Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Eletrônica

MODULO LDPC PARA SISTEMAS DE TELECOMUNICAÇÕES DVB-S2X

Autor: Marcos Adriano Nery de Abrantes
Orientador: Professor titular José Camargo da Costa - PHD, ENE/UnB

Brasília, DF
2019

Marcos Adriano Nery de Abrantes

MODULO LDPC PARA SISTEMAS DE TELECOMUNICAÇÕES DVB-S2X

Relatório submetido ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Professor titular José Camargo da Costa - PHD, ENE/UnB

Coorientador: Professor Daniel Mauricio Muñoz Arboleda - PHD, FGA/UnB

Brasília, DF

2019

Marcos Adriano Nery de Abrantes
MODULO LDPC PARA SISTEMAS DE TELECOMUNICAÇÕES DVB-S2X/ Marcos Adriano Nery de Abrantes. – Brasília, DF, 2019-
110 p. : il. (algumas color.) ; 30 cm.

Orientador: Professor titular José Camargo da Costa - PHD, ENE/UnB

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2019.

1. FPGA. 2. comunicação satelital. 3. DVB-S2X. 4. DVB-S2X.
I. Professor titular José Camargo da Costa - PHD, ENE/UnB. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. MODULO LDPC PARA SISTEMAS DE TELECOMUNICAÇÕES DVB-S2X

CDU 02:141:005.6

Marcos Adriano Nery de Abrantes

MODULO LDPC PARA SISTEMAS DE TELECOMUNICAÇÕES DVB-S2X

Relatório submetido ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Trabalho aprovado. Brasília, DF, 10 de julho de 2019:

Professor titular José Camargo da Costa - PHD, ENE/UnB
Orientador

Professor Daniel Mauricio Muñoz Arboleda - PHD, FGA/UnB
coorientador

Professor Leonardo Aguayo - PHD, FGA/UnB
Convidado 1

Professor Guillermo Alvarez Bestard - Doutor, FGA/UnB
Convidado 2

Brasília, DF
2019

Dedico a Deus que em tudo me susteve e
aos meus pais, pois sem eles nada disso seria possivel.

Agradecimentos

A Deus por ter me dado força e animo para que até aqui eu esteja com considerável saúde mental e física.

Ao professor Daniel que sempre se mostrou disponível e me orientou nas atividade e no desenvolvimento dessas.

Ao professor Camargo que me orientou junto as atividades cumpridas. Pelo suporte mesmo com sua estrita agenda.

Ao Zoé que sempre esteve disponível e me auxiliou quanto as minhas dificuldades.

A meus pais, irmãos e família.

Um especial agradecimento a Julie minha namorada que me deu suporte e esteve ao meu lado sempre que precisei, sem quem eu não seria capaz de cumprir com essa jornada.

Aos amigos em especial o SAMU.

E a todos que fizeram parte desse aprendizado e do cumprimento de mais uma jornada.

"Então Calebe fez calar o povo perante Moisés, e disse:
Certamente subiremos e a possuiremos em herança;
porque seguramente prevaleceremos contra ela.
(Bíblia Sagrada, Números 13, 30)

Resumo

A comunicação sempre foi fundamental para a história humana e hoje não é diferente. Atualmente, a comunicação não depende mais exclusivamente das relações pessoais, mas acontece principalmente através da tecnologia. Com o advento das tecnologias, tornou-se possível aos indivíduos manter contato, realizar ações e até mesmo cumprir suas atividades profissionais à distância. Como tal, os avanços no campo das comunicações são primordiais para o desenvolvimento profissional da sociedade moderna.

Neste cenário tem-se o acesso a informação sem necessidade de estar fisicamente ligado a uma rede, neste caso com a tecnologia de envio de informações sem fio, aonde um dos maiores protagonistas se condiz ao satélite. Para que isso seja possível é necessário que exista um sistema eficaz e que seja capaz de cumprir com isto. Assim o sistema de comunicação via satélite nos dias atuais deixou de ser opcional ao homem e passou a ser uma necessidade.

Em qualquer comunicação via satélite é necessário a integração e existências de diversos sistemas. O satélite e a unidade em terra são essenciais. Não atrás destes está o meio e como ocorre a comunicação. A comunicação em questão é feita tendo como canal o ar enquanto como ocorre tem sua consistência em um padrão. Padrão é o estabelecimento de protocolos que permitem que a comunicação aconteça de forma que formaliza uma comunicação.

Existem diversos padrões existentes, não apenas na tecnologia de telecomunicação mas em todos o ato de mandar e receber sinal. Dentre esses padrões um de demasiada importância é o DVB-S2X, este apresenta uma operação muito próxima do limite de Shannon e baseia seu FEC na correção de erro com o uso de um LDPC concatenado com um BCH. Aqui vale iterar sobre o LDPC, esse pertence a uma classe de códigos de bloco linear e tem a habilidade de detectar e corrigir erros causados por ruídos do canal de comunicação. Este trabalho traz a implementação do decodificador de erros LDPC do padrão DVB-S2X com a utilização de recursos de *hardware* de forma compatível com as taxas de transmissão do sistema referente. Tendo como intuito a implementação em FPGA de arquiteturas compatíveis com o LDPC usado no padrão de comunicação DVB-S2X.

O LDPC de forma geral apresenta uma característica de serem altamente paralelizáveis, normalmente na literatura com 180 ou 360 FUs. No caso deste trabalho a implementação ocorre com 360 células. Para tal será usado a plataforma de *hardware Xilinx - NEXYS 4* com uso do *software VIVADO 2018.3*. Foi feita a implementação das arquiteturas, sua simulação, síntese, mapeamento e roteamento em FPGA, e obtido informações acerca da frequência máxima de operação, consumo de recursos e energia acerca do funcionamento do LDPC.

Palavras-chaves:Comunicação. Satélite. FPGA. RDS. Arquiteturas integradas. DVB-S2X.

Abstract

Communication has always been pivotal to the human history, and today it is no different. In these days, communication no longer depends exclusively in the personal relationships, but happens mostly through technology. With the advent of technologiesm it has become possible for individuals to maintain contact, carry out actions and even comply with their professional activities at a distance. As such, advancements in the field of communications are primordial to the professional development of the modern society.

In this scenario one has the access to information without the need to be physically physically connected to a network, in this case the technology of wireless information sending, where one of the main protagonists corresponds to the satellite. In order to enter this world, there must be an effective system that is capable of accomplishing this. So today's satellite communication system becomes essential to man.

For satellite communication to operate, integration and existences of several factors are necessary. The satellite and ground unit are essential. Not behind is the medium and how communication occurs. The communication in question is made by having as a channel the air while it occurs has its consistency in a pattern. Standard is what sets itself is the establishment of protocols that allow communication to happen and formalize a form of communication.

A very important standard is the DVB-S2X, which presents an operation very close to the Shannon boundary and bases its FEC on error correction with the use of a LDPC concatenated with a BCH. LDPC belongs to a linear block code class, and has the ability to detect and correct errors caused by communication channel noise. This work brings the implementation of the LDPC error decoder of the DVB-S2X standard with the use of textit hardware resources in a way compatible with the transmission rates of the referring system.

This work intends to implement FPGA-compatible architectures with the LDPC used in the DVB-S2X communication standard. LDPC in general has a feature of being highly parallelizable, usually in the literature with 180 or 360 FUs. In the case of this work the implementation occurs with 360 cells. Using the hardware textit xilinx - NEXYS 4 platform using the VIVADO software 2018.3. It was made the implementation of the architectures, and from the results of simulation, synthesis, mapping and routing in FPGA, we obtained information about the maximum frequency of operation, resource consumption and energy.

Key-words: Communication. Satellite. FPGA. RDS. Integrated architectures. DVB-S2X.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACM	Codificação e Modulação Adaptativa
AD	Analógico - Digital
ASIC	Circuitos Integrados de Aplicações Especıficas
BCH	*Bose Chaudhuri Hocquenghem*
BER	*Bit Error Rate*
CN	*Nó de verificação*
DA	Digital - Analógico
DDC	*Digital Down Converter*
DSP	Processadores Digitais de Sinais
DUC	*Digital Up Converter*
Downlink	Caminho das ondas do satélite
DVB-RCS	Digital Video Broadcasting Return Channel by Satellite
DVB-RCS2	Digital Video Broadcasting - Interactive Satellite System, Return Channel by Satellite second generation
DVB-S	Digital Video Broadcasting by Satellite
DVB-S2	Digital Video Broadcasting - Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications
ETSI	*European Telecommunications Standards Institute*
FEC	*Forward Error Correction*
FI	Frequência Intermediaria
FF	*Flip Flops*
FPGA	Conjuntos de Portas Logicas Programáveis em Campo
FU	*Unidade Funcional*
IDE	*Integrated development environment*

IP	*Intellectual Property*
LDPC	*Low Density Parity Check*
LLR	*Log Likelihood ratio*
LUT	*Look Up Tables*
MAC	*Medium Access Control*
MUX	*Multiplexadores*
NF	Figura de Ruido
PLD	Dispositivo Lógico Programável
QEF	Quasi Error-Free
RAM	*Random access memory*
RDS	Rádio definido por software
RF	Rádio frequência
RMS	*Root Mean Square*
ROM	*Read only memory*
RTL	*Registro nível de transferência*
SMA	*Algoritmo Min-Sum*
SNR	*Relação sinal-ruído*
SO	*SoftOutput*
SPA	*Algorítimo Soma Produto*
ULA	Unidade Logica e Aritmética
Uplink	Caminho percorrido pelas ondas eletromagnéticas até o satélite
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very-High Speed Integrated Circuits*
VN	*Nó de variável*
VSAT	*Very Small Aperture Terminal*

Sumário

1 Introdução

1.1 Contextualização

Atualmente a utilização de satélites para a comunicação é extremamente difundida. Possibilita com um grande diferencial de qualquer outro meio de comunicação o enlace entre locais distantes com o mínimo de pontos e terminam apresentando a melhor solução para comunicação entre uma dada estação base e seus terminais. Essa comunicação se apresenta em serviços de telecomunicações na terra, mar e ar, também em transmissão de televisão, internet, telefonia, rastreamento de veículos, barcos e aviões. (MARAL; BOUSQUET, 2002).

Existe uma grande indústria formada em torno da comunicação como um todo. Isso possibilitou que a comunicação comercial via satélite progredisse nos últimos 50 anos para uma tecnologia de comunicação de massa. (COSTA et al., 2017). Há, assim, uma grande importância no uso e administração desses recursos tecnológicos e sua cobertura.

1.2 Descrição do Problema

Este trabalho está inserido como parte do desenvolvimento de um rádio definido por software (RDS), em que são recebidos *frames* codificados segundo o padrão DVB-S2X. A Figura 1 apresenta um diagrama de blocos com as etapas de processamento que devem ser realizados pelo terminal na etapa de decodificação de dados, com destaque para os blocos corretores de erro.

1.2.1 Erros em comunicação satelital

Um sistema de comunicação é nada mais que elementos trocando sinais. Esses sinais representam informações, e são propagados em um meio que é o canal. São sensíveis a distorções e ruídos podendo ser alterados durante a transmissão. As distorções implicadas são de forma geral operações aplicadas ao sinal, que podem ser corrigidas aplicando a operação inversa. Já os ruídos são perturbações que tem caráter imprevisível, de forma que nem sempre podem ser completamente removidas, sendo necessário que no receptor tenha um mecanismo que permita associar os dados e consequentemente sua recuperação (SHANNON, 1948).

Assim como em uma conversa, em uma comunicação satelital é necessário que as informações recebidas correspondam às enviadas ou ao menos possibilitem a recuperação

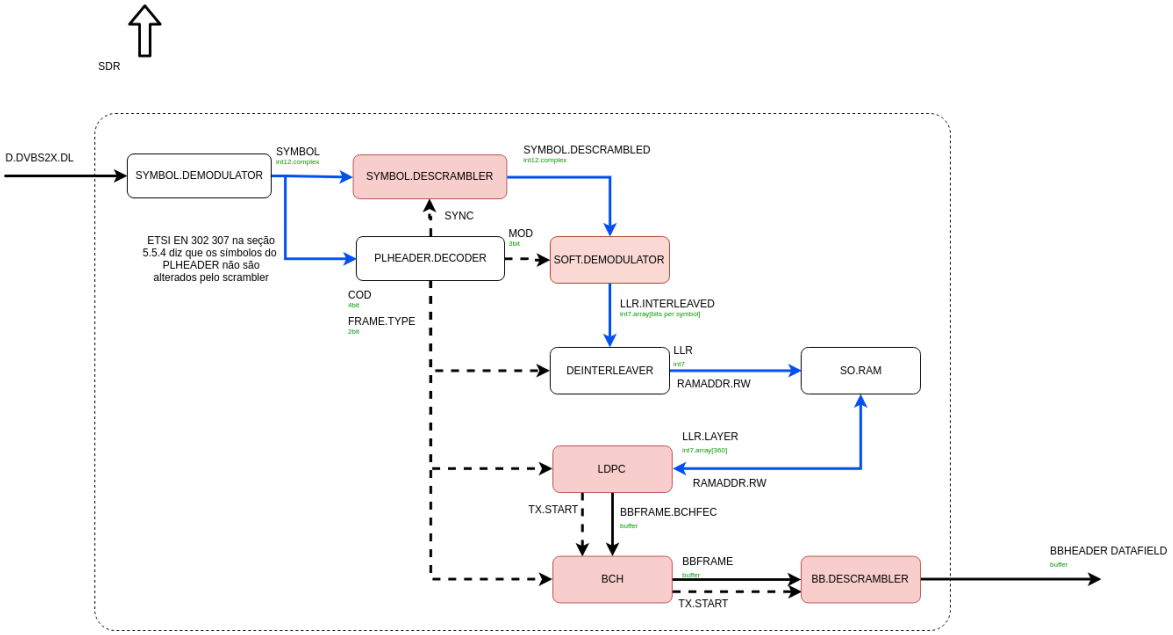


Figura 1 – Diagrama de blocos com as etapas de processamento realizados pelo terminal para decodificação no padrão DVB-S2X (ETSI, 2009).

do que foi enviado. Para que isso ocorra, os sistemas precisam oferecer um desempenho satisfatório na taxa de erros, de acordo com o cenário e aplicação (MARINS et al., 2004).

Para se obter uma transmissão de alta confiabilidade, é necessário o uso de técnicas de processamento digital de sinais. Existem técnicas para correção de erros, e estas técnicas são capazes de detectar e corrigir erros inseridos pelo canal de comunicação (PIMENTEL, 2007).

As técnicas que se apresentam como solução para falhas e erros de comunicação, ou mais precisamente, técnicas de correção de erros, costumam apresentar um baixo desempenho computacional. Em diversas abordagens o tempo de execução de um algoritmo é essencial para o funcionamento e velocidade do sistema. A taxa de solução de erros será baixa dependendo da velocidade do algoritmo e de seu baixo desempenho. Nesse sentido, é necessário que os erros sejam corrigidos até que uma taxa admissível de erros seja atingida, impedindo assim o fluxo de novas entradas de dados (MOLYNEAUX, 2009).

1.3 Justificativa

Na área de telecomunicações existem algumas organizações que têm o papel de estabelecer padrões e protocolos de comunicação para os sistemas de transmissão e recepção de dados. Uma dessas organizações é a ETSI - *European Telecommunications Standards Institute*, considerada a Organização Europeia de Padrões.

A ETSI tem alguns padrões que são difundidos em todo o mercado, em diferentes áreas. Um desses padrões é de interesse deste trabalho, a saber o padrão de transmissão

de TV digital conhecido como DVB-S2X, que é uma extensão do padrão DVB-S2 (do ingles *Digital Video Broadcasting - Satellite - Second Generation*).

O padrão DVB ganhou mais importância com a implementação do padrão DVB-RCS, tornando o que era até então estações de recepção em estações de comunicação bidirecionais, resultando no fornecimento de serviços como TV interativa e de banda larga. O padrão DVB-S2X, com relação ao DVB-S2 apresenta novos esquemas de codificação de canais, níveis de proteção contra erros a diferentes componentes de serviço, flexibilidade estendida para lidar com outros formatos de dados de entrada e correção de erros baseada em códigos LDPC (do inglês *Low Density Parity Check*) concatenados com códigos BCH (do inglês *Bose–Chaudhuri–Hocquenghem*) (MARAL; BOUSQUET, 2002). "As correções de erros baseada em códigos LDPC concatenados com códigos BCH, permitem uma operação QEF(do inglês, Quasi Error-Free) a cerca de 0.7 dB a 1 dB do limite de Shannon"(MARAL; BOUSQUET, 2002).

O LDPC possui um algoritmo rápido capaz de recuperar informações mesmo em um canal com altas taxas de ruído, o que torna os códigos LDPC uma alternativa bem difundida para aplicações práticas e, portanto, um assunto bem conhecido na teoria de codificação e correção de erros (MATTOS, 2012).

Outro componente essencial para as comunicações via satélite são os dispositivos de processamento. No que se refere a comunicações moveis merecem destaque as plataformas de processamento baseado nos Arranjos de Portas Logicas Programáveis em Campo (FPGA - do inglês *Field Programmable Gate Array*) e os Circuitos Integrados de Aplicações Especıficas (ASIC - do inglês *Application specific integrated circuit*). Esses dispositivos apresentam entre si vantagens e desvantagens (LIMA, 2006).

Os dispositivos de processamento, são um dos maiores limitadores em um sistema de comunicação. Fazendo com que a escolha de qual hardware será utilizado seja de suma importância. Nessa escolha deve ser levada em consideração além dos recursos disponíveis em cada dispositivo, a finalidade do mesmo. Fatores como tempo de reconfiguração, capacidade de processamento, consumo de potencia, custo, programabilidade devem ser levadas em consideração (ISOMÄKI; AVESSTA, 2004).

O FPGA usa linguagens de descrição de hardware que permitem um desenvolvimento rápido e de baixo custo. Isso permite uma implementação em *hardware* dedicada que não limita o potencial dos algoritmos, o que diminui os custos do projeto e torna o processo mais rápido, viabilizando mais propostas de soluções. Trabalha a baixas frequências (100 a 300 MHz), mantendo o desempenho computacional dos algoritmos mapeados diretamente em *hardware* e reduzindo o consumo de energia. Atualmente, os kits de desenvolvimento vem acompanhados de sistemas em chip, que permitem implementar abordagens de particionamento de *hardware* e *software*, algo que flexibiliza as soluções. Possui uma implementação que permite explorar o paralelismo intrínseco dos algoritmos,

que proporciona o aumento do desempenho. Além de serem amplamente utilizadas para prototipagem de soluções baseadas em ASICs.

1.4 Objetivos

O presente trabalho visa o estudo e a implementação em linguagem de descrição de *hardware* VHDL(do inglês, VHSIC Hardware Description Language) do código corretor de erros LDPC, segundo o padrão DVB-S2X.

1.4.1 Objetivos específicos

- Implementação individual da arquitetura de *hardware* da Unidade Logica Aritmética (ULA) do LDPC;
- Implementação em FPGA da arquitetura de *hardware* do *SHIFT* do LDPC do padrão DVB-S2X.
- Desenvolvimento e verificação em VHDL da célula de processamento do LDPC;
- Desenvolvimento, implementação e verificação em FPGA de um modelo integrado da primeira varredura do LDPC usando blocos de BRAMs da XILIX.

1.5 Contribuições

Inserido em um projeto junto com a industria local, que tem como fim a criação de um Radio Definido por Software (RDS), este trabalho tem o proposito de disponibilizar um modelo de referencia em alto nível da implementação de parte da construção de um LDPC. Além disso, visa a realização de um estudo das arquiteturas em termos da máxima frequência de operação, do consumo de energia e do consumo de recursos, tais como *Look Up Tables*, *Flip Flops*, DSPs, blocos de RAM e multiplexadores. A realização de simulações, síntese e implementação dos sistemas. Constituindo parte de uma solução para o desenvolvimento de um radio definido por *software* (RDS) no padrão de comunicação DVB-S2X.

1.6 Organização do documento

A organização do trabalho se condiz em cinco capítulos e o restante do presente documento esta organizado da seguinte maneira. O segundo capítulo a Fundamentação Teórica, com a definição dos conceitos básicos para a compreensão do desenvolvimento. O terceiro capítulo é a Metodologia, com a explicação dos métodos aplicados que garante a

qualidade do mesmo e quais ferramentas foram utilizadas no desenvolvimento deste trabalho. No quarto capítulo se apresentam os resultados alcançados, um estudo comparativo das arquiteturas em termos da máxima frequência de operação, do consumo de energia e do consumo de recursos. O quinto e ultimo capítulo, traz as Conclusões, onde tem-se as discussões dos assuntos tratados, uma análise dos resultados obtidos e a proposta de trabalhos futuros.

2 Fundamentação teórica

2.1 Comunicação Satelital

A comunicação via satélite tem sido sinônimo de tecnologia e qualidade nas transmissões e recepções de qualquer tipo de informação. Alguns satélites de comunicação podem oferecer serviços de transmissão de dados em velocidades muito altas e a grandes distancias (BAPTISTA; MARINS, 2012). Com a redução do custo, tamanho e potência necessários e devido aos constantes avanços da microeletrônica, tornou-se possível criar equipamentos com uma estratégia de difusão mais sofisticada. Cada transmissão para a Terra pode ser focalizada em uma área geográfica pequena, de modo que múltiplas transmissões possam ocorrer simultaneamente. (SBIZERA, 2003)

Neste pensamento apresenta-se abaixo algumas vantagens e desvantagens deste tipo de comunicação. Com base em (SBIZERA, 2003), algumas vantagens da utilização de satélites a depender da aplicação, são:

- Grande largura de banda disponível;
- Facilidade de utilização em comunicação móvel;
- Custo efetivo: transmissões de dados via satélite a longas distâncias apresentam custo mais baixo que transmissões utilizando outros meios de comunicação como, por exemplo, o sistema de telefonia fixa ou celular. O custo de um canal independe da distância entre os pontos que integram a rede. A multiplexação dos dados permite a recuperação dos mesmos independentemente de sua localização geográfica;
- Alta disponibilidade;
- Cobertura de grandes áreas e de regiões não atendidas por sistemas terrestres devido a baixas densidades populacionais ou a dificuldades geográficas;
- Qualidade de transmissão;
- Flexibilidade de instalação e mudança de área;
- Superação de obstáculos naturais;
- Alta capacidade de transmissão *broadcast*.

Entre as desvantagens estão (SBIZERA, 2003):

- Atraso na propagação (mais comum aos satélites geoestacionários);
- Atenuação por chuva;
- Interferências solares e de outros sistemas;
- Congestionamento.
- Distorção não linear;
- Grande atenuação do espaço livre.

2.1.1 Arquitetura de um sistema de comunicação via satélite

Na figura 2 é possível verificar os principais itens existentes em um sistema de comunicação via satélite. O desenvolvimento do presente trabalho concentra-se no terminal de recepção e nos parâmetros de conexão.

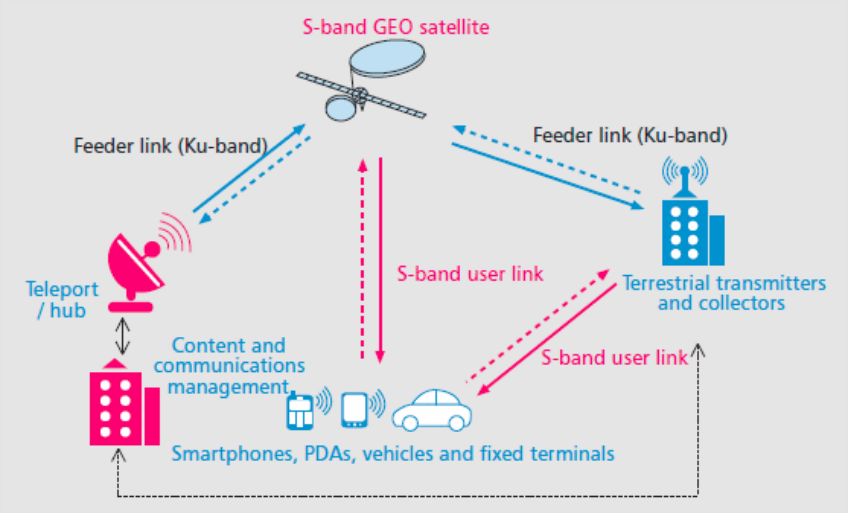


Figura 2 – Arquitetura de um sistema via satélite (SCALISE et al., 2013).

O terminal de recepção é divido em duas principais áreas, *front-end* que é relacionado aos aspectos de frequência e *back-end*, relacionado ao processamento do sinal (TUTTLEBEE, 2003).

O satélite realiza o envio do sinal que é recebido pelo *front-end*. Este é enviado pelo satélite em uma frequência elevada. O conversor Analógico - Digital (AD) tem certos limitantes que o impossibilita de cumprir com a devida conversão. Assim, é necessário determinados dispositivos de processamento para realizar o deslocamento dessa frequência que entra no conversor até um valor de trabalho, uma frequência intermediaria. Nessa frequência o sistema é capaz de realizar a conversão de valores analógicos para digitais. Após a digitalização, o sinal é processado pelo processador digital (BARROS, 2007.),

2.1.2 Módulos macro para um sistema de comunicação por satélite

O sistema de comunicação via satélite pode ser dividido em quatro partes principais, o satélite, o Front-End RF, Front-end digital e a plataforma de processamento. O satélite e o Front-End RF são os módulos responsáveis pelo processamento analógico enquanto o Front-End digital e a plataforma de processamento são responsáveis pelo processamento digital. Na figura 3 é possível ver o sistema completo que fica presente em terra.

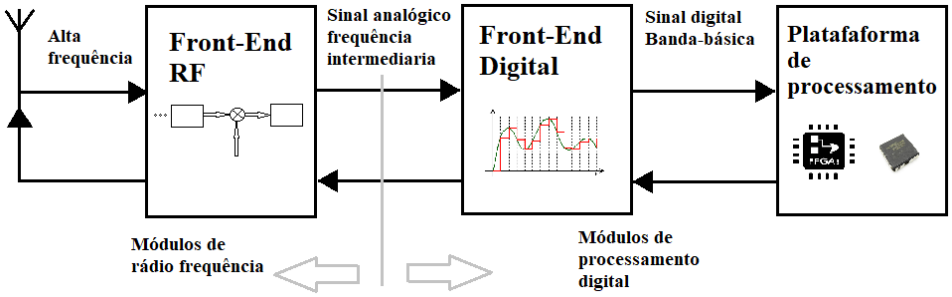


Figura 3 – Sistema completo parte terra.

2.1.3 Bandas de frequência para comunicação satelital

Os satélites geoestacionários operam em variadas faixas de frequência tais como as bandas C, X, Ku, K (BAPTISTA; MARINS, 2012). Na tabela 1 é possível ver as bandas existentes e suas referentes faixas de frequência (DUARTE; LEãO, 2002).

Banda	Faixa de frequência
Banda P	200-400 Mhz
Banda L	1530-2700 Mhz
Banda S	2700-3500 Mhz
Banda C	3700-4200 Mhz
	4400-4700 Mhz
	5725-6425 Mhz
Banda X	7900-8400 Mhz
Banda Ku1 (Banda PSS)	10.7-11.75 Ghz
Banda Ku2 (Banda DBS)	11.75-12.5 Ghz
Banda Ku3 (Banda Telecom)	12.5-12.75 Ghz
Banda Ka	17.7-21.2 Ghz
Banda K	27.5-31.0 Ghz

Tabela 1 – Bandas e suas respectivas faixas de frequência (DUARTE; LEãO, 2002).

A frequência tanto para o *uplink* (caminho percorrido pelas ondas eletromagnéticas até o satélite) quanto para o *downlink* (caminho das ondas do satélite) possuem frequências diferentes a fim de não causar interferência mútua (DUARTE; LEãO, 2002).

2.2 Front-end RF

As limitações tecnológicas dos conversores AD e DA (Digital - Analógico) dos sinais recebidos e transmitidos pelo rádio já são conhecidas. Neste contexto a introdução de um módulo que trata da radio frequência, entre a antena e o conversor, torna-se de grande importância no desenvolvimento de um RDS (ISOMÄKI; AVESSTA, 2004).

Em relação ao sinal recebido o módulo front-end RF deve amplificar o sinal, realizar a translação de frequência para uma frequência intermediária, realizar um controle do ganho e a filtragem anti-alias. Sendo assim, o conversor AD pode operar na frequência intermediária mais baixa, não comprometendo seu funcionamento devido às altas frequências e a provável oscilação de amplitude do sinal (ISOMÄKI; AVESSTA, 2004). O modulo macro do *Front-End* RF pode ser visto na figura 4.

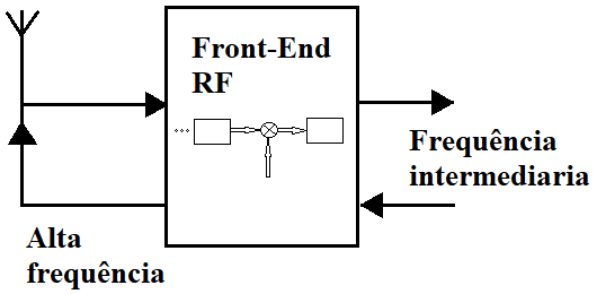


Figura 4 – Front-End RF.

A frequência intermediária (FI) deve ser suficientemente baixa de modo a atender as especificações do rádio, para não prejudicar o processamento do sinal (BARROS, 2007.). Outro aspecto relevante no projeto do RDS são as frequências imagem, as quais devem ser rejeitadas no processamento do sinal.

2.3 Front-end digital

O *front-end* digital é uma ponte entre processamento em rádio frequência e processamento em banda-base (BARROS, 2007.). O papel do front-end digital é digitalizar o sinal, amostrá-lo e passar o sinal para banda-base (TUTTLEBEE, 2003), tal como na figura 3.

2.4 Plataforma de processamento

Em um RDS, merecem destaque as seguintes plataformas de processamento de dados: (a) Processadores Digitais de Sinais (DSP); (b) Conjuntos de Portas Logicas Programáveis em Campo (FPGA); (c) Circuitos Integrados de Aplicações Especıficas (ASIC) (LIMA, 2006).

Na escolha do tipo de plataforma para processamento dos dados deve-se considerar os seguintes fatores: Recursos disponíveis em cada dispositivo, a finalidade de cada rádio, tempo de reconfiguração, capacidade de processamento, consumo de potencia, custo, programabilidade, entre outros (ISOMÄKI; AVESSTA, 2004).

Seja qual for o dispositivo adotado, deve sempre ser buscada uma maior capacidade de processamento para atender aos sistemas cada vez mais complexos, uma maior facilidade de desenvolvimento e uma melhor capacidade de reconfiguração (LIMA, 2006).

Todas essas plataformas apresentam vantagens e desvantagens, para uma escolha sensata é necessário pontuar pesos para as vantagens do projeto de aplicação. Na tabela 2 é possível ver de forma simplificada quais as maiores vantagens e desvantagens de cada dispositivo.

Característica	DSP	FPGA	ASIC
Frequência (MHz)	100 - 600	100– 300	>1000
Consumo de energia	Muito alta	Alto	Moderado
Execução paralela	Serial	Máxima (flexível)	Máxima
Complexidade	Programas complexos	Muito alta	Muito alta
Tamanho	Moderado	Muito grande	Grande
Evolução	Alta	Alta	Sem
Customização	Muito fácil	Fácil	Difícil
Verificação do projeto	Moderada	Moderada	Muito difícil
Ferramentas	Bom	Muito bom	Bom

Tabela 2 – Comparação entre DSP, FPGA e ASIC (HARUYAMA, 2002).

Os FPGAs são costumeiramente utilizados como ferramentas de prototipagem do ASICs. Sendo que o ASICs possuem como vantagem o fato de serem mais rápidos. Entretanto, não são reconfiguráveis, sendo voltados somente para a execução de tarefas fixas. Dessa forma as FPGAs atuam como um refinador do projeto. Por conta dos ASICs serem voltados para tarefas especıficas, em longa escala terminam sendo mais baratos (LIMA, 2006).

Os ASICs dentre todos apresentam o melhor desempenho enquanto os DSPs uma maior flexibilidade e como um meio termo destes tem-se os FPGAs (LIMA, 2006).

2.4.1 FPGA

FPGA são dispositivos lógicos programáveis (PLD). Consiste em um chip configurável, muito potente e que programa circuitos digitais. Trata-se de um arranjo de módulos, composto por portas lógicas programáveis e *flip-flops*, conhecido como blocos lógicos configuráveis que permitem as conexões dos módulos por meio de um protocolo simples (ZELENOVSKY; MENDONÇA, 2007). Tem o grande advento de ser reconfigurável e tem sido amplamente utilizada em termos de criação de protótipos.

Em sua formação o FPGA tem conjunto de blocos configuráveis que permite ao usuário a construção de circuitos complexos, e também em aplicações com circuitos formados a partir destes sendo possível considerar a partir da declaração de componentes em uma dada linguagem de programação, estes componentes podem compor circuitos com determinadas especificações de projeto e elevada complexidade.

2.5 Padrões de comunicação

Para uma comunicação via satélite existem técnicas que permitem que os sinais sejam transportados à distância, e adaptadas as características dos sinais transmitidos com consideração às restrições do canal. Essas restrições são basicamente sobre potencia e largura de banda e esses recursos podem ser trocados entre si. Este é um aspecto importante das comunicações via satélite pois a potencia impacta tanto no tamanho de satélites quanto o da estação terrestre, além da necessidade de respeitar largura de banda pelos limites das regulações. Nesses termos, o objetivo é alcançar um compromisso, visando minimizar o custo do sistema (MARAL; BOUSQUET, 2002).

2.6 Enlace de descida

O projeto de RDS em que o presente trabalho está inserido, tem em sua recepção o uso do padrão DVB-S2X. Esse padrão tem como alvo as principais áreas de aplicação do DVB-S2 *(Digital Video Broadcasting)*. Esses dois padrões protocolam sistemas de transmissão digital por satélite e utilizam sofisticadas técnicas de modulação e codificação, cobrindo de forma confiável satélites em todo o mundo. (ETSI, 2006) (ETSI, 2015)

Historicamente o DVB-S foi introduzido como um padrão em 1994 enquanto o DVB-DSNG em 1997. O padrão DVB-S especifica modulação QPSK, codificação convolucional e de canal *Reed-Solomon* concatenada, sendo amplamente usado pelas operadoras de satélites em todo o mundo para serviços de transmissão de dados e televisão. O DVB-DSNG especifica, além do formato DVB-S, o uso de modulação 8PSK e 16QAM para coleta de notícias via satélite, com o advento dessas novas tecnologias construiu-se um sistema de codificação de modulação e canal de "segunda geração"(denominado "DVB-S2") constituindo-se em um padrão único e flexível. (ETSI, 2006). As principais características do padrão DVB-S2X são as seguintes:

- Um poderoso sistema FEC baseado nos códigos LDPC concatenados com códigos BCH, que permitem uma operação quase sem erros entre 0,7 dB e 1 dB do limite de Shannon;

- 4 constelações, variando em eficiência de espectro de 2 bit/s/Hz a 5 bit/s/Hz, otimizadas para operação em traz uma ampla gama de taxas de código (de 1/4 até 9/10);
- um conjunto de três formas espectrais com fatores de roll-off 0.35, 0.25 e 0.20;
- Funcionalidade de Codificação e Modulação Adaptativa (ACM), otimizando a codificação de canal e modulação numa base quadro a quadro.

2.7 DVB-S2X

O padrão DVB-S2X reutiliza a arquitetura do sistema DVB-S2 conforme pode ser visto na figura 5, e são adicionadas etapas MODCOD mais refinadas, filtragem mais precisa, meios técnicos que permitem o fatiamento de tempo de sinais de banda larga e para colagem de múltiplos transpondes, entre outras tecnologias (ETSI, 2015).

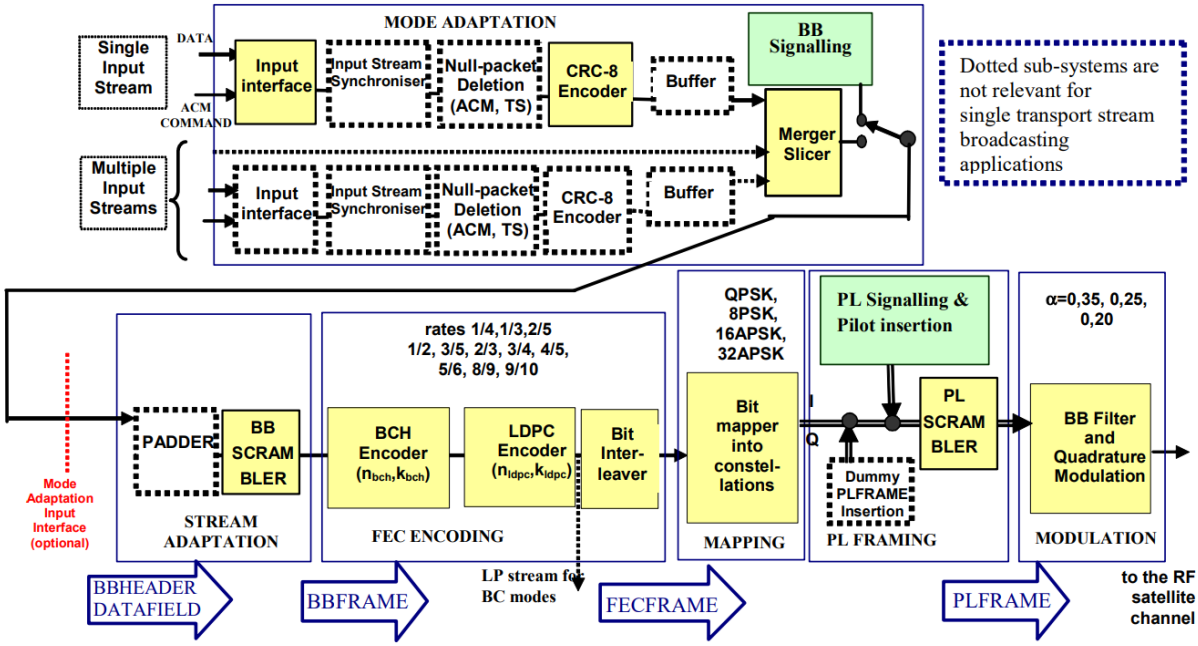


Figura 5 – Diagrama de blocos funcionais do sistema DVB-S2 (ETSI, 2006).

2.7.1 DVB e a sua importância

Vale ressaltar um pouco acerca do próprio DVB e sua importância. De acordo com o texto normativo do *Digital Video Broadcasting Project* (DVB) (ETSI, 2014), o DVB é um consórcio liderado pelo setor de emissoras, fabricantes, operadores de rede, desenvolvedores de software, órgãos reguladores, proprietários de conteúdo e outros, comprometidos com a criação de padrões globais para a prestação de serviços de dados e televisão digital. O DVB promove soluções voltadas para o mercado que atendem às necessidades e circunstâncias econômicas das partes interessadas e dos consumidores do setor de transmissão.

Tamanha é a importância desses padrões que os mesmos cobrem todos os aspectos da interface de transmissão da TV digital, do acesso condicional e interatividade para video, áudio e dados digitais em todo o mundo. E desde 1993 tem fornecido especificações globais de padronização (ETSI, 2014).

2.8 Base teórica para introdução ao LDPC

Em um sistema de comunicação se tem a tarefa de transmitir informações de uma extremidade a outra do sistema, a uma taxa e nível de confiabilidade aceitáveis. Na pratica, muitas vezes um esquema de modulação não é capaz de prove-lo, sendo a única opção pratica disponível o uso de codificação para correção de erros. O uso de codificadores também reduz a $\frac{E_b}{N_0}$ (BER - Bit Error Rate) necessária para uma taxa de erros fixa, o que proporciona uma redução na potencia transmitida necessária e os custos de *hardware* (HAYKIN, 2004).

O FEC (do inglês, Forward Error Correction) do padrão DVB-S2X é baseado nos códigos LDPC concatenados com códigos BCH, que permitem uma operação muito próxima do limite de Shannon, na figura 6 é exibido como ocorre a comunicação com essa abordagem. Tem-se uma fonte de símbolos binários. O codificador de canal recebe os dados e adiciona redundância de acordo com uma regra predefinida. Isso permite que na recepção o decodificador explore a redundância para decidir quais *bits* apresentam erros. Essa atuação de codificador e decodificador com a adição de redundância diminui o efeito de ruido do canal, fazendo com que o numero de erros também diminua (HAYKIN, 2004) (ETSI, 2006).

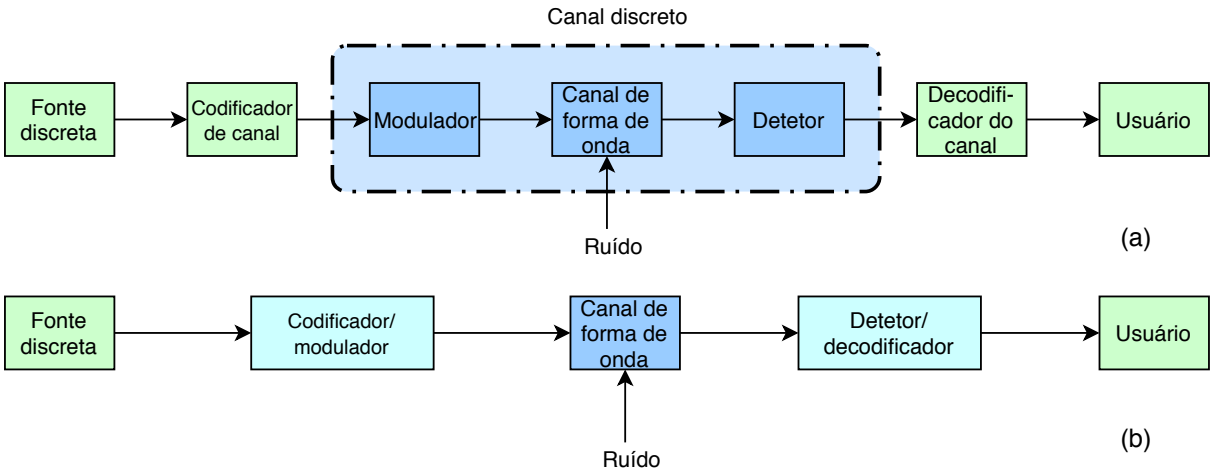


Figura 6 – Modelo simplificado de comunicação digital. (a) Codificação e modulação separadamente. (b) Codificação e modulação combinados (HAYKIN, 2004).

2.8.1 Códigos de blocos lineares

O LDPC é considerado uma classe de códigos de bloco linear. Um código é linear quando quaisquer duas palavras-códigos somadas em aritmética modulo de 2 produzem uma terceira palavra-código. Esses tem a habilidade de detectar e corrigir erros causados por ruídos no canal de comunicação e são amplamente empregados nos sistemas de comunicação comerciais (HAYKIN, 2004) (LATHI; DING, 2009).

> Segundo (HAYKIN, 2004, p. 660), um código de blocos linear *(n,k)*, no qual *k bits* dos *n bits* do código sempre são idênticos à sequencia de mensagem a ser transmitida. Os *n-k bits* na fração restante são computados a partir dos *bits* de mensagem de acordo com uma regra de codificação predefinida que determina a estrutura matemática do código. Consequentemente, estes *n-k bits* são chamados *bits* de verificação da paridade generalizados ou simplesmente bits de paridade.

Em um bloco com k *bits*, tem-se 2^k blocos de mensagens distintas. Essa sequencia de *bits* aplicada a um codificador produz uma palavra-código de n *bits*, sendo *(n-k)* os *bits* de paridade. Assim, tem-se um código que possui estrutura sistemática que é dividida em duas partes, uma com os *bits* de paridade e a outro pelos de mensagem (HAYKIN, 2004) (LATHI; DING, 2009). Na figura 7, o formato da palavra-código.

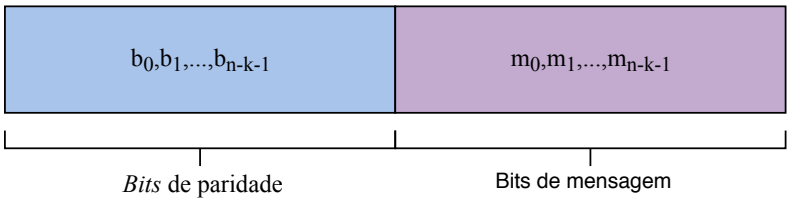


Figura 7 – Estrutura da palavra código sistemática (HAYKIN, 2004).

Num código sistemático, os k dígitos de uma palavra-código são os dígitos de dados(ou informação) e os restantes dígitos $m = n - k$ são os dígitos de verificação de paridade, formados por combinações lineares de dígitos de dados (LATHI; DING, 2009). Com base na figura 7, tem-se que o formato da palavra-código para um código sistemático pode ser tido como a relação 2.1 (HAYKIN, 2004).

$$c_i = \begin{cases} b_i, & i = 0, 1, ..., n-k-1 \\ m_{i+k-n}, & i = n-k, n-k+1, ..., n-1 \end{cases} \tag{2.1}$$

Pela relação, tem-se a forma generalizada,

$$b_i = p_{0,i}m_0 + p_{1,i}m_1 + ... + p_{k-1,i}m_{k-1} \tag{2.2}$$

Onde os coeficientes são definidos como,

$$p_{ij} = \begin{cases} 1, & \text{se } b_i \text{ depender de } m_j \\ 0, & \text{caso contrário} \end{cases} \tag{2.3}$$

É possível identificar nas equações 2.2 e 2.3 que p_{ij} é a relação de conexões, e são escolhidas de forma que as linhas da matriz geradora seja linearmente independente e as equações de paridade únicas. Usando notação matricial essas relações ficam da seguinte maneira (HAYKIN, 2004) (LATHI; DING, 2009) (SANTOS et al., 2014):

$$m = [m_0, m_1, ..., m_{k-1}] \tag{2.4}$$
$$b = [b_0, b_1, ..., b_{n-k-1}] \tag{2.5}$$
$$c = [c_0, c_1, ..., c_{n-1}] \tag{2.6}$$

Considerando as notações em 2.4 e 2.5, a reescrita da equação 2.3 de forma compacta, é

$$b = mP \tag{2.7}$$

Onde P é a matriz com coeficiente k, definida por

$$P = \begin{bmatrix} P_{00} & P_{01} & ... & P_{0,n-k-1} \\ P_{10} & P_{11} & ... & P_{1,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ P_{k-1,0} & P_{k-1,1} & ... & P_{k-1,n-k-1} \end{bmatrix}, \tag{2.8}$$

onde P_{ij} é 0 ou 1. Pelas definições c pode ser escrito como vetores verticais, equação 2.9. Substituindo as equações 2.7 e considerando a equação 2.9, obtém-se a equação 2.10, onde I_k é a matriz identidade com tamanho k.

$$c = [b \ \vdots \ m] \tag{2.9}$$
$$c = m[P \ \vdots \ I_k] \tag{2.10}$$

Sendo $G = [P \vdots I_k]$, substituindo em 2.10 tem-se a equação 2.11.

$$c = mG \tag{2.11}$$

Assim c denominam-se as palavras-código, o conjunto das palavras-código é denominado como código, e é gerado por meio da equação 2.11. A matriz G denomina-se como a matriz geração de paridade. A matriz de verificação de paridade nasce da transposta de P, e pode ser vista na equação 2.12 (HAYKIN, 2004) (LATHI; DING, 2009).

$$H = [I_{n-k} \quad \vdots \quad P^T] \tag{2.12}$$

As equações geradoras e de verificação de paridade são fundamentais para a operação de códigos de bloco linear, sendo importantes para a compreensão e funcionamento do LDPC (HAYKIN, 2004). O modelo diagramático de como funciona a geração e detecção pode ser visto na figura 8.

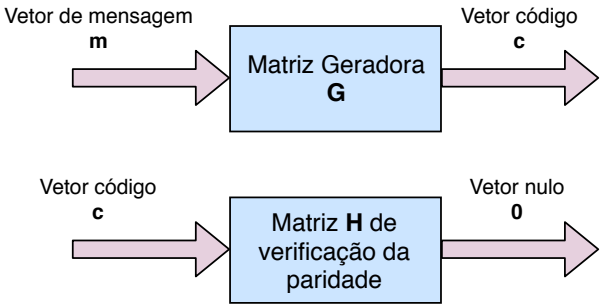


Figura 8 – Diagrama de blocos da equação geradora e da equação de verificação de paridade (HAYKIN, 2004).

2.8.2 Matriz geradora G e matriz de verificação de paridade H

Uma representação para a base de palavras-código *(n,k)* de um determinado código c é $g_0, g_1, ..., g_{k-1}$. Sua matriz G pode ser definida como (MATTOS, 2012),

$$G = \begin{bmatrix} g_0 \\ \vdots \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{10} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{2.13}$$

A relação presente em 2.13 apresenta a matriz geradora para um código gerador c com blocos de dados de k símbolos (MATTOS, 2012). A matriz G gera por meio da equação 2.11 as palavras-código. Na equação 2.12 tem-se como a matriz de verificação de paridade é gerada, por meio da transposta de P e a ligação que está tem com a matriz geradora $G = [P \vdots I_k]$. Dessa forma, a matriz de verificação de paridade pode ser vista na

relação 2.14.

$$H = \begin{bmatrix} h_0 \\ \vdots \\ \vdots \\ h_{k-1} \end{bmatrix} = \begin{bmatrix} h_{00} & h_{0,1} & ... & h_{0,n-1} \\ h_{10} & h_{1,1} & ... & h_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & ... & h_{n-k-1,n-1} \end{bmatrix} \tag{2.14}$$

A matriz verificadora tem tal importância que o vetor c é uma palavra código somente se $cH^T = 0$ (MATTOS, 2012).

2.8.3 Distancia mínima

Entre dois vetores código há uma distancia que os difere entre si, sendo essa distancia definida com o numero de localizações. Essa é conhecida como distancia de *Hamming* e simbolizada por $d(c_1, c_2)$ onde c_1 e c_2 são os vetores-código. O peso de *Hamming* de um vetor-código é o numero de elementos diferentes de zero que de forma semelhante é a distancia entre um certo vetor e o vetor que possui apenas zeros(HAYKIN, 2004) (MATTOS, 2012).

A distancia miníma d_{min} é definida como a menor distancia *Hamming* que equivalentemente é o menor peso de *Hamming* entre qualquer par de vetores código. A distancia mínima d_{min} se relaciona de maneira fundamental com a estrutura da matriz H de verificação de paridade do código. Por meio de manipulações matemáticas é possível provar que $cH^T = 0$, disso sabe-se que um código de bloco linear é definido pelo conjunto de todos os vetores código que obedecem essa relação (HAYKIN, 2004).

A distancia miníma de um código de bloco linear determina a capacidade de correção de erro do código. Para detectar e corrigir todos os padrões de erro é necessário um código bloco linear *(n,k)*, no caso de um peso de *Hamming* menor ou igual ao raio de correção t, onde um vetor de código c_i foi transmitido, porém o valor recebido foi $r = c_i + e$. Deve-se exigir que a saída do decodificador seja $\hat{c} = c_i$, sempre que $w(e) \leq t$. Assim, a melhor estrategia é usar o vetor código mais próximo do vetor r recebido, dessa forma, usa-se a menor distancia de *Hamming* de *d(c_i, r)*(HAYKIN, 2004) (MATTOS, 2012).

Assim a estrategia deve ser tal que a distancia de *Hamming* *d(c_i, r)* $\leq t$ e $d(c_i, c_j) \geq 2t+1$. Como pode ser visto na figura 9a, o vetor mais próximo do vetor r é o vetor código c_i pois se tem certeza que c_j está à uma distancia maior ou igual a $2t$ de c_i. Já na figura 9b, não se tem um vetor mais próximo podendo ocorrer do decodificador decidir por c_j propagando um erro (HAYKIN, 2004).

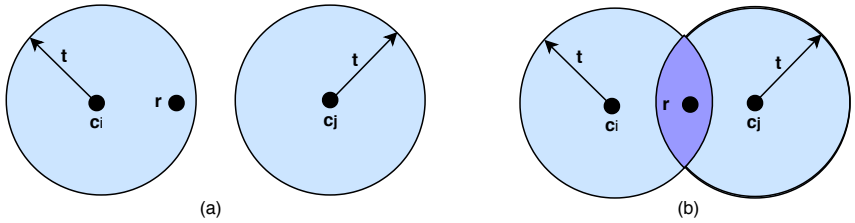


Figura 9 – (a)Distancia de *Hamming* $d(c_i, c_j) \geq 2t + 1$. (b) Distancia de *Hamming* $d(c_i, r) \leq t$ (HAYKIN, 2004).

2.9 Codigos de verificação de paridade de baixa densidade - LDPC

Os códigos de verificação de paridade de baixa densidade (LDPC) pertencem a uma ampla família de técnicas de codificação para controle de erros chamadas códigos compostos. O LDPC apresenta algumas importantes vantagens em relação aos códigos turbo, ausência de palavras-código de baixo peso e decodificação iterativa de menor complexidade. Quanto à questão de palavras-código de baixo peso considerasse que em um código turbo um certo numero de palavras-código estão indesejavelmente próximas. Isso permite que o ruido de canal faça com que a as palavras-código sejam confundidas com suas vizinhas. A não existência de palavras-código de baixo peso permite que o LDPC possa ser facilmente construído de forma a obter taxas de erro de *bits* extremamente pequenas (HAYKIN, 2004). O peso da palavra código é o número de coordenadas não nulas do bloco (MATTOS, 2012).

A segunda vantagem condiz na complexidade computacional, enquanto um decodificador turbo opera sobre treliça do código convolucional, o numero de computações em cada recursão do algorítimo se dimensiona linearmente com o numero de treliças, normalmente treliças com 16 estados ou mais. Os códigos LDPC, por sua vez, usam uma única treliça de verificação da paridade simples que tem apenas dois estados. O que permite que os códigos LDPC sejam significativamente mais simples. Além do grande advento de ser "paralelizável", que permite sua execução em maiores velocidades. Entretanto, para grandes blocos, a complexidade de codificação do LDPC é maior quando comparado (HAYKIN, 2004).

O nome LDPC vem da característica de possuírem uma Matriz de Verificação de paridade H esparsa, isto é, a matriz H possui poucos 1's em comparação à quantidade de 0's. Esta característica advém do fato de estes códigos possuírem uma grande distância mínima, ausência de palavras-código de baixo peso, e consequentemente uma baixa probabilidade de erro (MATTOS, 2012).

2.9.1 Códigos LDPC

Códigos LDPC são especificados por uma matriz de verificação de paridade, que é esparsa, consistindo apenas de 0's e poucos 1's. Para a caracterização, tem-se que n indica o tamanho de blocos, t_c o peso em cada coluna da matriz e t_r o peso de cada linha da matriz esparsa, com $t_r > t_c$. A taxa do LDPC com esse formato poder ser medida como na equação 2.15 (HAYKIN, 2004).

$$r = 1 - \frac{t_c}{t_r} \tag{2.15}$$

Isso pode ser justificado, de forma que se admita que ρ indique a densidade da matriz esparsa é possível definir que $t_c = \rho(n-k)$ e $t_r = \rho(n)$. Onde (n-k) é justamente o numero de linhas na matriz esparsa A e n o numero de colunas. Relacionando essas equações, obtém-se $\frac{t_c}{t_r} = 1 - \frac{k}{n}$, por definição a taxa de código de um código de blocos é $\frac{k}{n}$, validando o resultado da equação 2.15. Para a matriz deve-se garantir que as linhas sejam linearmente independentes (HAYKIN, 2004).

Para motivos de maior compreensão as figuras *grafos bipartites* são utilizadas, essas conseguem uma ótima retratação dos códigos LDPC. Para um caso com $n = 10$, $t_c = 3$ e $t_r = 5$, tem-se seu grafo na figura 10. Os nós a esquerda, são os nós de variáveis e correspondem a elementos de palavra-código. Os nós a direita, são nós de verificação, os quais correspondem ao conjunto de restrições de verificação de paridade. Esse tipo de LDPC é regular, porque todos os nós são similares, apresentando o mesmo grau. Para esse grafo o tamanho de bloco dos nós de variáveis é $t_c = 3$, e o grau dos nós de verificação é $t_r = 5$. À medida que n se aproxima do infinito, cada nó de verificação é conectado a uma fração cada vez menor de nós de variáveis, daí o termo baixa densidade (HAYKIN, 2004).

A matriz A de verificação de paridade do LDPC não é sistemática, não tem *bits* de verificação da paridade que aparecem na forma diagonal. Mesmo diante disso é possível se obter uma matriz geradora G, por meio da eliminação *gaussiana* em aritmética com módulo 2. Considerando o que foi iniciado em Códigos de blocos lineares, 2.8.1, o vetor código c é particionado como $c = [b \vdots m]$, onde m é o vetor mensagem e b, o vetor paridade. Dessa forma, a matriz A de verificação de paridade pode ser particionada tal como na equação 2.16 (HAYKIN, 2004).

$$A^T = \begin{bmatrix} A_1 \\ \dots \\ A_2 \end{bmatrix} \tag{2.16}$$

Em 2.16, a matriz A_1 é quadrada com dimensões $(n-k)\times(n-k)$, e A_2 é uma matriz retangular de dimensões $k \times (n-k)$, dessa forma obtém-se a equação 2.17 e resolvendo

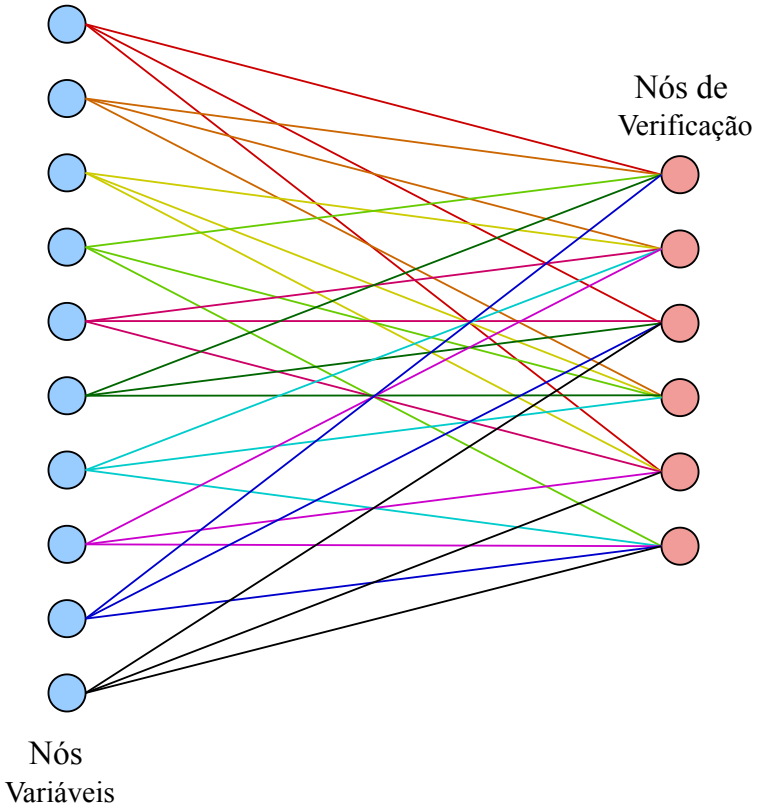


Figura 10 – Grafo bipartite do código LDPC(10,3,5) (HAYKIN, 2004).

obtém-se 2.18 (HAYKIN, 2004).

$$[b \quad \vdots \quad m] \begin{bmatrix} A_1 \\ \dots \\ A_2 \end{bmatrix} = 0 \tag{2.17}$$

$$bA_1 + mA_2 = 0 \tag{2.18}$$

Os vetores m e b se relacionam por $b = mP$, onde P é a matriz coeficiente. Substituindo em 2.18, tem-se a equação 2.19, onde é facilmente visto que para qualquer vetor mensagem m diferente de zero, a matriz coeficiente de códigos LDPC é satisfeita. Resolvendo para P, obtém-se a equação 2.20. Sendo a matriz geradora do LDPC definida como $G = [P \quad \vdots \quad I_k]$, pela relação presente em 2.20, encontra-se a equação 2.21 (HAYKIN, 2004).

$$PA_1 + A_2 = 0 \tag{2.19}$$

$$P = A_2 {A_1}^{-1} = 0 \tag{2.20}$$

$$G = [A_2 {A_1}^{-1} \vdots I_k] \tag{2.21}$$

2.9.1.1 Considerações

Considerando o modelo LDPC(10,3,5) enunciado anteriormente cujo o grafo esta presente na figura 10, a matriz de verificação de paridade é definida por 2.22. Mesmo que pareça aleatória, mantém as restrições quanto à regularidade presente no grafo (HAYKIN, 2004).

$$A = \underbrace{\left[\begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right.}_{A_1{}^T} \vdots \underbrace{\left.\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right]}_{A_2{}^T} \tag{2.22}$$

Particionando a matriz em suas partes, obtém-se as matrizes 2.23 e 2.24 (HAYKIN, 2004).

$$A_1 = \left[\begin{array}{cccccc} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{array}\right] \tag{2.23}$$

$$A_2 = \left[\begin{array}{cccccc} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \tag{2.24}$$

Devendo agora fazer o inverso de A_1, pode ser visto na equação 2.25 e o produto $A_2{A_1}^{-1}$ em 2.26. E por fim o gerador LDPC(10,3,5) na equação 2.27. Esses passos ilustram o procedimento envolvido na geração do código (HAYKIN, 2004). Com referencia ao modelo de LDPC deste trabalho, tem-se um tamanho de bloco muito maior que este.

$${A_1}^{-1} = \left[\begin{array}{cccccc} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{array}\right] \tag{2.25}$$

$$A_2{A_1}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.26}$$

$$G = \underbrace{\left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{array}\right.}_{A_2{A_1}^{-1}} \vdots \underbrace{\left.\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]}_{I_k} \tag{2.27}$$

2.9.2 Distancia mínima de códigos LDPC

A análise algébrica do LDPC é difícil, sendo que este tem um grande tamanho de bloco, cerca de 10^3 a 10^6, isso faz com que o numero de palavras-código também seja grande. Dessa forma, é muito mais produtivo a realização de uma analise estatística, onde pode se encontrar informações acerca de suas propriedades, tal como a distancia mínima dos códigos, que é a menor distancia de *Hamming* (HAYKIN, 2004).

Como mostrado, a medida que aumenta-se o tamanho de bloco n, para $t_c \geq 3$ e $t_r > t_c$ fixos, a distribuição de probabilidade pode ser delimitada por uma função próxima da função degrau unitária a uma fração fixa $\Delta_{t_c t_r}$ do bloco n. Para um n grande, praticamente todos os códigos LDPC do conjunto tem uma distancia miníma de pelo menos $n\Delta_{t_c t_r}$ (HAYKIN, 2004).

2.9.3 Código LDPC e uma consideração probabilística do Algoritmo SPA (Sum Product Algorithm)

O SPA é a forma mais comum de decodificar dados a partir de um algorítimo LDPC, esse apresenta grande eficiência, sendo um algoritmo de decodificação iterativo baseado na propagação de crença. Para melhor entendimento, o SPA pode ser visto como um jogo de gangorra, considera-se duas etapas ora um lado sobe e o outro desce, outrora o contrario. No SPA na primeira etapa, cada nó de variável(CN - *check node*) passa informações por meio de suas arestas para os nós de verificação conectados no fluxo de passagem de cima para baixo. Na próxima etapa, cada nó de verificação(CN - *check node*) retorna as informações para todos os nós de variáveis aos quais estão conectados em um fluxo de passagem de baixo para cima (LATHI; DING, 2009). Esses nós podem ser visualizados na figura 10, os nós de verificação são conectados aos nós de variáveis por meio de arestas.

No processo de decodificação considera-se a matriz de paridade H, que já foi enunciada nas subseções anteriores, aonde H tem tamanho $J \times n$, e $J = n - k$. Para um LDPC com código de bloco (n,k), a palavra-código é representada pelos bits dos nós de variável $v_j, j = 1, ..., n$, assim, para o j-ésimo nó de variável v_j, tem-se a relação presente em 2.28, essa equação denota um grupo de nós de paridade conectados a v_j. Para o i-ésimos nó de verificação z_i tem-se a equação 2.29, essa denota o grupo de nós de variáveis conectados a z_i (LATHI; DING, 2009).

$$\mu_j = \{i : h_{ij} = 1, 1 \leq i \leq J\} \tag{2.28}$$

$$\sigma_i = \{j : h_{ij} = 1, 1 \leq j \leq n\} \tag{2.29}$$

A probabilidade de satisfazer o nó de verificação $z_i = 0$ quando $v_j = u$ é definida na equação 2.30, denotando o vetor de *bits* de variável como $\mathbf{v}$, pelo teorema de Bayes obtem-se a equação 2.31, onde $R_{i,j}$ é a mensagem passada dos nós de verificação para os nós de variável (LATHI; DING, 2009).

$$R_{i,j}(u) = P[z_i = 0 | v_j = u] \qquad u = 0, 1 \tag{2.30}$$

$$\begin{aligned} R_{i,j}(u) &= \textstyle\sum_{\mathbf{v}:v_j=u} P[z_i = 0|\mathbf{v}] \cdot P[\mathbf{v}|v_j = u] \\ &= \textstyle\sum_{v_l : l \in \sigma_i, l \neq j} P[z_i = 0 | v_j = u, \{v_l : l \in \sigma_i, l \neq j\}] \cdot P[\{v_l : l \in \sigma_i, l \neq j\} | v_j = u] \end{aligned} \tag{2.31}$$

Quanto aos nós de verificação z_i, para estimar a probabilidade $P[\{v_l : l \in \sigma_i, l \neq j\}|v_j = u]$, os nós de verificação z_i coletam informações do conjunto de nós de variável σ_i. Então define-se a probabilidade $v_l = x$ obtida dos nós de verificação com exceção do i-ésimo, o que pode ser visto na equação 2.34 (LATHI; DING, 2009).

$$Q_{i,l}(x) = P[v_i = x | \{z_m = 0 : m \in \mu_l, m \neq i\}] \qquad u = 0, 1 \tag{2.32}$$

Além disso, e assumindo que a probabilidade dos nós de variáveis são aproximadamente independentes (LATHI; DING, 2009), É possível estimar:

$$P[\{v_l : l \in \sigma_i, l \neq j\} | v_j = u] = \prod_{l \in \sigma_i, l \neq j} Q_{i,l}(v_l) \tag{2.33}$$

Isso significa que os nós de verificação podem atualizar as mensagens por meio de (LATHI; DING, 2009):

$$R_{i,j}(u) = \sum_{v_l : l \in \sigma_i, l \neq j} P[z_i = 0 | v_j = u, \{v_l : l \in \sigma_i, l \neq j\}] \cdot \prod_{l \in \sigma_i, l \neq j} Q_{i,l}(v_l) \tag{2.34}$$

A probabilidade de $P[z_i = 0|v_j = u, \{v_l : l \in \sigma_i, l \neq j\}]$ é 0 ou 1. Isto é, o conjunto de nós de verificação $Z_i = 0$ pode ter sucesso ou falha. A equação 2.34 permite $R_{i,j}(u)$ ser atualizado quando o i-ésimo nó de verificação receber $Q_{i,j}(v_l)$. Com a atualização de $R_{i,j}(u)$, este poderá passar para os nós de variável a mensagem necessária para atualizar $Q_{i,j}(x)$. Então novamente usando o teorema de Bayes, obtém-se a equação 2.35 (LATHI; DING, 2009).

$$Q_{i,l}(x) = \frac{P[v_l = x|\{z_m = 0 : m \in \mu_l, m \neq i\}|v_l = x]}{P[v_l = x|\{z_m = 0 : m \in \mu_l, m \neq i\}]} \tag{2.35}$$

Dessa forma novamente assumindo que as verificadores de paridade são independentes, obtém-se a equação 2.36 (LATHI; DING, 2009).

$$P[\{z_m = 0 : m \in \mu_l, m \neq i\}|v_l = x] \cdot \prod_{m \in \mu_l, m \neq i} R_{m,l}(x) \tag{2.36}$$

Definindo a probabilidade do bit de variável como $P_l(x) = P(v_l = x)$. Considerando $\alpha_{i,l}$ fator de normalização tal que $Q_{i,l}(1) + Q_{i,l}(0) = 1$. É possível atualizar $Q_{i,l}(x)$ nos nós de variável baseado na equação 2.34, isso se revela na equação 2.37 (LATHI; DING, 2009).

$$Q_{i,l}(x) = \alpha_{i,l} \cdot p_l(x) \prod_{m \in \mu_l, m \neq i} R_{m,l}(x) \tag{2.37}$$

Esta mensagem será passada de volta para os nós de verificação. Está comunicação pode ser vista na figura 11, tal como ocorre. Na figura $R^k_{i,j}$ representa a mensagem enviada do nó de verificação para o nó de variável, enquanto $Q^{k+l}_{i,j}$ é a mensagem enviada do nó de variável para o nó de verificação (LATHI; DING, 2009).

Para um maior entendimento é possível sumarizar o SPA, a seguir tem-se os passos que devem ser tomados em ordem (LATHI; DING, 2009):

Inicialização: Considerando $m = 0$ e um m_{max} tal que seja o numero de iterações. Para todo $h_{i,l} = 1$ em $\mathbf{H}$, usa-se as probabilidades anteriores para definir as equações 2.38.

$$Q^0_{i,l}(1) = P_l(1) \tag{2.38a}$$

$$Q^0_{i,l}(0) = P_l(0) \tag{2.38b}$$

Passo 1: O nó de verificação i atualiza as informações, equações 2.39.

$$R_{i,j}(1) = \sum_{v_l : l \in \sigma_i, l \neq j} P[z_i = 0|v_j = 1, \{v_l\}] \cdot \prod_{l \in \sigma_i, l \neq j} Q^{(m)}_{i,l}(v_l) \tag{2.39a}$$

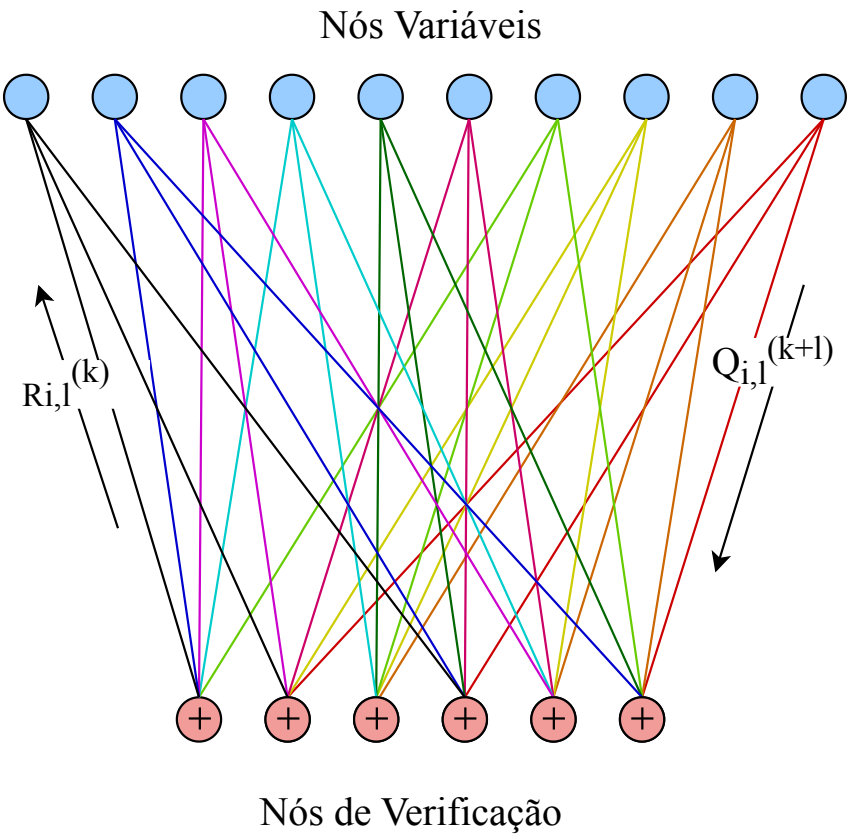


Figura 11 – Troca de mensagens no SPA (LATHI; DING, 2009).

$$R_{i,j}(0) = \sum_{v_l : l \in \sigma_i, l \neq j} P[z_i = 0 | v_j = 0, \{v_l\}] \cdot \prod_{l \in \sigma_i, l \neq j} Q_{i,l}^{(m)}(v_l) \tag{2.39b}$$

Passo 2: Para cada nó de variável indexado por l, atualiza com as equações 2.40.

$$Q_{i,j}^{(m+1)}(0) = \alpha_{i,l}^{(m+1)} \cdot p_l(0) \prod_{m \in \mu_l, m \neq j} R_{i,l}^{(m)}(0) \tag{2.40a}$$

$$Q_{i,j}^{(m+1)}(1) = \alpha_{i,l}^{(m+1)} \cdot p_l(1) \prod_{m \in \mu_l, m \neq j} R_{i,l}^{(m)}(1) \tag{2.40b}$$

Onde o fator de normalização $\alpha_{i,l}^{(m+1)}$ é selecionado segundo a equação 2.41.

$$Q_{i,j}^{(m+1)}(0) + Q_{i,j}^{(m+1)}(0) = 1 \tag{2.41}$$

Passo 3: Nos nós de variável, estimar as probabilidades a posteriori. É possível ver nas equações 2.42.

$$p^{(m+1)}[v_l = 0 | \mathbf{r}] = \alpha_{i,l}^{(m+1)} \cdot p_l(0) \prod_{m \in \mu_l} R_{i,l}^{(m)}(0) \tag{2.42a}$$

$$p^{(m+1)}[v_l = 1|\mathbf{r}] = \alpha_{i,l}^{(m+1)} \cdot p_l(1) \prod_{m \in \mu_l} R_{i,l}^{(m)}(1) \tag{2.42b}$$

Onde o fator de normalização $\alpha_l^{(m+1)}$ é selecionado segundo a equação 2.43.

$$p^{(m+1)}[v_l = 0|\mathbf{r}] + p^{(m+1)}[v_l = 1|\mathbf{r}] = 1 \tag{2.43}$$

Passo 4: Realiza a difícil decisão de qual o bit de código 2.44.

$$\hat{v}_l = sign \left\{ \log \frac{p^{(m+1)}[v_l = 1|\mathbf{r}]}{p^{(m+1)}[v_l = 0|\mathbf{r}]} \right\} \tag{2.44}$$

Nesse momento se a palavra-código satisfazer todos as verificações de paridade, para-se a decodificação. Em outro caso, retorna-se ao passo 1 para uma outra iteração.

2.9.4 Algoritmos *Sum-Product* e *Min-Sum*

Na seção 2.9.3 foi trazido de forma probabilística acerca do funcionamento do algoritmo SPA, a mesma traz um teor matemático complexo a se considerar em analises mais generalizadas para a construção do código. Nesta seção será enunciado os mesmo passos tomadas anteriormente com uma notação mais simplista e de mais fácil entendimento e compreensão, além de ser enunciado acerca do algoritmo Min-Sum que será o utilizado na construção do LDPC neste texto.

Antes é necessário que se introduza uma outra notação, para tal L_n é a priori informação do bit de nó, $\overline{L_n}$ a informação posterior do bit de nó, $E_{m,n}$ e a verificação para o bit mensagem de m para n e $F_{n,m}$ o bit para a mensagem de verificação de n para m. O SPA pode ser sumarizado como (ISLAM et al., 2011):

Passo 1: Inicialização

1 - Recebe-se a priori informação, $L_n = -r_n$; 2 - O bit para verificação de mensagem inicializa, $F_{n,m} = L_n$.

Passo 2: Passo horizontal

Processa-se o nó de verificação (ISLAM et al., 2011):

$$E_{m,n} = \log \frac{1 + \prod_{n' \in N(m) \backslash n} \tanh\left(\frac{F_{n',m}}{2}\right)}{1 - \prod_{n' \in N(m) \backslash n} \tanh \frac{F_{n',m}}{2}} \tag{2.45}$$

Passo 3: Passo vertical

A posteriori informação (ISLAM et al., 2011):

$$\overline{L_n} = L_n + \sum_{m \in M(n)} E_{m,n} \tag{2.46}$$

O bit para a mensagem de verificação (ISLAM et al., 2011):

$$F_{m,n} = \overline{L_n} + \sum_{m' \in M(n) \setminus m} E_{m,n} \tag{2.47}$$

Passo 4: Limiar de decodificação

$$\overline{c_n} = \begin{cases} 0, & \text{se } \overline{L_n} > 0 \\ 1, & \text{caso contrário} \end{cases} \tag{2.48}$$

Assim se $\overline{c_n} = 0$, o algorítimo para e o resultado da decodificação é validade. Caso não, uma nova iteração é iniciada. Caso sejam necessárias novas iterações essas continuaram até um limite máximo pré definido (ISLAM et al., 2011).

2.9.4.1 Min-sum

O algoritmo min-sum é na verdade uma modificação especifica do SPA, essa modificação reduz a complexidade da implementação do decodificador. A alteração se condiz em alterar no passo horizontal a equação 2.45 utilizando a relação $2\tanh^{-1} p = \log \frac{1+p}{1-p}$, dessa forma passa-se a ter a equação 2.49 (ISLAM et al., 2011).

$$E_{m,n} = 2\tanh^{-1} \prod_{n' \in N(m) \setminus n} \tanh \frac{F_{n',m}}{2} \tag{2.49}$$

A equação 2.49 pode ser modificada, de tal forma que se obtém a equação 2.50 (ISLAM et al., 2011).

$$E_{m,n} = 2\tanh^{-1} \prod_{n' \in N(m) \setminus n} sgn(F_{n',m}) \prod_{n' \in N(m) \setminus n} \tanh \frac{|F_{n',m}|}{2} \tag{2.50a}$$

$$E_{m,n} = \prod_{n' \in N(m) \setminus n} sgn(F_{n',m}) 2\tanh^{-1} \prod_{n' \in N(m) \setminus n} \tanh \frac{|F_{n',m}|}{2} \tag{2.50b}$$

Então o algoritmo min-sum usa a simplificação da equação 2.50b, relaciona os termos similares e reconhece que o menor termo $F_{n',m}$ domina o produtório, assim o produtório pode ser aproximado pelo mínimo do termo dominante. E a equação passa a ter a forma da equação 2.51 (ISLAM et al., 2011).

$$E_{m,n} = \prod_{n' \in N(m) \setminus n} sgn(F_{n',m}) min_{n' \in N(m) \setminus n} |F_{n',m}| \tag{2.51}$$

2.10 Construção do LDPC e sua implementação para o padrão DVB-S2X

Trazendo para o modelo especifico do DVB-S2X, para a consideração de como se relaciona a matriz de paridade, vale considerar um exemplo de codificação de LDPC com palavra-código de tamanho N=8 e uma rate de $\frac{1}{2}$. Este pode ser especificado pela matriz presente em 2.52 (ETSI, 2015).

$$H = \begin{array}{c} \begin{matrix} n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & n_7 & n_8 \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{matrix} \tag{2.52}$$

As relações presente na matriz pode ser também vista no gráfico bipartite na figura 12. O gráfico conserva as considerações do LDPC com as suas características, no caso tem-se 8 nós de variáveis e 4 nós de verificação, o que corresponde a um taxa de $\frac{1}{2}$.

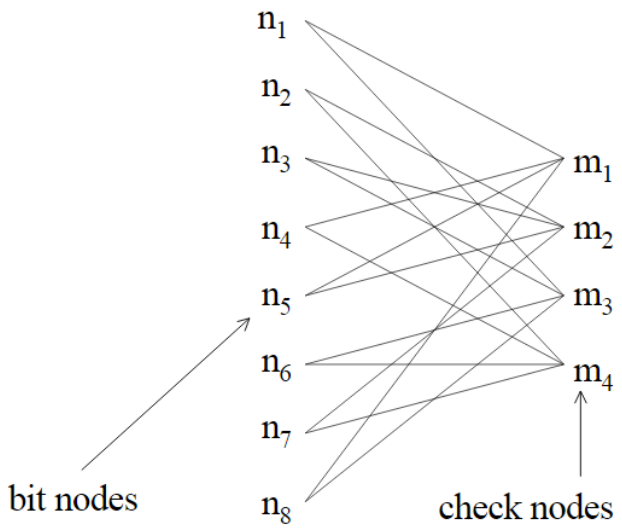


Figura 12 – *Bipartite* exemplo LDPC DVB-S2X (ETSI, 2015).

> Segundo (ETSI, 2015, p. 65), a finalidade do decodificador é determinar os valores transmitidos dos bits. Os nós de variáveis e os nós de verificação se comunicam entre si para realizar isso. A decodificação começa atribuindo o valor do canal recebido de cada bit a todas as bordas de saída do nó de variável correspondente aos nós de verificação adjacentes. Ao receber isso, os nós de verificação fazem uso das equações de verificação de paridade para atualizar as informações do nó de variável e enviá-lo de volta. Cada nó de variável, em seguida, executa uma votação por maioria simples entre as informações que chegam de seus nós de verificação adjacentes. Neste ponto, se as decisões difíceis sobre os bits satisfizerem todas as equações de verificação de paridade, isso

significa que uma palavra de código válida foi encontrada e o processo é interrompido. Caso contrário, os nós de variáveis continuam enviando o resultado de seus votos por maioria simples para os nós de verificação.

De forma resumida os passos que são tomados e que já foram previamente enunciados são

- Inicialização

$$v_{n\to k_i} = u_n, \qquad n = 0, 1, ..., N-1, \qquad i = 1, 2, ..., \ deg \ (bitnoden) \tag{2.53}$$

Na equação 2.53, tem-se a mensagem que vai do nó de variável n para o nó de verificação adjacente k_i, anota o valor do canal para o bit n e N é o tamanho da palavra-código. O processo de inicialização também é mostrado na figura 13 (ETSI, 2015).

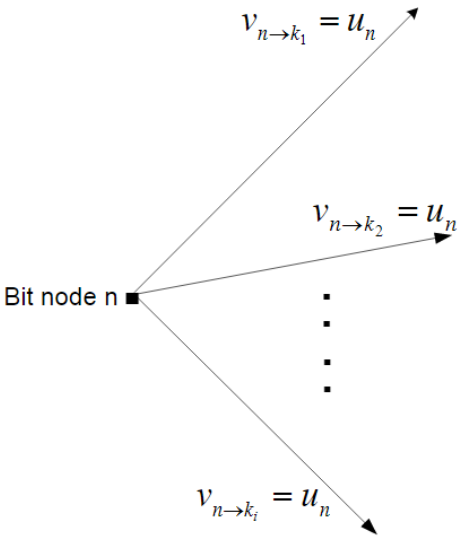


Figura 13 – Inicialização (ETSI, 2015).

- Atualização do nó de verificação

Denota-se as mensagens recebidas para o nó de verificação k de seus nós de variáveis adjacentes d_c por $V_{n_1\to k}, V_{n_2\to k}, ..., V_{n_{dc}\to k}$ que pode ser visto na figura 14 - (a). Então calcula-se as mensagens de saída do nó de verificação k de volta para nós de variáveis d_c adjacentes, com as mensagens $W_{k\to n_1}, W_{k\to n_2}, ..., W_{k\to n_{dc}}$ (ETSI, 2015).

$$W_{k\to n_i} = g(V_{n_1\to k}, V_{n_2\to k}, ..., V_{n_{i-1}\to k}, V_{n_{i+1}\to k}, .., V_{n_{dc}\to k}) \tag{2.54}$$

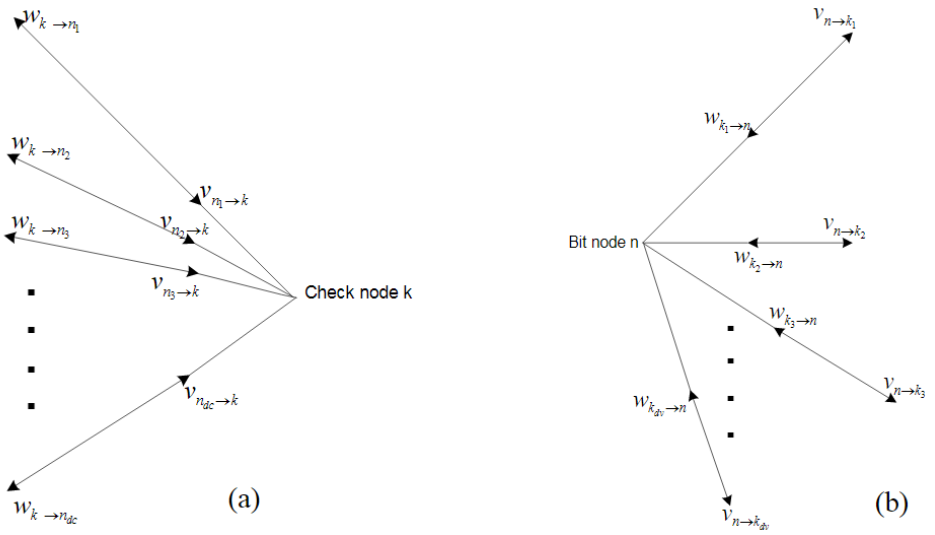


Figura 14 – (a) Atualização do nó de verificação, (b) Atualização do nós de variáveis (ETSI, 2015).

Onde:

$$g(a,b) = sign(a) \times sign(b) \times min(|a|,|b|) + LUT_g(a,b) \tag{2.55}$$

E:

$$LUT_g(a,b) = \log(1 + e^{-|a+b|}) - \log(1 + e^{-|a-b|}) \tag{2.56}$$

- Atualização do nó de *bit*

Denota-se as mensagens recebidas para o nó de variável n de seus nós de verificação d_v adjacentes por $W_{k_1 \to n}, W_{k_2 \to n}, ..., W_{k_{dv} \to}$ que pode ser visto na figura 14 - (b). Então calcula-se as mensagens de saída do nó de variável n de volta para nós de verificação adjacentes d_v. Denotar essas mensagens por $V_{n \to k_1}, V_{n \to k_2}, ..., V_{n \to k_{d_V}}$. Então é computado com a equação 2.57 (ETSI, 2015).

$$V_{n \to k_i} = u_n + \sum_{j \neq i} W_{k_j} \to n \tag{2.57}$$

- Decisão para próxima iteração

Após as atualizações do nó de variável, a decisão difícil pode ser feita para cada *bit* n observando o sinal de $V_{n \to k_i} + W_{k_i \to n}$ para qualquer k_i. Se as decisões difíceis satisfizerem todas as equações de verificação de paridade, significa que uma palavra de código válida

foi encontrada, portanto o processo é interrompido. Caso contrário, outro nó de verificação/atualização de nó de *bit* é executado. Se nenhuma convergência for alcançada após um número predeterminado de iterações, a saída de corrente será distribuída. Conforme a Relação sinal-ruído (SNR) aumenta, o decodificador converge com menos iterações (ETSI, 2015).

2.11 Estado da arte

Esta seção visa a apresentação de resultados obtidos em trabalhos que tiveram como tema a implementação em FPGA dos algoritmos utilizados no processo de decodificação do LDPC. Os resultados referentes aos decodificadores LDPC estão apresentados na Tabela 3.

(LOI, 2010), foram utilizados dois diferentes dispositivos o FPGA *Xilinx Virtex-II Pro XC2VP100* e a *Xilix Virtex-6 XC6LX240T*. Nesta tese os autores se preocuparam em realizar testes para a construção de algoritmos com um esquema de mapeamento de memória que permite que seja utilizado 180 ou 360 unidades funcionais (FUs) em paralelo, que são usadas na decodificação usando o Algoritmo de Soma de Produto (SPA).

(LIMA et al., 2014), traz a construção de todo o padrão DVBS2-X, em especial o LDPC que representa cerca de 80% da área consumida, sendo justificado por todo o processamento necessário e pelo nível de paralelismo.

Em (MENGARDA et al., 2016) é apresentado o desenvolvimento de um core em lógica programável de um *codec* LDPC (Low-Density Parity-Check) compatível com o padrão DVB-S2. O core opera com os dois tamanhos de *frames* e as vinte e uma taxas de codificação previstas no padrão. O mesmo usa (LOI, 2010) como parâmetros de teste e construção.

Autor	Ano	Tecnologia	Algoritmo	Consumo de Recursos
Loi (2010)	2010	*Virtex-II Pro XC2VP100* - 180 (FUs)	SPA	FFs: 26458; LUTs: 67026; BRAMs: 191; $f_{max}(MHz)$: 83.4
Loi (2010)	2010	*Virtex-II Pro XC2VP100* - 360 (FUs)	SPA	FFs: 51151; LUTs: 120823; BRAMs: 191; $f_{max}(MHz)$: 87.1
Loi (2010)	2010	*Virtex-6 XC6LX240T* - 180 (FUs)	SPA	FFs: 27130; LUTs: 51245; BRAMs: 129; $f_{max}(MHz)$: 83.4
Loi (2010)	2010	*Virtex-6 XC6LX240T* - 360 (FUs)	SPA	FFs: 51245; LUTs: 90432; BRAMs: 129; $f_{max}(MHz)$: 87.1
Lima et al. (2014)	2014	*Alteras FPGA (Stratix IV)*	Minimum-Sum algorithm	FFs: 59692; LUTs: 53607; BRAMs: 118;
Mengarda et al. (2016)	2016	*Xilinx XC7K160*	Minimum-Sum algorithm	FFs: 45183; LUTs: 79524; BRAMs: 334; $f_{max}(MHz)$: 100,058

Tabela 3 – Tabela comparativa de implementações de codificação do LDPC.

3 Metodologia

3.1 Introdução

O presente trabalho realiza o estudo e construção da arquitetura de um LDPC próprio do padrão de comunicação DVB-S2X. Neste trabalho foi considerada apenas a construção da primeira varredura em FPGA com a utilização da linguagem de descrição de *hardware* VHDL. Para a organização e gerenciamento das tarefas relacionadas ao desenvolvimento do projeto, foi adotada a metodologia ágil *SCRUM* com *Sprints* quinzenais. As primeiros *Sprints* foram dedicadas ao desenvolvimento do ALU que com a incrementação na segunda etapa posteriori deste trabalho vem a ser as Unidades Funcionais (FU).

O trabalho contou com a existência de um modelo de referencia dos componentes do sistema que foram codificados com o auxilio de uma ferramenta de *software*, usando a linguagem de programação C. Um diferencial desse modelo é que em sua implementação buscou-se uma aproximação da implementação em linguagem de descrição de *hardware*.

A partir do modelo de referência foram especificados os blocos e partes necessárias para construção, então, foi realizada sua codificação em VHDL. As arquiteturas implementadas foram submetidas a uma análise comportamental por meio de simulações, visando a verificação de seu funcionamento, essa análise consiste em instanciar as entidades como componentes em um arquivo de simulação. Neste arquivo, são criados sinais que são responsáveis por gerar estímulos nas entradas do sistema, os valores utilizados como entrada estão salvos previamente em uma memoria RAM *(Random-access memory)*, essas apresentam uma grande dimensão por conta do fator de paralelismo intrínseco, e as conexões e deslocamento que devem ser realizadas, foram salvas em uma memoria ROM *(Read-only memory)* com duas portas.

As implementações dos códigos, assim como o acompanhamento da evolução do trabalho, foram realizadas com o uso da ferramenta *Git* para versionamento de projetos. O *Git* é um sistema que armazena as mudanças feitas em arquivos ao longo de um período, visando a recuperação de versões específicas e o monitoramento das modificações ao longo do ciclo de vida do projeto (CHACON; STRAUB, 2014). Adicionalmente, foi utilizada uma pasta com compartilhamento em nuvem para a manutenção dos códigos, prevendo qualquer falha do computador utilizado.

3.2 Requisitos

Por meio de estratégias para sincronização dos subsistemas do decodificador FEC - DVB-S2X, foi definido requisitos de interface e funcionamento para os sistemas. Os requisitos do LDPC estão descritos a seguir.

- *Reset* assíncrono

  O decodificador deve retomar as condições inicias e mantê-las enquanto o sinal de *reset* estiver habilitado. Quando o sinal de *reset* assíncrono $reset\_in$ for acionado, o LDPC deve permanecer no estado ocioso em que aguarda o início do recebimento da próximo entrada para decodificação.

- Reconhecimento do início do *frame* e dos parâmetros de codificação

  O LDPC deve iniciar o processo de decodificação quando nova entrada for recebida. Tendo em vista que o processo de decodificação depende do tamanho do *frame* e da taxa de código, Uma entrada chamada $sync\_in$ indica à arquitetura que pode ler o tipo de *frame* e taxa de código, então o LDPC deve iniciar a decodificação a partir das novas entradas.

- Aguardar disponibilidade dos *bits*

  A leitura da entrada de recebimento do *frame* deve ser feita apenas quando os sinais de entrada estiverem estáveis. Quando uma leitura for realizada, o decodificador deve aguardar que a entrada $data\_read\_in\_width$ tenha nível logico alto. Esse sinal habilita que percorra-se todas as conexões de um CN, e o fim de cada CN é indicado por meio da saida end_width. Outra entrada se torna necessária para a consideração dos demais CNs. Assim, quando uma leitura for realizada, o decodificador deve aguardar que a entrada $data\_read\_in\_width$ e $data\_read\_in\_deep$ tenha nível logico alto para que um novo CN seja considerado.

- *Bits* recebidos

  Os *bits* recebidos devem ser corretamente alocados, de forma que possam ser lidos durante todos os passos seguintes. Assim, é necessário alocar as entradas recorrentes a todas as iterações necessárias. As entradas serão escritas em uma memoria baseada em BRAMs, essa permite escrita e leitura por meio de sua porta. Na arquitetura da memoria o sinal r_w_memory informa se os dados devem ser escritos ou lidos.

3.3 Arquitetura do LDPC

Os códigos LDPC possuem um algoritmo de decodificação facilmente paralelizável, que consiste em operações simples, como adição, comparação e busca de tabela. Além

disso, o grau de paralelismo é "ajustável", o que facilita o intercâmbio e a complexidade (ETSI, 2004). Neste trabalho apresenta-se a construção da arquitetura com grau de paralelismo de 360, ou seja, as operações que o LDPC realiza acontecerão de uma única vez paralelamente.

O comprimento de bloco de código LDPC apresenta tamanhos que variam de 64800 *bits* para o *normal frame* até 16200 *bits* para o *short frame*. Nesse trabalho será considerado apenas o comprimento de bloco máximo do *short frame*. Contudo, a arquitetura proposta é parametrizável e, portanto, permite alteração para o comprimento de bloco do *normal frame*. Bastaria alterar em principio a capacidade da memória RAM que armazena os dados de entrada.

Na figura 15 está o RTL (Register Transfer Level) da arquitetura completa do sistema a mesma tem sua versão expandida no Apêndice B. A primeira varredura aqui referida tem o efeito de recolher os LLRs *(log-likelihood ratio)* e a partir das confecções predefinidas entre os nós de verificação e os nós de variável definem qual valor de LLR corresponde ao menor e segundo menor valor. Os LLRs avaliam com base na razão de probabilidades se a essa relação de verossimilhança das variáveis é significativamente diferente, ou equivalentes, se o seu logaritmo natural é significativamente diferente de zero (KING, 1998).

Quanto aos algorítimos LDPC, uma intuição por trás é que os nós de variáveis cujos valores mais variam são aquelas que precisam ser atualizados primeiro. Nós altamente confiáveis, cuja magnitude do (LLR) é grande e não muda significativamente de uma atualização para outra, não exigem atualizações com a mesma frequência que outros nós (RICHARDSON, 2003). Na primeira varredura determinasse a partir de um conjunto de LLR quais nós apresentam menor confiabilidade, ou seja, quais nós apresentam menores magnitudes do LLR, além de também serem determinados o sub-mínimo, o endereço do mínimo LLR e o produto dos sinais de entrada que serão utilizados na segunda varredura.

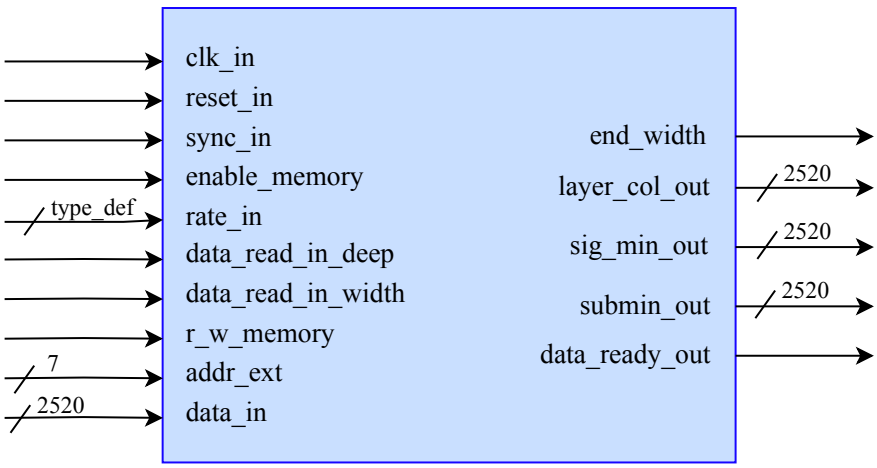


Figura 15 – Arquitetura geral da primeira varredura do LDPC.

No primeiro instante o bloco deve ter em suas entradas os sinais que determinam qual tarefa será realizada, indicando por meio de *r_w_memory* se será realizada uma escrita ou se no dado instante os dados devem serem lidos para que seja realizada a primeira varredura. A entrada *rate_in* indica qual é a taxa de código.

Realizado a escrita dos LLRs de acordo com o tamanho do *frame* da *rate* em questão, o fluxo de ocorrência interna da arquitetura é apresentada na figura 16. Na mesma memoria RAM que acabou-se de escrever, passa-se a ler. O acesso as posições do nós de variáveis é realizado de acordo com as conexões que são disponibilizadas pela norma, ou seja de acordo com uma conjuntura de nós de verificação, cujas conexões são fixas e armazenadas em uma memoria ROM. Essas conexões e o formato que foram salvos pode ser encontrado no apêndice A.

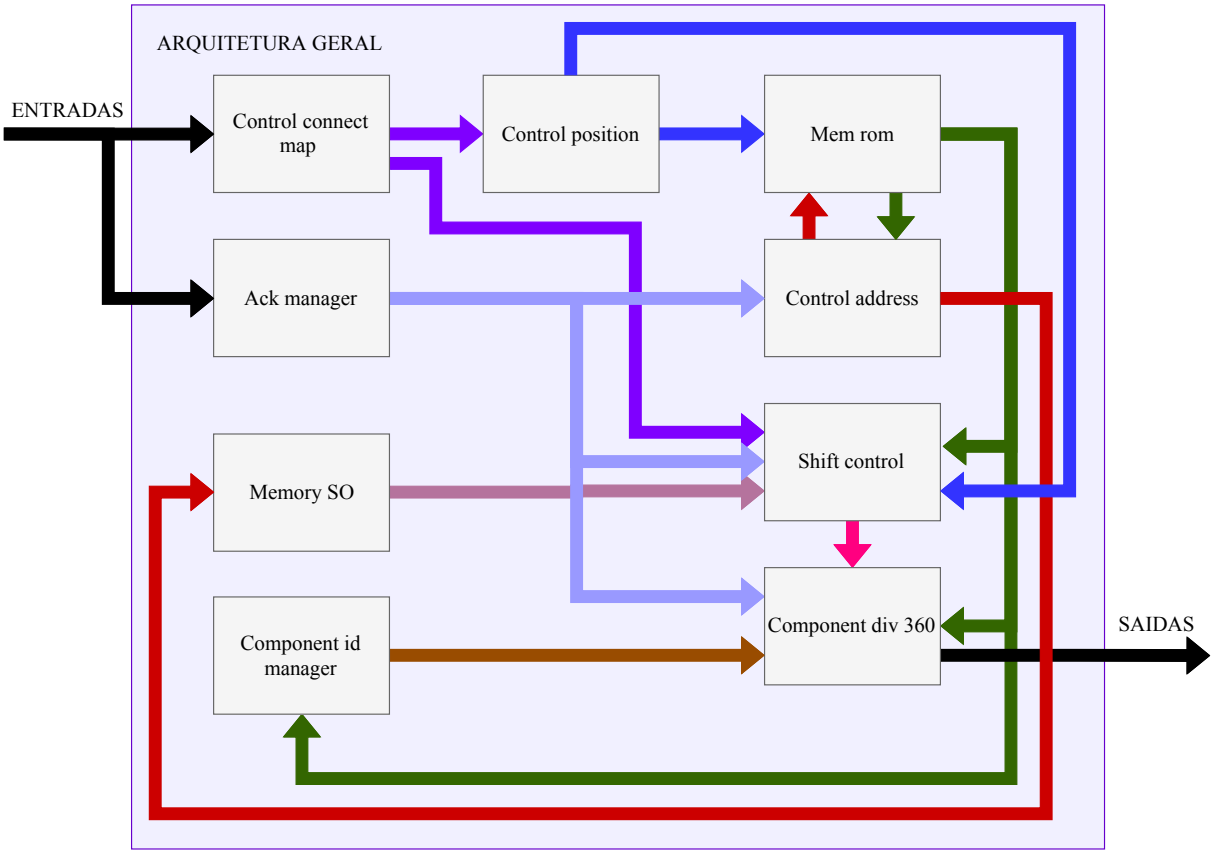


Figura 16 – Arquitetura geral interna da primeira varredura do LDPC.

Vale considerar a figura 17, na qual é possível observar a matriz esparsa, além da relação entre os VNs e os CNs. Observa-se que essa matriz é justamente para o pior caso aqui tratado onde requer um bloco de tamanho igual a 16200 *bits*. Cada posição onde se tem uma linha diagonal corresponde a uma matriz identidade, no primeiro CN é possível ver a existência de 10 matrizes, isso pode ser visto também no apêndice A, percebe-se ainda que essas podem se repetir diferindo do inicio da matriz identidade, no caso o valor do deslocamento da matriz. Essas posições onde estão as matrizes identidades são tidas como *ID*. Outra característica é que as matrizes não estão necessariamente centralizadas,

na verdade apresentam em sua maioria um deslocamento, esse deslocamento se condiz na característica aqui tida como *shift*, e também é guardado na memoria ROM. Esses valores estão presente de forma individual para cada *rate*, e caracterizam a matriz de verificação, sendo um formato numérico da figura 17 para o caso da *rate* $\frac{2}{3}$.

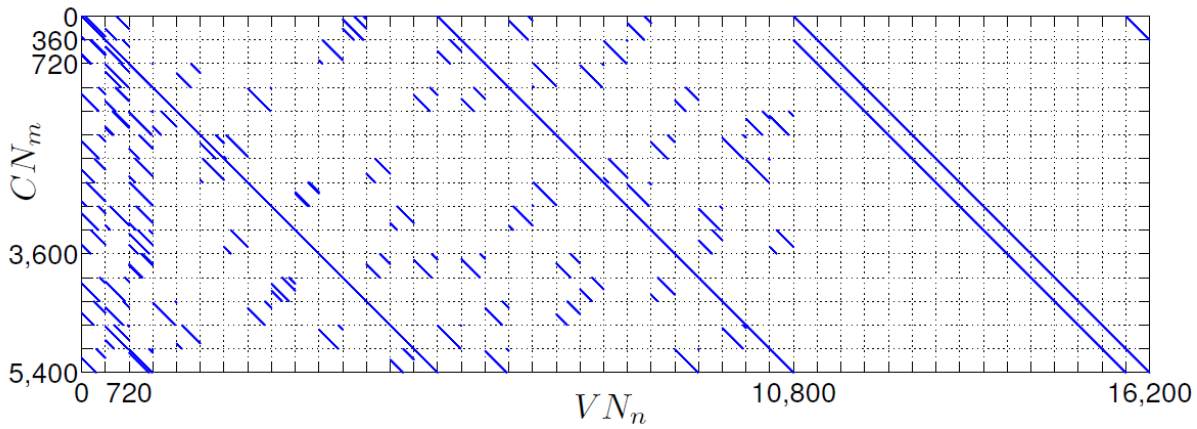


Figura 17 – Estrutura da matriz *rate* $\frac{2}{3}$ DVB-S2 (MARCHAND, 2010).

Ainda na figura 17 pode ser visto que cada quadrado menor, aonde se encontram as matrizes identidades, tem tamanho de 360×360 sendo desta característica a possibilidade de paralelização. Cada posição dentre essas 360 da matriz menor tem um LLR, e cada LLR tem 7 *bits*, ao total tem-se para a *rate* $\frac{2}{3}$ 16200 LLRs na entrada, o que resulta em um total de 113400 *bits*. Olhando-se atentamente a matriz com relação a seus deslocamentos é possível ver que deslocar os VNs de forma que se trata cada *ID* independentemente é semelhante a deslocar a própria matriz, o deslocamento é facilmente visto como exemplificado na figura 18, a figura tem apenas papel didático.

Atentando-se a figura 16 é possível identificar os blocos *Mem rom* e *Memory SO*, o mesmo pode ser visto na arquitetura presente no apêndice B. O bloco *Mem rom* tem salvo as características de *ID* e *shift* e no bloco *Memory SO* tem-se salvo os LLRs. A partir desses blocos é feito a leitura dos LLRs e das característica *ID* e *shift*. Logo após no bloco *Shift control*, a entrada é deslocada com o valor de *shift* relacionado aquele *ID*. Depois de deslocados os SOs são passados para o bloco *Component div 360* que também pode ser identificado na figura 19. Esse bloco tem 360 unidade da ULA paralelizada, assim, o *frame* que foi deslocado é separado de forma que cada SO é avaliado individualmente e na saída do bloco *Component div 360* tem-se o valor de mínimo, sub-mínimo e o produto dos sinais de todos as 360 unidades ULAs concatenados em vetores referentes a cada um. Na figura 19 tem-se como é o funcionamento interno do bloco *Component div 360*.

Dentro do modulo *Component div 360* tem-se o total de 360 unidades da ULA, dentro de cada uma dessas unidades tem-se o processamento mostrado na figura 20. É necessário realizar o complemento de dois dos números negativos recebidos tal que possa-se retirar o modulo e sinal para determinação de quais SOs tem menor valor, ou seja,

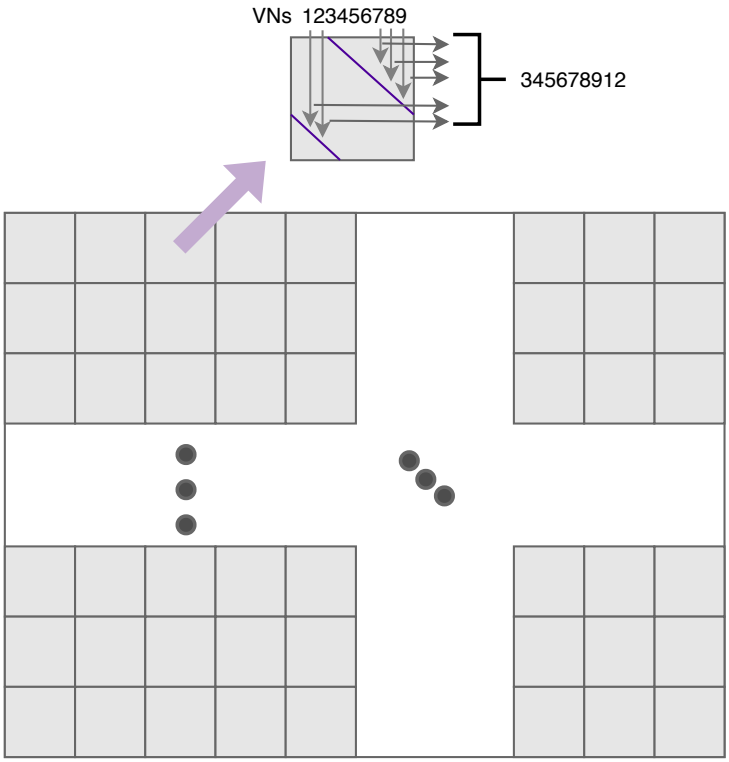


Figura 18 – Exemplo didatico do deslocamento.

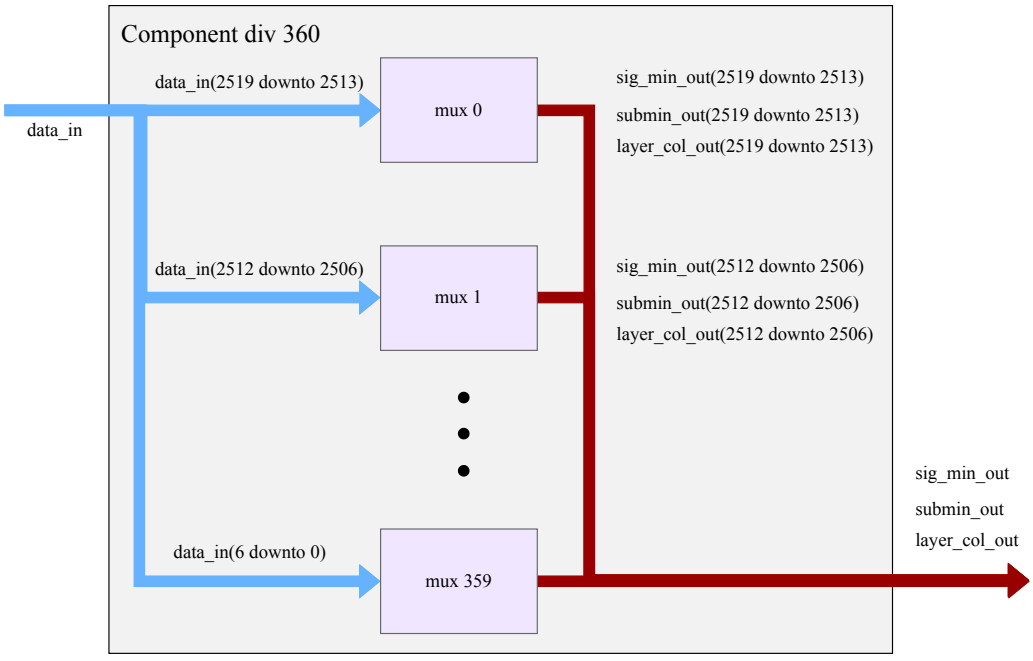


Figura 19 – Processamento individual dos SOs.

qual tem maior necessidade de ser alterado, o complemento a dois do sinal de entrada é realizado pelo bloco *Complement two* e o bloco ALU faz o processamento desses SOs referente a todo um CNs.

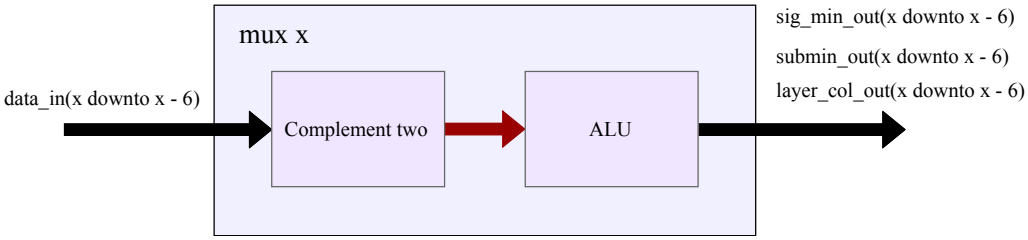


Figura 20 – Célula paralelizada.

3.3.1 Arquitetura da célula paralelizável e ULA

Na figura 21 tem-se a arquitetura da célula paralelizada, ao total 360 como já mencionado no modulo *Component div 360* e apresentado na figura 19. A entidade trabalha de maneira síncrona e conta com um *reset* assíncrono.

Na entrada desta entidade são recebidos os sinais referentes aos 360 SOs já deslocados, estes 360 SOs são então separados e manipulados de forma individual, passando pelo processo presente na arquitetura apresentada na figura 21. Os SOs negativos passam pela operação de complemento a dois separando o *bit* de sinal e os de magnitude. Posteriormente, os SOs são direcionados a ALU que faz o processamento de forma a determinar os valores do produto dos sinais, o mínimo e seu endereço, e o valor sub-mínimo.

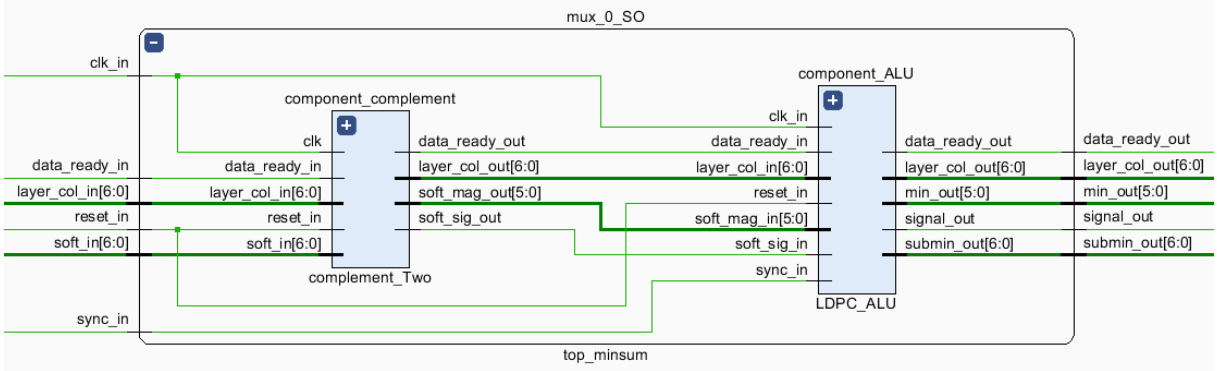


Figura 21 – Arquitetura da célula paralelizada.

Nas tabelas 4 e 5, são apresentadas as entradas e saida referentes as entidades *component div 360* e *ALU* e a descrição dessas respectivamente.

3.3.2 Arquitetura memoria RAM

A memoria RAM é utilizada para salvar os 360 LLRs e para a leituras desses em todas as iterações necessárias para o LDPC. Foi-se utilizado em sua construção um total

N°de bits	Nome	Descrição
1	clk_in	Sinal de *clock*.
1	reset_in	*Reset* assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
1	data_ready_in	Indica uma nova entrada.
7	layer_col_in	Indica o endereço referente ao valor do SO de entrada.
2520	dados_in	Entrada contendo os 360 SOs.
1	data_ready_out	Indica uma nova saída.
2520	sig_min_out	Saída com o produto "*signal*"e valor mínimo do SO recorrente de todas as células.
2520	submin_out	Saída com o valor sub-mínimo do SO recorrente de todas as células.
2520	layer_col_out	Saída com o endereço referente ao valor do SO mínimo.

Tabela 4 – Descrição das entradas e saídas do conjunto de células paralelas.

N°de bits	Nome	Descrição
1	clk_in	Sinal de *clock*.
1	reset_in	*Reset* assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
1	data_ready_in	Indica uma nova entrada.
6	layer_col_in	Indica o endereço referente ao valor do SO de entrada.
1	soft_sig_in	Entrada contendo o sinal do SO.
6	soft_mag_in	Entrada contendo magnitude do SO.
1	data_ready_out	Indica uma nova saída.
1	signal_out	Saída com o produto "*signal*"dos SOs.
6	min_out	Saída com o valor mínimo dos SOs.
7	submin_out	Saída com o valor sub-mínimo dos SOs.
7	layer_col_out	Saída com o endereço referente ao valor do SO mínimo.

Tabela 5 – Descrição das entradas e saídas da ALU.

de 35 blocos de RAM de 36K. A memoria tem largura de 2520 *bits* e uma profundidade de 45 camadas, assim, é escrito na memoria os SOs em passos de 2520 *bits* o que corresponde a um quadrado da matriz e a um único ID. Dessa forma o numero máximo de escritas para um dado *frame* é de $\frac{16200}{360} = 45$, que corresponde com a profundidade da memória.

Na figura 22 tem-se a arquitetura da memoria RAM e na tabela 6 tem-se a descrição das entradas e das saídas recorrentes.

3.3.3 Arquitetura memoria ROM

A memoria ROM é utilizada para salvar o mapeamento da matriz esparsa de paridade, sendo salvo o *ID* e o *shift*, dados que categorizam a matriz, de acordo com a matriz presente na figura 17. Foram mapeadas as matrizes referentes a todas as *rates*

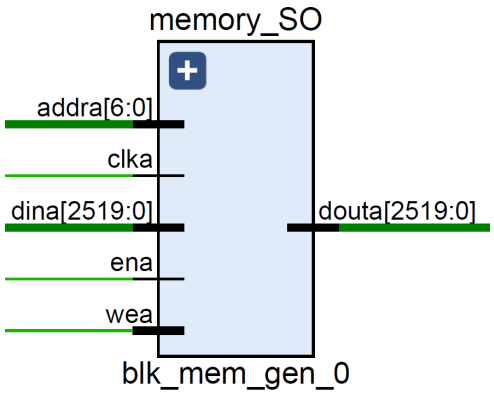


Figura 22 – Arquitetura da memoria RAM.

N°de bits	Nome	Descrição
1	clka	Sinal de *clock*.
7	addra	*Reset* assíncrono.
2520	dina_in	Porta de entrada e saída de dados.
1	ena	*Enable*, tão somente se igual a '1' sera realizado alguma ação.
1	wea	Se igual a '0' é realizado leitura, se igual a '1' será feito uma escrita.

Tabela 6 – Descrição das entradas e saídas da memoria RAM.

a serem utilizada no *short frame*. Neste trabalho foi implementada uma ROM de duas portas baseada em BRAMs, a memoria tem uma largura de 12 *bits* em ambas as portas e uma profundidade de 6000 camadas no toal. O uso de duas portas da-se pela a agilidade de se ler o *ID* ao mesmo passo que se lê o *shift*. Na memoria ROM está salvo as conexões recorrentes da matriz *LAYER MAP*, esses dados estão inseridos na memoria em um arquivo *.coe*, e podem ser vistos no apêndice A.

Na figura 23 tem-se a arquitetura da memoria ROM e na tabela 7 tem-se a descrição das entradas e das saídas recorrentes.

3.3.4 Controladores das memorias

O controle das memorias é realizado pelos blocos *Control position* e *Control address*, na figura 16 é possível ver a ligação entre esses blocos e as memorias. O controle da memoria ROM é feito por ambos os blocos. De acordo com a *rate* presente o bloco *Control position* envia para a memoria ROM quais endereços devem ser acessados no momento em questão, e ao mesmo tempo envia os endereços tanto do *ID* quanto o endereço do *shift*. A estrategia do endereçamento depende do tamanho de cada *frame*, portanto, a memoria ROM teve seu espaço particionado em duas partes uma para o *ID* e outra para o *shift*. O bloco *Control address* determina quando deve ocorrer a leitura, baseado no controle de

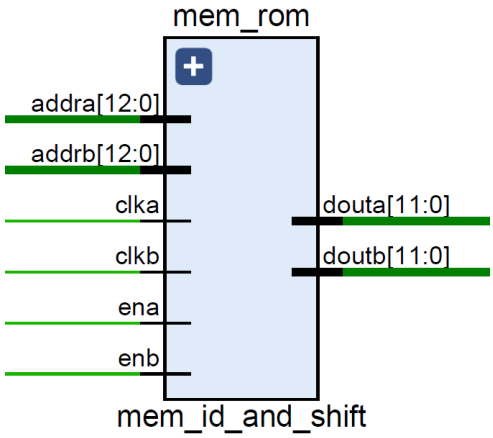


Figura 23 – Arquitetura da memoria ROM.

N°de bits	Nome	Descrição
1	clka	Sinal de *clock*.
1	clkb	Sinal de *clock*.
13	addra	*Reset* assíncrono.
13	addrb	*Reset* assíncrono.
12	douta_in	Porta de saída do *ID* da matriz.
12	doutb_in	Porta de saída do *shift* da matriz.
1	ena	*Enable*, tão somente se igual a '1' sera realizado alguma ação.
1	enb	*Enable*, tão somente se igual a '1' sera realizado alguma ação.

Tabela 7 – Descrição das entradas e saídas da memoria ROM.

enable.

O controle da memoria RAM é realizado unicamente pelo bloco *control adress*, recebendo da memoria ROM quais os valores de *ID* e a partir deste determina quais posições da memoria RAM devem ser acessadas. Além de realizar o controle de *enable* da memoria.

A arquitetura dos blocos *Control position* e *Control address* podem serem visualizadas na figura 24a e 24b respectivamente, e a descrição de suas entradas e saídas nas tabelas 8 e 9.

3.3.5 Arquitetura da maquina de transição

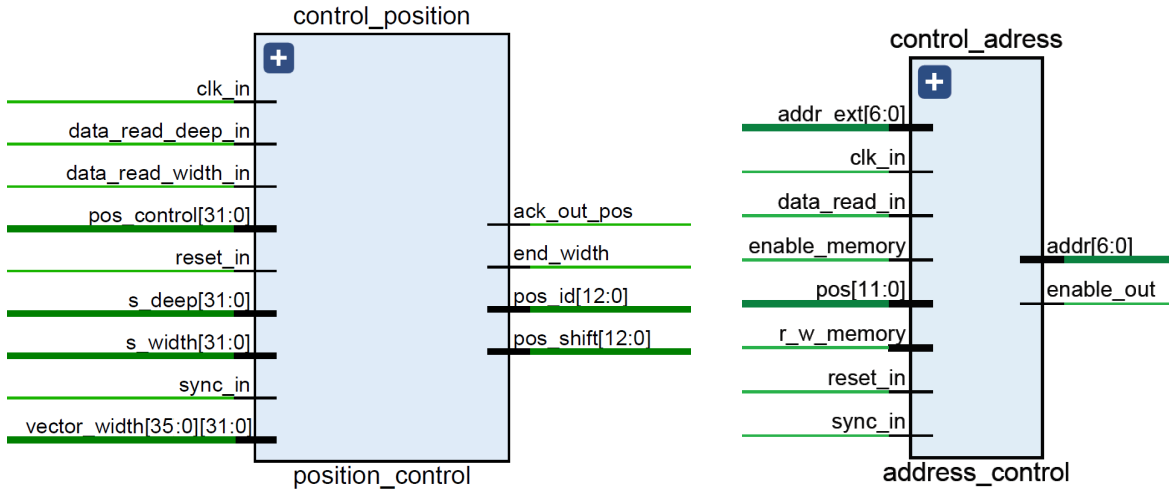
Foi-se construído um modelo de maquina de transição que faz a relação entre o sinal recebido *data_read_in* e os sinais a serem enviados aos módulos que podem ser vistos na figura 16. A maquina faz uso do advento de todos os blocos que faz referencia terem o numero de ciclos de *clock* para funcionamento fixo. O funcionamento do bloco é

N°de bits	Nome	Descrição
1	clk_in	Sinal de *clock*.
1	reset_in	*Reset* assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
1	data_read_deep_in	Entrada que avança para o próximo *CN*. Será o caso quando todos os *IDs* tiverem sido percorridos no numero de iterações necessário.
1	data_read_width_in	Entrada que avança para o próximo *ID*.
inteiro	pos_control	Relaciona a posição relativa de cada *rate* na memoria.
inteiro	s_deep	Entrada que determina a profundidade do mapa da matriz referente a cada *rate*.
inteiro	s_width	Entrada que determina a largura máxima do mapa da matriz, referente ao *rate* presente.
36 × inteiro	vector_width	Entrada que determina a largura do mapa da matriz em cada linha, referente ao *rate* presente.
1	ack_out_pos	Indica uma saída.
1	end_width	Indica o fim de um *layer* e que é o ultimo *ID*.
13	pos_id	Passa para a memoria a posição do *ID*.
13	pos_shift	Passa para a memoria a posição do *shift*.

Tabela 8 – Descrição das entradas e saídas do *control position*.

N°de bits	Nome	Descrição
1	clk_in	Sinal de *clock*.
1	data_read_in	Indica uma nova entrada.
1	reset_in	*Reset* assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
7	addr_ext	Entrada de endereço externa para o caso de escrita.
1	enable_memory	Entrada de *enable* externa para escrita na memoria.
12	pos	Posição do *ID* lida da memoria ROM.
1	sr_w_memory	Entrada que alterna entre escrita e leitura.
7	addr	Saida com endereço geral a ser lido.

Tabela 9 – Descrição das entradas e saídas do *control address*.



(a) Control position (b) Control adress

Figura 24 – Controladores de memoria.

semelhante ao de um registrador de deslocamento sem realimentação, assim sempre que um há um sinal de entrada em *data_read_in* esse é deslocado a cada ciclo de *clock* por um total de oito ciclos de *clock*. Na figura 25 é possível ver a arquitetura do bloco e na tabela 10 tem-se a descrição dos sinais de entrada e saída.

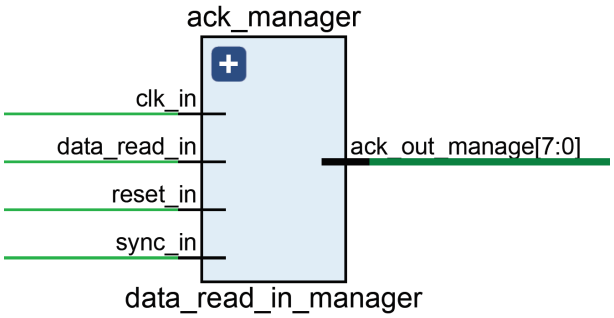


Figura 25 – Arquitetura da maquina de transição.

N°de bits	Nome	Descrição
1	clk_in	Sinal de *clock*.
1	data_read_in	Indica uma nova entrada.
1	reset_in	*Reset* assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
8	ack_out_manage	Saída com *data_read_in* transitado.

Tabela 10 – Descrição das entradas e saídas do *ack_manager*.

3.3.6 Arquitetura *control connect map*

Para a compreensão de toda a arquitetura é necessário citar que o projeto conta com dois pacotes, e nesses pacotes estão definidas constantes e variáveis gerais que são

utilizadas em diversas partes da arquitetura. Para o LDPC criou-se um pacote com o nome *LdpcPkg.vhd*, no qual foram definidas todas as constantes e tipos definidos usados no projeto. O outro pacote é *Dvbs2xPkg.vhd* que é usado para todo o projeto do RDS. No referente ao modulo *control connect map* torna-se essencial citar que alguns tipos definidos são usados, na figura 26 é possível ver a entrada *rate_in* que usa o seguinte tipo:

```
type T_DVBS2X_FRAME_TYPE is record
      FRAME_TYPE_SHORT    :   DVBS2X_FRAME_TYPE_SHORT;
      FRAME_TYPE_MEDIUM   :   DVBS2X_FRAME_TYPE_MEDIUM;
      FRAME_TYPE_NORMAL   :   DVBS2X_FRAME_TYPE_NORMAL;
 end record T_DVBS2X_FRAME_TYPE;
```

Os tipos presentes na construção do tipo *T_DVBS2X_FRAME_SIZE* são também tipos definidos, no caso aqui referido é usado para o *FRAME_TYPE_SHORT*. A seguir tem-se o modelo referido:

```
type DVBS2X_FRAME_TYPE_SHORT is(
        DVBS2X_CODE_RATE_1_4,   --! 1/4 of code rate
        DVBS2X_CODE_RATE_1_3,   --! 1/3 of code rate
        DVBS2X_CODE_RATE_2_5,   --! 2/5 of code rate
        DVBS2X_CODE_RATE_1_2,   --! 1/2 of code rate
        DVBS2X_CODE_RATE_3_5,   --! 3/5 of code rate
        DVBS2X_CODE_RATE_2_3,   --! 2/3 of code rate
        DVBS2X_CODE_RATE_3_4,   --! 3/4 of code rate
        DVBS2X_CODE_RATE_4_5,   --! 4/5 of code rate
        DVBS2X_CODE_RATE_5_6,   --! 5/6 of code rate
        DVBS2X_CODE_RATE_8_9,   --! 8/9 of code rate
        DVBS2X_CODE_RATE_1_45,  --! 1/45 of code rate
        DVBS2X_CODE_RATE_4_15,  --! 4/15 of code rate
        DVBS2X_CODE_RATE_4_45,  --! 8/15 of code rate
        DVBS2X_CODE_RATE_7_15,  --! 7/15 of code rate
        DVBS2X_CODE_RATE_8_15,  --! 8/15 of code rate
        DVBS2X_CODE_RATE_6_45,  --! 6/45 of code rate
        DVBS2X_CODE_RATE_2_45   --! 2/45 of code rate
);
```

Assim o bloco *control connect map* recebe e trabalha com esse tipo definido, e partir desse tipo define os sinais *deep*, *width*, *vector_width* e *pos_geral*. Na tabela 11 tem-se a descrição das entradas e saídas da arquitetura.

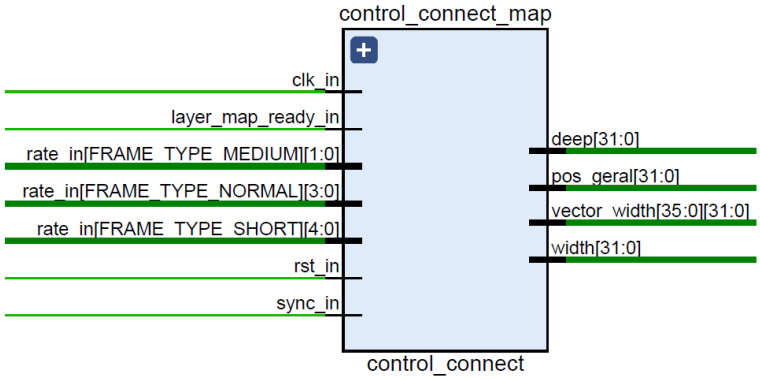


Figura 26 – Arquitetura do *control connect map*.

N°de bits	Nome	Descrição
1	clk_in	Sinal de *clock*.
1	rst_in	*Reset* assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
1	layer_map_ready_in	Entrada de *enable* externa.
type_def	rate_in [FRAME_TYPE_MEDIUM	Entrada para quando considerado uma *Rate medium*.
type_def	rate_in [FRAME_TYPE_NORMAI	Entrada para quando considerado uma *Rate normal*.
type_def	rate_in [FRAME_TYPE_SHORT]	Entrada para quando considerado uma *Rate short*.
1	deep	Saída que indica a profundida do *rate* atual.
1	width	Saída que indica a largura máxima do *rate* atual.
inteiro	pos_geral	Saída que indica a posição geral das característica *ID* e *shift* na memoria.
36×inteiro	vector_width	Saída que com a largura variável de cada linha.

Tabela 11 – Descrição das entradas e saídas do *control connect map*.

3.3.7 Arquitetura do *Shift control*

No *Shift control* é onde ocorre a maior parte do processamento, sendo ainda o bloco que consome mais recursos dentre todos, sua arquitetura pode ser vista na figura 27. A complexidade do mesmo se da por apresentar um deslocamento variável, ou seja, este deve ter em sua arquitetura uma forma que o possibilite deslocar um dado *frame* de 0 a $359 \times 7 = 2513$ *bits*. Varias estrategias foram traçadas mas a mais razoável se baseia em usar uma função própria da biblioteca *numeric.std*. Essa função a partir do valor informado realiza um deslocamennte circular no *frame* informado. A descrição das entradas e saídas é apresentado na tabela 12.

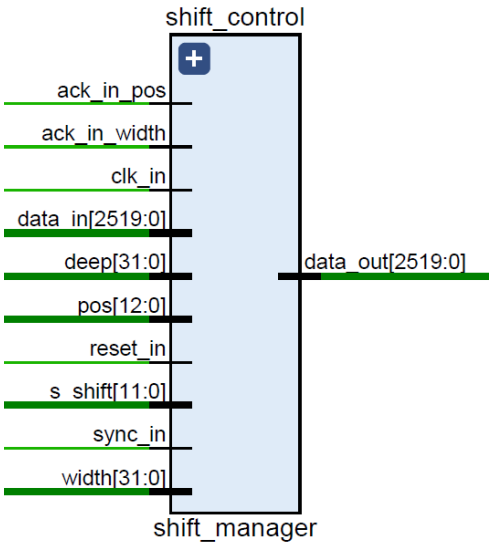


Figura 27 – Arquitetura do *Shift control*.

N°de bits	Nome	Descrição
1	clk_in	Sinal de *clock*.
1	reset_in	*Reset* assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
2520	data_in	Entrada dos dados a serem deslocados.
1	ack_in_pos	Entrada da posição.
1	data_read_width_in	Entrada indicando um novo *ID*.
12	s_shift	Entrada que informa qual o deslocamento a ser realizado.
inteiro	width	Largura do *rate*, quantidade máxima de *ID*.
inteiro	deep	Profundidade do *rate*.
13	pos	Entrada que indica a posição.
2520	data_out	Saída com os dados deslocados.

Tabela 12 – Descrição das entradas e saídas do *Shift control*.

3.4 Fluxo de desenvolvimento

Como já mencionado, este trabalho contou com um modelo de referencia já existente construído utilizando a linguagem de programação C. Esse modelo foi construído propositalmente tendo em mente o mapeamento do algoritmo em hardware, sendo construído pensando em um formato que facilitasse a visualização de uma futura construção da arquitetura usando uma linguagem de descrição de *Hardware*. A estrategia de construção foi tomada com uma abordagem *bottom-up*, aonde ocorre a junção de sistemas menores para dar origem a sistemas maiores e mais complexos.

O modelo de simulação das arquiteturas foi baseado em uma análise comportamental, na qual as entidades foram instanciadas e simuladas. Uma vez feita a integração, a arquitetura de *hardware* proposta para a primeira varredura do LDPC foi testada, utilizou-se na simulação do artificio de se ter uma memoria RAM que justamente é onde se guarda os dados referentes as entradas. As memoria tanto RAM quanto a ROM permitem a inicialização de seus dados por meio de um arquivo com extensão *.coe*. Nas simulações ambas as memorias foram inicializadas desta forma. Os arquivos para verificação de entrada e saída foram gerados e verificados com o auxilio do *software Matlab*.

3.5 Plataforma de desenvolvimento

O modelo utilizado para o projeto foi o Kit de desenvolvimento Nexys4 da Digilent, que contem um dispositivo FPGA Artix7 (chip Artix-7 FPGA XC7A100T-1CSG324C) da xilinx. O modelo referido pode ser visto na figura 28 e a relação de periféricos na tabela 13.

3.6 Velocidade de Arquitetura

Para designar a funcionalidade e operação de uma arquitetura é necessário, mensurar baseado em alguns parâmetros de forma a caracterizar o sistema e restringir aspectos mínimos de funcionamento. Para arquiteturas em FPGA tem-se alguns aspectos importantes a se considerar.

> Segundo (KILTS, 2007, p. 1), no contexto do processamento de dados em um FPGA, a taxa de transferência (*throughput*) refere-se à quantidade de dados processados por ciclo de *clock*. Uma métrica comum para taxa de transferência *(throughput)* é de bits por segundo. A latência refere-se ao tempo entre a entrada de dados e a saída de dados processados. A métrica típica para latência será tempo ou ciclos de *clock*. *Timing* refere-se aos atrasos lógicos entre elementos sequenciais. Quando dizemos que um projeto não "atende a tempo", queremos dizer que o atraso do caminho crítico, ou seja, o maior atraso entre *flip-flops* (composto de atraso combinatório, atraso entre cliques, atraso de roteamento, tempo

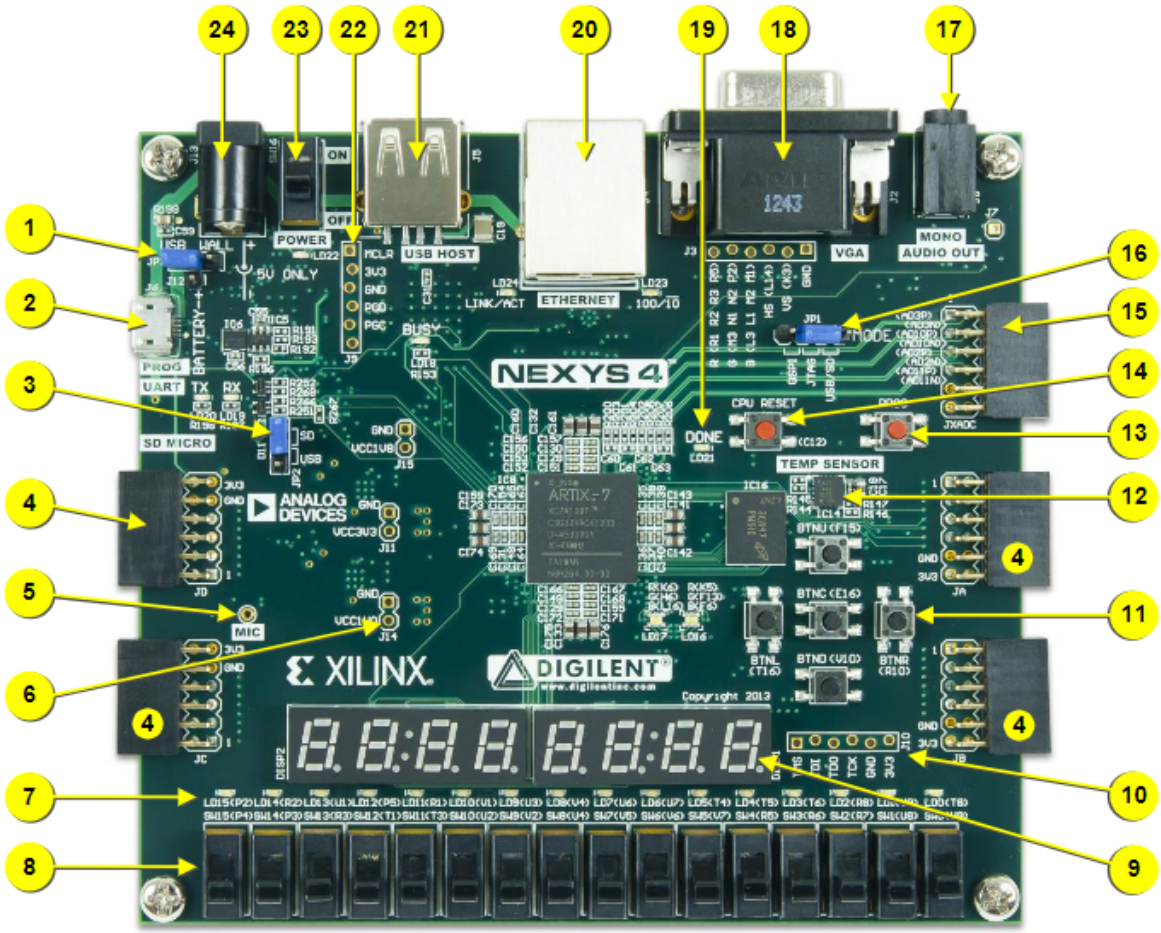


Figura 28 – Kit *NEXYS 4 - Artix-7 FPGA XC7A100T-1CSG324C* utilizado (DILIGENT, 2016).

de configuração, *clock* inclinação, e assim por diante) é maior que o período do *clock* alvo. As métricas padrão para o tempo são o período do *clock* e a frequência.

A latência, o *throughput* e a frequência de operação caracterizam a arquitetura quanto a quantidade de dados por tempo esses serão capazes de processar, dando aspectos de velocidade e escalabilidade, a latência de forma resumida corresponde quanto tempo ou ciclos de *clock* o sistema leva para processar uma entrada e apresentar a saída correspondente a essa entrada e o *throughput* corresponde a taxa de transferência correspondente ao sistema, ou seja, quantos *bits* o sistema processa a cada período de *clock* ou mesmo no período de um segundo e apresentado tal como $\frac{bits}{clock}$ ou $\frac{bits}{s}$.

3.7 Metodologia de implementação

Para a implementação foi necessário testar o circuito por meio de interfaces que permitiam a comunicação deste com os periféricos presentes na FPGA. Dessa forma construiu-se um *top level* para realizar o interfaceamento.

Referencia	Descrição	Referencia	Descrição
1	*Jumper* de seleção de energia	13	Botão de reset
2	Porta USB compartilhada UART/JTAG	14	CPU
3	*Jumper* de configuração externa	15	Porta Pmod de sinal analógico
4	Porta(s) Pmod	16	Jumper de modo de programação
5	Microfone	17	Conector de áudio
6	Ponto (s) de teste da fonte	18	Conector VGA
7	LEDs (16)	19	LED de programação
8	Interruptores	20	Conector *Ethernet*
9	7-seg	21	Conector *host* USB
10	Porta JTAG	22	Porta de programação PIC24
11	Cinco botões	23	Interruptor de alimentação
12	Sensor de temperatura	24	Conector de energia

Tabela 13 – Tabela de perifericos kit *NEXYS 4* (DILIGENT, 2016).

Dado todo o projeto e sua complexidade, a validação da arquitetura proposta foi realizada em duas partes. Primeiro foi Validado o sistema em sua construção até o passo em que ocorre o deslocamento e em segundo o funcionamento da ALU.

3.7.1 Teste da ALU

Para a ALU tem-se entradas e saídas essenciais para o *top level* essa organização está apresentada na arquitetura presente na figura 29.

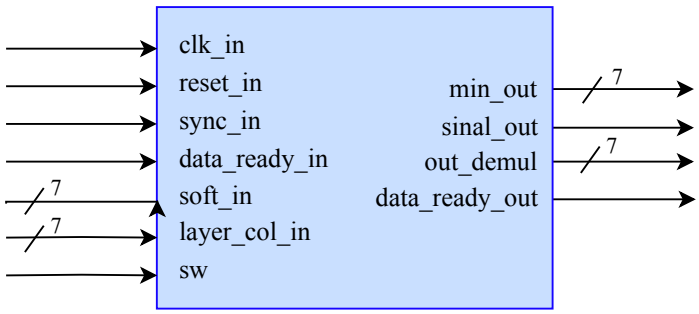


Figura 29 – Arquitetura do *top level* para implementação do ALU.

As mesmas foram mapeadas no arquivo de *.xdc* tais como nas figuras 30, 31 e 32. Na figura 30 é apresentado os sinais *reset_in*, *sync_in* e *data_ready_in*. Na figura 31

tem-se as entradas referentes as chaves, nestas entram os sinais *soft_in* que tem consigo o sinal e o modulo do SO, o sinal *layer_col_in* que insere o endereço e o sinal *sw* que faz a multiplexação da saída, de forma que se *sw* = '0' *out_demul* apresentará o *s_submin_out* referente ao valor do SO sub-mínimo e se *sw* = '1' apresentará na saída *s_layer_col_out*.

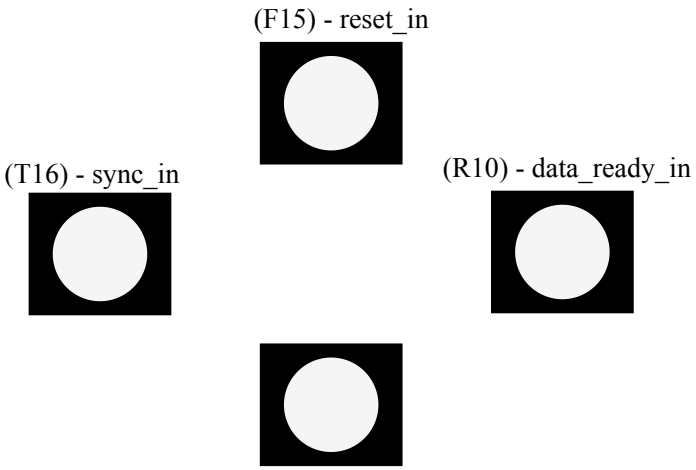


Figura 30 – Interface do *top level* da ALU, *bottons*.

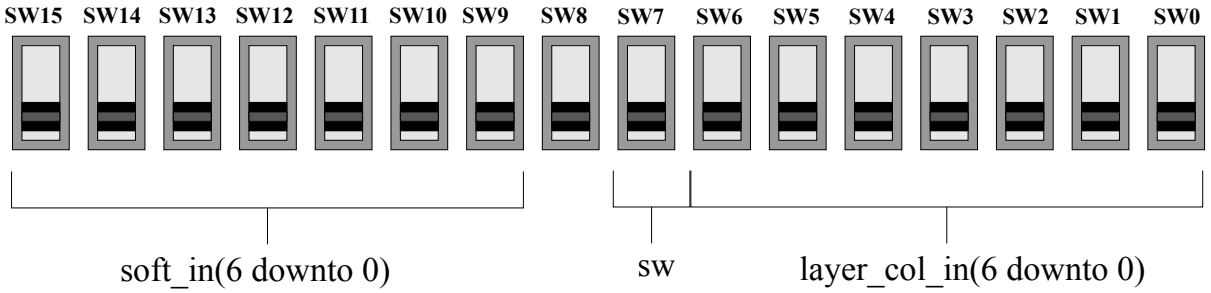


Figura 31 – Interface do *top level* da ALU, *switches*.

E por fim as saídas são apresentadas nos *leds*, no formato apresentado na figura 32.

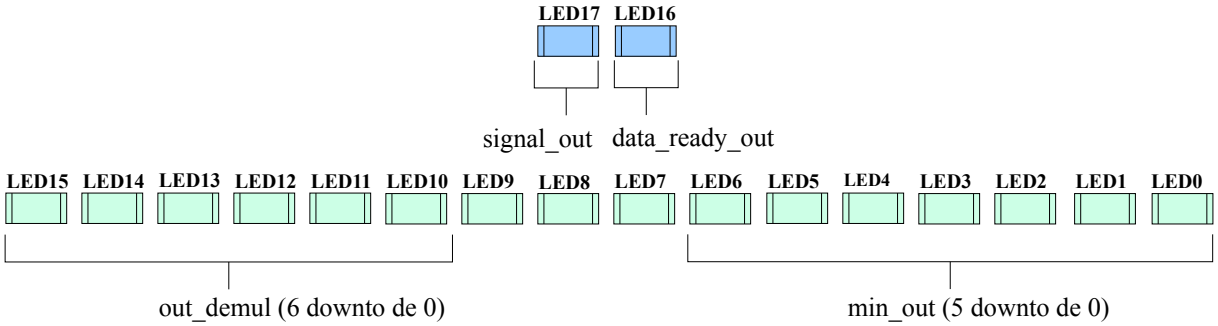


Figura 32 – Interface do *top level* da ALU, *LEDs*.

3.7.2 *Shift*

Para o *shift* tomou-se entradas e saídas essenciais para o *top level* essa organização está apresentada na arquitetura presente na figura 33.

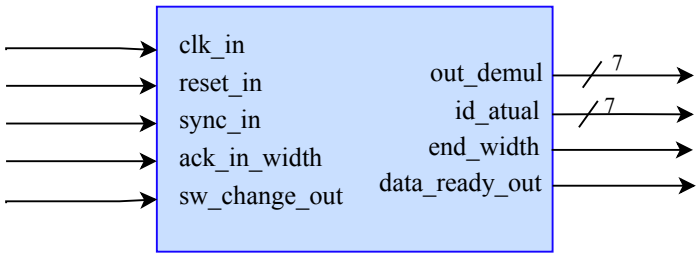


Figura 33 – Arquitetura do *top level* para implementação do *deslocamento*.

As mesmas foram mapeadas no arquivo de *.xdc* tais como nas figuras 34, 35 e 36. Na figura 34 são apresentados os sinais *reset_in*, *sync_in* e *ack_in_width*. Na figura 35 tem-se as entradas referentes as chaves, nestas entram o sinal *sw_change_out* que multiplexa as saídas de forma que se for igual a '0' *out_demul* apresentará o *s_entrada_teste* que refere-se ao valor da entrada na dada posição considerada e se *sw* = '1' apresentará na saída *s_shift_out* que condiz no valor na mesma posição mas depois do deslocamento realizado.

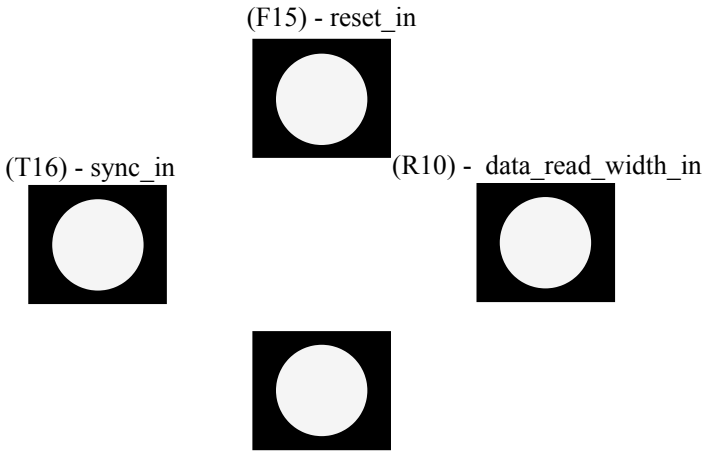


Figura 34 – Interface do *top level* da *deslocamento*, *bottons*.

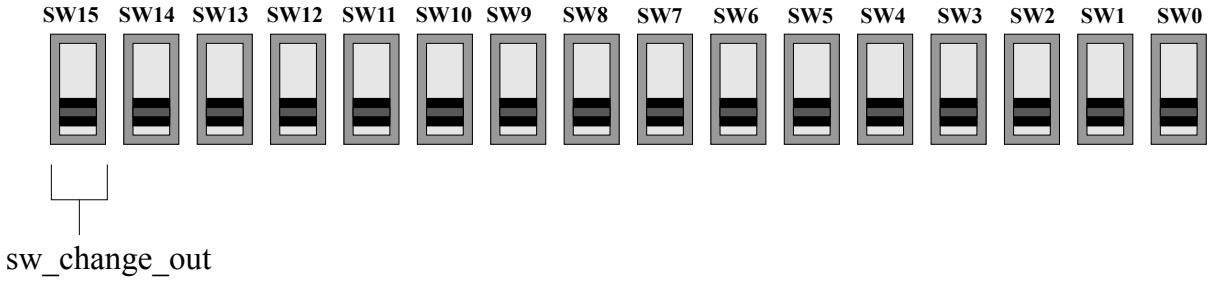


Figura 35 – Interface do *top level* da *deslocamento*, *switches*.

Por fim, as saídas são apresentadas nos *leds*. Tal que seguem o formato apresentado na figura 36.

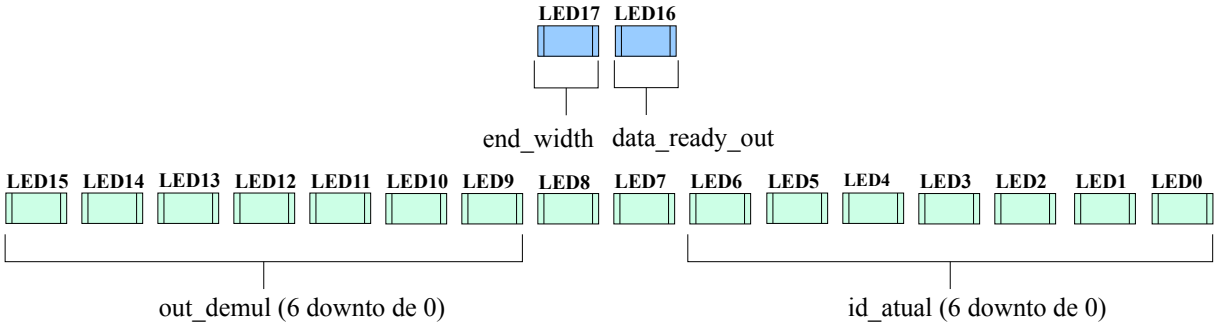


Figura 36 – Interface do *top level* da *deslocamento*, *LEDs*.

4 Resultados

4.1 Resultados de simulação

Para teste e validação preliminar das arquiteturas implementadas, submeteu-se essas a uma analise comportamental por meio de simulações. Como explicado no capítulo anterior, foram utilizadas memórias para injetar os vetores de teste. Para a criação dos arquivos, em especial o arquivos que tem as entradas foi-se feito um modelo no *Matlab*, sendo que este arquivo apresenta 45 entradas de tamanho de 2520 *bits* correspondente a maior entrada possível para o *short frame*(*rate* $\frac{2}{3}$), assim, pode ser usado para testar todas as demais. Todas as simulações obedecem a um *short frame* com taxa de código de $\frac{1}{4}$.

Na figura 37, é possível verificar todas as entradas e saídas mais externas da arquitetura. Há uma interação entre os sinais de entrada e de saída, tal que estes tem sincronia com o sinal de *clock*. O sistema apresenta o sinal de entrada *rate_in* que define a taxa de código. No escopo é possível atentar-se a 4 macro sinais, *s_douta* que é referente ao sinal de entrada e que no momento em questão que é apresentado está entrando no *Shift control*, o sinal *s_dados_out_shift* que é referente a saída do *Shift control*, o *sig_min_out* que é a saída contendo o produto dos sinais e os valores mínimos encontrados nas 360 *ULAs* e por ultimo o *submin_out* que é referente aos sub-mínimos encontrados. Na figura 38 é possível ver o mesmo *top level* com estimulos na entrada de *reset* é possível perceber que os sinais voltam as condições iniciais de forma que os valores de *min* e *submin* passam a ter o valor máximo possível e o valor do sinal passa a ser igual a 0. É importante denotar que a variável *sig_min_out* é onde se encontra os valores referentes ao sinal e ao mínimo que saem de todas as ALUs.

Na figura 39 tem-se o bloco *Shift control*, este é de fato o bloco mais complexo e é responsável pela maior parte do processamento. No caso recebe em *data_in* os dados a serem deslocados e disponibiliza em *data_out* os dados já deslocados.

Os resultados de simulação do bloco que contem as 360 ALUs em paralelo são apresentados na figura 40. Recebe o sinal e a magnitude de cada SO e disponibiliza o mínimo, sub-mínimo e o produto dos sinais.

Para a melhor compreensão da maquina de transição, na figura 41 é possível ver quando *data_read_in* é acionado e como é o funcionamento do bloco que tem grande semelhança a um registrador de deslocamento.

Por fim, quanto as simulações é importante dar atenção as memorias. Na figura 42 pode ser visto a leitura e iterações com os *Intellectual Property* (IPs) de memorias *ROM*

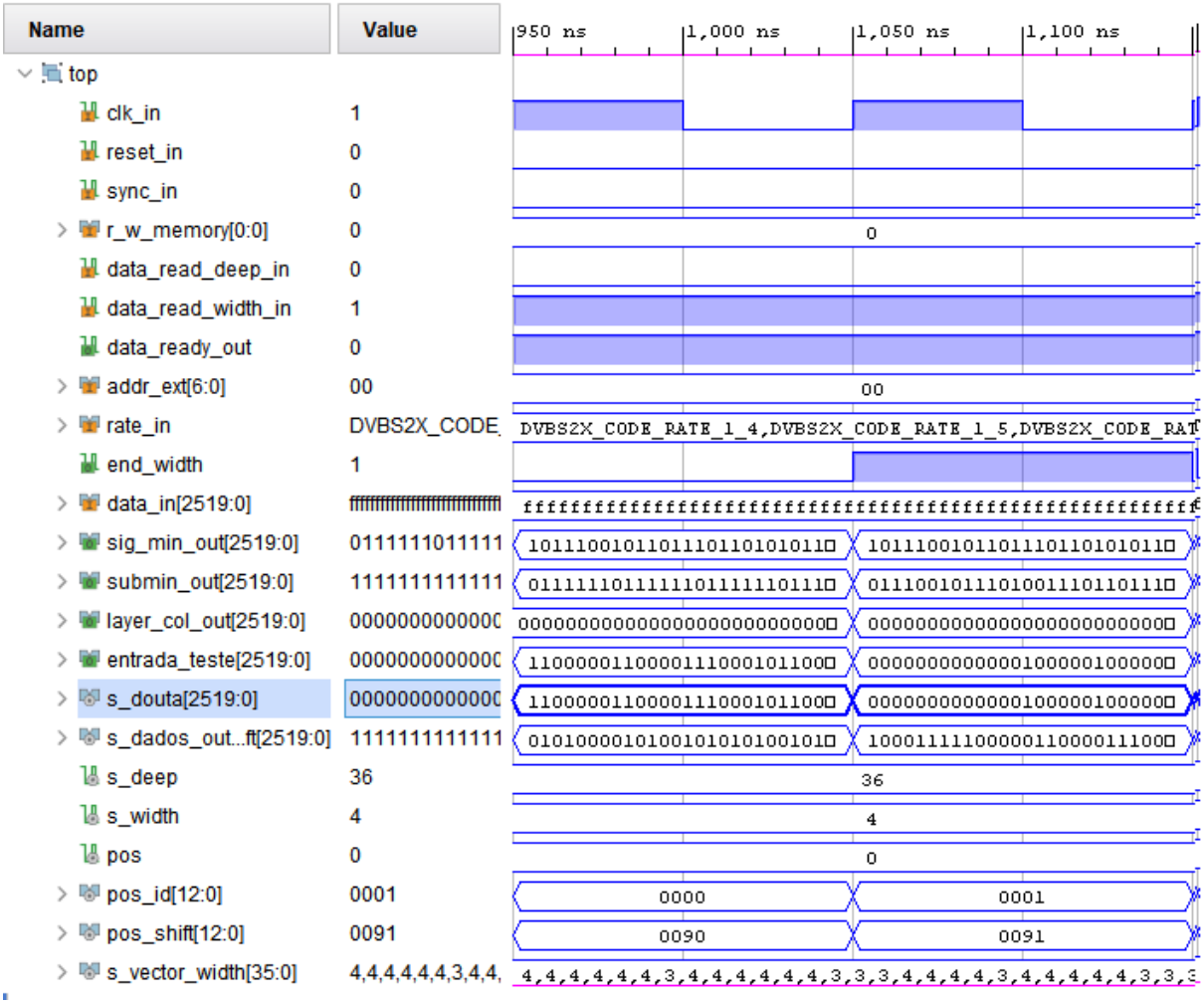


Figura 37 – Resultado de simulação, *Top level*.

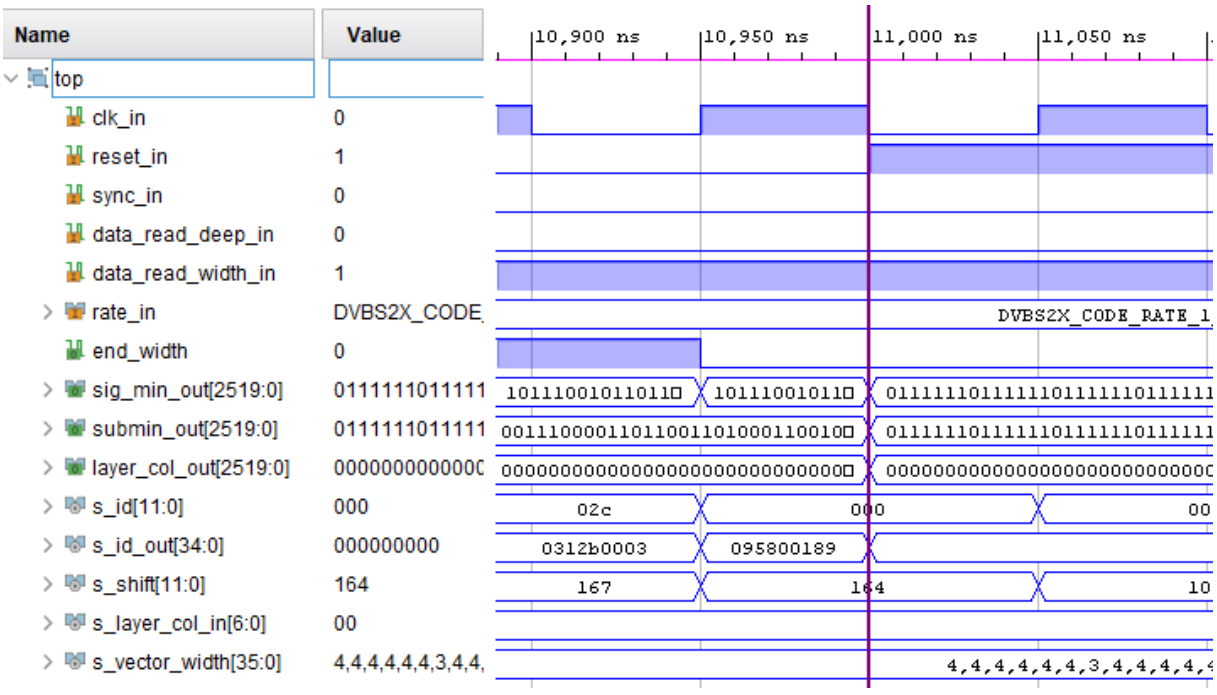


Figura 38 – Resultado de simulação, *Top level* com estimulo no sinal de *reset*.

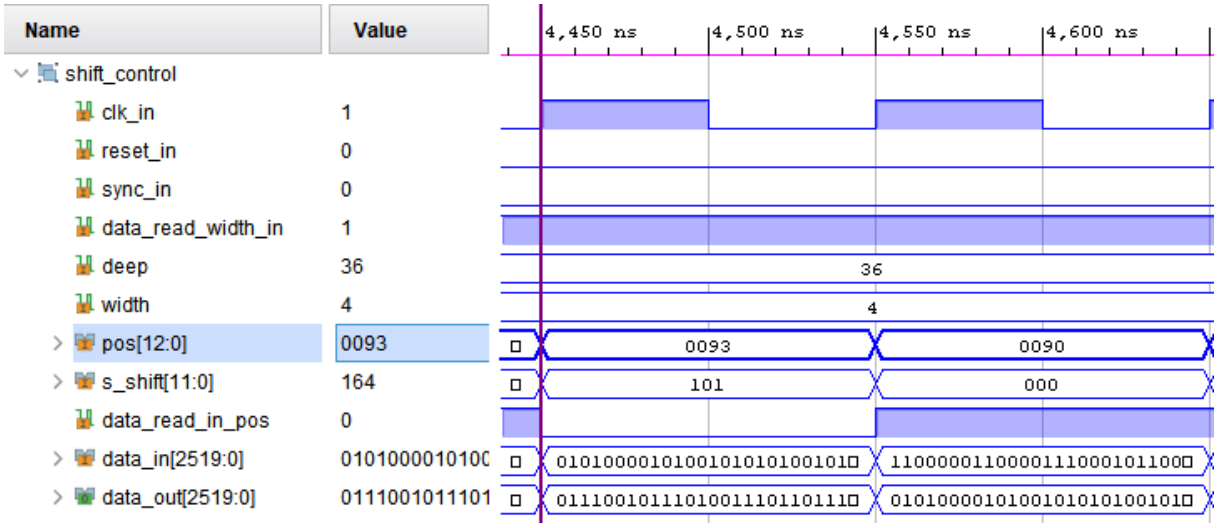


Figura 39 – Resultado de simulação, *Shift control*.

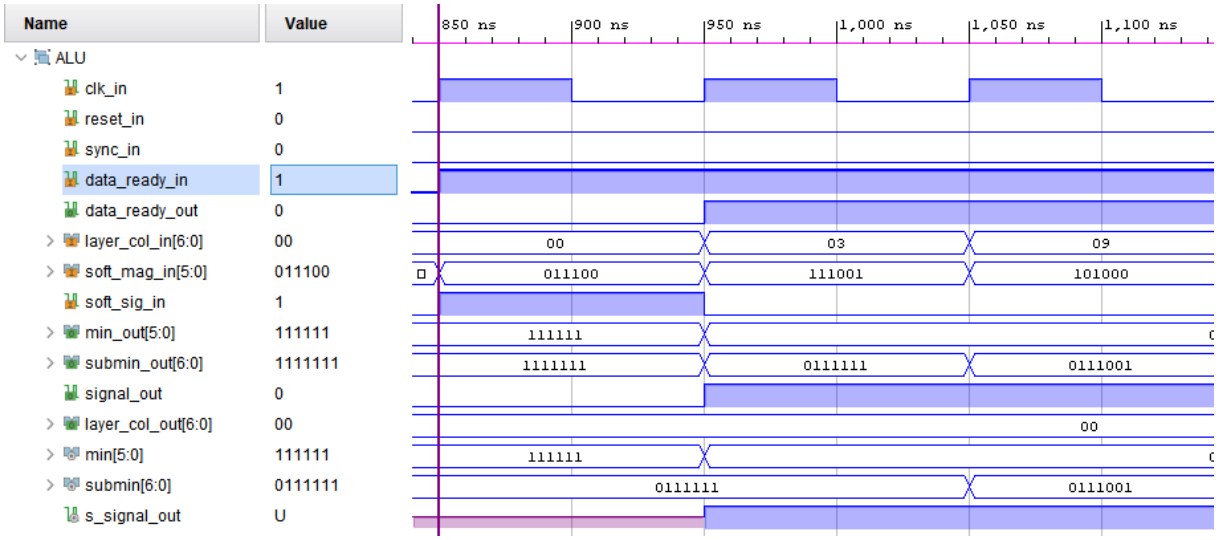


Figura 40 – Resultado de simulação, *ALU*.

e *RAM*.

4.2 Caracterização das arquiteturas

Com a análise de síntese, mapeamento dos pinos de entrada e saída das arquiteturas e o roteamento de conexão entre os componentes, foram obtidas informações referentes ao consumo de recursos totais e parciais do sistema. A tabela 14 apresenta a quantidade de *Look Up Tables*(LUTs), *Flip Flops* (FFs), multiplexadores (MUXes), entradas e saídas (IOs), blocos de memória RAM (BRAMs) e DSPs utilizados.

O consumo de IOs se dá tão somente pelo *Top level*, já que apenas esse faz interface com o exterior, sendo apresentado um total de 12620 IO o que não é possível ser feito com a implementação em um kit comum de FPGA. Tomando esse parâmetro e considerando a

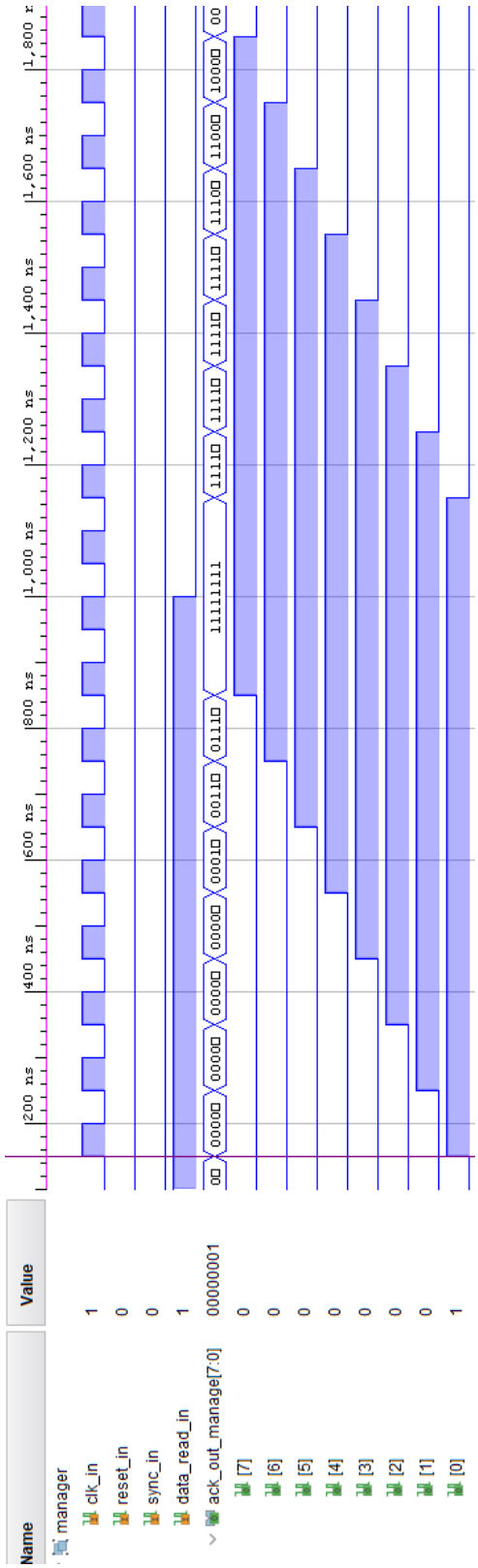


Figura 41 – Resultado de simulação, *ack_manager*.

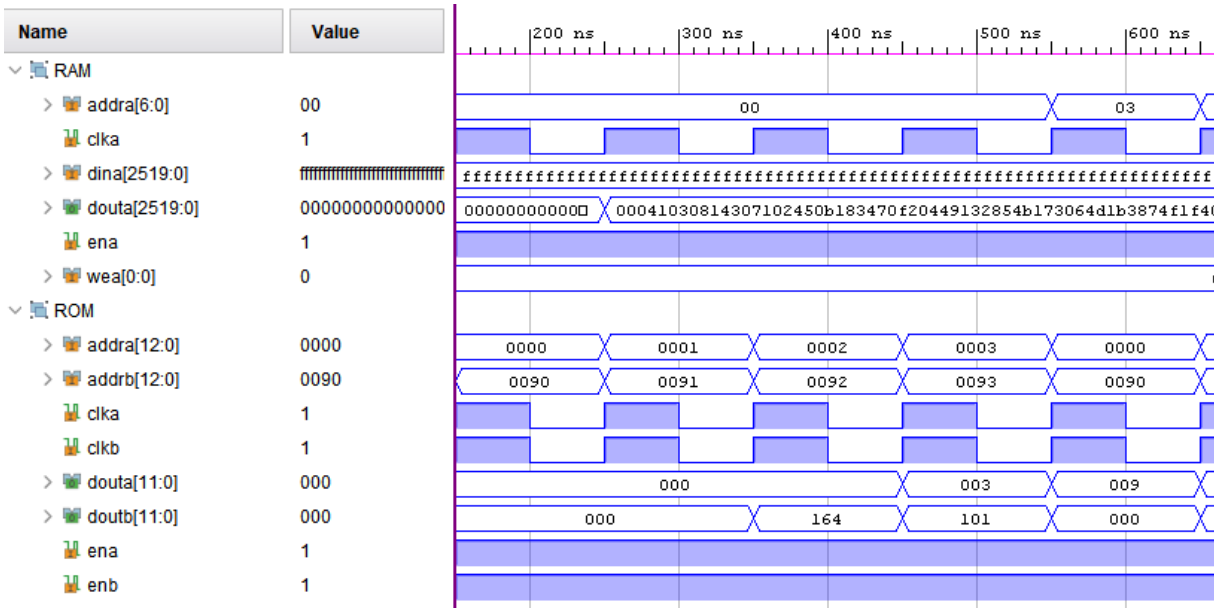


Figura 42 – Resultado de simulação das memorias.

impossibilidade, tomou-se como o consumo referente a arquitetura o consumo apresentado pela síntese.

4.3 Resultados de implementação

Os testes de implementação foram realizados na FPGA *NEXYS 4 - Xilinx Artix-7 FPGA XC7A100T-1CSG324C* a implementação foi divida em duas partes.

4.3.1 Resultados de implementação da ALU

Para a ALU deu-se bastante atenção aos testes na FPGA. Os testes realizados foram semelhantes aos esperados, sendo que todas as funcionalidades previstas funcionaram e rodaram sem problemas.

Na implementação encontrou-se para uma frequência de 200MHz um consumo de energia total igual a 0.121W, sendo 0.097W(80%) correspondente a energia estática, e os 20% restantes corresponde à potência dinâmica que considera o consumo dos *clocks*, sinais, lógica implementada, entradas e saídas.

Foram realizados outros testes até que se chegou a uma máxima frequência de 298.329 MHz, com um consumo de energia de 0.134W, sendo 0.097W(89%) correspondente a energia estática. Deu-se especial cuidado aos caminhos criticos e para as duas frequência, a tabela 15 referencia os dados obtidos.

Instancia	LUTs	FFs	DSPs	BRAMs	IO
Top level	39581 (62.43%)	13216 (10.42%)	4 (1.67%)	37.5	12620
ack_manager	2408 (3.80%)	8 (3.80%)	0	0	0
component_div_360	6840 (10.79%)	10460 (8.25%)	0	0	0
component_id_manager	7 (0.01%)	7 (0.01%)	0	0	0
control_adress	6 (0.01%)	7 (0.01%)	0	0	0
control_connect_map	71 (0.11%)	75 (0.06%)	0	0	0
control_position	201 (0.32%)	78 (0.06%)	4	0	0
mem_rom	22 (0.03%)	8 (0.01%)	0	2.5	0
memory_SO (ram)	0 (0.03%)	0 (0.01%)	0	35	0
shift_control	30012 (47.34%)	2580 (2.03%)	0	0	0

Tabela 14 – Consumo de recursos das arquiteturas para o modelo FPGA *NEXYS 4 - Artix-7 FPGA XC7A100T-1CSG324C*.

Freq. (MHz)	WNS (ns)	TNS (ns)	WHS (ns)	THS (ns)	TPWS (ns)	Cons. (W)
200	0.771	0.000	0.181	0.000	0.000	0.121
298.329	0.008	0.000	0.164	0.000	0.000	0.134

Tabela 15 – *Timming* e frequência para arquitetura ALU.

4.3.2 Resultados de implementação do bloco de *shift*

Os testes realizados foram semelhantes aos esperados, sendo que todas as funcionalidades previstas funcionaram e rodaram sem problemas.

Na implementação encontrou-se para uma frequência de 50MHz um consumo de energia total igual a 0.145W, sendo 0.099W(68%) correspondente a energia estática. Para uma frequência de 100MHz foram encontrados problemas de *timming* ocasionados pela complexidade presente no bloco *Shift_manager*.

Foram realizados diversos testes até que se chegou a uma máxima frequência de 60.606 MHz, com um consumo de energia de 0.162W, sendo 0.099W(61%) correspondente a energia estática. Esta respeitou todas as restrições de *timming* e na tabela 16 apresenta-se os dados referentes a cada frequência.

,

Freq. (MHz)	WNS (ns)	TNS (ns)	WHS (ns)	THS (ns)	TPWS (ns)	Cons. (W)
50	0.432	0.000	0.156	0.000	0.000	0.145
60.606	0.261	0.000	0.048	0.000	0.000	0.162
100	-6.638	-45.138	0.094	0.000	0.000	0.204

Tabela 16 – *Timming* e frequência para arquitetura *Shift*.

4.3.3 Implementação geral

Foi-se realizado uma implementação geral do bloco de *shift* que durou cerca de 12 horas até seu termino, porém está apresentou problemas de *timming*. Tendo sido o primeiro teste realizado operou-se com uma frequência de 100 MHz que como é possível ver em na subseção 4.3.2 é mais alta que a máxima frequência. As estrategias anteriores eram previstas e foram de fato adotadas pela inviabilidade de testar o sistema por inteiro, principalmente pela quantidade de IOs. Porém alguns resultados foram possíveis se obter, o mais importante consta na figura 43, que apresenta na área ocupada do *chip* FPGA.

4.4 Velocidade de arquitetura

Como dito na seção 3.6, e considerando informações obtidas na simulação, síntese e implementação, é possível obter dados importantes para a caracterização do sistema, que são a latência e a taxa de processamento *throughput*. Para o sistema da primeira varredura do LDPC tem-se para o processamento da entrada um total de 9 ciclos de *clock*, esse ciclos são recorrentes a todos os blocos apresentados em 15, sendo 1 ciclo para o *Control connect map*, 1 ciclo para o *control position*, 2 ciclos para o *Mem rom*, 1 ciclo para o *Control address*, 1 ciclo para o *Shift control* e 2 ciclos para o *Component div 360*. O *Memory SO* é acessada paralelamente nesse processo e o mesmo leva 2 ciclos de *clock* para disponibilizar a primeira saída, e 1 ciclo de *clock* para as saídas subsequentes, assim, como a memoria ROM. Para a parte do *shift* que considera até o passo onde a entrada é deslocada, tem-se 6 ciclos de *clock*, e para o bloco que contem o ALU tem-se 2 ciclos de *clock*.

Esse cenário considerado para a latência, se diz respeito a um único *layer* processado, e a quantidade desses varia de acordo com a *rate* usada podendo variar de acordo com o CN, isso pode ser melhor visualizado na figura 17 e no apêndice A. Para a taxa de processamento do $CODE\_RATE\_1\_4$ tem-se entre 3 e 4 conexões para cada CN, portanto, a latência total para a primeira varredura considerando todos as conexões presentes, é a soma da latência da saída para a primeira conexão com o total de conexões menos uma, já que a saída do processamento da conexão seguinte ocorre no ciclo de *clock* subsequente. Para a *rate* considerada tem-se um total de 135 conexões, assim, para a saída

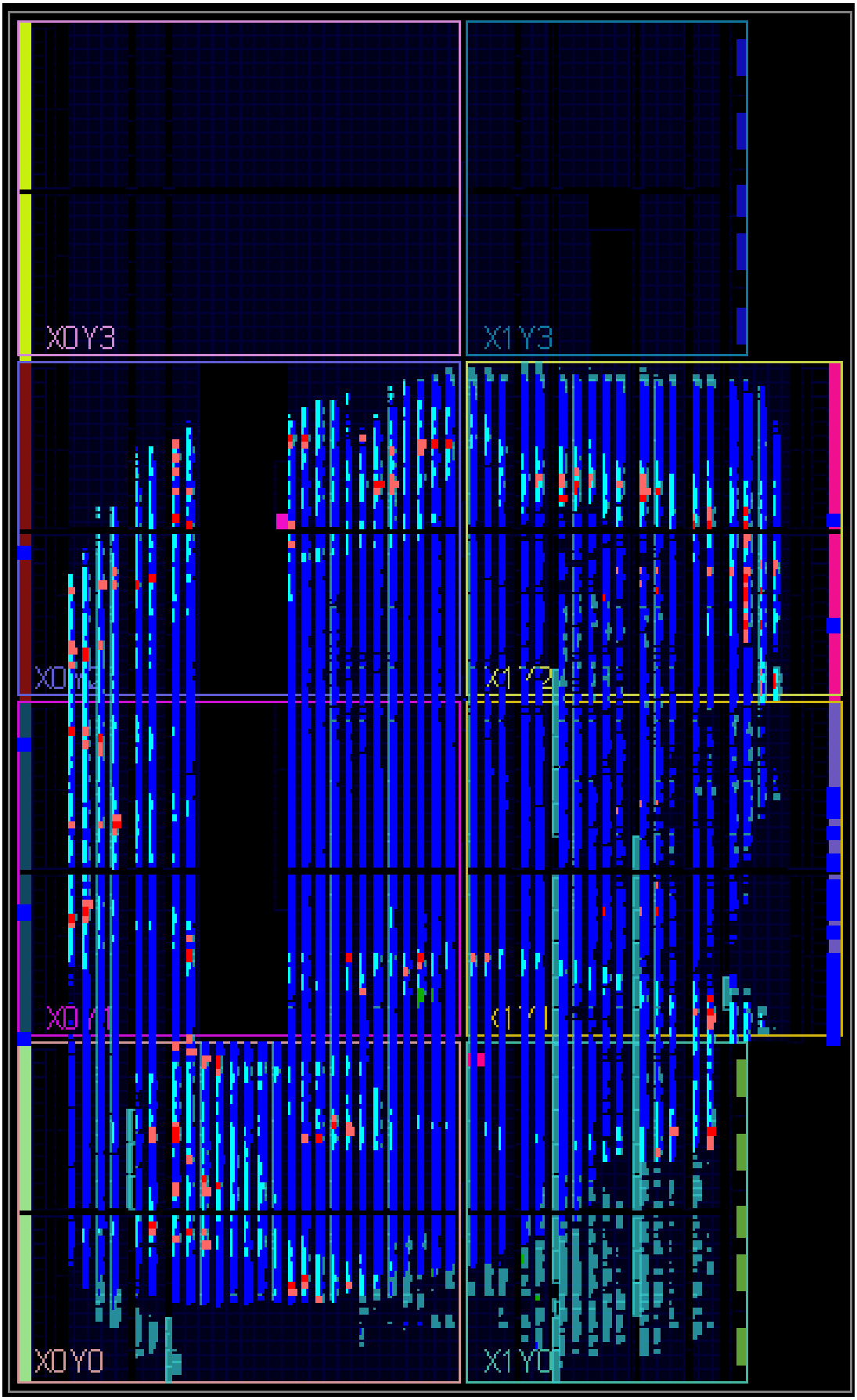


Figura 43 – Área ocupada na placa.

da primeira varredura tem-se um total de $135 - 1 + 9 = 143$ ciclos de *clock*.

A frequência máxima encontrada na implementação realizada em 4.3.2 para o deslocamento é de 60.606 MHz, essa frequência é a menor frequência do sistema. A entrada de cada conexão contem 2520 *bits* que são 360 SOs, o processamento desses 2520 *bits* leva um ciclo de *clock*, isso corresponde a $\frac{2520bits}{16.5ns} = 152.727$ Gbit/s o que equivalentemente é igual a 21.818 G"SO"/s. Toda via, o que interessa é saber a velocidade para que todo um *FRAME* seja processado, para a *rate* aqui considerada tem-se um total de 3240 SOs que é 22680 bits e como já mencionado há 135 conexões. Considerando o tempo que demora para processar todo uma "matriz"semelhante as presentes no apêndice A, se cada conexão demora 1 ciclo de *clock*, então as 135 conexões demorariam considerando a latência inicial 143 ciclos de *clock*, assim como na equação 4.1, isso para percorrer todas as conexões presentes no *rate*.

$$143(\frac{1}{60.606MHz}) = (143)(16.5ns) = 2.3595\ \mu s \tag{4.1}$$

Normalmente tem-se atrelado a esse processo e ao processo da segunda varredura 50 iterações, esse é o numero limite de iterações, ou seja, assim quando considerado uma *rate* por completo tem-se um tempo total de $(2.3595\mu s)(50) = 117.975\mu s$, isso desconsiderando a segunda varredura ainda não implementada. Com esse tempo serão processados $\frac{1}{117.975\mu s} = 8476.372$ *frames* por segundo, e se a *rate* tem um *frame* de 3240 bits, é processado um total de $(3240)(8476.372) = 27.463$ Mbits/s ou Mbps.

É importante atentar-se que esses cálculos consideram a latência inicial da arquitetura, mas para os *frames* seguintes quando considerado um funcionamento ininterrupto tem-se um total de 135 ciclos de *clock* por *frame*, disso o tempo necessário para percorrer uma única vez todas as conexões de uma matriz recorrente a *frame* está presenta na equação 4.2. Considerando um total de 50 iterações, o tempo total percorrido pode ser visto na equação 4.3, o total de vezes para cada *frame* completa por segundo pode ser visto na equação 4.4 e a taxa de processamento é apresentado na equação 2.15.

$$(135)(\frac{1}{60.606MHz}) = (135)(16.5ns) = 2.2275\mu s \tag{4.2}$$

$$(50)(2.2275\mu s) = 111.375\mu s \tag{4.3}$$

$$\frac{1}{111.375\mu s} = 8978.675\ \ frames\ \ por\ \ segundo \tag{4.4}$$

$$(3240)(8978.675) = 29.091 Mbits/s\ \ ou\ \ Mbps \tag{4.5}$$

4.4.1 Taxa de aceleração

Todo o trabalho de codificação como já dito foi realizado utilizando um modelo de referencia codificado com a linguagem C. Vale considerar as taxas de processamento de ambos os projetos, tal que se faça uma comparação entre os trabalhos. Para o modelo em C, ou seja, o modelo descrito em *software* que foi implementado em ARM com uma frequência de 666,67 MHz, teve-se uma taxa de processamento de 20Kbps. Enquanto para o modelo descrito em *hardware* a taxa é de cerca de 27.463Mbps, quando considerado a latência inicial. Vale conotar que o modelo descrito em *software* tem as duas varreduras implementadas e algumas diferentes em termos de comparação devem serem levadas em consideração.

Fazendo a comparação segundo a equação 4.6 tem-se uma aceleração dado os dados atuais de 1373.15 vezes. Essa diferença com a implementação no modelo em descrição de *hardware* deve cair em certa proporção ainda não mensurada.

$$Taxa\ \ de\ \ aceleração = \frac{27.463Mbps}{20Kbps} = 1373.15\ \ vezes \tag{4.6}$$

5 Conclusões

Este trabalho se manteve focado na construção parcial de um LDPC, que tem sua construção especifica para o padrão DVB-S2X. As arquiteturas foram construídas de forma a possibilitar o maior nível de paralelismo presente em diferente literaturas, onde cada *FRAME* é executado de uma única vez não tendo qualquer processo serial para qualquer parte do sistema. Sendo implementado uma arquitetura de *hardware* especifica da Unidade Logica Aritmética (ULA), e também da arquitetura de *hardware* do deslocamento(*SHIFT*) do LDPC.

O desenvolvimento do projeto em linguagem de descrição de hardware teve como principal referencia o modelo codificado em alto nível. Os algoritmos que se tinham, descritos em linguagem C, já estavam em um contexto próximo do que é realizado a nível de registradores o que possibilitou a verificação das características da implementação em hardware. Sendo possível desenhar uma arquitetura especifica de forma a dar a cada sub sistema uma tarefa condizente, o que possibilitou uma maior fluidez de construção.

De forma geral o trabalho se condiz na construção da primeira varredura do LDPC tal que para efeitos de comparação pode-se atentar no atual cenário o consumo presente e as máximas frequências que os blocos apresentam. No trabalho aqui realizado é possível perceber que há uma certa complexidade quanto ao deslocamento dos *bits* de acordo com o valor de *shift* recebido o bloco apresenta grande complexidade e consumo de recursos. Valendo ressaltar que foram testadas varias estrategias de arquiteturas, e em principio o consumo ultrapassava 119000 LUTs. A estrategia tomada tornou o consumo do bloco 30012 LUTs dado presente na tabela 14, dados encontrados em síntese. Usou-se na arquitetura uma função especifica da biblioteca *numeric.std*

Foi-se usado em todos os testes tanto para simulações quanto para as implementações o uso de arquivos tipo *.coe*, estes foram gerados em simulação de alto nível, isso proporcionou a validação mais rápida do funcionamento do sistema, e tornou o processo de simulação das arquiteturas menos trabalhoso e permitiu a implementação e testes na FPGA já que seria inviável se fosse necessario entrar com os sinais pelos periféricos da FPGA já que a entrada do *frame* tem 2520 *bits*. Os dados colocados como entradas foram obtidos por meio de um *script* no *Matlab*.

As simulação apresentaram o funcionamento da lógica do sistema, por meio das quais foi realizado a verificação de sincronia e funcionamento geral dos sistemas. Onde também foi possível obter outras informações tais como o tempo necessários para se ter uma saída válida, que foi de 8 ciclos de *clock*, e as saídas subsequentes que são obtidas no ciclo de *clock* seguinte. O que corresponde ao grau de paralelismo das arquiteturas.

Foi possível realizar a verificação da técnica de sincronização adotada no sistema FEC, baseado no reconhecimento de entrada e confirmação de recebimento.

A análise de síntese do mapeamento e roteamento em FPGA, permitiu a análise de parâmetros mais específicos dos circuitos, tais como *timming*, frequência máxima e consumo de potencia. Além do próprio consumo de área e de energia da própria FPGA *NEXYS 4* para diferentes frequências. A arquitetura utilizada atingiu uma frequência máxima de operação de 60.606MHz, que é limitada pela máxima frequência do deslocamento. A célula ALU atingiu uma frequência máxima de 298.329MHz. Esses valores foram obtidos levando em consideração as estrategias tomadas e descritas na seção 3.7.

Por fim, na subseção 4.4.1 é avaliado quanto mais rápido é a implementação do sistema em *hardware* para o codificado em *software*. O resultado preliminar apresenta um total de 1373.15 vezes quando considerado 50 iterações e a latência inicial do sistema, porém esse valor se condiz a o que a implementação da primeira varredura, que provavelmente diminuirá quando considerado a segunda varredura. Dessa forma com todo o exposto teve-se o desenvolvimento, implementação e verificação em FPGA do modelo integrado da primeira varredura do LDPC usando blocos de BRAMs da XILIX.

5.1 Considerações finais

A arquitetura desenvolvida neste trabalho se insere em um contexto da construção de um RDS, de forma que todo o padrão DVB-S2X é construído e o LDPC aqui tratado é parte integrante. No caso é necessário que a segunda varredura seja implementada para que o mesmo seja concluído. Sendo ainda necessário trabalhos de otimização e melhoramento para o sistema. Sendo que deve-se buscar um melhoramento em termos de uso de recursos, máxima frequência de *clock* e consumo de energia. Os trabalhos futuros consistirão em realizar na implementação da segunda varredura do LDPC sua otimização e o retrabalho deste com reconfiguração dinâmica, algo que é indispensável para um RDS.

Referências

BAPTISTA, R.; MARINS, C. N. M. Sistemas de comunicação via satélite operando em banda ka. *Outubro*, p. 16, 2012. Citado 2 vezes nas páginas 31 e 33.

BARROS, L. G. O rádio definido por software. Trabalho de Conclusão de Curso em Engenharia Elétrica - UnB, Brasília, Distrito Federal, 2007. Citado 2 vezes nas páginas 32 e 34.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Apress, 2014. Citado na página 59.

COSTA, J. C. et al. Plano de trabalho, fase 2 projeto unb-autotrac. *Acesso privado.*, 2017. Citado na página 25.

DILIGENT. *Nexys 4™ FPGA Board Reference Manual*. 2016. <https://store.digilentinc.com/nexys-4-artix-7-fpga-trainer-board-limited-time-see-nexys4-ddr/>. Acessado em 20/06/2019. Citado 4 vezes nas páginas 15, 17, 75 e 76.

DUARTE, O. C. M. B.; LEãO, F. L. *VSAT (Very Small Aperture Terminal)*. 2002. <http://www.gta.ufrj.br/grad/02_2/vsat>. Acessado em 26/09/2018. Citado 2 vezes nas páginas 17 e 33.

ETSI. *Digital Video Broadcasting (DVB)Implementationguidelines for the second generation system forBroadcasting, Interactive Services, News Gathering and otherbroadband satellite applications*. [S.l.], 2015. Citado 5 vezes nas páginas 15, 53, 54, 55 e 56.

ETSI, E. 302 307:"digital video broadcasting (DVB). *Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications*, v. 1, 2006. Citado 4 vezes nas páginas 15, 36, 37 e 38.

ETSI, E. *Digital video broadcasting (dvb); second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications(DVB-S2)*. [S.l.], 2009. Citado 2 vezes nas páginas 15 e 26.

ETSI, E. 302 307-2: Digital video broadcasting (DVB); second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications; part 2: DVB-S2 extensions (DVB-S2X). *Air Interface for S-band Mobile Interactive Multimedia (S-MIM)*, v. 1, 2015. Citado 2 vezes nas páginas 36 e 37.

ETSI, T. *Digital video broadcasting (DVB): Implementation guidelines for DVB terrestrial services*. [S.l.], 2004. Citado na página 61.

ETSI, T. 101 545-1. *Digital Video Broadcasting (DVB), Second Generation DVB Interactive Satellite System (DVB-RCS2). Part*, v. 1, 2014. Citado 2 vezes nas páginas 37 e 38.

HARUYAMA, S. Software-defined radio technologies. In: *Wireless Communication Technologies: New Multimedia Systems*. [S.l.]: Springer, 2002. p. 131–145. Citado 2 vezes nas páginas 17 e 35.

HAYKIN, S. *Sistemas de comunicação*. [S.l.]: Bookman, 2004. Citado 11 vezes nas páginas 15, 38, 39, 40, 41, 42, 43, 44, 45, 46 e 47.

ISLAM, M. R. et al. Optimized min-sum decoding algorithm for low density parity check codes. *International Journal of Advanced Computer Science and Applications*, v. 2, n. 12, p. 168–174, 2011. Citado 2 vezes nas páginas 51 e 52.

ISOMÄKI, P.; AVESSTA, N. *Technical Reports: An overview of software defined radio technologies*. [S.l.]: Publisher: Turku Centre for Computer Science, 2004. Citado 3 vezes nas páginas 27, 34 e 35.

KILTS, S. *Advanced FPGA design: architecture, implementation, and optimization*. [S.l.]: John Wiley & Sons, 2007. Citado na página 74.

KING, G. *Unifying political methodology: The likelihood theory of statistical inference*. [S.l.]: University of Michigan Press, 1998. Citado na página 61.

LATHI; DING. *Modern Digital and Analog Communication Systems*. 4. ed. [S.l.]: Oxford University Press, 2009. (The Oxford Series in Electrical and Computer Engineering). ISBN 0195331451. Citado 8 vezes nas páginas 15, 39, 40, 41, 47, 48, 49 e 50.

LIMA, A. G. M. Reconfigurabilidade em um sistema de comunicação móvel baseado na arquitetura de um rádio definido por software, tese (doutorado em engenharia elétrica) - universidade de brasília, brasilia, df. 2006. Citado 3 vezes nas páginas 27, 34 e 35.

LIMA, E. R. D. et al. A detailed DVB-S2 receiver implementation: Fpga prototyping and preliminary asic resource estimation. In: IEEE. *2014 IEEE Latin-America Conference on Communications (LATINCOM)*. [S.l.], 2014. p. 1–6. Citado 2 vezes nas páginas 56 e 57.

LOI, K. C. C. *Field-programmable Gate-array (FPGA) Implementation of Low-density Parity-check (LDPC) Decoder in Digital Video Broadcasting-Second Generation Satellite (DVB-S2)*. Tese (Doutorado) — University of Saskatchewan, 2010. Citado 2 vezes nas páginas 56 e 57.

MARAL, G.; BOUSQUET, M. *Satellite Communications Systems: Systems, Techniques and Technology, 5a ed.* [S.l.]: Wiley, 2002. (Novartis Foundation Symposium). ISBN 9780471496540. Citado 3 vezes nas páginas 25, 27 e 36.

MARCHAND, C. *Implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards*. Tese (Doutorado), 12 2010. Citado 2 vezes nas páginas 15 e 63.

MARINS, C. N. M. et al. Estudo analítico e numérico de um enlace digital de comunicação via satélite em condição orbital geoestacionária. Instituto Nacional de Telecomunicações, 2004. Citado na página 26.

MATTOS, W. D. d. Códigos de verificação de erros de paridade de baixa densidade (LDPC). Universidade Tecnológica Federal do Paraná, 2012. Citado 4 vezes nas páginas 27, 41, 42 e 43.

MENGARDA, A. C. et al. Core LDPC para o padrão DVB-S2-digital video broadcasting-satellite generation 2. Pontifícia Universidade Católica do Rio Grande do Sul, 2016. Citado 2 vezes nas páginas 56 e 57.

MOLYNEAUX, I. *The art of application performance testing: Help for programmers and quality assurance*. [S.l.]: "O'Reilly Media, Inc.", 2009. Citado na página 26.

PIMENTEL, C. J. L. *Comunicação Digital*. [S.l.]: Brasport, 2007. Citado na página 26.

RICHARDSON, T. Error floors of LDPC codes. In: THE UNIVERSITY; 1998. *Proceedings of the annual Allerton conference on communication control and computing*. [S.l.], 2003. v. 41, n. 3, p. 1426–1435. Citado na página 61.

SANTOS, L. F. d. et al. Decodificadores de baixa complexidade para códigos LDPC q-ários. [sn], 2014. Citado na página 40.

SBIZERA, D. d. O. *Um sistema de comunicação para transmissão de dados a longa distância em aeronaves do Projeto ARARA*. Dissertação (Mestrado) — (Ciências da Computação e Matemática Computacional) - Universidade de São Paulo, São Carlos, SP. 2003, 2003. Citado na página 31.

SCALISE, S. et al. S-mim: a novel radio interface for efficient messaging services over satellite. *IEEE Communications Magazine*, IEEE, v. 51, n. 3, p. 119–125, 2013. Citado 2 vezes nas páginas 15 e 32.

SHANNON, C. E. A mathematical theory of communication. *Bell system technical journal*, Wiley Online Library, v. 27, n. 3, p. 379–423, 1948. Citado na página 25.

TUTTLEBEE, W. H. *Software defined radio: enabling technologies, 1a Ed.* [S.l.]: John Wiley & Sons, 2003. Citado 2 vezes nas páginas 32 e 34.

ZELENOVSKY, R.; MENDONÇA, A. *ELETRONICA DIGITAL: CURSO PRATICO E EXERCICIOS, 1a ED.* [S.l.]: MZ EDITORA, 2007. ISBN 9788587385130. Citado na página 35.

Apêndices

APÊNDICE A – *LAYERS MAP*, arquivo *.coe*

$memory\_initialization\_radix = 10;$

$memory\_initialization\_vector =$

; ::

; $constantid\_1\_4 : type\_id\_1\_4 :=$

0,3,9,44,3,5,10,9,1,1,11,10,2,7,12,11,0,0,13,12,1,3,14,13,3,15,14,999,0,1,16,15,
2,2,17,16,4,8,18,17,2,2,19,18,0,1,20,19,3,8,21,20,2,22,21,999,0,23,22,999,0,24,
23,999,0,1,25,24,1,2,26,25,5,8,27,26,1,4,28,27,1,29,28,999,2,3,30,29,3,6,31,30,
0,5,32,31,1,2,33,32,2,3,34,33,1,35,34,999,0,36,35,999,6,37,36,999,0,7,38,37,3,
39,38,999,0,2,40,39,3,4,41,40,3,7,42,41,2,6,43,42,1,3,44,43,

; $constantshift\_1\_4 : type\_shift\_1\_4 :=$

356,257,0,359,171,266,0,0,263,52,0,0,159,295,0,0,223,314,0,0,136,194,0,0,118,0,
0,999,49,253,0,0,147,241,0,0,61,222,0,0,61,286,0,0,184,275,0,0,224,87,0,0,260,0,
0,999,93,0,0,999,226,0,0,999,352,130,0,0,74,50,0,0,59,7,0,0,147,275,0,0,222,0,0,
999,262,70,0,0,268,27,0,0,206,201,0,0,195,216,0,0,20,31,0,0,257,0,0,999,147,0,0,
999,323,0,0,999,12,243,0,0,65,0,0,999,186,178,0,0,52,143,0,0,103,189,0,0,118,22,
0,0,64,288,0,0,

; ::

; $constantid\_1\_3 : type\_id\_1\_3 :=$

2,3,10,15,44,1,4,4,16,15,0,4,6,17,16,0,3,9,18,17,2,12,14,19,18,1,3,5,20,19,0,1,1,
21,20,2,3,11,22,21,1,5,13,23,22,2,2,11,24,23,0,3,8,25,24,1,2,7,26,25,0,1,5,27,26,
0,4,12,28,27,2,8,11,29,28,0,2,4,30,29,0,3,4,31,30,1,14,14,32,31,1,2,10,33,32,0,1,
4,34,33,3,4,6,35,34,3,4,7,36,35,0,3,9,37,36,2,3,6,38,37,2,4,10,39,38,1,3,7,40,39,
0,2,4,41,40,3,12,13,42,41,1,9,13,43,42,0,4,8,44,43,

; $constantshift\_1\_3 : type\_shift\_1\_3 :=$

217,232,29,0,359,159,75,299,0,0,263,11,121,0,0,136,237,108,0,0,323,303,249,0,0,

241,153,310,0,0,253,147,286,0,0,22,267,270,0,0,61,101,231,0,0,25,189,40,0,0,275,
189,18,0,0,260,284,229,0,0,130,50,171,0,0,74,181,131,0,0,346,330,199,0,0,147,295,
310,0,0,222,70,217,0,0,262,23,85,0,0,20,27,192,0,0,195,216,198,0,0,180,157,133,0,
0,296,246,118,0,0,257,208,240,0,0,248,336,140,0,0,48,114,209,0,0,178,107,244,0,0,
347,243,196,0,0,125,253,27,0,0,118,48,202,0,0,64,42,72,0,0,

; ::

;$constantid\_2\_5 : type\_id\_2\_5 :=$

3,4,5,17,18,44,1,3,8,14,19,18,0,5,5,13,20,19,1,2,2,14,21,20,3,6,8,16,22,21,4,4,6,
12,23,22,2,3,4,13,24,23,0,3,5,12,25,24,2,3,4,9,26,25,0,5,7,10,27,26,0,1,5,17,28,
27,1,4,5,15,29,28,0,0,2,9,30,29,1,3,5,11,31,30,0,2,5,17,32,31,1,2,5,11,33,32,0,0,
2,12,34,33,3,3,6,10,35,34,3,5,7,11,36,35,1,4,7,16,37,36,1,2,4,15,38,37,0,1,4,10,
39,38,2,2,4,13,40,39,2,3,5,15,41,40,0,1,4,14,42,41,0,3,8,16,43,42,1,1,4,9,44,43,

;$constantshift\_2\_5 : type\_shift\_2\_5 :=$

196,348,1,30,0,359,174,220,188,111,0,0,36,293,244,162,0,0,104,279,291,151,0,0,208,
307,45,171,0,0,143,68,299,177,0,0,28,108,123,20,0,0,151,15,211,87,0,0,133,143,203,
28,0,0,333,267,118,116,0,0,104,345,170,309,0,0,298,352,289,196,0,0,207,295,33,146,
0,0,264,277,278,218,0,0,337,289,342,262,0,0,295,332,15,17,0,0,339,114,72,39,0,0,80,
73,58,294,0,0,56,182,236,211,0,0,153,34,222,11,0,0,329,294,199,194,0,0,311,242,3,
197,0,0,92,328,184,287,0,0,213,161,331,47,0,0,112,64,328,27,0,0,62,34,17,89,0,0,
150,101,29,87,0,0,

; ::

;$constantid\_1\_2 : type\_id\_1\_2 :=$

5,9,19,20,44,999,999,1,2,2,6,21,20,999,3,3,4,7,14,22,21,1,4,8,17,23,22,999,0,9,24,
23,999,999,999,6,10,13,25,24,999,999,11,13,26,25,999,999,999,2,12,18,27,26,999,999,
0,8,13,14,28,27,999,5,14,15,29,28,999,999,4,15,16,30,29,999,999,0,0,16,19,31,30,999,
0,0,7,17,32,31,999,2,4,18,18,33,32,999,1,8,17,19,34,33,999,2,4,10,35,34,999,999,3,6,
36,35,999,999,999,3,4,11,12,37,36,999,1,2,15,38,37,999,999,2,7,39,38,999,999,999,0,
0,3,12,40,39,999,1,5,16,41,40,999,999,1,2,4,9,42,41,999,1,1,3,43,42,999,999,3,3,4,10,
11,44,43,

;$constantshift\_1\_2 : type\_shift\_1\_2 :=$

0,206,222,0,359,999,999,125,323,132,0,0,0,999,248,217,112,0,45,0,0,107,280,0,239,
0,0,999,106,0,0,0,999,999,999,246,0,237,0,0,999,999,0,176,0,0,999,999,999,220,0,
318,0,0,999,999,154,314,0,175,0,0,999,83,0,205,0,0,999,999,313,0,3,0,0,999,999,265,
198,0,64,0,0,999,332,318,352,0,0,0,999,263,310,0,121,0,0,999,237,223,330,0,0,0,999,
233,155,349,0,0,999,999,317,358,0,0,999,999,999,174,171,302,271,0,0,999,259,213,86,
0,0,999,999,350,93,0,0,999,999,999,0,159,180,48,0,0,999,0,199,161,0,0,999,999,168,
0,101,184,0,0,999,131,267,0,0,0,999,999,148,183,0,124,199,0,0,
; ::
; $constantid\_3\_5 : type\_id\_3\_5 :=$
0,2,3,6,8,8,9,11,18,27,44,1,2,4,5,6,8,9,10,26,28,27,1,2,2,6,8,8,11,16,25,29,28,1,1,2,
4,5,5,10,12,22,30,29,0,0,1,8,8,8,13,17,24,31,30,0,1,3,3,4,7,14,14,21,32,31,0,2,5,5,7,
8,15,15,23,33,32,0,2,3,4,6,6,9,16,19,34,33,0,5,6,6,7,7,16,17,21,35,34,3,4,4,5,5,6,13,
18,23,36,35,0,2,3,4,5,7,15,19,22,37,36,0,3,3,4,6,7,12,20,20,38,37,1,1,2,3,5,7,10,21,
26,39,38,1,1,5,7,7,8,12,20,22,40,39,0,0,2,3,4,7,13,18,23,41,40,0,1,3,6,6,7,17,24,25,
42,41,2,3,4,4,4,5,14,24,25,43,42,1,2,6,7,8,8,11,19,26,44,43,
; $constantshift\_3\_5 : type\_shift\_3\_5 :=$
3,188,122,148,241,338,0,221,89,0,359,85,1,276,249,38,134,240,0,128,0,0,338,9,137,
270,46,311,0,307,107,0,0,320,49,298,16,189,33,279,0,126,0,0,239,330,291,203,261,97,
0,44,172,0,0,93,215,96,329,357,298,0,264,199,0,0,284,66,212,118,239,285,0,150,71,
0,0,43,86,185,127,315,232,213,0,180,0,0,121,149,54,199,47,89,122,0,161,0,0,129,194,
149,75,60,178,28,0,99,0,0,172,14,171,112,153,26,253,0,218,0,0,207,246,225,265,312,
231,140,0,96,0,0,238,198,349,46,6,258,284,0,88,0,0,349,346,38,266,161,143,302,142,0,
0,0,161,203,98,141,337,125,24,346,0,0,0,318,42,25,291,164,60,171,0,77,0,0,123,9,226,
320,98,109,26,142,0,0,0,28,40,294,345,289,197,155,88,0,0,0,
; ::
; $constantid\_2\_3 : type\_id\_2\_3 :=$
0,0,1,11,11,15,18,23,30,44,0,0,1,1,10,15,16,22,31,30,1,1,2,2,4,17,19,21,32,31,0,1,3,
7,14,16,18,25,33,32,1,1,2,3,4,19,28,29,34,33,0,2,5,5,6,20,24,27,35,34,0,2,5,6,12,21,
22,28,36,35,0,2,7,9,9,19,22,23,37,36,0,1,2,8,13,18,23,26,38,37,0,2,2,6,9,24,26,29,39,
38,2,2,10,12,14,16,20,25,40,39,0,1,8,8,11,21,24,26,41,40,0,1,3,7,12,17,20,27,42,41,1,

1,4,10,13,15,27,28,43,42,0,2,2,13,14,17,25,29,44,43,

;$constantshift\_2\_3 : type_shift\_2\_3 :=$

0,52,168,171,313,0,293,256,0,359,147,130,0,259,53,191,0,345,0,0,352,237,0,353,217,0,19,
337,0,0,257,153,0,344,244,188,0,165,0,0,107,125,343,125,0,0,248,285,0,0,263,152,0,235,
33,0,251,318,0,0,195,94,38,0,132,0,74,345,0,0,74,8,0,216,193,13,0,328,0,0,253,70,105,
0,104,66,0,323,0,0,136,134,303,103,0,0,203,101,0,0,184,204,0,203,270,282,217,0,0,0,275,
296,160,265,0,181,99,0,0,0,186,267,338,258,0,258,166,0,0,0,131,336,81,294,0,350,315,0,
0,0,222,299,350,196,0,327,351,0,0,0,

; ::

;$constantid\_3\_4 : type\_id\_3\_4 :=$

0,8,9,11,12,21,27,32,33,44,999,999,999,0,0,3,5,10,17,20,22,24,25,34,33,999,0,4,7,11,13,
16,23,26,29,35,34,999,999,0,10,12,14,24,24,30,36,35,999,999,999,999,0,1,13,18,18,25,31,
31,37,36,999,999,999,0,0,1,2,4,12,13,14,23,26,28,38,37,0,3,6,9,15,19,26,27,30,39,38,999,
999,0,2,4,8,9,16,17,21,25,28,40,39,999,5,5,6,10,15,17,21,22,29,41,40,999,999,3,6,11,15,
18,22,29,30,42,41,999,999,999,0,7,7,14,16,19,27,28,31,43,42,999,999,0,1,2,8,19,20,20,23,
32,32,44,43,999,

;$constantshift\_3\_4 : type\_shift\_3\_4 :=$

142,278,0,217,349,0,149,126,0,359,999,999,999,276,210,79,16,0,198,15,0,277,80,0,0,999,
314,41,271,0,307,315,0,278,252,0,0,999,999,0,110,0,73,0,35,309,0,0,999,999,999,999,
200,0,0,198,326,0,223,101,0,0,999,999,999,237,74,137,0,73,50,209,0,301,0,160,0,0,94,0,
132,239,0,174,268,0,346,0,0,999,999,10,25,0,174,108,0,276,104,167,0,0,0,999,0,189,12,
250,58,0,154,270,0,0,0,999,999,141,0,266,256,0,109,204,0,0,0,999,999,999,321,0,32,213,
53,0,349,220,0,0,0,999,999,304,183,102,0,278,0,202,246,0,243,0,0,999,

; ::

;$constantid\_4\_5 : type\_id\_4\_5 :=$

2,5,6,10,15,16,16,25,27,31,35,44,999,6,12,14,16,17,22,26,26,27,36,35,999,999,2,4,7,9,
13,15,17,18,25,27,28,37,36,1,5,8,8,18,20,20,28,28,30,38,37,999,1,7,9,11,19,21,23,25,
29,30,39,38,999,0,0,6,7,10,18,20,24,30,31,34,40,39,0,1,10,11,11,21,21,24,31,32,33,41,
40,2,3,5,12,14,15,22,23,26,32,33,42,41,3,4,12,13,13,17,19,23,29,33,34,43,42,3,4,8,9,
14,19,22,24,29,32,34,44,43,

; $constantshift\_4\_5 : type\_shift\_4\_5 :=$

39,0,337,149,0,180,32,0,199,14,0,359,999,0,265,218,0,327,348,0,172,90,0,0,999,999,
339,299,0,274,6,133,0,105,288,0,187,0,0,111,94,0,332,0,222,326,0,32,336,0,0,999,
342,81,0,157,0,351,83,202,0,53,0,0,999,0,204,91,158,0,314,0,175,0,266,6,0,0,271,
0,323,0,132,0,337,149,0,186,304,0,0,0,288,166,0,330,241,0,211,84,0,218,0,0,0,18,
154,0,50,130,257,0,344,0,244,0,0,227,0,57,72,0,113,99,0,251,156,0,0,0,

; ::

; $constantid\_5\_6 : type\_id\_5\_6 :=$

0,2,5,5,8,13,15,15,21,21,26,29,34,36,37,44,999,999,999,0,0,6,7,12,14,14,19,22,23,
24,29,30,30,38,37,999,999,999,3,3,7,9,11,15,16,17,21,23,23,31,31,33,39,38,999,
999,999,0,0,7,8,12,16,18,20,24,26,27,30,32,35,40,39,999,999,999,0,0,0,1,4,6,9,10,
17,17,19,25,25,28,33,33,36,41,40,0,0,2,4,9,10,11,13,13,18,22,22,26,29,34,35,42,
41,999,0,0,1,2,3,6,10,11,18,19,20,24,27,27,31,32,35,43,42,0,1,4,5,8,12,14,16,20,
25,28,28,32,34,36,44,43,999,999,

; $constantshift\_5\_6 : type\_shift\_5\_6 :=$

77,207,0,96,202,0,148,222,0,328,70,0,61,147,0,359,999,999,999,59,175,0,292,350,
0,339,298,0,203,207,64,0,353,0,0,999,999,999,96,255,0,78,346,0,121,179,50,0,153,
0,229,143,0,0,999,999,999,0,298,258,0,80,0,349,163,0,26,306,300,0,340,0,0,999,
999,999,247,271,47,0,334,67,0,240,0,294,82,0,238,332,0,352,30,0,0,319,10,0,289,
148,0,59,205,163,0,221,203,0,125,0,280,0,0,999,202,148,120,309,0,75,340,0,239,0,
70,176,0,201,256,112,0,0,0,291,55,0,63,343,0,264,7,0,47,0,298,207,242,0,0,0,999,
999,

; ::

; $constantid\_8\_9 : type\_id\_8\_9 :=$

0,1,2,3,4,6,13,21,24,44,0,1,2,3,4,5,14,22,25,24,0,1,2,3,4,8,12,19,26,25,0,1,2,
3,4,9,11,19,27,26,0,1,2,3,4,9,15,20,28,27,0,1,2,3,4,5,16,19,29,28,0,1,2,3,4,7,
12,22,30,29,0,1,2,3,4,7,15,23,31,30,0,1,2,3,4,6,16,18,32,31,0,1,2,3,4,5,12,23,
33,32,0,1,2,3,4,6,14,17,34,33,0,1,2,3,4,7,10,17,35,34,0,1,2,3,4,6,13,17,36,35,
0,1,2,3,4,8,14,21,37,36,0,1,2,3,4,5,11,18,38,37,0,1,2,3,4,9,15,18,39,38,0,1,2,
3,4,9,10,22,40,39,0,1,2,3,4,16,20,41,40,999,0,1,2,3,4,8,10,20,42,41,0,1,2,3,4,

7,11,21,43,42,0,1,2,3,4,8,13,23,44,43,

; $constantshift\_8\_9 : type\_shift\_8\_9 :=$

138,4,17,38,222,115,11,259,0,359,122,27,83,227,141,166,228,212,0,0,266,22,73,

79,5,116,88,7,0,0,332,275,81,37,328,70,103,34,0,0,265,4,103,44,321,225,220,1,

0,0,187,32,76,7,17,12,140,284,0,0,342,88,105,200,316,208,114,120,0,0,18,8,63,

333,5,129,232,312,0,0,114,82,221,25,318,20,207,10,0,0,89,93,44,83,298,161,

262,5,0,0,340,146,226,201,352,3,275,280,0,0,359,14,29,262,13,28,0,69,0,0,257,

271,132,291,303,74,19,36,0,0,147,50,125,34,154,32,194,35,0,0,160,126,156,247,

0,1,16,35,0,0,299,231,13,1,17,24,40,122,0,0,259,90,2,125,113,6,103,268,0,0,22,

124,161,83,52,80,257,0,0,999,29,24,90,7,49,93,183,9,0,0,21,2,228,263,29,26,

242,25,0,0,156,270,6,274,6,210,271,348,0,0,

; ::

; $constantid\_11\_45 : type\_id\_11\_45 :=$

0,6,11,44,4,8,12,11,4,6,13,12,9,10,14,13,2,4,15,14,0,3,16,15,0,1,17,16,2,3,

18,17,8,9,19,18,0,1,20,19,0,2,21,20,7,8,22,21,3,5,23,22,1,2,24,23,0,6,25,24,

2,4,26,25,2,7,27,26,0,4,28,27,3,4,29,28,1,3,30,29,1,2,31,30,5,7,32,31,1,3,

33,32,1,3,34,33,4,10,35,34,4,10,36,35,3,5,37,36,3,4,38,37,1,2,39,38,2,4,40,

39,0,3,41,40,0,9,42,41,1,2,43,42,0,1,44,43,

; $constantshift\_11\_45 : type\_shift\_11\_45 :=$

272,347,0,359,109,49,0,0,113,139,0,0,157,305,0,0,300,266,0,0,143,209,0,0,90,

283,0,0,220,111,0,0,318,107,0,0,121,232,0,0,94,194,0,0,23,274,0,0,208,52,0,0,

122,256,0,0,331,251,0,0,115,271,0,0,276,37,0,0,3,9,0,0,334,330,0,0,38,212,0,

0,36,138,0,0,205,338,0,0,205,79,0,0,89,305,0,0,118,195,0,0,334,173,0,0,123,

41,0,0,262,249,0,0,9,8,0,0,103,226,0,0,30,156,0,0,222,29,0,0,1,155,0,0,171,

151,0,0,

; ::

; $constantid\_4\_15 : type\_id\_4\_15 :=$

0,2,6,12,44,0,2,8,13,12,1,2,5,14,13,0,1,6,15,14,0,1,8,16,15,0,1,3,17,16,0,

2,9,18,17,0,1,5,19,18,0,1,10,20,19,1,2,3,21,20,1,2,8,22,21,0,1,3,23,22,1,2,

7,24,23,0,1,4,25,24,2,10,26,25,999,0,1,3,27,26,0,2,7,28,27,0,1,7,29,28,0,2,

6,30,29,1,2,11,31,30,2,4,32,31,999,0,2,9,33,32,1,7,34,33,999,0,2,9,35,34,0,
1,4,36,35,1,2,5,37,36,1,2,5,38,37,0,1,4,39,38,0,2,8,40,39,0,2,11,41,40,1,2,
6,42,41,1,2,11,43,42,0,2,10,44,43,
; $constantshift\_4\_15 : type\_shift\_4\_15 :=$
219,193,4,0,359,105,223,291,0,0,73,260,99,0,0,55,203,117,0,0,283,44,247,0,0,
127,22,58,0,0,301,154,224,0,0,82,122,154,0,0,70,84,26,0,0,132,61,301,0,0,288,
83,57,0,0,152,11,87,0,0,286,136,7,0,0,95,298,0,0,0,291,6,0,0,999,130,9,131,
0,0,281,321,44,0,0,6,296,92,0,0,71,250,57,0,0,345,308,43,0,0,90,135,0,0,999,
290,346,119,0,0,27,211,0,0,999,165,115,38,0,0,188,160,107,0,0,337,64,22,0,0,
16,160,153,0,0,122,154,313,0,0,13,34,171,0,0,209,119,266,0,0,258,126,119,0,
0,70,233,59,0,0,60,1,91,0,0,
; ::
; $constantid\_14\_45 : type\_id\_14\_45 :=$
6,9,11,14,44,2,3,4,15,14,0,1,3,16,15,0,1,3,17,16,2,3,4,18,17,3,5,13,19,18,
0,2,3,20,19,4,5,10,21,20,0,2,12,22,21,0,4,9,23,22,0,4,6,24,23,0,3,5,25,24,
1,7,8,26,25,1,2,4,27,26,4,10,12,28,27,5,6,11,29,28,3,4,5,30,29,1,3,7,31,30,
1,2,4,32,31,1,8,12,33,32,1,2,4,34,33,0,3,4,35,34,0,2,5,36,35,2,5,13,37,36,
0,1,2,38,37,0,2,9,39,38,4,5,8,40,39,2,5,13,41,40,0,1,7,42,41,1,3,11,43,42,
1,3,10,44,43,
; $constantshift\_14\_45 : type\_shift\_14\_45 :=$
187,204,50,0,359,40,177,341,0,0,63,66,97,0,0,223,210,315,0,0,263,190,265,
0,0,228,330,255,0,0,103,312,275,0,0,149,269,235,0,0,213,317,1,0,0,54,326,
142,0,0,143,0,130,0,0,108,107,155,0,0,96,204,188,0,0,228,87,340,0,0,175,5,
157,0,0,259,120,228,0,0,84,274,115,0,0,235,125,345,0,0,146,109,176,0,0,6,
308,221,0,0,113,185,165,0,0,244,240,354,0,0,58,196,358,0,0,201,207,106,0,
0,225,331,156,0,0,309,334,176,0,0,293,356,88,0,0,150,348,238,0,0,348,246,
61,0,0,179,178,185,0,0,329,2,217,0,0,
;::
; $constantid\_7\_15 : type\_id\_7\_15 :=$
0,1,2,3,5,9,13,21,44,0,1,2,3,7,18,22,21,999,0,1,2,3,10,13,23,22,999,0,1,

2,3,10,16,24,23,999,0,1,2,3,4,7,13,25,24,0,1,2,3,11,16,26,25,999,0,1,2,3,
10,18,27,26,999,0,1,2,3,4,8,15,28,27,0,1,2,3,5,8,17,29,28,0,1,2,3,6,12,20,
30,29,0,1,2,3,5,12,15,31,30,0,1,2,3,5,11,19,32,31,0,1,2,3,7,19,33,32,999,
0,1,2,3,11,20,34,33,999,0,1,2,3,6,14,35,34,999,0,1,2,3,4,9,15,36,35,0,1,2,
3,8,20,37,36,999,0,1,2,3,6,9,17,38,37,0,1,2,3,11,18,39,38,999,0,1,2,3,9,
14,40,39,999,0,1,2,3,4,12,14,41,40,0,1,2,3,10,16,42,41,999,0,1,2,3,8,17,
43,42,999,0,1,2,3,6,7,19,44,43,

; $constantshift\_7\_15 : type\_shift\_7\_15 :=$

1,247,263,8,235,108,229,0,359,226,109,301,247,33,58,0,0,999,347,25,5,98,2,
140,0,0,999,0,102,159,216,252,15,0,0,999,61,149,93,226,23,4,167,0,0,189,181,
24,120,46,352,0,0,999,5,11,22,15,50,46,0,0,999,88,281,129,233,1,225,28,0,0,
87,310,28,11,358,253,1,0,0,14,213,79,117,285,112,277,0,0,103,2,94,36,17,0,
222,0,0,22,190,314,93,279,5,19,0,0,276,97,47,237,268,271,0,0,999,294,76,202,
129,273,152,0,0,999,36,82,3,247,190,11,0,0,999,347,284,40,4,30,15,143,0,0,52,
232,112,75,70,320,0,0,999,355,38,62,30,7,270,136,0,0,63,95,0,331,206,323,0,0,
999,234,3,353,5,36,116,0,0,999,110,346,265,21,217,49,123,0,0,32,311,166,87,
302,167,0,0,999,1,46,215,222,319,336,0,0,999,320,156,300,14,28,92,310,0,0,

; ::

; $constantid\_8\_15 : type\_id\_8\_15 :=$

0,1,2,3,4,6,13,21,24,44,0,1,2,3,4,5,14,22,25,24,0,1,2,3,4,8,12,19,26,25,0,1,
2,3,4,9,11,19,27,26,0,1,2,3,4,9,15,20,28,27,0,1,2,3,4,5,16,19,29,28,0,1,2,3,
4,7,12,22,30,29,0,1,2,3,4,7,15,23,31,30,0,1,2,3,4,6,16,18,32,31,0,1,2,3,4,5,
12,23,33,32,0,1,2,3,4,6,14,17,34,33,0,1,2,3,4,7,10,17,35,34,0,1,2,3,4,6,13,
17,36,35,0,1,2,3,4,8,14,21,37,36,0,1,2,3,4,5,11,18,38,37,0,1,2,3,4,9,15,18,
39,38,0,1,2,3,4,9,10,22,40,39,0,1,2,3,4,16,20,41,40,999,0,1,2,3,4,8,10,20,
42,41,0,1,2,3,4,7,11,21,43,42,0,1,2,3,4,8,13,23,44,43,

; $constantshift\_8\_15 : type\_shift\_8\_15 :=$

138,4,17,38,222,115,11,259,0,359,122,27,83,227,141,166,228,212,0,0,266,22,
73,79,5,116,88,7,0,0,332,275,81,37,328,70,103,34,0,0,265,4,103,44,321,225,
220,1,0,0,187,32,76,7,17,12,140,284,0,0,342,88,105,200,316,208,114,120,0,0,

18,8,63,333,5,129,232,312,0,0,114,82,221,25,318,20,207,10,0,0,89,93,44,83,
298,161,262,5,0,0,340,146,226,201,352,3,275,280,0,0,359,14,29,262,13,28,0,
69,0,0,257,271,132,291,303,74,19,36,0,0,147,50,125,34,154,32,194,35,0,0,
160,126,156,247,0,1,16,35,0,0,299,231,13,1,17,24,40,122,0,0,259,90,2,125,
113,6,103,268,0,0,22,124,161,83,52,80,257,0,0,999,29,24,90,7,49,93,183,9,0,
0,21,2,228,263,29,26,242,25,0,0,156,270,6,274,6,210,271,348,0,0,
; ::
; $constantid\_26\_45 : type\_id\_26\_45 :=$
0,1,2,4,5,6,7,13,26,44,0,2,3,4,5,6,7,11,27,26,0,2,4,7,8,11,15,18,28,27,1,6,
7,8,9,12,21,22,29,28,0,1,3,4,5,6,7,17,30,29,0,2,3,5,6,7,16,23,31,30,1,2,3,6,
7,9,14,19,32,31,0,2,4,5,6,7,12,20,33,32,0,3,4,6,16,20,21,22,34,33,1,2,4,5,6,
16,24,25,35,34,1,4,5,13,17,19,23,24,36,35,1,2,3,4,5,12,13,21,37,36,0,1,7,8,
10,18,22,23,38,37,0,1,3,4,6,9,14,15,39,38,0,1,2,3,5,7,14,15,40,39,0,1,2,3,5,
6,7,10,41,40,0,2,3,5,10,17,18,20,42,41,0,1,2,3,4,6,11,25,43,42,1,3,4,5,7,19,
24,25,44,43,
; $constantshift\_26\_45 : type\_shift\_26\_45 :=$
273,37,257,344,0,248,3,321,0,359,246,237,348,13,295,180,0,193,0,0,37,180,209,
354,15,91,153,35,0,0,139,280,68,175,316,76,0,141,0,0,5,334,322,94,222,8,183,
316,0,0,29,178,220,178,153,242,338,346,0,0,236,349,102,264,97,264,264,145,0,
0,39,295,95,244,66,119,229,260,0,0,256,99,57,160,112,42,331,334,0,0,263,288,
37,162,288,0,38,310,0,0,294,314,26,211,197,92,68,191,0,0,173,208,56,168,70,
292,349,323,0,0,77,215,43,18,47,258,136,307,0,0,159,155,257,224,167,128,140,
262,0,0,324,333,347,202,3,168,217,349,0,0,40,187,243,214,139,56,25,37,0,0,39,
98,17,316,75,220,21,179,0,0,231,10,257,92,273,178,123,80,0,0,27,82,11,213,321,
356,316,217,0,0,
; ::
; $constantid\_32\_45 : type\_id\_32\_45 :=$
0,1,2,3,4,8,9,10,13,15,16,32,44,0,1,2,3,4,6,12,14,21,23,27,33,32,0,1,2,3,4,
10,11,17,18,25,31,34,33,0,1,2,3,4,8,12,13,19,23,30,35,34,0,1,2,3,11,15,16,
17,19,20,21,36,35,0,1,2,3,4,6,9,22,24,25,28,37,36,0,2,3,4,5,7,12,18,20,24,

26,38,37,0,1,4,5,6,9,14,21,23,26,28,39,38,0,1,2,3,4,7,10,19,26,27,29,40,39,

0,1,2,3,4,7,15,20,22,30,31,41,40,0,1,2,3,4,5,17,24,25,28,29,42,41,0,1,2,3,4,

5,11,13,16,18,27,43,42,1,2,3,4,5,8,14,22,29,30,31,44,43,

;$constantshift\_32\_45 : type\_shift\_32\_45 :=$

36,338,350,161,90,242,211,250,298,106,118,0,359,226,53,137,2,67,277,261,27,

317,199,137,0,0,263,336,40,146,171,314,210,221,90,93,243,0,0,346,186,20,35,

49,179,120,10,265,223,66,0,0,237,130,296,162,196,9,322,284,341,311,228,0,0,

310,165,155,264,10,313,21,291,268,240,204,0,0,87,185,234,334,166,38,129,358,

233,120,291,0,0,183,247,33,282,341,151,130,138,53,292,245,0,0,154,148,268,

246,25,132,128,326,175,180,265,0,0,147,351,84,17,230,248,307,304,191,337,106,

0,0,313,10,145,80,68,194,311,200,302,128,318,0,0,335,142,84,126,48,150,86,

184,179,30,185,0,0,161,148,216,105,144,132,330,231,176,190,331,0,0;

APÊNDICE B – Arquitetura geral da implementação em descrição de *hardware*

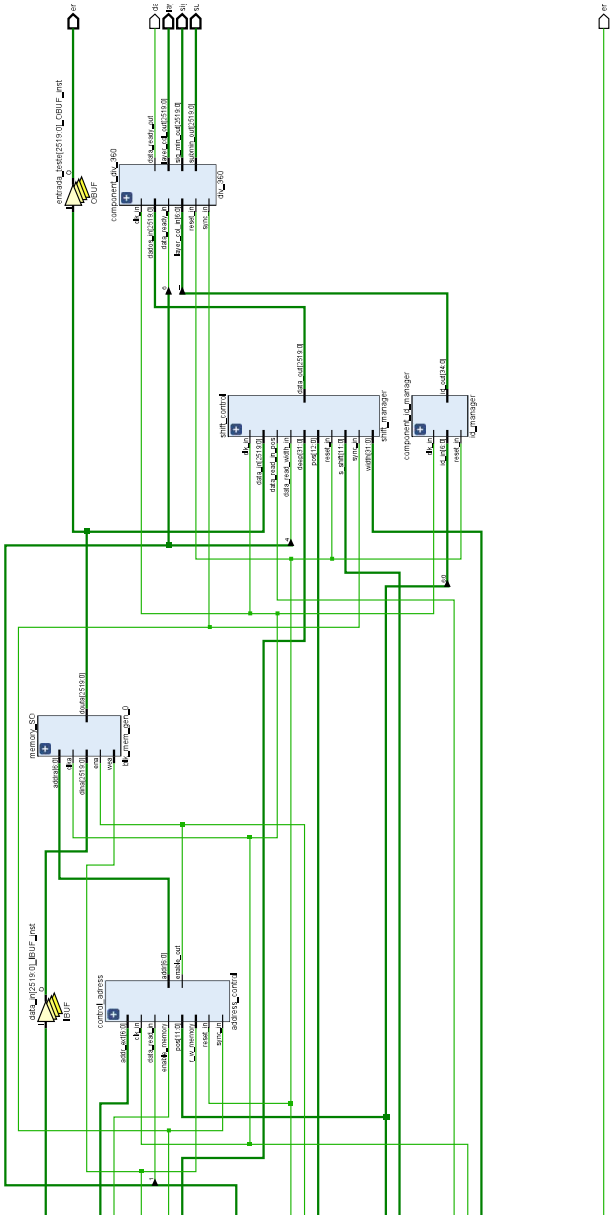


Figura 44 – *Schematic* da arquitetura geral da primeira varredura do LDPC - Parte A.

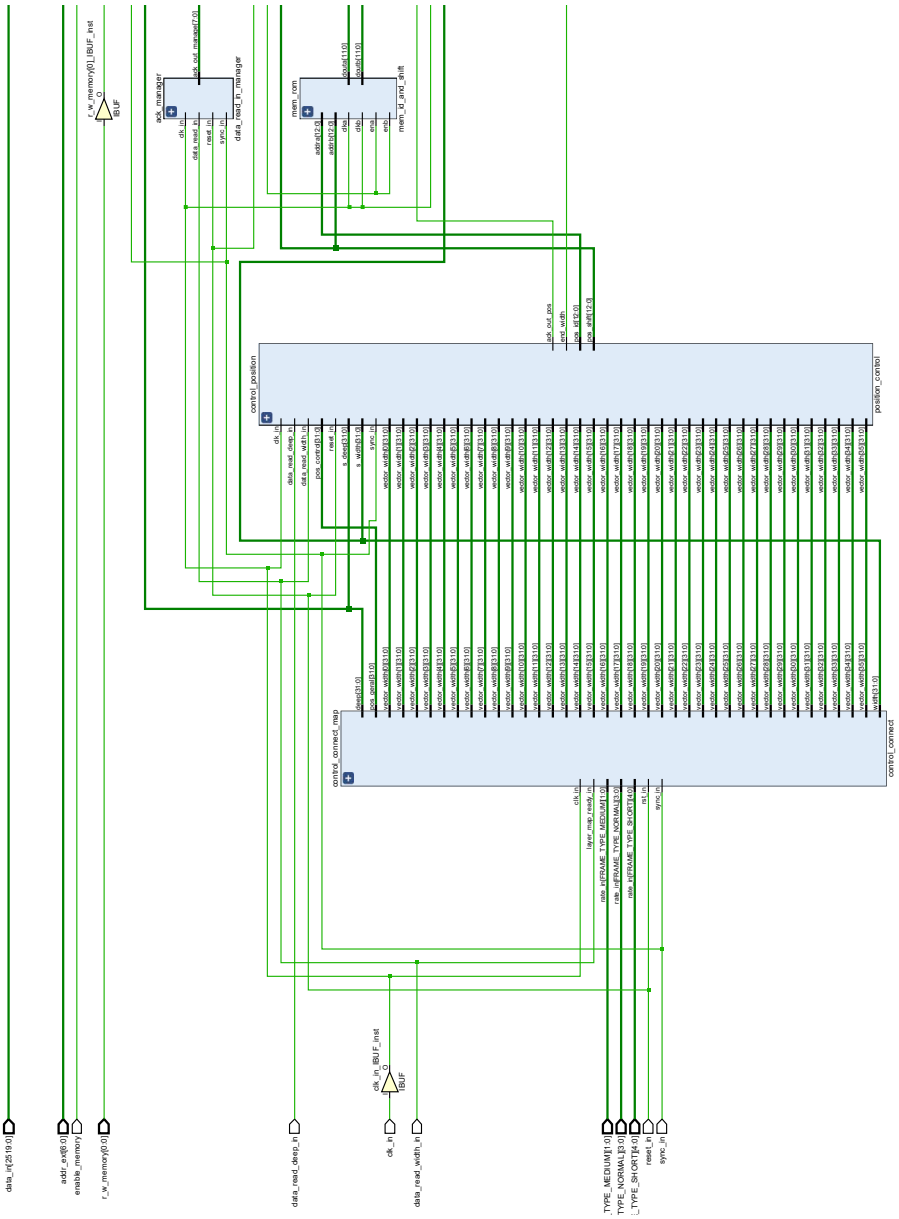


Figura 45 – *Schematic* da arquitetura geral da primeira varredura do LDPC - Parte B.