UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Hiago Murilo Horstmann

Especificação e Projeto de Blocos Digitais para um Módulo Transmissor USB em VHDL

Florianópolis
2020

Hiago Murilo Horstmann

Especificação e Projeto de Blocos Digitais para um Módulo Transmissor USB em VHDL

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Elétrica.
Orientador: Prof. Walter Pereira Carpes Júnior, Dr.

Florianópolis
2020

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

Horstmann, Hiago Murilo
Especificação e Projeto de Blocos Digitais para um Módulo Transmissor USB em VHDL / Hiago Murilo Horstmann ; orientador, Walter Pereira Carpes Júnior, 2020.
63 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2020.

Inclui referências.

1. Engenharia Elétrica. 2. Blocos Digitais. 3. Módulo Transmissor. 4. Universal Serial Bus. I. Carpes Júnior, Walter Pereira. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Hiago Murilo Horstmann

Especificação e Projeto de Blocos Digitais para um Módulo Transmissor USB em VHDL

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Elétrica" e aprovado, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 26 de novembro de 2020.

Documento assinado digitalmente
Jean Vianei Leite
Data: 30/11/2020 15:04:36-0300
CPF: 003.474.909-80

Prof. Jean Vianei Leite, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Documento assinado digitalmente
Walter Pereira Carpes Junior
Data: 29/11/2020 18:35:13-0300
CPF: 572.566.599-20

Prof. Walter Pereira Carpes Júnior, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Jean Vianei Leite
Data: 30/11/2020 15:05:21-0300
CPF: 003.474.909-80

Prof. Jean Vianei Leite, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Renato Lucas Pacheco
Data: 27/11/2020 17:54:02-0300
CPF: 341.751.489-49

Prof. Renato Lucas Pacheco, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Fernando Effting da Silva
Data: 27/11/2020 13:57:38-0300
CPF: 079.304.779-08

Eng. Eletric. Fernando Effting da Silva, Me.
Fruitkeep

Aos meus pais, alicerces da minha essência, fontes de amor e inspiração, meus maiores exemplos.

AGRADECIMENTOS

Primeiramente, à minha família, em especial aos meus pais, Maria Janete Folster Horstmann e Sérgio Murilo Horstmann, por todo apoio e estímulo que me deram, me influenciando a seguir neste caminho e não medindo esforços para que eu tivesse as melhores condições possíveis de obter esta conquista. Sem eles eu não teria chegado aonde estou. E à minha irmã, Karolyne Horstmann, por toda a ajuda dedicada à mim, principalmente ao meu bem-estar em horas difíceis de desabafos e frustrações. Uma pessoa com quem eu sei que sempre poderei contar.

Aos meus amigos da graduação, Arthur Merlo, Eduardo Steffens, Felipe Zimmermann Severino, Gabriel Piazera Hessman, Indiara Pitta Correa da Silva, Jaqueline Clamer, Lucas Comelli, Matheus Schlischting Ferreira, Unirio Machado dos Santos, Willian Jamir da Silva e tantos outros que conheci ao longa da graduação, companheiros de muitos momentos de dificuldades, mas também de diversão e descontração, dentro e fora da Universidade. Em especial, agradeço ao Guilherme Luis Medeiros que, além de tudo, também contribuiu diretamente na construção deste trabalho.

À toda a família do meu tio, Jaime Pedro Folster, que sempre me acolheu, evitando muito tempo desperdiçado nessa longa jornada de casa até a Universidade e me deram um segundo lar.

À Universidade Federal de Santa Catarina, em especial aos professores e funcionários do Departamento de Engenharia Elétrica e Eletrônica, por um curso de qualidade com todo o conhecimento transmitido e serviços prestados.

Ao meu orientador Prof. Walter Pereira Carpes Júnior e aos coordenadores Prof. Renato Lucas Pacheco e Prof. Jean Vianei Leite, pelo suporte para este trabalho, assim como toda empatia e compreensão das situação adversas.

Por fim, a todos que apoiaram e contribuíram de alguma forma para a minha formação acadêmica, o meu muito obrigado.

"Toda conquista começa com a decisão de tentar."
(Gail Devers)

RESUMO

A transmissão e armazenamento de informação por meio digital possuem algumas vantagens, tais como, maior imunidade ao ruído; ausência da deterioração da qualidade da informação; e tratamento de dados por técnicas computacionais de processamento digital de sinais. Por isso, o Barramento Serial Universal (USB), além de ser um sistema de comunicação que permite a conexão de dispositivos periféricos a um *Host* (computador), é uma das interfaces mais populares atualmente, com larga escala de aplicações. Tanto as interfaces do *Host*, quanto as de dispositivos USB, podem ser divididas em três camadas. Este trabalho focou na camada física de dispositivos, especificamente na construção dos blocos digitais de transmissão, que fazem parte do protocolo USB gerenciados pelo Mecanismo de Interface Serial - SIE. O módulo transmissor é responsável por enviar dados do dispositivo para o cabo USB. Pesquisou-se sobre funcionamento e comportamento dos circuitos internos de um transmissor USB, criou-se diagramas, tabelas de codificação de estados e tabelas de transições de estados para suportar os circuitos especificados e gerar o circuito digital dos blocos, buscando a divisão explícita em *Datapath* e *Control Path*. Cada circuito que compõe o transmissor - de conversão paralelo-serial, *bit-stuffing*, codificação NRZI, conversão diferencial e controle - foi descrito em VHDL. Por fim, foi realizado o agrupamento e simulação em conjunto dos blocos validando o comportamento destes.

Palavras-chave: Blocos Digitais, Módulo Transmissor, Universal Serial Bus.

ABSTRACT

Data transmission and storage through digital media has some advantages such as: better noise shielding, lack of data integrity degradation, and data processing through digital signal processing computational techniques. The Universal Serial Bus (USB), which is a communication system that allows peripheral devices to communicate to a host (computer), is one of the most used interfaces up to date, with a large spectrum of applications. Both host and USB devices are categorized in three layers. This work is focused on the device's physical layer, specifically on building the digital transmission blocks, which are part of the USB protocol managed by the Serial Interface Engine (SIE). The transmission module is responsible for sending the device's data to the USB cable. One researched the operation and behavior of internal circuits in a USB transmitter, created diagrams, codification of states and state transition tables to back up those specificated circuits and to generate the block's digital circuits, aiming for the explicit separation between Datapath and Control Path. Each circuit that constitutes the transmitter - parallel-to-serial converter, bit-stuffing, NRZI encoder, differential encoder and control - were described in VHDL. Finally, all the circuits were put together and simulated, validating their behavior.

Keywords: *Digital Blocks, Transmissor Module, Universal Serial Bus.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CDR	*Clock and Data Recovery*
EOP	*End Of Packet*
FPGAs	*Field Programmable Gate Arrays*
GND	*Ground*
I2C	*Inter-Integrated Circuit*
NAK	*Negative Acknowledgment packet*
NRZI	*Non Return to Zero Inverted*
PID	*Packet Identifier*
RS-232	*Recommended Standard 232*
RTL	*Register-Transfer Level*
SE0	*Single-Ended Zero*
SE1	*Single-Ended One*
SIE	*Serial Interface Engine*
USB	*Universal Serial Bus*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, o modo como vivemos tem sofrido constantes transformações causadas pela evolução da comunicação por vias digitais. Com a internet, a troca ou o acesso à informação disponível se tornou mais prática e rápida (FOROUZAN, 2009) e tem proporcionado um desenvolvimento tecnológico cada vez mais veloz.

A transmissão e/ou armazenamento de informação por meio digital possuem algumas vantagens, tais como, maior imunidade ao ruído gerado por interferências causadas durante a transmissão; ausência da deterioração da qualidade da informação armazenada ao longo do tempo; e tratamento de dados por técnicas computacionais de processamento digital de sinais (RUSCHEL, 1996). Por estas razões que, apesar da grande maioria dos sinais ser de natureza analógica (imagens, sons e outros), cada vez mais dispositivos convertem os sinais analógicos captados em sinais digitais, antes de qualquer processamento ou transmissão.

A comunicação digital pode ser paralela ou serial. Na transmissão paralela, considerando dados binários, vários bits podem ser enviados por vezes a cada ciclo de *clock* (relógio). Normalmente, para $\boldsymbol{n}$ bits são necessários $\boldsymbol{n}$ fios, ou seja, um canal para cada bit.

Na transmissão serial, os bits são enviados um após o outro, a cada ciclo de *clock*, necessitando, então, de apenas de um canal para transmissão. Ou seja, o custo e o número de pinos utilizados na transmissão serial é muito menor em relação à transmissão paralela. Por exemplo, em um *chip*, onde a quantidade de pinos é limitada, a transmissão serial, apesar de ter uma complexidade maior (SZECÓWKA; PYRZYNSKI, 2012), é vantajosa em relação à paralela.

Em sistemas de comunicação, a padronização é essencial (LECHEMINANT, 2002), pois assegura a interoperabilidade. Diversos padrões de comunicação serial são conhecidos, como *Recommended Standard 232* (RS-232), *Inter-Integrated Circuit* (I2C) e *Universal Serial Bus* (USB).

O Barramento Serial Universal (USB), além de ser um sistema de comunicação que permite a conexão de dispositivos periféricos a um *Host* (computador), utilizando quatro condutores, é uma das interfaces mais populares atualmente, com larga escala de aplicações (AXELSON, 2005).

Tanto as interfaces do *Host*, quanto as de dispositivos USB, podem ser divididas em três camadas. Este trabalho focou na camada física de dispositivos, especificamente na construção dos blocos digitais de transmissão, que fazem parte do protocolo USB, gerenciados pelo Mecanismo de Interface Serial - *Serial Interface Engine* (SIE) (ANDERSON; DZATKO, 2001).

Nos requisitos do USB, o barramento possui condutores para o envio apenas de sinais de dados. Para que exista o funcionamento correto, é necessário que o dispositivo

possua, no bloco receptor, um método de sincronizar os dados recebidos sem *clock*, por exemplo, um *Clock and Data Recovery* (CDR).

O módulo transmissor ou bloco de transmissão é responsável por enviar dados do dispositivo para o cabo USB. Os dados paralelos do dispositivo passam inicialmente pelo bloco de conversão paralelo-serial, em seguida, passam pelo bloco de *bit-stuffing* e, por fim, os dados seriais passam pelo bloco de codificação *Non Return to Zero Inverted* (NRZI) (MCGOWAN, 2001).

O bloco receptor é um bloco dual ao bloco de transmissão. Os dados passam primeiramente pelo CDR (SHIN *et al.*, 2013) e, depois de sincronizados, passam pelo bloco de decodificação NRZI. Em seguida, esses dados passam pelo bloco de *bit-unstuffing* e, por fim, os dados seriais passam pelo bloco de conversão serial-paralelo, sendo "entregues" paralelamente aos níveis lógicos mais altos do dispositivo.

Além de estudar os requisitos impostos pelo padrão USB, especificar, descrever as funções e simular os blocos do sistema, o presente trabalho teve como objetivo projetar blocos digitais que compõem um módulo transmissor USB em *VHSIC Hardware Description Language* (VHDL) - uma linguagem de descrição de *hardware*.

Realizou-se pesquisa bibliográfica sobre o funcionamento e comportamento dos circuitos internos de um transmissor USB em artigos e periódicos de base de dados como IEEE Xplore e nas especificações USB do site oficial `https://www.usb.org`. Criaram-se diagramas para suportar os circuitos especificados, tabelas de codificação de estados e tabelas de transições de estados, para gerar o circuito digital dos blocos, buscando a divisão explícita em *Datapath* e *Control Path*. Cada bloco foi descrito em VHDL através da ferramenta *Quartus II* e simulado através da ferramenta *ModelSim*. Finalizou-se com o agrupamento e simulação em conjunto dos blocos, compondo o módulo de transmissão.

A maior dificuldade no projeto foi a implementação do bloco de Controle, sendo necessária diligência nas simulações de integração dos blocos do módulo Transmissor devido ao fato de não haver um modelo predefinido. Apesar disso, obteve-se êxito no trabalho, uma vez que, ao final, o módulo Transmissor teve sua funcionalidade adequada, conforme os requisitos do sistema USB.

Para trabalhos futuros, sugere-se a especificação e projeto de blocos digitais para um módulo receptor USB, a implementação dos módulos Transmissor e Receptor em *Field Programmable Gate Arrays* (FPGAs) e, por fim, a integração do transmissor e receptor em um transceptor USB sintetizado e testado em circuito integrado.

1.1 OBJETIVOS

Esta seção traz os objetivos do trabalho, divididos em objetivo geral e objetivos específicos.

1.1.1 Objetivo Geral

- Projetar blocos digitais que compõem um módulo transmissor USB em VHDL.

1.1.2 Objetivos Específicos

- Estudar os requisitos impostos pelo padrão USB;
- Especificar os blocos do Sistema;
- Descrever as funções comportamentais e/ou estruturais dos blocos do sistema, usando linguagem de descrição de *hardware*;
- Realizar simulações de cada bloco individualmente e do sistema.

1.2 METODOLOGIA

A pesquisa teve como foco a especificação e o desenvolvimento de blocos digitais de um módulo transmissor USB em VHDL.

A metodologia adotada para a realização do trabalho deu-se primeiramente com maior obtenção de conhecimento sobre o funcionamento/comportamento dos circuitos internos de um transmissor USB. Para isso, foi realizada revisão bibliográfica sobre o tema, em artigos e periódicos de base de dados como IEEE Xplore e nas especificações USB do site oficial `https://www.usb.org`, através do link `https://www.usb.org/document-library/usb-20-specification`.

Para a execução deste estudo, foram criados diagramas para suportar os circuitos especificados. Para os diagramas mais simples, foram criadas tabelas de codificação e de transições de estados, através do diagrama de estados inicialmente proposto. A partir da tabela de transições, gerou-se o circuito digital do bloco de maneira manual, utilizando técnica de redução de circuito lógico com o método do mapa de Karnaugh. O circuito foi então descrito e sintetizado em VHDL com a ferramenta de desenvolvimento Quartus II.

Para diagramas mais complexos, o diagrama de estados proposto foi descrito em VHDL e a própria ferramenta Quartus II gerou o circuito do bloco utilizando a síntese *Register-Transfer Level* (RTL).

Buscou-se na construção dos blocos digitais e do sistema desenvolvido a divisão explícita em *Datapath* (caminho de dados) e *Control Path* (caminho de controle). Isso quer dizer que os blocos não trocam sinais de controle entre si, somente dados passam por eles (*Datapath*), e os sinais de controle (*Status* e Comandos) são de responsabilidade do bloco de controle.

Cada bloco, individualmente, foi descrito em VHDL e posteriormente simulado através da ferramenta *ModelSim* para verificar se havia o comportamento adequado do circuito. Quando o funcionamento do bloco não estava adequado, foi necessário reavaliar o

comportamento do mesmo, especificá-lo e descrevê-lo novamente, para então obter sucesso com a simulação.

Por fim, os blocos foram agrupados, compondo o módulo de transmissão, e simulados em conjunto para verificar o funcionamento adequado do módulo transmissor como um todo.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo é destinado à exposição dos conceitos mais importantes, de maneira geral, para o entendimento do trabalho.

2.1 USB E SUAS CAMADAS

O Barramento Serial Universal - USB é um sistema de comunicação que permite a conexão de dispositivos periféricos a um *Host* (computador) sem a necessidade de desligá-lo. Esses dispositivos periféricos tem a finalidade de adicionar funcionalidades específicas ao *Host*. Teclados, *mouses*, e *pen-drives* são exemplos desses dispositivos (AXELSON, 2005).

Em sua concepção, o barramento utiliza quatro condutores, sendo eles VBUS - que fornece alimentação ao dispositivo, *Ground* (GND) - referência e as linhas diferencias D− e D+ - meio de transmissão e recepção de dados durante a comunicação USB.

Tanto as interfaces do *Host*, quanto as de dispositivos USB, podem ser divididas em três camadas, sendo elas: Camada de Aplicação (*Aplication Layer*), Camada Link de Dados (*Data Link Layer*) e Camada Física (*Physical Layer*) (ANDERSON; DZATKO, 2001), como se pode observar na Figura 1.

Figura 1 – Camadas USB.

Fonte: Elaborada pelo autor.

A camada de Aplicação é a camada de mais alto nível e nela encontram-se os procedimentos que gerenciam e distribuem as informações apropriadas para uma função ou aplicação de um dispositivo (LUEKER *et al.*, 2000).

A camada Link de Dados faz a integração entre a camada de *software* (Camada de Aplicação) com a camada de *hardware* (Camada Física). Nela estão contidas as informações de operação dos dispositivos.

A camada Física ocupa-se com a transmissão e recepção de dados entre o *Host* e os dispositivos (LUEKER *et al.*, 2000). Ela é composta basicamente por um Transceptor (*Transceiver*) e pelo Mecanismo de Interface Serial (SIE). Este trabalho é focado na camada física de dispositivos, especificamente na construção dos blocos de transmissão.

2.2 MECANISMO DE INTERFACE SERIAL

O Mecanismo de Interface Serial (SIE) lida com o protocolo USB. Os módulos Transmissor e Receptor fazem parte do protocolo USB de baixo nível, onde se encontram blocos de funções tais como conversão paralelo-serial/serial-paralelo de dados, bit *stuffing/unstuffing* e codificação/decodificação NRZI (BABULU; RAJAN, 2008).

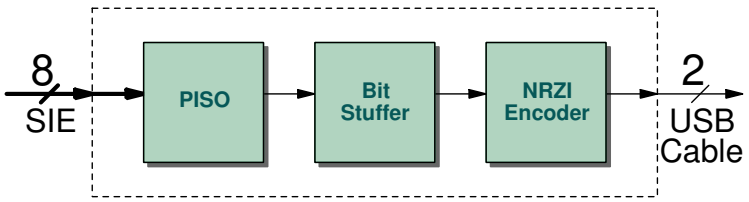
Nos requisitos do USB, o barramento possui condutores para o envio apenas de sinais de dados, sem um condutor para o sinal de *clock*, de modo que o número de condutores do sistema seja bem reduzido. Portanto, para que exista o funcionamento correto, é necessário que o dispositivo possua, no bloco receptor, um método de sincronizar os dados recebidos sem *clock* com a frequência de operação do seu próprio *clock*, como por exemplo, o circuito de *Clock and Data Recovery* (CDR) (SHIN *et al.*, 2013).

O CDR faz o sincronismo no início de cada recepção, após "extrair o *clock*" de uma sequência de transições (entre nível lógico alto e nível lógico baixo) recebida através do cabo USB. Basta que não ocorram longos períodos sem transições do sinal para que o CDR possa manter o sincronismo. Isso justifica a necessidade da codificação do sinal utilizada ser a NRZI, assim como a necessidade de existir o *bit stuffing*. A combinação dessas funções (codificação NRZI e *bit sutffing*) proporciona ao CDR as condições necessárias para funcionar adequadamente.

2.2.1 Transmissor

O módulo transmissor é responsável por enviar dados do dispositivo para o cabo USB. Para isso, os dados são "processados" por três blocos principais. Inicialmente, os dados paralelos do dispositivo passam pelo bloco de conversão paralelo-serial, onde são convertidos em dados seriais. Em seguida, os dados seriais passam pelo bloco de *bit-stuffing*, onde os dados com sequências longas de bits '1's são limitadas. Por fim, os dados seriais passam pelo bloco de codificação NRZI, onde são codificados e convertidos em dados seriais diferenciais (MCGOWAN, 2001). A Figura 2 ilustra de forma simples a composição do nódulo Transmissor.

Figura 2 – Bloco Transmissor.



Fonte: Adaptada de (SHIN *et al.*, 2013).

A seguir são descritas as funções dos blocos que compõem a Figura 2:

Paralelo-Serial - realiza a conversão paralelo-serial dos bits a serem enviados do dispositivo para o cabo USB. O bloco deve realizar aquisição paralela 8 bits de dados do dispositivo em sua entrada e deslocá-los serialmente em sua saída. Ao terminar o deslocamento de 8 bits, o bloco deve realizar a aquisição de dados novamente.

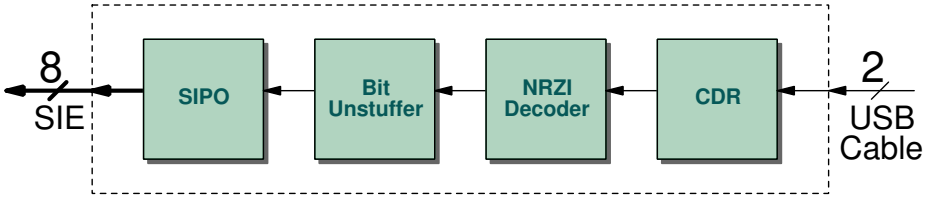
Bit-Stuffing - limita o número de consecutivos bits '1's, evitando a falta de transição no sinal transmitido, com a inserção forçada de um bit '0' após a detecção de uma sequência de seis '1's consecutivos. Na comunicação USB, sequências de dados com seis ou mais '1's consecutivos antes da codificação NRZI, acarretam em uma sequência muito longa sem transições durante a transmissão (pois a codificação NRZI converte '1's em "não-transições"), o que pode provocar a perda de sincronismo no CDR do receptor.

Codificação NRZI - A codificação NRZI converte um bit '0' de sua entrada em uma transição do valor da saída (se a saída é '0', vai para '1', ou vice-versa), enquanto que um bit '1' na entrada é convertido na "não transição" do valor de sua saída (se a saída é '0', mantém '0', se a saída é '1', mantém '1'). Por meio das transições presentes no sinal, o CDR sincronizará a taxa de dados (bits recebidos) com o *clock* de operação do receptor.

2.2.2 Receptor

O bloco receptor é um bloco dual ao bloco de transmissão, responsável pela admissão de dados na comunicação, porém com a adição extra do *Data and Clock Recovery* (CDR). Os dados passam primeiramente pelo CDR para que a taxa de dados seja sincronizada com o *clock* do receptor. Os dados (agora sincronizados) passam então pelo bloco de decodificação NRZI, onde são decodificados. Em seguida, esses dados passam pelo bloco de *bit-unstuffing*, onde os bits '0' inseridos durante a transmissão no bloco de *bit-stuffing* são removidos. Por fim, os dados seriais passam pelo bloco de conversão serial-paralelo e são "entregues" aos níveis lógicos mais altos do dispositivo (NAM *et al.*, 2003). A Figura 3 ilustra de forma simples a composição do módulo Receptor - Rx.

Figura 3 – Bloco Receptor.



Fonte: Adaptada de (SHIN *et al.*, 2013).

A seguir são descritas as funções dos blocos que compõem a Figura 3:

Decodificação NRZI - converte uma transição do valor da entrada em um bit ‘0’ em sua saída. A “não-transição” é convertida em um bit ‘1’.

Bit-Unstuffing - remove um bit ‘0’, inserido forçadamente durante a transmissão, após uma sequência de seis ’1’s consecutivos.

Serial-Paralelo - realiza a conversão serial-paralelo dos bits recebidos. O bloco deve realizar aquisição serial dos bits em sua entrada e armazená-los até possuir 8 bits para entregá-los paralelamente em sua saída. O bloco repete continuamente o procedimento.

3 PROJETO DOS BLOCOS DIGITAIS

Este capítulo destinou-se ao projeto dos blocos digitais de transmissão, no qual são apresentados detalhes da implementação.

As arquiteturas das implementações foram desenvolvidas pelo autor, seguindo a metodologia proposta, baseando-se nas descrições comportamentais dos circuitos, citadas em (LUEKER *et al.*, 2000) e (MCGOWAN, 2001).

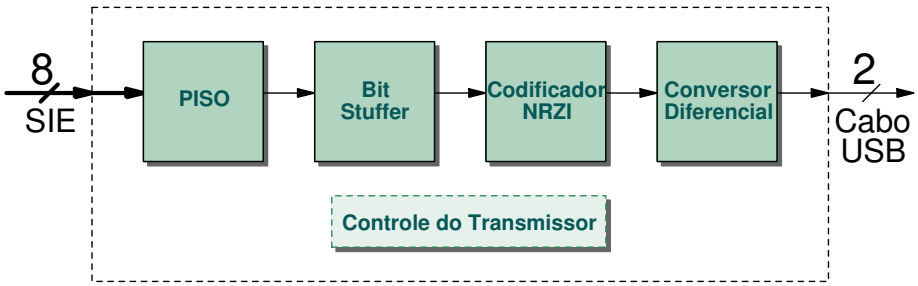
Durante a implementação dos blocos, foi necessário realizar simulações para avaliar seu funcionamento e, por vezes, combinados com outro bloco. Quando encontradas divergências no funcionamento esperado, melhorias foram realizadas nas propostas iniciais dos blocos, até que o circuito obtivesse o adequado funcionamento.

Nesse processo, visou-se a divisão explícita dos circuitos em *Datapath* e *Control Path*. Esse é um método de projeto de circuitos digitais largamente utilizado, com a vantagem de apresentar um projeto bem estruturado, com um único bloco de controle "comandando" o sistema.

Porém, nos blocos **PISO** e ***Bit Stuffer***, ao realizar simulações com a combinação dos dois, percebeu-se a necessidade de adicionar um sinal de controle entre eles. Esse sinal representa um *stuffing* e será explicado mais a frente no trabalho.

Como foi mencionado no Capítulo 2, os componentes básicos para a implementação do Transmissor são os blocos **PISO**, ***Bit Stuffer*** e **Codificador NRZI**. Mas ao optar pelo método de projeto utilizando a divisão dos circuitos em *Datapath* e *Control Path*, foi obviamente necessário adicionar o bloco de **Controle do Transmissor**. Outro bloco extra também implementado foi o bloco **Conversor Diferencial**, que poderia ser integrado ao bloco **Codificador NRZI** mas, por questões de projeto, optou-se por fazê-lo separadamente. Assim, a Figura 4 ilustra de forma simples a nova composição proposta do bloco Transmissor.

Figura 4 – Bloco Transmissor.



Fonte: Elaborada pelo autor

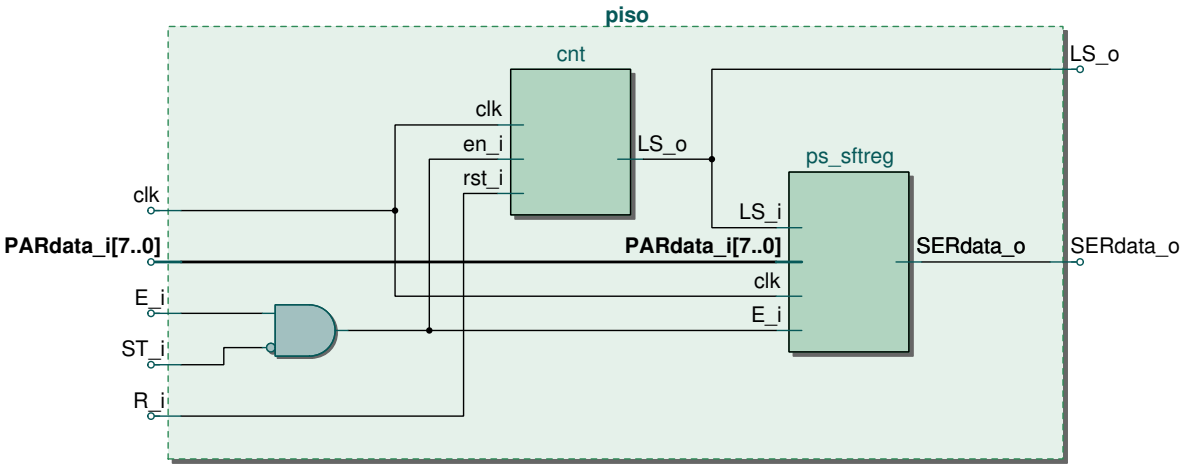
.

3.1 BLOCO PISO

O bloco *Parallel-In to Serial-Out* (nomeado **PISO**) realiza a conversão paralelo-serial dos bits a serem enviados do dispositivo para o cabo USB.

Baseado na especificação comportamental do bloco, chegou-se à proposta do diagrama esquemático da Figura 5, que é composto por blocos com funções mais básicas: um contador (nomeado **cnt**), um *shift register* - registrador de deslocamento (nomeado **ps_sftreg**) e uma porta lógica **AND**.

Figura 5 – Diagrama do bloco **PISO**.



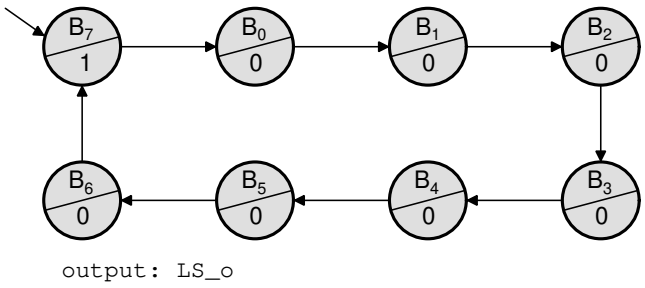
Fonte: Elaborada pelo autor

.

O bloco **ps_sftreg**, um registrador de deslocamento, foi implementado de maneira comportamental. Ele é um registrador de 8 bits que tem a função de armazenar os 8 bits de dados carregados paralelamente do barramento **PARdata_i** ou de deslocar serialmente esses 8 bits nele armazenados. A seleção entre carregar (*Load*) OU deslocar (*Shift*) os dados, vai depender do sinal **LS**, indicador proveniente do bloco **cnt**.

Já o bloco **cnt**, um contador de bits, teve seu comportamento descrito por uma Máquina de Moore, representado pelo diagrama de estados na Figura 6. Nesse diagrama, pode-se observar os estados de **B0** a **B7**, que representam a contagem decimal (de "0" a "7"), e o sinal **LS**, que é um indicador de que o contador terminou a contagem de 8 valores decimais (de "0" a "7"), ou seja, de "000" a "111" binário). No estado **B7**, é indicado o fim de um ciclo de 8 bits e o sinal **LS** recebe '1' (**LS** = '1'); durante todos os outros estados **LS** recebe '0' (**LS** = '0').

Figura 6 – Máquina de Moore do bloco **cnt**.



Fonte: Elaborada pelo autor

.

3.1.1 Funcionamento do bloco PISO

O bloco **PISO** é resetado e habilitado pelo bloco de controle. Ao receber o sinal de *reset*, o bloco **cnt** é resetado e sua máquina de estados vai para o estado inicial (**B7**), enviando **LS** = '1' na saída. A contagem é então iniciada (**B0** - bit0, **B1** - bit1, ..., **B7** - bit7, **B0** - bit0...) e, após 8 ciclos de *clock*, o contador volta para o estado inicial (**LS** = '1'). Em todos os outros 7 estados o sinal **LS** é igual a "zero" (**LS** = '0').

O bloco **ps_sftreg** recebe o sinal **LS** e sempre que **LS** = '1', o registrador carrega os dados paralelos do barramento **PARdata_i**. Quando **LS** = '0', o registrador desloca para a saída os dados nele armazenados.

Desse modo, o bloco **PISO** realiza a carga paralela de 8 bits do barramento de entrada e os envia serialmente para a saída. Após 8 ciclos, uma nova carga paralela é realizada e convertida em dados seriais.

3.1.2 Símbolo do bloco PISO

Ao concluir as etapas anteriores, foi criado o diagrama do símbolo do bloco, onde entradas e saídas do bloco são nomeadas. O símbolo do bloco **PISO** é apresentado na Figura 7.

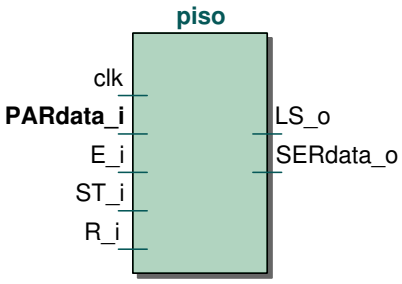
Buscou-se, desde o início, a divisão explícita dos sinais em *Datapath* e *Control Path*, mas por questões de praticidade, optou-se por realizar o controle do sinal **ST** diretamente do bloco ***Bit Stuffer*** para o bloco **PISO**.

Todos os sinais com a terminação "_i" são sinais de entrada e com "_o", sinais de saída do bloco.

clk - sinal de *clock* fornecido pelo dispositivo.

PARdata_i - barramento paralelo de 8 bits, sinal de dados (*Datapath*) oriundo da seleção entre padrão **SYNC** e **DataIn_i** através do multiplexador (Figura 17), no bloco **Transmissor**.

Figura 7 – Símbolo do bloco **PISO**.



Fonte: Elaborada pelo autor

.

E_i - *enable*, sinal de comando (*Control Path*) proveniente do bloco de controle.

R_i - *reset*, sinal de comando (*Control Path*).

ST_i - *Stuffing*, sinal de comando proveniente do bloco ***Bit Stuffer*** (este sinal tem uma função específica e será explicado na seção do bloco ***Bit Stuffer***). Enquanto este sinal está ativo (**ST_o** = '1') o bloco **PISO** é desabilitado.

LS_o - *Load/Shift*, sinal de status (*Control Path*), indica se o bloco **PISO** está carregando (*Load*) o registrador com dados **PARdata_i** ou se está realizando o deslocamento (*Shift*) dos dados do registrador.

SERdata_o - barramento serial de dados, sinal de dados (*Datapath*) enviado para o bloco ***Bit Stuffer***.

3.2 BLOCO *BIT STUFFER* (BITSTUFF)

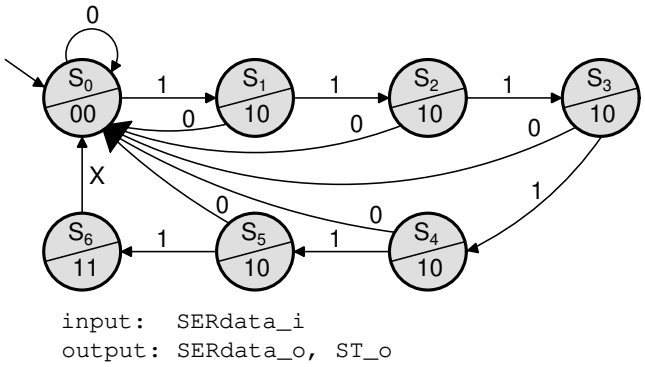
Bit Stuffing é um sistema que limita o número de consecutivos '1's, evitando a falta de transição no sinal transmitido após a codificação NRZI.

Baseado na especificação comportamental do bloco, chegou-se à proposta do diagrama de estados do bloco ***Bit Stuffer*** da Figura 8, descrito em uma Máquina de Moore. Nele pode-se observar os estados de **S0** a **S6**, que representam a quantidade de bits '1' consecutivos (state0 - nenhum bit '1', state1 - um bit '1', state2 - dois bits '1', ..., state6 - seis bits '1').

Durante a transmissão, o ***Bit Stuffer*** verifica a existência de uma sequência de seis bits '1's consecutivos no sinal de dados de entrada e insere um bit '0' ("*stuffed bit*"), quebrando a sequência de '1's. No momento do *stuffing*, após seis bits '1', o bit presente na entrada do bloco ***Bit Stuffer*** não importa, pois a saída deverá apresentar um "zero" (essa é a função do sistema de *stuffing*).

Após criar o diagrama de estados do bloco ***Bit Stuffer*** da Figura 8, transcreveu-se esse diagrama em uma tabela de transições de estados (Tabela 1), a qual apresenta as

Figura 8 – Máquina de Moore do bloco ***Bit Stuffer***.



Fonte: Elaborada pelo autor

.

transições de estados e as respostas de saída do bloco, baseada nas entradas do mesmo. Essas transições demonstram qual será o próximo estado e quais os valores das saídas, dependendo do estado atual e dos valores de entrada do bloco. Essa é uma maneira de apresentar o comportamento do bloco, para facilitar o entendimento.

Tabela 1 – Tabela de Transições do bloco ***Bit Stuffer***.

Entradas		**Saídas**		
SERdata_i	Estado Atual	Próximo Estado	SERdata_o	ST_o
0	S0	S0	0	0
1	S0	S1	0	0
0	S1	S0	1	0
1	S1	S2	1	0
0	S2	S0	1	0
1	S2	S3	1	0
0	S3	S0	1	0
1	S3	S4	1	0
0	S4	S0	1	0
1	S4	S5	1	0
0	S5	S0	1	0
1	S5	S6	1	0
0	S6	S0	1	1
1	S6	S0	1	1

Fonte: Elaborada pelo autor

.

3.2.1 Funcionamento do Bloco *Bit Stuffer*

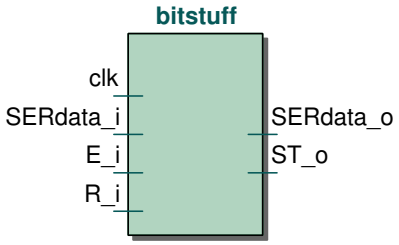
O bloco ***Bit Stuffer*** verifica o valor lógico do sinal de dados presente em sua entrada (**SERdata_i**). Ao receber '1' na entrada, ele avança para o próximo estado, caso contrário, ele retorna para o estado inicial **S0**. Após seis '1's consecutivos, ou seja, após passar por todos os estados e chegar ao último estado **S6** (**S1, S2, ..., S6**), o sinal de *status* de saída do bloco (**ST_o**) é igual a '1', indicando essa sequência de seis bits '1'. Ao atingir o último estado (**S6**), não importa o valor da entrada, o próximo estado será **S0**. O valor do sinal de dados de saída (**SERdata_o**) é igual a '1' para todos os estados, com exceção do estado **S0**, em que é igual a '0'.

Para evitar a perda, ou a necessidade de armazenar o bit presente na entrada do bloco, no momento do *stuffing*, o sinal **ST_o** foi utilizado como sinal de comando pelo o bloco **PISO**, com a função de "*pausar*" ou "congelar" o deslocamento serial de dados, desabilitando o bloco enquanto o sinal **ST_o** = '1'. Dessa maneira, no momento em que o bit '0' é inserido na sequência de dados no bloco ***Bit Stuffer***, o bit presente na sua própria entrada, que seria perdido, é "repetido" ao desabilitar o bloco **PISO** momentaneamente.

3.2.2 Símbolo do bloco *Bit Stuffer*

Ao concluir as etapas anteriores, foi criado o diagrama do símbolo do bloco, onde entradas e saídas do bloco são nomeadas. O símbolo do bloco ***Bit Stuffer*** (**bitstuff**) é apresentado na Figura 9.

Figura 9 – Símbolo do bloco ***Bit Stuffer***.



Fonte: Elaborada pelo autor

.

Todos os sinais com a terminação "_i" são sinais de entrada e com "_o", sinais de saída do bloco.

clk - sinal de *clock* fornecido pelo dispositivo.

SERdata_i - barramento serial de dados, sinal de dados (*Datapath*) proveniente do bloco **PISO**.

E_i - *enable*, sinal de comando (*Control Path*).

R__i - *reset*, sinal de comando (*Control Path*).

SERdata__o - barramento serial de dados, sinal de dados (*Datapath*) enviado para o bloco **Codificador NRZI**.

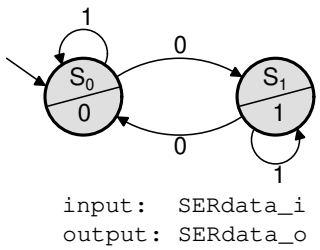
ST__o - *Stuffing*, sinal de status, indica se o bloco ***Bit Stuffer*** detectou uma sequência de seis 1's consecutivos no barramento serial de entrada **SERdata__i**. Enquanto este sinal está ativo (**ST__o** = '1') o bloco **PISO** é desabilitado.

3.3 BLOCO CODIFICADOR NRZI (NRZIENCOD)

A codificação NRZI é utilizada para codificar o sinal de modo que, com o auxílio de um CDR (*Clock Data Recovery*) na recepção, ocorra a sincronização da taxa de dados (bits recebidos) com o *clock* de operação do receptor.

Baseado na especificação comportamental do bloco, chegou-se à proposta do diagrama de estados do bloco **Codificador NRZI** da Figura 10, descrito em uma Máquina de Moore. Nele podem-se observar os únicos dois estados **S0** e **S1**, que representam o valor do sinal de dados de saída do bloco.

Figura 10 – Máquina de Moore do bloco **Codificador NRZI**.



Fonte: Elaborada pelo autor

.

Ao receber um bit '1' na entrada, a máquina de estados permanece no estado em que se encontra e, ao receber um '0', ela troca de estado. Ou seja, um bit '0' representa uma transição e um bit '1' representa uma "não-transição" do sinal de dados de saída do bloco.

Após criar o diagrama de estados do bloco **Codificador NRZI** da Figura 10, por ser um diagrama de estados bem simples, criou-se uma tabela de codificação de estados (Tabela 2) do bloco **Codificador NRZI**, para gerar o circuito manualmente através da simplificação do circuito lógico utilizando, mapa de Karnaugh.

O número total de estados indica a quantia necessária de registradores para gerar o circuito do bloco. Como só existem dois estados, e cada registrador pode representar dois valores binários ('0' e '1'), basta apenas um registrador.

Tabela 2 – Tabela de Codificação de Estados do bloco **Codificador NRZI**.

Entradas	Código
	Q
S0	0
S1	1

Fonte: Elaborada pelo autor

.

Em seguida, criou-se também uma tabela de transições de estados (Tabela 3) do bloco **Codificador NRZI**.

Tabela 3 – Tabela de Transições de Estados do bloco **Codificador NRZI**.

Entradas			**Saídas**		
Q	SERdata_i	Estado Atual	Próximo Estado	D	SERdata_o
0	0	S0	S1	1	0
0	1	S0	S0	0	0
1	0	S1	S0	0	1
1	1	S1	S1	1	1

Fonte: Elaborada pelo autor

.

A partir dessas informações, extraíram-se as equações lógicas do circuito e utilizou-se o método do mapa de Karnaugh para redução do circuito mínimo das saídas, chegando aos seguintes resultados:

- O sinal de dados de saída do bloco (**SERdata_o**) recebe o sinal de saída do registrador (**Q**).

  SERdata_o <= Q.

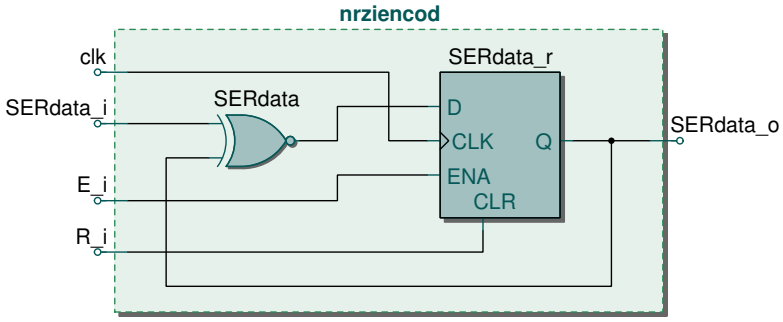
- A entrada do registrador (**D**) recebe o valor da operação **xnor** (ou exclusivo negado) entre os sinais **Q** e **SERdata_i**.

  $D <= Q.SERdata\_i + \overline{Q}.\overline{SERdata\_i}$

  $D <= \overline{Q \oplus SERdata\_i}$

Por fim, o circuito do bloco **Codificador NRZI** foi obtido através dessas equações e é apresentado na Figura 11, onde se pode perceber a presença do registrador com a sua saída (**Q**) conectada à saída do sinal de dados do bloco (**SERdata_o**) e também à porta lógica que realiza a operação **XNOR** (lê-se "ou exclusivo negado") entre os sinais **Q** e **SERdata_i**.

Figura 11 – Diagrama do bloco **Codificador NRZI**.



Fonte: Elaborada pelo autor

.

3.3.1 Funcionamento do bloco Codificador NRZI

A porta lógica **xnor** compara o valor entre suas duas entradas ("**Q**" e "**SERdata_i**").

Se ambas são iguais, então o registrador recebe '1' (**D** <= '1'), pelo princípio de funcionamento de um registrador (**Q** <= **D**), o que implica na saída do registrador em **Q** = '1'. Caso as entradas sejam diferentes, então **D** recebe '0' e, consequentemente, **Q** = '0'. Outra maneira seria dizer que, sempre que a entrada **SERdata_i** for '1', a saída do registrador **Q**, e também do bloco **SERdata_o**, não vai se alterar, e sempre que a entrada **SERdata_i** for '0', a saída do registrador **Q**, e também do bloco **SERdata_o**, vai trocar de valor.

Observando a Tabela 3, percebe-se a afirmação do parágrafo anterior. Quando a entrada **SERdata_i** for igual a '1', então a saída **SERdata_o** mantém o seu valor, pois se **Q** também for '1', tem-se "entradas iguais" e assim a saída (**SERdata_o**) continua '1'. Caso **Q** seja '0', tem-se "entradas diferentes" e assim a saída (**SERdata_o**) continua '0'.

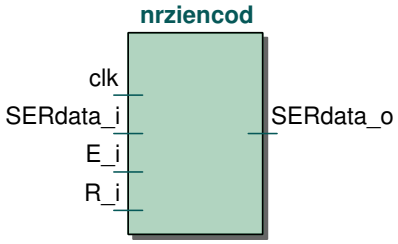
Já quando a entrada **SERdata_i** for igual a '0', então a saída **SERdata_o** tem seu valor alterado, pois se **Q** também for '0' tem-se "entradas iguais" e assim a saída (**SERdata_o**) torna-se '1'. Caso **Q** seja '1', tem-se "entradas diferentes" e assim a saída (**SERdata_o**) torna-se '0'.

3.3.2 Símbolo do bloco Codificador NRZI

Ao concluir as etapas anteriores, foi criado o diagrama do símbolo do bloco, onde entradas e saídas do bloco são nomeadas. O símbolo do bloco **Codificador NRZI** (**nrziencod**) é apresentado na Figura 12.

Todos os sinais com a terminação "_i" são sinais de entrada e com "_o", sinais de saída do bloco.

Figura 12 – Símbolo do bloco **Codificador NRZI**.



Fonte: Elaborada pelo autor

.

clk - sinal de *clock* fornecido pelo dispositivo.

SERdata_i - barramento serial de dados, sinal de dados (*Datapath*) proveniente do bloco ***Bit Stuffer***.

E_i - *enable*, sinal de comando (*Control Path*).

R_i - *reset*, sinal de comando (*Control Path*).

SERdata_o - barramento serial de dados, sinal de dados (*Datapath*) enviado para o bloco **Conversor Diferencial**.

3.4 BLOCO CONVERSOR DIFERENCIAL (DIFFENCOD)

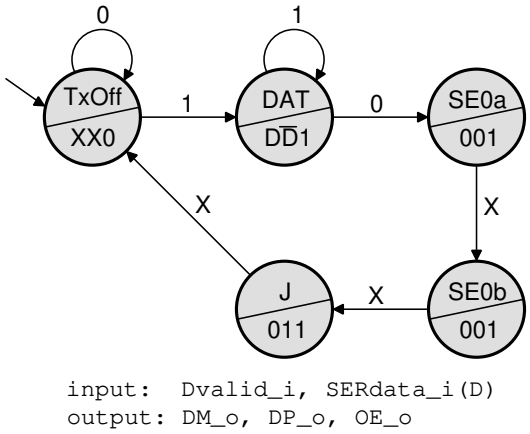
Por fazer parte da especificação USB, a comunicação de dados ocorre por vias diferenciais. É necessário, então, a conversão dos dados para saídas diferenciais. Optou-se em criar um bloco separado para realizar a conversão diferencial, apesar desta também poder estar "embutida" no bloco **Codificador NRZI**.

Num sistema USB Full-speed, a combinação das linhas diferenciais positiva e negativa (**DP** e **DM**), possuem denominações. Para **DP** = '0' e **DM** = '0', tem-se um ***Single-Ended Zero*** **(SE0)**. Para **DP** = '0' e **DM** = '1' tem-se um estado '**K**' e para **DP** = '1' e **DM** = '0' tem-se um estado '**J**', igualmente ao estado IDLE do sistema. O estado ***Single-Ended One*** **(SE1)**, para ambos **DP** e **DM** iguais a '1', é um estado que não deve ocorrer e é visto como um erro.

O bloco **Conversor Diferencial** foi implementado com uma Máquina de Moore, conforme Figura 13. Nela pode-se observar os estados **TxOff, DAT, SE0a, SE0b** e **J**. A fim de simplificar o diagrama, a entrada **SERdata_i** foi substituída pela letra *D*.

O estado **TxOff** representa a situação em que os dados presentes na entrada do bloco de **Conversor Diferencial** não são válidos para serem transformados em um par diferencial. Ou seja, nesse estado, não importam os valores das linhas diferenciais **DM_o**

Figura 13 – Máquina de Moore do bloco **Conversor Diferencial**.



Fonte: Elaborada pelo autor

.

e **DP_o**, pois um estado '**J**' (*Idle*) ocorre naturalmente (está relacionado com resistores pull-up/pull-down do USB, não fazendo parte do escopo desse trabalho).

No estado **DAT**, a máquina de estados converte os dados de entrada em par diferencial nas linhas **DM**, que recebem o valor da entrada, e **DP**, que recebe o valor da entrada **SERdata_i** barrada/negada.

A sucessão dos estados **SE0a**, **SE0b** e **J** compõe a sequência que representa, nas linhas diferenciais, o fim de uma transmissão. Esta sequência (2x**SE0** e 1x**J**) é denominada padrão de Encerramento de Pacote - *End Of Packet* (EOP).

Após criar o diagrama de estados do bloco **Conversor Diferencial** da Figura 13, transcreveu-se esse diagrama em uma tabela de transições de estados (Tabela 4).

Nela, é possível observar que, no estado **DAT**, as saídas **DM** e **DP** do bloco **Conversor Diferencial** são derivadas da entrada **SERdata_i**, representada como D. Nesse caso, "D" tem o valor do próprio sinal de dados de entrada **SERdata_i**, "$\overline{D}$" tem o valor negado/barrado e "X" representa um valor indiferente e que não importa. Caso a entrada **SERdata_i** seja '1', então "D" = '1' e "$\overline{D}$" = '0'.

Já a Tabela 5 apresenta especificamente as transições do estado **DAT** do bloco **Conversor Diferencial**, referente à Tabela 4, explicitando os valores das saídas (**DM** e **DP**), esclarecendo o parágrafo anterior quanto à questão do sinal de dados de entrada D. Em qualquer outro estado do bloco **Conversor Diferencial**, a entrada **SERdata_i** (representada como D) não tem influência nas saídas.

Tabela 4 – Tabela de Transições de Estados do bloco **Conversor Diferencial**.

Entradas		**Saídas**			
Dvalid_i	Estado Atual	Próximo Estado	DM_o	DP_o	OE_o
0	TxOff	TxOff	X	X	0
1	TxOff	TxOff	X	X	0
0	DAT	SE0a	D	$\overline{D}$	1
1	DAT	DAT	D	$\overline{D}$	1
0	SE0a	SE0b	0	0	1
1	SE0a	SE0b	0	0	1
0	SE0b	J	0	0	1
1	SE0b	J	0	0	1
0	J	TxOff	0	1	1
1	J	TxOff	0	1	1

Fonte: Elaborada pelo autor.

Tabela 5 – Tabela de Transições do Estado **DAT** no bloco **Conversor Diferencial**.

Entradas			**Saídas**			
Dvalid_i	SERdata_i (D)	Estado Atual	Próximo Estado	DM_o	DP_o	OE_o
0	0	DAT	SE0a	0	1	1
0	1	DAT	SE0a	1	0	1
1	0	DAT	DAT	0	1	1
1	1	DAT	DAT	1	0	1

Fonte: Elaborada pelo autor

.

3.4.1 Funcionamento do bloco Conversor Diferencial

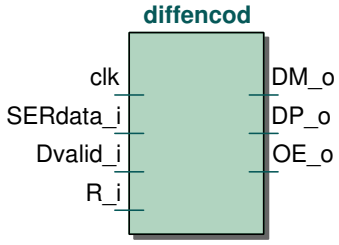
O bloco **Conversor Diferencial** é controlado pelo bloco de controle. Ao receber o sinal de *reset* no início de uma transmissão, a Máquina de estados é iniciada no estado **TxOff** e ali ela permanece até que o "*enable*" (**Dvalid_i** = '1') seja detectado, indicando que os dados recebidos na entrada **SERdata_i** são válidos e podem ser convertidos diferencialmente.

A Máquina de estados passa, então, para o estado **DAT**, estado em que ocorre a conversão diferencial para as saídas **DM** e **DP**. A conversão é realizada até que não existam mais dados válidos a serem convertidos (**Dvalid_i** = '0'). Esse é um indicador de que a transmissão deve encerrar e deve ser concluída com a transmissão do padrão de Encerramento de Pacote, passando pelos estados **SE0a**, **SE0b** e, por fim, pelo estado **J**.

3.4.2 Símbolo do bloco Conversor Diferencial

Ao concluir as etapas anteriores, foi criado o diagrama do símbolo do bloco, onde entradas e saídas do bloco são nomeadas. O símbolo do bloco **Conversor Diferencial** (**diffencod**) é apresentado na Figura 14.

Figura 14 – Símbolo do bloco **Conversor Diferencial**.



Fonte: Elaborada pelo autor

.

Todos os sinais com a terminação "_i" são sinais de entrada e com "_o", sinais de saída do bloco.

clk - sinal de *clock* fornecido pelo dispositivo.

SERdata_i - barramento serial de dados, sinal de dados (*Datapath*) proveniente do bloco **nrziencod**.

Dvalid_i - *Data Valid*, sinal de comando (*Control Path*), funciona como um *enable* do bloco. Com **Dvalid_i** = '1' indica-se a existência de dados válidos a serem convertidos para os sinais **DM** e **DP** (linhas diferenciais). Após inicio da conversão, **Dvalid_i** = '0' indica o que não existem mais dados válidos e deve-se enviar o Pacote de Encerramento (EOP). **Dvalid** é proveniente do bloco de **Controle**.

R_i - *reset*, sinal de comando (*Control Path).*

DM_o - sinal diferencial "negativo" de dados (-), saída para a interface do bloco **Transmissor** e o cabo USB.

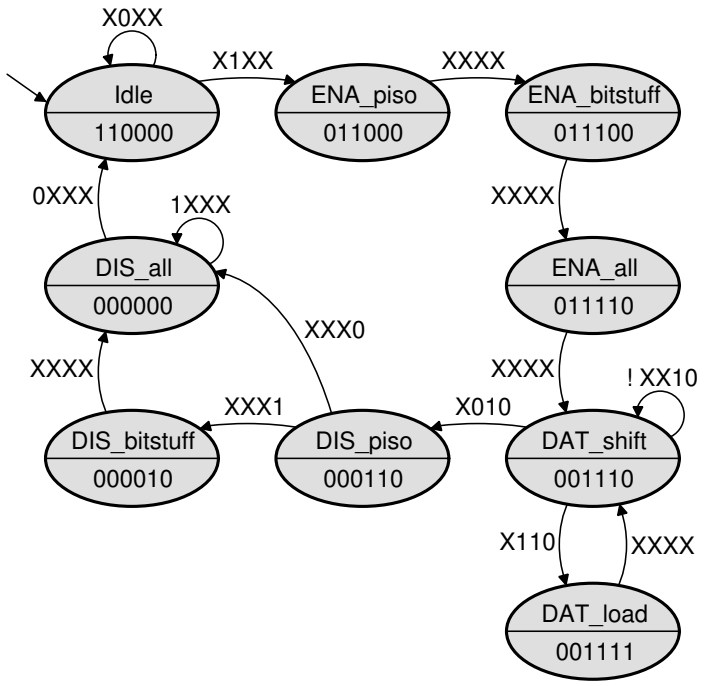
DP_o - sinal diferencial "positivo" de dados (+), saída para a interface do bloco **Transmissor** e o cabo USB.

OE_o - *Output Enable*, sinal de *status* (*Control Path*), indica se há dados válidos a transmitir. Quando **OE_o** está ativo, o barramento USB está transmitindo e os dados do barramento diferencial são reconhecidos como sinais de saída e não de entrada (no caso de uma recepção). **OE_o** também é uma saída para a interface do bloco **Transmissor**.

3.5 CONTROLE DO TRANSMISSOR (TX_CONTROL)

O bloco **Controle do Transmissor** é responsável por operar e controlar os componentes do bloco **Transmissor**. Ele foi implementado com uma Máquina de Moore conforme a Figura 15.

Figura 15 – Máquina de Moore do bloco de **Controle do Transmissor**.




```
input:  OE_i, Valid_i, LS_i, ST_i
output: R_o, SD_0, E1_o, E2_o, E3_o, Ready_o
```

Fonte: Elaborada pelo autor

.

Após criar o diagrama de estados do bloco de **Controle do Transmissor** da Figura 15, transcreveu-se esse diagrama nas tabelas de transições de estados (Tabela 6) e de saídas (Tabela 7).

3.5.1 Funcionamento do bloco Controle do Transmissor

A Máquina de estados inicia no estado **IDLE** ao receber o sinal de **rst_i** antes de começar uma transmissão. No estado **IDLE**, o controle faz o *reset* (**R_o**) dos blocos controlados e mantém o multiplexador a selecionar o padrão *Sync*.

A Máquina de Estados então permanece no estado **IDLE** até que o sinal **Valid_i** seja habilitado. Ao receber o sinal **Valid_i** ativo, a Máquina de Estados passa para o estado **ENA_piso**, terminando o *reset* dos blocos controlados e habilitando o bloco **PISO**.

Tabela 6 – Tabela de Transições de Estados do bloco de **Controle do Transmissor**.

Entradas				**Saída**
OE_i	Valid_i	LS_i	ST_i	Estado Atual	Próximo Estado
X	0	X	X	IDLE	IDLE
X	1	X	X	IDLE	ENA_piso
X	X	X	X	ENA_piso	ENA_bittuff
X	X	X	X	ENA_bitstuff	ENA_all
X	X	X	X	ENA_all	DAT_shift
X	0	0	0	DAT_shift	DAT_shift
X	0	0	1	DAT_shift	DAT_shift
X	0	1	0	DAT_shift	DIS_piso
X	0	1	1	DAT_shift	DAT_shift
X	1	0	0	DAT_shift	DAT_shift
X	1	0	1	DAT_shift	DAT_shift
X	1	1	0	DAT_shift	DAT_load
X	1	1	1	DAT_shift	DAT_shift
X	X	X	X	DAT_load	DAT_shift
X	X	X	0	DIS_piso	DIS_all
X	X	X	1	DIS_piso	DIS_bitstuff
X	X	X	X	DIS_bitstuff	DIS_all
0	X	X	X	DIS_all	IDLE
1	X	X	X	DIS_all	DIS_all

Fonte: Elaborada pelo autor

.

Tabela 7 – Tabela de Saídas do bloco de **Controle do Transmissor**.

Entrada	**Saídas**					
Estado Atual	R_o	SD_o	E1_o	E2_o	E3_o	Ready_o
IDLE	1	1	0	0	0	0
ENA_piso	0	1	1	0	0	0
ENA_bitstuff	0	1	1	1	0	0
ENA_all	0	1	1	1	1	0
DAT_shift	0	0	1	1	1	0
DAT_load	0	0	1	1	1	1
DIS_piso	0	0	0	1	1	0
DIS_bitstuff	0	0	0	0	1	0
DIS_piso	0	0	0	0	0	0

Fonte: Elaborada pelo autor

.

Na sequência, o estado **ENA_bitstuff** habilita o bloco ***Bit Stuffer*** e o estado **ENA_all** habilita os blocos **Codificador NRZI** e **Conversor Diferencial**.

Depois de habilitar todos os blocos, a máquina de estados entra no estado **DAT_shift**, alterando o multiplexador para selecionar **DataIn**. Nesse estado, a máquina de estados aguarda o conjunto de sinais **Valid_i** = '1', **LS_i** = '1' e **ST_i** = '0', para entrar no estado **DAT_load**, ou **Valid_i** = '0', **LS_i** = '1' e **ST_i** = '0', para entrar no estado **DIS_piso**. Para qualquer outra combinação, a máquina de estados permanece no estado **DAT_shift**.

Durante o estado **DAT_shift**, os dados são apenas deslocados serialmente. Já no **DAT_load**, o sinal **Ready_o** é ativado e, logo em seguida, a máquina de estados volta para o estado **DAT_shift**. O sinal **Ready_o** é o indicador para que o SIE altere os dados paralelos do barramento **DataIn**, dados que continuarão a ser convertidos serialmente.

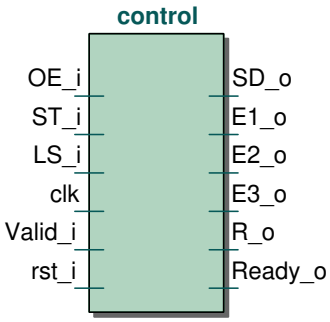
Ao entrar no estado **DIS_piso**, o bloco **PISO** é desabilitado e, caso um *Stuffing* **ST_i** seja detectado, a máquina de estados entra no estado **DIS_bitstuff** e, em seguida, no estado **DIS_all**, desabilitando o restante dos blocos (**Codificador NRZI** e **Conversor Diferencial**). Caso nenhum *Stuffing* seja detectado, a máquina de estados passa diretamente do estado **DIS_piso** para o estado **DIS_all**, desabilitando todos os blocos no mesmo estado.

Por fim, durante o estado **DIS_all**, a máquina de estados aguarda o sinal **OE_i** ser desabilitado para concluir a transmissão e retornar para o estado **IDLE**.

3.5.2 Símbolo do bloco Controle do Transmissor

Ao concluir as etapas anteriores, foi criado o diagrama do símbolo do bloco, onde entradas e saídas do bloco são nomeadas. A Figura 16 apresenta o símbolo do bloco de **Controle do Transmissor**.

Figura 16 – Símbolo do bloco **Controle do Transmissor**.



Fonte: Elaborada pelo autor

.

Todos os sinais com a terminação "_i" são sinais de entrada e com "_o", sinais de

saída do bloco.

clk - sinal de *clock* fornecido pelo dispositivo.

OE_i - *Output Enable*, sinal de status, indica se há dados válidos na saída do bloco **Transmissor** a transmitir. Proveniente do bloco **Conversor Diferencial**.

ST_i - *Stuffing*, sinal de status, indica a detecção de seis '1's consecutivos para realizar a inserção de um bit '0' forçadamente. Proveniente do bloco ***Bit Stuffer***.

LS_i - *Load/Shift*, sinal de status, indica a carga ou deslocamento serial no bloco **PISO**. Proveniente do bloco **PISO**.

Valid_i - sinal que indica o início de uma transmissão, proveniente de uma camada de alto nível do dispositivo USB através da interface do bloco **Transmissor**.

rst_i - *reset*, proveniente do sistema.

SD_o - *Sync/Data*, sinal de comando, controla o multiplexador presente no bloco **tx** (Figura 17). O multiplexador seleciona os dados paralelos (**Sync** ou **DataIn_i**), que chegam à entrada do bloco **PISO**. Com **SD_o** = '0', multiplexador seleciona **Sync** e para **SD_o** = '1', seleciona **DataIn_i**.

E1_o - *enable*, sinal de comando, habilita o bloco **PISO**.

E2_o - *enable*, sinal de comando, habilita o bloco ***Bit Stuffer***.

E3_o - *enable*, sinal de comando, habilita os blocos **Codificador NRZI** e **Conversor Diferencial**.

R_o - *reset*, sinal de comando, "reseta" os blocos que compõem o bloco **Transmissor (tx)**.

Ready_o - sinal indica que os 8 bits paralelos carregados do barramento **DataIn** já foram deslocados serialmente pelo bloco **PISO** e o SIE deve colocar novos à disposição para que nova carga seja realizada para a transmissão. **Ready_o** é uma saída para a interface do bloco **Transmissor**.

3.6 BLOCO TRANSMISSOR (TX)

O bloco Transmissor é responsável por converter os dados paralelos do barramento de entrada em uma saída de sinal de dados serial diferencial, com a inexistência de sequências longas sem transições (nas linhas diferenciais) e com a codificação adequada, de modo que os dados sejam recebidos corretamente no receptor, seguindo os requisitos do USB.

A construção do bloco transmissor consistiu-se basicamente da integração dos blocos vistos anteriormente, com a adição de um multiplexador. Esse multiplexador tem a

função de selecionar os dados paralelos que chegam à entrada do bloco **PISO** para serem "serializados". As opções são o barramento de entrada **DataIn_i** ou o padrão *Sync*.

O padrão *Sync* é uma sequência determinada, pelos requisitos USB, de 8 bits ("00000001" - antes da codificação NRZI) que, após ser codificada e transmitida para o cabo USB, torna-se ("KJKJKJKK" - byte de sincronização de *clock*) o indicador da recepção de que será iniciada uma transmissão de dados válidos, deixando o receptor apto para a recepção. Além do padrão *Sync* ser o indicador de que existe uma transmissão acontecendo, de modo que o receptor não deve ignorar esses dados, ele também é utilizado pelo CDR para sincronizar a taxa de dados com o *clock* de operação do receptor.

Após a integração de todos os blocos, chegou-se à construção do diagrama apresentado na Figura 17.

Ao detectar o início de uma transmissão, através do sinal de status **TxValid_i** = '1', o bloco de controle envia **SD_o**= '1' para o multiplexador, selecionando o padrão *Sync* a ser enviado para o bloco **PISO**, iniciando o caminho do *Datapath*.

Em seguida, conforme a mudança dos estados do bloco de controle vai progredindo, os blocos vão sendo habilitados, até que o *Datapath* esteja completo. O *Datapath* é um "caminho" entre a entrada de dados e a saída de dados do bloco **Transmissor**.

O bloco **PISO** realiza a primeira conversão serial dos 8 bits do padrão *Sync*, ativando o sinal **LS_o** para informar ao controle que os dados foram "serializados" e uma nova carga deve ser realizada. Nesse momento, após a primeira conversão de 8 bits, o controle envia **SD_o** = '0' para o multiplexador selecionando agora os dados do barramento de entrada vindos do SIE. De agora em diante, o multiplexador não tem sua seleção alterada, até que a transmissão seja encerrada. Desse modo, o bloco **PISO** continua a repetir seu ciclo de carregar novos dados e convertê-los serialmente até que seja detectado o fim dos dados para transmissão, através do sinal de status **TxValid_i** = '0'.

Os dados passam, então, pelos blocos seguintes, conforme já foi detalhado nas sessões anteriores.

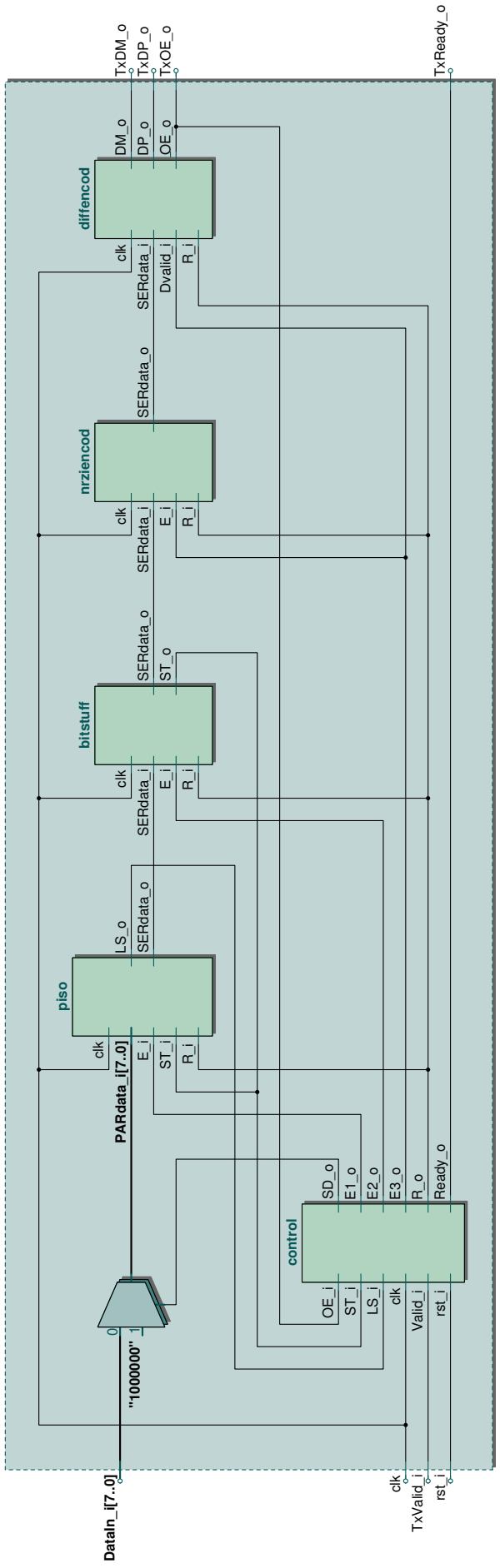


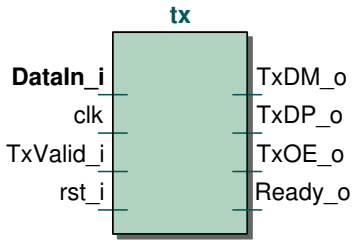
Figura 17 – Diagrama do bloco **Transmissor**.

Fonte: Elaborada pelo autor

.

Por fim, criou-se o diagrama do símbolo do bloco **Transmissor (tx)** apresentado na Figura 18.

Figura 18 – Símbolo do bloco **Transmissor**.



Fonte: Elaborada pelo autor

.

Todos os sinais com a terminação "_i" são sinais de entrada e com "_o", sinais de saída do bloco.

clk - sinal de *clock* fornecido pelo dispositivo.

PARdata_i - barramento paralelo de 8 bits, sinal de dados (*Datapath*) proveniente da seleção entre padrão **SYNC** e **DataIn_i** através do multiplexador (Figura 17), no bloco **Transmissor (tx)**.

DataIn_i - barramento paralelo de 8 bits, sinal de dados (*Datapath*) proveniente do SIE.

TxValid_i - sinal de status (*Control Path*), indica o início de uma transmissão, proveniente do SIE.

rst_i - sinal de status *reset* (*Control Path)*, proveniente do sistema.

TxDM_o - sinal diferencial "negativo" de dados (-), saída para o cabo USB.

TxDP_o - sinal diferencial "positivo" de dados (+), saída para o cabo USB.

TxOE_o - sinal de *status* (*Control Path*), indica que há dados válidos na saída do bloco **Transmissor**. (Este sinal também serve de aviso para o sistema, indicando ao receptor que há uma transmissão no momento e que os dados presentes no barramento não devem ser interpretados como dados recebidos. Ou seja, só ocorre uma recepção quando o sinal **TxOE_o** estiver desativado.

TxReady_o - sinal de comando (*Control Path*), indica que os 8 bits paralelos do barramento **DataIn** já foram carregados pelo bloco Transmissor (especificamente pelo registrador do bloco **PISO**) e que o SIE deve apresentar os próximos dados à disposição no barramento de entrada, para que nova carga seja realizada. Quando não existem mais novos dados a serem transmitidos, o sinal **TxValid_i** é desativado, indicando que se deve iniciar o encerramento da transmissão.

4 SIMULAÇÕES E RESULTADOS

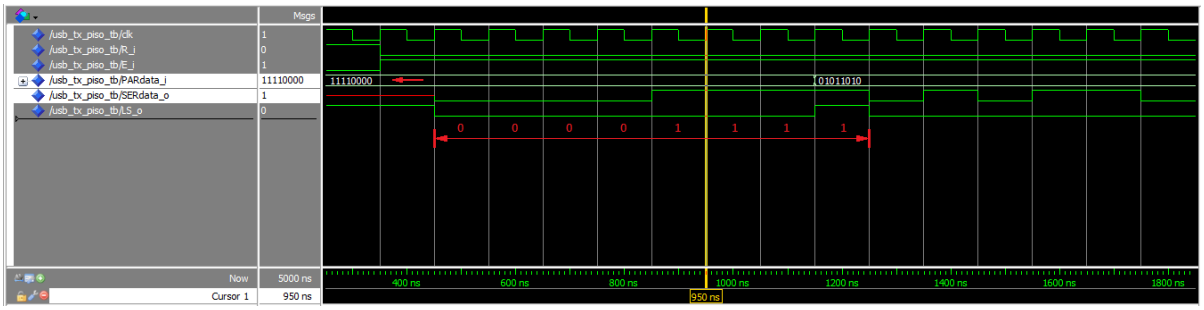
Este capítulo é dedicado a mostrar resultados e simulações obtidos, com o propósito de validar a funcionalidade dos blocos implementados no projeto.

Os valores das entradas nas simulações foram forçadas especificamente para gerar e analisar individualmente o comportamento (já esperado) em cada bloco. Além disso, a frequência do *clock* utilizada não corresponde ao valor real, pois as simulações visam apenas uma análise lógica do circuito. Com o uso do *software* de simulação *ModelSim*, as simulações foram realizadas e, a partir destas, foi possível observar coerência ou, algumas vezes, erros no comportamento dos blocos. Com isso, realizaram-se as devidas alterações na implementação para corrigir os problemas, até se chegar a um resultado satisfatório de cada componente implementado.

4.1 SIMULAÇÃO DO BLOCO PISO

A simulação do bloco **PISO** deu-se inicialmente com a escolha de uma entrada qualquer de dados, a fim de verificar o comportamento adequado do bloco. Nota-se que o sinal serial de dados de saída do bloco **SERdata_o** apresentou a conversão serial dos dados paralelos adquiridos na entrada **PARdata_i** corretamente, do bit menos significativo para o mais significativo, assim como esperado, conforme apresentado na Figura 19.

Figura 19 – Simulação do bloco **PISO**.

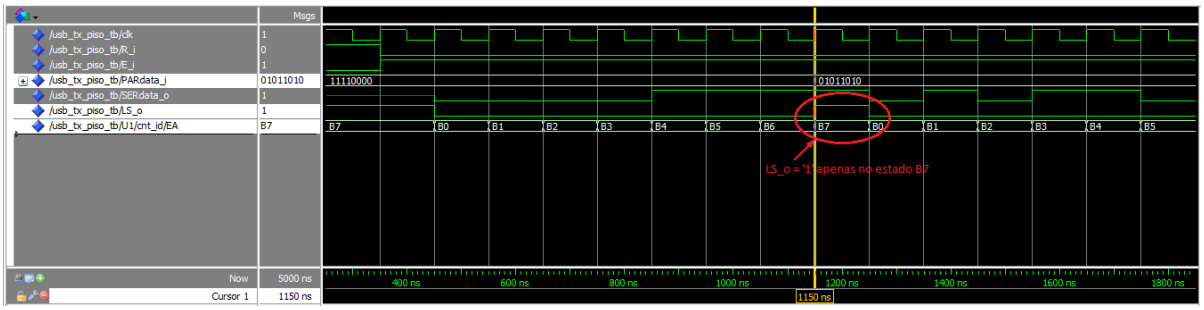


Fonte: Elaborada pelo autor.

Com o intuito de observar o comportamento do sinal interno do contador, presente no bloco **PISO**, a Figura 20 apresenta a simulação dos mesmos dados da Figura 19, com detalhe do sinal interno do contador, que comanda a carga de novos dados na entrada ou o deslocamento dos dados para a saída serial através do sinal **LS_o**. Quando o contador encontra-se no estado **B7**, o sinal **LS_o** = '1'.

Novamente a simulação é apresentada na Figura 21, desta vez, destacando o sinal interno do registrador de deslocamento **SERdata_r**. Neste detalhe, é possível notar o

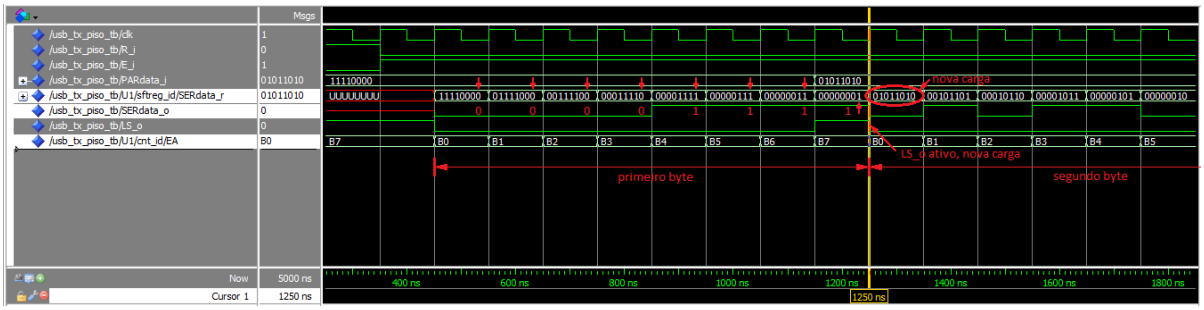
Figura 20 – Simulação do bloco **PISO**, detalhe do contador.



Fonte: Elaborada pelo autor.

momento da carga do registrador com os dados da entrada e do deslocamento dos dados no registrador. Agora se nota claramente que a saída do bloco recebe o bit menos significativo do registrador de deslocamento.

Figura 21 – Simulação do bloco **PISO**, detalhe do registrador de deslocamento.



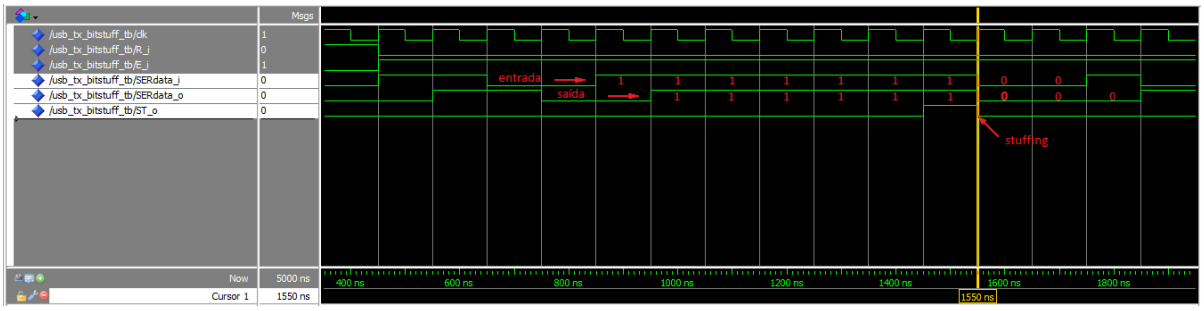
Fonte: Elaborada pelo autor.

4.2 SIMULAÇÃO DO BLOCO *BIT STUFFER*

Na simulação do bloco ***Bit Stuffer***, os dados de entrada foram escolhidos de modo a visualizar a ocorrência de um *stuffing* (a inserção forçada de um bit '0' após uma sequência de 6 bits '1' consecutivos).

Na Figura 22, encontram-se presentes na sequência de dados da entrada, sete bits '1' consecutivos, seguidos de dois bits '0' - "111111100". Na saída tem-se seis bits '1' consecutivos, seguidos de três bits '0' - "111111000". Aqui se pode notar que o sinal de saída será igual ao sinal de entrada se não houver um *stuffing*. E, no momento da inserção do bit '0' após o sexto bit '1' consecutivo, tem-se a "perda"de um bit da entrada, que não é "entregue" à saída.

Figura 22 – Simulação do bloco ***Bit Stuffer***.



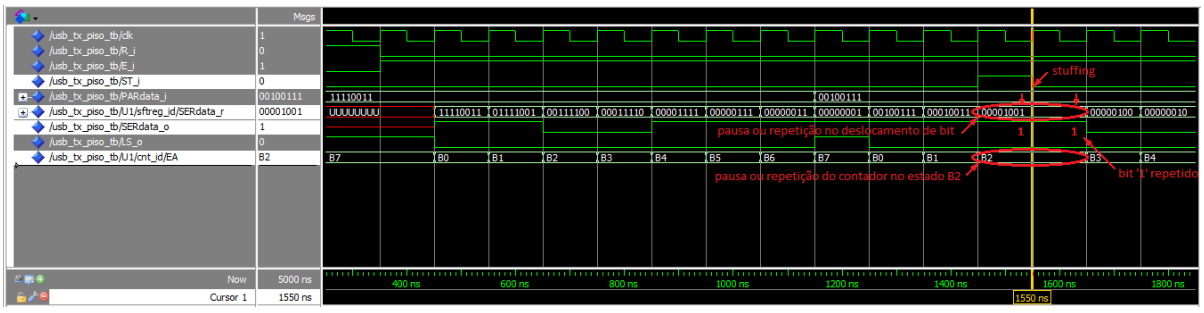
Fonte: Elaborada pelo autor.

Porém, ao ocorrer um *stuffing*, o número de bits deve aumentar, uma vez que existe a inserção de um bit extra na sequência de dados. Portanto, não deve ocorrer a "perda" de nenhum bit e a saída correta deveria ser de seis bits '1', um bit '0' (*stuffing*), um bit '1' (o sétimo bit '1' da entrada) e dois bits '0' - "1111110100".

A Figura 22 mostra a primeira simulação deste bloco e, para resolver o problema da "perda" de bits após um *stuffing*, optou-se pela solução de corrigir a entrada do bloco ***Bit Stuffer***, de modo que o bit sucessor ao *stuffing* fosse repetido. Ou seja, presentes na entrada do bloco ***Bit Stuffer*** (ou na saída do bloco **PISO**) devem aparecer oito bits '1' consecutivos (seis bits '1' do stuffing + dois vezes o bit sucessor '1' repetido), seguidos de dois bits '0'. Para tanto, utilizou-se o sinal **ST_i** (gerado no bloco ***Bit Stuffer*** como **ST_o**) como sinal de comando do bloco **PISO**, de modo que no momento do *stuffing* o bloco **PISO** seja desabilitado, reenviando o bit "perdido" para o bloco ***Bit Stuffer***.

Na Figura 23, é apresentada a simulação do bloco **PISO** utilizando o sinal **ST_o**, com dados de entrada que geram a sequência de saída para a entrada do bloco ***Bit Stuffer***. Nota-se que, no momento do *stuffing*, o registrador de deslocamento e o contador são desabilitados e o bit sucessor ao *stuffing* é repetido na saída.

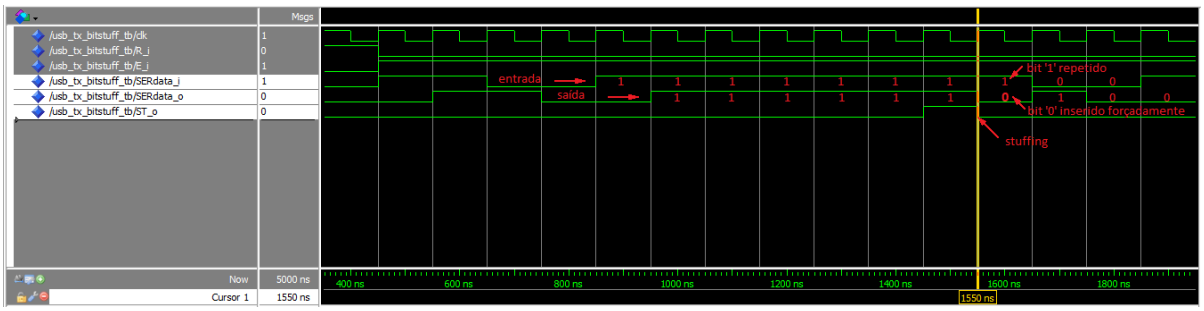
Figura 23 – Simulação do bloco **PISO**, corrigido com *stuffing* (**ST_i**).



Fonte: Elaborada pelo autor.

Depois de realizar os ajustes do bloco **PISO**, a simulação do bloco ***Bit Stuffer*** é apresentada novamente na Figura 24, chegando-se ao resultado correto e esperado.

Figura 24 – Simulação do bloco ***Bit Stuffer*** corrigido.

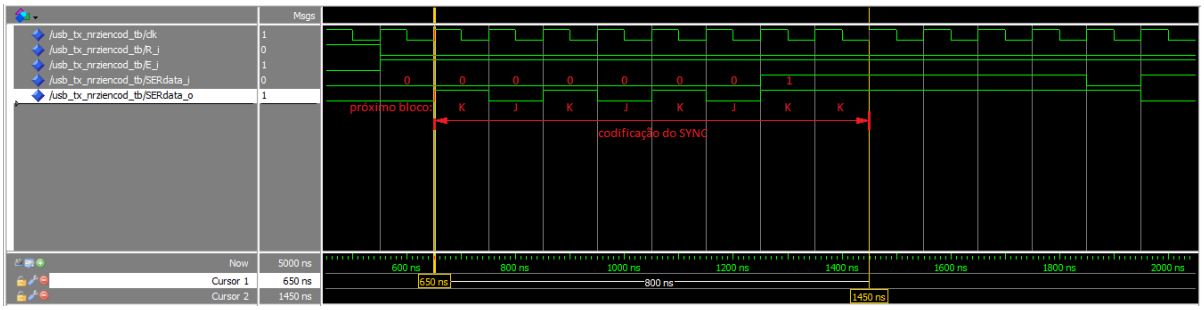


Fonte: Elaborada pelo autor.

4.3 SIMULAÇÃO DO BLOCO CODIFICADOR NRZI

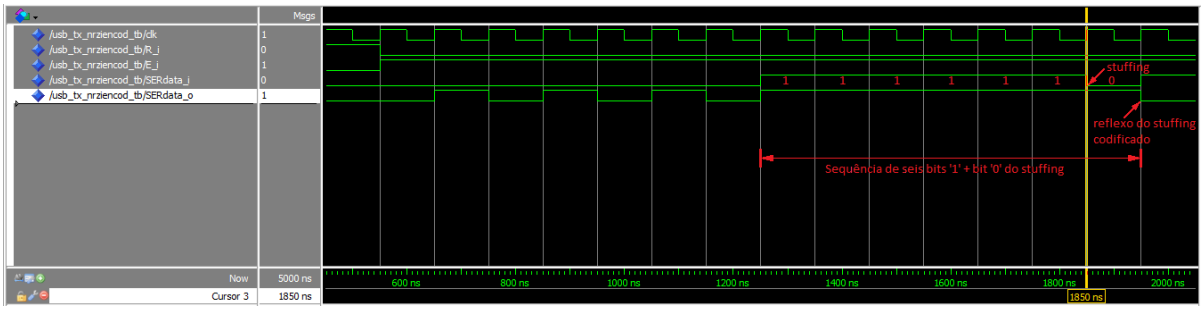
Na simulação do bloco **Codificador NRZI**, os dados de entrada foram selecionados de modo a emular o padrão SYNC (“00000001” - byte enviado no início de cada pacote de transmissão, com o bit menos significativo primeiro), acompanhado de um *bit stuffing*. Com isso, é possível observar na Figura 25 que o padrão SYNC começa a tomar forma codificada do byte de sincronização de *clock* (KJKJKJKK) na saída. Após o valor inicial (zero), os valores de saída alternam a cada *clock* sete vezes e o último valor ‘1’ do padrão SYNC (‘**K**’ repetido) marca o fim do padrão. Além disso, é possível observar no detalhe da Figura 26 o efeito de um *stuffing* na codificação, da sequência de entrada de seis bits ‘1’ consecutivos, seguida de um bit ‘0’.

Figura 25 – Simulação do bloco **Codificador NRZI**.



Fonte: Elaborada pelo autor.

Figura 26 – Simulação do bloco **Codificador NRZI**, detalhe *stuffing*.



Fonte: Elaborada pelo autor.

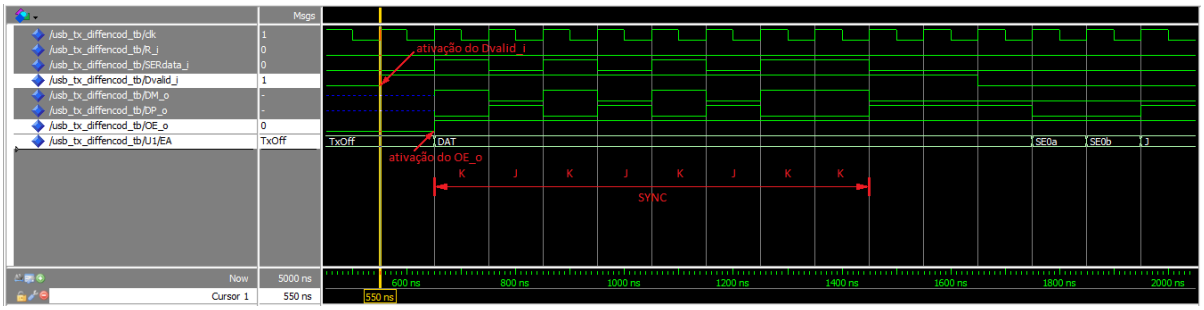
4.4 SIMULAÇÃO DO BLOCO CONVERSOR DIFERENCIAL

Com a ativação do sinal de comando **Dvalid_i** no bloco **Conversor Diferencial**, o sinal de status *Output Enable* (**OE_o**) é ativado e os dados diferenciais (na saída do bloco) são passados para o cabo USB.

Com o encerramento de dados a serem transmitidos pelo dispositivo, o sinal de comando **Dvalid_i** é desativado e o pacote de encerramento - *End of Packet* (**EOP**) é enviado. Ao concluir o envio do **EOP**, o sinal de status **OE_o** é desativado, marcando o fim da transmissão. Enquanto **OE_o** está desativado, não importam os valores das linhas diferenciais **DM_o** e **DP_o**, pois um estado 'J' (*IDLE*) ocorre naturalmente (está relacionado com resistores pull-up/pull-down do USB, não fazendo parte do escopo desse trabalho).

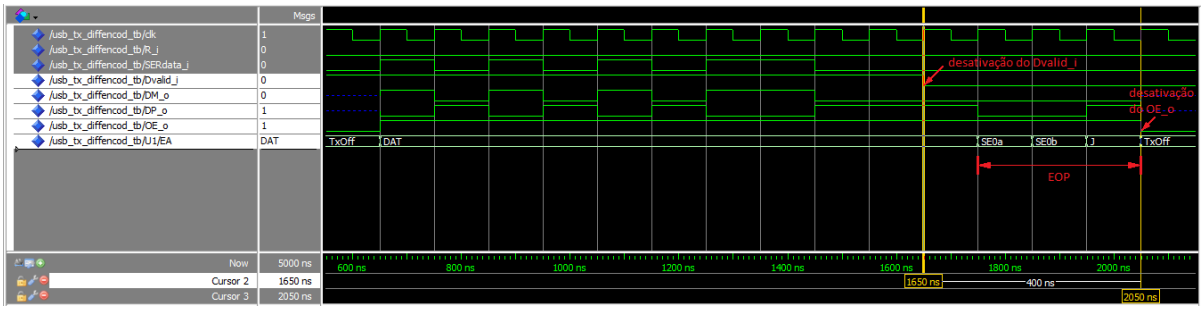
Na Figura 27, pode-se observar a ativação do sinal de status **OE_o**, iniciada com o byte de sincronização de *clock* nas linhas diferenciais, **DM_o** e **DP_o**, após a ativação do sinal de comando **Dvalid_i**. Na Figura 28, após a desativação do **Dvalid_i**, observa-se nas linhas diferenciais o pacote de encerramento (**EOP**), encerrando a ativação do **OE_o**.

Figura 27 – Simulação do bloco **Conversor Diferencial**, ativação *Output Enable*.



Fonte: Elaborada pelo autor.

Figura 28 – Simulação do bloco **Conversor Diferencial**, encerramento *Output Enable*.



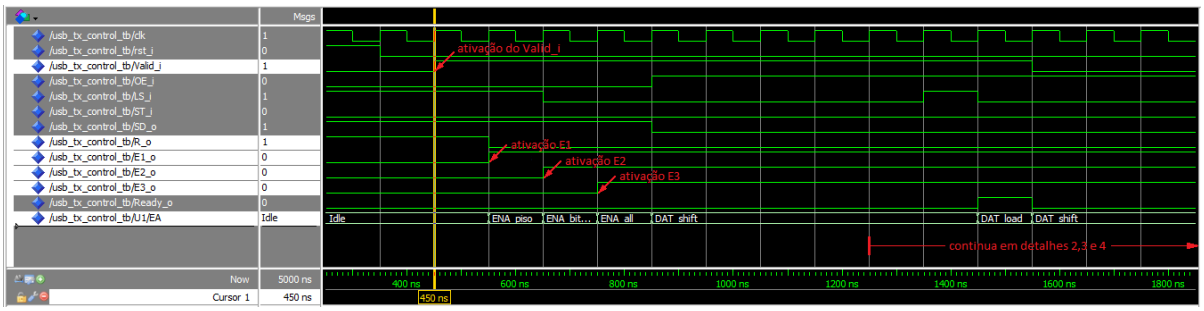
Fonte: Elaborada pelo autor.

4.5 SIMULAÇÃO DO BLOCO DE CONTROLE DO TRANSMISSOR

A simulação no bloco **Controle do Transmissor** foi realizada a fim de se observar alguns efeitos dos sinais de *status* (entradas) sobre os sinais de comando (saídas) do bloco.

O sinal **Valid_i** indica a presença de dados a serem transmitidos e, ao ser ativado, o bloco sai do estado inativo (*Idle State* - o qual mantém o reset (**R_o**) dos blocos controlados ativo), passando pelos estados “ENA piso”, “ENA bitstuff”, “ENA all”, desativando o reset dos outros blocos e ativando os sinais de controle **E1_o**, **E2_o** e **E3_o**, respectivamente, em cada estado, como se pode observar na Figura 29.

Figura 29 – Simulação do bloco de **Controle do Transmissor**, detalhe 1.

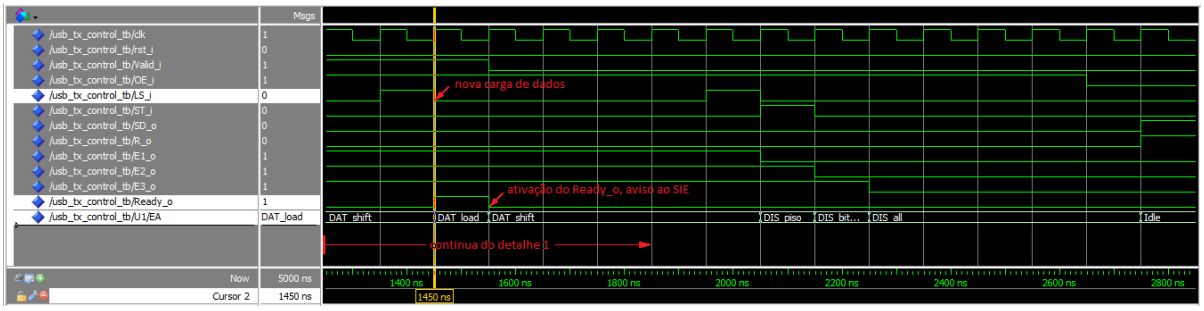


Fonte: Elaborada pelo autor.

O sinal **LS_i**, ao ser ativado, indica que uma carga de dados foi completamente deslocada e, enquanto o sinal **Valid_i** estiver ativo, uma nova carga de dados será realizada para a transmissão. Nesse caso, como mostra a Figura 30, pode-se observar que o bloco de **Controle do Transmissor** vai para o estado “DAT load”, efetuando o novo carregamento, e o sinal de saída **Ready_o** será ativado, indicando ao SIE que os dados foram carregados e novos dados devem estar a disposição para transmissão.

Quando não existem novos dados a serem enviados, **Valid_i** é então desativado.

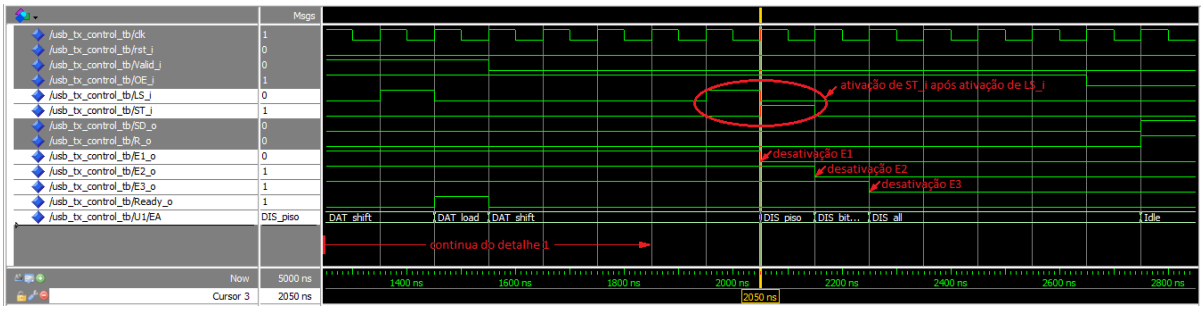
Figura 30 – Simulação do bloco de **Controle do Transmissor**, detalhe 2.



Fonte: Elaborada pelo autor.

Portanto, na próxima ativação do sinal **LS_i**, o bloco de **Controle do Transmissor** não vai para o estado "DAT load" e se inicia o procedimento de encerramento de pacote, para finalizar a transmissão. A Figura 31 mostra uma situação específica dessa simulação, em que ocorre um *stuffing* - ativação do sinal **ST_i** - logo após a ativação de **LS_i** e enquanto o sinal **Valid_i** já está desativado. Desse modo, o bloco de **Controle do Transmissor** passa pela sequência de estados "DIS piso", "DIS bitstuff" e "DIS all", desativando os sinais **E1_o**, **E2_o** e **E3_o**, respectivamente, em cada estado. Sem essa condição específica (em que ocorre um *stuffing* após a conclusão do último deslocamento de dados), o bloco de controle não precisa de um estado extra para o *stuffing*, "pulando" do estado "DIS piso" para o estado "DIS all", de maneira que primeiramente **E1_o** é desativado e, em seguida, **E2_o** e **E3_o** são desativados juntos.

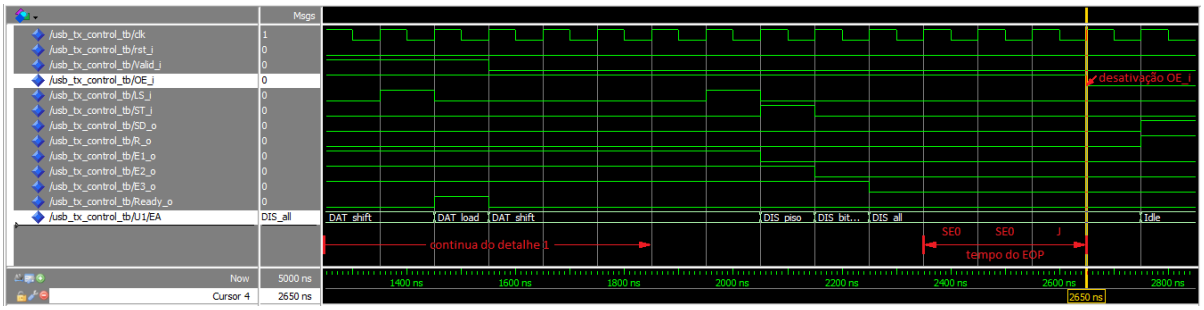
Figura 31 – Simulação do bloco de **Controle do Transmissor**, detalhe 3.



Fonte: Elaborada pelo autor.

Por fim, na Figura 32, o bloco de **Controle do Transmissor** aguarda pela desativação do sinal **OE_i** - indicador de que todos os dados foram transmitidos, inclusive o pacote de encerramento - **EOP**. Assim, a transmissão é encerrada, com o retorno do bloco de controle para o estado "IDLE", à espera do recomeço de um novo ciclo de transmissão.

Figura 32 – Simulação do bloco de **Controle do Transmissor**, detalhe 4.



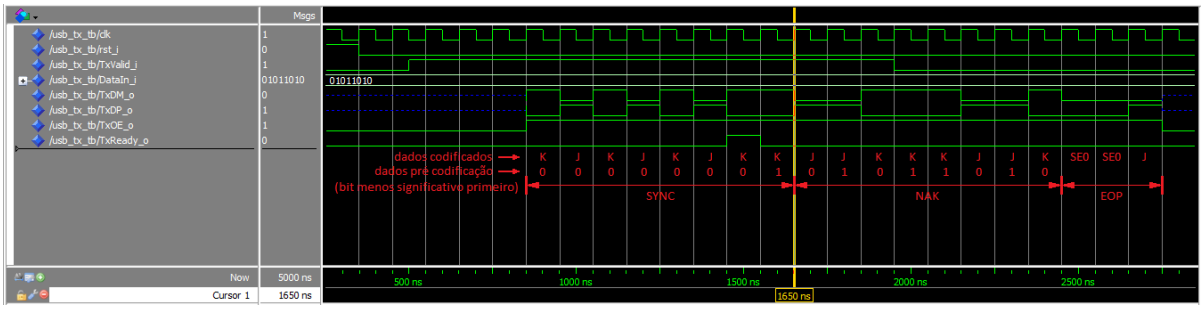
Fonte: Elaborada pelo autor.

4.6 SIMULAÇÃO DO BLOCO TRANSMISSOR

A simulação desta seção é bem simples e tem como objetivo observar o bloco **Transmissor** operando como um todo.

A Figura 33, simulação do bloco **Transmissor** - composto pelos blocos vistos anteriormente, apresenta um exemplo de um *Negative Acknowledgment packet* (NAK), um pacote simples de comunicação USB. NAK é uma das possíveis respostas do dispositivo ao *Host* e é composto pelos campos SYNC (KJKJKJKK), seguido do *Packet Identifier* (PID) - NAK (01011010) e pelo EOP (2x**SE0** + 1x**J**), nas linhas diferenciais de saída do **Transmissor**.

Figura 33 – Simulação do bloco **Transmissor**, NAK.



Fonte: Elaborada pelo autor.

Todos os sinais internos foram ocultados e apenas os sinais de entrada e saída do bloco **Transmissor** aparecem na simulação.

5 CONSIDERAÇÕES FINAIS

A pesquisa bibliográfica sobre os requisitos impostos pelo padrão USB possibilitou a compreensão sobre a especificação dos blocos do sistema, além da descrição das funções comportamentais deles. E, juntamente com as simulações, pôde-se projetar os cinco blocos digitais (**PISO**, ***Bit Stuffer***, **Codificador NRZI**, **Conversor Diferencial** e **Controle do Transmissor**) que compõem o módulo **Transmissor** USB.

A construção dos blocos digitais e do sistema desenvolvido foi realizada através da divisão explícita em *Datapath* (caminho de dados) e *Control Path* (caminho de controle). Ou seja, os blocos não trocam sinais de controle entre si, sendo esta a principal função do bloco de controle, tornando-o o bloco desenvolvido mais complexo. No entanto, entre os blocos **PISO** e ***Bit Stuffer***, optou-se em realizar um controle direto com o sinal **ST** - *Stuffing*, uma vez que, desta forma foi possível corrigir o problema da perda de bit durante o *stuffing* com mais praticidade, conforme mencionado na Seção 3.2.1 e mostrado na Figura 23 da Seção 4.2.

A maior dificuldade encontrada no projeto foi a implementação do bloco de **Controle do Transmissor**, pois, como mencionado anteriormente, apresentou mais complexidade em sua construção, além da necessidade de diligência nas simulações de integração com os blocos do módulo **Transmissor**, devido ao fato de não haver um modelo pré definido. Ainda assim, se obteve êxito no trabalho, uma vez que, ao final, o módulo **Transmissor** teve sua funcionalidade adequada, conforme os requisitos do sistema USB.

Por fim, pode-se concluir que este trabalho foi importante para melhor compreensão sobre o tema abordado, gerando desenvolvimento pessoal, acadêmico e, consequentemente, profissional.

5.1 SUGESTÕES PARA TRABALHOS FUTUROS

A partir deste trabalho, sugere-se:

- Primeiramente, a especificação e projeto de blocos digitais para um módulo receptor USB, concluindo um módulo Transceptor USB.

- A implementação dos módulos Transmissor e Receptor em FPGAs - *Field Programmable Gate Arrays*, a fim de não se limitar às simulações e realizar testes com dados reais de comunicação entre as placas FPGAs (transmissor enviar - receptor receber).

- Integrar, por fim, transmissor e receptor em um único módulo USB, sintetizado e testado em circuito integrado.

REFERÊNCIAS

ANDERSON, Don; DZATKO, Dave. **Universal Serial Bus System Architecture**. 2. ed. Boston: Addison-Wesley Longman Publishing Co., Inc., 2001.

AXELSON, Jan. **USB Complete: Everything You Need to Develop USB Peripherals**. 3. ed. Madison: Lakeview Research LLC, 2005.

BABULU, K.; RAJAN, K. S. **FPGA Implementation of USB Transceiver Macrocell Interface with USB2.0 Specifications,** First International Conference on Emerging Trends in Engineering e Technology. Nagpur, jul. 2008, P. 966-970. DOI: 10.1109/ICETET.2008.77.

FOROUZAN, B. A. **Comunicação de Dados e Redes de Computadores**. 4. ed. Porto Alegre: AMGH Editora, 2009.

LECHEMINANT, Greg D. **The evolution of test methodologies for high-speed digital communications components,** 24th Annual Technical Digest Gallium Arsenide Integrated Circuit (GaAs IC) Symposiu. Monterey, out. 2002, P. 203-206. DOI: 10.1109/GAAS.2002.1049060.

LUEKER, J. *et al.* **Universal Serial Bus Specification**: Revision 2.0. 2000. Disponível em: https://www.usb.org/document-library/usb-20-specification.

MCGOWAN, Steve. **USB 2.0 Transceiver Macrocell Interface (UTMI) Specification**: Version 1.05. 2001. Disponível em: https://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/usb2-transceiver-macrocell-interface-specifications.pdf.

NAM, J. J. *et al.* **A UTMI-Compatible Physical-Layer USB2.0 Transceiver Chip,** International [Systems-on-Chip] SOC Conference. Portland, set. 2003, P. 309-312. DOI: 10.1109/SOC.2003.1241532.

RUSCHEL, O. T. **Princípios da Comunicação Digital**. Porto Alegre: EDIPUCRS, 1996. v. 3.

SHIN, K. *et al.* **Verilog Synthesis of USB 2.0 Full-Speed Device PHY IP,** International SoC Design Conference (ISOCC). Busan, nov. 2013, P. 162-165. DOI: 10.1109/ISOCC.2013.6863961.

SZECÓWKA, P. M.; PYRZYNSKI, K. J. **USB Receiver/Transmitter for FPGA Implementation,** International Conference on Signals e Electronic Systems (ICSES). Wroclaw, set. 2012, P. 1-6. DOI: 10.1109/ICSES.2012.6382226.