**UNIVERSIDADE ESTADUAL PAULISTA**

**“JÚLIO DE MESQUITA FILHO”**

**Câmpus de Ilha Solteira - SP**

**VICTOR AFONSO DOS REIS**

**PROF. DR. AÍLTON AKIRA SHINODA**

**(Orientador)**

**ESTUDO E DESENVOLVIMENTO DA CODIFICAÇÃO 8B10B EM VHDL**

**APLICADA À SISTEMAS DE ALTAS ENERGIAS**

RELATÓRIO DE INICIAÇÃO CIENTÍFICA

FUNDUNESP

**Ilha Solteira - SP**

**2017**

**Resumo**

Em sistemas com alta taxa de eventos, onde a informação provém de um grande número de sensores e os eventos de interesse são raros, usualmente uma codificação 8b10b é recomendada para implementação no canal de transmissão.

Para uma sequência digital gerada e transmitida por fibra ótica em alta velocidade, pode ocorrer uma série de problemas na transmissão do dado. Estes problemas são caracterizados por ruídos devido a radiações, interferências eletromagnéticas ou ionizações indesejáveis e uma de-sincronização entre o transmissor e receptor dada por uma longa sequência de zeros (0’s) ou um (1’s) no canal de transmissão. Esta longa sequência interfere nos circuitos adicionais presentes no canal de transmissão de realizarem a sincronização, sendo necessário realizar um balanço nos bits (1’s) e nos bits (0’s) transmitidos. Neste projeto é realizado um estudo da codificação 8b10b através de um sistema que implementa a codificação no *software* MatalabTM dentro do ambiente do Simulink e um estudo da linguagem de descrição de *harware* VHDL juntamente com uma implementação da codificação nesta linguagem.

Com a implementação dos sistemas é possível realizar um balanceamento dos dados e a detecção de erros no canal de transmissão. Com o balanceamento dos dados possibilita a recuperação do *clock,* sincronizando o dispositivo transmissor e receptor, ao mesmo tempo possibilita um canal de transmissão mais confiável por conta da detecção dos erros.

**SUMÁRIO**

Capítulo 1……………………………………………………………………………………………2

Introdução..………………………………………………………………………………….2

1.1 Motivação..……………………………………………………………………………...21.2 Trabalho Desenvolvido….…………………………………………………………….5

Capítulo 2………………………………………..………………………………………………….6

Colisor LHC e o Laboratório CERN………………………………………………………6

2.1 O LHC………………………………………………………...…………………………7

2.3 O detector CMS………………………………………………………………………..8

2.4 O ambiente de Física de Altas Energias………………...…………………………11

2.5 O Sistema Level 1 Trigger………………………………...…………………………15

Capítulo 3………………………………………………………………..………………………...18

Codificação 8b/10b……………………………………………………………………….18

3.1 Transmissões Digitais em Alta Velocidade………………………………………...18

3.2 Codificação 8b/10b…………………………………………………………………...23

3.3 Algoritmo desenvolvido para a simulação e descrição do sistema da codificação 8b10b……………………………………………………….………………………..28

3.4 Sistema implementado no MATLABTM (SIMULINK)…...………………………..32

Capítulo 4………………………………………………………………..………………………...40

Sistema implementado no VHDL………………………..……………………………...40

4.1 A linguagem de descrição de hardware VHDL……………………………………40

4.2 Field Programmable Gate Array (FPGA)………….……………………………….43

4.2.1 Estrutura Interna do FPGA………………..……………………………...46

4.2.2 Técnicas de Programação do FPGA……………………………………..51

4.3 Sistema implementado no VHDL…..……………………………………………….53

Capítulo 5……………………………..…………………………………………………………...55

Resultados e Conclusões Finais………………………………………………………………..55

Referências Bibliográficas……..……………………………….………………………………..65

Anexo I…………………..………………………………………………………………………...69

Anexo II…………………..………………………………………………………………………..74

Anexo III………………...…………………………………………………………………………79

Anexo IV………………...…………………………………………………………………………86

Anexo V………………………..…………………………………………………………………..97

**CAPÍTULO 1**

**INTRODUÇÃO**

Em ambientes modernos, diversos recursos tecnológicos tendem a trocar dados a pequenas ou longas distâncias. Dessa forma, deve-se garantir uma alta confiabilidade dos dados na transmissão, de forma que no lado do receptor sejam recebidos o mesmo dado enviado pelo transmissor. Esta preocupação ocorre devido aos problemas apresentados nas transmissões como por exemplo: a atenuação do sinal, dessincronização entre o transmissor e o receptor e ruídos apresentados no canal (MACHADO,2016). Para amenizar os efeitos da dessincronização e dos ruídos, desenvolveu-se codificações de linha a fim de aumentar a confiabilidade da transmissão. Dessa forma, torna-se possível a detecção de erros e a implantação de circuitos que gerem uma sincronização entre os dispositivos comunicantes.

Uma codificação inteligente em um canal de transmissão, possibilita transmitir uma maior quantidade de dados por unidade de tempo (COMER,2016). Estas codificações são extremamente úteis em sistemas para física de altas energias. Portanto, a presença de um *clock* elevado e o interesse de uma alta confiabilidade no canal, necessita-se implementar uma codificação no canal de transmissão. Esta codificação deve fornecer mecanismos para identificar possíveis erros, juntamente com a inserção de circuitos corretores de erros, e que possibilitem que circuitos externos sincronizem os dispositivos comunicantes.

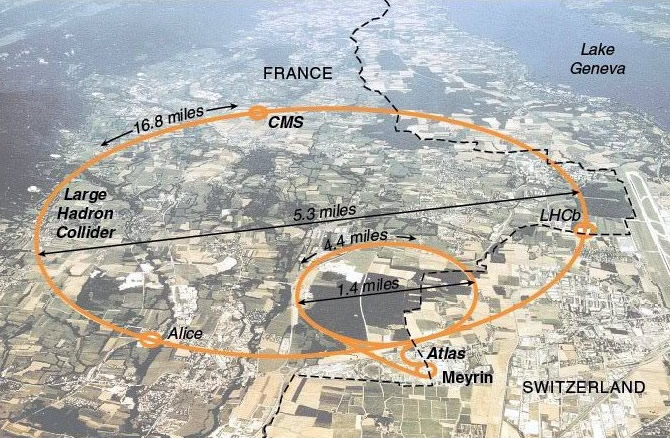
**1.1 MOTIVAÇÃO**

Este trabalho é parte de uma colaboração com o laboratório “*São Paulo Research and Analysis Center*” (SPRACE) (SPRACE,2017) ao qual realiza um projeto de pesquisa com o grupo CMS/*Level 1 Tracking Trigger* (L1TT) no “*Fermi National Accelerator Laboratory*” (Fermilab) (PAIVA,2016). Um dos proprósitos do Fermilab é participar da grande atualização que irá ocorrer nos próximos anos no “*Larger Hadron Collider*” (LHC) de forma a aumentar a taxa de colisão dentro do colisor.

O LHC tem uma extensão de 27 km de circunferência, localizado na fronteira Franco-Suiça e o colisor tem por objetivo descobrir a origem da massa das partículas elementares e outras dimensões do espaço (WIKIPÉDIA,2017). O colisor é o maior equipamento já construído para pesquisa em física de altas energias do mundo, obtendo resultados expressivos como a descoberta do Bóson de Higgs. Este Bóson é uma partícula elementar prevista pelo modelo padrão de partículas e que ajuda a explicar a massa de outras partículas elementares (RANDALL,2013).

No percurso do colisor há 4 detectores : ATLAS, CMS, Alice, e LHCb. O acelerador de partículas fornece velocidade e os detectores captam os produtos do impacto das partículas. Dessa forma, pode-se observar a existência de traços de partículas elementares que explicam teorias importantes sobre a física de altas energias (FERREIRA,2009). Na figura 1.1 é apresentado um esquema do grande colisor de Hádrons que está localizado a 175 metros abaixo do solo.

Figura 1.1 – Esquema do Grande Colisor de Hádrons.



Fonte: (TEMPLETON, 2017).

Para aumentar o progresso das descobertas, o LHC desde 2015 está em um processo de atualização denominado de “Grande Luminosidade” ao qual irá aumentar o número de colisões gerando mais dados para estudos. O grande volume de dados torna-se um problema pela necessidade de armazenamento para serem estudados posteriormente, tendo em vista que alguns dados gerados já existem estudos concluídos e resolvidos ou até não são objeto de estudo no momento. Os detectores devem serem atualizados para suportar o aumento dos dados gerados no colisor. Portanto, deve-se desenvolver novas tecnologias para filtrar o grande volume de dados armazenando somente o necessário. Os dispositivos que serão implementados para trabalhar com esses dados deverão serem projetados para trabalharem com alta velocidade que os dados são produzidos (BRÜNING et al., 2017).

O trabalho desenvolvido na colaboração entre o SPRACE e Fermilab está diretamente ligado ao detector “Solenóide de Muon Compacto” (CMS) do LHC. Os detectores do LHC tem estruturas diferentes e cada um obtém dados de partículas específicas, dessa forma a junção de todos os dados de todos os detectores formam uma imagem completa do experimento ajudando a realizar novas descobertas. O detector tem 6 metros de diâmetro e 13 metros de comprimento captando ao longo do diâmetro as posições das partículas. Dessa forma, é possível traçar o caminho das partículas por produzirem dados de posições ao longo do diâmetro do detector. As partículas carregadas seguirão caminho em espiral no campo magnético de 4(T) do detector possibilitando calcular os seus momentos, uma vez que momentos diferentes indicam partículas diferentes. Portanto, por meio desses dados é possível coletar evidências para comprovar as teorias de novas partículas (TSESMELIS, 2017).

A colaboração entre o SPRACE e o Fermilab tem o objetivo de ajudar no desenvolvimento de um novo sistema para o detector CMS que realize o filtro dos dados produzidos no detector. O atual sistema do detector CMS possui restrições contra o aumento da taxa de dados, uma vez que atualmente o sistema precisa de 3,4 μs de latência para processar os dados do detector e filtrá-los. O processamento deve-se na tentativa de coletar os dados de pontos no detector e traçar o caminho da partícula, posteriormente filtra-se o dado armazenando somente os de interesse. Com o aumento das colisões estima-se que o novo sistema deverá ter uma latência de 12,5 μs para processar os dados. O sistema que está em desenvolvimento é denominado “*Level 1 Track Trigger*” (L1TT) e objetiva eliminar as restrições que o atual sistema possui (CMS Collaboration, 2015).

O sistema em desenvolvimento no projeto do Fermilab com a colaboração do SPRACE é o L1TT baseado em memórias associativas com “*Field Programmable Get Array*” (FPGA) que se aproxima do sistema L1TT. Como os laboratórios não possuem a disponibilidade do detector, está em desenvolvimento um sistema em um FPGA que emula os dados provindos do detector CMS. Este sistema foi definido como “Data Source System” (DSS). Portando, há duas placas FPGA se comunicando, uma com o DSS que emula os dados e a outra com o sistema L1TT que realiza o filtro do sinal (PAIVA,2016).

**1.2 TRABALHO DESENVOLVIDO**

No ambiente do detector há uma alta taxa de radiação eletromagnética por conta da alta velocidade dos átomos presentes no tubo, dessa forma em transmissões de alta velocidade os dados transmitidos de uma placa para outra podem acarretar a presença de ruídos no canal de transmissão. Os ruídos presentes no canal de transmissão danificam os dados originais, acarretando no armazenamento de dados incoerentes para um posterior estudo. Portanto, uma codificação presente no canal de transmissão possibilita a detecção de erros e a solução de problemas envolvidos na transmissão em altas velocidades como por exemplo a dessincronização entre o transmissor (DSS) e o receptor (L1TT).

Este trabalho desenvolve o estudo da codificação 8b10b e suas características por meio do Simulink(MATLAB) e em conjunto realiza o estudo da linguagem de descrição de Hardware “Very High Speed Integrated Circuits(VHSIC) Harware Description Hardware” (VHDL) descrevendo um sistema baseado na codificação 8b10b na linguagem VHDL. Dessa forma, o sistema desenvolvido pode ser usado como codificação do canal de transmissão entre as duas placas FPGA garantindo uma maior confiabilidade dos dados transmitidos.

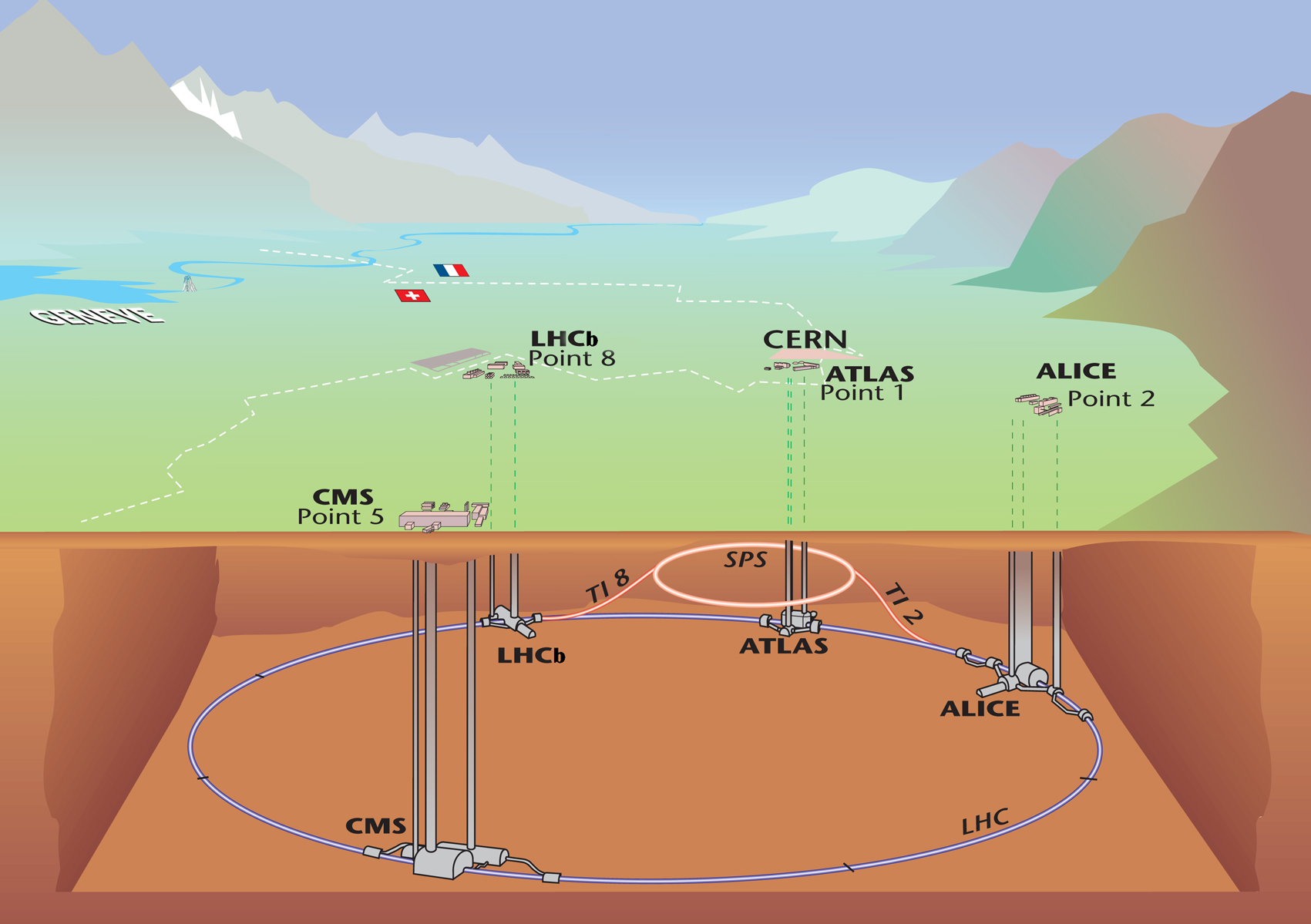
**CAPÍTULO 2**

**COLISOR LHC E O LABORATÓRIO CERN**

A física de altas energias trata da parte experimental da física, ao qual tem o objetivo de estudar as quatro interações fundamentais da natureza (Gravitacional, Eletromagnetismo, Fraca e Forte) e as partículas elementares descritas no modelo padrão.

Por volta do final da segunda guerra mundial, pesquisadores Europeus tiveram a ideia pioneira de montar um laboratório que contaria com a colaboração de pesquisadores do mundo inteiro e que os países participantes dividissem a despesa do experimento nuclear. No ano de 1950 foi proposto para Organização das Nações Unidas para a Educação, a Ciência e a Cultura (UNESCO) a criação de laboratórios regionais, ao qual um deles era específico para a ciência nuclear. Dois anos mais tarde cientistas europeus continuaram a proposta criando o “Conselho Europeu de Pesquisa Nuclear” (CERN), ao qual posteriormente virou uma organização. Em 1954 os organizadores decidiram juntar os esforços em um único laboratório em Geneva na Suíça, com a proposta central da criação de um enorme acelerador de partículas, conforme demonstrado na figura 2.1 (HEILBRON et al., 2003).

Figura 2.1 – Esquema do Grande Acelerador de Partículas denominado LHC.



Fonte: (BULLETIN, 2017).

O objetivo do CERN é estudar os fundamentos da matéria, especificamente as partículas físicas elementares. Portanto, o laboratório desenvolve, opera e constrói o acelerador e os detectores sendo ferramentas necessárias para evidenciar as novas partículas. Atualmente, a ajuda vem de vários países do mundo aos quais ajudam a comprovar a teoria do modelo padrão.

O laboratório também ajuda as outras áreas do conhecimento a desenvolverem produtos que é de grande ajuda atualmente. Dentro do CERN surgiu a “World Wide Web” (WWW) em 1990, sendo atualmente de grande ajuda para o mundo inteiro. No ambiente do laboratório também observa-se o desenvolvimento de tecnologias na área da Engenharia Civil, Eletrônica, Mecânica, bem como detectores de imagem para medicina e novos aceleradores, menos invasivos, para radioterapia (FERREIRA,2009).

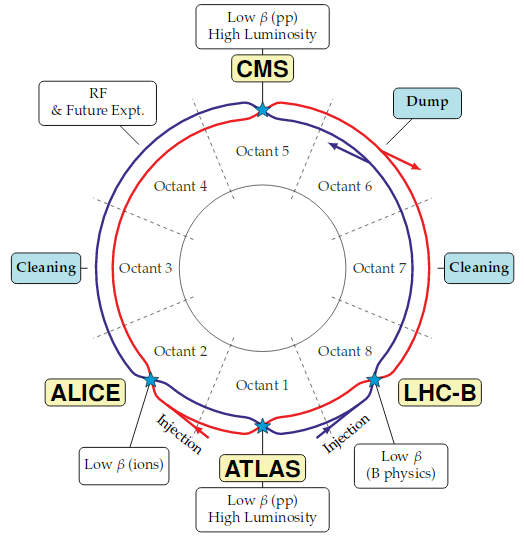
**2.1 O LHC**

Os aceleradores de partículas são máquinas criadas para fornecerem velocidade às partículas perto da velocidade da luz, podendo produzir matéria com a colisão das mesmas. Os aceleradores são classificados em circulares, os quais colidem dois feixes de partículas, ou lineares os quais colidem uma partícula contra um alvo visando estudar a estrutura da matéria. Para os aceleradores circulares ou mais conhecidos como síncronos, as partículas são aceleradas em sentidos contrários por meio de campos magnéticos intensos, fazendo-as colidirem produzindo matéria ao qual é o objetivo de estudo (LUIZ; MONTEIRO; BATISTA, 2017).

O LHC é um colisor do tipo circular ou *síncronton* fornecendo colisões próton-próton a uma energia no centro de massa de aproximadamente 14 TeV. Para colisões chumbo-chumbo fornece uma energia aproximadamente de 5,52 TeV no centro de massa núcleon-núcleon. Os detectores ATLAS e CMS são de propósito geral e possuem estrutura de forma a observar qualquer evento físico acima de 1 TeV. Pela estrutura dos detectores de propósito geral possibilitarem captarem qualquer evento físico, é possível realizar estudos não somente sobre o modelo padrão mas de outros modelos propostos.

A luminosidade do colisor é dee é obtida pela aceleração até 7 TeV de grupos de prótons em sentidos opostos, resultando eminterações próton-próton por segundo. Para obter a colisão os feixes de luzes se cruzam em ângulo de aproximadamente de 150 a 200 μrad, possuindo 4 pontos de colisão no colisor de acordo com a figura 2.2 (TOMEI T. R. F. P., 2012).

Figura 2.2 – Descrição dos pontos de colisão em torno do colisor LHC.



Fonte: (TOMEI T. R. F. P., 2012).

**2.3 O DETECTOR CMS**

O detector CMS objetiva-se obter a lucidez do mecanismo de quebra da simetria eletrofraca, investigar teorias além do modelo padrão e obter medidas descritas no modelo padrão, como por exemplo a massa do quark T e do bóson W. A sua colaboração envolve milhões de cientistas e engenheiros de 40 países de 4 continentes. O CMS é constituído pelo detector propriamente dito e o sistema computacional que trata dos dados obtidos no detector. Os dados obtidos no detector provém de diversas camadas, obtendo as diferentes características das partículas produzidas. Estas características produzidas no detector CMS são: o Sistema de Detecção de Traços, o Calorímetro Eletromagnético, o Calorímetro Hadrônico e o Sistema de Múons, dispostos de forma concêntrica em relação à linha de feixe como pode-se visualizar na figura 2.3.

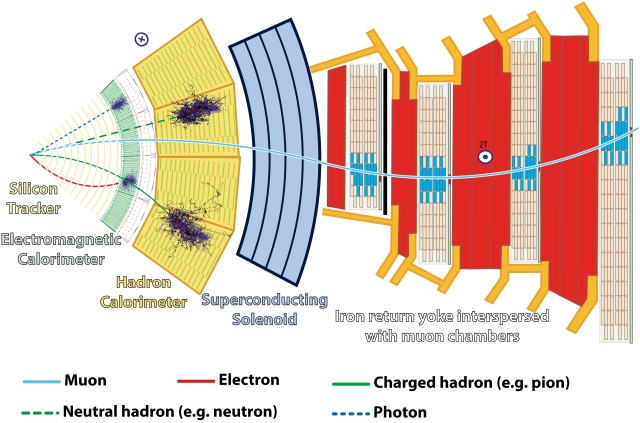
Os dados obtidos das camadas são digitalizados e combinadas para fornecerem uma visão geral da colisão. Estes dados digitais são gravados e algoritmos realizam a reconstrução do caminho da partícula nos sistemas computacionais. Porém, pela alta taxa de interações, em torno de , gera-se uma alta taxa de aquisição que gira em torno de 1 Petabyte por segundo. Por conta do grande volume de dados, é necessário a adoção de um sistema Trigger ao qual filtra em tempo real somente os dados que são interessantes do ponto de vista físico ou são objeto de estudo no momento. Os dados que são eliminados pelo Trigger são perdidos e os que são selecionados são armazenados permanentemente para posterior estudo.

As camadas do detector são dispostas de forma concêntrica em relação ao feixe de partículas e estima-se que a área de detecção cubra completamente 98% do ângulo azimutal 2π. Em torno das camadas de detecção de Traços, calorímetro eletromagnético e a camada do calorímetro hadrônico, há um solenóide supercondutor de 13m de comprimento e 5,9 de diâmetro interno, tendo uma corrente de 19,5 kA com um campo magnético resultante de 4 T.

O sistema de detecção ou “Silicon Tracker” representado na figura 2.3 , detecta a trajetória da partícula e determina o sinal de todas as partículas carregadas usando elementos de detecção e algorítimos de reconstrução do caminho. O calorímetro eletromagnético é responsável por medir a energia de fótons e elétrons, por meio de fotodiodos na região do barril e fototriodos, na tampa, sobre cristais de PbWO4 (chumbo-tungstato). Os elétrons e os fótons depositam toda a energia nos cristais de forma a gerar um chuveiro eletromagnético, com essa energia depositada nos cristais há um processo de reconstrução da mesma ao qual é feita de forma distinta para fótons e elétrons. As partículas de fóton, não possuindo carga, não deixam caminhos no sistema de detecção, portanto para reconstruir a energia do fóton procura-se traços obtidos nos cristais aos quais não possuem caminhos obtidos no sistema de detecção (Silicon Tracker) e para os elétrons procura-se traços que tenham caminhos formados no sistema de detecção.

O calorímetro Hadrônico absorvem energia, no material absorvedor do calorímetro, dos hádrons que o atravessam por meio de interação forte residual dando origem a chuveiros hadrônicos. O sistema de Múons é localizado fora do solenoide do detector, isto deve-se aos múons possuírem massa de aproximadamente 105 MeV e tempo médio de vida de 2,2 μs e portanto possuírem energia e tempo de vida suficiente para chegarem ao sistema de múons. Este sistema obtém a trajetória da partícula pelo mesmo algorítimo do sistema de detecção, possuindo duas partes a online e a offline ao qual possuem algorítimos de reconstrução distintos. Na figura 2.3 pode-se visualizar um esquema do detector CMS com suas várias camadas de instrumentação (TOMEI T. R. F. P., 2012).

Figura 2.3 – Subdetectores do CMS: Sistema de Traços, ECAL, HCAL, Sistema de Múons.



Fonte: (BARNEY, 2017).

**2.4 O AMBIENTE DE FÍSICA DE ALTAS ENERGIAS**

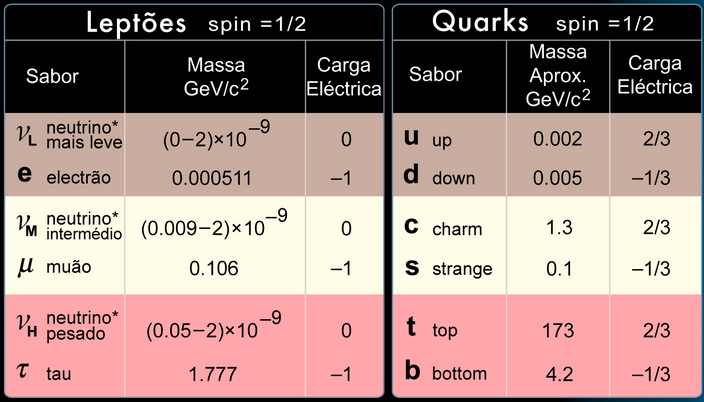
O modelo padrão de partículas foi elaborado durante a segunda metade do século XX, descrevendo o funcionamento das partículas elementares. Estas partículas são assim chamadas por não terem organização interna, ou seja, não são formadas por outras partículas sendo usadas para explicar a maioria dos fenômenos físicos atuais. O modelo padrão tem obtido grande sucesso por conta das atuais comprovações experimentais de sua teoria através dos experimentos no LHC, dessa forma pode-se dizer que o modelo padrão atualmente é o mais próximo dos dados obtidos experimentalmente.

Atualmente no modelo são descritas 17 partículas elementares divididas em 2 grupos: os bósons e os férmions. Para cada partícula elementar descrita há uma antipartícula equivalente possuindo carga inversa. Férmions são partículas as quais possuem spin semi-inteiros (½ ,3/2 , 5/2, ...) e Bósons são partículas que possuem spin inteiros (0,1,2, ...). O spin das partículas é o momento angular intrínseco existente dado em unidades da constante de planck (ℏ), ou seja o spin é um valor múltiplo da constante de planck (SPRACE,2017). A constante de planck equivale à:

…...……………… (1)

Ao todo há 12 Férmions que são divididos em *quarks* e *léptons* sendo constituidores de toda matéria conhecida. Os *quarks* são partículas que interagem pela força nuclear forte e são mais massivas, possuindo seis “sabores”: *up, down, charm, strange, top e o bottom*. Os *léptons* possuem spin semi-inteiros e carga elétrica inteira ou nula, tendo o elétron como a partícula elementar mais conhecida. Na figura 2.4 é mostrado um esquema dos símbolos e as cargas das partículas que representam os *quarks*. *Quarks* e *léptons* são divididos em 3 gerações e as duas primeiras linhas da tabela da figura 1.5 representam a I geração, a terceira e quarta representa a II geração, a quinta e sexta representa a III geração. A primeira geração é representada por elementos comuns no meio como prótons, elétrons e nêutrons, os quais são elementos estáveis. A segunda e terceira geração são partículas instáveis por decaírem facilmente e só são observadas por experimentos, como os realizados no LHC (SPRACE,2017).

Figura 2.4 – Tabela das massas e cargas das partículas elementares do grupo dos fermions.



Fonte: (PROJECT,2017).

Os *quarks* são partículas formadoras de *hádrons* aos quais são divididos em duas classes: *mésons* e *bários*. Os *quarks* se diferenciam por possuírem cargas de cor divididas em três tipos: vermelha, azul e verde. Cada partícula de *quark* pode ser de cada uma das cargas de cores e a junção das 3 cores formam uma partícula de carga de cor nula. Para formar os *prótons* unem-se duas partículas de *quark up* e uma partícula de *quark down*, cada uma com carga de cor diferente. Já para os *nêutrons* unem-se dois *quarks down* e um *up*. A formação dos *hádrons* dá-se de dois modos: junção de um *quark* com um *anti-quark* formando um *méson*, ou a junção de três quarks (ou *anti-quarks*) com cargas de cor distintas entre si formando um *bárion* (SPRACE,2017).

Na figura 2.5 pode-se visualizar exemplos de *bárions* com suas respectivas massas, cargas elétricas e composição das partículas de *quarks*. Ressalta-se que os *bários* possuem carga de cor neutra.

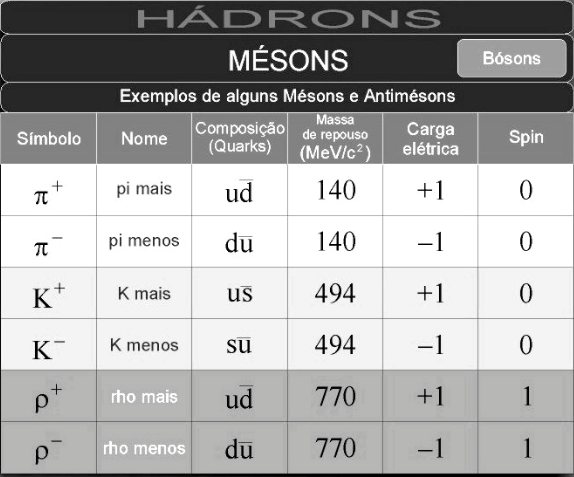
Figura 2.5 – Tabela das partículas de *Bárions*.



Fonte: (OSTERMANN; CAVALCANTI, 2017).

Na figura 2.6 pode-se visualizar exemplos de *mésons* com suas respectivas massas, cargas elétricas, composição das partículas de quarks.

Figura 2.6 – Tabela das partículas de *Mésons*.



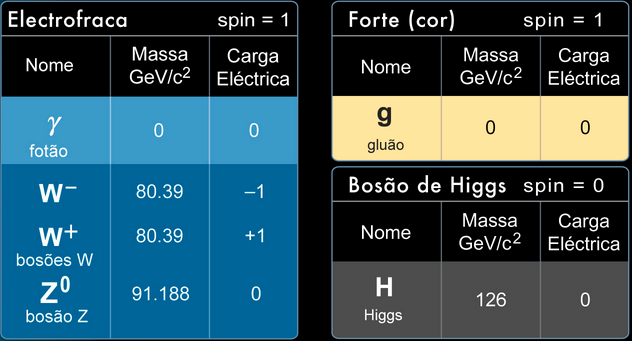
Fonte: (OSTERMANN; CAVALCANTI, 2017).

Diferentemente dos *quarks*, os *léptons* não possuem carga de cor e somente três partículas possuem carga elétrica: os elétrons, os *muons* e os *taus*. Os *léptons* que não possuem massa são definidos como neutrinos e possuem três sabores de acordo com a tabela da figura 2.4. O estudo dos neutrinos são de extrema importância pois ajudam a compreensão da *matéria-antimatéria* e a compreensão do surgimento das estrelas e galáxias. Observa-se que a massa das partículas é dada em GeV/c² (PROJECT,2017). Essa unidade corresponde à:

………………..……... (2)

Sabe-se que na natureza há 4 tipos de interações fundamentais: a gravitacional, a eletromagnética, forte e a fraca. Essas interações são possíveis de serem realizadas graças as partículas mediadoras, que são capazes de serem trocadas entre um conjunto de partículas elementares produzindo forças e campos fundamentais. Na interação eletromagnética a partícula relacionada é o *fóton*, na interação forte o *glúon* ena interação fraca as partículas W, Z e os *grávitons* (ainda não identificados) relacionados com a interação gravitacional. Portanto, as partículas da matéria física interagem entre si pelas interações fundamentais. Esta interação é possível por conta da troca de partículas mediadoras realizada entre si. Na figura 2.7 é apresentado uma tabela com as partículas mediadoras e as suas respectivas massas, cargas elétricas, e spins (MOREIRA, 2017).

Figura 2.7 – Tabela das partículas mediadoras.

Fonte: (PROJECT,2017).

Os *bósons* são responsáveis pela interação entre os férmions, tendo especificamente o *bóson de higgs* a partícula responsável pela massa de todas as outras partículas. Esta partícula é o elemento fundamental do modelo padrão pois sem ela as outras partículas não teriam massa uma vez que não havia explicação de como as partículas adquirem massa (SPRACE,2017).

**2.5 O SISTEMA *LEVEL 1 TRIGGER***

O LHC foi projetado para realizar colisões a cada 25 ns, possuindo uma taxa de cruzamento de 40 MHz, dessa forma, dentro do detector é geradointerações próton-próton por segundo em um feixe com luminosidade nominal depossuindo aproximadamente 20 colisões simultâneas. Essa taxa de dados é imensamente alta para o sistema de armazenamento, sendo impossível armazenar essa quantidade de dados na tecnologia atual. Portanto, há a necessidade de implementação de um sistema *trigger* que realize a filtragem dos dados “interessantes”, reduzindo a taxa dados armazenados para 100 Hz.

As colisões geram dados em um conjunto de sensores no detector denominados *Outer Silicon track*. Este sistema possui seis linhas de sensores em que a partícula passa gerando dadosde acordo com a figura 2.9. A partícula ao atravessar os sensores determina uma trajetória permitindo calcular o ângulo dessa trajetória e o momento da partícula. Um conjunto de dados dos sensores de duas camadas consecutivas pode definir se a partícula aceitável, isto depende do momento angular da mesma obtido pelo pelo caminho feito nas camadas. Esse pré-trigger é definido como *stub* eao unir o conjunto de três *stubs* pode-se pré-determinar se a partícula é objeto do estudo realizado (RAMALHO,2017).

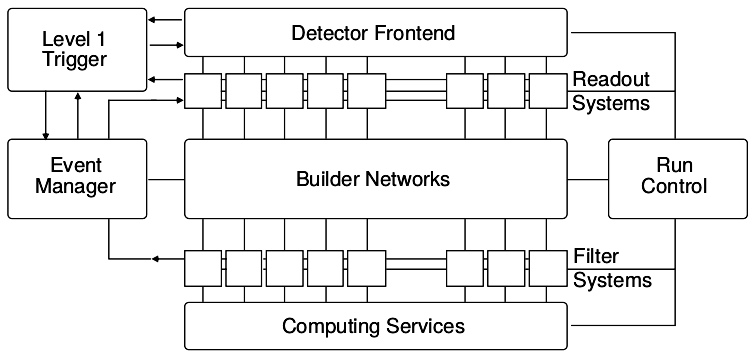
Figura 2.9 – Em cima um esquema do *stub* um esquema da geração do caminho da partícula.



Fonte: (RAMALHO, 2017).

O sistema trigger do detector CMS é dividido em 2 fases: O *level 1 trigger* (L1T) e *High Level Trigger* (HLT). O L1T recebe os dados das seis camadas e realiza o processamento dos dados e determina se o dado é objeto de estudo. Este sistema é formado por eletrônicos personalizados (FPGA’s, etc) e possui uma latência de 3.2 μs contados desde a coleta dos *stubs,* processamento da informação até obter a resposta se o dado será armazenado e enviá-lo ao buffer em que o HLT irá em seguida coletar e processar o dado. Na figura 2.10 é mostrado um esquema dessa comunicação entre o sistema L1T, os detectores e os *buffers* de *readout* (CMS,2017)*.*

Figura 2.10 – Esquema da comunicação interna do sistema de *Trigger.*

**

Fonte: (CMS,2017).

O sistema HLT é um sistema de software que realiza um processamento mais complexo dos dados enviados pelo L1T. A taxa de dados recebidos pelo HLT é de 100 kHz e por algorítimos complexos tendo acesso à toda informação usada pelo sistema L1T. O HLT também usa informações que não podem ser acessadas pelo L1T por conta da escala de tempo deste sistema, dessa forma o HLT tem a possibilidade de acessar as informações do *Tracker* e dos calorímetros.

Observa-se que o sistema L1T deve estar disponível para nova entrada de dados a cada 25 ns e é capaz de processar dados de fótons, elétrons, *muons* usando informações do calorímetro eletromagnético e do sistema de *muons*. Dessa forma, o sistema L1T pode ser classificado como um filtro grosso, ou seja, identifica a maioria da partículas que possuem traços compatíveis e as manda para o HLT. Este pode ser classificado como um filtro fino, ou seja, analisa perfeitamente se os dados enviados pelo L1T podem serem realmente armazenados. Essa separação de filtro grosso e fino deve ser feita por conta que o sistema L1T deve ser o mais rápido possível, não possuindo tempo suficiente para filtrar profundamente as informações (CMS,2017).

**CAPÍTULO 3**

**CODIFICAÇÃO 8B/10B**

Desde a segunda metade do século XX, a área de telecomunicações sofreu um grande avanço com o surgimento de sistemas cada vez mais modernos pelo fato da velocidade e a quantidade de dados aumentarem. Para as transmissões digitais usadas nos detectores do LHC, há a necessidade de garantir uma alta confiabilidade no canal de transmissão. Essa preocupação, deve-se pelo fato de os *hardware*s estarem imersos as radiações emitidas pelas colisões causando ruídos no canal e os problemas de transmissão de dados em altas velocidades.

A solução para a confiabilidade do canal de transmissão e a dessincronização entre o transmissor e o receptor é a implantação de uma codificação no canal de linha. Com a implantação da codificação no canal é possível que outros hardwares interliguem-se no canal de transmissão, garantindo a confiabilidade e resolvendo os problemas de transmissões em altas velocidades. A codificação 8b10b é interessante por solucionar estes problemas, sendo utilizada em vários protocolos de alta velocidade como: 1000Base-X gigabit Ethernet interface ( para comunicações em Fibra Ótica), PCI Express, SerialATA, DVI, HDMI, USB3.

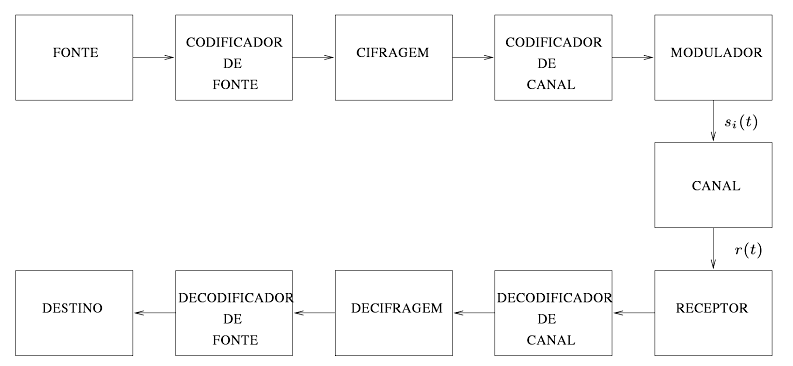
Para sincronizar o emissor e o receptor usa-se o circuito *phase-locked loop* (PLL) que realiza a sincronização pelos níveis lógicos de entrada. Caso em uma transmissão possua muitos níveis lógicos altos ou níveis lógicos baixos seguidos o circuito não é capaz de sincronizar os dois dispositivos, por conta do modo como é projetado. A codificação 8b10b realiza o balanço DC nos dados inseridos no canal de transmissão, equilibrando, na transmissão de bits seguidos, os níveis lógicos altos e baixos. Por meio da sua descrição, também é possível detectar possíveis erros nos dados transmitidos.

**3.1 TRANSMISSÕES DIGITAIS EM ALTA VELOCIDADE**

Sistemas de transmissões digitais possuem um número finito de transições discretas e formas de onda constantes durante a transmissão. O objetivo de uma transmissão digital é transferir uma informação na forma de pulsos de um emissor para um receptor por meio de um canal de transmissão. Todo canal de transmissão não é perfeito e apresenta ruídos somados aos ruídos gerados pelo emissor, portanto todo receptor trabalha deve ser projetado para trabalhar com dados ruidosos (ALMEIDA,2017).

Durante as últimas décadas, com o avanço dos sistemas de telecomunicações, vários sistemas foram propostos para transmitirem dados de forma eficiente e com alta confiabilidade. Na figura 3.1 descreve-se um modelo de transmissão a qual foi formado este concepção durante décadas. Neste esquema está presente uma fonte de informação responsável por inserir dados, de forma analógica ou digital, no canal de transmissão. Por questões que serão explicadas posteriormente, costuma-se em alguns casos converter os dados analógicos para dados digitais, ou discretos, para realizar a transmissão (PIMENTEL,2007).

Figura 3.1 – Esquema de uma transmissão de dados digitais.



Fonte: (PIMENTEL,2007).

O codificador de fonte transforma o sinal da fonte em um sinal codificado, sempre com o menor número de bits possível de forma a obter o maior número de transições possíveis no dado. A cifragem na transmissão é usado quando deseja-se transformar os dados enviados incompreensível à algum agente externo que por acaso deseja captar o dado transmitido, dessa forma preserva-se o dado possibilitando somente o alvo final processar as informações transmitidas. O codificador de canal introduz uma redundância controlada no dado com o objetivo de reduzir os efeitos de ruído e interferências na transmissão (PIMENTEL,2007).

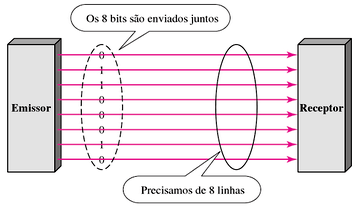
A transmissão digital é melhor do que a transmissão analógica, em alguns casos, por conta da facilidade de trabalhar com sinais digitais e das diversas técnicas existentes para tratá-los. Em qualquer tipo de transmissão o ruído sempre estará presente, portanto em transmissões de forma analógica o dado transmitido estará misturado com o ruído sendo mais difícil trabalhar com os dados. Em transmissões digitais os ruídos analógicos são misturados com o dado digital, dessa forma é mais fácil identificar o ruído no meio da transmissão. Adicionalmente, as transmissões digitais possuem repetição regenerativa a qual um dado ao ser transmitido para o receptor é previamente conhecido possibilitando identificar um erro mais facilmente.

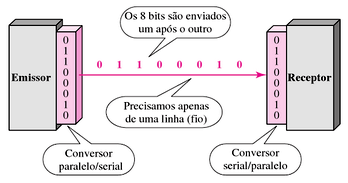
A comunicação digital possui técnicas de processamento digitais de sinais as quais possibilita a filtragem do sinal, eliminação de ruídos e interferências e modulação. Também possuem outros processamentos por software e um baixo consumo de potência quando comparado com sistemas de transmissão analógica. Em transmissões digitais também há desvantagens, como a necessidade de algorítimos complexos e protocolos, uso de circuitos de sincronização e necessidade de equalização por conta da interferência intersimbólica para taxas elevadas.

Porém as vantagens no uso de sistemas de transmissão digital são extremamente importantes, quando entra-se no campo das transmissões em alta velocidade e necessita-se de uma alta confiabilidade na recepção dos dados como no caso dos sistemas implantados no LHC (RUSCHEL, 1996).

A transmissão paralela é mais rápida do que a serial em sistemas digitais. Porém, a transmissão paralela exige custos maiores por conta dos cabos de transmissões que devem ser instalados. Portanto, no caso dos sistemas implementados no LHC prefere-se a transmissão serial por ser o menor custo depositado no sistema. Entretanto, em uma transmissão digital serial bem projetada e com os recursos de hardware certos, pode-se aumentar o *clock* do sistema compensando a diferença de velocidade para a transmissão paralela. Na figura 3.2 é descrito esquemas de transmissões digitais paralelas e seriais.

Figura 3.2 – Esquema de transmissões digitais paralelas e digitais.

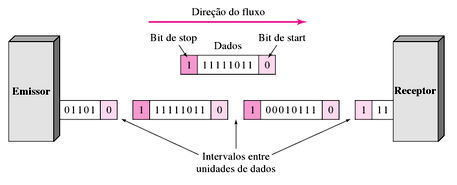




Fonte: (FOROUZAN, 2010).

Transmissões digitais seriais podem ser realizadas de duas formas: assíncrona ou síncrona. A forma assíncrona não importa-se com o tempo entre as mensagens, sendo enviado um grupo de n bits e o receptor tratando de forma independente. Como o receptor não é capaz de prever quando irá receber o dado, torna-se necessário o uso de *bit start* e *bit stop,* dessa forma é possível o receptor sincronizar com o fluxo de dados por ser sinalizado quando o dado começa e onde termina. Este tipo de sistema é classificado como assíncrono pois o emissor e o receptor não precisam estar sincronizados para receber ou enviar os dados, portanto o emissor envia os dados em um instante e o receptor é capaz de receber o dado em outro instante. Na figura 3.3 é descrito um esquema de transmissões digitais seriais assíncronas (FOROUZAN, 2010).

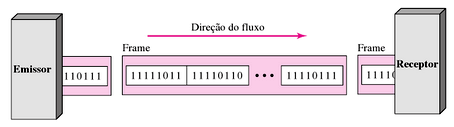
Figura 3.3 – Esquema de transmissão digital assíncrona.



Fonte: (FOROUZAN, 2010).

Em transmissões digitais seriais síncronas são enviados *bits* ininterruptos de acordo com um *clock,* dessa forma emissor e receptor estão sincronizados pelo *clock*, ou seja, o tempo torna-se importante. Quando necessita-se introduzir estados de espera, introduz-se bits 0’s podendo existir intervalos irregulares na transmissão porém sempre o emissor e o receptor estarão sincronizados. Dessa forma, não existe a necessidade de bits extras para indicar início ou fim dos *frames* enviados, tornando a transmissão síncrona mais rápida do que a assíncrona. Na figura 3.4 é descrito um esquema de transmissões digitais seriais síncronas (FOROUZAN, 2010).

Figura 3.4 – Esquema de transmissão digital síncrona.

Fonte: (FOROUZAN, 2010).

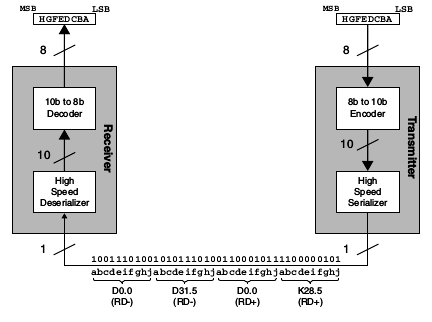
Dessa forma como a transmissão síncrona é mais rápida, nos sistemas do LHC usa-se essa forma por conta da alta velocidade das colisões e a necessidade do hardware estar disponível para um novo evento o mais rápido possível. Portanto com a transmissão digital serial síncrona é possível captar e processar os dados no detector.

**3.2 CODIFICAÇÃO 8B/10B**

A codificação 8b10b foi criada por Albert X. Widmer e Peter A. Franaszek, ambos da IBM em 1983. A codificação foi descrita para se adequar à locais com transmissões em alta velocidades, promovendo sistemas de baixo custo e com transmissões confiáveis. Pela descrição, a codificação promove um balanceamento DC no sinal, ou seja, o dado a ser transmitido não possui níveis lógicos altos ou baixos por muito tempo. Esse balanço torna-se importante para a recuperação do relógio e consequentemente sincronizar o emissor e o receptor.

Os dados de 8 *bits* são codificados em 10 bits de forma que haja o maior número de transições possíveis. Dessa forma, as 256 possibilidades de dados de entrada tornam-se em 1024 possibilidades de dados codificados. Neste mapeamento, nota-se que alguns dados de 10 bits não possuem equivalentes em dados de 8 bits, portanto alguns dados de 10 bits, que possuem o mesmo número de bits 1’s e 0’s, são usados como dados de controle da transmissão. No sistema da codificação, os dados a serem codificados entram paralelamente e posteriormente são serializados para serem transmitidos. Posteriormente, realiza-se a deserialização, entrando com os 10 bits em paralelo no decoder, conforme descrito na figura 3.5 (LATTICE SEMICONDUCTOR, 2015).

Figura 3.5 – Esquema da Transmissão usando a codificação 8b10b.

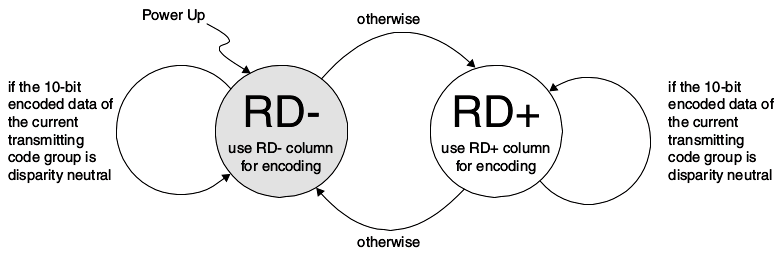


Fonte: (LATTICE SEMICONDUCTOR, 2015)

A codificação é descrita de forma que os dados de 10 bits da codificação só possuam disparidade de +2, -2 ou 0 (nula). A disparidade é o cálculo do número de 1’s menos o número de 0’s do dado de 10 bits codificado. Dessa forma, caso um dado codificado chegar no receptor com disparidade diferente da descrita pode-se detectar que houve um erro durante a transmissão e deve-se descartar o dado. Essa característica da transmissão possibilita a detecção de erro pelo receptor e uma possível recuperação do dado usando circuitos adicionais.

O sistema da codificação deve codificar os dados de forma que a linha de transmissão nunca possua disparidade diferente de +2, -2, 0 entre os dados de 10 bits enviados. Portanto, para a codificação dos dados usa-se o *running diparity* (RD) ao qual segue uma lógica na codificação dos dados em que pode ser visualizado na figura 3.6. O RD sempre irá ser inicializado como -1 no sistema e irá se alternar entre +1 e -1 ou irá permanecer o mesmo de acordo com as regras descritas na figura 3.6. Se o RD é -1 e a disparidade do novo dado de 10 bits for +2, o RD troca para +1. Caso o RD for +1 e a disparidade do novo dado de 10 bits for -2, logo o RD troca para -1. Qualquer outra situação não pertencente a esses casos o rd permanece o mesmo (LATTICE SEMICONDUCTOR, 2015).

Figura 3.6 – Máquina de Estados do RD da codificação 8b10b.

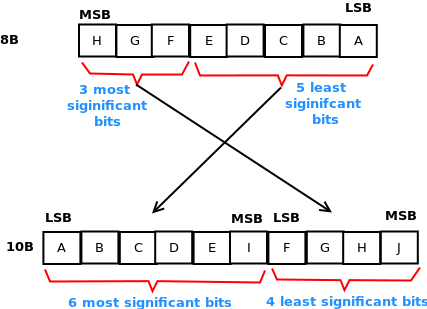
Fonte: (LATTICE SEMICONDUCTOR, 2015).

Cabe ressaltar que de acordo com a descrição da codificação, se o RD é -1 o dado de 10 *bits* codificado só poderá possuir disparidade 0 ou +2. Representa-se um erro no lado do receptor caso receba um dado de 10 bits com disparidade -2 no momento que o RD do sistema é -1, ou seja, em nenhum momento que o RD for -1 irá ser codificado dados com disparidade -2. Da mesma forma, representa um erro no lado do receptor caso receba um dado com disparidade +2 se o RD for +1. Pela descrição não haverá dados codificados para 10 bits com disparidade +2 quando o RD for +1, portanto caso o receptor receba um dado nas condições descritas irá caracterizar um erro na transmissão.

Quando há um erro na transmissão o RD do receptor e do transmissor entra-se em descompasso. Porém, de acordo com a codificação o RD é recuperado não levando mais do que 5 ciclos de *clock*, no pior dos casos, para entrar em compasso novamente com o transmissor.

A codificação dos dados é feita separando os dados de 8 bits em duas partes. Uma parte pega-se os 5 bits menos significativos e convertendo-os para 6 bits e a outra parte pega-se os 3 bits mais significativos e convertendo-os para 4 bits. Posteriormente junta-se os 6 bits com os 4 bits, formando um dado de 10 bits. Esta junção é feita coletando a parte de 6 bits e a parte de 4 bits, inserindo-os na parte mais significativa e na parte menos significativa do dado de 10 bits respectivamente. Um esquema dessa separação da codificação pode ser visualizada na figura 3.7.

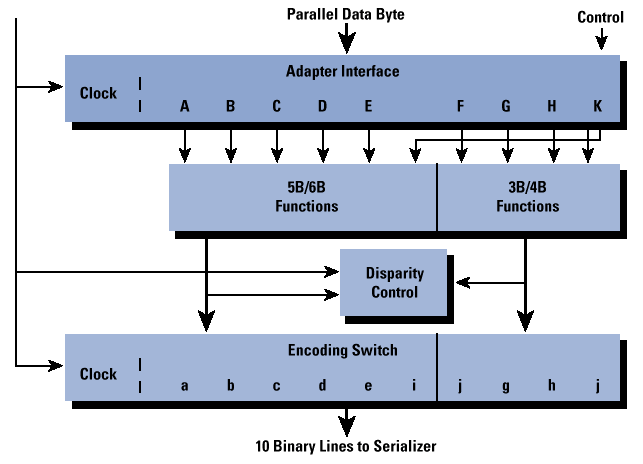
Figura 3.7 – Esquema da separação para codificação 8b/10b.



Fonte: Elaborado pelo Autor.

Na figura 3.8 é apresentado um esquema de um sistema que realiza a codificação 8b/10b junto com o sistema de controle de disparidade.

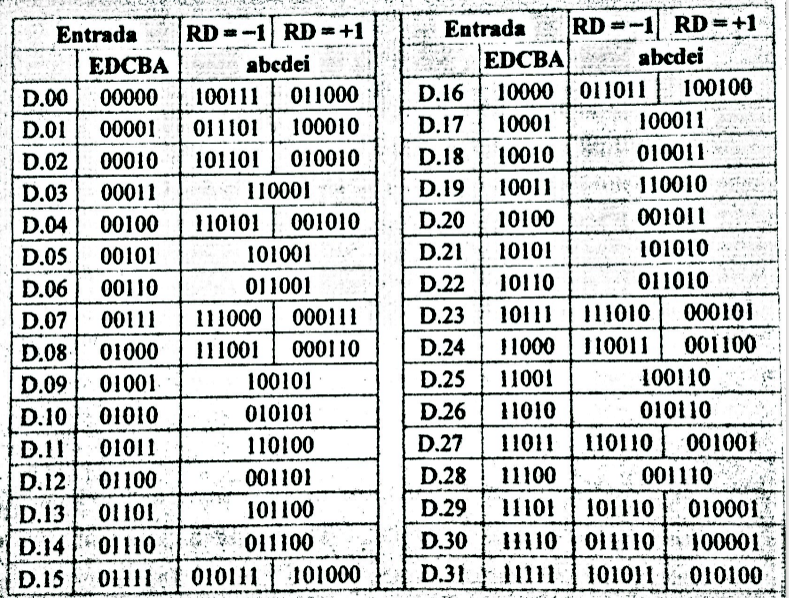
Figura 3.8 – Esquema de um Sistema da codificação 8b10b.



Fonte: (STALLINGS, 2017).

Os sistemas para a codificação podem serem implementados pelas funções descritas por Widmer e Franaszek (1983), ou usando memórias ROM’s com as tabelas de codificações retiradas das funções descritas por Widmer e Franaszek (1983). Neste trabalho usou-se o método pelas tabelas, portando na figura 3.9 é mostrado a tabela da codificação dos 5 bits para os 6 bits de acordo com o RD.

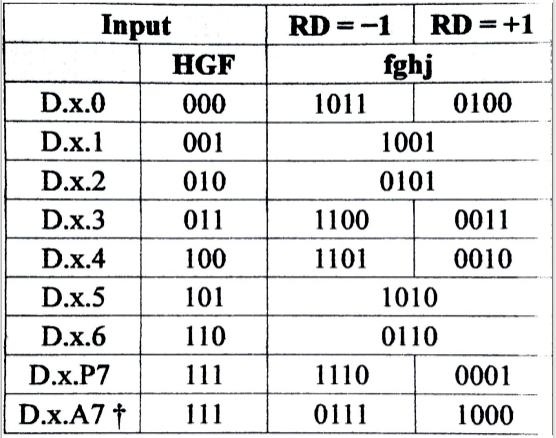
Figura 3.9 – Tabela de codificação dos 5 bits para 6 bits.



Fonte: (SATA COMMUNICATION, 2017).

Na figura 3.10 é mostrado a tabela da codificação de 3 bits para 4 bits de acordo com a codificação. Nesta tabela para alguns valores de 5b a tabela é invertida, ou seja, os valores que estão no RD +1 vão para o RD -1 e os valores do RD -1 vão para o RD +1. Para os valores de 5 bits (D.3, D.5, D.6, D.9, D.10, D.11, D.12, D.13, D.14, D.17, D.18, D.19, D.20, D.21, D.22, D.25, D.26 e D.28) da tabela da figura 3.10 deve-se inverter a tabela da figura 3.10 para que a codificação permaneça corretamente descrita.

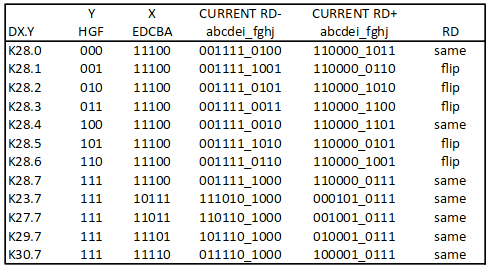
Figura 3.10 – Tabela de codificação dos 3 bits para 4 bits.



Fonte: (SATA COMMUNICATION, 2017).

Na tabela da figura 3.11 são apresentados alguns códigos de controle os quais usa-se para estabilizarem a sincronização dos bits, marcar o começo e o fim dos pacotes e algumas vezes o controle das funções dos sinais como *reset, abort, shut off e Idle.* Na tabela são descritos 12 codificações de acordo com a codificação.

Figura 3.11 – Tabela de Caracteres Especiais



Fonte:(CORNELL, 2017).

**3.3 ALGORITIMO DESENVOLVIDO PARA A SIMULAÇÃO E DESCRIÇÃO DO SISTEMA DA CODIFICAÇÃO 8B10B**

De acordo com a descrição da codificação 8b/10b, elaborou-se fluxogramas para o encoder e um decoder a qual implemente a codificação, fornecendo a possibilidade de implementar no software. Portanto, com esse fluxograma é possível descrever um programa no MATLAB(Simulink) para simular e obter as características da codificação 8b10b. Também é possível descrever um hardware no VHDL de forma a implementar a codificação fornecendo a possibilidade de implementação futura em um hardware.

O fluxograma do *encoder* está descrito na figura 3.12 e o do *decoder* está descrito na figura 3.13. Com a análise do algorítimo do *encoder*, observa-se que no início do sistema são definidos as tabelas de codificação e verificação de flip. As tabelas de codificação são separadas em duas partes, a primeira refere-se as tabelas de codificação 5B/6B e a segunda as tabelas de codificação 3B/4B.

As tabelas 5B/6B contém os dados de entrada 5B provindos dos 5 bits menos significativos do dado de entrada de 8 bits e os dados codificados para 6 bits para o RD negativo e positivo. As tabelas 3B/4B seguem o mesmo padrão possuindo os dados dos 3 bits mais significativos do dado de entrada e os dados da codificação para 4 bits referentes ao RD positivo e negativo. A tabela de verificação de flip refere-se a necessidade de inverter a tabela dos dados codificados de 3 bits para 4 bits, acontecendo nos casos da entrada de 5 bits listados na secção 3.2 deste capítulo.

Posteriormente, deve-se separar o dado de entrada como descrito pela codificação para o sistema codificar os 5 bits menos significativos em 6 bits menos significativos e os 3 bits mais significativo para 4 bits menos significativos do dado de 10 bits. Portanto, usa-se 2 variáveis que coletam esses valores para serem comparados com as tabelas de dados de entrada de 5 bits e 3 bits, respectivamente.

Figura 3.12 – Fluxograma de um sistema Encoder que implemente a codificação 8b10b.

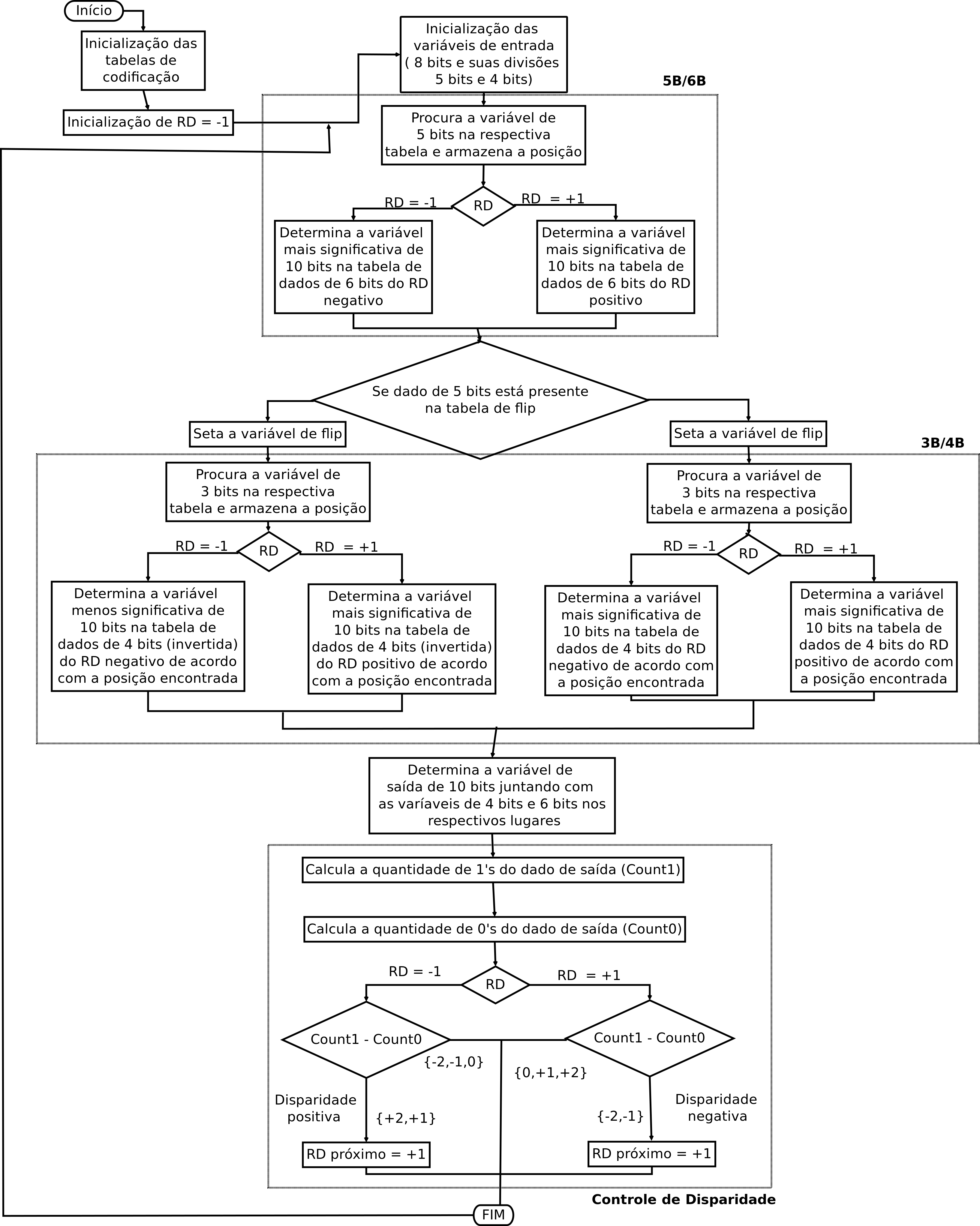
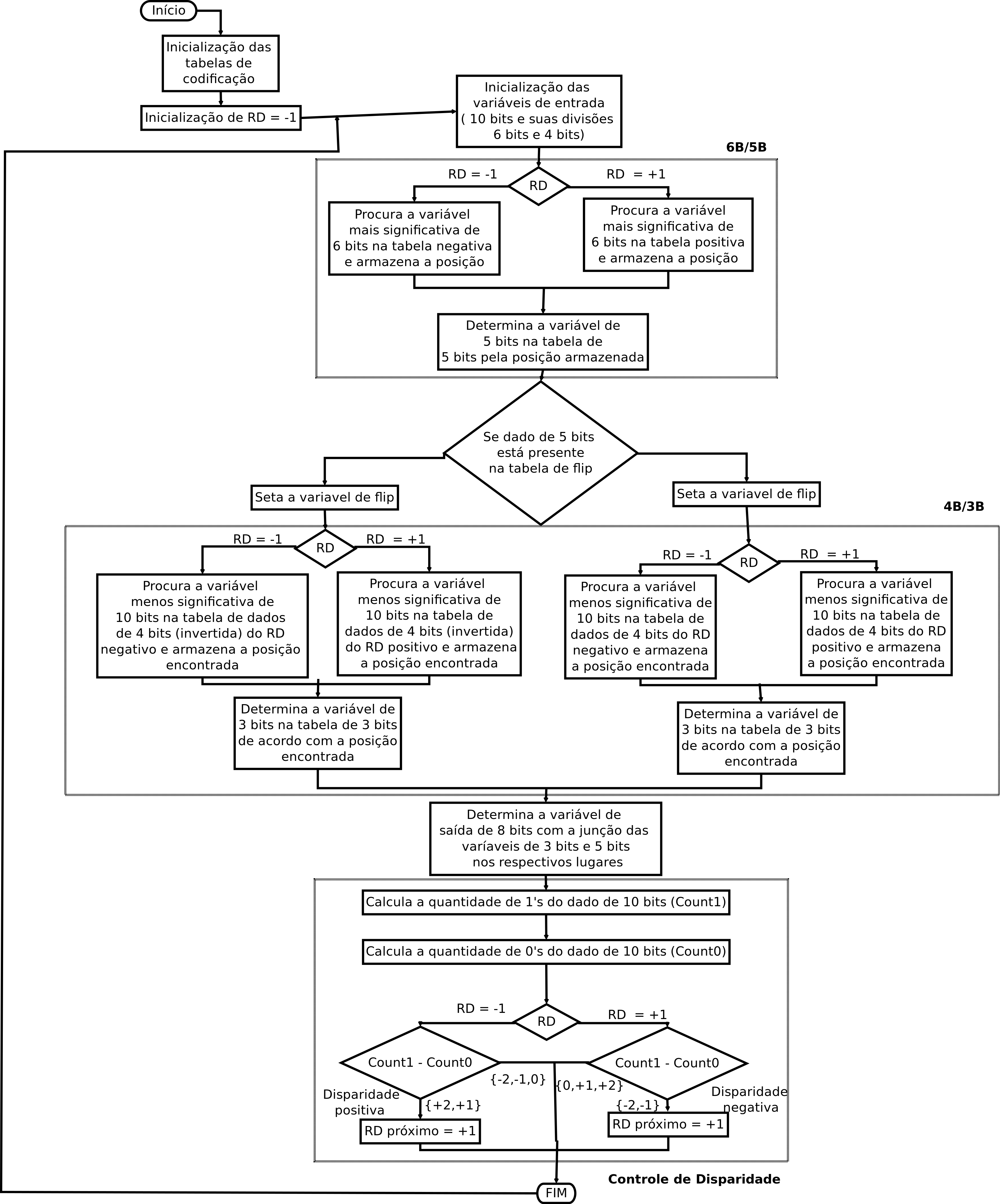
Fonte: Elaborado pelo Autor.

Figura 3.13 – Fluxograma de um sistema *Decoder* que implemente a codificação 8b10b.

Fonte: Elaborado pelo Autor.

Essas variáveis ao serem comparadas com as tabelas, pode-se armazenar as posições pelos valores correspondentes na tabela e usar as tabelas de codificação para definir os valores das variáveis de 6 bits e 4 bits codificadas por meio da posição encontrada e do RD do sistema. Após encontrar as variáveis preliminares de 10 bits agrupa-se em uma variável de saída, inserindo a variável de 6 bits nos bits mais significativos e a variável de 4 bits nos bits menos significativos do dado de saída de 10 bits.

Por meio do dado de saída inicia-se a definição do novo RD para o sistema. Extrai-se o número de 1’s no sistema e o número de 0’s, subtraindo o número de 1’s do número de 0’s e obtendo o novo RD como descrito na secção 3.2 deste capítulo. Para realizar a contagem usa-se variáveis para armazenar o valor dos respectivos níveis lógicos analisados.

O algorítimo que descreve o *decoder* exemplificado na figura 3.13 executa da mesma forma que no algorítimo da figura 3.12, no início do sistema, o carregamento das tabelas para codificação dos dados, a comparação dos dados de entrada e a verificação do *flip.*

Diferentemente do empregado no algorítimo do sistema do *encoder* da figura 3.12, após carregar as tabelas separa-se o dado de entrada de 10 bits em duas partes. A primeira parte são os 6 bits mais significativos e a segunda os 4 bits menos significativos, posteriormente esses valores são comparados com os dados presentes nas tabelas de codificação o qual são escolhidas dependendo do RD do sistema. Ao encontrar o dado correspondente na tabela, a posição é salva para posteriormente definir a variável de saída de 8 bits ao qual é definida em duas partes. Primeiramente define-se os 5 bits menos significativos do sinal de saída de 8 bits. Usa-se essa variável encontrada, para verificar se a tabela de codificação dos 4 bits menos significativos do dado de entrada é invertida para análise.

Posteriormente define-se os 3 bits mais significativos do dado de saída de 8 bits, encontrando o valor correspondente na tabela de codificação de 4 bits, de acordo com o RD, já levando em conta se as tabelas são invertidas. Dessa forma, os dados das variáveis de 5 bits e 3 bits são unidos na posição de menos significativos e mais significativos no dado de saída de 8 bits respectivamente. O novo RD para o fluxograma do *decoder* é calculado como no fluxograma da figura 3.12 porém nesse caso o RD é calculado com o dado de entrada de 10 bits.

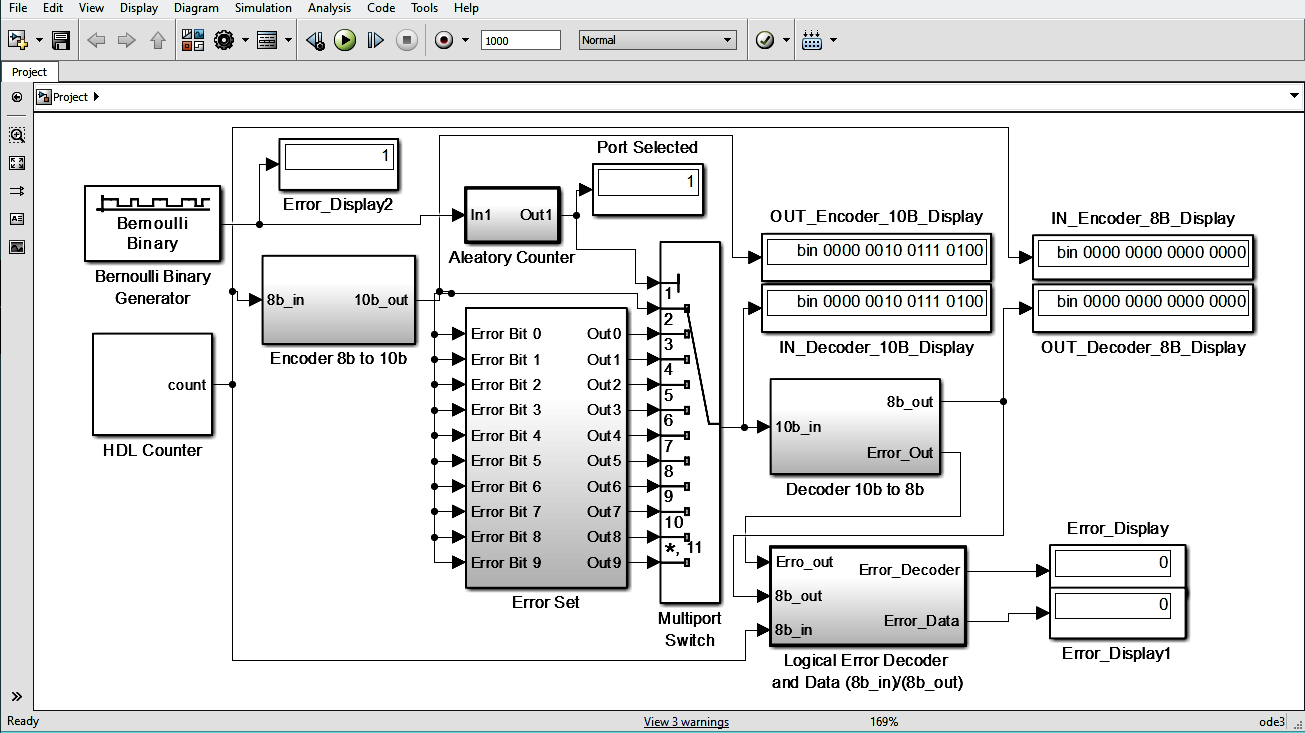
**3.4 SISTEMA IMPLEMENTADO NO MATLABTM (SIMULINK)**

O sistema foi implementado no MatlabTM usando os recursos *Embedded MatlabTM Function Block*  (EMFB) do *Simunlink*. Nesta ferramenta pode-se descrever o *encoder* e o *decoder* de acordo com os fluxogramas das figuras 3.12 e 3.13, respectivamente, com os EMFBs do MatlabTM. Pela grande variedade de recursos que esta ferramenta possui é possível criar mecanismos para obter as características da codificação. A obtenção destas características provém de inserção de erros na transmissão de dados do *encoder* para o *decoder*, analisando o número de erros obtido pelo número de dados transmitidos. Estes erros inseridos devem ser aleatórios para a obtenção de uma característica que se aproxime da realidade.

Na figura 3.14 observa-se o sistema desenvolvido usando a ferramenta EMFB do *Simulink.* Neste sistema observa-se um *HDL counter* que realiza a inserção dos dados em 8 bits, ou seja, insere dados binários no sistema que vão desde dos equivalentes inteiros “0” até “255”. Os subsistemas *Bernoulli Binary Generator* (BBG) e *Aleatory Counter* (ALC) são responsáveis por gerarem o erro na transmissão depois que os dados são codificados para 10 bits no subsistema *Encoder 8b to 10b.*

No subsistema *Error Set* é aplicado o erro no dado transmitido e no subsistema *Multiport Switch* (MPS)é selecionado se há ou não erro na transmissão. No subsistema Decoder *10b to 8b* os dados são decodificados para 8 bits novamente. No subsistema *Logical Error Decoder and Data (8b\_in)/(8b\_out)* (LEDD) o número de erros do sistema é obtido de duas formas. Primeiramente analisa-se o sinal de saída do *decoder Error\_out,* caso estiver com valor lógico alto aciona um contador.A segunda forma de obter o número de erros é a comparação entre os dados de entrada do subsistema *HDL counter*  com os dados de saída do *decoder,* caso forem diferentes aciona um contador registrando o erro. Dessa forma, há dois contadores para obter erros no sistema.

Figura 3.14 – Sistema desenvolvido no MatlabTM usando a EMFB do *Simulink.*

**

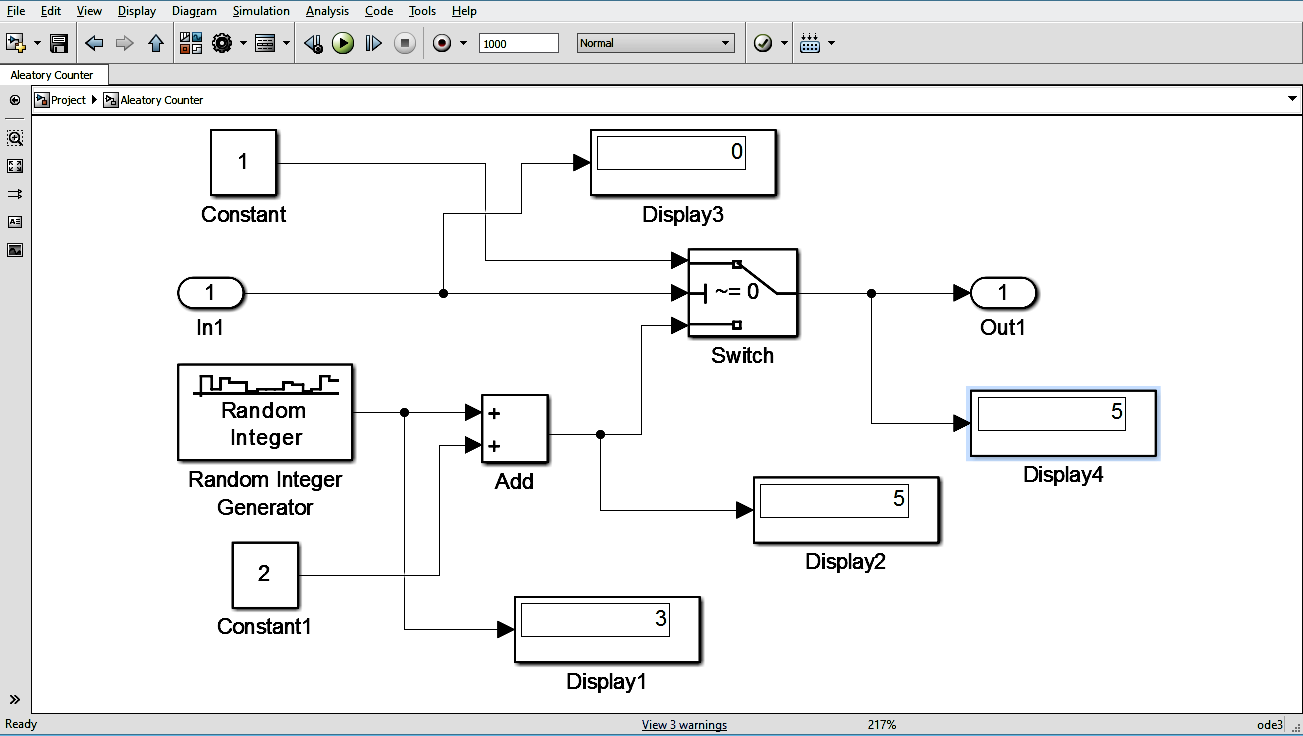
Fonte: Elaborado pelo Autor.

O subsistema BBGgera números binários de 1 bit (0’s ou 1’s) aleatoriamente de acordo com uma porcentagem pré-definida. Pode-se dizer que este subsistema é como se fosse uma moeda viciada, em que tem-se um número total de eventos e define-se a porcentagem do número de vezes que cada lado da moeda vai sair do total de vezes que a moeda for jogada. O erro é gerado quando o subsistema gerar o bit 0, dessa forma dentro do subsistema define-se a probabilidade de ocorrer erro no canal.

Na figura 3.15 observa-se a estrutura dentro do subsistema ALC querecebe o bit do BBGselecionando as portas do MPS dependendo do bit recebido. Dessa forma, pode -se introduzir um erro no canal de transmissão ou transmitir o dado sem erros. Pela figura 3.15 observa-se a entrada *In1* a qual recebe o bit do subsistema BBGselecionando, através do bloco *Switch*, entre a constante de valor inteiro “1” ou o subsistema *Random Integer generator* (RIG) adicionado com uma constante de valor inteiro “2” como saída.

O RIG está configurado para gerar números de “0” até “8”, dessa forma somado com a constante de valor inteiro “2” são gerados números de “2” até “10” aleatoriamente. O subsistema *Switch* seleciona a constante de valor 1 se na entrada *In1* possuir o bit 1, caso possuir o bit 0 é selecionado o RIG adicionado com a constante 2.

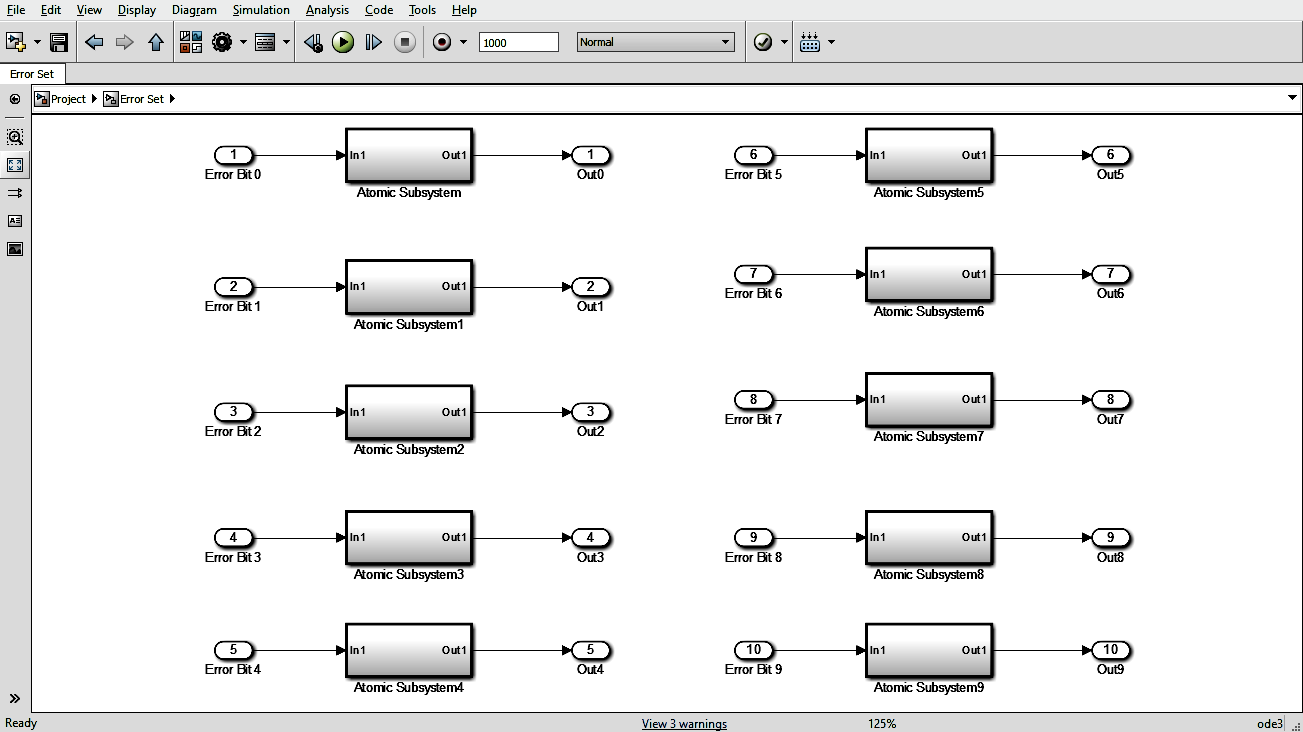
Figura 3.15 – Estrutura Interna do bloco *Aleatory Counter*.



Fonte: Elaborado pelo Autor.

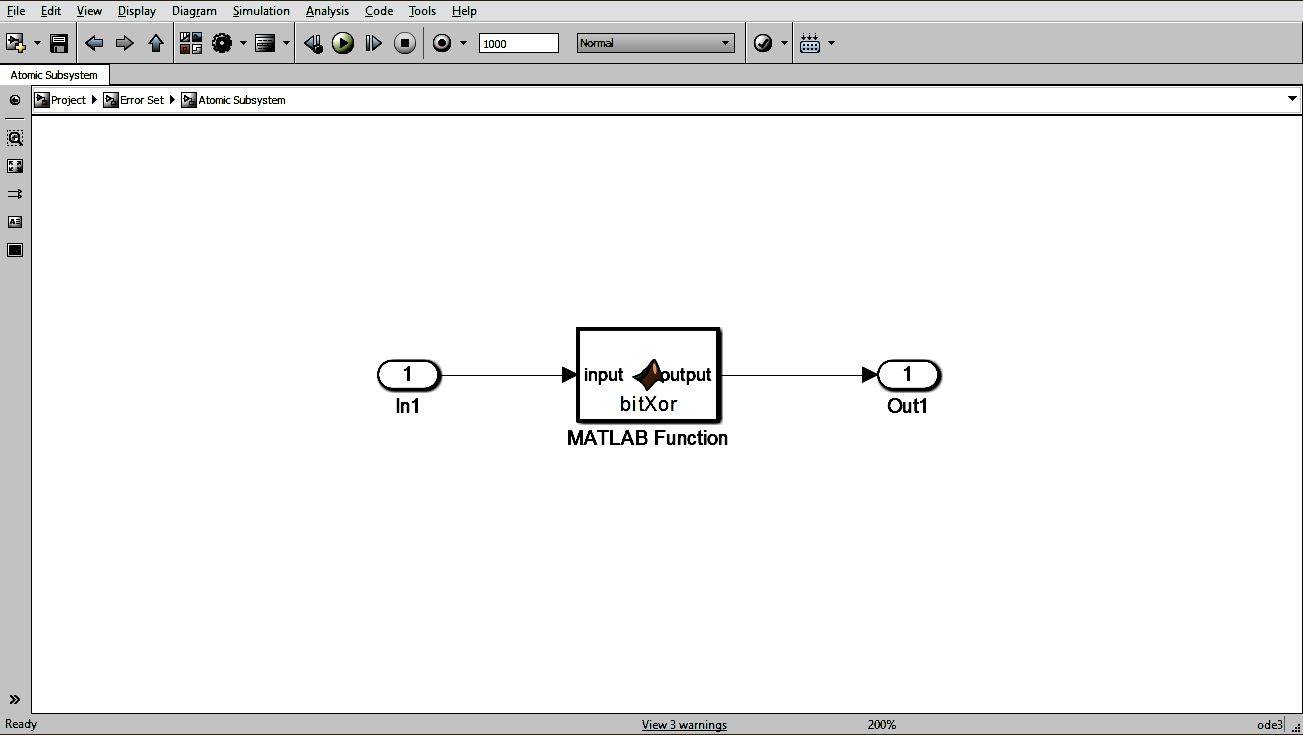
No MPS a porta 1 é a que seleciona as portas 2 até a porta 10, porém se na porta 1 é inserido o valor inteiro “1” logo é selecionado a porta 2 e assim por diante. Pela figura 3.14, no MPS o erro é inserido quando na porta 1 estiverem presentes valores inteiros de “2” até “10”. Dessa forma, pela lógica presente no subsistema ALC pode-se inserir um erro aleatoriamente no canal de transmissão. Na figura 3.16 apresenta-se a estrutura do subsistema *Error Set* que insere erros no dado de saída.

Figura 3.16 – Estrutura interna do subsistema *Error Set*.

Fonte: Elaborado pelo Autor.

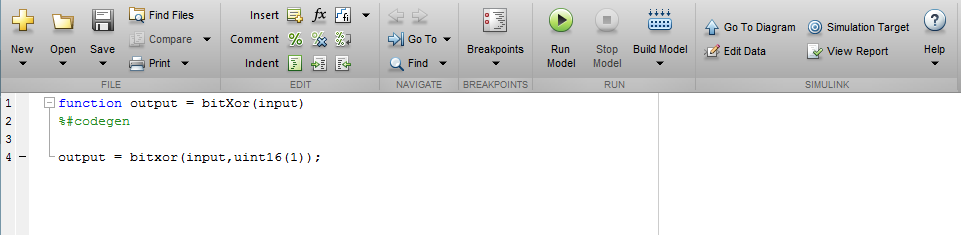
Na figura 3.17 é apresentado a estrutura interna de todos os *Atomic Subsystem* do subsistema *Error Set*. A estrutura é formada por uma entrada e saída ao qual há um bloco que implementa um XOR em um bit específicado do dado de saída de 10 bits do *encoder*.

Figura 3.17 – Estrutura interna dos *Atomics Subsystem* do subsistema Error Set.

Fonte: Elaborado pelo Autor.

Na figura 3.18 é apresentado a programação do subsistema *Matlab Function* da figura 3.17 em que usa-se o comando “bitxor” do matlab para realizar a operação XOR. Neste comando realiza uma operação XOR entre dois dados, no caso há o dado de entrada (*input)* e um valor inteiro “1” transformado em binário pela função *bitxor.* Neste caso introduziu-se um erro no primeiro bit, para introduzir o erro no terceiro bit deve-se realizar uma operação XOR com a entrada *input* e o valor inteiro “4” transformado em binário. Esse mesmo procedimento é seguido para introduzir o erro nos outros bits, sempre inserindo números inteiro que ao serem transformados em binários deixem possuam o bit 1 na mesma casa binária que deseja-se introduzir o erro no dado de entrada.

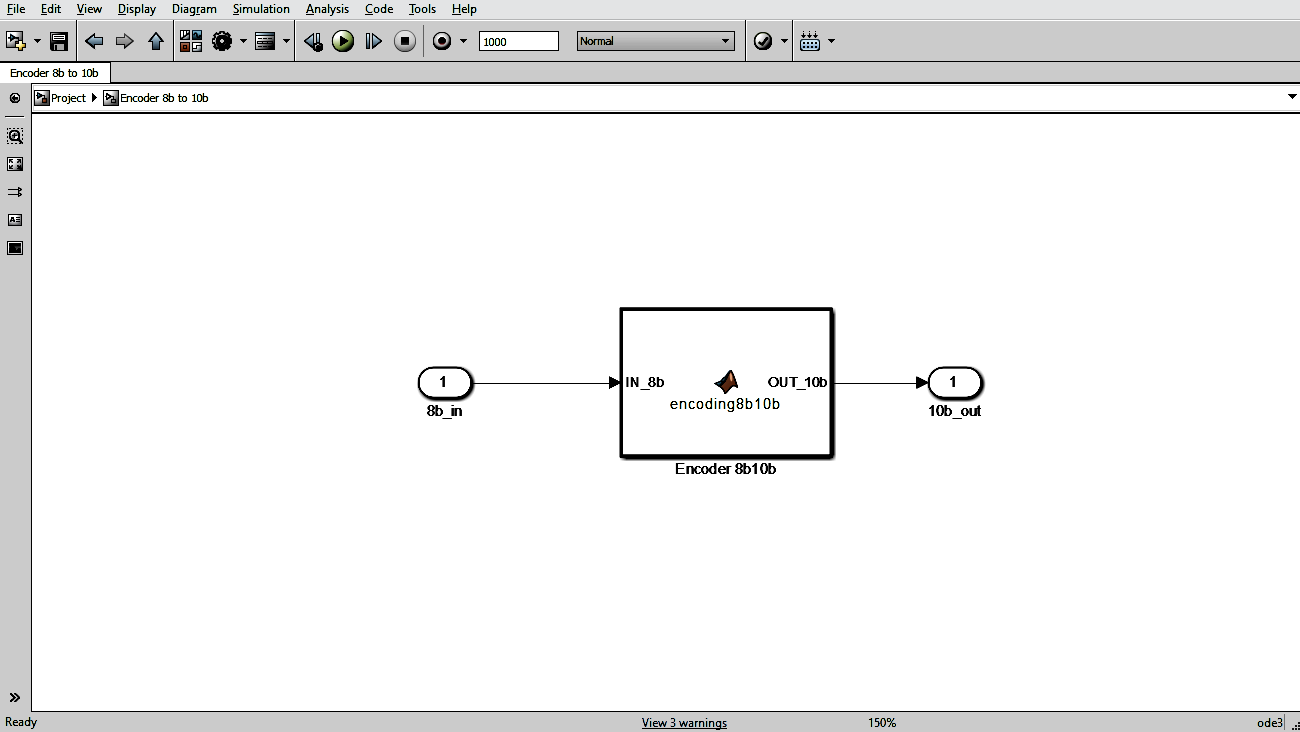
Figura 3.18 - Programação interna do subsistema Matlab Function.



Fonte: Elaborado pelo Autor.

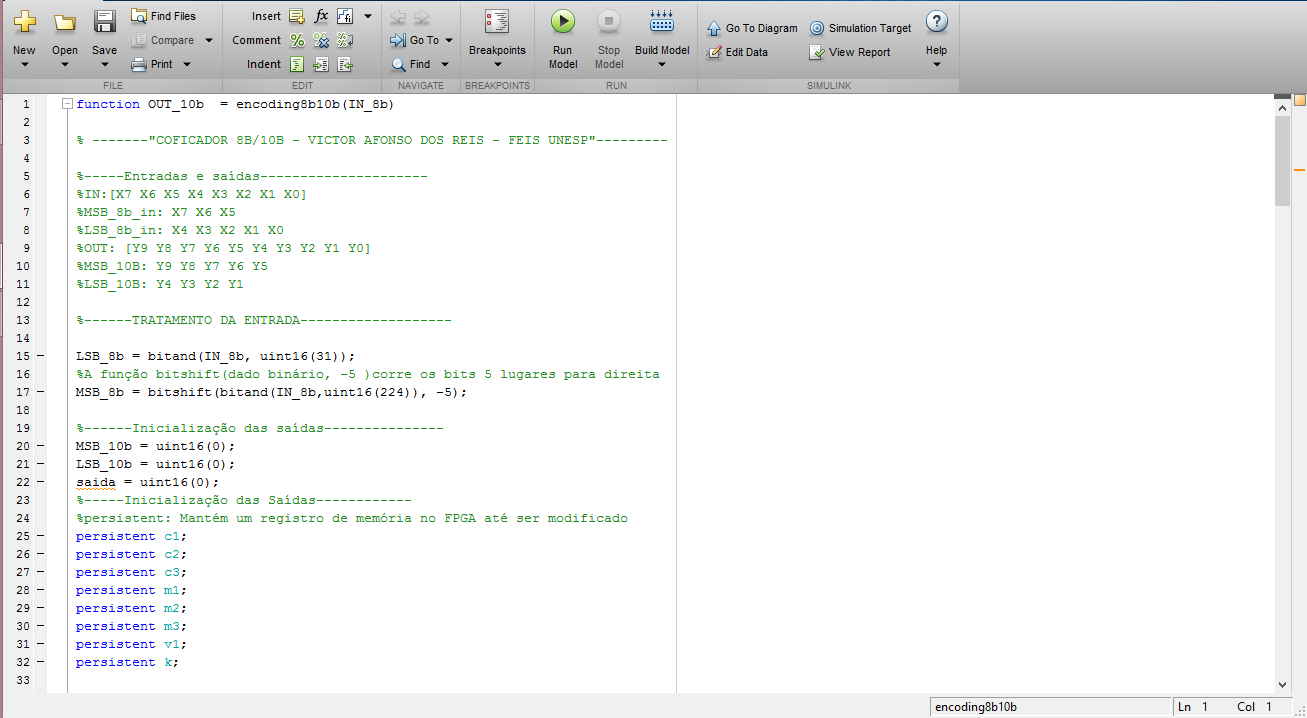
Na figura 3.19 é apresentado a estrutura interna do subsistema *Encoder 8b to 10b.* Nesta estrutura está presente um *Function Block* em que está descrito a codificação do *encoder* 8 bits para 10 bits.

Figura 3.19 – Estrutura interna do *Encoder 8b to 10b.*

Fonte: Elaborado pelo Autor.

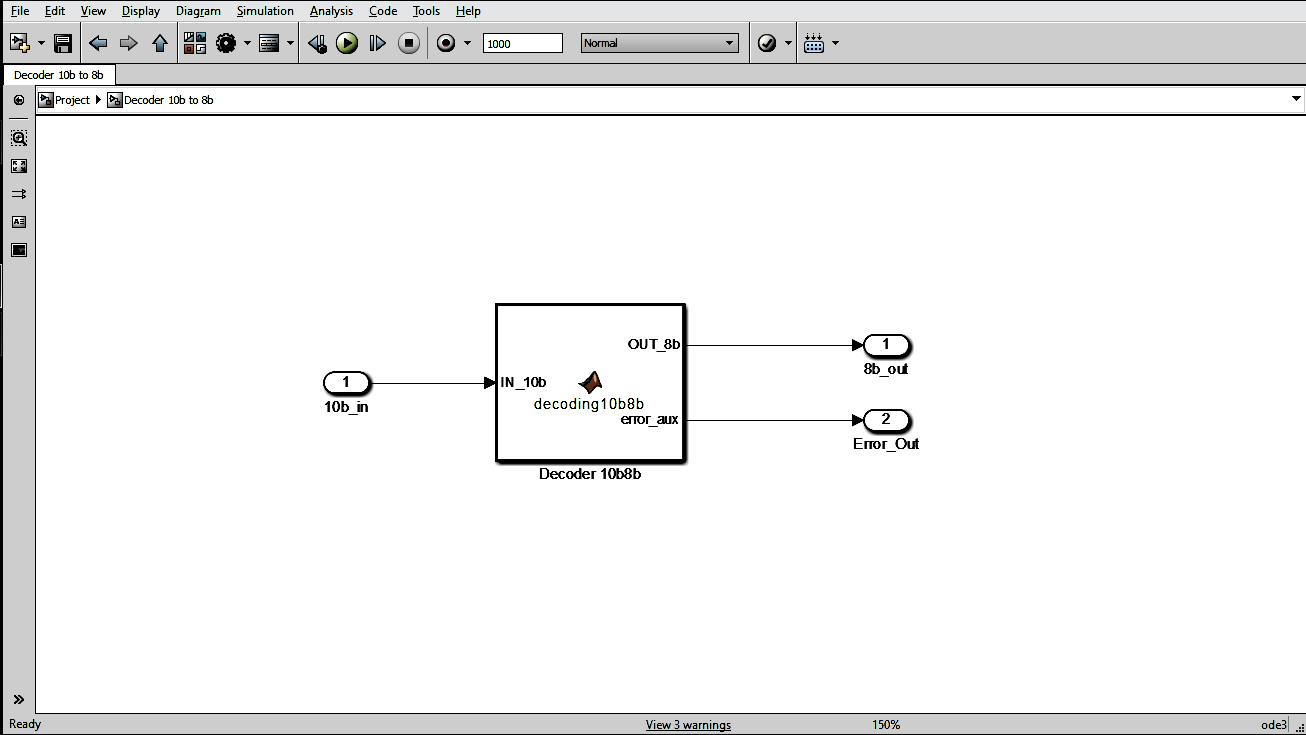
Na figura 3.20 é apresentado uma parte da programação do *Function Block Encoder 8b10b.* A programação completa está descrita no Anexo I.

Figura 3.20 – Programação do *Function Block Encoder 8b10b.*

Fonte: Elaborado pelo Autor.

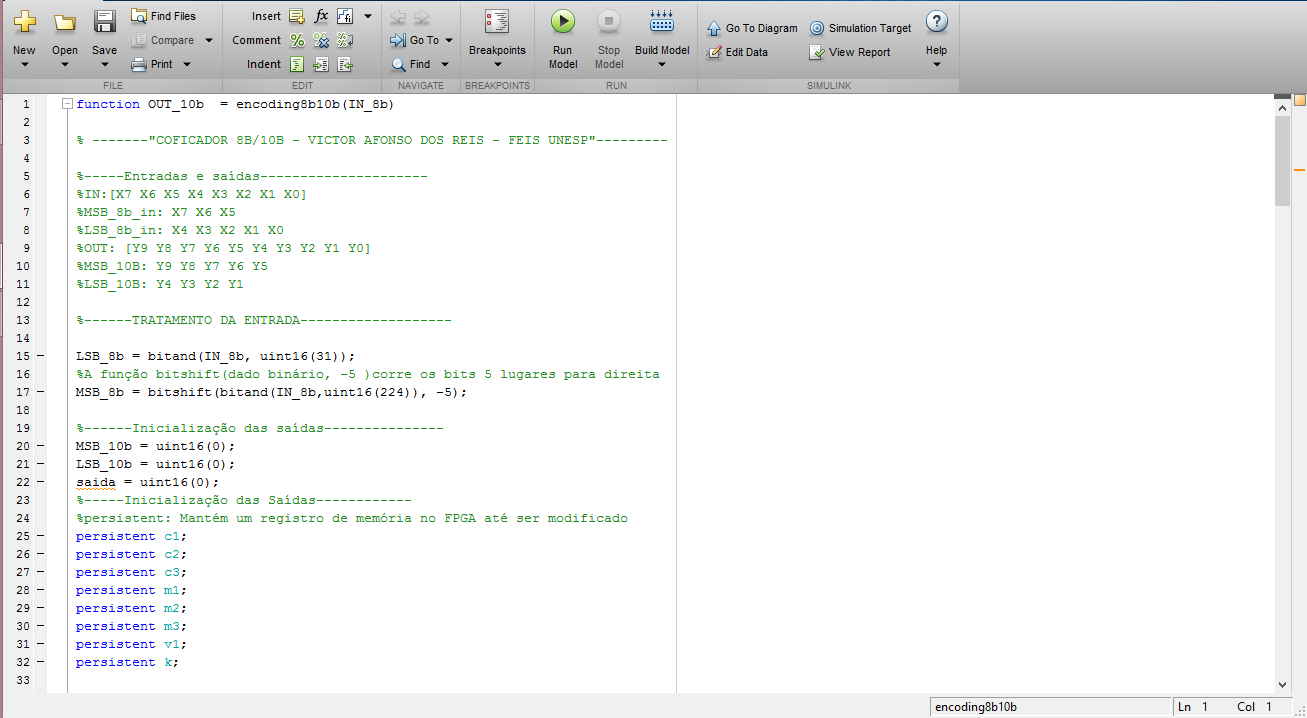
Na figura 3.21 é apresentado a estrutura interna do subsistema *Decoder 10b to 8b.* Nesta estrutura está presente um *Function Block* em que está descrito a codificação do D*ecoder* 10 bits para 8 bits.

Figura 3.21 – Estrutura interna do De*coder 8b to 10b.*

Fonte: Elaborado pelo Autor.

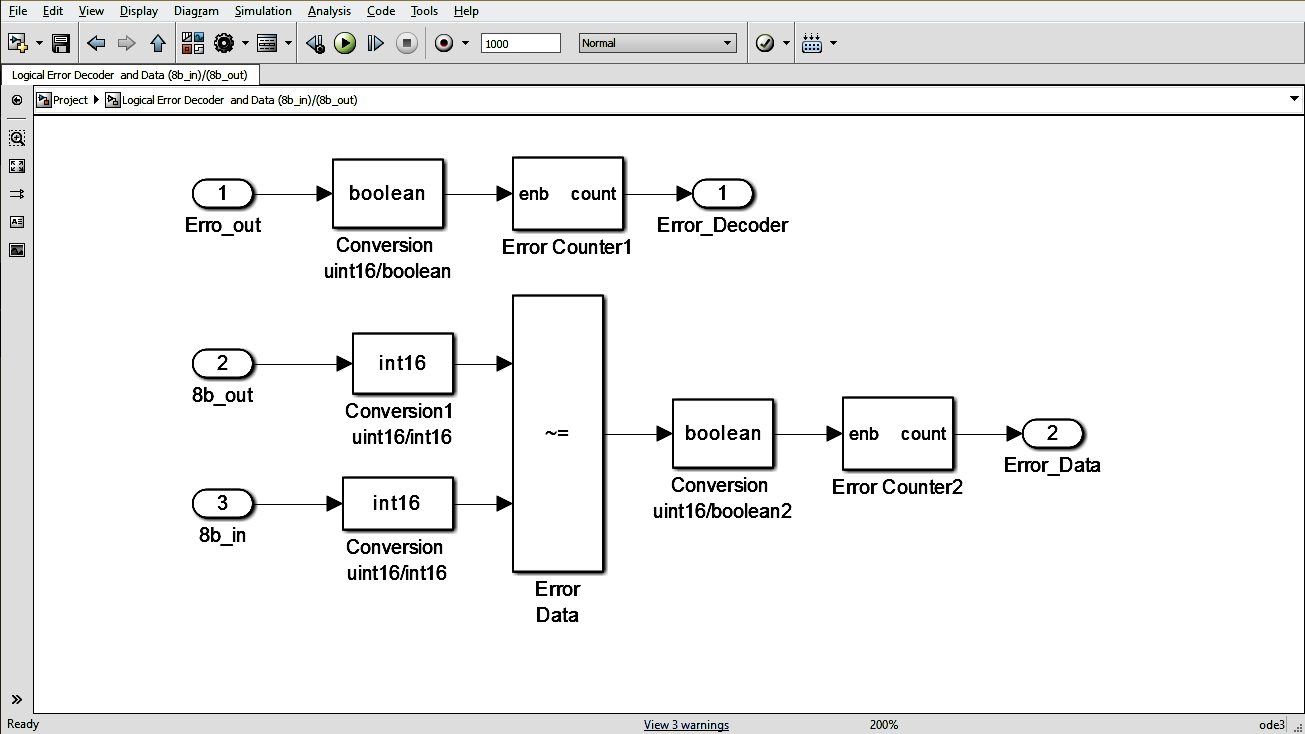
Na figura 3.22 é apresentado uma parte da programação do *Function Block Decoder 10b8b.* A programação completa está descrita no Anexo II.

Figura 3.22 – Programação do *Function Block Decoder 10b8b.*

Fonte: Elaborado pelo Autor.

Na figura 3.23 é apresentado a estrutura interna do subsistema *Logical Error.* A estrutura possui 3 entradas sendo a primeira o sinal de saída *Error\_Out* do *decoder* ao qual indica um sinal de erro do dado recebido, a segunda o dado de saída do *decoder* e a terceira o dado inserido no *encoder.* Estas duas últimas entradas são inseridas em um comparador gerando um sinal lógico que ativa um contador, da mesma forma o sinal de erro do *decoder* ativa um contador e obtém-se o número de erros contido na transmissão.

Figura 3.23 – Estrutura inter do subsistema *Logical Error.*



Fonte: elaborado pelo Autor.

**CAPÍTULO 4**

**SISTEMA IMPLEMENTADO NO VHDL**

A linguagem de descrição de harware (HDL) permite várias aplicações de documentação e descrição de circuitos digitais. Os projetos de circuitos lógicos descritos nessa linguagem podem serem implementados em dispositivos lógicos programáveis complexos, possibilitando posteriormente a aplicação de circuitos integrados com o projeto. Para os sistemas implementados no LHC, torna-se necessário o uso de placas “Fiel Programmable Gate Array” (FPGA) por suportarem a alta velocidade que os dados devem serem processados. Através da linguagem VHDL pode-se descrever e simular circuitos descritos para o sistema L1TT, possuindo a versatilidade para descrever outro sistema a qualquer momento somente reprogramando as placas que estão implementadas o circuito descrito.

**4.1 A LINGUAGEM DE DESCRIÇÃO DE HARDWARE VHDL**

O desenvolvimento da linguagem de descrição de *hardware* VHDL, foi motivado por conta do departamento de defesa americano necessitar de uma ferramenta que disponibilizasse um padrão na documentação para o projeto *Very high Speed Integrated Circuit* (VHSIC). Dessa forma, em 1983 foi definido uma linguagem padrão para descrição de circuitos lógicos e quatro anos mais tarde o *Institute of Electrical and Electronic Engineer* estabeleceu o padrão IEEE 1076-1987. Anos posteriores, surgiram vários outros padrões com novos pacotes com o intuito de introduzir novos recursos e resolver problemas presentes nas versões antigas.

A linguagem de descrição de *hardware* VHDL descreve o que um sistema faz e como ele faz. Esta linguagem suporta diferentes níveis de hierarquias, ou seja, a linguagem é capaz de mesclar uma descrição estrutural e comportamental. As ferramentas de síntese suportam os diferentes estilos desde que o nível de abstração seja moderado, ou seja, se um sistema descrito só possuir o que é observado nas saídas sem ser descrito como é produzido internamente as ferramentas de síntese podem não suportar.

Todos os comandos descritos na linguagem são executados concorrentemente, ou seja, não importa a ordem em que os comandos apareçam eles serão executados ao mesmo tempo. É possível definir em algumas regiões do código para que os comandos sejam executados de forma sequencial. Na simulação a avaliação interna é realizada de forma sequencial, loco necessita-se de alguma ferramenta interna que armazene o resultado de cada comando até que todos os comandos forem avaliados (D’AMORE, 2012).

A maior dificuldade nos métodos tradicionais de projeto é a conversão manual da descrição do projeto em um conjunto de equações Booleanas. Esta dificuldade é eliminada com o uso da linguagem de descrição de hardware VHDL. Pode-se, por exemplo, a partir de uma tabela verdade, ou da descrição de uma máquina de estado, implementar um circuito usando-se HDL. Dentre as várias HDLs, as mais populares são VHDL, Verilog e Abel.

Um processo de síntese refere-se a uma inferência de estruturas necessárias para atender a descrição do projeto. A linguagem VHDL não foi concebida originalmente para a síntese, dessa forma nem todas as construções presentes na linguagem podem serem utilizadas para síntese. Há várias etapas na síntese e seu produto final é um arquivo em que uma ferramenta do fabricante consiga realizar o posicionamento e a interligação dos componentes. Dessa forma, com o arquivo e algumas ferramentas do fabricante é possível implementar o circuito descrito em FPGAs e *Application Specific Integrated Circuits* (ASIC).

Num projeto de descrição de hardware em VHDL as duas principais interfaces são: a *entity* e a *architecture.* A declaração da entidade define a interface entre a entidade e o ambiente exterior como por exemplo entradas e saídas. A arquitetura contém especificação das relações entre entrada e saídas de uma entidade (GRUPO DE MICROELETRÔNICA UNIFEI, 2009).

As estruturas mais importantes do VHDL são :

***Entity:*** A *entity* é a parte principal de qualquer projeto, pois descreve a interface do sistema. Tudo que é descrito na *entity* fica automaticamente visível a outras unidades associadas com a *entity*. O nome do sistema é o próprio nome da *entity,* dessa forma deve-se sempre iniciar um projeto em VHDL pela *entity*. Na *entity* é onde descreve-se as portas e a características das mesmas, ou seja, se são de entrada, saída, entrada-saída ou *buffer.* Também descreve-se a característica da entrada se é bit ou um vetor de bits, etc (GRUPO DE MICROELETRÔNICA UNIFEI, 2009).

***Architecture:*** As *architectures* definem comportamentos, logo uma arquitetura é definida em função de uma *entity*. Dentro de uma *architecture* pode-se ter várias sub entidades de projeto ( *entity* e architecture)que juntas formam um sistema mais complexo. Um sistema pode ser descrito em termos de funcionalidade, isto é, o que o sistema faz, ou em termos de estrutura, isto é, como o sistema é composto. A descrição funcional específica as respostas nas saídas em termos das excitações aplicadas nas entradas. Neste caso não há nenhuma informação de como o sistema deverá ser implementado. A descrição estrutural, por sua vez, especifica quais componentes devem ser usados e como devem ser ligados. Esta descrição é mais facilmente sintetizada, porém exige mais experiência do projetista (GRUPO DE MICROELETRÔNICA UNIFEI, 2009).

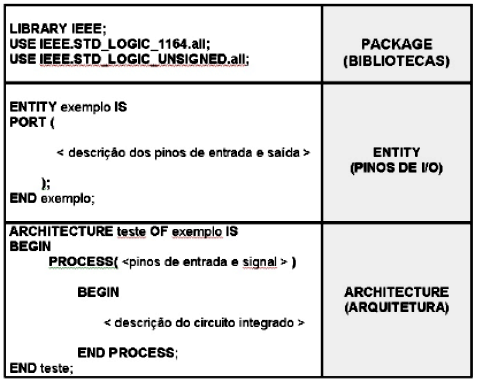
***Package:*** Está contido subprogramas, definição de constantes e as definições dos tipos de *and/or.*  Dessa forma, quando declara-se um *package* importa-se constantes, definições e subprogramas para utilizar na descrição fazendo-se necessário declarar outro *package* quando não há os recursos necessários disponíveis. O *package* deve ser previamente definido antes do início da entity e o uso é feito por meio de duas declarações: library e use. Dos vários packages existentes, o mais conhecido e usado é o STD\_LOGIC\_1164 da IEEE que contém a maioria dos comandos adicionais mais usados em VHDL (GRUPO DE MICROELETRÔNICA UNIFEI, 2009). O uso deste package é dado por:

library IEEE;

use IEEE.std\_logic \_1164.all;

Na figura 4.1 pode-se visualizar a estrutura de um programa em VHDL com a parte do *package, entity e architecture.*

Figura 4.1 – Estrutura de um programa em VHDL.



Fonte: (BENFICA, 2009).

Portanto, para ser possível testar o sistema da codificação 8b10b futuramente em uma FPGA deve-se descrevê-lo em VHDL. A descrição desenvolvida está presente na secção 4.3, juntamente com os softwares e os recursos utilizados para a descrição.

**4.2 FIELD PROGRAMMABLE GATE ARRAY (FPGA)**

O processamento de dados está diretamente relacionado com o tipo de chip usado no projeto desenvolvido. A escolha do chip leva em conta particularidades do projeto como volume de dados processados, número de pinos requisitados, tamanho da palavra processada e a velocidade do processamento. Conforme a aplicação desenvolvida, são necessários maiores recursos financeiros para obter chips mais robustos possibilitando a implementação da aplicação desenvolvida.

Muitos microcontroladores possuem vários recursos internos possibilitando a comunicação com dispositivos externos. Esta versatilidade dos microcontroladores possibilita a conexão com sensores que obtém dados do meio externo gerando uma saída em forma de sinais. Dessa forma, pode-se usar um microcontrolador para processar dados e controlar sistemas. Muitas aplicações no cotidiano são empregados microcontroladores como sistemas indicadores de níveis do carro, máquinas de lavar, semáforos , rádios, portas eletrônicas, entre outras.

Sistemas simples e de baixo rendimento justifica-se o uso de um microcontrolador por possuir baixo valor comercial, consumir baixa potência e possuir recursos internos suficientes para o sistema desenvolvido. Porém, para sistemas mais robustos o uso de microcontroladores torna-se inviável pela necessidade do uso de chips mais desenvolvidos. Estes circuitos integrados com alto rendimento possuem um elevado custo necessitando de soluções mais econômicas (MORIMOTO, 2017).

Os chips FPGA são implementados quando necessita-se de um processamento rápido, alto desempenho e paralelismo. No LHC os eventos estão na casa de nanosegundos, dessa forma necessita-se de dispositivos que possuam um alto desempenho e paralelismo no processamento dos dados sendo necessário o uso de chips FPGA. O ambiente do colisor possui constantes atualizações de software e hardware, portanto a versatilidade de reprogramação dos chips FPGA atende aos requisitos do sistema.

A estrutura de confecção das trilhas e dos circuitos internos dos chips FPGA, fornece uma alta velocidade de processamento de dados sendo possível, em alguns casos, realizar uma instrução por ciclo de *clock.* Porém, um circuito customizado para uma aplicaçãoespecífica é mais rápido do que um chip FPGA implementando a mesma aplicação por conta de chips FPGAs serem projetados para aplicações em geral. (PRADO, 2017).

Atualmente, chips modernos de FPGA possuem recursos extras como uma pequena quantidade de memória RAM e alguns circuitos de apoio. Devido à essa evolução é possível descrever qualquer dispositivo digital que vão desde funções lógicas, portas lógicas e circuitos de memórias até mesmo circuitos mais complexos como microprocessadores. Para a implementação de um projeto com FPGA deve-se possuir um chip previamente programado, um local para armazenar o software com a descrição do circuito e uma fonte ou bateria.

Os dispositivos lógicos podem serem classificados em duas categorias de acordo com a Xilinx (XILINX INC, 2017):

**Lógica Fixa:** Dispositivos cuja função lógica é definida no momento da fabricação permanecendo inalterada após a definição. Esta categoria estão presentes os dispositivos ASIC (de “Application Specific Integrated Circuit”), que são concebidos e construídos, desde a sua máscara de silício, com uma finalidade pré-definida, imutável. Os dispositivos ASIC têm um altíssimo custo de desenvolvimento, justificando sua utilização somente para uma produção em larga escala.

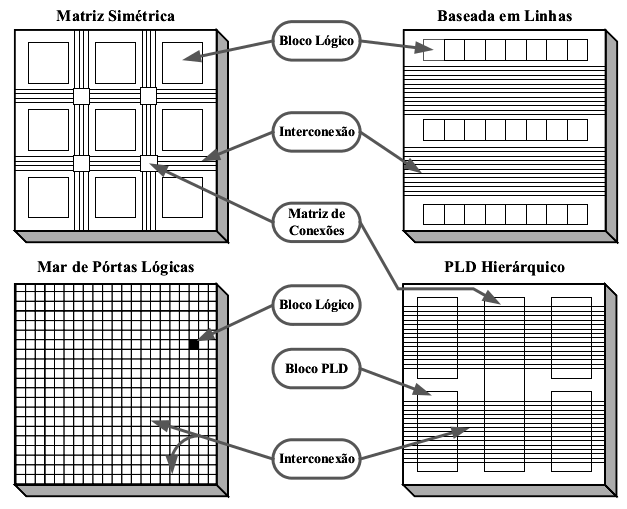
**Lógica Programável:** Os dispositivos lógicos programáveis (PLDs) são dispositivos que possuem uma versatilidade no seu comportamento pois pode-se projetar através de uma IDE (de “Integrated Development Environment”), ou seja, um software de desenvolvimento e testada no mesmo instante diretamente no dispositivo. Por conta desta versatilidade, dispositivos PLDs tornaram-se uma solução para a integração de projetos em sistemas de menor escala de produção.

Os PLDs são classificados em três categorias: “*Simple Programmable Logic Devices”*(SPLDs), “*Complex Programmable Logic Devices”* (CPLDs) e “*Field Programmable Gate Arrays”* (FPGA). Os SPLDs são dispositivos de baixa capacidade lógica, sendo constituídos basicamente por uma matriz de lógica programável do tipo *AND-OR*. Os CPLDs são constituídos por um conjunto de blocos matriciais de lógica compostos e por um conjunto de macrocélulas lógicas, possuindo uma interconexão entre os blocos de lógica e blocos de controle de entrada e saída. Os FPGAs são os dispositivos mais avançados da categoria dos PLDs, uma vez que possuem em um modelo básico uma composição matricial de blocos de lógica, segmentos de conexão, chaves de interconexão e blocos de controle de entrada e saída (MÉLO; DIAS; STEINBACH, 2017).

4.2.1 Estrutura Interna do FPGA

Os FPGAs foram introduzidos pela empresa Xilinx Inc. no ano de 1985 e possuem quatro arquiteturas principais: matriz simétrica, *sea-of-gates, row-based* e PLD hierárquico, como descrito na figura 4.2.

Figura 4.2 – Arquiteturas principais de um FPGA.



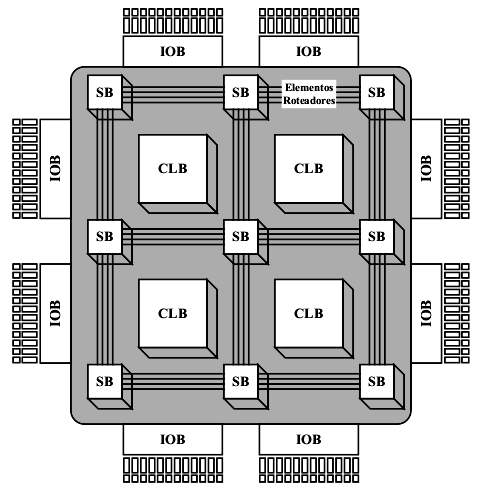
Fonte: Adaptado (ORDONEZ; PENTEADO; SILVA, 2005).

A arquitetura s*ea-of-gates* é um circuito composto por transistores ou blocos lógicos de baixa complexidade possuindo a vantagem de grande disponibilidade de portas lógicas por área. O roteamento é feito sobre as células por não possuir área específica para o mesmo, inutilizando áreas disponíveis para implementar uma determinada lógica. A arquitetura *Row-based*, há blocos lógicos organizados horizontalmente possuindo uma área entre as linhas de blocos lógicos dedicada ao roteamento (ORDONEZ; PENTEADO; SILVA, 2005).

A arquitetura do tipo PLD hierárquico há blocos lógicos denominados “logic array blocs”, podendo serem agrupados pelo recurso de roteamento conhecido como matriz programável de interconexão (PIA). A arquitetura matriz simétrica é flexível no roteamento já que possui canais verticais e horizontais.

Os FPGAs são compostos internamente por blocos lógicos configuráveis (CLBs), chaves de interconexão (*Switch Boxes)*, blocos de entrada e saída (IOB) e canais de roteamento como pode-se visualizar na figura 4.3.

Figura 4.3 – Esquema dos elementos básicos do chip FPGA.



Fonte: (GONÇALVES, 2005).

Os CLBs são idênticos entre si, possuindo a capacidade de implementar funções combinacionais através de *look up tables* (LUT) ou sequenciais através de *flip-flops.*  Os IOBs são componentes responsáveis por realizar a comunicação externa, formados por estruturas bidirecionais que realizam a entrada e saída dos dados. Nos IOBs estão presentes buffer, *flip-flop* de entrada, *buffer tri-state* e *flip-flop* de saída. As *Switch Boxes* são responsáveis pelas conexões entre os blocos lógicos programáveis e os blocos de entrada e saída, através dos elementos roteadores (GONÇALVES, 2005).

O roteamento é uma interconexão entre os blocos lógicos por intermédio de uma rede de camadas de metal. Na família XC4000 da Xilinx há 4 elementos básicos do roteamento:

**Conexões Globais:** São redes de linhas e colunas que realizam a interconexão entre si por chaves de interconexão. Estas redes circundam os CLBs e os IOBs.

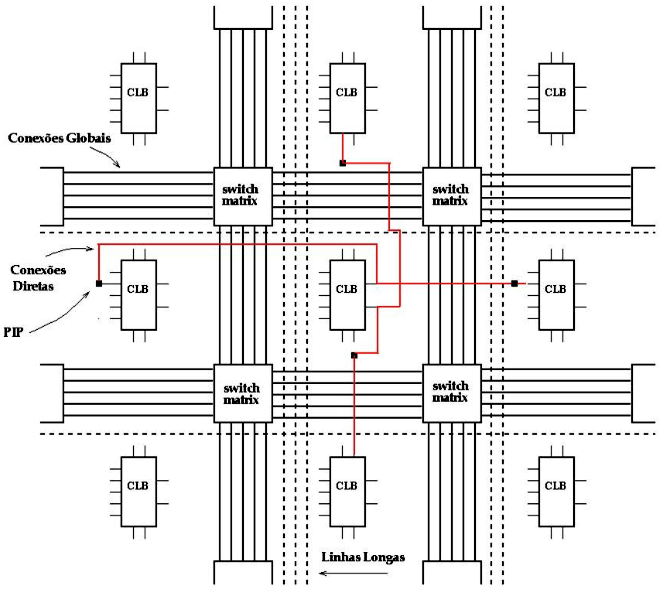
**Matriz de conexão (*Switch Matrix*):** São chaves de interconexão que permitem o roteamento entre os CLBs pelas conexões globais.

**Conexões diretas:** Interligam os CLBs sem utilizar as conexões globais, dessa forma a conexão é mais rápida.

**Linhas Longas:** São conexões que atravessam o circuito inteiro sem atravessar pela matriz de conexão.

Os recursos de roteamento podem serem visualizados na figura 4.4 em que mostra um esquemático do roteamento em um FPGA. As conexões físicas são realizadas por transistores controlados por bits de memória (PIP) ou por chaves de interconexão as denominadas *Switch Matrix* (ORDONEZ; PENTEADO; SILVA, 2005).

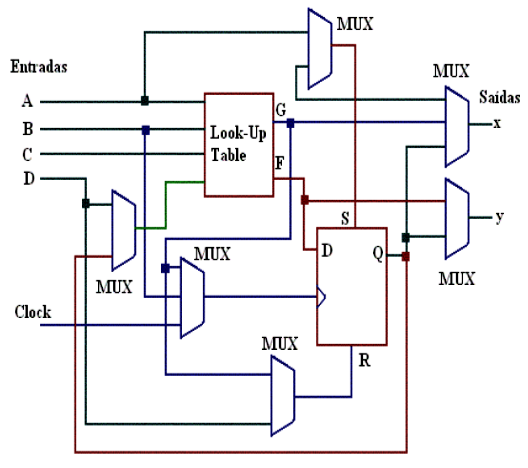
Figura 4.4 – Esquemático de um roteamento em um circuito de FPGA.



Fonte: (NOVELETTO, 2013).

Os blocos lógicos internamente possuem LUTs ou *flip-flops* de acordo com um modelo de implementação de um CLB descrito na figura 4.5. Para estruturas na forma combinacional, ou seja, portas AND, NOR, OR, NAND, XOR, XNOR e NOT, há tabelas dentro de memórias LUTs que descrevem as saídas combinacionais dependendo das entradas. Em estruturas de lógica síncrona usa-se os registradores de deslocamento binário ou mais conhecidos como *Flip-Flops.* Possuem a função de travar o valor na saída, dependendo da entrada e do tipo de *flip-flop* implementado, de acordo com os pulsos de clock (NOVELETTO, 2013).

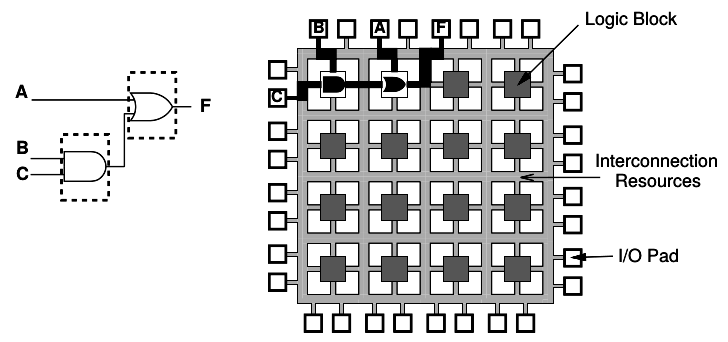
Figura 4.5 – Esquemático de um modelo da estrutura interna do CLB.



Fonte: (NOVELETTO, 2013).

Ao ser descrito um circuito lógico e implementado no FPGA, os CLBs são programados para realizarem funções específicas configurando os canais de roteamento para interligarem os CLBs realizando as funções descritas. Normalmente as LUTs possuem de 2 a 5 entradas o que exige de 4 a 32 células de armazenamento, uma vez que ao programar o CLB são geradas tabelas verdade para as LUTs realizarem uma determinada função. Na figura 4.6 pode-se observar um modelo de implementação de um chip FPGA. O circuito descrito é implementado separando-o em partes, armazenando dentro de cada CLB uma ou várias funções do circuito por meio de tabelas verdade. Estas tabelas verdade são armazenadas dentro de memórias *Static Random Access Memory* (SRAM) da LUT realizando a função que o CLB foi designado (CHUNG, 1994).

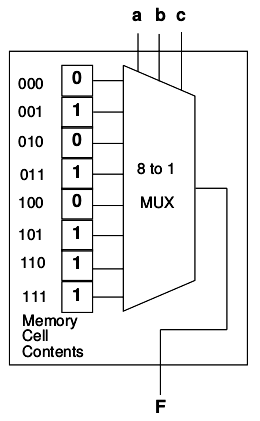
Figura 4.6 – Esquema de implementação de um circuito em um chip FPGA.



Fonte: (CHUNG, 1994).

Uma LUT é composta basicamente por células de memórias de K entradas, necessitando de 2K  células de memórias na LUT. As 2K  células são entradas para um multiplexador de 2K  para 1 em que a seleção é feita pelas K entradas. Nestas células de memória estão presentes as tabelas verdade desejadas de acordo com o circuito descrito. Dessa forma, tendo em mente o circuito lógico descrito na figura 4.6 um esboço de uma LUT que implementa este circuito está descrito na figura 4.7. A programação deve ser refeita toda vez que é desligado o FPGA pois as memórias SRAM que armazenam as tabelas verdades na LUT são voláteis. Normalmente há uma memória *Programmable Read Only Memory* (PROM), ou seja uma memória não volátil, armazenando as tabelas verdades descritas (CHUNG, 1994).

Figura 4.7 – Modelo de LUT de implementação de um circuito lógico.



Fonte: (CHUNG, 1994).

4.2.2 Técnicas de Programação do FPGA

Após a descrição de um sistema em linguagem de descrição de hardware HDL, deve-se realizar uma simulação para ter-se uma ideia inicial se o sistema condiz com o objetivo inicial. Após a obtenção de uma descrição funcional, realiza-se o processo de compilação em duas partes. A primeira parte, denominada síntese, em que o compilador transforma toda a descrição em um modelo de implementação, ou seja, transcreve-se todo o texto da descrição em funções definindo uma estrutura que será implementada nas CLBs. Posteriormente, é criado um *bitstream,* ou seja, são posicionadas as funções em CLBs definidos, definindo todos os tipos de roteamento para realizar as funções lógicas do sistema descrito. Os fabricantes normalmente disponibilizam um software para carregar o *bitstream* gerado na FPGA.

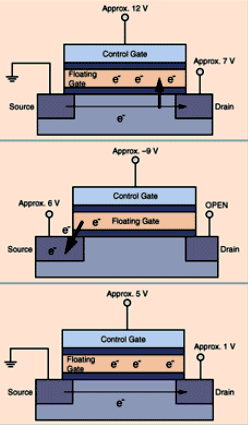
Algumas propriedades dos comutadores presentes no FPGA afetam o desempenho do dispositivo. Estes fatores permeiam em torno do tamanho, resistência, capacitância e a tecnologia empregada na construção. Estes fatores influenciam o desempenho e definem características como volatilidade e capacidade de reprogramação. Há três tipos de tecnologia de programação: SRAM, *Antifuse* e *Floating Gate.*

Na tecnologia SRAM o comutador é controlado por um transistor ativado por células de bits selecionadas por um multiplexador. Como são voláteis necessitam de reprogramação toda vez que o chip é ligado, portanto normalmente utiliza-se uma memória *Read Only Memory* (ROM) ou discos magnéticos para armazenar a descrição do sistema. Dessa forma, pode-se reprogramar o chip automaticamente quando houver um religamento. Esse tipo de tecnologia possui a desvantagem de usar uma área grande, por conta do uso de transistores para formar as células da SRAM (em torno de 5 transistores) e os comutadores (em torno de 1 transistor). A maior vantagem é uma rápida reprogramação e o uso de circuitos integrados padrão.

Um *antifuse* é um dispositivo oposto ao fusível, ou seja, ele começa com uma resistência alta, quando não está programado, e ao aplicar-se uma tensão em torno de 11 a 20 volts entre os terminais o dispositivo adquire uma baixa resistência sendo programado. Na tecnologia *antifuse,* necessita-se de um circuito extra para a programação por conta da alta tensão e correntes em torno de 5 mA. As vantagens dessa tecnologia são menores áreas de implementação do *antifuse* do que as outras tecnologias, uma baixa resistência e uma capacitância parasita menor do que as outras tecnologias.

Na tecnologia *Floating Gate* utiliza-se um transistor *mosfet* ao qual possui um *gate* flutuante isolado eletricamente. Na figura 4.8 pode-se observar em primeiro a operação de escrita no transistor, embaixo a operação de desprogramação do transistor e em último a operação de leitura. Esse tipo de transistor é aplicado com a tecnologia EEPROM, realizando a operação de leitura e escrita de forma elétrica (ROSE; GAMAL; VINCENTELLI, 2017).

Figura 4.8 – Processos de leitura, escrita e apagamento no transistor de porta flutuante.



Fonte: (HITEQUEST, 2004).

Esse transistor é programado aplicando uma tensão de aproximadamente 12 V no *gate* e aproximadamente 7 V no dreno, dessa forma os elétrons vão para o *gate* flutuante. Quando há elétrons no *floating gate,* não permite a passagem de elétrons do *source* para o dreno dessa forma em uma leitura obtém-se o bit 0. Para desprogramar o *gate* flutuante deve-se aplicar uma tensão negativa de 9 V no *gate* e uma tensão de 6 V no *source* e deixar o dreno aberto. Portanto, os elétrons vão do *gate* flutuante para o *source* e ao realizar uma leitura obtém-se o bit 1.

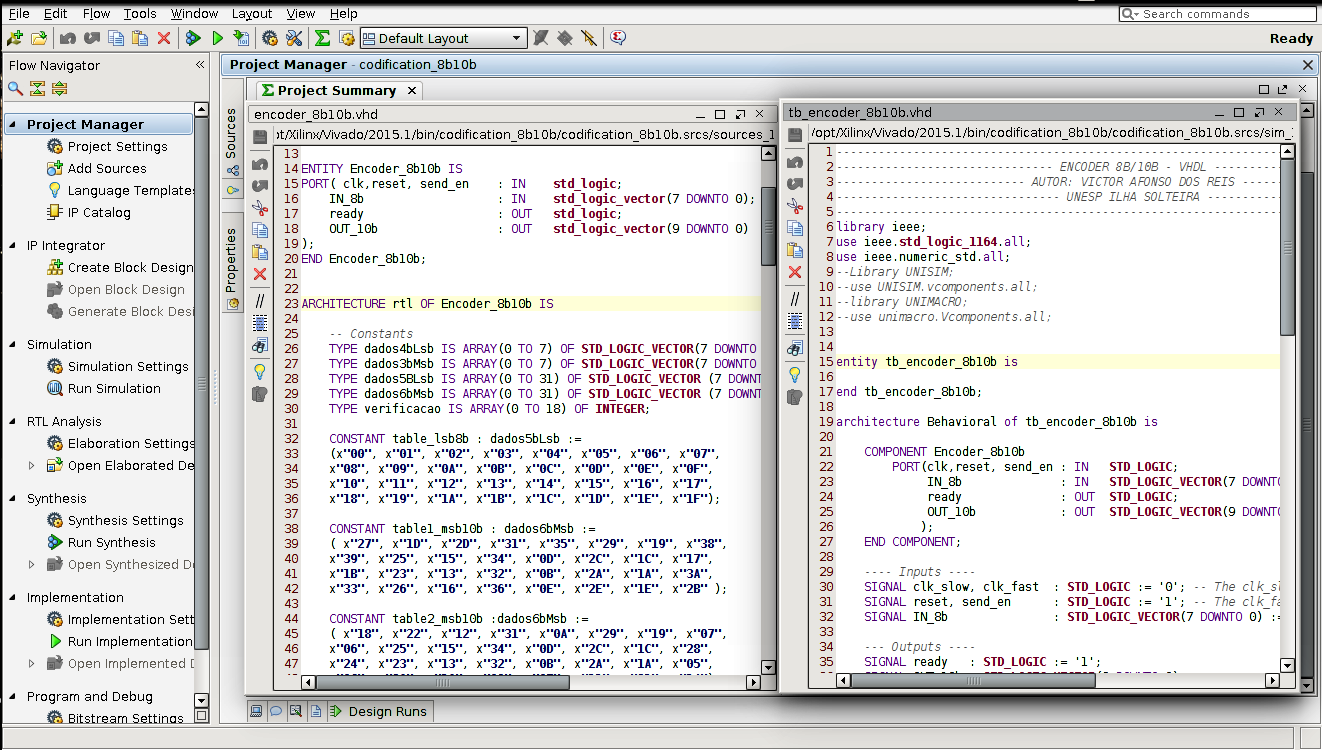
Também são fabricados transistores com *gate* flutuante que usam raios ultra-violetas para inserir ou retirar elétrons do mesmo, neste tipo de dispositivo é empregado uma tecnologia parecida com as memórias EPROMs. Para programação é usado o mesmo procedimento elétrico, porém nessa tecnologia os transistores são apagados por raios ultra-violetas. A maior vantagem é a fácil reprogramabilidade do chip, não necessitando de uma memória PROM externa para reprogramá-lo toda vez que liga-se o sistema. As desvantagens dessa tecnologia é a alta resistência e o alto consumo de energia estática, devido ao resistor de pull-up presente. As tecnologias EEPROM usam o dobro da área para implementar os circuitos do que a tecnologia EPROM (ROSE; GAMAL; VINCENTELLI, 2017).

**4.3 SISTEMA IMPLEMENTADO NO VHDL**

Para descrever o sistema em VHDL utilizou-se o software VivadoTM da empresa Xilinx. A placa alvo selecionado para este projeto foi a *Zynq-7000,* uma placa da Xilinx disponível para o projeto e que futuramente será implementado o circuito descrito. Na figura 4.9 apresenta-se uma parte da descrição do *encoder* que implementa a codificação 8b10b no VHDL. Na esquerda está presente a descrição do sistema e na direita o *testbench* do sistema.

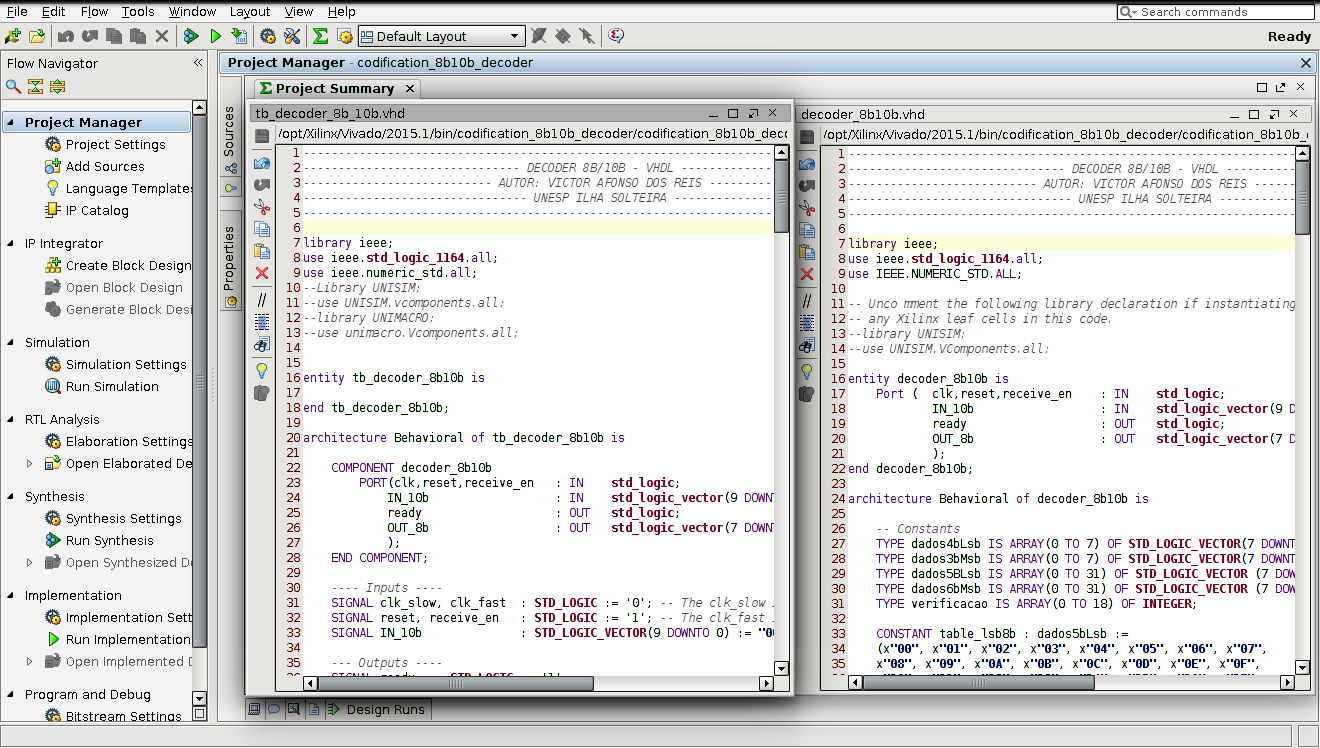
O código completo do *encoder* e do seu *testbench* está no Anexo III. Neste *testbench* são gerados dados de 8 bits que vão desde do equivalente inteiro “0” até o equivalente inteiro “255”, sequencialmente. Para o testbench há dois sinais de clock, um serve para a máquina de estados do sistema e o outro serve para a inserção de dados no sistema. Essa separação deve-se a uma possível utilização de FIFOs para a transmissão.

Figura 4.9 – Descrição do *encoder* da codificação 8b10b em VHDL.

Fonte: Elaborado pelo Autor.

Na figura 4.10 apresenta-se uma parte da descrição do *decoder* que implementa a codificação 8b10b no VHDL. Na direita está presente a descrição do sistema e na esquerda o *testbench* do sistema. O código completo do dec*oder* e do seu respectivo *testbench* está no Anexo IV. Neste *testbench,* são gerados dados de 10 bits de acordo com tabelas da codificação 8b10b retiradas de Cornell (2017). Por meio destas tabelas, pode-se inserir os dados de 10 bits no *decoder* de acordo com o RD.

Figura 4.10 – Descrição do *decoder* da codificação 8b10b em VHDL.

Fonte: Elaborado pelo Autor.

**CAPÍTULO 5**

**RESULTADOS E CONCLUSÕES FINAIS**

O acelerador de partículas LHC buscar encontrar respostas sobre os fundamentos da matéria, mais especificamente as partículas físicas elementares. O estudo envolve uma enorme quantidade de dados e consequentemente uma enorme quantidade de colisões de partículas. Por conta dessa alta taxa de colisões, os sistemas eletrônicos desenvolvidos para o colisor trabalham em uma velocidade muito alta de processamento e estão imersos em um ambiente de uma taxa de radiação elevada. Esta alta taxa é provocada por conta das partículas estarem em alta velocidade, para realizarem a colisão.

A transmissão de dados entre os dispositivos eletrônicos do LHC possui diversos problemas. Estes problemas referem-se à alta velocidade de processamento e a alta radiação que os dispositivos estão expostos. A alta velocidade de processamento gera diversos problemas nas comunicações digitais, como por exemplo a dessincronização entre os dispositivos emissor e receptor. O ambiente com alta radiação induz ruídos nos dados transmitidos, sendo necessário um mecanismo de detecção de erros. A codificação 8b10b possibilita a detecção de erros nos dados transmitidos e manter o sincronismo entre os dispositivos comunicantes. Estas características são obtidas por conta do balanço DC produzido no dado transmitido e por meio da descrição da codificação, ajudando outros circuitos a sincronizarem os dispositivos comunicantes e detecção de erros respectivamente.

O estudo das características da codificação 8b10 foi obtido testando o sistema descrito no programa MatlabTM, dentro do ambiente do Simulink. Primeiramente introduziu-se um dado de acordo com a tabela da figura 5.1, ao qual também está presente o RD vigente no sistema ao introduzir o dado.

Figura 5.1 – Dado de 8 bits inserido no *encoder* e a condição inicial do RD.

|  |  |  |
| --- | --- | --- |
| Entrada | | RD Inicial |
| Decimal | Binário | -1 |
| 51 | 00110011 |

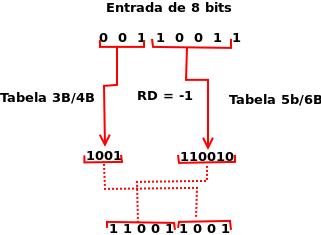
Fonte: Elaborado pelo Autor.

O programa desenvolvido separa os 3 bits mais significativos e os 5 bits menos significativos do dado de entrada. De acordo com o RD do sistema vigente no momento, os dados são codificados para 4 bits e 6 bits de acordo com as tabelas das figuras 3.10 e 3.9, respectivamente.

De acordo com o dado inserido da tabela da figura 5.1, os 5 bits menos significativos são “10011”. Analisando a tabela da figura 3.9 para um RD igual a “-1” obtém-se uma saída de 6 bits igual a “110010”. Para os 5 bits menos significativos (D.3, D.5, D.6, D.9, D.10, D.11, D.12, D.13, D.14, D.17, D.18, D.19, D.20, D.21, D.22, D.25, D.26 e D.28) do dado de entrada da tabela da figura 3.9, a tabela da figura 3.10 inverte. Para os 5 bits menos significativos “10011” a tabela será invertida. Os 3 bits mais significativos do dado de entrada são “001”. Analisando a tabela da figura 3.10 invertendo-a para um RD igual a “-1”, obtém-se uma saída de 4 bits igual a “1001”. A junção do dado codificado em 4 bits com o dado codificado de 6 bits de acordo com a figura 5.2, obtém-se o dado codificado de 10 bits.

O sistema da decodificação do dado de 10 bits possui o caminho inverso da figura 5.2. Primeiramente decodifica-se os 6 bits mais significativos, do dado de 10 bits, para os 5 bits equivalentes, de acordo com a tabela da figura 3.9 para um RD igual a “-1”. Posteriormente analisa-se o dado de 5 bits invertendo a tabela da figura 3.10 ,igual feito no processo de codificação por se tratar do mesmo dado, decodificando o dado de 4 bits para 3 bits usando a tabela conforme um RD igual a “-1”. Caso não haja erro na decodificação do dado, os 3 bits e os 5 bits decodificados são unidos seguindo o caminho inverso da codificação de acordo com a figura 5.2. Porém, caso haja erro há um sinal de erro de saída e a porta de saída apresenta o dado inteiro “256” transformado em binário no *decoder*.

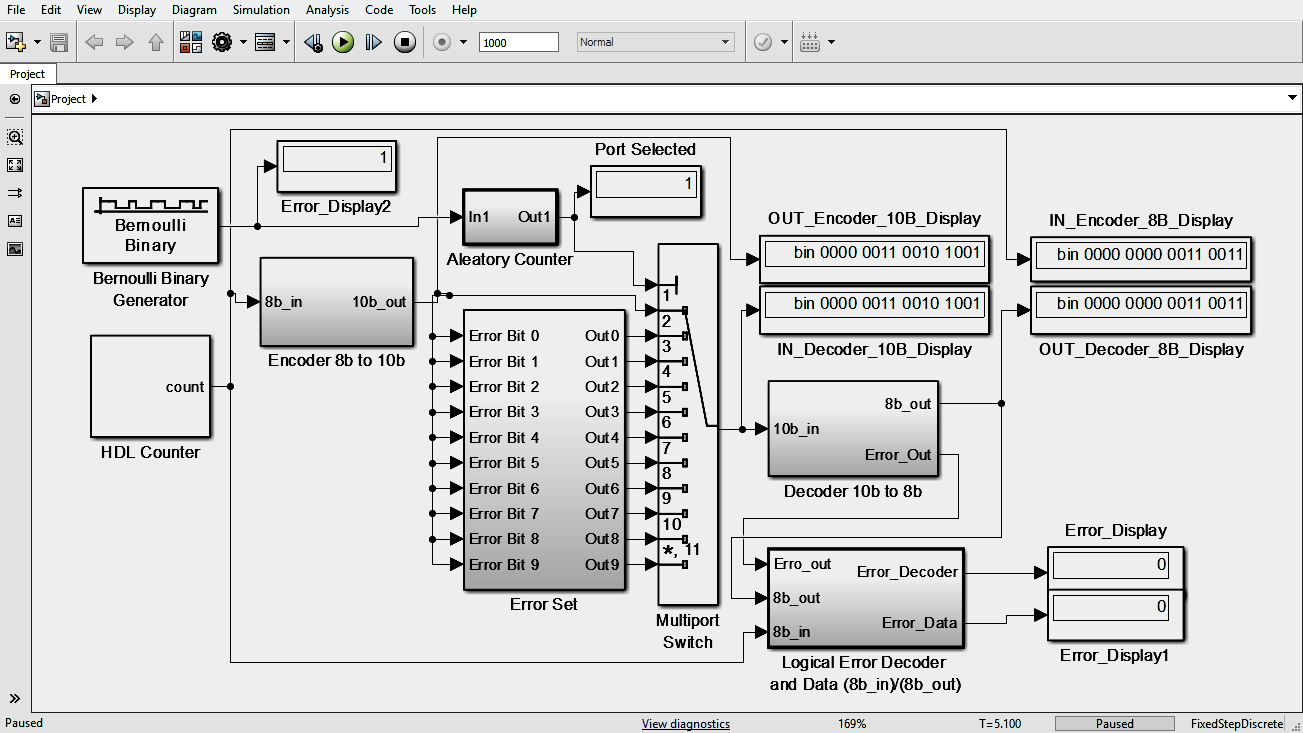
Figura 5.2 – Processo de codificação 8b10b para a simulação do dado de entrada.



Fonte: Elaborado pelo Autor.

Simulando os dados contidos na tabela da figura 5.1 no programa desenvolvido como descrito na secção 3.4 obteve-se o resultado esperado como está descrito na figura 5.3.

Figura 5.3 – Simulação da inserção do dado de 8 bits na entrada do encoder.

Fonte: Elaborado pelo Autor.

Nota-se que o sistema cumpre as características da codificação, pois ao codificar obtêm-se dados para serem transmitidos com a quantidade de bits 0’s e bits 1’s balanceada. Dessa forma, possibilita o *decoder* detectar um possível erro na transmissão de acordo com a descrição da codificação. No sistema implementado a detecção de erro ocorre de duas formas A primeira acontece com a contagem dos erros detectados pelo *decoder,* já a segunda a detecção do erro acontece comparando o dado inserido no *encoder* e os de saída do *decoder* gerando uma contagem caso forem diferentes.

Em uma transmissão de dados em alta velocidade é impossível não haver erros na transmissão, mesmo com a implementação de uma codificação muito robusta. Dessa forma, ao simular o sistema implementado observa-se que o número de erros que o subsistema *decoder* identifica é menor do que o número de erros ao comparar os dados de entrada e saída do sistema. Isto acontece, pois ao gerar ruídos no dado transmitido é possível que o erro gerado transforme o dado em um dado diferente que esteja contido nas tabelas de codificação. Portanto, o dado será decodificado para um valor diferente do que foi inserido. O erro coletado na comparação dos dados contém o número de erros fornecidos pelo *decoder* somados com o número de erros entre o dado de entrada no *encoder.*

Para observar a proporção de erros em relação ao número de dados transmitidos no sistema, configurou-se no ambiente *Simulink* o número máximo de simulações para 1000 com passos de 0,1 totalizando 10000 simulações. Variou-se a probabilidade de erro no canal no bloco *Bernoulli Binary Generator* de 0 até 100 por cento, coletando o número de erros da comparação entre o dado de entrada e saída do sistema, ou seja, o número obtido no *display* *Error\_Display1* de acordo com a figura 5.3. Na tabela da figura 5.4, observa-se a porcentagem de erro obtido na transmissão de acordo com a variação da probabilidade de erro no canal.

Figura 5.4 – Porcentagem de Erros Obtidos com a Variação da Probabilidade de Erro no Canal.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Probabilidade de Erro no Canal (%)** | **Porcentagem de Erros Obtidos na Transmissão (%)** |  | **Probabilidade de Erro no Canal (%)** | **Porcentagem de Erros Obtidos na Transmissão (%)** |  | **Probabilidade de Erro no Canal (%)** | **Porcentagem de Erros Obtidos na Transmissão (%)** |
| **1** | 1,80 | **27** | 38,99 | **53** | 64,60 |
| **2** | 3,49 | **28** | 40,20 | **54** | 65,43 |
| **3** | 5,30 | **29** | 41,41 | **55** | 66,37 |
| **4** | 6,93 | **30** | 42,58 | **56** | 67,30 |
| **5** | 8,93 | **31** | 43,72 | **57** | 68,18 |
| **6** | 10,70 | **32** | 44,85 | **58** | 68,93 |
| **7** | 12,09 | **33** | 45,90 | **59** | 69,77 |
| **8** | 13,54 | **34** | 46,79 | **60** | 70,58 |
| **9** | 15,15 | **35** | 47,94 | **61** | 71,43 |
| **10** | 16,60 | **36** | 49,20 | **62** | 72,40 |
| **11** | 18,02 | **37** | 50,14 | **63** | 73,01 |
| **12** | 19,54 | **38** | 50,92 | **64** | 73,80 |
| **13** | 20,64 | **39** | 51,87 | **65** | 74,59 |
| **14** | 22,15 | **40** | 52,59 | **66** | 75,51 |
| **15** | 23,56 |  | **41** | 53,53 |  | **67** | 76,36 |
| **16** | 24,82 | **42** | 54,74 | **68** | 77,12 |
| **17** | 26,38 | **43** | 55,63 | **69** | 78,00 |
| **18** | 28,08 | **44** | 56,73 | **70** | 78,80 |
| **19** | 29,41 | **45** | 57,64 | **71** | 79,71 |
| **20** | 30,50 | **46** | 58,60 | **72** | 80,49 |
| **21** | 32,04 | **47** | 59,70 | **73** | 81,21 |
| **22** | 33,12 | **48** | 60,43 | **74** | 81,87 |
| **23** | 34,03 | **49** | 61,18 | **75** | 82,53 |
| **24** | 35,30 | **50** | 62,10 | **76** | 83,06 |
| **25** | 36,73 | **51** | 62,92 | **77** | 83,85 |
| **26** | 37,86 |  | **52** | 63,80 |  | **78** | 84,73 |

Fonte: Elaborado pelo Autor.

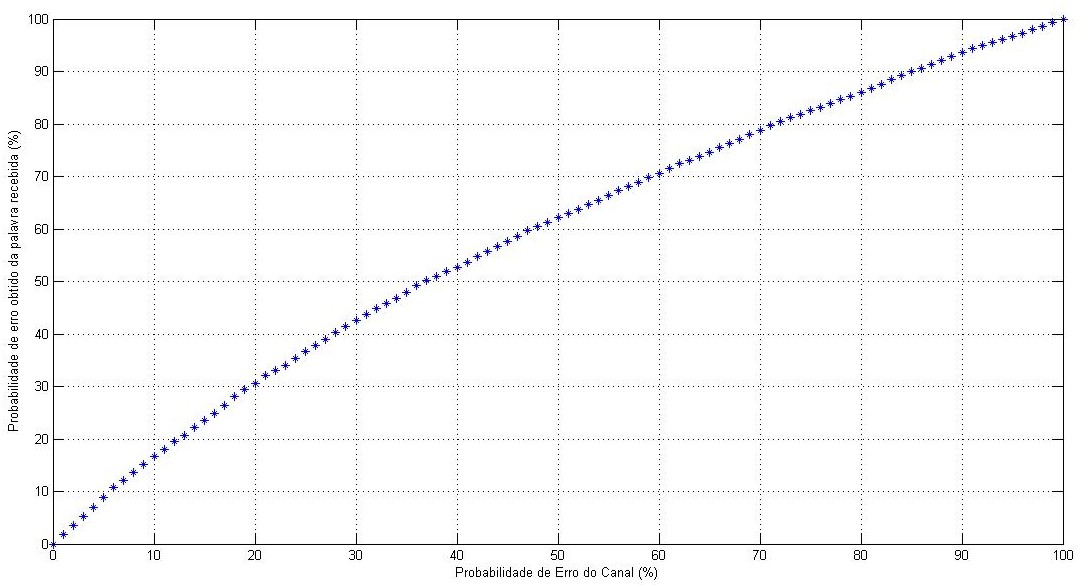
Figura 5.4 – Porcentagem de Erros Obtidos com a Variação da Probabilidade de Erro no Canal (Continuação).

| **Probabilidade de Erro no Canal (%)** | **Porcentagem de Erros Obtidos na Transmissão (%)** |  | **Probabilidade de Erro no Canal (%)** | **Porcentagem de Erros Obtidos na Transmissão (%)** |  | **Probabilidade de Erro no Canal (%)** | **Porcentagem de Erros Obtidos na Transmissão (%)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **79** | 85,22 |  | **87** | 91,38 |  | **95** | 96,70 |
| **80** | 86,06 |  | **88** | 92,14 |  | **96** | 97,32 |
| **81** | 86,82 |  | **89** | 92,85 |  | **97** | 97,96 |
| **82** | 87,61 |  | **90** | 93,58 |  | **98** | 98,64 |
| **83** | 88,41 |  | **91** | 94,29 |  | **99** | 99,35 |
| **84** | 89,30 |  | **92** | 94,86 |  | **100** | 100 |
| **85** | 90,02 |  | **93** | 95,53 |  |  |  |
| **86** | 90,59 |  | **94** | 96,13 |  |  |  |

Fonte: Elaborado pelo Autor.

Na figura 5.5 é apresentado um gráfico com os dados da tabela da figura 5.4. Observa-se que o comportamento da codificação em relação aos erros acometidos em um bit do dado transmitido, é aproximadamente polinomial quando a probabilidade de erro no canal ultrapassa aproximadamente 30%. Pelos dados observados, a diferença entre os erros coletados pela comparação entre os dados de entrada do *encoder* e de saída do *decoder* e os erros pelo sinal de saída do *decoder*  não ultrapassaram uma diferença maior que 20 %, nos casos que ultrapassam 50 % de probabilidade de erro no canal. Em casos de baixa probabilidade de erro no canal de transmissão, a diferença não ultrapassa 10%. Dessa forma, observa-se que a codificação é adequada para a transmissão uma vez que possibilita uma redução drástica dos erros no canal de transmissão.

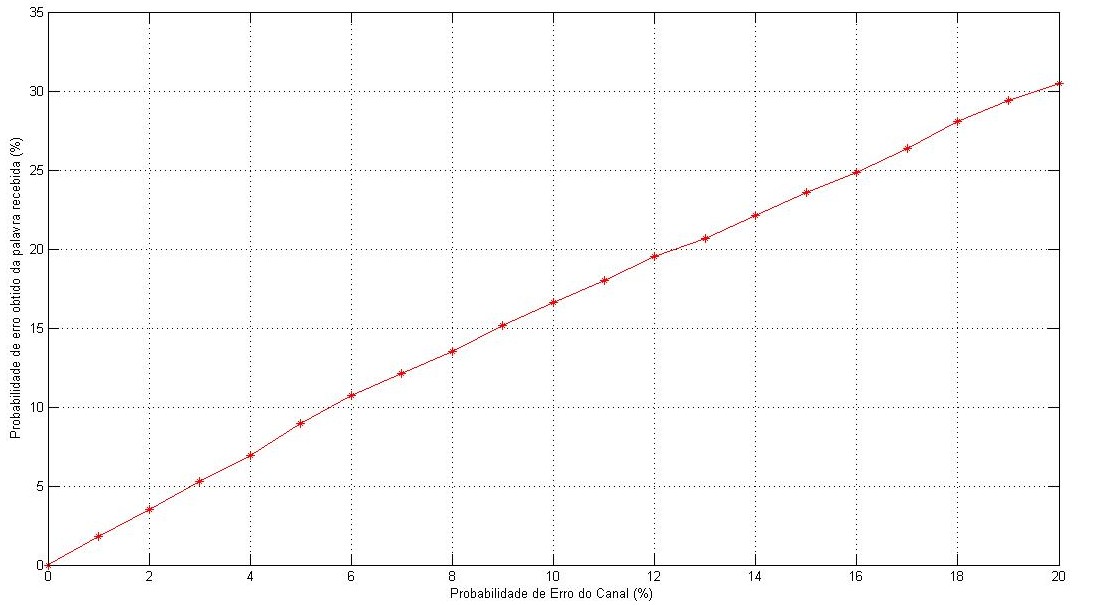
Figura 5.5 – Gráfico do comportamento da codificação em relação aos erros.



Fonte: Elaborado pelo Autor.

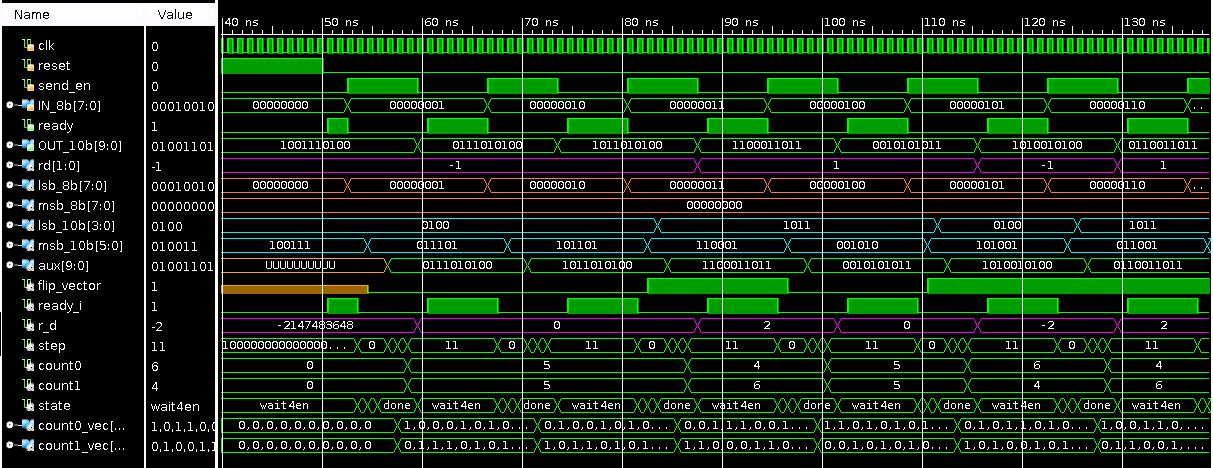
Para transmissões em alta velocidade, normalmente trabalha-se com uma probabilidade de erro de aproximadamente 5% no canal de transmissão. Na figura 5.6 é apresentado o gráfico com os dados da tabela da figura 5.3 traçados com uma probabilidade de erro no canal de transmissão de até 20%. Neste caso, observa-se um comportamento aproximadamente linear e é obtido uma porcentagem de erro no canal de 8.93% para uma probabilidade de erro no canal de 5 % , de acordo com a tabela da figura 5.3. Para uma probabilidade de erro no canal de 5 %, o *decoder* identificou erros em torno de 6 %, dessa forma a porcentagem de erro na transmissão foi reduzida drasticamente por meio da codificação.

Figura 5.3 - Comportamento da codificação com a probabilidade de erros de até 20%.

Fonte: Elaborado pelo Autor.

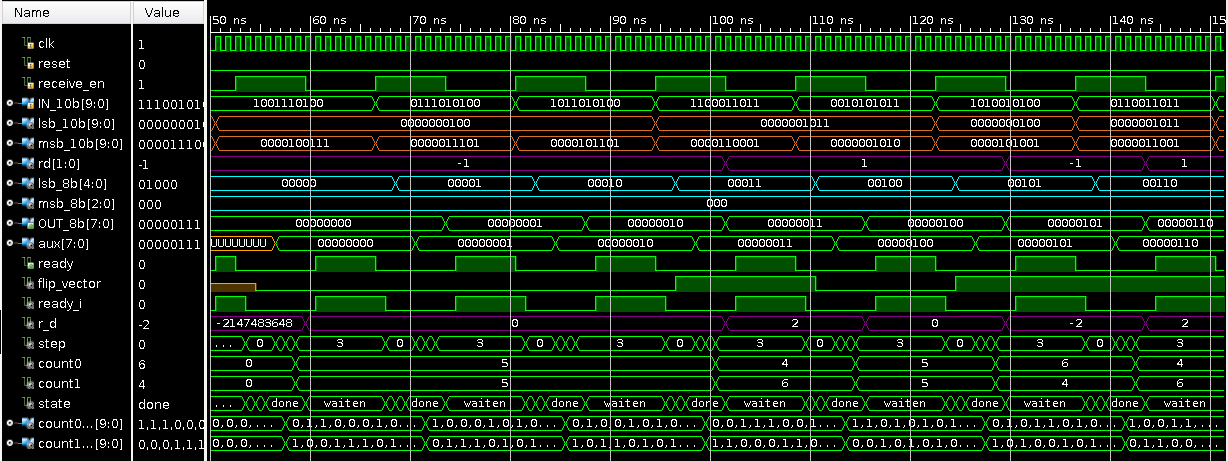
Para o sistema da codificação 8b10b implementado em VHDL de acordo com a secção 4.3, simulou-se inserindo dados de 8 bits e 10 bits na entrada do *encoder* e do *decoder*, respectivamente. Na figura 5.4 é apresentado a simulação do *encoder* implementado em VHDL. Observa-se pela figura que o sinal RD representado na cor roxa não varia somente em 0, +2 e -2. Este sinal representa a diferença entre os os bits 1’s e os bit 0’s do dado codificado para 10 bits, dessa forma revela que a codificação foi descrita de forma correta por satisfazer as características propostas.

Figura 5.4 – Simulação do sistema do *encoder* Descrito em VHDL.

Fonte: Elaborado pelo Autor.

Na figura 5.5 apresenta-se a simulação do *decoder* implementado em VHDL. Para a simulação introduziu-se dados sequenciais da codificação, ou seja, introduziu-se dados de 10 bits referentes aos dados de 8 bits que vão ,segundo os equivalentes inteiro, desde “0” até “255”, de acordo com o RD do sistema. Estes dados foram introduzidos por meio de tabelas obtidas de acordo com Cornell (2017), dentro do *testbench* do *decoder*. Pelos dados de saída de 8 bits do *decoder* observou-se uma sequência nos dados verificando que o sistema é funcional e realiza o objetivo proposto da descrição.

Figura 5.5 - Simulação do sistema do de*coder* Descrito em VHDL.



Fonte: Elaborado pelo Autor.

No *encoder* e no *decoder* observa-se que há um tempo entre a entrada do dado e a saída produzida pelo mesmo. Este fato deve-se à máquina de estados descrita nos dois sistemas e pelos dois sinais de *clock* presentes, um designado ao sistema e o outro para a entrada do dado. A proporção entre os 2 *clocks* é de 1 para o *clock* do dado para 7 do *clock* do sistema, ou seja, para 1 período do *clock* da inserção dos dados já aconteceu 7 períodos do *clock* do sistema.

Tanto o sistema do *encoder* e do *decoder,* foram intensamente testados e analisados por meio das entradas e saídas produzidas, comparadas com as tabelas descritas nas figuras 3.9 e 3.10. Observou-se que o sistema descrito no software MatlabTM no ambiente do Simulink é capaz de codificar e decodificar dados seguindo a descrição da codificação 8b10b. As características obtidas da codificação revelam que a codificação é aplicável para a transmissão de dados no sistema que está em desenvolvimento pelo SPRACE e Fermilab para uma possível atualização do LHC.

No sistema desenvolvido no VHDL observou-se o bom funcionamento tanto do encoder quanto do *decoder* após longos testes e análise dos dados descritos. Para o desenvolvimento houve um estudo da linguagem de descrição de hardware através de resoluções de exercícios do livro D’AMORE (2012), que estão presentes no anexo V. Portanto conclui-se que o sistema descrito em VHDL da codificação está pronto para começar uma implementação em um FPGA, havendo um grande aprendizado do software MatlabTM,juntamente com o ambiente Simulink, e da linguagem de descrição de hardware VHDL.

**REFERÊNCIAS** **BIBLIOGRÁFICAS**

COMER, Douglas E. Redes de Computadores e Internet. 6. ed. São Paulo: Bookman, 2016. 557 p.

MACHADO, Renato. Problemas de Transmissão. 2016. Disponível em: <http://coral.ufsm.br/gpscom/professores/Renato Machado/ComunicacaoDeDados/ComDados09Renato.pdf>. Acesso em: 5 jan. 2017.

SPRACE. SPRACE research group. Disponível em: <http://sprace.org.br/>. Acesso em:

5 jan. 2017.

ROSSI, L; BRÜNING, O. High Luminosity Large Hadron Collider A description for the

European Strategy Preparatory Group. Geneva, Aug. 2012. Disponível em:

<https://cds.cern.ch/record/1471000>. Acesso em : 5 jan. 2017.

PAIVA, Thiago Costa de. Remote Development Environment With Reconfigurable Components In The Advanced Telecom Computing Architecture Context. 2016. 126 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Engenharia Elétrica, Universidade Estadual Paulista (UNESP), Ilha Solteira, 2016

WIKIPÉDIA. Grande Colisor de Hádrons. Disponível em: <https://pt.wikipedia.org/wiki/Grande\_Colisor\_de\_Hádrons>. Acesso em: 6 jan. 2017

RANDALL, Lisa. Batendo à Porta do Céu: O bóson de higgs e como a física moderna ilumina o universo. São Paulo: Companhia das Letras, 2013. 576 p.

FERREIRA, Bruno Carneiro. DETECÇÃO DE RAIOS CÓSMICOS COM CALORIMETRIA DE ALTAS ENERGIAS. 2009. 118 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Niversidade Federal do Rio de Janeiro, Rio de Janeiro, 2009

TEMPLETON, Graham. How does the Large Hadron Collider work? Disponível em: <https://www.extremetech.com/extreme/210215-extremetech-explains-what-is-the-large-hadron-collider>. Acesso em: 05 jan. 2017.

BRÜNING, Lucio Rossi And Oliver et al. High Luminosity Large Hadron Collider A description for the European Strategy Preparatory Group. Disponível em: <https://cds.cern.ch/record/1471000/files/CERN-ATS-2012-236.pdf>. Acesso em: 05 jan. 2017

TSESMELIS, Emmanuel. THE COMPACT MUON SOLENOID (CMS) EXPERIMENT: THE LHC FOR HIGH ENERGY AND LUMINOSITY. Disponível em: <http://www.desy.de/~garutti/LECTURES/ParticleDetectorSS12/JournalClub/lhc-CMS.pdf>. Acesso em: 05 jan. 2017

CMS Collaboration. Techninal Proposal for the Phase-II Upgrade of the Compact Muon

Solenoid. 2015. Disponível em: <<https://cds.cern.ch/record/2020886/files/LHCC-P-008.pdf>>.

HEILBRON, John Lewis et al. THE HISTORY OF THE MODERN SCIENCE. New York: Oxford University Press, 2003. 940 p.

BULLETIN, Cern. Some LHC milestones... Disponível em: <http://cds.cern.ch/journal/CERNBulletin/2008/38/News Articles/1125888?ln=en>. Acesso em: 05 jan. 2017.

LUIZ, Leandro da Conceição; MONTEIRO, Keila Thaís da Silva; BATISTA, Rafaela Tavares. Os aceleradores de partículas e sua utilização na p rodução de radiofármacos. Disponível em: <http://rbfarma.org.br/files/rbf-2011-92-3-2.pdf>. Acesso em: 05 jan. 2017.

TOMEI T. R. F. P. Busca por Dimensões Extras no Detector CMS do Large Hadron Collider.

Tese (Doutorado) — IFT UNESP, São Paulo, 6 2012. An optional note. Disponível em:

<<http://base.repositorio.unesp.br/handle/11449/102537>>.

BARNEY, David. CMS Detector Slice. Disponível em: <https://cds.cern.ch/record/2120661>. Acesso em: 05 jan. 2017.

SPRACE. O que é o Modelo Padrão? Disponível em: <https://www.sprace.org.br/divulgacao/o-que-e-o-modelo-padrao>. Acesso em: 05 jan. 2017.

PROJECT, Comtemporary Education. O modelo padrão das partículas e interações fundamentais. Disponível em: <http://www.cpepphysics.org/images/particle\_chart\_PT.jpg>. Acesso em: 05 jan. 2017.

OSTERMANN, Fernanda; CAVALCANTI, Cláudio J. de H.. FÍSICA MODERNA E CONTEMPORÂNEA NO ENSINO MÉDIO: ELABORAÇÃO DE MATERIAL DIDÁTICO, EM FORMA DE PÔSTER, SOBRE PARTÍCULAS ELEMENTARES E INTERAÇÕES FUNDAMENTAIS. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/85023/000269902.pdf?sequence=1>. Acesso em: 05 jan. 2017.

MOREIRA, Marco Antonio. O Modelo Padrão da Física de Partículas. Disponível em: <http://www.scielo.br/pdf/rbef/v31n1/v31n1a06>. Acesso em: 05 jan. 2017.

RAMALHO, Lucas Arruda. Development of CMS L1 Tracker Data Sourcing and Serial Link Comissioning System. 2017. 150 f. Tese (Doutorado) - Curso de Engenharia Elétrica, Ilha Solteira, Ilha Solteira, 2017.

CMS. The Trigger Systems. Disponível em: <http://cmsdoc.cern.ch/cms/TDR/TRIGGER-public/trigger.html>. Acesso em: 05 jan. 2017.

ALMEIDA, Celso de. Ransmissão Digital em Banda Base. Disponível em: <http://www.decom.fee.unicamp.br/~celso/mac/bandabase.pdf>. Acesso em: 07 jan. 2017.

PIMENTEL, Cecilio José Lins. Comunicação Digital. Rio de Janeiro: Brasport, 2007. 397 p.

RUSCHEL, Orlando T.. Princípios da Comunicação Digital. Porto Alegre: Edipucrs, 1996. 227 p.

FOROUZAN, Behrouz A.. Comunicação de Dados e Redes de Computadores. 4. ed. São Paulo: Amgh, 2010. 1134 p.

LATTICE SEMICONDUCTOR. 8b/10b Encoder/Decoder. Hillsboro: Lattice Semiconductor, 2015.

STALLINGS, William. Gigabit Ethernet. Disponível em: <http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-2/ipj-archive/article09186a00800c85a6.html>. Acesso em: 09 jan. 2017.

SATA COMMUNICATION. Encoding. Disponível em: <https://sites.google.com/site/ee3550usbesata/encoding>. Acesso em: 10 jan. 2017.

CORNELL. Table 8b10b. Disponível em: <http://www.cs.cornell.edu/courses/cs3410/2013sp/lab/tables/dy\_encoding\_table.pdf>. Acesso em: 11 jan. 2017.

MORIMOTO, Carlos E.. FPGA. Disponível em: <http://www.hardware.com.br/termos/fpga>. Acesso em: 17 jan. 2017.

PRADO, André Castelan. FPGAs substituindo microcontroladores simples? Disponível em: <https://www.embarcados.com.br/fpgas-substituindo-microcontroladores-simples/>. Acesso em: 17 jan. 2017.

XILINX INC. What is Programmable Logic? Disponível em: <http://www.xilinx.com/ company/about/programmable.html>. Acesso em: 17 jan. 2017.

MÉLO, Francisco Édson Nogueira de; DIAS, Roberto Alexandre; STEINBACH, Reginaldo. FPGA PARA TODOS: UM PROJETO PARA A DISSEMINAÇÃO DA TECNOLOGIA DE LÓGICA PROGRAMÁVEL. Disponível em: <http://www.abenge.org.br/CobengeAnteriores/2011/sessoestec/art1986.pdf>. Acesso em: 17 jan. 2017.

ORDONEZ, Edward David Moreno; PENTEADO, Cesar Giacomini; SILVA, Alexandre César Rodrigues da. Microcontroladores e FPGAs: Aplicações em Automação. São Paulo: Novatec, 2005. 378 p.

NOVELETTO, Fabrício. Dispositivos de Lógica Programável. Joinville: Udesc, 2013. Color. Disponível em: <http://www.joinville.udesc.br/portal/professores/noveletto/materiais/Dispositivos\_Logica\_Prog.pdf>. Acesso em: 18 jan. 2017.

GONÇALVES, Flávio Alessandro Serrão. Pré-Regulador Retificador Entrelaçado (Interleaved) ZCS-FM Boost, com Controle Digital Através de Dispositivo FPGA e Linguagem VHDL. 2005. 277 f. Tese (Doutorado) - Curso de Engenharia Elétrica, Engenharia Elétrica, Universidade Estadual Paulista (UNESP), Ilha Solteira, 2005.

CHUNG, Kevin Charles Kenton. Architecture and Synthesis of Field-Programmable Gate Arrays with Hard-wired Connections. 1994. 156 f. Tese (Doutorado) - Curso de Electrical Engineering, Electrical And Computer Engineering, University Of Toronto, Toronto, 1994.

ROSE, Jonathan; GAMAL, Abbas El; VINCENTELLI, Alberto Sangiovanni. Architecture and Synthesis of Field-Programmable Gate Arrays with Hard-wired Connections. Disponível em: <http://isl.stanford.edu/groups/elgamal/abbas\_publications/J029.pdf>. Acesso em: 19 jan. 2017.

HITEQUEST, Alex Paikin For. Flash memory. 2004. Disponível em: <http://www.hitequest.com/Kiss/Flash\_terms.htm>. Acesso em: 20 jan. 2017.

GRUPO DE MICRO ELETRÔNICA. Universidade Federal de Itajubá. Tutorial VHDL. Disponível em: <https://www.passeidireto.com/arquivo/18536081/apostila\_vhdl\_unifei>. Acesso em: 20 jan. 2017.

BENFICA, Juliano D’ornelas. LINGUAGEM DE DESCRIÇÃO DE HARDWARE: VHDL. 2009. Disponível em: <https://www.passeidireto.com/arquivo/1756338/apostila-vhdl-juliano-benfica>. Acesso em: 20 jan. 2017.

D'AMORE, Roberto. VHDL: Descrição e Síntese de Circuitos Digitais. 2. ed. Rio de Janeiro: Ltc, 2012. 292 p.

WIDMER, Albert. X.; FRANASZEK, Peter. A. **A DC-balanced, partitioned-block, 8B/10B transmission code**, IBM Journal of Research and Development, vol. 27, no. 5, pp. 440-451, Sep. 1983.

**ANEXO I**

Algorítimo desenvolvido para o subsistema *Encoder 8b to 10b* para realizar a codificação 8b10b, utilizando a ferramenta *Embedded MATLABTM Function Block*:

function OUT\_10b = encoding8b10b(IN\_8b)

% -------"COFICADOR 8B/10B - VICTOR AFONSO DOS REIS - FEIS UNESP"---------

%-----Entradas e saídas---------------------

%IN:[X7 X6 X5 X4 X3 X2 X1 X0]

%MSB\_8b\_in: X7 X6 X5

%LSB\_8b\_in: X4 X3 X2 X1 X0

%OUT: [Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0]

%MSB\_10B: Y9 Y8 Y7 Y6 Y5

%LSB\_10B: Y4 Y3 Y2 Y1

%------TRATAMENTO DA ENTRADA-------------------

LSB\_8b = bitand(IN\_8b, uint16(31));

%A função bitshift(dado binário, -5 )corre os bits 5 lugares para direita

MSB\_8b = bitshift(bitand(IN\_8b,uint16(224)), -5);

%------Inicialização das saídas---------------

MSB\_10b = uint16(0);

LSB\_10b = uint16(0);

saida = uint16(0);

%-----Inicialização das Saídas------------

%persistent: Mantém um registro de memória no FPGA até ser modificado

persistent c1;

persistent c2;

persistent c3;

persistent m1;

persistent m2;

persistent m3;

persistent v1;

persistent k;

if isempty (c1) %Se c1 não está inicializado (Ou seja, quando começa a

%funcionar o FPGA)

%C1 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

c1 = [uint16(0); uint16(1);uint16(2);uint16(3);uint16(4);uint16(5);

uint16(6);uint16(7);uint16(8);uint16(9);uint16(10);uint16(11);

uint16(12);uint16(13);uint16(14);uint16(15);uint16(16);uint16(17);

uint16(18);uint16(19);uint16(20);uint16(21);uint16(22);uint16(23);

uint16(24);uint16(25);uint16(26);uint16(27);uint16(28);uint16(29);

uint16(30);uint16(31)];

end

if isempty (c2) %Se c2 não está inicializado (Ou seja, quando começa a

%funcionar o FPGA)

%C2 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

c2=[uint16(39); uint16(29);uint16(45);uint16(49);uint16(53);uint16(41);

uint16(25);uint16(56);uint16(57);uint16(37);uint16(21);uint16(52);

uint16(13);uint16(44);uint16(28);uint16(23);uint16(27);

uint16(35);uint16(19);uint16(50);uint16(11);uint16(42);uint16(26);

uint16(58);uint16(51);uint16(38);uint16(22);uint16(54);uint16(14);

uint16(46);uint16(30);uint16(43)];

end

if isempty (c3) %Se c3 não está inicializado (Ou seja, quando começa a

%funcionar o FPGA)

%C2 vetor coluna com valor em fixed point inteiro sem sinal 16 bits

c3=[uint16(24);uint16(34);uint16(18);uint16(49);uint16(10);uint16(41);

uint16(25);uint16(7);uint16(6);uint16(37);uint16(21);uint16(52);

uint16(13);uint16(44);uint16(28);uint16(40);uint16(36);uint16(35);

uint16(19);uint16(50);uint16(11);uint16(42);uint16(26);uint16(5);

uint16(12);uint16(38);uint16(22);uint16(9);uint16(14);uint16(17);

uint16(33);uint16(20)];

end

if isempty(m1) %Se m1 não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%m1 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

m1=[uint16(0);uint16(1);uint16(2);uint16(3);uint16(4);uint16(5);

uint16(6); uint16(7)];

end

if isempty(m2) %Se m2 não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%m2 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

m2=[uint16(4);uint16(9);uint16(5);uint16(3);uint16(2);uint16(10);

uint16(6);uint16(1)];

end

if isempty(m3) %Se m3 não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%m3 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

m3=[uint16(11);uint16(9);uint16(5);uint16(12);uint16(13);uint16(10);

uint16(6); uint16(14)];

end

if isempty(v1) % Se v1 não está inicializado

% v1 vetor para verificação da codificação 3b/4b

v1 = [uint16(4);uint16(6);uint16(7);uint16(8);uint16(10);uint16(11);

uint16(12); uint16(13);uint16(14);uint16(15);uint16(18);uint16(19);

uint16(20);uint16(21);uint16(22); uint16(23);uint16(26);uint16(27);

uint16(29)];

end

if isempty (k) %Se k não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%k vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

k = [uint16(488); uint16(535)];

end

%------------- Inicialização RD -----------------

persistent rd;

if isempty(rd) %Se rd não está inicializado (Ou seja, quando começar

%a funcionar o FPGA)

rd = fi(-1,1,8,0); %RD fixed-point inteiro com sinal 8 bits

end

%----------- Convert 5b/6b do LSB --------------------------

j = 0;

for i = 1:1:32

if (LSB\_8b == c1(i))

j=i;

end

end

if (rd == fi(1,1,8,0))

MSB\_10b = c3(j);

end

if (rd == fi(-1,1,8,0))

MSB\_10b = c2(j);

end

% --------------- Convert 3b/4b of MSB ----------------------

flip = uint8(0);

for i=1:1:19

if(uint16(j) == v1(i))

flip = uint8(1); % Procura na matriz v1 se o dado 5b inserido

% inverte ou não a tabela. Caso o dado inverta

% o flip será 1 caso contrário será 0.

end

end

%Procura na matriz m1 a posição que coincida com o dado de entrada

if (flip == uint8(0)) % Verifica se inverte as tabelas de codificação

for i=1:1:8

if (MSB\_8b == m1(i))

j=i; % Coincide, armazena-se em j essa posição

end

end

if(rd == fi(1,1,8,0)) % Se rd = 1, então LSB\_10b fica com m3(j),

%sendo j, a posição achada anteriormente

LSB\_10b = m3(j);

end

if(rd == fi(-1,1,8,0)) % Se rd =-1, então LSB\_10b fica com m2(j),

%sendo j, a posição achada anteriormente

LSB\_10b = m2(j);

end

else

for i=1:1:8

if (MSB\_8b == m1(i))

j=i; % Coincide, armazena-se em j essa posição

end

end

if(rd == fi(1,1,8,0)) % Se rd = 1, então LSB\_10b fica com m3(j),

%sendo j, a posição achada anteriormente

LSB\_10b = m2(j);

end

if(rd == fi(-1,1,8,0)) % Se rd =-1, então LSB\_10b fica com m2(j),

%sendo j, a posição achada anteriormente

LSB\_10b = m3(j);

end

end

% ------------- SAÍDA --------------------------------

%A função bitshift(dado binário,4) corre 4 lugares para a esquerda.

%A função bitor realiza uma operação OR com o dado MSB\_10b e o dado

% LSB\_10b, dessa forma soma os dois dados para gerar "saida"(10 bits).

saida = bitor(bitshift(MSB\_10b,4),LSB\_10b);

%---------------- Cálculo do próximo RD ----------------

count1 = int8(0);

count2 = int8(0);

%Faz a contagem de 1's e 0's que tem em "saida"

%Quantidade de 1's fica com count1

%Quantidade de 0's fica com count2

for i=1:1:10

if (bitget(saida,i) == 1)

count1 = count1 + int8(1);

end

if (bitget(saida,i) == 0)

count2 = count2 + int8(1);

end

end

%Se RD anterior é -1 e a disparidade atual é positiva, RD(prox) = +1

if (rd == fi(-1,1,8,0)) && (count1 - count2 > int8(0))

rd = fi(1,1,8,0);

end

%Se RD anterior é -1 e a disparidade atual é positiva, RD(prox) = +1

if (rd == fi(1,1,8,0)) && (count1 - count2 < int8(0))

rd = fi(-1,1,8,0);

end

OUT\_10b = saida;

**ANEXO II**

Algorítimo desenvolvido para o subsistema De*coder 10b to 8b* para realizar a decodificação 8b10b, utilizando a ferramenta *Embedded MATLABTM Function Block*:

function [OUT\_8b,error\_aux] = decoding10b8b(IN\_10b)

% ------"DECOFICADOR 8B/10B - VICTOR AFONSO DOS REIS - FEIS UNESP"--------

%-----Entradas e saídas---------------------

%IN:[X9 X8 X7 X6 X5 X4 X3 X2 X1 X0]

%MSB\_10b\_in: X9 X8 X7 X6 X5 X4

%LSB\_8b\_in: X3 X2 X1 X0

%OUT: [Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0]

%MSB\_8B: Y7 Y6 Y5

%LSB\_8B: Y4 Y3 Y2 Y1 Y0

%------TRATAMENTO DA ENTRADA-------------------

%A função bitand realiza uma operação AND com o dado MSB\_10b

%com "0000 0011 1111 0000"

MSB\_10b = bitshift(bitand(IN\_10b, uint16(1008)), -4);

%A função bitshift(dado binário, -4) corre 4 lugares para a direita

% A função bitand realiza uma operação AND com o dado MSB\_10b

%com "0000 0000 0000 1111"

LSB\_10b = bitand(IN\_10b,uint16(15));

%------Inicialização das saídas---------------

MSB\_8b = uint16(0);

LSB\_8b = uint16(0);

error\_aux = uint16(0); % Saída de Erro na decodificação

count1 = uint8(0);

count2 = uint8(0);

%-----Inicialização das Saídas------------

%persistent: Cria um registro de memória no FPGA, ele é mantifo até ser

%modificado

persistent c1;

persistent c2;

persistent c3;

persistent m1;

persistent m2;

persistent m3;

persistent v1;

persistent k;

persistent k\_exit;

if isempty (c1) %Se c1 não está inicializado (Ou seja, quando começa a

%funcionar o FPGA)

%C1 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

c1 = [uint16(0); uint16(1);uint16(2);uint16(3);uint16(4);uint16(5);

uint16(6);uint16(7);uint16(8);uint16(9);uint16(10);uint16(11);

uint16(12);uint16(13);uint16(14);uint16(15);uint16(16);uint16(17);

uint16(18);uint16(19);uint16(20);uint16(21);uint16(22);uint16(23);

uint16(24);uint16(25);uint16(26);uint16(27);uint16(28);uint16(29);

uint16(30);uint16(31)];

end

if isempty (c2) %Se c2 não está inicializado (Ou seja, quando começa a

%funcionar o FPGA)

%C2 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

c2=[uint16(39); uint16(29);uint16(45);uint16(49);uint16(53);uint16(41);

uint16(25);uint16(56);uint16(57);uint16(37);uint16(21);uint16(52);

uint16(13);uint16(44);uint16(28);uint16(23);uint16(27);

uint16(35);uint16(19);uint16(50);uint16(11);uint16(42);uint16(26);

uint16(58);uint16(51);uint16(38);uint16(22);uint16(54);uint16(14);

uint16(46);uint16(30);uint16(43)];

end

if isempty (c3) %Se c3 não está inicializado (Ou seja, quando começa a

%funcionar o FPGA)

%C2 vetor coluna com valor em fixed point inteiro sem sinal 16 bits

c3=[uint16(24);uint16(34);uint16(18);uint16(49);uint16(10);uint16(41);

uint16(25);uint16(7);uint16(6);uint16(37);uint16(21);uint16(52);

uint16(13);uint16(44);uint16(28);uint16(40);uint16(36);uint16(35);

uint16(19);uint16(50);uint16(11);uint16(42);uint16(26);uint16(5);

uint16(12);uint16(38);uint16(22);uint16(9);uint16(14);uint16(17);

uint16(33);uint16(20)];

end

if isempty(m1) %Se m1 não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%m1 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

m1=[uint16(0);uint16(1);uint16(2);uint16(3);uint16(4);uint16(5);

uint16(6); uint16(7)];

end

if isempty(m2) %Se m2 não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%m2 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

m2=[uint16(4);uint16(9);uint16(5);uint16(3);uint16(2);uint16(10);

uint16(6);uint16(1)];

end

if isempty(m3) %Se m3 não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%m3 vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

m3=[uint16(11);uint16(9);uint16(5);uint16(12);uint16(13);uint16(10);

uint16(6); uint16(14)];

end

if isempty(v1) % Se v1 não está inicializado

% v1 vetor para verificação da codificação 3b/4b

v1 = [uint16(4);uint16(6);uint16(7);uint16(8);uint16(10);uint16(11);

uint16(12); uint16(13);uint16(14);uint16(15);uint16(18);uint16(19);

uint16(20);uint16(21);uint16(22); uint16(23);uint16(26);uint16(27);

uint16(29)];

end

if isempty (k) %Se k não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%k vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

k = [uint16(488); uint16(535)];

end

if isempty (k\_exit) %Se k\_exit não está inicializado(ou seja, quando começa a

%funcionar o FPGA)

%k\_exit vetor coluna com valor em fixed-point inteiro sem sinal 16 bits

k\_exit = uint16(254);

% Saída quando há erro na decodificação

end

%------------- Inicialização RD -----------------

persistent rd;

if isempty(rd) %Se rd não está inicializado (Ou seja, quando começar

%a funcionar o FPGA)

rd = fi(-1,1,8,0); %RD fixed-point inteiro com sinal 8 bits

end

%----------- Convert 6b/5b do LSB --------------------------

j=-1;

if (rd == fi(1,1,8,0))

for i = 1:1:32

if (MSB\_10b == c3(i))

j=i;

end

end

end

if (rd == fi(-1,1,8,0))

for i = 1:1:32

if (MSB\_10b == c2(i))

j=i;

end

end

end

if (j ~= -1 ) %Se o valor recebido estiver na tabela

LSB\_8b = c1(j);

end

% ------- Convert 3b/4b of MSB -----------------

flip = uint8(0);

if (j ~= -1) %Se o valor recebido estiver na tabela

for i=1:1:19

if(uint16(j) == v1(i))

flip = uint8(1); % Procura na matriz v1 se o dado 5b inserido

% inverte ou não a tabela. Caso o dado inverta

% o flip será 1 caso não será 0.

end

end

j = -1;

% Procura na matriz m1 a posição que coincida com o dado de entrada

if (flip == uint8(0)) % Verifica se inverte as tabelas de codificação

if(rd == fi(1,1,8,0)) % Se rd = 1, então MSB\_8b fica com m3(j),

%sendo j, a posição achada anteriormente

for i=1:1:8

if (LSB\_10b == m3(i))

j=i; % Coincide, armazena-se em j essa posição

end

end

end

if(rd == fi(-1,1,8,0)) % Se rd =-1, então LSB\_10b fica com m2(j),

%sendo j, a posição achada anteriormente

for i=1:1:8

if (LSB\_10b == m2(i))

j=i; % Coincide, armazena-se em j essa posição

end

end

end

else

if(rd == fi(1,1,8,0)) % Se rd = 1, então MSB\_8b fica com m3(j),

%sendo j, a posição achada anteriormente

for i=1:1:8

if (LSB\_10b == m2(i))

j=i; % Coincide, armazena-se em j essa posição

end

end

elseif (rd == fi(-1,1,8,0)) % Se rd =-1, então LSB\_10b fica com

% m2(j) sendo j, a posição achada anteriormente

for i=1:1:8

if (LSB\_10b == m3(i))

j=i; % Coincide, armazena-se em j essa posição

end

end

end

end

if (j~= -1)

MSB\_8b = m1(j);

end

end

% ---------------------- SAÍDA --------------------------------

if ( j ~= -1) %Se o valor inserido estiver nas tabelas

%A função bitshift(dado binário,5) corre 5 lugares para a esquerda.

%A função bitor realiza uma operação OR com o dado MSB\_8b e o dado

% LSB\_8b, dessa forma soma os dois dados para gerar OUT\_8b.

OUT\_8b = bitor(bitshift(MSB\_8b,5),LSB\_8b);

%Caso não há erro e os valore estiverem nas tabelas o erro\_aux é 0.

error\_aux = uint16(0);

else % Se o valor inserido não estiver na tabela.

OUT\_8b = uint16(256);

error\_aux = uint16(1);

end

%---------------- Cálculo do próximo RD ----------------

count1 = int8(0);

count2 = int8(0);

if (j ~= -1) %Só calcula o próximo RD se o valor estiver na tabela

%Faz a contagem de 1's e 0's que tem em OUT\_10b

%Quantidade de 1's fica com count1

%Quantidade de 0's fica com count2

for i=1:1:10

if (bitget(IN\_10b,i) == 1)

count1 = count1 + int8(1);

end

if (bitget(IN\_10b,i) == 0)

count2 = count2 + int8(1);

end

end

%Se RD anterior é -1 e a disparidade atual é positiva, RD(prox) = +1

if (rd == fi(-1,1,8,0)) && (count1 - count2 > int8(0))

rd = fi(1,1,8,0);

end

%Se RD anterior é +1 e a disparidade atual é positiva, RD(prox) = -1

if (rd == fi(1,1,8,0)) && (count1 - count2 < int8(0))

rd = fi(-1,1,8,0);

end

end

**ANEXO III**

Algorítimo desenvolvido para o sistema *encoder* descrito em VHDL no software VivadoTM da empresa Xilinx:

----------------------------------------------------------------------------------------

------------------------------- ENCODER 8B/10B - VHDL ----------------------------------

--------------------------- AUTOR: VICTOR AFONSO DOS REIS ------------------------------

-------------------------------- UNESP ILHA SOLTEIRA -----------------------------------

----------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Library UNISIM;

--use UNISIM.vcomponents.all;

--library UNIMACRO;

--use unimacro.Vcomponents.all;

ENTITY Encoder\_8b10b IS

PORT( clk,reset, send\_en : IN std\_logic;

IN\_8b : IN std\_logic\_vector(7 DOWNTO 0);

ready : OUT std\_logic;

OUT\_10b : OUT std\_logic\_vector(9 DOWNTO 0)

);

END Encoder\_8b10b;

ARCHITECTURE rtl OF Encoder\_8b10b IS

-- Constants

TYPE dados4bLsb IS ARRAY(0 TO 7) OF STD\_LOGIC\_VECTOR(7 DOWNTO 0);

TYPE dados3bMsb IS ARRAY(0 TO 7) OF STD\_LOGIC\_VECTOR(7 DOWNTO 0);

TYPE dados5BLsb IS ARRAY(0 TO 31) OF STD\_LOGIC\_VECTOR (7 DOWNTO 0);

TYPE dados6bMsb IS ARRAY(0 TO 31) OF STD\_LOGIC\_VECTOR (7 DOWNTO 0);

TYPE verificacao IS ARRAY(0 TO 18) OF INTEGER;

CONSTANT table\_lsb8b : dados5bLsb :=

(x"00", x"01", x"02", x"03", x"04", x"05", x"06", x"07",

x"08", x"09", x"0A", x"0B", x"0C", x"0D", x"0E", x"0F",

x"10", x"11", x"12", x"13", x"14", x"15", x"16", x"17",

x"18", x"19", x"1A", x"1B", x"1C", x"1D", x"1E", x"1F");

CONSTANT table1\_msb10b : dados6bMsb :=

( x"27", x"1D", x"2D", x"31", x"35", x"29", x"19", x"38",

x"39", x"25", x"15", x"34", x"0D", x"2C", x"1C", x"17",

x"1B", x"23", x"13", x"32", x"0B", x"2A", x"1A", x"3A",

x"33", x"26", x"16", x"36", x"0E", x"2E", x"1E", x"2B" );

CONSTANT table2\_msb10b :dados6bMsb :=

( x"18", x"22", x"12", x"31", x"0A", x"29", x"19", x"07",

x"06", x"25", x"15", x"34", x"0D", x"2C", x"1C", x"28",

x"24", x"23", x"13", x"32", x"0B", x"2A", x"1A", x"05",

x"0C", x"26", x"16", x"09", x"0E", x"11", x"21", x"14");

CONSTANT table\_msb8b :dados3bMsb :=

( x"00", x"20", x"40", x"60", x"80", x"A0", x"C0", x"E0");

CONSTANT table1\_lsb10b : dados4bLsb :=

( x"04", x"09", x"05", x"03", x"02", x"0A", x"06", x"01" );

CONSTANT table2\_lsb10b : dados4bLsb :=

( x"0B", x"09", x"05", x"0C", x"0D", x"0A", x"06", x"0E" );

CONSTANT verification : verificacao :=

( 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21,

22, 25, 26, 28);

-- Signals

SIGNAL rd : STD\_LOGIC\_VECTOR(1 DOWNTO 0); -- Runing Disparity of the sistem

SIGNAL lsb\_8b : STD\_LOGIC\_VECTOR(7 DOWNTO 0); -- Part of the input 8B

SIGNAL msb\_8b : STD\_LOGIC\_VECTOR(7 DOWNTO 0); -- Part of the input 8B

SIGNAL lsb\_10b : STD\_LOGIC\_VECTOR(3 DOWNTO 0); -- Part of the output 10B

SIGNAL msb\_10b : STD\_LOGIC\_VECTOR(5 DOWNTO 0); -- Part of the output 10B

SIGNAL aux : STD\_LOGIC\_VECTOR(9 DOWNTO 0); -- Signal auxiliar to read the output

SIGNAL flip\_vector : STD\_LOGIC; -- Signal to flip the tables

SIGNAL ready\_i : STD\_LOGIC; -- Signal to indicate the data is in the output

SIGNAL r\_d : INTEGER; -- Only show the diference of the bits 1's and 0's in the output

SIGNAL step : INTEGER; -- Help to count the new rd

SIGNAL count0, count1 : INTEGER := 0; -- Counters of bits 1's and 0's

-- States of the Finite States Machine

TYPE fsm IS (wait4en, msb0\_10b\_calc,lsb0\_10b\_calc, done); -- States of the finite States Machine

SIGNAL state : fsm;

-- Type to help the count of bits of the data output

TYPE bit\_check IS ARRAY (integer RANGE 9 DOWNTO 0) OF INTEGER;

SIGNAL count0\_vec, count1\_vec : bit\_check;

BEGIN

---------- Processing input Data -----------------

lsb\_8b <= "00011111" AND IN\_8b WHEN send\_en = '1';

msb\_8b <= "11100000" AND IN\_8b WHEN send\_en = '1';

--------------------------------------------------

----- The data is ready in the output ------------

ready <= '0' WHEN send\_en = '1' ELSE

'1' WHEN ready\_i = '1' ELSE

'0' WHEN reset = '1';

--------------------------------------------------

fsm\_8b\_10b : PROCESS(clk)

BEGIN

IF RISING\_EDGE(clk) THEN

IF (reset = '1') THEN

state <= wait4en;

rd <= "11";

lsb\_10b <= "0100";

msb\_10b <= "100111";

ready\_i <= '0';

count0\_vec <= (OTHERS => 0);

count1\_vec <= (OTHERS => 0);

OUT\_10b <= "1001110100";

ELSE

CASE state IS

------------ Receiving the new data in the input ---------

WHEN wait4en =>

IF send\_en = '1' THEN

state <= msb0\_10b\_calc;

ready\_i <= '0';

step <= 0;

ELSE

state <= wait4en;

ready\_i <= '1';

END IF;

----------------------------------------------------------

--- Processing the 5 bits LSB of the input data ----------

WHEN msb0\_10b\_calc =>

state <= lsb0\_10b\_calc;

FOR i IN 0 TO 31 LOOP

IF (lsb\_8b = table\_lsb8b(i)) THEN

IF (rd = "01") THEN --- (1)

msb\_10b <= table2\_msb10b(i)(5 DOWNTO 0);

ELSIF (rd = "11") THEN ---(-1)

msb\_10b <= table1\_msb10b(i)(5 DOWNTO 0);

END IF;

--- Verific if flip the tables of 4 bits LSB output data ----

FOR j IN 18 downto 0 LOOP

IF (i = verification(j)) THEN

flip\_vector <= '1';

exit;

ELSE

flip\_vector <= '0';

END IF;

END LOOP;

-------------------------------------------------------------

END IF;

END LOOP;

----------------------------------------------------------

---- Processing the 3 bits MSB of the input data ---------

WHEN lsb0\_10b\_calc =>

state <= done;

IF (flip\_vector = '0') THEN -- If the tables not flip

FOR i IN 0 TO 7 LOOP

IF (msb\_8b = table\_msb8b(i)) THEN

IF rd = "01" THEN -- rd = 1

lsb\_10b <= table2\_lsb10b(i)(3 DOWNTO 0);

ELSIF rd = "11" THEN -- rd = -1

lsb\_10b <= table1\_lsb10b(i)(3 DOWNTO 0);

END IF;

END IF;

END LOOP;

ELSIF (flip\_vector = '1') THEN -- If the tables flip

FOR i IN 0 TO 7 LOOP

IF (msb\_8b = table\_msb8b(i)) THEN

IF (rd = "01") THEN --(1)

lsb\_10b <= table1\_lsb10b(i)(3 DOWNTO 0);

ELSIF (rd = "11") THEN --(-1)

lsb\_10b <= table2\_lsb10b(i)(3 DOWNTO 0);

END IF;

END IF;

END LOOP;

END IF;

----------------------------------------------------------

------- Processing the new RD of the sistem --------------

WHEN done =>

------ Put the output in the signal auxiliar ---------

IF step = 0 THEN

aux(9 DOWNTO 4) <= msb\_10b; -- OUTPUT 5B/6B

aux(3 DOWNTO 0) <= lsb\_10b; -- OUTPUT 3B/4B

step <= 1;

state <= done;

------------------------------------------------------

------ Count the number os bits 1's and 0's in the output ------

ELSIF step = 1 THEN

FOR i IN 9 DOWNTO 0 LOOP

IF (aux(i) = '0') THEN

count0\_vec(i) <= 1;

count1\_vec(i) <= 0;

ELSIF (aux(i) = '1') THEN

count0\_vec(i) <= 0;

count1\_vec(i) <= 1;

END IF;

END LOOP;

step <= 2;

state <= done;

--------------------------------------------------------------

---- Put the number of bits 1's and 0's in the signals and sum -------

ELSIF step = 2 THEN

count0 <= count0\_vec(9) + count0\_vec(8) + count0\_vec(7) +

count0\_vec(6) + count0\_vec(5) + count0\_vec(4) +

count0\_vec(3) + count0\_vec(2) + count0\_vec(1) +

count0\_vec(0);

count1 <= count1\_vec(9) + count1\_vec(8) + count1\_vec(7) +

count1\_vec(6) + count1\_vec(5) + count1\_vec(4) +

count1\_vec(3) + count1\_vec(2) + count1\_vec(1) +

count1\_vec(0);

step <= 3;

state <= done;

---------------------------------------------------------------------

---- Verific the numbers of 1's and 0's and change or not the RD ----

ELSIF step = 3 THEN

r\_d <= (count1 - count0);

IF (rd = "01") AND (count1 < count0) THEN

rd <= "11";

ELSIF (rd = "11") AND (count1 > count0) THEN

rd <= "01";

END IF;

state <= wait4en;

OUT\_10b <= aux;

END IF;

------------------------------------------------------------------------------

end case;

end if;

end if;

end process;

END rtl;

***TestBench* do *encoder* descrito em VHDL:**

-------------------------------------------------------------------------------------------------

------------------------- TESTBENCH ENCODER 8B/10B - VHDL ------------------

--------------------------- AUTOR: VICTOR AFONSO DOS REIS -------------------

-------------------------------- UNESP ILHA SOLTEIRA -------------------------------

-------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Library UNISIM;

--use UNISIM.vcomponents.all;

--library UNIMACRO;

--use unimacro.Vcomponents.all;

entity tb\_encoder\_8b10b is

end tb\_encoder\_8b10b;

architecture Behavioral of tb\_encoder\_8b10b is

COMPONENT Encoder\_8b10b

PORT(clk,reset, send\_en : IN STD\_LOGIC;

IN\_8b : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ready : OUT STD\_LOGIC;

OUT\_10b : OUT STD\_LOGIC\_VECTOR(9 DOWNTO 0)

);

END COMPONENT;

---- Inputs ----

SIGNAL clk\_slow, clk\_fast : STD\_LOGIC := '0'; -- The clk\_slow is the clock of the user to insert data

SIGNAL reset, send\_en : STD\_LOGIC := '1'; -- The clk\_fast is the clock of the sistem

SIGNAL IN\_8b : STD\_LOGIC\_VECTOR(7 DOWNTO 0) := "00000000";

--- Outputs ----

SIGNAL ready : STD\_LOGIC := '1';

SIGNAL OUT\_10b : STD\_LOGIC\_VECTOR(9 DOWNTO 0);

----- Constants of the periods -----

CONSTANT clk\_slow\_period : TIME := 7 ns; -- The ratio is 7 periods to clk\_slow to 1 period to clk\_fast

CONSTANT clk\_fast\_period : TIME := 1 ns; -- The ratio have to be this ratio because of the FIFO

CONSTANT reset\_period : TIME := 50 ns;

begin

--Instância da Unit Under test (UUT)

uut: Encoder\_8b10b PORT MAP(

clk => clk\_fast,

reset => reset,

IN\_8b => IN\_8b,

ready => ready,

send\_en => send\_en,

OUT\_10b => OUT\_10b

);

-- Processo de estímulo

---------- Generation of de data input ------------

stp: PROCESS (clk\_slow)

BEGIN

IF rising\_edge(clk\_slow) THEN

IF reset = '1' THEN

IN\_8b <= "00000000";

send\_en <= '0';

ELSE

IF ready = '1' THEN -- If the last data is in the output --

In\_8b <= STD\_LOGIC\_VECTOR(UNSIGNED(IN\_8B) +1);

send\_en <= '1'; -- Reset the output ready

ELSE

send\_en <= '0';

END IF;

END IF;

END IF;

END PROCESS;

----------------------------------------------------

------------- Generation of clk\_slow ---------------

clk\_slow\_gen: PROCESS

BEGIN

clk\_slow <= '0';

WAIT FOR clk\_slow\_period/2;

clk\_slow <= '1';

WAIT FOR clk\_slow\_period/2;

END PROCESS;

----------------------------------------------------

------------- Generation of clk\_fast ---------------

clk\_fast\_gen: PROCESS

BEGIN

clk\_fast <= '0';

WAIT FOR clk\_fast\_period/2;

clk\_fast <= '1';

WAIT FOR clk\_fast\_period/2;

END PROCESS;

----------------------------------------------------

------------- Generation of reset ------------------

stp2: PROCESS

BEGIN

WAIT FOR reset\_period;

reset <= '0';

END PROCESS;

----------------------------------------------------

END Behavioral;

**ANEXO IV**

Algorítimo desenvolvido para o sistema do de*coder* descrito em VHDL no software VivadoTM da empresa Xilinx:

----------------------------------------------------------------------------------------

------------------------------- DECODER 8B/10B - VHDL ----------------------------------

--------------------------- AUTOR: VICTOR AFONSO DOS REIS ------------------------------

-------------------------------- UNESP ILHA SOLTEIRA -----------------------------------

----------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use IEEE.NUMERIC\_STD.ALL;

-- Unco mment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity decoder\_8b10b is

Port ( clk,reset,receive\_en : IN std\_logic;

IN\_10b : IN std\_logic\_vector(9 DOWNTO 0);

ready : OUT std\_logic;

OUT\_8b : OUT std\_logic\_vector(7 DOWNTO 0)

);

end decoder\_8b10b;

architecture Behavioral of decoder\_8b10b is

-- Constants

TYPE dados4bLsb IS ARRAY(0 TO 7) OF STD\_LOGIC\_VECTOR(7 DOWNTO 0);

TYPE dados3bMsb IS ARRAY(0 TO 7) OF STD\_LOGIC\_VECTOR(7 DOWNTO 0);

TYPE dados5BLsb IS ARRAY(0 TO 31) OF STD\_LOGIC\_VECTOR (7 DOWNTO 0);

TYPE dados6bMsb IS ARRAY(0 TO 31) OF STD\_LOGIC\_VECTOR (7 DOWNTO 0);

TYPE verificacao IS ARRAY(0 TO 18) OF INTEGER;

CONSTANT table\_lsb8b : dados5bLsb :=

(x"00", x"01", x"02", x"03", x"04", x"05", x"06", x"07",

x"08", x"09", x"0A", x"0B", x"0C", x"0D", x"0E", x"0F",

x"10", x"11", x"12", x"13", x"14", x"15", x"16", x"17",

x"18", x"19", x"1A", x"1B", x"1C", x"1D", x"1E", x"1F");

CONSTANT table1\_msb10b : dados6bMsb :=

( x"27", x"1D", x"2D", x"31", x"35", x"29", x"19", x"38",

x"39", x"25", x"15", x"34", x"0D", x"2C", x"1C", x"17",

x"1B", x"23", x"13", x"32", x"0B", x"2A", x"1A", x"3A",

x"33", x"26", x"16", x"36", x"0E", x"2E", x"1E", x"2B" );

CONSTANT table2\_msb10b :dados6bMsb :=

( x"18", x"22", x"12", x"31", x"0A", x"29", x"19", x"07",

x"06", x"25", x"15", x"34", x"0D", x"2C", x"1C", x"28",

x"24", x"23", x"13", x"32", x"0B", x"2A", x"1A", x"05",

x"0C", x"26", x"16", x"09", x"0E", x"11", x"21", x"14");

CONSTANT table\_msb8b :dados3bMsb :=

( x"00", x"20", x"40", x"60", x"80", x"A0", x"C0", x"E0");

CONSTANT table1\_lsb10b : dados4bLsb :=

( x"04", x"09", x"05", x"03", x"02", x"0A", x"06", x"01" );

CONSTANT table2\_lsb10b : dados4bLsb :=

( x"0B", x"09", x"05", x"0C", x"0D", x"0A", x"06", x"0E" );

CONSTANT verification : verificacao :=

( 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21,

22, 25, 26, 28);

-- Signals

SIGNAL rd : STD\_LOGIC\_VECTOR(1 DOWNTO 0); -- Runing Disparity of the sistem

SIGNAL lsb\_8b : STD\_LOGIC\_VECTOR(4 DOWNTO 0); -- Part of the output 8B

SIGNAL msb\_8b : STD\_LOGIC\_VECTOR(2 DOWNTO 0); -- Part of the output 8B

SIGNAL lsb\_10b : STD\_LOGIC\_VECTOR(9 DOWNTO 0); -- Part of the input 10B

SIGNAL msb\_10b : STD\_LOGIC\_VECTOR(9 DOWNTO 0); -- Part of the input 10B

SIGNAL aux : STD\_LOGIC\_VECTOR(7 DOWNTO 0); -- Signal auxiliar to read the output

SIGNAL flip\_vector : STD\_LOGIC; -- Signal to flip the tables

SIGNAL ready\_i : STD\_LOGIC; -- Signal to indicate the data is in the output

SIGNAL r\_d : INTEGER; -- Only show the diference of the bits 1's and 0's in the output

SIGNAL step : INTEGER; -- Help to count the new rd

SIGNAL count0, count1 : INTEGER := 0; -- Counters of bits 1's and 0's

-- States of the Finite States Machine

TYPE fsm IS (waiten, msb\_8b\_calc,lsb\_8b\_calc, done); -- States of the finite States Machine

SIGNAL state : fsm;

-- Type to help the count of bits of the data output

TYPE bit\_check IS ARRAY (integer RANGE 9 DOWNTO 0) OF INTEGER;

SIGNAL count0\_vec, count1\_vec : bit\_check;

begin

----- Processing data Input --------

lsb\_10b <= "0000001111" AND IN\_10b WHEN ready\_i = '1';

msb\_10b <= STD\_LOGIC\_VECTOR(UNSIGNED("1111110000" AND IN\_10b) ROL 6) WHEN ready\_i = '1';

------ The data is ready in the output ----

ready <= '0' WHEN receive\_en = '1' ELSE

'1' WHEN ready\_i = '1' ELSE

'0' WHEN reset = '1';

-------------------------------------------

fsm\_10b\_8b : PROCESS(clk)

BEGIN

IF RISING\_EDGE(clk) THEN

IF (reset = '1') THEN

lsb\_8b <= "00000";

msb\_8b <= "000";

state <= waiten;

rd <= "11";

ready\_i <= '0';

count0\_vec <= (OTHERS => 0);

count1\_vec <= (OTHERS => 0);

OUT\_8b <= "00000000";

ELSE

CASE state IS

------------ Receiving the new data 10b in the input ---------

WHEN waiten =>

IF receive\_en = '1' THEN

state <= lsb\_8b\_calc;

step <= 0;

ready\_i <= '0'; -- The data isn't avaible in the output

ELSE

state <= waiten;

ready\_i <= '1'; -- If the data is in the output

END IF;

----------------------------------------------------------- --- Processing the 6 bits MSB of the input 10b data ----------

WHEN lsb\_8b\_calc =>

state <= msb\_8b\_calc;

FOR i IN 0 TO 31 LOOP

IF (rd = "01") THEN -- rd = (1) --- Verific wich position of the input 6b MSB data is in the table

IF (msb\_10b = "00" & table2\_msb10b(i)) THEN

lsb\_8b <= table\_lsb8b(i) (4 DOWNTO 0);

------ Verific if flip tables of 3 bits LSB output data ------

FOR j IN 18 DOWNTO 0 LOOP

IF (i = verification(j)) THEN

flip\_vector <= '1';

EXIT;

ELSE

flip\_vector <= '0';

END IF;

END LOOP;

END IF;

ELSIF (rd = "11") THEN -- rd = (-1)

--- Verific wich position of the input 6b MSB data is in the table

IF (msb\_10b = "00" & table1\_msb10b(i)) THEN

lsb\_8b <= table\_lsb8b(i)(4 DOWNTO 0);

----- Verific if flip tables of 3 bits LSB output data ------

FOR j IN 18 DOWNTO 0 LOOP

IF (i = verification(j)) THEN

flip\_vector <= '1';

EXIT;

ELSE

flip\_vector <= '0';

END IF;

END LOOP;

END IF;

END IF;

-------------------------------------------------------------

END LOOP;

----------------------------------------------------------------

-------- Processing the 4 bits LSB of the input data -----------

WHEN msb\_8b\_calc =>

state <= done;

IF (flip\_vector = '0') THEN -- If the tables not flip

FOR i IN 0 TO 7 LOOP

IF (rd = "01") THEN -- rd = 1

IF (lsb\_10b = "00" & table2\_lsb10b(i)) THEN --- Verific wich position of the input 4b LSB data is in the table

msb\_8b <= table\_msb8b(i)(7 DOWNTO 5);

END IF;

ELSIF (rd = "11") THEN -- rd = -1

--- Verific wich position of the input 4b LSB data is in the table

IF (lsb\_10b = "00" & table1\_lsb10b(i)) THEN

msb\_8b <= table\_msb8b(i)(7 DOWNTO 5);

END IF;

END IF;

END LOOP;

ELSIF (flip\_vector = '1') THEN -- If the tables flip

FOR i IN 0 TO 7 LOOP

IF (rd = "01") THEN -- rd = (1)

--- Verific wich position of the input 4b LSB data is in the table

IF (lsb\_10b = "00" & table1\_lsb10b(i)) THEN

msb\_8b <= table\_msb8b(i)(7 DOWNTO 5);

END IF;

ELSIF (rd = "11") THEN -- rd = (-1)

--- Verific wich position of the input 4b LSB data is in the table

IF (lsb\_10b = "00" & table2\_lsb10b(i)) THEN

msb\_8b <= table\_msb8b(i)(7 DOWNTO 5);

END IF;

END IF;

END LOOP;

END IF;

--------------------------------------------------------------

-------- Processing the new RD of the sistem -----------------

WHEN done =>

------ Put the output in the auxiliar signal ---------

IF step = 0 THEN

aux(7 DOWNTO 5) <= msb\_8b; -- OUTPUT 3B

aux(4 DOWNTO 0) <= lsb\_8b; -- OUTPUT 5B

step <= 1;

state <= done;

------------------------------------------------------

------ Count the number os bits 1's and 0's in the input ------

ELSIF step = 1 THEN

FOR i IN 9 DOWNTO 0 LOOP

IF (IN\_10b(i) = '0') THEN

count0\_vec(i) <= 1;

count1\_vec(i) <= 0;

ELSIF (IN\_10b(i) = '1') THEN

count0\_vec(i) <= 0;

count1\_vec(i) <= 1;

END IF;

END LOOP;

step <= 2;

state <= done;

--------------------------------------------------------------

---- Put the number of bits 1's and 0's in the signals and sum -------

ELSIF step = 2 THEN

count0 <= count0\_vec(9) + count0\_vec(8) + count0\_vec(7) +

count0\_vec(6) + count0\_vec(5) + count0\_vec(4) +

count0\_vec(3) + count0\_vec(2) + count0\_vec(1) +

count0\_vec(0);

count1 <= count1\_vec(9) + count1\_vec(8) + count1\_vec(7) +

count1\_vec(6) + count1\_vec(5) + count1\_vec(4) +

count1\_vec(3) + count1\_vec(2) + count1\_vec(1) +

count1\_vec(0);

step <= 3;

state <= done;

---------------------------------------------------------------------

---- Verific the numbers of 1's and 0's and change or not the RD ----

ELSIF step = 3 THEN

r\_d <= (count1 - count0);

IF (rd = "01") AND (count1 < count0) THEN

rd <= "11";

ELSIF (rd = "11") AND (count1 > count0) THEN

rd <= "01";

END IF;

state <= waiten;

OUT\_8b <= aux;

END IF;

------------------------------------------------------------------------------

END CASE;

END IF;

END IF;

END PROCESS fsm\_10b\_8b;

end Behavioral;

**TestBench do decoder descrito em VHDL:**

----------------------------------------------------------------------------------------------

------------------------- TESTBENCH DECODER 8B/10B - VHDL ----------------

--------------------------- AUTOR: VICTOR AFONSO DOS REIS ------------------

-------------------------------- UNESP ILHA SOLTEIRA -------------------------------

------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Library UNISIM;

--use UNISIM.vcomponents.all;

--library UNIMACRO;

--use unimacro.Vcomponents.all;

entity tb\_decoder\_8b10b is

end tb\_decoder\_8b10b;

architecture Behavioral of tb\_decoder\_8b10b is

COMPONENT decoder\_8b10b

PORT(clk,reset,receive\_en : IN std\_logic;

IN\_10b : IN std\_logic\_vector(9 DOWNTO 0);

ready : OUT std\_logic;

OUT\_8b : OUT std\_logic\_vector(7 DOWNTO 0)

);

END COMPONENT;

---- Inputs ----

SIGNAL clk\_slow, clk\_fast : STD\_LOGIC := '0'; -- The clk\_slow is the clock of the user to insert data

SIGNAL reset, receive\_en : STD\_LOGIC := '1'; -- The clk\_fast is the clock of the sistem

SIGNAL IN\_10b : STD\_LOGIC\_VECTOR(9 DOWNTO 0) := "0000000000";

--- Outputs ----

SIGNAL ready : STD\_LOGIC := '1';

SIGNAL OUT\_8b : STD\_LOGIC\_VECTOR(7 DOWNTO 0);

----- Constants of the periods -----

CONSTANT clk\_slow\_period : TIME := 7 ns;-- The ratio is 7 periods to clk\_slow to 1 period to clk\_fast

CONSTANT clk\_fast\_period : TIME := 1 ns;--The ratio have to be this ratio because of the FIFO

CONSTANT reset\_period : TIME := 50 ns;

--- Signals

SIGNAL position : INTEGER := 0;

SIGNAL rdtb : STD\_LOGIC\_VECTOR(1 DOWNTO 0); -- Runing Disparity of the testbench

-- Constants

TYPE dataRD IS ARRAY(0 TO 255) OF STD\_LOGIC\_VECTOR(11 DOWNTO 0);

TYPE dataRDchange IS ARRAY(0 TO 255) OF STD\_LOGIC;

CONSTANT tableRDminus : dataRD := -- Insert data for RD (-)

(x"274", x"1D4", x"2D4", x"31B", x"354", x"29B", x"19B", x"38B",

x"394", x"25B", x"15B", x"34B", x"0DB", x"2CB", x"1CB", x"174",

x"1B4", x"23B", x"13B", x"32B", x"0BB", x"2AB", x"1AB", x"3A4",

x"334", x"26B", x"16B", x"364", x"0EB", x"2E4", x"1E4", x"2B4",

x"279", x"1D9", x"2D9", x"319", x"359", x"299", x"199", x"389",

x"399", x"259", x"159", x"349", x"0D9", x"2C9", x"1C9", x"179",

x"1B9", x"239", x"139", x"329", x"0B9", x"2A9", x"1A9", x"3A9",

x"339", x"269", x"169", x"369", x"0E9", x"2E9", x"1E9", x"2B9",

x"275", x"1D5", x"2D5", x"315", x"355", x"295", x"195", x"385",

x"395", x"255", x"155", x"345", x"0D5", x"2C5", x"1C5", x"175",

x"1B5", x"235", x"135", x"325", x"0B5", x"2A5", x"1A5", x"3A5",

x"335", x"265", x"165", x"365", x"0E5", x"2E5", x"1E5", x"2B5",

x"273", x"1D3", x"2D3", x"31C", x"353", x"29C", x"19C", x"38C",

x"393", x"25C", x"15C", x"34C", x"0DC", x"2CC", x"1CC", x"173",

x"1B3", x"23C", x"13C", x"32C", x"0BC", x"2AC", x"1AC", x"3A3",

x"333", x"26C", x"16C", x"363", x"0EC", x"2E3", x"1E3", x"2B3",

x"272", x"1D2", x"2D2", x"31D", x"352", x"29D", x"19D", x"38D",

x"392", x"25D", x"15D", x"34D", x"0DD", x"2CD", x"1CD", x"172",

x"1B2", x"23D", x"13D", x"32D", x"0BD", x"2AD", x"1AD", x"3A2",

x"332", x"26D", x"16D", x"362", x"0ED", x"2E2", x"1E2", x"2B2",

x"27A", x"1DA", x"2DA", x"31A", x"35A", x"29A", x"19A", x"38A",

x"39A", x"25A", x"15A", x"34A", x"0DA", x"2CA", x"1CA", x"17A",

x"1BA", x"23A", x"13A", x"32A", x"0BA", x"2AA", x"1AA", x"3AA",

x"33A", x"26A", x"16A", x"36A", x"0EA", x"2EA", x"1EA", x"2BA",

x"276", x"1D6", x"2D6", x"316", x"356", x"296", x"196", x"386",

x"396", x"256", x"156", x"346", x"0D6", x"2C6", x"1C6", x"176",

x"1B6", x"236", x"136", x"326", x"0B6", x"2A6", x"1A6", x"3A6",

x"336", x"266", x"166", x"366", x"0E6", x"2E6", x"1E6", x"2B6",

x"171", x"1D1", x"2D1", x"31E", x"351", x"29E", x"19E", x"38E",

x"391", x"25E", x"15E", x"34E", x"0DE", x"2CE", x"1CE", x"171",

x"1B1", x"237", x"137", x"32E", x"0B7", x"2AE", x"1AE", x"3A1",

x"331", x"26E", x"16E", x"361", x"0EE", x"2E1", x"1E1", x"2B1");

CONSTANT tableRDplus : dataRD := -- Insert data for RD (+)

(x"18B", x"22B", x"12B", x"314", x"0AB", x"294", x"194", x"074",

x"06B",x"254",x"154",x"344",x"0D4",x"2C4",x"1C4",x"28B",

x"24B",x"234",x"134",x"324",x"0B4",x"2A4",x"1A4",x"05B",

x"0CB",x"264",x"164",x"09B",x"0E4",x"11B",x"21B",x"14B",

x"189",x"229",x"129",x"319",x"0A9",x"299",x"199",x"079",

x"069",x"259",x"159",x"349",x"0D9",x"2C9",x"1C9",x"289",

x"249",x"239",x"139”,x"329",x"0B9",x"2A9",x"1A9",x"059",

x"0C9",x"269",x"169",x"099”,x"0E9”,x"119",x"219",x"149",

x"185",x"225",x"125",x"315",x"0A5",x"295",x"195",x"075",

x"065",x"255",x"155",x"345",x"0D5",x"2C5",x"1C5",x"285",

x"245",x"235",x"135",x"325",x"0B5",x"2A5",x"1A5",x"055",

x"0C5",x"265",x"165",x"095",x"0E5",x"115",x"215",x"145",

x"18C",x"22C",x"12C",x"313",x"0AC",x"293",x"193",x"073",

x"06C",x"253",x"153",x"343",x"0D3",x"2C3",x"1C3",x"28C",

x"24C",x"233",x"133",x"323",x"0B3",x"2A3",x"1A3",x"05C",

x"0CC",x"263",x"163",x"09C",x"0E3",x"11C",x"21C",x"14C",

x"18D",x"22D",x"12D",x"312",x"0AD",x"292",x"192",x"072",

x"06D",x"252",x"152",x"342",x"0D2",x"2C2",x"1C2",x"28D",

x"24D",x"232",x"132",x"322",x"0B2",x"2A2",x"1A2",x"05D",

x"0CD",x"262",x"162",x"09D",x"0E2",x"11D",x"21D",x"14D",

x"18A",x"22A",x"12A",x"31A",x"0AA",x"29A",x"19A",x"07A",

x"06A",x"25A",x"15A",x"34A",x"0DA",x"2CA",x"1CA", x"28A",

x"24A",x"23A",x"13A",x"32A",x"0BA",x"2AA",x"1AA",x"0AA",

x"0CA",x"26A",x"16A",x"09A",x"0EA",x"11A",x"21A", x"14A",

x"186",x"226",x"126",x"316",x"0A6",x"296",x"196", x"076",

x"066",x"256",x"156",x"346",x"0D6",x"2C6",x"1C6", x"286",

x"246",x"236",x"136",x"326",x"0B6",x"2A6",x"1A6", x"056",

x"0C6",x"266",x"166", x"096",x"0E6",x"116",x"216", x"146",

x"18E",x"22E",x"12E",x"311",x"0AE",x"291",x"191",x"071",

x"06E",x"251",x"151",x"348",x"0D1",x"2C8",x"1C8",x"28E",

x"24E",x"231",x"131",x"321",x"0B1",x"2A1",x"1A1",x"05E",

x"0CE",x"261",x"161",x"09E",x"0E1",x"11E",x"21E",x"14E");

CONSTANT tableRDnew : dataRDchange := -- '1' if the rd change or '0' if the rd not change

('0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '1', '1', '1', '1', '0',

'0', '1', '1', '1', '1', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '0',

'1', '1', '1', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',

'1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '1', '1',

'1', '1', '1', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',

'1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '1', '1',

'1', '1', '1', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',

'1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '1', '1',

'0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '1', '1', '1', '1', '0',

'0', '1', '1', '1', '1', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '0',

'1', '1', '1', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',

'1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '1', '1',

'1', '1', '1', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',

'1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '1', '1',

'0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '1', '1', '1', '1', '0',

'0', '1', '1', '1', '1', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '0');

begin

--Instância da Unit Under test (UUT)

uut: decoder\_8b10b PORT MAP(

clk => clk\_fast,

reset => reset,

IN\_10b => IN\_10b,

ready => ready,

receive\_en => receive\_en,

OUT\_8b => OUT\_8b

);

-- Processo de estímulo

---------- Generation of de data input ------------

stp: PROCESS (clk\_slow)

BEGIN

IF rising\_edge(clk\_slow) THEN

IF reset = '1' THEN

In\_10b <= tableRDminus(0)(9 DOWNTO 0);

receive\_en <= '0';

position <= 0;

rdtb <= "11";

ELSE

IF ready = '1' THEN -- If the last data is in the output --

receive\_en <= '1'; -- Reset the output ready

------- Inserting the 10b data in the input ------------

IF rdtb = "11" THEN -- If the RD is negative

In\_10b <= tableRDminus(position)(9 DOWNTO 0);

IF tableRDnew(position) = '0' THEN -- If not change RD

rdtb <= "11";

ELSE -- If change RD

rdtb <= "01";

END IF;

ELSIF rdtb = "01" THEN -- If the RD is positive

In\_10b <= tableRDplus(position)(9 DOWNTO 0);

IF tableRDnew(position) = '0' THEN -- If not change RD

rdtb <= "01";

ELSE -- If change RD

rdtb <= "11";

END IF;

END IF;

---------------------------------------------------------

----------- Reseting the counter position ---------------

IF position < 255 THEN

position <= position + 1;

ELSE

position <= 0;

END IF;

---------------------------------------------------------

ELSE -- If the data isn't in the output

receive\_en <= '0';

END IF;

END IF;

END IF;

END PROCESS;

----------------------------------------------------

------------- Generation of clk\_slow ---------------

clk\_slow\_gen: PROCESS

BEGIN

clk\_slow <= '0';

WAIT FOR clk\_slow\_period/2;

clk\_slow <= '1';

WAIT FOR clk\_slow\_period/2;

END PROCESS;

----------------------------------------------------

------------- Generation of clk\_fast ---------------

clk\_fast\_gen: PROCESS

BEGIN

clk\_fast <= '0';

WAIT FOR clk\_fast\_period/2;

clk\_fast <= '1';

WAIT FOR clk\_fast\_period/2;

END PROCESS;

----------------------------------------------------

------------- Generation of reset ------------------

stp2: PROCESS

BEGIN

WAIT FOR reset\_period;

reset <= '0';

END PROCESS;

----------------------------------------------------

END Behavioral;

**ANEXO V**

Algorítimos descritos em VHDL das resoluções de alguns exercícios do livro do D’Amore (2012).

**Capítulo 2:**

**Ex 2.7.1:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity teste is

Port (a, b,c,d: in bit;

s1,s2,s3,s4: out bit

);

end teste;

architecture Behavioral of teste is

begin

s1 <= a or not b;

s2 <= a or (not b and c) or d;

s3 <= (a or not b) and ( c or d);

s4 <= (a or not b) and not(c or (a and d));

end Behavioral;

**Ex 2.7.3:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity teste is

Port ( a , b : in bit\_vector(3 downto 0);

c ; out bit\_vector(7 downto 0)

);

end teste;

architecture Behavioral of teste is

begin

c(7 downto 4) <= a(3 downto 0);

c(3 downto 0) <= b(3 downto 0);

end Behavioral;

**Ex 2.7.5:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity teste is

Port (a\_bit, b\_bit : in bit\_vector(2 DOWNTO 0);

not\_bit, and\_bit, or\_bit : out BIT\_VECTOR(2 downto 0)

);

end teste;

architecture Behavioral of teste is

type vetor\_booleano is array (0 to 2) of boolean;

signal not\_bool, and\_bool, or\_bool : vetor\_booleano;

constant a\_bool : vetor\_booleano := (true, false, true);

constant b\_bool : vetor\_booleano := (false, true, false);

begin

not\_bit <= not a\_bit;

and\_bit <= a\_bit and b\_bit;

or\_bit <= a\_bit or b\_bit;

not\_bool <= not a\_bool;

and\_bool <= a\_bool and b\_bool;

or\_bool <= a\_bool or b\_bool;

end Behavioral;

**Ex 2.7.6:**

**library IEEE;**

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity test is

Port (a, b ,c , d : in BIT;

s : out BIT\_VECTOR (5 downto 0) );

end test;

architecture Behavioral of test is

begin

s(0) <= (a AND b) OR (c AND d);

s(1) <= (a NOR b) NOR c;

s(2) <= (a AND b) OR c;

s(3) <= (NOT (a AND b)) NAND c;

s(4) <= (a XOR b) OR c;

end Behavioral;

**Ex 2.7.8:**

**library IEEE;**

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity errad\_2 is

Port (a ,b ,c ,d : IN BIT;

r , s, t, u, v, x : OUT BIT\_VECTOR(0 to 5) );

end errad\_2;

architecture Behavioral of errad\_2 is

begin

r(0 to 2) <= a & b & c;

s <= a & b & c & "010";

t(0 to 2) <= "101";

u(3 to 5) <= "101";

v <= (a ,'0','0','0','0', '0');

x <= (a ,'0','0','0','0', '1');

end Behavioral;

**Capítulo 3 :**

**Ex 3.10.1:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ex\_3\_10 is

Port (a ,b : IN BIT;

x : OUT BIT);

end ex\_3\_10;

architecture teste of ex\_3\_10 is

begin

x <= b XOR a;

b <= not a;

end teste;

**Ex 3.10.1b:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ex\_3\_10b is

Port (a,b,c : IN BIT;

x : OUT BIT);

end ex\_3\_10b;

architecture teste of ex\_3\_10b is

begin

x <= c XOR a;

c <= a AND b;

b <= NOT x;

end teste;

**Ex 3.10.2:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ex\_3\_10\_2 is

Port (d : IN BIT\_VECTOR (3 downto 0) ;

en : IN BIT ;

q : OUT BIT\_VECTOR(3 downto 0));

end ex\_3\_10\_2;

architecture Behavioral of ex\_3\_10\_2 is

begin

q <= d when en = '1' else

q;

end Behavioral;

**Ex 3.10.3:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ex\_3\_10\_3 is

Port (d : IN BIT\_VECTOR(3 downto 0);

en : IN BIT;

q : IN BIT\_VECTOR(3 downto 0));

end ex\_3\_10\_3;

architecture Behavioral of ex\_3\_10\_3 is

begin

with en select

q <= d when '1',

q when '0';

end Behavioral;

**Ex 3.10.4:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_3104 is

Port (p : IN BIT\_VECTOR(0 to 3);

x : OUT BIT\_VECTOR(0 to 1);

int : OUT BIT);

end Ex\_3104;

architecture Behavioral of Ex\_3104 is

begin

comandif : process (p,int)

begin

if p = "0000" then

int <= 0;

else

int <= 1;

end if;

end process comandif;

x <= "00" when p(0) = '1' else

"01" when p(1) = '1' else

"10" when p(2) = '1' else

"11" when p(3) = '1' else

"11" when int = '0';

end Behavioral;

**------------------------------------------------------------------------------------------------------------------------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_3104 is

Port (p : IN BIT\_VECTOR(0 to 3);

x : OUT BIT\_VECTOR(0 to 1);

int : OUT BIT);

end Ex\_3104;

architecture Behavioral of Ex\_3104 is

begin

comandif : process (p,int)

begin

if p = "0000" then

int <= 0;

else

int <= 1;

end then;

end process comandif;

x <= "00" when p(0) = '1' else

"01" when p(1) = '1' else

"10" when p(2) = '1' else

"11" when p(3) = '1' else

"11" when int = '0';

end Behavioral;

**------------------------------------------------------------------------------------------------------------------------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Ex\_3\_10\_4 is

Port (p : IN BIT\_VECTOR(0 to 3);

x : OUT BIT\_VECTOR(0 to 1);

int : OUT BIT);

end Ex\_3\_10\_4;

architecture Behavioral of Ex\_3\_10\_4 is

begin

comandif : process (p)

begin

if (p ="0000") then

int <= 0;

else

int <= 1;

end then;

end process comandif;

x <= "00" when p(0) = '1' else

"01" when p(1) = '1' else

"10" when p(2) = '1' else

"11" when p(3) = '1' else

"11" when int = '0';

end Behavioral;

**Ex 3.10.6:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ex\_3\_10\_6 is

Port (a : IN BIT\_VECTOR (17 downto 0);

cs : OUT BIT\_VECTOR (3 downto 0));

end ex\_3\_10\_6;

architecture Behavioral of ex\_3\_10\_6 is

signal global : BIT\_VECTOR (2 downto 0);

begin

global(2 downto 0) <= a (17 downto 15);

cs <= "011" when global <= "011" else

"101" when global = "100" else

"110" when global = "101" else

unaffected;

end Behavioral;

**Ex 3.10.5:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ex\_3\_10\_5 is

Port (p : IN BIT\_VECTOR(0 to 3);

x : OUT BIT\_VECTOR(1 downto 0);

int : OUT BIT);

end ex\_3\_10\_5;

architecture Behavioral of ex\_3\_10\_5 is

begin

int <= '1';

with p select

x <= "00" when "1000"|"1001"|"1010"|"1011"|"1100"|"1101"|"1110"|"1111",

"01" when "0100"|"0101"|"0110"|"0111",

"10" when "0010"|"0011",

"11" when "0001";

int <= "0" when p = "0000";

x <= "11" when int = "0";

end Behavioral;

**Ex 3.10.7:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ex\_3\_10\_7 is

Port (a : IN BIT\_VECTOR (17 downto 0);

cs : OUT BIT\_VECTOR (3 downto 0));

end ex\_3\_10\_7;

architecture Behavioral of ex\_3\_10\_7 is

signal global : BIT\_VECTOR (2 downto 0);

begin

global(2 downto 0) <= a (17 downto 15);

with global select

cs <= "011" when "011"|"000"|"001"|"010",

"101" when "100",

"110" when "101",

unaffected when others ;

end Behavioral;

**Ex 3.10.8:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ex\_3\_10\_8 is

Port (a : IN BIT\_VECTOR(0 to 3);

o : OUT BIT\_VECTOR(0 to 6));

end ex\_3\_10\_8;

architecture Behavioral of ex\_3\_10\_8 is

begin

with a select

o <= "1111110" when "0000",

"0110000" when "0001",

"1101101" when "0010",

"1111001" when "0011",

"0110011" when "0100",

"1011011" when "0101",

"1011111" when "0110",

"1110000" when "0111",

"1111111" when "1000",

"1111011" when "1001",

unaffected when others;

end Behavioral;

**Ex 3.10.9:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ex\_3\_10\_9 is

Port (bcd : IN BIT\_VECTOR(3 downto 0);

seg : OUT BIT\_VECTOR(0 to 6);

sel : IN BIT\_VECTOR(1 downto 0);

mst : OUT BIT\_VECTOR(0 to 3));

end ex\_3\_10\_9;

architecture Behavioral of ex\_3\_10\_9 is

begin

decodificador:BLOCK

with bcd select

seg <= "1111110" when "0000",

"0110000" when "0001",

"1101101" when "0010",

"1111001" when "0011",

"0110011" when "0100",

"1011011" when "0101",

"1011111" when "0110",

"1110000" when "0111",

"1111111" when "1000",

"1111011" when "1001",

unaffected when others;

END BLOCK decodificador;

selecao: BLOCK

with sel select

mst <= "0111" when "00",

"1011" when "01",

"1101" when "10",

"1110" when "11";

END BLOCK selecao;

end Behavioral;

**Capítulo 4**

**Ex 4.9.1:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_1 is

Port (sa : IN BIT;

sb, sc : BUFFER BIT);

end Ex\_4\_9\_1;

architecture Behavioral of Ex\_4\_9\_1 is

begin

PROCESS (sa,sb)

BEGIN

sc <= sb;

sb <= sa;

END PROCESS;

end Behavioral;

-- Neste código percebe-se que a lista de sensibilidade do processo criado está de acordo pois

-- gerou resultados esperados, aos quais representa o circuito descrito.

**Testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity test is

end test;

architecture Behavioral of test is

COMPONENT Ex\_4\_9\_1

PORT(sa : IN BIT;

sb, sc : BUFFER BIT);

END COMPONENT;

signal sa : BIT := '0';

signal sb : BIT ;

signal sc : BIT ;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_1 PORT MAP(

sa => sa,

sb => sb,

sc => sc);

-- Processo de estímulo

stp: PROCESS

BEGIN

sa <= NOT sa;

WAIT FOR 50 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.2:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_SIGNED.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Ex\_4\_9\_2 is

Port (a, b : IN BIT;

c: IN BIT;

x, y, z, k : OUT BIT );

end Ex\_4\_9\_2;

architecture Behavioral of Ex\_4\_9\_2 is

begin

abc: PROCESS (a, b, c)

BEGIN

IF a = '1' THEN

x <= '1';

ELSE

x <= '0';

END IF;

IF c = '1' THEN

z <= '1';

ELSE

z <= '0';

END IF;

IF a = '1' AND b = '1' THEN

y <= '0';

ELSE

y <= '1';

END IF;

IF c = '1' AND b = '1' THEN

k <= '1';

ELSE

k <= '0';

END IF;

END PROCESS abc;

end Behavioral;

-- O código original é apresentado abaixo. Foi convertido a entrada "c" para que fosse

-- possível usar a função IF e também foi introduzido as condições que não foram descritas

-- quando nenhuma das condições fossem satisfeitas.

--

-- Código:

--ENTITY erro1 IS

-- PORT(a,b: IN BIT;

-- c : IN BOOLEAN;

-- x , y, z, k : OUT BIT);

--END erro1;

--ARCHITECTURE teste OF erro1 IS

--BEGIN

-- abc: PROCESS (a,b,c)

-- BEGIN

-- IF a THEN x <= '1';

-- END IF;

-- IF c THEN z <= '1';

-- END IF;

-- IF a AND b = '1' THEN y <= '0';

-- END IF;

-- IF c AND b = '1' THEN k <= '1';

-- END IF;

-- END PROCESS abc;

--END teste;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_Ex\_4\_9\_2 is

end tb\_Ex\_4\_9\_2;

architecture Behavioral of tb\_Ex\_4\_9\_2 is

COMPONENT Ex\_4\_9\_2

PORT(a, b : IN BIT;

c: IN BIT;

x, y, z, k : OUT BIT);

END COMPONENT;

signal a : BIT := '0';

signal b : BIT := '0';

signal c : BIT := '0';

signal x : BIT;

signal y : BIT := '1';

signal z : BIT;

signal k : BIT;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_2 PORT MAP(

a => a,

b => b,

c => c,

x => x,

y => y,

z => z,

k => k);

-- Processo de estímulo

stp: PROCESS

BEGIN

a <= NOT a;

WAIT FOR 100 ns;

END PROCESS;

stp1: PROCESS

BEGIN

b <= NOT b;

WAIT FOR 50 ns;

END PROCESS;

stp2: PROCESS

BEGIN

c <= NOT c;

WAIT FOR 25 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.2:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_3 is

Port (x : IN INTEGER RANGE 0 TO 15;

s : OUT INTEGER RANGE 0 TO 15);

end Ex\_4\_9\_3;

architecture Behavioral of Ex\_4\_9\_3 is

begin

abc: PROCESS (x)

BEGIN

CASE x IS

WHEN 7 TO 10 => s <= 0;

WHEN 3 DOWNTO 0 => s <= 1;

WHEN 4 | 5 | 6 => s <= 2;

WHEN 15 => s <= 3;

WHEN OTHERS => s <= 4;

END CASE;

END PROCESS abc;

end Behavioral;

----------------------------------------------------------------------------------

-- Para solucionar o problema foi retirado o a opção 9 da linha 46 e também

-- adicionado a opção para as opções não atingidas, levando a saída s para 4 para

-- as outras opções não terem o mesmo sinal de saída.(mesmo com a opção null)

-- O código original era:

--library IEEE;

--use IEEE.STD\_LOGIC\_1164.ALL;

--

--entity Ex\_4\_9\_3 is

-- Port (x : IN INTEGER RANGE 0 TO 15;

-- s : OUT INTEGER RANGE 0 TO 15);

--end Ex\_4\_9\_3;

--

--architecture Behavioral of Ex\_4\_9\_3 is

--begin

-- abc: PROCESS (x)

-- BEGIN

-- CASE x IS

-- WHEN 7 TO 10 => s <= 0;

-- WHEN 3 DOWNTO 0 => s <= 1;

-- WHEN 4 | 5 | 9 => s <= 2;

-- WHEN 15 => s <= 3;

-- END CASE;

-- END PROCESS abc;

--end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_Ex\_4\_9\_3 is

end tb\_Ex\_4\_9\_3;

architecture Behavioral of tb\_Ex\_4\_9\_3 is

COMPONENT Ex\_4\_9\_3

PORT(x : IN INTEGER RANGE 0 TO 15 ;

s : OUT INTEGER RANGE 0 TO 15);

END COMPONENT;

--INPUT

signal x : INTEGER RANGE 0 TO 15 := 0;

--OUTPUT

signal s : INTEGER RANGE 0 TO 15;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_3 PORT MAP(

x => x,

s => s

);

-- Processo de estímulo

stp: PROCESS

BEGIN

x <= x + 1;

WAIT FOR 50 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.3:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_4 is

Port (i0, i1, i2, i3 : IN BIT;

s0, s1 : IN BIT;

ot : OUT BIT);

end Ex\_4\_9\_4;

architecture Behavioral of Ex\_4\_9\_4 is

begin

abc: PROCESS (i0,i1,i2,i3,s0,s1)

BEGIN

IF s1 & s0 = "00" THEN ot <= i0;

ELSIF s1 & s0 = "01" THEN ot <= i1;

ELSIF s1 & s0 = "10" THEN ot <= i2;

ELSE ot <= i3;

END IF;

END PROCESS abc;

end Behavioral;

-- Sem as saídas de seleção o código gera uma saída errada em "ot", pois ao mudar as saídas

-- de seleção a saída "ot" não acompanha, portanto faz-se necessário estarem presente na

-- lista de sensibilidade.

-- O código original era:

--

--library IEEE;

--use IEEE.STD\_LOGIC\_1164.ALL;

--

--entity Ex\_4\_9\_4 is

-- Port (i0, i1, i2, i3 : IN BIT;

-- s0, s1 : IN BIT;

-- ot : OUT BIT);

--end Ex\_4\_9\_4;

--

--architecture Behavioral of Ex\_4\_9\_4 is

--

--begin

-- abc: PROCESS (i0,i1,i2,i3)

-- BEGIN

-- IF s1 & s0 = "00" THEN ot <= i0;

-- ELSIF s1 & s0 = "01" THEN ot <= i1;

-- ELSIF s1 & s0 = "10" THEN ot <= i2;

-- ELSE ot <= i3;

-- END IF;

-- END PROCESS abc;

--end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_Ex\_4\_9\_3 is

end tb\_Ex\_4\_9\_3;

architecture Behavioral of tb\_Ex\_4\_9\_3 is

COMPONENT Ex\_4\_9\_3

PORT(x : IN INTEGER RANGE 0 TO 15 ;

s : OUT INTEGER RANGE 0 TO 15);

END COMPONENT;

--INPUT

signal x : INTEGER RANGE 0 TO 15 := 0;

--OUTPUT

signal s : INTEGER RANGE 0 TO 15;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_3 PORT MAP(

x => x,

s => s

);

-- Processo de estímulo

stp: PROCESS

BEGIN

x <= x + 1;

WAIT FOR 50 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.4:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_4 is

Port (i0, i1, i2, i3 : IN BIT;

s0, s1 : IN BIT;

ot : OUT BIT);

end Ex\_4\_9\_4;

architecture Behavioral of Ex\_4\_9\_4 is

begin

abc: PROCESS (i0,i1,i2,i3,s0,s1)

BEGIN

IF s1 & s0 = "00" THEN ot <= i0;

ELSIF s1 & s0 = "01" THEN ot <= i1;

ELSIF s1 & s0 = "10" THEN ot <= i2;

ELSE ot <= i3;

END IF;

END PROCESS abc;

end Behavioral;

-- Sem as saídas de seleção o código gera uma saída errada em "ot", pois ao mudar as saídas

-- de seleção a saída "ot" não acompanha, portanto faz-se necessário estarem presente na

-- lista de sensibilidade.

-- O código original era:

--

--library IEEE;

--use IEEE.STD\_LOGIC\_1164.ALL;

--

--entity Ex\_4\_9\_4 is

-- Port (i0, i1, i2, i3 : IN BIT;

-- s0, s1 : IN BIT;

-- ot : OUT BIT);

--end Ex\_4\_9\_4;

--

--architecture Behavioral of Ex\_4\_9\_4 is

--

--begin

-- abc: PROCESS (i0,i1,i2,i3)

-- BEGIN

-- IF s1 & s0 = "00" THEN ot <= i0;

-- ELSIF s1 & s0 = "01" THEN ot <= i1;

-- ELSIF s1 & s0 = "10" THEN ot <= i2;

-- ELSE ot <= i3;

-- END IF;

-- END PROCESS abc;

--end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_Ex\_4\_9\_4 is

end tb\_Ex\_4\_9\_4;

architecture Behavioral of tb\_Ex\_4\_9\_4 is

COMPONENT Ex\_4\_9\_4

PORT(i0, i1, i2, i3 : IN BIT;

s0, s1 : IN BIT;

ot : OUT BIT);

END COMPONENT;

signal i0 : BIT := '0';

signal i1 : BIT := '0';

signal i2 : BIT := '0';

signal i3 : BIT := '0';

signal s0 : BIT := '0';

signal s1 : BIT := '0';

signal ot : BIT;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_4 PORT MAP(

i0 => i0,

i1 => i1,

i2 => i2,

i3 => i3,

s0 => s0,

s1 => s1,

ot => ot);

-- Processo de estímulo

stp1: PROCESS

BEGIN

i0 <= NOT i0;

WAIT FOR 400 ns;

END PROCESS;

stp2: PROCESS

BEGIN

i1 <= NOT i1;

WAIT FOR 200 ns;

END PROCESS;

stp3: PROCESS

BEGIN

i2 <= NOT i2;

WAIT FOR 100 ns;

END PROCESS;

stp4: PROCESS

BEGIN

i3 <= NOT i3;

WAIT FOR 50 ns;

END PROCESS;

stp5: PROCESS

BEGIN

s0 <= NOT s0;

WAIT FOR 25 ns;

END PROCESS;

stp6: PROCESS

BEGIN

s1 <= NOT s1;

WAIT FOR 12.5 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.5:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_5 is

Port (x, y : IN BIT;

a ,b, c : IN BIT;

s : OUT BIT);

end Ex\_4\_9\_5;

architecture Behavioral of Ex\_4\_9\_5 is

begin

abc:PROCESS (a,b,c,x,y)

BEGIN

IF a = b THEN

s <= x AND y;

ELSIF b = c THEN

s <= x OR y;

ELSIF a = c THEN

s <= x XOR y;

END IF;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_Ex\_4\_9\_5 is

end tb\_Ex\_4\_9\_5;

architecture Behavioral of tb\_Ex\_4\_9\_5 is

COMPONENT Ex\_4\_9\_5

PORT(x, y : IN BIT;

a ,b, c : IN BIT;

s : OUT BIT);

END COMPONENT;

signal x : BIT := '0';

signal y : BIT := '0';

signal a : BIT := '0';

signal b : BIT := '0';

signal c : BIT := '0';

signal s : BIT;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_5 PORT MAP(

a => a,

b => b,

c => c,

x => x,

y => y,

s => s

);

-- Processo de estímulo

stp3: PROCESS

BEGIN

x <= NOT x;

WAIT FOR 200 ns;

END PROCESS;

stp4: PROCESS

BEGIN

y <= NOT y;

WAIT FOR 100 ns;

END PROCESS;

stp: PROCESS

BEGIN

a <= NOT a;

WAIT FOR 50 ns;

END PROCESS;

stp1: PROCESS

BEGIN

b <= NOT b;

WAIT FOR 25 ns;

END PROCESS;

stp2: PROCESS

BEGIN

c <= NOT c;

WAIT FOR 12.5 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.6:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_6 is

Port (d : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

en : IN BIT;

q : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0));

end Ex\_4\_9\_6;

architecture Behavioral of Ex\_4\_9\_6 is

begin

abc: PROCESS (d,en)

BEGIN

IF en = '1' THEN

q <= d;

END IF;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_6 is

end tb\_Ex\_4\_9\_6;

architecture Behavioral of tb\_Ex\_4\_9\_6 is

COMPONENT Ex\_4\_9\_6

PORT(d : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

en : IN BIT;

q : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0));

END COMPONENT;

signal d : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";

signal en : BIT := '0';

signal q : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_6 PORT MAP(

d => d,

en => en,

q => q);

-- Processo de estímulo

stp: PROCESS

BEGIN

d <= d + "0001";

WAIT FOR 100 ns;

END PROCESS;

stp1: PROCESS

BEGIN

en <= NOT en;

WAIT FOR 50 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.7:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_7 is

Port (d : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

en: IN BIT;

q : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0));

end Ex\_4\_9\_7;

architecture Behavioral of Ex\_4\_9\_7 is

begin

abc: PROCESS(d,en)

BEGIN

CASE en IS

WHEN '1' => q <= d;

WHEN OTHERS => null;

END CASE;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_7 is

end tb\_Ex\_4\_9\_7;

architecture Behavioral of tb\_Ex\_4\_9\_7 is

COMPONENT Ex\_4\_9\_7

PORT(d : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

en : IN BIT;

q : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0));

END COMPONENT;

signal d : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";

signal en : BIT := '0';

signal q : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_7 PORT MAP(

d => d,

en => en,

q => q);

-- Processo de estímulo

stp: PROCESS

BEGIN

d <= d + "0001";

WAIT FOR 100 ns;

END PROCESS;

stp1: PROCESS

BEGIN

en <= NOT en;

WAIT FOR 50 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.8:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_8 is

Port (p : IN STD\_LOGIC\_VECTOR(0 TO 3);

x : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

int: OUT BIT);

end Ex\_4\_9\_8;

architecture Behavioral of Ex\_4\_9\_8 is

begin

abc: PROCESS (p)

BEGIN

IF p(0) = '1' THEN

int <= '1';

x <= "00";

ELSIF p(1) = '1' THEN

int <= '1';

x <= "01";

ELSIF p(2) = '1' THEN

int <= '1';

x <= "10";

ELSIF p(3) = '1' THEN

int <= '1';

x <= "11";

ELSIF p = "0000" THEN

int <= '0';

x <= "11";

END IF;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_8 is

end tb\_Ex\_4\_9\_8;

architecture Behavioral of tb\_Ex\_4\_9\_8 is

COMPONENT Ex\_4\_9\_8

PORT(p : IN STD\_LOGIC\_VECTOR(0 TO 3);

x : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

int: OUT BIT);

END COMPONENT;

signal p : STD\_LOGIC\_VECTOR(0 TO 3) := "0000";

signal x : STD\_LOGIC\_VECTOR(1 DOWNTO 0);

signal int : BIT;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_8 PORT MAP(

p => p,

x => x,

int => int

);

-- Processo de estímulo

stp: PROCESS

BEGIN

p <= p + "0001";

WAIT FOR 100 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.9:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_9 is

Port (p : IN STD\_LOGIC\_VECTOR(0 TO 3);

x : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

int: OUT BIT );

end Ex\_4\_9\_9;

architecture Behavioral of Ex\_4\_9\_9 is

begin

abc:PROCESS (p)

BEGIN

CASE p IS

WHEN "1000"|"1001"|"1010"|"1011"|"1100"|"1101"|"1110"|"1111" => x <= "00"; int <= '1';

WHEN "0100"|"0101"|"0110"|"0111" => x <= "01"; int <= '1';

WHEN "0010"|"0011" => x <= "10"; int <= '1';

WHEN "0001" => x <= "11"; int <= '1';

WHEN others => x <= "11"; int <= '0';

END CASE;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_9 is

end tb\_Ex\_4\_9\_9;

architecture Behavioral of tb\_Ex\_4\_9\_9 is

COMPONENT Ex\_4\_9\_9

PORT(p : IN STD\_LOGIC\_VECTOR(0 TO 3);

x : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

int: OUT BIT);

END COMPONENT;

signal p : STD\_LOGIC\_VECTOR(0 TO 3) := "0000";

signal x : STD\_LOGIC\_VECTOR(1 DOWNTO 0);

signal int : BIT;

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_9 PORT MAP(

p => p,

x => x,

int => int

);

-- Processo de estímulo

stp: PROCESS

BEGIN

p <= p + "0001";

WAIT FOR 100 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.10:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_10 is

Port (a : IN STD\_LOGIC\_VECTOR(17 DOWNTO 0);

cs: OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0));

end Ex\_4\_9\_10;

architecture Behavioral of Ex\_4\_9\_10 is

begin

abc:PROCESS (a)

BEGIN

IF a(17 DOWNTO 15) = "000" OR a(17 DOWNTO 15) = "011" THEN

cs <= "110";

ELSIF a(17 DOWNTO 15) = "100" THEN

cs <= "101";

ELSIF a(17 DOWNTO 15) = "101" THEN

cs <= "011";

ELSE cs <= "111";

END IF;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_10 is

end tb\_Ex\_4\_9\_10;

architecture Behavioral of tb\_Ex\_4\_9\_10 is

COMPONENT Ex\_4\_9\_10

PORT(a : IN STD\_LOGIC\_VECTOR(17 DOWNTO 0);

cs: OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0)

);

END COMPONENT;

signal a : STD\_LOGIC\_VECTOR(17 DOWNTO 0) := "000000000000000000";

signal cs : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_10 PORT MAP(

a => a,

cs => cs

);

-- Processo de estímulo

stp: PROCESS

BEGIN

a <= a + "000000000000000001";

WAIT FOR 100 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.11:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_11 is

Port (a : IN STD\_LOGIC\_VECTOR(17 DOWNTO 0);

cs: OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0)

);

end Ex\_4\_9\_11;

architecture Behavioral of Ex\_4\_9\_11 is

begin

abc:PROCESS (a)

BEGIN

CASE a(17 DOWNTO 15) IS

WHEN "000" | "011" => cs <= "110";

WHEN "100" => cs <= "101";

WHEN "101" => cs <= "011";

WHEN OTHERS => cs <= "111";

END CASE;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_11 is

end tb\_Ex\_4\_9\_11;

architecture Behavioral of tb\_Ex\_4\_9\_11 is

COMPONENT Ex\_4\_9\_11

PORT(a : IN STD\_LOGIC\_VECTOR(17 DOWNTO 0);

cs: OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0)

);

END COMPONENT;

signal a : STD\_LOGIC\_VECTOR(17 DOWNTO 0) := "000000000000000000";

signal cs : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_11 PORT MAP(

a => a,

cs => cs

);

-- Processo de estímulo

stp: PROCESS

BEGIN

a <= a + "000000000000000001";

WAIT FOR 100 ns;

END PROCESS;

END Behavioral;

**Ex 4.9.12:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_12 is

Port (i : IN STD\_LOGIC\_VECTOR(0 to 3);

o : OUT STD\_LOGIC\_VECTOR(0 to 6) );

end Ex\_4\_9\_12;

architecture Behavioral of Ex\_4\_9\_12 is

begin

abc:PROCESS (i)

BEGIN

CASE i IS

when "0000" => o <= "1111110";

when "0001" => o <= "0110000";

when "0010" => o <= "1101101";

when "0011" => o <= "1111001";

when "0100" => o <= "0110011";

when "0101" => o <= "1011011";

when "0110" => o <= "1011111";

when "0111" => o <= "1110000";

when "1000" => o <= "1111111";

when "1001" => o <= "1111011";

when others => o <= "0000000";

END CASE;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_12 is

end tb\_Ex\_4\_9\_12;

architecture Behavioral of tb\_Ex\_4\_9\_12 is

COMPONENT Ex\_4\_9\_12

PORT(i : IN STD\_LOGIC\_VECTOR(0 to 3);

o : OUT STD\_LOGIC\_VECTOR(0 to 6));

END COMPONENT;

signal i : STD\_LOGIC\_VECTOR(0 to 3) := "0000";

signal o : STD\_LOGIC\_VECTOR(0 to 6);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_12 PORT MAP(

i => i,

o => o

);

-- Processo de estímulo

stp: PROCESS

BEGIN

i <= i + "0001";

WAIT FOR 100 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.13:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_13 is

Port (bcd : IN STD\_LOGIC\_VECTOR(3 downto 0);

seg : OUT STD\_LOGIC\_VECTOR(0 to 6);

sel : IN STD\_LOGIC\_VECTOR(1 downto 0);

mst : OUT STD\_LOGIC\_VECTOR(0 to 3));

end Ex\_4\_9\_13;

architecture Behavioral of Ex\_4\_9\_13 is

begin

decodificador:PROCESS (bcd)

BEGIN

CASE bcd IS

WHEN "0000" => seg <= "1111110";

WHEN "0001" => seg <= "0110000";

WHEN "0010" => seg <= "1101101";

WHEN "0011" => seg <= "1111001";

WHEN "0100" => seg <= "0110011";

WHEN "0101" => seg <= "1011011";

WHEN "0110" => seg <= "1011111";

WHEN "0111" => seg <= "1110000";

WHEN "1000" => seg <= "1111111";

WHEN "1001" => seg <= "1111011";

WHEN others => null;

END CASE;

END PROCESS decodificador;

selecao:PROCESS(sel)

BEGIN

CASE sel IS

WHEN "00" => mst <= "0111";

WHEN "01" => mst <= "1011";

WHEN "10" => mst <= "1101";

WHEN "11" => mst <= "1110";

WHEN others => null;

END CASE;

END PROCESS selecao;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_13 is

end tb\_Ex\_4\_9\_13;

architecture Behavioral of tb\_Ex\_4\_9\_13 is

COMPONENT Ex\_4\_9\_13

Port (bcd : IN STD\_LOGIC\_VECTOR(3 downto 0);

seg : OUT STD\_LOGIC\_VECTOR(0 to 6);

sel : IN STD\_LOGIC\_VECTOR(1 downto 0);

mst : OUT STD\_LOGIC\_VECTOR(0 to 3));

END COMPONENT;

signal bcd : STD\_LOGIC\_VECTOR(3 downto 0) := "0000";

signal sel : STD\_LOGIC\_VECTOR(1 downto 0):= "00";

signal seg : STD\_LOGIC\_VECTOR(0 to 6);

signal mst : STD\_LOGIC\_VECTOR(0 to 3);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_13 PORT MAP(

bcd => bcd,

seg => seg,

sel => sel,

mst => mst

);

-- Processo de estímulo

stp: PROCESS

BEGIN

bcd <= bcd + "0001";

WAIT FOR 100 ns;

END PROCESS;

stp1: PROCESS

BEGIN

sel <= sel + "01";

WAIT FOR 100 ns;

END PROCESS;

end Behavioral;

**Ex 4.9.14:**

- Descrição de um Barrel shift com aninhamento dos sinais. Para realizar sem ani

-- nhamento basta não atribuir as entradas "d1" e "d2" para d. Poderia ser descrito

-- declarando somente uma entrada "d" como "STD\_LOGIC\_VECTOR". Neste caso foi usado

-- o comando sequencial "IF ELSE".

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_14 is

Port (v : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

d1,d0 : IN BIT;

s : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0));

end Ex\_4\_9\_14;

architecture Behavioral of Ex\_4\_9\_14 is

signal d : BIT\_VECTOR(1 Downto 0 );

begin

d <= d1 & d0;

abc:PROCESS (v,d)

BEGIN

IF d = "00" THEN

s(3) <= v(3);

s(2) <= v(2);

s(1) <= v(1);

s(0) <= v(0);

ELSIF d = "01" THEN

s(3) <= v(2);

s(2) <= v(1);

s(1) <= v(0);

s(0) <= v(3);

ELSIF d = "10" THEN

s(3) <= v(1);

s(2) <= v(0);

s(1) <= v(3);

s(0) <= v(2);

ELSIF d = "11" THEN

s(3) <= v(0);

s(2) <= v(3);

s(1) <= v(2);

s(0) <= v(1);

END IF;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity tb\_Ex\_4\_9\_14 is

end tb\_Ex\_4\_9\_14;

architecture Behavioral of tb\_Ex\_4\_9\_14 is

COMPONENT Ex\_4\_9\_14

PORT(v : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

d1,d0 : IN BIT;

s : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0));

END COMPONENT;

signal v : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";

signal d1 : BIT := '0';

signal d0 : BIT := '0';

signal s : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_14 PORT MAP(

v => v,

d1 => d1,

d0 => d0,

s => s

);

-- Processo de estímulo

**Ex 4.9.14a:**

- Descrição de um Barrel shift com aninhamento dos sinais. Para realizar sem ani

-- nhamento basta não atribuir as entradas "d1" e "d2" para d. Poderia ser descrito

-- declarando somente uma entrada "d" como "STD\_LOGIC\_VECTOR". Neste caso foi usado

-- o comando sequencial "CASE WHEN". PErcebe-se que é menos trabalhoso do que o

-- comando sequencial "IF ELSE".

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ex\_4\_9\_14a IS

Port (v : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

d1,d0 : IN BIT;

s : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0));

end Ex\_4\_9\_14a;

architecture Behavioral of Ex\_4\_9\_14a is

signal d : BIT\_VECTOR(1 Downto 0 );

begin

d <= d1 & d0;

abc:PROCESS (v,d)

BEGIN

CASE d IS

WHEN "00" => s(3) <= v(3); s(2) <= v(2); s(1) <= v(1); s(0) <= v(0);

WHEN "01" => s(3) <= v(2); s(2) <= v(1); s(1) <= v(0); s(0) <= v(3);

WHEN "10" => s(3) <= v(1); s(2) <= v(0); s(1) <= v(3); s(0) <= v(2);

WHEN "11" => s(3) <= v(0); s(2) <= v(3); s(1) <= v(2); s(0) <= v(1);

WHEN others => null;

END CASE;

END PROCESS abc;

end Behavioral;

**testbench:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_Ex\_4\_9\_14a is

end tb\_Ex\_4\_9\_14a;

architecture Behavioral of tb\_Ex\_4\_9\_14a is

COMPONENT Ex\_4\_9\_14a

PORT(v : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

d1,d0 : IN BIT;

s : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0));

END COMPONENT;

signal v : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";

signal d1 : BIT := '0';

signal d0 : BIT := '0';

signal s : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

begin

--Instância da Unit Under test (UUT)

uut: Ex\_4\_9\_14a PORT MAP(

v => v,

d1 => d1,

d0 => d0,

s => s

);

-- Processo de estímulo

stp: PROCESS

BEGIN

WAIT FOR 100 ns;

v <= v + "0001";

END PROCESS;

stp1: PROCESS

BEGIN

WAIT FOR 100 ns;

d1 <= NOT d1;

END PROCESS;

stp2: PROCESS

BEGIN

WAIT FOR 50 ns;

d0 <= NOT d0;

END PROCESS;

end Behavioral;