UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

RAMON HERRERO MARTINS

# PROPOSIÇÃO DE TESTES PARA FLATSAT DA MISSÃO FLORIPASAT UTILIZANDO DISPOSITIVO PROGRAMÁVEL

Joinville
2022

RAMON HERRERO MARTINS

# PROPOSIÇÃO DE TESTES PARA FLATSAT DA MISSÃO FLORIPASAT UTILIZANDO DISPOSITIVO PROGRAMÁVEL

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Engenharia Mecatrônica, no curso Engenharia Mecatrônica da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Prof. Dr. Anderson Wedderhoff Spengler

Joinville
2022

## RESUMO

Este trabalho apresenta a montagem e testes da FlatSat2 que será utilizada para validar os módulos que compõem o satélite FloripaSat2 e futuros projetos envolvendo Cubesats. Tal plataforma tem o propósito de avaliar e prever eventuais comportamentos da integração dos diferentes módulos que compõem o satélite como sistema de comunicação, sistema de controle de atitude, sistema de fornecimento de energia e demais placas. Este trabalho sucede a etapa de desenvolvimento do hardware da FlatSat2, desenvolvida dentro do laboratório SpaceLab. Com isso e devido aos vários outros projetos que estão em desenvolvimento, esta universidade torna-se capaz não só de desenvolver e montar hardware para satélites mas também de testar e prever seu comportamento durante seu ciclo de vida.



**Palavras-chave**: Sistemas Embarcados. CubeSat. FloripaSat2. FlatSat. Xilinx Zynq. Teste de integração de sistemas. FPGA. VHDL. I2C.

**ABSTRACT**

This paper presents the assembly and testing of FlatSat2 that will be used to validate the modules that compose the FloripaSat2 satellite and future projects involving Cubesats. Such platform has the purpose of evaluating and predicting eventual integration behaviors of the different modules that compose the satellite as communication system, attitude control system, power supply and other boards. This work succeeds the stage of hardware development of FlatSat2, developed within the SpaceLab laboratory. With that and due to the various other projects that are under development, this university becomes capable not only of developing and assembling hardware for satellites but also of testing and predicting their behavior during their life cycle.

**Keywords**: Embedded Systems. CubeSat. FloripaSat2. FlatSat. Xilinx Zynq. System Integration Tests. FPGA. VHDL. I2C.

## AGRADECIMENTOS

Muitas pessoas contribuíram para o desenvolvimento direto e indireto deste trabalho, e, mesmo correndo o risco de esquecer alguns, não posso deixar de expressar aqui minha gratidão.

Primeiramente agradeço à UFSC pela estrutura fornecida para o meu desenvolvimento intelectual e pessoal ao longo de todos estes anos; ao LISHA, seus organizadores e quadro de alunos, pela concessão do espaço e tempo necessário para a elaboração deste trabalho.

Em relação aos agradecimentos pessoais, começo pelo meu orientador, o Professor Doutor Anderson Wedderhoff Spengler, o qual me acolheu sob sua tutela e se dispôs a me orientar e que viu em mim algo que talvez nem eu mesmo soubesse que existia, colocando sob minha responsabilidade o desenvolvimento de algo tão grandioso e ousado quanto este projeto o foi.

À Manueli de Souza Sabatke, minha companheira que abriu mão de muitos momentos e eventos em que poderíamos estar juntos, visando um bem maior que é a concretização deste trabalho. Agradeço por último a todo o corpo técnico-administrativo da UFSC Campus Joinville por proporcionarem sempre que possível a solução dos problemas que todo estudante de engenharia enfrenta ao longo do curso.

À todos o meu muito obrigado.

Só sei que nada sei, e o fato de saber isso, me coloca em vantagem sobre aqueles que acham que sabem alguma coisa.

SÓCRATES

## LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

## LISTA DE SIGLAS

ACP Accelerator Coherency Port

ACS Passive Attitude Control System

ADCS Attitude Determination and Control Subsystem

ARM Advanced RISC Machines

ASIC Application Specific Integrated Circuit

C&DH Command and Data Handling

CAN Controller Area Network

CISC Complete Instruction Set Computer

CLB Configuration Logical Block

DDR2 Double Data Rate versão 2 - Padrão de memórias

DDR3 Double Data Rate versão 3 - Padrão de memórias

DDR3L Double Data Rate with Low Voltage versão 3 - Padrão de memórias

DMA Acesso direto à memória

EDC Environmental Data Collector

EM Engineering Model

EPS Electrical Power Subsystem

EQM Engineering Qualification Model

FM Flight Model

FPGA Field Programable Gate Array

GPIO General Purpouse Input Output

IDE Integrated Development Environment

IO Input/Output que significam suas entradas e saídas

IOB Input/Output Block

LEO Low Earth Orbit

LPDDR2 Low Power Double Data Rate versão 2 - Padrão de memórias

LVDS Low Voltage Differential Signal

MDVE Model-based Development and Verification Environment

NAND Flash Padrão de memória não volátil baseado em Flip-Flops Não-e

NOR Flash Padrão de memória não volátil baseado em Flip-Flops Não-ou

PL Programmable Logic - o controlador do FPGA

RISC Reducec Instruction Set Computer

RISC Reduced Instruction Set Computer

RTD Resistive Temperature Detector

SBC Sigle Board Computer

SMD Surface Mount Device

SMT Surface Mount Technology

SoC System-on-a-Chip

SRAM Memória estática de acesso aleatório

THR Through Hole Reflow

TTC Telemetry, Tracking and Control

TTC2 Telemetry, Tracking and Control

UART Universal Assinchronous Receptor Transmissor

USB Universal Serial Bus

**SUMÁRIO**

# 1 INTRODUÇÃO

O projeto, integração, teste e lançamento de satélites envolvem muitas etapas, pessoas e recursos para que tenha êxito. Com o desenvolvimento de componentes eletrônicos cada vez mais complexos e também mais baratos, foi possível o desenvolvimento de satélites de dimensões reduzidas, fazendo com que o custo de todo o projeto diminuísse e se tornasse acessível também para instituições de ensino e pesquisa.

Surge então, a partir de esforços de pesquisadores, alguns padrões para que diferentes instituições pudessem se desenvolver nesta área, mesmo inicialmente sem especialistas no assunto. Um desses padrões foi divulgado em 1999 pelos professores Bob Twiggs da Universidade Stanford e Jordi Puig-Suari da California Polytechnic State University (CUBESAT.ORG, 2017).

Segundo dados da organização Cubesat (CUBESAT.ORG, 2017), cerca de 6% desses satélites lançados não conseguiram uma única comunicação e tantos outros não duraram o esperado. Muitas vezes estas falhas podem acabar encerrando todo um projeto que poderia gerar muita visibilidade e trazer recursos e geração de conhecimento para a instituição de ensino.

Espera-se que tais falhas possam ser evitadas caso alguns testes, entre a etapa de projeto e de voo, sejam realizadas.

Por ser um conteúdo muito denso, no Capítulo 2 é apresentada uma breve revisão teórica acerca das normas do setor aeroespacial, a Filosofia de Modelos adotada no desenvolvimento dos CubeSats e os principais componentes dele.

No Capítulo 3, mostra-se o método de resolução escolhido, os recursos necessários para alcançá-los e uma breve explicação do funcionamento da MicroZed.

No Capítulo 4, são descritos os resultados dos testes implementados com as devidas considerações sobre os mesmos.

Por fim, no Capítulo 5, são apresentadas as conclusões sobre os resultados obtidos, as dificuldades enfrentadas ao longo do desenvolvimento, os conhecimentos adquiridos durante o desenvolvimento e as considerações sobre os próximos passos antecedendo o lançamento ao espaço.

## 1.1 Objetivo Geral

Validação do hardware da FlatSat2 com testes eletromecânicos e de programação, de continuidade e de qualidade de sinal entre os slots.

## 1.2 Objetivos Específicos

1. Teste das trilhas da FlatSat2:
   A placa foi confeccionada e recebida sem conferência por parte dos desenvolvedores dela. Sendo assim, ficou-se responsável de conferência dos corretos caminhos das trilhas. Além disso, garantiu-se que os sinais fossem transmitidos de um dispositivo a outro da placa, tanto entre os barramentos PC104 quanto entre PC104 e a entrada dos transdutores de tensão que direcionados à MicroZed.
2. Montagem dos componentes na placa:
   Como a placa foi enviada sem os componentes, também foi necessária a instalação dos mesmos com os devidos cuidados e técnicas de montagem, levando-se em consideração as normas do setor aeroespacial que serão citadas mais adiante.
3. Testes com dispositivos programáveis
   Devido a muitas variáveis que podem interferir no resultado devido à complexidade envolvida na implementação de um dispositivo programável como a Microzed, implementou-se como uma forma de teste incremental, dispositivos programáveis mais simples como um Arduino ou uma Raspberry Pi, plataformas com extensos testes e bibliotecas disponíveis.
4. Testes dos transdutores de tensão:
   A comunicação entre a MicroZed e o microHeader, o barramento na parte inferior da placa, são em sua maioria feita através de sinais entre 0 e 1.8 V. Já o barramento PC104 do FloripaSat2 e outros dispositivos da FlatSat2 se utilizam de sinais entre 0 e 3.3 V. Sendo assim, é necessária uma conversão dos níveis de sinais e isto é feito através dos transdutores de tensão bidirecionais comuns ou com função open-drain/push-pull (como exige-se no $I^2C$) montados na FlatSat2. Sendo assim é necessário garantir o correto funcionamento de tais componentes.

# 2 REVISÃO TEÓRICA

Neste capítulo é apresentado um conteúdo teórico para melhor entendimento do que se propõe atingir neste trabalho, já que o mesmo envolve tanto o desenvolvimento de satélites quanto as etapas de teste de hardware e programação para automação de testes e análise comportamental de equipamentos. As rotinas de teste apresentadas são para uma FlatSat, neste caso a FlatSat2, desenvolvida para interconectar e testar os módulos do FloripaSat2 que é um CubeSat.

Algumas das diferenças entre a FlatSat2 e uma FlatSat comum como as existentes no mercado, são alguns sensores de temperatura e módulos de conversão para comunicação como RS485, CAN e USB . Além disso, a FlatSat2 possibilita a automação dos testes através do uso de uma placa MicroZed que é conectada diretamente através do conector microHeader.

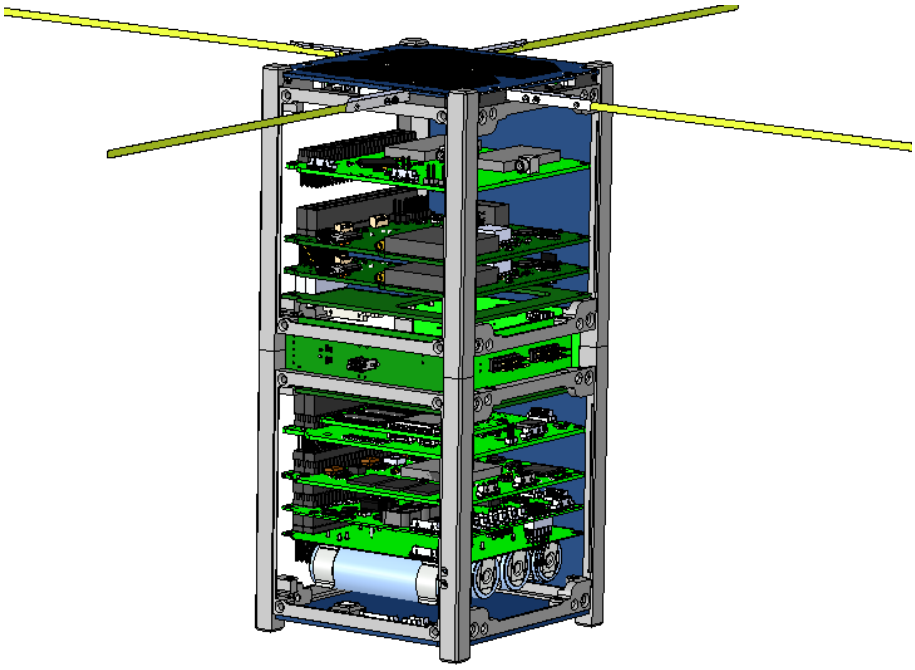
As rotinas de testes embarcadas na placa MicroZed da Avnet são parte implementadas em VHDL e fazem parte da FPGA contida no SoC Zynq 7000 e parte em C, C++ e Python que são linguagens interpretadas pelos processadores ARM, também contido no Zynq 7000. Sendo assim, cada conceito apresentado neste parágrafo será melhor detalhado.

## 2.1 CubeSats

CubeSats são, segundo Chantal Cappelletti et. al(CAPPELLETTI; BATTISTINI; MALPHRUS, 2020), um padrão proposto pelos professores Jordi Puig-Suari da California Polytechnic State University e Bob Twiggs da Universidade Stanford no ano de 1999 para que estudantes de doutorado e instituições de ensino pudessem participar da compreensão e do desenvolvimento de satélites a um custo muito abaixo do praticado por empresas e governos, com valores na época partindo de 30 mil dólares entre construção e lançamento. Algo muito menor comparado aos milhões investidos por governos em um único lançamento.

Um lançamento pode acomodar uma centena de CubeSats em suas plataformas (Pods) de lançamento. O estabelecimento deste padrão também disponibiliza oportunidades de lançamento do projeto em missões com slots disponíveis no sistema de fixação e lançamento dos CubeSats, também chamado de deployer. Na maioria dos lançamentos é utilizado o PPOD desenvolvido pela CalTech(CUBESAT.ORG, 2017).

Figura 1 – FloripaSat2 - Cubesat 2U



Fonte: Github do SpaceLab(UFSC-SPACELAB, 2020).

Cada CubeSat apresenta dimensões de 10 cm de largura, 10 cm de altura e seu comprimento pode variar entre múltiplos de 10 cm com a menor unidade sendo chamada de 1U, podendo chegar até 3U (30 cm de comprimento). O CubeSat visto na Figura 1 é o FloripaSat2 sendo um satélite 2U.

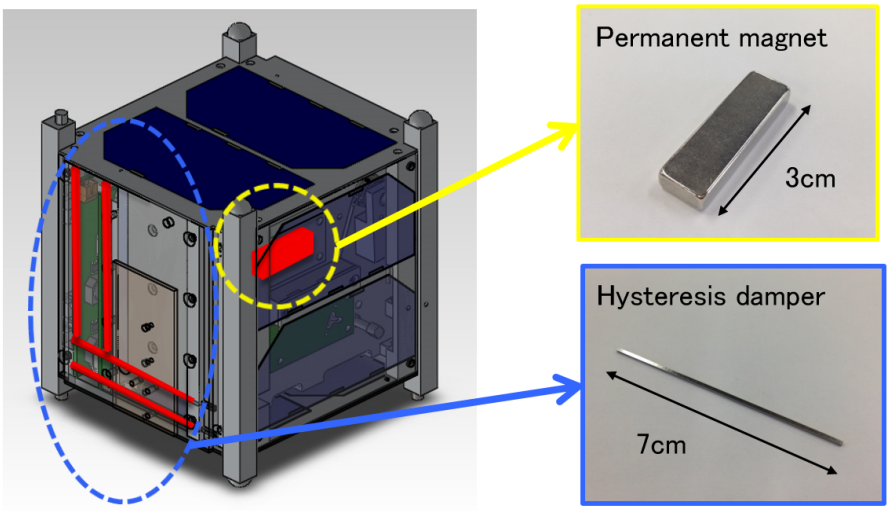
Para que tais componentes se acomodem dentro de um CubeSat 1U, eles são empilhados e conectados a um barramento de 104 conexões dispostas em duas linhas de 52 canais chamado de PC/104, desenvolvido e gerido pelo PC/104 Embedded Consortium e com especificação disponível no site da organização (PC/104 EMBEDDED CONSORTIUM, 2008).

Os módulos necessários para o funcionamento correto de um CubeSat são, seguindo a nomenclatura adotada pela organização CubeSat(CUBESAT.ORG, 2017) serão apresentados nas próximas seções:

### 2.1.1 ADCS

O ADCS, também chamado de Controle de Atitude, do FloripaSat2 a princípio será composto do mesmo módulo enviado com o FloripaSat1, que é um controle passivo de atitude (ACS) que faz com que o satélite se alinhe com o campo magnético da terra. Um sistema similar já foi utilizado no satélite ITF-1 de 2014 (UNIVERSIDADE DE TSUKUBA, 2014) e pode ser visto na Figura 2.

Figura 2 – Exemplo de controle de atitude passivo



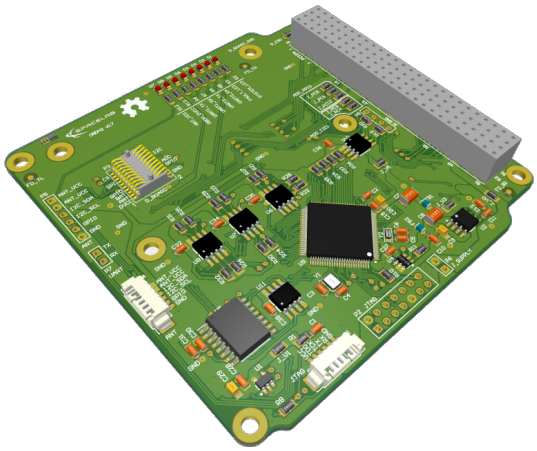

Fonte: Universidade de Tsukuba(UNIVERSIDADE DE TSUKUBA, 2014).

Tal módulo não requer alimentação, ocupa pouco espaço e pesa pouco (cerca de 50 gramas). É capaz de estabilizar o sistema em poucas horas, eliminando qualquer rotação do satélite e mantendo um alinhamento com os campos da Terra dentro de uma tolerância de 15º(UFSC-SPACELAB, 2020).

### 2.1.2 C&DH

O dispositivo de Comando e Manipulação de dados (C&DH) é chamado no projeto do FloripaSat2 de On-board Data Handling (OBDH2) e desenvolvido dentro dos laboratórios da UFSC. Seu projeto pode ser visto na Figura 3. É responsável por gerar os pacotes de dados, armazená-los em memória não volátil ou transmitir à Terra através do módulo de comunicação. Também analisa os comandos enviados da Terra e toma as ações apropriadas ou direciona os dados ao módulo de destino.

Figura 3 – OBDH2 do FloripaSat2



Fonte: SpaceLab(UFSC-SPACELAB, 2020).

Nele está armazenado um Sistema Operacional de Tempo Real (RTOS) que executa todas as funções básicas de um sistema operacional como gerenciamento de memória, pilha e processos e realiza diagnósticos dos módulos que integram o satélite. Também implementa uma série de protocolos de comunicação como SPI e $I^2C$.

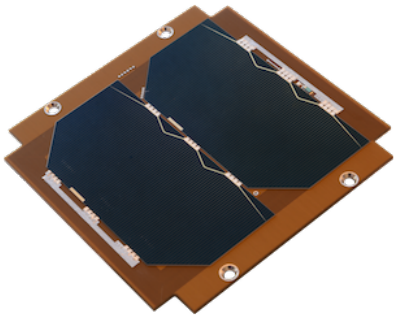
Como o componente centralizador de dados, ele é portanto o componente que tem o maior número de interfaces.

É importante, portanto, a implementação da configuração de hardware com os protocolos de comunicação existentes no OBDH2 para a simulação e teste do mesmo.

### 2.1.3 EPS

O EPS é o módulo que fornece potência para todos os demais sistemas do CubeSat. Para o FloripaSat2, foi desenvolvido um novo módulo de potência, chamado de EPS2, capaz de coletar a energia de 10 placas solares modelo P110 da GomSpace (GOMSPACE. . . , 2022) (Figura 4) dispostas nas superfícies do FloripaSat2, armazenar em 4 baterias de ion-lítio e disponibilizar tal energia sempre que necessário.

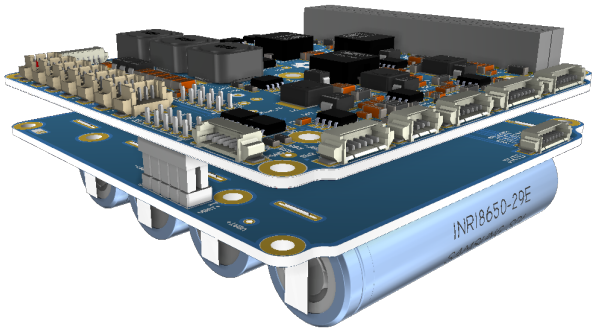
Figura 4 – Placas solares P110 da Gomspace



Fonte: Gomspace(GOMSPACE. . . , 2022)

Cada conjunto de placas é capaz de gerar até 2,3 Watts em órbita LEO. Os cálculos da quantidade de energia necessária e da quantidade de placas e bateria é feito levando-se em consideração seu ciclo de translação e posição em relação ao sol durante o ano.

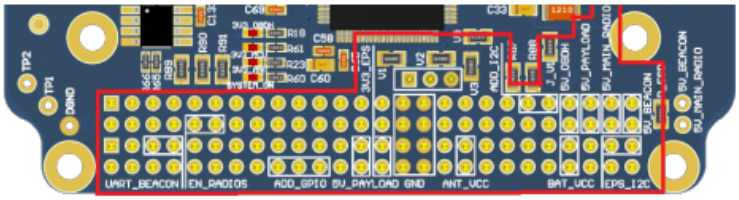
Figura 5 – EPS2, módulo de bateria e baterias




Fonte: Spacelab(UFSC-SPACELAB, 2020)

O EPS2 se comunica através do protocolo $I^2C$ com o OBDH2 através dos pinos H2-49 e H2-51 da interface PC104 e com o módulo TTC2 via UART através dos pinos H2-5 (EPS-Tx) e H2-7 (EPS-Rx).

Figura 6 – Interface PC104 do módulo EPS2




Fonte: SpaceLab(UFSC-SPACELAB, 2020)

Mais informações podem ser encontradas na documentação criada por Gabriel Marcelino e Yan Azeredo (MARCELINO; AZEREDO, 2020).

### 2.1.4 COMM

O módulo de comunicação no FloripaSat2 é chamado de TTC2, que pode ser entendido como telemetria, rastreamento e comando. É o responsável pela comunicação do satélite com a estação base.

## 2.2 Filosofia de Modelos para CubeSats

Existem inúmeras maneiras de se desenvolver um satélite partindo do levantamento de requisitos, características, restrições até o desenvolvimento de um modelo que será enviado ao espaço. Uma das metodologias satisfatórias em termos dos resultados já apresentados e à qualidade e durabilidade dos projetos enviados ao espaço é a "Filosofia de Modelos"conforme explicada por Adam Taylor (TAYLOR, 2013). Dentre os modelos, três são fundamentais para o desenvolvimento de módulos de satélites:

1. Modelo de engenharia (EM):
   Este modelo tem o mesmo formato e funcionalidades do modelo final (modelo de voo), porém é desenvolvido tanto com componentes de categoria de voo quanto comerciais, sendo portanto um modelo muito mais barato e passível de revisão.
2. Modelo de qualificação (EQM):
   Este modelo já se utiliza de componentes idênticos ao de voo e apresenta todas as melhorias percebidas no modelo de engenharia porém não precisa ser preparada em uma sala livre de poeira como no modelo de voo e o revestimento da PCB também pode ser mais simples.
3. Modelo de voo (FM):
   Modelo desenvolvido com componentes próprios para as condições do espaço, montado em um ambiente livre de poeira e com todos os testes de aprovação realizados. É o sistema completo que será enviado ao espaço.

Seguindo a Filosofia de Modelos, alguns testes são realizados até chegar-se no FM. A FlatSat é utilizada tanto no modelo EM quanto no EQM, sendo o foco deste trabalho, os testes do EM seguindo as normas já mencionadas. Já a FlatSat2 em si, foi projetada e já gerado seu FM (só em termos gerais, pois a mesma não irá ao espaço) sem as etapas de EM e EQM.

## 2.3 Padrões de testes

Para que se possa avaliar melhor os resultados obtidos e alcançar melhores resultados em termos de hardware e software, alguns padrões de testes surgiram no setor aeroespacial. Assim, para que o lançamento e a operação de um satélite possa ser bem sucedido, ele precisa seguir uma série de normas, garantindo que os resultados obtidos em cada etapa sejam satisfatórios e garantam o bom funcionamento do equipamento e pelo tempo estimado. O padrão adotado no SpaceLab, o laboratório da Universidade Federal de Santa Catarina diretamente envolvido no desenvolvimento de Cubesats, e também seguido neste documento é uma norma resumida da Agência Espacial Europeia (ESA).

Por se tratar de um projeto, apesar de complexo, muito menor que o desenvolvimento de um satélite de comunicação global geoestacionário ou um telescópio orbital, não se faz necessário aplicar normas do setor espacial tão rígidas quanto as dispostas pela Agência Espacial Européia (ESA) através do Comitê de Cooperação Européia para Normas Espaciais (ECSS) (ECSS, 2021). Conforme citado por Gouveia Júnior (JúNIOR, 2021), o documento TEC-SY/128/2013/SPD/RW escrito pela ESA apresenta as normas da ECSS que podem ser aplicadas para CubeSats e os projetos do SpaceLab seguem estas normas.

Quando se fala em testes de integração e avaliação do comportamento elétrico dos módulos, não há uma norma específica aplicável, deixando muitas lacunas quanto ao comportamento do equipamento em uma situação normal ou de estresse no espaço. Sendo assim, pode-se com a FlatSat2 implementar os módulos e monitorar a troca de mensagens de falha ou então simular falhas e analisar a resposta dos módulos a elas. Outro ponto é levar o fator tempo em consideração, deixando os módulos o mais tempo possível ligados. Pode-se assim verificar se não há esgotamento de memória ou corrupção de dados.

## 2.4 FPGAs

O nome FPGA segundo L. H. Crocket(CROCKETT et al., 2014) quer dizer “Arranjo de Portas Programáveis em Campo”, ou seja, um hardware que sai de fábrica e que pode ser programado pelo usuário e depois implementado, diferente dos Circuitos Integrados de Aplicações Específicas (ASICs) que já vem configurados de fábrica. Um FPGA é composta basicamente de três estruturas:

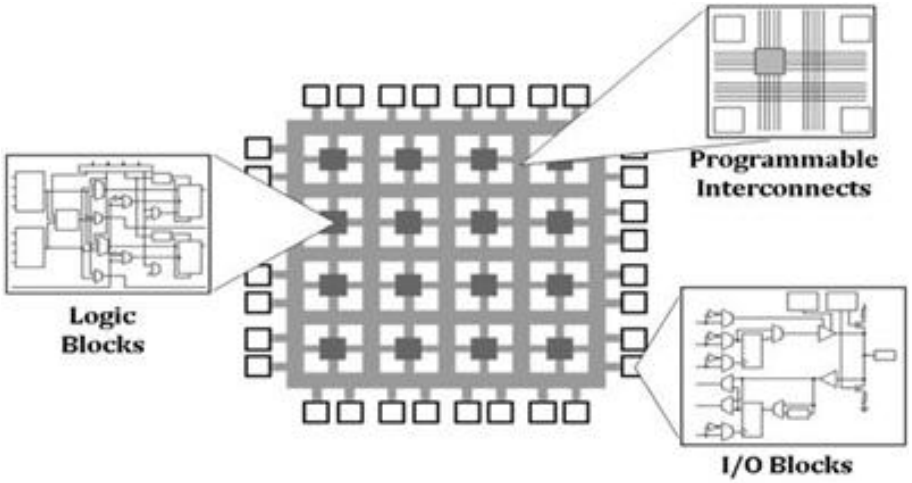
1. Configuration Logical Blocks(CLB): são a base de tudo. São células construídas em forma de matriz pela união de flip-flops e a utilização de lógica combinacional.

2. Input/Output Blocks (IOB): são os pinos de entrada e saída das células.

3. Switch Matrix: trilha que conecta os CLBs e os IOBs. Nesta camada é feito o roteamento que determina as composições de funções e conecta as entradas com as saídas. Tecnologia baseada em SRAM .

Um único FPGA pode comportar milhares destas células. O SoC utilizado neste projeto, o Zynq Z-7010 conta com 28 mil células.

Figura 7 – Matriz da FPGA




Fonte: http://www.xilinx.com/images/fpga-block-structure.gif.

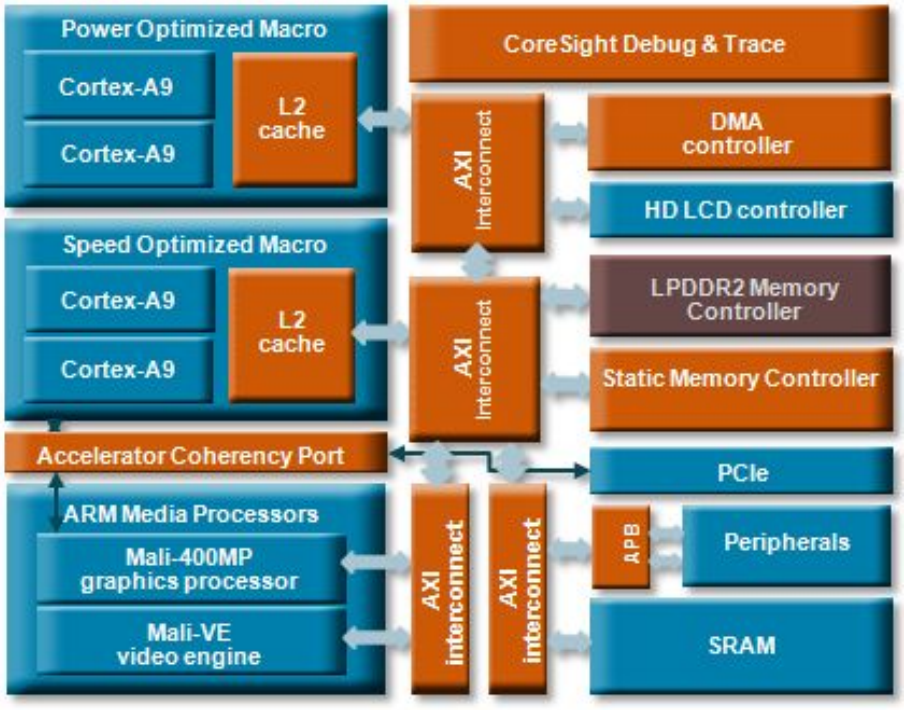
FPGAs são feitos para terem um desempenho superior à uma implementação em software e terem paralelismo em suas tarefas (entre suas entradas e saídas). Assim, são principalmente utilizados para sistemas que precisam de respostas rápida como Switches Ethernet, televisores digitais, carros autônomos, CLPs e Machine Vision. A abstração do sistema FPGA pode ser chamada de Programable Logic (PL).

## 2.5 O que são processadores ARM

A arquitetura ARM é desenvolvida por uma empresa britânica chamada ARM Holdings e licenciada para vários fabricantes de chips como Samsung, Huawei, Qualcomm e AllWinner. É a arquitetura mais comercializada até hoje com cerca de 100 bilhões de núcleos processadores vendidos até 2017 segundo o website da Statista (STATISTA. . . , 2022).

Dentre as principais vantagens frente aos competidores, está o baixo consumo, uma menor geração de calor, um alto desempenho e multiprocessamento. Muito devido a um conjunto restrito de instruções (RISC), diferente dos computadores pessoais que implementam um conjunto completo de instruções (CISC). Tudo isso fez com que fosse escolhida como a principal plataforma para celulares e outros dispositivos móveis que tem restrições de consumo de energia e dissipação de calor. Existe uma ampla gama de famílias que apresentam diferentes características devido às suas aplicações.

Figura 8 – Arquitetura ARM Cortex A9




Fonte: https://perspectives.mvdirona.com/2009/09/arm-cortex-a9-smp-design-announced/.

A família Cortex dentro dos processadores ARM existentes é a utilizada no projeto. Suas instruções e dados são de 32 bits e mais que suficiente para a maioria das aplicações. Existem 3 linhas principais de processadores ARM Cortex sendo comercializados hoje em dia: Cortex-A para uso geral (o mais comum entre os dispositivos móveis como celulares), Cortex-R para aplicações em tempo real e Cortex-M para processamento de sinais. O modelo Z-7010 conta com 2 processadores ARM Cortex-A9. Sua organização lógica pode ser vista na Figura 8

# 3 MATERIAIS E MÉTODOS

Conforme já citado, o método de teste escolhido para esta etapa é o teste de conformidade da FlatSat2 com a conferência das trilhas, montagem dos componentes e testes de transdutores de tensão com implementação em um dispositivo programável.

## 3.1 Fonte de alimentação Keysight 8032A

Para a alimentação dos transdutores de tensão e demais componentes da placa, a fonte de alimentação Keysight 8032A foi utilizada com os valores de 3.3V e 1.8V além de um valor fixo de 5V que pode ser utilizado para alimentar outros componentes como o sensor de temperatura TMP112A.

## 3.2 Osciloscópio Keysight 3014A

Para a aquisição dos sinais em tempo real, foi utilizado o Osciloscópio Keysight 3014A com dois canais disponíveis, o suficiente para ver a leitura de referência e a saída ou SCL e SDA do protocolo $I^2C$.

## 3.3 Gerador de função KeySight 33600A

Para simular sinais como o clock (SCL) do $I^2C$ e verificar possíveis atenuações de sinais, um gerador de função modelo KeySight 33600A foi utilizado, o que permitiu o controle total da onda a ser emitida.

## 3.4 Multímetro Minipa ET-1002

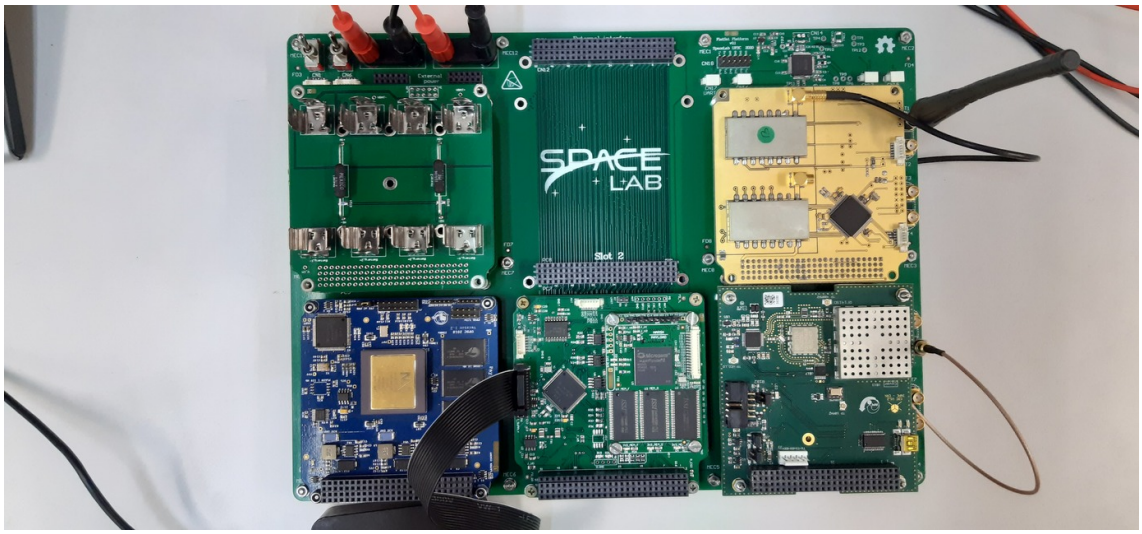
Para testes de continuidade e referenciais de tensão RMS, foi utilizado o multímetro Minipa que atende as necessidades atuais.

## 3.5 FlatSat2

Todo projeto enviado ao espaço, deve ser antes testado e validado. Diferentes metodologias podem ser implementadas, desde um ambiente virtual como em um MDVE (GABER; EL_MASHADE; AZIZ, 2020) ao modelo de engenharia com a construção física e integração de todos os componentes do satélite.

Para o projeto do FloripaSat2, feita a escolha pelo modelo de engenharia, segue-se para uma série de testes com os módulos de uma forma que estes fiquem expostos em um ou ambos os lados das placas, diferentemente do modelo em sanduíche no modelo de voo (FM) onde as placas são empilhadas. É então empregada uma placa que possa proporcionar esta integração e exposição dos módulos, a FlatSat que pode ser vista na Figura9.

Figura 9 – Placa de testes FlatSat2



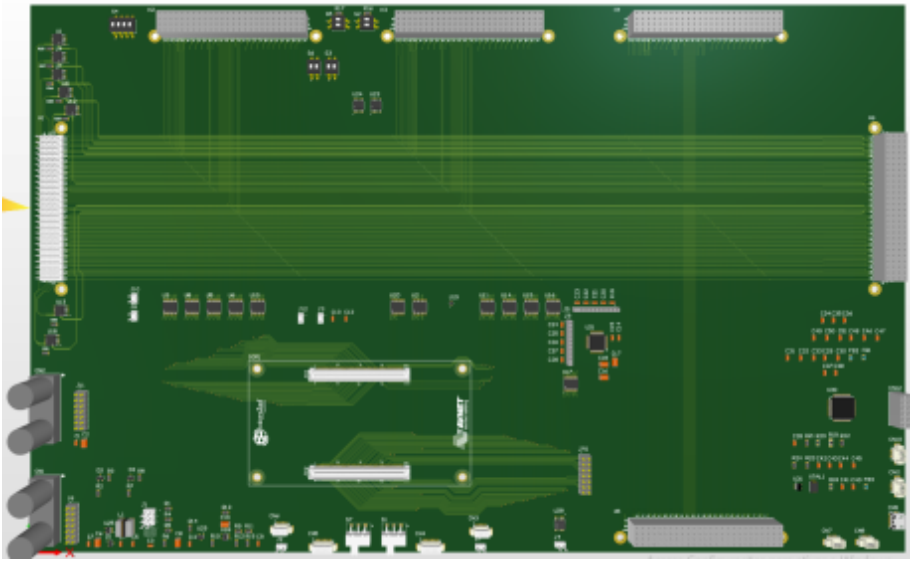

Fonte: SpaceLab(UFSC-SPACELAB, 2020).

A FlatSat pode ser simplesmente um circuito de trilhas que interligue os módulos ou então algo que apresente um processamento e uma maior imunidade a ruídos externos causados até mesmo pelas dimensões da placa. Para o projeto FloripaSat-2 foi desenvolvida uma outra placa, a FlatSat2 (Figura10) com a possibilidade de integração de uma placa de desenvolvimento onde parte dela está conectada à um processador e parte, à uma FPGA. A placa de desenvolvimento escolhida foi a MicroZed conforme já explicado anteriormente.

A MicroZed é conectada através de seus dois barramentos disponíveis na parte inferior da placa, os chamados microHeaders. Alguns se conectam diretamente a parte do processador ARM e a grande maioria é conectado à FPGA, também chamada de PL, o que dá uma grande flexibilidade, já que essas entradas e saídas podem ser reconfiguradas conforme a aplicação.

Por se tratar de um equipamento onde a maioria dos sinais são de até 1,8 V e o padrão adotado nas FlatSats ser de 3,3V, é necessário que todas essas interfaces da MicroZed passem por um circuito que converta os sinais para os dois níveis de tensão, os chamados transdutores de tensão bidirecional.

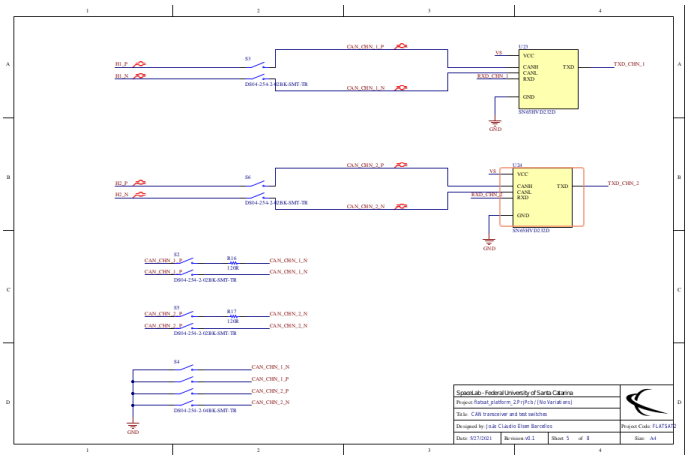
Figura 10 – Placa de testes FlatSat2



Fonte: SpaceLab(UFSC-SPACELAB, 2020).

Alguns deles são do tipo coletor aberto para, por exemplo, comunicação $I^2C$ (como os componentes U2 e U20) utilizando-se os componentes TXB0108EWPR e outros são de uso geral (U3, U5, U6, U8, U10, U11, U14, U15, U16 e U17) utilizando-se os componentes TXB0108PWR. A placa conta ainda com transdutores CAN modelo SN65HVD232D e com switches para simular falhas de comunicação na rede CAN (de S2 a S6) tanto de atenuação do sinal quanto de interrupção conforme a Figura 11.

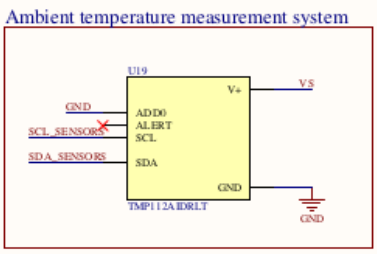
Figura 11 – Esquemático Rede CAN



Fonte: SpaceLab(UFSC-SPACELAB, 2020).

Há também dois sensores de temperatura instalados na placa. Um digital, o TMP112AIDLRT (Figura 12) que serve como um sensor de temperatura ambiente e se comunica com a MicroZed através de $I^2C$ pelos pinos JX2-49 (I2C_SCL) e JX2-50 (I2C_SDA).

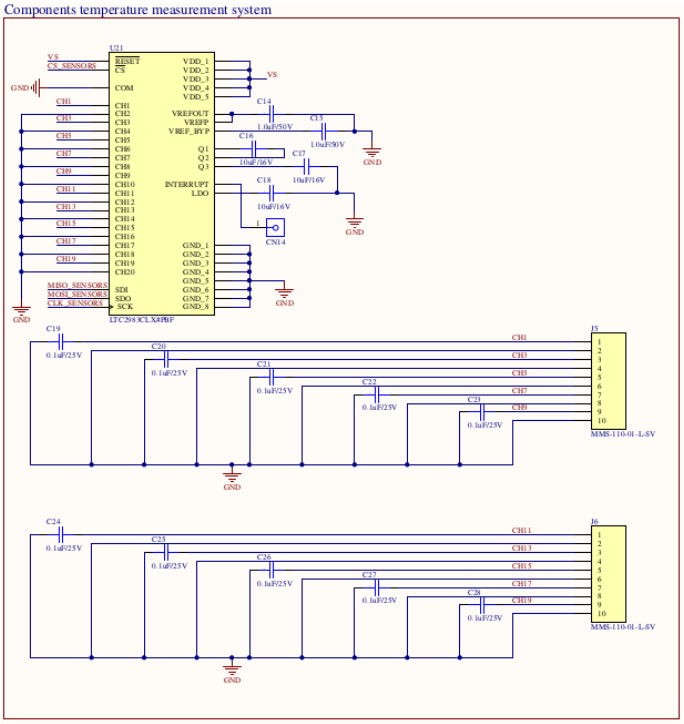
Figura 12 – Esquemático TMP112AIDLRT




Fonte: SpaceLab(UFSC-SPACELAB, 2020).

O outro é o LTC2983CLX#PBF (componente U21) que pode ler sinais de RTDs , termistores, termopares e diodos, transformando em sinais digitais de 24 bits e enviando esses dados através de comunicação SPI.

Figura 13 – Esquemático Sensor de Temperatura com comunicação SPI




Fonte: SpaceLab(UFSC-SPACELAB, 2020).

Este dispositivo é capaz de ler simultaneamente até 20 sensores em canais separados, sendo um ótimo componente de bancada para detecção de pontos de aquecimento nos diversos módulos que compõem o FloripaSat2.

Por fim, como a FlatSat2 é uma placa de testes, é muito importante levantar informações do consumo de cada módulo, o que é feito através dos sensores de corrente INA219AID instalados na FlatSat2. Esta leitura de corrente pode ser feita tanto pelo módulo EPS2 no barramento PC104 quanto pela MicroZed através do conversor bidirecional U2 que envia para os pinos JX2-49 e JX2-51 do microHeader.

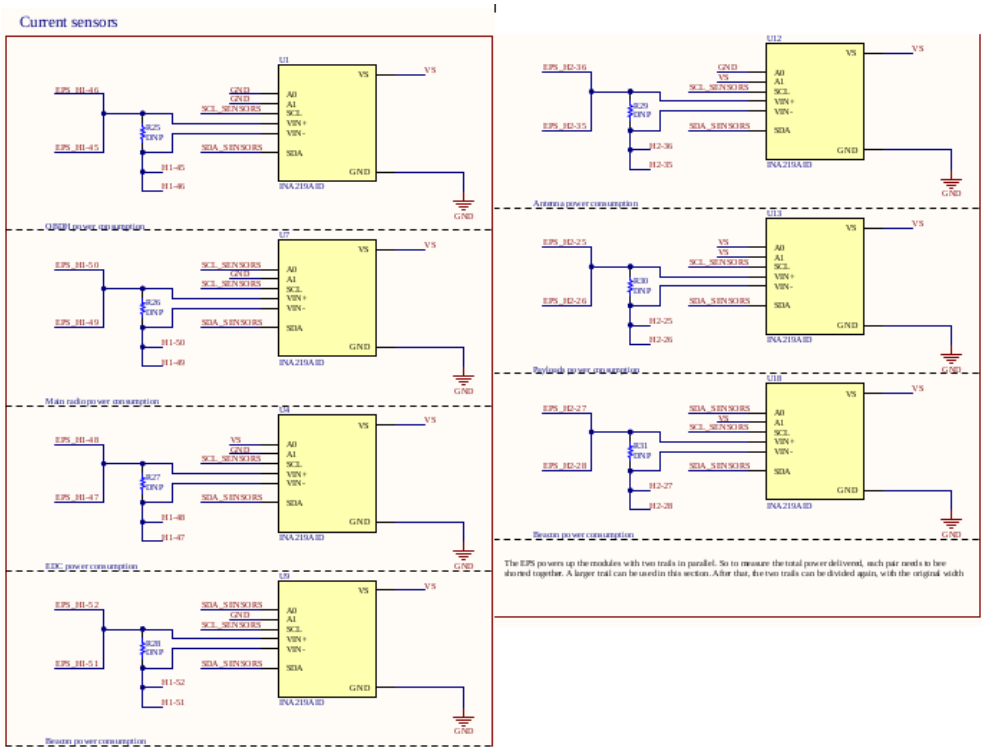
Figura 14 – Tabela de endereçamento do INA219AID

| A1 | A0 | SLAVE ADDRESS |
|---|---|---|
| GND | GND | 1000000 |
| GND | $V_{S+}$ | 1000001 |
| GND | SDA | 1000010 |
| GND | SCL | 1000011 |
| $V_{S+}$ | GND | 1000100 |
| $V_{S+}$ | $V_{S+}$ | 1000101 |
| $V_{S+}$ | SDA | 1000110 |
| $V_{S+}$ | SCL | 1000111 |
| SDA | GND | 1001000 |
| SDA | $V_{S+}$ | 1001001 |
| SDA | SDA | 1001010 |
| SDA | SCL | 1001011 |
| SCL | GND | 1001100 |
| SCL | $V_{S+}$ | 1001101 |
| SCL | SDA | 1001110 |
| SCL | SCL | 1001111 |

Fonte: (INA219AID. . . , 2022)

Conforme pode ser visto na Figura 15 a forma de diferenciação entre os sensores é feita através dos pinos A0 e A1 conforme a tabela fornecida pelo fabricante.

Figura 15 – Sensores de corrente INA219




Fonte: SpaceLab(UFSC-SPACELAB, 2020).

Cada componente realiza a leitura de corrente de um módulo sendo:

1. U1 - Consumo do OBDH2;
2. U4 - Consumo do EDC, uma payload do FloripaSat2;
3. U7 - Consumo do COMM principal;
4. U9 - Consumo do Beacon;
5. U12 - Consumo do sistema da Antena;
6. U13 - Consumo das demais payloads;
7. U18 - Consumo do segundo Beacon.

Com isso e com as rotinas de comunicação e simulação dos módulos implementados na MicroZed, pode-se levantar informações de consumo em algumas situações e verificar se os valores teóricos estão de acordo com os obtidos na FlatSat2.

Para entender como a FlatSat2 funciona em relação a MicroZed, faz-se necessário entender como é a arquitetura do SoC ZYNQ-7000, uma arquitetura ARM+FPGA com a comunicação entre ela e o barramento PC104 através dos transdutores de tensão.

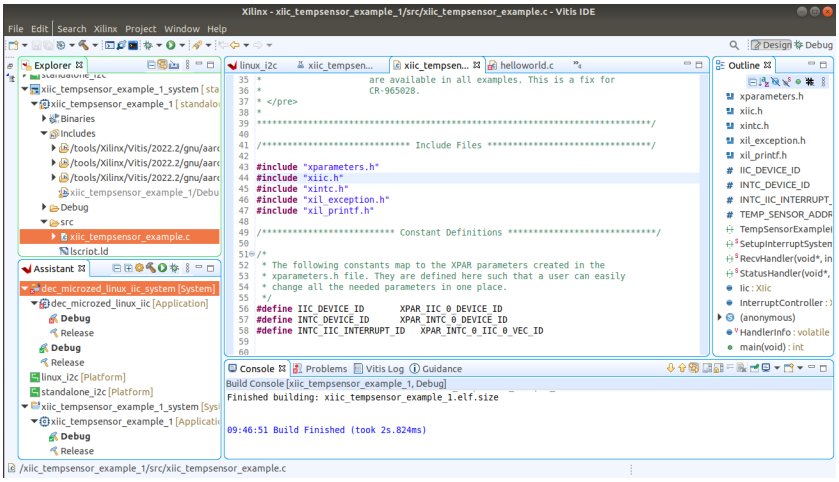
## 3.6 Ambiente de desenvolvimento

Toda a comunicação entre a placa MicroZed e os módulos do satélite através do barramento PC104 é feito por meio dos microHeaders disponíveis na parte inferior da placa MicroZed e na parte superior da FlatSat2. Todas as portas dos microHeaders estão ligados ao SoC Z-7010 através das Portas AXI. A configuração do hardware (tanto do PS quanto do PL) é feita no Vivado em sua versão 2022.2 para Ubuntu Linux 18.04.6 LTS. Com o hardware configurado, é necessário então gerar o sistema operacional e passar este sistema para a placa. Ele pode ser feito de duas formas:

1. **Vitis**

   O Vitis é o ambiente de desenvolvimento integrado (IDE) gráfico que é integrado ao Vivado a partir da versão 2020 em diante. Seu desenvolvimento muito se assemelha às IDEs de programação conforme visto na Figura 16. Nele, se pode selecionar o tipo de sistema operacional que se quer trabalhar como por exemplo Linux, FreeRTOS e uma versão Baremetal, sem depender de um Sistema Operacional. Para a transferência de dados, utiliza-se um conversor UART-Serial fornecido pelo fabricante que reconhece o dispositivo conectado e transfere o sistema operacional gerado para a memória integrada da MicroZed.

Figura 16 – Vitis IDE

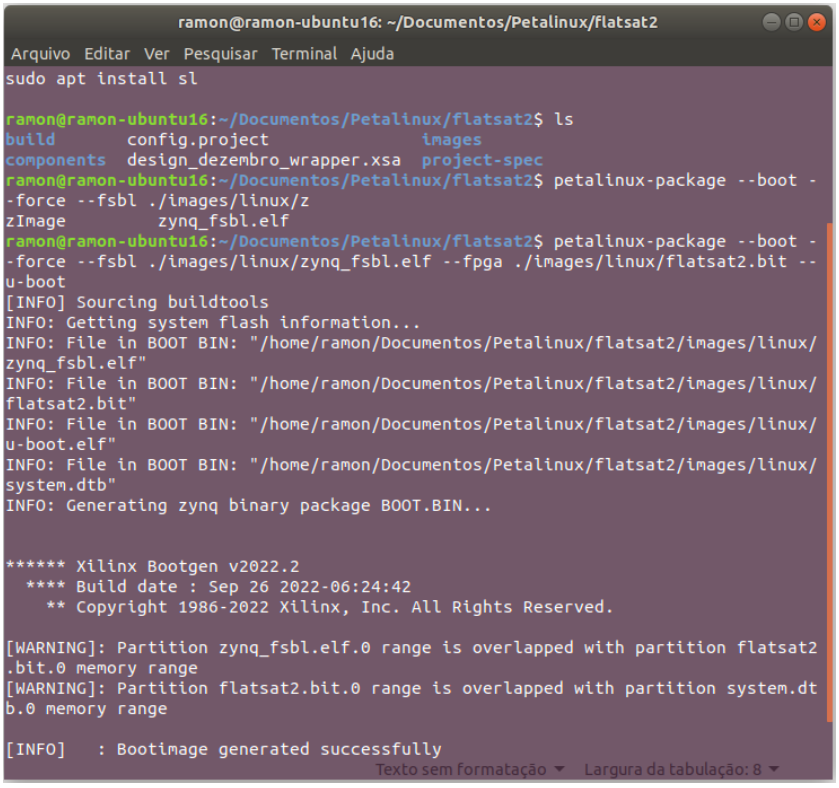



Fonte: o autor

2. **Petalinux**

   Para quem prefere o Linux como sistema operacional e está acostumado a trabalhar com comandos do shell, o Petalinux é uma boa opção. Exige um pouco mais de conhecimento de sistema operacional mas ganha-se em personalização e eficiência na implementação. Nele você configura o hardware, gera o kernel, escolhe como quer que seu SO se comporte e gera os arquivos em um cartão microSD que vai na MicroZed.

Figura 17 – Petalinux Tools




Fonte: o autor

Os resultados obtidos em ambos os ambientes (Vitis e Petalinux) serão apresentados na seção Resultados.

## 3.7 Arquitetura do Zynq-7000

Agora que já se conseguiu compreender o funcionamento de um FPGA e de um processador ARM, é preciso entender como funciona o SoC Zynq-7000 e então partir para o desenvolvimento dos códigos em VHDL para testar os módulos embarcados no FloripaSat-2. Esta seção irá descrever como o hardware se organiza dentro do chip para, no capítulo seguinte, apresentar a forma como a placa MicroZed se comunica com os módulos do FloripaSat-2.

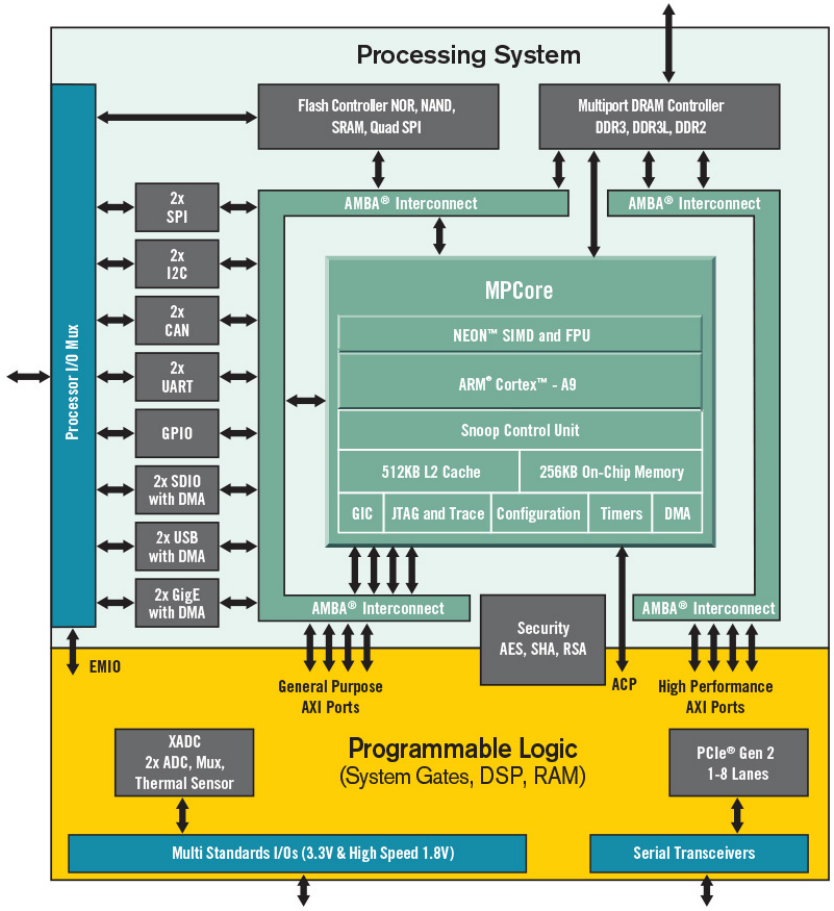
Conforme visto, um FPGA trabalha de forma diferente de um processador, microcontrolador e de um pensamento linear. Esse hardware se assemelha aos neurônios do cérebro humano e em muitos casos são usados para emular tal comportamento segundo Wolfgang Maass(MAASS, 1997). A proposta do SoC da família Zynq 7000, é reunir em um chip os dois tipos de hardware (ARM e FPGA) e os integrar de uma forma nunca antes feita(SILVA, 2014).

### 3.7.1 Xilinx Zynq Z-7010

O chip Z-7010 (Figura 18) é o presente na MicroZed. Como pode ser visto, o SoC conta com dois processadores ARM Cortex-A9 de 667 MHz, um dos mais avançados disponíveis atualmente, cada um com sua memória cache L1 dedicada de 32 kB para instrução e dados separados e com uma cache L2 compartilhada de 512 kB unificada. Conta também com 256 kB de RAM on-chip que pode ser usada para troca de dados entre o processador ARM e o PL (o controlador do FPGA). Suporte à memórias de 16 e 32 bits padrão DDR3, DDR3L, DDR2 ou LPDDR2 e também 2xQuad SPI, NOR ou NAND Flash.

Na parte de armazenamento, possui 8 canais de acesso direto à memória (DMA) — dos quais 4 são dedicados ao PL. Já com relação aos periféficos, o SoC Z-7010 conta com 2 conexões UART, 2 conexões CAN 2.0B, 2 conexões I2C, 2 conexões SPI de uso geral, e 4 conexões de 32 bits de entrada-saída de uso geral (GPIO). Vale ressaltar que uma característica interessante deste SoC é que ele é chamado de "All Programmable SoC"((XILINX, 2019), p.7), sendo tanto seu hardware, quanto seu software e suas IOs podendo ser configuradas, proporcionando uma alta flexibilidade.

Figura 18 – Comunicação entre ARM e Artix-7




Fonte: (XILINX, 2019)

Além disso, seu processador ARM funciona independente do PL, ou seja, funciona sem o PL estar configurado e até mesmo sem estar ligado (o fornecimento de energia para partes do chip também pode ser configurada). Já na parte do Programmable Logic (o controlador do FPGA), o SoC conta com um Artix-7 e 28 mil células programáveis.

O principal motivo de se construir um chip com um processador e um PL junto é a possibilidade de se ter uma uma interface entre ARM e PL extremamente rápida. A Figura 18 ajuda a entender melhor a comunicação entre PS (Processing System - compreendido pelo ARM) e PL (Programmable Logic - o controlador Artix-7).

A comunicação entre os dois ocorre pela interface AXI que oferece uma alta taxa de transferência tanto com o PS quanto com o barramento de memórias. Através do canal Accelerator Coherency Port (ACP), há um canal direto entre o PL e o Snoop Control Unit do PS permitindo a implementação de aceleradores no FPGA. Além disso, há mais de 3000 interconexões entre o PS e o PL permitindo uma largura de banda de dados de até 100 gigabits por segundo, algo que nunca poderia ser atingida em uma solução com 2 chips.

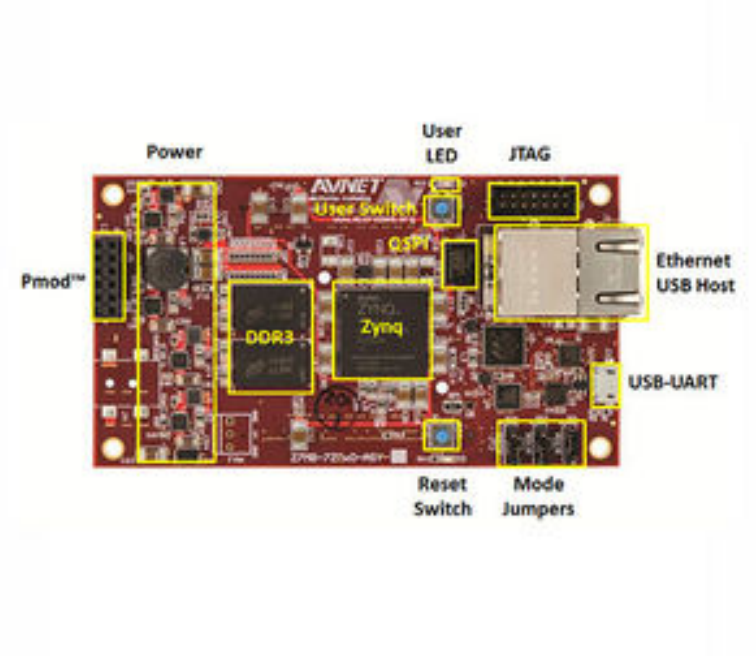
Já com relação as entradas e saídas, o SoC conta com 128 IOs controladas pelo ARM, das quais 54 são de uso geral para conexão externa e as demais estão

ligadas ao barramento AMBA que podem ser utilizadas para comunicação com memória RAM ou com o PL. Já na parte do PL, conta com até 100 IOs configuráveis para conexão externa também chamadas de Portas AXI e que, como será visto a frente, as mais importantes neste projeto.

### 3.7.2 Escolha pela SBC MicroZed

Vários dispositivos programáveis poderiam ter sido escolhidos para integrar a FlatSat2, porém uma SBC com chip Zynq-7000 foi vencedora devido à várias características já citadas como a grande variedade de IOs (104 do PL e 8 do PS nos microHeaders), a capacidade de utilizar os IOs do PL conforme se necessite redefinindo-os como entrada, saída ou bidirecional, ou como por exemplo um protocolo $I^2C$ ou SPI, a capacidade de processamento de até 12,5 GB/s de dados e a facilidade de se programar rotinas de testes e simular módulos, já que as mesmas podem ser feitas em C, C++ ou Python, recursos não disponíveis em uma FPGA pura. Sem esse SoC, algumas dessas características seriam perdidas. Já devido ao custo e necessidades do projeto, a MicroZed (Figura 19) é suficiente.

Figura 19 – Parte superior da MicroZed




Fonte: www.avnet.com.

Esta placa conta portanto com 1 GB de memória RAM DDR3, 128 MB de memória flash QSPI, interface microSD onde pode ser colocado o sistema operacional, arquivos e programas, interface USB 2.0, USB-UART e Ethernet. Estes são os pontos relevantes para este projeto.

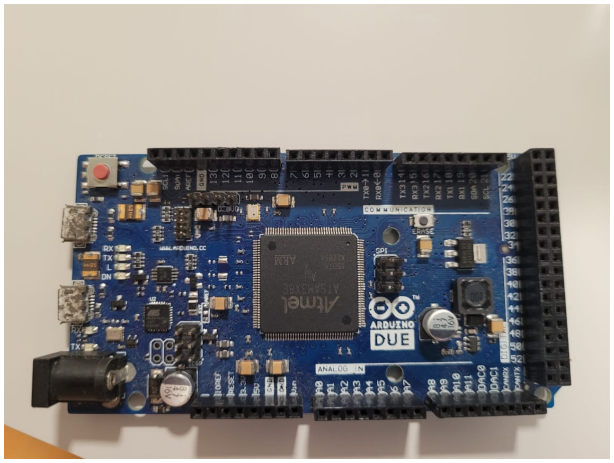
Figura 20 – Parte inferior da MicroZed com destaque para microHeaders


Fonte: www.avnet.com.

## 3.8 Uma alternativa à MicroZed nos testes da FlatSat2

Uma boa opção para validação da FlatSat2 é o uso de dispositivos programáveis menos complexos. Um deles é o Arduino Due, uma placa composta de um processador Cortex-M (já citado neste documento). Este Arduino dispõe de portas de comunicação $I^2C$, SPI e CAN.

Figura 21 – Arduino Due com processador Cortex-M (SAM3X8E)




Fonte: o autor

Outra opção é o uso de um Raspberry Pi 3B+ que pode substituir o Arduino Due ou trabalhar em conjunto com um deles conectado por exemplo no Slot 1 e o outro no Slot 2, validando o barramento PC104, já que ambas as plataformas funcionam com

sinais de 3.3 V.

Figura 22 – RaspberryPi 3B+

Fonte: O autor

Para os testes iniciais, serão utilizados ambos com o Arduino substituindo um sensor $I^2C$ e fazendo o papel do TTC2 e o Raspberry Pi fazendo o papel do OBHD2 interrogando o TTC2. Com os transdutores de tensão testados e operando, pode-se observar o sinal do lado da MicroZed para ver se a mesma tem a possibilidade de ler os sinais dos módulos, validando assim o funcionamento da FlatSat2.

# 4 RESULTADOS

## 4.1 Conferência das trilhas

Há dois tipos de conferências que precisam ser realizadas quando se trata de um projeto novo. O primeiro, que só é necessário ser feito uma vez, é verificar se o que foi projetado é o que está implementado nas placas. O outro é se a placa em questão não tem alguma trilha defeituosa ou se há um mal funcionamento de algum componente.

### 4.1.1 Verificação de continuidade

Através de um multímetro na função continuidade que emite um sinal sonoro quando a continuidade é detectada, verificou-se todas as portas dos circuitos instalados na placa partindo-se da parte superior esquerda até a parte inferior direita. Todos os pinos dos circuitos foram conferidos conforme o esquemático da FlatSat2 disponível no Github do SpaceLab. Os pinos dos slots também devem ser conferidos, conectando-se uma ponteira do multímetro no pino H1-1 do slot 1 e a outra no pino H1-1 do slot 2 e depois do slot 3 e assim até o slot 6, repetindo o processo para os demais pinos como na Figura 23.

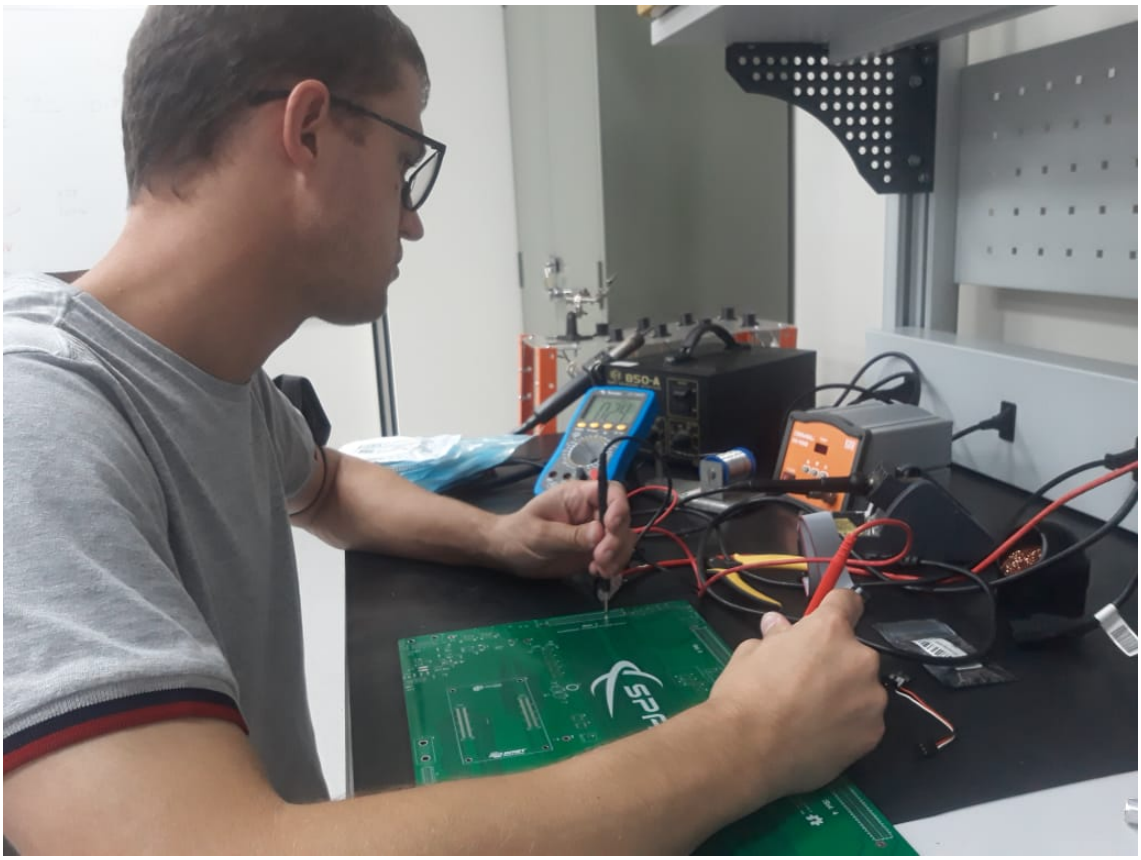
Inicialmente, sem a soldagem dos componentes, as trilhas foram conferidas seguindo-se os seguintes procedimentos:

1. Todos os sinais de tensão e terra foram conferidos nos componentes e no barramento PC104.
2. Teste das conexões entre os 6 slots disponíveis. Os slots são divididos em dois conectores (H1 e H2) com numeração de 1 a 52 para ambos, totalizando-se 104 conexões. Como as conexões são diretas sendo sua ligação feita apenas por trilhas, procedeu-se conectando o pino H1-1 do slot 1 com o pino H1-1 do slot 2, depois o pino H1-1 do slot 3 e assim sucessivamente, repetindo-se o processo também para todos os demais pinos.
3. Teste dos caminhos entre as 3 interfaces $I^2C$ da MicroZed, os transdutores de tensão comuns e sua correspondência na placa ou no PC104 e o caminho entre U2 e o sensor de temperatura ambiental instalado em U19.
4. Os caminhos entre os controladores CAN (U23 e U24), PC104 e switches (S3 e S6). Tudo correto conforme o projeto.
5. Caminhos entre MicroZed e sensor de temperatura LTC2983 (U21) com

comunicação SPI passando pelos transdutores.

Esta etapa levou cerca de 4 horas e não há divergência de projeto (com relação ao que foi projetado e o que foi implementado), prosseguindo para a verificação da qualidade do sinal.

Figura 23 – Realização dos testes de continuidade



Fonte: o autor

### 4.1.2 Verificação de qualidade do sinal

A segunda conferência ainda sem os componentes instalados é da qualidade do sinal entre alguns terminais dos slots, como por exemplo, aqueles dedicados à comunicação $I^2C$ e SPI.

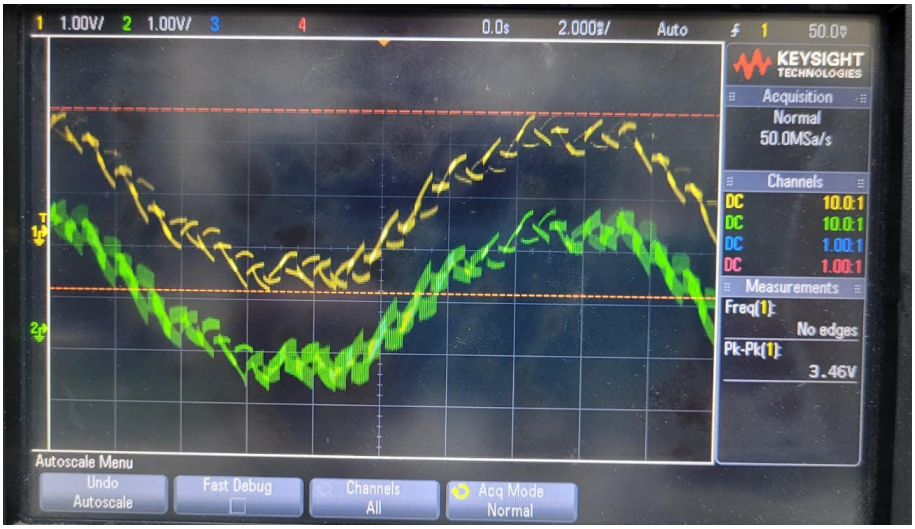
Com o auxílio de um gerador de função modelo KeySight 33600A e um osciloscópio Keysight 3014A, foi verificada a qualidade do sinal. O gerador de função foi ajustado para nivel lógico baixo em 0 V e para nível lógico alto em 2V com duty cycle de 50% conforme pode ser visto na Figura 24. Foram realizados testes nas frequências de 1 kHz, 10 kHz e 100 kHz para verificar se a atenuação é diferente nessas frequências.

Figura 24 – Ajuste do gerador de função


Fonte: o autor

Com relação ao osciloscópio, é importante ressaltar que é necessário um ajuste dos parâmetros para que o que é mostrado na tela do osciloscópio não seja a realidade do que acontece como quando se tem um aterramento mal conectado (Figura 25), quando não se aplicam filtros de baixa frequência (Figura 26) ou quando o trigger do osciloscópio está na posição errada.

Figura 25 – Teste de qualidade sem aterramento (inadequado)



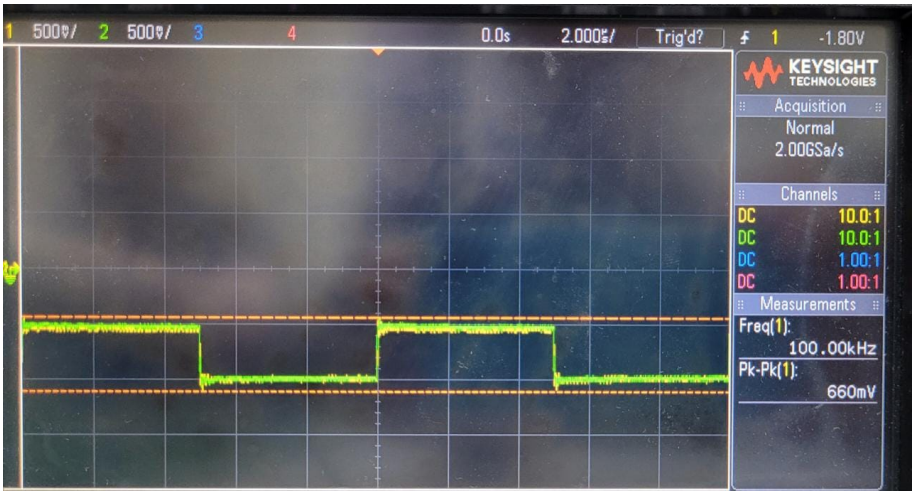

Fonte: o autor

Figura 26 – Teste de qualidade sem filtros corretos (inadequado)


Fonte: o autor

O resultado verdadeiro acaba se apresentando de uma forma bem diferente conforme é visto na Figura 27. Para gerar esta imagem, o gerador foi ajustado para um nível de tensão de 600 mV e com a frequência de 100 kHz, o que foi obtido no osciloscópio conforme a Figura 27.

Figura 27 – Realização dos testes de continuidade




Fonte: o autor

O teste foi repetido também para as frequências de 1 kHz e 10 kHz mas não houve variação no resultado de saída, sendo sua resposta idêntica a da entrada, comprovando uma baixíssima atenuação do sinal e nível de ruido para estas frequências. Com isso, constata-se que as trilhas irão se comportar bem sob tais níveis de tensão e frequência.

## 4.2 Soldagem dos componentes

Conforme a norma resumida da ECSS adotada, a composição da solda deve ser de somente chumbo-estanho sendo permitido apenas traços de outros metais na proporção inferior a 1% a fim de se evitar problemas como de corrosão por *red plague*.

O material de solda apresenta-se em estado pastoso, o que facilita sua aplicação em circuitos SMD mesmo sem o uso de stencil e sua composição está dentro do permitido pela norma ECSS, sendo composta de 63% de chumbo e 37% de estanho conforme pode ser visto na Figura 28.

Figura 28 – Pasta de Soldagem

Fonte: O autor

Apenas uma quantidade ínfima deve ser aplicada com cuidado ou com o auxílio de um cotonete, já que as trilhas são muito pequenas e próximas, o que acarreta no curto-circuito (contato entre dois pinos) quando aplicada uma quantidade muito grande de solda ou aquecimento inadequado. Na maioria das situações uma estação de solda com transferência de calor por contato é o mais indicado com movimentos no sentido longitudinal dos pinos, requerendo uma certa experiência e firmeza das mãos do operador.

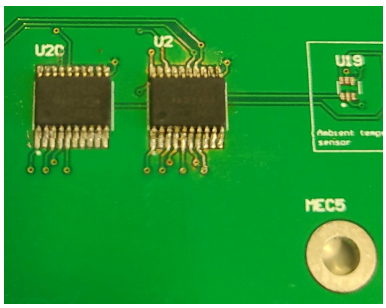
Uma estação de retrabalho que opera com fluxo de ar quente também pode ser utilizada para limpeza e garantir que terminais contíguos não sejam soldados. Nesta situação, aplica-se um pouco de fluxo pastoso nos terminais. Em seguida, com uma pinça, aplica-se uma leve pressão sob o componente e passa-se o bico do soprador de ar sob os terminais observando-se quando a solda inicia-se a se fundir deslocando o bico da estação para os próximos pinos e com cuidado para não sobreaquecer o componente.

### 4.2.1 Rotina de montagem

Para garantir o funcionamento correto após a montagem de cada placa, uma rotina de montagem e conferência foi desenvolvida baseada em boas práticas do setor. Os passos a serem seguidos são:

1. **Instalação dos resistores e capacitores:** Pode-se procecer com a instalação desses componentes que são os menos complexos por apresentarem apenas dois terminais. Tanto a solda em pasta quanto a de fio com fluxo podem ser utilizadas desde que sua composição seja de chumbo-estanho e dentro das normas da ECSS. A estação de retrabalho pode ser utilizada com moderação para que o componente não seja ejetado para longe, seguindo-se as recomendações iniciais desta seção.
2. **Teste de continuidade dos resistores e capacitores:** Após a fixação dos componentes disponíveis, pode-se desligar os equipamentos de solda e utilizar um multímetro para verificar se a solda não está fechando curto por debaixo do componente. Assim, conecta-se os terminais do multímetro na função continuidade entre os dois terminais dos componentes soldados e o resultado esperado é que não surja sinal sonoro. Por fim, testa-se o multímetro conectando os dois terminais neles mesmos e deve-se ouvir o sinal sonoro.
3. **Instalação dos transdutores de tensão:** Existem 12 transdutores de tensão instalados na placa, sendo dois deles (U2 e U20) do modelo TXS0108EPWR (para $I^2C$) e os demais do modelo TXB0108PWR de uso geral. Por apresentarem a mesma configuração de pinos conforme a Figura 30 este procedimento pode ser aplicado para todos. Eles devem ser posicionados sob a placa na posição correta conforme a Figura 29 e aplicada a solda pastosa e soldados conforme as recomendações iniciais desta seção iniciando-se pelo processo com estação de solda e podendo-se finalizar com a estação de retrabalho, o que nem sempre é necessário.

Figura 29 – Modo de instalação dos transdutores




Fonte: O autor

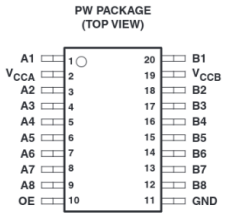
4. **Conferência inicial dos transdutores:** Por apresentarem vários pinos e com dimensões reduzidas, após a soldagem é necessário garantir que os pinos vizinhos não estejam soldados.
   Com o multímetro na função continuidade inicia-se testando o próprio multímetro conectando-se seus terminais e aguardando o sinal sonoro. Em seguida, conecta-se um dos terminais no pino 1 e o outro no pino 2. Depois entre o pino 2 e pino

3 e assim por diante até o último pino da linha. Nenhum sinal sonoro deve ser observado e caso positivo, proceder retirando o excesso com o auxílio de uma malha de cobre dessoldadora ou aplicando o chamado fluxo e com uma estação de retrabalho em casos onde uma inspeção visual não detecta o excesso de solda.

Em contextos mais drásticos onde o contato entre os pinos ocorre e não se identifica o defeito, faz-se necessário a remoção e limpeza do componente, que pode ser feito com o correto uso da estação de uma estação de retrabalho e uma amálgama, comumente chamada de "salva chip", de um metal de menor ponto de fusão que a liga chumbo-estanho com a função de facilitar a transferência de calor entre os pinos. O uso da amálgama só é necessário onde não se consegue aquecer todos os pinos com o fluxo de ar, como no caso dos microHeaders. Só após a conferência visual e com o auxílio do multímetro na função continuidade e em caso negativo é que se procede-se para a conferência do próximo componente.

5. **Funcionamento dos transdutores:** Deve-se agora alimentar a placa com a fonte com dois níveis de tensão: 3.3 V e 1.8V respectivamente em VCCB e VCCA. Conforme dados do fabricante do TXS0108EPWR e conforme pode ser visto na Figura 30, com o multímetro na função tensão contínua e limite superior a 5V, conectamos o terminal preto em qualquer terra e testamos os pinos 2 que deve apresentar a tensão de 1.8V fornecida pela fonte e 3.3 V também fornecida pela fonte. Qualquer valor diferente disso indica que há um curto ou o componente é defeituoso, inicialmente retrabalhando a solda e caso não seja solucionado, trocando o componente.

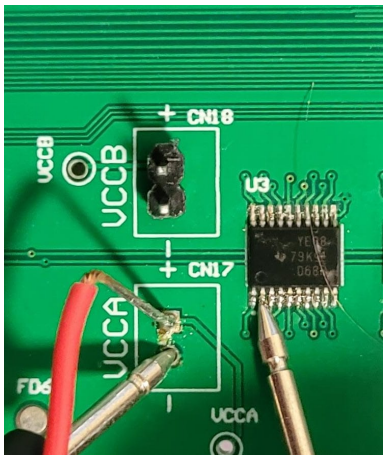
Figura 30 – Conexões do TXS0108EPWR




Datasheet do TXS0108EPWR

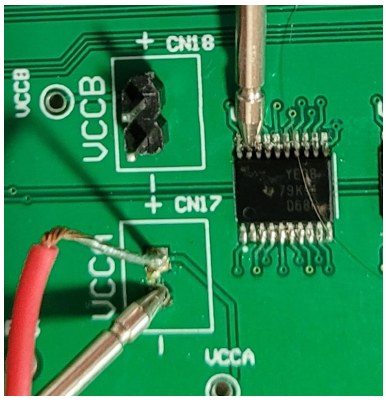
Durante a montagem, foi necessário o retrabalho de alguns componentes. Um cuidado que se deve também tomar é que se houver alguma conexão errada, o componente pode sobreaquecer e até estourar. Para evitar isso, liga-se a fonte por cerca de 1 segundo e novamente desliga-se. Com o dedo indicador, é aplicado

Figura 31 – Tensão em VCCA: 1.8V




Fonte: o autor

Figura 32 – Tensão em VCCB: 3.3V




Fonte: o autor

um leve contato sob o circuito para se observar se o mesmo aqueceu. Este tempo em geral é suficiente para aquecer o componente mas não provocar sua perda, podendo então se corrigir o erro sem a necessidade de troca do componente.

6. **Conexão dos microHeaders** Este talvez seja o componente mais complicado em se instalar devido a quantidade, dimensões e posição dos pinos. Deve-se proceder com a soldagem e testes dos pinos vizinhos igualmente feito com os transdutores.

A instalação dos transdutores de tensão e dos conectores microHeaders, demais conectores para alimentação, resistores e capacitores SMD permitiram os testes dos transdutores de tensão e do protocolo $I^2C$, já que a Microzed pode ser conectada à FlatSat e outros dispositivos podem ser conectados aos slots. Os demais testes devem ser realizado assim que os componentes chegarem.

O próximo passo para a continuação do projeto é a conexão dos módulos CAN. Esta etapa não foi realizada devido à falta dos componentes. Segundo Emerson da Trindade Lemos(LEMOS, 2021), o protocolo CAN será adotado para a comunicação entre os componentes dos futuros satélites por apresentar, conforme os testes

realizados, uma melhor confiabilidade para a situação, substituindo por exemplo os protocolos $I^2C$ e SPI.

Os demais componentes serão montados assim que adquiridos, já que, devido a escassez de componentes muito em função da pandemia de COVID-19, não constam em estoque nos principais fornecedores globais e não tem previsão de fornecimento.

Como as soldas foram todas manuais, sem a utilização de um estêncil e uso de solda pastosa, foram testados os pinos vizinhos de cada componente para ver se a solda não estava os conectando. Os que apresentavam alguma inconsistência foram refeitos. Cerca de 30% dos componentes foram montados.

Devido a escolha da forma de execução adotada aqui, este trabalho levou cerca de 10 horas em função do retrabalho de alguns componentes. Acredita-se que com o uso de estêncil e uma forma de aquecimento automatizada e por completo da placa, como nos processos THR e SMT, este tempo possa ser reduzido para menos de 1 hora.

## 4.3 Testes com dispositivos programáveis

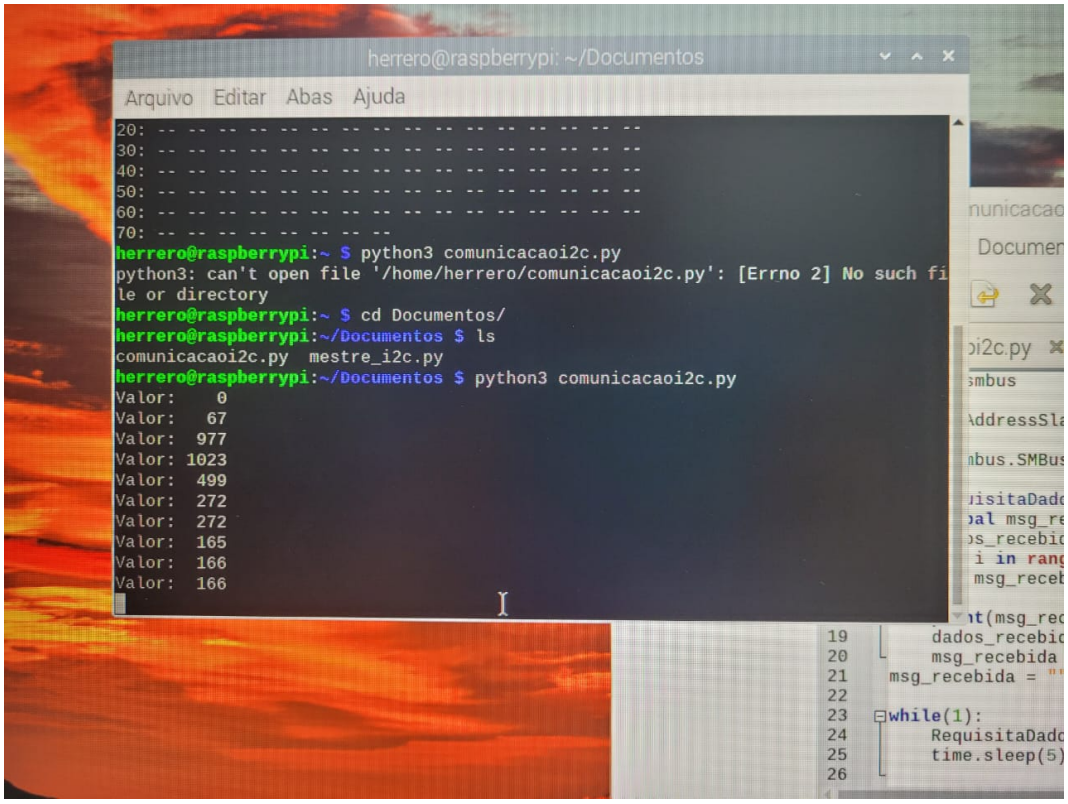
Quando se implementa um protocolo de comunicação através de um barramento e coordenado por um dispositivo programável, muitas variáveis são inseridas e muita coisa pode dar errado. Na tentativa de minimizar esses erros e possibilitar testes com incremento de complexidade, pode-se utilizar dispositivos programáveis já testados e mais simples e que contam com bibliotecas prontas e testadas como é o caso do Arduino e do Raspberry Pi.

### 4.3.1 Dispositivos de menor complexidade

Desta forma, partiu-se de algo simples e aumentou-se a complexidade à medida que os resultados foram positivos, seguindo-se os seguintes passos:

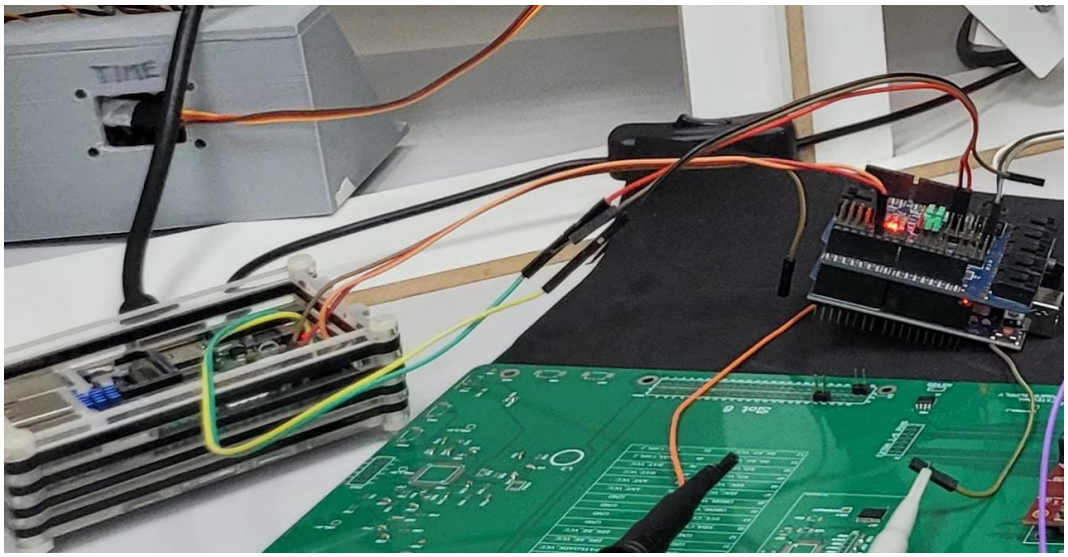
1. Arduino escravo $I^2C$ e Raspberry Pi mestre:
   Foi implementada a aplicação com o Arduino sendo um escravo $I^2C$ com endereço 0x18 e nele conectado um potenciômetro em A0 para simular uma variação de temperatura ou corrente. Os pinos utilizados para comunicação são A4 (SCL) e A5 (SDA). Com um Raspberry Pi como mestre $I^2C$, pode-se interrogar o Arduino uma determinada quantidade de vezes e ter-se uma análise de perda de pacotes. Os pinos do Raspberry Pi a serem conectados são pino 3 (SCL) e 5 (SDA) do conector de 40 pinos conforme indicação numérica do fabricante.
   A comunicação entre os dois dispositivos foi feita através de alguns pinos do barramento PC104 conforme a montagem da Figura 34 e na situação onde os slots estão mais separados (o que seria o pior cenário pois é onde se tem mais interferências).

Figura 33 – Troca de dados I²C entre Arduino e Raspberry Pi




Fonte: o autor

Figura 34 – Testes com Arduino e Raspberry Pi



Fonte: o autor

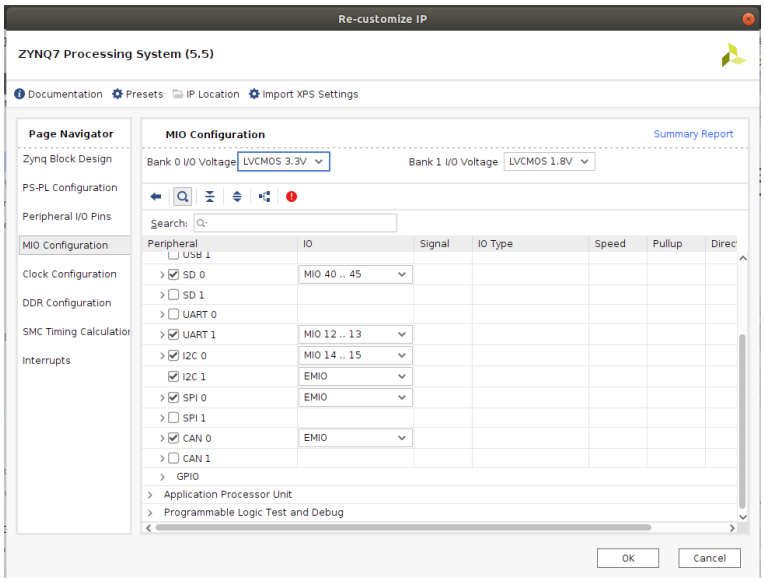
Com uma frequência padrão de 100kHz do clock (SCL), não houveram perdas para uma quantidade de 1000 pacotes nos pinos testados e o resultado da leitura foi obtido conforme a Figura 33.

### 4.3.2 Implementação com a MicroZed

Após confirmação das primeiras etapas (conferência das trilhas e análise da qualidade do sinal) e testes com os dispositivos de menor complexidade, prosseguimos com a implementação na MicroZed, também de forma incremental para se detectar falhas, de acordo com os seguintes passos:

1. **Funcionamento com um sistema operacional simples:** Primeiramente, precisamos garantir que a placa de desenvolvimento está funcionando. O fabricante tem em seu site (www.avnet.com) uma série de arquivos e dentre eles a imagem do sistema operacional e um manual *MicroZed Getting Started* onde se tem um pequeno tutorial com alguns comandos de *blink* e *echo* e a forma de se configurar os *jumpers* para buscar o SO no cartão SD. Estes passos foram realizados para um pequeno aprendizado e garantiu-se assim que a placa e o computador utilizado no desenvolvimento estão funcionando e são capazes de trocar dados através da USB e com um software de terminal serial (Putty). Por serem os mesmos passos e os mesmos resultados obtidos no manual citado, não foram documentadas essas etapas.
2. **Funcionamento do SO com um hardware personalizado:** Com o auxílio do Petalinux, pode-se gerar um SO Linux baseado em um hardware desenvolvido no Vivado e exportado no formato XSA. Realizou-se então a configuração do hardware com um barramento $I^2C$ implementado na FPGA para a leitura da temperatura do sensor TMP112 e os dados sendo enviados ao processador ARM. Além disso, a comunicação serial UART e outros dois protocolos $I^2C$ também foram configurados no PS clicando-se sob o processador no diagrama de blocos e na aba *MIO Configuration* conforme pode ser visto na Figura 35.

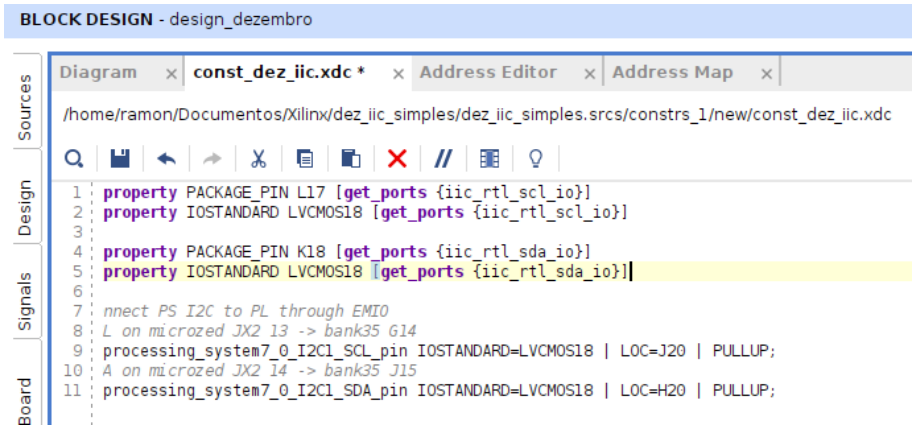
Figura 35 – Configuração dos protocolos no PS




Fonte: o autor

O canal I2C_1 foi configurado como uma EMIO que vai para o PL. Assim, foi necessário criar-se um arquivo de *constraints* (.xdc) para a conexão física do hardware no PL com os dois protocolos I²C, transdutor do TMP112 e sensores de corrente nos pinos JX2-49 e JX2-50 e a comunicação com o TTC2 nos pinos JX2-68 (I2C_1_SCL) e JX2-70 (I2C_1_SDA) do microHeader. Nesta etapa foi encontrada uma necessidade de reprojeto da placa e que será discutido melhor no final deste capítulo. A configuração apresentada aqui já implementa esta correção.

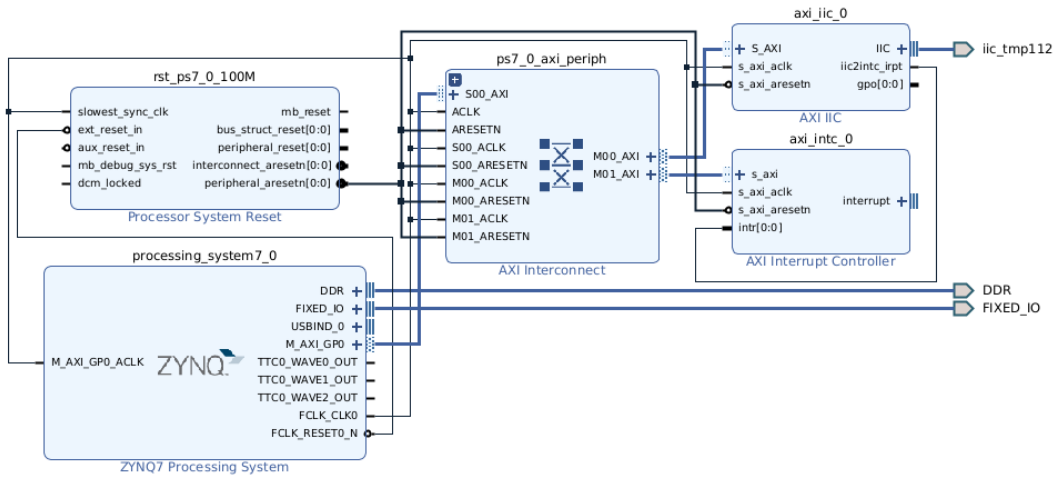
Figura 36 – Configuração do PL para os microHeaders




Fonte: o autor

Além do arquivo de *constraints*, é necessário gerar o diagrama de blocos com a configuração do PL. Em função da praticidade, foram utilizados os blocos disponibilizados pelo Vivado e sem interrupções (apesar de uma versão com controlador de interrupção também ter sido desenvolvida). O hardware ficou conforme a Figura 37.

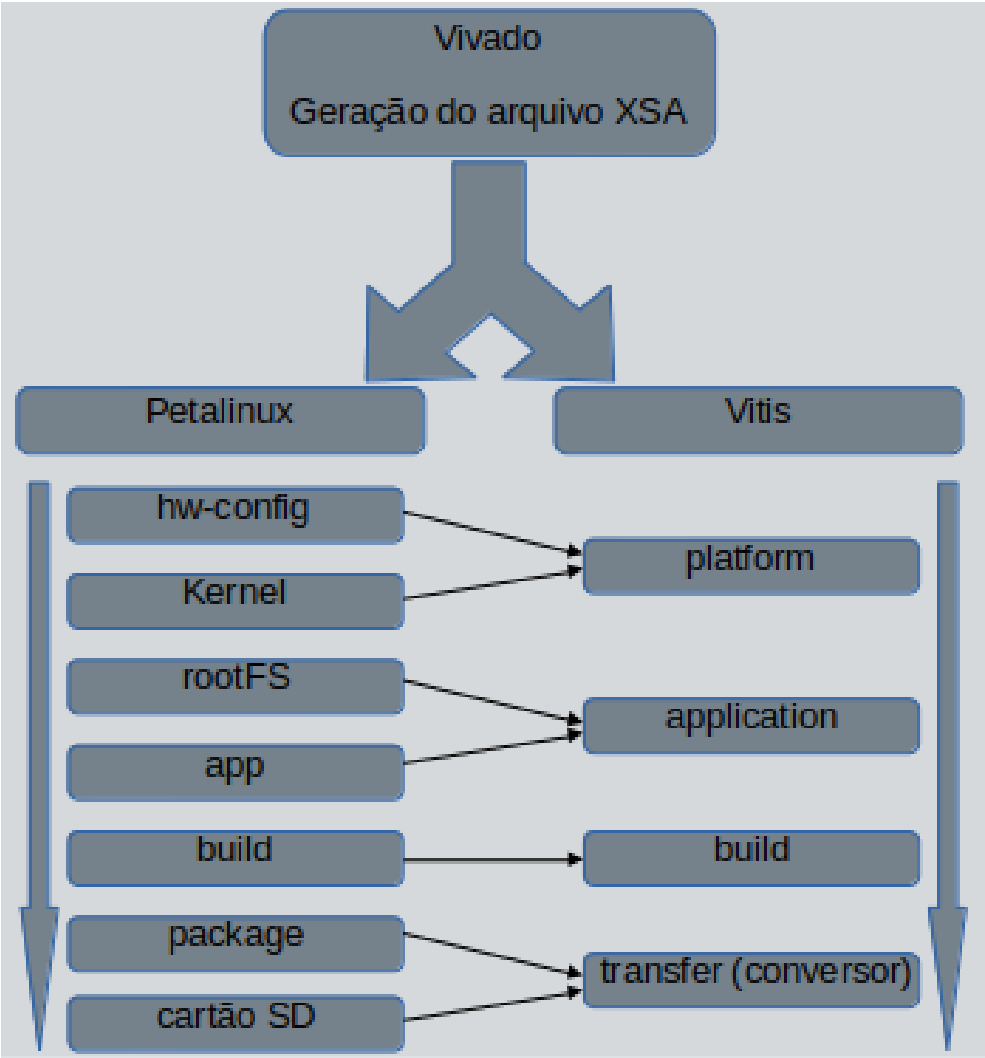
Figura 37 – Diagrama de Blocos do I²C




Fonte: o autor

Após a construção do diagrama e do *wrapper* através do comando *Create HDL Wrapper...*, gerou-se o *bitstream* que é o arquivo necessário para se gravar as conexões físicas (rotas) das células da FPGA e saídas do PL e o PS. Após a geração, é necessário selecionar a opção *Exportar hardware* que cria o arquivo XSA utilizado tanto pelo Vitis quanto pelo Petalinux. Neste momento então, deve-se optar pela implementação em um dos dois sistemas. Os passos necessários são descritos conforme a Figura 38.

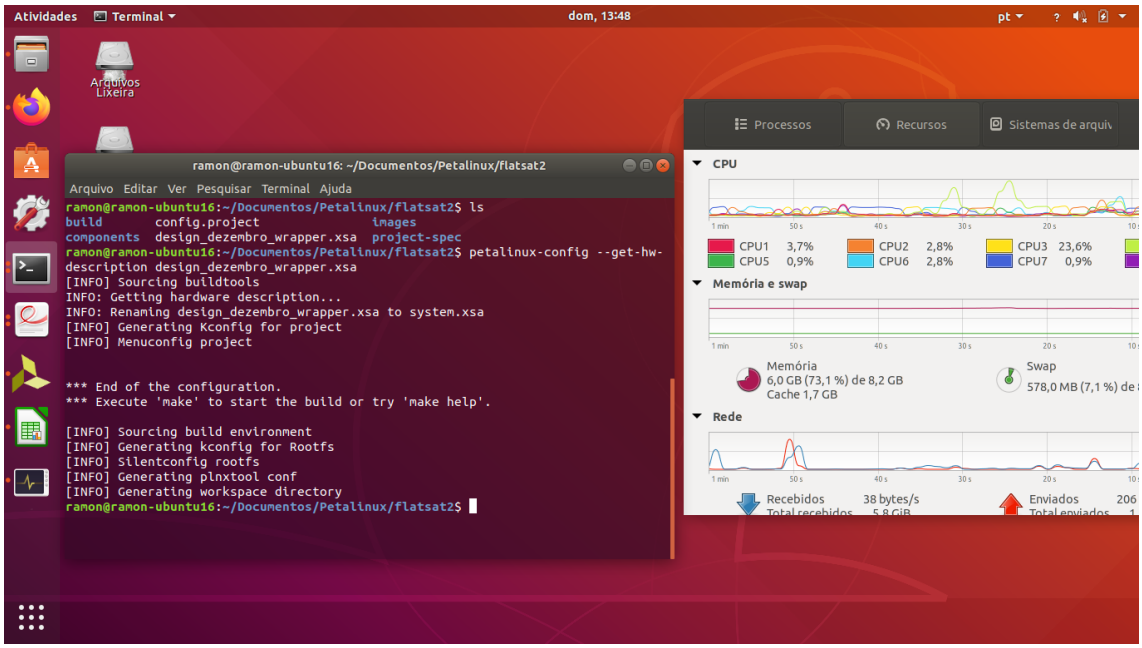
Figura 38 – Diagrama de construção do Sistema Operacional




Fonte: o autor

No segundo caso, precisa-se mover este arquivo para o diretório onde o projeto foi criado e a configuração do hardware ocorre com o comando "petalinux-config –get-hw-description <nome_do_XSA>". Após o comando, é aberto o menu de configuração e inicialmente pode-se só aceitar as configurações padrão.

Figura 39 – Configuração do hardware no Petalinux



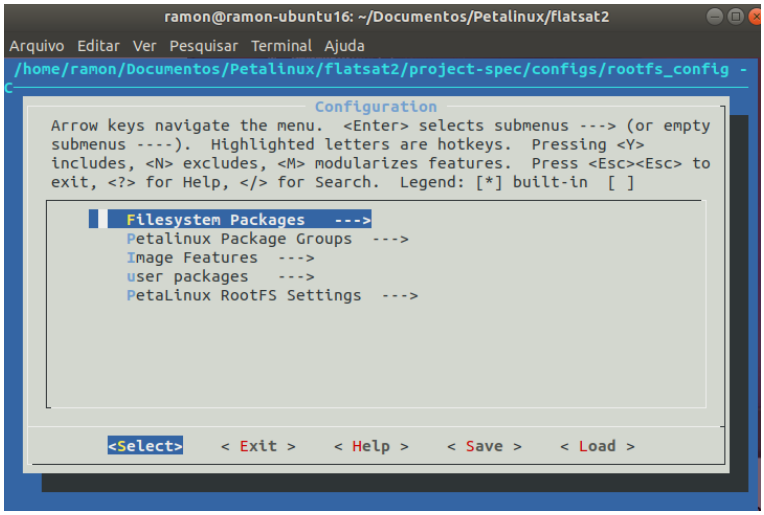

Fonte: O autor

O resultado foi positivo conforme a Figura 39. Caso alguma alteração de hardware seja feita no Vivado, é necessário executar novamente este passo. Inicialmente o hardware é implementado com uma aplicação tipo *hello world*. Esse processo foi realizado no Petalinux. Os passos para geração de um sistema operacional no Petalinux são:

a) **Criação do projeto:** O projeto é criado no Petalinux com o comando *petalinux-create* e com os parâmetros adequados.
b) **Configuração do hardware:** São os mesmos passos descritos anteriormente e apresentados na Figura 39.
c) **Criação das aplicações:** Nesta etapa, aplicações (ou comumente chamados de programas) podem ser criados e construidos junto com o sistema operacional. Para isso, utiliza-se o comando *petalinux-create -t apps –name <nome do aplicativo> –template c*. Assim ele cria uma aplicação *hello world* que pode ser alterada em um editor de textos ou IDE de programação. Sua compilação ocorre no último passo ao se gerar o SO.
d) **Geração do Kernel:** Com o hardware configurado, as configurações do kernel do Linux baseado no processador ARM e no hardware gerado são feitas. Assim como em qualquer sistema operacional vários parâmetros estão disponíveis como tratamento de deadlocks, tamanho de pilhas, gerenciamento de memória, tratamento de erros e outros estão disponíveis. O utilizado foi o padrão, sem qualquer alteração e também obteve-se resultado positivo.

e) **Geração do rootFS:** Nesta etapa é construida a árvore de processos com as bibliotecas que se deseja adicionar ao SO, selecionadas as aplicações que deseja-se compilar e as configurações de login. Isto é feito executando o comando *petalinux-config -c rootfs* e precisa-se, após criada a aplicação, selecionar para compilar a mesma na aba *apps*.

Figura 40 – Configuração do rootFS no Petalinux

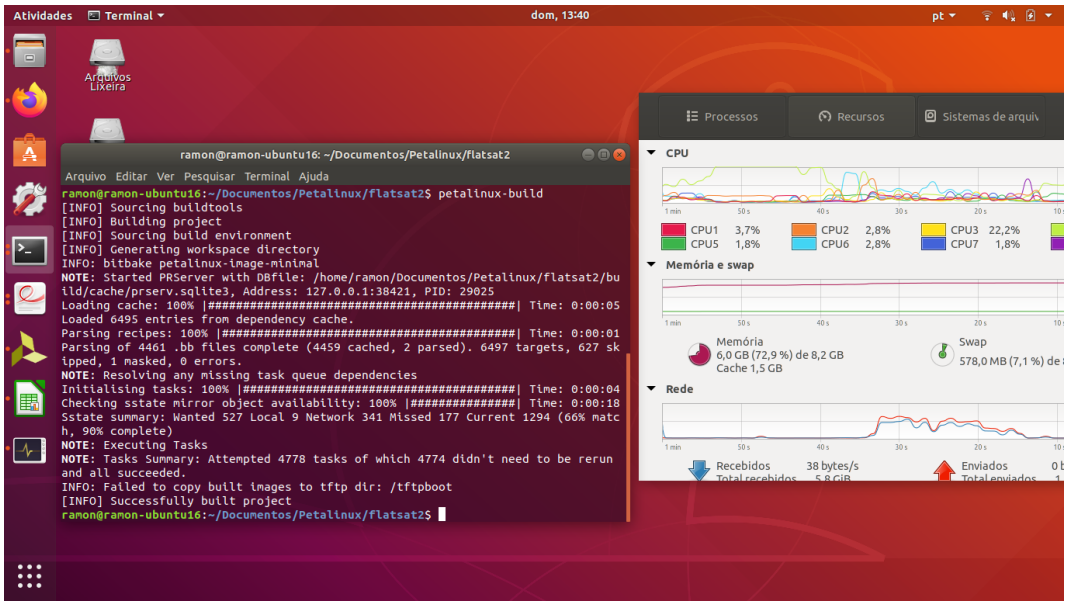



Fonte: o autor

Conforme visto na Figura 40 tudo está correto e funcionando.

f) **Compilação do Sistema Operacional:** A compilação do sistema operacional é feita através do comando *petalinux-build* e também deu tudo certo com a aplicação sendo compilada conforme a Figura 41.

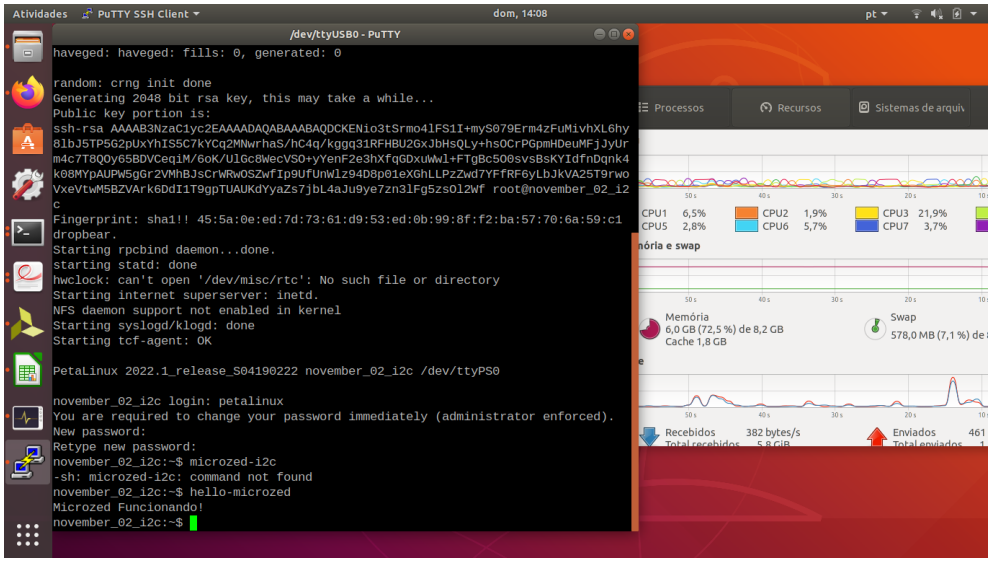
Figura 41 – Sistema operacional compilado com o hardware criado no Vivado




Fonte: o autor

g) **Geração do pacote e transferência para cartão SD:** Por fim, é necessário gerar o arquivo *rootfs.cpio* que é feito executando-se o comando *petalinux-package* e descompactá-lo dentro da partição rootFS através do comando *sudo pax -rfv rootfs.cpio*. Todos estes passos estão no manual do Petalinux disponível no site da Xilinx.
Até aqui, tudo foi implementado com exito, estando o atual cartão SD com esta imagem. Para abrir o sistema operacional como usuário deve-se digitar *petalinux* e inserir uma nova senha. Para checar se está tudo correto deve-se digitar no terminal do Putty *microzed-i2c* e o resultado esperado é o mostrado na Figura 42.

Figura 42 – Sistema operacional compilado com o hardware criado no Vivado




Fonte: o autor

3. Hardware específico e software de aplicação:
Por último, gerar uma aplicação que utiliza este hardware específico implementado no PL para que o PS possa processar e mostrar resultados ao usuário.
Para gerar a aplicação no Petalinux, basta alterar o código da aplicação criada incluindo as bibliotecas necessárias como *xiic.h*
Caso se queira configurar todo o hardware, precisa-se seguir tanto o que foi projetado na FlatSat2 como pode ser visto na correspondência da memória fornecida pelo fabricante e pode ser visto nas Figuras 43 e 44.

Figura 43 – Pinagem do MicroHeader JX1

| JX1 Pin | MicroZed Net Name | Zynq Pin | Zynq Name | VCCO Bank | JX1 Pin | MicroZed Net Name | Zynq Pin | Zynq Name | VCCO Bank |
|---|---|---|---|---|---|---|---|---|---|
| 1 | JTAG_TCK | F9 | TCK_0 | -1 | 51 | GND | A8 | GND | -1 |
| 2 | JTAG_TMS | J6 | TMS_0 | -1 | 52 | GND | A8 | GND | -1 |
| 3 | JTAG_TDO | F6 | TDO_0 | -1 | 53 | JX1_LVDS_14_P | T20 | IO_L15P_T2_DQS_34 | 34 |
| 4 | JTAG_TDI | G6 | TDI_0 | -1 | 54 | JX1_LVDS_15_P | V20 | IO_L16P_T2_34 | 34 |
| 5 | NetJX1_5 | #N/DISP | #N/DISP | #N/DISP | 55 | JX1_LVDS_14_N | U20 | IO_L15N_T2_DQS_34 | 34 |
| 6 | NetJX1_6 | #N/DISP | #N/DISP | #N/DISP | 56 | JX1_LVDS_15_N | W20 | IO_L16N_T2_34 | 34 |
| 7 | FPGA_VBATT | F11 | VCCBATT_0 | -1 | 57 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 8 | FPGA_DONE | R11 | DONE_0 | -1 | 58 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 9 | JX1_SE_0 | R19 | IO_0_34 | 34 | 59 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 10 | JX1_SE_1 | T19 | IO_25_34 | 34 | 60 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 11 | JX1_LVDS_0_P | T11 | IO_L1P_T0_34 | 34 | 61 | JX1_LVDS_16_P | Y18 | IO_L17P_T2_34 | 34 |
| 12 | JX1_LVDS_1_P | T12 | IO_L2P_T0_34 | 34 | 62 | JX1_LVDS_17_P | V16 | IO_L18P_T2_34 | 34 |
| 13 | JX1_LVDS_0_N | T10 | IO_L1N_T0_34 | 34 | 63 | JX1_LVDS_16_N | Y19 | IO_L17N_T2_34 | 34 |
| 14 | JX1_LVDS_1_N | U12 | IO_L2N_T0_34 | 34 | 64 | JX1_LVDS_17_N | W16 | IO_L18N_T2_34 | 34 |
| 15 | GND | A8 | GND | -1 | 65 | GND | A8 | GND | -1 |
| 16 | GND | A8 | GND | -1 | 66 | GND | A8 | GND | -1 |
| 17 | JX1_LVDS_2_P | U13 | IO_L3P_T0_DQS_PUDC_B_34 | 34 | 67 | JX1_LVDS_18_P | R16 | IO_L19P_T3_34 | 34 |
| 18 | JX1_LVDS_3_P | V12 | IO_L4P_T0_34 | 34 | 68 | JX1_LVDS_19_P | T17 | IO_L20P_T3_34 | 34 |
| 19 | JX1_LVDS_2_N | V13 | IO_L3N_T0_DQS_34 | 34 | 69 | JX1_LVDS_18_N | R17 | IO_L19N_T3_VREF_34 | 34 |
| 20 | JX1_LVDS_3_N | W13 | IO_L4N_T0_34 | 34 | 70 | JX1_LVDS_19_N | R18 | IO_L20N_T3_34 | 34 |
| 21 | GND | A8 | GND | -1 | 71 | GND | A8 | GND | -1 |
| 22 | GND | A8 | GND | -1 | 72 | GND | A8 | GND | -1 |
| 23 | JX1_LVDS_4_P | T14 | IO_L5P_T0_34 | 34 | 73 | JX1_LVDS_20_P | V17 | IO_L21P_T3_DQS_34 | 34 |
| 24 | JX1_LVDS_5_P | P14 | IO_L6P_T0_34 | 34 | 74 | JX1_LVDS_21_P | W18 | IO_L22P_T3_34 | 34 |
| 25 | JX1_LVDS_4_N | T15 | IO_L5N_T0_34 | 34 | 75 | JX1_LVDS_20_N | V18 | IO_L21N_T3_DQS_34 | 34 |
| 26 | JX1_LVDS_5_N | R14 | IO_L6N_T0_VREF_34 | 34 | 76 | JX1_LVDS_21_N | W19 | IO_L22N_T3_34 | 34 |
| 27 | GND | A8 | GND | -1 | 77 | GND | A8 | GND | -1 |
| 28 | GND | A8 | GND | -1 | 78 | VCCO_34 | N19 | VCCO_34 | 34 |
| 29 | JX1_LVDS_6_P | Y16 | IO_L7P_T1_34 | 34 | 79 | VCCO_34 | N19 | VCCO_34 | 34 |
| 30 | JX1_LVDS_7_P | W14 | IO_L8P_T1_34 | 34 | 80 | VCCO_34 | N19 | VCCO_34 | 34 |
| 31 | JX1_LVDS_6_N | Y17 | IO_L7N_T1_34 | 34 | 81 | JX1_LVDS_22_P | N17 | IO_L23P_T3_34 | 34 |
| 32 | JX1_LVDS_7_N | Y14 | IO_L8N_T1_34 | 34 | 82 | JX1_LVDS_23_P | P15 | IO_L24P_T3_34 | 34 |
| 33 | GND | A8 | GND | -1 | 83 | JX1_LVDS_22_N | P18 | IO_L23N_T3_34 | 34 |
| 34 | GND | A8 | GND | -1 | 84 | JX1_LVDS_23_N | P16 | IO_L24N_T3_34 | 34 |
| 35 | JX1_LVDS_8_P | T16 | IO_L9P_T1_DQS_34 | 34 | 85 | GND | A8 | GND | -1 |
| 36 | JX1_LVDS_9_P | V15 | IO_L10P_T1_34 | 34 | 86 | GND | A8 | GND | -1 |
| 37 | JX1_LVDS_8_N | U17 | IO_L9N_T1_DQS_34 | 34 | 87 | BANK13_LVDS_0_P | U7 | IO_L11P_T1_SRCC_13 | 13 |
| 38 | JX1_LVDS_9_N | W15 | IO_L10N_T1_34 | 34 | 88 | BANK13_LVDS_1_P | T9 | IO_L12P_T1_MRCC_13 | 13 |
| 39 | GND | A8 | GND | -1 | 89 | BANK13_LVDS_0_N | V7 | IO_L11N_T1_SRCC_13 | 13 |
| 40 | GND | A8 | GND | -1 | 90 | BANK13_LVDS_1_N | U10 | IO_L12N_T1_MRCC_13 | 13 |
| 41 | JX1_LVDS_10_P | U14 | IO_L11P_T1_SRCC_34 | 34 | 91 | BANK13_LVDS_2_P | V8 | IO_L15P_T2_DQS_13 | 13 |
| 42 | JX1_LVDS_11_P | U18 | IO_L12P_T1_MRCC_34 | 34 | 92 | BANK13_LVDS_3_P | T5 | IO_L19P_T3_13 | 13 |
| 43 | JX1_LVDS_10_N | U15 | IO_L11N_T1_SRCC_34 | 34 | 93 | BANK13_LVDS_2_N | W8 | IO_L15N_T2_DQS_13 | 13 |
| 44 | JX1_LVDS_11_N | U19 | IO_L12N_T1_MRCC_34 | 34 | 94 | BANK13_LVDS_3_N | U5 | IO_L19N_T3_VREF_13 | 13 |
| 45 | GND | A8 | GND | -1 | 95 | GND | A8 | GND | -1 |
| 46 | GND | A8 | GND | -1 | 96 | GND | A8 | GND | -1 |
| 47 | JX1_LVDS_12_P | N18 | IO_L13P_T2_MRCC_34 | 34 | 97 | NetJX1_97 | K9 | VP_0 | 0 |
| 48 | JX1_LVDS_13_P | N20 | IO_L14P_T2_SRCC_34 | 34 | 98 | NetJX1_98 | M9 | DXP_0 | -1 |
| 49 | JX1_LVDS_12_N | P19 | IO_L13N_T2_MRCC_34 | 34 | 99 | NetJX1_99 | L10 | VN_0 | 0 |
| 50 | JX1_LVDS_13_N | P20 | IO_L14N_T2_SRCC_34 | 34 | 100 | NetJX1_100 | M10 | DXN_0 | -1 |

Fonte: (AVNET, 2019)

Isto é necessário porque no arquivo de *constraints* como foi mostrado na Figura 36, os valores são aqueles da terceira coluna da memória do Zynq, como K12, J20 e assim por diante.

Figura 44 – Pinagem do MicroHeader JX2

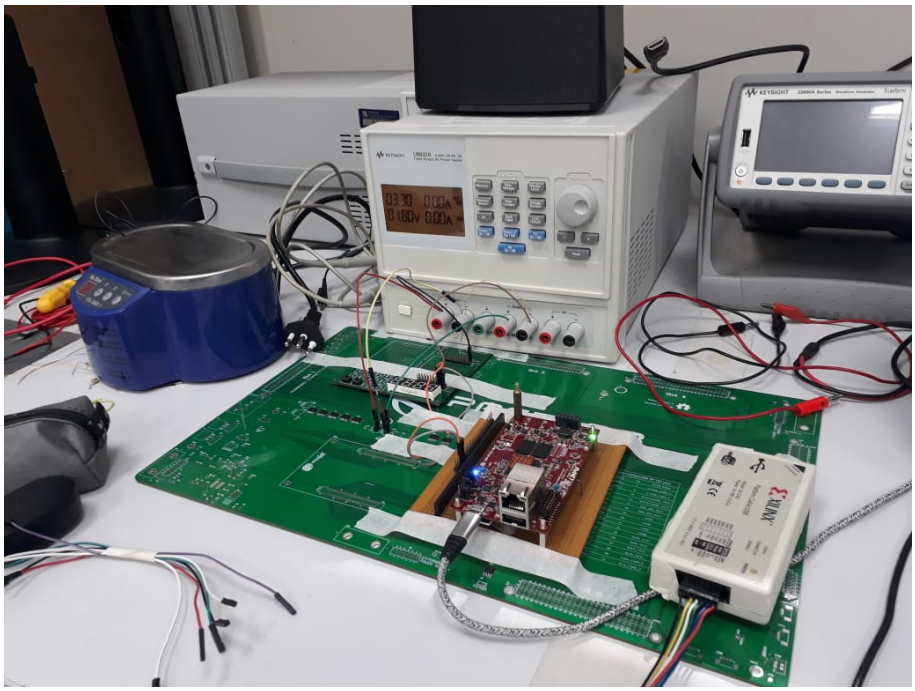
| JX2 Pin | MicroZed Net Name | Zynq Pin | Zynq Name | VCCO Bank | JX2 Pin | MicroZed Net Name | Zynq Pin | Zynq Name | VCCO Bank |
|---|---|---|---|---|---|---|---|---|---|
| 1 | PMOD_D0 | E8 | PS_MIO13_500 | -1 | 51 | GND | A8 | GND | -1 |
| 2 | PMOD_D1 | E9 | PS_MIO10_500 | -1 | 52 | GND | A8 | GND | -1 |
| 3 | PMOD_D2 | C6 | PS_MIO11_500 | -1 | 53 | JX2_LVDS_12_P | H16 | IO_L13P_T2_MRCC_35 | 35 |
| 4 | PMOD_D3 | D9 | PS_MIO12_500 | -1 | 54 | JX2_LVDS_13_P | J18 | IO_L14P_T2_AD4P_SRCC_35 | 35 |
| 5 | PMOD_D4 | E6 | PS_MIO0_500 | -1 | 55 | JX2_LVDS_12_N | H17 | IO_L13N_T2_MRCC_35 | 35 |
| 6 | PMOD_D5 | B5 | PS_MIO9_500 | -1 | 56 | JX2_LVDS_13_N | H18 | IO_L14N_T2_AD4N_SRCC_35 | 35 |
| 7 | PMOD_D6 | C5 | PS_MIO14_500 | -1 | 57 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 8 | PMOD_D7 | C8 | PS_MIO15_500 | -1 | 58 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 9 | NetJX2_9 | R10 | INIT_B_0 | -1 | 59 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 10 | NetJX2_10 | L6 | PROGRAM_B_0 | -1 | 60 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP |
| 11 | PG_MODULE | C7 | PS_POR_B_500 | -1 | 61 | JX2_LVDS_14_P | G17 | IO_L16P_T2_35 | 35 |
| 12 | VIN_HDR | #N/DISP | #N/DISP | #N/DISP | 62 | JX2_LVDS_15_P | F19 | IO_L15P_T2_DQS_AD12P_35 | 35 |
| 13 | JX2_SE_0 | G14 | IO_0_35 | 35 | 63 | JX2_LVDS_14_N | G18 | IO_L16N_T2_35 | 35 |
| 14 | JX2_SE_1 | J15 | IO_25_35 | 35 | 64 | JX2_LVDS_15_N | F20 | IO_L15N_T2_DQS_AD12N_35 | 35 |
| 15 | GND | A8 | GND | -1 | 65 | GND | A8 | GND | -1 |
| 16 | GND | A8 | GND | -1 | 66 | GND | A8 | GND | -1 |
| 17 | JX2_LVDS_0_P | C20 | IO_L1P_T0_AD0P_35 | 35 | 67 | JX2_LVDS_16_P | G19 | IO_L18P_T2_AD13P_35 | 35 |
| 18 | JX2_LVDS_1_P | B19 | IO_L2P_T0_AD8P_35 | 35 | 68 | JX2_LVDS_17_P | J20 | IO_L17P_T2_AD5P_35 | 35 |
| 19 | JX2_LVDS_0_N | B20 | IO_L1N_T0_AD0N_35 | 35 | 69 | JX2_LVDS_16_N | G20 | IO_L18N_T2_AD13N_35 | 35 |
| 20 | JX2_LVDS_1_N | A20 | IO_L2N_T0_AD8N_35 | 35 | 70 | JX2_LVDS_17_N | H20 | IO_L17N_T2_AD5N_35 | 35 |
| 21 | GND | A8 | GND | -1 | 71 | GND | A8 | GND | -1 |
| 22 | GND | A8 | GND | -1 | 72 | GND | A8 | GND | -1 |
| 23 | JX2_LVDS_2_P | E17 | IO_L3P_T0_DQS_AD1P_35 | 35 | 73 | JX2_LVDS_18_P | K14 | IO_L20P_T3_AD6P_35 | 35 |
| 24 | JX2_LVDS_3_P | D19 | IO_L4P_T0_35 | 35 | 74 | JX2_LVDS_19_P | H15 | IO_L19P_T3_35 | 35 |
| 25 | JX2_LVDS_2_N | D18 | IO_L3N_T0_DQS_AD1N_35 | 35 | 75 | JX2_LVDS_18_N | J14 | IO_L20N_T3_AD6N_35 | 35 |
| 26 | JX2_LVDS_3_N | D20 | IO_L4N_T0_35 | 35 | 76 | JX2_LVDS_19_N | G15 | IO_L19N_T3_VREF_35 | 35 |
| 27 | GND | A8 | GND | -1 | 77 | GND | A8 | GND | -1 |
| 28 | GND | A8 | GND | -1 | 78 | VCCO_35 | C19 | VCCO_35 | 35 |
| 29 | JX2_LVDS_4_P | E18 | IO_L5P_T0_AD9P_35 | 35 | 79 | VCCO_35 | C19 | VCCO_35 | 35 |
| 30 | JX2_LVDS_5_P | F16 | IO_L6P_T0_35 | 35 | 80 | VCCO_35 | C19 | VCCO_35 | 35 |
| 31 | JX2_LVDS_4_N | E19 | IO_L5N_T0_AD9N_35 | 35 | 81 | JX2_LVDS_20_P | N15 | IO_L21P_T3_DQS_AD14P_35 | 35 |
| 32 | JX2_LVDS_5_N | F17 | IO_L6N_T0_VREF_35 | 35 | 82 | JX2_LVDS_21_P | L14 | IO_L22P_T3_AD7P_35 | 35 |
| 33 | GND | A8 | GND | -1 | 83 | JX2_LVDS_20_N | N16 | IO_L21N_T3_DQS_AD14N_35 | 35 |
| 34 | GND | A8 | GND | -1 | 84 | JX2_LVDS_21_N | L15 | IO_L22N_T3_AD7N_35 | 35 |
| 35 | JX2_LVDS_6_P | L19 | IO_L9P_T1_DQS_AD3P_35 | 35 | 85 | GND | A8 | GND | -1 |
| 36 | JX2_LVDS_7_P | M19 | IO_L7P_T1_AD2P_35 | 35 | 86 | GND | A8 | GND | -1 |
| 37 | JX2_LVDS_6_N | L20 | IO_L9N_T1_DQS_AD3N_35 | 35 | 87 | JX2_LVDS_22_P | M14 | IO_L23P_T3_35 | 35 |
| 38 | JX2_LVDS_7_N | M20 | IO_L7N_T1_AD2N_35 | 35 | 88 | JX2_LVDS_23_P | K16 | IO_L24P_T3_AD15P_35 | 35 |
| 39 | GND | A8 | GND | -1 | 89 | JX2_LVDS_22_N | M15 | IO_L23N_T3_35 | 35 |
| 40 | GND | A8 | GND | -1 | 90 | JX2_LVDS_23_N | J16 | IO_L24N_T3_AD15N_35 | 35 |
| 41 | JX2_LVDS_8_P | M17 | IO_L8P_T1_AD10P_35 | 35 | 91 | GND | A8 | GND | -1 |
| 42 | JX2_LVDS_9_P | K19 | IO_L10P_T1_AD11P_35 | 35 | 92 | GND | A8 | GND | -1 |
| 43 | JX2_LVDS_8_N | M18 | IO_L8N_T1_AD10N_35 | 35 | 93 | BANK13_LVDS_4_P | V12 | IO_L20P_T3_13 | 13 |
| 44 | JX2_LVDS_9_N | J19 | IO_L10N_T1_AD11N_35 | 35 | 94 | BANK13_LVDS_5_P | V11 | IO_L21P_T3_DQS_13 | 13 |
| 45 | GND | A8 | GND | -1 | 95 | BANK13_LVDS_4_N | Y13 | IO_L20N_T3_13 | 13 |
| 46 | GND | A8 | GND | -1 | 96 | BANK13_LVDS_5_N | V10 | IO_L21N_T3_DQS_13 | 13 |
| 47 | JX2_LVDS_10_P | L16 | IO_L11P_T1_SRCC_35 | 35 | 97 | BANK13_LVDS_6_P | V6 | IO_L22P_T3_13 | 13 |
| 48 | JX2_LVDS_11_P | K17 | IO_L12P_T1_MRCC_35 | 35 | 98 | VCCO_13 | T8 | VCCO_13 | 13 |
| 49 | JX2_LVDS_10_N | L17 | IO_L11N_T1_SRCC_35 | 35 | 99 | BANK13_LVDS_6_N | W6 | IO_L22N_T3_13 | 13 |
| 50 | JX2_LVDS_11_N | K18 | IO_L12N_T1_MRCC_35 | 35 | 100 | BANK13_SE_0 | V5 | IO_L6N_T0_VREF_13 | 13 |

Fonte: (AVNET, 2019)

Inicialmente foi montada uma bancada com um conversor USB-UART da Xilinx, e conseguiu-se gerar uma plataforma e uma aplicação. A única forma encontrada de transferir os dados do Vitis é através do conversor Serial/UART, porém o

componente não reconhecia a placa no Xilinx SDK. Este conversor estava avariado (com o plastico de proteção saindo e talvez esteja danificado, o que impossibilitou a gravação da FPGA e testes do software através do Vitis. A versão gerada ficará disponível para futuros testes no GitHub do autor.

Figura 45 – Montagem da bancada para testes do $I^2C$



Fonte: o autor

Partiu-se então para a outra alternativa que é a gravação com o Petalinux. O hardware para os testes do $I^2C$ já estava configurado e no momento da geração do projeto, foram adicionadas bibliotecas auxiliares como o python3, smBus e i2c-tools, sendo que com este último, pode-se executar o comando *i2cdetect -y 1* que lista todos os endereços de dispositivos conectados aos canais $I^2C$ ativos. Executando todos os passos para criação do SO no Petalinux, conseguiu-se também uma versão estável e operante, o qual está atualmente no cartão SD da placa e também ficará disponível no Github do autor.

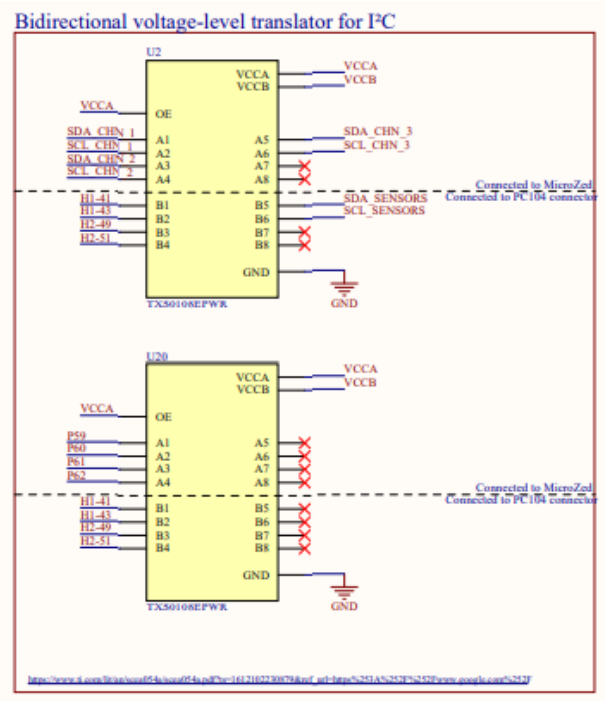
## 4.4 Testes dos transdutores de tensão

Este teste é feito alimentando-se os transdutores com as tensões de 1.8V em VCCA e 3.3V em VCCB. Conecta-se então um gerador de função do lado da menor tensão (onde é conectada a MicroZed) e o osciloscópio no lado B para a análise do sinal. Para um adequado comparativo, também pode-se conectar outro canal do osciloscópio no lado A para ter-se a proporção entre os níveis de tensão gerado e transmitido e uma análise da degradação ou atraso do sinal, causado por interferências

na placa e das características do componente TXS0108EPWR.

O principal a se testar aqui são os componentes que fazem a comunicação $I^2C$ (U2 e U20) conforme esquemático da Figura 46 que são responsáveis pelos sinais dos sensores de corrente, sensor de temperatura ambiente e canal entre os módulos e a MicroZed.

Figura 46 – Esquemático do transdutor de tensão para $I^2C$

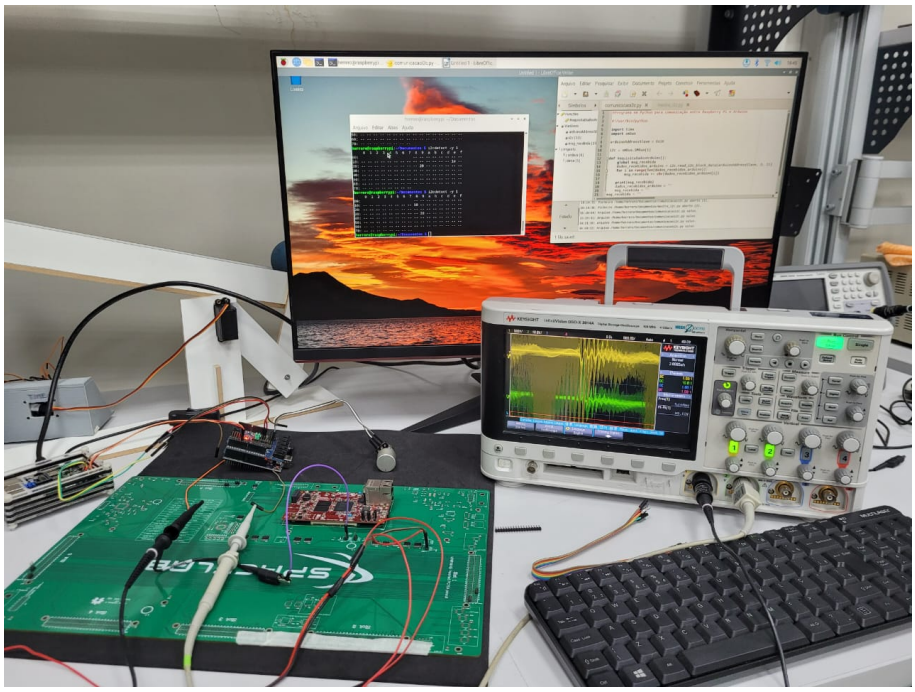



Fonte: https://github.com/spacelab-ufsc/flatsat-platform2/tree/dev_hardware

Para este teste, em uma outra placa (do mesmo lote da placa inicial), foram montados apenas um transdutor bidirecional com open-drain (U2) e um transdutor bidirecional comum (U15) que conta com uma comunicação serial (TX_2 e RX_2) e outras portas de uso geral. Esta escolha foi feita para se reduzir a probabilidade de erros durante a montagem, interferindo nos resultados e possivelmente gastando-se muito tempo para encontrá-los.

Os dispositivos Arduino e Raspberry Pi estavam se comunicando, com os endereços funcionando corretamente até que a fonte foi ligada para alimentar os transdutores. Neste momento, a comunicação parou e o programa que estava sendo executado no Raspberry Pi parou em função desta falha. Ao executar algumas vezes o comando *i2cdetect -y 1* no Raspberry Pi conforme a Figura 48, percebeu-se que endereços aleatórios estavam aparecendo como resposta muito provavelmente em função de ruídos por falta de aterramento ou filtro adequado. Por falta de tempo hábil, não conseguiu-se encontrar a fonte do distúrbio e futuras investigações serão necessárias.

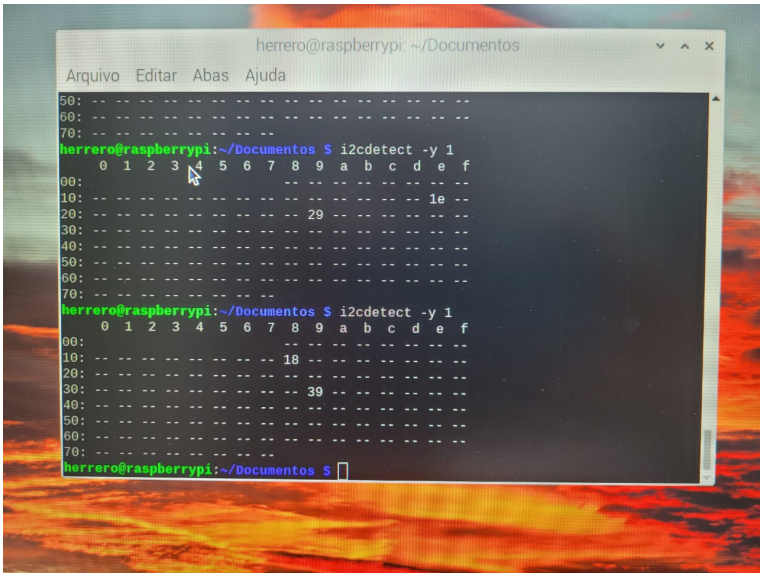
Figura 47 – Testes finais com Arduino, Raspberry Pi e MicroZed



Fonte: o autor

Mesmo com os transdutores desligados, ao se inserir as ponteiras do Osciloscópio no barramento $I^2C$ e buscando por dispositivos, nenhum endereço era encontrado, mostrando que o Osciloscópio também causa perturbações, impossibilitando que maiores testes possam ser realizados. Um desses momentos é como pode ser visto na Figura 47 onde, com o Osciloscópio conectado ao barramento, pode-se visualizar tal perturbação.

Figura 48 – Execução do comando *i2cdetect* múltiplas vezes



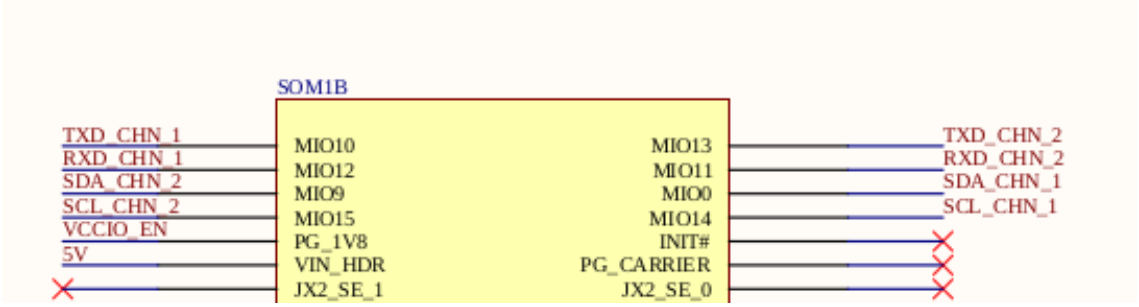

Fonte: o autor

Com isso não se pode validar totalmente os transdutores de tensão e eventuais correções ou acréscimo de dispositivos de filtragem sejam necessários. A implentação em FPGA também possibilita a inserção de filtros dentro do circuito, sendo uma também possível solução.

## 4.5 Necessidade de reprojeto da FlatSat2

Conforme dito na seção de *Configuração do sistema com um hardware personalizado*, um reprojeto será necessário caso todas as funcionalidades da MicroZed sejam implementadas, principalmente em relação as MIOs que seguem pelo microHeader. Conforme pode ser visto na Figura 49, as MIOs que estão disponíveis em tal conector são MIO 0 e de MIO 9 a MIO15.

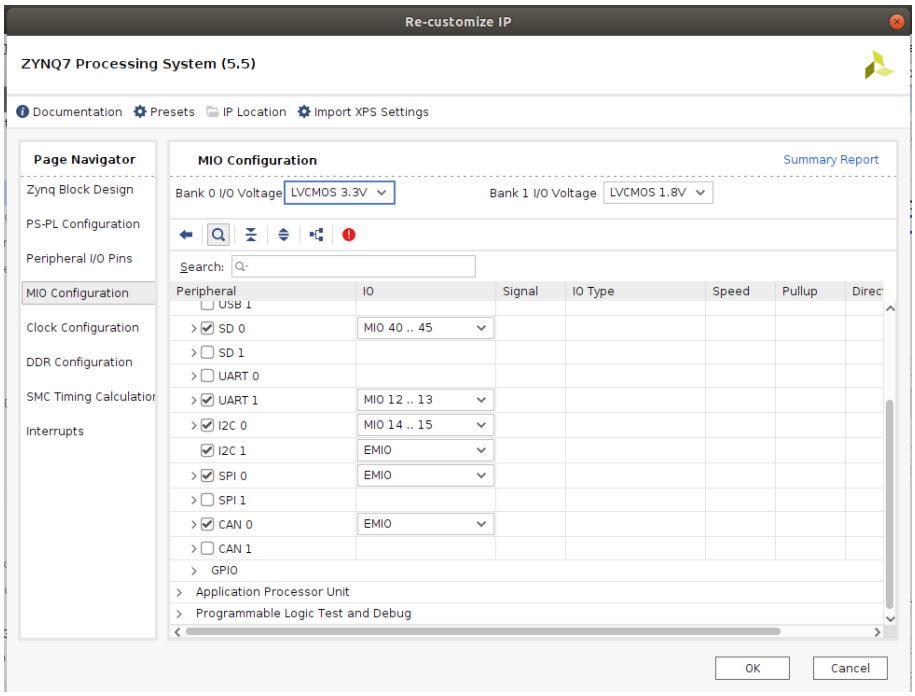
Figura 49 – Detalhe do esquemático de JX2




Fonte: o autor

Como qualquer dispositivo programável, algumas funcionalidades, principalmente no tocante a barramentos de comunicação, devem estar em pinos específicos. O pino definido para SCL_CHN_1 é o MIO 14 e o SDA_CHN_1 é MIO 0. Já para CHN2, foram definidos os pinos MIO 15 e MIO 9. Tal configuração não é permitida no Vivado, conforme pode ser visto na Figura 50, sendo somente permitidos para I2C_0 ser definido como, dentro do disponível em JX2, MIO 10 e 11 ou MIO 14 e 15, sendo o primeiro SCL e o segundo SDA. Da mesma forma, I2C_1 só pode ser definido em MIO 12 e 13, concorrendo para a escolha com o barramento UART_0.

Figura 50 – Configuração do PS no Vivado




Fonte: o autor

Desta forma, para os testes implementados aqui, optou-se por configurar I2C_0 em MIO 14 e 15, o barramento UART_0 em MIO 12 e 13 e a implementação do I2C_1 sendo reconfigurada para o PL. Apenas a saída da I2C_1 do PS é conectada ao PL, diferentemente do apresentado no diagrama onde há uma outra implementação de $I^2C$ totalmente no PL (chamado de I2C_3) e que pode ser visto na Figura 37. A configuração do I2C_1 foi feita no arquivo de *constraints* conforme visto na Figura 36. Da forma, conforme a numeração adotada no projeto da FlatSat2, o I2C_1 fica em H1-43 e H2-51, o I2C_2 fica em H1-41 e H2-49 e I2C_3 continua como antes, já que o mesmo não ia para o barramento PC104. Vale lembrar que no projeto I2C_1 corresponde ao I2C_0 da configuração do PS e assim sucessivamente.

# 5 CONCLUSÕES

Como o objetivo geral deste documento era verificar se eventuais falhas de projeto da FlatSat2 pudessem ter ocorrido e validar o uso desta placa para a missão atual e missões futuras, tal objetivo foi cumprido através de todos os testes implementados.

Com relação ao primeiro objetivo específico de teste das trilhas, os testes de continuidade e qualidade de sinal provaram que a mesma não apresenta falhas de projeto, somente sendo sugerida uma alteração no final do capítulo de resultados quando se quer utilizar as portas de IO do PS disponíveis nos microHeaders.

Com relação à montagem dos componentes, todos os que estavam disponíveis foram montados seguindo as normas da ECSS resumidas e a atual placa encontra-se nas dependências do LISHA em Joinville, disponível para futuros testes e continuação da montagem dos componentes. Com relação à rotina de montagem, foram descritos os principais pontos à serem observados e a forma como foram inseridos os componentes até o momento. Seguindo-se os passos descritos, pode-se conseguir o mesmo resultado positivo. Já com relação ao terceiro objetivo específico, de testes com dispositivos programáveis, foram implementados tanto testes com dispositivos mais simples como Arduino e Raspberry Pi quanto com a MicroZed e acredita-se que os resultados só não foram melhores em função da indisposição de alguns componentes que auxiliariam na filtragem de ruídos dos barramentos de comunicação. Na etapa de testes com a MicroZed, foi implementado o hardware com geração do arquivo de configuração XSA e a implementação no Vitis também foi bem suscedida com os arquivos estando disponíveis no GitHub. Só não foram possíveis demais testes em função da falta de comunicação entre a MicroZed e o Vitis através do conversor Serial/UART.

Com relação ao último objetivo específico dos testes dos transdutores de tensão, quando houve a alimentação, os resutados também foram positivos, gerando-se uma onda do lado dos módulos do FloripaSat (tensão máxima de 3.3V) e obteve-se um sinal semelhante do lado da MicroZed (tensão máxima de 1.8V). Demais testes necessitam ser realizados com a inserção dos filtros para uma correta interpretação dos protocolos existentes.

## 5.1 Trabalhos futuros

Como sugestão para trabalhos futuros, levando-se em consideração os contratempos enfrentados em termos de fornecimento de componentes em função de uma crise global no setor de fornecimento, sugere-se a aplicação de um passo intermediário, considerando-se a filosofia de modelos também apresentada neste documento, entre o modelo de engenharia (EM) e o modelo em placa universal (EBB) no desenvolvimento da FlatSat. Como o intuito deste trabalho não é focar no hardware em si e sim nos testes com este hardware, um modelo de engenharia mais simples ou onde se consegue trocar componentes escassos por componentes com a mesma funcionalidade e disponíveis no mercado, seria o ideal. A escolha dos tamanhos e posição dos componentes também deve ser observada. Como componentes entram e saem de linha, aqueles que tem pinagem idêntica, independente do fabricante é recomendada. Assim, na escassez de um, consegue-se continuar o projeto com os recursos disponíveis sem a necessidade da confeção de uma placa diferente. Consegue-se assim um desenvolvimento mais rápido da lógica sem a dependência da chegada de tais componentes. Pode-se implementar também uma pinagem entre os sensores e transdutores de tensão e uma outra forma de conectar a MicroZed à FlatSat2. Uma melhor escolha dos pinos e quantidade dos pontos de teste também é recomendada.

Devido também à dificuldade de aquisição e importação dos componentes em função da pandemia do COVID-19, o recomendado é que a placa já venha com os componentes montados, já que não se exige a montagem em *cleanroom* (já que a mesma não será lançada ao espaço). Isto porque não contamos ainda com uma forma fácil e adequada à complexidade exigida deste projeto. Algumas empresas chinesas que realizam a confeção da placa, tem facilidade em adquirir esses componentes e ela já vem testada.

Para um futuro trabalho, pode-se migrar para outra placa (mas continuando na linha Zynq-7000) que se tenha mais suporte, exemplos, literatura e com uma conexão física mais simples (pois levou-se muito tempo para adquirir devido à escassez atual de componentes) seja uma melhor opção. Devido a facilidade em se reprogramar um SoC Zynq-7000, mesmo que os protocolos de comunicação mudem para CAN ou outro protocolo, esta reconfiguração leva pouco tempo no Vivado e não é necessária uma alteração física desde que as mesmas estejam conectadas ao PL. Caso o nível de tensão mude para 1.8 V (LVDS), a placa poderá ser utilizada, apenas fazendo um bypass dos transdutores de tensão. A placa também permite que algumas portas trabalhem com tensões superiores como 2.5V e 3.3V, configuração esta que pode ser feita tanto no bloco do PS como através do arquivo de *constraints*.

# REFERÊNCIAS

AVNET. **MicroZed Getting Started Guide Version 1.2**. 2019. Disponível em: <https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/microzed/microzed-board-family>. Acesso em: 16 de junho de 2019.

CAPPELLETTI, C.; BATTISTINI, S.; MALPHRUS, B. K. **CUBESAT HANDBOOK: From Mission Design to Operations**. [S.l.]: Academic Press, 2020.

CROCKETT, L. H. et al. **The Zynq Book**: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC. [S.l.]: Strathclyde Academic Media, 2014.

CUBESAT.ORG. **CubeSat Standard Organization**. 2017. Último acesso 16 de Julho de 2022. Disponível em: <http://www.cubesat.org/>.

ECSS. **European Cooperation for Space Standardization**. 2021. Disponível em: <https://ecss.nl/standards/>. Acesso em: 12 de julho de 2022.

GABER, K.; EL_MASHADE, M. B.; AZIZ, G. A. A. Hardware-in-the-loop real-time validation of micro-satellite attitude control. **Computers & Electrical Engineering**, Science Direct, v. 85, 2020.

GOMSPACE P110 Solar Panels. 2022. Último acesso 16 de Julho de 2022. Disponível em: <https://gomspace.com/shop/subsystems/power/p110-solar-panel.aspx>.

INA219AID Bi-Directional Power/Current Monitor with $I^2C$ Interface. 2022. Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/249610/TI/INA219AID.html>. Acesso em: 04 de dezembro de 2022.

JúNIOR, K. R. G. **Fluxo de apoio à concepção de hardware para o segmento espacial visando melhoria de confiabilidade de missões cubesat**. 2021. Disponível em: <https://repositorio.ufsc.br/handle/123456789/227056>. Acesso em: 25 de junho de 2022.

TRINDADE LEMOS Émerson da. **Análise e reformulação do arcabouço de interconexão do FloripaSat visando mitigação de falhas nas camadas física e de enlace de dados**. 2021. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/229813/PEEL2042-D.pdf>. Acesso em: 4 de dezembro de 2022.

MAASS, W. **Neural Networks - Networks of spiking neurons**: The third generation of neural network models(Pages 1659-1671. [S.l.]: Elsevier, 1997.

MARCELINO, G.; AZEREDO, Y. C. de. **EPS 2.0 Documentation (SpaceLab)**. 2020. Último acesso 16 de Julho de 2022. Disponível em: <https://github.com/spacelab-ufsc/eps2>.

PC/104 EMBEDDED CONSORTIUM. **PC/104 Specification - Version 2.6**. 2008. Disponível em: <https://pc104.org/wp-content/uploads/2015/02/PC104_Spec_v2_6.pdf>. Acesso em: 6 de junho de 2022.

SILVA, J. P. S. da. Processamento de dados em zynq apsoc. Universidade de Aveiro, 2014.

STATISTA, Market and Consumer Data. 2022. Disponível em: <https://www.statista.com/statistics/1131983/arm-based-chip-unit-shipments-as-reported-by-licensees-worldwide/>. Acesso em: 28 de novembro de 2022.

TAYLOR, A. **The 'Model Philosophy' Used by Space Engineering Companies**. 2013. Último acesso 16 de Julho de 2022. Disponível em: <https://www.eetimes.com/the-model-philosophy-used-by-space-engineering-companies/>.

UFSC-SPACELAB. **FloripaSat2**. 2020. Disponível em: <https://github.com/spacelab-ufsc/>. Acesso em: 25 de outubro de 2022.

UNIVERSIDADE DE TSUKUBA. **ITF-1**. 2014. Disponível em: <https://yui.kz.tsukuba.ac.jp/>. Acesso em: 6 de junho de 2022.

XILINX. **Zynq-7000 SoC Data Sheet: Overview(DS190)**. 2019. Disponível em: <https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf>. Acesso em: 6 de junho de 2019.

# APENDICE A

## .1 Raspberry Pi requisitando dados I2C

```
#!/usr/bin/python

import time, smbus

slaveAddress = 0x18
#este endereÃğo deve ser o mesmo que foi colocado no arduino

i2c = smbus.SMBus(1) #cria o canal para comunicacao

def RequisitaDadosArduino():
    global msg_recebida
    dados_recebidos_Arduino = i2c.read_i2c_block_data(slaveAddress, 0,11)
 #endereÃğo, 0 para ler e 1 para escrever e tamanho dos dados (12 bits)
    for i in range(len(dados_recebidos_Arduino)):
        msg_recebida += chr(dados_recebidos_Arduino[i])

    print(msg_recebida)
    dados_recebidos_Arduino =""
    msg_recebida = ""

msg_recebida = ""

while 1:
    RequisitaDadosArduino()
    time.sleep(100)
```

# APENDICE B

## .2 Arduino Due como escravo I2C

```
#include <Wire.h> //biblioteca necessaria para I2C
char str[15];     //Por padrao, SCL ÃĽ A4 e SDA A5
int valor;
void setup()
{
  Serial.begin(9600);
  //Utilizado no monitor serial
  Wire.begin(0x18);
  //endereÃğo dado ao Arduino no I2C
  Wire.onRequest(requestEvent);
  //Arduino sÃş responde quando requisitado
}
void requestEvent()
{
  valor = analogRead(A0);
  //Conectar o pino do meio do potenciometro em A0
  //e os outros em VCC e GND
  Serial.println("Requisicao recebida!");
  //Mostra na serial se houve solicitacao
  sprintf(str, "Potenciometro em %4dn", valor);
  //envia ao I2C mestre
  Wire.write(str);
}
void loop()
{
  //como o escravo I2C trabalha por requisiÃğao,
  //nÃčo hÃą cÃşdigo no loop
  delay(10);
}
```

# APENDICE C

## .3 Programa para MicroZed de leitura de temperatura pelo I2C

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf"
#include "xiic"

#define IIC_DEV_ID XPAR_IIC_0_DEVICE_ID
#define IIC_TEMP_ADDR 0x24 //o endereÃğo real Ãľ 48 mas precisa ser dividido por 2
#define BUFFER_SIZE 6
#define WHOAMI  0xA0

XIic iic;
u8 SendBuffer[2];
u8 RecvBuffer[2];
u16 result;
float temp;

int main(int argc, char **argv)
{
 XIic_Config *iic_conf;

 //init_platform();
 print("Rodando a aplicaÃğÃčo de leitura de temperatura pelo I2C");

 iic_conf = XIic_LookupConfig(IIC_DEV_ID);
 XIic_CfgInitialize(&iic, iic_conf, iic_conf->BaseAddress);

 SendBuffer[0]=0x01;
 XIic_Send(iic.BaseAddress, IIC_TEMP_ADDR, (u8 *)&SendBuffer, 1, XIIC_REPEATED_START);
 XIic_Recv(iic.BaseAddress, IIC_TEMP_ADDR, (u8 *)&RecvBuffer, 1, XIIC_STOP);

 if(RecvBuffer[0]==WHOAMI){
  printf("Sensor de temperatura detectado");
 }
 else{
```

```
35   printf("Sensor de temperatura nÃčo detectado");
36   return 0;
37   }
38
39
40  SendBuffer[0]=0x04;
41  SendBuffer[1]=0x0C;
42  XIic_Send(iic.BaseAddress, IIC_TEMP_ADDR, (u8 *)&SendBuffer, 1,
      XIIC_STOP);
43
44  SendBuffer[0]=0x04;
45  XIic_Send(iic.BaseAddress, IIC_TEMP_ADDR, (u8 *)&SendBuffer, 1,
      XIIC_REPEATED_START);
46  XIic_Recv(iic.BaseAddress, IIC_TEMP_ADDR, (u8 *)&RecvBuffer, 1,
      XIIC_STOP);
47
48  while(1){
49   SendBuffer[0]=0x06;
50   XIic_Send(iic.BaseAddress, IIC_TEMP_ADDR, (u8 *)&SendBuffer, 1,
      XIIC_REPEATED_START);
51   XIic_Recv(iic.BaseAddress, IIC_TEMP_ADDR, (u8 *)&RecvBuffer, 1,
      XIIC_STOP);
52
53   result=RecvBuffer[1] << 8 | RecvBuffer[0];
54   temp = (float)result/100;
55   printf("A temperatura Ãľ %f \n\r", temp);
56   usleep(1000000);
57   }
58  //cleanup_platform();
59
60  return 0;
61 }
```

# ANEXO A

## .1 Datasheet do transdutor para I2C - TXS0108EPWR

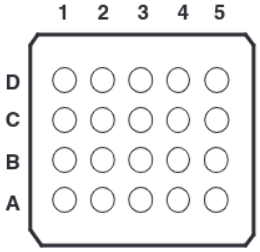
# TXS0108E
# 8-BIT BIDIRECTIONAL VOLTAGE-LEVEL TRANSLATOR FOR OPEN-DRAIN AND PUSH-PULL APPLICATIONS

SCES642–DECEMBER 2007

## FEATURES

- **1.2 V to 3.6 V on A Port and 1.65 V to 5.5 V on B Port ($V_{CCA} \leq V_{CCB}$)**
- **$V_{CC}$ Isolation Feature – If Either $V_{CC}$ Input Is at GND, All Outputs Are in the High-Impedance State**
- **Latch-Up Performance Exceeds 100 mA Per JESD 78, Class II**
- **ESD Protection Exceeds JESD 22 (A Port)**
  - **2000-V Human-Body Model (A114-B)**
  - **150-V Machine Model (A115-A)**
  - **1000-V Charged-Device Model (C101)**
- **IEC 61000-4-2 ESD (B Port)**
  - **±8-kV Contact Discharge**
  - **±6-kV Air-Gap Discharge**

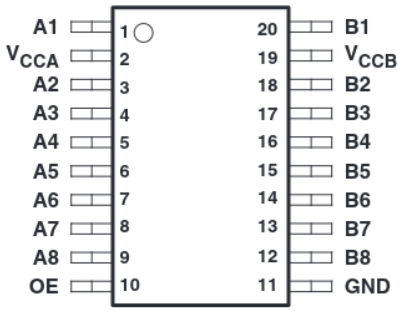
**ZXY PACKAGE**
**(BOTTOM VIEW)**



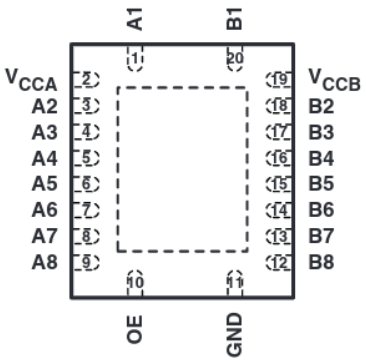

**TERMINAL ASSIGNMENTS**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **D** | $V_{CCB}$ | B2 | B4 | B6 | B8 |
| **C** | B1 | B3 | B5 | B7 | GND |
| **B** | A1 | A3 | A5 | A7 | OE |
| **A** | $V_{CCA}$ | A2 | A4 | A6 | A8 |

**PW PACKAGE**
**(TOP VIEW)**




**RGY PACKAGE**
**(TOP VIEW)**




## DESCRIPTION/ORDERING INFORMATION

This 8-bit noninverting translator uses two separate configurable power-supply rails. The A port is designed to track $V_{CCA}$. $V_{CCA}$ accepts any supply voltage from 1.2 V to 3.6 V. The B port is designed to track $V_{CCB}$. $V_{CCB}$ accepts any supply voltage from 1.65 V to 5.5 V. This allows for low-voltage bidirectional translation between any of the 1.2-V, 1.5-V, 1.8-V, 2.5-V, 3.3-V, and 5-V voltage nodes.

When the output-enable (OE) input is low, all outputs are placed in the high-impedance state.

To ensure the high-impedance state during power up or power down, OE should be tied to GND through a pulldown resistor; the minimum value of the resistor is determined by the current-sourcing capability of the driver.