOMSPACE, 2016) e ainda uma atualização na camada de confiabilidade. O projeto prevê a seleção e a escolha de um processador que possua, entre suas características: (i) redundância de seu núcleo interno; (ii) detecção e correção de erros das memórias embarcadas; (iii) monitoramento do *clock*; (iv) monitoramento da tensão de alimentação e (v) auto-teste em tempo de aplicação do próprio núcleo do processador e sua memória RAM.

Assim, será desenvolvida uma plataforma confiável e preparada para o ambiente hostil no qual a aplicação será submetida. Todas essas novas características serão acrescentadas, obedecendo as restrições intrínsecas de aplicações espaciais e custo da aplicação.

1.1 Objetivos

Este projeto tem como objetivo o desenvolvimento de um computador de bordo referenciado ao projeto CONASAT (Constelação Nano-Satélites Ambientais) (INPE, 2016), no qual o mesmo pode ser responsável pelo controle e comunicação existente no nano satélite, gerenciando assim todo o tráfego de dados entre os demais subsistemas. Pelo fato do conceito do padrão CubeSat basear-se em pesquisa de soluções inovadoras e de baixo custo, será estudado e implementado um OBC compatível com os já existentes no mercado, mas sempre seguindo a linha de inovação científica/tecnológica e buscando a melhor relação custo-benefício. Abaixo são apresentados alguns dos objetivos específicos.

- O desenvolvimento de uma arquitetura otimizada baseada em um processador robusto e compatível com a aplicação;
- A pesquisa dos componentes eletrônicos críticos;
- O estudo e implementação de circuitos eletrônicos;
- O desenvolvimento da Placa de Circuito Impresso (PCI) baseada no padrão CubeSat;
- Análise de tolerância a falhas do OBC;
- O desenvolvimento dos *device-drivers* das interfaces utilizadas;
- O *porting* do Sistema Operacional de Tempo Real FreeRTOS;
- Os testes sistêmicos dos *device-drivers* do OBC;
- Fomentar a área de engenharia espacial através do desenvolvimento deste projeto Open Source;

1.2 Organização do Trabalho

Este trabalho é assim organizado: no Capítulo 2 será introduzido a origem do padrão CubeSat e suas principais vantagens, em seguida, realiza-se um estudo sobre computadores de bordo para nano satélites, verificando suas qualidades e deficiências, propondo melhorias baseando-se nos aspectos de tolerância a falhas, tecnologias utilizadas e custo benefício da solução.

Para atender os requisitos mínimos de um OBC, discute-se no Capítulo 3 o desenvolvimento de uma arquitetura de *hardware* e *software* adequado ao padrão CubeSat. O capítulo é complementado com a descrição dos componentes críticos utilizados, a implementação do projeto de *hardware*, o desenvolvimento dos *device-drivers* e do *porting* do FreeRTOS.

No Capítulo 4, consta o detalhamento dos testes da PCI, validando as fontes de alimentação e visualizando os sinais vitais do processador, obtendo assim a sua inicialização, a validação dos circuitos eletrônicos do projeto através dos *device-drivers* desenvolvidos e os testes do *porting* do FreeRTOS. Ainda neste capítulo serão demonstrados algumas técnicas de roteamento, as imagens do protótipo da PCI montada e referências contendo toda a documentação do projeto. Finalmente, as conclusões e trabalhos futuros serão discutidos no Capítulo 5.

1.3 Produção Intelectual

Nesta dissertação de mestrado foi publicado um artigo, conforme descrito abaixo, no *2nd IAA Latin American Cubesat Workshop* suportado pela sociedade científica *International Academy of Astronautics*.

MOTA DAVID; AMERICO, J. M. D. S. J. M. J. M. C. OpenOBC: An open-source, low-cost and highly-reliable architecture for a cubesat on-board computer. *2nd IAA Latin American Cubesat Workshop*, 2016.

2 FUNDAMENTOS TEÓRICOS

Neste capítulo, serão expostos a origem do padrão CubeSat e a revolução pertinente a seu uso. Em seguida, será discutido a importância de um computador de bordo em um satélite e, por fim, serão apresentadas as características técnicas de alguns OBCs existentes no mercado, bem como alguns modelos desenvolvidos por universidades.

2.1 A Origem do Padrão CubeSat

Em 1999, os professores Jordi Puig-Suari da Universidade Politécnica da Califórnia (Cal Poly) e Bob Twiggs da Universidade de Stanford desenvolveram um conceito de satélite muito pequeno para ajudar as universidades em todo o mundo a permitir e incentivar que estudantes e pesquisadores se inserissem e realizassem pesquisas na área da ciência espacial. O CubeSat, como inicialmente proposto, não se propôs a tornar-se um padrão. No entanto, tornou-se um padrão ao longo do tempo por um processo de emergência, e culminou em um conceito de satélite muito popular e padronizado. Os primeiros CubeSats foram lançados em Junho de 2003 num veículo de lançamento chamado Rockot (ISIS, 2016b).

O objetivo deste padrão era criar um conceito que permitisse não só aos grupos universitários de implementar rapidamente uma pequena missão espacial, mas também assegurar que as possibilidades de embarcar num lançamento espacial como passageiro secundário fossem maximizadas, através da normalização das interfaces, proibindo ou limitando aspectos de projeto que poderiam ser potencialmente perigosos, reduzindo as chances de ser lançado próximo de veículos espaciais maiores e mais caros (ISIS, 2016b).

O CubeSat é um tipo de satélite muito pequeno que é baseado em uma unidade padronizada de massa e volume. A unidade básica inicial do padrão CubeSat mede 10x10x10 centímetros cúbicos (1U, uma unidade), de acordo com as interfaces específicas, para permitir que um lançamento seja padronizado em contêiner e tenha massa máxima de 1 kg (a massa foi posteriormente aumentada para 1,33 kg). Foi rapidamente percebido que tais unidades básicas do CubeSat poderiam ser combinadas para formar veículos espaciais ligeiramente maiores, enquanto na maior parte seguem os mesmos requisitos e restrições. Múltiplos da unidade básica do padrão CubeSat foram combinados para estabelecer CubeSats maiores. Por exemplo, um CubeSat de 1 Unidade mede uma única unidade básica de CubeSat como descrito acima, enquanto um CubeSat de 3 Unidades consiste em 3 unidades de CubeSat padrão empilhadas juntas (ISIS, 2016b).

O conceito CubeSat tornou-se muito popular, tanto em grupos universitários, como para pesquisadores, agências espaciais, governos e empresas. Os CubeSats oferecem uma maneira rápida e acessível para uma ampla gama de interessados na área aeroespacial permitindo uma rápida inovação. As características típicas ao padrão CubeSat seguem logo abaixo (ISIS, 2016b):

- **Modularização:** Dado o tamanho padronizado do CubeSat, tornou-se possível desenvolver muitos módulos funcionais padronizados, tais como computadores de bordo, sistemas de comunicação, etc. O uso desses módulos comerciais (COTS) é uma das principais razões pelas quais o CubeSat pode ser implementado rapidamente e também ser rentável;
- **Lançamento:** Os CubeSats normalmente são lançados no espaço como uma carga útil em um contêiner, dentro de um *deployer*, como um meio para reduzir a complexidade do lançamento e os custos associados;
- **Custo:** Os CubeSats, normalmente, são utilizados em missões que são implementadas usando um orçamento muito baixo;
- **Componentes:** Componentes eletrônicos que não são feitos para uso em aplicações espaciais são frequentemente usados e aceitos nas missões de um CubeSat. Isso permite uma abordagem numa implementação curta e de baixo custo, permitindo que os desenvolvedores façam uso dos mais recentes componentes comerciais e industriais. O fato de que a maioria das missões de um CubeSat estão em órbita baixa (LEO, do inglês *Low Earth Orbit*) e têm uma vida útil de missão curta, torna possível usar tais componentes;
- **Desenvolvimento:** Devido à baixa complexidade e ao pequeno escopo e escala de projetos de um CubeSat, muitas vezes é possível utilizar métodos e metodologias de projeto menos formais. Muitas vezes é possível trabalhar em equipes de projeto compactas, eliminando ainda mais a necessidade de grandes documentações e outras despesas gerais;
- **Riscos:** O padrão CubeSat é o satélite com o maior risco técnico aceito em troca de um menor custo, implementação mais rápida, com inovação ou uma combinação desses elementos. Tais elementos também permitem diferentes abordagens de mitigação de risco (re-voos, peças sobressalentes em órbita, etc) do que as abordagens tradicionais de mitigação de riscos;
- **Nicho de aplicação:** A vantagem dos CubeSats não está em ter o melhor desempenho em termos de largura de banda ou de amostragem de sinal, pois isso não é compatível com as grandes restrições que o padrão CubeSat impõe ao tamanho do satélite. Porém,

quando usado em redes ou constelação, os satélites CubeSats são capazes de fornecer muito melhor resolução temporal a preços acessíveis.

2.2 Computador de Bordo

Um OBC é um sistema computacional, ou um conjunto de sistemas computacionais, que processa as informações enviadas para o satélite e as informações geradas pelos outros subsistemas do satélite (RAZZAGHI, 2012). Sendo diretamente responsável pelo sucesso da missão, o OBC deve ser desenvolvido para ser o mais robusto possível, mas obedecendo as restrições de projeto do padrão CubeSat.

Os principais elementos de um OBC são: processador, memória de *boot*, memória de segurança (*safeguard*), memória de trabalho, memória de armazenamento para os dados obtidos, barramento de dados, barramento de controle, interface de depuração, fonte de alimentação e *clock* (EICKHOFF, 2012).

Ao desenvolver os elementos de um OBC, temos que: a memória de *boot* é a memória que guarda o programa que será executado; a memória de trabalho é a memória RAM; a memória de armazenamento é uma memória flash ou EPROM; no qual serão armazenados os dados da carga útil da operação. Os barramentos de dados e controle, interface de depuração, fonte e *clock* são os periféricos necessários para o funcionamento de um sistema computacional. A memória de segurança é onde ficam armazenadas as configurações do OBC e as informações sobre o estado do satélite. O processador é o elemento principal do OBC em que, durante o projeto, as interfaces serão definidas obedecendo às restrições e as opções que o processador oferece, e por isso, a escolha desse componente deve ser cuidadosamente analisada.

As principais funções de um OBC são (WELLS G. J.; STRAS, 2003):

- Monitoramento e armazenamento da telemetria gerada pelo satélite para transmissão futura;
- Codificação e decodificação dos pacotes oriundos da comunicação com a estação base;
- Processamento de telecomandos da estação base;
- Monitoramento dos subsistemas, analisando o seu correto funcionamento e reinicializando o sistema quando necessário.

Existem vários tipos de falhas que podem afetar um OBC em ambiente espacial. As falhas mais significativas e as suas taxas de ocorrência são:

- Desenvolvimento – 24,8%

- Falhas em entender todas as variáveis do ambiente – 21,4%;
- Operação (comandos errados) – 4,7%;
- Falhas aleatórias, tais como:
 - Peças defeituosas – 16,3%;
 - Problemas de qualidade – 7,7%;
 - Outros problemas – 6,3%;
 - Causas desconhecidas – 18,9%.

Esses dados foram obtidos através de uma coleção de informações em cerca de 2500 missões espaciais entre 1962 e 1988 (BRIDGES, 2012).

O maior índice de falhas está no processo de desenvolvimento do produto, seguido de problemas em entender as reais condições na qual a aplicação será submetida.

Os principais desafios de ambiente que um satélite do padrão CubeSat irá enfrentar são vibração na fase de lançamento, condições térmicas e radiação em LEO (RAZZAGHI, 2012).

O projeto de um OBC deve atender a esses três critérios. A vibração durante o lançamento é devido à aceleração de aproximadamente 20g por 90 segundos, necessária para escapar do campo gravitacional da terra (TRIBBLE, 1995). As condições térmicas variam em uma grande faixa, sendo o sol o grande responsável por essa variação, com uma intensidade média de 1367 W/m² (GILMORE, 2002). A radiação em LEO pode ser dividida em dois principais efeitos em aplicações espaciais: Dose Total Ionizante (TID, do inglês *Total Ionizing Dose*) e Efeitos de evento único (SEE, do inglês *Single-Event Effects*).

TID provoca a degradação dos circuitos eletrônicos devido à energia depositada no material de forma cumulativa quando exposto por um longo período a radiação. Um dos métodos para evitar erros provenientes desse efeito é usar versão de dispositivos chamados *Radiation Hardened*, que possuem uma certa tolerância a essa radiação. Devido ao período de missão considerado pelo CubeSat, componentes COTS são considerados ao invés dos mais caros com características de *Radiation Hardened*.

SEE são efeitos quase instantâneos que acontecem através do impacto de um simples íon, depositando energia suficiente para ocasionar erros como a simples inversão de um bit na memória (*soft errors*), ou até provocar um defeito permanente no dispositivo (*hard errors*).

Porém, como o padrão CubeSat adota o uso de componentes eletrônicos de uso geral, o desafio está em atender os requisitos de tolerância a falhas sem grandes custos podendo ser na implementação de um *hardware*, de *software* ou de ambos para solucionar este problema sem a

necessidade de se utilizar componentes com características de *Radiation Hardened*.

2.3 OBCs existentes

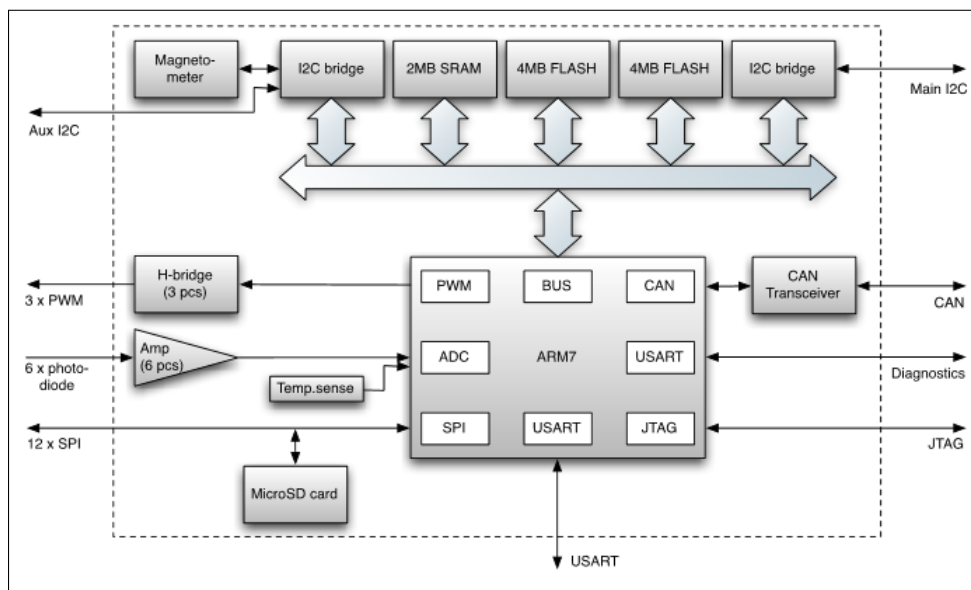
Nesta seção serão analisados alguns OBCs comerciais e acadêmicos averiguando suas principais características e, por fim, uma tabela de comparação entre os mesmos.

2.3.1 GomSpace NanoMind A712C

O computador de bordo NanoMind A712C do fabricante GomSpace é projetado para ser um sistema eficiente para aplicações espaciais com recursos limitados, tais como Cubesat ou nano-satélite. Além de um sistema de computação totalmente eficiente, ele fornece um magnetômetro de três eixos para detectar o campo magnético da Terra e os *drivers* de bobina que podem ser usados para implementar controle de atitude com base em sensor magnético e atuadores. Sua interface principal para outros subsistemas são um barramento CAN (do inglês *Controller Area Network*) e um I2C (do inglês *Inter-Integrated Circuit*) (GOMSPACE, 2016).

O NanoMind A712C é baseado no processador AT91SAM7A1 do fabricante Atmel que contém o núcleo ARM7TDMI. Este processador possui uma arquitetura RISC de 32 *bits* de alto desempenho com um conjunto de instruções de 16 bits de alta densidade e consumo de energia muito baixo. A arquitetura deste OBC está ilustrada na Figura 2.

Figura 2 – Arquitetura do NanoMind A712C



Fonte – Gomspace (2016).

2.3.2 *ESL CubeComputer V3*

O CubeComputer V3 do fabricante ESL (do inglês *Electronic Systems Laboratory*) é um computador de bordo de propósito geral destinado a ser utilizado em nanosatélites. Ele foi projetado para ser totalmente compatível com o padrão CubeSat, bem como outros subsistemas CubeSat dos fabricantes ISIS, Pumpkin, ClydeSpace, Gomspace e outros produtos ESL. Este OBC é baseado em um processador com o núcleo ARM Cortex-M3 que oferece alto desempenho em potência muito baixa. Para proteger o OBC contra efeitos de radiação como SEUs (do inglês *Single-Event Upsets*) e SELs (do inglês *Single-Event Latch-up*), técnicas de tolerância a falhas foram implementadas para maior confiabilidade (ISIS, 2016a).

Como dito logo acima, este OBC faz uso de um processador que é baseado na arquitetura ARM Cortex-M3 de 32 *bits* e opera entre 4 à 48 MHz. Este processador também é otimizado para a eficiência energética e, portanto, tem um consumo de energia muito pequena de 200 uA/MHz ao executar a partir da memória Flash e 0,9 uA quando está no modo *sleep*. O processador também possui, em sua arquitetura interna, todos os periféricos necessários para um computador a bordo de um nanosatélite: RTC (do inglês *Real Time Clock*), Temporizadores, ADC (do inglês *Analog-Digital Converter*), Interface de Memória Externa, UART (do inglês *Universal Asynchronous Receiver/Transmitter*), SPI (do inglês *Serial Peripheral Interface*) e I2C (ISIS, 2016a). A Figura 3 ilustra a arquitetura do OBC CubeComputer V3.

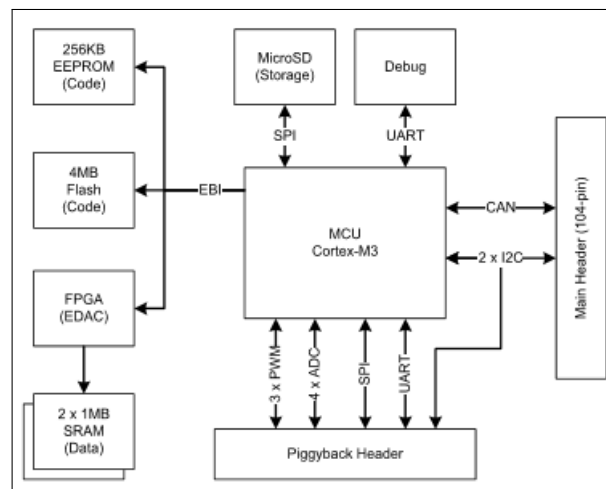
Para proteger o processador contra travamentos de *software* e *hardware*, é implementado um *watchdog* interno e externo. No caso de o travamento ser causado por um bloqueio induzido por radiação dentro do processador, uma reposição normal não irá recuperar o processador. Portanto, a ação do *watchdog* externo faz com que o OBC remova o *latchup* (ISIS, 2016a).

2.3.3 *FloripaSat OBDH*

O OBDH é o computador de bordo do satélite FloripaSat desenvolvido pela Universidade Federal de Santa Catarina (UFSC). Este OBC é constituído de uma unidade central de processamento (CPU, do inglês *Central Processing Unit*), de uma memória não volátil, de *drivers* de corrente, de uma unidade de medição inercial (IMU, do inglês *Inertial Measurement Units*), sensor de corrente e de interfaces de comunicação (FLORIPASAT, 2016).

Quem executa o código principal do OBDH é um SoC (do inglês *System on Chip*) que

Figura 3 – Arquitetura do CubeComputer V3



Fonte – ISIS (2016a).

contém um núcleo, memória RAM e memória Flash (usado para armazenamento de programas e registros de status não-volátil). O processador deste OBC é o MSP430F6659IPZ do fabricante Texas Instruments que possui o núcleo RISC de 16 *bits* com características de baixa potência. Contém sete modos de operação de consumo de energia, quatro temporizadores de 16 *bits*, ADC de 12 *bits* e DAC (do inglês *Digital-Analogic Converter*), seis interfaces de comunicação serial universal (USCI, do inglês *Universal Serial Communication Interfaces*), um RTC e até 74 pinos de entrada e saída (I/O, do inglês *Input/Output*). O *clock* de funcionamento do OBDH é 32 MHz (FLORIPASAT, 2016).

Para proteção do processador, há um monitor de tensão externo com um *Watchdog Timer* para garantir um certo nível de tensão para ligar o sistema e fornecer um *Watchdog* redundante (FLORIPASAT, 2016). Segue na Figura 4 a seguir a arquitetura do OBDH.

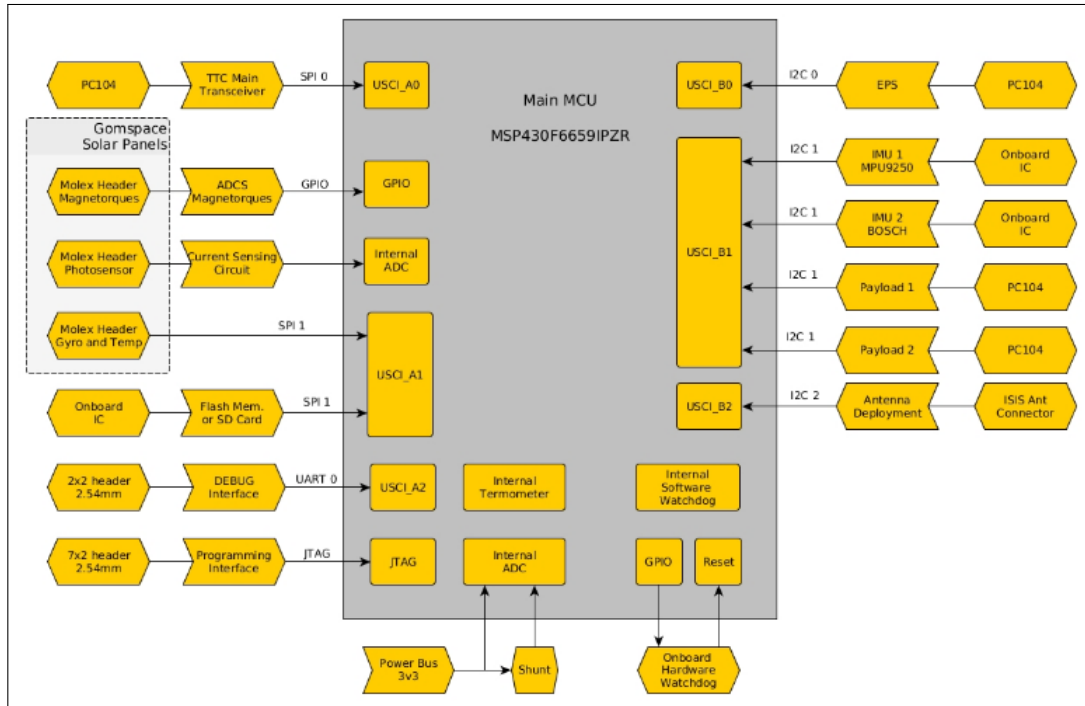
2.3.4 Tabela Comparativa

A Tabela 1 compara as características existentes na arquitetura interna de cada processador dos OBCs citados anteriormente.

Apesar de serem processadores bem consolidados e de grande eficiência, não foi detectado em suas estruturas características de tolerância a falhas, sendo elas implementadas através de *hardwares* externos, ou simplesmente as desconsiderando.

No capítulo 3 será apresentada uma solução de OBC buscando aliar as melhores qualidades desses processadores às características de tolerância a falhas internas ao processador para poder obter uma boa relação custo-benefício sem perder a alta confiabilidade necessária em

Figura 4 – Arquitetura do CubeComputer V3



Fonte – Floripasat (2016).

sistemas aeroespaciais.

Tabela 1 – Comparação entre os processadores

Itens	Descrição	Descrição	Descrição
Projeto	NanoMind A712C	CubeComputer V3	FloripaSat OBDH
Processador	AT91SAM7A1	EFM32GG280F1024	MSP430F6659IPZ
Fabricante	Atmel	Silicon Labs	Texas Instruments
Memória Flash interna	X	1024 KB	512 KB
Memória RAM interna	4 KB	128 KB	66 KB
Memória EEPROM interna	X	X	X
Clock	40 MHz	48 MHz	8 MHz
Watchdog Timer	Sim	Sim	Sim
BIST	X	X	X
Monitoramento de operação	X	Tensão	Tensão
I2C	X	2	3
Canais ADC	8 canais de 10 bits	8 canais de 12 bits	16 canais de 12 bits
CAN	1	X	X
SPI	1	2	3
PWM	4 canais	4 canais	5 canais
UART	3	2	3

3 DESENVOLVIMENTO DO OPENOBC

Neste capítulo será descrito o desenvolvimento utilizado na implementação do OpenOBC. Inicialmente será apresentada a arquitetura proposta da solução. Em seguida será exibido o processador escolhido, juntamente com suas interfaces de comunicação que atendem o padrão CubeSat, assim como as unidades de armazenamento e fontes de alimentação. Na sequência será apresentada a arquitetura de *software* e os *devices-drivers* que a implementam. Por fim, serão analisadas as características de tolerância a falhas, discutidos os aspectos de baixo custo e o conceito de um projeto totalmente aberto à comunidade, através de um controlador de versão.

3.1 Arquitetura do OpenOBC

Para implementar o computador de bordo apresentado neste trabalho, foi necessário elaborar uma arquitetura bem detalhada com objetivo de diminuir o tempo de desenvolvimento, aumentar a eficiência da implementação reduzindo custos em versões de placas de circuito impresso e garantindo que todos os requisitos estarão elencados nessa arquitetura (MOTA D.; AMERICO, 2016b).

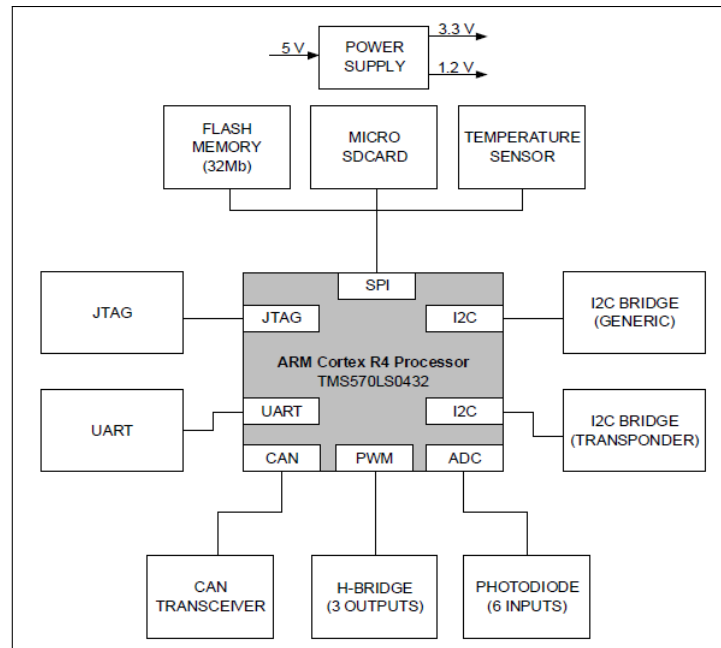
A arquitetura de um sistema embarcado é uma abstração do dispositivo em questão, à qual está associada um diagrama de fluxo de informação, significando uma simplificação do sistema geral, em que normalmente não mostra informações detalhadas de implementação, tais como código-fonte do *software* ou esquema elétrico do circuito do *hardware*.

Em nível de arquitetura, os componentes de *hardware* e *software*, em um sistema embarcado, são muitas vezes representados pela composição de elementos que interagem entre si. Estes elementos são representações de *hardware* e *software* cujos detalhes de implementação foram abstraídos, deixando apenas as informações de comunicação. Uma boa definição e um bom entendimento da arquitetura de um sistema embarcado são essenciais para se obter bons resultados em projetos que fazem uso desta solução (NOERGAARD, 2005). A Figura 5 ilustra a arquitetura desenvolvida.

Os nano e pico-satélites, normalmente, utilizam uma arquitetura de processamento descentralizada composta por diversos subsistemas, os quais têm capacidade de processamento local e são equipados com uma infraestrutura de comunicação que torna possível a sua coordenação por uma entidade centralizada (normalmente o computador de bordo) (CUBESAT,

2016). Essa arquitetura é vantajosa, pois facilita o cumprimento dos requisitos de tempo real das aplicações, uma vez que as diversas tarefas são divididas entre as diversas unidades de processamento, e isola os pontos de falha, de tal modo que o mau funcionamento de um subsistema não compromete, necessariamente, todo o sistema.

Figura 5 – Arquitetura do OpenOBC



Nesse tipo de arquitetura, como mencionado, normalmente se utiliza um computador de bordo para coordenar e sincronizar os diversos subsistemas do satélite. As funções de gerência, desempenhadas pelo OpenOBC, são realizadas através de trocas de mensagens e comandos entre o computador de bordo e as demais unidades de processamento do satélite. Isso exige que o OpenOBC seja dotado de interfaces de comunicação que viabilizem essa troca de informação (MOTA D.; AMERICO, 2016b).

Os esquemas elétricos projetados que implementam a arquitetura do OpenOBC, juntamente com o layout da PCI desenvolvida, encontram-se no Apêndice A.

No próximo item, será apresentado o processador TMS570LS0432 utilizado neste trabalho, o qual desempenha o papel principal do OpenOBC realizando as tarefas discutidas nesta seção.

3.2 O processador TMS570LS0432

O elemento principal da arquitetura é o processador. Além de executar todo o sistema operacional e as aplicações a serem desenvolvidas (*firmware*), é através dele que poderão ser implementadas as interfaces definidas no padrão CubeSat e no OpenOBC proposto para realizar a comunicação com os outros módulos do satélite. Como um computador de bordo é um equipamento que executa tarefas em tempo real de modo tolerante a falhas, e ainda controla as interfaces de entrada e saída, é necessário um processador que seja capaz de executar, sem perder desempenho, todas essas funções de forma a garantir o êxito da aplicação desejada.

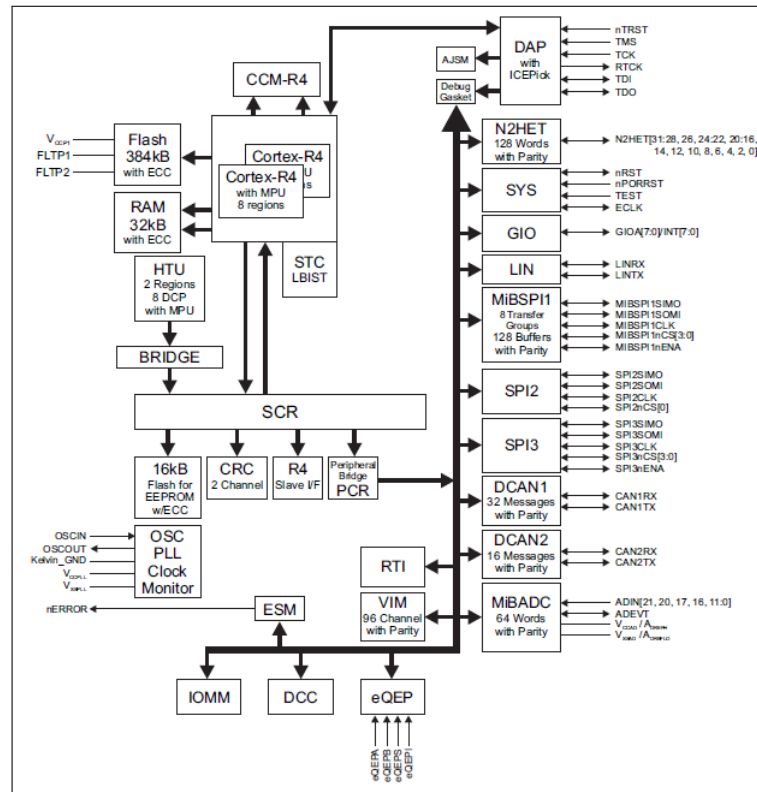
Sabe-se que existem vários computadores de bordo disponíveis no mercado, porém muitos deles têm baixa capacidade de processamento. Apesar de já terem sido submetidos a diversas missões em órbita, apresentarem eficiência mínima e ainda serem bem consolidados no mercado de microeletrônica, em sua maioria são processadores de uso geral, não contendo assim requisitos de tolerância a falhas em sua construção (GOMSPACE, 2016; ISIS, 2016a; FLORIPASAT, 2016). Neste sentido, buscou-se um novo processador, que além de adequado aos requisitos de aplicações espaciais, possibilitasse uma alta integração, diminuindo o uso de componentes externos. Ademais, o uso de circuitos integrados de alta integração é condizente com os próximos passos da evolução do segmento tecnológico, essencial nos dias de hoje.

Desta forma, foi realizada uma procura em diversos fabricantes de processadores objetivando encontrar um que atendesse os requisitos do padrão CubeSat e da missão ao qual o satélite será destinado. O processador TMS570LS0432 do fabricante Texas Instruments se mostrou adequado a esses requisitos. Ele possui dois núcleos ARM Cortex-R4, com detecção e correção de falhas em suas memórias RAM e ROM internas, *hardware BIST* (do inglês *Built-in Self Test*) tanto nos núcleos quanto na memória RAM e outras características de segurança como o monitoramento do *clock* e da tensão de alimentação. O núcleo ARM Cortex-R4 se mostrou uma solução bastante robusta para aplicações espaciais por possuir características de tolerância a falhas (TEXAS INSTRUMENTS, 2015b). A arquitetura interna deste processador está ilustrada na Figura 6, onde são mostradas todas suas interfaces de comunicação assim como os módulos de seu funcionamento.

Do ponto de vista estrutural, uma limitação deste processador é a falta da interface I2C, que será utilizado na comunicação com os subsistemas. Para superar esta limitação, foi utilizada uma ponte SPI/I2C, pelo fato de já existir, na arquitetura interna do processador, esta interface. Outra limitação é não poder reiniciar o processador manualmente, tendo em vista

que ele opera em um ambiente espacial. Com isso, a forma de reverter essa situação é utilizar o *watchdog timer* e os controles de detecção e correção de erros existentes neste processador. Porém, em caso de falhas por falta ou baixa alimentação, este processador possui o *power-on reset* como solução deste inconveniente.

Figura 6 – Arquitetura interna do TMS570LS0432



Fonte – Texas Instruments (2015b).

3.3 Interfaces do OpenOBC

O padrão CubeSat, como exposto no Capítulo 2, determina todas especificações e estruturas que serão utilizadas em satélites que queiram adotar esta solução. Para a comunicação entre os subsistemas existentes dentro do satélite, foi padronizado dois conectores com a disponibilidade de sinais dispostos conforme a Figura 7.

No OpenOBC, são disponibilizadas as interfaces de comunicação I2C, CAN, UART e SPI. A interface UART é disponibilizada basicamente para depuração, embora possa, eventualmente, ser utilizada para comunicação com algum subsistema.

A interface I2C permite a comunicação entre um mestre e diversos escravos através de um esquema de endereçamento definido nativamente em seu protocolo. Ainda assim, no

Figura 7 – Lista de sinais existentes nos conectores utilizados no padrão CubeSat

Pino	Nome	Descrição	Pino	Nome	Descrição	Pino	Nome	Descrição	Pino	Nome	Descrição
H1.1	CAN-	Barramento CAN (nível dominante baixo)	H1.27	#	Reservado para compatibilidade	H2.1	EPS0	Uso Específico do SS Energia	H2.27	+3.3V	Tensão de Saída Regulada de +3,3 V
H1.2	CAN+	Barramento CAN (nível dominante alto)	H1.28	VREF2+	Tensão de Referência 1	H2.2	EPS1		H2.28	+3.3V	
H1.3		Uso Específico da Missão	H1.29	-RESET		H2.3	EPS2		H2.29	DGND	Terra Digital
H1.4			H1.30	VREF-		H2.4	EPS3		H2.30	DGND	
H1.5			H1.31	#	Reservado para compatibilidade	H2.5	EPS4		H2.31	AGND	Terra Analógico
H1.6			H1.32	+5VCHR		H2.6	EPS5		H2.32	DGND	Terra Digital
H1.7			H1.33	#	Reservado para compatibilidade	H2.7	EPS6		H2.33	S0	Pino RBF - contato NF
H1.8			H1.34	#		H2.8	EPS7	H2.34	S0		
H1.9			H1.35	#		H2.9	COM0	H2.35	S1	Interruptor de Ejeção - contato NF	
H1.10			H1.36	#		H2.10	COM1	H2.36	S1		
H1.11			H1.37	#	H2.11	COM2	H2.37	S2	Pino RBF - contato NA		
H1.12			H1.38	#	H2.12	COM3	H2.38	S2			
H1.13		H1.39	TXD0	Transmissão de Dados Sistema	H2.13	COM4	H2.39	S3	Interruptor de Ejeção - contato NA		
H1.14		H1.40	RXD0	Recepção de Dados Sistema	H2.14	COM5	H2.40	S3			
H1.15		H1.41	SDA0	I ² C Sistema	H2.15	COM6	H2.41	S4	Pino RBF - contato Comum		
H1.16		H1.42	VBCKUP	Tensão de Backup	H2.16	COM7	H2.42	S4			
H1.17	TXD1	Transmissão de Dados Payload	H1.43	SCL0	Clock I ² C Sistema	H2.17	ADC80	H2.43	S5	Interruptor de Ejeção - contato Comum	
H1.18	RXD2	Recepção de Dados Payload	H1.44	#	Reservado para compatibilidade	H2.18	ADC81	H2.44	S5		
H1.19	#	Reservado para uso futuro	H1.45	#		H2.19	ADC82	H2.45	VBAT	Tensão Direta da Bateria	
H1.20	#	Tensões de Saída Protegidas	H1.46	#	Uso Específico do SS Controle Atitude	H2.20	ADC83	H2.46	VBAT		
H1.21	SCL1		Clock I ² C Payload	H1.47		VCC0	H2.21	ADC84	H2.47	PAYLD0	Uso Específico do Payload
H1.22	#		Reservado par auso futuro	H1.48		VCC1	H2.22	ADC85	H2.48	PAYLD1	
H1.23	SDA1		Dados I ² C Payload	H1.49		VCC2	H2.23	ADC86	H2.49	PAYLD2	
H1.24	#		Reservado para uso futuro	H1.50		VCC3	H2.24	ADC87	H2.50	PAYLD3	
H1.25	-FAULT		Falha de Alimentação (Coletor Aberto)	H1.51		VCC4	H2.25	+5V	H2.51	PAYLD4	
H1.26	VREF1+		Tensão de Referência 1	H1.52		VCC5	H2.26	+5V	H2.52	PAYLD5	

Fonte – INPE (2016).

OpenOBC, optou-se por disponibilizar duas interfaces I2C independentes, uma exclusiva para a comunicação com o *transponder* e outra para a comunicação com os demais subsistemas. Essa separação aumenta a robustez e a flexibilidade da comunicação, tornando possível, por exemplo, priorizar a troca de informação com o *transponder*. A interface I2C do OpenOBC suporta taxas de dados de até 1.2 Mbps.

Também é possível utilizar o OpenOBC em uma rede CAN, o que possibilita a troca de mensagens de forma robusta, com controle de erro e garantia de recebimento das mensagens. O transmissor CAN presente no OpenOBC suporta taxas de dados de até 1Mbps e possui diversos mecanismos de proteção, como *cross-wire*, desligamento térmico, reinicialização e circuito aberto (TEXAS INSTRUMENTS, 2015a).

Uma interface SPI é utilizada para realizar a leitura de um sensor de temperatura que realiza as medições das temperaturas existentes na placa de circuito impresso do OpenOBC. O circuito integrado que implementa esta solução é o LM70 do também fabricante Texas Instruments (TEXAS INSTRUMENTS, 2016).

O OpenOBC é capaz de realizar aquisição de sinais analógicos, para isso são disponibilizadas 6 entradas analógicas que são multiplexadas para a entrada de um ADC de 12 bits. Quando essas entradas analógicas são utilizadas para leitura de fotodiodos, o *hardware* do OpenOBC já possui circuito de condicionamento de sinais adequado para esse tipo de aplicação, não sendo, portanto, necessário nenhum tipo de circuito externo adicional. Quando utilizado para

outro propósito que não a leitura de fotodiodos, o OpenOBC possui apenas um filtro *anti-aliasing* e algum tipo de circuito de condicionamento externo pode ser necessário. Por padrão, as entradas analógicas são conectadas ao circuito de condicionamento dos fotodiodos. Assim, caso não seja o propósito desejado, aquele circuito deve ser desviado através de resistores de 0 Ohm presentes na placa.

A arquitetura de *hardware* do OpenOBC também possui três saídas de *drivers* de ponte H. A aplicação principal para essas saídas é, mas não está limitada, o acionamento de bobinas de torque (*magnetorquers*) externas que são utilizadas para implementar controle de altitude. Os *drivers* da ponte H operam com tensão de 5 V e podem fornecer até 2 A de corrente por saída. Quando a carga acionada exige valores de corrente que superam a capacidade da fonte principal de 5 V, pode-se utilizar uma tensão auxiliar de 5 V destinada a alimentar exclusivamente o circuito da ponte H. A escolha entre a fonte principal e a fonte auxiliar é feita através de um seletor presente na placa.

A ponte H é controlada por sinais PWM (do inglês, *Pulse Width Modulation*) provenientes do processador. Assim, a corrente de saída pode ser controlada variando-se o *duty cycle* dos sinais PWM de controle. Os *drivers* de ponte H presentes no OpenOBC suportam diversos modos de operação como: *stand-by*, para a frente, reverso, freio e controle de PWM. Também são implementados diversos circuitos de proteção como: Circuito de condução cruzada, circuito de bloqueio de subtensão, circuito de proteção de sobretensão, circuito de desligamento térmico e circuito de sobre corrente (ROHM SEMICONDUCTOR, 2014).

3.4 Unidades de Armazenamento do OpenOBC

Ainda sobre a arquitetura de *hardware* do OpenOBC, foram utilizados dois tipos de memórias para armazenamento de dados, uma Flash com interface de comunicação SPI e um cartão SDcard, também fazendo uso da interface SPI. A memória Flash irá armazenar o sistema operacional do computador de bordo. Já o cartão armazenará os dados coletados pelo OBC. As memórias ROM e RAM internas do processador, pelo fato de possuírem detecção e correção de falhas, terão um papel muito importante, pois no ambiente espacial, os níveis de radiação podem causar uma mudança de bit (*bit-flip*), provocando ações inesperadas no comportamento do software, podendo acarretar travamento do sistema, ocasionando a operação de *reset* do *watchdog*. Na memória ROM será armazenado o sistema operacional como forma de *backup*. A memória RAM serve como memória temporária, enquanto os programas são executados, e

também como armazenamento de variáveis dos cálculos que estão sendo realizados.

A memória Flash utilizada foi a S25FL132K0XMFI041 do fabricante Cypress Semiconductor. Este circuito integrado possui 32 Mbit de armazenamento, *clock* de 108 MHz, padrões de temperatura industrial, automotiva e estendida, além de várias opções de seguranças tais como três registros de segurança de 256 bytes com proteção OTP (do inglês, *On Time Programmable*), proteção contra baixa tensão, recurso de proteção de segurança com base em ponteiros, etc (CYPRESS SEMICONDUCTOR, 2016).

3.5 Fontes de Alimentação do OpenOBC

Para alimentar o *hardware* do OpenOBC foi utilizada, por meio do conector H2 padrão da especificação CubeSat, através dos pinos 25 e 26, uma tensão de saída regulada de 5 V. A partir dessa tensão de entrada, o *hardware* deriva, por intermédio de um regulador chaveado, as tensões de 1,2 V para o núcleo do processador e 3,3 V para os demais periféricos, com exceção da ponte H, que é alimentada por uma tensão de 5 V, o qual utilizou a tensão de saída protegida proveniente dos pinos 49 e 51 do conector H1, conforme o padrão CubeSat.

As fontes chaveadas apresentam níveis consideráveis de ruído de alta frequência. Essa característica, embora pouco relevante para os circuitos digitais, é representativa para os circuitos analógicos de precisão, como conversores AD. Uma solução comum para esse problema é utilizar reguladores lineares para alimentar os circuitos sensíveis a tais sinais ruídos. No OpenOBC, no entanto, optou-se por utilizar um regulador chaveado seguido de um filtro PI (Proporcional Integrativo) (IDT, 2013) para derivar a tensão de 3,3V de baixo ruído, necessária para alimentar os circuitos de condicionamento de sinais. Com essa solução é possível atingir bom desempenho e baixo custo.

A capacidade de corrente da fonte de entrada deve ser definida segundo os requisitos da aplicação, dada a restrição de que a máxima corrente que o conversor DC/DC da placa pode regular é de 1 A.

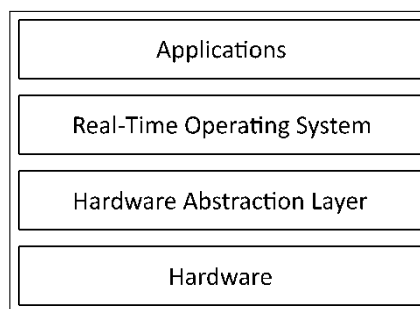
3.6 Software e device-drivers do OpenOBC

Como exposto anteriormente, os diversos requisitos e objetivos do projeto OpenOBC culminaram em uma arquitetura de *hardware* que atendesse às funcionalidades e peculiaridades de sistemas espaciais que, devido a sua natureza, são considerados sistemas críticos do ponto

de vista da confiabilidade. Desta forma, faz-se necessária uma arquitetura de *software* que implementasse estas funcionalidades em seus diversos níveis.

A arquitetura de *software* proposta baseia-se em um modelo de camadas em que as camadas inferiores prestam algum tipo de serviço para as camadas superiores. Através deste modelo, as funções e procedimentos de uma camada que fornece serviços a uma camada superior podem ter suas implementações modificadas sem que a esta precise ser alterada, o que garante ao software desenvolvido uma maior modularidade. A arquitetura foi dividida em três camadas: a camada de abstração do hardware (HAL, do inglês *Hardware Abstraction Layer*), o Sistema Operacional de Tempo Real (RTOS, do inglês *Real-Time Operating System*) e as aplicações. A Figura 8 ilustra a relação entre as camadas e o *hardware*.

Figura 8 – Distribuição dos módulos de *software* em camadas e sua distribuição sobre o *hardware*



A camada que interage diretamente com o hardware é a camada HAL. É nesta camada que residem os módulos de software conhecidos como *device-drivers* que controlam o funcionamento em mais baixo nível do microprocessador e seus periféricos. Devido a esta proximidade com o *hardware*, os módulos de *software* desta camada são conhecidos também como *firmware*. São eles que fornecem APIs (do inglês *Application Programming Interface*) que possibilitam as camadas superiores acessarem o *hardware* sem manipular detalhes sobre o seu funcionamento.

Os *device-drivers* podem ser de dois tipos em relação aos periféricos que estes acessam: internos ou externos. Os primeiros normalmente controlam periféricos de *hardware* que já vem integrados ao processador. São exemplos clássicos desses *device-drivers* os controladores de módulos de comunicações seriais e conversores analógico-digitais. Já o segundo tipo de *driver* controla dispositivos auxiliares que estão conectados externamente ao processador. Normalmente, um *device-driver* externo faz uso de algum *device-driver* interno para implementar sua funcionalidade. A Tabela 2 enumera os *device-drivers* desenvolvidos para o projeto bem

como sua classificação e utilização.

Tabela 2 – *Device-drivers* do projeto OpenOBC

<i>Device-driver</i>	Classificação	Descrição
UART	Interno	Envio e recebimento de dados seriais para auxiliar no desenvolvimento e depuração do sistema
CAN	Interno	Barramento com alta imunidade a interferências elétricas utilizada na comunicação com outros subsistemas
SPI	Interno	Barramento de comunicação serial do tipo mestre-escravo
GPIO	Interno	Controla os pinos de uso de geral do processador
ADC	Interno	Realiza leituras analógicas de sinais advindos de sensores que captam a intensidade do sol nas faces do satélite
PWM	Interno	Modulação de Largura de Pulso para as pontes H que controlam os magneto-torques
<i>Watchdog Timer</i>	Interno	Temporizador destinado a reinicializar o sistema em caso de falha
Ponte SPI/I2C	Externo	Periférico de conversão entre o barramento SPI e o I2C
Memória Flash	Externo	Dispositivo de armazenamento em massa
Micro SD Card	Externo	Dispositivo de armazenamento em massa
<i>Transponder</i>	Externo	Comunicação entre o satélite e as estações receptoras

A partir da Tabela 2 podemos notar a presença de *device-drivers* para os módulos de comunicação CAN, UART e I2C que fazem parte do padrão de conexão sugerido pelo documento *CubeSat Board Specification* (CUBESAT, 2016), padrão este que está sendo amplamente adotado pelos maiores fabricantes de nano-satélites.

O projeto OpenOBC faz uso de um sistema operacional de tempo real para realizar a gerência dos recursos computacionais e do escalonamento de tarefas. Ao contrário de sistemas operacionais de uso geral, que se preocupam principalmente em manter o sistema responsivo ao usuário, um RTOS possui como característica principal o atendimento de prazos de forma determinística possibilitando uma maior previsibilidade do comportamento das aplicações de tempo real executando sobre o OpenOBC. Esta característica é obtida através de algoritmos de escalonamento específicos e atribuição de prioridades às tarefas que serão executadas.

O sistema operacional de tempo real adotado para o projeto OpenOBC é o FreeRTOS. Diversos são os motivos que tornaram este RTOS um padrão na indústria sendo os mais proeminentes: código fonte de excelente qualidade, comunidade ativa de desenvolvedores, vasta documentação, baixo consumo de memória e licenciamento que não obrigam desenvolvedores a exporem seus códigos fontes.

O FreeRTOS fornece uma API que facilita o desenvolvimento de aplicações em que diversas tarefas são executadas de forma concorrente. São fornecidas funções e procedimentos para criação e manutenção das tarefas bem como mecanismos de comunicação entre elas. Outra vantagem associada ao uso de um RTOS é que o desenvolvimento do *software* pode ser melhor distribuído entre os diversos desenvolvedores que integram o projeto, possibilitando que cada um deles se concentre no aspecto principal da tarefa e não na temporização global do sistema.

3.7 Características de Tolerância a Falhas

Para alcançar a segurança funcional do sistema, o risco de perigos causados por sistemas elétricos e eletrônicos com defeito deve ser removido. Indústrias têm padrões de segurança funcionais para garantir que os sistemas relacionados com a segurança vão oferecer a redução do risco necessária para alcançar a segurança para o equipamento.

Com isso, o projeto do OpenOBC procura aliar tecnologias para adicionar características de tolerância a falhas, juntamente com o baixo custo proposto nesse trabalho em relação aos OBCs comerciais.

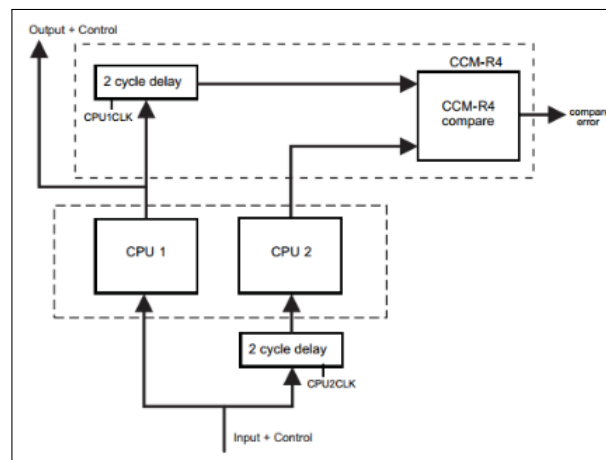
A seleção dos componentes eletrônicos foi baseada em características internas desses dispositivos que nos pudessem conceder um mínimo de tolerância a falhas como, por exemplo, redundâncias, correção de erros, *radhard*, limites de temperatura elevada, aplicação do componente, etc. Algumas dessas tecnologias foram implementadas devido a seu custo-benefício tal como redundância, mas outros não puderam pelo fato de ser inviável ao escopo do projeto, que foi o caso do *radhard*. Exemplos de componentes utilizados no OpenOBC que fazem uso de tecnologias de tolerância a falhas são o processador e a ponte H, entre outros.

Este processador foi projetado para ambientes automotivos, com foco em aplicações de segurança crítica, como por exemplo em aeronáutica. Assim como em operações no ambiente automotivo, em que não há baixa tolerância para erros, o TMS570LS0432 pode ser utilizado conforme os requisitos existentes em ambientes espaciais. Este processador conta com dois núcleos (redundantes) operando com o mesmo conjunto de operações em paralelo (*lockstep*). Assim as saídas das operações realizadas em cada núcleo podem ser comparadas, possibilitando detecção e correção de erros. A Figura 9 ilustra o funcionamento dos dois núcleos.

A memória RAM e a memória Flash internas do processador possuem recursos de redundância da informação armazenada, através do ECC (do inglês *Error-Correcting Code*) que possibilita a correção de dados incorretos armazenados no processador e mitiga as ocorrências de *bit-flip* nas memórias do sistema.

Rotinas de BIST possibilitam o diagnóstico de falhas antes do chip entrar efetivamente em operação, em que parâmetros são verificados e comparados. Já as falhas severas podem ser identificadas através de uma entrada do processador (*Error Pin*), a qual pode ser monitorada externamente como indicador de uma condição de falhas. Mensagens de erro no processador podem ser enviadas através de protocolos de comunicação como SPI e UART para indicar uma ocorrência de erro e possibilitar a tomadas de decisões no próprio sistema.

Figura 9 – Implementação dos dois núcleos do processador



Fonte – Texas Instruments (2015b).

Além disso, o processador realiza monitoramento do *clock* e das tensões de alimentação do componente, comparando os valores esperados com os valores coletados periodicamente.

Outro componente que possui aspectos de tolerância a falhas e é responsável pelo acionamento dos atuadores magnéticos para a movimentação do sistema, é o *driver* da ponte H com a referência BD6211, do fabricante Rohm Semiconductor, o qual fornece potência para o acionamento dos atuadores através de sinais PWM gerados pelo processador. O *driver* já fornece quatro formas de proteção que resguardam o sistema das falhas mais prováveis: OCP (do inglês *Over Current Protection*), OVP (do inglês *Over Voltage Protection*), TSD (do inglês *Thermal Shutdown*), UVLO (do inglês *Under Voltage Lock Out*) (ROHM SEMICONDUCTOR, 2014).

3.8 Requisito de Baixo Custo

O termo “Baixo Custo” é bastante disseminado e de interpretação aleatória, gerando dúvidas se há realmente ganho em optar por um produto ou outro quando não sabemos verdadeiramente o custo da matéria prima utilizada para implementar esses produtos.

O custo da matéria prima do OpenOBC é dividido em duas partes: o custo dos componentes eletrônicos e o custo da fabricação da PCI. Dado que este projeto foi desenvolvido com o objetivo de ser totalmente aberto, é possível adquirir toda lista de materiais com seus respectivos preços ao mesmo tempo em que se pode obter os arquivos de fabricação da PCI sabendo o valor real que está sendo empregado para a montagem de um protótipo de um OpenOBC. A lista dos componentes eletrônicos (BOM do inglês *Bill of Materials*) tem um custo de US\$ 65,07 no sitio de compras www.digikey.com e encontra-se disponível no Apêndice B. Já

a fabricação da PCI pode variar bastante o preço pelo fato de haver muitas empresas disponíveis no mercado nacional e internacional capazes de executar esse serviço.

3.9 OpenOBC: um projeto *Open Source*

O conceito de *Open Source* é uma aplicação cujos arquivos fontes de um determinado projeto estão “abertos”, ou seja, estão disponíveis para a comunidade, fazendo com que um usuário desta solução possa, além de fazer uso desses arquivos, também cooperar com a evolução e melhorias do projeto proposto.

Como a GNU License que representa o *software* livre, a CERN *Open Hardware License* representa o *hardware* livre. Em arquivos que estão protegidos sob essa licença, como a GPL (do inglês *General Public License*), garantem que o usuário final, ou seja, empresa, grupos ou pessoas, tenham a liberdade de estudar, rodar, compartilhar, copiar e modificar esse *software*. Esquemáticos e *Layouts* devem estar presentes entre os arquivos distribuídos em formato original, bem como os arquivos necessários para fabricação, e também disponíveis em formato para que todos possam acessar de forma fácil, como arquivo PDF.

A documentação que deve ser disponibilizada pelo projeto protegido sob licença CERN *Open Hardware License* (OHL) v1.2 devem possuir os seguintes arquivos:

1. Licença;
2. Guia da Licença;
3. Arquivo que contenha o contato no qual se pode receber mais informações sobre o produto, por exemplo, um arquivo chamado PRODUCT.TXT;
4. Arquivo que documenta todas as modificações realizadas nos arquivos até este ponto do projeto, por exemplo, um arquivo chamado CHANGES.TXT.

Nos arquivos que compõem a documentação de *hardware*, é importante que se coloque um cabeçalho dizendo que esses arquivos estão protegidos sob essa licença. É sugerido o seguinte texto:

Copyright CERN 2013.

This documentation describes Open Hardware and is licensed under the CERN OHL v. 1.2. You may redistribute and modify this documentation under the terms of the CERN OHL v.1.2. (<http://ohwr.org/cernohl>). This documentation is distributed WITHOUT ANY EXPRESS OR IMPLIED WARRANTY, INCLUDING OF MERCHANTABILITY, SATISFACTORY QUALITY AND FITNESS FOR A PARTICULAR PURPOSE. Please see the CERN OHL v.1.2 for applicable

conditions.

Em uma parte visível da placa, coloque a seguinte frase escrito, por exemplo, no lado de cima da placa de circuito impresso:

Licensed under CERN OHL v.1.2

Caso sejam feitas alterações nesses arquivos, o que implica em trabalho que derivam desse *hardware*, esses arquivos somente poderão ser distribuídos sob essa mesma licença. É necessário documentar as modificações e manter todas as referências de autoria e de licença intactas (LIMA, 2016).

Com isso, o projeto OpenOBC visa contribuir com a comunidade acadêmica oferecendo todo o seu conjunto de artefatos em pesquisa e desenvolvimento desde o entendimento em sistemas embarcados tolerantes a falhas para aplicações espaciais até a construção do seu próprio protótipo, sendo necessário somente implementar a aplicação na qual o OBC proposto irá realizar. Conseqüentemente, este usuário irá relatar suas experiências e acrescentar novos códigos fontes ou implementações de *hardware* e assim fomentar ainda mais com um dos objetivos deste projeto, impulsionando a área de engenharia espacial no mercado brasileiro. O *link* para ter acesso a toda documentação do projeto está disponível em (MOTA D.; AMERICO, 2016a).

Após exposto toda a metodologia de desenvolvimento do OpenOBC, será apresentado, no próximo capítulo, os resultados pertinentes a arquitetura proposta.

4 RESULTADOS DO OPENOBC

No Capítulo 4 será exposto a metodologia empregada na validação dos circuitos eletrônicos através de medição de sinais, no funcionamento dos *device-drivers* desenvolvidos e do *porting* do freeRTOS para o OpenOBC. Como resultado final será exibido as técnicas principais de roteamento utilizadas na PCI do OpenOBC.

4.1 Validação da fonte de alimentação e Sinais vitais do processador

Como em todo sistema embarcado, é necessário iniciar a depuração da PCI através da validação da fonte de alimentação. O regulador de tensão chaveado utilizado no OpenOBC é o TPS62402 do fabricante Texas Instruments que é recomendado pelo processador, do mesmo fabricante, para aumentar a compatibilidade evitando qualquer mau funcionamento. Este regulador, conforme explicado no capítulo 3, possui duas saídas que foram projetadas para regular suas tensões em 3,3 V e 1,2 V. Essas saídas possuem um resistor de 0 Ohm (*jumper*) em cada uma delas, com o objetivo de isolar as tensões dos seus respectivos destinos. Essa isolação se faz necessária para testar a fonte e verificar se está realmente de acordo com o projetado, antes de injetar as tensões em suas respectivas cargas e assim não danificar o restante dos circuitos. Em caso de bom funcionamento, os resistores serão soldados e a energia será alcançada em toda placa e pronta para continuar a depuração do restante do sistema. A Figura 10 mostra os níveis de tensões adequados ao projeto garantindo a normalidade deste circuito.

Com a fonte em perfeitas condições, o próximo passo é a verificação dos sinais vitais do processador que são o *reset* e o *clock*. O TMS570LS0432 possui dois tipos de *reset*. O *reset* padrão, denominado por nRST, no qual é necessário um circuito externo para gerar a reinicialização do sistema como mostrado na Figura 11. É recomendado pelo fabricante que seja utilizado um resistor de *pull-up* neste circuito para não ser gerado um *reset* arbitrariamente (TEXAS INSTRUMENTS, 2015b). Já o outro *reset* é proveniente da falha da fonte de tensão. O sinal dele proveniente é o nPORRST, e para acioná-lo é necessário um circuito capaz de identificar uma divergência de tensão de alimentação na fonte fora do intervalo especificado e assim resetar o processador evitando danos no sistema quando se há sub ou sobre tensão. A Figura 12 ilustra o circuito projetado para essa tarefa. Esses dois sinais possuem um filtro de *Glitch* interno ao processador que tem como objetivo eliminar picos de tensão com pequenos intervalos de tempo do sinal de entrada (TEXAS INSTRUMENTS, 2015b).

Figura 10 – Tensões de alimentação do OpenOBC

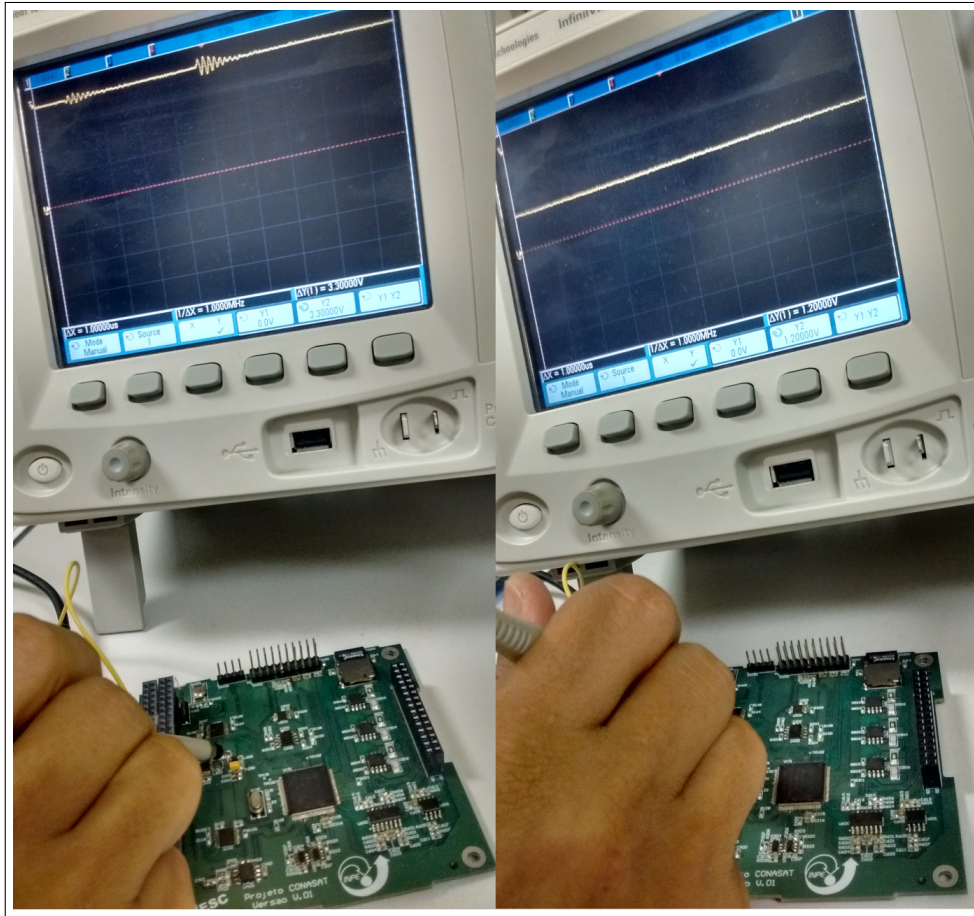
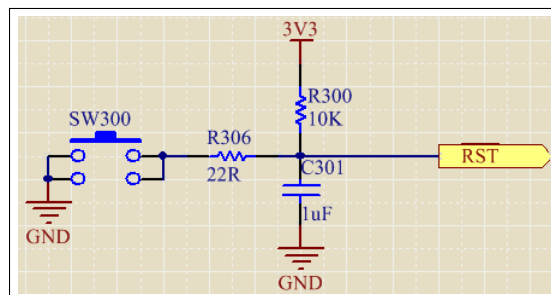


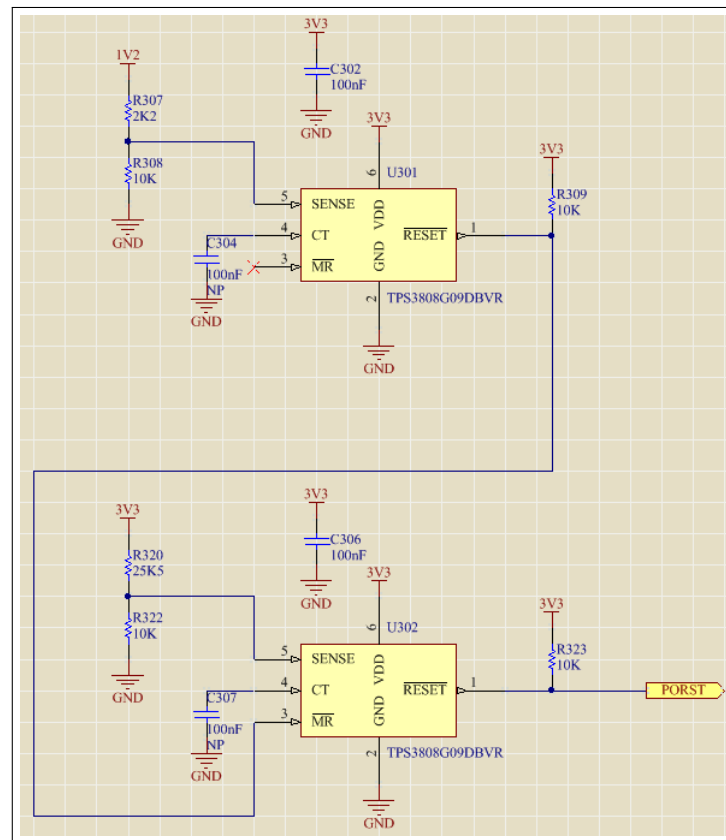
Figura 11 – Circuito projetado para a geração do sinal nRST



O processador do OpenOBC possui em sua arquitetura interna a opção de utilizar um oscilador ou um cristal para a geração do *clock*. Neste projeto foi utilizado um cristal para fazer uso da funcionalidade existente no TMS570LS0432 que é um pino conectado ao terra dedicado ao uso em cristais e assim evitar qualquer ruído que seja proveniente do terra do sistema (TEXAS INSTRUMENTS, 2015b).

Tanto o circuito de *reset* quanto o do *clock* foram testados utilizando um osciloscópio, operando conforme o esperado e seus sinais de funcionamento estão ilustrados na Figura 13. O

Figura 12 – Circuito projetado para a geração do sinal nPORRST



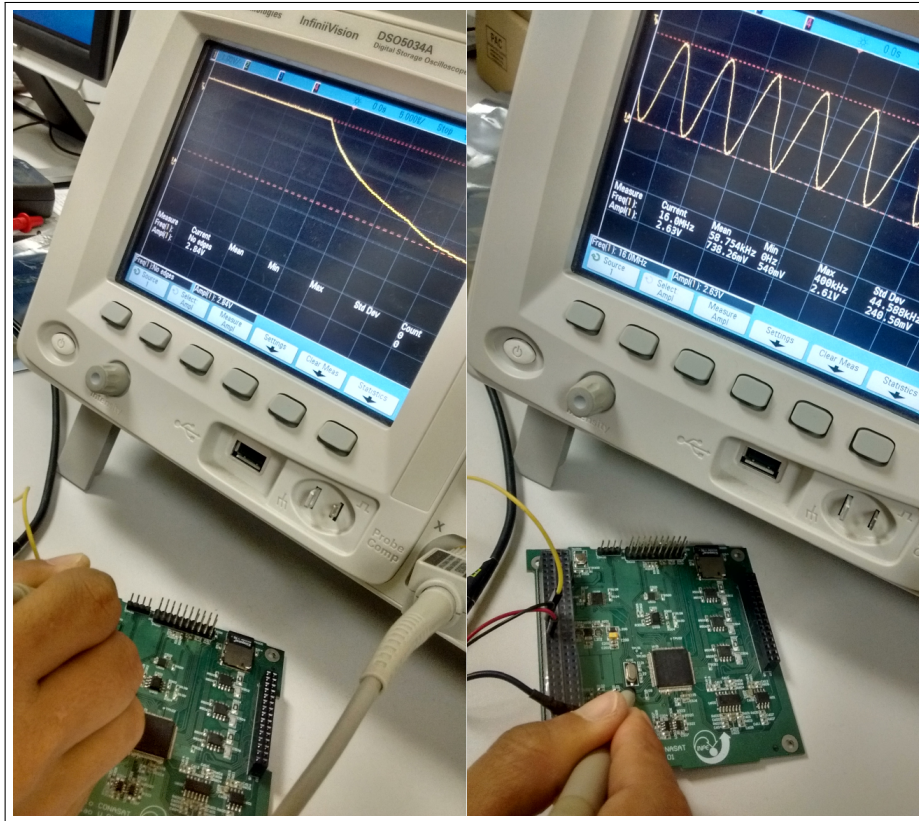
o sinal de nRST faz a transição do nível lógico alto para baixo quando o sistema é iniciado ou quando pressionada a chave e o sinal de nPORRST cai para o nível lógico baixo assim que ele detecta uma variação inesperada na tensão de alimentação. O *clock* proveniente do cristal opera em 16 MHz.

4.2 Comunicação JTAG

Após a validação da fonte e dos sinais vitais do processador, é preciso fazer uso da comunicação JTAG (do inglês *Joint Test Access Group*) que tem como função realizar a programação e testes de circuitos digitais tais como processadores e microcontroladores. Para isso, foi utilizado o Emulador XDS100 do fabricante Texas Instruments que, ao conectá-lo ao OpenOBC através da interface JTAG, efetua a programação do processador e ainda possui a funcionalidade de *debugger* para depuração do código implementado.

O Ambiente de desenvolvimento (IDE, do inglês *Integrated Development Environment*) usado para a codificação dos *devices-drivers* foi o *Code Composer Studio* (CCS). Neste *software*, antes de iniciar a gravação do processador, é necessário configurá-lo estabelecendo

Figura 13 – Sinais de nRST e de clock



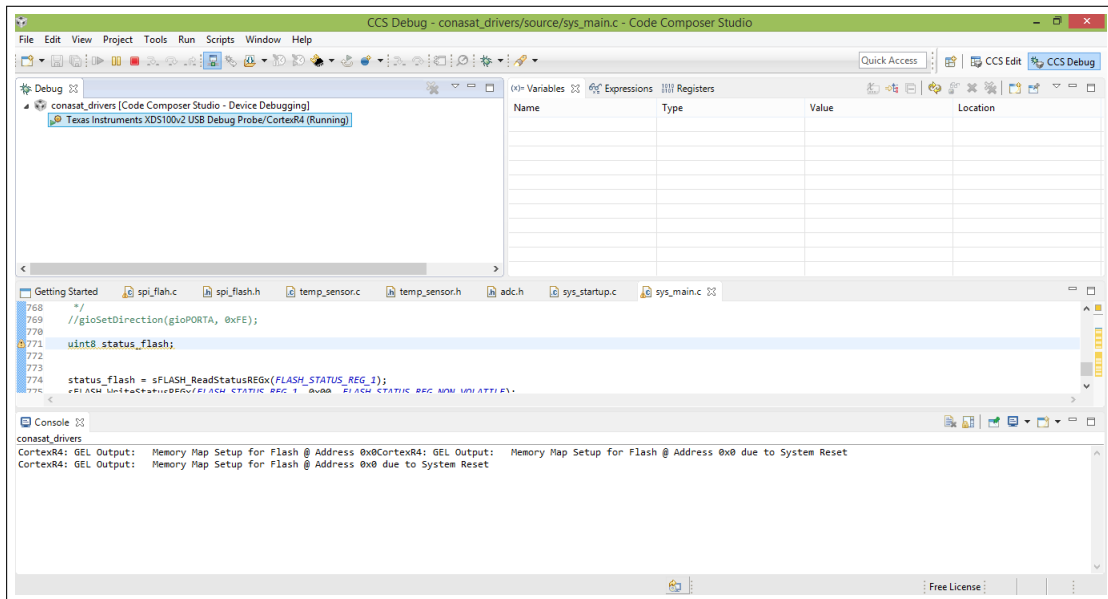
alguns parâmetros tais como a referência do processador a ser gravado e a interface que será utilizada na gravação para, em seguida, iniciá-la de fato. A Figura 14 ilustra a IDE utilizada e um programa em execução, validando assim a comunicação JTAG do OpenOBC.

4.3 Comunicação SPI

A SPI é um protocolo serial síncrono do tipo mestre-escravo que permite a comunicação entre o processador (mestre) e os periféricos externos (escravos) através de um barramento. Ele é dito síncrono por que necessita que os dispositivos mestre e escravos estejam sincronizados através de um sinal de *clock* comum gerado pelo mestre durante as transações de envio e recepção de dados (SACCO, 2014).

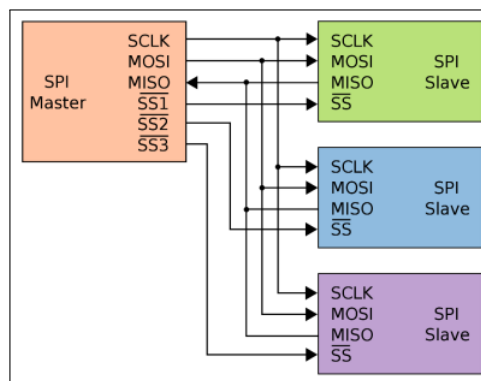
O barramento SPI é composto de quatro sinais: MISO (do inglês *Master In Slave Out*), MOSI (do inglês *Master Out Slave In*), CLK (do inglês *Clock*), CS (do inglês *Chip Select*). Na arquitetura interna do processador é delimitada uma quantidade de CSs para cada canal de SPI e com isso, é possível utilizar, numa mesma SPI, várias comunicações com dispositivos escravos, dado a quantidade de CSs existente (SACCO, 2014). Normalmente, cada ativação de

Figura 14 – A IDE CCS rodando um código para demonstração da validação da comunicação JTAG e gravação do processador



um CS é uma transação SPI. A Figura 15 exemplifica um barramento SPI. Notar que o sinal de CS possui um nome alternativo chamado de SS (do inglês *Slave Select*). A barra horizontal sobre o sinal SS denota sua ativação e nível lógico baixo.

Figura 15 – Barramento SPI com três escravos

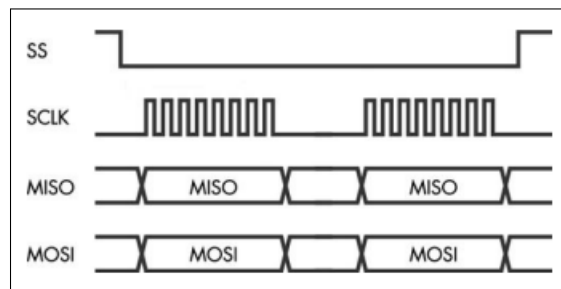


Fonte – Sacco (2014).

A cada transação SPI, os dados são transmitidos e/ou recebidos por meio de palavras de tamanho pré-definido nos *datasheets* de cada um dos escravos. Cada palavra de dados que é transmitida terá um significado peculiar para cada escravo. Logo, apesar de o barramento ser padronizado, não existe padronização alguma sobre os dados que serão trafegados nele. A maioria dos processadores permitem a configuração do tamanho das palavras de dados. Normalmente, este tamanho é configurado para 8 ou 16 bits. Vale ressaltar também que, em

muitos casos, quando o mestre está transmitindo dados na linha MOSI, alguns desses dados também poderão ser colocados na linha MISO. A Figura 16 ilustra uma transação SPI que, para este caso específico, o sinal SS permaneceu ativo durante a transmissão das duas palavras de dados.

Figura 16 – Transação SPI transmitindo/recebendo duas palavras de dados, cada uma com 8 bits.



Fonte – Sacco (2014).

Mesmo sendo um barramento padronizado e amplamente adotado, a SPI ainda possui alguns detalhes que devem ser observados antes de iniciar a comunicação propriamente dita. O primeiro deles é a escolha do modo de operação do mestre SPI. Estes modos de operação estão ligados a duas características do sinal de *clock*: polaridade e fase. A primeira determina qual o nível lógico do sinal do *clock* quando o mestre não está se comunicando com nenhum escravo, já a segunda determina se os dados presentes no MOSI e no MISO serão amostrados na borda de subida ou de descida do *clock* (SACCO, 2014). Para conhecer o modo correto de operação dos dispositivos escravos deve-se consultar o *datasheet* de cada um deles.

Outro fator que deve ser observado durante a utilização do barramento SPI é o conjunto de temporizações que devem ser seguidas para cada um dos escravos. Por exemplo, para um determinado escravo o sinal de CS deve ser ativado (nível lógico baixo) 100 ns antes de o *clock* começar a pulsar. Além disso, deve-se observar a frequência máxima suportada em cada escravo. Por fim, deve-se verificar no *datasheet* de cada escravo se o CS estará ativo na transmissão de uma palavra ou durante a transação inteira, que poderá conter mais de uma palavra de dados.

No projeto OpenOBC foram utilizados três módulos SPI master: SPI1, SPI2 e SPI3. Desta forma, a Tabela 3 mostra como estão distribuídos os dispositivos escravo nos módulos SPI e CS disponíveis.

Estes módulos foram criados através do desenvolvimento de três arquivos que compõem o driver da SPI. O arquivo **reg_spi.h** contém a definição dos endereços base de acesso

Tabela 3 – Mapeamento dos módulos SPI do processador nos dispositivos escravos do projeto OpenOBC

Periférico	SPI	CS
Ponte SPI/I2C <i>Transponder</i>	SPI2	CS2
Ponte SPI/I2C Subsistemas	SPI3	CS0
Memória Flash	SPI1	CS1
Micro SD Card 4	SPI1	CS0
Sensor de temperatura	SPI1	CS2

ao módulo SPI. É neste arquivo que está definida a estrutura que engloba todos os registradores do módulo SPI. Outro arquivo é o **spi.h** que é o arquivo de definição do *driver*, definindo algumas constantes e os protótipos das funções implementadas no arquivo **spi.c**. E por fim, o arquivo **spi.c**, que é o arquivo de implementação do *driver* propriamente dito. As funções contidas neste arquivo e toda a documentação estão disponível em (MOTA D.; AMERICO, 2016a) no formato Doxygen.

Para testar o *driver* gerado para a SPI, configurou-se dois formatos para SPI2 e verificaram-se as formas de onda para os sinais MOSI, *Clock* e CS com o auxílio de um osciloscópio digital. O primeiro formato (Format 0) testado possui uma frequência de 1 MHz, 8 bits por palavra de dados e nível lógico alto para a polaridade do clock. Já o segundo formato (Format 1), possui uma frequência de 500 KHz, 16 bits por palavra de dados e a polaridade do *clock* em nível lógico baixo.

Com isso, foram testados todos os periféricos conectados ao barramento SPI. A memória Flash, por ser o elemento que irá armazenar todo o código desenvolvido (*devices-drivers* e FreeRTOS) teve que ser testada de diferentes maneiras para garantir que ela seja tolerante a falhas e que a incidência de erros seja o mínimo possível. Foi desenvolvida uma tabela de testes, cujos resultados foram bem sucedidos. Todos os testes foram desenvolvidos baseados em parâmetros encontrados no *datasheet* da memória (CYPRESS SEMICONDUCTOR, 2016). A Tabela 4 ilustra os testes executados na memória Flash do projeto OpenOBC.

O cartão Micro SD Card foi testado de tal maneira que o *driver* desenvolvido, primeiramente, verifica a presença do cartão na placa do OpenOBC. Após o cartão ser detectado, a inicialização, a nível de protocolo, é inicializada. O protocolo SD consiste no envio de comandos para o cartão que enviará respostas de certos tipos dependendo do comando enviado pelo *host*.

A seleção do modo SPI é realizada na inicialização do cartão quando o comando

Tabela 4 – Relação de testes executados na memória Flash do projeto OpenOBC

Caso de Teste	Descrição
1	Leitura e escrita dos registros de status
2	Escrita volátil e não-volátil
3	Escrita alinhada em uma página
4	Escrita alinhada em duas páginas
5	Escrita desalinhada em uma página
6	Escrita desalinhada em duas páginas
7	Escrita desalinhada em três páginas
8	Escrita não bloqueante
9	Suspensão de <i>erase</i> e <i>write</i>
10	Teste dos modos de proteção
11	Modo de proteção por ponteiro

CMD0 é enviado. Após o envio do CMD0, um comando CMD8 é enviado para verificar as condições de operação da interface do cartão. Estas condições são checadas quando se enviam determinados parâmetros (e.g. tensão de operação) no CMD8 e o cartão responde corretamente. Através desta resposta se conclui que o cartão ou é do tipo V1.0 ou do tipo V2.0+. Após isso, o processo de inicialização do cartão é disparado (ACMD41) e sua classe de capacidade obtida (CMD58). Caso algum destes comandos falhe, a função de inicialização retorna um código de erro para a função chamadora.

Para a escrita de um bloco se utilizou o comando CMD24. Este comando requer o envio de um *Start Block Token* antes do envio dos dados a serem gravados. Após o envio dos dados, envia-se dois *bytes dummy* de CRC. Por fim, espera-se um *Response Token* para verificar se os dados foram aceitos. Os processos de leitura e apagamento seguem estratégias de comunicação similares.

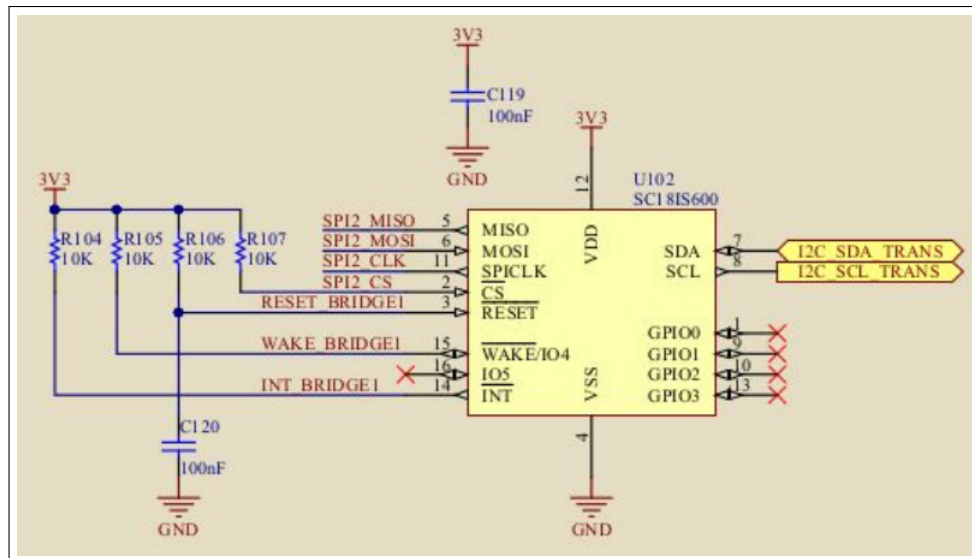
Já os testes do sensor de temperatura foram realizados utilizando um ferro de solda em aproximação do circuito integrado, o qual implementa este sensor e com isso foi verificado, através da leitura do processador, o valor real da temperatura. O valor lido é representado no formato Q9.2 e é descartado os 5 bits menos significativos, que sempre são lidos como 1 e não trazem informação sobre a temperatura (TEXAS INSTRUMENTS, 2016).

4.4 Comunicação I2C

Um das principais funções do projeto OpenOBC é fornecer conectividade I2C a um *transponder* e outros subsistemas que possam se acoplar ao computador de bordo. Para prover essa funcionalidade, o OpenOBC disponibiliza duas pontes SPI/I2C de referência SC18IS600

do fabricante NXP Semiconductors que implementam o barramento I2C com os dispositivos escravos. A utilização dessas pontes se fez necessária devido à ausência de um controlador I2C interno no processador selecionado. A Figura 17 apresenta a parte do esquema elétrico que contém uma das pontes SPI/I2C e seus sinais de entrada e saída.

Figura 17 – Esquema elétrico da ponte SPI/I2C com seus sinais de comunicação



A comunicação entre o processador e a ponte é feita através da interface SPI composta pelos seguintes sinais: SPI2_MISO, SPI2_MOSI, SPI2_CLK e SPI2_CS. O SC18IS600 apresenta algumas restrições quanto às configurações dos sinais desta interface SPI. A primeira delas é em relação ao modo de operação da SPI que diz respeito às configurações de polaridade e fase do *clock*. Este componente aceita uma comunicação SPI de modo 3 que indica que o sinal de *clock* é sempre alto/baixo quando a SPI está ociosa e que a amostragem dos dados é realizada a cada borda de subida/descida do sinal de *clock*. O sinal CS também apresenta configurações especiais para que a comunicação com a ponte SPI/I2C se dê de maneira correta. A primeira delas diz a respeito à ativação ou desativação do CS entre a transmissão de palavras de dados consecutivas. O SC18IS600 necessita que, em uma mesma transação SPI, o sinal de CS não seja desativado, isto é, permaneça em nível lógico baixo. Outra configuração importante do sinal CS diz respeito aos tempos de *hold* e *setup* que deverão ser atendidos. Estes tempos podem ser configurados no controlador SPI presente no processador TMS570LS0432.

Além da interface de acesso para mestre SPI qualquer, o SC18IS600 apresenta alguns GPIOs (do inglês *General Purpose IN/OUT*) auxiliares. O sinal RESET_BRIDGE é ativo em nível lógico BAIXO e é responsável por reiniciar o *hardware* dos barramentos SPI e I2C e

colocar os registradores internos do SC18IS600 em valores padrões. Neste sinal estão ligados um resistor de *pull-up* e um capacitor que levam a ponte de um nível lógico baixo até um nível lógico alto assim que o OpenOBC é ligado. Isso garante que as pontes estejam em um estado conhecido nos primeiros momentos da operação do OpenOBC. Estes sinais de *reset* também são ligados em GPIOs que podem executar o *reset* através do *firmware* a qualquer momento. Outro sinal utilizado na inicialização das pontes SPI/I2C foi o WAKE_BRIDGE. Quando este sinal está ativo, o SC18IS600 é retirado do estado de desligamento. Os sinais de WAKEUP também são ligados em resistores de *pull-up* a fim de que estes possuam um estado lógico alto conhecido durante o *power-up* do sistema. Estes sinais de *reset* também são ligados em GPIOs que podem ligar as pontes através do *firmware*.

Por último, tem-se o sinal INT que comunica o processador a existência de uma interrupção no SC18IS600 que precisa ser servida. Este sinal é ativo em nível lógico baixo e por este motivo possui em seu pino um resistor de *pull-up*. Além disso, este sinal está conectado em um GPIO de entrada no processador do OBC. Este GPIO está configurado de modo que o processador sofra uma interrupção na transição do nível lógico alto para o nível lógico baixo.

Com isso, foram desenvolvidas algumas funções que implementam essa comunicação e toda documentação e arquivos do *device-driver* desta interface encontra-se disponível em (MOTA D.; AMERICO, 2016a) no formato Doxygen.

Durante o desenvolvimento do *device-driver* foram realizados testes que pudessem minimizar a existência de erros de codificação e, conseqüentemente, erros na comunicação com as pontes SPI/I2C e nas operações de leitura e escrita dos dispositivos escravos do barramento I2C.

O teste advém do uso do periférico mestre da SPI presente no processador e consiste na observação dos sinais presentes nos pinos que compõem a interface SPI de acesso às pontes. Esta observação foi realizada utilizando-se um osciloscópio com canais digitais de forma que os sinais da SPI, bem como os outros sinais auxiliares, pudessem ser capturados todos ao mesmo tempo. Desta forma, todas as transações SPI e seus conteúdos (dados transmitidos e recebidos) puderam ser comparados com aqueles considerados ideais pelo *datasheet* da ponte SPI/I2C.

Após a constatação de que os dados estavam trafegando corretamente pelo barramento SPI, o barramento I2C de saída também foi analisado. Foram verificadas as formas de onda dos sinais SDA e SDL a fim de verificar que estes estavam de acordo com o protocolo I2C. Foram analisadas as condições de início e fim de transmissão, os reconhecimentos

(*acknowledgments*), endereçamento dos escravos e o conteúdo dos pacotes transmitidos.

Por fim, para aplicar o *device-driver* desenvolvido, foi conectado ao OpenOBC um sensor de luminosidade TSL2561. Para se certificar do bom funcionamento dos dados ligou-se este sensor a um kit desenvolvimento Arduino e utilizou-se uma API disponível na comunidade para ler o sensor. Observou-se a saída do sensor em diversas situações diferentes de luminosidade. Após isso, um código similar foi utilizado no OpenOBC para ler o sensor. Constatou-se que a faixa de valores era bem próxima para todos os cenários de luminosidade.

4.5 Conversor Analógico Digital

Existe, na arquitetura interna do processador do OpenOBC, canais de ADC que, por sua vez, possuem três grupos de conversão, os quais são ordenados de acordo com a prioridade de amostragem: EventGroup, Group 1 e Group 2. A cada grupo de conversão pode ser associado um conjunto qualquer de canais de forma que todos esses canais sejam amostrados a partir do mesmo evento de disparo. Os grupos Group 1 e Group 2 podem ser também disparados por *software* ou por oito eventos que são específicos para cada dispositivo. No caso do TMS570LS0432 os eventos estão relacionados principalmente com o pino de disparo ADEVT e alguns pinos do HET (TEXAS INSTRUMENTS, 2015b). Por último, as configurações tais como o modo de conversão (conversão única ou contínua), o tempo de amostragem e o de geração de interrupções, podem ser configuradas individualmente para cada grupo de conversão.

A conversão realizada pelo ADC se dá em dois períodos: um período de amostragem e um período de conversão. Um período de amostragem é iniciado quando o sequenciador gera um sinal interno de START para o núcleo do ADC. Esse sinal é ativado quando este passa do nível lógico alto para o nível lógico baixo. Desta forma, enquanto o sinal de START estiver ativado, pode-se considerar que o ADC está em um período de amostragem. O tempo total em que o sinal de START fica ativo é determinado pelos registradores de controle de tempo de amostragem (ADEVSAMP, ADG1SAMP e ADG2SAMP). Após o fim do tempo de amostragem o sinal de START é desativado e após um período de ADCLK o ADC entra no modo de conversão. Durante esta fase, o núcleo do ADC finaliza a conversão de um bit por ACLK, ou seja, se o ADC está configurado para 12 bits, o núcleo do ADC levará o tempo de $12 \times \text{ADCLK}$ para converter os 12 bits. Ao final do período de conversão o núcleo do ADC sinaliza ao bloco sequenciador o fim da conversão através do sinal de EOC (do inglês *End-Of-Conversion*).

A faixa de conversão do ADC é baseada nas tensões presentes em dois pinos do pro-

cessador: ADREFHI e ADREFLO. Eles são o maior e o menor valor que podem ser convertidos pelo ADC, respectivamente. Para valores analógicos maiores que ADREFHI a leitura digital apresentada será 0x3FF (resolução de 10 bits) ou 0xFFF (resolução de 12 bits). Para valores iguais ou menores que ADREFLO a conversão será lida como 0x000. As Equações 4.1 e 4.2 determinam os valores digitais em função dos valores analógicos para cada uma das resoluções, cuja a escolha é definida em projeto.

$$\text{Resultado Digital} = \frac{1024 \times (\text{Tensao de Entrada} - \text{ADREFLO})}{\text{ADREFHI} - \text{ADREFLO}} - 0,5 \quad (4.1)$$

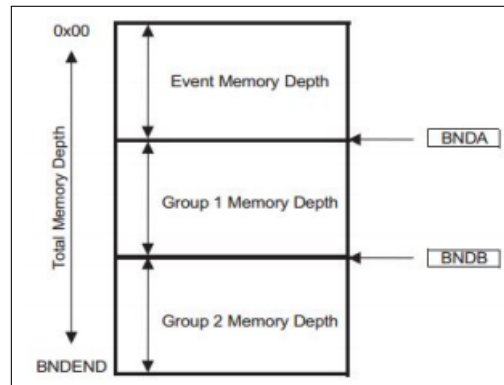
$$\text{Resultado Digital} = \frac{4096 \times (\text{Tensao de Entrada} - \text{ADREFLO})}{\text{ADREFHI} - \text{ADREFLO}} - 0,5 \quad (4.2)$$

Os resultados das conversões são armazenados em uma memória do tipo FIFO (do inglês *First In First Out*) de 64 palavras. Essa memória é dividida em três regiões, uma para cada grupo de conversão do ADC. A forma como este espaço de 64 palavras está distribuída para cada um dos grupos de conversão pode ser configurada através de software utilizando-se os registradores ADBNDCR e ADBNDEND. O registrador ADBNDCR possui dois ponteiros de 9 bits chamados de BNDA e BNDB que apontam, respectivamente, para o fim da memória reservada para o EventGroup e para o fim da memória reservada para o EventGroup somado à quantidade de espaços reservados ao Group 1. Por fim, o registrador ADBNDEND contém um campo chamado de BNDEND que configura a memória total disponível. Deve-se notar que BNDA e BNDB são referenciados a partir do começo da memória de resultados e são dados em múltiplos de dois *buffers*. A memória suporta um total de 1024 *buffers*. A Figura 18 ilustra a distribuição da memória de resultados.

O acesso aos dados produzidos pelo ADC pode ser realizado através de dois mecanismos: leitura através da FIFO ou leitura direta. No primeiro caso utiliza-se uma FIFO para tirar proveito da instrução LDM (do inglês *Load Multiple*). Já no segundo caso lê-se diretamente o mapa de memória do processador a partir do endereço 0xFF3E0000.

O módulo ADC possui basicamente dois tipos de interrupções: as básicas e as de limiar. As básicas são disparadas normalmente em situações que envolvem a leitura de valores já adquiridos pelo ADC. Já as de limiar são disparadas quando os valores das conversões atingem determinado valor, seja ele fixo ou advindo de outro canal. São possíveis até três interrupções de limiar. A Tabela 5 apresenta as interrupções básicas.

Figura 18 – Distribuição dos espaços das memórias de resultados a partir da definição dos registradores ADBNDCR e ADBNDEND



Fonte – Texas Instruments (2015b).

Tabela 5 – Interrupção básicas relacionadas aos eventos de leitura das conversões e ao acesso à memória de resultados

Nome da Interrupção	Descrição
<i>Group Conversion End Interrupt</i>	Disparada quando as conversões de todos os canais em um grupo de conversão é finalizada
<i>Group Memory Threshold Interrupt</i>	Disparada quando um determinado número de conversões atinge um limiar pré-definido
<i>Group Memory Overrun Interrupt</i>	Gerada quando o número de conversões ultrapassa a quantidade de espaços na memória para um dado grupo de conversão

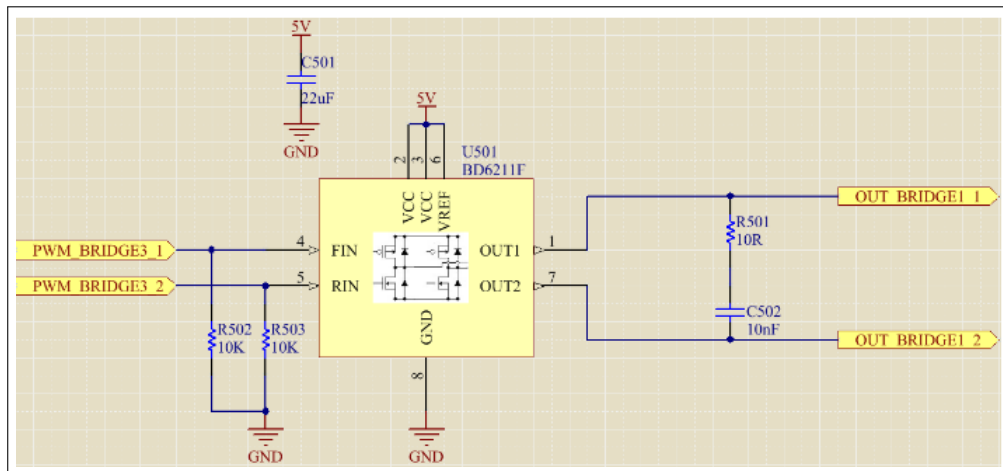
Para as interrupções de limiar, diversos parâmetros podem ser configurados de forma a realizar diversos tipos de configuração. O primeiro campo, denominado CHN_THR_COMP, indica se um canal específico será comparado com outro canal ou com um valor de limiar pré-definido. Já os campos MAG_CHID e COMP_CHID identificam, respectivamente, o canal que será monitorado e o canal com o qual este último será comparado. Além disso, o campo MAG_THR define o limiar de comparação caso o canal identificado por MAG_CHID esteja configurado para ser comparado com um limiar. Por último, o campo CMP_GE_LT define se a comparação será do tipo "maior ou igual que" ou "menor que" (TEXAS INSTRUMENTS, 2015b).

Foram desenvolvidos *devices-drivers* para as três situações comentadas acima. Na primeira delas realizou-se através de *polling* a leitura da conversão de apenas um canal do Grupo de Conversão 1. Já no segundo projeto realizou-se a leitura de dois canais simultaneamente através da leitura do campo que identifica o canal que se está lendo. Por fim, no terceiro projeto, habilitou-se a interrupção do ADC e aguardou-se o recebimento de novas amostras através do mecanismo de notificações. Toda essa implementação está disponível em (MOTA D.; AMERICO, 2016a) no formato Doxygen.

4.6 Sinais PWM

O *hardware* do projeto OpenOBC possui três saídas de ponte H que são responsáveis por energizar os *magnetotors* que controlam a atitude do nano-satélite. A comunicação entre o processador e as pontes H é feita através de sinais PWM, conforme ilustrado na Figura 19.

Figura 19 – Circuito de acionamento dos *magnetotors* através de sinais PWM



O circuito integrado BD6211F possui diversos modos de operação de acordo com os valores de entrada fornecidos em FIN e RIN. O modo de operação *Stand-by* faz com que toda circuitaria interna fique desligada, logo a saída da ponte vai para o estado de alta impedância. O modo *Forward* faz com que a saída da ponte H execute rotação progressiva. Os sinais OUT1 e OUT2 permanecem nos níveis lógicos alto e baixo, respectivamente. O modo *Reverse* faz com que a saída da ponte H execute rotação retrógrada. Os sinais OUT1 e OUT2 permanecem nos níveis lógicos baixo e alto, respectivamente. O modo *Brake* é utilizado para parar o acionamento rapidamente e, diferentemente do modo *Stand-by*, a circuitaria interna não é desligada (ROHM SEMICONDUCTOR, 2014).

Para executar estes modos de funcionamento, as entradas do BD6211F devem receber sinais PWM de dois modos. No modo A, um dos sinais de entrada (FIN ou RIN) é mantido em nível lógico baixo enquanto o outro está chaveando em modo PWM. A frequência do PWM deve estar entre 20KHz e 200KHz. Já no modo B, um dos sinais de entrada (FIN ou RIN) é mantido em nível lógico alto enquanto o outro está chaveando em modo PWM. A frequência do PWM deve estar entre 20KHz e 200KHz (ROHM SEMICONDUCTOR, 2014).

O módulo HET (do inglês *High-End Timer*), que possui a capacidade de gerar sinais de saída que podem ter suas temporizações configuradas, foi utilizado para a geração dos

sinais PWM necessários ao controle dos *magnetotorquers*. Através do HET, foi configurado a polaridade, o *duty cycle* e o período dos sinais PWM gerados (TEXAS INSTRUMENTS, 2015b).

Como teste deste circuito, foi escrito um código que realizasse as chamadas das funções variando os diversos parâmetros de temporização dos sinais PWM, os quais devem controlar as pontes H. Para verificar a correta execução das funções, executou-se o código passo-a-passo através do mecanismo de depuração do ambiente de desenvolvimento e, a cada passo, os sinais foram verificados com o auxílio de um osciloscópio digital. O código para este projeto está disponível em (MOTA D.; AMERICO, 2016a) no formato Doxygen.

4.7 Comunicação CAN

Para a realização de comunicações robustas, foi disponibilizado no OpenOBC a interface CAN. O padrão CAN é um protocolo padronizado pela ISO que foi criado pela indústria automobilística com o objetivo de simplificar a ligação física dos componentes da rede CAN. Ele é um barramento serial com alta imunidade às interferências elétricas, baseado em sinais diferenciais e com a capacidade de autodiagnosticar e reparar erros de dados (CIA: CONTROLLER AREA NETWORK, 2016).

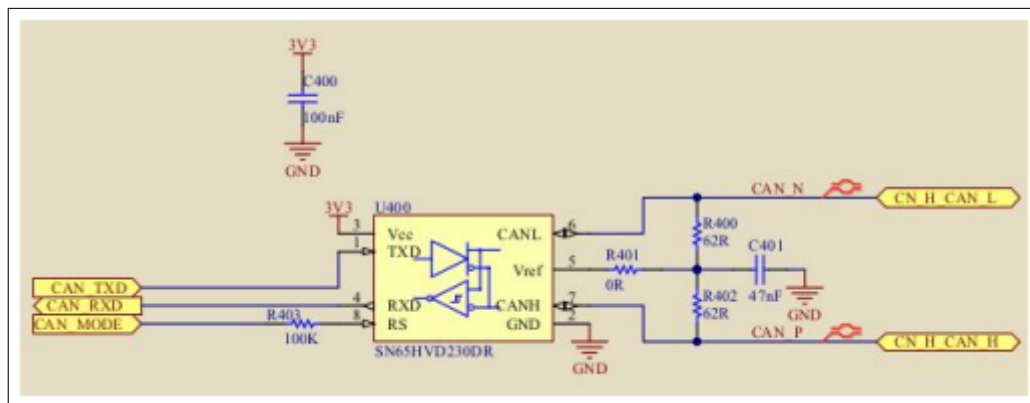
Basicamente, o protocolo CAN possui quatro tipos de mensagens. O primeiro tipo é chamado de *Data Frame* que é o tipo mais comum e trafega dados entre os nós. O segundo tipo é conhecido como *Remote Frame* e é utilizado para solicitar dados de outros nós. Já o *Error Frame* é enviado quando uma mensagem recebida fere as regras de formatação do CAN. Após o envio do *Error Frame* o nó que transmitiu a mensagem formatada incorretamente irá executar uma retransmissão. Por fim, o *Overload Frame* é utilizado para avisar a rede que o nó está ocupado e precisa de um tempo maior entre as mensagens.

O pacote CAN possui uma série de campos que podem ser divididos em campos de controle ou de dados. Fazem parte dos campos de controle os bits que compõem o Identificador da Mensagem, o *bit RTR* (indica se a mensagem é ou não um *Remote Frame*) e os *bits* de CRC. Já o campo de Dados pode ou não existir dependendo do tipo da mensagem a ser transmitida. O tamanho deste campo é determinado por outro campo de controle, o DLC (do inglês *Data Length Code*), que indica o número de *bytes* que estão sendo trafegados. A quantidade máxima de dados por mensagem é de 8 *bytes* (CIA: CONTROLLER AREA NETWORK, 2016).

Para implementar um barramento CAN são necessários dois componentes: um controlador CAN e um *transceiver*. O controlador implementa o protocolo propriamente dito,

enquanto o *transceiver* cuida das questões relacionadas à camada física. Nem sempre o controlador e o *transceiver* estão presentes na arquitetura do processador, de forma que o *transceiver* ou ambos estejam presentes como circuitos integrados externos ao processador. No caso do projeto OpenOBC utilizou-se um *transceiver* externo compatível. A Figura 20 traz a parte do esquemático que ilustra o *transceiver* e suas ligações.

Figura 20 – Circuito de comunicação da interface CAN



Para verificar o correto funcionamento da interface CAN2 foram criados dois projetos: cantx e canrx. O primeiro projeto corresponde ao transmissor enquanto o segundo corresponde ao receptor. Existem duas diferenças básicas entre os dois projetos: a configuração da Message 1 quanto à sua direção e o código executado dentro da função principal main(). No primeiro caso, o campo TX da Message 1 é marcado enquanto no segundo o campo RX está ativo.

Após a compilação dos dois projetos, executou-se o receptor para que este aguardasse dados pelo barramento. Depois disso, executou-se o *firmware* do transmissor. Quando um *breakpoint* atingiu o *loop* infinito do receptor, verificou-se o *buffer* de recepção e constatou-se que os dados enviados foram recebidos com sucesso. Para maiores detalhes sobre o código executado em ambos os projetos, está disponível em (MOTA D.; AMERICO, 2016a) toda documentação dessa implementação em formato Doxygen.

4.8 Porting do FreeRTOS

O FreeRTOS é um Sistema Operacional de Tempo Real *open-source* e livre, desenvolvido pela Real Time Engineers. Ele fornece uma solução compatível para muitas arquiteturas diferentes e geralmente é utilizado para sistemas embarcados, os quais os recursos disponíveis

são limitados. O sistema fornece um conjunto de APIs para ser utilizado no desenvolvimento da aplicação, algumas delas são: *Tasks*, filas, semáforos e mutex (FREERTOS, 2016a).

A arquitetura do sistema influencia na configuração do FreeRTOS, dependendo do processador utilizado, alguns recursos do FreeRTOS podem não estar disponíveis. A configuração do sistema é realizado através do arquivo FreeRTOSConfig.h que adapta o kernel RTOS para a aplicação que está sendo construída (FREERTOS, 2016a).

A configuração do FreeRTOS para o TMS570LS0432 é auxiliada pela ferramenta Halcogen da Texas Instruments, ela gera os arquivos fontes de acordo com a configuração na ferramenta e disponibiliza as principais funcionalidades, entre elas:

- Modo preemptivo e colaborativo;
- Prioridade das *Tasks*;
- Filas;
- Semáforos binário e contador;
- Análise de tolerância a falhas do OBC;
- Funções *Tick hook* e *Idle hook*;
- *Trace hook* macros;
- Checagem de estouro de pilha.

4.8.1 *Modo de Operação*

O *clock* utilizado no FreeRTOS pode ser configurado pela opção `configCPU_CLOCK_HZ`, em relação ao TMS570LS0432, este valor corresponde ao RTICK, caso a fonte do *clock* para o RTICK seja o VCLK, definido com padrão, o valor do *clock* para o RTICK deve ser menor ou igual a RTICK/3.

O modo de operação das *tasks* é um dos fatores fundamentais do funcionamento do FreeRTOS e pode ser configurado pela opção `configUSE_PREEMPTION`, isto define o comportamento das *tasks* e o responsável pela sua interrupção, seja a própria *task* ou o *Kernel*. Para um maior nível de abstração em relação a interrupção das *tasks*, estas são configuradas em modo preemptivo, e caso uma *task* tenha prioridade mais alta que a *task* em execução, essa é interrompida. O número de prioridade das *tasks* é definido pela opção `configMAX_PRIORITIES` e a configuração das *tasks* é de uma margem de 0 (menor prioridade) até `configMAX_PRIORITIES - 1` (maior prioridade), quanto maior for a prioridade de uma *task* maior será o consumo de energia e de memória RAM. Assim, definiu-se para a aplicação um valor máximo de 5.

O tempo de execução de cada *task* é influenciado pelo modo de operação e determinado pelo inverso do valor em Hertz da opção `configTICK_RATE_HZ` que apresenta um valor padrão de 1 KHz, utilizado para realizar testes nas funcionalidades do FreeRTOS. Contudo, este valor é relativamente alto para a maioria das aplicações, pois a troca de contexto ocorre com uma maior frequência, aumentando o processamento da CPU e a escrita de dados na memória RAM. O valor ideal dessa opção é configurado de acordo com ensaios realizados com base nos fatores apresentados anteriormente, mas na maioria das aplicações utilizam uma margem de 100 Hz até 200 Hz.

4.8.2 Gerenciamento de Recursos

O FreeRTOS fornece algumas funcionalidades para gerenciamento de interrupções e recursos, entre elas: filas, semáforo binário e semáforo contador. O semáforo binário pode ser habilitado utilizando a opção `configUSE_MUTEXES`, o qual é útil caso a frequência de interrupções seja baixa. No entanto, quando a frequência é alta, existe a possibilidade de perder a ocorrência de algumas interrupções. Nestes casos, o uso de semáforos contador e filas são mais recomendados, o semáforo contador pode ser habilitado pela opção `configUSE_COUNTING_SEMAPHORES`, ele é utilizado para comunicar eventos entre tarefas, ou entre uma tarefa e uma interrupção, enquanto as filas, que armazenam um conjunto finito de itens, são usadas para comunicar eventos e transferir dados. Caso a fila esteja cheia, a tarefa passa para o estado de bloqueado e fica na espera de um espaço na fila.

4.8.3 Funções de Callback

O FreeRTOS disponibiliza algumas funções de *callback* que são executadas em determinados estados do sistema e podem possuir rotinas para tratar diversas situações. O *tick interrupt hook* é uma função executada durante o intervalo da troca de contexto entre as *tasks*. Assim, as rotinas nele implementadas devem ser de curta duração. O FreeRTOS pode ser utilizado ainda para resetar o *watchdog timer*, o qual pode ter as condições de funcionamento compartilhadas através de uma fila de mensagens e eventos. Quando a maioria das *tasks* estão em estado "bloqueado" é possível utilizar a função `vApplicationIdleHook`, habilitando a opção `configUSE_IDLE_HOOK` para executar algumas rotinas no sistema, tais como: verificar o estado do sistema e deixar o processador em modo de baixo consumo. Este tipo de função é definido como a mais baixa prioridade 0 e pode ser implementada de 2 modos diferentes:

- Utiliza diretamente a função `vApplicationIdleHook` para a execução das rotinas: neste tipo é recomendado a não utilização de APIs que causem o bloqueio da *task*, tais como `vTaskDelay`, filas e semáforos;
- Utiliza a *task* com prioridade para a implementação das funcionalidades: há uma maior flexibilidade, mas consome uma maior quantidade de memória RAM.

O FreeRTOS utiliza o espaço da memória RAM para armazenar dados de tarefas, filas e semáforos. Tal espaço é conhecido como *Heap* e seu tamanho pode ser configurado pela opção `configTOTAL_HEAP_SIZE`. O tamanho da *Heap* pode ser subdimensionado, causando problemas de alocação de memória na habilitação de opção `configUSE_MALLOC_FAILED_HOOK` é possível utilizar a função `vApplicationMallocFailedHook()` a qual realiza o monitoramento para este tipo de situação.

O armazenamento de variáveis, parâmetros e registradores é realizado em uma região de memória denominada de *Stack*, em que cada tarefa é alocada em sua própria *Stack* e seu valor mínimo definido através da opção `configMINIMAL_STACK_SIZE`. A especificação em *word* a ser utilizada nas funções e cada item na *Stack* é dimensionado em 32 *bits* ou 4 *bytes*. A *Stack*, ao ser subdimensionado, pode gerar um estouro de pilha na criação da tarefa, se esta ultrapassar o espaço de memória alocado para ela. O freeRTOS disponibiliza alguns métodos para depurar e checar o estouro de pilha, os quais são descritos abaixo:

- A função `uxTaskGetStackHighWaterMark()` disponibiliza o valor do espaço restante para a ocorrência do estouro de pilha;
- A função `vApplicationStackOverflowHook()` pode ser usada para executar a depuração na aplicação. Ao ser habilitada pela opção `configCHECK_FOR_STACK_OVERFLOW`, seu valor pode ser 1, e neste caso, verifica se o ponteiro da *Stack* esta dentro dos limites da *Stack* para a *task*. Quando o seu valor for 2, neste caso, verifica se este padrão foi sobre-escrito nos últimos 20 *bytes* da *Stack*. O método 2 é mais lento, mas tem uma maior margem de acerto.

4.8.4 Gerenciamento de memória

A alocação de dados na memória RAM poder ser realizada de forma estática e dinâmica, habilitando a opção `configSUPPORT_STATIC_ALLOCATION` e `configSUPPORT_DYNAMIC_ALLOCATION`. A criação de objetos RTOS dinamicamente tem uma maior simplicidade, minimiza o uso de RAM máxima da aplicação e pode escolher a forma de imple-

mentação das rotinas de gerenciamento da memória, habilitando a opção `configAPPLICATION_ALLOCATED_HEAP`. Neste caso, é utilizado a `HEAP_4` que diminuiu os riscos de fragmentação e combina regiões adjacentes de memória. A alocação de forma estática oferece um maior controle, mas aumenta as chances de erros para aplicações mais extensas.

4.8.5 *Coleta e análise de dados*

Algumas vezes é necessário coletar dados sobre o funcionamento da aplicação, os quais podem ser usados para entender o fluxo de execução de uma aplicação, diagnosticar e corrigir problemas. A opção `configUSE_TRACE_FACILITY` oferece alguns macros que possibilitam capturar as trocas de contexto, medir os tempos de execução de cada tarefa, fazer *log* de eventos do *Kernel*, entre outros. Os macros disponíveis são encontrados em (FREERTOS, 2016b) e seus dados podem ser analisados por algumas ferramentas, como: `Stateviewerplugin`.

4.8.6 *Software Timer*

O *Software Timer* é um *timer* utilizado para executar uma determinada função em um tempo futuro. Esta funcionalidade não faz parte do núcleo do *Kernel*, mas não se torna *overhead* para a aplicação. A sua implementação funciona como uma *task* que utiliza recursos do FreeRTOS, não utilizando o contexto de interrupção, e sua comunicação entre as *tasks* é por meio de *queues*. Esta funcionalidade é habilitada através da opção `configUSE_TIMERS` e possui algumas sub-opções:

- `configTIMER_TASK_PRIORITY`: prioridade da *task* de *timer*;
- `configTIMER_TASK_STACK_DEPTH`: tamanho do *stack* da *task* de *timer*;
- `configTIMER_QUEUE_LENGTH`: tamanho do *queue* de comandos da *task* de *timer*.

4.8.7 *Configuração do FreeRTOS*

A configuração adotada para o sistema teve como base as informações apresentadas anteriormente e podem ser observadas na Tabela 6.

4.9 **Placa de Circuito Impresso**

A PCI do projeto OpenOBC foi desenvolvida levando-se em consideração o baixo custo e a alta confiabilidade em que a aplicação se propõe. Há nisso, prós e contras, e a

Tabela 6 – Tabela de Configuração do FreeRTOS para o OpenOBC

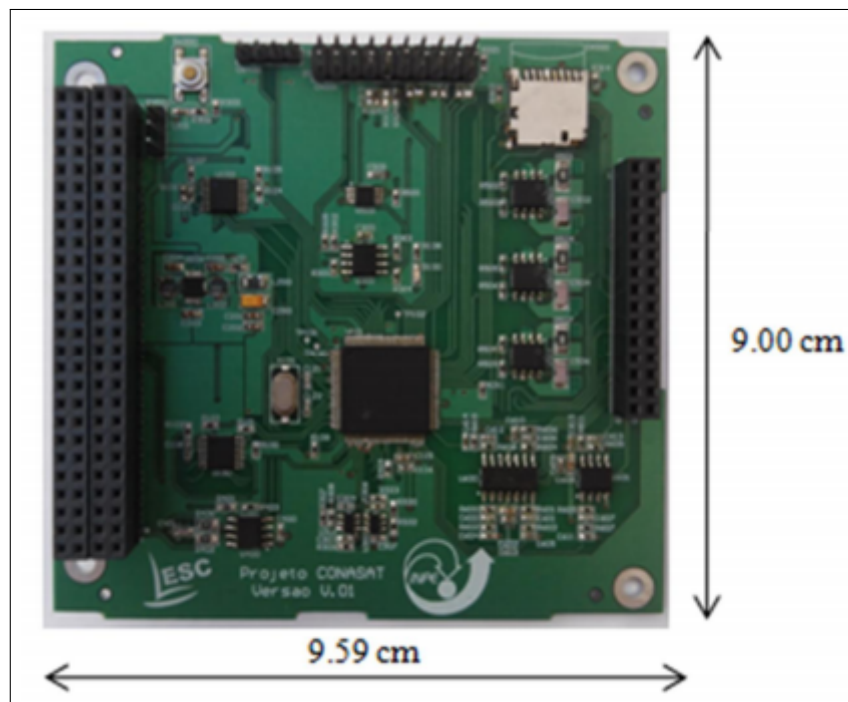
Opção	Configuração
configUSE_PREEMPTION	Habilitado
configUSE_IDLE_HOOK	Habilitado
configUSE_TICK_HOOK	Habilitado
configUSE_MALLOC_FAILED_HOOK	Habilitado
configUSE_COUNTING_SEMAPHORES	Habilitado
configUSE_MUTEXES	Habilitado
configUSE_TIMERS	Habilitado
configCHECK_FOR_STACK_OVERFLOW	Utiliza o segundo modo
configCPU_CLOCK_HZ	5 MHz
configMAX_PRIORITIES	5
configTICK_RATE_HZ	100 Hz
configMINIMAL_STACK_SIZE	128 (valor padrão)
configTOTAL_HEAP_SIZE	25 KB
configTIMER_TASK_PRIORITY	2
configTIMER_QUEUE_LENGTH	4
configTIMER_TASK_STACK_DEPTH	configMINIMAL_STACK_SIZE (padrão)

avaliação dessas características foi feita baseada em técnicas de roteamento. A placa possui duas camadas, barateando seu custo. Porém, a PCI não terá a mesma blindagem de tráfego de sinais que uma placa multicamadas possui. Para solucionar este inconveniente, foram utilizadas técnicas de roteamento e vários capacitores de desacoplamento com o objetivo de eliminar ruídos provenientes da fonte de alimentação (MONTROSE, 2000). Dois planos foram utilizados, um plano de 3,3 V no lado de cima (*top*) da placa e outro de terra no lado de baixo (*bottom*), aumentando, assim, a capacitância da PCI.

Outra técnica utilizada foi a segregação tanto de sinais quando de tensão de alimentação de circuitos digitais e circuitos analógicos. Um exemplo disso, foi a tensão utilizada no pino de alimentação do ADC do processador, a qual a mesma do circuito de leitura de sinal analógico das entradas de ADC. Com isso, o ruído existente nesse circuito não se espalhará para o circuito digital e vice versa (MONTROSE, 2000).

Com relação aos sinais de comunicação, seus roteamentos foram feitos de maneira em que houvesse a menor distância possível entre o processador e o seu circuito de destino e a maior distância possível entre os sinais analógicos e digitais. Destaca-se o roteamento dos sinais de entrada dos *transceivers* CAN, pelo fato de serem sinais diferenciais, os quais foram roteados obedecendo a regra de haver a distância entre suas duas trilhas a mesma que a largura da própria trilha (MONTROSE, 2000). A Figura 21 ilustra a PCI do OpenOBC.

Figura 21 – Placa de Circuito Impresso do OpenOBC com suas dimensões



Após a realização dos testes do que se foi proposto no projeto OpenOBC, será exposto no capítulo 5 a conclusão deste trabalho analisando suas contribuições e a sua continuidade através de trabalhos futuros, como novas características e melhorias sugeridas.

5 CONCLUSÕES E TRABALHOS FUTUROS

O objetivo deste trabalho é a concepção e implementação de uma arquitetura para um computador de bordo que atendesse às restrições do padrão CubeSat. O desenvolvimento de uma arquitetura bem definida foi de grande importância para a eficiência deste projeto, pois além de incorporar diversas qualidades existentes em outros OBCs, ainda foi possível adicionar outras, tais como as características de tolerância a falhas no processador. Ademais, a definição bem elaborada da arquitetura, no qual se realizou ainda uma pesquisa minuciosa dos componentes eletrônicos críticos, foi de grande importância na eliminação de erros na implementação dos esquemas elétricos e, conseqüentemente, na redução de custos com novas versões de PCI para a correção de erros.

Conclui-se que, de acordo com a proposta de origem, este projeto foi um sucesso, uma vez que os resultados dos testes realizados foram de acordo com a literatura. Além disso, o domínio da tecnologia proposta, através da agregação de conhecimento na elaboração deste documento, servirá de insumo para novos projetistas que também queiram desenvolver projetos na área de sistemas embarcados para aplicações espaciais. A proposta de fazer deste trabalho um projeto *Open Source* é uma contribuição bastante significativa para a comunidade acadêmica, pelo fato de diminuir o tempo de pesquisa e desenvolvimento para um próximo estudante ou profissional, podendo o mesmo agregar novas implementações ao projeto e expandir ainda mais conhecimentos para toda comunidade.

Outro ponto importante foi o baixo custo inerente a este OBC, fazendo com que a reprodução do mesmo seja feita sem grandes investimentos e, conseqüentemente, favorecendo a continuidade de pesquisas pertinentes a este objeto de estudo.

No entanto, é importante salientar que fases importantes ainda são necessárias para tornar este OBC apto para lançamento. Dentre estas propomos testes de vibração, temperatura, autonomia e de irradiação. Propõe-se ainda, como trabalhos futuros, o desenvolvimento de uma aplicação voltada para a missão do projeto CONASAT, com o objetivo de validar o funcionamento integrado que um OBC deve ser capaz de realizar.

Finalmente, um outro aspecto importante a ser considerado neste trabalho é o uso de um componente COTS como elemento principal de processamento. Neste sentido, propõe-se um conjunto de testes de inserção de falhas de radiação do tipo SEE e do TID, com o objetivo de avaliar os recursos de tolerância a falhas do processador Cortex R4 da Texas Instruments.

REFERÊNCIAS

- BRIDGES, C. **CubeSat course presentation on On-Board Computing Challenges on in LEO**. 2012. Disponível em: <<http://www.youtube.com/watch?v=6qlue8-O5i4>>. Acesso em: 23 nov. 2014.
- CIA: CONTROLLER AREA NETWORK. **CAN in Automation**. 2016. Disponível em: <<https://www.can-cia.org/can-knowledge/>>. Acesso em: 28 nov. 2016.
- CUBESAT. **The CubeSat Program**. 2016. Disponível em: <<http://www.cubesat.org/about/>>. Acesso em: 28 nov. 2016.
- CYPRESS SEMICONDUCTOR. **Datasheet do S25FL132K0XMF1041**. 2016. Disponível em: <<http://www.cypress.com/file/196886/download>>. Acesso em: 28 nov. 2016.
- EICKHOFF, J. **Onboard Computers, Onboard Software and Satellite Operations: An introduction**. [S. l.]: Springer Aerospace Technology, 2012.
- FLORIPASAT. **Computador de bordo do FloripaSat**. 2016. Disponível em: <<https://tuliogomesp.gitbooks.io/floripasat-technical-manual/content/obdh/hardware.html>>. Acesso em: 28 nov. 2016.
- FREERTOS. **Free Real Time Operating System**. 2016. Disponível em: <<http://www.freertos.org/>>. Acesso em: 28 nov. 2016.
- FREERTOS. **Free Real Time Operating System**. 2016. Disponível em: <<http://www.freertos.org/rtos-trace-macros.html>>. Acesso em: 28 nov. 2016.
- GILMORE, D. **Spacecraft Thermal Control Handbook: Volume i: Fundamental technologies**. [S. l.]: American Institute of Aeronautics and Astronautics, Inc, 2002.
- GOMSPACE. **Computador de bordo da GomSpace**. 2016. Disponível em: <<http://gomspace.com/index.php?p=products-a712c>>. Acesso em: 28 nov. 2016.
- IDT. An-806: Power supply noise rejection. Revisão A, 2013.
- INPE. **Constelação de Nano-Satélites Ambientais**. 2016. Disponível em: <<http://www.crn2.inpe.br/conasat1/index.php>>. Acesso em: 28 nov. 2016.
- ISIS. **Computador de bordo da ESL**. 2016. Disponível em: <https://www.isispace.nl/brochures/CubeComputerV3_Brochure.pdf>. Acesso em: 28 nov. 2016.
- ISIS. **Innovative Solutions In Space**. 2016. Disponível em: <<https://www.isispace.nl/cubesats/>>. Acesso em: 28 nov. 2016.
- LIMA, T. **Como publicar um projeto Open Hardware**. 2016. Disponível em: <<https://www.embarcados.com.br/hardware-livre/>>. Acesso em: 28 nov. 2016.
- MONTROSE, M. I. **Printed Circuit Board Design Techniques for EMC Compliance: 2ª edição**. [S. l.]: Wiley Inter-Science, 2000.
- MOTA D.; AMERICO, J. **Repositório do projeto OpenOBC**. 2016. Disponível em: <https://bitbucket.org/conasat_obc_team/conasat_obc>. Acesso em: 23 dez. 2016.

MOTA D.; AMERICO, J. M. D. S. J. M. J. M. C. **OpenOBC: An Open-Source, Low-Cost and Highly-Reliable Architecture for a CubeSat On-Board Computer**. 2nd IAA Latin American Cubesat Workshop, 2016.

NASA. **Nano-Satélite no padrão CubeSat**. 2013. Disponível em: <<https://www.flickr.com/photos/145538433@N02/27930799044/>>. Acesso em: 28 nov. 2016.

NOERGAARD, T. **Embedded Systems Architecture: A comprehensive guide for engineers and programmers**. [S. l.]: Elsevier, 2005.

RAZZAGHI, E. **Design and Qualification of On-Board Computer for Aalto-1 CubeSat**. Dissertação (Mestrado) — Aalto University, Finlândia, 2012.

ROHM SEMICONDUCTOR. **Datasheet do BD6211F-E2**. 2014. Disponível em: <<http://rohms.rohm.com/en/products/databook/datasheet/ic/motor/dc/bd621x-e.pdf>>. Acesso em: 28 nov. 2016.

SACCO, F. **Comunicação SPI – Parte 1**. 2014. Disponível em: <<https://www.embarcados.com.br/spi-parte-1/>>. Acesso em: 28 nov. 2016.

TEXAS INSTRUMENTS. **Datasheet do SN65HVD230DR**. 2015. Disponível em: <<http://www.ti.com/lit/ds/symlink/sn65hvd230.pdf>>. Acesso em: 28 nov. 2016.

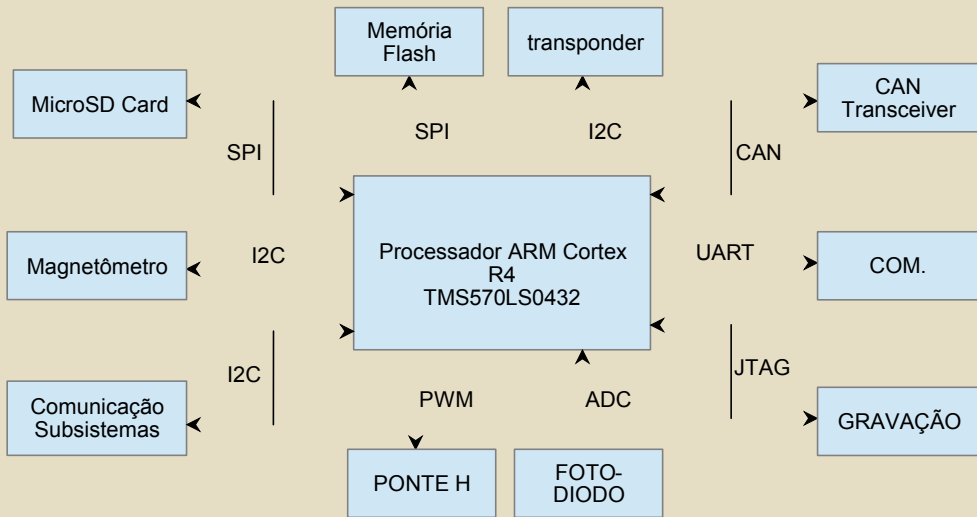
TEXAS INSTRUMENTS. **Datasheet do TMS570LS0432**. 2015. Disponível em: <<http://www.ti.com/lit/ds/symlink/tms570ls0432.pdf>>. Acesso em: 28 nov. 2016.


TEXAS INSTRUMENTS. **Datasheet do LM70**. 2016. Disponível em: <<http://www.ti.com/lit/ds/symlink/lm70.pdf>>. Acesso em: 28 nov. 2016.

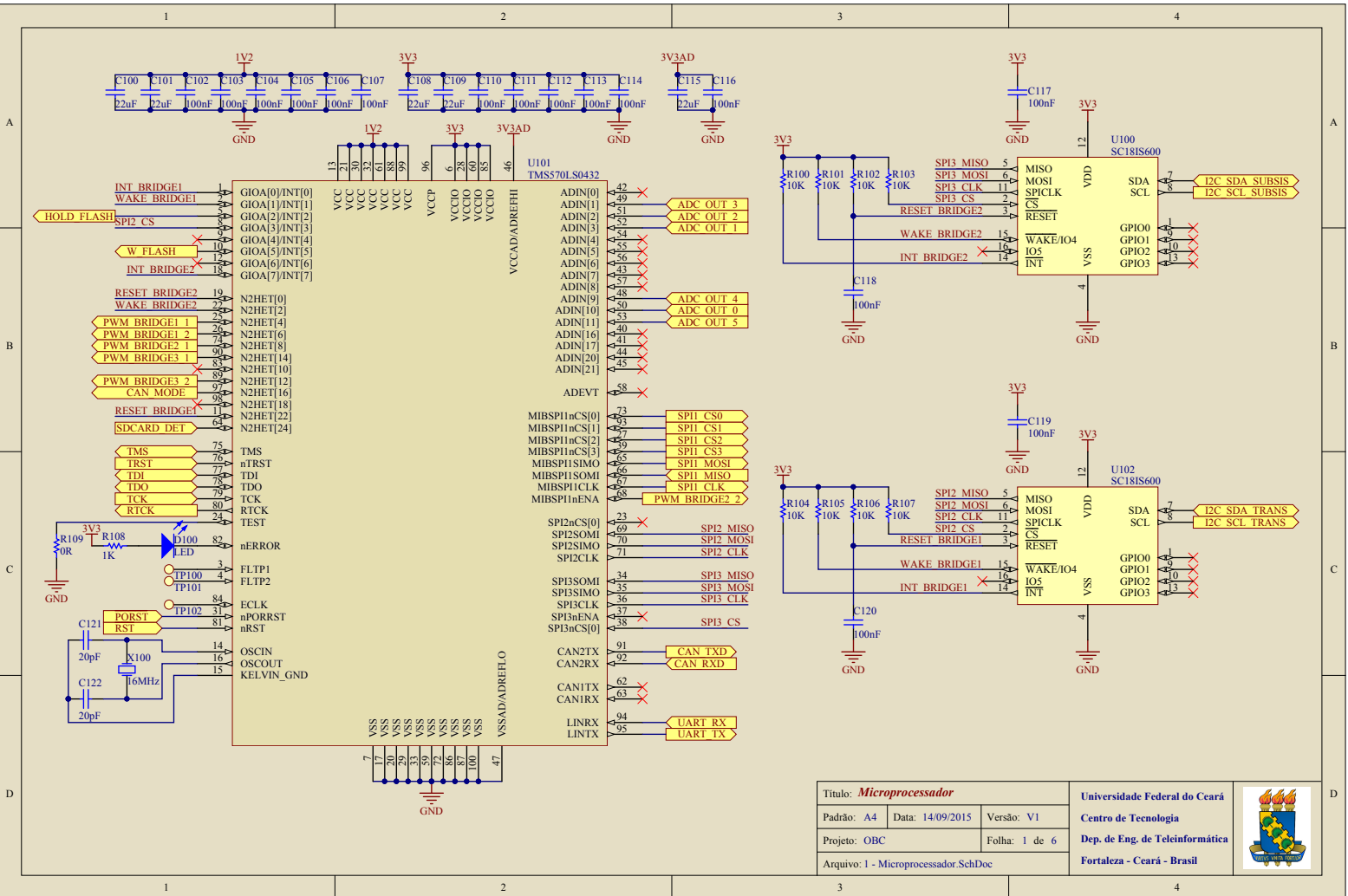
TRIBBLE, A. **The Space Environment: Implications for Spacecraft Design**. [S. l.]: Princeton University Press, 1995.

WELLS G. J.; STRAS, L. J. T. **Canada's Smallest Satellite; The Canadian Advanced Nanospace eXperiment (Can X-1)**. University of Toronto Institute for Aerospace Studies, 2003.

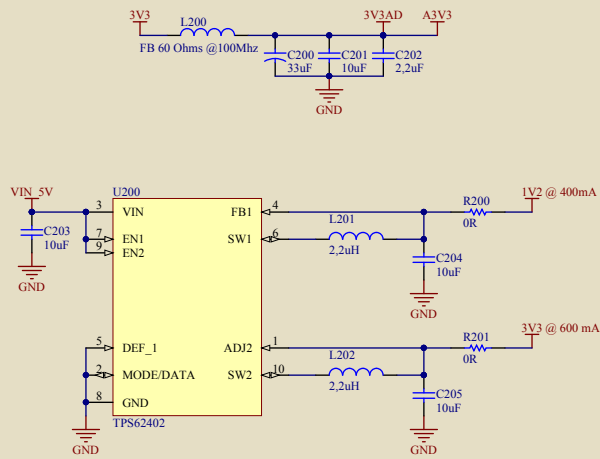
APÊNDICE A – ESQUEMAS ELÉTRICOS



Título: Arquitetura			Universidade Federal do Ceará Centro de Tecnologia Dep. de Eng. de Teleinformática Fortaleza - Ceará - Brasil	
Padrão: A4	Data: 14/09/2015	Versão: V1		
Projeto: OBC		Folha: 0 de 6		
Arquivo: 0 - Arquitetura.SchDoc				

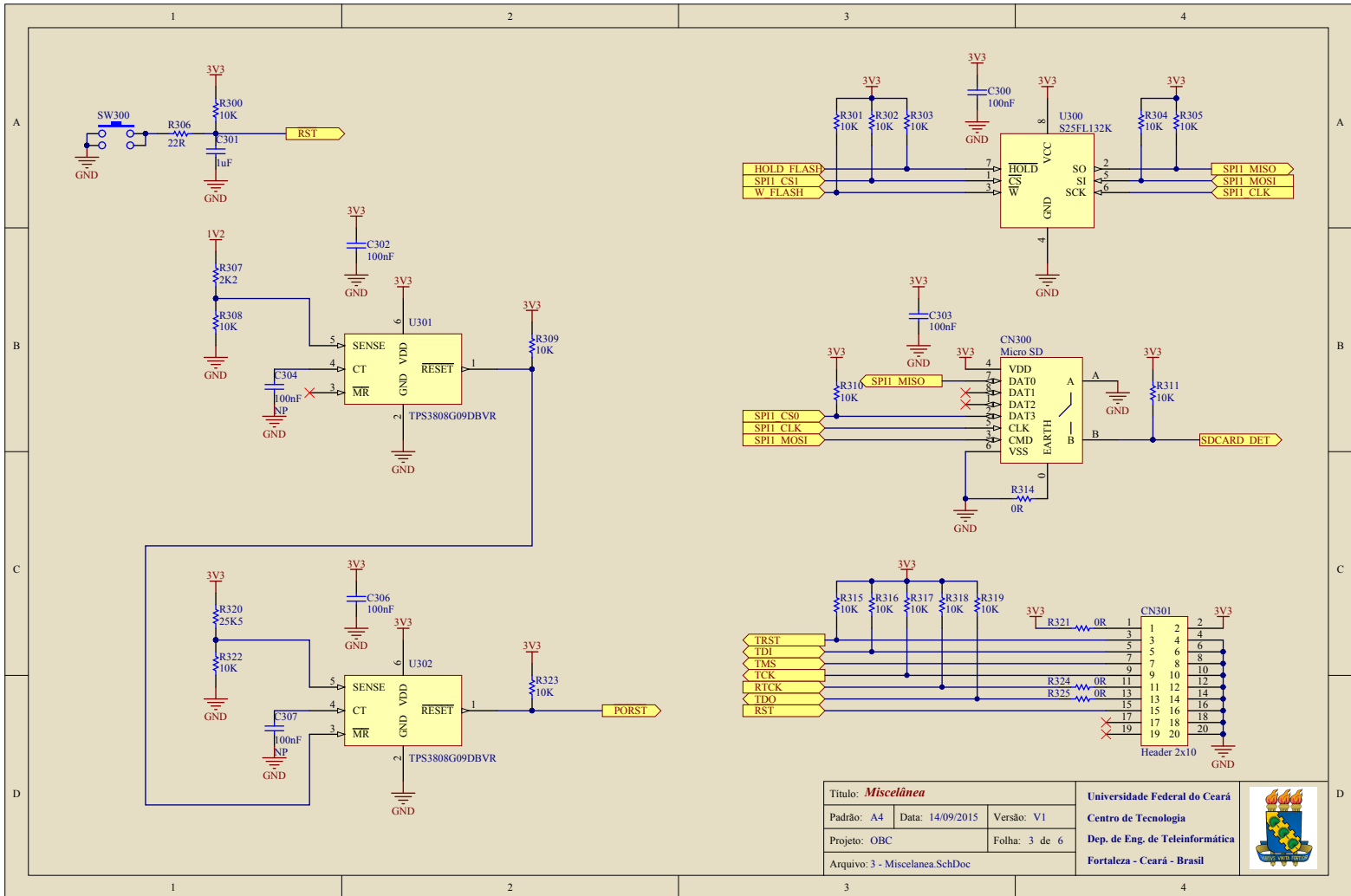


Título: Microprocessador			Universidade Federal do Ceará Centro de Tecnologia Dep. de Eng. de Telemática Fortaleza - Ceará - Brasil	
Padrão: A4	Data: 14/09/2015	Versão: V1		
Projeto: OBC		Folha: 1 de 6		
Arquivo: 1 - Microprocessador.SchDoc				



Título: Fonte			Universidade Federal do Ceará Centro de Tecnologia Dep. de Eng. de Teleinformática Fortaleza - Ceará - Brasil
Padrão: A4	Data: 14/09/2015	Versão: V1	
Projeto: OBC	Folha: 2 de 6		
Arquivo: 2 - Fonte.SchDoc			

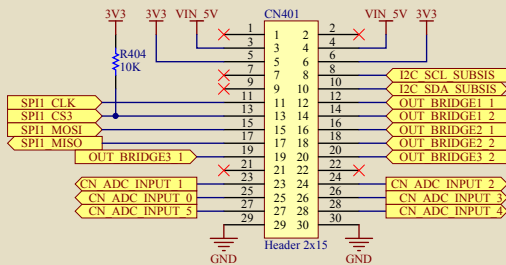
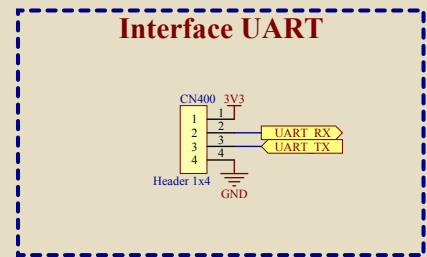
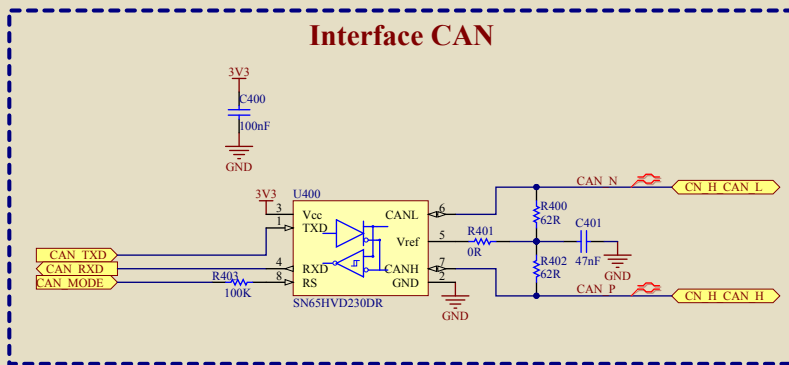





Título: Miscelânea		
Padrão: A4	Data: 14/09/2015	Versão: V1
Projeto: OBC	Folha: 3 de 6	
Arquivo: 3 - Miscelanea.SchDoc		

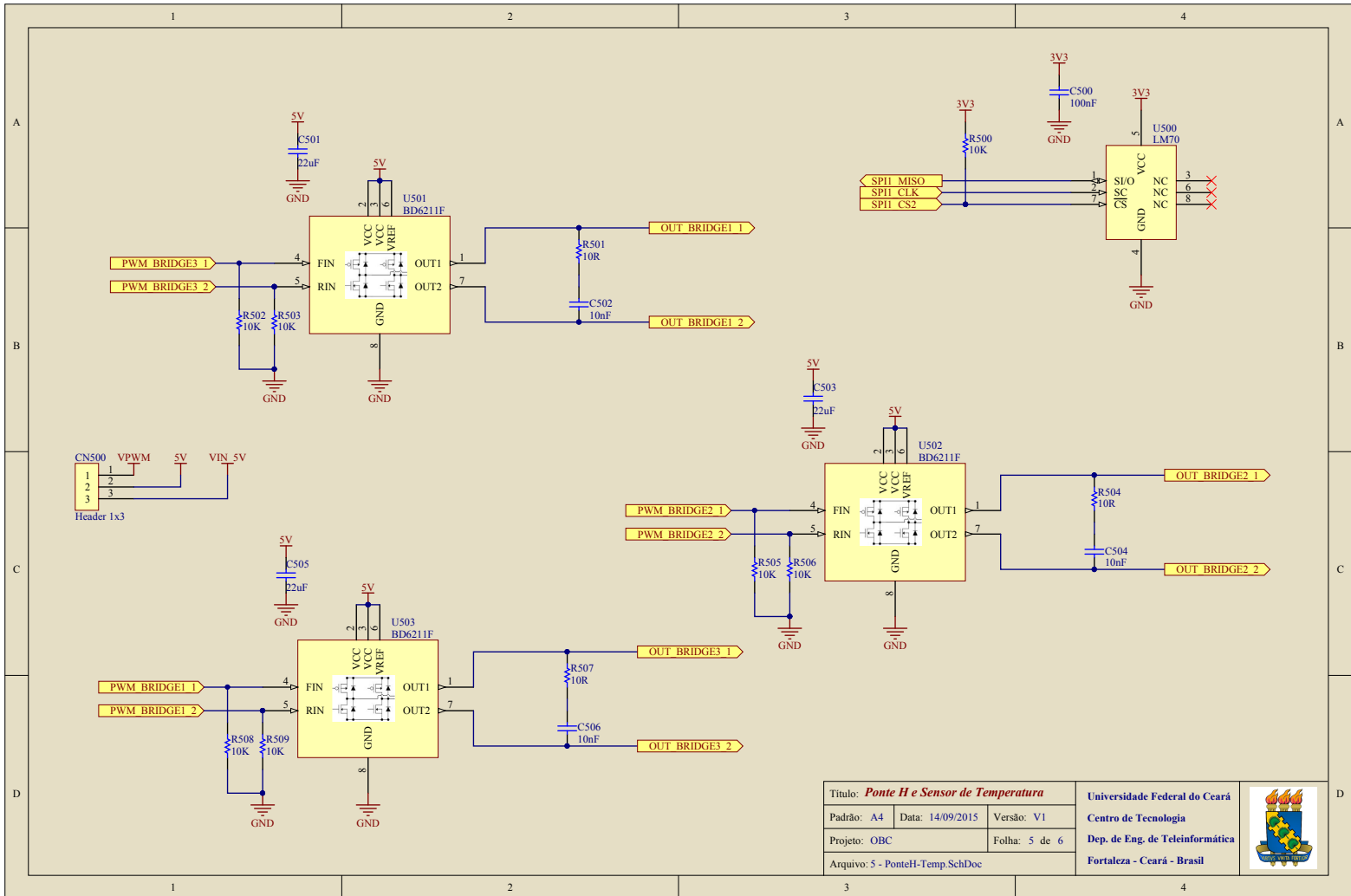
Universidade Federal do Ceará
 Centro de Tecnologia
 Dep. de Eng. de Teleinformática
 Fortaleza - Ceará - Brasil





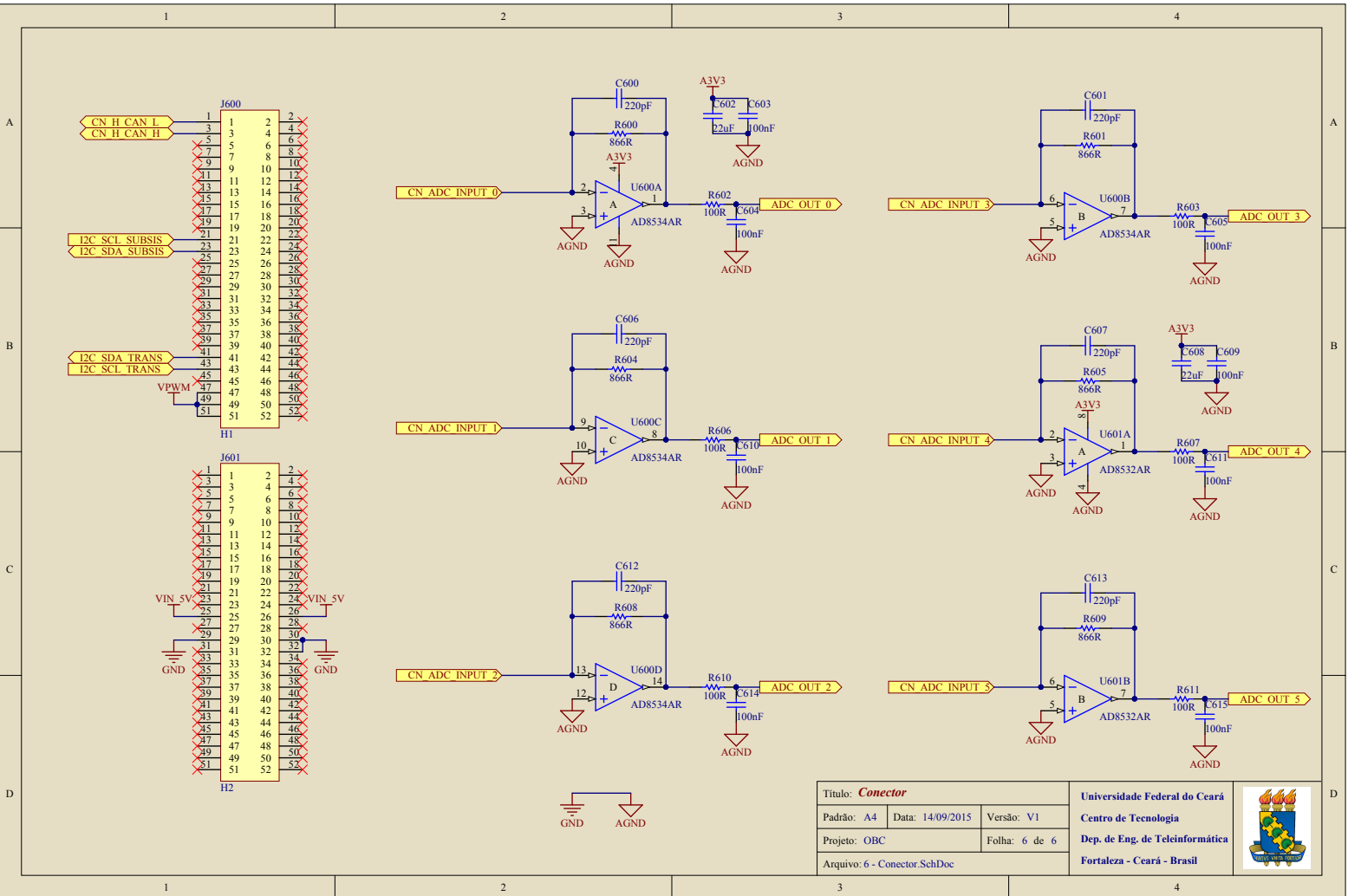
Título: Interfaces de Comunicação			
Padrão: A4	Data: 14/09/2015	Versão: V1	
Projeto: OBC	Folha: 4 de 6		
Arquivo: 4 - Comunicação.SchDoc			
Universidade Federal do Ceará			Centro de Tecnologia Dep. de Eng. de Teleinformática Fortaleza - Ceará - Brasil
Centro de Tecnologia			
Dep. de Eng. de Teleinformática			





Título: Ponte H e Sensor de Temperatura			Universidade Federal do Ceará Centro de Tecnologia Dep. de Eng. de Teleinformática Fortaleza - Ceará - Brasil
Padrão: A4	Data: 14/09/2015	Versão: V1	
Projeto: OBC	Folha: 5 de 6		
Arquivo: 5 - PonteH-Temp.SchDoc			





Título: Conector		
Padrão: A4	Data: 14/09/2015	Versão: V1
Projeto: OBC	Folha: 6 de 6	
Arquivo: 6 - Conector.SchDoc		

Universidade Federal do Ceará
 Centro de Tecnologia
 Dep. de Eng. de Teleinformática
 Fortaleza - Ceará - Brasil



APÊNDICE B – LISTA DE MATERIAIS

Lista de Materiais

Universidade Federal do Ceará
 Centro de Tecnologia
 Dep. de Engenharia de Teleinformática

Projeto: OBC
 Documento gerado em: 22/12/2016

1 Placa (S):	85,07
1000 Placas (S)	26,09
Itens por Placa	155

#	Designator	Comment	Part#	Description	Manufacturer	eVendor	Quantity	MSRP	Total MSRP	eVendor Part#	eVendor Price	Total Vendor	MOQ
1	C100, C101, C108, C109, C115, C501	2.2uF	CL10A222M2R9NC	CAP CER 22uF 2.2V 20% XSR 0805	Samsung Electro-Mechanics America, Inc	Digley	10,00081	0,61	1228-1165-ND	0,25	2,50	1000	
2	C102, C103, C104, C105, C106, C107	1000F	GRM156R71C10K48SD	CAP CER 0.1UF 16V 10% X7R 0402	Murata Electronics North America	Digley	30,000375	0,11	490-3261-ND	0,1	3,00	1000	
3	C121, C122	20pF	GRM1555C1H20QJAG1D	CAP CER 20PF 50V 5% NP0 0402	Murata Electronics North America	Digley	2,000510	0,01	490-6930-ND	0,1	0,20	1000	
4	C200	33uF	TPSA330K060R0600	CAP TANT 33UF 6.3V 10% 1206	AVX Corporation	Digley	1,0,18180	0,18	478-1755-ND	0,57000	0,57	1000	
5	C201, C203, C204, C205	10uF	C1068X5R1A106K080AC	CAP CER 10UF 16V 10% X5R 0603	TDK Corporation	Digley	4,0,138	0,55	445-7486-ND	0,48	1,92	1000	
6	C302	2.2uF	CC06032R2V068B225	CAP CER 2.2UF 10V 10% 0603	Yageo	Digley	1,0,21137	0,03	311-1450-ND	1,12000	0,12	1000	
7	C301, C305	1uF	GRM156R6J105KE18D	CAP CER 1UF 6.3V 10% X5R 0402	Murata Electronics North America	Digley	2,000870	0,02	490-1926-ND	0,1	0,20	1000	
8	C304, C307	100nF	GRM156R71C104K48SD	CAP CER 0.1UF 16V 10% X7R 0402	Murata Electronics North America	Digley	2,000375	0,01	490-3261-ND	0,1	0,20	1000	
9	C401	47nF	C1068C0J1H472J080AA	CAP CER 4700PF 50V 5% C0G 0603	TDK Corporation	Digley	1,0,05875	0,09	445-7400-ND	0,24	0,24	1000	
10	C502, C504, C508	100pF	C3216C01H103J030AA	CAP CER 10000PF 50V 5% C0G 1206	TDK Corporation	Digley	3,0,1059	0,32	445-7358-ND	0,37	1,11	1000	
11	C600, C601, C606, C607, C612, C613	220pF	GRM1555C1H221JAG1D	CAP CER 220PF 50V 5% NP0 0402	Murata Electronics North America	Digley	6,000600	0,04	490-1923-ND	10,000	0,60	1000	
12	CH300	Micro SD	DM30-3F	CONN MICRO SD R/A PUSH-PULL SMD	Hirose Electric Co Ltd	Digley	1,0,7847	0,78	HR1941CT-ND	1,92	1,92	1000	
13	CH301	Header 2x10	M20-8971046	2X10 POS DR VERM PIN HDR	Hawain Inc	Digley	1,0,4964	0,49	962-2327-ND	0,88	0,88	1000	
14	CH400	Header 1x4	M20-9904448	SIL VERTICAL PC TAIL PIN HEADER	Hawain Inc	Digley	1,0,57770	0,09	952-2266-ND	0,19	0,19	1000	
15	CH401	Header 2x15	PH7C152LFBN-RC	CONN HEADER FMAL 30P5 1" DL TN	Sullins Connector Solutions	Digley	1,0,99617	1,00	57093-ND	2,17	2,17	1000	
16	CH500	Header 1x3	M20-9903446	SIL VERTICAL PC TAIL PIN HEADER	Hawain Inc	Digley	1,0,5588	0,08	962-2264-ND	0,14	0,14	1000	
17	D100	LED	LTST-C191BKT	LED 468NM BLUE CLEAR 0603 SMD	Lite-On Inc	Digley	1,0,0855	0,09	160-1647-ND	0,48	0,48	1000	
18	R000	HT	ESQ_12L_49_GD			Digley	1	0,00			0,00		
19	R001	HZ	ESQ_12L_49_GD			Digley	1	0,00			0,00		
20	L200	FB 60 Ohms @100MHz	F8M3216M600-T	FERRITE BEAD 60 OHM 1206	Taiyo Yuden	Digley	1,0,03300	0,03	587-1770-ND	0,12000	0,12	1000	
21	L201, L202	2.2uH	VLFP301AT-2R2M1R0	FIXED IND 2.2UH 1A 120 MOHM SMD	TDK Corporation	Digley	2,0,44	0,88	445-3215-ND	0,9	1,80	1000	
22	R100, R101, R102, R103, R104, R105	10K	ERJ2RKF1002X	RES SMD 10K OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	34,0,05544	0,19	P1018LCT-ND	0,1	3,40	1000	
23	R108	1K	ERJ2RKF1011X	RES SMD 1K OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	10,0,05544	0,01	P1108LCT-ND	0,1	0,10	1000	
24	R109, R314, R321, R324, R325	0R	ERJ2RKE0R00X	RES SMD 0.0 OHM JUMPER 1/10W	Panasonic Electronic Components	Digley	5,0,00384	0,02	P010CT-ND	0,1	0,50	1000	
25	R200, R201, R401	0R	ERJ2RKE0R00V	RES SMD 0.0 OHM JUMPER 1/10W	Panasonic Electronic Components	Digley	3,0,00352	0,01	P010CT-ND	0,1	0,30	1000	
26	R306, R313	22R	ERJ2RKF22R0X	RES SMD 22 OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	2,0,05544	0,01	P221LCT-ND	0,1	0,20	1000	
27	R307	0R2	ERJ2RKF0R21X	RES SMD 0.2K OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	1,0,05544	0,01	P201LCT-ND	0,1	0,10	1000	
28	R320	25K5	ERJ2RKF25K2X	RES SMD 25.5K OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	1,0,05544	0,01	P255LCT-ND	0,1	0,10	1000	
29	R400, R402	62R	ERJ2RKE62R0V	RES SMD 62 OHM 5% 1/2W 0805	Panasonic Electronic Components	Digley	2,0,02432	0,05	PE240CT-ND	0,23	0,46	1000	
30	R403	100K	ERJ2RKF1003X	RES SMD 100K OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	1,0,05544	0,01	P100KCT-ND	0,1	0,10	1000	
31	R601, R604, R607	10R	RNCP126R710R19R	RES SMD 10 OHM 1% 1/2W 1206	Resistor Electronics Inc	Digley	3,0,00902	0,03	RNCP126R710R19R-ND	0,1	0,30	1000	
32	R600, R601, R604, R605, R608, R609	896R	ERJ2RKF8960X	RES SMD 896 OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	6,0,05544	0,03	P896LCT-ND	0,1	0,60	1000	
33	R602, R603, R606, R607, R610, R611	100R	ERJ2RKF1000X	RES SMD 100 OHM 1% 1/10W 0402	Panasonic Electronic Components	Digley	6,0,05544	0,03	P100LCT-ND	0,1	0,60	1000	
34	SW300, SW301	SW	PT5525SM10SMTR LFS	Switch tact. SMT. SPST. 160gf. 5.25 mm L x 5.25 mm W x 1.5 mm H	C&K Components	Mouser	2,0,244	0,49	611-PT5525SM10LFS	0,48	0,96	1000	
35	U100, U102	IC1655000	SC1655000PW112	IC BRIDGE SPI2C 16-ESSOP	NXP Semiconductors	Digley	2,1,42	2,80	528-332-ND	0,90	5,60	1000	
36	U101	ARM Micocontroller	TM5700432APZ001	ARM Micocontrollers - MCU 16/32-Bit RISC Flash MCU	Texas Instruments	Digley	1,6,91	6,91	695-TM5700432APZ001	11,42	11,42	1000	
37	U200	TPS62402	TPS62402DRCT	IC REG BUCK SYNC PROG DL 10SON	Texas Instruments	Digley	1,0,2416	2,04	298-21646-ND	2,81	2,81	1000	
38	U300	S25FL132K	S25FL132KXKMPD14	Flash 32Mb. 3V. 108MHz CMOS SPI Flash	Spansion	Mouser	1,0,44	0,44	787-25FL132KXKMPD14	0,58	0,58	1000	
39	U301, U302	TPS3800	TPS3800DWR	IC VOLT SUPERVISOR 0.5V SOT23-6	Texas Instruments	Digley	2,0,785	1,57	298-17385-ND	1,96	3,96	1000	
40	U400	SN65LVDS32DR	SN65LVDS32DR	IC 3.3V CAN TX/RX W/STDBY 8-SOIC	Texas Instruments	Digley	1,0,9	0,90	298-11654-ND	0,30	2,30	1000	
41	U500	LM70	LM70DMM-SNOPB	IC TEMP SENSOR DIGITAL BVSSOP	Texas Instruments	Digley	1,0,83875	0,84	LM70DMM-SNOPBCT-ND	0,73	2,73	1000	
42	U501, U502, U503	BD621F-E2	BD621F-E2	IC MOTOR DRIVER PAR 850P	Rohm Semiconductor	Digley	3,0,91039	2,73	BD621F-E2CT-ND	1,96	8,80	1000	
43	U500	AD9634AR	AD9634ARZ-REEL	IC OPAMP GP 3MHZ RRIO 14SOIC	Analog Devices	Digley	1,0,84930	0,80	AD9634ARZ-REELCT-ND	1,82	1,82	1000	
44	U601	AD9634AR	AD9634ARZ-REEL	IC OPAMP GP 3MHZ RRIO 14SOIC	Analog Devices	Digley	1,0,63318	0,63	AD9634ARZ-REELCT-ND	1,47	1,47	1000	
45	X100	16MHz	EC5-160-30-3X-TR	CRYSTAL 16MHZ 20PF SMD	EC5 Inc	Digley	1,0,21	0,21	XC1776CT-ND	0,53	0,53	1000	