UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FELIPE RUBBO TRAMONTINA

Implementação de um Receptor ARINC 429 utilizando noções do Processo de Desenvolvimento de Hardware para a Aviação DO-254

Monografia apresentada como requisito parcial para a obtenção do grau de Engenheiro Eletricista

Orientador: Prof. Dr. Raphael Martins Brum

Porto Alegre
2021

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões
Vice-Reitora: Prof[a].Patricia Pranke
Pró-Reitora de Graduação: Prof[a]. Cíntia Inês Boll
Diretora da Escola de Engenharia: Prof[a]. Carla Schwengber ten Caten
Coordenador do Curso de Engenharia Elétrica: Prof. Raphael Martins Brum
Bibliotecária-chefe da Escola de Engenharia: Rosane Beatriz Allegretti Borges

O metal mais forte é forjado no fogo mais quente.

— DESCONHECIDO

AGRADECIMENTOS

Agradeço a Deus pela oportunidade de existir. Agradeço aos meus pais pela educação e sustento. Agradeço ao meu irmão pelo apoio. Também agradeço ao meu orientador, Raphael Martins Brum, pela orientação.

RESUMO

Esse trabalho tem como objetivo o desenvolvimento da lógica programável de um receptor ARINC 429 utilizando noções do processo de desenvolvimento de *hardware* para a aviação DO-254. Por referenciar-se a sistemas e normas utilizados na aviação, o texto apresenta inicialmente conceitos básicos de arquitetura de sistemas eletrônicos presentes na aviação (aviônica). Em seguida, por tratar-se de *hardware* embarcado em aeronaves e que pode ser crítico, isto é, sua falha pode ocasionar uma falha catastrófica na aeronave, o texto apresenta uma metodologia de desenvolvimento de *hardware* utilizada na aviação que visa certificar o sistema desenvolvido como seguro, a norma DO-254. Após isso, a interface ARINC 429 é detalhada ao leitor. Então, no capítulo de desenvolvimento, é apresentado o fluxo de atividades que foram executadas para o projeto e verificação do receptor ARINC 429. Esse fluxo é baseado na DO-254, que por sua vez é baseada na criação de requisitos de *design* que devem ser implementados em lógica. Para verificar o sistema, simulações foram feitas com o intuito de provar que os requisitos estão de fato implementados e que o sistema funciona como foi especificado nos requisitos. Por fim, concluiu-se que a metodologia utilizada para o desenvolvimento do receptor ARINC 429 seguiu de fato noções da DO-254. Também conclui-se que, embora os requisitos do receptor tenham sido provados que estão implementados em lógica através das simulações, para de fato provar a funcionalidade do sistema na prática, é necessário que realize-se testes com os sistemas que fazem interface com o receptor ARINC 429 desenvolvido.

Palavras-chave: Aviônica, ARINC 429, DO-254, Hardware, Requisitos, VHDL.

Implementation of an ARINC 429 Receiver using notions of the Hardware Development Process for the Aviation DO-254

ABSTRACT

This work has as objective the development of the programmable logic of an ARINC 429 receiver using notions of the hardware development process for the aviation DO-254. As it refers to systems and standards used in aviation, the text initially presents basic concepts of electronic systems architecture present in aviation (avionics). Then, because it is hardware embedded in aircraft and which can be critical, that is, its failure can cause a catastrophic failure in the aircraft, the text presents a hardware development methodology used in aviation that aims to certify the developed system as safe, the DO-254 standard. After that, the ARINC 429 interface is detailed to the reader. Then, in the development chapter, the flow of activities that were performed for the design and verification of the ARINC 429 receiver is presented. This flow is based on DO-254, which is based on creating design requirements that must be implemented in logic. To verify the system, simulations were carried out in order to prove that the requirements are in fact implemented and that the system works as specified in the requirements. In conclusion, this methodology used for the development of the ARINC 429 receptor actually followed notions of DO-254. It is also concluded that, although the receiver requirements have been proven that they are implemented in logic through simulations, to actually prove the functionality of the system in practice, it is necessary to carry out tests with the systems that interface with the developed ARINC 429 receiver.

Keywords: Avionics, ARINC 429, DO-254, Hardware, Requirements, VHDL.

LISTA DE ABREVIATURAS E SIGLAS

AEE	Airlines Electronic Engineering Committee
ANAC	Agência Nacional de Aviação Civil
ARINC	Aeronautical Radio Incorporated
ASIC	Application-Specific Integrated Circuit
BCD	Binary-coded Decimal
BNR	Binary
BRZ	Bipolar Return to Zero
CPU	Central Processing Unit
DAL	Design Assurance Level
DME	Distance measuring equipment
EASA	European Union Aviation Safety Agency
EUROCAE	European Organization for Civil Aviation Equipment
FAA	Federal Aviation Administration
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
HDL	Hardware Description language
IF	Interface
LRU	Line-replaceable Unit
PLD	Programmable Logic Device
PHAC	Plan for hardware Aspects of Certification
RBV	Requirements-Based Verification
RF	Radiofrequência
RTCA	Radio Technical Commission for Aeronautics
RTL	Register Transfer Level

SDI	Source/Destination Identifier
SRU	Shop-replaceable Unit
SSM	Sign/Status Matrix
UART	Universal Asynchronous Receiver/Transmitter
UVM	Universal Verification Methodology
VHDL	Very High Speed Integrated Circuit hardware Description Language

LISTA DE FIGURAS

Figura 1.1 Exemplo de LRU: Avionics Network Computing (ANC) Platform 13

Figura 2.1 Design Assurance Levels de acordo com a DO-254 15
Figura 2.2 DO-254 Flow 16
Figura 2.3 Comunicação via ARINC 429 entre LRUs 21
Figura 2.4 Modulação RZ Bipolar 22
Figura 2.5 Protocolo ARINC 429 23
Figura 2.6 Campos da Tabela de Labels da norma ARINC 429 24
Figura 2.7 Label 201 24
Figura 2.8 Anexo 6-1-1 da norma 25
Figura 2.9 Trecho da tabela de dados BCD da norma 25

Figura 3.1 Diagrama de blocos - Etapa de Planejamento 30
Figura 3.2 Diagrama de blocos - Etapa de Captura de Requisitos 30
Figura 3.3 Diagrama de blocos - Etapa de Design Conceitual 34
Figura 3.4 ARINC 429 - Digital 35
Figura 3.5 Estratégia de CDR 36
Figura 3.6 Máquina de estados do bloco de CDR 37
Figura 3.7 Design Conceitual - Clock and Data Recovery 38
Figura 3.8 Design Conceitual - Parity Checker 39
Figura 3.9 Design Conceitual - FIFO 39
Figura 3.10 Máquina de estados do bloco de CPU IF 41
Figura 3.11 Design Conceitual - CPU IF 41
Figura 3.12 Máquina de estados do bloco de controle 43
Figura 3.13 Design Conceitual - Control Unit 43
Figura 3.14 Design Conceitual - Topo 44

Figura 4.1 Simulação - CDR 48
Figura 4.2 Detalhe da saída do bloco CDR 48
Figura 4.3 Simulação do bloco Parity Checker 51
Figura 4.4 Simulação do bloco FIFO 53
Figura 4.5 Simulação do bloco CPU IF 56
Figura 4.6 Simulação dos blocos integrados 57
Figura 4.7 Simulação dos blocos integrados com ênfase nos sinais de controle 57

Figura 5.1 Tabela ISO Alphabet Number 5 61

LISTA DE TABELAS

Tabela 2.1 Tabela de tensões na saída do transmissor sem carga....................................22
Tabela 2.2 Tensões na entrada do receptor sem considerar ruído. 22
Tabela 2.3 Tensões na entrada do receptor considerando ruído. 22
Tabela 2.4 Tabela SDI para instalação multi-sistema. 26
Tabela 2.5 Tabela mostrando a distribuição do dado. 26
Tabela 2.6 BNR SSM: bits 30 e 31 27
Tabela 2.7 BNR SSM: bit 29...27
Tabela 2.8 BCD SSM: bits 30 e 31 28

Tabela 3.1 Exemplos de paridade ímpar para uma palavra de 5 bits 38

Tabela 4.1 Resultado dos testes do bloco CDR..48
Tabela 4.2 Resultado dos testes do bloco Parity Checker 51
Tabela 4.3 Resultado dos testes do bloco FIFO 53
Tabela 4.4 Resultado dos testes do bloco CPU IF 55

SUMÁRIO

1 INTRODUÇÃO **12**
2 REVISÃO BIBLIOGRÁFICA **14**
2.1 DO-254 .... **14**
2.2 Verificação e Validação em Aviônica **18**
2.2.1 Requirements-Based Verification (RBV) 19
2.3 ARINC 429 .... **20**
2.3.1 Aspectos Elétricos 20
2.3.1.1 Características Gerais 20
2.3.2 Modulação 21
2.3.3 Níveis de Tensões 22
2.3.4 Aspectos Lógicos 23
2.3.4.1 Formato da palavra 23
2.3.5 Bit Rate 28
2.3.6 Sincronização e Clocking 28
3 DESENVOLVIMENTO **29**
3.1 Planejamento **29**
3.2 Captura de Requisitos **30**
3.2.1 Requisitos - Clock and Data Recovery 31
3.2.2 Requisitos - Paridade 31
3.2.3 Requisitos - FIFO 32
3.2.4 Requisitos - CPU IF 33
3.3 Design Conceitual **34**
3.3.1 Clock and Data Recovery 34
3.3.2 Parity Checker 38
3.3.3 FIFO 39
3.3.4 CPU IF 40
3.3.5 Control Unit 42
3.3.6 Diagrama Geral 44
3.4 Design Detalhado **45**
3.5 Implementação **45**
3.6 Transição para Produção **45**
4 RESULTADOS **46**
4.1 CDR **46**
4.2 Parity Checker **49**
4.3 FIFO **51**
4.4 CPU IF **53**
4.5 Integração **56**
4.6 Discussões **57**
5 CONCLUSÃO **59**
ANEXO A - TABELA ISO ALPHABET NUMBER 5 .... **61**
APÊNDICE A - CABOS ARINC 429 .... **62**
REFERÊNCIAS **63**

1 INTRODUÇÃO

Pela questão de segurança ou pela pequena margem à falhas, sistemas eletrônicos na aviação (aviônica) sempre estiveram na vanguarda da tecnologia. Radares, sistemas de processamento de vídeo, sensores, rádios, entre outros, fazem parte desse contexto. Esses sistemas eletrônicos são agrupados e distribuídos na aeronave através de *Line Replaceable Units* (LRUs). LRUs são módulos, normalmente caixas, que agrupam sistemas distintos da aeronave que podem ser removidos e colocados rapidamente a nível de campo. Por exemplo, uma LRU que contém o sistema de rádio da aeronave conterá todo o sistema eletrônico responsável pela comunicação do piloto com a torre de comando, dentro de uma caixa. Cada LRU é formada por *Shop-Replaceable Units* (SRUs). SRUs são componentes modulares das LRUs. No exemplo da LRU do rádio dado, as placas de circuito impresso de radiofrequência e de amplificação seriam SRUs. Uma aeronave é composta por várias LRUs. Essas LRUs precisam conversar entre si através de interfaces de comunicação. A comunicação entre LRUs é vital para o desempenho correto dos sistemas da aeronave; logo, é uma necessidade que sempre esteve presente. Com essa necessidade crescendo, em 1979, tendo em vista a criação de três novas aeronaves (*AirBus* A310, *Boeing* B-757 e B-767) por duas das grandes empresas da época, *Airbus* e *Boeing*, a *Airlines Electronic Engineering Committee* (AEEC), comitê de engenharia responsável pelo desenvolvimento de normas de sistemas usados na aviação, desenvolveu o padrão de interface de comunicação entre LRUs, *Mark33 Digital Information Transfer System*, publicado pela *Aeronautical Radio Incorporated* (ARINC), um dos principais provedores de soluções para a indústria de defesa e aeroespacial na época e que detém a AEEC, como padrão ARINC 429. Esse padrão é utilizado até hoje por apresentar intercambiabilidade e interoperabilidade entre LRUs. Devido a esses fatores, a interface ARINC 429 foi escolhida como contexto do projeto.

O objetivo do trabalho consiste em implementar a parte lógica de um receptor ARINC 429. Parte lógica, nesse contexto, refere-se a um *design* digital desenvolvido em linguagem de descrição de *hardware* capaz de realizar o recebimento de dados a fim de enviá-los a um dispositivo capaz de processá-los. Como trata-se de um *design* que engloba aviônica, o trabalho será baseado em uma abordagem simplificada da norma DO-254 (RTCA; EUROCAE, 2000). A norma DO-254, desenvolvida pela *Radio Technical Commission for Aeronautics* (RTCA) e a European Organization for Civil Aviation Equipment (EUROCAE), organizações sem fins lucrativos que desenvolvem guias téc-

Figura 1.1: Exemplo de LRU: Avionics Network Computing (ANC) Platform

Fonte: Leonardo Aerospace (2021).

nicos para entidades governamentais regulamentadoras e indústrias, define um processo de desenvolvimento de *hardware* para a aviação que busca certificar um *hardware* como seguro. Essa norma será vista em detalhes ao longo do trabalho. Esse trabalho tem um intuito puramente acadêmico e não busca certificar o *design* apresentado para ser usado em uma aeronave comercial.

Dado o contexto, pode-se resumir os objetivos principais do trabalho como:

- Desenvolver o código de um receptor ARINC 429 utlizando uma linguagem de descrição de *hardware*. O objetivo é ter apenas o código e não gravar o *design* de fato em um *Field Programmable Gate Array* (FPGA).
- Utilizar noções do processo de desenvolvimento de *hardware* para a aviação DO-254 priorizando a parte de *design*. Como será explicado ao longo do texto, a parte de verificação também será feita, porém seu escopo será reduzido.
- Fornecer conhecimento de aviônica à comunidade.

Esse trabalho está estruturado da seguinte forma: revisão bibliográfica, desenvolvimento, resultados e conclusão. No capítulo de revisão bibliográfica são apresentados os conceitos referentes às normas ARINC 429 e DO-254. Já no capítulo de desenvolvimento é apresentado o projeto do receptor ARINC 429. No capítulo de resultados são exibidos os testes realizados no *design* desenvolvido. Por fim, no capítulo de conclusão é tratado o desfecho do projeto.

2 REVISÃO BIBLIOGRÁFICA

Esse capítulo tem como objetivo fornecer a revisão bibliográfica básica a respeito do tema em questão. Como o assunto em questão engloba aviônica, é importante detalhar a respeito do processo de desenvolvimento de *hardware* na aviação. Esse processo é diferente de um processo de desenvolvimento de *hardware* de sistemas comuns do dia a dia, pois se baseia em normas específicas para aviação. Uma dessas normas, reconhecida por diversas entidades certificadoras, como a *Federal Aviation Administration* (FAA), a DO-254, é amplamente utilizada na aviação civil. Nesse capítulo será apresentado uma explicação da DO-254. Uma vez entendido o processo de desenvolvimento de *hardware* para aviação, a interface ARINC 429 será explicada em mais detalhes.

2.1 DO-254

Com o advento de sistemas eletrônicos em aeronaves comandando operações cada vez mais críticas, a exigência de segurança sobre aviônica aumentou consideravelmente. Cada vez mais, *hardware* tomava conta de funções críticas da aeronave, como controle da aeronave, monitoramento de combustível e comunicação. Dado o contexto, era necessário um método de desenvolvimento diferenciado que pudesse assegurar que o equipamento desenvolvido era seguro. Desse modo, a RTCA e EUROCAE publicaram a norma *DO 254 - Design Assurance Guidance for Airborne Electronic Hardware* (RTCA; EUROCAE, 2000).

DO-254 é uma norma que estabelece um processo de desenvolvimento de *hardware* orientado a requisitos que busca certificar o *hardware* como seguro. O objetivo da DO-254 é fornecer a garantia de que o sistema desenvolvido atende todas as funções pretendidas sob todas as condições operacionais previsíveis. Para tal, a DO-254 classifica diferentes partes do *hardware* (placas de circuito impresso, FPGAs) da aeronave de acordo com seu nível de criticidade, os chamados *Design Assurance Levels* (DALs). Cada nível recebe uma probabilidade de ocorrência. O grau mais alto de criticidade, quando a falha no *hardware* impede a aeronave de continuar voando com segurança ou de aterrizar em segurança, recebe o nível DAL-A (NATIONAL INSTRUMENTS, 2021). Em outras palavras, a falha em um *hardware* DAL-A pode gerar a queda da aeronave, ou seja, uma falha catastrófica. Uma falha em um *hardware* DAL-A deve ter uma probabilidade de ocorrência de menos de 1×10^{-9}. Um exemplo de *hardware* DAL-A é o computador de

controle da aeronave. Já o grau DAL B, a probabilidade de ocorrer mortes ou queda é menor, mas ainda assim existe, logo permite uma probabilidade um pouco maior de falha, na faixa de 1×10^{-7}. Para o DAL C, a falha do *hardware* ocasionaria ferimentos ou estresse. A falha nível DAL D causaria apenas desconforto e falha nível DAL E não causaria problemas. A figura 2.1 ilustra esse contexto. Os DALs impactam diretamente em testes, requisitos e custo de cada *hardware*, sendo assim, uma parte vital do desenvolvimento.

Figura 2.1: Design Assurance Levels de acordo com a DO-254

Design Assurance Level (DAL)	Description	Target System Failure Rate	Example System
Level A (Catastrophic)	Failure causes crash, deaths	<1 x 10-9 chance of failure/flight-hr	Flight controls
Level B (Hazardous)	Failure may cause crash, deaths	<1 x 10-7 chance of failure/flight-hr	Braking systems
Level C (Major)	Failure may cause stress, injuries	<1 x 10-5 chance of failure/flight-hr	Backup systems
Level D (Minor)	Failure may cause inconvenience	No safety metric	Ground navigation systems
Level E (No effect)	No safety effect on passengers/crew	No safety metric	Passenger entertainment

Fonte: Cadence (2019).

Como a DO-254 é um padrão baseado em requisitos, é importante definir o conceito de requisito. Requisito é uma frase que traz uma definição no projeto. Essas definições devem ser provadas a fim de garantir que o *design* em questão é seguro. Para exemplificar esse conceito, uma lista de três requisitos para uma LRU hipotética é apresentada abaixo:

- Requisito 1: A LRU deve possuir uma interface ARINC 429.
- Requisito 2: Para um estado lógico *HIGH*, a tensão no receptor ARINC 429 deve ser de +6.5V até +13V.
- Requisito 3: O receptor deve possuir um *buffer First-In-First-Out* (FIFO) de até 512 palavras de 32 bits.

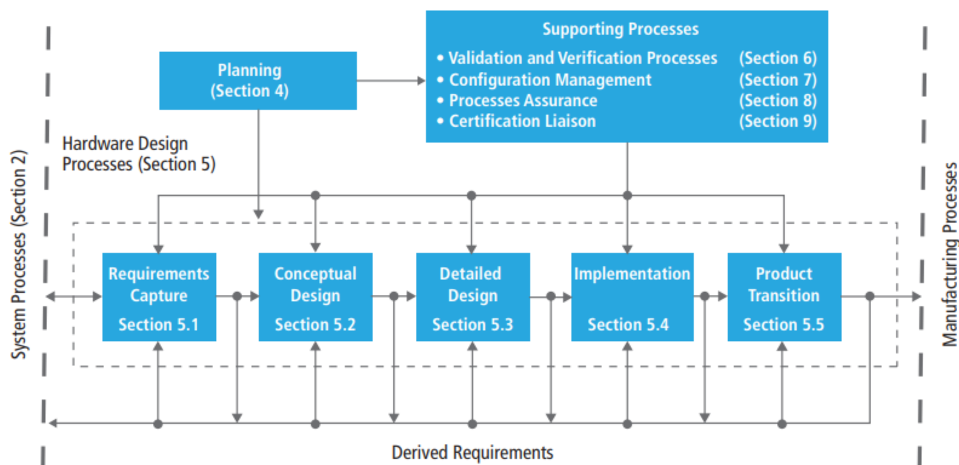
Observando esses três requisitos é possível notar que há uma certa hierarquia sistêmica entre os requisitos. O primeiro requisito, que estabelece a necessidade de uma interface ARINC 429, está indicando um nível de abstração mais alto, um nível de LRU; logo, esse requisito indica uma necessidade da LRU como um todo e não de seus componentes. O segundo requisito indica um requisito que discorre sobre níveis de tensão; logo, esse requisito manifesta uma necessidade de uma SRU, ou seja, de uma placa de

circuito impresso. É nítido que o nível de abstração do segundo requisito é menor do que o primeiro requisito. O terceiro requisito trata de uma necessidade vinculada à lógica do sistema, ou seja, vinculada a um *Programmable Logic Device* (PLD), que pode ser um FPGA, por exemplo. Esse requisito é de um nível ainda mais baixo do que o requisito anterior. É importante ressaltar que nesse contexto a palavra abstração faz referência a uma hierarquia sistêmica e não a uma hierarquia de *hardware* e *software*, como foi mencionado anteriormente.

Em um projeto que segue a DO-254, existem diversos tipos de requisitos de diferentes hierarquias. Os requisitos sistêmicos de mais alto nível vão sendo "quebrados"em requisitos cada mais específicos, seja de SRU (placa) ou de PLD. Nesse trabalho, será apenas tratado dos requisitos de PLD, ou seja, os requisitos de lógica.

Em seguida será apresentado resumidamente o processo da DO-254 para FPGAs, pois como o projeto se trata de lógica programável, ele se enquadra na categoria de PLD. É importante ressaltar que a DO-254 se aplica a qualquer *hardware* na aviação, como *Application-Specific Integrated Circuits* (ASIC), placas de circuito impresso e outros. O fluxo do processo da DO-254 é dado pela figura 2.2.

Figura 2.2: DO-254 Flow



Fonte: Cadence (2019).

- **Planejamento**

  O planejamento é uma etapa crítica no processo de desenvolvimento de *hardware* pois nessa fase a empresa responsável pelo *design* declara a abordagem que utilizará para atingir a certificação com base na DO-254. Exemplos de entidades regulamentadoras na aviação que tem o poder de fornecer a certificação para determinado *hardware* desenvolvido seguindo a DO-254 são a FAA e a EASA. Nessa etapa é apresentado para as entidades certificadoras um documento chamado PHAC (*Plan*

for hardware Aspects of Certification). Esse documento ilustra como os aspectos da DO-254 serão abordados. Essa etapa é tratada na seção 4 da DO-254 (RTCA; EUROCAE, 2000).

- **Captura de Requisitos**
  A captura de requisitos é uma etapa fundamental e decisiva do processo. Nela os requisitos são escritos. Visto que a DO-254 é baseada em requisitos, o processo será baseado nos requisitos que serão escritos nessa etapa; logo, os requisitos precisam ser claros e devem incluir critérios como testabilidade, confiabilidade e coerência visando os critérios de validação e verificação de *hardware*. A diferença entre verificação e validação será abordada posteriormente. Essa etapa é tratada na seção 5.1 da DO-254 (RTCA; EUROCAE, 2000).
- **Design Conceitual**
  Nessa etapa, com base nos requisitos escritos na etapa anterior e uma ideia definida do sistema, o *design* é quebrado em diversas partes menores, diversos componentes (blocos). Nessa etapa são desenvolvidos os diagramas de blocos do sistema a nível de PLD e outros diagramas de apoio, como diagramas de máquinas de estado. Também nessa etapa são definidos os *intellectual property* (IPs) que serão usados e o próprio FPGA que será usado, caso a lógica desenvolvida seja embarcada em um FPGA. Essa etapa é tratada na seção 5.2 da DO-254 (RTCA; EUROCAE, 2000).
- **Design Detalhado**
  Nessa etapa ocorre a implementação do código usando alguma linguagem de descrição de *hardware*. Cada bloco definido no *design* conceitual será associado a algum ou alguns requisitos. O *design* em *Register Transfer Level* (RTL) desenvolvido nessa etapa deve ser capaz de implementar os requisitos de cada componente. Ao mesmo tempo, um time de verificação deve implementar testes para verificar se o RTL de fato reproduz a sua determinada função. A verificação inclui desde métodos de verificação utilizando linguagens como systemVerilog até procedimento de testes físicos em FPGAs. O *design* detalhado é tratado na seção 5.3 da DO-254 (RTCA; EUROCAE, 2000).
- **Implementação** Essa etapa depende do tipo de tecnologia utilizada. Para *designs* baseados em RTL, como FPGAs e ASICs, a implementação envolve o processo de síntese lógica. Nessa etapa, para FPGAs, é produzido o arquivo de gravação do *design*. Essa etapa é tratada na seção 5.4 da DO-254 (RTCA; EUROCAE, 2000).
- **Transição para Produto** Nessa fase, o *design* é transferido para a produção. Vários

aspectos precisam ser verificados, como por exemplo, a versão correta de FPGA para se gravar, como será transferido essa versão, entre outros. Essa etapa é tratada na seção 5.5 da DO-254 (RTCA; EUROCAE, 2000).

Nesse trabalho, embora se trate de aviônica, não será considerado o escopo inteiro da DO-254 como processo de desenvolvimento, pois esse processo demora anos para ser executado e tem um custo muito elevado. Para se ter uma ideia, estima-se que o custo de um projeto que utiliza o método da DO-254 pode chegar a custar até quatro vezes o custo do projeto sem considerar a DO-254 (CADENCE, 2019). Ao invés disso, tentará se utilizar de uma abordagem simplificada da DO-254 elucidando as suas principais etapas. Desse modo, o *design* desenvolvido no presente trabalho não é certificável.

2.2 Verificação e Validação em Aviônica

A verificação e validação da solução são constantemente realizadas em processos de desenvolvimento de aviônica. Elas fazem parte de todo o processo de desenvolvimento seguindo a DO-254 seja diretamente ou indiretamente. Entretanto, para se compreender, é necessário diferenciar validação de verificação.

- **Validação**
  O conceito de validação está relacionado com a premissa de que o produto atende as necessidades do cliente. Em outras palavras, se o produto é útil para o cliente. O processo de validação garante que os requisitos escritos estão alinhados com a demanda do cliente e com os requisitos de mais alto nível, como os requisitos de SRUs e LRU (ALDEC, 2021). Normalmente o termo validação é usado em um nível de abstração sistêmica mais alto de projeto.
- **Verificação**
  O conceito de verificação aplicado à aviônica resume-se em provar que o sistema em questão foi desenvolvido da forma correta, ou seja, se ele funciona corretamente. Entretanto, a verificação não busca apenas saber se o sistema funciona corretamente, mas também busca assegurar que os requisitos foram de fato implementados no design (LUNA; ZALEWSK, 2011). Verificação é associado a um nível de abstração mais baixo do projeto.

Pode-se sumarizar a metodologia de verificação segundo a DO-254 como sendo do tipo *Requirements-Based Verification* (RBV).

2.2.1 Requirements-Based Verification (RBV)

A verificação baseada em requisitos tem como finalidade provar que os requisitos em questão são atendidos garantindo que o *design* construído pelo time de lógica é seguro. Para tal, um time de verificação realiza simulações e testes em cima do *design*. Alguns exemplos de metodologias de verificação são: revisão de código, *functional simulation*, *timing simulation*, *static timing analysis* e *physical testing*. A seguir, é apresentada uma breve explicação sobre esses termos:

- **Revisão de código**: Como o nome sugere, trata-se de revisão de código. Uma pessoa que não participou do desenvolvimento do código realiza uma revisão do código.
- **Functional Simulation**: Trata-se de simulações que possuem a finalidade de averiguar o funcionamento do *design*, ou seja, se o sistema funciona da forma que deveria funcionar em diversas situações. Diversas metodologias são usadas como *test benches* desenvolvidos em linguagem de descrição de *hardware* e *Universal Verification Methodology* (UVM).
- **Timing Simulation e Static Timing Analysis**: Trata-se de simulações que consideram os atrasos de sinal no sistema, isto é, analisa se o *design* é capaz de operar na velocidade em que se espera operar. Esse método, por exemplo, é capaz de quebrar o *design* em *timing paths* e calcular o atraso de propagação do sinal em cada caminho (SYNOPSYS, 2021). Com isso, o método consegue checar se há violações nas restrições de tempo do design (*setup and hold time*).
- **Physical Testing**: A lógica desenvolvida é testada de uma maneira "física". Se a lógica desenvolvida tiver a finalidade de ser embarcada em um FPGA por exemplo, para realizar esse teste, gravaria-se o design em um FPGA e se realizaria medidas através de instrumentos como osciloscopios. Também se realizariam testes de acesso à interfaces e testes de integração do sistema com as SRUs envolvidas. Essa etapa é executada quando as SRUs e placas de teste já foram projetadas e manufaturadas.

Para garantir a cobertura dos requisitos, utilizam-se métodos de rastreabilidade que garantem que o *design* está ligado aos requisitos de lógica, requisitos de circuito de placa e aos procedimentos de testes. Também utilizam-se métodos que fornecem uma métrica de quanto os requisitos estão de fato sendo atingidos nas mais diversas situações. Por

exemplo, para um *design* DAL A, uma métrica razoável é de 97% de cobertura funcional (BUTKA, 2015).

Nesse trabalho será adotado como método de verificação apenas a simulação funcional (*functional simulation*). Para isso, *test benches* serão criados a fim de provar os requisitos escritos.

2.3 ARINC 429

A especificação ARINC 429, também conhecida como "*Mark33 Digital Information Transfer System* (DITS)", é um padrão técnico que define uma interface diferencial digital para transmissão de dados entre sistemas usados na aviação, predominantemente na aviação civil. Esses sistemas são na prática LRUs. A especificação foi publicada pela ARINC em 1977 e, desde lá, se tornou uma das interfaces mais comuns na aviação civil, sendo usada até hoje. A discussão seguinte será baseada na norma ARINC 429, AEEC (2004), no livro *Avionics Handbook*, Spitzer (2001), e no datasheet do core de ARINC 429 vendido pela Actel (2006).

A apresentação da interface será divida em duas partes: aspectos elétricos e aspectos lógicos. Como o foco do trabalho é a parte lógica, a parte elétrica será apresentada resumidamente.

2.3.1 Aspectos Elétricos

2.3.1.1 Características Gerais

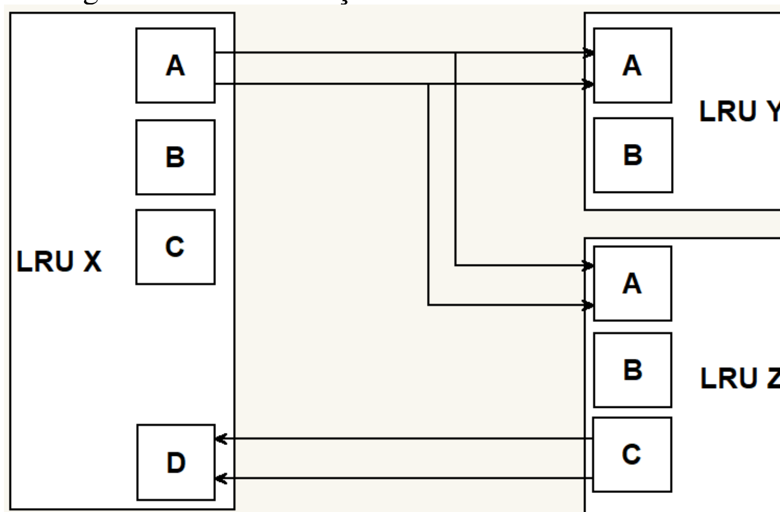
o primeiro passo para se entender uma interface é entender suas características gerais. As características gerais da interface ARINC 429 são:

- A Interface é serial digital;
- O sinal é diferencial;
- A comunicação é unidirecional (simplex).

Para ilustrar o funcionamento dessas características, considere a figura 2.3:

O transmissor da porta A da LRU X envia dados através de um par trançado para os receptores da porta A das LRUs Y e Z. Em cada LRU que está recebendo dados (Y e Z) existe um decodificador apropriado para interpretação dos dados recebidos. Na porta

Figura 2.3: Comunicação via ARINC 429 entre LRUs



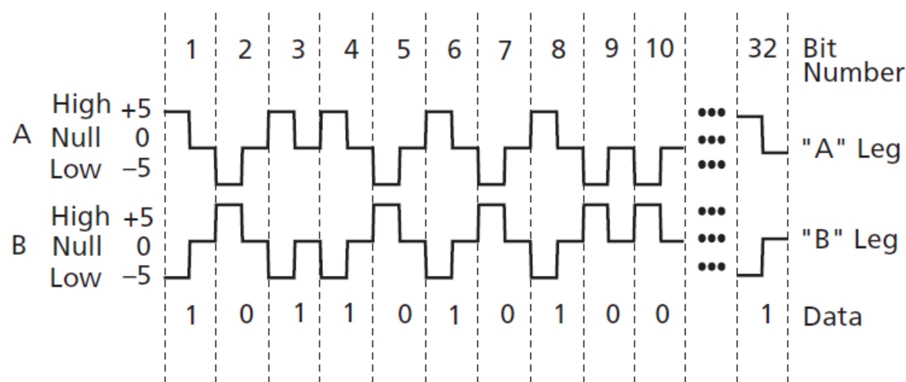
Fonte: Autoria própria.

C da LRU Z existe um transmissor que envia dados para o receptor da porta D da LRU X. Como a interface é unidirecional, a LRU X pode enviar dados através da porta A para a porta A das LRUs Y e Z mas não pode receber dados através dessa porta. Desse modo, para a LRU Z se comunicar com a LRU X, ela precisa usar outra porta. No caso, ela faz uso do transmissor da porta C que está ligado no receptor da porta D da LRU X. Esse comportamento é chamado de unidirecional ou *simplex*. *Simplex* faz referência a uma interface que só é capaz de enviar dados em uma única direção. Esse comportamento é diferente de uma interface I2C, por exemplo, que é capaz de enviar e receber dados pelo mesmo canal, embora não simultaneamente. Esse comportamento é chamado de *half-duplex*.

2.3.2 Modulação

A modulação que deve ser usada com a interface ARINC 429 é a *Return to Zero Bipolar* (AEEC, 2004). Na figura 2.4 é ilustrado esse tipo de modulação já para um sinal ARINC 429. Nota-se na figura 2.4 que existem dois estados lógicos: *HIGH* e *LOW*. Entretanto, existe um terceiro estado lógico não retratado na figura que se chama *NULL*. Esse estado ocorre quando tanto a linha A como a linha B ficam em zero por um período completo.

Figura 2.4: Modulação RZ Bipolar



Fonte: Actel (2006).

2.3.3 Níveis de Tensões

A norma estabele que, dado uma interface ARINC 429 composta pelo par diferencial A e B, as tensões na saída do transmissor sem carga devem ser de acordo com a tabela 2.1. A tensão da linha A para a linha B é uma tensão diferencial e a tensão A ou B com referência ao ground são tensões *single ended*.

Tabela 2.1: Tabela de tensões na saída do transmissor sem carga.

	HI (V)	NULL (V)	LO (V)
Line A to Line B	+10 ± 1	0 ± 0.5	-10 ± 1
Line A to Ground	+5 ± 0.5	0 ± 0.25	-5 ± 0.5
Line B to Ground	-5 ± 0.5	0 ± 0.25	+5 ± 0.5

Fonte: Confecção do autor com base em AEEC (2004).

As tensões aceitáveis no receptor sem considerar ruído, isto é, em uma situação hipotética, são ilustradas na tabela 2.2.

Tabela 2.2: Tensões na entrada do receptor sem considerar ruído.

HI	+7.25V to +11V
NULL	-0.5V to +0.5V
LO	-7.25V to -11V

Fonte: Confecção do autor com base em AEEC (2004).

Considerando ruído e instalações reais, se aceita uma tolerância. As tensões aceitáveis no receptor, considerando ruído e situações reais, são ilustradas na tabela 2.3.

Tabela 2.3: Tensões na entrada do receptor considerando ruído.

HI	+6.5V to +13V
NULL	-2.5V to +2.5V
LO	-6.5V to -13V

Fonte: Confecção do autor com base em AEEC (2004).

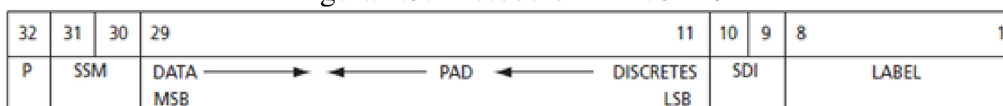
2.3.4 Aspectos Lógicos

Essa seção tem o intuito de detalhar os aspectos lógicos da interface ARINC 429, como tipos de dados e o formato da palavra.

2.3.4.1 Formato da palavra

As palavras transmitidas no padrão ARINC 429 contém 32 bits de tamanho e são divididas conforme a figura 2.5:

Figura 2.5: Protocolo ARINC 429



Fonte: Actel (2006).

Antes de detalhar cada campo da palavra, é necessário enfatizar a sequência de transmissão da palavra. A sequência de transmissão do primeiro campo, o *label*, se dá pelo MSB. Os campos seguintes se dão pelo LSB. Desse modo, a sequência de transmissão de uma palavra ARINC 429 é: 8-7-6-5-4-3-2-1-9-10-11 ... 30-31-32. Essa característica de *label* "reverso"é um legado dos sistemas passados onde a codificação em octal do campo *label* era, aparentemente, sem significância (AEEC, 2004).

A seguir será detalhado as componentes da palavra.

- **Label**

  Os primeiros 8 bits da palavra ARINC 429 são destinados ao *label*. O *label* é um código em octal que identifica o tipo de informação contido na palavra e o formato da palavra. Por exemplo, para se transmitir uma informação de altitude, deve-se usar o *label* de altitude. Através do *label* de altitude, o decodificador do receptor sabe que a informação contida no resto da palavra é sobre altitude e não de outro tipo de dado, como velocidade por exemplo. Para entender-se como os *labels* funcionam, é necessário entender o conceito de *Equipment ID*.

  Como foi mencionado anteriormente, o campo é composto de 8 bits; logo, é possível ter 256 labels. Isso gera um problema, pois em uma aeronave com diversos sistemas (LRUs) diferentes, existem mais de 256 tipos de dados. Entretanto, o número de tipos de dados de um único sistema (LRU) é inferior a 256. Desse modo, surge a ideia de criar um código identificador para cada equipamento (LRU) e reaproveitar *labels*. Os identificadores de equipamento são chamados de *Equipment*

IDs e são dados em hexadecimal. Para exemplificar esse conceito, considere o label 056. O label 056, para o *Equipment* ID 002 (*Flight Management Computer*) significa o tempo estimado de chegada da aeronave ao seu destino. Já para o *Equipment* ID 05 (*Attitude and Heading Ref. System*), o mesmo label significa direção do vento em relação ao norte magnético.

A norma explica que o *label* é na verdade formado por seis caracteres. Os 3 primeiros são os 8 bits em octal explicados anteriormente. Os outros 3 caracteres dados em hexadecimal compõem o Equipment ID. Como o engenheiro responsável pela interface ARINC 429 sabe qual equipamento transmissor será conectado em cada equipamento receptor, o *Equipment* ID não se torna necessário, pois já se espera labels de um único equipamento. Por exemplo, considere um transmissor ARINC 429 de um sistema de Navegação Inercial que quer enviar informações para o Computador de Missão e para o Computador de Controle de Voo. Os receptores ARINC 429 do Computador de Missão e de Controle de Voo não precisam receber o *Equipment* ID do Sistema de Navegação Inercial, pois como estão conectados unicamente a um Sistema de Navegação Inercial, eles só podem receber labels do Sistema de Navegação Inercial. A conclusão aqui é que o *label*, na prática, acaba sendo constituído apenas dos 8 bits em octal. Para todo caso, existe um label especial para envio do Equipment ID, que é o 377. Esse *label* não será abordado na monografia.

Para exemplificar, será considerado o *Equipment ID* 09 (*Airborne Distance Measuring Equipment*), que é um equipamento que mede a distância entre a aeronave e uma estação no solo. Também será considerado o *label* 201 desse equipamento (distância que esse equipamento mediu - *DME Distance*). Nas figuras 2.6 e 2.7 é possível ver as informações que o *label* 201 para o *Equipment ID* 09 faz referência.

Figura 2.6: Campos da Tabela de Labels da norma ARINC 429

Code No. (Octal)	Eqpt. ID (Hex)	Transmission Order Bit Position								Parameter	Data				Notes & Cross Ref. to Tables in Att. 6
		1	2	3	4	5	6	7	8		BNR	BCD	DISC	SAL	

Fonte: AEEC (2004).

Figura 2.7: Label 201

Code No.	Eqpt. ID	1	2	3	4	5	6	7	8	Parameter	BNR	BCD	DISC	SAL	Notes
2 0 1	0 0 9	1	0	0	0	0	0	0	1	DME Distance		X			6-1-1
	0 5 A	1	0	0	0	0	0	0	1	Fuel Temperature Right Wing Tank	X				
	1 1 2	1	0	0	0	0	0	0	1	TACAN Distance		X			
	1 1 4	1	0	0	0	0	0	0	1	Inner Tank 3 Fuel Temp & Advisory Warning	X				
	1 1 5	1	0	0	0	0	0	0	1	DME		X			6-25
	1 4 0	1	0	0	0	0	0	0	1	Mach Maximum Operation (Mmo)	X				
	1 4 2	1	0	0	0	0	0	0	1	Projected Future Latitude	X				
		1	0	0	0	0	0	0	1	GPS/GNSS Sensor - System Address Label				X	See Attachment 11

Fonte: AEEC (2004).

Os *labels*, além de fornecerem o tipo de dado, também fornecem informações como codificação, resolução, número de dígitos significativos, unidades, *range* e etc. Na figura 2.7, o *label* 201 para o *Equipment ID* 9, além de mostrar que se trata de um dado de *DME Distance* codificado em BCD (*Binary Coded Decimal* - os tipos de codificação serão tratados posteriormente), também faz referência ao anexo 6-1-1 da norma. Nesse anexo, figura 2.8, encontra-se o formato da palavra para esse *label* já considerando a codificação em BCD. Já na figura 2.9, encontra-se que a unidade é dada em milhas náuticas com *range* de -1 a 399.99, que o dado possui 5 dígitos e que a resolução é de 0.01 (AEEC, 2004). Esse exemplo serve para ilustrar que através do *label*, a informação contida no campo de dados da palavra ARINC 429 recebe seu sentido; portanto, o campo *label* é de extrema importância.

Figura 2.8: Anexo 6-1-1 da norma

P	SSM	BCD CH #2	BCD CH #2	BCD CH #3	BCD CH #4	BCD CH #5	SDI	8 7 6 5 4 3 2 1
		4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1		
0	0 0	0 1 0	0 1 0 1	0 1 1 1	1 0 0 0	0 1 1 0	0 0	1 0 0 0 0 0 0 1
Example		2	5	7	8	6		DME DISTANCE (201)

Fonte: AEEC (2004).

Figura 2.9: Trecho da tabela de dados BCD da norma

Label	Eqpt ID (Hex)	Parameter Name	Units	Range (Scale)	Sig Bits	Pos Sense	Reso- lution	Min Transit Interval (msec) 2	Max Transit Interval (msec) 2	Max Trans- port Delay (msec) 3	Notes & Cross Ref. to Tables and Attachments
2 0 1	0 0 9	DME Distance	N.M.	-1-399.99	5		0.01	83.3	167		6-1-1
	1 1 2	TACAN Distance	N.M.	0-399.99	5		0.01	190	210		
	1 1 5	DME Distance	N.M.	0-399.99	5		0.01	50	50		

Fonte: AEEC (2004).

- **Source Destination/Identifider (SDI)**

  Os bits 9 e 10 são usados quando se precisa transmitir uma palavra específica para um determinado sistema de uma instalação multi-sistema ou quando o transmissor de uma instalação multi-sistema precisa ser identificado (AEEC, 2004). Por exemplo, em um cenário onde existam três LRUs receptoras e uma LRU transmissora, a LRU transmissora pode escolher para qual instalação quer enviar a palavra de acordo com a tabela 2.4. Esses bits também servem para aumentar a resolução de determinados *labels*.

- **Data**

  Bits 11 até 29 são reservados ao dado que está sendo transmitido. Existem 4 tipos de codificação de dados: BNR, BCD, Discrete e ISO Alphabet Number 5 (figura 5.1). A descrição de cada grupo se encontra abaixo.

  BNR: Codificação expressa em números binários que representa um valor fracionário. Para exemplificar, será usado o *label* 366 (*North South Velocity*) para o

Tabela 2.4: Tabela SDI para instalação multi-sistema.

Bit 10	Bit 9	Installation No.
0	0	All
0	1	1
1	0	2
1	1	3

Fonte: Autoria própria com base em AEEC (2004).

Equipment ID 38 com o dado 00 1000 0000 0000 0000. A norma estabelece que para essa combinação de *label* e *Equipment ID*, a unidade do dado é expressa em nós, o *range* é de 4096, e é composto de 15 bits significativos (AEEC, 2004). É importante salientar que para a maioria dos dados em BNR, com raras exceções, o bit 29 não faz parte do campo de dados, faz parte do campo *Sign State Matrix*(SSM) que será explicado posteriormente. Desse modo, para a grande maioria dos dados codificados em BNR, os bits disponíveis para o campo de dados são do 11 ao 28. Outro aspecto importante é que o dado é distribuído do bit 28 ao bit 11 conforme o seu número de bits com o MSB no bit 28. Para o exemplo, a tabela 2.5 ilustra esse comportamento.

Tabela 2.5: Tabela mostrando a distribuição do dado.

28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fonte: Autoria própria.

Como esse *label* para esse *Equipment ID* possui apenas 15 bits, os bits 13, 12 e 11 não são usados; logo, são preenchidos com zeros. Eles não serão incluídos no cálculo da informação. O cálculo do valor codificado para um dado que contém o MSB no bit 28 e é composto de 15 bits é feito com base na lógica a seguir:

$$(bit(28) * 2^{-1} + bit(27) * 2^{-2} + ... + bit(14) * 2^{-15}) * Range = VBNR$$

Sendo VBNR o valor codificado no dado. Para o caso do exemplo, tem-se:

$$(0 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} + 0 * 2^{-4} + 0 * 2^{-5} + 0 * 2^{-6} + 0 * 2^{-7} + 0 * 2^{-8} + 0 * 2^{-9} + 0 * 2^{-10} + 0 * 2^{-11} + 0 * 2^{-12} + 0 * 2^{-13} + 0 * 2^{-14} + 0 * 2^{-15}) * 4096 = 512$$

O dado, nesse exemplo, é 512 nós.

BCD: *Binary Coded Decimal* (BCD) é uma codificação binária de números decimais onde cada dígito do número decimal é formado por 4 bits. Para o exemplo que foi usado anteriormente na seção 2.3.4.1, *label* 201 e *Equipment ID* 9 (*DME Distance*), o tipo de codificação é BCD. Nessa codificação, de acordo com as figuras 2.8 e 2.9, o bitstream será dividido em 5 dígitos com duas casas decimais de

resolução (0.01) e o dado será dado em milhas náuticas. Se o bitstream em questão for 000 0010 0010 0011 0101, tem-se respectivamente o número de 022.35 milhas náuticas.

Discrete: Essa codificação estabelece valores de 1 bit que indicam situações que possuem apenas dois significados opostos, como por exemplo, ligado ou desligado (ON/OFF), presença de falha ou ausência de falha e etc.

ISO Alphabet Number 5 Data: Essa codificação permite o envio de caracteres utilizando a tabela ISO Alphabet Number 5 (Figura 5.1). Essa codificação não será abordada nesse trabalho pois é pouco utilizada.

- **Sign/Status Matrix (SSM)**

  Para todos os tipos de codificação, exceto BNR, os bits 30 e 31 são destinados ao campo de *Sign/Status Matrix* (SSM). Para a codificação BNR, como foi visto anteriormente, o campo de SSM também pode abranger o bit 29.

  Os bits de SSM tem como objetivo identificar características de determinado dado como direção e sinal. Também servem para informar a condição do *hardware*, modo de operação ou a validade do dado contido na palavra. Para a codificação em BNR, os bits 31 e 30 tem o significado de status de acordo com a tabela 2.6. Já o bit 29, possui o significado de direção de acordo com a tabela 2.7.

Tabela 2.6: BNR SSM: bits 30 e 31

Bit 31	Bit 30	Function
0	0	Failure Warning
0	1	No Computed Data
1	0	Functional Test
1	1	Normal Operation

Fonte: Autoria própria com base em AEEC (2004).

Tabela 2.7: BNR SSM: bit 29

Bit 29	Function
0	Plus, North, East, Right, To, Above
1	Minus, South, West, Left, From, Below

Fonte: Autoria própria com base em AEEC (2004).

Considerando o mesmo exemplo usado para explicar a codificação BNR, ou seja, o *label* 366 para o *Equipment ID* 38 (*North South Velocity*) com o dado sendo de 512 nós e com SSM de 110, tem-se, com base nas tabelas 2.6 e 2.7, que o dado completo é velocidade de 512 nós na direção norte e com operação normal.

Já para a codificação BCD, os bits 30 e 31 tem seu significado de acordo com a

Tabela 2.8: BCD SSM: bits 30 e 31

Bit 31	Bit 30	Function
0	0	Plus, North, East, Right, To, Above
0	1	No Computed Data
1	0	Functional Test
1	1	Minus, South, West, Left, From, Below

Fonte: Autoria própria com base em AEEC (2004).

tabela 2.8. Considerando o mesmo exemplo usado para explicar a codificação BCD, ou seja, o *label* 201 para o *Equipment ID* 9 (*DME Distance*) com o dado sendo de 022.35 milhas náuticas e com SSM de 00, tem-se, com base na tabela 2.8, que o dado completo é +22.35 milhas náuticas. O *range*, de acordo com a figura 2.9, é de -1 a 399.99 milhas náuticas. As opções "*North, East, Right, To, Above*"não fazem sentido para o dado pois o *range* abrange apenas sinal negativo e positivo; portanto, a única opção que faz sentido para o dado é a "*plus*".

- **Parity (P)**

  O último bit da palavra é o bit de paridade que serve para garantir a integridade da palavra transmitida ou recebida. A paridade adotada é ímpar (AEEC, 2004).

2.3.5 Bit Rate

Existem duas possibilidades de bit rate: *High Speed* e *Low Speed*. Na opção de *High Speed*, o *bit rate* é de 100kbps e, na opção de *low speed*, o *bit rate* é de 12 a 14kbps (AEEC, 2004).

2.3.6 Sincronização e Clocking

O *clock* é inerente à palavra transmitida (AEEC, 2004). A identificação do intervalo entre palavras é feita com base no início de um estado lógico *HIGH* ou *LOW* em comparação com um estado *NULL* anterior. A sincronização é feita de acordo com no mínimo um *gap* de 4 intervalos de bit entre as palavras. Esse *gap* é, na verdade, o estado *NULL*; portanto, a sincronização é feita com base nos estados *NULL* contido entre as palavras enviadas.

3 DESENVOLVIMENTO

Como foi mencionado anteriormente, o objetivo do trabalho consiste em implementar a parte lógica de um receptor ARINC 429 utilizando noções do processo de desenvolvimento de *hardware* DO-254. Para tal, se utilizou o fluxo de atividades descritos na seção 2.1. A seguir será detalhada a solução para cada etapa do fluxo.

3.1 Planejamento

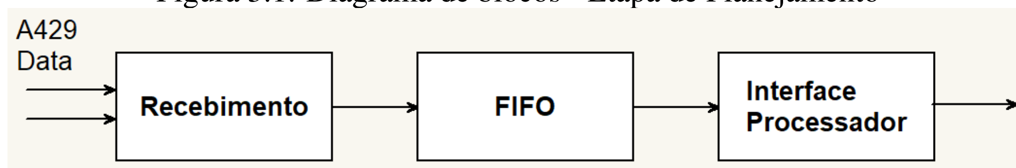
A principal finalidade dessa etapa é definir a abordagem que será utilizada para atingir a certificação perante uma unidade certificadora. Como o intuito desse trabalho não é produzir de fato um sistema que possa ser usado em uma aeronave comercial, não há necessidade de definir uma abordagem perante uma entidade certificadora, pois não haverá uma entidade que certificará esse *design*. No entanto, essa etapa foi utilizada para definir a abordagem geral do projeto. Alguns pontos importantes foram definidos. São eles:

- O aluno adotará o papel de um desenvolvedor de *hardware* responsável pelo desenvolvimento da lógica do receptor ARINC 429. A linguagem de descrição de *hardware* utilizada será VHDL.
- O *design* será quebrado em diversos blocos. Cada bloco receberá uma lista de requisitos. Esses requisitos serão responsáveis pelo escopo de projeto de cada bloco.
- O método para verificação dos requisitos será via simulação funcional, utilizando *test benches*. Os *test benches* também serão implementados em VHDL. Não será utilizado outro método de verificação.
- O funcionamento do sistema será considerado suficiente quando uma única palavra for recebida, armazenada e transmitida com sucesso.
- Como não há informações sobre a LRU em que pretende se usar o receptor, não há como definir o *Design Assurance Level* (DAL) do sistema. Obviamente, fabricantes que desenvolvem IPs que são comercializados para uso segundo a DO 254 apresentam o DAL do IP em sua documentação. No caso do presente trabalho, se escolheu DAL-D para não ter que adentrar profundamente o processo de verificação, visto que o objetivo do trabalho é mais focado na parte de *design*.

Também nessa etapa, já existe uma noção da arquitetura do projeto pois essa noção

servirá de *input* para a etapa de captura de requisitos. Obviamente, nessa etapa não se espera ter profundos detalhes do *design*, mas sim uma noção mais do ponto de vista funcional. A parte lógica dos receptores ARINC 429 é constituída por um bloco que é capaz de receber e recuperar a mensagem ARINC 429, convertê-la para uma modulação mais fácil de ser usada e então armazená-la em um *buffer* como um FIFO. Um bloco de interface com um processador também é comum nesse tipo de *design*. Pode-se resumir o *design* nos seguintes blocos: bloco receptor, FIFO e bloco de interface com processador. Na etapa de *design* conceitual, esses blocos vão ser definidos em detalhe. Até essa etapa, o diagrama de blocos da solução consiste na imagem 3.1.

Figura 3.1: Diagrama de blocos - Etapa de Planejamento

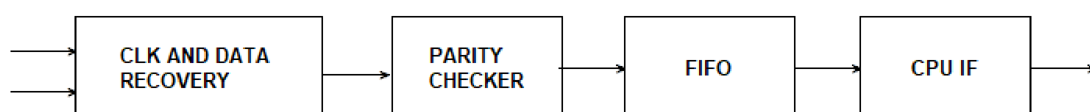


Fonte: Autoria própria.

3.2 Captura de Requisitos

Na etapa anterior, o sistema foi dividido em três blocos: bloco receptor, FIFO e bloco de interface com processador. Com o passar das etapas de desenvolvimento, tem-se um aprofundamento e uma adição de complexidade ao *design*. Considerando isso, foi adicionado um bloco de verificação de paridade, além de detalhes nos blocos já existentes. Um desses detalhes é que o bloco de recebimento recebeu o nome de *Clock and Data Recovery* já se referindo a sua função. A figura 3.2 ilustra o diagrama de blocos até essa etapa. Cada bloco receberá seus requisitos com base em sua função nas sub-seções a seguir.

Figura 3.2: Diagrama de blocos - Etapa de Captura de Requisitos



Fonte: Autoria própria.

3.2.1 Requisitos - Clock and Data Recovery

No transmissor de uma interface genérica, os bits que compõem a mensagem são transmitidos a uma taxa que é baseada em um *clock* no transmissor. Para conseguir recuperar esses bits no receptor, é necessário extrair informações de *timing* dos dados recebidos que remetam ao *clock* do transmissor. Desse modo, se o *clock* do transmissor for recuperado com a mesma fase, a amostragem dos dados recebidos é feita com base nesse *clock*, o que faz com que seja possível recuperar os bits no receptor. A esse mecanismo é dado o nome de *clock and data recovery*. Em alguns casos, não é necessariamente obrigatório a recuperação do *clock* do transmissor no receptor, mas sim estabelecer um sincronismo do *clock* global já presente no receptor com os dados recebidos. Desse modo, a amostragem é feita corretamente sem a recuperação do clock. Esse foi o método adotado no presente trabalho. Esse método será explicado em detalhes na etapa de *design* conceitual. O objetivo da presente explicação foi dar contexto aos requisitos escritos nessa etapa. A lista de requisitos do bloco de *clock and data recovery* é apresentada abaixo.

1. **[CDR01]** O bloco deve ser capaz de receber palavras ARINC 429 a uma taxa de 100kbps do transmissor.
2. **[CDR02]** O bloco deve contar com um mecanismo de sincronização do *clock* do bloco com os dados recebidos para que a amostragem seja feita corretamente.
3. **[CDR03]** O bloco deve ser capaz de converter as palavras ARINC 429 recebidas diferencialmente na codificação *Return to Zero Bipolar* para *single ended Non Return to Zero Polar*.
4. **[CDR04]** O bloco deve prover meios de informar os blocos adjacentes quando ele terminou de receber uma palavra.
5. **[CDR05]** O bloco deve prover um sinal de *input* de habilitação de leitura da palavra recebida.

3.2.2 Requisitos - Paridade

Como foi mencionado anteriormente, foi adicionado um bloco de verificação de paridade. Isso se deve ao fato de que a interface ARINC 429 possui um bit de paridade, o bit 31. A paridade, segundo a norma ARINC 429, é ímpar (AEEC, 2004). Paridade ímpar significa que o número de "1"s na palavra, incluindo o bit de paridade, deve ser ímpar. Por

exemplo, se em uma palavra recebida tem três vezes o estado lógico "1", ela tem paridade ímpar. Isso será explicado em detalhes na etapa de *design* conceitual. A lista de requisitos do bloco de verificação de paridade se encontra abaixo.

1. **[P01]** A interface deve prover meios de verificar a paridade da palavra de 32 bits recebida, isto é, verificar se a palavra está corrompida ou não.
2. **[P02]** A paridade deve ser do tipo ímpar conforme a norma ARINC 429.
3. **[P03]** Se a paridade da palavra não for ímpar, a palavra está incorreta; logo, a saída de dados do bloco não deve receber essa palavra.
4. **[P04]** Se a paridade da palavra for ímpar, a palavra está correta; logo, a saída de dados do bloco deve receber essa palavra.
5. **[P05]** O bloco deve prover meios de informar se a palavra está ou não corrompida ao bloco de controle.
6. **[P06]** O bloco deve prover meios de informar se o processo de verificação da paridade foi terminado ou não ao bloco de controle.
7. **[P07]** Deve ser possível resetar o bloco.
8. **[P08]** No estado de *reset*, a saída do bloco deve ficar em estado lógico *LOW*.
9. **[P09]** Se o bloco receber uma palavra com todos os bits em nível lógico *LOW*, a saída de dados deve ficar no estado lógico *LOW* e a interface não deve sinalizar palavra corrompida.
10. **[P10]** Deve existir um sinal de habilitação do bloco.

3.2.3 Requisitos - FIFO

O *buffer* do tipo FIFO tem o objetivo de armazenar as palavras recebidas. Desse modo, quando o processador deseja realizar uma leitura nas mensagens recebidas, é do FIFO que as palavras são retiradas. A lista de requisitos do FIFO se encontra abaixo.

1. **[FIFO01]** A interface receptora ARINC 429 deve ser capaz de armazenar as palavras recebidas com paridade correta em um *buffer* do tipo FIFO.
2. **[FIFO02]** O *buffer* do tipo FIFO deve ser capaz de armazenar até 4 palavras de 32 bits.
3. **[FIFO03]** Deve ser possível ler e escrever no FIFO usando sinais de controle de leitura e escrita.

4. **[FIFO04]** Deve ser possível ler e escrever ao mesmo tempo no FIFO.
5. **[FIFO05]** Enquanto o FIFO não estiver sendo lido, sua saída de dados deve permanecer em estado *LOW*.
6. **[FIFO06]** O FIFO deve prover meios de informar o bloco de controle se está cheio ou vazio.
7. **[FIFO07]** Uma tentativa de ler o FIFO estando ele vazio deve retornar estado *LOW* na saída de dados.
8. **[FIFO08]** Uma escrita no FIFO, estando ele cheio, não deve ser possível, mantendo ele assim suas palavras originais.
9. **[FIFO09]** Deve ser possível resetar o FIFO.
10. **[FIFO10]** No estado de *reset*, o FIFO deve ficar vazio e a saída de dados em estado *LOW*.

3.2.4 Requisitos - CPU IF

O bloco de CPU IF tem o objetivo de realizar a interface do receptor ARINC 429 com um processador de 8 bits. O processador realiza leituras no FIFO a fim de receber as palavras armazenadas. Desse modo, o processador pode começar o processo de decodificação da mensagem. É também nesse bloco que os bits de SDI são checados. A lista de requisitos para esse bloco encontra-se abaixo.

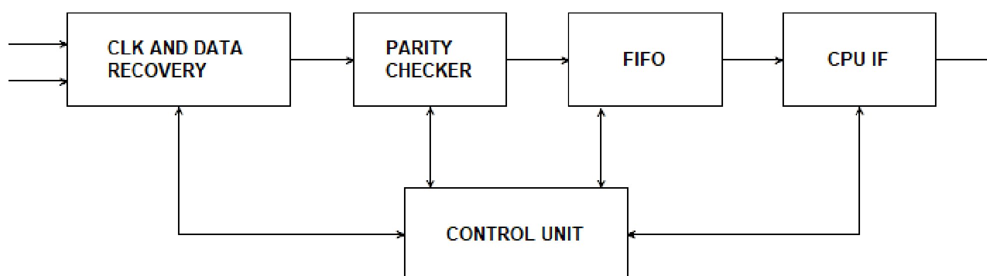
1. **[CPU01]** O bloco deve prover meios de interfacear o bloco FIFO com um processador que possui barramento de dados de 8 bits, possibilitando assim a leitura da palavra recebida pelo processador.
2. **[CPU02]** O bloco deve prover meios de configurar o endereço da instalação receptora (LRU).
3. **[CPU03]** O bloco deve prover meios de verificar se os bits de SDI (bit 9 e 10) da palavra recebida correspondem aos bits de identificação da LRU receptora (requisito CPU02).
4. **[CPU04]** Se os bits de SDI corresponderem ao endereço da LRU, a saída do bloco deve receber a palavra A429.
5. **[CPU05]** Se os bits de SDI não corresponderem ao endereço da LRU, a saída deve permanecer em nível lógico *LOW*.

6. **[CPU06]** Como o processador que realiza a interface possui barramento de 8 bits e a palavra A429 possui 32 bits, os *bytes* que formam a palavra A429 devem ser multiplexados no barramento de dados do processador.
7. **[CPU07]** A cada *byte* recebido pelo processador, o processador enviará um sinal confirmando o recebimento; o *byte* seguinte só deve ser enviado pelo bloco após a confirmação do recebimento pelo processador.
8. **[CPU08]** A palavra só deve ser transferida ao processador se requisitada; caso contrário, a saída deve ficar em estado lógico *LOW*.
9. **[CPU09]** Deve ser possível resetar o bloco.
10. **[CPU10]** No estado de *reset*, a saída do bloco deve ficar em estado lógico *LOW*.

3.3 Design Conceitual

Com base nos requisitos criados na etapa de captura de requisitos e uma ideia já definida do sistema, nessa etapa definiu-se os diagramas de blocos finais, suas funcionalidades e atributos como máquinas de estado. O diagrama de blocos final da solução se encontra na figura 3.3:

Figura 3.3: Diagrama de blocos - Etapa de Design Conceitual



Fonte: Autoria própria.

A seguir, a estratégia de *design* de cada bloco será explicada.

3.3.1 Clock and Data Recovery

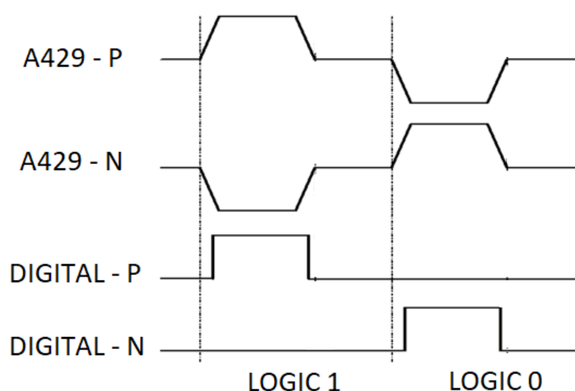
Como foi mencionado anteriormente, no transmissor de uma interface genérica, os bits que compõem a mensagem são transmitidos a uma taxa que é baseada em um *clock* no transmissor. Esses dados então seguem para o receptor. Para o receptor conseguir

amostrar corretamente esses dados, é necessário extrair informações de *timing* dos dados recebidos para que de alguma maneira o *clock* do transmissor seja recuperado e seja usado para realizar a amostragem nos dados recebidos. É importante ressaltar que não apenas a frequência deve ser recuperada, mas também a fase do *clock* recuperado deve estar alinhada com a fase dos dados recebidos. Desse modo, é possível obter um sinal de *clock* que é capaz de amostrar corretamente os dados recebidos. A esse processo de extrair informações de *timing* dos dados recebidos, obtendo assim um sinal de *clock* capaz de amostrar corretamente os dados recebidos, é dado o nome de *clock and data recovery* (CDR).

O processo de CDR é padrão para interfaces de alta velocidade em que o *clock* recuperado é maior do que o *clock* global do bloco receptor. Um exemplo de interface que dispõe de CDR é a PCIe Gen2, que opera a 5Gbps (SDA, 2020). Para interfaces que não são de alta velocidade e que o *data rate* é conhecido, pode-se adotar uma estratégia diferente. Essa estratégia será explicada tendo como base o que foi feito para a interface ARINC 429, já que ela opera em sua forma mais rápida a 100kbps (AEEC, 2004).

Antes de explicar a estratégia adotada para o bloco de CDR, é necessário ilustrar o formato do sinal ARINC 429 depois de passar por algum receptor que converta os níveis de tensão ARINC 429 para algum nível de tensão que o FPGA entenda. Considerando isso, é esperado o formato de onda presente na imagem 3.4.

Figura 3.4: ARINC 429 - Digital

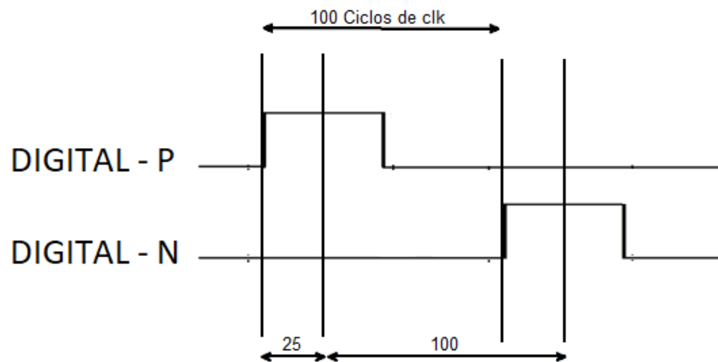


Fonte: Imagem modificada, Ashware (2021).

A estratégia de CDR que foi adotada nesse trabalho consiste na premissa de que o *clock* global do receptor é muito maior do que o *clock* embutido nos dados recebidos. Como não há requisito para a frequência do *clock* global do receptor, escolheu-se um *clock* de 10MHz. Considerando que a taxa adotada é de 100kbps e que um bit ARINC 429 consiste no tempo de informação mais o tempo de retorno a zero, como é ilustrado na

figura 3.4, pode-se assegurar que ocorrem 100 ciclos do *clock* global por bit ARINC 429 (10M/100k = 100). Desse modo, a estratégia para amostrar os dados recebidos consiste em detectar a borda do primeiro bit recebido, contar até 25 para chegar na quarta parte do primeiro bit e então realizar uma amostragem. Após isso, conta-se até 100 para que se chegue até o próximo bit. A figura 3.5 ilustra essa técnica.

Figura 3.5: Estratégia de CDR



Fonte: Imagem modificada, Ashware (2021).

Essa técnica tem como objetivo sincronizar o *clock* global com os dados a partir da detecção da borda do primeiro bit. Ela não consiste em uma técnica que recupera o *clock* de fato. Isso só é possível pois a taxa de transmissão é muito baixa (100Kbps), consequentemente o *clock* do transmissor é muito baixo também. Considerando que a transmissão é de 100Kbps, pode-se estimar que no transmissor exista um *clock* de 200KHz enviando um bit de informação e um bit em nível lógico baixo para representar o tempo de retorno a zero. Esse número é muito menor que 10MHz; portanto, pode-se usar essa estratégia nesse caso. Essa estratégia é usada também em *universal asynchronous receiver-transmitters* (UARTs), por exemplo. Apesar dessa abordagem não recuperar o clock, o nome do bloco foi mantido como *clock and data recovery* pois sua função é a mesma de um bloco de CDR padrão.

Na etapa de *design* conceitual também se realiza o projeto das máquinas de estado usadas na solução. No caso do bloco de CDR, foi desenvolvido uma máquina de estados que apresenta os seguintes estados já ordenados:

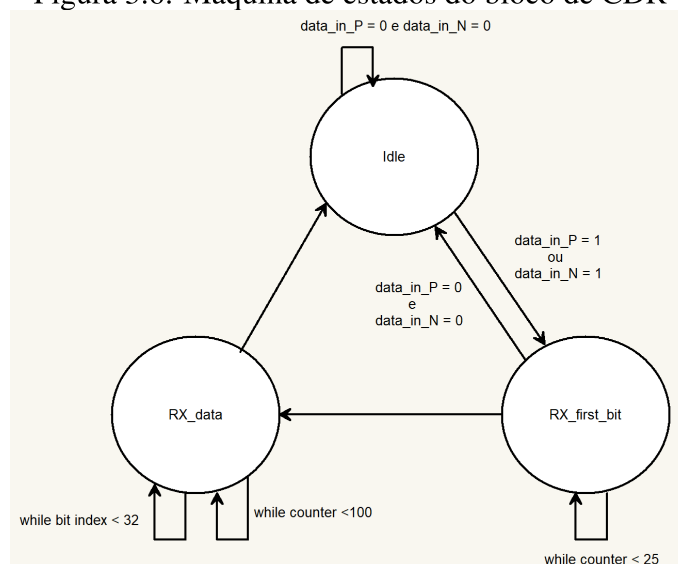
- **Idle**: Estado em que o bloco fica esperando algum dado ser recebido. Nesse estado é feita a detecção de borda do primeiro bit a ser recebido. Caso algum bit seja recebido, a máquina de estados vai para o próximo estado; caso contrário, ela permanece no estado *idle*.
- **RX_first_bit**: Tendo sido detectada a borda do primeiro bit, a máquina de estados

inicia o estado de RX_first_bit. Nesse estado, o contador é iniciado e incrementado em uma unidade. Esse estado segue se repetindo até o contador chegar em 25 unidades. Quando o contador chega em 25 unidades, significa que chegou-se até a quarta parte do primeiro bit recebido, que é onde existe a informação relevante. Então é realizada a amostragem do primeiro bit. A transformação de diferencial para *single ended* também é feita nesse estado, isto é, se o sinal P está em "1"e o sinal N está em "0", o bit amostrado é "1"; caso o sinal P esteja em "0"e o sinal N esteja em "1", o bit amostrado é "0". A partir da amostragem do primeiro bit, a máquina de estados vai para o próximo estado.

- **RX_data**: Para chegar-se até o próximo bit, é necessário incrementar em 100 unidades o contador a partir da quarta parte do primeiro bit. Como nesse estado é feito o incremento de apenas uma unidade do contador, esse estado se repete por 100 vezes para que chegue-se até o próximo bit. Uma vez chegado no próximo bit, a amostragem e a transformação de diferencial para *single ended* são feitas novamente. Então, esse processo se repete por 31 vezes para que os 32 bits sejam amostrados. Após o recebimento da mensagem, a máquina de estados volta para o estado inicial (*idle*).

A imagem 3.6 demonstra o comportamento simplificado da máquina de estados. Não foi incluído o sinal de *reset* na máquina para simplificar o desenho.

Figura 3.6: Máquina de estados do bloco de CDR



Fonte: Autoria própria.

O diagrama de blocos do bloco CDR se encontra na figura 3.7.

A descrição dos sinais se encontra a seguir:

Figura 3.7: Design Conceitual - Clock and Data Recovery

Fonte: Autoria própria.

- data_in_P e data_in_N: sinal diferencial digital de entrada.
- clk: *clock* global de 10MHz.
- rst: sinal de *reset* do bloco.
- rd: sinal de habilitação de leitura.
- data_out: palavra recebida (32 bits).
- ready: sinal que informa que o bloco terminou sua função.

3.3.2 Parity Checker

Esse bloco tem o objetivo de checar a paridade da palavra recebida para verificar se ela é válida ou não. Como foi mencionado anteriormente, a paridade da interface ARINC 429 é ímpar. Paridade ímpar significa que a soma do número de bits da palavra recebida que estão em nível lógico "1"deve ser ímpar. Desse modo, se algum bit mudar de seu estado lógico inicial ao longo do caminho de transmissão, a soma de bits em nível lógico "1"não será ímpar. A seguir, na tabela 3.1, alguns exemplos são ilustrados para uma palavra de 5 bits.

Tabela 3.1: Exemplos de paridade ímpar para uma palavra de 5 bits

Bits	Bit de Paridade	Validade
1001	1	Palavra válida
1000	0	Palavra válida
1001	0	Palavra inválida
1000	1	Palavra inválida

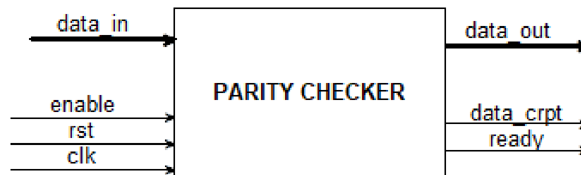
Fonte: Autoria própria.

A estatégia adotada para construir esse bloco é a utilização da operação lógica XOR entre todos os bits da palavra recebida. Por exemplo, para a palavra de 5 bits "10011", se for realizada a operação XOR entre todos os bits, o resultado será "1", indicando que a paridade é ímpar, isto é, está correta. Entretanto, se for feito a mesma operação para a palavra "10001", o resultado será "0", indicando que a paridade é par,

isto é, está incorreta. Essa estratégia foi utilizada para as palavras de 32 bits ARINC 429.

O diagrama de blocos da interface se encontra na figura 3.8:

Figura 3.8: Design Conceitual - Parity Checker



Fonte: Autoria própria.

A descrição dos sinais se encontra a seguir:

- data_in: palavra de entrada (32 bits).
- data_out: palavra de saída (32 bits).
- enable: sinal de habilitação do bloco.
- rst: sinal de *reset* do bloco.
- clk: *clock* global de 10MHz.
- data_crpt: sinal que informa se a palavra está corrompida.
- ready: sinal que informa que o bloco terminou sua função.

3.3.3 FIFO

O bloco FIFO é um *buffer* do tipo FIFO que pode armazenar até 4 palavras de 32 bits. Sua função é armazenar as palavras recebidas. Desse modo, quando o processador desejar ler as palavras recebidas, ele irá realizar leituras no FIFO.

O diagrama de bloco se encontra na figura 3.9 abaixo.

Figura 3.9: Design Conceitual - FIFO

Fonte: Autoria própria.

A descrição dos sinais se encontra a seguir:

- data_in: palavra de entrada (32 bits).
- data_out: palavra de saída (32 bits).

- wr_en: sinal de habilitação de escrita no FIFO.
- rd_en: sinal de habilitação de leitura no FIFO.
- fifo_full: sinal que indica que o FIFO está cheio.
- fifo_empty: sinal que indica que o FIFO está vazio.
- reset: sinal de *reset*.
- fifo_clk: *clock* global de 10MHz.

3.3.4 CPU IF

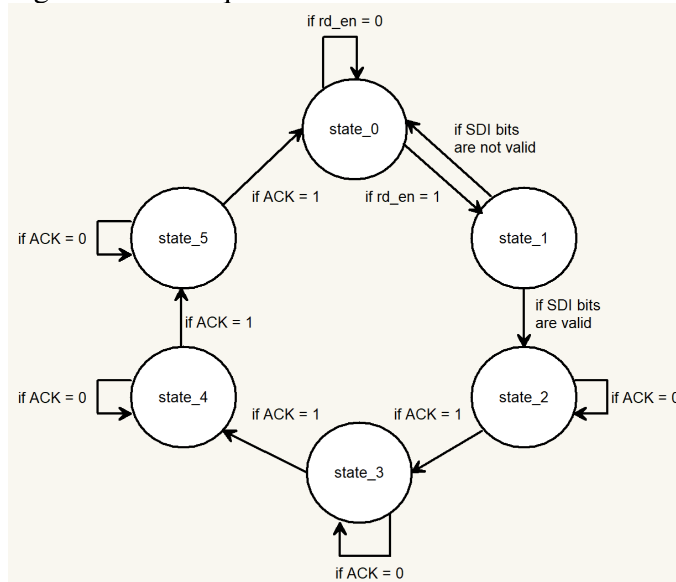
O bloco de CPU IF, como foi mencionado anteriormente na seção 3.2.4, além de verificar e permitir a configuração dos SDI bits da palavra, tem o objetivo de estabelecer interface entre um processador de 8 bits e o receptor ARINC 429. Desse modo, o processador pode ler as palavras do FIFO quando desejar. Como o processador possui um barramento de dados de 8 bits e as palavras ARINC 429 são de 32 bits, é necessário ocorrer 4 envios de 8 bits separados. Para tal, a estratégia adotada foi a de desenvolver uma máquina de estados capaz de cuidar dessa tarefa. Essa máquina de estados possui os seguintes estados:

- **State_0**: Estado em que se espera uma requisição de leitura do FIFO pelo processador. Quando ocorre a requisição de leitura através do sinal rd_en, a máquina vai para o próximo estado. Caso contrário, a máquina permanece esperando o sinal de requisição de leitura do processador.
- **State_1**: Estado em que os SDI bits da palavra recebida são comparados com os SDI bits configurados. Desse modo, as palavras que não possuem os SDI bits corretos podem ser descartadas. Se os SDI bits da palavra recebida forem iguais aos SDI bits configurados, a palavra é válida e a máquina segue para o próximo estado; caso contrário, a palavra é inválida e volta para o primeiro estado.
- **State_2**: Estado em que o primeiro *byte* da palavra é lido do FIFO pelo processador. Se o processador receber o *byte*, ele envia um sinal que indica o correto recebimento (sinal ACK). Uma vez recebido esse sinal ACK, a máquina segue para o próximo *byte*. Caso não receber, a máquina continua nesse estado até receber o sinal de ACK.
- **State_3**: Estado que possui o mesmo comportamento do state_2, com a diferença de estar tratando do segundo *byte* da palavra.

- **State_4**: Estado que possui o mesmo comportamento do state_2, com a diferença de estar tratando do terceiro *byte* da palavra.
- **State_5**: Estado que possui o mesmo comportamento do state_2, com a diferença de estar tratando do quarto *byte* da palavra. Entretanto, nesse caso, quando o sinal de ACK for recebido do processador, a máquina volta para o primeiro estado.

O diagrama de estados que descreve essa máquina se encontra na figura 3.10. O sinal de *reset* foi ocultado para melhor entendimento da imagem.

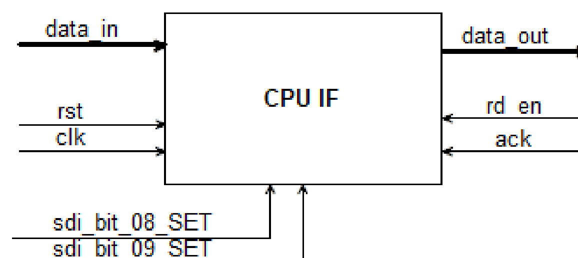
Figura 3.10: Máquina de estados do bloco de CPU IF



Fonte: Autoria própria.

O diagrama de blocos dessa interface se encontra na figura 3.11.

Figura 3.11: Design Conceitual - CPU IF



Fonte: Autoria própria.

A descrição dos sinais se encontra a seguir:

- data_in: palavra de entrada (32 bits).
- data_out: palavra de saída (8 bits).
- rd_en: sinal de habilitação de leitura.

- ack: sinal informativo de recebimento de um *byte* do processador.
- rst: sinal de *reset*.
- clk: *clock* global de 10MHz.
- sdi_bit_08_SET: Sinal para configurar o primeiro bit de SDI (bit 9 da palavra).
- sdi_bit_09_SET: Sinal para configurar o segundo bit de SDI (bit 10 da palavra).

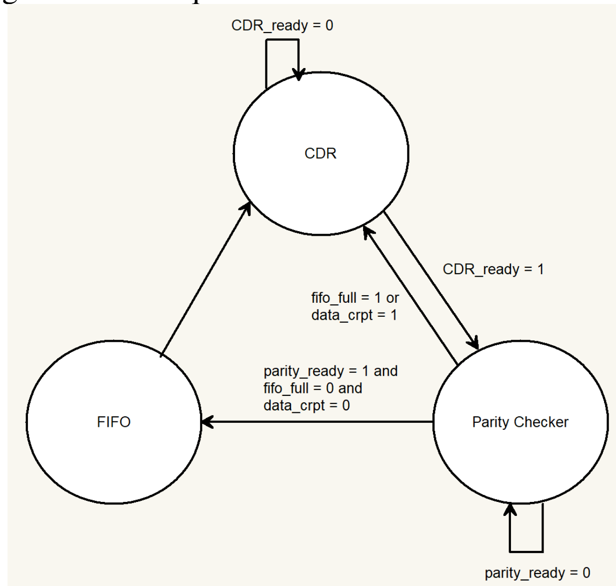
3.3.5 Control Unit

O bloco da unidade de controle tem como objetivo controlar a sequência de execução dos blocos descritos anteriormente. Desse modo, o recebimento da informação acontece de forma organizada respeitando a ordem de execução de cada bloco. Como esse bloco é mais um bloco de apoio do que um bloco de arquitetura, não é necessário escrever requisitos para ele. A estratégia adotada para controlar o receptor foi a de usar uma máquina de estados. Os estados da máquina de estados em questão são:

- **CDR**: Estado em que se espera o recebimento de dados pelo bloco de CDR descrito anteriormente. Uma vez que o bloco de CDR termina sua execução, ele informa o bloco de controle. Desse modo, o bloco de controle fornece os sinais necessários para a execução do próximo do bloco, isto é, do bloco de *parity checker*. A máquina de estados, nesse caso, vai para o próximo estado. Caso o bloco de controle não receba a informação de que o bloco de CDR terminou sua execução, a máquina de estados continua no estado de CDR até que a informação de que a execução no bloco de CDR foi terminada seja recebida.
- **Parity Checker**: Estado em que o bloco de controle fornece os sinais necessários para a execução do bloco de *parity checker*. Uma vez terminada a execução do bloco de paridade, o bloco de controle é informado. Caso a paridade da palavra recebida seja válida e o FIFO não esteja cheio, o FIFO é habilitado para a escrita e a máquina de estados vai para o próximo estado. Enquanto o bloco de paridade não termina sua execução, a máquina de estados fica no estado de *parity checker*. Caso a paridade da palavra recebida seja inválida ou o FIFO esteja cheio, a máquina volta para o estado de CDR.
- **FIFO**: Nesse estado, a máquina de estados volta para o estado inicial (CDR). Esse estado é usado para fins de sincronização.

A figura 3.12 ilustra a máquina de estados.

Figura 3.12: Máquina de estados do bloco de controle

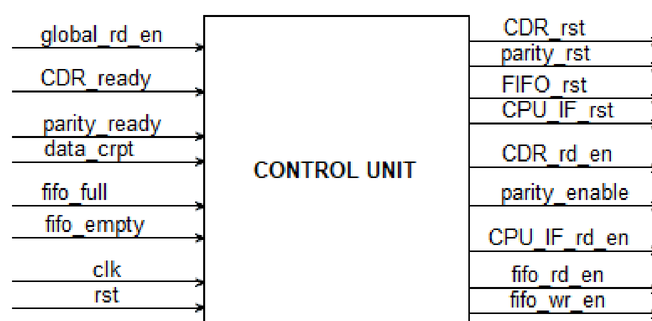


Fonte: Autoria própria.

Além da função descrita, o bloco de controle também estabelece o mecanismo de leitura. Esse mecanismo consiste na requisição de leitura do processador. Basicamente, o processador faz a requisição de leitura. Caso o FIFO não esteja vazio, o bloco de controle fornece os sinais necessários para a leitura do FIFO e a execução do bloco de CPU IF. Desse modo, a palavra é transferida para o processador.

O diagrama de blocos dessa interface se encontra na figura 3.13.

Figura 3.13: Design Conceitual - Control Unit



Fonte: Autoria própria.

A descrição dos sinais se encontra abaixo:

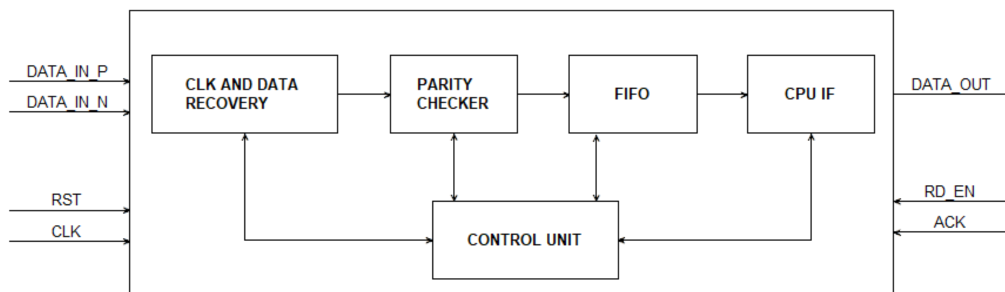
- global_rd_en: sinal de requisição de leitura vindo do processador.
- CDR_ready: sinal que informa que o bloco de CDR terminou sua execução.
- parity_ready: sinal que informa que o bloco de paridade terminou sua execução.
- data_crpt: sinal que informa se a palavra recebida esta corrompida ou não.

- fifo_full: sinal que informa se o FIFO está cheio ou não.
- fifo_empty: sinal que informa se o FIFO está vazio ou não.
- clk: *clock* global de 10MHz.
- rst: sinal de *reset* global.
- CDR_rst: sinal de *reset* do bloco de CDR.
- parity_rst: sinal de *reset* do bloco de paridade.
- FIFO_rst: sinal de *reset* do bloco FIFO.
- CPU_IF_rst: sinal de *reset* do bloco CPU IF.
- CDR_rd_en: sinal de habilitação de leitura dos dados do bloco de CDR.
- parity_enable: sinal de habilitação do bloco de paridade.
- fifo_rd_en: sinal de habilitação de leitura do FIFO.
- fifo_wr_en: sinal de habilitação de escrita no FIFO.
- CPU_IF_rd_en: sinal de habilitação de leitura dos dados do bloco CPU IF.

3.3.6 Diagrama Geral

Com toda a informação anterior, o diagrama geral de blocos se encontra na figura 3.14.

Figura 3.14: Design Conceitual - Topo



Fonte: Autoria própria.

A descrição dos sinais se encontra abaixo:

- DATA_IN_P e DATA_IN_N: sinal diferencial digital de entrada.
- RST: sinal de *reset*.
- CLK: *clock* global de 10MHz.
- DATA_OUT: sinal de saída de dados (8 bits).
- RD_EN: sinal de requisição de leitura vindo do processador.

- ACK: sinal informativo de recebimento de um *byte* do processador.

3.4 Design Detalhado

Nessa etapa, com base nas informações adquiridas, desenvolveu-se os códigos em VHDL. Além dos códigos de *design*, também desenvolveu-se *test benches* para a comprovação dos requisitos. A verificação do projeto será explicada no capítulo de resultados.

3.5 Implementação

Essa fase não foi executada pois o processo de síntese, *place and route* e de geração de um arquivo de gravação de FPGA não faz parte do escopo do trabalho.

3.6 Transição para Produção

Essa fase não foi executada pois o trabalho não é um produto; logo, ele não foi transferido para a produção.

4 RESULTADOS

Esse capítulo tem o intuito de mostrar os resultados obtidos, isto é, o capítulo ilustrará a verificação feita no sistema. Como foi comentado anteriormente, a verificação de sistemas DO-254 se baseia na comprovação dos requisitos escritos na etapa de captura de requisitos. Como a verificação segundo a DO-254 é um processo extremamente longo que envolve desde simulações até testes físicos dependendo do *Design Assurance Level* (DAL) do sistema, procurou-se limitar o escopo para que o trabalho se torne possível de ser concluído por uma única pessoa dentro do cronograma estabelecido. Pode-se listar os seguintes tópicos que foram adotados como metodologia de verificação:

- Seguir a premissa estabelecida na etapa de planejamento que salienta que o sistema será considerado suficiente quando for capaz de receber, armazenar e transmistir uma palavra.
- Comprovar que os requisitos estão de fato implementados e funcionando utilizando *test benches* implementados em VHDL (simulação funcional). Não serão adotados métodos de verificação como UVM.

A estratégia adotada para a escrita dos *test benches* foi a de fazer um código por bloco. Cada código contém diversos testes que comprovam diferentes requisitos do bloco em questão. Dentro do próprio código, cada teste foi "linkado"com o requisito que ele está testando. A simulação foi feita utilizando o *software* ModelSim, da *Mentor Graphics*. Com a utilização desse *software*, é possível ver o comportamento dos sinais testados (forma de onda); portanto, é possível ver se os sinais estão funcionando do modo que deveriam. A seguir, serão discutidos os testes implementados por bloco. Embora os requisitos já tenham sido listados na captura de requisitos, eles serão repetidos para que se possa compreender o que cada teste faz.

4.1 CDR

O teste do bloco de CDR consiste em alimentar os sinais DATA_IN_P e DATA_IN_N com uma palavra ARINC 429 diferencial a uma taxa de 100kbps considerando a modulação *return to zero bipolar*. Para tal, se criou um *clock* de 200kHz para que seja possível enviar além da informação da palavra, o retorno a zero, como ilustra a figura 3.4. Após o envio da palavra, o sinal que habilita a leitura no bloco é acionado para que a palavra

recebida possa ser transmitida ao próximo bloco através do sinal de data_output. Desse modo, analisando o sinal de data_output na simulação, é possível verificar o que o bloco recebeu de fato. Se o bloco transmite a mesma palavra que recebeu mas na modulação *single ended non return to zero polar* (modulação padrão da eletrônica digital), quer dizer que o bloco cumpre a sua funcionalidade. Em outras palavras, quer dizer que o bloco consegue amostrar a palavra de entrada, decodificar ela e transmitir ela em um formato mais otimizado. A seguir, é apresentada a análise da metodologia adotada para cada requisito:

- *[CDR01] O bloco deve ser capaz de receber palavras ARINC 429 a uma taxa de 100kbps do transmissor.*

  No *test bench*, os dados são enviados a 100kbps. Desse modo, se o dado de saída é igual ao dado de entrada, significa que o bloco é capaz de receber palavras ARINC 429 a uma taxa de 100kbps.

- *[CDR02] O bloco deve contar com um mecanismo de sincronização do clock do bloco com os dados recebidos para que a amostragem seja feita corretamente.*

  Para provar esse requisito, basta que o mecanismo de amostragem explicado na seção de *design* conceitual atue corretamente. Desse modo, a amostragem é feita corretamente.

- *[CDR03] O bloco deve ser capaz de converter as palavras ARINC 429 recebidas diferencialmente na codificação Return to Zero Bipolar para single ended Non Return to Zero Polar.*

  A palavra transmitida através dos sinais DATA_IN_P e DATA_IN_N é gerada de modo que seja diferencial e com modulação *return to zero bipolar*. Para provar o requisito, basta checar se o dado de saída se encontra na modulação *single ended non return to zero polar* (modulação padrão na eletrônica digital).

- *[CDR04] O bloco deve prover meios de informar os blocos adjacentes quando ele terminou de receber uma palavra.*

  Para provar esse requisito, basta que o sinal de saída ready mude seu estado lógico quando a operação do bloco for terminada, isto é, quando o bloco terminou de receber a palavra. Para ver o comportamento desse sinal, basta acompanhá-lo através da simulação.

- *[CDR05] O bloco deve prover um sinal de input de habilitação de leitura da palavra recebida.*

  Para provar esse requisito, basta que exista um sinal que seja capaz de habilitar a

leitura da palavra recebida.

A tabela 4.1 ilustra os resultados utilizando o teste citado no início dessa seção e a metodologia de análise por requisito desse bloco.

Tabela 4.1: Resultado dos testes do bloco CDR

Requisitos	Resultado
CDR01	OK
CDR02	OK
CDR03	OK
CDR04	OK
CDR05	OK

Fonte: Autoria própria.

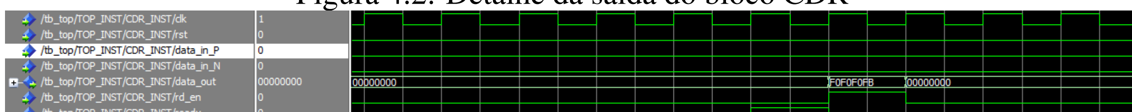
Um ponto a ser destacado é que na simulação foi possível ver que o contador usado para sincronização do primeiro bit da palavra recebida teve seu valor atualizado de 25 para 17. Por algum motivo, com o valor de 25, a sincronização não acontecia corretamente; logo, foi necessário esse ajuste fino. Também foi adicionado ao bloco um processo que realiza uma amostragem dupla dos dados de entrada para evitar problemas de metaestabilidade, pois os dados são recebidos a 100kbps (200kHz) e o *clock* global atua em 10MHz.

A figura 4.1 apresenta a simulação completa feita para o bloco de CDR. Como o *clock* global de 10MHz possui um período muito pequeno em relação ao tempo de simulação (0.8ms), não é possível ver em detalhes a saída de dados e o sinal de ready e rd_en. Para conseguir visualizá-los, utilizou-se de uma aproximação visual no momento em que a palavra de saída é mostrada (figura 4.2). Nessa simulação, foi enviada a palavra ARINC 429 em hexadecimal F0F0F0FB. Após o sinal ready sinalizar que o bloco terminou sua função, é possível ver essa palavra na saída de dados no momento em que a leitura do bloco é habilitada através do sinal de rd_en. O comportamento mostrado na imagem se encontra de acordo com o esperado.

Figura 4.1: Simulação - CDR

Fonte: Imagem gerada através do software ModelSim.

Figura 4.2: Detalhe da saída do bloco CDR



Fonte: Imagem gerada através do software ModelSim.

4.2 Parity Checker

Diversos testes são feitos dentro do *test bench* do bloco de paridade. Eles serão explicados de acordo com a análise de cada requisito apresentada abaixo.

- *[P01] A interface deve prover meios de verificar a paridade da palavra de 32 bits recebida, isto é, verificar se a palavra está corrompida ou não.*
- *[P02] A paridade deve ser do tipo ímpar conforme a norma ARINC 429.*
- *[P04] Se a paridade da palavra for ímpar, a palavra está correta; logo, a saída de dados do bloco deve receber essa palavra.*
- *[P05] O bloco deve prover meios de informar se a palavra está ou não corrompida ao bloco de controle.*
- *[P06] O bloco deve prover meios de informar se o processo de verificação da paridade foi terminado ou não ao bloco de controle.*
- *[P10] Deve existir um sinal de habilitação do bloco.*

  Para provar os requisitos P01, P02, P04, P08, P09 e P10, implementou-se um teste que carrega uma palavra com paridade ímpar, portanto uma palavra válida, e verificou-se se o bloco é capaz de interpretá-la como válida. Se o bloco interpretar essa palavra como válida, fica provado que o sistema é capaz de verificar que palavras de 32 bits com paridade ímpar não estão corrompidas ([P01] e [P02]). Adicionalmente, se a saída de dados receber essa palavra e o sinal ready mudar seu estado, indicando que a operação foi concluída, [P04] e [P06] também ficam provados. Aditivamente, espera-se que o estado de data_crpt, sinal que informa se a palavra está corrompida ou não, fique em "0"pois a palavra é válida ([P05]). Por último, se esse teste for iniciado por meio de um sinal de habilitação (*enable*), [P10] também é provado. Para aumentar a confiabilidade, dois testes com duas palavras diferentes com paridade ímpar foram feitos. O teste descrito prova que o sistema funciona para palavras válidas. Entretanto, para que o bloco seja provado de fato, é necessário alimentar o bloco com palavras inválidas. Esse é o contexto da próxima análise. Embora os mesmos requisitos sejam analisados para palavras inválidas, o requisito [P03] apresentado a seguir, também é incluído na análise.

- *[P03] Se a paridade da palavra não for ímpar, a palavra está incorreta; logo, a saída de dados do bloco não deve receber essa palavra.*

  Para provar que o bloco é capaz de interpretar palavras não válidas, isto é, pala-

vras corrompidas, desenvolveu-se um teste que alimenta o bloco com uma palavra inválida (paridade par). Para aumentar a confiabilidade, o teste foi repetido duas vezes com palavras inválidas diferentes. Se o bloco interpretar essas palavras como inválidas, fica provado que o sistema é capaz de verificar que palavras de 32 bits com paridade par estão corrompidas [P01] e [P02]. Adicionalmente, se a saída de dados não receber essa palavra, isto é, ficar em nível lógico *LOW*, o requisito [P03] é provado. Também, espera-se que o estado de data_crpt fique em '1' pois a palavra não é válida ([P05]).

- *[P09] Se o bloco receber uma palavra com todos os bits em nível lógico LOW, a saída de dados deve ficar estado lógico LOW e a interface não deve sinalizar palavra corrompida.*

  Para provar esse requisito, se desenvolveu um teste que carrega no bloco uma palavra composta inteiramente de bits em nível lógico *LOW* e observou-se a saída de dados e o sinal que informa se a palavra está corrompida ou não. Se a saída de dados permanecer em "0"e o sinal que informa que a palavra está corrompida não mudar seu nível lógico, isto é, permanecer em "0", acusando que não há palavra corrompida, o requisito está provado.

- *[P07] Deve ser possível resetar o bloco.*
- *[P08] No estado de reset, a saída do bloco deve ficar em estado lógico LOW.*

  Para provar esses dois requisitos, se desenvolveu um teste que reseta o bloco e verificou-se se a saída fica em nível lógico *LOW*.

A tabela 4.2 ilustra os resultados obtidos utilizando os testes citados para esse bloco.

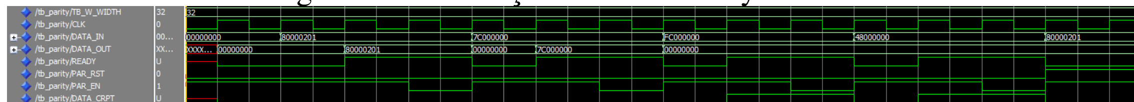
Tabela 4.2: Resultado dos testes do bloco Parity Checker

Requisitos	Resultado
P01	OK
P02	OK
P03	OK
P04	OK
P05	OK
P06	OK
P07	OK
P08	OK
P09	OK
P10	OK

Fonte: Autoria própria.

A figura 4.3 ilustra a simulação dos testes citados. É possível ver que as duas primeiras palavras em hexadecimal (80000201 e 7C000000) são palavras válidas; logo, o sinal data_crpt permanece em "0"e as palavras são transferidas para a saída (data_out). Já para as duas palavras seguintes, FC000000 e 48000000, o sinal data_crpt muda seu estado para "1"pois as palavras são inválidas. Portanto, elas não aparecem na saída. Por fim, o teste de *reset* é apresentado.

Figura 4.3: Simulação do bloco Parity Checker



Fonte: Imagem gerada através do software ModelSim.

4.3 FIFO

Os testes realizados para os requisitos do bloco FIFO são explicados abaixo.

- *[FIFO01] A interface receptora ARINC 429 deve ser capaz de armazenar as palavras recebidas com paridade correta em um buffer do tipo FIFO.*
- *[FIFO02] O buffer do tipo FIFO deve ser capaz de armazenar até 4 palavras de 32 bits.*
- *[FIFO03] Deve ser possível ler e escrever no FIFO usando sinais de controle de leitura e escrita.*
- *[FIFO05] Enquanto o FIFO não estiver sendo lido, sua saída de dados deve permanecer em estado LOW.*
- *[FIFO06] O FIFO deve prover meios de informar o bloco de controle se está cheio*

ou vazio.

Para provar esses requisitos, pode-se realizar um único teste. Esse teste consiste em escrever 4 palavras diferentes e, após a escrita, realizar a leitura dessas 4 palavras. Se o FIFO for capaz de armazenar essas 4 palavras de 32 bits (ARINC 429), os requisitos [FIFO01] e [FIFO02] são comprovados. Adicionalmente, se a escrita e a leitura das palavras nesse FIFO forem ditadas por sinais de controle, no caso rd_en para leitura e wr_en para escrita, o requisito [FIFO03] é provado. Também é analisado o comportamento dos sinais fifo_full e fifo_empty. Quando o FIFO está vazio, espera-se que o sinal fifo_empty assuma o valor "1"e quando não estiver vazio, o valor "0". O mesmo comportamento se espera para o sinal fifo_full, porém com os estados de FIFO cheio e FIFO não cheio. Observando o comportamento desses sinais, o requisito [FIFO06] pode ser provado. Outro ponto a se observar nesse teste é a saída do FIFO enquanto o FIFO não é lido; se ela permanecer em nível lógico "0", o requisito [FIFO05] é provado.

- *[FIFO04] Deve ser possível ler e escrever ao mesmo tempo no FIFO.*

  A fim de provar esse requisito, desenvolveu-se um teste que lê e escreve no FIFO ao mesmo tempo. Basicamente, nesse teste, após a escrita de duas palavras iguais, tenta-se realizar uma nova escrita com uma palavra diferente ao mesmo tempo que tenta-se realizar uma leitura. Se a leitura que ocorre durante a escrita for feita corretamente, isto é, se a palavra escrita anteriormente for transferida para a saída de dados, é provado que a leitura ocorreu corretamente. Após isso, realizasse a leitura das palavras restantes no FIFO para averiguar se a escrita feita durante a leitura foi de fato realizada com sucesso.

- *[FIFO07] Uma tentativa de ler o FIFO estando ele vazio deve retornar estado LOW na saída de dados.*

  Com intuito de provar esse requisito, se desenvolveu um teste que tenta ler o FIFO quando ele está vazio e observou-se a saída de dados. Se ela se manter em estado lógico *LOW*, o requisito é provado.

- *[FIFO08] Uma escrita no FIFO, estando ele cheio, não deve ser possível, mantendo ele assim suas palavras originais.*

  A fim de provar esse requisito, criou-se um teste que primeiramente envia 4 palavras iguais para preencher o FIFO. Após isso, tenta-se escrever uma quinta palavra diferente das demais enviadas anteriormente. Então, o FIFO é lido inteiramente. Se as 4 palavras lidas forem as mesmas palavras enviadas inicialmente sem a substi-

tuição de uma delas pela quinta palavra escrita, quer dizer que a quinta palavra não foi escrita; logo, o requisito está provado.

- *[FIFO09] Deve ser possível resetar o FIFO.*
- *[FIFO10] No estado de reset, o FIFO deve ficar vazio e a saída de dados em estado LOW.*
  O teste para provar esse requisito consiste em resetar o FIFO e verificar se o FIFO se encontra vazio e a saída de dados em estado lógico *LOW*.

A tabela 4.3 ilustra os resultados obtidos utilizando os testes citados para esse bloco.

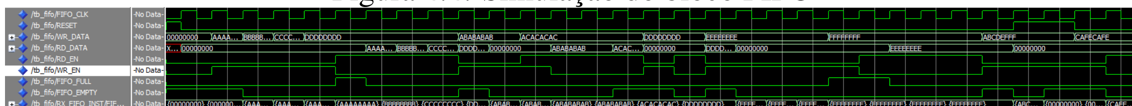
Tabela 4.3: Resultado dos testes do bloco FIFO

Requisitos	Resultado
FIFO01	OK
FIFO02	OK
FIFO03	OK
FIFO04	OK
FIFO05	OK
FIFO06	OK
FIFO07	OK
FIFO08	OK
FIFO09	OK
FIFO10	OK

Fonte: Autoria própria.

A figura 4.4 ilustra a simulação que apresenta os testes citados acima.

Figura 4.4: Simulação do bloco FIFO



Fonte: Imagem gerada através do software ModelSim.

4.4 CPU IF

Os testes realizados para os requisitos do bloco CPU IF são explicados abaixo.

- *[CPU02] O bloco deve prover meios de configurar o endereço da instalação receptora (LRU).*
  Como o requisito não entra em detalhes de como configurar o endereço da instalação receptora, simplesmente criou-se dois sinais de entrada de um bit para a

configuração. Esses sinais, na verdade, são os bits de SDI da LRU. Para provar esse requisito, basta ser possível configurar esses dois sinais. Desse modo, o bloco fornece meios de configurar o endereço da LRU (SDI bits). Para os testes seguintes, esses bits foram configurados com o endereço 01.

- *[CPU01] O bloco deve prover meios de interfacear o bloco FIFO com um processador que possui barramento de dados de 8 bits, possibilitando assim a leitura da palavra recebida pelo processador.*
- *[CPU03] O bloco deve prover meios de verificar se os bits de SDI (bit 9 e 10) da palavra recebida correspondem aos bits de identificação da LRU receptora (requisito [CPU02]).*
- *[CPU04] Se os bits de SDI corresponderem ao endereço da LRU, a saída do bloco deve receber a palavra A429.*
- *[CPU06] Como o processador que realiza a interface possui barramento de 8 bits e a palavra A429 possui 32 bits, os bytes que formam a palavra A429 devem ser multiplexados no barramento de dados do processador.*
- *[CPU07] A cada byte recebido pelo processador, o processador enviará um sinal confirmando o recebimento; o byte seguinte só deve ser enviado pelo bloco após a confirmação do recebimento pelo processador.*

  Para provar esses requisitos, desenvolveu-se um teste que carrega uma palavra na entrada de dados (data_in) que contém os bits de SDI válidos, isto é, os bits de SDI 01, conforme configurados no teste descrito acima. Então, simulou-se que o processador solicitou a realização de uma leitura, portanto o sinal de rd_en recebeu "1". Se o bloco for capaz de entender que os bits de SDI são válidos, enviando assim a palavra de forma multiplexada conforme o recebimento do sinal de ACK do processador por *byte* enviado, condizente com a descrição da seção 3.3.4, os requisitos acima são provados. Para aumentar a testabilidade do requisito [CPU07], simulou-se também o cenário onde o processador não envia o sinal de confirmação do recebimento. Com isso, espera-se que o sistema mantenha o mesmo *byte* na saída até que o processador envie o sinal de confirmação do recebimento.
- *[CPU05] Se os bits de SDI não corresponderem ao endereço da LRU, a saída deve permanecer em nível lógico LOW.*

  Para provar esse requisito, criou-se um teste que carrega uma palavra na entrada de dados (data_in) que não contém os bits de SDI válidos; logo, se a saída permanecer em nível lógico *LOW*, o requisito é provado.

- *[CPU08] A palavra só deve ser transferida ao processador se requisitada; caso contrário, a saída deve ficar em estado lógico LOW.*

  O teste desenvolvido para provar esse requisito consiste na simulação de que o processador não enviou a solicitação de leitura (rd_en = 0); logo, se a saída permanecer em "0", uma parte do requisito está provado. A outra parte foi provada no teste explicado anteriormente, onde existe uma requisição de leitura por parte do processador.

- *[CPU09] Deve ser possível resetar o bloco.*
- *[CPU10] No estado de reset, a saída do bloco deve ficar em estado lógico LOW.*

  O teste para provar esses requisitos consiste em resetar o bloco e verificar se a saída fica em estado lógico *LOW*.

A tabela 4.4 ilustra os resultados obtidos utilizando os testes citados para esse bloco.

Tabela 4.4: Resultado dos testes do bloco CPU IF

Requisitos	Resultado
CPU01	OK
CPU02	OK
CPU03	OK
CPU04	OK
CPU05	OK
CPU06	OK
CPU07	OK
CPU08	OK
CPU09	OK
CPU10	OK

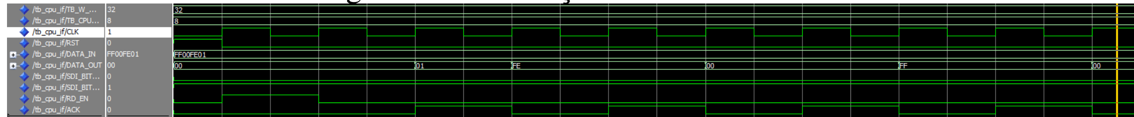
Fonte: Autoria própria.

Existe uma situação diferente na verificação desse bloco. A simulação feita não consegue de fato provar o requisito [CPU01] e a função geral do bloco. O que a simulação prova é que o comportamento do bloco está de acordo com o esperado, ou seja, os requisitos estão implementados em lógica. Entretanto, para realmente provar o requisito [CPU01] e a função do bloco, que é realizar a interface com um processador de 8 bits, seria necessário implementar essa lógica em um FPGA e de fato interfacear o FPGA com um processador de 8 bits que seja capaz de responder o protocolo apresentado. O mesmo se aplica ao requisito [CPU07]. O diferencial nesse bloco é que existe uma dependência externa, no caso, o processador que precisa estar sincronizado com a lógica do bloco para enviar o sinal ACK no momento certo. Esse detalhe faz com que a simulação não tenha um grau de confiabilidade alto como ocorre nos outros blocos. Na seção 4.6, essa questão

será abordada de forma mais ampla.

A figura 4.5 apresenta a simulação feita com base nos testes citados anteriormente. É possível ver que a palavra de entrada FF00FE01 em hexadecimal é transferida para a saída *byte* por *byte* conforme o sinal de ACK do processador.

Figura 4.5: Simulação do bloco CPU IF



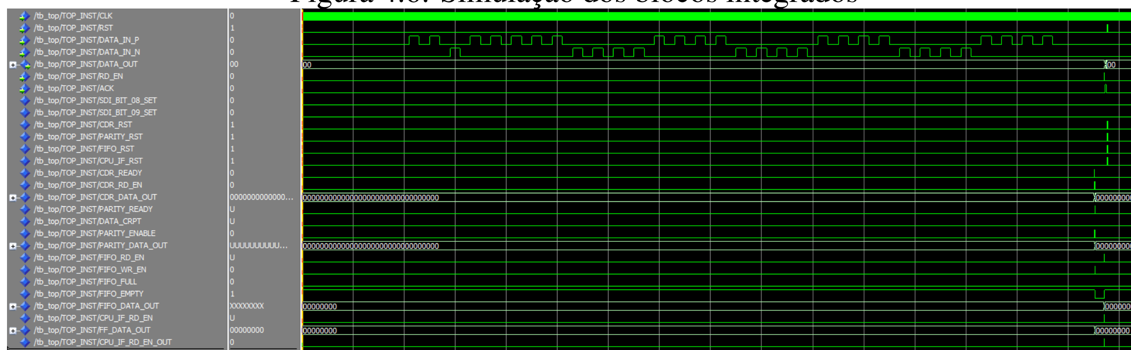
Fonte: Imagem gerada através do software ModelSim.

4.5 Integração

Nas seções anteriores, foi descrito a estratégia de verificação para provar os requisitos por bloco. Uma vez que foi verificado que os requisitos estão de fato implementados em lógica de descrição de *hardware* e funcionando de acordo o esperado, a etapa de integração teve início. Essa etapa consiste em conectar todos os blocos citados anteriormente para que funcionem de acordo com o bloco de controle. O resultado dessa integração consiste na lógica final do bloco.

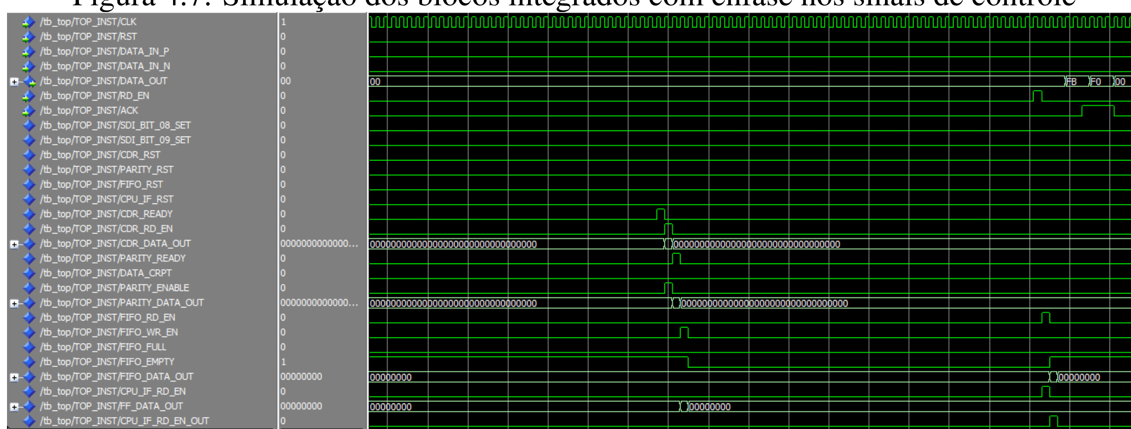
Para verificar que os blocos integrados funcionam de fato como deveriam, desenvolveu-se um teste em que uma palavra ARINC 429 é enviada (F0F0F0FB em hexadecimal), recebida, armazenada e transferida para o processador. O resultado foi conforme o objetivo inicial: o bloco foi capaz de receber, armazenar e enviar a palavra pronta para ser processada. A figura 4.6 apresenta essa simulação. Nessa figura apenas mostrou-se os sinais externos de cada bloco. Os sinais internos foram ocultados para evitar excesso de informação na imagem. A figura 4.7 ilustra os sinais de controle dos blocos em mais detalhe. Nessa figura, é possível visualizar que a palavra F0F0F0FB é multiplexada na saída *byte* por *byte* conforme o sinal de ACK (sinal DATA_OUT na figura 4.7), indicando o correto funcionamento do sistema.

Figura 4.6: Simulação dos blocos integrados



Fonte: Imagem gerada através do software ModelSim.

Figura 4.7: Simulação dos blocos integrados com ênfase nos sinais de controle



Fonte: Imagem gerada através do software ModelSim.

4.6 Discussões

Na seção de verificação do bloco CPU IF, foi levantada a questão de que a simulação desse bloco não seria suficiente para provar alguns requisitos e a funcionalidade do bloco. Isso se deve ao fato de que o bloco CPU IF tem dependências externas, como é o caso do processador. Nesse caso, seria necessário interfacear o bloco com um processador de 8 bits para provar de fato que o bloco é capaz de se comunicar com um processador. A simulação, como foi comentado anteriormente, prova que o bloco possui os requisitos necessários para interagir com um processador implementados, mas não prova que ele é capaz de fazer isso. Entretanto, se esse raciocínio for aplicado para o sistema como um todo, chegar-se-à conclusão de que apenas a simulação não é o suficiente para comprovar que a lógica realmente funciona como deveria. E isso, de forma alguma, inválida o que foi feito no presente trabalho. As simulações realizadas comprovam que o comportamento ditado pelos requisitos está implementado em lógica. Para esse trabalho, isso é o suficiente. Entretanto, se essa lógica realmente buscasse a certificação, além de implementar e executar mais testes no âmbito da simulação para cobrir as mais diversas possibilida-

des do sistema, o próximo passo seria implementar essa lógica em um FPGA e realizar um teste de *Proof of Design* (POD). Testes de POD são testes formais que visam testar na prática a implementação. Nesse caso, um gerador de pacotes ARINC 429, como o ES-429TK-101, da Excalibur Systems (2021), seria colocado para fornecer as palavras ARINC 429 ao sistema. O sistema seria também interfaceado com um processador de 8 bits rodando um *software* que seja capaz de reconhecer o protocolo descrito no bloco CPU IF.

5 CONCLUSÃO

No presente trabalho, desenvolveu-se um projeto simplificado que aborda o contexto de sistemas eletrônicos usados na aviação (aviônica). Aspectos como nomenclaturas, processos e análises foram cobertos ao longo da monografia. Em especial, foi apresentada a interface ARINC 429 e o processo de desenvolvimento de *hardware* DO-254. Dessa maneira, através da leitura desta monografia, é possível adquirir noções básicas sobre os requisitos de projetos em aviônica. Cumpre-se assim, o primeiro dos objetivos do trabalho.

Além de fornecer noções básicas de aviônica, o ponto central do trabalho consistia em desenvolver a lógica de um receptor ARINC 429 tendo como base noções do processo de desenvolvimento de *hardware* DO-254. Com base no processo da DO-254, a ideia do projeto foi sendo desenvolvida e detalhada a medida que as etapas da DO-254 eram cumpridas. A cada etapa, o *design* foi ganhando complexidade e detalhamento. Começando com a etapa de planejamento, pôde-se definir uma arquitetura simples do sistema e determinar a abordagem utilizada no projeto. Essas considerações da etapa de planejamento serviram de "entrada"para a escrita dos requisitos. Para escrever os requisitos, considerou-se dois elementos: a lógica que o requisito implementaria e sua testabilidade. Com isso, a captura de requisitos foi realizada. A etapa de captura de requisitos representou uma etapa de suma importância no trabalho, pois a partir dos requisitos, o desenvolvimento da arquitetura do sistema tornou-se possível na etapa de *design* conceitual. Adicionalmente, os requisitos também serviram de guia para a confecção dos testes da etapa de verificação. Uma vez definida a arquitetura do sistema na etapa de *design* conceitual, o código foi implementado em VHDL na etapa de *design* detalhado. Devido à execução dessas etapas, pode-se concluir que a metodologia apresentada nesse trabalho foi condizente com a norma DO-254.

Para verificar o sistema, adotou-se a estratégia de utilizar uma metodologia de verificação funcional fazendo o uso de testes escritos em VHDL. Para cada requisito escrito, foi criado um teste que pretendia provar que o requisito está de fato implementado em lógica, funcionando de acordo com sua descrição. Com os testes desenvolvidos, foi possível provar, através de simulações, que o sistema se comportou como os requisitos solicitavam, ou seja, os requisitos estavam implementados em lógica; portanto, pode-se afirmar que foi desenvolvida a lógica de um receptor ARINC 429. Entretanto, como foi mencionado na seção 4.6, para de fato provar que o sistema funciona como especificado, é

necessário realizar testes com a lógica desenvolvida implementada em algum dispositivo de lógica programável, como um FPGA por exemplo, para que seja possível interfacear o sistema desenvolvido com um transmissor ARINC 429 e um processador de 8 bits, conforme estabelecem os requisitos das seções 3.2.1 e 3.2.4. Desse modo, o sistema seria testado na prática. Portanto, para um sistema em que se deseja atingir a certificação e embarcar-se em uma aeronave, é necessário que sejam executados testes com os sistemas adjacentes a esse desenvolvido.

Caso deseje-se, em trabalhos futuros, alcançar a certificação do sistema desenvolvido, alguns aspectos precisam ser observados. Primeiramente, precisa-se saber o quão crítica é a operação em que o receptor ARINC 429 irá ser usado. Com base nisso, pode-se definir um DAL para o sistema. Aditivamente, mais testes a nível de simulação precisam ser feitos. Simulações abordando questões relacionadas à análise de tempo e simulações funcionais utilizando metodologias de verificação como UVM, podem ser feitas. Essas simulações adicionais devem ser feitas tendo como base os requisitos já escritos. Após isso, a etapa de implementação, onde é realizada a síntese lógica e o processo de *place and route*, pode ser executada. Logo em seguida, testes utilizando geradores de mensagens ARINC 429 devem ser executados já utilizando os sistemas adjacentes (testes de POD), como processadores por exemplo. Por fim, a integração deve ser feita com a LRU em que pretende-se utilizar o receptor. Uma vez cumpridos e documentados esses aspectos adicionais, o receptor poderá ser submetido ao processo de certificação.

ANEXO A - TABELA ISO ALPHABET NUMBER 5

Figura 5.1: Tabela ISO Alphabet Number 5

BIT 7 → BIT 6 → BIT 5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
BIT 4 ↓	BIT 3 ↓	BIT 2 ↓	BIT 1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	•	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

Fonte: AEEC (2004).

APÊNDICE A - CABOS ARINC 429

Para se conectar um transmissor em um receptor, deve ser usado um par de fios trançados e blindado com uma impedância diferencial de 70 a 80 Ohms (AEEC, 2004). Um transmissor pode se comunicar com até 20 receptores (AEEC, 2004).

REFERÊNCIAS

ACTEL. **Arinc 429 Bus Interface Datasheet**. 2006. Disponível em: <https://web.archive.org/web/20091007185947/http://www.actel.com/ipdocs/CoreARINC429_DS.pdf>.

AEEC. Arinc specification 429 part 1-17: Mark33 digital information transfer system (dits). ARINC, 2004.

ALDEC. **Developing high-reliability FPGAs for DO-254**. 2021. Disponível em: <https://www.aldec.com/en/company/blog/102--developing-high-reliability-fpgas-for-do-254>.

ASHWARE. **ARINC 429 Transmitter-Receiver Details**. 2021. Disponível em: <https://www.ashware.com/arinc-429-transmitter-receiver-details>.

BUTKA, B. Advanced verification methods for safety-critical airborne electronic hardware. FAA, 2015.

CADENCE. **DO-254 Explained**. 2019. Disponível em: <https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/aerospace-and-defense/do-254-explained-wp.pdf>.

EXCALIBUR SYSTEMS. **ARINC 429 Tester**. 2021. Disponível em: <https://www.mil-1553.com/429-tester>.

LEONARDO AEROSPACE. **Avionics Network Computing (ANC) Platform**. 2021. Disponível em: <https://www.leonardocompany.com/en/products/anc?f=/air/airborne-systems/avionics-and-cni>.

LUNA, L. D.; ZALEWSK, Z. Fpga level in-hardware verification for do-254 compliance. IEEE, 2011.

NATIONAL INSTRUMENTS. **What is DO-254?** 2021. Disponível em: <https://www.ni.com/pt-br/innovations/white-papers/11/what-is-do-254-.html>.

RTCA; EUROCAE. Do-254 - design assurance guidance for airborne electronic hardware. 2000.

SDA. **congatec Application Note**. 2020. Disponível em: <https://www.congatec.com/fileadmin/user_upload/Documents/Application_Notes/AN45_PCI_Express_Reference_Clock_Design_Considerations.pdf>.

SPITZER, C. R. The avionics handbook. CRC Press LLC, 2001.

SYNOPSYS. **What is Static Timing Analysis (STA)?** 2021. Disponível em: <https://www.synopsys.com/glossary/what-is-static-timing-analysis.html>.