Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

IMPLEMENTAÇÃO DE UMA CLASSE DE CÓDIGOS PRODUTO COM DECODIFICAÇÃO TURBO EM FPGA

IVAN SIMÕES GASPAR

JULHO / 2006

Implementação de uma classe de códigos produto com decodificação turbo em FPGA

IVAN SIMÕES GASPAR

Dissertação submetida ao **Departamento de Telecomunicações** do **Instituto Nacional de Telecomunicações – Inatel**, como requisito parcial para a obtenção do título de **Mestre em Engenharia Elétrica**.

Orientador: Prof. Dr. Dayan Adionel Guimarães.

Santa Rita do Sapucaí

2006

Ivan Simões Gaspar

Implementação de uma classe de códigos produto com decodificação turbo em FPGA

Banca Examinadora:

_____

Orientador:
Prof. Dr. Dayan Adionel Guimarães
DTE – INATEL, MG.

_____

Dr. Durval Zandonadi Júnior
LIT – INPE, SP.

_____

Prof. Dr. Geraldo Gil Ramundo Gomes
DTE – INATEL, MG.

Santa Rita do Sapucaí, 31 de julho de 2006.

Este trabalho é dedicado à minha família, em especial ao meu pai, que me inspirou a ser um engenheiro e a gostar do que faço.

"***We shall not cease from exploration***

And the end of all our exploring

Will be to arrive where we started

And know the place for the first time."

Thomas Stearns Eliot (1888 – 1965)

Four Quartets IV, 239-245

Agradecimentos

Agradeço a Deus, minha consciência do caminho. Aos meus pais e irmãos, primos, avó e tias, grandes companheiros, exemplos no objetivo de vida, tão presentes e merecedores desta conquista. A minha esposa Ana, pelo amor, incentivo e pela compreensão do tempo que lhe fiquei ausente. Ao Prof. Dayan, pela perseverante orientação e amizade. Ao Prof. Adonias pela oportunidade concedida ainda na graduação de cursar minhas primeiras disciplinas do mestrado. Ao apoio financeiro proporcionado pelo convênio nº 22.02.0431.00, celebrado entre o Inatel, a Linear Equipamentos Eletrônicos S/A e a FINEP (Financiadora de Estudos e Projetos) e em especial ao Sr. José de Souza Lima pelo atendimento em todos os momentos em que lhe solicitei auxílio.

Índice

Capítulo III

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ADC	*Analog to Digital Converter*
ADSL	*Asymmetric Digital Subscriber Line*
AGC	*Automatic Gain Control*
ARQ	*Automatic Repeat Request*
AWGN	*Additive White Gaussian Noise*
BER	*Bit Error Rate*
BPSK	*Binary Phase Shift Keying*. Modulação binária por deslocamento de fase de 180º de um símbolo em relação ao outro.
BTC	*Block Turbo Codes*
CCSDS	*Consultative Committee for Space Data Systems*
CD	*Compact Disk*
codec	Codificador e decodificador
CRSC	*Circular Recursive Systematic Convolutional*
CTC	*Convolutional Turbo Codes*
dB	Decibel ou decibéis
dibit	associação de um par de bits
DVB-RCS	*Digital Vídeo Broadcasting – Return Channel over Satellite*
E_b	Energia de bit

fdp	função densidade probabilidade
FEC	*Forward Error Correction*
FER	*Frame Error Rate*
FPGA	*Field Programmable Gate Array*
GCC	*Generalized Code Concatenation*
GF	*Galois Field*
IP	*Intellectual Property*
I/O	*Input/Output*
ISI	*Intersymbol Interference*
JTAG	*Joint Test Action Group*. Padrão de interface elétrica e de protocolo de comunicação
LAB	*Logic Array Blocks*
LAN	*Local Area Network*
LDPC	*Low Density Parity-Check*
LE	*Logical Element*
LUT	*Look Up Table*
N_0	Densidade espectral (unilateral) de potência de ruído, medida em Watts por Hertz
MAP	*Maximum a posteriori*
MC-DS-CDMA	*Multi-Carrier Direct Sequence Code Division Multiple Access*
Mbps	Mega bits por segundo
merge	bloco do *VisSim/Comm* onde a saída recebe a entrada *t* ou *f* se o sinal aplicado em *b* valer respectivamente 1 ou 0
ML	*Maximum Likelihood* (o mesmo que MV – Máxima Verossimilhança)
MV	Máxima Verossimilhança (o mesmo que ML – *Maximum Likelihood*)
PN	*Pseudo Noise*

PRBS	*Pseudo Random Binary Sequence*
QPSK	*Quadrature Phase Shift Keying.* Modulação na qual há um deslocamento de 90º entre cada um dos quatro símbolos possíveis.
RAM	*Random Access Memory*
RTL	*Register Transfer Level*
SIHO	*Soft-Input Hard-Output*
SISO	*Soft-Input Soft-Output*
SNR	*Signal to Noise Ratio*
S&H	*Sample and Hold*
UMTS	*Universal Mobile Telecommunications System*
VHDL	*Very high speed integrated circuit Hardware Description Language*
XOR	Operação lógica “ou-exclusiva”
3G	Terceira Geração de tecnologias para redes de telefonia sem fio.

Resumo

Este trabalho, patrocinado pela empresa LINEAR Equipamentos Eletrônicos S/A, descreve um projeto de implementação de um esquema de correção de erro com código turbo de bloco de baixa complexidade elaborado em um FPGA de baixo custo. A decodificação turbo é baseada na combinação dos algoritmos de Pyndiah e Wagner. Taxas úteis de dados até 60 Mbps foram atingidas utilizando 60% da lógica disponível no FPGA EP1C6T144C8 da fabricado pela Altera. Como resultado complementar, o uso de conhecidas ferramentas de simulação didáticas são exploradas, permitindo a compreensão de uma classe específica de códigos produto e seu adequado processo de implementação. O *codec* foi criado com grande flexibilidade usando modelos estruturais e comportamentais, traduzidos posteriormente com simplicidade para linguagem VHDL.

Abstract

This work describes a project sponsored by LINEAR Equipamentos Eletrônicos S/A and carried out at INATEL. The project aimed to implement a turbo forward error correction scheme with a low complexity block turbo codec assembled in a popular low cost FPGA. The turbo decoding is based on a combination of Pyndiah's and Wagner's algorithms. Data rates of up to 60 Mbps were achieved using 60% of the resources in the FPGA EP1C6T144C8 produced by Altera. As a complementary result, the use of well-known and didactic simulation tools is explored, allowing the understanding of the specific class of product code and the proper FPGA implementation process. The codec was created with great flexibility using structural and behavioral models, easily translated later into the VHDL language.

Capítulo I

Introdução

NESTE capítulo é apresentada uma breve abordagem sobre a origem e o princípio de operação dos códigos turbo, contextualizando-os na necessidade cada vez maior de se ter sistemas de comunicação a altas taxas de dados e com baixa taxa de erros. Aspectos relacionados à caracterização do desempenho da codificação de canal, como convergência, complexidade, latência, versatilidade, soluções de implementação e suas implicações nos sistemas atuais são também abordados. Por fim descreve-se as características básicas do código turbo implementado e são listadas as principais contribuições e a estrutura desta dissertação de mestrado.

I.1 Histórico e princípio de operação dos códigos turbo

A teoria dos códigos corretores de erro tem seu marco histórico com a publicação contemporânea de dois importantes estudos ([Sha48] e [Ham50]), pioneiros no campo da teoria da informação e inusitadamente complementares. O primeiro mostrou que,

adicionando redundância controlada à informação, é possível reduzir a quantidade de erros na recepção, produzidos por ruído, a um patamar tão pequeno quanto desejado, desde que a taxa de transmissão esteja abaixo da capacidade do canal, essa determinada por um limite hoje conhecido por limite de Shannon [Gal68], [Pro95], [Wic95]. O segundo estudo descrevia justamente um esquema FEC (*Forward Error Correction*) através do qual essa redundância poderia ser adicionada à informação de modo a permitir a detecção e correção de erros, caracterizando um dos primeiros sistemas práticos de código de bloco, conhecido como código de Hamming [Pro95].

A redundância controlada refere-se à restrição das possíveis seqüências de bits de informação esperadas na recepção. Assim, quando uma seqüência é detectada com um padrão imprevisto, a função do decodificador de canal é procurar dentre as seqüências esperadas aquela que mais se assemelha à recebida. Essa semelhança pode ser estimada através da correta utilização de critérios que minimizem o erro de decisão sobre os bits transmitidos, como o de máximo a posteriori (MAP, *Maximum a posteriori*) e o de máxima verossimilhança (ML, *Maximum Likelihood*) [Lee94].

Os códigos turbo foram apresentados à comunidade científica por C. Berrou e A. Glavieux no início da década de 90 [Ber93]. Sua estrutura envolveu a concatenação paralela de dois codificadores convolucionais, alimentados com arranjos diferentemente entrelaçados da mesma seqüência de informação, e um processo iterativo de decodificação baseada em um algoritmo de entrada e saída suaves SISO (*Soft-Input Soft-Output*). Foram apresentados resultados que viabilizaram a operação do sistema a valores de E_b/N_0 muito próximos do limite de Shannon.

A invenção dos códigos turbo foi o resultado de uma construção pragmática baseada na intuição dos pesquisadores Battail, Hagenauer e Hoeher, os quais, no final dos anos 80, reacenderam o interesse do processamento probabilístico em receptores

para comunicação digital [Bat89], [Hag89]. Anteriormente, outros pesquisadores, principalmente Gallager [Gal62] e Tanner [Tan81], já haviam imaginado técnicas de codificação e decodificação cujos princípios são estritamente relacionados aos códigos turbo. Já a concatenação de códigos foi proposta inicialmente por Forney [For66] como uma alternativa para se atingir altos ganhos de codificação através da combinação de dois ou mais códigos, mantendo a capacidade de correção de erro desta combinação equivalente a de um extenso código isolado, com a vantagem de se conservar o processo de decodificação em um nível de complexidade moderadamente contornável.

Até a introdução dos códigos turbo, o mais difundido esquema de concatenação de códigos utilizava um código Reed-Solomon seguido de um código convolucional, o primeiro sendo chamado de "código externo", por ser aplicado primeiro e removido por último, e o segundo sendo chamado de "código interno". Tal configuração permitiu por muitos anos que desempenhos distantes 4 dB do limite teórico pudessem ser obtidos em aplicações reais, como os sistemas utilizados por sondas espaciais [Skl88]. Claramente esse esquema é sub-ótimo, pois embora a combinação utilizada resulte em uma palavra-código com grande distância mínima, o que garante elevado ganho assintótico, o processo de decodificação interna é isolado e não explora a redundância do código externo. Como conseqüência, a convergência fica distante do limite teórico [Ber03]. De fato, a dualidade formada pela capacidade de convergência *versus* distância mínima é um dos principais enclaves no dimensionamento de um bom *codec* (codificador-decodificador). A desejável melhora de uma destas características em certo aspecto leva geralmente ao enfraquecimento da outra [Ber03].

Os códigos turbo podem ser vistos como um aprimoramento da técnica de concatenação pois, diferente da forma de decodificação totalmente isolada dos códigos componentes, emprega iterativamente resultados parciais e complementares dos

estágios de decodificação SISO no refinamento da confiabilidade da decisão das etapas seguintes. A informação de credibilidade ou qualidade da decodificação (conhecida como informação extrínseca) de um dos códigos componentes (*soft-output*) são passadas para o processo de decodificação do outro código na forma de entradas suaves (*soft-input*), de modo que a cada iteração tenha-se maior precisão na estimativa do bit, palavra ou seqüência transmitida. Um esquema de decodificação iterativa em tempo real, adaptado para a tradicional codificação concatenada usando Reed-Solomon e código convolucional, foi proposta por Barbulesco [Bar96], prevendo mínimas modificações em *hardwares* já existentes. Tal abordagem resultaria, com uma única iteração, em um desempenho adicional de 0,8 dB no ganho de codificação com relação ao esquema original.

A decodificação iterativa constituiu uma descoberta real no que diz respeito ao problema da convergência. Com emprego desta técnica de correção de erro pôde-se então obter, sob a taxa de erro de bit de 10^{-5}, uma E_b/N_0 distante apenas 0,7 dB da capacidade de Shannon em canal AWGN (*Additive White Gaussian Noise*) [Ber96]. Este resultado foi superado novamente por Nickl, Hagenauer e Burket [Nic97a], apresentando um desempenho igual a uma E_b/N_0 distante 0,27 dB da capacidade do canal AWGN, e também por Mackay e Neal [Mac96], com a proposta de um potente código LDPC (*Low Density Parity-Check*) [Gal63].

Mais recentemente os trabalhos de Chung [Chu01] e Boutros [Bou02] mostram desempenhos surpreendentes obtendo resultados distantes apenas alguns centésimos de decibel do limite de Shannon. Acertadamente demonstram que o potencial de codificação de canal empregando códigos turbo ou códigos baseados no princípio turbo podem conduzir a eficiência dos sistemas de comunicação para resultados muito próximos a essa barreira intransponível e de tão difícil aproximação.

Apesar dos significativos resultados já publicados, a existência de palavras-código de baixo peso (pequena distância mínima), que resulta em uma severa mudança de inclinação na curva de BER (*Bit Error Rate*), fenômeno conhecido como "joelho" (Figura I.1), sugere a aplicação eficiente dos códigos turbo para alvos de BER e FER (*Frame Error Rate*) entre média ($10^{-2} > BER > 10^{-6}$ ou $1 > FER > 10^{-4}$) e baixa taxa ($10^{-6} > BER > 10^{-11}$ ou $10^{-4} > FER > 10^{-9}$). Em taxas de erro de bit muito baixas ($BER < 10^{-11}$ ou $FER < 10^{-9}$) uma adequada utilização de extensos blocos de entrelaçamento temporal entre os códigos utilizados pode conduzir a resultados significativos, como mostrados por Barbulesco e Pietrobon [Bar98]. Porém um dos grandes obstáculos a ser ainda superado se refere à adequação dos algoritmos de decodificação visando a implementação do mesmo em aplicações reais de altas taxas e/ou baixa latência.

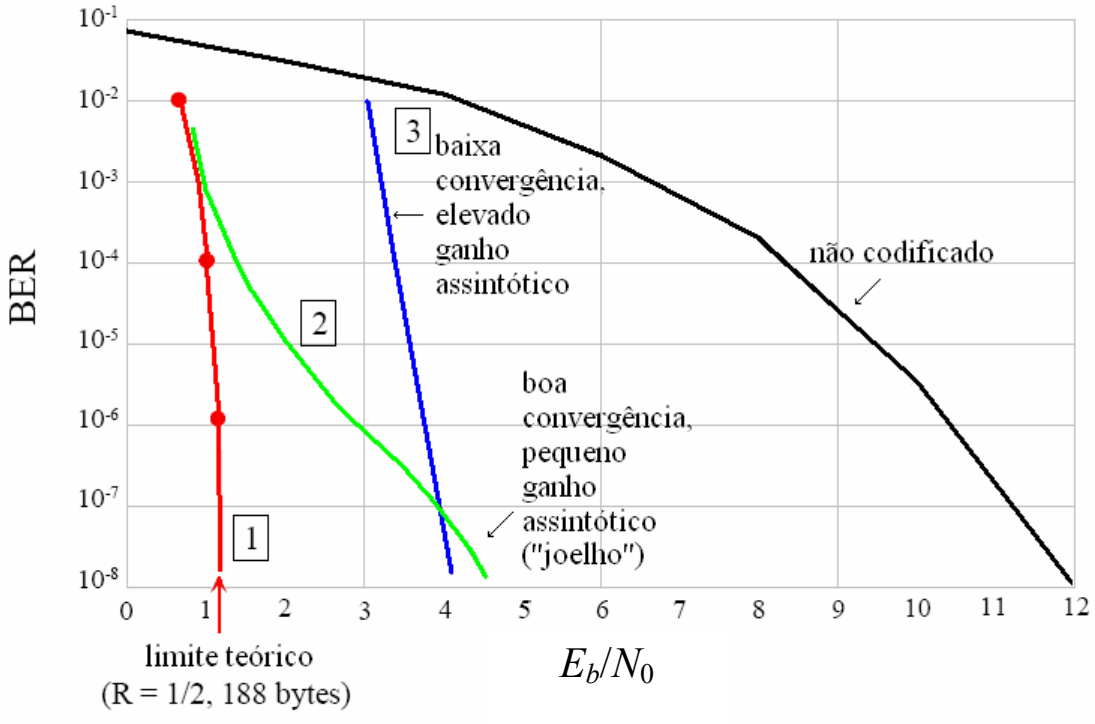


Figura I.1 *Adaptado de Berrou [Ber03]. Três possíveis comportamentos para um esquema FEC em um canal AWGN com modulação BPSK ou QPSK, taxa de código 1/2 e blocos de 188 bytes. (1) Limite teórico. (2) Código turbo. (3) Concatenação clássica entre um código Reed-Solomon e um código convolucional.*

Em relação à taxa de informação, duas soluções de *hardware* podem ser contempladas. Para baixas taxas, um único processador operando em alta freqüência pode fazer todas as iterações necessárias com uma tolerável adição de atraso. Para altas taxas, uma série de módulos idênticos pode ser implementada monoliticamente para habilitar um processamento em *pipe-line* [Ber03].

I.2 Códigos turbo convolucionais e de bloco

Basicamente têm-se duas famílias de códigos turbo, uma baseada na concatenação de códigos convolucionais (CTC, *Convolutional Turbo Code*), introduzida por Berrou [Ber93], e outra baseada na concatenação de códigos de bloco (BTC, *Block Turbo Code*), proposta por Pyndiah [Pyn94]. Grande atenção passou a ser dirigida à implementação de códigos turbo com códigos convolucionais após sua publicação introdutória e, mais recentemente, grande interesse tem sido demonstrado por implementações de processos de decodificação iterativa baseados em códigos de bloco. Dentre várias iniciativas podem-se mencionar as referências [Hag96], [Nic97a], [Nic97b], [Pyn98], [Hun98a], [Hun98b], [Dav01], [Ran01] e [Gui03]. Os principais objetivos dessas implementações se referem à possibilidade de redução na complexidade e aumento na velocidade de decodificação e também à possibilidade de aumento de desempenho em relação aos códigos turbo convolucionais (CTCs), principalmente para códigos de taxas altas [Pyn98], [Hag96], [Dav01].

A concatenação dos códigos componentes dos BTCs e CTCs pode ser obtida na forma serial ou paralela. Para os BTCs, a concatenação serial pode ser mais vantajosa que a concatenação paralela, conforme pode ser verificado em [Dav01]. Quando há

concatenação serial de dois ou mais códigos de bloco separados por entrelaçadores temporais, o processo de decodificação de BTCs pode-se valer de algoritmos derivados de algoritmos iterativos de decodificação de códigos produto, desde que estes algoritmos forneçam decisões suaves de saída.

I.3 Desempenho, complexidade, latência e versatilidade dos códigos turbo

O desempenho do *codec* é avaliado em função do tipo de perturbação (ruído gaussiano, impulsivo, desvanecimento etc.), da taxa do código e do comprimento dos blocos. Em geral, o resultado da comparação que pode ser feita entre diferentes códigos em um canal AWGN é hierarquicamente respeitado para outros tipos de degradação [Ber03]. Para taxas de erro baixas, os CTCs geralmente apresentam melhor desempenho, mas há uma aparente vantagem dos códigos turbo de bloco (BTCs) no que se refere à possibilidade, para esses códigos, de redução do "joelho" percebido nas curvas de taxa de erro *versus* relação sinal-ruído média por bit (E_b/N_0).

Como regra geral, decodificadores considerados ótimos, segundo o critério MAP símbolo a símbolo, são também exponencialmente complexos. A decodificação turbo contorna a complexidade exponencial da decodificação de máxima verossimilhança, isto é, se o objetivo é pela aproximação de um desempenho ideal, a combinação de decodificações parciais e iterativas pode oferecer uma considerável simplificação. A complexidade adicional, gerada pelo processo iterativo que a decodificação turbo demanda, em relação a, por exemplo, um único decodificador de Viterbi, abundantemente usado com 16 a 64 estados nas últimas décadas, é praticamente compatível com os novos recursos oferecidos pelos progressos na área de

microeletrônica, como mostram os recentes produtos oferecidos no mercado [Aha06][Tur06].

Em aplicações de tempo real com baixa taxa de dados a latência é geralmente uma restrição à elaboração de bons códigos turbo, pois necessitam de grande quantidade de cálculos repetitivos no lado da recepção e seu desempenho é tanto melhor quanto maior o tamanho dos blocos de dados [Ben96]. Um estudo de Koorapaty [Koo97] aponta a existência de um limiar de desempenho para códigos com comprimento em torno de 200 bits. Se o tamanho do bloco é menor do que este limiar então um código convolucional irá superar a atuação de um código turbo de comparável complexidade. Caso contrário, sendo o tamanho do bloco maior do que o limiar, o código turbo terá desempenho melhor em termos de BER para uma dada E_b/N_0. Esta é a razão pela qual em algumas aplicações, como a transmissão de voz no sistema 3G, um código convolucional singular é preferido ao invés de um código turbo.

No momento várias soluções empregando códigos turbo são oferecidas no mercado, como os circuitos integrados da empresa AHA [Aha06] ou as propriedades intelectuais (*IP cores*) da *Turbo Concept* [Tur06] para FPGAs de diversos fabricantes. Estes circuitos operam a elevadas freqüências e exploram as possibilidades de paralelismo para se adaptarem às restrições de latência de certos grupos de aplicação em tempo real. Entretanto, assim como em todos os códigos orientados em blocos, o decodificador tem que esperar pela completa seqüência codificada para iniciar o processo, e esta é uma limitação inevitável.

Outro atributo cada vez mais solicitado para aplicações reais é a versatilidade da codificação de canal, isto é, a possibilidade de que um mesmo princípio de código e um mesmo decodificador possam satisfazer várias necessidades em termos de taxa de código e tamanho de bloco. Essa versatilidade é benéfica, por exemplo, em aplicações

com satélites, onde o recurso do enlace utilizado não é bilateralmente uniforme, e onde códigos adaptáveis podem compensar a desigualdade de taxa, ou para o caso de uma rede de pacotes de tamanho variável na qual blocos codificados podem conter diferentes quantidades de informação.

Os princípios de formação dos códigos convolucionais são bastante versáteis no que diz respeito à taxa de redundância, e essa característica pode ser encontrada também nos códigos produto e LDPC. Por meio de técnicas de puncionamento, ou por concatenação de códigos, a quantidade de redundância pode ser ajustada com precisão.

I.4 Aplicações dos códigos turbo

Os acessos aos serviços de telecomunicações têm demonstrado um crescimento extraordinário nos últimos anos. Um dos grandes motivadores desse crescimento é notadamente a Internet. Antes dos anos 90 predominava a necessidade pelo chamado acesso básico (serviços de voz e, em alguns casos, fax). A partir dos anos 90, a explosão na popularização da Internet mudou rapidamente esse paradigma. Os usuários ambicionam agora não somente os serviços básicos, mas também e-mail, áudio em tempo real, imagens e multimídia, acessíveis com qualidade, a qualquer momento e de qualquer parte do mundo.

Nestas aplicações, tipicamente atendidas com sistemas ARQ (*Automatic Repeat Request*), têm-se condições propícias de taxa de erro ($10^{-2} < BER < 10^{-6}$) para o emprego dos códigos turbo. E de fato os padrões para 3ª geração de telefonia móvel, UMTS na Europa e Japão e CDMA-2000 nos EUA já adotaram os códigos turbo em sua estrutura e, devido ao seu expressivo desempenho, também estão sendo vistos como

fortes candidatos na padronização das futuras gerações de redes sem fio.

Prevê-se que a tecnologia sem fio venha a superar a cobertura de acesso fixo convencional por volta do ano 2010 [Sin00]. Nesta projeção o sistema sem fio alcançaria a penetração de mercado atingida em cerca de 100 anos pelos serviços de acesso fixo convencional [Itu00]. Atualmente, sistemas de múltiplo acesso com multiportadoras são os candidatos mais cotados às futuras gerações de comunicação sem fio, oferecendo robustez contra desvanecimentos seletivos com a possível combinação de diversidade temporal e em freqüência, contornando o uso de complexa equalização no controle de ISI (*Intersymbol Interference*), além de flexibilidade na composição do espectro final (similar ao espectro de Nyquist).

Técnicas de apoio vêm sendo desenvolvidas para firmar esta tendência. Entre elas, estratégias de codificação de canal eficientes e simples representam um desafio.

Os atuais serviços já são capazes de oferecer limitadamente desde comércio eletrônico (*e-commerce*), passando pelo acesso a informações pessoais e de negócios, até entretenimento, tudo isso podendo ser realizado através de diversos meios e com mobilidade.

As tecnologias de 3ª geração se valem das vantagens dos recentes avanços da comunicação móvel, como sistemas com modulação adaptativa, que permitem que nós de rádio aperfeiçoem as taxas de transmissão com base em um laço de realimentação "instantânea" estabelecida com os terminais. Isso, aliado à codificação turbo, permite que as velocidades de *download* cheguem próximas aos limites teóricos do canal móvel sem fio.

Os recentes avanços de implementação de códigos turbo estão sendo levados em consideração nas novas gerações de transmissão via satélite (CCSDS - *Consultative Committee for Space Data Systems*, Inmarsat M4 – *Satellite Phones*, Eutelsat Skyplex -

Video over Satellite) com taxas de código desde 1/2 a 6/7. Graças às vantajosas propriedades de puncionamento e terminação, os códigos turbo baseados em CRSC (*Circular Recursive Systematic Convolutional*) representam uma forma muito flexível de codificação [Ber03], como demonstrou a recente padronização DVB-RCS (*Digital Video Broadcasting – Return Channel over Satellite*).

Da mesma forma, aplicações já consagradas como ADSL, LAN etc. provavelmente escolherão uma das possíveis variações dos códigos turbo ou LDPC para sua evolução.

I.5 Contribuições e estrutura da dissertação

Esta dissertação descreve o desenvolvimento em *hardware* de um *codec* turbo destinado ao aprimoramento do sistema MC-DS-CDMA (*Multi-Carrier Direct Sequence Code Division Multiple Access*) sugerido por Sourour e Nakagawa[Sou96], substituindo-se o código de repetição nele existente por um código produto multidimensional decodificado de forma iterativa [Gui03].

Utilizou-se como dispositivo de implementação um FPGA (*Field Programmable Gate Array*) da família Cyclone (EP1C6T144C8), fabricado pela ALTERA. Esta tecnologia de custo relativamente baixo permite que taxas úteis de decodificação até 60 Mbps, utilizando 4 iterações, possam ser atingidas na prática.

O esquema de codificação proposto é constituído de uma classe de código produto multidimensional obtida com a concatenação serial generalizada de um mesmo código componente, um exemplo típico de GCC (*Generalized Code Concatenation*) [Bos00], assegurando, entre outras particularidades, uma grande distância mínima. A

característica chave deste arranjo é que é possível construir para ele uma estrutura de decodificação de entrada suave aplicando o algoritmo de decodificação por distância mínima de Wagner [Sil54].

A complexidade de decodificação resultante em cada dimensão é semelhante à de um código produto de paridade simples [Skl97] [Ran98] e o desempenho é equiparável àquele obtido por meio de procura exaustiva. Uma forma adaptada por Guimarães [Gui03] do algoritmo de decodificação iterativa de Pyndiah [Pyn98] é aplicada ao arranjo objetivando-se reduzir ainda mais a complexidade. Algumas considerações para a quantização dos coeficientes de ponderação e a pesquisa de uma forma de construção paralelizada da estrutura repetitiva de decodificação são investigadas com auxílio das ferramentas de simulação *VisSim/Comm* e *Mathcad* para se atingir o mais elevado fluxo de informações (*throughput*) possível no FPGA. Um arranjo combinacional foi a proposta encontrada para uma decodificação paralela que oferecesse equilíbrio entre desempenho e tempo de propagação no FPGA.

Neste capítulo destacou-se a apresentação de um breve histórico sobre os códigos turbo. Características como desempenho e complexidade foram mencionadas com o propósito de colaborar para o esclarecimento da utilização cada vez mais maciça dos códigos turbo. Foram citados também alguns dos trabalhos mais abordados na literatura e o objeto do estudo [Gui03] será destacado e analisado no Capítulo II, através da descrição da classe de códigos produto e de sua decodificação turbo.

O Capítulo III corresponde à construção prática do *codec* em FPGA. Inicialmente são apresentadas as etapas da elaboração de um diagrama funcional genérico, tendo como auxílio ferramentas computacionais de simulação como *VisSim/Comm*, *Mathcad* e *Matlab*, e também aquelas especificamente associadas ao dispositivo FPGA, como o compilador *Quartus II*. Algumas observações sobre tais

ferramentas são apresentadas no intuito de mostrar como o trabalho pôde evoluir para geração de um código VHDL, utilizando-se interfaces ora voltadas para uma implementação estrutural em blocos, ora comportamental por meio de uma linguagem matemática (sistemas de equações) ou usando linguagem texto.

A partir de diagramas em bloco em mais alto nível o projeto é detalhado em uma abordagem didática e em conformidade com as diretrizes da Linear[1]. As simulações estão armazenadas em um CD, que é parte integrante da dissertação. No CD, arquivos organizados de forma estruturada são disponibilizados em conjunto com uma documentação suficiente para sua correta navegação e aproveitamento.

Por fim, no Capítulo IV apresentam-se as conclusões finais a partir dos resultados obtidos com a experiência de implementação e são listadas também algumas propostas de trabalhos futuros. Resultados preliminares desta dissertação foram também publicados no artigo de Gaspar e Guimarães [Gas06].

[1] Empresa que colaborou com o projeto disponibilizando tempo e recursos para a pesquisa e que pode eventualmente vir a utilizá-lo na concepção de um equipamento para comunicação digital com espalhamento espectral. Maiores detalhes sobre a Linear podem ser encontrados no site www.linear.com.br.

Capítulo II

Construção e decodificação turbo do código produto $(8,4,4)^2$

O ESQUEMA de codificação de canal $(8,4,4)^2$ e sua decodificação turbo são apresentados neste capítulo. A formação do bloco codificado é obtida a partir de um arranjo de concatenação serial do código componente segundo uma estrutura de códigos produto. Entre suas vantagens, destaca-se a simplicidade de codificação aliada à simplicidade e ao ganho apresentados pelo correspondente processo de decodificação turbo.

II.1 Código componente (n,n/2,4)

O código componente (n,n/2,4) é um código de bloco com construção multinível baseado na regra de partição de conjuntos (Figura II.1), com adição de uma regra de rotulamento conforme a Concatenação Generalizada de Códigos, (GCC, *Generalized Code Concatenation*) [Bos00]. Sua construção combina um código interno de repetição $\mathbf{C}_i$ = (2,2,1), que codifica o bit mais significativo do rotulamento através de um código

de repetição, com um código de paridade simples externo, que codifica os outros bits.

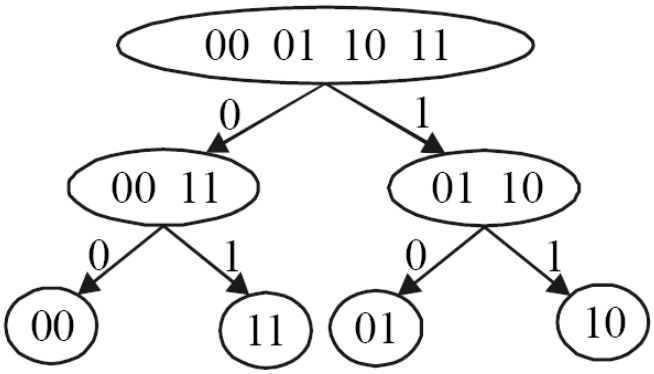


Figura II.1 *Partição do conjunto {00, 01, 10, 11} utilizado na composição do código (n,n/2,4).*

Seja $\mathbf{c}_r$ uma palavra-código do código de repetição $\mathbf{C}_r = (n/2,\ 1,\ n/2)$ e $\mathbf{c}_p$ uma palavra-código do código de paridade simples $\mathbf{C}_p = (n/2,\ n/2-1,\ 2)$. A palavra-código $\mathbf{c}$ do código não-sistemático $\mathbf{C} = (n,\ k,\ d_{\min}) = (n,\ n/2,\ 4)$ é determinada por

$$\mathbf{c} = [01]\mathbf{c}_r \oplus [11]\mathbf{c}_p \qquad \text{(II.1)}$$

onde a soma $\oplus$ é em GF(2) e o produto $[01]\mathbf{c}_r$ é calculado substituindo-se um 0 em $\mathbf{c}_r$ por 00 e um 1 por 01. O mesmo é feito para $[11]\mathbf{c}_p$, onde agora um 1 torna-se 11 e um 0 torna-se 00. O bit mais à esquerda da palavra de mensagem é utilizado como entrada para o código de repetição $\mathbf{C}_r$ e os demais são utilizados como informação para o código de paridade simples $\mathbf{C}_p$.

Por exemplo, os bits de informação [1110] resultam em $\mathbf{c}_r$ = [1111] e $\mathbf{c}_p$ = [1100] e produzem como palavra-código $\mathbf{c} = [01]\mathbf{c}_r \oplus [11]\mathbf{c}_p$ = [01 01 01 01] $\oplus$ [11 11 00 00] = [10 10 01 01].

A Figura II.2 ilustra a treliça construída para o código (8,4,4). As sub-treliças de

códigos de paridade simples, cuja estrutura básica se repete a cada seção dessas sub-treliças, são arranjadas em dois ramos paralelos, cada qual relacionado a uma das partições do código (alfabetos {00,11} e {01,10}).

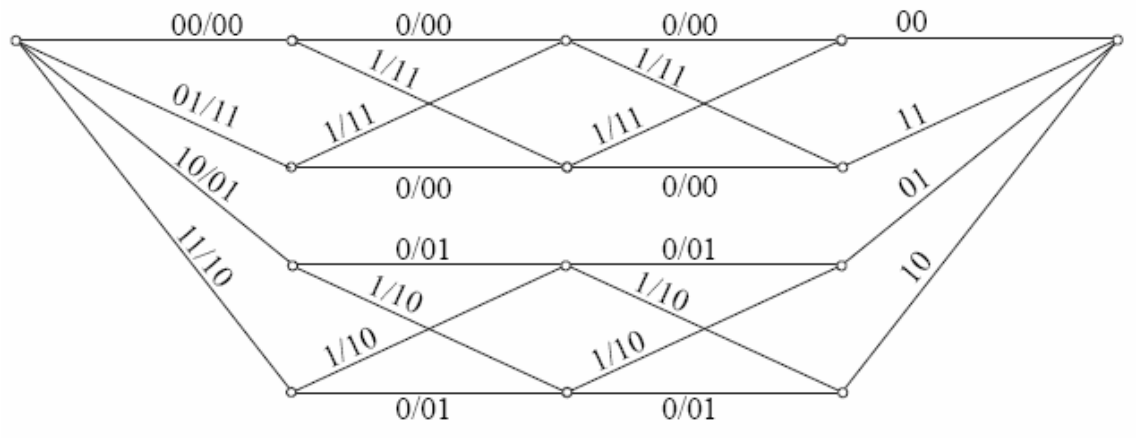


Figura II.2 *Treliça para o código (8,4,4), extraído de Guimarães [Gui03].*

Grupos de 2 bits, um do código $\mathbf{C}_r$ e outro do código $\mathbf{C}_p$, selecionam o caminho nas sub-treliças. Os rótulos *xx*/*yy* e *x*/*yy* correspondem respectivamente aos dois primeiros bits (*xx*) e os demais bits (*x*) da palavra de informação associados aos bits codificados *yy*. Esta construção possibilita que o código (*n*,*n*/2,4) seja candidato a uma decodificação de máxima verossimilhança utilizando o algoritmo de Wagner [Sil54].

Segundo o Teorema 9.2 de Bossert [Bos00], a distância mínima de Hamming desta classe de código vale $d \geq \min(d_0 d_r, d_1 d_p) = (1\times4, 2\times2) = 4$, onde d_0 é a distância mínima de Hamming entre as palavras do código interno $\mathbf{C}_i = (2,2,1)$, que é o código base da partição de conjunto ilustrada na Figura II.1, antes da primeira partição, d_1 é a distância mínima de Hamming entre as palavras-código interno $\mathbf{C}_i$ após a primeira partição, d_r é a distância mínima de Hamming entre as palavras-código do código de repetição $\mathbf{C}_r$ e d_p é a distância mínima de Hamming entre as palavras-código de paridade simples $\mathbf{C}_p$.

Segundo observado por Guimarães [Gui03], tem-se para os códigos $(n, n/2, 4)$ um decréscimo no desempenho com o aumento do comprimento das palavras-código. Os que apresentam melhores resultados têm tamanhos n entre 8 e 16. Isto ocorre porque, para uma dada relação E_b/N_0, a probabilidade de se ter mais erros em uma palavra de comprimento maior é também maior, podendo ultrapassar mais freqüentemente a capacidade de correção de erros do código. A escolha pelo código (8,4,4) para implementação foi tomada considerando que este, enquanto mantém um comportamento de desempenho médio com relação a outros possíveis comprimentos de código investigados por Guimarães [Gui03], seria o de mais simples elaboração.

II.2 Decodificação do código (*n*,*n*/2,4) usando o algoritmo de Wagner

O algoritmo de Wagner [Sil54] é apropriado à decodificação de códigos de verificação de paridade e sua utilização pode ser estendida para quaisquer casos em que a decodificação possa ser baseada em processos de verificação de paridade. No algoritmo de decodificação de Wagner para códigos de paridade simples as probabilidades *a posteriori* $p(x_1|y)$ e $p(x_2|y)$ de cada dígito y recebido são calculadas e a estimativa dos dígitos transmitidos é obtida com base no critério de máxima verossimilhança ML (*Maximum Likelihood*), isto é, escolhe-se x_1 se $p(x_1|y) > p(x_2|y)$ e x_2 caso contrário. Se $p(x_1|y) = p(x_2|y)$ escolhe-se por x_1 ou x_2 arbitrariamente com igual probabilidade. O critério ML é implementado a partir da distância euclidiana quadrática entre a amplitude do sinal recebido e a amplitude do sinal transmitido normalizada em relação à média do sinal recebido, ou seja, escolhe-se x_1 se $(y - x_1)^2 < (y - x_2)^2$ e x_2 caso contrário. Tendo-se decidido pelos n dígitos, verifica-se a paridade. Quando correta, a palavra estimada é

considerada como a mais provável. Em caso contrário é invertido o dígito mais duvidoso da palavra anteriormente estimada, formando a estimação final. O dígito mais duvidoso é aquele que apresenta a menor diferença $|(y - x_1)^2 - (y - x_2)^2|$ onde $|x|$ é o módulo de x.

Guimarães [Gui03] utiliza um processo concorrente de decodificação do código componente (n,n/2,4) empregando a regra de Wagner nas duas possíveis partições em que o código pode ser gerado, {00,11} (ramo 0) e {01,10} (ramo 1), obtendo as decisões $\hat{\mathbf{c}}$ *e* $\hat{\mathbf{c}}$' sobre palavra recebida **r**.

Cada *dibit* de **r** é interpretado como um símbolo x na entrada do processo de decodificação descrito anteriormente e a distância euclidiana (d_E) é estimada simultaneamente sobre o alfabeto $y = \{-1-1, +1+1\}$ e $y' = \{-1+1, +1-1\}$ (os símbolos "0" são esperados com amplitude "−1"e os símbolos "1" com amplitude "+1"). Os valores das métricas calculadas em cada subseção escolhida entre os ramos {00,11} e {01,10} são analisados de forma que a decisão final seja tomada pela partição com menor diferença acumulada, ou seja, aquela em que o somatório das distâncias euclidianas quadráticas calculadas para cada dígito em cada partição resulta em menor valor.

Seja, por exemplo, o vetor **r** abaixo uma palavra-código recebida na saída de um canal AWGN vetorial.

$$\mathbf{r} = (\overbrace{0.47 \quad -1.92}^{1° \textit{ dibit}} \quad 0.62 \quad 0.49 \quad -1.44 \quad -2.47 \quad 1.44 \quad 1.56)$$

O valor da distância euclidiana, d_E, de cada *dibit* da palavra-código recebida é calculado em relação ao rótulo de cada umas das seções {00}, {11}, {01} e {10} que compõe o código (8,4,4) como mostrado na matriz a seguir:

$$d_E := \begin{bmatrix} \left[(-1-r_0)^2+(-1-r_1)^2\right] & \left[(-1-r_2)^2+(-1-r_3)^2\right] & \left[(-1-r_4)^2+(-1-r_5)^2\right] & \left[(-1-r_6)^2+(-1-r_7)^2\right] \\ \left[(1-r_0)^2+(1-r_1)^2\right] & \left[(1-r_2)^2+(1-r_3)^2\right] & \left[(1-r_4)^2+(1-r_5)^2\right] & \left[(1-r_6)^2+(1-r_7)^2\right] \\ \left[(-1-r_0)^2+(1-r_1)^2\right] & \left[(-1-r_2)^2+(1-r_3)^2\right] & \left[(-1-r_4)^2+(1-r_5)^2\right] & \left[(-1-r_6)^2+(1-r_7)^2\right] \\ \left[(1-r_0)^2+(-1-r_1)^2\right] & \left[(1-r_2)^2+(-1-r_3)^2\right] & \left[(1-r_4)^2+(-1-r_5)^2\right] & \left[(1-r_6)^2+(-1-r_7)^2\right] \end{bmatrix}$$

$$d_E = \begin{pmatrix} 3.02 & 4.84 & 2.35 & 12.51 \\ 8.8 & 0.41 & 17.99 & 0.51 \\ 10.69 & 2.88 & 12.22 & 6.27 \\ 1.12 & 2.36 & 8.12 & 6.75 \end{pmatrix} \begin{matrix} \rightarrow \text{Métrica da seção } \{00\} \\ \rightarrow \text{Métrica da seção } \{11\} \\ \rightarrow \text{Métrica da seção } \{01\} \\ \rightarrow \text{Métrica da seção } \{10\} \end{matrix}$$

A escolha das subsecções em função das métricas calculadas é didaticamente mostrada na Figura II.3 com a justaposição dos dados calculados na matriz d_E sobre a treliça da Figura II.2. Os percursos traçados em cada um dos ramos da treliça representam a seleção das menores distâncias euclidianas.

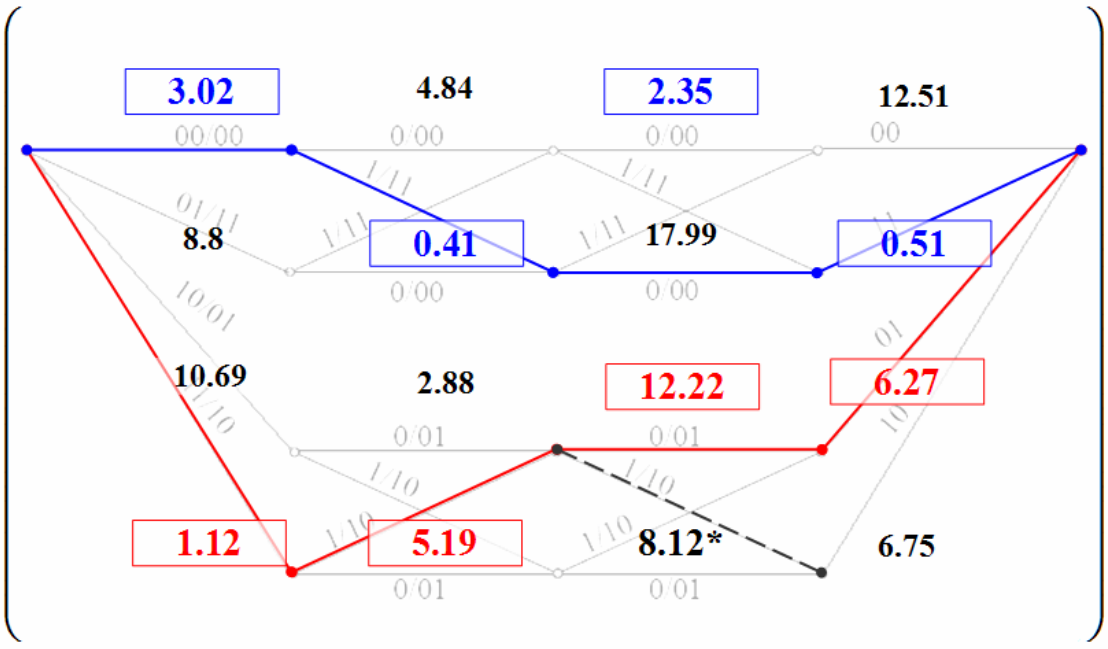


Figura II.3 *Processo concorrente de decodificação do código componente (8,4,4) empregando a regra de Wagner.*

Na Figura II.3, o caminho que aparece "tracejado" na treliça do ramo formado pelo subconjunto de *dibits* {01,10} seria a escolha a ser tomada segundo a menor métrica daquela subseção, mas foi preterido no julgamento final do processo de decodificação. Conforme a regra de Wagner, quando não há verificação da paridade, inverte-se a decisão tomada no dígito mais duvidoso, dessa forma, embora para aquela subseção a menor distância euclidiana estivesse relacionada ao par {+1−1}, a seleção final foi pelo par {−1+1}.

As palavras obtidas $\hat{\mathbf{c}}$ e $\hat{\mathbf{c}}$' e suas respectivas métricas acumuladas são mostradas abaixo. Com base no somatório das distâncias euclidianas de cada subseção a decisão final da decodificação é pela palavra $\hat{\mathbf{c}}$.

$$\hat{\mathbf{c}} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$\sum d_E = 3.02 + 0.41 + 2.35 + 0.51 = 6.28$$

$$\hat{\mathbf{c}}` = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\sum d_E = 10.69 + 2.36 + 12.22 + 6.75 = 21.97$$

A Figura II.4, ilustra a curva de desempenho de decodificação do código (8,4,4) pela regra de Wagner e por um decodificador ótimo usando o método de procura exaustiva (Máxima Verossimilhança), obtidos com uma simulação no programa *Mathcad* originalmente elaborada por Guimarães [Gui03]. Observa-se que ambas as decodificações apresentam aparentemente o mesmo desempenho em termos de taxa de erro de bit. De fato, o algoritmo de Wagner tem desempenho de Máxima Verossimilhança para códigos de paridade simples e para códigos com construção similar, como é o caso dos códigos (n,n/2,4) [Gui03].

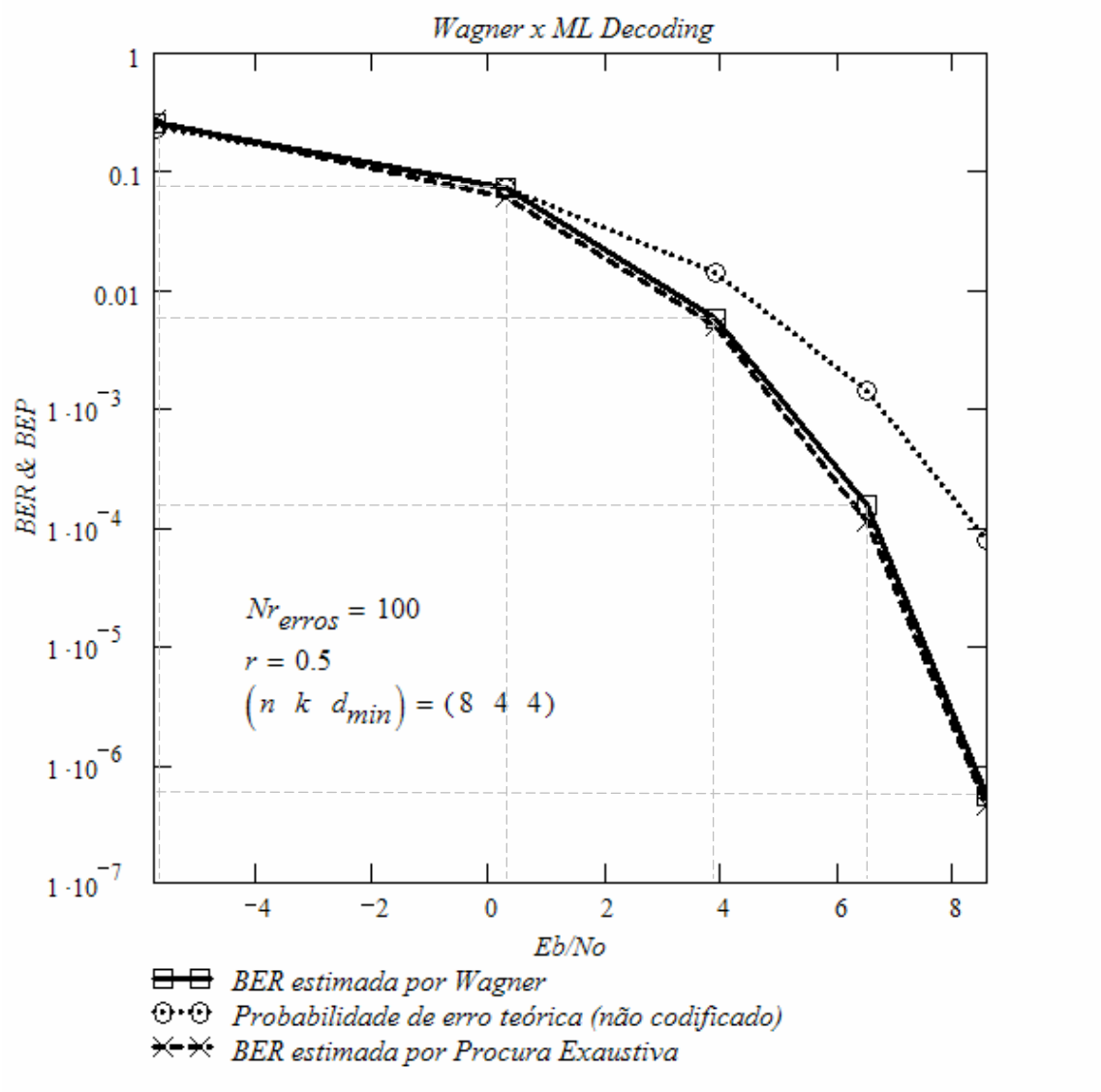


Figura II.4 *Desempenho do algoritmo de Wagner para o código (8,4,4).*

II.3 Construção do código produto bidimensional

A estrutura típica de um código produto bidimensional, ilustrada na Figura II.5, é construída com a concatenação serial de dois códigos de bloco lineares $\mathbf{C}_1$ com parâmetros (n_1,k_1,d_{min1}) e $\mathbf{C}_2$ com parâmetros (n_2,k_2, d_{min2}). Os símbolos são arranjados

em k_1 linhas e k_2 colunas. O codificador inicialmente codifica as k_1 linhas usando o codificador $\mathbf{C}_2$ e então processa as n_2 colunas usando o código $\mathbf{C}_1$.

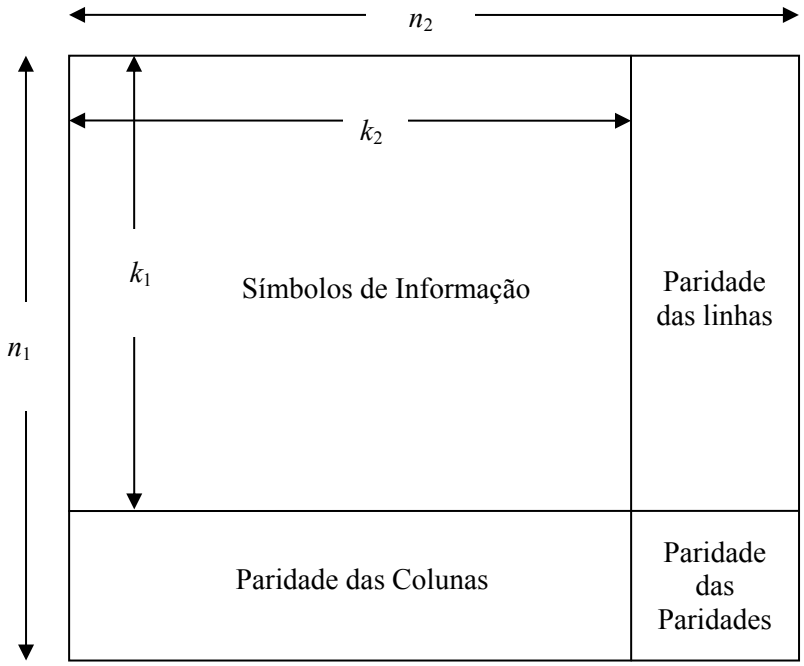


Figura II.5 *Construção do código produto $C_1 \times C_2$.*

O comprimento, a taxa e a distância mínima de Hamming para este código são respectivamente:

$$n = n_1 \times n_2 \tag{II.2}$$

$$R_c = R_{c_1} \times R_{c_2} \tag{II.3}$$

$$d_{\min} = d_{\min_1} \times d_{\min 2} \tag{II.4}$$

Guimarães [Gui03] propõe a criação de um arranjo *D*-dimensional correspondente ao código produto $(n,n/2,4)^D$. Entretanto, o código alvo de implementação nesta dissertação possui somente duas dimensões ($D = 2$) e suas etapas de formação são listadas abaixo:

1. Forme o vetor de entrada composto de k^D bits de informação;
2. Na dimensão $d = 0$, codifique k a k os bits de entrada, gerando um vetor com $N_d = n^{d+1} \times k^{D-d-1}$ bits;
3. Faça o entrelaçamento temporal nesse vetor de N_d bits, entre as dimensões 0 e 1, alimentando, pelas linhas, um arranjo do tipo linha-coluna com $(N_{linhas} \times N_{colunas})_{d,d+1} = (n^d k^{D-d-1} \times n)$ bits; a leitura dos bits entrelaçados é feita pelas colunas desse arranjo;
4. Na dimensão $d = 1$, codifique k a k os N_d bits anteriormente entrelaçados, gerando um vetor com $N_d = n^{d+1} \times k^{D-d-1}$ bits.

As características da concatenação, definidas em função do comprimento do bloco de informação em cada dimensão $\{k_1, k_2, \ldots k_i, \ldots k_D\}$, $i = 1,\ldots,D$, fazem com que o código produto $(8,4,4)^2$ resultante possua blocos de comprimento:

$$n = \prod_{i=1}^{D} n_i = n_1 \times n_2 = 8 \times 8 = 64$$

Sendo $R_{ci} = k_i/n_i$ a taxa do código componente na dimensão *i*, a taxa do código produto $(8,4,4)^2$ é

$$R_c = \prod_{i=1}^{D} R_{ci} = R_{c1} \times R_{c2} = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

e a distância mínima de Hamming para este código é:

$$d_{\min} = \prod_{i=1}^{D} d_{\min_i} = d_{\min_1} \times d_{\min_2} = 4 \times 4 = 16$$

A Figura II.6 ilustra a construção do código produto $(8,4,4)^2$, formado pela concatenação serial de dois códigos componentes (8,4,4) separados por um entrelaçador temporal.

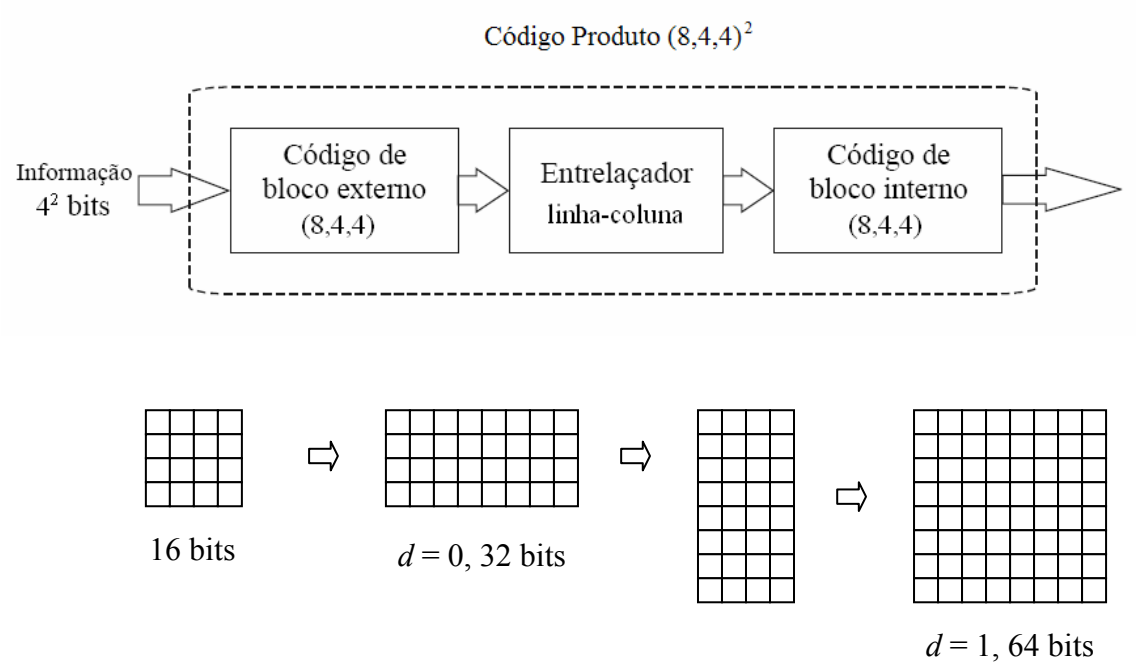


Figura II.6 *Diagrama da Concatenação serial de códigos de bloco (8,4,4).*

II.4 Decodificação iterativa

Para uma palavra-código $(n,k,d_{\min})$ de um código de bloco linear $\mathbf{C}=(c_1,c_2,\ldots,c_n)$, transmitida como $\mathbf{X}=(x_1,x_2,\ldots,x_n)$, onde $x_j=+1$ se $c_j=1$ e $x_j=-1$ se $c_j=0$, sobre um canal AWGN (*Additive White Gaussian Noise*), a decodificação de Máxima Verossimilhança (MV) do vetor recebido $\mathbf{R}=(r_1,r_2,\ldots,r_n)$, onde $r_j=x_j+n_j$, $j=1,2,\ldots,n$ e n_j é uma variável aleatória gaussiana com desvio padrão σ, é obtida com a procura exponencialmente complexa em função de k

$$\mathbf{D}=\mathbf{C}^i \text{ se } \left|\mathbf{R}-\mathbf{C}^i\right|^2 < \left|\mathbf{R}-\mathbf{C}^l\right|^2, \text{ com } l\in\left[1\ldots 2^k\right] \text{ e } l\neq i \qquad \text{(II.5)}$$

onde $\{\mathbf{C}^i\}$ é o conjunto de todas as palavras-código válidas e

$$\left|\mathbf{R}-\mathbf{C}^i\right|^2=\sum_{l=1}^{n}\left(r_l-c_l^i\right)^2 \qquad \text{(II.6)}$$

Como solução de contorno à complexidade do processo de MV, Pyndiah apresenta em [Pyn98] um algoritmo sub-ótimo para decodificação iterativa de códigos de bloco baseados em códigos produto. Esse algoritmo utiliza um decodificador SIHO (*Soft-Input Hard-Output*) fundamentado no algoritmo 2 de Chase [Cha72], seguido de cálculos para obter informações de confiabilidade (saída suave) a partir das decisões abruptas do decodificador. Cada estágio de decodificação opera separadamente sobre um dos códigos componentes do código produto, apresentando uma solução de

compromisso entre desempenho e complexidade bastante atrativa para aplicações práticas.

A idéia chave do algoritmo 2 de Chase é a redução da complexidade computacional através da limitação do processo de procura de palavras-código dentro de uma esfera de raio $(\delta - 1)$ centrada na seqüência binária $Y = (y_1, y_2, \ldots, y_n)$, onde $y_i = 0{,}5 \times (1 + \text{sign}(r_j))$, i=1,2,…,n. As informações suaves são computadas com base na confiabilidade estimada a partir de uma lista disponível de palavras-código.

Pyndiah [Pyn98] e [Pyn99] apresenta expressões apropriadas para o cálculo da confiabilidade da decisão tomada pelo algoritmo de Chase. Segundo Pyndiah, a confiabilidade da decisão d_j (saída suave) em canal AWGN pode ser determinada com a equação II.7 abaixo [Pyn98, eq. 18], onde **B** é uma palavra-código concorrente à palavra decodificada **D**.

$$r'_j = r_j + i_{ej} = \left(\frac{|\mathbf{R} - \mathbf{B}|^2 - |\mathbf{R} - \mathbf{D}|^2}{4} \right) \cdot d_j \qquad \text{(II.7)}$$

Quando o algoritmo 2 de Chase não encontra uma palavra-código concorrente, a saída suave é então calculada por intermédio do fator de ponderação β.

$$r'_j = r_j + i_{ej} = \beta \cdot d_j \qquad \text{(II.8)}$$

O algoritmo de Pyndiah usa ainda um outro parâmetro essencial para o bom desempenho do decodificador, o fator de ponderação α, aplicado à informação obtida após a decodificação em uma das dimensões do código. A composição básica do esquema de decodificação turbo proposto por Pyndiah é mostrada na Figura II.7.

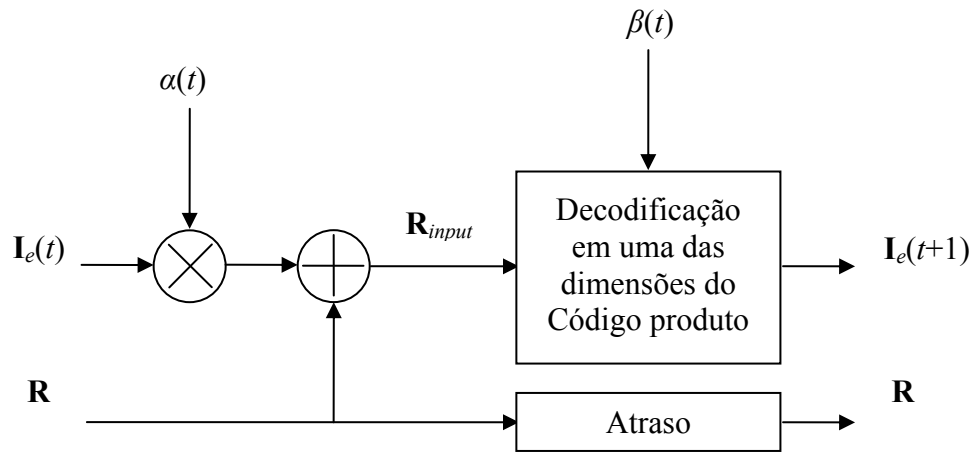


Figura II.7 *Esquema básico de decodificação iterativa proposto por Pyndiah [Pyn98].*

O arranjo **R** mostrado na Figura II.7 corresponde a um bloco de dados de um código produto, de dimensão *D*, interferido pelo canal. Um número de passos de decodificação igual ao produto do número de iterações desejado, *NI*, pela dimensão do código, *D*, é indexado na Figura II.7 como *t*, ou seja,

$$t = 0,1,\ldots,(NI \times D) - 1 = 0,1,\ldots,(NI \times 2) \qquad \text{(II.9)}$$

A cada iteração completa têm-se *D* passos de decodificação correspondentes à decodificação em todas as dimensões do código. No primeiro passo (passo 0) os valores da informação extrínseca, $\mathbf{I}_e(0)$, e do fator de ponderação dessa informação extrínseca, $\alpha(0)$, são nulos e a entrada suave é o próprio arranjo $\mathbf{R}_{input}(0) = \mathbf{R}$. O decodificador, implementado por Pyndiah [Pyn98] com o algoritmo de Chase, efetua, a partir de $\mathbf{R}_{input}(0)$, a estimação de cada uma das palavras-código no sentido correspondente à primeira dimensão. A confiabilidade da decisão de cada um dos bits estimados é então

calculada conforme a equação II.9 ou, se necessário, fazendo uso do fator de ponderação da saída abrupta, $\beta(0)$, para formar a saída suave do decodificador. Desta saída suave é subtraída a entrada suave, gerando a informação extrínseca $\mathbf{I}_e(1)$ que será utilizada no segundo passo de decodificação. No segundo passo a informação extrínseca calculada no passo anterior é ponderada pelo fator $\alpha(1)$ e somada ao arranjo **R** de tal sorte que seja formada a nova entrada suave $\mathbf{R}_{input}(1)$ para o decodificador de Chase. O decodificador então efetua a estimação de cada uma das palavras-código no sentido correspondente à segunda dimensão e gera a confiabilidade da decodificação, utilizando novamente a equação II.9 ou $\beta(1)$ agora. Uma nova informação extrínseca é então calculada e assim o processo se repete para todas as dimensões (iteração completa) e em cada iteração seguinte. Ao final da última iteração a decisão abrupta fornecida pelo decodificador é considerada como a decisão final.

Guimarães [Gui03] utiliza a idéia básica do algoritmo de Pyndiah, porém substituindo o algoritmo de Chase do decodificador SIHO pelo algoritmo de Wagner. Propõe-se ainda que a confiabilidade da decisão seja calculada apenas fazendo uso do fator de ponderação da saída abrupta, $\beta(t)$.

Tais mudanças foram investigadas por Guimarães [Gui03] tendo-se em vista a construção de um decodificador que, embora menos flexível se comparado ao algoritmo de Chase (aplicável a qualquer código de bloco linear), apresentasse, com uma considerável redução de complexidade, uma atrativa solução de codificação para um esquema MC-DS-CDMA, permitindo elevar seu desempenho em relação ao sistema não codificado sem alterar a banda ou a taxa de informação.

Os fatores de escala $\alpha(t)$ e $\beta(t)$ do decodificador turbo de bloco são crescentes em t ($t = 0 \ldots NI \times D$) e escolhidos de tal sorte que a cada passo e a cada iteração a confiabilidade da decisão seja melhorada. As leis de variação dos valores de $\alpha(t)$ e $\beta(t)$

que proporcionaram melhores resultados para os casos considerados por Guimarães [Gui03] são, respectivamente, logarítmica e linearmente crescentes em *t* (equações II.10 e II.11). Os termos $K\beta$ e $K\alpha$ foram determinados empiricamente por Guimarães [Gui03] de tal forma a compensar, em termos do desempenho final obtido, a exclusão do estado de canal[2] no cálculo da entrada suave do algoritmo de decodificação turbo.

$$\beta(t) = \frac{t+2}{NI \times D} \times K\beta \tag{II.10}$$

$$\alpha(t) = \frac{\log(t+1)}{NI \times D} \times K\alpha \tag{II.11}$$

A decodificação iterativa do código $(8,4,4)^2$ proposta por Guimarães [Gui03] utilizando a estrutura da Figura II.7 pode ser resumida na seguinte seqüência de passos:

1. Utilizando o algoritmo de Wagner, decodifique a entrada suave na dimensão $d = 0$;
2. Calcule, utilizando β, a confiabilidade de cada decisão tomada pelo algoritmo de Wagner, formando a saída suave;
3. Calcule a informação extrínseca (entrada suave subtraída da saída suave) e pondere-a pelo fator de escala α;
4. Some a informação extrínseca ponderada ao arranjo $\boldsymbol{R}$, formando uma nova entrada suave;
5. Utilizando o algoritmo de Wagner, decodifique a nova entrada suave na dimensão $d = 1$;

[2] O estado de canal, também denominado confiabilidade do canal, nada mais é do que a porção de interferência na entrada suave de um decodificador SISO estimada a partir da potência média do ruído gaussiano aditivo do canal [Gui03, eq 4.4].

6. Calcule, utilizando β, a confiabilidade de cada decisão tomada pelo algoritmo de Wagner, formando nova saída suave;
7. Calcule uma nova informação extrínseca e pondere-a pelo fator α;
8. Some a informação extrínseca ponderada ao arranjo **R**, formando nova entrada suave;
9. Repita os passos anteriores (iniciando no passo 2) tantas vezes quanto determinar o número de iterações desejado, recomeçando sempre com a decodificação da entrada suave mais recente, na dimensão $d = 0$.

Os resultados previstos para implementação do esquema de decodificação turbo do código produto $(8,4,4)^2$ estão representados pelas curvas mostradas na Figura II.8. Para este código, de apenas 64 bits de comprimento e construção bastante simples, são esperados ganhos de codificação de mais de 4 dB, medidos em relação a taxas de BER de 1×10^{-5} em canal AWGN, utilizando-se 16 iterações.

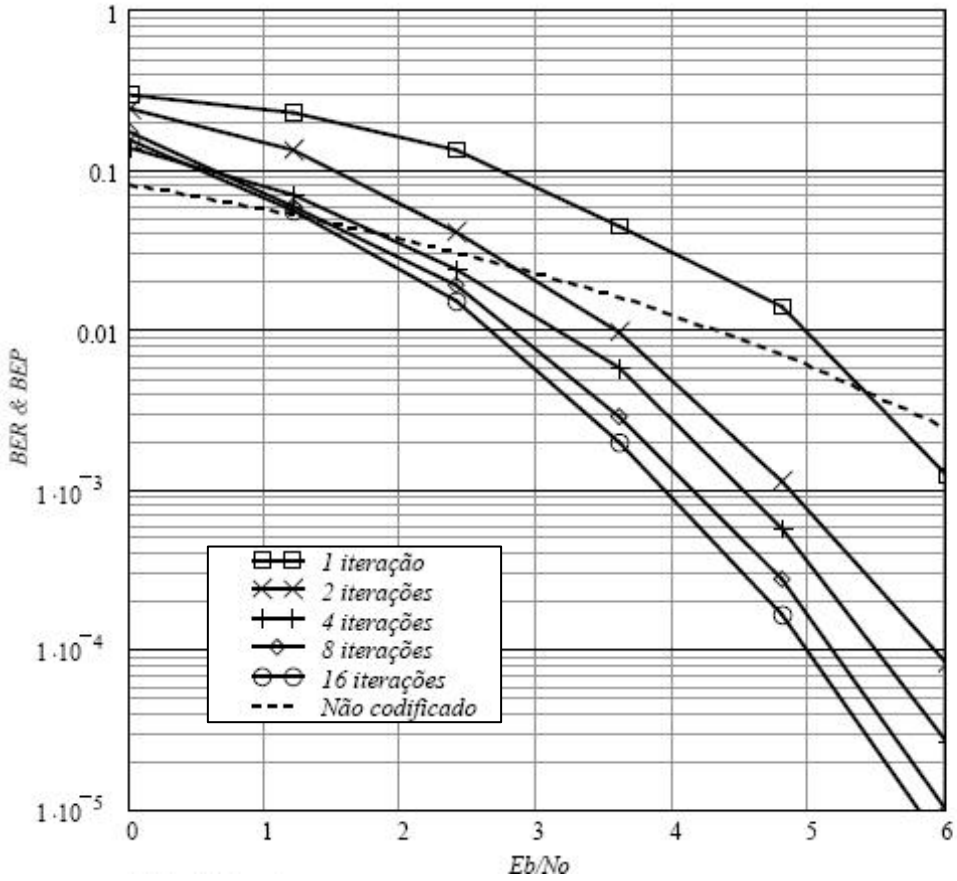


Figura II.8 *Evolução no desempenho do código produto 2D com componentes (8,4,4) em canal AWGN, em função do número de iterações.*

Capítulo III

Implementação do código produto $(8,4,4)^2$ com decodificação turbo

ESTE capítulo descreve as etapas da implementação do código produto $(8,4,4)^2$, de um canal AWGN vetorial e de um esquema de decodificação iterativa em um dispositivo FPGA (*Field Programmable Gate Array*), tendo como auxílio as ferramentas computacionais *Mathcad*, *VisSim/Comm* e *Quartus II*. Detalhes da construção de modelos alternativos às simulações originais em *Mathcad* de Guimarães [Gui03], utilizando o software *VisSim/Comm*, e o fluxo de projeto para programação do dispositivo FPGA em linguagem de descrição de *hardware*, utilizando o compilador *Quartus II*, são abordados à medida que as implementações das partes constituintes do projeto são apresentadas. Este aspecto do texto, ao mesmo tempo em que mais didático, foi considerado mais adequado pelo autor, em detrimento da adoção de anexos, por oferecer uma forma ininterrupta e um pouco mais detalhada para narração do desenvolvimento.

III.1 Descrição do dispositivo FPGA utilizado no projeto

A tecnologia de lógica digital programável (FPGA), adotada na implementação do *codec* $(8,4,4)^2$, apresenta uma atrativa combinação de desempenho e flexibilidade para construção de protótipos e verificação de operação de algoritmos em tempo real. O *hardware* do dispositivo FPGA, uma matriz de portas lógicas programáveis configuradas por *software*, atende a uma ampla faixa de aplicações que vão desde as que requerem alta flexibilidade, tradicionalmente implementadas com a arquitetura de microcontroladores, àquelas dedicadas e com alto desempenho como é o caso dos ASICs (*Application-Specific Integrated Circuit*). De fato, os FPGAs podem oferecer uma solução completa de projeto, envolvendo a utilização de microcontroladores embarcados, comercializados pelos fabricantes como IPs (*Intelectual Properties*), aliados à construção de funções dedicadas ao processamento digital de sinais. É possível ainda utilizá-los para aceleração de simulações por *hardware*, como é o caso do método denominado *hardware in the loop* explorado pela Altera [Alt05], o que pode ser muito benéfico em investigações de caráter empírico.

As arquiteturas de FPGAs disponíveis no mercado se apresentam classificadas em diversas famílias, com atributos adaptados a certos nichos de mercado, algumas voltadas para aplicações de baixo custo, outras com foco em desempenho. Por esse motivo pode-se escolher entre uma grande variedade de recursos de capacidade, velocidade e faixas de preço. O FPGA adotado na implementação, EP1C6T144C8 da Altera, comumente chamado de Cyclone C6, foi fornecido pela empresa Linear Equipamentos Eletrônicos S/A na forma de uma placa de testes genérica. A escolha do Cyclone C6 também representa uma boa solução de compromisso para aplicações de

intermediária complexidade, oferecendo razoável desempenho a um custo relativamente baixo (da ordem de algumas dezenas de dólares).

Os dispositivos da família Cyclone são compostos basicamente por uma matriz de circuitos lógicos (*Logic Array*), blocos de memória RAM (*Random Access Memory*), vias de interconexão e pinos configuráveis de I/O (*input/output*). A composição elementar dos circuitos lógicos, denominada elemento lógico (LE – *Logical Element*), pode ser configurada para operar de forma combinacional e/ou seqüencial através da união da saída de uma LUT (*Look-Up Table*) com um registrador de deslocamento (*flip-flop* tipo D), como mostrado na Figura III.1.

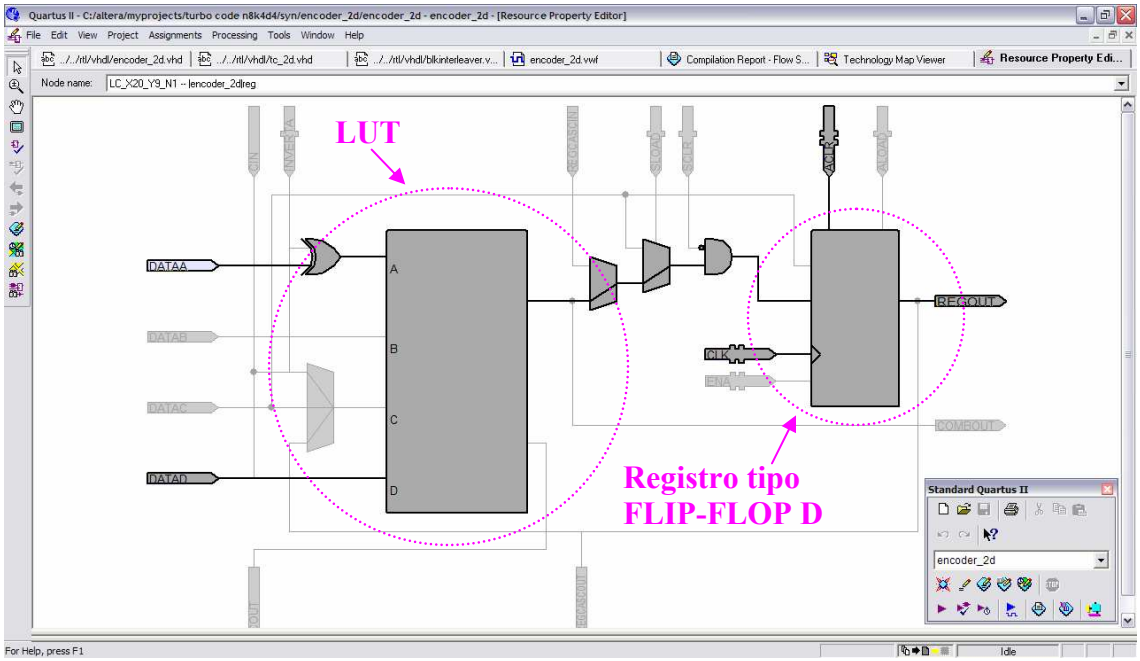


Figura III.1 *Detalhe de um elemento lógico do dispositivo Cyclone EP1C6 configurado como componente de uma implementação do codificador (8,4,4)*[2].

A arquitetura do Cyclone EP1C6T144C8 (Figura III.2) é composta por 5.980 elementos lógicos (de onde vem a designação EP1C6, o algarismo 6 indica a ordem de milhar do número de elementos lógicos), pertence a um lote de *chips* com desempenho típico (denominado traço 8 − C8, enquanto que os mais rápidos são classificados como

traço 7 e 6), 20 blocos de memória M4K RAM (memórias de 4608 bits – composta por dados mais paridade), dois PLLs (*Phase Locked Loop*) internos e encapsulamento TQFP de 144 pinos (de onde vem a identificação T144), onde 98 pinos são I/Os configuráveis.

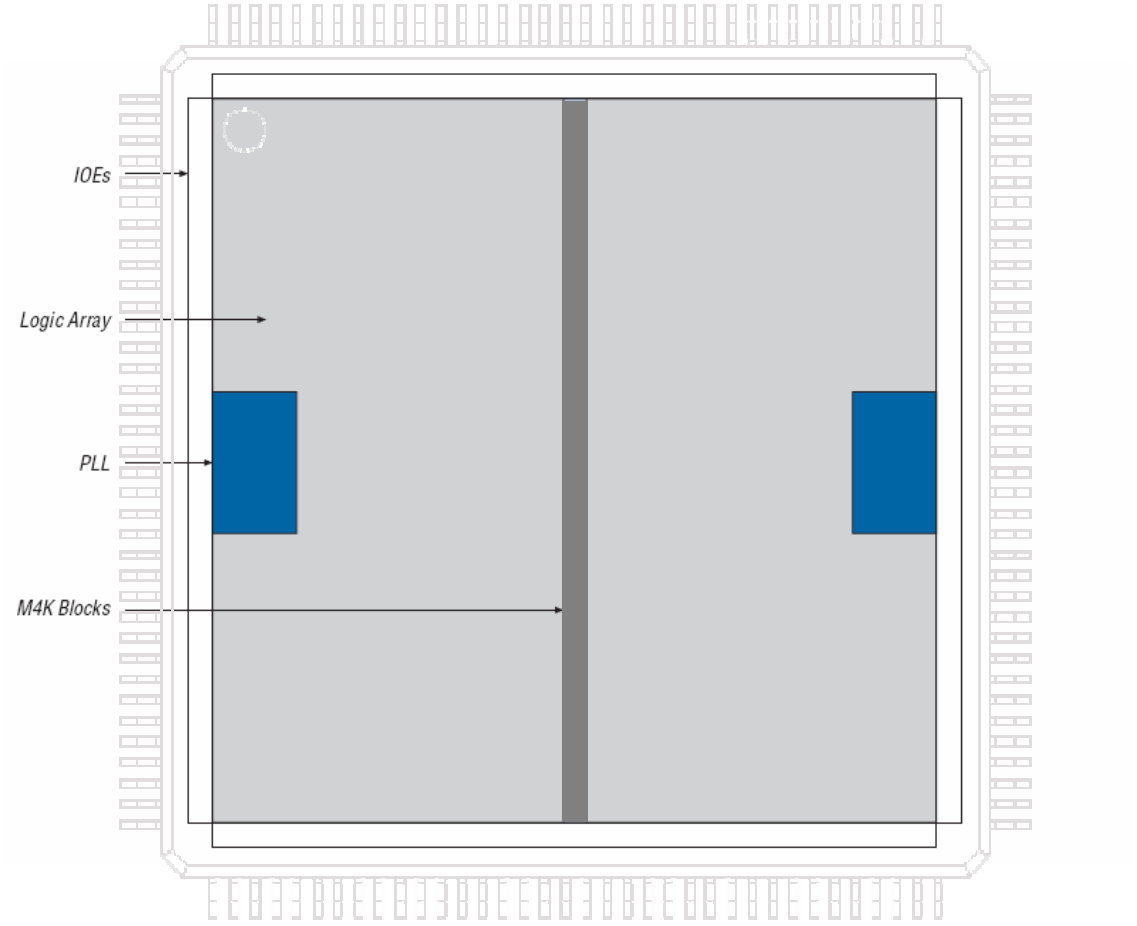


Figura III.2 *Arquitetura do dispositivo Cyclone C6 em uma pastilha com encapsulamento TQFP 144.*

Os pinos de I/O são alimentados por um elemento de controle (IOE – *Input Output Element*) que suporta vários tipos de interface como LVTTL (*Low Voltage TTL*) ou padrões diferenciais LVDS (*Low-Voltage Differential Signaling*) de alta velocidade. O dispositivo conta também com 8 vias dedicadas para *clock*, estrategicamente

distribuídas ao longo de toda a pastilha. Estas vias estão interconectadas aos dois PLLs internos, mas podem eventualmente ser utilizadas como vias globais de dados.

Os LEs são agrupados em conjuntos de 10 elementos lógicos, formando os chamados LABs (*Logic Array Blocks*), e distribuídos em um arranjo bidimensional de linhas e colunas. Cada LAB pode utilizar as diversas vias de interconexão para se interligar aos blocos de memória RAM, do tipo *dual port*, obtendo acesso de leitura e escrita independentes de até 36 bits a velocidades de 250 MHz.

A implementação do *codec* $(8,4,4)^2$ no Cyclone C6 é intermediada pelo uso do software *Quartus II*. Essa ferramenta, desenvolvida para os dispositivos da Altera, contempla uma solução de integração completa do fluxo de projeto: *design entry* [3], simulação funcional, compilação e síntese, *place & route* [4], análise de tempos de propagação (*timing*), simulação com restrições de *timing* e até captura de sinais lógicos gerados internamente no dispositivo programado através de interface padrão JTAG (*Joint Test Action Group*).

III.2 Implementação do código produto $(8,4,4)^2$

III.2.1 Construção do código componente (8,4,4)

Para simular e programar em FPGA o circuito do código componente (8,4,4), explorado originalmente no *Mathcad* por Guimarães [Gui03] (Figura III.3), investigou-se com o auxílio do software *VisSim/Comm* uma estrutura serial de codificação de comprimento *n*

[3] Definição do tipo do projeto, descrição comportamental ou estrutural.
[4] Mapeamento de recursos primitivos (flip-flops, portas lógicas etc.) em locais específicos dentro de uma tecnologia alvo, especificando as rotas e conexões necessárias com base nas restrições de área e velocidade de propagação.

qualquer capaz de entregar 2 bits codificados para cada bit de mensagem recebido (Figura III.4).

Comparada à construção elaborada no *Mathcad*, o modelo retrabalhado no *VisSim/Comm* se assemelha bem mais a um diagrama esquemático digital e isso contribui para que as propostas investigadas em caráter de simulação possuam uma construção condizente com a forma de implementação final.

A semelhança com o nível de construção esquemática motivou a adoção do *VisSim/Comm* na reelaboração de todas as etapas que compõem o projeto do *codec* $(8,4,4)^2$. Outros benefícios da utilização dessa ferramenta de projeto se referem à simplicidade e à capacidade de abstração na criação de blocos hierárquicos parametrizáveis, à transparência da depuração com o uso de avançados recursos de análise e verificação e à similaridade com que tais blocos podem ser posteriormente convertidos em VHDL[5].

Geração do código componente (1xn) pela regra de construção multinível

$$C_{1xn}(Data) := \left| \begin{array}{l} Cp \leftarrow stack\left(submatrix(Data,1,k-1,0,0), mod\left(\sum_{j=1}^{k-1} Data_j, 2\right)\right) \\ Ref \leftarrow \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \text{for } \ j \in 0..n-1 \\ \quad \left| \begin{array}{l} C_j \leftarrow Cp_{floor\left(\frac{j}{2}\right)} \ \text{if } Data_0 = 0 \\ C_j \leftarrow \left| \begin{array}{ll} Ref_{0,mod(j,2)} & \text{if } Cp_{floor\left(\frac{j}{2}\right)} = 1 \\ Ref_{1,mod(j,2)} & \text{otherwise} \end{array} \right. \end{array} \right. \\ C \end{array} \right.$$

Figura III.3 *Trecho responsável pela codificação da mensagem segundo o código componente (n,n/2,4), selecionado da simulação de Guimarães [Gui03].*

[5] A linguagem de descrição de *hardware* VHDL (*Very high speed integrated circuit Hardware Description Language*), adotado pelo IEEE (*Institute of Electrical and Electronics Engineers*) como um padrão em 1987 (IEEE 1076-1987), é capaz de caracterizar um circuito digital qualquer da mesma forma que um desenho esquemático. Foi desenvolvida sob o programa VHSIC (*Very High Speed Integrated Circuit*) do Departamento de Defesa dos Estados Unidos para viabilizar o desenvolvimento, simulação, síntese e documentação de circuitos digitais de grande complexidade, podendo suportar abstração e reuso parametrizado do projeto.

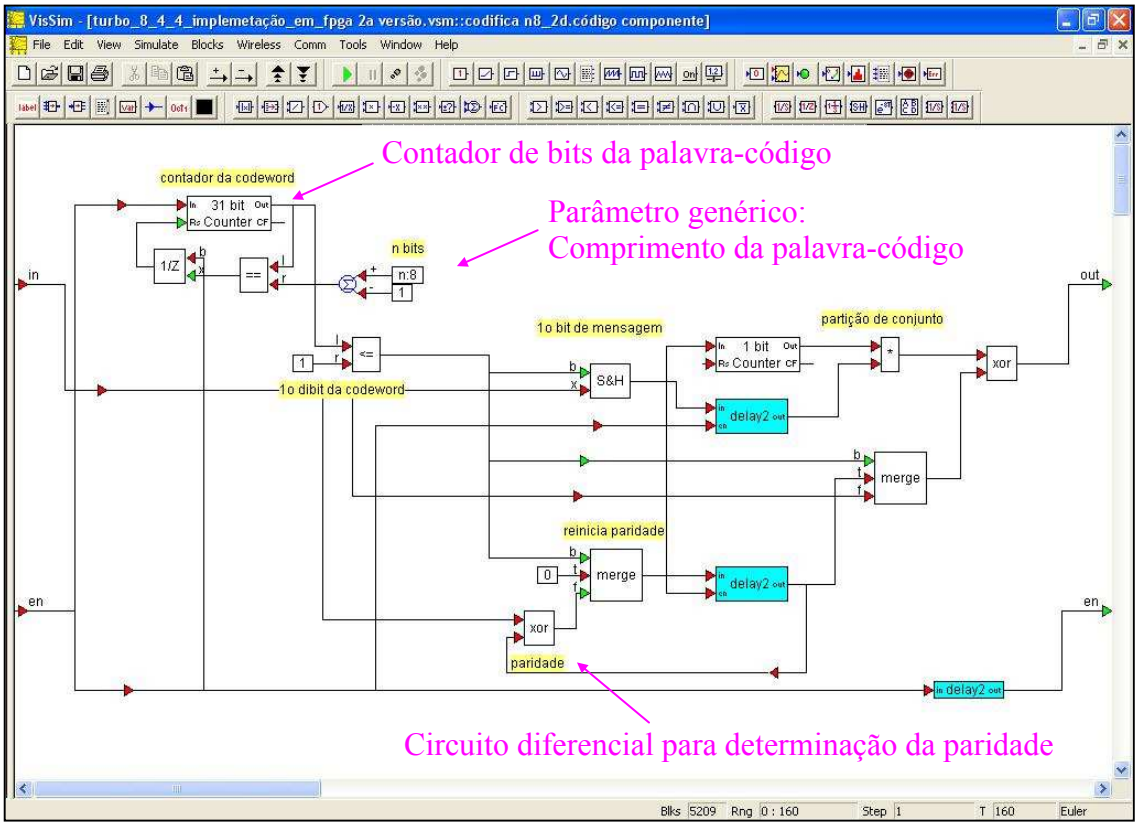


Figura III.4 *Estrutura proposta para o código componente (n,n/2,4), elaborada com o software VisSim/Comm.*

O circuito proposto para codificação opera a partir do recebimento do primeiro bit de mensagem definindo a partição de conjunto do código componente (*n*,*n*/2,4). Na Figura III.4 esse bit é registrado na saída do bloco "S&H" (*Sample and Hold*) quando o contador de bits da palavra-código (que varia de 0 a *n*-1) é menor ou igual a 1 (posição equivalente ao primeiro *dibit* codificado). Seqüencialmente os demais bits da mensagem são repassados para a saída, podendo ser invertidos ou não após meio intervalo de bit de entrada, segundo o alfabeto dessa partição - {00,11} ou {01,10}. Se o primeiro bit de entrada for 0, a partição do conjunto corresponde aos *dibits* {00,11} e nenhuma atitude de inversão adicional nos bits do código de paridade simples é necessária. No outro caso, quando o primeiro bit é igual a 1 e a partição {01,10}, a saída de um contador de 1 bit é combinada ao código de paridade simples em uma porta XOR, invertendo assim o

sinal de saída codificado a cada metade do tempo de bit de mensagem na entrada. Durante o intervalo do primeiro bit da palavra de mensagem se encerra também o processamento do cálculo da paridade da mensagem anterior. Esse cálculo é reinicializado a partir do segundo bit de mensagem. O bloco denominado "merge" na Figura III.4 atua como uma chave combinacional, exercendo uma operação equivalente a um "*IF Statement*" em VHDL, dependendo do valor da entrada *b* (*boolean*) a saída recebe o sinal da entrada *t* (*true*) ou *f* (*false*).

A seleção de processos VHDL mostrados na Figura III.5 correspondem à implementação da proposta previamente construída e simulada no *VisSim/Comm*, conforme ilustrado na Figura III.4. Pode-se observar a utilização do parâmetro *n* (tamanho da palavra-código) na definição do módulo do contador *cont*, que é direta ou indiretamente utilizado como sinal de controle em todos os processos destacados. Detalhes de alinhamento dos sinais de controle de saída *enc_out* e *res_enc_out*, iniciados somente após o recebimento e processamento do primeiro bit de mensagem (*cont* > 2), foram também previamente considerados e reproduzidos no *VisSim/Comm* com os blocos denominados "*delay2*" (ver Figura III.4).

A Figura III.6 ilustra basicamente a operação dos sinais de habilitação *en_enc_in* e *en_enc_out* indicando o momento adequado de captura dos bits de mensagem e da palavra codificada. Os sinais de *res_enc_in* e *res_enc_out* são utilizados na inicialização do processo de codificação e alinham o início da operação de formação do código com o estágio precedente que fornece a mensagem. O estado do sinal de *reset* recebido pelo circuito do código componente é então propagado para o próximo estágio assim que é iniciado o fornecimento da palavra codificada.

```
-- contador de bits de mensagem do codificador
PROCESS
BEGIN
    WAIT UNTIL mck='1';
        IF res_enc_in = '1' THEN
            cont<=(OTHERS=>'0');
        ELSIF en_enc_in='1' THEN
            IF cont=n-1 THEN
                cont<=(OTHERS=>'0');
            ELSE
                cont<=cont+1;
            END IF;
        END IF;
END PROCESS;
```

```
-- processo de codificação
PROCESS
BEGIN
    WAIT UNTIL mck='1';
        IF res_enc_in = '1' THEN
            enc_outi<='0';
        ELSIF en_enc_in='1' THEN
            IF cont=0 THEN
                enc_outi<=p;
            ELSIF cont(0)='0' THEN
                enc_outi<=enc_in;
            ELSIF cont(0)='1' THEN
                IF m(1)='1' THEN
                    enc_outi<=NOT(enc_outi);
                END IF;
            END IF;
        END IF;
END PROCESS;
```

```
-- determinação da partição de conjunto e paridade
PROCESS
BEGIN
    WAIT UNTIL mck='1';
        IF res_enc_in = '1' THEN
            p<='0';
            m<=(OTHERS=>'0');
        ELSIF en_enc_in='1' THEN
            IF cont=0 THEN
                m(0)<=enc_in;
                p<='0';
            ELSIF cont(0)='0' THEN
                p<=p xor enc_in;
            END IF;
            m(1)<=m(0);
        END IF;
END PROCESS;
```

```
-- gera sinais de reset e enable de saída
PROCESS
BEGIN
    WAIT UNTIL mck='1';
        IF res_enc_in = '1' THEN
            res_enc_out<='1';
            en_enc_out<='0';
            start<='0';
        ELSE
            IF cont=2 THEN
                start<='1';
            END IF;
            IF start='1' THEN
                res_enc_out<='0';
                en_enc_out<=en_enc_in;
            END IF;
        END IF;
END PROCESS;
```

"delay2"

Figura III.5 *Trechos selecionados do projeto VHDL do código componente (n,n/2,4).*

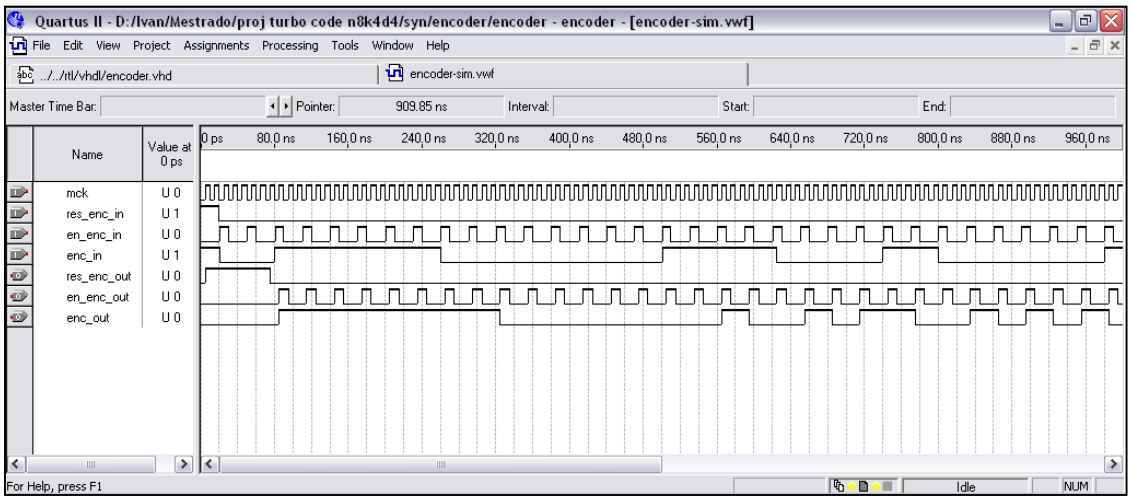


Figura III.6 *Simulação realizada com o Quartus II: sinais de entrada e saída e sinais internos do projeto VHDL do código componente (n,n/2,4).*

III.2.2 Construção do código produto $(8,4,4)^2$

A implementação do codificador $(8,4,4)^2$ é obtida basicamente com a concatenação serial do circuito do código componente (8,4,4) em 2 dimensões (Figura III.7). A descorrelação entre as etapas de codificação é assegurada por meio de um processo de entrelaçamento temporal da informação codificada a cada dimensão.

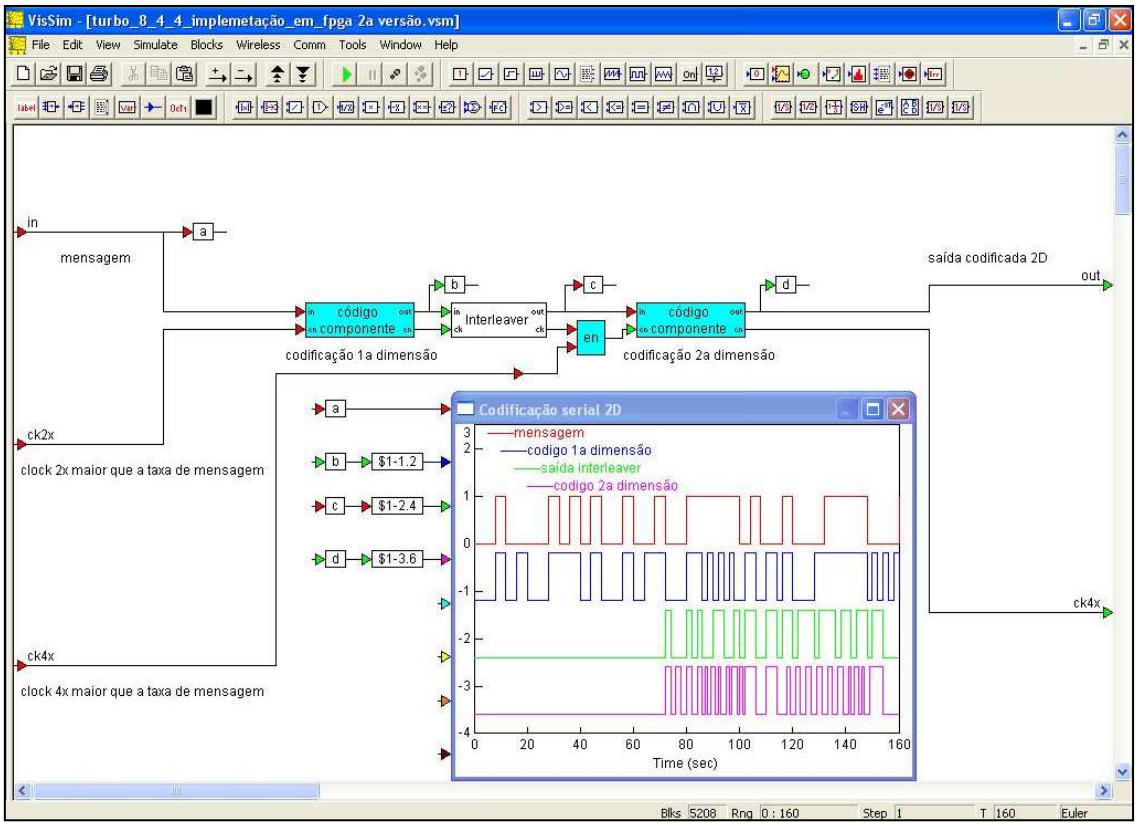


Figura III.7 *Implementação no VisSim/Comm código produto $(8,4,4)^2$ formado pela concatenação serial do código (8,4,4) em 2 dimensões.*

Na saída do primeiro estágio de codificação cada grupo de 16 bits de mensagem, considerado como um arranjo bidimensional 4×4, resulta em um bloco bidimensional codificado de 32 bits (4×8) utilizando o circuito de codificação (8,4,4). No bloco de entrelaçamento temporal (*interleaver*) os dados de saída do primeiro codificador são armazenados linha-a-linha e entregues coluna-a-coluna para o próximo codificador. A

saída resultante do segundo estágio de codificação é um bloco codificado de 64 bits na forma bidimensional 8×8.

III.2.3 Construção do entrelaçador de bloco

O entrelaçador temporal é construído como parte da memória RAM disponível no dispositivo FPGA e opera sobre os 4×8 bits entregues pelo codificador da primeira dimensão. A memória do bloco de entrelaçamento é dividida em duas partes: enquanto uma é usada para armazenar a saída do primeiro codificador linha-a-linha, a segunda entrega coluna-a-coluna os dados para o próximo codificador.

O controle de endereços de escrita da memória RAM, representado pelo sinal *cont* no código VHDL mostrado na Figura III.8, varia linearmente de 0 a $2\times N_c\times N_l-1$, onde N_c é o número de colunas e N_l *é* o número de linhas do bloco entrelaçador.

Enquanto os dados são escritos na primeira parte da memória (faixa de 0 a $N_c \times N_l - 1$), os endereços de leitura da outra parte, E_L, são determinados pela equação abaixo:

$$E_L = C_c+(C_l\times N_c)+(N_c\times N_l) \qquad \text{(III.1)}$$

onde C_c e C_l são, respectivamente, os contadores de colunas e de linhas. Quando a faixa de escrita vai de $N_c\times N_l$ a $2\times N_c\times N_l - 1$ o termo $N_c\times N_l$ na equação é ignorado.

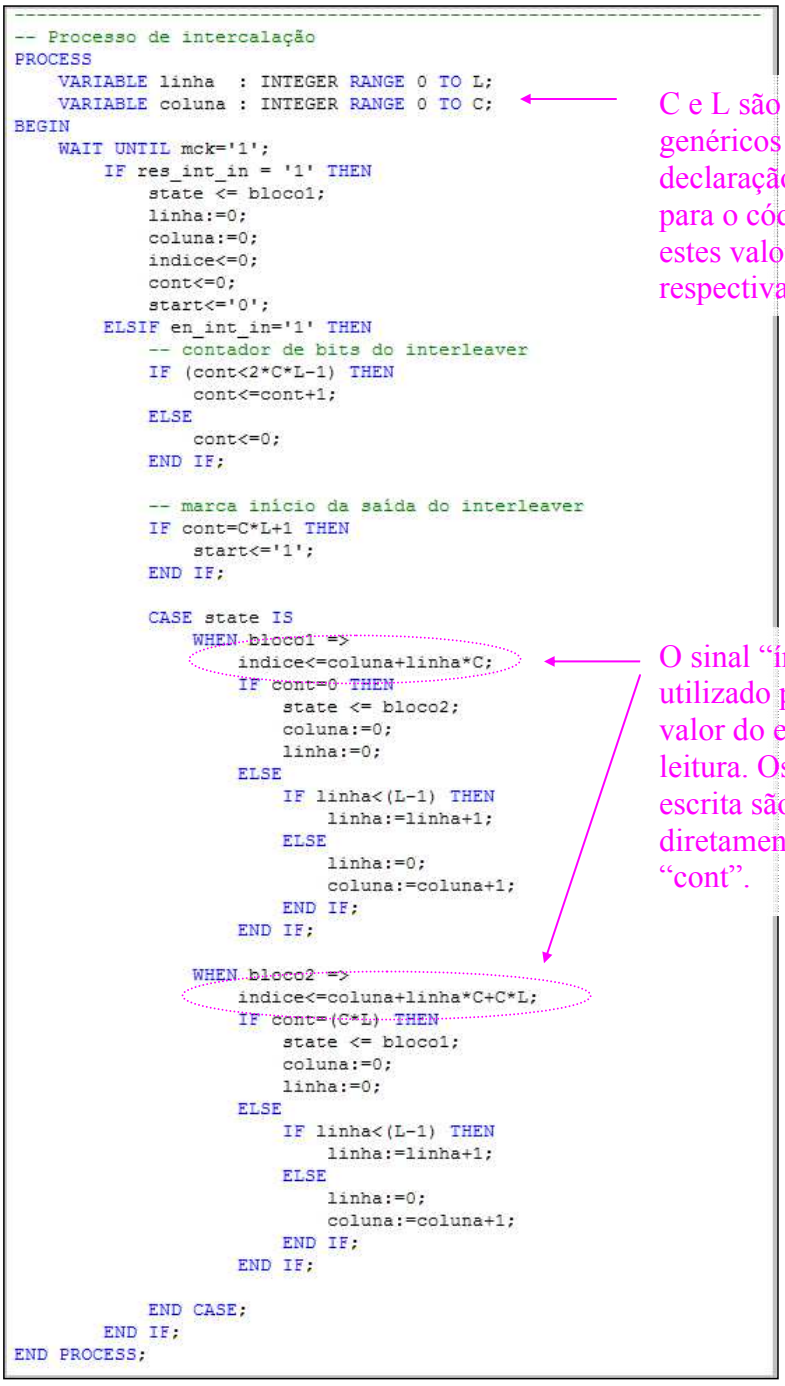


Figura III.8 *Processo VHDL para geração dos endereços de escrita e leitura da memória usada no circuito do entrelaçador.*

O resultado da equação (III.1) é atribuído ao sinal "*indice"* no processo VHDL na Figura III.8. Seu comportamento é detalhado na simulação funcional mostrada na Figura III.9 ("endereço para leitura em coluna").

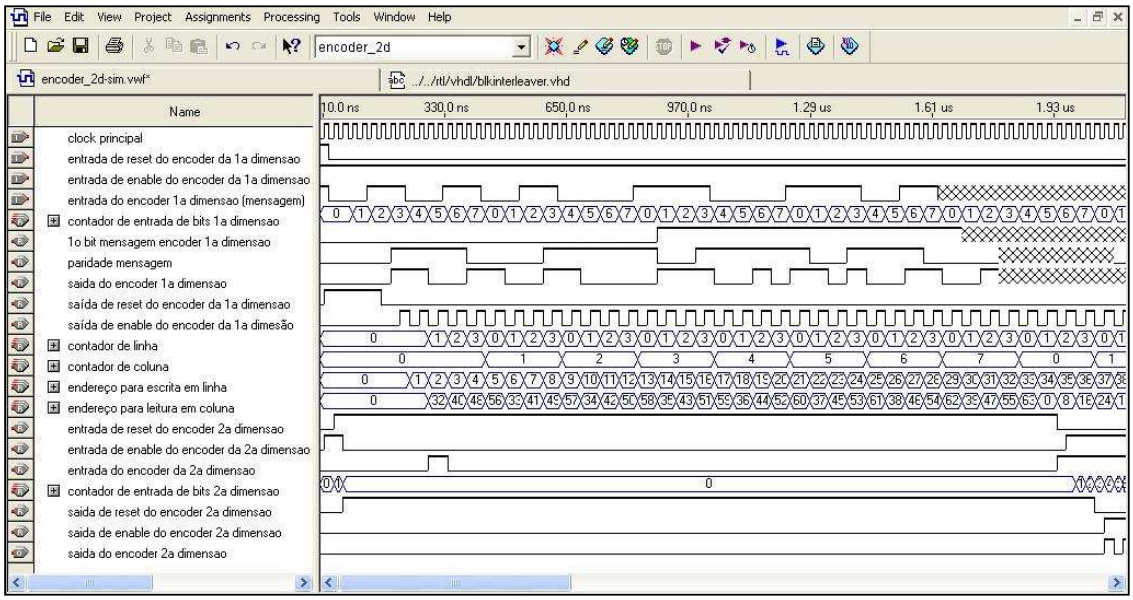


Figura III.9 *Sinais de entrada e saída dos componentes do codificador (8,4,4)^2 simulados com o software Quartus II.*

A utilização das entradas e saídas de controles de habilitação e de reinício (associados aos sinais de *enable* e *reset*) contorna a necessidade de se criar um controle central para integração dos blocos. Essa medida mantém a modularidade do conjunto e permite que a construção para um número de dimensões, *D,* qualquer seja uma simples concatenação dos sinais de entrada e de saída dos blocos de códigos componentes e entrelaçadores devidamente parametrizados.

III.2.4 Síntese do código produto (8,4,4)^2 no FPGA

A freqüência máxima de operação calculada pelo *Quartus II* para o circuito do codificador bidimensional (2D), composto por 2 codificadores (8,4,4) interligados pelo bloco entrelaçador de 4 linhas e 8 colunas, implementado de forma serial no FPGA da família Cyclone EP1C6T144C8, é de até 200 MHz, embora na prática valores maiores que 250 MHz tenham sido experimentados com sucesso na placa de teste

(provavelmente porque o componente na placa testada tenha características um pouco melhores do que a tipicamente assegurada).

Com o resultado calculado, taxas úteis de entrada de até 50 Mbps podem ser admitidas em qualquer dispositivo Cyclone C6 traço 8. Taxas ainda maiores podem ser atingidas em *chips* com maior *speed grade* (traço 7 ou 6).

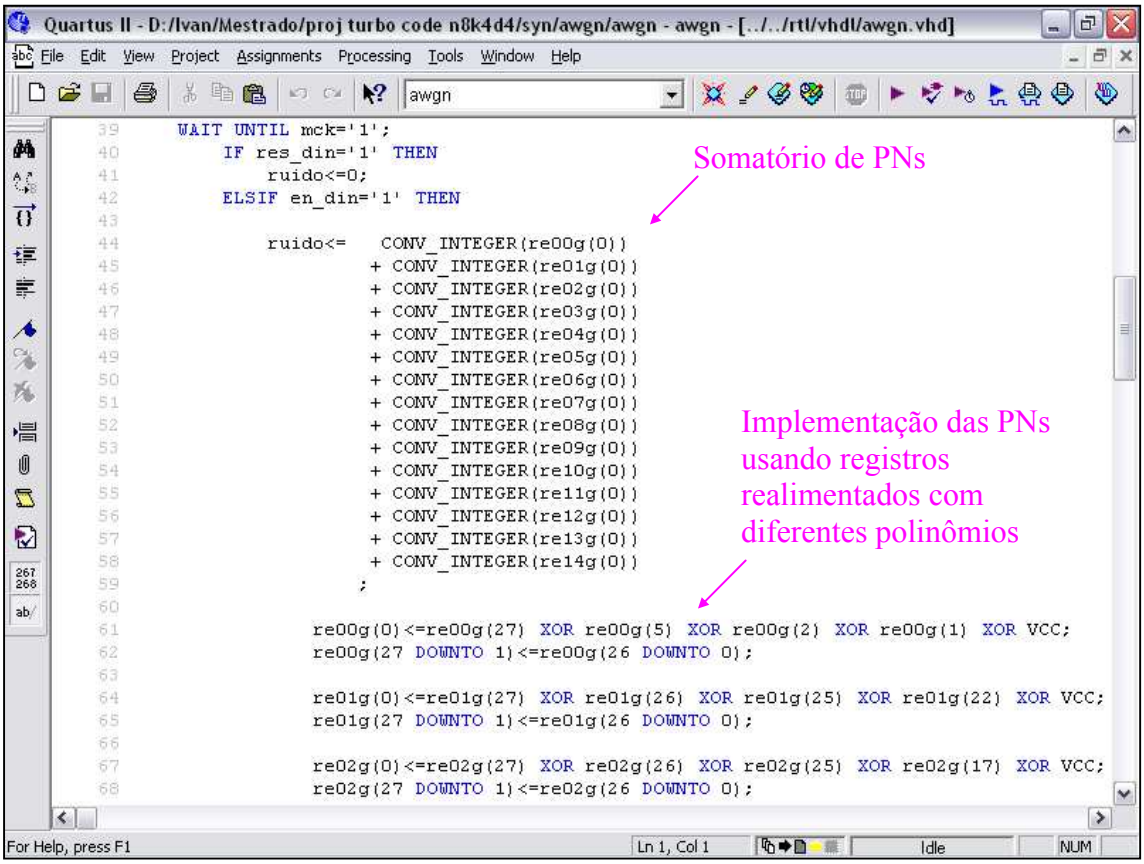
A montagem do projeto ocupou um recurso de 77 elementos lógicos e 64 bits de memória (menos de 1% da composição lógica disponível). Uma implementação do código produto $(8,4,4)^2$ por meio do mapeamento combinacional é ainda uma alternativa que pode ser investigada em trabalhos futuros para se elevar a freqüência de codificação: mensagens paralelas de *k* bits endereçando uma LUT com saída paralela de *n* bits podem operar com *clocks* acima de 200 MHz.

III.3 Implementação do canal AWGN vetorial

No intuito de se controlar e experimentar a degradação na palavra codificada da forma mais simples possível, construiu-se no FPGA um canal vetorial aproximadamente gaussiano para simular o comportamento do sinal recebido em um circuito ADC (*Analog to Digital Converter*) com 5 bits de quantização.

Para construir este canal AWGN, implementou-se em VHDL um circuito contendo 15 seqüências PNs (*Pseudo Noise*) de comprimento de 2^l-1 (Figura III.10). A soma destas PNs resulta em um sinal cuja distribuição se aproxima de uma curva Normal. O número de registro *l* adotado na implementação permite gerar seqüências de $2^{28}-1=268.435.455$ amostras aleatórias. Com este intervalo é possível medir com

assegurada aleatoriedade taxas de erro da ordem de até 10^{-6}, considerando em média a ocorrência de 100 erros.




```
39      WAIT UNTIL mck='1';
40          IF res_din='1' THEN
41              ruido<=0;
42          ELSIF en_din='1' THEN
43
44              ruido<=   CONV_INTEGER(re00g(0))
45                      + CONV_INTEGER(re01g(0))
46                      + CONV_INTEGER(re02g(0))
47                      + CONV_INTEGER(re03g(0))
48                      + CONV_INTEGER(re04g(0))
49                      + CONV_INTEGER(re05g(0))
50                      + CONV_INTEGER(re06g(0))
51                      + CONV_INTEGER(re07g(0))
52                      + CONV_INTEGER(re08g(0))
53                      + CONV_INTEGER(re09g(0))
54                      + CONV_INTEGER(re10g(0))
55                      + CONV_INTEGER(re11g(0))
56                      + CONV_INTEGER(re12g(0))
57                      + CONV_INTEGER(re13g(0))
58                      + CONV_INTEGER(re14g(0))
59                     ;
60
61                  re00g(0)<=re00g(27) XOR re00g(5) XOR re00g(2) XOR re00g(1) XOR VCC;
62                  re00g(27 DOWNTO 1)<=re00g(26 DOWNTO 0);
63
64                  re01g(0)<=re01g(27) XOR re01g(26) XOR re01g(25) XOR re01g(22) XOR VCC;
65                  re01g(27 DOWNTO 1)<=re01g(26 DOWNTO 0);
66
67                  re02g(0)<=re02g(27) XOR re02g(26) XOR re02g(25) XOR re02g(17) XOR VCC;
68                  re02g(27 DOWNTO 1)<=re02g(26 DOWNTO 0);
```

Figura III.10 *Trecho do programa VHDL que implementa o ruído aditivo do canal AWGN vetorial.*

A Figura III.11 mostra o histograma (10^6 amostras) obtido com o esquema proposto. A distribuição na verdade é binomial, mas para fins de comparação o contorno de uma fdp (função densidade probabilidade) Normal e de um histograma obtido com variáveis gaussianas geradas com o algoritmo de Box Muller [Box58] são sobrepostas a esta figura para ilustrar a aproximação.

Figura III.11 *Em cinza: histograma (10^6 amostras) obtido com a soma de 15 PNs com 28 registros. Em preto: sobreposição do contorno do histograma obtido com amostras produzidas com o algoritmo de Box-Muller. A curva contínua é de uma fdp Normal.*

As expressões que definem a taxa de ocorrência dos eventos $P(X = x)$, a média μ_X e a variância $\sigma_X{}^2$ na distribuição da Figura III.13 são mostradas a seguir, sendo que p é a probabilidade de ocorrência dos bits 0 e 1 na saída do gerador PN, l é o total de PN somadas e x é um evento específico sob análise.

$$P(X = x) = \binom{l}{x} p^x (1-p)^{l-x} \tag{III.2}$$

onde

$$\binom{l}{x} = \frac{l!}{x!(l-x)!} \tag{III.3}$$

$$\mu_X = l \times p \tag{III.4}$$

$$\sigma_X{}^2 = l \times p \times (1-p) \tag{III.5}$$

Com base nas equações (III.2) e (III.3), a seguir estão calculadas as probabilidades de alguns dos eventos mostrados no histograma da Figura III.11, bem como sua média e variância:

$$P(X=0)=P(X=15)=\binom{15}{0}0{,}5^{0}(1-0{,}5)^{15-0}=\frac{15!}{0!(15-0)!}\times 0{,}5^{15}\cong 3{,}05\cdot 10^{-5}$$

$$P(X=1)=P(X=14)=\binom{15}{1}0{,}5^{1}(1-0{,}5)^{15-1}=\frac{15!}{1!(15-1)!}\times 0{,}5^{15}\cong 4{,}57\cdot 10^{-4}$$

$$P(X=2)=P(X=13)=\binom{15}{2}0{,}5^{2}(1-0{,}5)^{15-2}=\frac{15!}{2!(15-2)!}\times 0{,}5^{15}\cong 3{,}2\cdot 10^{-3}$$

...

$$P(X=7)=P(X=8)=\binom{15}{7}0{,}5^{7}(1-0{,}5)^{15-7}=\frac{15!}{7!(15-7)!}\times 0{,}5^{15}\cong 1{,}96\cdot 10^{-1}$$

$$\mu_X = l\times p = 15\times 0{,}5 = 7{,}5$$

$$\sigma_X{}^2 = l\times p\times(1-p) = 15\times 0{,}5\times(1-0{,}5) = 3{,}75$$

O circuito que simula o canal AWGN usa a distribuição e os valores de probabilidade de ocorrência dos eventos obtidos com a soma das PNs para conferir à forma de onda de saída uma taxa de erro conhecida, equivalente a uma dada relação E_b/N_0. Isto é feito alocando-se o histograma Normal para certas posições dentro da faixa de quantização de 5 bits, em função do estado do bit de entrada.

Se a entrada do canal AWGN vetorial for o bit 0, a saída é o próprio resultado da soma das PNs somada a uma constante negativa. Quando o bit de entrada é 1, a saída é gerada pela diferença entre uma constante positiva e o sinal obtido com a soma das PNs. A estrutura desse circuito e as possíveis configurações de probabilidade de erro são mostradas na Figura III.12.

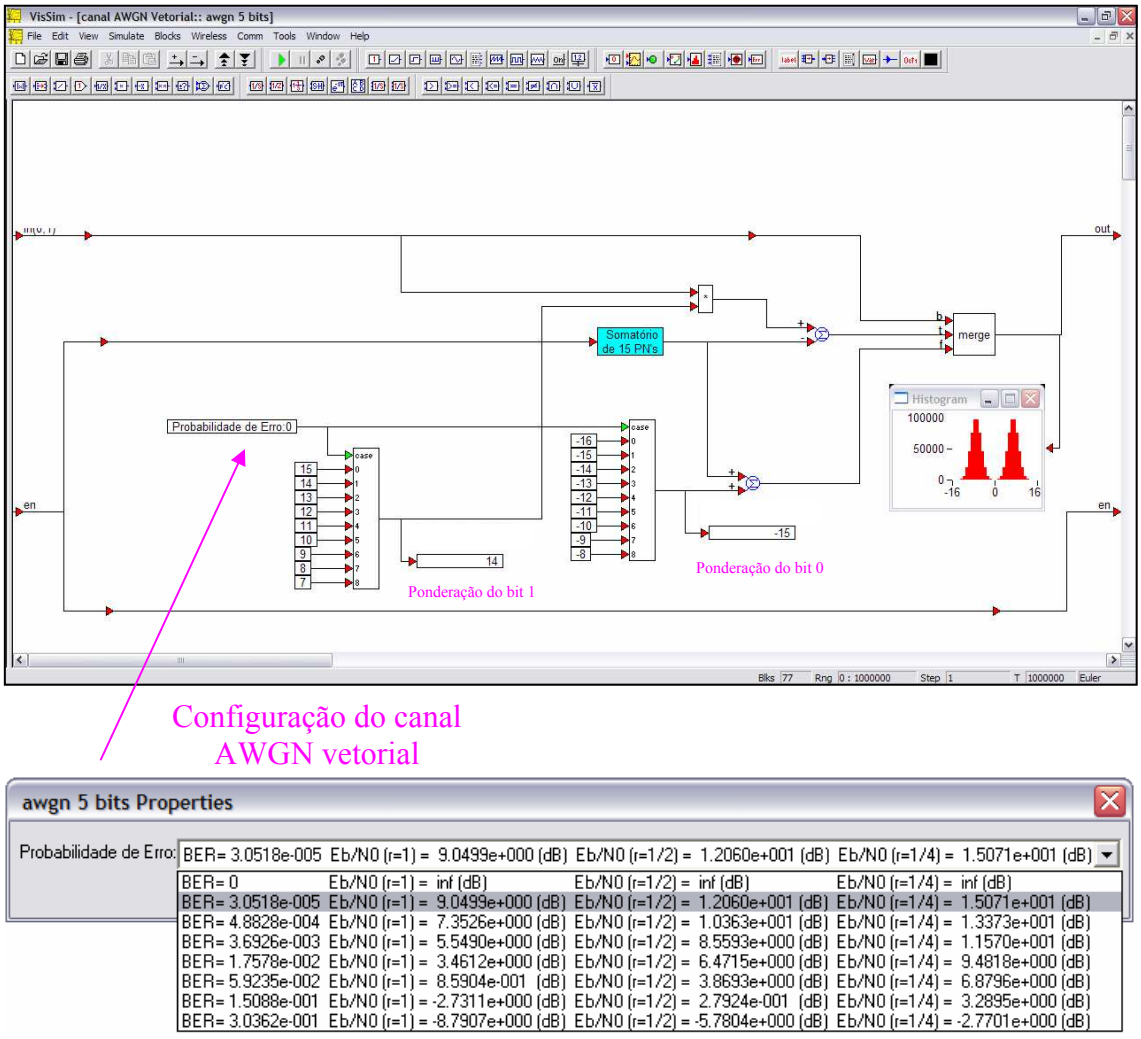


Figura III.12 *Circuito para implementação do canal AWGN, elaborado com o VisSim/Comm.*

Para simular uma condição de taxa de erro na palavra-código igual a, por exemplo, $3{,}05\times10^{-5}$ ($P(X=0)$), a saída do canal AWGN vetorial é composta de maneira que no limiar de comparação ocorra sobreposição dos valores de amplitude

assumidos pelos bits 0 e 1 (Figura III.13). Quando o bit de entrada for 1, subtrai-se da constante +14 o valor do somatório das PNs (que pode levar à condição $14-\sum PN=14-15=-1$, pertencente a região de valores do bit 0). Quando o bit de entrada for bit 0 adiciona-se a essa soma uma constante de valor −15 (que pode levar à condição $-15+\sum PN=-15+15=0$, cujo valor de amplitude é interpretado como bit 1).

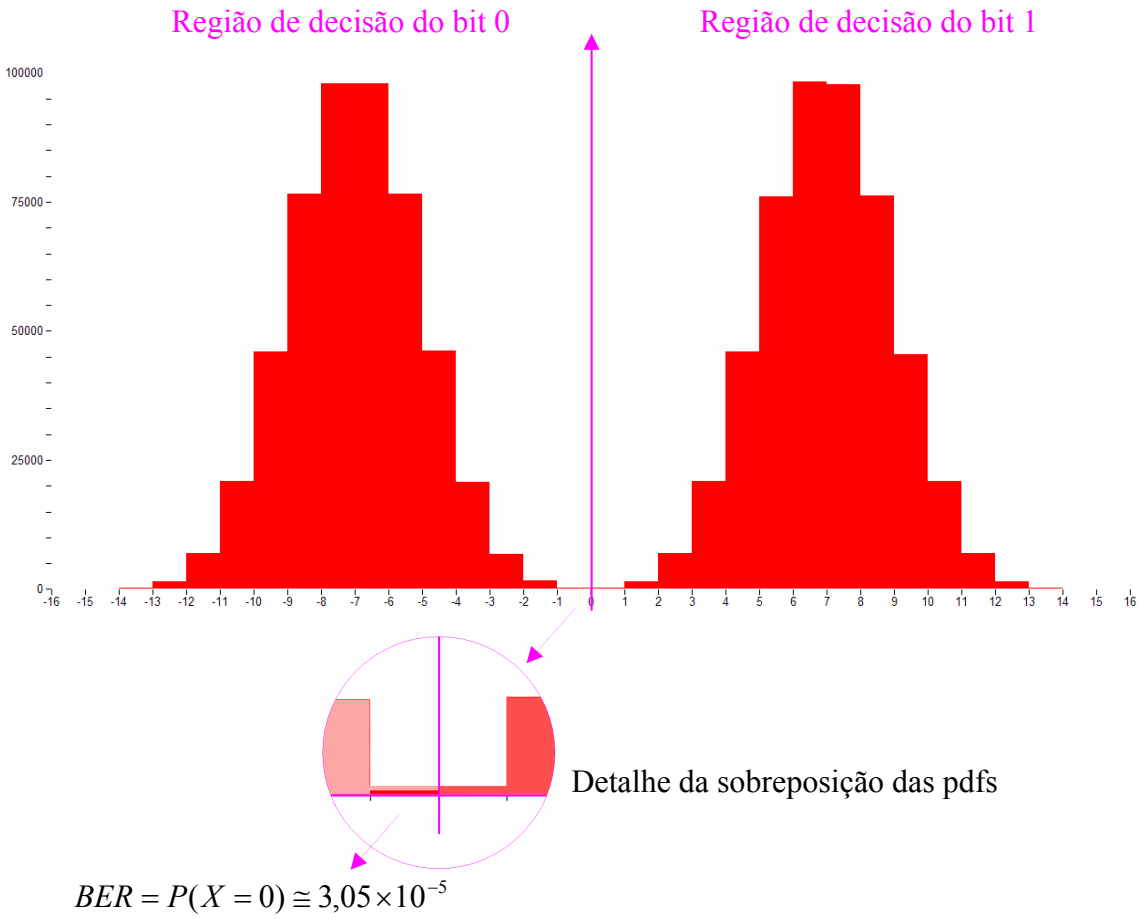


Figura III.13 *Histograma obtido na saída do canal AWGN vetorial configurado para taxa de erro igual a* $3{,}05\times10^{-5}$.

A Figura III.14 mostra o histograma na saída do canal AWGN vetorial configurado para uma probabilidade erro igual a $1{,}768\times10^{-2}$. Nesse caso, quando o bit

de entrada no canal for 1 subtrai-se da constante +11 o valor do somatório das PNs (que pode resultar nos valores -1 a -4, pertencentes a região de valores do bit 0). Quando o bit de entrada for bit 0 adiciona-se a essa soma uma constante de valor -12 (que pode levar aos resultados 0 a $+3$, cujos valores de amplitude são interpretados como bit 1). A probabilidade de erro provocada por esta combinação é igual à soma das probabilidades dos eventos $P(X=0)+P(X=1)+P(X=2)+P(X=3)$.

$$BER \cong 3{,}05\times10^{-5}+4{,}57\times10^{-4}+3{,}20\times10^{-3}+1{,}96\times10^{-1}\cong 1{,}758\times10^{-2}$$

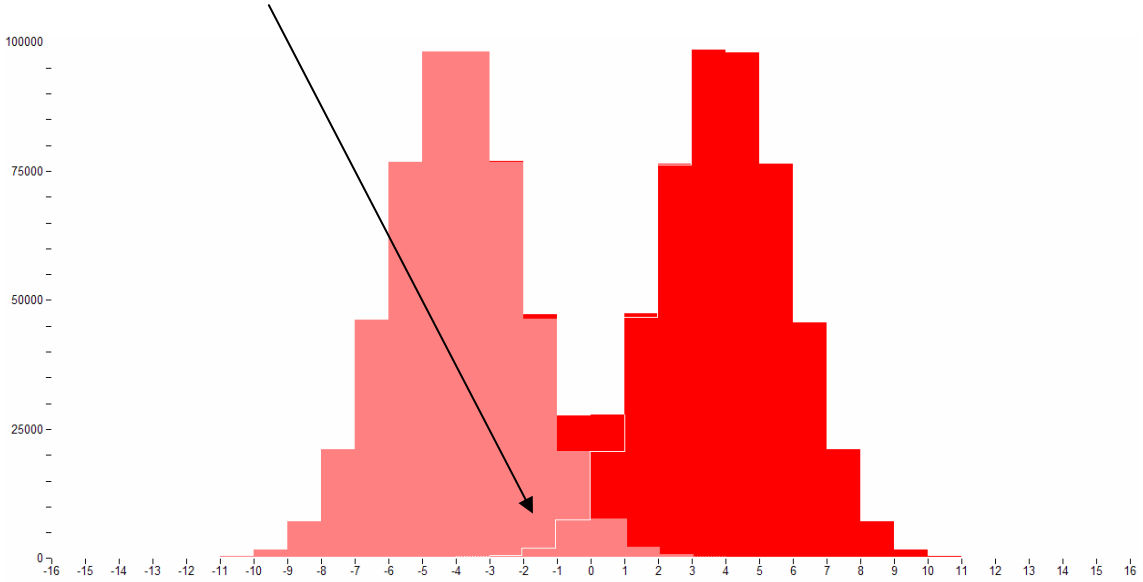


Figura III.14 *Histograma obtido na saída do canal AWGN vetorial configurado para taxa de erro igual a* $1{,}76\times10^{-2}$.

Os demais valores de taxa de erro do canal AWGN proposto, que aparecem listadas na Figura III.12, são obtidos em condições similares aos casos exemplificados e estão resumidos na curva de probabilidade acumulada mostrada na Figura III.15.

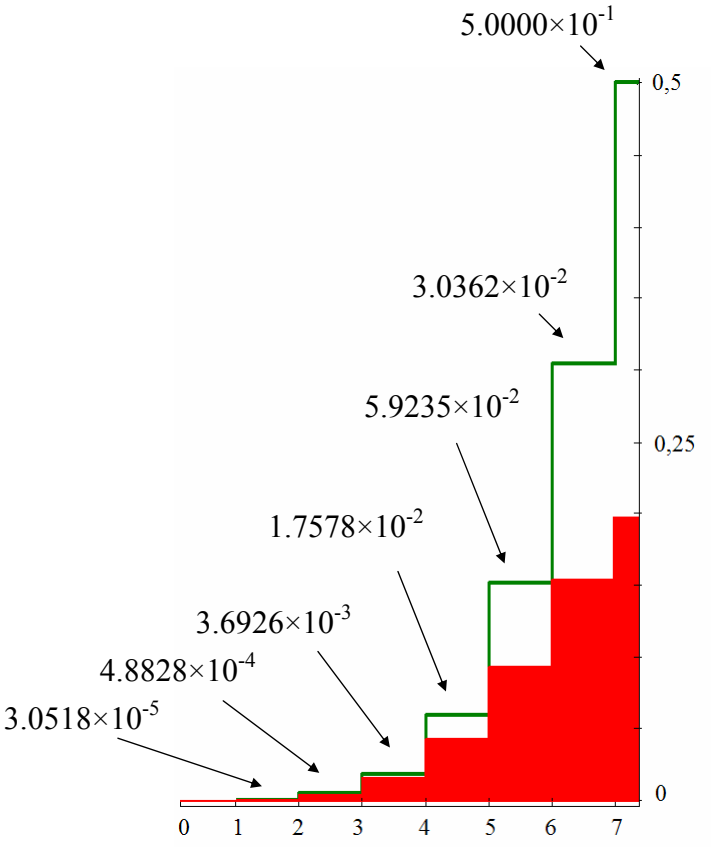


Figura III.15 *Curva de probabilidade de erro acumulada que define as possíveis configurações do canal AWGN vetorial.*

O valor de taxa de erro para sinalização antipodal não codificada na saída de canal AWGN, com uma dada relação E_b/N_0, pode ser encontrado com auxílio da expressão de probabilidade de erro de bit:

$$BER = \frac{1}{2}\left(1 - \mathrm{erf}\left(\frac{\sqrt{E_b}}{N_0}\right)\right) \qquad \text{(III.6)}$$

Reescrevendo a expressão (III.6) em termos da relação sinal/ruído SNR (*Signal to Noise Ratio*) em decibel tem-se:

$$BER = \frac{1}{2}\left(1 - \mathrm{erf}\left(\sqrt{10^{\frac{SNR}{10}}}\right)\right) \qquad \text{(III.6)}$$

E para uma dada condição de taxa de erro (BER) no canal AWGN o valor da SNR em decibel é igual a

$$SNR = 20\log_{10}\left(\mathrm{erf}^{-1}\left(1 - 2\times BER\right)\right) \quad \text{(dB)} \qquad \text{(III.7)}$$

onde $\mathrm{erf}^{-1}(x)$ é uma função de cálculo numérico inversa de erf(x). Esta função pode ser invocada, por exemplo, pelo comando "erfinv" no software *Matlab*.

Com base na equação (III.7), construiu-se então a tabela que é mostrada na Figura III.12. Nessa figura os valores de E_b/N_0 listados estão organizados em três colunas, cada qual relacionada a uma determinada taxa de codificação (R_c = 1, 1/2 e 1/4). Esta forma de descrever o canal AWGN vetorial é utilizada devido à necessidade de se compensar a redução de energia média por bit codificado (influência da taxa do código) no cálculo da equivalente relação *SNR* não codificada (equação III.8). Isto é, quando o canal AWGN vetorial se aplica, por exemplo, para um código com taxa 1/2 (como é o caso do código componente (8,4,4)), o valor de E_b/N_0 calculado pela expressão (III.7), obtido a partir da BER no canal, deve ser acrescido de 3 dB para normalizá-lo em relação ao acréscimo de banda, e quando a taxa do código é 1/4 (como é o caso do código produto $(8,4,4)^2$) o acréscimo é igual a aproximadamente 6 dB.

$$SNR_{não\,codificado} = 20\log_{10}\left(erfinv\left(1 - 2\times BER_{canal}\right)\right) + 10\log\left(\frac{1}{R_c}\right) \text{ (dB)} \qquad \text{(III.8)}$$

Dessa forma uma dada taxa de erro de bit do canal AWGN vetorial, como por exemplo, $1{,}76\times10^{-5}$, produz um efeito de degradação equivalente a uma relação de E_b/N_0 de 6,47 dB se aplicada no código de taxa 1/2, como o (8,4,4), e de 9,47 se aplicada no código produto de taxa 1/4, como o $(8,4,4)^2$.

A Figura II.4 apresentada no Capítulo II, apresenta um exemplo aplicado da relação entre a probabilidade de erro gerada pelo canal e sua correspondente relação E_b/N_0. No eixo horizontal os valores de E_b/N_0 tomados como referência na avaliação da decodificação usando a regra de Wagner são iguais a aproximadamente -5,7 dB, 0,3 dB, 3,8 dB, 6,5 dB e 8,6 dB. Esses valores de E_b/N_0, obtidos a partir da equação III.8, são respectivamente referentes às condições de probabilidade de erro $3{,}0\times10^{-1}$, $1{,}5\times10^{-1}$, $5{,}9\times10^{-2}$, $1{,}7\times10^{-2}$ e $3{,}7\times10^{-3}$, calculadas com auxílio das equações III.2 e III.3, na saída do canal AWGN vetorial atuando sobre um sinal codificado com taxa $R_c = 1/2$.

III.4 Implementação do algoritmo de Wagner

III.4.1 Construção do algoritmo de Wagner em uma estrutura seqüencial

A primeira versão do decodificador do código componente (8,4,4) implementado em FPGA foi concebida para operar serialmente, de forma que durante o carregamento de uma nova palavra se processasse a saída decodificada da palavra anterior. Basicamente duas estruturas paralelas, relacionadas às partições {00,11} e {01,10}, são alimentadas com os dados suaves quantizados em 5 bits nessa estrutura. De forma sincronizada, dois a dois os bits são agrupados para que as métricas (associadas às distâncias euclidianas) das seções em cada ramo sejam calculadas. À medida que as decisões são tomadas, um acumulador de métricas em cada ramo é atualizado. O decodificador aplica então a

regra de Wagner para o código de paridade simples de comprimento n/2, sobre o alfabeto binário {00, 11} e sobre o alfabeto {01, 10}, obtendo as decisões **ĉ** e **ĉ'**. Comparando as métricas entre a palavra-código recebida, **r**, e as decisões **ĉ** e **ĉ'** escolhe-se como decisão final a palavra que estiver mais próxima de **r**.

O trecho de código VHDL, mostrado na Figura III.16, apresenta apenas parte do código referente a estimação da palavra **c** (decodificação do ramo 0 – partição {00,11}) mas ilustra, em termos de estrutura comportamental, a forma como a idéia de decodificação seqüencial apresentada no parágrafo anterior foi implementada em FPGA.

Cerca de 130 elementos lógicos foram empregados na síntese deste circuito e o tempo de decodificação é de *n* pulsos de *clock* da palavra-código (*n*,*n*/2,4), podendo operar a taxas de até 115 MHz.

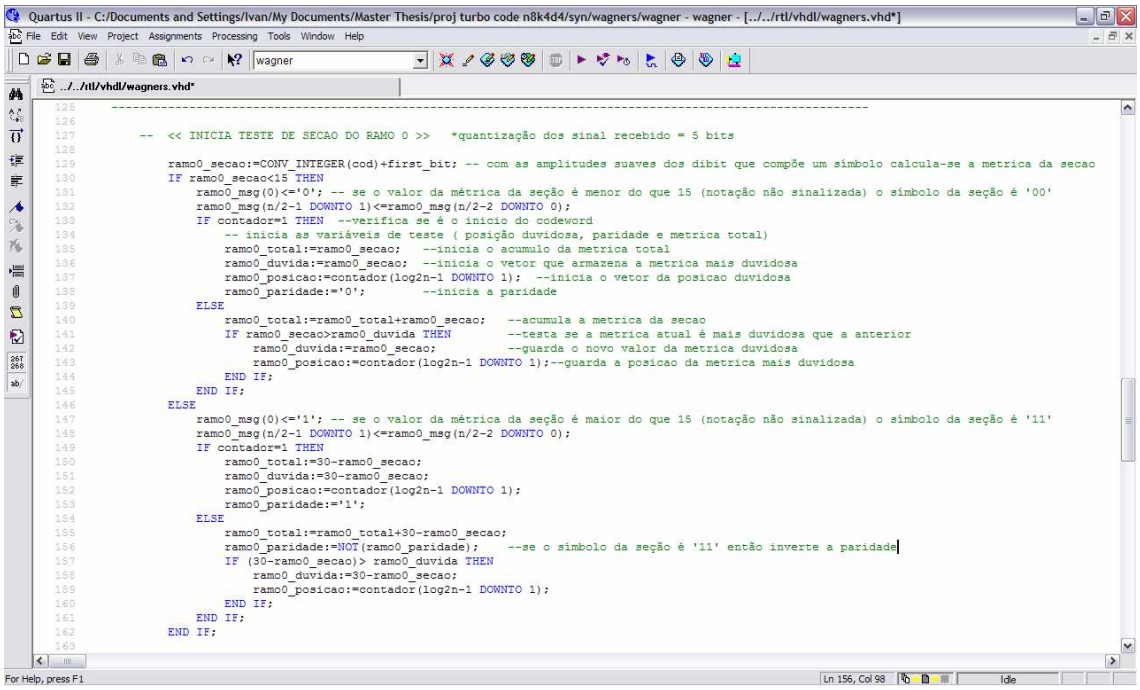


Figura III.16 *Implementação de um esquema seqüencial de decodificação da palavra-código (n,n/2,4) empregando a regra de Wagner.*

Essa primeira proposta, no entanto, foi preterida logo no começo dessa pesquisa e uma segunda versão do projeto foi elaborada para decodificar o código componente (8,4,4) usando uma forma puramente combinacional (paralela) do algoritmo de Wagner. O registro feito aqui tem o objetivo de incitar maiores investigações sobre este esquema de decodificação seqüencial em possíveis trabalhos futuros.

III.4.2 Construção do algoritmo de Wagner em uma estrutura combinacional

Nesse esquema de implementação os dados suaves recebidos serialmente são paralelizados em blocos de *n* bits, aplicados ao esquema combinacional de decodificação e em um único pulso de *clock* obtêm-se a saída abrupta decodificada.

O tempo necessário para decodificação isolada do código componente (8,4,4) é basicamente restringido pelo tempo de carregamento da palavra-código e pelo respectivo processo de resserialização na saída do decodificador.

Comparado ao sistema de decodificação seqüencial, a decodificação combinacional viabiliza de forma bem mais simples a construção de arranjos paralelos (processamento "força bruta") adaptados à decodificação turbo e por isso foi preferido para implementação.

Associando-se vários blocos de Wagner com estrutura combinacional para decodificação do código componente (8,4,4), é possível construir, com simplicidade, um sistema de decodificação turbo de um código produto $(8,4,4)^2$ que necessite de apenas um pulso de *clock* para perfazer uma iteração parcial completa, assunto este que será melhor apresentado no tópico III.5.

A Figura III.17 ilustra o arranjo proposto com o circuito combinacional para decodificação do código componente (8,4,4), elaborado com o auxílio do software *VisSim/Comm*. O bloco denominado "wagner (8,4,4)" é internamente constituído de dois circuitos de decodificação em paralelo, cada qual preparado para uma das partições de conjunto (ramos) utilizadas na composição do código (8,4,4). Dessa forma a palavra-código recebida é decodificada simultaneamente sobre os alfabetos {00,11} (ramo 0) e {01,10} (ramo 1) e a decisão final se dá pelo ramo com menor métrica acumulada.

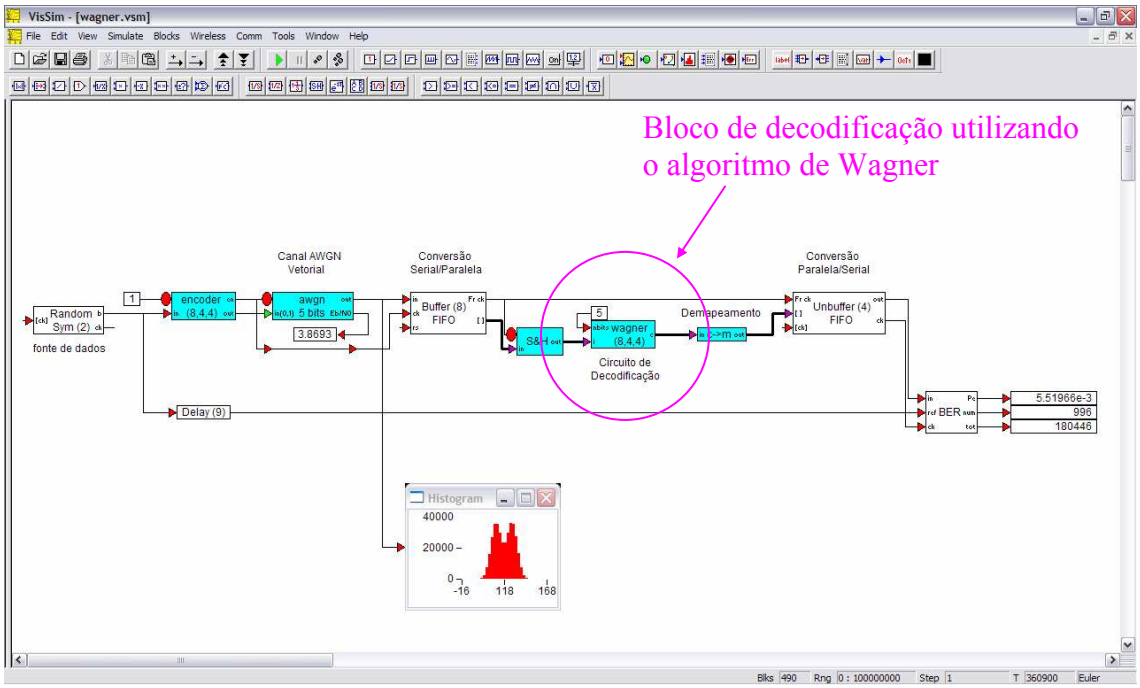


Figura III.17 *Projeto construído com auxílio do Software VisSim/Comm para decodificação do código componente (8,4,4) usando o algoritmo de Wagner.*

A Figura III.18 ilustra a estrutura de decodificação simultânea dos ramos 0 e 1 e o resultado de decodificação SIHO (*Soft Input Hard Output*) de uma palavra do código (8,4,4), quantizada em 5 bits com notação sinalizada em complemento 2, recebida após passar pelo canal AWGN vetorial.

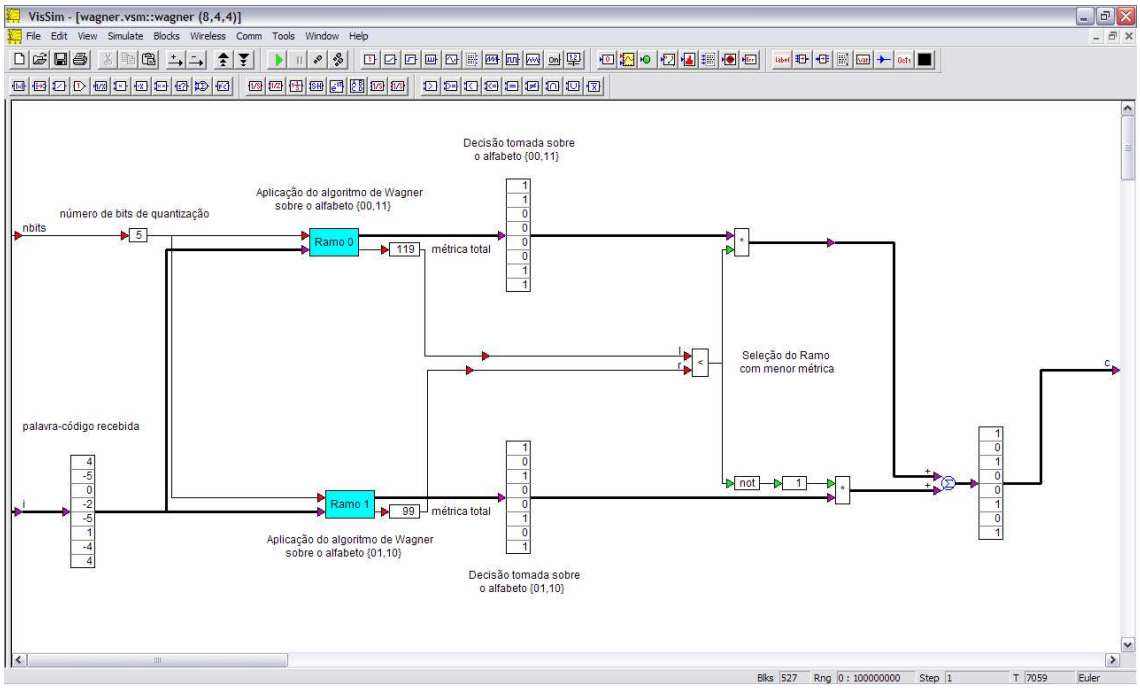


Figura III.18 *Implementação combinacional do algoritmo de Wagner no simulador VisSim/Comm.*

Nos blocos denominados “Ramo 0” e “Ramo 1” (Figura III.18) o vetor contendo a versão paralelizada da palavra-código recebida é analisado segundo a regra de Wagner. Assim em cada ramo ocorrem a seguinte sucessão de processos:

a) Os elementos do vetor são agrupados dois a dois (*dibits* da palavra-código recebida) e interpretados como símbolos de um código de paridade simples;

b) Calculam-se para estes *dibits* as correspondentes métricas em relação às seções {00} e {11} (ramo 0) ou {01} e {10} (ramo 1);

c) Aplica-se a verificação de paridade do código com base nas decisões tomadas pelas menores métricas em cada seção;

d) As menores métricas de cada seção são analisadas para determinação do símbolo mais duvidoso;

e) Em caso de não verificação da paridade inverte-se a decisão tomada na seção que contém o símbolo mais duvidoso;

f) A métrica total da palavra decodificada é igual ao somatório das métricas dos símbolos decididos em cada seção considerando o resultado da paridade.

Durante a elaboração do circuito combinacional para decodificação do código componente (8,4,4) usando a regra de Wagner foi possível investigar as possíveis limitações e as implicações da proposta de operação totalmente paralela dos processos listados há pouco. A Figura III.19 ilustra os sub-blocos que compõem o bloco de decodificação do ramo 0 e também alguns sinais intermediários obtidos com as operações de a) a f) listadas acima.

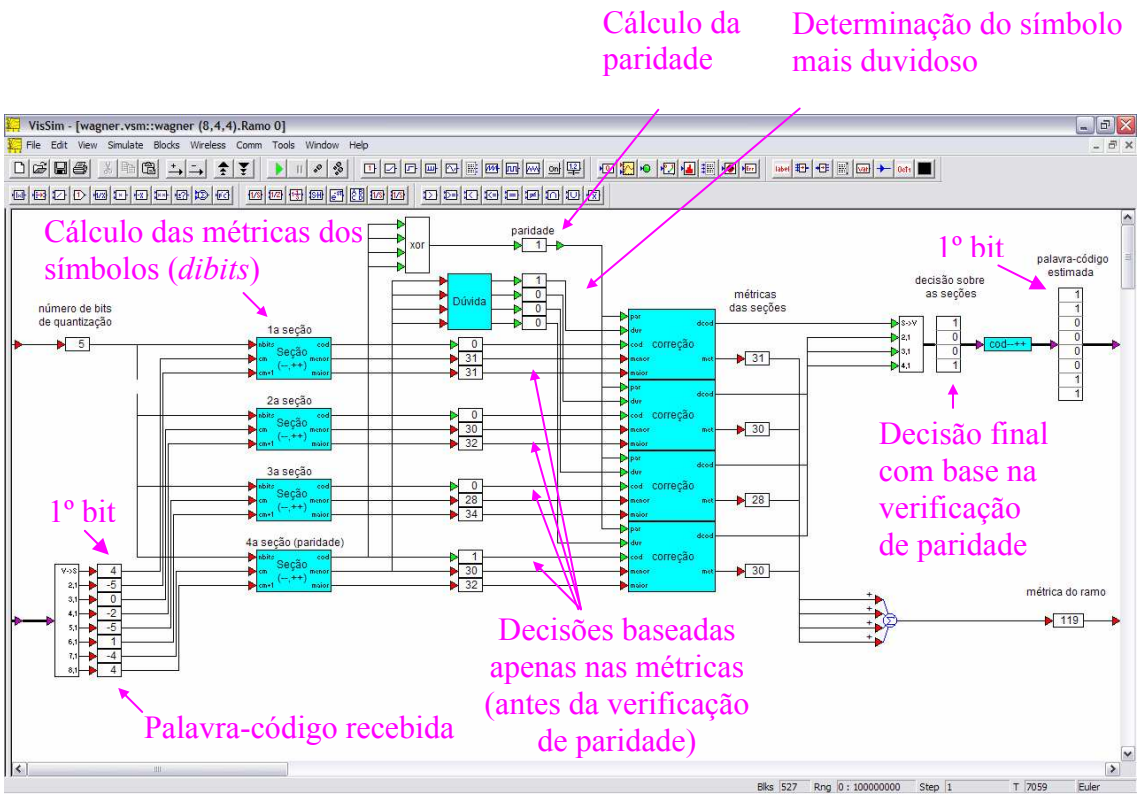


Figura III.19 *Estrutura interna para decodificação da palavra-código recebida em relação ao alfabeto {00,11} (ramo 0).*

Pôde-se constatar que, embora as etapas de decodificação sejam seqüencialmente dependentes, não existem laços de realimentação entre os processos e

que, tal como na simulação, eles poderiam ser implementados no dispositivo FPGA utilizando apenas lógica combinacional.

III.4.2.1 Cálculo das métricas das seções

O cálculo da métrica das seções implementado em VHDL difere um pouco do processo adotado nas investigações de Guimarães [Gui03], que é o originalmente proposto por Wagner. Ao invés de se encontrar a distância euclidiana quadrática em relação ao valor esperado do símbolo, investigou-se nesta dissertação um cálculo alternativo, sem o uso de potenciação ou de estimação do valor médio do valor recebido. Objetivou-se reduzir significativamente a quantidade de lógica a ser utilizada no FPGA, mesmo que sob uma possível redução de desempenho na decodificação.

Na implementação desse cálculo considera-se que o sinal recebido na entrada do conversor ADC (*Analog to Digital Converter*) do circuito de decodificação seja ajustado dinamicamente, através de um sistema AGC (*Automatic Gain Control*), a uma faixa pré-estabelecida de valores. A atuação que se espera do circuito AGC é ilustrada na Figura III.20.

Na Figura III.20 os valores máximos $+V$ e $-V$ assumidos pelo sinal na entrada do circuito ADC são mantidos constantes através da atuação do sistema AGC. Os valores médios dos símbolos recebidos variam como conseqüência da acomodação da dispersão do ruído gaussiano aditivo dentro da faixa dinâmica $\pm V$. Os símbolos da palavra-código são então quantizados em 2^{nbits} níveis discretos de amplitude, uniformemente distribuídos entre $\pm V$. Para uma quantização com *nbits* = 5 bits, os valores máximos que

podem ser assumidos pelos símbolos quantizados correspondem aos inteiros +15 e -16 (notação em complemento de 2).

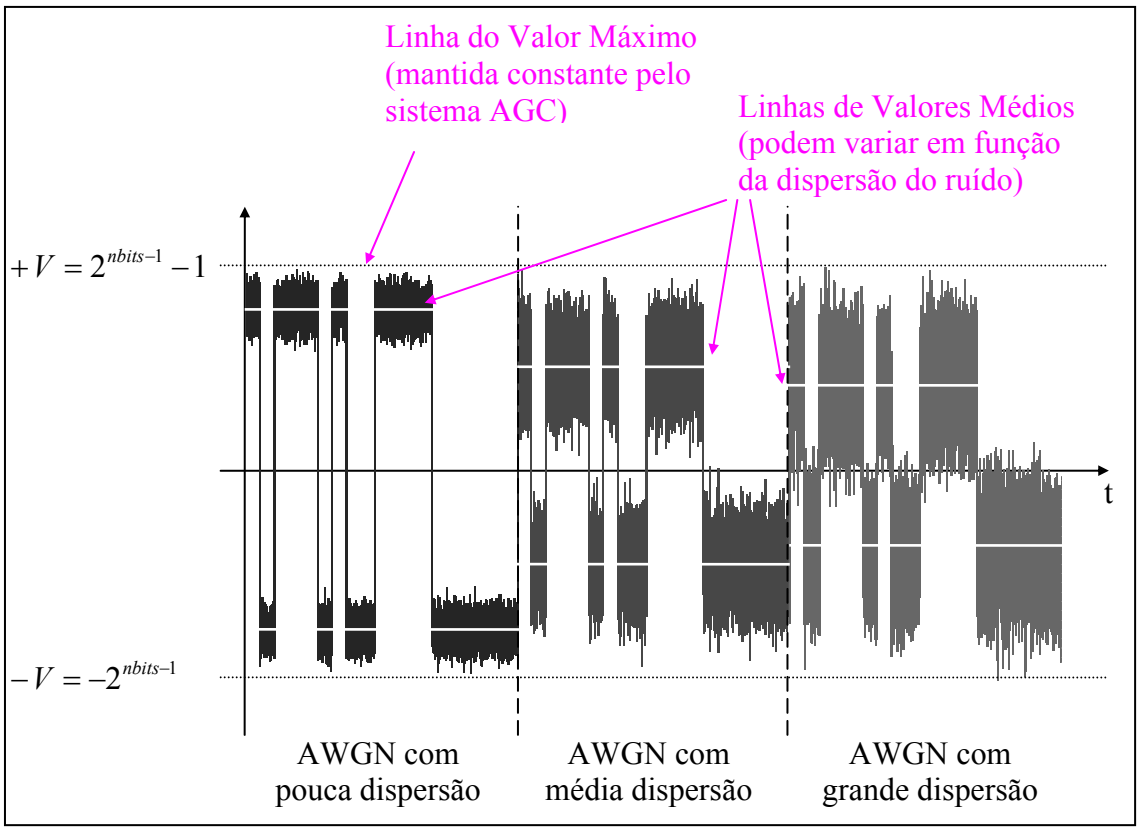


Figura III.20 *Atuação de um circuito AGC sobre um sinal antipodal contaminado por ruído gaussiano aditivo.*

No método aqui utilizado, ao invés de se calcular as distâncias euclidianas quadráticas em relação aos valores médios, a medida das métricas é então simplesmente estimada com base na distância entre o valor recebido quantizado e o valor máximo afixado pelo circuito AGC/ADC.

O trecho de programa, mostrado na Figura III.21, apresenta a forma como é construído, em VHDL, o cálculo da métrica de um *dibit* da palavra-código em uma das seções do ramo 0 (alfabeto {00,11}) usando lógica combinacional. Este sub-bloco é denominado de "seção(−−,++)" no sistema construído no *VisSim/Comm* (Figura III.19).

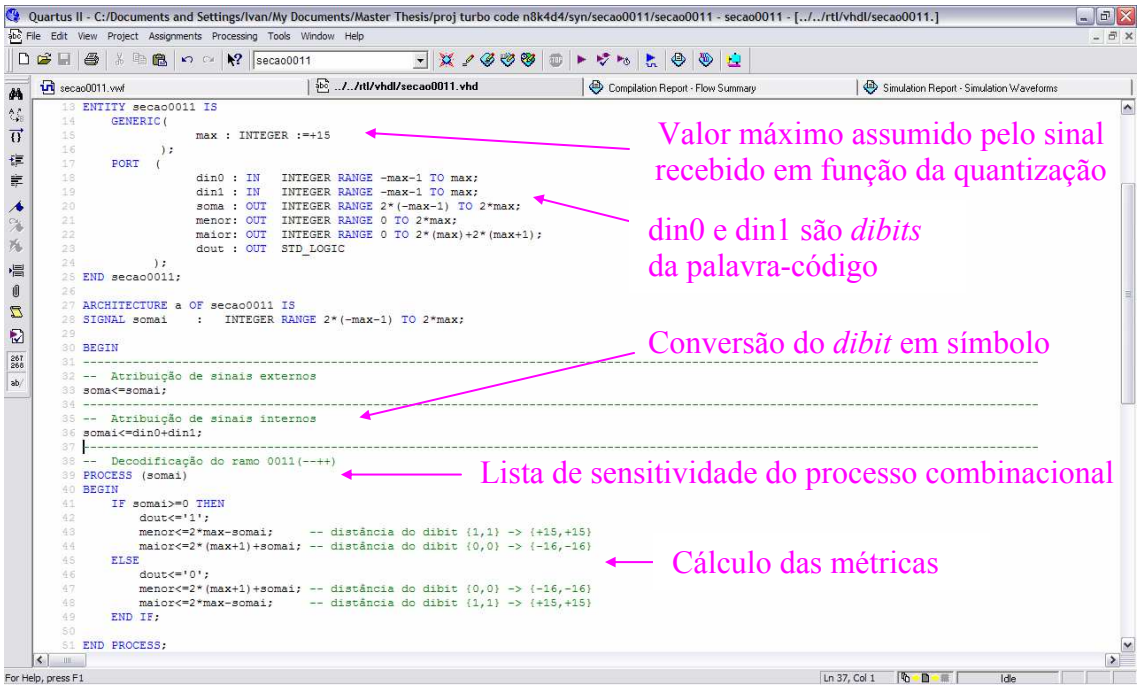


Figura III.21 *Cálculo da métrica de uma das seções do ramo 0 (alfabeto {00,11}).*

A Figura III.22 mostra uma simulação feita no *Quartus II* que exemplifica a operação do circuito de cálculo da métrica em uma seção do ramo 0. O tempo de propagação da resposta do circuito combinacional simulado para o Cyclone C6 traço 8 é de aproximadamente 15 nanossegundos. Este tempo de propagação corresponde ao tempo necessário à mudança e estabilização da resposta na saída do circuito.

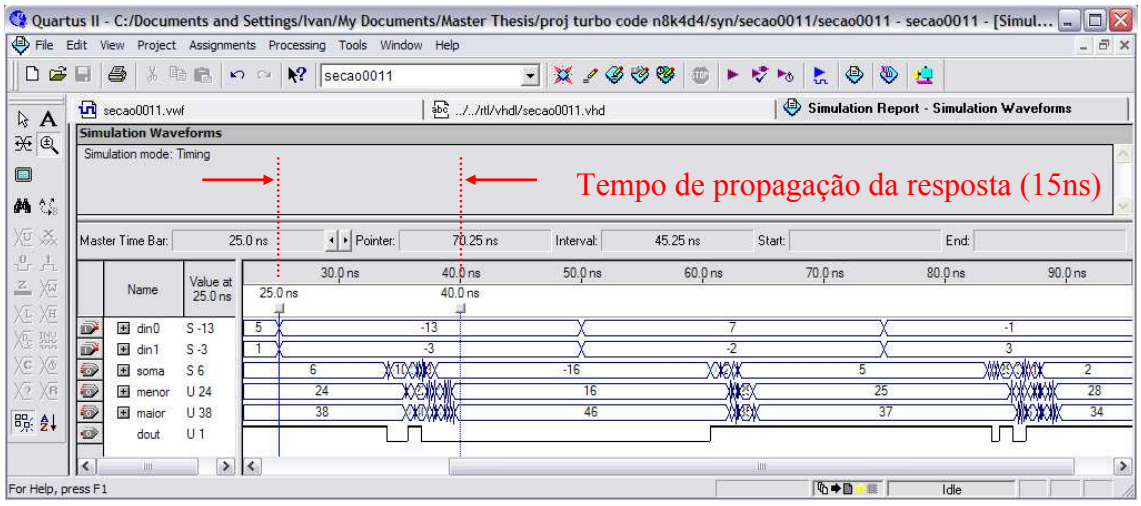


Figura III.22 *Simulação do cálculo da métrica de uma seção do ramo 0.*

O *dibit* recebido, {−13,−3} por exemplo, é comparado aos valores máximos {+15,+15} e {−16,−16}, que são respectivamente os valores esperados dos símbolos {11} e {00} do alfabeto do ramo 0. O sinal denominado "soma" apresenta o resultado da soma das amplitudes dos bits do *dibit*, $soma = -13 - 3 = -16$, e o sinal "dout" a decisão tomada na seção, $dout = 0$. Ou seja, quando o resultado da soma é maior ou igual a zero a menor métrica da seção é aquela medida em relação ao par {11} e a maior em relação ao {00}, se o resultado da soma é negativo (como é o caso do exemplo). Então, para o *dibit* {−13, −3}, a menor métrica é em relação ao par {00},

$$(-16) - (-13) + (-16) - (-3) = 16,$$

enquanto a maior é em relação ao par {11},

$$15 - (-13) + 15 - (-3) = 46.$$

No caso do ramo 1 o mesmo raciocínio é aplicado em relação aos valores esperados {+15, −16} e {−16,+15}, relativos aos pares {10} e {01}.

III.4.2.2 Determinação do símbolo mais duvidoso e seleção final da palavra decodificada

As lógicas internas dos blocos "Dúvida" e "Correção" (Figura III.19), transcritos em VHDL na Figura III.23 em entidades renomeadas como "maior"e "decisão", são funções bem simples, constituídas apenas de blocos comparadores para determinação da maior métrica e seleção final da palavra decodificada. Como pode ser visto na entidade

"decisão", a escolha da métrica final da seção toma por base o resultado de paridade e a confiabilidade resultante da definição da métrica mais duvidosa.

Código VHDL referente ao bloco "correção" da Figura III.19

Código VHDL referente ao bloco "dúvida" da Figura III.19

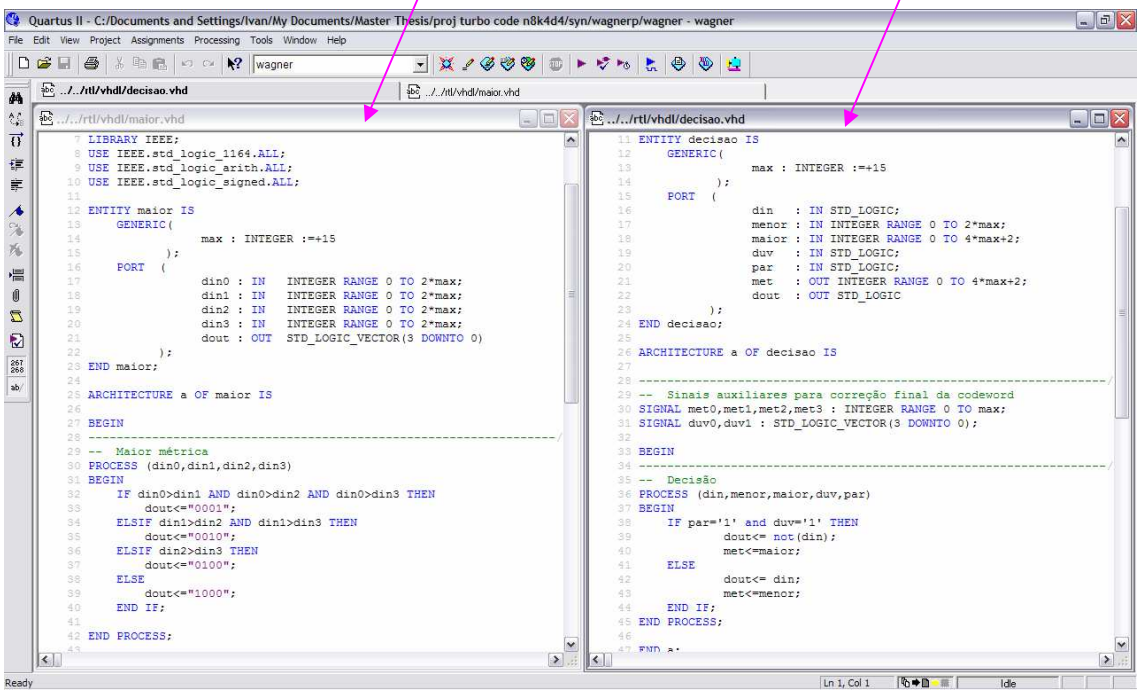


Figura III.23 *Transcrição em VHDL dos blocos "Dúvida" e "Correção" mostrados na Figura III.19.*

O programa VHDL que efetua a operação completa de decodificação do ramo 0 utiliza os componentes "secao0011", "maior" e "decisão" compondo um esquema semelhante ao mostrado na Figura III.19. A construção da decodificação do ramo 1 é feita da mesma forma, apenas troca-se o alfabeto avaliado utilizando então a entidade "secao0110".

III.4.3 Síntese do algoritmo de Wagner no FPGA

A integração dos blocos "ramo 0" e "ramo 1" é mostrada no esquema RTL[6] da entidade denominada "wagnerp" (Figura III.24).

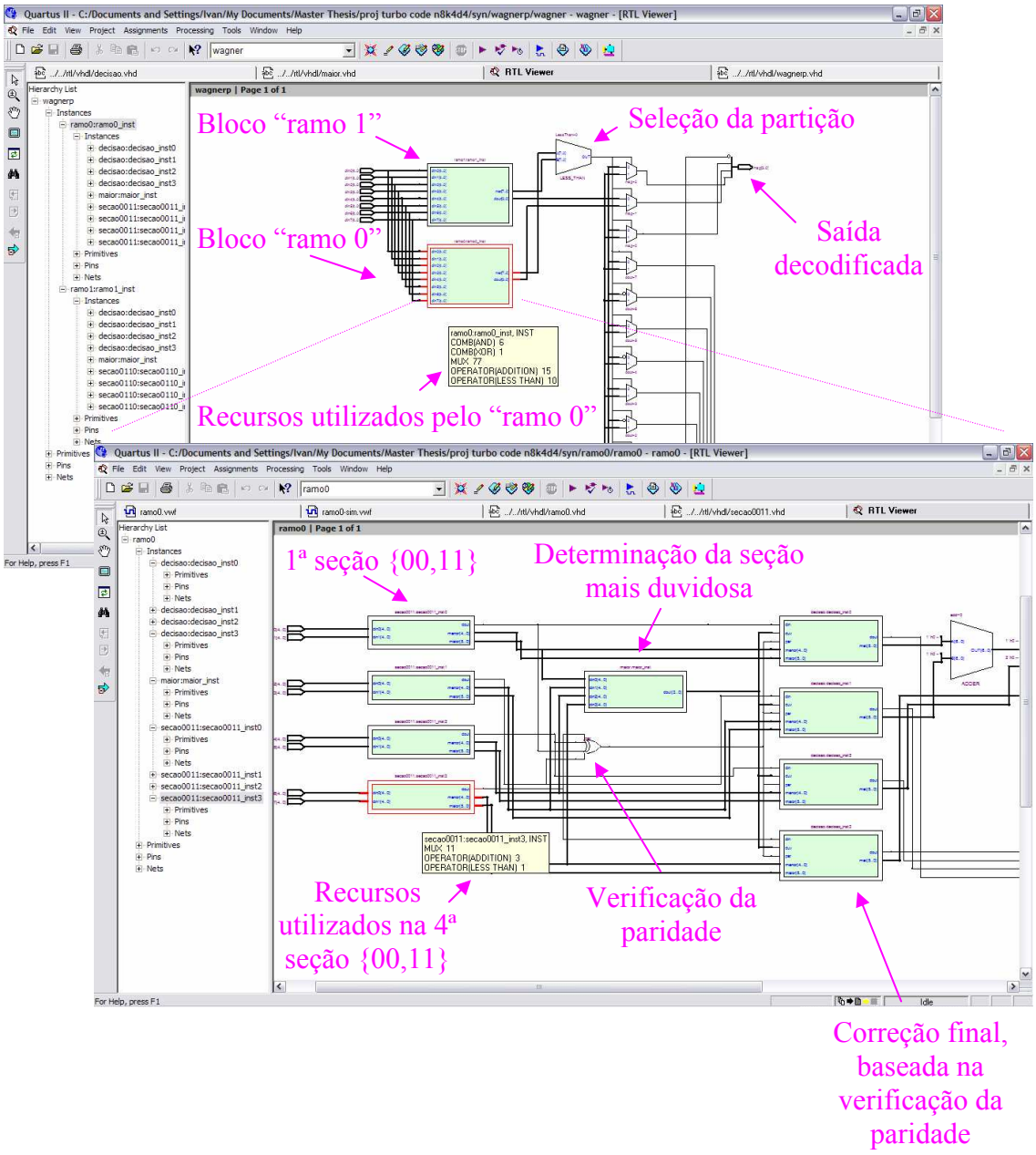


Figura III.24 *Visualização do circuito esquemático do algoritmo de Wagner utilizando a ferramenta RTL Viewer do Quartus II.*

[6] RTL - *Register Transfer Level* é um tipo intermediário de descrição de hardware (HDL - *hardware description language*) contendo mapeamento de portas lógicas e registradores. A ferramenta *RTL Viewer* da ALTERA permite visualizar, na forma de um esquema elétrico, um projeto sintetizado a partir de um código VHDL.

Como esperado, o diagrama de conexão dos registros lógicos sintetizados, que é gerado automaticamente pelo compilador, conservou a mesma forma da construção planejada com o *VisSim/Comm* (comparar a Figura III.24 com as Figuras III.18 e III.19).

III.4.3.1 Recursos lógicos utilizados pelo decodificador combinacional usando a regra de Wagner

A utilização de recursos do FPGA Cyclone pelo decodificador de Wagner implementado está detalhada na tabela mostrada na Figura III.25.

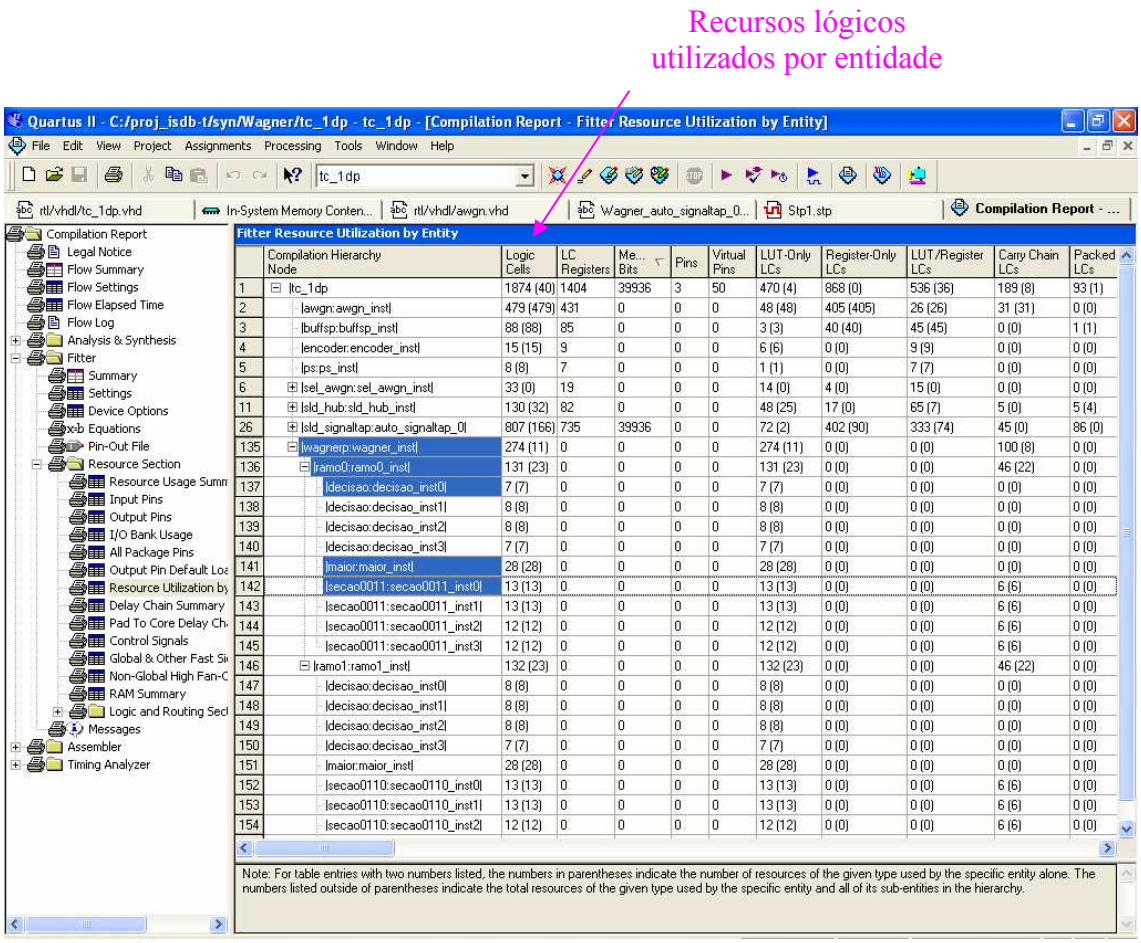


Figura III.25 *Recursos do FPGA empregados no projeto do decodificador de Wagner.*

Um total de 274 elementos lógicos (4,5% da lógica disponível no Cyclone C6) foram empregados no circuito de decodificação combinacional quantizado em 5 bits. Os circuitos de cada partição, blocos "ramo 0" e "ramo 1", utilizaram praticamente a mesma razão do recurso, respectivamente 131 e 132 elementos lógicos.

A baixa complexidade de implementação é assegurada pelo pequeno número de elementos lógicos utilizados pelos componentes elementares do projeto. Individualmente os circuitos para cálculo de seções, "secao00", o circuito para determinação do símbolo mais duvidoso, "maior", e o circuito de verificação de paridade e decisão do ramo, "decisão", não utilizam mais do que 28 elementos lógicos cada.

III.4.3.2 Freqüência máxima de operação

O tempo de decodificação do código componente (8,4,4) com o circuito combinacional implementado no Cyclone C6 traço 8 de forma otimizada ficou em torno de 19 ns. Isto corresponde a um taxa de operação do *clock* do circuito até um pouco mais de 50 MHz e taxa de transferência útil (4 bits de mensagem decodificados entregues em paralelo) maior que 200 Mbps.

III.4.3.3 Verificação da operação do circuito usando SignalTap

Outro recurso também muito utilizado no projeto foi a ferramenta *SignalTap* do *Quartus II* (Figura III.26). Essa ferramenta consiste numa função de análise lógica,

incorporada ao circuito projetado, capaz de capturar sinais de quaisquer nós internos ou pinos de I/O em tempo real e na velocidade de operação do sistema através da interface JTAG do FPGA. Além da possibilidade de escolha dos sinais a serem capturados, é possível configurar as condições de disparo (*trigger*) do circuito de aquisição, escrever e ler dados na memória interna do FPGA, bem como escolher a formatação dos dados apresentados na interface gráfica do *Quartus II*.

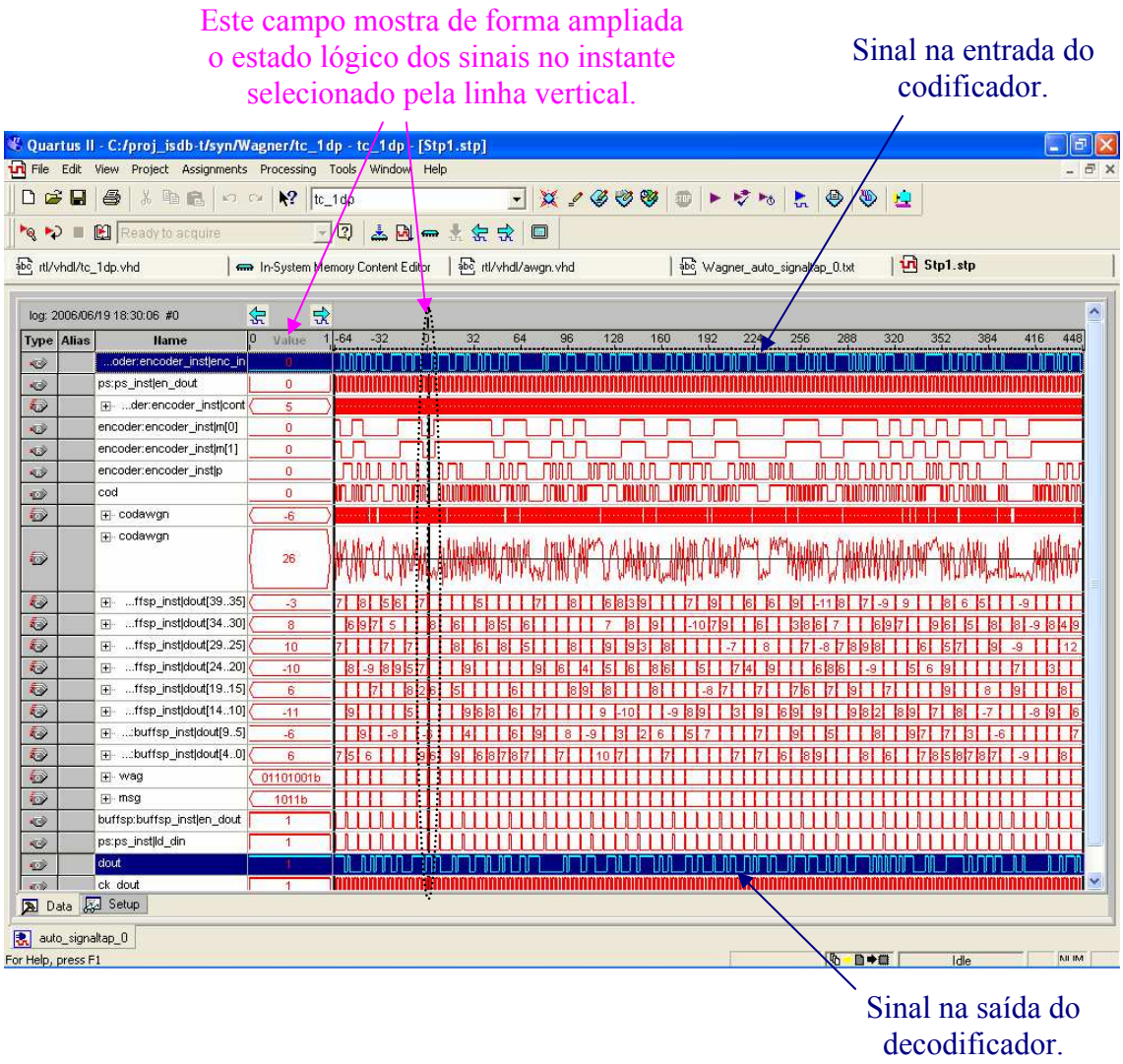


Figura III.26 *Sinais capturados internamente no FPGA durante uma operação real do circuito de decodificação implementado.*

Durante a fase inicial do projeto a constatação da correta operação dos sub-blocos do decodificador de Wagner foi realizada com auxílio do simulador do *Quartus*

II. Essa verificação consistia basicamente na análise de trechos envolvendo pequenos transitórios ou em situações pontuais. Na integração final dos blocos utilizou-se mais freqüentemente a ferramenta *SignalTap* como recurso de validação do sistema.

Uma das motivações dessa abordagem é que, embora o recurso de simulação do *Quartus II* caracterize com fidelidade o circuito criado em VHDL, o número de minutos necessário para simulação aumenta bastante na medida em que se estende o intervalo de tempo a ser analisado, limitando seu uso na análise de longos trechos de operação em regime permanente. Com uma adequada utilização dos recursos de *trigger* da ferramenta *SignalTap*, eventos específicos podem ser capturados para análise da operação do sistema em regime permanente e em tempo real.

III.4.3.4 Desempenho de decodificação

O resultado final da implementação do algoritmo de Wagner em FPGA se deu exatamente conforme planejado. Tanto na versão reelaborada no *VisSim/Comm* quanto nos experimentos realizados em bancada não se percebeu nenhuma redução de desempenho por conta dos processos de quantização e principalmente pelo esquema simplificado de cálculo de métrica.

Pôde-se inclusive constatar uma pequena melhora na taxa de erro de bit para uma mesma relação E_b/N_0 ao se comparar a curva de BER da Figura II.4, gerada com a simulação em *Mathcad* de Guimarães [Gui03], com a que foi obtida com modelo implementado no *VisSim/Comm* (Figura III.27) e posteriormente reproduzida no FPGA. Observando, por exemplo, as taxas de BER obtidas a 6,5 dB e 8,6 dB de relação E_b/N_0 na Figura III.27 (onde há uma reprodução da curva da Figura II.4), a diferença entre as

taxas de erro foi de quase 20%, de $1{,}1\times10^{-4}$ passou a $8{,}5\times10^{-5}$ e de $4{,}9\times10^{-7}$ para $3{,}9\times10^{-7}$ (um singelo ganho de +0,1 dB).

A razão para esta diferença é que os algoritmos implementados no *Mathcad* e *VisSim/Comm* diferem na forma como a decisão final pela partição é tomada. O critério adotado por Guimarães [Gui03] utiliza somente o somatório das menores métricas de cada seção em cada ramo e não se considera o efeito de uma eventual violação de paridade na composição final da métrica total. Na versão do algoritmo reelaborada no *VisSim/Comm*, antes da decisão final pela partição, a decodificação em cada ramo é completa e as métricas totais de cada ramo são calculadas com relação aos símbolos efetivamente adotados após a verificação da paridade.

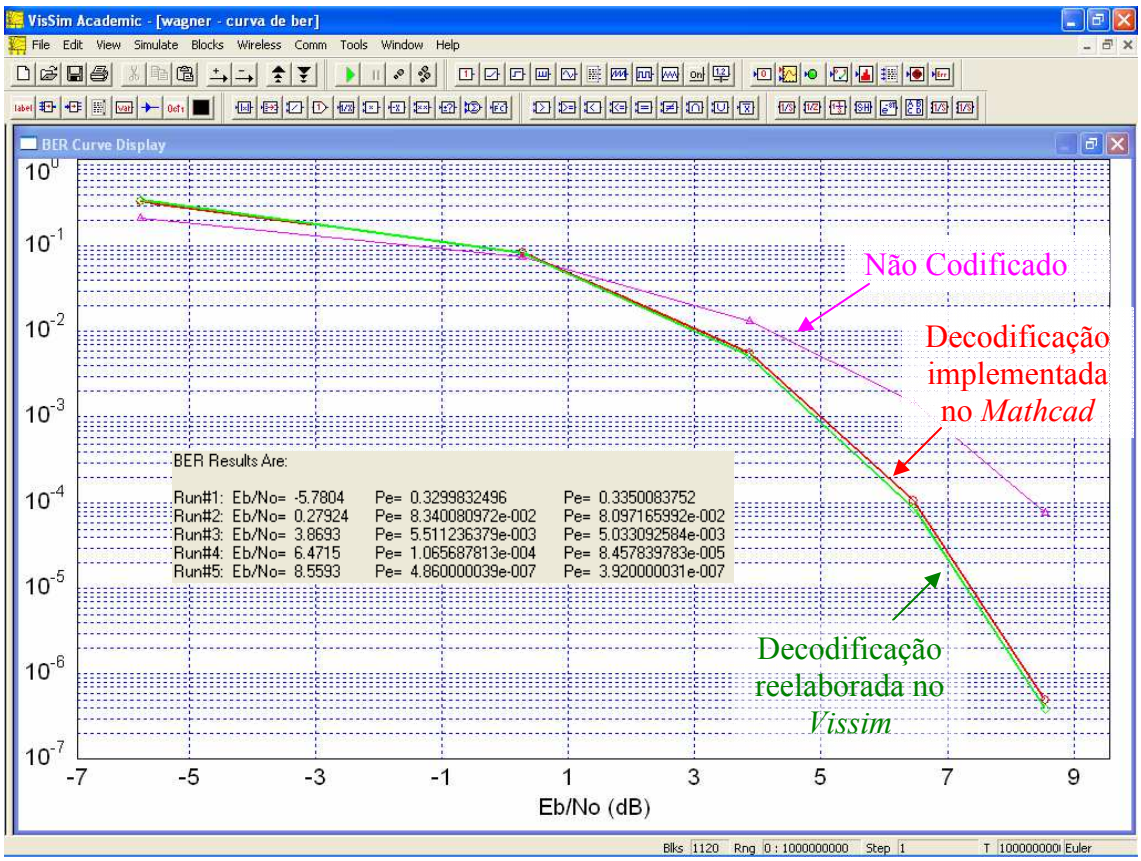


Figura III.27 *Curva de BER do circuito de decodificação do código componente (8,4,4).*

III.5 Decodificação turbo do código produto $(8,4,4)^2$

A simulação montada no *VisSim/Comm*, mostrada na Figura III.28, ilustra a concepção do sistema de codificação e decodificação turbo implementado nesta dissertação. Um feixe serial de dados codificados pelo código $(8,4,4)^2$, recebido após passar por um canal AWGN vetorial, é paralelizado em blocos de 64 símbolos, compondo a matriz recebida **R**, e aplicado à entrada do decodificador turbo. A matriz **R** é aplicada ao arranjo de decodificação iterativa e, antes que um novo bloco codificado seja recebido de forma completa, a saída decodificada em até 32 iterações completas é entregue ao circuito de serialização. O tempo total despendido com a operação é então menor do que um intervalo de 3×16 bits de mensagem, onde 1×16 são gastos na codificação, 1×16 gastos na paralelização dos dados no receptor e mais até 1×16 gastos em uma decodificação com 32 iterações.

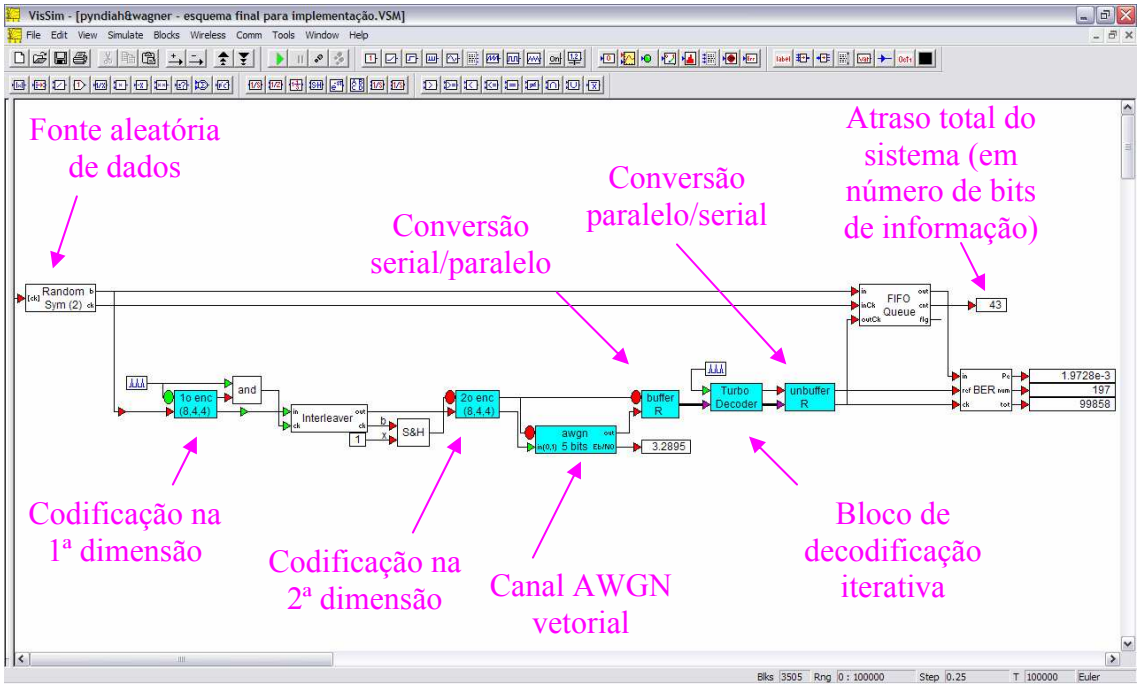


Figura III.28 *Projeto construído com auxílio do Software VisSim/Comm para decodificação do código componente $(8,4,4)^2$ usando decodificação iterativa.*

Como pode ser visto na Figura III.29, o bloco de decodificação turbo foi construído com alto grau de paralelização. Com um decodificador SISO para cada coluna da matriz **R**, o circuito é capaz de executar uma iteração parcial completa a cada pulso de *clock* do sistema. A baixa complexidade e o pequeno número de elementos lógicos utilizados na construção do decodificador de Wagner motivaram o emprego do método de "força bruta" na concepção desse esquema. Outro ponto favorável, encontrado em investigações empíricas, foi a adoção de meios de ponderação muito simples para a obtenção da informação extrínseca em cada iteração.

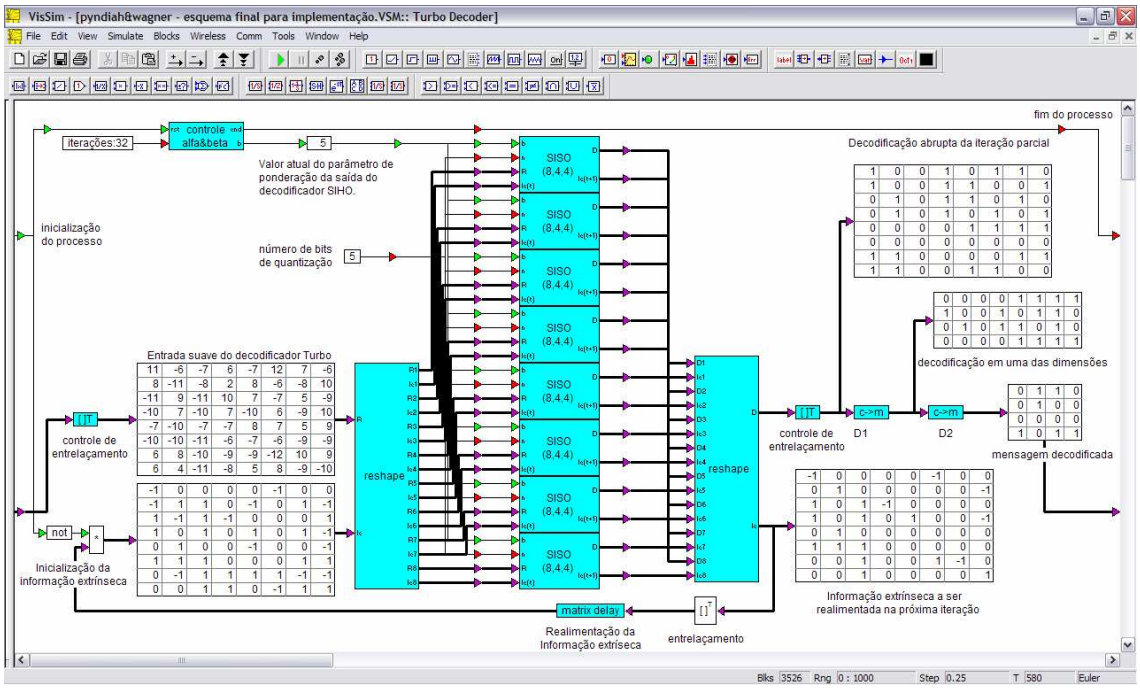


Figura III.29 *Circuito de decodificação iterativa do código $(8,4,4)^2$ implementado em FPGA.*

III.5.1 Alimentação dos decodificadores SISO

O bloco de inicialização da informação extrínseca garante que na primeira iteração os decodificadores SISO sejam alimentados apenas com a informação da matriz recebida

R, eliminando eventuais resíduos oriundos da decodificação turbo de um bloco de dados precedente.

Os blocos de controle de entrelaçamento perfazem a alternância da dimensão decodificada a cada iteração, alimentando os circuitos SISO ora pelo sentido das linhas, ora pelo sentido das colunas da matriz **R**. Este mesmo tratamento de entrelaçamento é aplicado à informação extrínseca.

Tais tarefas, se realizadas em estruturas seriais e compartilhadas, em geral necessitam de vários pulsos de *clock* para conclusão da operação, como foi o caso do entrelaçador implementado na construção do código produto (Figura III.8). Como a implementação aqui é totalmente paralelizada, o processo descrito em VHDL é na verdade um emaranhado de atribuições (Figura III.30) que ocorrem simultaneamente a cada pulso de *clock* do sistema.

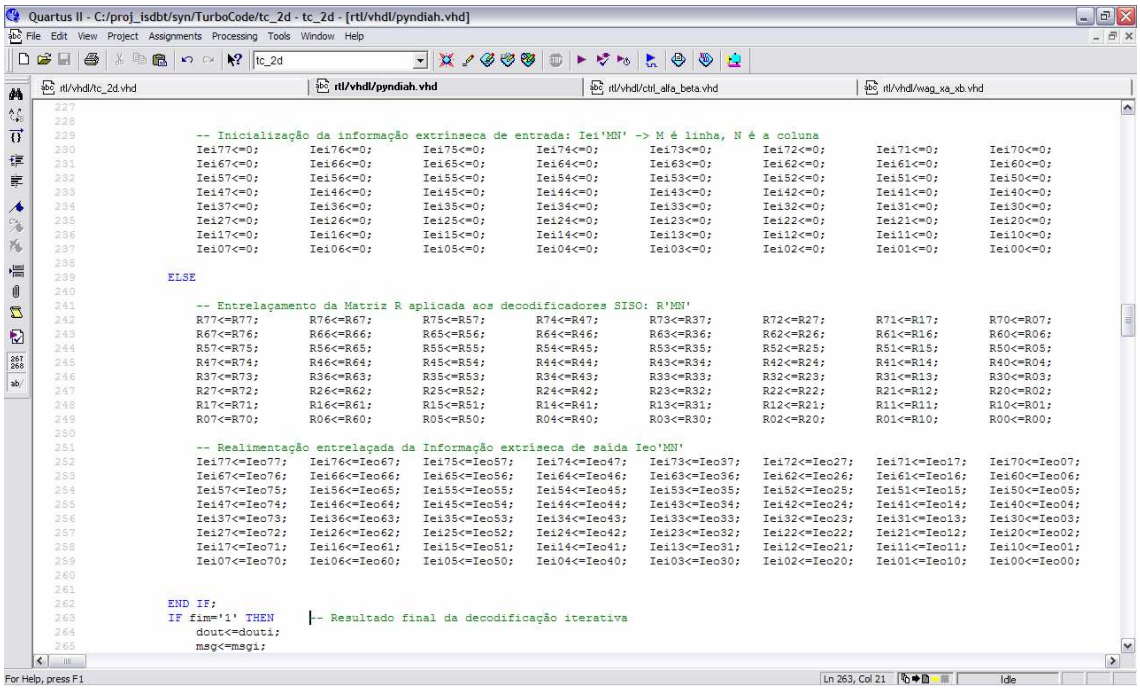


Figura III.30 *Trecho de código VHDL responsável pela inicialização da informação extrínseca e mudança da dimensão decodificada através de entrelaçamento.*

III.5.2 Construção dos decodificadores SISO

As figuras III.31 e III.32 apresentam em detalhes o esquema de decodificação SISO de uma das linhas da matriz **R** recebida, respectivamente implementados no VisSim/Comm e em VHDL. Em ambas as figuras é possível notar a presença do bloco de decodificação de Wagner.

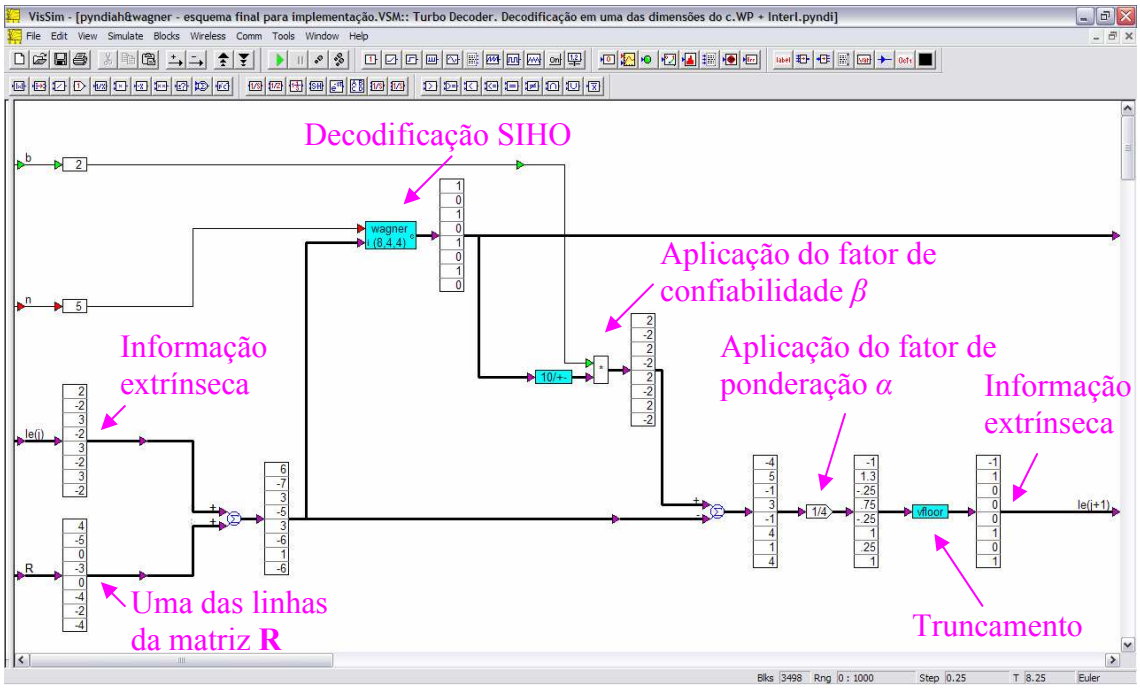


Figura III.31 *Modelo construído no VisSim/Comm para decodificação iterativa de uma das linhas da Matriz **R** utilizando a regra de Wagner sobre o código componente (8,4,4) e determinação da informação extrínseca com auxílio dos parâmetros α e β.*

No código VHDL a entrada suave do decodificador de Wagner (“wagnerp_inst”) é composta pela adição da informação extrínseca “Iei” à matriz **R** (“R”). O sinal de saída abrupta “wag_out”, obtido com a decodificação SIHO, é transformado na saída suave “xB” quando multiplicado pelo fator de confiabilidade β (sinal “beta”) no circuito denominado como “xbeta_inst”. A informação extrínseca de saída “Ieo”, que será realimentada ao decodificador SISO na próxima iteração, é finalmente composta pela

subtração das saídas suaves “xB” e do sinal aplicado na entrada do decodificador de Wagner “R”, ponderados por α (1/4).

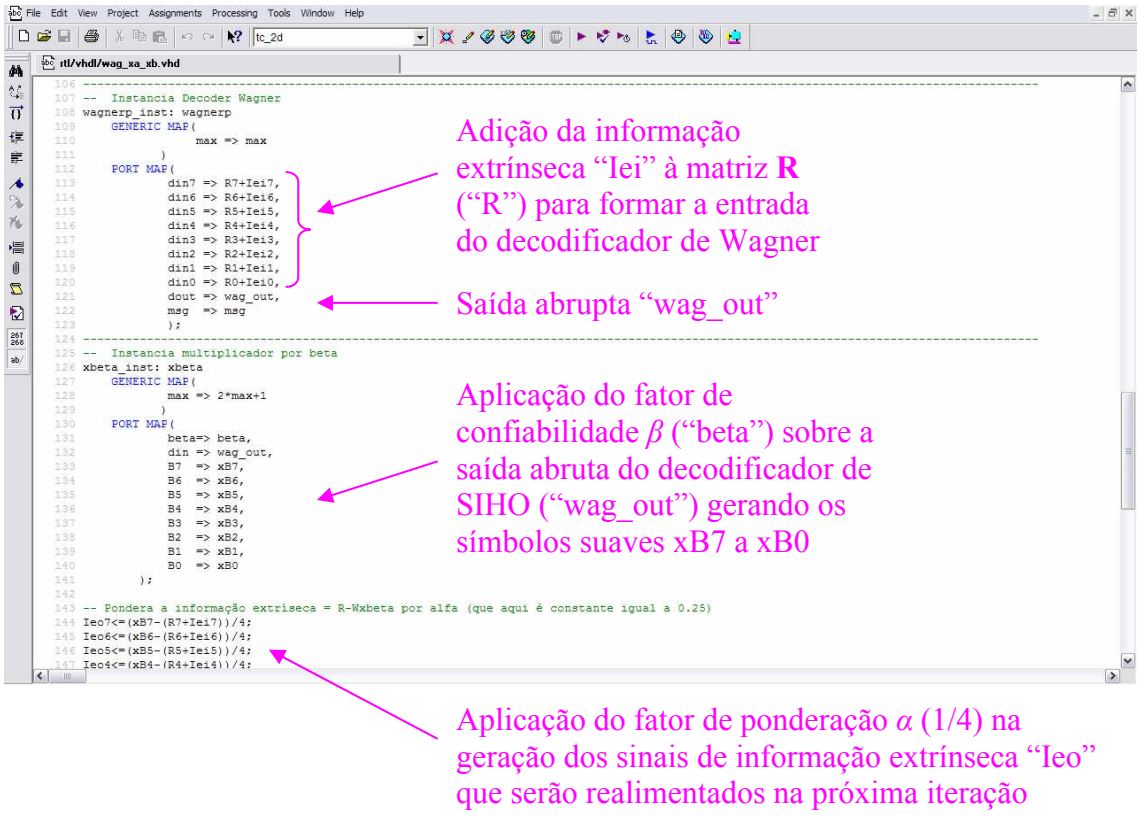


Figura III.32 *Código VHDL referente à decodificação iterativa de uma das linhas da Matriz **R** utilizando a regra de Wagner sobre o código componente (8,4,4) e determinação da informação extrínseca com auxílio dos parâmetros α e β.*

III.5.3 Geração dos fatores de ponderação

Na implementação, enquanto o parâmetro β foi truncado em 4 bits (assumindo na prática valores linearmente crescentes de 1 a 15), o fator de ponderação α, originalmente adotado com variação exponencial (equação II.8) nas simulações de Guimarães [Gui03], foi empiricamente aproximado para um valor constante igual a 1/4. Observou-se que após as primeiras iterações o valor de α rapidamente convergia para um valor constante em torno de 0,3. Após alguns testes de truncamento nos valores de α percebeu-se pouca

influência no desempenho final da decodificação, acabando por motivar a adoção de uma fração fixa, de apenas 25%, da amplitude suave obtida com aplicação do fator de confiabilidade β na formação da informação extrínseca a ser realimentada na próxima iteração. Esta aproximação resultou em uma drástica redução na quantidade de elementos lógicos relacionada à operação de multiplicação pelo parâmetro α no FPGA.

Enquanto o circuito de atribuição do fator de confiabilidade β consiste numa simples operação de mapeamento de um operador de 2 estados ($\pm 1 \times \beta$), ocupando pouca lógica, o circuito de multiplicação por fator α, truncado em 5 bits por exemplo, envolveria no mínimo um bloco com duas entradas de 5 bits e com isso seriam necessários na prática pelo menos 57 elementos lógicos do FPGA para a operação de cada símbolo da matriz **R**. Multiplicando este valor por 64 (o número de bits do bloco codificado) pode-se constatar que, a partir de uma aproximação básica, mais de 50% do FPGA estaria comprometido com esta operação, acabando por inviabilizar a proposta de paralelização. A ponderação em 1/4 do valor da informação extrínseca é obtida com uma operação de descarte de dois bits menos significativos da palavra binária e, sob o ponto de vista de síntese, este truncamento é uma simples omissão de rotas de conexão.

Obviamente o parâmetro α não é simplesmente descartável. Investigou-se aqui apenas uma simplificação específica para o código $(8,4,4)^2$ e o objetivo maior era redução do número de elementos lógicos utilizados, não o melhor desempenho de decodificação, embora o desempenho final tenha se mantido sobre a curva de BER esperada, como será mostrado adiante. De qualquer forma, pode-se constatar que os multiplicadores empregados para ponderação por α serão grandes consumidores de recursos lógicos do FPGA. Alternativas de implementação que contornem a necessidade destes são imprescindíveis.

Os trechos em VHDL abaixo (Figura III.33) ilustram os componentes responsáveis pela geração do fator de confiabilidade β e pela ponderação da saída abrupta do decodificador de Wagner.

A variação dos valores de β ao longo das iterações é parametrizada com o auxílio de duas variáveis, uma para definir o valor inicial e a outra diretamente responsável pela definição do valor final (equivalente à função de $K\beta$ na equação II.10), respectivamente os sinais "ini" e "inc" no trecho de VHDL mostrado na Figura III.33.

Um esquema de acumulação linear do valor de "inc" é utilizado para gerar os valores do sinal "betai" a cada iteração. O sinal de saída "beta" recebe então "betai" normalizado mais o valor inicial "ini". Uma terceira variável "i" é utilizada como parâmetro para alterar o número de iterações parciais realizadas pelo decodificador turbo.

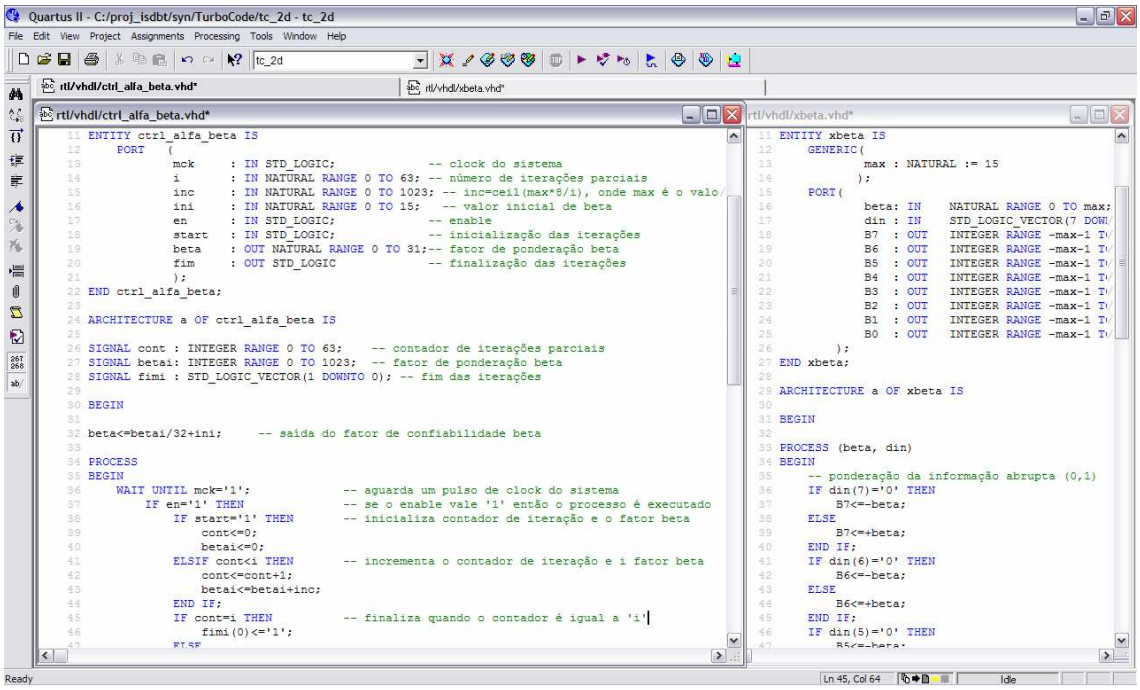


Figura III.33 *Códigos VHDL referentes à geração do fator de confiabilidade β e a sua aplicação na ponderação da saída abrupta decodificada.*

As variáveis “ini”, “inc” e “i” são armazenadas em posições de memória acessíveis via interface JTAG permitindo reconfiguração dos parâmetros do circuito em pleno funcionamento. Para alterá-las utiliza-se ferramenta *In-System Memory Editor*[7] do *Quartus II* (Figura III.34).

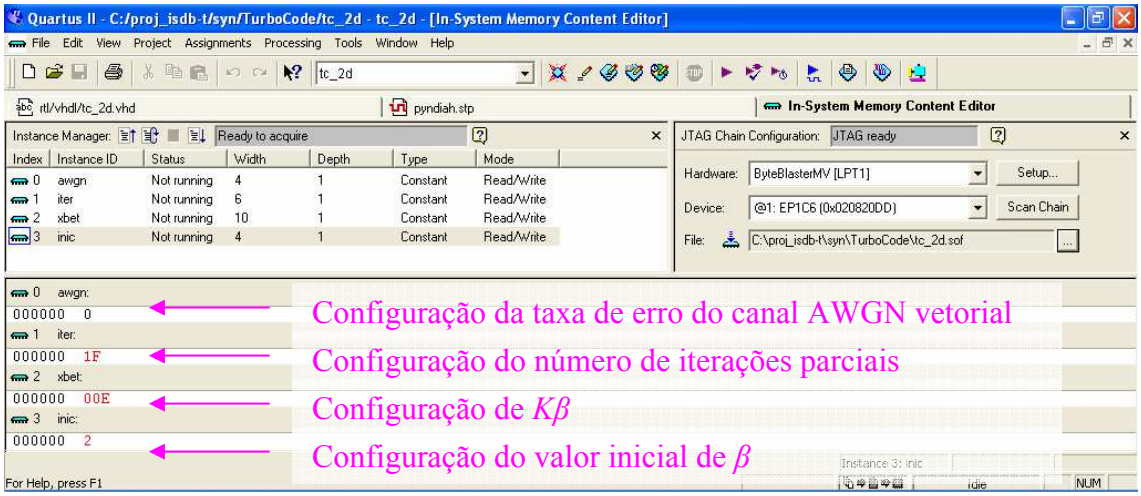


Figura III.34 *Acesso a variáveis do decodificador turbo via In-System Memory Editor (os valores das variáveis mostrados estão na base hexa).*

Essa flexibilidade de alteração do sistema em tempo real inspira buscas empíricas por combinações de parâmetros que resultem em melhores desempenhos de decodificação. Dependendo da freqüência de operação do *codec* e da taxa de erro de bit alvo da análise é possível monitorar dinamicamente a variação de desempenho e estabelecer uma relação imediata com a modificação de uma ou mais variáveis, como é o caso, por exemplo, da variação dos valores inicial e final de β e/ou do número de iterações utilizadas na decodificação turbo.

[7] Trata-se de um recurso bastante simples onde elementos de memória podem ser acessados e alterados com o circuito em operação através da interface JTAG.

III.5.4 Síntese do algoritmo de decodificação iterativa

III.5.4.1 Recursos lógicos utilizados pelo decodificador turbo

A tabela III.1 contém a hierarquia do projeto concebido na Figura III.28 e também os recursos utilizados por cada sub-bloco que o compõe. O circuito de decodificação iterativa especificamente utiliza cerca de 3500 elementos lógicos, um pouco menos de 60% da capacidade do FPGA Cyclone C6.

Tabela III.1 *Recursos do FPGA Cyclone C6 utilizados pelo projeto do codec turbo (8,4,4)*[2].

Fitter Resource Utilization by Entity

Compilation Hierarchy Node	Logic Cells	LC Registers	Memory Bits	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs	Carry Chain LCs	Packed LCs
⊟ |tc_2d	5884 (32)	2750	29450	3	0	3134 (0)	380 (0)	2370 (32)	2437 (0)	842 (1)
⊞ |encoder_2d:encoder_inst|	55 (3)	45	64	0	0	10 (0)	0 (0)	45 (3)	10 (0)	14 (0)
⊞ |awgn:awgn_inst|	213 (189)	183	202	0	0	30 (24)	121 (121)	62 (44)	50 (30)	19 (14)
⊞ |sel_awgn:sel_awgn_inst|	27 (0)	19	0	0	0	8 (0)	0 (0)	19 (0)	1 (0)	5 (0)
|buffsp:sp_inst|	653 (653)	648	0	0	0	5 (5)	140 (140)	508 (508)	6 (6)	3 (3)
⊞ |bini:bini_inst|	28 (0)	19	0	0	0	9 (0)	0 (0)	19 (0)	0 (0)	7 (0)
⊞ |iteracao:iteracao_inst|	31 (0)	23	0	0	0	8 (0)	0 (0)	23 (0)	0 (0)	5 (0)
⊞ |pondera:pondera_inst|	39 (0)	31	0	0	0	8 (0)	0 (0)	31 (0)	0 (0)	5 (0)
⊟ |pyndiah:pyndiah_inst|	3564 (652)	611	0	0	0	2953 (59)	0 (0)	611 (593)	2318 (305)	607 (0)
|ctrl_alfa_beta:ctrl_alfa_beta_inst|	26 (26)	18	0	0	0	8 (8)	0 (0)	18 (18)	22 (22)	3 (3)
⊟ |wag_xa_xb:wag_xa_xb_inst0|	746 (108)	0	0	0	0	746 (108)	0 (0)	0 (0)	584 (100)	82 (4)
⊟ |wagnerp:wagnerp_inst|	204 (14)	0	0	0	0	204 (14)	0 (0)	0 (0)	100 (8)	64 (2)
⊟ |ramo0:ramo0_inst|	95 (24)	0	0	0	0	95 (24)	0 (0)	0 (0)	46 (22)	31 (0)
|secao0011:secao0011_inst0|	9 (9)	0	0	0	0	9 (9)	0 (0)	0 (0)	6 (6)	3 (3)
|secao0011:secao0011_inst1|	8 (8)	0	0	0	0	8 (8)	0 (0)	0 (0)	6 (6)	4 (4)
|secao0011:secao0011_inst2|	9 (9)	0	0	0	0	9 (9)	0 (0)	0 (0)	6 (6)	4 (4)
|secao0011:secao0011_inst3|	8 (8)	0	0	0	0	8 (8)	0 (0)	0 (0)	6 (6)	3 (3)
|maior:maior_inst|	18 (18)	0	0	0	0	18 (18)	0 (0)	0 (0)	0 (0)	8 (8)
|decisao:decisao_inst0|	6 (6)	0	0	0	0	6 (6)	0 (0)	0 (0)	0 (0)	1 (1)
|decisao:decisao_inst1|	6 (6)	0	0	0	0	6 (6)	0 (0)	0 (0)	0 (0)	1 (1)
|decisao:decisao_inst2|	2 (2)	0	0	0	0	2 (2)	0 (0)	0 (0)	0 (0)	5 (5)
|decisao:decisao_inst3|	5 (5)	0	0	0	0	5 (5)	0 (0)	0 (0)	0 (0)	2 (2)
⊞ |ramo1:ramo1_inst|	95 (23)	0	0	0	0	95 (23)	0 (0)	0 (0)	46 (22)	31 (0)
|xbeta:xbeta_inst|	434 (434)	0	0	0	0	434 (434)	0 (0)	0 (0)	384 (384)	14 (14)
⊞ |wag_xa_xb:wag_xa_xb_inst1|	297 (101)	0	0	0	0	297 (101)	0 (0)	0 (0)	201 (101)	81 (11)
⊞ |wag_xa_xb:wag_xa_xb_inst2|	298 (101)	0	0	0	0	298 (101)	0 (0)	0 (0)	201 (101)	81 (11)
⊞ |wag_xa_xb:wag_xa_xb_inst3|	308 (101)	0	0	0	0	308 (101)	0 (0)	0 (0)	201 (101)	70 (11)
⊞ |wag_xa_xb:wag_xa_xb_inst4|	306 (101)	0	0	0	0	306 (101)	0 (0)	0 (0)	201 (101)	76 (11)
⊞ |wag_xa_xb:wag_xa_xb_inst5|	311 (102)	0	0	0	0	311 (102)	0 (0)	0 (0)	201 (101)	72 (11)
⊞ |wag_xa_xb:wag_xa_xb_inst6|	311 (102)	0	0	0	0	311 (102)	0 (0)	0 (0)	201 (101)	71 (11)
⊞ |wag_xa_xb:wag_xa_xb_inst7|	309 (101)	0	0	0	0	309 (101)	0 (0)	0 (0)	201 (101)	71 (11)
|ps:ps_inst|	17 (17)	17	0	0	0	0 (0)	0 (0)	17 (17)	0 (0)	1 (1)
⊞ |sld_hub:sld_hub_inst|	152 (33)	111	0	0	0	41 (26)	0 (0)	111 (7)	6 (0)	44 (35)
⊞ |sld_signaltap:auto_signaltap_0|	1073 (229)	1011	29184	0	0	62 (1)	119 (113)	892 (115)	46 (5)	131 (1)

For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

III.5.4.2 Freqüência máxima de operação e throughput

O tempo de decodificação do código componente (8,4,4) com o circuito combinacional implementado no Cyclone C6 traço 8 de forma otimizada ficou então em torno de 33 ns (Figura III.35), assegurando uma taxa de operação do *clock* do circuito de até aproximadamente 30 MHz.

Taxas úteis de dados de 10 Mbps foram testadas com o decodificador turbo operando com até 32 iterações completas. De fato umas das limitações do teste em bancada foram os processos de serialização e desserialização da informação dos dados provenientes do equipamento SMIQ04 da *Rohde&Schwarz* usado como medidor de BER.

Em uma condição de carga e descarga paralela de informação, taxas úteis de até 60 Mbps poderiam ser atingidas com o circuito de decodificação turbo configurado para 4 iterações. Isto é, com cada iteração parcial sendo realizada em 33 ns as 4 iterações completas levariam 8×33 ns para entregar 16 bits de mensagem decodificados, o *throughput* resultante seria de $16/(8 \times 3 \times 10^{-9}) = 60{,}6$ Mbps.

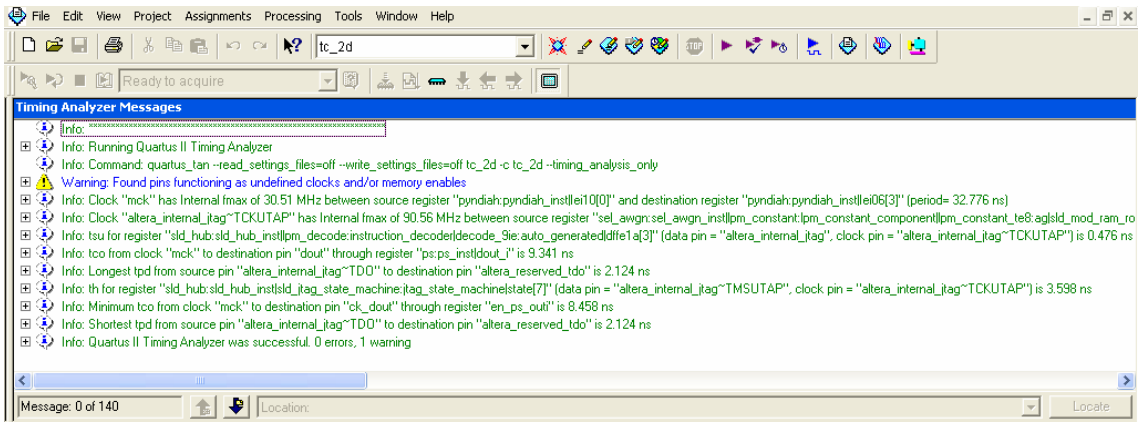


Figura III.35 *Freqüência máxima de operação estimada para o projeto do codec turbo (8,4,4)^2 no FPGA Cyclone C6 traço 8.*

III.5.4.3 Desempenho de decodificação

Na curva de BER referencial mostrada na Figura III.36 os valores de E_b/N_0 testados na simulação de Guimarães [Gui03] são os mesmos gerados pelo canal AWGN vetorial nos testes realizados no *VisSim/Comm* e em bancada (Figuras III.37 e III.38).

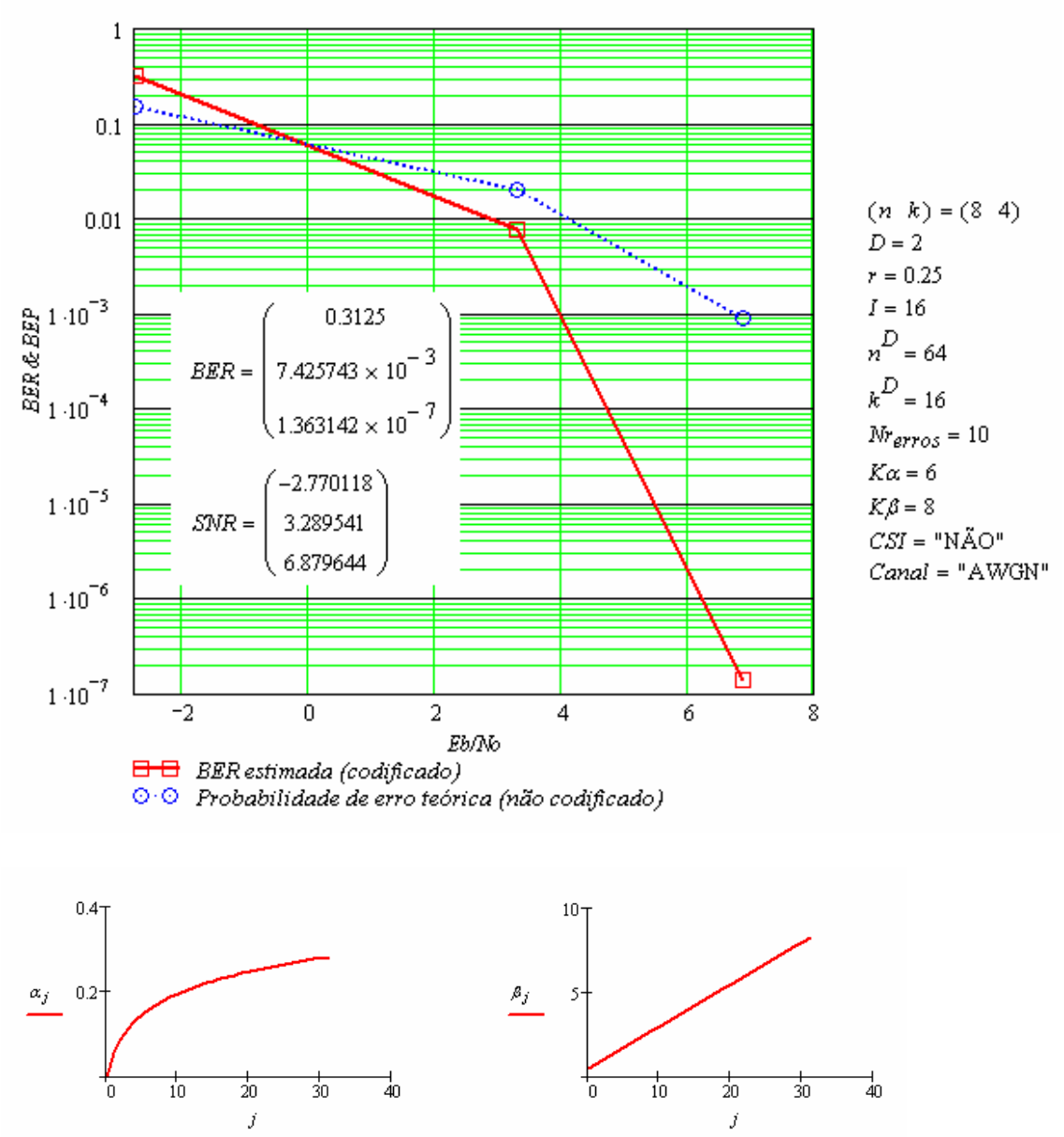
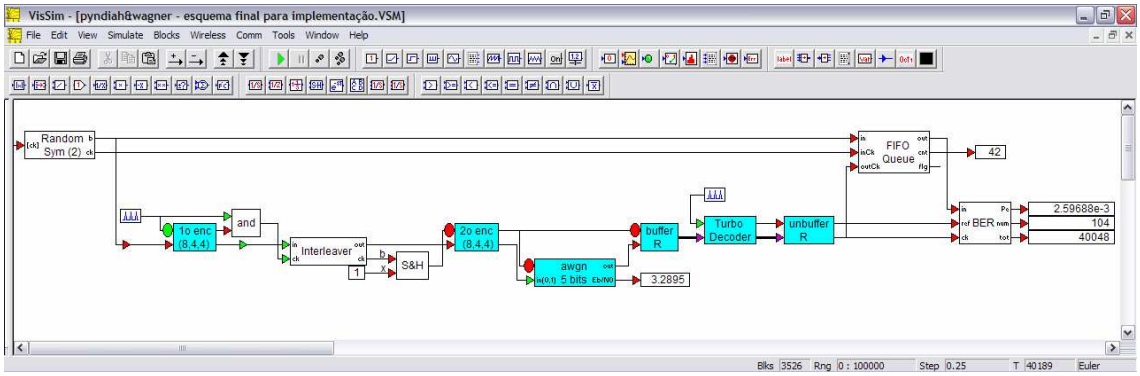
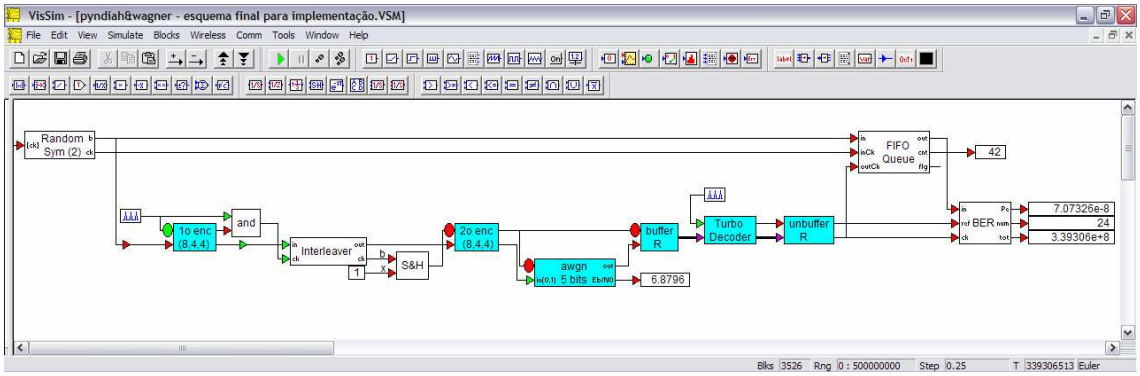


Figura III.36 *Desempenho da decodificação turbo (16 iterações) do código $(8,4,4)^2$ em canal AWGN e comportamento dos parâmetros α e β durante as j iterações parciais de decodificação turbo do código $(8,4,4)^2$.*

O desempenho final da implementação do *codec* $(8,4,4)^2$ no *VisSim/Comm* e em FPGA, usando apenas os parâmetros α e β limitados respectivamente a uma constante e a uma faixa de quantização de 4 bits, ficou compatível com o resultado simulado na proposta original de Guimarães [Gui03] usando o software *Mathcad*.



(a)



(b)

Figura III.37 *Desempenho da decodificação turbo (16 iterações) do código* $(8,4,4)^2$ *em canal AWGN vetorial com* E_b/N_0 *iguais a (a) 3,28 dB e (b) 6,88 dB.*

Enquanto o gráfico da Figura III.36 precisou de quase uma semana de simulação no *Mathcad* a simulação do mesmo número de eventos de decodificação no *VisSim/Comm* durou pouco mais de um dia. Não que os softwares tenham grandes discrepâncias em termos de desempenho, mas a própria natureza da simulação contribuiu para a diferença. Como já mencionado, cada passo de simulação no

VisSim/Comm equivale a uma iteração completa do decodificador turbo. No *Mathcad* um mesmo sub-bloco de decodificação de Wagner é utilizado para decodificação em todas as linhas e colunas da matriz **R**. Muitas instruções intermediárias de decodificação do código componente precisam ser feitas de forma seqüencial para completar uma iteração parcial, o que acaba naturalmente demandando um maior tempo de simulação.

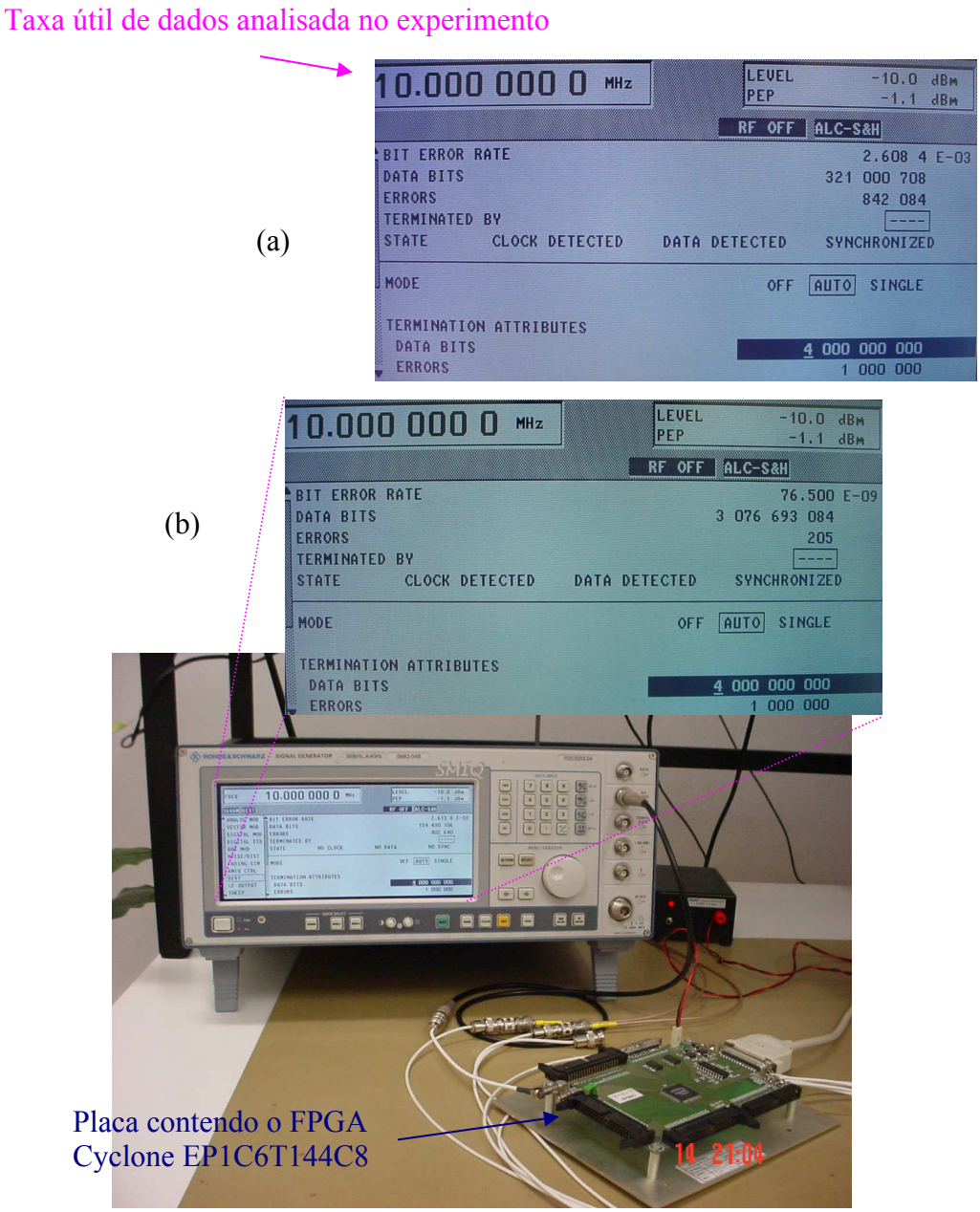


Figura III.38 *Desempenho da decodificação turbo (16 iterações) do código* $(8,4,4)^2$ *em canal AWGN: (a)* E_b/N_0 *igual a 3,28dB. (b)* E_b/N_0 *igual a 6,88 dB.*

Na implementação em *hardware* mostrada na Figura III.38 a taxa de informação avaliada foi de 10Mbps. Com alguns poucos minutos de observação foi possível constatar o desempenho de BER da ordem de 10^{-8} no decodificador turbo operando sobre um sinal recebido com E_b/N_0 de 6,9 dB.

Para facilitar os experimentos de bancada mostrados na Figura III.38, a implementação da fonte de dados aleatórios foi também incorporada no FPGA. Utilizou-se a seqüência PRBS23, cujo polinômio gerador $(x^{23} + x^{18} + 1)$ é padronizado pelo ITU para testes de BER. Como equipamento de medição e fonte de *clock* do sistema foi utilizado o SMIQ04 da *Rohde&Schwarz*.

Um outro experimento prático, motivado pela oportunidade de se utilizar o recurso de histograma do osciloscópio digital Agilent 54832B *Infiniium* (Figura III.39), registra também os efeitos de degradação obtidos com o canal AWGN vetorial implementado no FPGA. Nesse caso o gerador SMIQ04 da *Rohde&Schwarz* é utilizado como fonte de dados aleatórios e como medidor de BER e o gerador HAMEG é utilizado como fonte alternativa de *clock* para o sistema. A seqüência de bits codificada na saída do canal AWGN vetorial é exteriorizada para demonstração com o auxílio de um conversor D/A presente numa placa de testes disponibilizada pela empresa Linear.

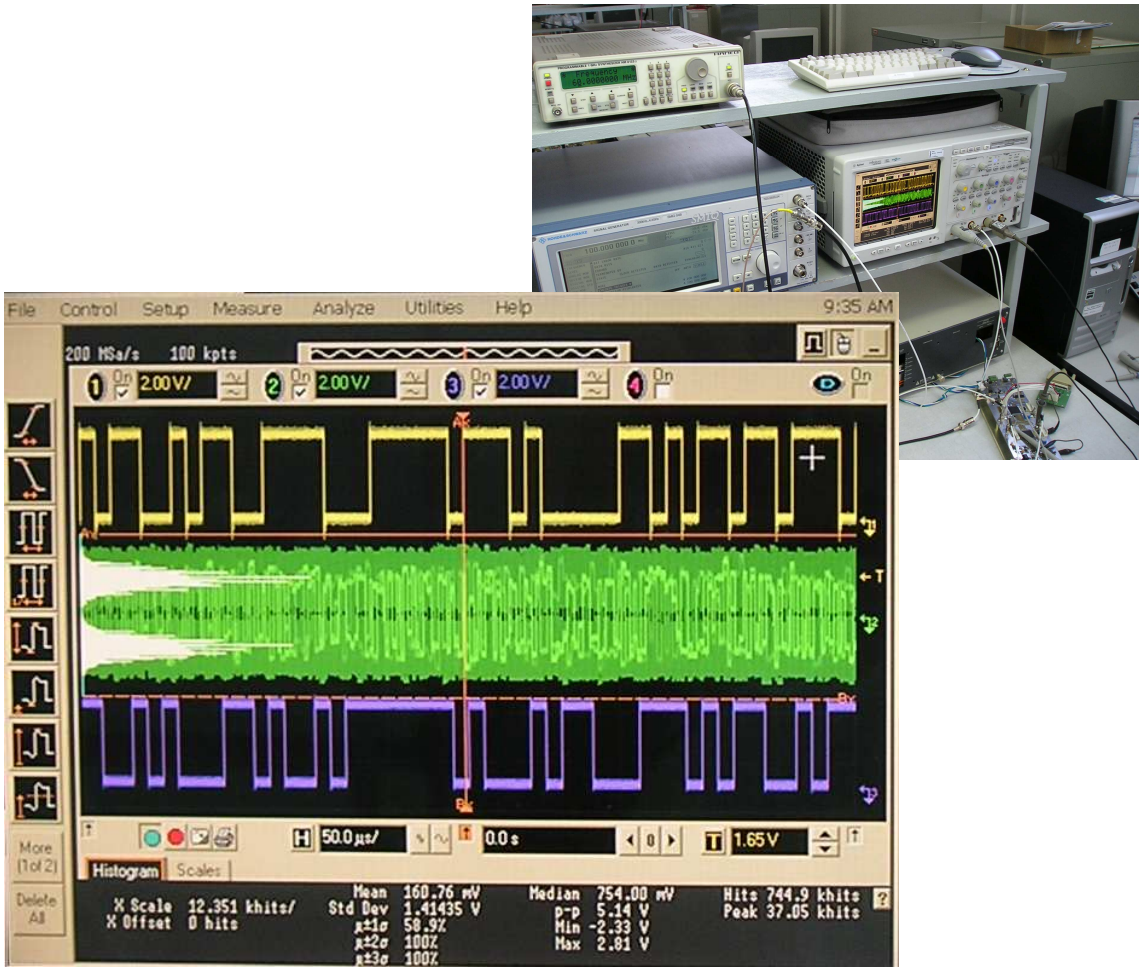


Figura III.39 *Experimento mostrando o histograma do sinal codificado na saída do canal AWGN vetorial implementado em FPGA.*

III.6 Alternativas de implementação do cálculo da confiabilidade da palavra decodificada sem o uso dos parâmetros α e β

O trabalho de mestrado de Le [Le05] introduz uma abordagem alternativa de cálculo de confiabilidade da palavra decodificada que não requer a tradicional procura por palavras concorrentes e/ou a utilização dos fatores de ponderação α e β. Baseando-se em uma propriedade da distância euclidiana da palavra decodificada, Le apresenta um esquema de cálculo de confiabilidade cuja implementação é de baixa complexidade, facilmente

construída com auxílio de LUTs. A propriedade explorada é denominada distância euclidiana destrutiva ($Dist_{des}$) e é definida como sendo o somatório das j diferenças quadráticas de amplitude medidas entre os símbolos $\pm r_j$ do vetor recebido **r** e os respectivos valores $\pm d_j$ do vetor decodificado **d**, quando $d_j > r_j$ somente.

$$Dist_{des} = \sum_{j \in DES} (r_j - d_j)^2 \qquad onde \qquad DES = \{ j \mid d_j > r_j \} \tag{III.9}$$

Essa medida é didaticamente ilustrada na Figura III.40 (adaptada de Le [Le05]), onde **x** representa a seqüência transmitida, **r** a palavra recebida na saída de um canal AWGN vetorial e **d** o resultado da decodificação.

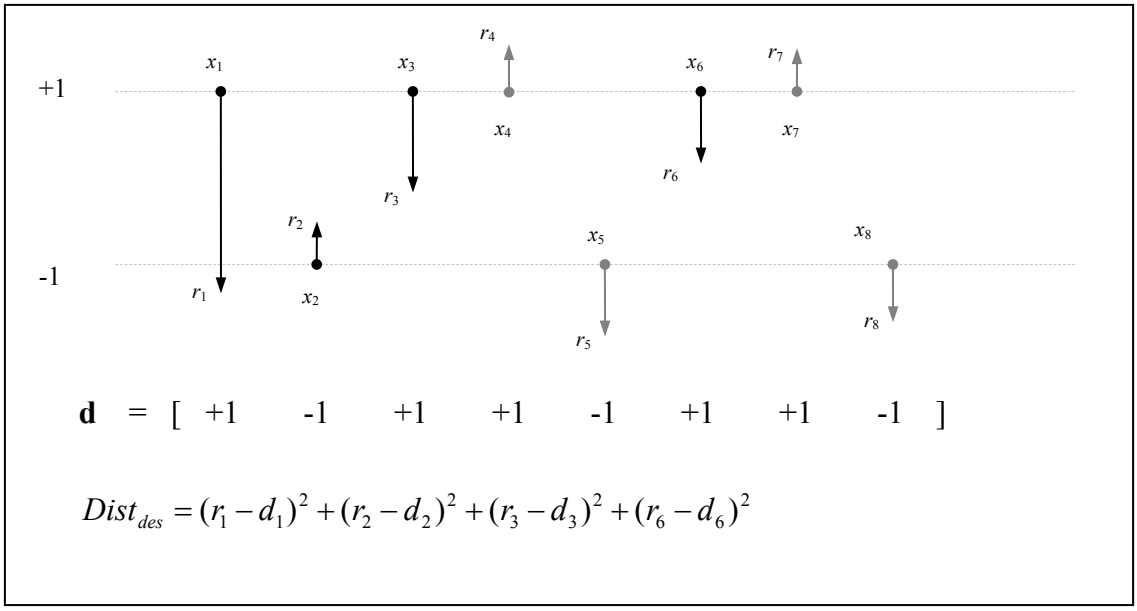


Figura III.40 *Distância euclidiana destrutiva ($Dist_{des}$).*

O objetivo almejado por Le [Le05] ao considerar apenas as distâncias euclidianas onde $d_j > r_j$ é remover os efeitos colaterais da adição de informação extrínseca de iterações passadas no cálculo da distância euclidiana de uma nova iteração. Ou seja, procura-se isolar apenas o efeito do ruído interferente com polaridade

contrária à do sinal transmitido, ruído aquele que acaba por reduzir o módulo original do sinal transmitido para um valor abaixo da média esperada, podendo inclusive provocar inversão da sua polaridade.

Na abordagem de Le [Le05] a relação entre o valor de confiabilidade e a distância euclidiana destrutiva é estabelecida estatisticamente através de simulação computando-se, em 10.000 amostras de resultados da decodificação de um código BTC(n,k,d)^2, as proporções de decodificações realizadas com sucesso associadas às suas respectivas condições de distância euclidiana destrutiva. Os resultados médios, obtidos para algumas condições de E_b/N_0 e algumas configurações do decodificador, são finalmente resumidos por Le [Le05] em uma tabela (Tabela III.2).

Tabela III.2 *Confiabilidade Φ versus distância euclidiana destrutiva $Dist_{des}$ obtida a partir de 10000 amostras de decodificação do código BTC(64,51,6)^2.*

$Dist_{des}$	<9	9	10	11	12	13	14	>14
$\phi(Dist_{des})$	0.99	0.93	0.9	0.82	0.65	0.42	0.21	0

A partir dos valores tabelados de confiabilidade $\phi(Dist_{des})$, dos valores da entrada suave r_j, dos elementos d_j da palavra decodificada e da potência do ruído no canal $\sigma^2/2$, a saída suave w_j do decodificador é então calculada pela expressão a seguir [Le05, eq. 3.21].

$$w_j = d_j \left(\frac{\sigma^2}{2} \ln \left(\frac{\phi + \exp(2 r_j d_j / \sigma^2)}{1 - \phi} \right) - r_j d_j \right) \qquad \text{(III.10)}$$

A proposta de cálculo de confiabilidade de Le [Le05] é mais apropriada do que a simples adoção de fatores de ponderação empíricos e poderiam apresentar interessantes resultados em trabalhos futuros. A possibilidade de utilização de LUTs aliada ao benefício de se determinar mais precisamente a confiabilidade da decodificação constitui uma solução viável de contorno às simplificações feitas aos fatores α e β nessa implementação por conta da necessidade de simplificação dos circuitos. Esse método apresenta ainda, segundo Le [Le05], uma melhoria de desempenho em relação ao método original de Pyndiah [Pyn98] se aplicada a códigos com grandes distâncias de Hamming, como é o caso da classe de códigos investigada nesta dissertação.

Capítulo IV

Conclusão

NESTA dissertação descreveu-se a implementação de um esquema de decodificação turbo de bloco proposto para o código produto $(8,4,4)^2$. O decodificador implementado pode proporcionar taxas úteis de mais de 60 Mbps, utilizando 4 iterações, e um atraso menor do que um intervalo equivalente a três blocos de informação codificada (basicamente o tempo gasto com o processo de serialização e desserialização). Utilizou-se neste projeto cerca de 60% da lógica disponível no FPGA Cyclone EP1C6T144C8. O desempenho mostrou-se compatível àquele obtido por Guimarães [Gui03] com simulação computacional, onde não se considerou os efeitos de quantização. O ganho de codificação para valores de BER de 1×10^{-5} foi da ordem de 4,5 dB, o mesmo registrado na simulação originalmente elaborada por Guimarães [Gui03] com 16 iterações.

Com relação à construção final, o produto implementado se manteve fiel às expectativas de possuir baixa complexidade. Como contribuição para uma simplificação ainda maior do esquema de decodificação, várias medidas foram investigadas e

incorporadas à construção do sistema originalmente proposto por Guimarães [Gui03]. As contribuições registradas abaixo foram ainda validadas com os experimentos realizados sem promover nenhuma perda significativa de desempenho.

- Reconstrução da proposta do *codec* $(8,4,4)^2$ no *VisSim/Comm*, disponibilizando um conjunto de simulações alternativas às originalmente construídas por Guimarães [Gui03] em *Mathcad*, com o benefício de oferecer uma interface mais condizente com uma implementação em *hardware*;
- Elaboração de um arranjo de decodificação totalmente combinacional;
- Truncamento do sinal suave de entrada do decodificador SIHO em 5 bits;
- Cálculo simplificado das métricas dos símbolos recebidos, contornando a necessidade de potenciação e estimação de valor médio;
- Implementação de um esquema de decodificação baseado na decodificação completa de cada ramo do código (8,4,4) segundo a regra de Wagner, resultando um uma decodificação um pouco mais eficiente do que a originalmente adotada por Guimarães [Gui03];
- Truncamento da constante de confiabilidade β em uma faixa de valores de 1 a 15, quantizada em 4 bits;
- Adoção de um fator de ponderação α com valor constante de 1/4;

Como subproduto também se desenvolveu neste trabalho um canal AWGN vetorial com princípio de operação simples para avaliar o desempenho do esquema de decodificação proposto para código produto $(8,4,4)^2$.

A metodologia de projeto adotada nesta dissertação, empregando o *VisSim/Comm* como etapa intermediária para reelaboração e validação de estruturas

originalmente construídas no *Mathcad*, foi bem sucedida. Na opinião do autor, a conversão das simulações em *Mathcad* de Guimarães [Gui03] diretamente para VHDL seria uma tarefa bastante difícil, uma vez que o nível de abstração permitido pelo *Mathcad* está muito acima da capacidade de descrição em VHDL. A semelhança com o nível de construção esquemática do *VisSim/Comm*, a capacidade de abstração na criação de blocos hierárquicos parametrizáveis e a transparência da depuração com o uso de avançados recursos de análise foram essenciais para a elaboração de códigos VHDL comportamentais simples e consistentes com as propostas originais de Guimarães [Gui03]. Os resultados simulados puderam ser reproduzidos com grande fidelidade nos experimentos realizados em *hardware*.

Outras ferramentas de projetos em FPGA utilizando o *Simulink* do *Matlab* já há algum tempo são oferecidas pela Altera e pela Xilinx. São elas, respectivamente, os softwares *DSP Builder* e *System Generato*. Em ambos os casos, blocos especialmente desenvolvidos para a interface do *Simulink*, como portas lógicas, registradores, pinos de I/O e *MegaCores* (IPs), são disponibilizados em uma biblioteca de projeto permitindo a construção de um diagrama funcional que pode ser automaticamente convertido em código VHDL estrutural. Uma grande vantagem destas propostas é a possibilidade de utilização de qualquer recurso do *Matlab* na geração de estímulos e na análise de resultados de simulações além, é claro, do fato de contornar a necessidade do pesquisador ter que desenvolver habilidades específicas em VHDL. Por outro lado, o código VHDL estrutural não é facilmente interpretável, não pode ser avaliado ou reeditado, como ocorre com o VHDL comportamental e, por ser gerado a partir de uma descrição gráfica, é mais limitado em termos de parametrização. O que se pôde perceber na investigação de tais ferramentas é que, de uma forma geral, elas ainda estão em processo de amadurecimento. Nem todos os recursos de compilação e síntese estão

disponíveis na interface do *Simulink* e ainda há a necessidade de alguma experiência prévia em VHDL para a construção de estruturas condizentes com a forma de funcionamento de um processo seqüencial.

IV.1 Propostas para investigações futuras

- Procura por uma arquitetura de decodificação que resulte em maiores *throughputs*: os resultados de tempo de propagação obtidos na ordem de 30 ns podem ser reduzidos para valores tipicamente em torno de 10 ns com simples emprego de alguns *pipelines* em pontos estratégicos da decodificação.
- Melhorias na implementação do algoritmo de decodificação: o ponto chave de melhoria do processo está relacionado a se encontrar um meio de efetuar de maneira mais exata o cálculo de confiabilidade da palavra decodificada sem o uso de fatores de ponderação puramente empíricos. Nesse propósito a breve abordagem ao trabalho de Le [Le05] no Capítulo III pode servir como proposta de uma nova direção de pesquisa, com a adoção de cálculos exatos de confiabilidade utilizando LUTs.
- Avaliação da possibilidade de construção de um esquema de decodificação analógico baseado no arranjo combinacional proposto nesta dissertação. Neste projeto os circuitos de serialização digital poderiam ser substituídos por uma sucessão de blocos de amostragem e retenção (*Sample and Hold*). As medidas das métricas e os processos de decisão seriam livres de quantização e a velocidade de processamento seria limitada unicamente ao *slew rate* do circuito analógico.

- Reescrever os códigos VHDL para torná-los mais genéricos, permitindo mudar parâmetros como o comprimento *n* da palavra-código.
- Implementação de um código $(n,n/2,4)^3$, utilizando talvez o *Simulink* do *Matlab* e as ferramentas *DSP Builder* da Altera ou o *System Generator* da Xilinx.

Referências Bibliográficas

[Aha06] www.aha.com (último acesso em abril 2006).

[Alt05] www.altera.com/support/examples/dsp-builder/exm-viterbi-ber.html (último acesso em abril de 2006).

[Bar96] BARBULESCU, S. A. **Iterative Decoding of Turbo Codes and Other Concatenated Codes**, Ph.D. Thesis, Faculty of Engineering, University of South Australia, Fevereiro/1996.

[Bar98] BARBULESCU, S. A. e S. S. PIETROBON **Turbo Codes: a Tutorial on a New Class of Powerful Error Correcting Coding Schemes. Part I: Code Structures and Interleaver Design**, Institute for Telecommunications Research, University of South Australia, Outubro/1998.

[Bat89] BATTAIL, G. **Coding for the Gaussian Channel: the Promise of Weighted-Output Decoding**, Int'l Journal of Satellite Communications, vol. 7, pp. 183-92, 1989.

[Ben96] BENEDETTO, S. e MONTORSI, G. **Unveiling turbo codes: Some results on parallel concatenated coding schemes**, IEEE Trans. Inform. Theory, vol. 42, no. 2, pp. 409-428, Março/1996.

[Ber93] BERROU, C. e A. GLAVIEUX e P. THITIMAJSHIMA **Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes**, Proceedings of the 1993 Communication Conference, ICC'93, Genebra, Suiça, pp. 1064-1070, Maio/1993.

[Ber96] BERROU, C. e A. GLAVIEUX. **Near Optimum Error Correcting Coding And Decoding: Turbo-Codes**, IEEE Transactions on Communications, pp. 1261-1271, Vol. 44, n° 10. Outubro/1996.

[Ber03] BERROU, C. **The Ten-Year-Old Turbo Codes are Entering into Service**, IEEE Commun. Mag., pp. 110-116, Agosto/2003.

[Bos00] BOSSERT, M. **Channel-Coding for Telecommunications**. John Wiley & Sons, Ltd., England, Julho/2000.

[Bou02] BOUTROS J. et al. **Turbo Code at 0.03 dB from Capacity Limit**, Proc. of Int'l Symposium on Information Theory, p. 56, Julho/2002.

[Box58] BOX, G. E. P. e MULLER, M. E. **A Note on the Generation of Random Normal Deviates**. Ann. Math. Stat. 29, 610-611, 1958.

[Cha72] CHASE, D., **A Class of Algorithms for Decoding Block Codes With Channel Measurement Information**. IEEE Transactions on Information Theory, Vol. IT- 18, No. 1, pp. 170-182, Janeiro/1972.

[Chu01] CHUNG, S.Y. et al. **On the Design of Low-Density Parity-Check Codes Within 0.0045 dB of the Shannon Limit**, IEEE Commun. Letters, vol. 5, no. 2, pp. 58-60, Fevereiro/2001.

[Dav01] DAVE, S., KIM, J. e KWATRA, S. C. **An Efficient Decoding Algorithm for Block Turbo Codes**. IEEE Transactions on Communications, pp. 41-46, Vol. 49, n° 1, Janeiro/2001.

[For66] FORNEY, G. D. Jr., **Concatened Codes**, Cambridge, MA: MIT. Press, 1966.

[Gal62] GALLAGER, R. G. **Low-Density Parity-Check Codes**, IRE Trans. Inform. Theory, vol. IT-8, pp. 21-28, Janeiro/1962.

[Gal63] GALLAGER, R. G., **Low density parity check codes**. Number 21 in Research monograph series. MIT Press, Cambridge, Mass., 1963.

[Gal68] GALLAGER, R. G. **Information Theory and Reliable Communication**, John Wiley & Sons, 1968.

[Gas06] GASPAR, I. S. e GUIMARÃES, D. A., **Implementação em FPGA de uma Classe de Códigos Produto com Decodificação Turbo**, WCCSETE'2006 – World Congress on Computer Science, Engineering and Technology Education, Itanhaém / Santos, SP, Brasil, Março/2006.

[Gui03] GUIMARÃES, D. A. **Uma Classe de Códigos Produto e sua Decodicação Turbo Aplicada em um Sistema CDFMA Multiportadora**, Tese de Doutorado: Universidade Estadual de Campinas-Unicamp, Campinas, SP, Junho/2003.

[Hag89] HAGENAUER, J. e P. HOEHER **A Viterbi Algorithm with Soft-Decision Outputs and its Applications**, Proc. of Globecom '89, Dallas, Texas, Novembro/1989.

[Hag96] HAGENAUER, J., OFFER, E. e PAPKE, L. **Iterative Decoding of Binary Block and Convolutional Codes**. IEEE Transactions on Information Theory, pp. 429-445, Vol. 42 N° 2, Março/1996.

[Ham50] HAMMING, R. **Error Detecting and Error Correcting Codes**, Bell System Technical Journal, vol. 29, pp.147-160, 1950.

[Hun98a] HUNT, A., **Hyper-codes: High-performance Low-Complexity Error-Correcting Codes**. M. Sc. Thesis, Faculty of Engineering, Ottawa-Carleton Institute of Electrical Engineering, Carleton University, Ottawa, Ontario, Canada, Maio/1998.

[Hun98b] HUNT, A., CROZIER, S. e FALCONER, D. **Hyper-codes: High-performance Low-Complexity Error-Correcting Codes**. Proceedings of the 19th Biennial Symposium on Communications, pp. 263-267, Kingston, Canada, Junho/1998.

[Itu00] www.itu.int/imt , 2000. (último acesso em julho de 2006).

[Koo97] KOORAPATY, H. e WANG, Y. E. e BALACHANDRAN, K. **Performance of Turbo Codes with Short Frame Sizes**, in Proc., IEEE Veh. Tech. Conf., pp. 329-333, Phoenix, 1997.

[Le05] LE, N. **A New Distance-Based Algorithm for Block Turbo Codes from Concept to Implementation** Tese de Mestrado em Ciências Aplicadas, Concordia University Montreal, Quebec, Canada, 2005.

[Lee94] LEE, E. A. e MESSERSCHMITT, D. G. **Digital Communication**, 2nd Edition - Kluwer Academic Publishers, 1994.

[Mac96] MACKAY, D. J. C. e NEAL, R.M. **Near Shannon Limit Performance of Low Density Parity Check Codes**, IEE Electronics Letters, vol. 32, no. 18, pp. 1645-1655, 29th Aug.1996.

[Nic97a] NICKL, H., HAGENAUER, J. e BURKET, F. **Approaching Shannon's Capacity Limit by 0.27 dB Using Hamming Codes in a 'Turbo'-Decoding Scheme**, In Proceedings of the 1997 International Symposium on Information Theory, Germany, Julho/1997.

[Nic97b] NICKL, H., HAGENAUER, J. e BURKET, F. **Approaching Shannon's Capacity Limit by 0.2 dB Using Simple Hamming Codes**, IEEE Communications Letters, Vol. 1 5, pp. 130-132, Setembro/1997.

[Pro95] PROAKIS, J. G. **Digital Communications**, 3rd Edition - McGraw Hill. New York, 1995.

[Pyn94] PYNDIAH, R., GLAVIEUX, A., PICART, A. e JACQ, S. **Near Optimum Decoding of Product Codes**, Proc. of IEEE GLOBECOM '94 Conference, San Francisco, vol. 1/3, pp. 339-343 , Dezembro/1994.

[Pyn98] PYNDIAH, R. M. **Near-Optimum Decoding of Product Codes: Block Turbo Codes.** IEEE Transactions on Communication, pp. 1003-1010, Vol. 46, n° 8. Agosto/1998.

[Pyn99] PYNDIAH, R. M. e A. PICART. **Adapted Iterative Decoding of Product Codes**, pp. 2357-2362, Global Telecommunications Conference – Globecom'99.

[Ran98] RANKIN, D. M. e GULLIVER, T. A. **Single Parity Check Product Codes**, IEEE Transactions on Communications, Vol. 49, No. 8, pp. 1354-1362, Agosto/1998.

[Ran01] RANKIN, D. M. **Single Parity Check Product Codes and Iterative Decoding**. Ph.D. Thesis, University of Canterbury, Christchurch, New Zealand, Maio/2001.

[Sha48] SHANNON, C. E. **A Mathematical Theory of Communication**, reprinted with corrections from The Bell System Technical Journal, Vol. 27, Outubro/1948.

[Sin00] SINGH, Manjit e SINGH, Manoneet **3G WirelessWith Respect to IMT-2000 and Beyond**", ITU documentation on-line (www.itu.int), Fevereiro/2000.

[Sil54] SILVERMAN, R., A. e BALSER, M. **Coding for Constant-Data-Rate Systems**, IRE Transactions on Information Theory, PGIT-4, pp. 50-63, 1954.

[Skl97] SKLAR, B. **A Primer on Turbo Code Concepts**. IEEE Communication Magazine, pp.94-101, Dezembro/1997.

[Skl88] SKLAR, B. **Digital Communications Fundamentals and Aplications,** PTR Prentice Hall, Upper Saddle River, New Jersey 07458, pp. 357-365 1988.

[Sou96] SOUROUR, E., e NAKAGAWA, M. **Performance of Orthogonal Multicarrier CDMA in a Multipath Fading Channel**, IEEE Transactions on Communications, Vol. 44, No. 3, pp. 356-367, Março/1996.

[Tan81] TANNER, R. M. **A Recursive Approach to Low Complexity Codes**, IEEE Trans. Inform. Theory, vol. IT-27, pp. 533-47, Setembro/1981.

[Tur06] www.turboconcept.com (último acesso em abril 2006).

[Wic95] WICKER, S. B. **Error Control Systems for Digital Communication and Storage**, Prentice Hall. New Jersey, 1995.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo