UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
E INFORMÁTICA INDUSTRIAL

BRUNO UMBRIA PEDRONI

IMPLEMENTAÇÃO EM HARDWARE DE UM DECODIFICADOR DE VITERBI BASEADO NAS TRELIÇAS CONVENCIONAL, PUNCIONADA E MÍNIMA

DISSERTAÇÃO DE MESTRADO

CURITIBA
2009

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

BRUNO UMBRIA PEDRONI

IMPLEMENTAÇÃO EM HARDWARE DE UM DECODIFICADOR DE VITERBI BASEADO NAS TRELIÇAS CONVENCIONAL, PUNCIONADA E MÍNIMA

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de “Mestre em Ciências” – Área de Concentração: Telemática.

Orientador: Prof. Dr. Richard Demo Souza
Co-Orientador: Prof. Dr. Volnei Antonio Pedroni

CURITIBA
2009

Ficha catalográfica elaborada pela Biblioteca da UTFPR – Campus Curitiba

P372i Pedroni, Bruno Umbria
Implementação em hardware de um decodificador de Viterbi baseado nas treliças convencional, puncionada e mínima / Bruno Umbria Pedroni. – 2009.
105 f. : il. ; 30 cm

Orientador: Richard Demo Souza
Co-orientador: Volnei Antonio Pedroni
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. Área de Concentração: Telemática, 2009
Bibliografia: f. 103-5

1. Viterbi, decodificadores de. 2. Dispositivos lógicos programáveis. 3. Decodificadores (Eletrônica). 4. Telecomunicações. 5. Telemática. I. Souza, Richard Demo, orient. II. Pedroni, Volnei A. (Volnei Antonio), co-orient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. Área de Concentração: Telemática. IV. Título.

CDD 621.3

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Campus Curitiba

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Título da Dissertação Nº 510:

"Implementação em Hardware de Um Decodificador de Viterbi Baseado nas Treliças Convencional, Puncionada e Mínima"

por

Bruno Umbria Pedroni

Esta dissertação foi apresentada, às 09h do dia 23 de outubro de 2009, como requisito parcial para a obtenção de grau de MESTRE EM CIÊNCIAS – Área de Concentração: Telemática, pelo Programa de Pós Graduação em Engenharia Elétrica e Informática Industrial – CPGEI –, na Universidade Tecnológica Federal do Paraná – UTFPR. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:

Prof. Dr. Richard Demo Souza
(Orientador - UTFPR)

Prof. Dr. Fernando Cardoso Castaldo
(UTFPR)

Prof. Dr. Cecílio José Lins Pimentel
(UFPE)

Visto da coordenação:

Prof. Dr. Humberto Remígio Gamba
(Coordenador do CPGEI)

Av. Sete de Setembro, 3165 - 80230-901 - Curitiba – PR.
Tel. +55 (41) 3310 4680 / e-mail: cpgei-ct@utfpr.edu.br / http://www.cpgei.ct.utfpr.edu.br

Ao Mestre na minha vida, cujo raciocínio e inteligência transcendem a lógica; cuja prontidão e dedicação transcendem a paternidade; e cuja aceitação e satisfação pelos momentos da vida, desapercebidas inclusive por ele mesmo, me inspiram em respirar fundo e ver o todo.

Minha admiração e metade do meu sangue serão sempre seus.

AGRADECIMENTOS

À Universidade Tecnológica Federal do Paraná – UTFPR, por me dar a oportunidade de desfrutar do seu excelente ensino, tanto em termos da qualificação dos professores, quanto em infra-estrutura, que propiciam uma educação completa.

Ao Richard Demo Souza, o qual foi o orientador preciso para meus estudos, apoiando-me em uma base quase diária, mostrando sua eterna disposição e interesse e dando seu voto de confiança em mim. Além de professor e orientador, tornou-se um grande amigo meu ao longo destes últimos anos.

Ao Volnei Antonio Pedroni, por ter co-orientado o meu trabalho e por ser meu pai e professor nestes últimos 27 anos.

Ao Cecílio Pimentel e ao Fernando Castaldo, por mostrarem interesse no meu trabalho e participarem da banca de defesa.

À Companhia Paranaense de Energia – COPEL, principalmente nas pessoas de José Maria Tiepolo, Gilmar Alfredo Ribas e Cley José Vaz Nascimento, pelo incentivo na realização deste estudo.

À Cláudia, Patrícia e Ricardo, por compartilharem comigo o mundo acadêmico e me apoiarem na minha decisão de segui-lo.

À minha namorada e futura esposa, Luciana, por mostrar compreensão em todos os momentos em que os livros e o computador venceram, participando avidamente dos meus estudos e de todo preparo final. Consegui concluir com êxito e manter o foco por causa do seu apoio e dos nossos planos. Obrigado, querida.

Aos amigos e colegas, que mostraram interesse durante o processo.

RESUMO

PEDRONI, Bruno Umbria. Implementação em hardware de um decodificador de Viterbi baseado nas treliças convencional, puncionada e mínima. 2009. 105 f. Dissertação – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná. Curitiba, 2009.

O decodificador de Viterbi (VD) é amplamente utilizado nos sistemas atuais de comunicação, principalmente os sem fio, onde o consumo de energia é crítico. O VD realiza a decodificação de códigos convolucionais, os quais podem ser representados por treliças, sendo que é através destas que o VD processa a decodificação. Conforme avaliado por McEliece e Lin, as treliças puncionada e mínima de um código de taxa 2/3 apresentam complexidade reduzida em relação à treliça convencional. Nesta dissertação, foram construídos VDs baseados nestas três estruturas de treliça do código de taxa 2/3. Os circuitos dos decodificadores foram implementados em dispositivos de lógica programável (PLDs) e em circuitos integrados de aplicação específica (ASICs), avaliando o impacto da treliça – e sua complexidade – sobre o consumo de energia, o consumo de hardware e a freqüência máxima de operação de cada VD.

Palavras-chave: Treliça de código convolucional. Decodificador de Viterbi. Desempenho de decodificador. Dispositivo de lógica programável. Circuito integrado de aplicação específica.

ABSTRACT

PEDRONI, Bruno Umbria. Hardware implementation of a Viterbi decoder based on the conventional, punctured, and minimal trellises. 2009. 105 f. Dissertação – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná. Curitiba, 2009.

The Viterbi decoder (VD) is widely used in modern communication systems, especially wireless systems, in which power consumption is critical. The VD decodes convolutional codes, which can be represented by trellis structures, and it is based on these trellises that the decoder processes. According to McEliece and Lin, the punctured and minimal trellis structures of a rate 2/3 code present reduced complexity in comparison to the conventional trellis. In this thesis, VDs based on these three trellis structures of the rate 2/3 code were constructed. The decoder circuits were implemented in Programmable Logic Devices (PLDs) and in Application Specific Integrated Circuits (ASICs), analyzing the impact that each trellis – and its complexity – has on power consumption, hardware consumption and maximum operating frequency.

Keywords: Convolutional code trellis. Viterbi decoder. Decoder performance. Programmable Logic Device. Application Specific Integrated Circuit.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

De tempos em tempos, invenções revolucionárias surgem e alteram a história, de modo que as vidas das pessoas nunca voltam a ser as mesmas. Na história pessoal das telecomunicações surgiram inventores e feitos como: Morse e o seu código na telegrafia; Marconi e a rádio telegrafia; Tesla e o principio do radar; e Korolev (chefe do Programa Sputnik Soviético) e o primeiro satélite artificial, Sputnik 1. Todas estas invenções e seus autores são conhecidos por disseminar uma tecnologia nova, deixando seu marco na história da humanidade. Entretanto, os feitos realizados por estes inventores e pioneiros advém da contribuição de muitos outros cientistas e inventores, cujo trabalho propiciou um meio para que idéias e aplicações pudessem surgir, pessoas que muitas vezes passam desapercebidas na história. Salvá i Campillo e Sömmering e o telégrafo eletromecânico, Braun e sua contribuição conjunta para a rádio telegrafia e até Fessenden e a modulação de amplitude (AM) na r[adio transmissão de áudio: menos reconhecidos (e conhecidos), mas todos contribuintes ávidos para o que as telecomunicações são hoje.

Observando assim, faz-se parecer que injustiças históricas foram cometidas. Entretanto, as invenções sozinhas muitas vezes não são suficientes para registrar um marco, onde uma tecnologia nova é consolidada apenas com contribuições que a viabilizam e tornam-na algo realmente concebível. Eis a razão de tanta fama dada aos “pós-inventores” que exploraram as aplicações, capacidades e limites das novidades recém-criadas, mostrando que isto pode causar um impacto ainda maior que a própria novidade.

Atualmente, sistemas de comunicação extremamente evoluídos são utilizados sem o usuário sequer dar conta das centenas de anos que foram despendidos para que a facilidade que ele possui pudesse existir. Dentre as “pós-invenções” que contribuíram para viabilizar as tecnologias modernas de comunicação aparece o decodificador de Viterbi (VD): um marco na história das telecomunicações e de extrema importância para a operação do mundo atual. Criado em 1967 por Andrew J. Viterbi [1], o VD é o método mais utilizado para decodificar códigos convolucionais [2], os quais são aplicados em sistemas de comunicação, para melhorar o seu desempenho frente ao ruído. Dentre

estes sistemas, destacam-se principalmente os sem fio, como enlaces de satélite, rádios digitais, televisão digital, redes de sensores sem fio (RSSF) e telefonia celular [2].

Sinais transmitidos dos satélites à Terra chegam altamente atenuados e corrompidos por ruído, porém a capacidade elevada de recuperação de informação do VD oferece muitas vantagens para este tipo de comunicação, permitindo alcances maiores, diminuindo o diâmetro da antena receptora e reduzindo a potência do transmissor. Como o consumo de energia nos sensores das RSSFs também é um fator crítico, a simples codificação utilizando códigos convolucionais e o VD são vitais para esta aplicação. Novamente, em cada um dos mais de um bilhão de aparelhos celulares no mundo, o VD também está presente. Como pode ser visto, em aplicações de comunicação sem fio onde o consumo de energia é crítico, o VD surge como uma solução extremamente eficaz.

Para realizar a decodificação, o VD opera de forma iterativa sobre a treliça do código convolucional, a qual pode ser obtida a partir do diagrama de estados do código. Todo código convolucional pode ser representado por uma treliça de estrutura regular, conhecida como a treliça convencional, porém duas outras representações de treliça são possíveis: a puncionada e a mínima. Além da treliça convencional, todo código possui uma representação através da treliça mínima; porém, a treliça puncionada só existe se o código for um código puncionado, os quais representam apenas uma parte dos códigos convolucionais [3]. Estas três estruturas de treliça são exploradas por Robert J. McEliece e Wei Lin em [4], onde a complexidade – medida em símbolos por bit – para as treliças de alguns códigos são propostas e comparadas, mostrando que a complexidade da treliça mínima é sempre inferior ou igual às das outras duas. Além desta demonstração, os autores deixam em aberto o impacto das treliças mínimas na eficiência do algoritmo de VDs construídos com estas. Desta maneira, permanece aqui a pergunta: treliças com complexidade reduzida possibilitam decodificadores melhores em termos de operação e construção? Nesta dissertação, esta pergunta pode ser traduzida da seguinte maneira: o consumo de energia em VDs é diretamente proporcional à complexidade da treliça utilizada para implementá-lo? Na busca desta resposta, duas outras características dos decodificadores serão exploradas: o consumo de hardware e a freqüência máxima de operação.

1.2 MOTIVAÇÃO

Devido à popularidade do VD, muitas técnicas já foram empregadas visando à redução do seu consumo de energia. O porquê disso pode ser explicado pela fração do consumo de energia em receptores de sistemas de comunicação que é representada pelo VD. Em aparelhos celulares de tecnologia CDMA de segunda geração, o VD consome aproximadamente um terço da energia do receptor [5]. Já em receptores para o padrão de redes sem fio IEEE 802.11, a decodificação através do VD corresponde a 35% do consumo de energia total [6]. Devido a estes percentuais consideráveis, a redução do consumo de energia do decodificador é algo muito explorado [7,8,9,10] e deverá continuar sendo, pois é de suma importância para sistemas de comunicação, principalmente os que possuem fonte de alimentação limitada (bateria).

Na busca por VDs mais econômicos em termos de energia, são três as diferentes estratégias de implementação: modificações no modo de construir o hardware; modificações do algoritmo; e implementações com diferentes estruturas de treliça. Para a primeira estratégia, destacam-se os trabalhos [7,8], onde melhorias nas estruturas das memórias e no roteamento entre os blocos do VD são geralmente os alvos. Já as modificações do algoritmo [9,10] visam redução de consumo de energia com implementações sub-otimizadas do VD, perdendo um pouco em termos de capacidade de decodificação, porém ganhando em termos de economia de energia dissipada. Nesta dissertação, entretanto, o terceiro tipo de estratégia é empregado, o qual analisa as diferentes estruturas de treliça. Para esta análise, os VDs foram construídos em circuitos integrados, podendo ser obtidas informações sobre o consumo de energia, além do consumo de hardware e da freqüência máxima de operação de cada decodificador. A comparação entre as treliças só é possível devido ao fato de que, apesar das estruturas diferentes entre elas, as três preservam o desempenho ótimo do VD, o qual não ocorre em implementações com modificações do algoritmo.

Como não há registros de aplicação da treliça mínima implementada em hardware, foi escolhido o próprio código apresentado em [4] para a análise. Neste artigo, os autores apresentam as três estruturas de treliça para um código de taxa 2/3,

onde a complexidade da treliça puncionada e da treliça mínima é metade do valor da complexidade da treliça convencional. Como em [5,6] os consumos dos VDs representam aproximadamente um terço do consumo total de energia do receptor, uma redução pela metade do consumo de VDs utilizando a treliça mínima (e puncionada, no caso do código de taxa 2/3 em questão) corresponderia a uma redução em torno de 15% do consumo total – algo realmente benéfico para os sistemas de comunicação sem fio.

1.3 CONTRIBUIÇÃO

Os resultados obtidos em [4] deixam em aberto quanto aos impactos que a complexidade reduzida da treliça mínima pode ter em VDs construídos com estas. Neste trabalho, o consumo de energia, o consumo de hardware e a freqüência máxima de operação de VDs para o código de taxa 2/3 considerado em [4] são analisados em circuitos integrados de lógica programável (PLDs) e de aplicação específica (ASICs). Além de ser o primeiro registro da implementação da treliça mínima em hardware, com os resultados obtidos aqui será possível avaliar se decodificadores construídos com as três diferentes estruturas de treliça deste código trazem benefícios para decodificadores de códigos convolucionais, podendo gerar impactos consideráveis para a indústria fabricante de circuitos integrados para sistemas de comunicação.

1.4 ORGANIZAÇÃO

A dissertação está dividida essencialmente em quatro partes: códigos convolucionais e o decodificador de Viterbi (VD); os circuitos dos VDs; processo de estimativa e medição do desempenho dos VDs; e a análise dos resultados.

Para poder explicar o funcionamento do VD, são apresentados conceitos da comunicação e da codificação convolucional no Capítulo 2. No tópico de codificação, é visto o circuito básico do codificador convolucional e o diagrama de estados para o código de taxa 2/3, além das três estruturas de treliça existentes – e suas

complexidades – para este código [4]. Com estes fundamentos expostos, os blocos e a operação do VD são apresentados no Capítulo 3.

As estratégias de construção e processamento dos decodificadores estão detalhadas no Capítulo 4, identificando as particularidades dos circuitos dos VDs de acordo com a treliça utilizada na implementação. O Capítulo 5 explora a maneira de estimar o consumo de energia, o consumo de hardware e a freqüência máxima de operação dos decodificadores. Em seguida, os Capítulos 6 e 7 mostram os resultados obtidos para os circuitos dos VDs construídos em FPGAs e CPLDs (em inglês, *Field Programmable Gate Array* e *Complex Programmable Logic Device*, respectivamente), através da linguagem de programação de hardware VHDL, e em ASIC (em inglês, *Application Specific Integrated Circuit*), usando o PSpice. Por último, as conclusões estão expostas no Capítulo 8.

O capítulo a seguir, portanto, apresenta um breve resumo da teoria da informação e comunicação, destacando onde e quando a codificação e decodificação de dados em sistemas de comunicação digital ocorrem e para que servem.

2. CÓDIGOS CONVOLUCIONAIS

Em 1948, Claude E. Shannon apresentou o teorema que identifica a taxa máxima teórica de transmissão de informação em um canal, de acordo com o nível de ruído [11]. Shannon demonstrou que em um canal ruidoso com capacidade *C* e transmitindo informação a uma taxa *R* (onde *R*<*C*) existem códigos que permitem que a probabilidade de erro no receptor seja suficientemente pequena, de modo que é possível transmitir informação sem erros a qualquer taxa inferior à capacidade.

Apesar de determinar o limite teórico da capacidade de um canal, Shannon não demonstrou como obter o método de detecção/correção de erros. Ao longo do tempo foram descobertos alguns destes métodos e, na prática, existem duas maneiras de se trabalhar com detecção e correção de erros [2]:

1. Pedido Automático de Repetição (em inglês, *Automatic Repeat Request*): o transmissor envia a mensagem com informação redundante e o receptor, ao decodificar a mensagem e identificar algum erro, solicita a retransmissão da mensagem original.

2. Correção Antecipada de Erros (em inglês, *Forward Error Correction*): o transmissor envia a mensagem com informação redundante de forma que o receptor, baseando-se nesta informação adicional, consegue corrigir certos tipos ou quantidades de erros e reconstruir a mensagem originalmente transmitida, sem a necessidade de solicitar a retransmissão.

A maneira como inserir redundância na mensagem original**,** para que a correção antecipada de erros possa ocorrer, é definida pelo tipo de codificação e pelo código escolhido para o sistema de comunicação digital. Esta transformação da mensagem original na mensagem (com redundância) a ser transmitida é também conhecida como o processo de codificação da palavra-informação na palavra-código, onde o tamanho desta, em bits, é sempre superior.

As duas principais categorias de correção antecipada de erros são os códigos de bloco e os códigos convolucionais [2]. Os códigos de bloco operam sobre blocos de informação de tamanho fixo e pré-determinado, sendo que as palavras-código são geradas baseando-se puramente na palavra-informação atual, não sendo necessários elementos de memória no codificador. Já os códigos convolucionais operam de maneira serial, onde os bits da palavra-informação são inseridos no codificador e a palavra-código é gerada a cada instante de tempo. Para isto, os codificadores convolucionais utilizam elementos de memória para armazenar os bits prévios para a formação da palavra-código atual.

As seguintes seções aprofundam sobre as características dos códigos convolucionais, incluindo a estrutura do codificador, o diagrama de estados e as treliças que representam um código.

2.1 CÓDIGO E CODIFICADOR CONVOLUCIONAL

Códigos convolucionais surgiram como uma alternativa para códigos de bloco e foram introduzidos por Peter Elias em 1955 [12]. Já em 1963, Aaron D. Wyner e Robert B. Ash trataram os códigos convolucionais sistematicamente [13]. Atualmente códigos convolucionais são amplamente utilizados em sistemas de comunicação com canais sem fio, como em enlaces de satélite, rádios digitais, televisão digital, redes de sensores sem fio e telefonia celular [2].

O processo de codificação convolucional é simples, ocorrendo de forma seqüencial. Em um código convolucional tem-se, por convenção, que o tamanho (em bits) da palavra-informação é definido como k e o tamanho da palavra-código como n, resultando em um código com taxa $r = k/n$ (sempre $r<1$) [2]. Desta maneira, k bits entram no circuito, são processados e armazenados (códigos convolucionais possuem memória), e n bits são gerados na saída do codificador. Este processo é repetido seqüencialmente, sempre entrando uma palavra-informação de k bits e saindo uma palavra-código de n bits.

O circuito do codificador convolucional é composto de dois elementos básicos: memórias (registradores) e somadores, os quais são representados pelos símbolos das

Figuras 2.1(a) e 2.1(b), respectivamente. O arranjo das memórias e suas interligações com os somadores são o que definem o codificador, ou seja, definem quais as palavras-código geradas de acordo com as palavras-informação de entrada [2]. No circuito do codificador (também conhecido como a forma canônica) de r=2/3 ilustrado na Figura 2.1(c) (extraído de e analisado por McEliece e Lin em [4]), pode-se observar que os k=2 bits de entrada possuem um elemento de memória cada, onde os somadores combinam os sinais para formar as n=3 saídas. Por definição, a quantidade de elementos de memória nos registradores de cada bit de entrada é conhecida como v_i, com $1 \leq i \leq k$ [2]. O código gerado por este codificador específico é chamado neste trabalho de código (3,2,2,3), e o porquê desta nomenclatura será explicado a seguir.

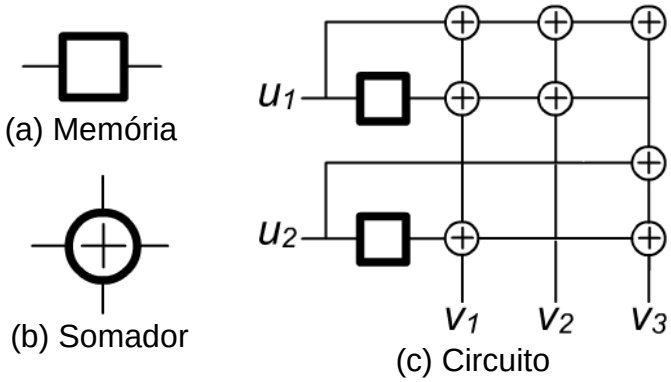


Figura 2.1 – Elementos e circuito do codificador para o código (3,2,2,3).

A quantidade de memória presente no circuito do codificador impacta em quanto uma palavra-código sendo gerada é influenciada pelos bits das palavras-informação anteriores, definindo assim outros dois parâmetros do código: a memória e o comprimento de restrição. A memória de um codificador m é definida em [2] como o tamanho do maior registrador presente no circuito do codificador, sendo que para o caso da Figura 2.1(c) tem-se m = máx(v_1,v_2) = máx(1,1) = 1. O comprimento de restrição v de um código é definido em [2] como a soma das memórias do circuito, ou seja, $v = \Sigma v_i$. Para o circuito da Figura 2.1(c), o comprimento de restrição v = 1 + 1 = 2.

Além dos parâmetros r, m e v, os códigos convolucionais também possuem algumas propriedades de distância entre as palavras geradas, destacando a distância livre mínima d_{free}. Para melhor compreender a origem deste parâmetro e sua

importância, é necessário primeiramente apresentar o diagrama de estados do codificador, o qual também auxiliará futuramente na construção da treliça do código.

Enquanto a forma canônica representa uma estrutura mais próxima do circuito físico do codificador, o diagrama de estados representa uma forma mais teórica da operação do codificador, demonstrando a transição entre os estados de suas memórias, os sinais necessários para percorrer estas transições e os sinais gerados ao percorrê-las. O diagrama de estados do codificador da Figura 2.1(c), gerador do código (3,2,2,3), está ilustrado na Figura 2.2.

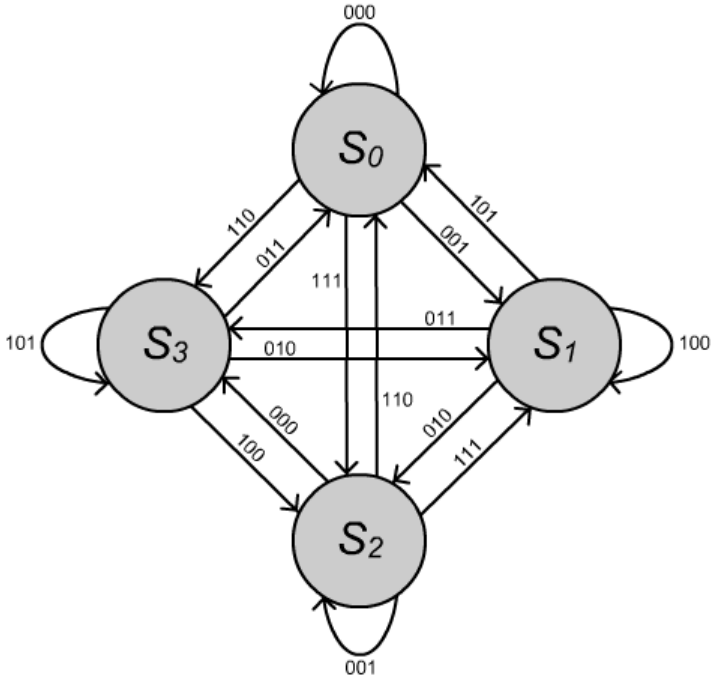


Figura 2.2 – Diagrama de estados do código (3,2,2,3).

A quantidade de estados do codificador é definida pela quantidade de elementos de memória presentes no mesmo, resultando em um número de estados igual a 2^{v} [2]. Como pode ser observado na Figura 2.2, o diagrama de estados deste codificador possui 2^{2} = 4 estados. Os nomes dos estados são definidos pelos valores atuais das memórias, podendo assumir, neste codificador, os valores de S_0=*A*="00", S_1=*B*="01", S_2=*C*="10" e S_3=*D*="11". O bit da esquerda representa a memória de u_1 e o bit da direita representa a memória de u_2. Os valores dos *n*=3 bits nos ramos, formados

por $v_1v_2v_3$, representam as palavras-código geradas pelo codificador ao tramitar de um estado da memória para outro.

Através do diagrama de estados, a distância livre mínima d_{free} de um código pode ser definida como a mensagem de menor peso (soma da quantidade de bits ‘1’ nas palavras-código que compõem a mensagem) originando no estado S_0=”00” e retornando ao estado S_0, desconsiderando o ciclo de peso nulo em torno do estado S_0 [2]. Para o código (3,2,2,3), pode ser observado que a mensagem de menor peso que pode ser gerada partindo de S_0 e retornando ao mesmo é a seqüência de palavras-códigos de $S_0 \rightarrow S_1 \rightarrow S_0$, equivalente a “001 101”. Somando todos os bits ‘1’ nesta mensagem resulta em uma d_{free}=3 para este código. Com isto é garantido que:

1) erros de $\lfloor (d_{free}/2) \rfloor$ = 1 bit podem ser corrigidos;
2) erros de d_{free}-1 = 2 bits podem ser detectados.

Com os quatro principais parâmetros do código definidos, é possível agora classificar o código da Figura 2.1(c) como um código (n,k,v,d_{free}) = (3,2,2,3), esclarecendo a origem de sua nomenclatura. Na próxima seção será demonstrada a construção da treliça convencional deste código, a qual é baseada em seu diagrama de estados, além das treliças puncionada e mínima.

2.2 TRELIÇAS

A forma canônica e o diagrama de estados do codificador foram apresentados na seção anterior, onde a primeira assemelha-se com o circuito real do codificador e o segundo representa a transição entre os estados das memórias do codificador. O diagrama de estados pode também ser convertido em uma terceira representação gráfica: a treliça. Nesta seção são apresentadas as três estruturas de treliça que representam um código convolucional: a treliça convencional, a treliça puncionada e a treliça mínima. As estruturas convencional e mínima existem para todos os códigos, enquanto as treliças puncionadas existem apenas para códigos puncionados [3].

Treliças são formas de representar códigos em estruturas de árvores, de modo que cada nó na estrutura representa um estado (valor atual das memórias) e os ramos

que ligam estes nós representam as transições entre os estados do codificador. Os rótulos dos ramos identificam as palavras-código geradas pelo codificador ao transitar de um estado para outro. Pode-se, assim, definir as seguintes características da construção de um módulo de treliça:

a. o número de nós é uma função de *v*;
b. o número de ramos que entra em cada nó é uma função de *k*;
c. e o tamanho de cada rótulo é uma função de *n*.

Os nós da treliça podem ser identificados por letras ou números, sendo estes representativos dos estados das memórias do codificador, podendo seguir uma nomenclatura conforme a Tabela 2.1.

Memória do codificador		**Nome do nó**	
m1	**m2**	**Número (decimal)**	**Texto**
0	0	0	A
0	1	1	B
1	0	2	C
1	1	3	D

Tabela 2.1 – Nomenclaturas possíveis para os nós da treliça do código (3,2,2,3).

A relação das operações do codificador com as operações da treliça está apresentada na Tabela 2.2.

Etapa	**Codificador**	**Treliça**
1	Estado atual da memória é M-	Nó atual é o que representa M-
2	Palavra-informação atual alterará a memória do codificador para M+	Transição do nó atual para o nó representando M+ devido à palavra-informação atual
3	As interligações das memórias e somadores definem a palavra-código gerada, de acordo com M- e a palavra-informação atual	O rótulo do ramo do nó atual (representando M-) ao novo nó (representando M+) define a palavra-código gerada

Tabela 2.2 – Relação da operação do codificador com a operação da treliça.

A seguir estão apresentados os três tipos de módulo de treliça para o código (3,2,2,3), identificando as suas particularidades.

- **A treliça convencional**

A representação clássica de um código em estrutura de árvore, a treliça convencional, pode ser facilmente obtida através do diagrama de estados do codificador. A estrutura (ou módulo) da treliça convencional é composta de uma fase (duas colunas de nós, interligados pelos ramos) e, como pode ser vista na Figura 2.3, possui as seguintes características:

a. número de fases = 1;
b. número de nós por fase = $2^v = 4$;
c. número de ramos que entra em cada nó = $2^k = 4$;
d. tamanho de cada rótulo = $n = 3$ bits.

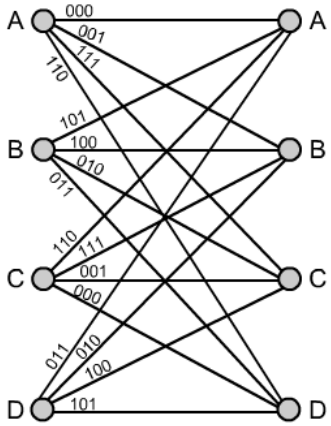


Figura 2.3 – Módulo da treliça convencional para o código (3,2,2,3).

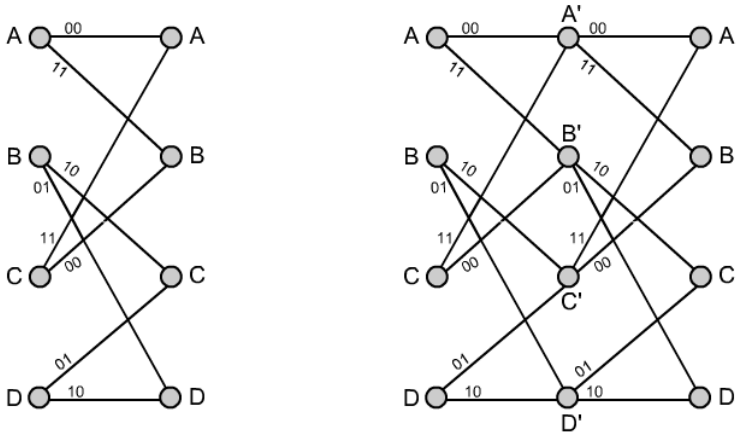
A geração de uma palavra-código na treliça convencional segue a seqüência apresentada na Tabela 2.2. Portanto, de acordo com o estado atual e o próximo estado (definido pelos valores das memórias e palavra-informação atuais), certo percurso é tomado na treliça, gerando a palavra-código identificada pelo rótulo do ramo deste percurso.

- **A treliça puncionada**

A classe de códigos puncionados [3] é uma subclasse dos códigos convolucionais, ou seja, nem todo código convolucional pode ser obtido através do

processo de puncionamento. Para criar um código puncionado, deve-se primeiro começar com um módulo de código convolucional (n,k,v,d_{free}) = (N,1,v,d_{free}) e repeti-lo k vezes, resultando em um código convolucional (kN,k,v,d_{free}). Se agora forem puncionados kN-n bits dos rótulos dos ramos, é obtido um código convolucional (n,k,v,d_{free}'), onde d_{free} e d_{free}' não são necessariamente iguais.

Para obter o código (3,2,2,3) em análise, inicia-se com o módulo de treliça do código (2,1,2,5) da Figura 2.4(a). Depois, mais um módulo deste mesmo código (2,1,2,5) é concatenado ao primeiro, obtendo um novo módulo, agora com duas fases. Desta maneira, o módulo de treliça do código (4,2,2,5) da Figura 2.4(b) é obtido [3].



(a) Código (2,1,2,5) (b) Código (4,2,2,5)

Figura 2.4 – Módulos de treliça dos códigos (2,1,2,5) (a) e (4,2,2,5) (b).

Como o objetivo é o código convolucional (3,2,2,3), basta puncionar o bit menos significativo (LSB) dos ramos da segunda fase da treliça exibida na Figura 2.4(b). Com isto, obtém-se o módulo da treliça puncionada da Figura 2.5, a qual possui as seguintes características:

a. número de fases = k = 2;
b. número de nós por fase = 2^v = 4;
c. número de ramos que entra em cada nó por fase = 2^{k^*} = 2 (onde k^*=1 é o k do código original (N,1,v,d_{free}));
d. tamanho de cada rótulo => n(fase 1)=2 bits e n(fase 2)=1 bit.

O número de nós por fase não foi alterado pois o código inicial já possuía *v*=2. Como o valor de *k* em cada fase é igual a 1, e não se altera com o puncionamento, apenas dois ramos entram em cada nó por fase agora [4]. Os rótulos dos ramos são os únicos elementos que variam de acordo com a fase.

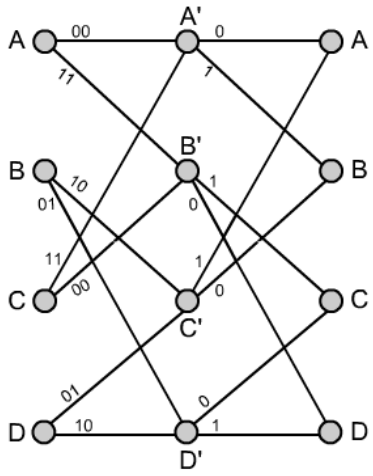


Figura 2.5 – Módulo da treliça puncionada para o código (3,2,2,3).

A palavra-código gerada pela treliça puncionada é formada pela concatenação dos dois bits do ramo selecionado na primeira fase com o bit do ramo selecionado na segunda fase. Portanto, se certo percurso for tomado na treliça puncionada, é possível perceber que a palavra-código gerada pelos rótulos de seus ramos é a mesma que aquela gerada pela treliça convencional, (para esta mesma transição de estados), demonstrando que ambas representam o mesmo código (3,2,2,3) gerado pelo circuito da Figura 2.1(c).

- **A treliça mínima**

O terceiro tipo de treliça considerada é a treliça mínima, cuja representação existe para todos os códigos. A treliça mínima para códigos convolucionais foi proposta por Robert J. McEliece, demonstrando a construção dos módulos de treliça mínima [14] e a complexidade dos mesmos [4], este em conjunto com o Wei Lin. No mesmo período em que os autores de [4] estavam estudando treliças mínimas, Sidorenko e Zyablov também propuseram sua própria maneira de construir treliças mínimas para códigos convolucionais [15]. O nome dado à treliça mínima advém da complexidade dela, a qual será o assunto da próxima seção.

Todo código pode ser representado por uma matriz, chamada de matriz geradora. Através de manipulações desta matriz, McEliece mostrou que é possível obter para todos os códigos uma treliça equivalente à convencional – ou seja, geradora do mesmo código –, porém com complexidade reduzida. Detalhes sobre matrizes geradoras e como obter as treliças mínimas não são apresentados nesta dissertação, pois o foco é a aplicação das treliças em hardware e os impactos que cada estrutura de treliça tem sobre os circuitos. Portanto, para o estudo é necessário apenas analisar as características do módulo de uma treliça mínima. O módulo da treliça mínima do código (3,2,2,3) aparece na Figura 2.6, extraído de [4].

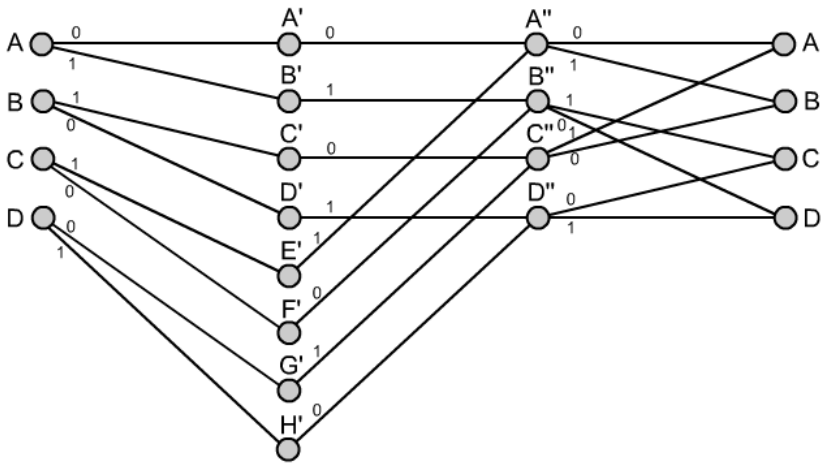


Figura 2.6 – Modulo da treliça mínima para o código (3,2,2,3).

Observando a Figura 2.6, é possível obter as seguintes informações sobre a estrutura da treliça:

a. número de fases = n = 3;
b. número de nós => inicia e termina com 2^v= 4 nós;
c. número de ramos que entra em cada nó por fase => ramos(fase1)=1, ramos(fase2)=2 e ramos(fase3)=2;
d. tamanho de cada rótulo = 1 bit.

Novamente, se uma transição específica entre dois estados for selecionada, concatenando-se os três bits dos ramos (um bit por ramo tomado), são geradas as mesmas palavras-código das treliças convencional e puncionada. Vale destacar que o número de fases da treliça mínima é sempre igual a n e o número de fases com nós de decisão (nós com mais de um ramo entrando) é sempre igual a k, tornando sua estrutura menos regular que as outras duas já analisadas. Apesar desta estrutura irregular da treliça mínima, ela possui a vantagem de apresentar complexidade no máximo igual à da treliça convencional, característica que será vista na seção 2.4.

2.3 EXEMPLO DE CODIFICAÇÃO

No exemplo a seguir é demonstrado o processo de codificação de uma mensagem utilizando o circuito do codificador.

Considere a mensagem $\boldsymbol{u}$ = “11 00 10 01 00”, composta de 5 palavras-informação, entrando no codificador (estado inicial = S_0). Após cinco progressões na codificação, a seqüência $\boldsymbol{v}$ = “110 011 111 111 101”, formada por 5 palavras-código, é gerada na saída do circuito. Analisando os cinco instantes diferentes do circuito da Figura 2.7, pode-se observar que os futuros valores dos elementos de memória (em verde) são os valores da palavra-informação atual (em vermelho), com os bits “X”, entrando em seguida no codificador, representando valores indiferentes dos bits de informação da próxima mensagem. A composição dos n=3 bits (em azul) é a palavra-código gerada.

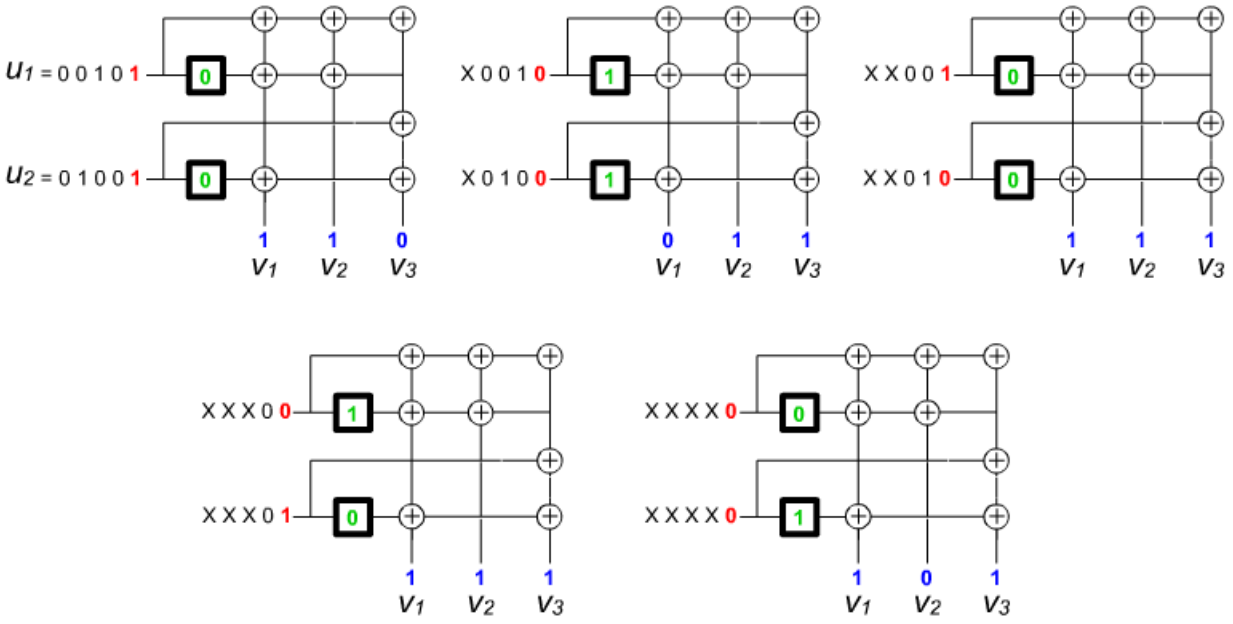


Figura 2.7 – Codificação de uma mensagem formada por 5 palavras.

A codificação iniciou no estado S_0=“00” e, por ter a última palavra-informação = ”00”, retornou as memórias para o estado S_0, de modo que a próxima mensagem a ser codificada conseqüentemente iniciará em S_0. Códigos com este característica de codificação são conhecidos como *códigos terminados*, sendo este um detalhe importante que será levado em consideração no processo de decodificação, visto no Capítulo 3.

2.4 COMPLEXIDADE DAS TRELIÇAS

O termo “complexidade de treliça” pode ser definido de várias maneiras [16], entretanto será utilizada a definição que é representada pelo *número de símbolos dos ramos por bit codificado*, chamada aqui de simplesmente *símbolos por bit*. Esta complexidade foi aplicada em [4] e os autores argumentam que ela é proporcional à complexidade computacional do algoritmo de Viterbi, utilizado na implementação do VD. Para obter o valor da complexidade, soma-se o número de símbolos (bits) nos rótulos dos ramos do módulo da treliça em todas as fases e divide-se o total por *k*. Para

as treliças convencional e puncionada é possível obter a complexidade através das seguintes equações [4]:

$$TC_{conv} = (n/k) \cdot 2^{v+k} \text{ símbolos/bit} \quad (2.1)$$

$$TC_{punc} = (n/k) \cdot 2^{v+1} \text{ símbolos/bit} \quad (2.2)$$

Para as treliças convencional e puncionada das Figuras 2.3 e 2.5, respectivamente, calcula-se que TC_{conv} = 24 símbolos/bit e TC_{punc} = 12 símbolos/bit. Portanto, para o código (3,2,2,3), a complexidade da treliça puncionada é metade do valor da complexidade da treliça convencional.

Já a complexidade da treliça mínima, por esta ser construída a partir de manipulações da matriz geradora e por não possuir uma estrutura regular, é calculada através da matriz geradora do código ou através de análise gráfica, sendo esta segunda maneira a utilizada aqui. Observando a Figura 2.6 e sabendo que cada ramo da treliça mínima representa apenas um bit, basta somar o número de ramos da treliça e dividir por k. Portanto, $TC_{min} = 24/k = 12$ símbolos/bit: a mesma complexidade que da treliça puncionada e metade da complexidade da treliça convencional.

Apesar de não ter sido apresentado o decodificador ainda, mas lembrando que o VD é construído a partir de uma treliça, pode-se começar a pensar sobre o impacto da complexidade sobre os circuitos decodificadores. Será que uma complexidade reduzida de treliça reflete em circuitos mais simples? Ou possivelmente circuitos que demandam menos computações para se obter os mesmos resultados? Será que a estrutura irregular da treliça mínima não implicará na construção de um VD mais complexo? Estas perguntas serão respondidas para o código (3,2,2,3) ao longo da dissertação e, para poder entender a importância delas um pouco melhor, o Capítulo 3 apresenta a seguir o processo de decodificação, especificamente através do decodificador de Viterbi.

3. DECODIFICAÇÃO DE CÓDIGOS CONVOLUCIONAIS

No Capítulo 2 foram apresentados códigos convolucionais, mostrando o codificador do código (3,2,2,3) e o processo de codificação. Neste capítulo serão abordados os temas necessários para converter uma mensagem codificada de volta na mensagem original, começando pelo algoritmo que torna possível a decodificação e depois o decodificador que realiza a operação do algoritmo.

3.1 ALGORITMO DE VITERBI

Códigos convolucionais foram criados por Peter Elias em 1955 como uma alternativa para códigos de bloco [12]. Logo em seguida John Wozencraft e Barney Reiffen propuseram a decodificação seqüencial como um método eficiente de decodificação para códigos convolucionais com comprimento de restrição elevado [17]. Em 1963 James Massey propôs a decodificação de limiar (do inglês, *threshold decoding*) [18], menos eficiente que a decodificação seqüencial, porém de implementação mais simples, gerando assim aplicações práticas com códigos convolucionais em transmissão digital por telefone, satélite e canais de rádio [2].

Apesar destes dois métodos sub-otimizados - métodos que não garantem a melhor seqüência decodificada - terem surgido, foi apenas em 1967 que Andrew J. Viterbi propôs um algoritmo de máxima verossimilhança, oferecendo o melhor desempenho do ponto de vista probabilístico [1]. De fácil implementação para códigos de comprimento de restrição baixo [2], o algoritmo de Viterbi, também conhecido como o decodificador de Viterbi (VD), realiza a decodificação de forma iterativa, não sendo necessário recorrer à extensiva análise de todas as palavras do alfabeto do código.

O algoritmo de Viterbi popularizou-se nos anos 70, quando passou a ser muito utilizado nos sistemas de comunicação espacial por satélite da NASA [19,20]. Hoje em dia, o VD pode ser encontrado principalmente em sistemas de transmissão digital de vídeo e aplicações de transmissão sem fio com fonte de energia limitada (bateria), como aparelhos celulares e redes de sensores sem fio.

No Capítulo 2 foi apresentado o processo de codificação, mostrando o circuito codificador e as treliças do código (3,2,2,3). O VD será utilizado, portanto, para realizar o processo reverso da codificação, a decodificação. Através da redundância inserida pelo codificador, o VD tenta obter uma mensagem decodificada igual à originalmente transmitida. Utilizando-se a treliça do código para decodificar a mensagem, o VD (para decisão abrupta) opera da seguinte maneira em cada fase do módulo da treliça:

1) A palavra-código é recebida.
2) As distâncias de Hamming entre a palavra-código recebida e as palavras-código esperadas são calculadas.
3) As distâncias calculadas são somadas às métricas atuais de seus respectivos ramos.
4) Cada nó é analisado individualmente: a menor soma (distância de Hamming + métrica atual) é considerada a vencedora. O ramo da métrica vencedora é armazenado e a métrica é atualizada.

Estas 4 etapas ocorrem para cada palavra-código recebida. Depois de gerada a treliça inteira, com um módulo para cada *n* bits da mensagem, a obtenção da mensagem original ocorre através do processo de *traceback*. O *traceback* analisa os percursos vencedores de trás para frente e as palavras-informação são obtidas de acordo com os ramos escolhidos para chegar no nó em questão.

Na etapa 2 da decodificação, para que a transição de um estado para outro ocorra, somente algumas palavras-código podem ser geradas: as palavras-código representadas pelos rótulos dos ramos que entram em cada nó. Para calcular as distâncias de Hamming entre a palavra-código recebida e as esperadas, como foi utilizada decodificação com decisão abrupta, basta somar a quantidade de bits diferentes em cada posição das palavras. Por exemplo, as palavras “1**01**” e “1**10**”, com o bit da esquerda sendo o mais significativo, possuem distância (entre si) de 2, pois diferem nos bits 1 e 0 (em negrito).

Para o código (3,2,2,3), as etapas da operação do VD utilizando a treliça convencional podem ser traduzidas da seguinte maneira:

1. A palavra-código de *n*=3 bits é recebida;

2. As distâncias de Hamming entre a palavra-código e os rótulos dos ramos da treliça são calculadas;
3. As distâncias são somadas às métricas atuais de cada nó;
4. As menores somas que entram em cada nó são consideradas as vencedoras, sendo que o ramo vencedor é armazenado e a métrica atualizada com esta soma.

Após o recebimento de todas as palavras-código, o processo de *traceback* ocorre na treliça completa. Iniciando no estado S_0=*A*="00", lembrando que o código é terminado, as palavras-informação necessárias para gerar as transições entre os ramos vencedores são obtidas pelo decodificador e a mensagem decodificada é formada.

Para os casos das treliças puncionada e mínima, o VD opera da mesma maneira no processo de *traceback*. Já nas atualizações das métricas, a treliça puncionada executa as atualizações *k*=2 vezes por palavra-código recebida, por possuir duas fases, e a treliça mínima executa as atualizações *n*=3 vezes por palavra-código, devido às suas três fases. Entretanto, na treliça mínima a comparação das somas (distância + métrica) não é necessária em *n-k* fases do módulo da treliça, por apenas um ramo entrar em cada nó nestas fases. Na treliça mínima do código (3,2,2,3), não há nós de decisão na primeira fase e na quarta etapa do VD é executada apenas a atualização das métricas com as menores, e únicas, somas.

3.2 EXEMPLO DE DECODIFICAÇÃO

Para validar a operação dos VDs para cada treliça, podem ser utilizadas as mensagens do exemplo da Seção 2.4. Considerando a mensagem ***u***="11 00 10 01 00" e sua versão codificada ***v***="110 011 111 111 101", suponhe que a transmissão de ***v*** pelo canal de comunicação resulte em uma mensagem corrompida, a qual é recebida como ***v'*** = "1**0**0 011 111 11**0** 101", todas exibidas na Figura 3.1.

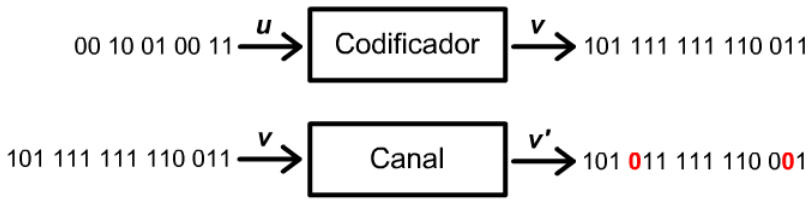


Figura 3.1 – Mensagem corrompida por ruído inserido pelo canal.

Como o segundo e décimo segundo bits (em vermelho) foram corrompidos pelo ruído presente no canal, o decodificador fica encarregado de gerar a mensagem ***u'***, de 5 palavras-informação, na expectativa de obter a mensagem original ***u***.

O processo de decodificação, com cada estrutura de treliça, está representado nas Figuras 3.2 a 3.5. Nestas, cada treliça é composta de 5 módulos, por estarem decodificando mensagens compostas de 5 palavras-código. Nas treliças, os números sobre os nós indicam as métricas atuais dos nós, os ramos com traços mais largos são os ramos vencedores e os ramos com traços mais finos são os ramos que foram rejeitados por apresentarem soma (distância + métrica) mais elevadas ou perderam no critério de desempate. O percurso formado pelos ramos de traços mais largos constitui o percurso vencedor, que será analisado pelo *traceback* para obtenção da palavra decodificada.

Vale destacar que na análise da operação dos decodificadores foi utilizado código terminado, o qual consiste em as memórias do codificador começarem e terminarem a codificação de uma mensagem no estado S_0. Isto ocorre simplesmente aplicando, para o código (3,2,2,3), "00" como a última palavra-informação da mensagem, de modo que a próxima mensagem a ser codificada já iniciará automaticamente no estado S_0. No processo de decodificação esta é uma informação adicional que é levada em consideração na formação da treliça (partindo do estado S_0) e no *traceback*, de modo que este também sempre inicia a sua operação no estado S_0.

- **Decodificação com a treliça convencional**

Iniciando sempre no estado $S_0=A$ por ser um código terminado, a treliça é formada de acordo com as palavras-código que são recebidas (os n=3 bits acima de

cada módulo da treliça, na Figura 3.2). No primeiro módulo, a palavra-código “100” foi recebida e analisada, gerando as seguintes distâncias:

1) distância *A*-*A* = distância entre “100” e “000” = 1;
2) distância *A*-*B* = distância entre “100” e “001” = 2;
3) distância *A*-*C* = distância entre “100” e “111” = 2;
4) distância *A*-*D* = distância entre “100” e “110” = 1.

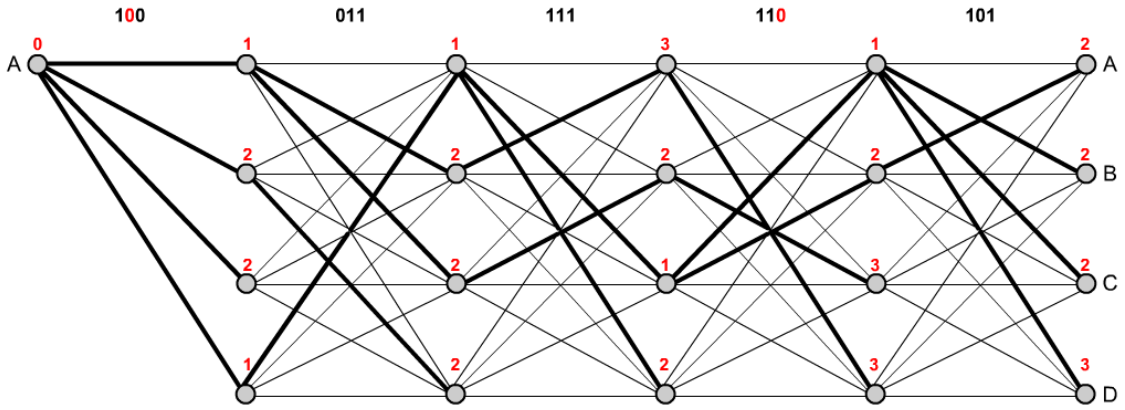


Figura 3.2 – Formação da treliça convencional para decodificação.

Como a métrica inicial de *A* é nula e não há nós de decisão neste primeiro módulo, as métricas de cada nó foram atualizadas com as próprias distâncias calculadas. Já nos demais módulos, há decisão entre qual o ramo com a menor soma que entra em cada nó. Por exemplo, analisando o nó *B* no segundo módulo, as seguintes somas entram nele:

1) distância *A*-*B* + métrica *A* = distância entre “011” e “001” + 1 = 2;
2) distância *B*-*B* + métrica *B* = distância entre “011” e “100” + 2 = 5;
3) distância *C*-*B* + métrica *C* = distância entre “011” e “111” + 2 = 3;
4) distância *D*-*B* + métrica *D* = distância entre “011” e “010” + 1 = 2.

Sendo que as somas dos ramos *A*-*B* e *D*-*B* apresentaram o mesmo valor, foi definido como critério de desempate que o ramo do nó mais acima é o vencedor. No caso do nó *B*, as somas de *A* e de *D* empataram (soma = 2), tornando o ramo do nó *A* o vencedor.

Formada a treliça inteira, a geração das palavras-informação ocorre através do *traceback*. A Figura 3.3 destaca em vermelho o percurso vencedor da decodificação, sendo que as palavras-informação geradas em cada módulo são simplesmente os valores das memórias neste instante; lembrando que isso ocorre devido às linhas dos bits de entrada do codificador possuírem apenas um elemento de memória em seus registradores.

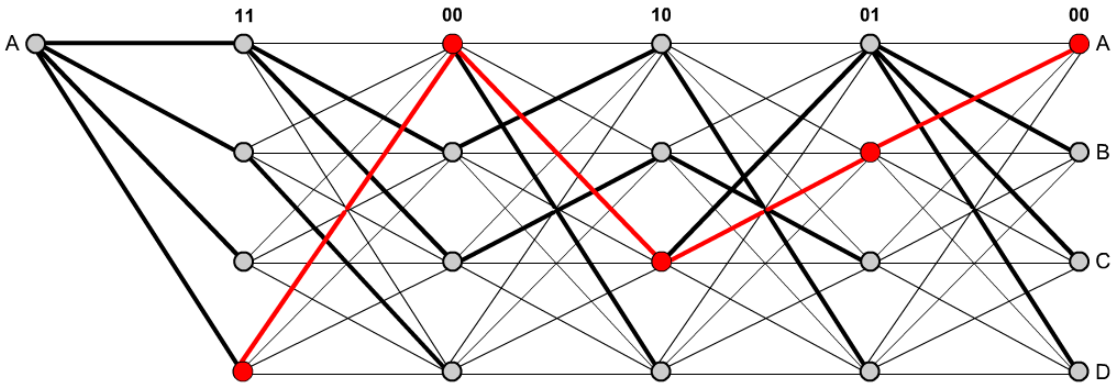


Figura 3.3 – *Traceback* na treliça convencional.

Iniciando sempre no estado *A* no final da treliça (código terminado), o *traceback* ocorre de trás para frente, verificando sempre qual foi a palavra-informação necessária para que a transição específica ocorresse. Por exemplo, do final do quarto módulo ao final do quinto módulo da treliça, houve uma transição do estado *B* para o estado *A*, a qual indica que a palavra-informação “00” foi codificada neste instante; e isto era esperado, pois é um código terminado. Já entre o terceiro e quarto módulos, houve uma transição de *C* para *B*, indicando que a palavra-informação neste instante foi a “01”. Desta maneira, repetindo este processo mais três vezes, a palavra-informação ***u’***=“11 00 10 01 00” é obtida e a decodificação, apesar dos dois bits errados na mensagem recebida, foi capaz de corrigir os erros inseridos pelo canal.

Em resumo, a decodificação é o processo de análise do percurso vencedor, de modo que as transições entre os estados (nós) indicam quais foram os bits de entrada necessários para gerar estas transições específicas. Os processos de decodificação utilizando as treliças puncionada e mínima estão apresentados a seguir, com a formação da treliça e o *traceback* presentes nas mesmas figuras.

- **Decodificação com a treliça puncionada**

Como foi visto anteriormente, a operação do VD com a treliça puncionada é a mesma que para a treliça convencional, com a diferença da atualização das métricas duas vezes por módulo, devido às duas fases da treliça puncionada. O *traceback* também opera da mesma maneira, porém apenas os nós em vermelho na Figura 3.4 são os que definem as palavras-informação decodificadas, estes ocorrendo uma vez por módulo. Apesar desta diferença, pode-se observar que a mensagem decodificada foi a seqüência ***u'***=“11 00 10 01 00”, a mesma que para a treliça convencional e, novamente, corrigida dos dois erros inseridos pelo canal.

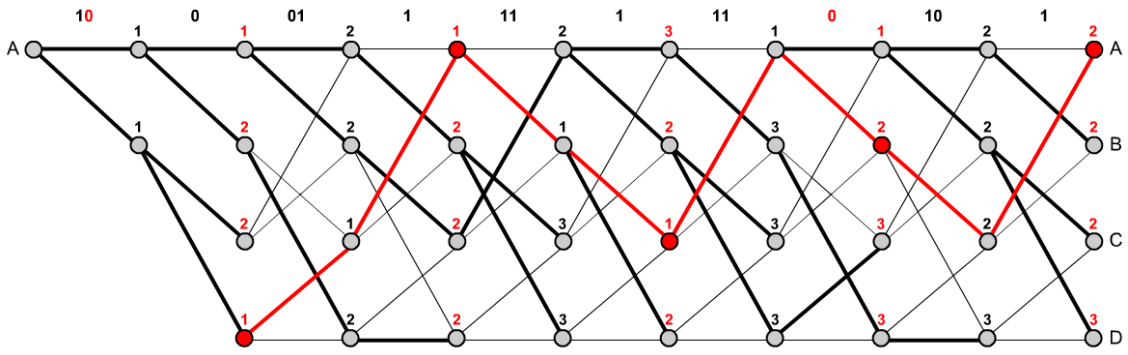


Figura 3.4 – Decodificação utilizando a treliça puncionada.

- **Decodificação com a treliça mínima**

Novamente, apesar dos bits serem analisados individualmente na treliça mínima conforme mostra a Figura 3.5, a mesma mensagem ***u'***=“11 00 10 01 00” foi corretamente obtida. Isso mostra que, apesar das estruturas dos módulos das treliças serem diferentes, os VDs obtiveram os mesmos resultados, o qual era previsto, pois as três treliças representam o mesmo código [4].

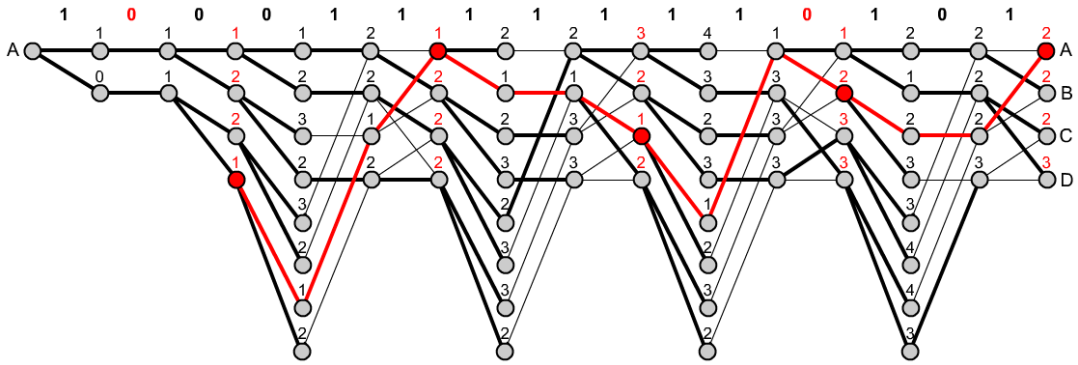


Figura 3.5 – Decodificação utilizando a treliça mínima.

3.3 COMPONENTES DO DECODIFICADOR

Para realizar as quatro etapas do VD, são necessários três componentes: o *Hamming distance calculator* (em inglês, calculador das distâncias de Hamming), o *Add-Compare-Store* (em inglês, Somar-Comparar-Armazenar) e o *Traceback* [7,8,21,22]. Na Figura 3.6 pode ser vista a disposição de cada um destes blocos dentro do decodificador do código (3,2,2,3), onde *cw* representa a palavra-código recebida, *hds* as distâncias de Hamming calculadas, *metrics* as métricas, *path* a memória de percursos vencedores e *dw* a palavra decodificada.

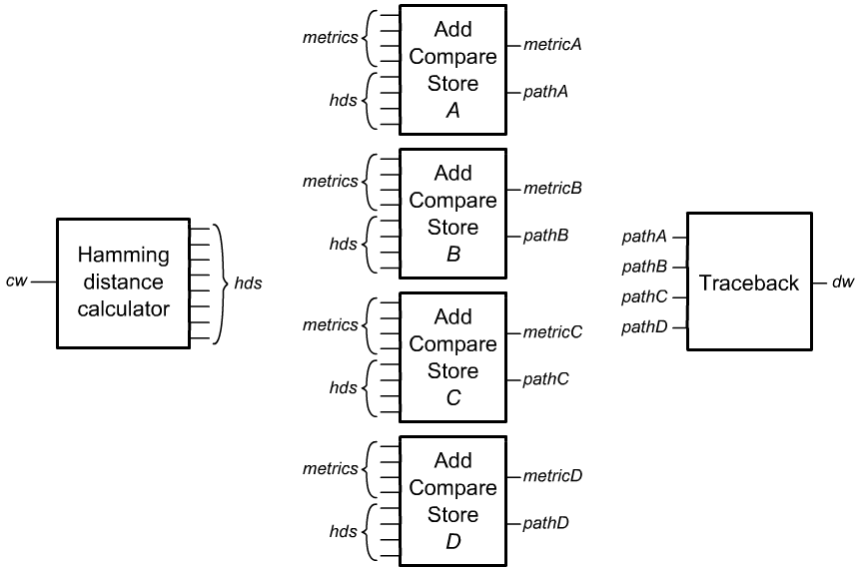


Figura 3.6 – Diagrama geral do VD.

Estes componentes realizam funções específicas no VD e suas particularidades estão detalhadas a seguir.

- ***Hamming distance calculator* (HD)**

O bloco HD do decodificador é responsável por calcular a diferença entre a palavra-código recebida e as palavras-código esperadas (aquelas presentes nos ramos da treliça). As distâncias de Hamming entre as palavras são passadas ao bloco *Add-Compare-Store*, onde são utilizadas para escolher os melhores percursos dentro do módulo da treliça e obter as novas métricas. Apenas um bloco HD é utilizado no decodificador.

- Sinais de entrada: a palavra-código recebida.
- Sinais de saída: as distâncias de Hamming entre a palavra recebida e as palavras esperadas (rótulos dos ramos).

- ***Add-Compare-Store* (ACS)**

O bloco ACS executa as funções de soma (*Add*), comparação (*Compare*) e armazenamento (*Store*). Apesar de este bloco ser usualmente chamado de *Add-Compare-Select* [8,22,23], o outro nome foi utilizado aqui pois é neste bloco que as métricas são armazenadas, enfatizando o ponto-de-vista de hardware do VD.

A parte *Add* do ACS soma as métricas atuais com as distâncias de Hamming calculadas, oriundas do HD. A cada ramo que entra no nó é atribuído um somador. Os resultados destas somas são repassados ao *Compare*, onde a busca pela menor métrica ocorre. O ramo vencedor é então armazenado e a métrica atualizada no *Store*. Portanto, cada nó possui uma memória de métrica e uma memória de percursos (conjunto de ramos) vencedores.

- Sinais de entrada: as distâncias de Hamming geradas pelo HD e as métricas atuais, além dos percursos intermediários no caso das treliças puncionada e mínima (a ser visto na Seção 3.4).
- Sinais de saída: as métricas atualizadas e os percursos vencedores armazenados, além dos percursos intermediários no caso das treliças puncionada e mínima.

- ***Traceback* (TB)**

Após a formação da treliça inteira, ou seja, após o recebimento de todas as palavras-código que compõem a mensagem codificada (incluindo a parte “terminada” do código), os percursos vencedores são repassados ao bloco TB, onde as palavras decodificadas serão obtidas. Começando de trás para frente e iniciando no estado S_0=*A*=“00” (código terminado), o TB analisa os percursos vencedores e decodifica as palavras em cada módulo.

- Sinais de entrada: os percursos vencedores.
- Sinais de saída: a mensagem decodificada.

3.4 PARTICULARIDADES DO VD DE ACORDO COM A TRELIÇA

Os blocos HD, ACS e TB do decodificador, definidos na seção anterior, foram vistos por enquanto de forma genérica. Aqui estão detalhadas as particularidades dos VDs de acordo com a treliça utilizada. Para todos os decodificadores, foram definidas métricas formadas por 3 bits, podendo variar entre 0 e 7 (decimal). Os somadores dos ACSs foram construídos com *lock-on-overflow* – LOV, limitando o valor máximo das métricas em 7 (decimal) mesmo quando um overflow da soma resultaria em uma métrica menor. Nas próximas figuras, o valor das métricas está identificado simplesmente pela letra do nó, ou seja, os sinais *A*, *B*, *C* e *D* representam as métricas dos nós *A*, *B*, *C* e *D*, respectivamente. Além destes sinais, *hd0* a *hd7* representam as distâncias de Hamming entre a palavra-código *cw* recebida e os rótulos “000” a “111” dos ramos, *path* representam os percursos vencedores de cada nó e *dw* representa a palavra decodificada.

- **Treliça convencional**

A treliça convencional possui uma estrutura regular, com apenas uma fase. Esta fase possui 2^v=4 nós, com 2^k=4 ramos entrando em cada nó. A Figura 3.7 mostra

o decodificador com a treliça convencional, também chamado de decodificador, ou VD, convencional.

- HD: este bloco calcula a distância de Hamming entre a palavra-código (n=3 bits) recebida e as 2^n=8 possíveis palavras-código. Como a distância entre a palavra-código recebida e uma palavra-código representada pelo rótulo de um ramo varia entre 0 e 3 (decimal), os sinais de saída deste bloco são 8 pares de bits.

- ACS: cada um dos 2^v=4 nós possui um somador por ramo (total de 2^k=4 somadores por nó), junto com um comparador de 2^k=4 entradas para a análise das métricas. Como cada nó possui uma métrica, são necessárias 4 memórias de métricas e 4 memórias de percursos vencedores.

- TB: depois de recebida e processada a mensagem codificada, as 2^v=4 memórias de percursos vencedores são repassadas ao bloco TB para a obtenção da mensagem original (baseando-se na mensagem codificada recebida).

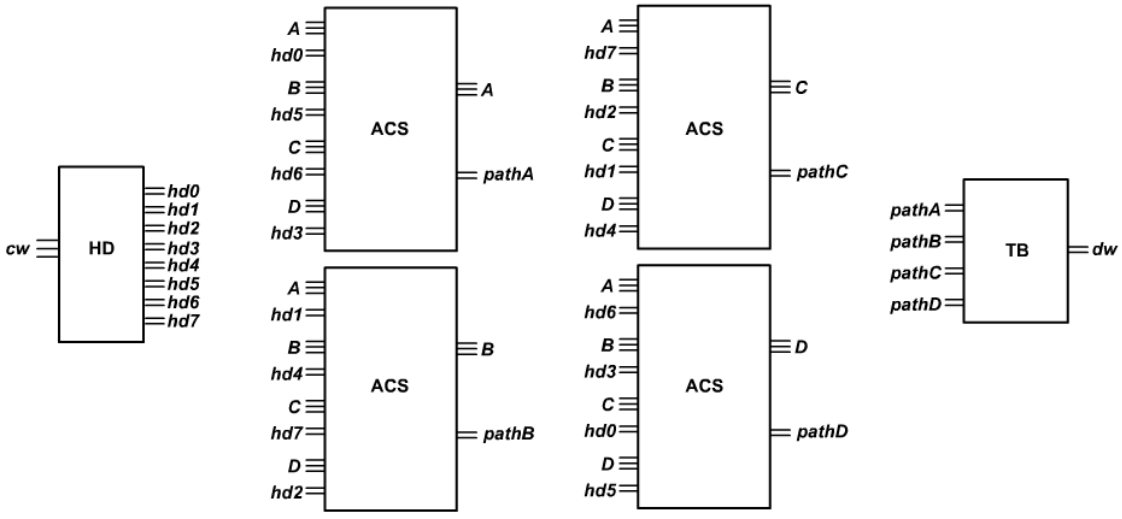


Figura 3.7 – Diagrama de blocos do VD convencional.

- **Treliça puncionada**

A estrutura da treliça puncionada é um tanto quanto regular por ela ser formada por uma concatenação de dois módulos de uma treliça cuja taxa de código r=1/2. A treliça puncionada é, portanto, composta de k=2 fases, com 2^v=4 nós em cada fase e 2^{k^*}=2 ramos entrando em cada nó, onde k^*=1 é o número de bits da palavra-informação da treliça que originou a treliça puncionada. A primeira fase da treliça possui ramos que representam os primeiros 2 bits da palavra-código. Já na segunda fase, como os bits mais significativos foram "puncionados" para totalizar 3 bits de palavra-código, os ramos representam apenas o terceito bit da palavra-código. Devido ao módulo da treliça puncionada ser formado por 2 fases, surgiram então duas maneiras diferentes de construir o decodificador puncionado. A primeira utiliza 4 blocos ACS na primeira fase e mais 4 blocos ACS na segunda fase, conhecida aqui como decodificador puncionado com múltiplos ACSs. A segunda maneira utiliza apenas 4 blocos ACS, os quais são reutilizados na segunda fase também, conhecida aqui como decodificador puncionado com ACSs únicos.

- HD: como as duas fases da treliça puncionada possuem rótulos dos ramos com quantidades diferentes de bits, o bloco HD calcula a distância de Hamming entre os primeiros dois bits da palavra-código e os bits dos rótulos dos ramos da primeira fase, podendo variar entre 0 e 2 (decimal). Depois o HD calcula a distância entre o terceiro bit da palavra-código e os bits dos rótulos dos ramos da segunda fase, variando apenas entre 0 e 1 (decimal). Os sinais de saída deste bloco são 4 pares de bits na primeira fase e 4 pares bits na segunda, com o bit mais significativo destes últimos igual a '0'.

- ACS (múltiplos ACSs): no decodificador utilizando a treliça puncionada, cada nó possui um somador por ramo (total de 2^{k^*}=2 somadores por nó), junto com um comparador de 2^{k^*}=2 entradas para a análise das métricas. Além da diferença na quantidade de componentes do ACS em relação ao do decodificador convencional, o VD puncionado utiliza uma memória intermediária (ver Figura 3.8,

sinal *int_path*) para armazenar os percursos vencedores entre as fases. Este conceito foi introduzido de modo a manter reduzido o tamanho da memória de percursos vencedores e aproveitar também o bloco TB utilizado no decodificador convencional. Salienta-se que só é viável a aplicação dos percursos intermediários para simplificar a memória de percursos vencedores caso o código apresente a característica de $2^k \leq 2^v$. Desta maneira, os percursos intermediários são os percursos vencedores da primeira fase da treliça. Na segunda fase, portanto, o percurso armazenado na memória definitiva de percursos vencedores (*path*) é o percurso intermediário do nó de origem do percurso vencedor da segunda fase. Devido ao código apresentar $2^k \leq 2^v$, armazenando apenas o percurso intermediário na memória de percursos vencedores é suficiente para conhecer-se o percurso inteiro pelo módulo da treliça.

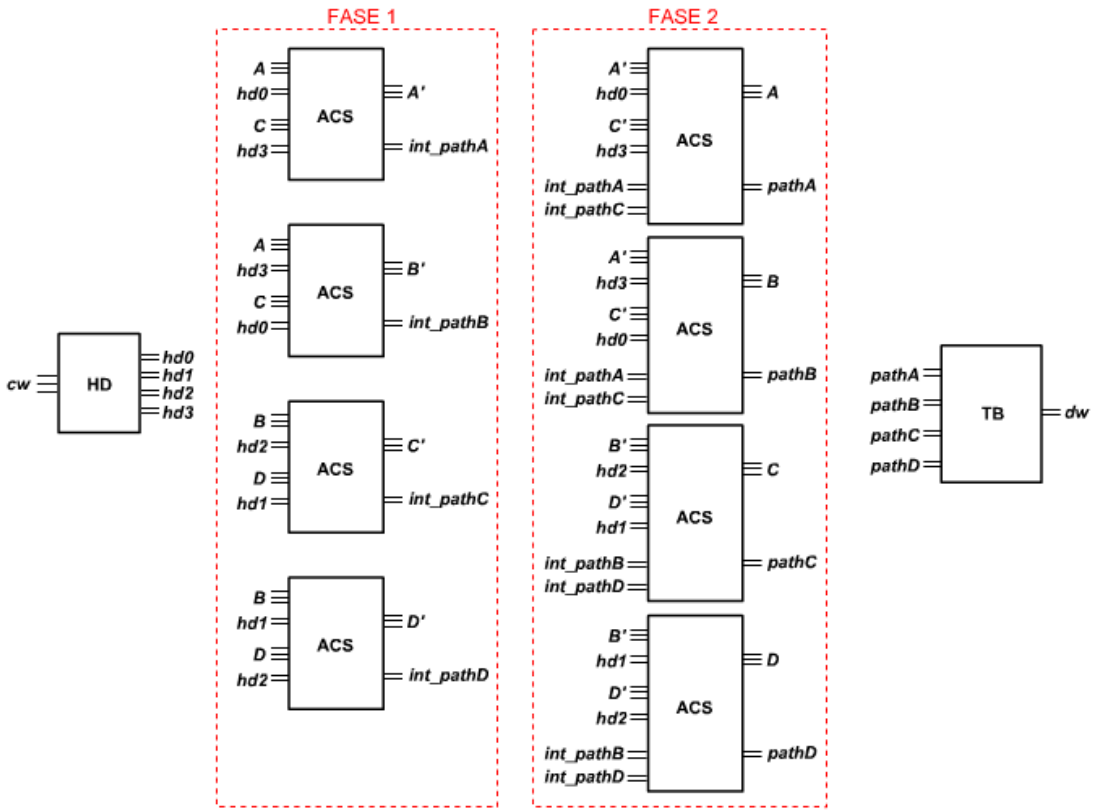


Figura 3.8 – Diagrama de blocos do VD puncionado com múltiplos ACSs.

Como será visto no próximo capítulo, os sinais gerados pelo HD para o VD puncionado com múltiplos ACSs são separados por fase da treliça. Isto possibilita a utilização de somadores mais simples nos ACSs da segunda fase da treliça, devido às distâncias calculadas nesta fase serem formadas por apenas um bit.

- ACS (ACSs únicos): como a quantidade de nós por fase é constante na treliça puncionada, é possível um decodificador onde a estrutura de somadores e comparadores é aproveitada em ambas as fases. Idem ao outro VD puncionado, há 2 somadores e 1 comparador por nó, porém estes são utilizados em ambas as fases da treliça puncionada. O VD puncionado com ACSs únicos está exibido na Figura 3.9, sendo que a principal diferença deste para o outro VD puncionado é na seleção dos sinais de percursos intermediários e no momento de atualização das métricas. Aqui a decisão para armazenar o percurso intermediário ou o percurso vencedor definitivo ocorre internamente no ACS, além das métricas serem atualizadas em ambas as fases.

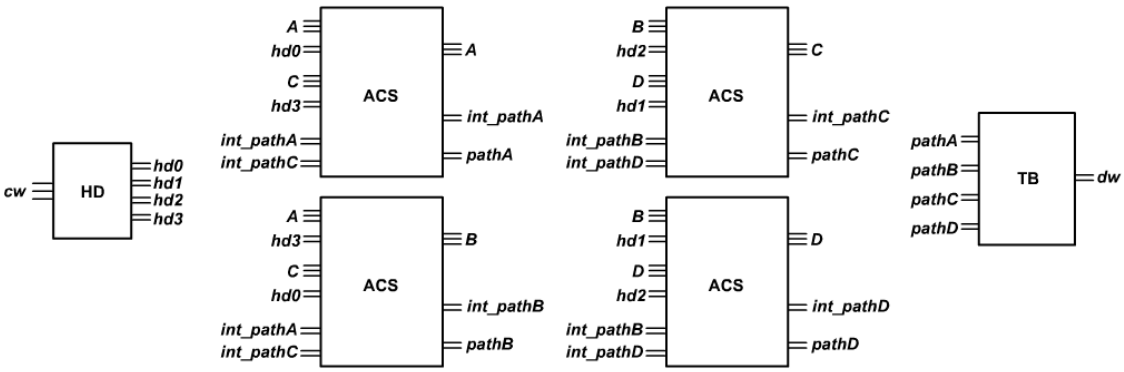


Figura 3.9 – Diagrama de blocos do VD puncionado com ACSs únicos.

- TB: este bloco é idêntico ao empregado no decodificador convencional.

- **Treliça mínima**

Devido à estrutura irregular da treliça mínima, a construção do VD também seguiu uma metodologia menos regular. Como ocorreu para os VDs puncionados, são

também possíveis duas formas diferentes de construir o VD utilizando a treliça mínima: uma que utiliza blocos ACS diferentes para cada fase da treliça e outra que reaproveita os mesmos ACSs em cada fase. Ambas formas de construção estão apresentadas a seguir.

- HD: como os rótulos dos ramos nas n=3 fases da treliça são formados por apenas um bit, as distâncias entre o bit da palavra-código recebida e os rótulos podem variar entre 0 e 1. Desta maneira, o HD gera sinais de apenas um bit: ‘0’ quando o bit recebido for igual ao esperado e ‘1’ quando for diferente.

- ACS (múltiplos ACSs): sendo que cada fase da treliça mínima possui uma quantidade diferente de nós, este primeiro método de construção utiliza um bloco ACS por nó do módulo da treliça (ver Figura 3.10). Como os nós da primeira fase da treliça não são nós de decisão, a parte *Compare* dos ACSs aqui não é necessária. Na segunda fase, há 4 nós com 2 ramos entrando em cada nó; portanto, há dois somadores em cada ACS, com um comparador de 2 entradas para a análise das métricas. Nesta fase da treliça, a memória intermediária é novamente empregada para armazenar os percursos vencedores, sendo requerida uma por nó. Na terceira fase, há 2^{v}=4 nós, de modo que cada ACS novamente analisa duas somas (2 somadores e um comparador com duas entradas por nó), agora armazenando na memória definitiva os percursos vencedores.

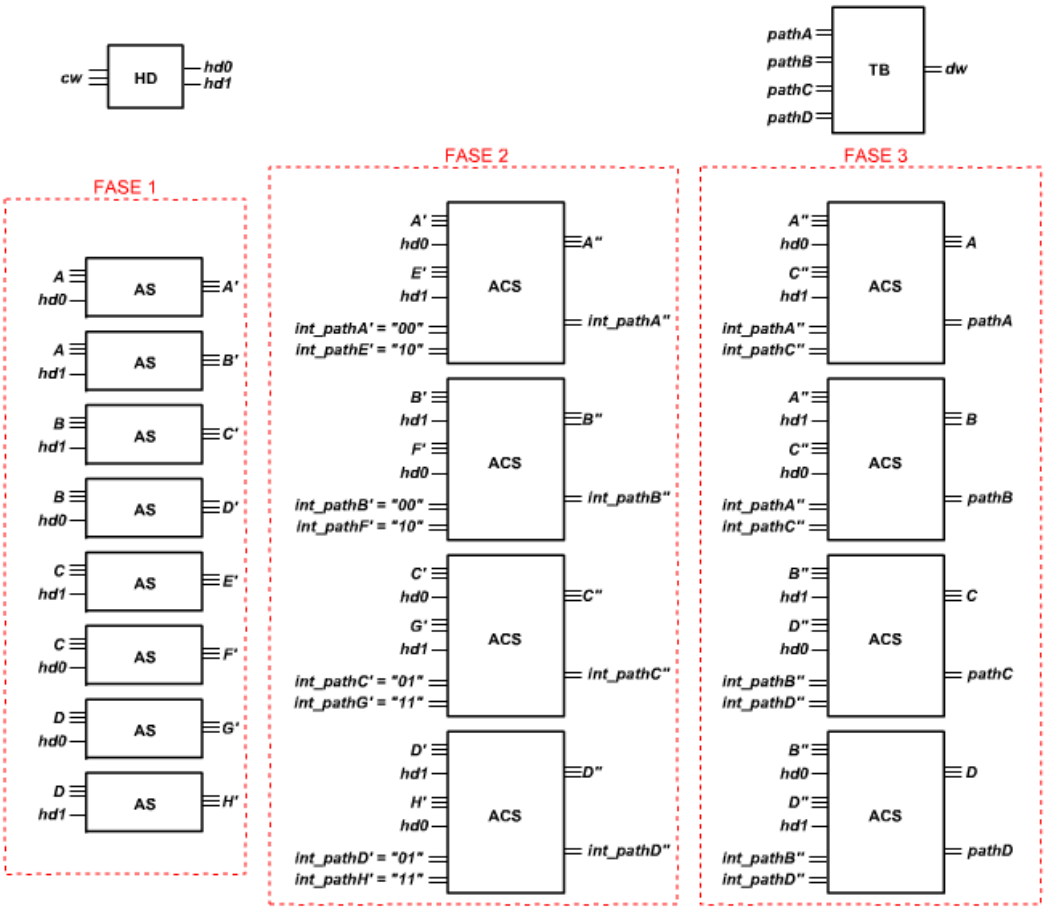


Figura 3.10 – Diagrama de blocos do VD mínimo com múltiplos ACSs.

- ACS (ACSs únicos): nesta segunda técnica de construção, ilustrada na Figura 3.11, quatro blocos ACS são utilizados nos 4 nós superiores da treliça, sendo re-aproveitados nas *n*=3 fases. Como os quatro nós inferiores da segunda fase da treliça simplesmente realizam um soma e armazenam a métrica, eles possuem apenas as partes *Add* (com um somador) e *Store* do ACS. A operação do VD com esta técnica ocorre simplesmente comparando uma soma com ela mesma na primeira fase, e realizando as comparações das duas somas que entram em cada nó na segunda e terceira fases. Ou seja, na primeira fase na realidade não ocorrem comparações, mas como os blocos ACSs são os mesmos nas 3 fases, se forem comparadas duas somas com os mesmos valores, evidentemente a métrica atualizada só poderá ser a própria soma. Na segunda e terceira fases da treliça, a operação do VD é a mesma que para a primeira técnica de construção. O sinal

phase (em inglês, fase) seleciona quais dados são inseridos no ACS de acordo com a fase atual.

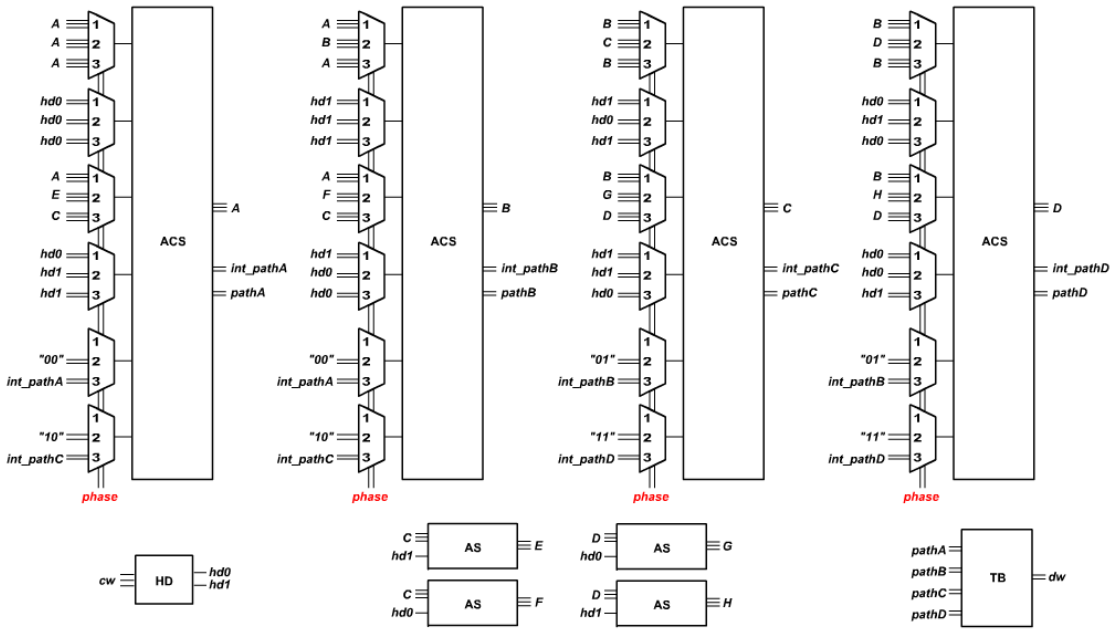


Figura 3.11 – Diagrama de blocos do VD mínimo com ACSs únicos.

- TB: este bloco é idêntico ao empregado no decodificador convencional.

Aparentemente a técnica de construção com ACSs únicos é mais útil e simples que a primeira. Entretanto, este técnica exige uma seleção mais complexa dos sinais que entram nos ACSs, de modo que há um roteamento maior nos circuitos, tornando-a também um tanto quanto elaborada.

O Capítulo 3 abordou os temas de decodificação de códigos convolucionais, mostrando os passos necessários para executar o algoritmo de Viterbi. Foram examinados também os decodificadores e suas particularidades de acordo com a treliça utilizada para implementá-los. À primeira vista, através dos diagramas de blocos, supõe-se que os dois decodificadores que utilizam múltiplos ACSs não serão vantajosos para serem construídos por utilizarem vários elementos a mais que os outros decodificadores. Já o VD convencional e os VDs com ACSs únicos

apresentaram diagramas de blocos mais concisos, de modo que a mesma estrutura dos ACSs pôde ser reutilizada em todas as fases da decodificação.

O próximo capítulo apresenta os circuitos específicos dos VDs, detalhando as estratégias utilizadas para implementá-los. Os sinais internos dos circuitos e a dimensão dos mesmos também serão apresentados, mostrando os instantes exatos em que as etapas de decodificação ocorrem. Além de verificar a validação da complexidade das treliças em termos de consumo de energia dos VDs, esta dissertação visa validar se o consumo de hardware e a freqüência máxima de operação dos decodificadores também estão ou não relacionados com a complexidade, de modo que é de fundamental importância analisar cada circuito digital utilizado na construção dos decodificadores para manter a comparação entre eles justa.

4. CIRCUITOS DOS DECODIFICADORES

No Capítulo 3 foi apresentado o VD, além de como ocorre a decodificação utilizando as treliças convencional, puncionada e mínima. Neste capítulo, os circuitos dos decodificadores utilizando os três tipos de treliças são apresentados, servindo de base para os circuitos implementados. Primeiramente, porém, serão vistas as estratégias empregadas na construção dos circuitos e no processamento de dados nos decodificadores, de acordo com a treliça utilizada.

4.1 ESTRATÉGIAS DE CONSTRUÇÃO E PROCESSAMENTO

Certas considerações sobre os decodificadores foram expostas no Capítulo 3. Além dos blocos HD e ACS variarem de acordo com a treliça selecionada para realizar a decodificação, o conceito de percurso intermediário foi também introduzido. Abaixo estão detalhadas estas estratégias utilizadas na construção dos decodificadores.

- **Estratégias de construção**

1) Percursos intermediários (do inglês, *intermediate paths*)

O conceito de percursos intermediários foi introduzido como uma estratégia de extrema vantagem para os decodificadores por duas razões: redução do tamanho dos registradores de percursos vencedores e redução de memória e processamento do bloco de *traceback* (TB). Nas treliças puncionada e mínima, os percursos intermediários foram inseridos para armazenar os nomes dos percursos vencedores nas fases anteriores à última. Isto é, a primeira fase da treliça puncionada possui nós de decisão, sendo necessário armazenar os percursos vencedores neste instante. Isto poderia ocorrer de duas maneiras: registrando os percursos vencedores em todas as fases da treliça ou armazenando temporariamente os percursos vencedores e registrando-os na memória definitiva dos percursos apenas ao final do módulo da treliça.

A primeira maneira é de implementação mais simples, porém acarreta em um consumo significativo de hardware (flip-flops do tipo D – DFFs), pois varia de acordo

com a quantidade de palavras sendo decodificadas L. Nas treliças puncionada e mínima, como o número de fases com nós de decisão é equivalente a k, os VDs puncionado e mínimo possuiriam então uma memória de percursos vencedores o dobro ($k=2$) do tamanho daquela do VD convencional. Como o número de bits (com um DFF utilizado por bit) para representar o nome de um nó é igual a 2, são necessários $2.2^{v}.L=8L$ DFFs para armazenar os percursos vencedores no decodificador convencional. Já nos decodificadores puncionado e mínimo, haveria um acréscimo de $8L$ DFFs. Além disto, a memória do *traceback* dos decodificadores sofreria também um aumento na mesma proporção.

A segunda maneira, apesar de tornar a operação do ACS mais complexa, reduz o consumo de hardware do circuito, de modo que o acréscimo de DFFs torna-se constante, independente do número de palavras sendo decodificadas. Como o nome de cada nó é representado por 2 bits, para o código (3,2,2,3) há um acréscimo de apenas $2.2^{v}=8$ DFFs. Caso fosse escolhido um código com um k mais elevado, a mesma regra seria válida, pois a memória de percursos intermediários seria reaproveitada (sobrescrita) a cada fase com nós de decisão. Vale lembrar que os percursos intermediários só podem ser aplicados desta maneira devido ao código (3,2,2,3) apresentar $2^{k}\leq 2^{v}$. Nos decodificadores puncionado e mínimo, foi utilizada esta segunda maneira de implementação.

2) ACSs: as duas maneiras de construir

Como foi visto no Capítulo 3, os decodificadores puncionado e mínimo possuem duas possíveis estruturas. No VD puncionado, a quantidade de nós é a mesma em ambas as fases, diferenciando os ACSs apenas no armazenamento do percurso intermediário (fase 1) ou do percurso vencedor definitivo (fase 2). O VD puncionado com ACSs únicos diferencia este armazenamento internamente, enquanto o VD puncionado com múltiplos ACSs armazena o percurso intermediário nos ACSs da primeira fase e armazena o percurso vencedor nos ACSs da segunda fase. Já em ambas as estruturas do VD mínimo, a primeira fase apresenta 4 métricas que são atualizadas apenas na primeira fase do módulo da treliça, com as outras 4 métricas sendo reaproveitadas no método com ACS únicos, mas triplicadas no método com

múltiplos ACSs. Além disto, o VD puncionado com múltiplos ACSs utiliza somadores um pouco mais simples nos ACSs da segunda fase, se comparado com os somadores (re-aproveitados na segunda fase) do VD puncionado com ACSs únicos.

3) ACS e TB: minimizando os registradores dos percursos

Esta estratégia foi utilizada para reduzir o consumo de hardware nos três tipos de decodificadores. Como foi empregado um código terminado, sabe-se de antemão que a última palavra a ser codificada é sempre “00”. Desta maneira, como o nó de origem da decodificação será sempre o S_0=A=“00”, o primeiro percurso vencedor não precisa ser armazenado, eliminando um par de DFFs por nó na memória de percursos vencedores. Na memória de *traceback* dois DFFs por nó também podem ser eliminados, pois a última palavra a ser decodificada será sempre “00”. Além destas duas situações, quando o último percurso vencedor está para ser armazenado na memória de percursos vencedores, ou seja, quando a última palavra codificada está para completar o L-ésimo módulo da treliça, o percurso vencedor dos nós pode ser armazenado diretamente na memória de *traceback*. Como a memória de percursos vencedores não precisa conter este último percurso, há uma redução de mais dois DFFs por nó da treliça. Apesar de esta técnica reduzir a quantidade de hardware despendido, a redução ocorre de forma igual para todos os decodificadores, servindo apenas como uma estratégia de redução da quantidade total de DFFs.

- **Estratégias de processamento**

1) Clocks do sistema: *gated clocks*

O termo *clock gating* origina-se do inglês e consiste na técnica de utilizar o clock apenas quando necessário, de forma a reduzir o consumo de energia. Para obter isto, pode-se utilizar, por exemplo, portas lógicas AND, onde o clock é um dos sinais de entrada da porta e o outro sinal habilita ou não a passagem do clock para partes específicas do circuito. Com isto, os DFFs que armazenam as métricas, os percursos vencedores e os percursos do *traceback* ficam estáveis durante uma parcela maior do

tempo, diminuindo a atividade de chaveamento dos circuitos e conseqüentemente reduzindo o consumo de energia.

Esta técnica é amplamente utilizada em processadores e ASICs em geral [24], porém em alguns circuitos integrados como CPLDs e FPGAs, devido à estrutura robusta de roteamento do clock nos mesmos, *clock gating* foi visto não ser a estratégia mais interessante. Os decodificadores elaborados em VHDL, portanto, não utilizaram esta técnica, sendo que a habilitação de seus DFFs foi realizada apenas através do pino de *enable*, mostrando uma redução no consumo de energia se comparado com os circuitos implementados com *clock gating*.

2) Bloco HD: atualizado apenas quando necessário

Similar ao que ocorre nos DFFs com *clock gating*, os HDs nos decodificadores foram construídos de forma a reduzir o consumo de energia, diminuindo a atividade de chaveamento dos circuitos que calculam as distâncias de Hamming. Para isto, os sinais de entrada dos HDs são atualizados apenas quando uma computação de distâncias deve ocorrer, ou seja, a palavra-código (ou bits dela) é armazenada serialmente em DFFs, sendo repassada a um outro conjunto de DFFs apenas quando o cálculo é necessário. No decodificador mínimo, como há n=3 fases, isto não ocorre; porém, nos decodificadores puncionado e convencional, o HD não necessita operar continuamente, trazendo grande vantagem em termos de consumo de energia – principalmente no convencional.

4.2 SINAIS COMUNS AOS VDs

Nos circuitos das Seções 4.3 a 4.5, alguns sinais estão presentes em vários circuitos dos decodificadores, sendo que as suas abreviações ou siglas facilitam a exibição nas figuras. Na Tabela 4.1, os sinais mais utilizados nos VDs estão listados, onde os valores dentro de colchetes após os nomes dos sinais indicam os bits que compõem os sinais, ou seja, a dimensão destes.

Função	**Nomes do sinal**	
Bit codificado (entrada)	*coded_bit*[0]	*cb*[0]
Palavra-código	*code_word*[2:0]	*cw*[2:0]
Palavra decodificada (saída)	*dec_word*[1:0]	*dw*[1:0]
Contador de bits	*bit_count*[1:0]	*bc*[1:0]
Contador de palavras	*word_count*[3:0]	*wc*[3:0]
Distância de Hamming	*hd*0[1:0] a *hd7*[1:0]	
Métricas	*mA*[2:0] a *mD*[2:0]	
Percursos vencedores	*pathA1*[*L*-1:0], *pathA0*[*L*-1:0] a *pathD1*[*L*-1:0], *pathD0*[*L*-1:0]	
Percursos intermediários	*int_pathA1*[*L*-1:0], *int_pathA0*[*L*-1:0] a *int_pathD1*[*L*-1:0], *int_pathD0*[*L*-1:0]	

Tabela 4.1 – Siglas e nomenclatura dos sinais globais dos VDs.

Os VDs foram construídos de modo que os bits da palavra-código são lidos e armazenados em cada borda positiva de clock; já as métricas e os percursos são atualizados na borda negativa de clock. As métricas em todos os VDs são reinicializadas no momento em que *word_count*=*L* e *bit_count*=3, ou seja, no processamento do último bit da mensagem codificada, sendo que esta informação foi omitida das figuras.

Vale destacar também que às métricas foi atribuída a dimensão de 3 bits e o código é formado por *L*=15 palavras. Para os percursos vencedores, como os “nomes” dos nós são formados por 2 bits, tem-se que *pathA1* é o bit mais significativo da memória de percursos vencedores do estado S_0=*A*=”00”, e *pathA0* o bit menos significativo.

4.3 O DECODIFICADOR CONVENCIONAL

Antes de apresentar o circuito do decodificador convencional, convém revisar a estrutura da treliça convencional, lembrando que ela apresenta uma disposição regular dos nós e ramos, como pode ser observado na Figura 4.1.

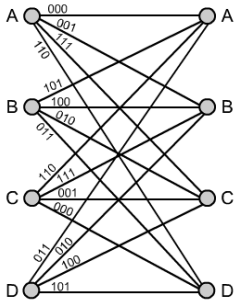


Figura 4.1 – Módulo da treliça convencional para o código (3,2,2,3).

As Figuras 4.2 e 4.3 ilustram o circuito VD convencional, com os comentários dos blocos e elementos do decodificador em vermelho. Como pode ser visto na Figura 4.2(a), a segunda estratégia de processamento – Bloco HD: atualizado apenas quando necessário – foi aplicada ao HD, de modo que os sinais *hd0* a *hd7* permanecem estáticos até o terceiro bit da palavra-código ser recebido. Quando uma borda positiva de clock ocorre e o bit atual é o segundo, o *bit counter* incrementa a contagem de bits para 3, ao mesmo tempo em que *n*=3 DFFs registraram a palavra-código *cw* recebida. Como somadores e comparadores utilizam estes sinais gerados pelo HD, manter estes estáticos reduz a atividade (chaveamento) dos circuitos combinacionais que os procedem, reduzindo conseqüentemente o consumo de energia. O circuito do comparador de 4 entradas do VD convencional pode ser observado na Figura 4.2(b), sendo que este circuito é composto, na realidade, de três comparadores de 2 entradas (*COMP*), além de um multiplexador que seleciona o bit menos significativo do percurso vencedor. Portanto, neste circuito quatro somas (*sumX*) são analisadas e os valores da métrica (*metric*) e do percurso vencedor (*path(1)* e *path(0)*) são repassados à parte *Store* do ACS.

A memória dos percursos vencedores *pathX* utilizou a terceira estratégia de construção – ACS e TB: minimizando os registradores dos percursos –, porém ainda manteve um par de DFFs a mais. Isto porque os blocos dos circuitos construídos em CPLDs e FPGAs ficariam um pouco mais complexos se o TB ficasse encarregado de obter este último percurso vencedor diretamente da saída do comparador (COMP). Além disto, esta estratégia é simplesmente para reduzir o consumo de hardware, tendo o mesmo impacto em todos os VDs, independentemente da treliça utilizada.

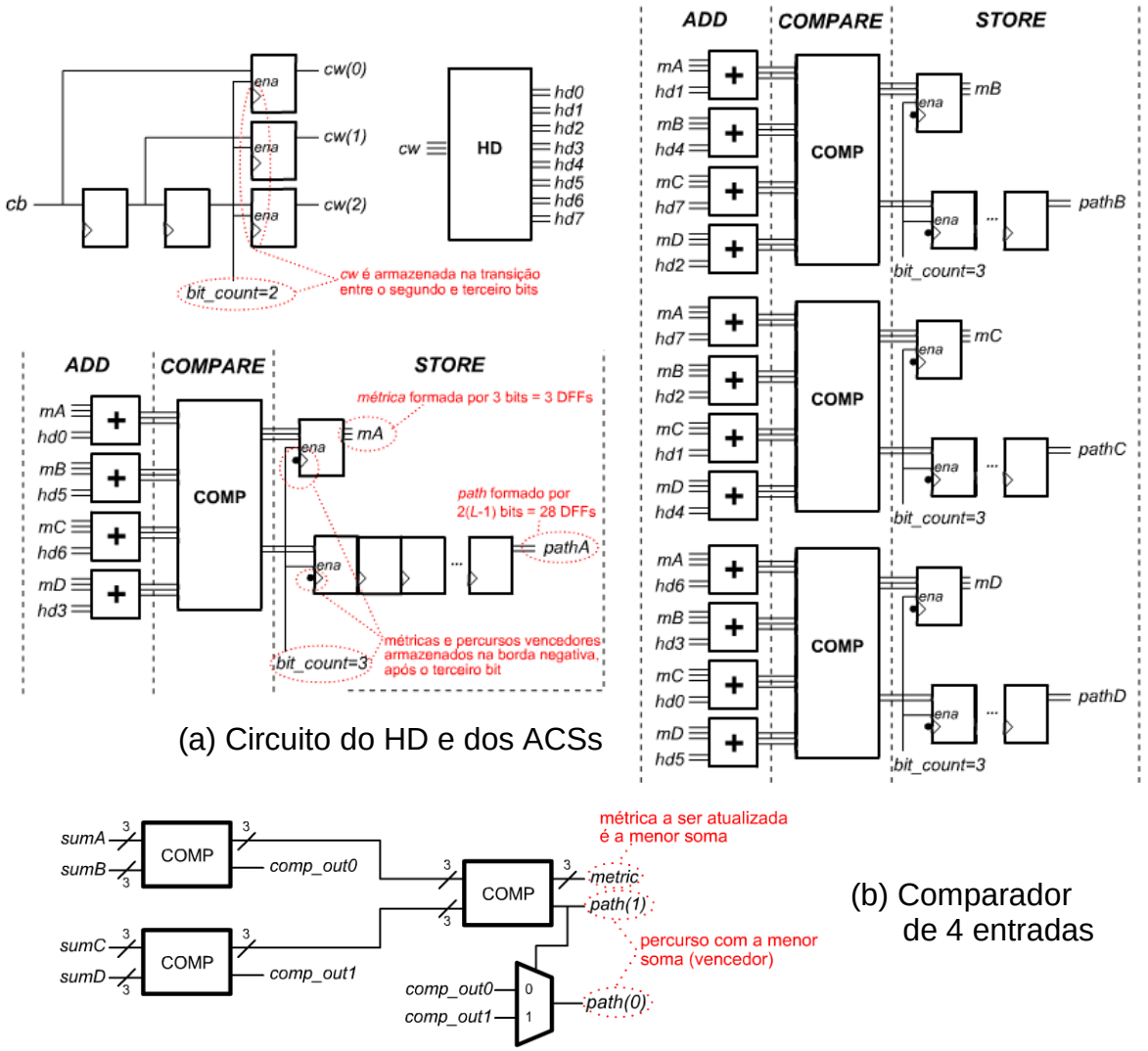


Figura 4.2 – HD e ACS do VD convencional.

O *traceback*, na Figura 4.3, opera na borda de descida de clock quando *bit_count*=1, e é inicializado (o estado atual do TB = “00” e a memória dos percursos vencedores é repassada à memória de TB) quando uma nova mensagem é recebida (*word_count*=1). Como o TB analisa os percursos vencedores de trás para frente, os valores armazenados nos *paths* são invertidos em relação aos armazenados na memória de *traceback*.

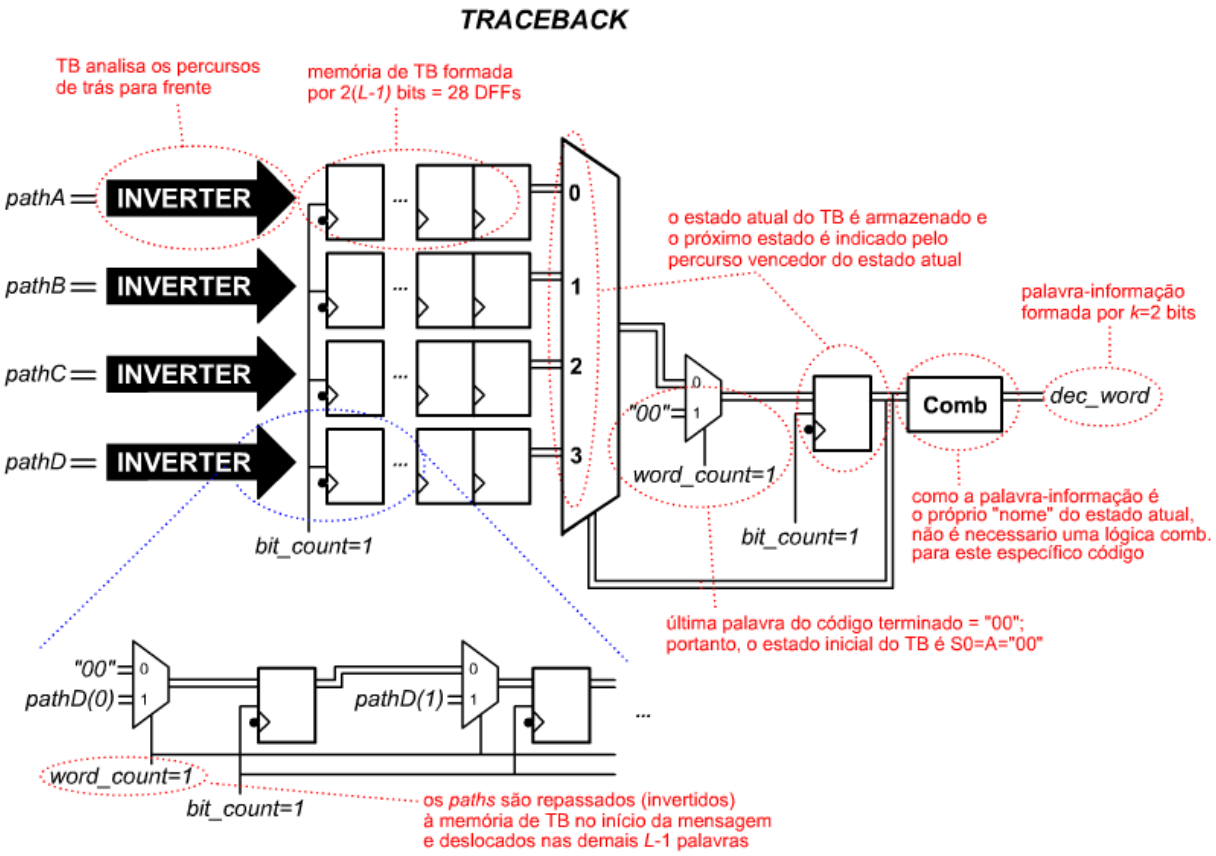


Figura 4.3 – TB do VD convencional.

Como na codificação o próximo estado na treliça é a própria palavra-informação atual, o TB utiliza-se apenas do “nome” do estado atual para deduzir qual foi a palavra-informação necessária para chegar até aquele ponto. Por esta razão, não é necessário um circuito combinacional *Comb* (na Figura 4.3) no TB do código (3,2,2,3).

4.4 O DECODIFICADOR PUNCIONADO

Vários conceitos aplicados ao VD convencional podem ser aproveitados nos VDs puncionado e mínimo, entretanto, o conceito de percursos intermediários aparece somente nestes dois. A treliça puncionada do código (3,2,2,3), vista na Figura 4.4, é formada por k=2 fases, com as quantidades de nós e ramos por nó constantes em ambas as fases. Devido a esta quantidade de fases, o VD puncionado possui um bloco HD que calcula distâncias para os dois primeiros bits da palavra-código, e depois

calcula distâncias para o terceiro bit. O bloco de *traceback* foi omitido nas figuras seguintes pois é o mesmo que foi construído para o VD convencional.

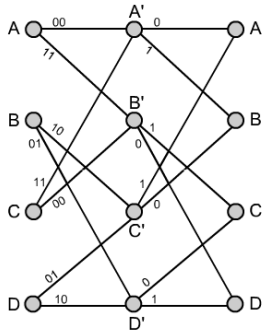


Figura 4.4 – Módulo da treliça puncionada para o código (3,2,2,3).

Como pode ser observado na Figura 4.5, a seguir, a segunda estratégia de processamento – Bloco HD: atualizado apenas quando necessário – foi novamente aplicada ao HD, sendo que agora são necessários dois sub-blocos (HD’ e HD’’) para realizar o cálculo das distancias de Hamming. O HD’ calcula as distâncias entre os primeiros dois bits da palavra-código e o rótulo dos ramos da primeira fase. Já o HD’’ calcula as distâncias entre o terceiro bit da palavra-código e o rótulo dos ramos da segunda fase, sendo que o bit menos significativo (LSB) dos rótulos destes foi puncionado. Com o puncionamento do LSB, resulta em calcular *hd0* para os ramos com os rótulos “00” e “01” na primeira fase, e calcular *hd1* para os ramos com os rótulos “10” e “11” na primeira fase.

Na prática, os sinais dos blocos HD’ e HD’’ do VD puncionado com múltiplos ACSs foram inseridos nos somadores da primeira e segunda fases, respectivamente, sem ser necessário o multiplexador (MUX). Com isto, foi possível utilizar somadores mais simples nos ACSs da segunda fase deste decodificador, realizando somas de 1 bit (distâncias calculadas) com 3 bits (métricas), conforme será visto na Figura 4.6. Além disto, foram construídos comparadores de duas entradas para os VDs puncionados por apenas dois ramos entrarem em cada nó na treliça puncionada, reduzindo em mais de um terço o tamanho deste circuito em relação ao do VD convencional.

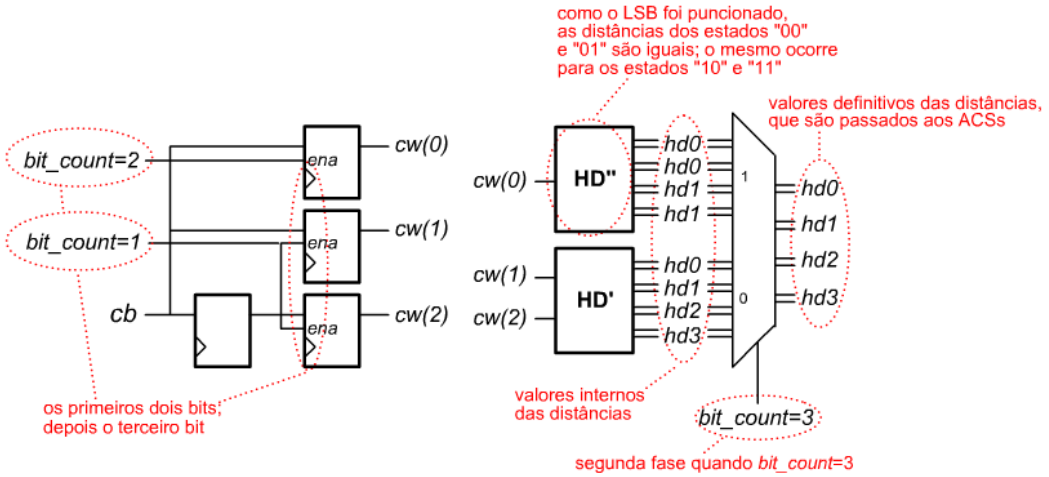


Figura 4.5 – HD do VD puncionado.

- **VD puncionado com múltiplos ACSs**

A Figura 4.6 mostra o circuito do ACS (do nó *A*) do VD puncionado com múltiplos ACSs. A principal diferença entre os dois está em os ACSs da primeira fase armazenarem os percursos intermediários e os ACSs da segunda fase armazenarem os percursos vencedores definitivos. Pode ser observado também que a quantidade de somadores e registradores de métricas foi duplicada, com somadores mais simples sendo utilizados na segunda fase.

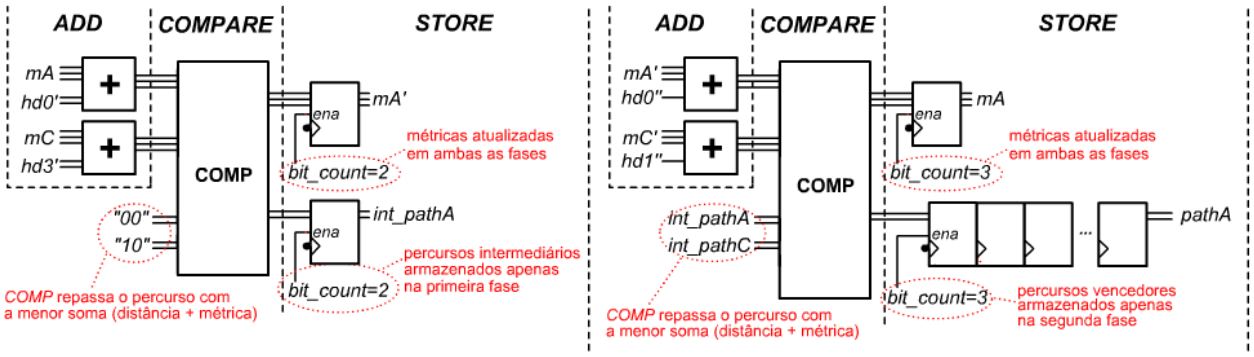


Figura 4.6 – ACS *A* do VD puncionado com múltiplos ACSs.

- **VD puncionado com ACSs únicos**

O circuito do ACS (do nó A) do VD puncionado com ACSs únicos está ilustrado na Figura 4.7. O processamento deste ACS é um pouco diferente do ACS do VD convencional devido à seleção dos sinais. Na primeira fase da treliça, apenas dois estados (ou nós) podem chegar ao estado *A*: o próprio *A* e o *C*. Devido a isto, o *int_pathA* pode assumir apenas os valores *A*=”00” ou *C*=”10”, sendo que o COMP seleciona qual será armazenado de acordo com a menor soma da primeira fase. Na segunda fase, quando *bit_count*=3, a memória de percursos vencedores receberá o valor de *int_pathA* ou de *int_pathC*, sendo que agora o COMP seleciona qual será armazenado de acordo com a menor soma da segunda fase.

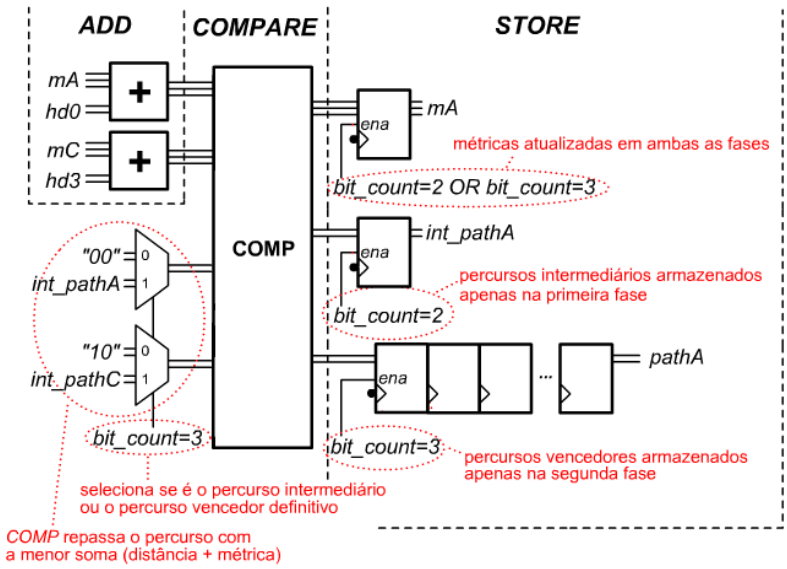


Figura 4.7 – ACS *A* do VD puncionado com ACSs únicos.

4.5 O DECODIFICADOR MÍNIMO

Expandindo os conceitos aplicados aos VDs puncionados, é possível então obter o circuito do VD mínimo. São duas as formas de construí-lo: com múltiplos ACSs e com ACSs únicos. As duas maneiras de construir o decodificador são bastante distintas, sendo a primeira mais simples, porém exigindo uma quantidade maior de

hardware; a segunda maneira implica em um roteamento complexo, além da seleção de vários sinais utilizando MUXs em cada uma das *n*=3 fases da treliça. A Figura 4.8 mostra novamente a estrutura da treliça mínima do código (3,2,2,3).

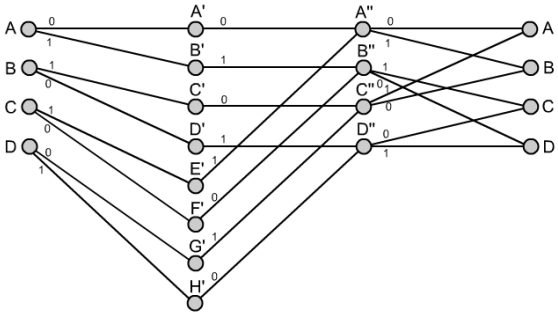


Figura 4.8 – Módulo da treliça mínima para o código (3,2,2,3).

Para compensar os blocos ACS mais complexos, o HD do VD mínimo é o mais simples dos três, consistindo de apenas um DFF para registrar o bit codificado *cb* e de um inversor, o qual gera *hd1*. O HD do decodificador mínimo está exibido na Figura 4.9.

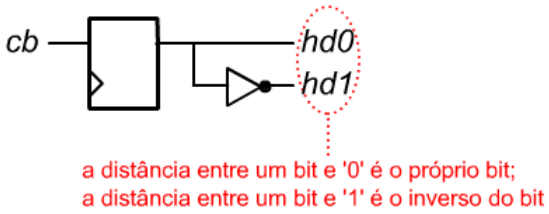


Figura 4.9 – HD do VD mínimo.

Como também ocorreu com os VDs puncionados, foram construídos comparadores de duas entradas para os VDs mínimos por apenas dois ramos entrarem em cada nó de decisão na treliça mínima, reduzindo em mais de um terço o tamanho deste circuito em relação ao do VD convencional.

- **VD mínimo com múltiplos ACSs**

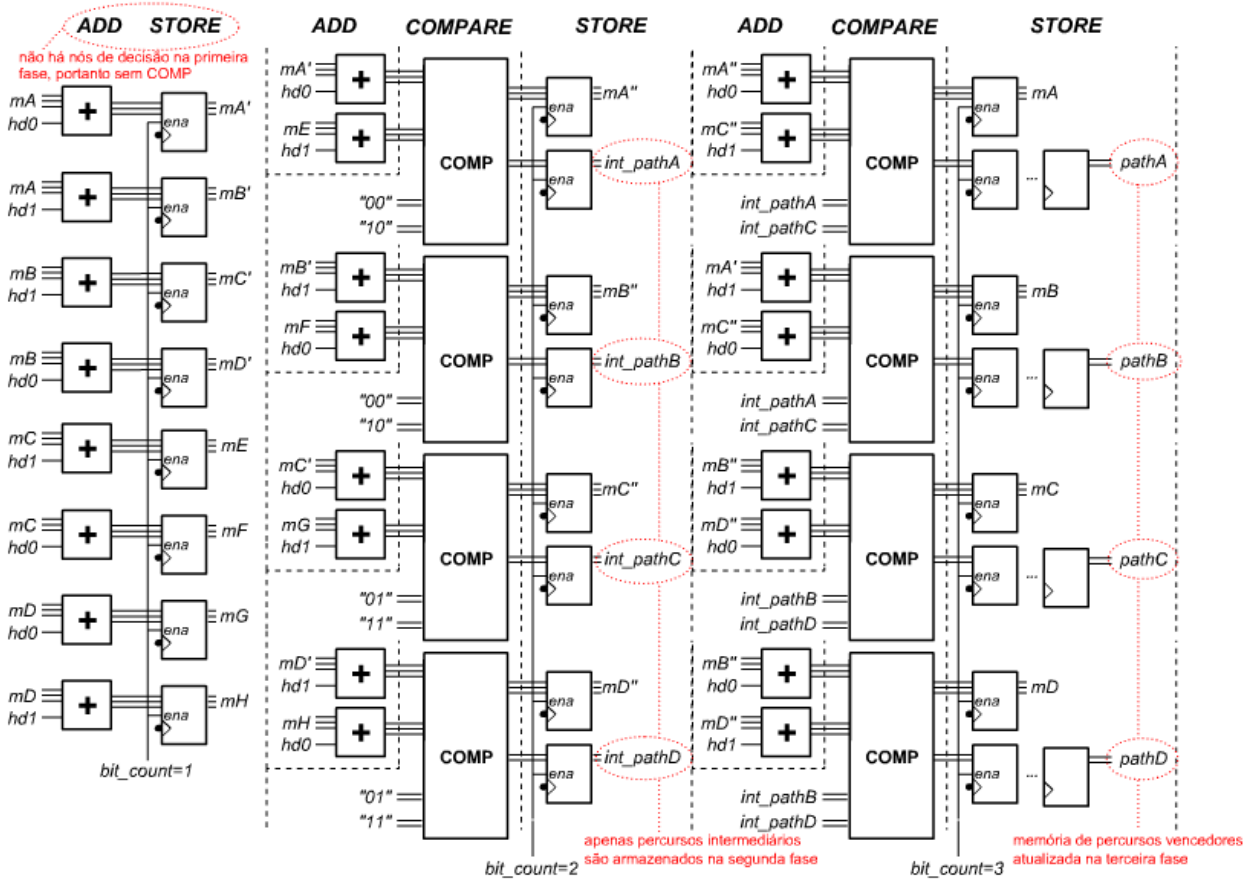


Figura 4.10 – ACS do VD mínimo com múltiplos ACS.

Como não há nós de decisão na primeira fase da treliça mínima, não foram armazenadas informações sobre os percursos nesta fase. Na segunda fase, a Figura 4.10 mostra que os percursos intermediários são os únicos elementos de memória de percursos (4 nós x 2 DFFs), enquanto as memórias definitivas dos percursos vencedores estão presentes apenas na terceira fase (4 nós x 2(*L*-1) DFFs).

- **VD mínimo com ACSs únicos**

O circuito do ACS (do nó *A*) do VD com ACSs únicos está representado na Figura 4.11, agora com uma quantidade reduzida de memória, porém com roteamento

e seleção de sinais mais complexos. Os ACSs dos demais nós estão ilustrados na Figura 4.12.

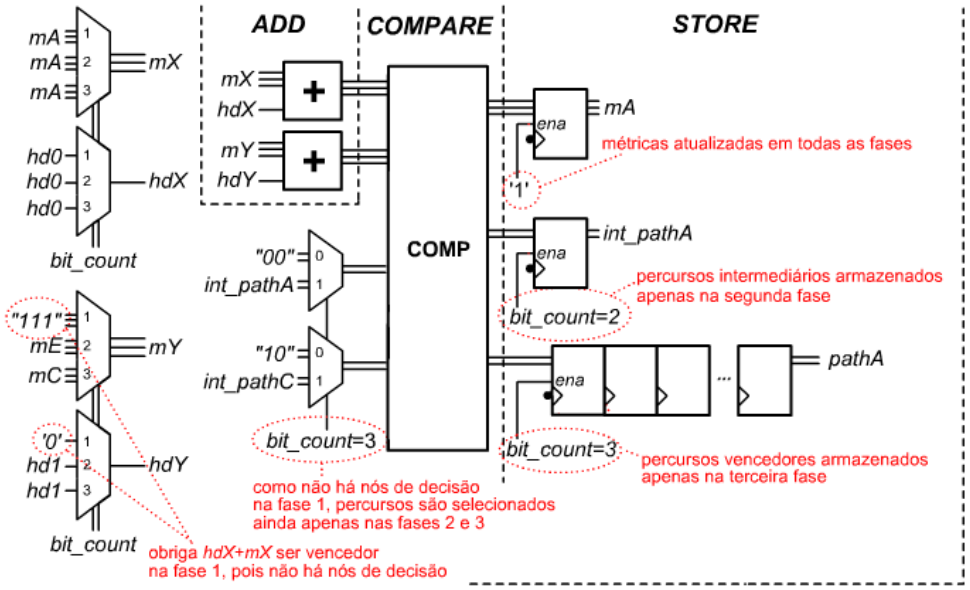


Figura 4.11 – ACS *A* do VD mínimo com ACSs únicos.

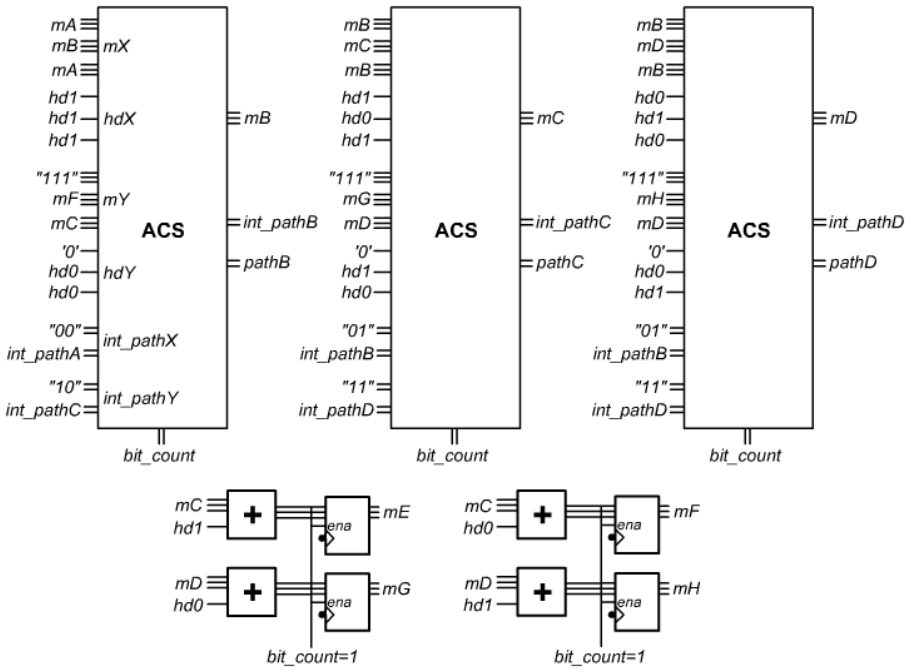


Figura 4.12 – ACSs *B*, *C*, *D*, *E*, *F*, *G* e *H* do VD mínimo com ACSs únicos.

Como são analisados no máximo dois ramos por nó em cada fase da treliça mínima, a estrutura do ACS do VD com ACSs únicos ficou bastante similar ao ACS do VD puncionado com ACSs únicos, mas com três principais diferenças: (i) é necessário armazenar as métricas *mE*, *mF*, *mG* e *mH*; (ii) as somas são de 1 bit (*hd0* ou *hd1*) com 3 bits (métricas); e (iii) a seleção dos sinais de entrada ocorre nas *n*=3 fases do módulo de treliça. A diferença (ii) implica em um somador um pouco mais simples, reduzindo um pouco o consumo de energia e de hardware. Devido à diferença (iii), os ACSs do VD mínimo atualizam as métricas nas três fases, portanto os MUXs dos ACSs devem analisar o valor de *bit_count*, e não apenas uma condição do mesmo (por exemplo, *bit_count*=3 no VD puncionado).

As principais diferenças entre os circuitos dos VDs foram apresentadas, dando enfoque individual aos três blocos dos decodificadores: HD, ACS e TB. Os primeiros dois sofrem modificações consideráveis de acordo com a treliça utilizada. O TB, entretanto, devido à utilização dos percursos intermediários, pôde ser reaproveitado, reduzindo também o seu consumo de hardware e facilitando o processo de decodificação. Com base nos circuitos definidos neste capítulo, foi possível construir os decodificadores em circuitos integrados. Para depois obter o consumo de energia, o consumo de hardware e a freqüência máxima de operação dos VDs, algumas informações específicas de circuitos eletrônicos serão apresentadas no Capítulo 5.

5. MEDIÇÕES E ESTIMATIVAS

Para obter valores de consumo de energia, consumo de hardware e freqüência máxima de operação dos decodificadores, é necessário um aprofundamento sobre como mensurar estes parâmetros. Os fatores que influenciam cada um destes serão apresentados, além de como gerar estimativas que se aproximam de valores práticos. Entretanto, para esclarecer a origem dos fatores, é necessário antes apresentar o elemento básico que torna possível a construção de circuitos digitais complexos em circuitos integrados de área extremamente reduzida: o transistor MOSFET. Além disto, quando os circuitos são construídos com MOSFETs aplicando a lógica CMOS, o consumo de energia – particularmente, a estática – pode tornar-se muito baixo.

5.1 O TRANSISTOR MOSFET E A LÓGICA CMOS

O transistor de efeito de campo – MOSFET (do inglês, *Metal Oxide Semiconductor Field Effect Transistor*), ou simplesmente MOS, popularizou-se no início dos anos 70 quando a Intel lançou seu primeiro microprocessador integrado, o Intel 4004, composto de apenas MOSFETs [25]. O MOSFET é composto de um semicondutor – tipicamente o silício (Si) – que é acrescentado de quantidades controladas de impurezas, de modo a criar um substrato “dopado”. Dependendo do tipo de dopante, o substrato pode ser do tipo *n*, quando acrescentado de elementos do grupo V da tabela periódica (P, As, Sb), ou do tipo *p*, quando acrescentado de elementos do grupo III (B, Al, Ga, In). Substratos do tipo *n* apresentam maior quantidade de elétrons livres, enquanto substratos do tipo *p* apresentam maior quantidade de lacunas livres.

No substrato são criadas ilhas com materiais de tipo contrário aos utilizados na fabricação do substrato. Em substratos do tipo *n* são criadas ilhas *p*+ altamente dopadas (o “+” indica que a quantidade de dopante é muito maior que a quantidade usada no substrato); em substratos do tipo *p* são criadas ilhas *n*+ altamente dopadas. Nos transistores, estas ilhas são chamadas de fonte e dreno, em inglês *source* e *drain*, respectivamente. Entre estas ilhas e paralela ao substrato existe uma lâmina,

conhecida como porta (do inglês, *gate*), que controla a passagem ou não de corrente entre a fonte e o dreno. A região de substrato abaixo da porta, entre a fonte e o dreno, é chamada de canal, pois é neste local que elétrons (canal *n*) ou lacunas (canal *p*) são induzidos de modo a criar um caminho para a passagem de corrente entre as duas ilhas. A Figura 5.1 ilustra um corte vertical de um transistor nMOS (substrato *p*, ilhas *n*+ e canal de elétrons) e de um pMOS (substrato *n*, ilhas *p*+ e canal de lacunas).

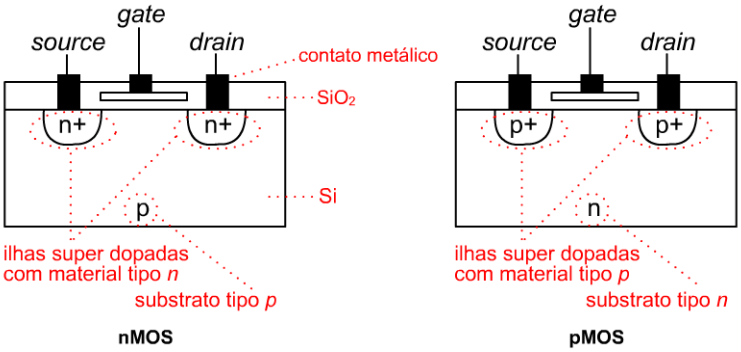


Figura 5.1 – Corte vertical de transistores nMOS e pMOS.

O tamanho de um transistor MOSFET é definido pelas dimensões de seu canal, particularmente a largura *W* e o comprimento *L* do mesmo. A Figura 5.2 ilustra o corte horizontal de um MOSFET, onde estas dimensões estão especificadas. A tecnologia MOS define qual o tamanho dos transistores construídos e é usualmente definida pelo parâmetro λ, o qual é metade da menor dimensão de tecnologia. A menor dimensão de canal é geralmente definida pela razão de largura por comprimento do canal e é definida como $W/L = 3\lambda/2\lambda$. Para uma tecnologia MOS de 90nm, por exemplo, tem-se que λ=45nm e W/L = 135nm/90nm. Estes parâmetros são importantes na construção dos transistores e têm impacto direto no consumo de energia.

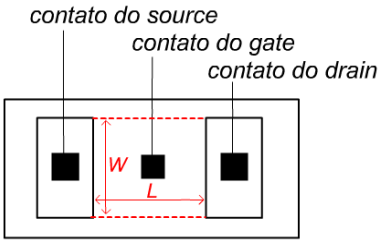


Figura 5.2 – Largura (*W*) e comprimento (*L*) do canal de um MOSFET.

Quando utilizados em circuitos digitais, a operação dos transistores MOSFET pode ser resumida de uma forma simples [26]:

- O transistor nMOS atua como uma chave fechada (ligado, conduzindo) quando sua tensão de entrada (na porta) está em nível alto (nível lógico ‘1’) e como uma chave aberta quando sua tensão de entrada está em nível baixo (nível lógico ‘0’).
- O transistor pMOS atua como uma chave fechada (ligado, conduzindo) quando sua tensão de entrada (na porta) está em nível baixo (nível lógico ‘0’) e como uma chave aberta quando sua tensão de entrada está em nível alto (nível lógico ‘1’).

A análise do comportamento dos transistores está baseada nestas considerações, sendo que detalhes mais profundos de como elétrons e lacunas são induzidos para formar os canais não serão abordados. Basta ter em mente que para que os MOSFETs atuem desta maneira, o corpo do tipo *p* do nMOS deve ser aterrado (nível de tensão baixo, nível lógico ‘0’) e o corpo do tipo *n* do pMOS deve ser ligado em V_{DD} (nível de tensão elevado, nível lógico ‘1’). Os símbolos dos transistores nMOS e pMOS, com os pinos identificados, estão ilustrados na Figura 5.3.

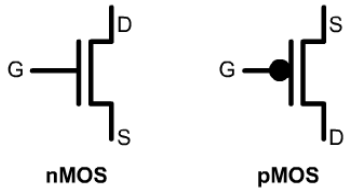


Figura 5.3 – Símbolos dos transistores nMOS e pMOS.

Dentre as várias configurações dos transistores MOSFET, ou seja, as maneiras de interligá-los, a CMOS (do inglês, *complementary* MOS) é a mais utilizada. Na lógica CMOS, para cada transistor nMOS existe o seu complementar pMOS. Com isto, além de não serem necessários resistores nos circuitos (os quais consomem muita área de Si), o consumo de energia fica extremamente reduzido, principalmente em termos de consumo estático, ou seja, quando os transistores não estão chaveando [26].

5.2 CONSUMO DE ENERGIA

Para demonstrar estas particularidades da lógica CMOS, uma explicação da operação do inversor digital é apresentada a seguir.

O inversor CMOS é o circuito lógico mais básico e é composto de apenas 2 transistores (1 nMOS e 1 pMOS). A Figura 5.4 ilustra o circuito do inversor, onde pode ser observado que o transistor pMOS (superior) está com a fonte (*source*) conectada ao V_{DD} (nível lógico ‘1’, indicado pela seta) e o transistor nMOS (inferior) com a fonte conectada ao terra (nível lógico ‘0’, indicado pela barra). Desta maneira, quando o sinal de entrada (nas portas dos transistores) for *a*=‘0’, o canal do pMOS é formado e este conduz (atuando como uma chave fechada), de modo que a tensão no dreno é igual à da fonte. Já o nMOS nesta situação atua como uma chave aberta e não afeta a tensão de seu dreno. Desta maneira, a tensão de saída, medida nos drenos, é praticamente a tensão da fonte do pMOS, ou seja, *a*’=‘1’. Quando o contrário ocorre, ou seja, *a*=‘1’, o pMOS fica aberto e o nMOS conduz, de modo que a tensão de saída é a tensão da fonte do nMOS, ou seja, *a*’=‘0’. Com isto, o sinal de saída do circuito é o inverso do sinal de entrada.

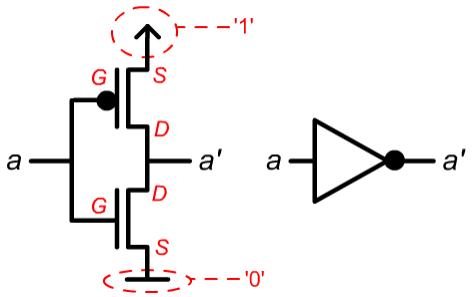


Figura 5.4 – Inversor CMOS.

Com o circuito do inversor CMOS em mente, pode-se definir dois tipos de energia: estática (P_{est}) e dinâmica (P_{din}). Como foi mencionado anteriormente, circuitos empregando lógica CMOS possuem consumo de energia estática muito reduzido, devido a um dos transistores – ora o pMOS, ora o nMOS – estar sempre desativado enquanto não há alteração de estado na entrada. Apesar de o consumo de energia estática apresentar valores consideravelmente inferiores aos da energia dinâmica, na

medida em que a tecnologia de fabricação de CIs evolui e os transistores tornam-se cada vez menores, as correntes de fuga destes começam a pesar no consumo total dos circuitos.

Constituinte da maior parte do consumo de energia em circuitos com lógica CMOS, o consumo de energia dinâmica pode ser dividido em duas: consumo de curto-circuito (P_{cc}) e consumo de capacitância (P_{cap}) [26], de modo que

$$P_{din} = P_{cc} + P_{cap} \tag{5.1}$$

O consumo de curto-circuito P_{cc} ocorre quando há uma mudança de estado dos transistores e, por um breve instante, ambos os transistores – pMOS e nMOS – estão ativos. A Figura 5.5 ilustra este evento, onde o canal dos transistores está representado como uma resistência neste instante. Para obter o valor de P_{cc} é necessário integrar a corrente de curto-circuito I_{cc} e multiplicar pela tensão de alimentação do circuito V_{DD}. Vale destacar que há consumo de energia dinâmica por curto-circuito tanto na transição de ‘0’ para ‘1’ quanto de ‘1’ para ‘0’.

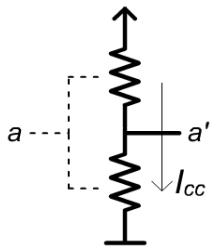


Figura 5.5 – Consumo de curto-circuito (energia dinâmica).

Quando ocorre uma transição de ‘0’ para ‘1’ em um nó do circuito, é despendida certa quantidade de cargas para carregar a capacitância equivalente deste nó. Na Figura 5.6, por exemplo, pode ser observada a capacitância equivalente C_D “vista” pelos drenos dos transistores. Enquanto *a*=‘1’, o nMOS conduz, de modo que a saída fica em *a*’=‘0’ e o capacitor C_D permanece descarregado. Quando há uma transição de *a*=‘1’ para *a*=‘0’, o pMOS passa a conduzir e o nível de tensão nos drenos começa a subir na medida que C_D vai carregando, até atingir o nível máximo de tensão V_{DD}.

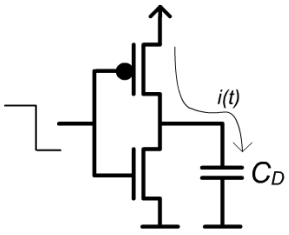


Figura 5.6 – Consumo de capacitância (energia dinâmica).

Portanto, para obter P_{cap} é necessário integrar o produto $i(t).V_{DD}$, onde $i(t)$ representa a corrente de carga de C_D. Sendo que a carga necessária para carregar completamente C_D é $Q_{cap}=C_D.V_{DD}$ e considerando ciclos de carga e descarga completos ocorrendo a cada T segundos (freqüência $f=1/T$), é possível concluir que [26]:

$$P_{cap} = C_D \, . \, V_{DD}^{\,2} \, . \, f \qquad (5.2)$$

Quando a transição dos sinais de entrada dos transistores é muito rápida (vertical) ou quando o capacitor C_D é grande (na ordem de pF), P_{cap} predomina sobre P_{cc} no consumo de energia dinâmica. Sendo também que o consumo de energia dinâmica é consideravelmente superior ao consumo de energia estática, pode-se concluir que o consumo de energia de um circuito é proporcional:

a. ao quadrado da tensão de alimentação;
b. à capacitância equivalente do circuito;
c. à freqüência de operação do circuito.

5.3 CONSUMO DE HARDWARE

O consumo de hardware de um circuito advém da quantidade e do tamanho dos transistores necessários para construí-lo, além do roteamento, em alguns casos. Os componentes de um circuito podem ser construídos de diversas maneiras, com o tipo de lógica definindo se são necessários elementos de memória para realizar a função especifica do circuito. Como será visto no Capítulo 6, a quantidade de hardware necessária para construir circuitos em *Programmable Logic Devices* (em inglês,

dispositivos de lógica programável), ou PLDs, é comumente dividida em funções, ou elementos, de lógica combinacional e em registradores. Registradores equivalem a elementos de memória e são circuitos de lógica seqüencial. Já em *Application Specific Integrated Circuits* (em inglês, circuitos integrados de aplicação especifica), ou ASICs, o consumo de hardware é traduzido diretamente pela quantidade de transistores utilizados na construção dos circuitos.

A definição básica de circuito combinacional é todo circuito cujos valores de saída dependem exclusivamente dos valores de entrada atuais [26]. Desta maneira, são circuitos que não possuem memória e podem ser criados a partir de portas lógicas e chaves. Exemplos de circuitos combinacionais são as próprias portas lógicas, multiplexador, somador/subtrator e multiplicador/divisor. Dentre estes citados, as portas lógicas e multiplexadores se enquadram em circuitos combinacionais lógicos, enquanto que somadores/subtratores e multiplicadores/divisores se enquadram em circuitos combinacionais aritméticos, por realizarem operações aritméticas. O modelo genérico de um circuito combinacional pode ser visto na Figura 5.7(a). O consumo de circuitos combinacionais depende da complexidade do circuito sendo implementado, e é mensurado geralmente em termos de elementos ou funções nos PLDs.

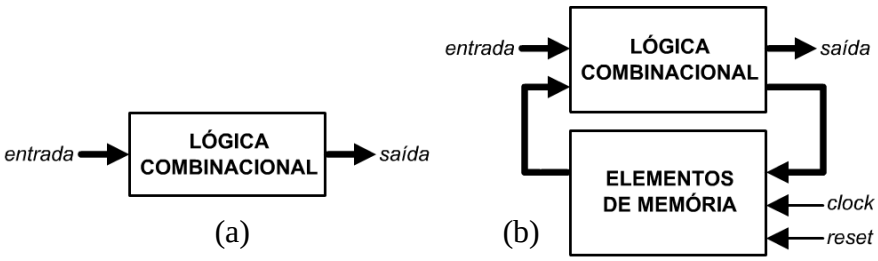


Figura 5.7 – Circuitos com lógica (a) combinacional e (b) seqüencial.

Em contrapartida, circuitos seqüenciais são aqueles cujas saídas dependem de algum valor anterior do sistema, portanto necessitam de memória. Além de elementos de memória, circuitos seqüenciais necessitam de um sinal de *clock* que controla o andamento do sistema, usualmente junto com um sinal de *reset*, que reinicializa o sistema para um estado pré-definido. Apesar de circuitos seqüenciais possuírem memória, eles quase sempre possuem alguma lógica combinacional. Conforme visto na Figura 5.7(b), circuitos seqüenciais podem ser modelados como um bloco

combinacional e um bloco seqüencial (contendo elementos de memória e suas interligações), com uma realimentação do segundo para o primeiro. Dentre os tipos de elementos de memória que constituem os circuitos seqüenciais, os flip-flops do tipo D (DFF) destacam-se entre os mais utilizados em circuitos integrados, inclusive nos dispositivos de lógica programável [26]. Portanto, para cada bit de informação que necessita ser armazenado nos circuitos dos decodificadores é consumido um DFF. Convém-se destacar, porém, que hoje em dia, com a redução do tamanho dos transistores, o consumo de hardware não é um fator tão significativo na construção de circuitos integrados quanto era até a alguns anos.

5.4 FREQÜÊNCIA MÁXIMA DE OPERAÇÃO

A freqüência máxima de operação de um circuito é definida pelo tempo de propagação e estabilização dos níveis de tensão (ou corrente) dos circuitos. Circuitos digitais, os quais operam com apenas dois níveis de tensão, sofrem impacto, principalmente, pelo tempo de propagação. O tempo de propagação é o tempo que demora para que um estímulo de entrada se propague até a saída do circuito, e é decorrente da quantidade de circuito que há entre os nós de entrada e saída do sinal. A Figura 5.8 demonstra este tipo de atraso.

Desta maneira, o percurso dentro de um circuito que possui o maior tempo de propagação é o que define o menor tempo entre amostras T_{min}. A freqüência máxima de operação f_{max} é definida, portanto, como o inverso do menor período de amostra, ou seja, $f_{max}=1/T_{min}$.

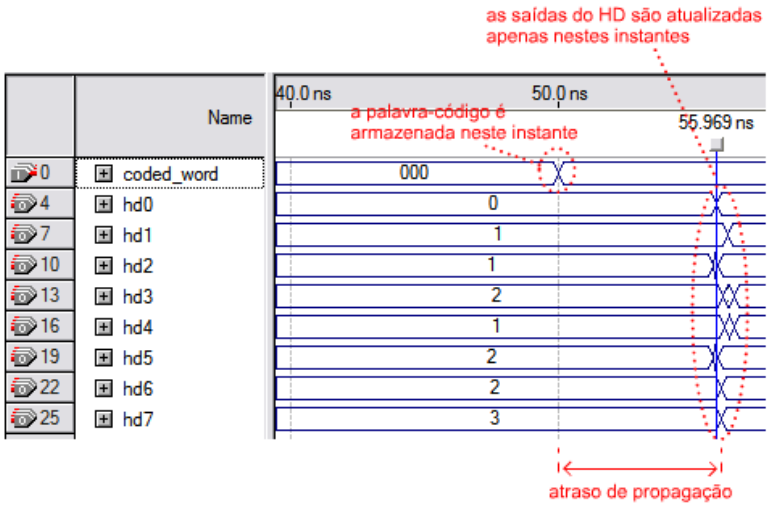


Figura 5.8 – Atraso de propagação.

No caso dos decodificadores, aqueles que apresentam uma freqüência máxima mais elevada podem operar com um maior *throughput*, ou seja, podem decodificar mais informação que decodificadores com menor freqüência máxima, no mesmo intervalo de tempo. A implicância, porém, de um circuito operar em uma freqüência maior é diretamente no consumo de energia, como foi visto na eq. (5.2).

De acordo com esta mesma equação, parecer-se-ia trivial simplesmente diminuir a tensão de alimentação do circuito para que o consumo de energia fosse reduzido. Porém, reduzindo a tensão dos transistores faz com que o tempo de carga dos nós do circuito torne-se maior, reduzindo a freqüência máxima de operação do mesmo. Há, portanto, uma medida de desempenho de circuitos que leva em consideração a velocidade do circuito e o seu consumo de energia, conhecida como *produto potência-atraso*. Desta maneira, a tensão de alimentação dos circuitos V_{DD} não pode ser reduzida indefinidamente, sendo que outros fatores intrínsecos dos transistores devem ser levados em consideração para que o produto potência-atraso seja otimizado [26].

Com estes detalhes em mente, é possível agora analisar os circuitos dos decodificadores individualmente. Os circuitos foram construídos em PLDs e em ASICs, sendo que em ambos o consumo de hardware é facilmente obtido. Já a freqüência máxima de operação é possível ser obtida através do software de desenvolvimento dos

circuitos dos PLDs, porém torna-se inviável analisá-la em ASICs pois o software utilizado para o desenvolvimento dos mesmos não possui esta funcionalidade e uma análise empírica torna-se dispendiosa. O consumo de energia é ainda mais complexo. O software de desenvolvimento dos PLDs dispõe de uma funcionalidade que estima o consumo de energia (estática e dinâmica) baseando-se em uma simulação do decodificador, porém apenas de forma aproximada. Já nos ASICs, os valores de consumo de energia foram obtidos através da análise da corrente consumida para realizar a decodificação de uma seqüência de palavras-código inseridas no sistema. Nos Capítulos 6 e 7 estão detalhados os resultados obtidos para os VDs construídos em PLDs e em ASICs, respectivamente.

6. IMPLEMENTAÇÃO DOS DECODIFICADORES EM PLDs

Para verificar o consumo de energia e de hardware, além da freqüência máxima de operação dos decodificadores, estes foram construídos em CIs de hardware programável, ou PLDs (do inglês, *Programmable Logic Devices*). Devido ao custo consideravelmente baixo e à simples maneira de programação, onde o desenvolvedor pode criar circuitos digitais complexos através de uma linguagem de programação, PLDs tornaram-se extremamente populares e estão atualmente presentes em um grande número de sistemas digitais comerciais. Através da linguagem de descrição VHDL é possível criar inúmeros circuitos digitais, incluindo decodificadores de Viterbi.

Neste capítulo, uma breve história sobre como surgiram os PLDs é apresentada, indicando as diferenças básicas entre CPLDs e FPGAs, além da linguagem de descrição de hardware VHDL, que torna possível a configuração de circuitos nos PLDs. Os circuitos dos VDs foram desenvolvidos utilizando a linguagem VHDL e os resultados de consumo de energia, consumo de hardware e freqüência máxima de operação foram analisados no Quartus II [27], software da Altera.

6.1 *PROGRAMMABLE LOGIC DEVICES* E VHDL

PLDs são circuitos integrados capazes de ser programados a nível de hardware. Introduzidos praticamente ao mesmo tempo nos anos 70, os primeiros CIs desta categoria foram os dispositivos *Programmable Array Logic* – PALs (em inglês, lógica de arranjo programável), desenvolvidos pela Monolithic Memories, e os dispositivos *Programmable Logic Array* - PLAs (em inglês, arranjo de lógica programável), desenvolvidos pela Signetics, os quais eram capazes de implementar apenas simples circuitos combinacionais. Logo após a introdução destes componentes, PLDs com registradores em seus terminais de saída foram criados, possibilitando a implementação de circuitos seqüenciais. No início dos anos 80, a Lattice Semiconductor lançou os *Generic PALs* – GALs (em inglês, PALs genéricos), onde portas lógicas e multiplexadores foram acrescentados aos registradores de saída, formando assim as macro-células. Todos estes circuitos integrados (PALs, PLAs e GALs) são conhecidos

hoje como *Simple PLDs* – SPLDs (em inglês, PLDs simples). Em meados dos anos 80, duas novas estruturas de PLDs foram criadas: os *Complex PLDs* - CPLDs (em inglês, PLDs complexos) e as *Field-Programmable Gate Arrays* - FPGAs (em inglês, arranjos de portas programáveis no campo).

CPLDs são componentes compostos de vários GALs, os quais são interligados através de sofisticados mecanismos de roteamento e são muito utilizados em aplicações de alta densidade, alta velocidade e baixo custo. Uma característica dos CPLDs é que sua configuração é não-volátil.

FPGAs diferem de CPLDs em vários aspectos, principalmente em estrutura (formadas basicamente por uma matriz de portas lógicas), tecnologia e custo. Devido à sua estrutura diferente e quantidade superior de registradores em relação aos CPLDs, as FPGAs são mais utilizadas em projetos complexos, que requerem alta velocidade de processamento, como sistemas de telecomunicações. FPGAs também diferem dos CPLDs em termos de memória de configuração, sendo neste caso volátil.

Apesar dos diferentes arranjos de lógica combinacional e da diferença na quantidade de registradores, CPLDs e FPGAs possuem algumas similaridades cruciais para a construção dos decodificadores. Em PLDs, o sinal de clock é roteado estrategicamente dentro dos CIs. A linha de clock é distribuída para os setores do componente visando reduzir as perdas do sinal de clock e fazer com que o atraso deste sinal, chegando nos diversos setores, seja mínimo e uniforme. Devido a isto, não convém aplicar a estratégia de *clock gating*, apresentada no Capítulo 4, pois se o sinal de clock passar por portas ANDs, por exemplo, o roteamento do clock será alterado de sua linha original, fazendo com que possam surgir diferenças no instante em que DFFs registram as informações. Com estas diferenças, informações podem ser incorretamente processadas e, conseqüentemente, armazenadas com valores incorretos.

PLDs possuem um grande número de células lógicas, as quais são compostas basicamente de portas lógicas, MUXs e registradores. O conteúdo, a disposição e a maneira de interligar estas células lógicas – o arranjo – é o que difere CPLDs de FPGAs. Apesar disto, em ambos os tipos de PLDs, os únicos registradores encontrados são *data* flip-flops (DFFs), com pinos de *enable*, *reset* e *preset*. Os circuitos dos

componentes dos VDs foram construídos baseando-se nos circuitos apresentados no Capìtulo 4. Entretanto, devido à disposição e disponibilidade das células lógicas dentro dos PLDs, nem sempre o circuito elaborado em VHDL é o efetivamente construído. O Quartus II, compilador do código VHDL e gerador do arquivo de programação do PLD, procura construir os circuitos da melhor maneira possível em termos de roteamento mais curto, de mínima área utilizada e de consumo de energia reduzido para o PLD, resultando em equações lógicas nem sempre iguais – porém, sempre compatíveis – com o que foi implementado em VHDL. Para este estudo, o Quartus II foi também utilizado para verificar o funcionamento dos decodificadores através de simulações e o consumo de energia através da função PowerPlay Power Analyzer [28].

Very High Speed Integrated Circuits Hardware Description Language – VHDL (em inglês, linguagem de descrição do hardware de circuitos integrados de velocidade altíssima) foi utilizada para o desenvolvimento dos VDs. A linguagem VHDL teve seu desenvolvimento iniciado em 1981, sob o patrocínio do Departamento de Defesa dos Estados Unidos, devido à necessidade de se ter uma linguagem padrão para descrever a estrutura e funções de CIs. Em 1987, o IEEE aprovou o primeiro padrão desta linguagem através da Norma IEEE 1076-87 [29]. Quatro revisões já foram realizadas desde 1987, e o padrão atual é a Norma IEEE 1076-2008.

6.2 CONSIDERAÇÕES SOBRE OS VDs

A seguir estão apresentados os resultados obtidos para os VDs. Vale lembrar que foram construídos dois contadores: o contador de bits e o contador de palavras. Como o código (3,2,2,3) possui n=3, o contador de bits conta de 1 a 3, iniciando sempre em 0; para isto foi criado um contador de 2 bits. Como foi utilizado um código formado por 15 palavras-códigos, foi construído um contador de 4 bits para realizar esta função. Os VDs foram analisados em 1 CPLD, da família MAX II, e em 4 FPGAs, das famílias Cyclone II, Cyclone III, Stratix II e Stratix III, todos decodificando uma seqüência de 500 mensagens (de 45 bits cada) aleatórias e operando com um clock de 10MHz.

6.3 RESULTADOS

Os resultados dos testes realizados nos PLDs estão separados por consumo de energia, consumo de hardware e freqüência máxima de operação.

- **Consumo de energia**

Utilizando do PowerPlay Power Analyzer, no Quartus II, foi possível obter uma estimativa da potência dissipada (conseqüentemente, consumo de energia) nos PLDs. O consumo nos PLDs é dividido em potência dinâmica, estática e nos pinos (I/Os). Como a atividade dos sinais nos pinos dos decodificadores é a mesma, independente da treliça utilizada para construí-lo, este consumo foi deixado de fora da relação. Desta maneira, foram apresentadas as potências dissipadas (em mW) de acordo com o dispositivo utilizado, segregando as potências dinâmica, estática e total dissipadas (incluindo consumo dos I/Os). A coluna da direita mostra a relação da potência dinâmica do VD com a potência dinâmica do VD convencional.

O Quartus II possui a função de otimização de energia durante a compilação do código VHDL, de modo que ele constrói o circuito visando redução de consumo de energia. Os resultados de consumo apresentados na Tabela 6.1 são referentes a decodificadores compilados sem esta função. Entretanto, os circuitos dos decodificadores foram também analisados com esta função habilitada, sendo que a redução de consumo de energia não mostrou alterações significativas frente os resultados já obtidos sem a otimização. Inclusive, em alguns casos, a otimização gerou circuitos cujo consumo de energia era superior ao consumo apresentado pelo circuito compilado sem otimização, razão pela qual estes resultados não constam na tabela seguinte.

Família	Dispositivo	Viterbi *decoder*	Potência Dissipada (mW)			
			Dinâmica	Estática	Total	%
MAX II	EPM570F100C4	Convencional	**12,83**	39,62	59,55	**100,0**
		Puncionado múltiplos ACSs	**9,35**	39,62	56,07	**72,9**
		Puncionado ACSs únicos	**10,65**	39,62	57,37	**83,0**
		Mínimo múltiplos ACSs	**9,09**	39,62	55,81	**70,8**
		Mínimo ACSs únicos	**12,12**	39,62	58,84	**94,5**
Cyclone II	EP2C5AF256A7	Convencional	**1,04**	18,00	31,19	**100,0**
		Puncionado múltiplos ACSs	**0,84**	18,00	30,86	**80,8**
		Puncionado ACSs únicos	**0,98**	18,00	30,98	**94,2**
		Mínimo múltiplos ACSs	**0,78**	18,00	30,79	**75,0**
		Mínimo ACSs únicos	**1,05**	18,00	31,03	**101,0**
Cyclone III	EP3C5E144A7	Convencional	**0,66**	46,14	56,36	**100,0**
		Puncionado múltiplos ACSs	**0,63**	46,12	56,35	**95,5**
		Puncionado ACSs únicos	**0,64**	46,12	56,34	**97,0**
		Mínimo múltiplos ACSs	**0,52**	46,12	56,22	**78,8**
		Mínimo ACSs únicos	**0,67**	46,12	56,37	**101,5**
Stratix II	EP2S15F484C3	Convencional	**1,83**	303,00	326,16	**100,0**
		Puncionado múltiplos ACSs	**1,45**	302,99	325,49	**79,2**
		Puncionado ACSs únicos	**1,92**	302,99	325,95	**104,9**
		Mínimo múltiplos ACSs	**1,68**	302,99	325,71	**91,8**
		Mínimo ACSs únicos	**2,05**	303,00	326,10	**112,0**
Stratix III	EP3SE50F484C2	Convencional	**1,01**	398,09	435,54	**100,0**
		Puncionado múltiplos ACSs	**0,80**	397,92	426,12	**79,2**
		Puncionado ACSs únicos	**0,80**	397,54	425,75	**79,2**
		Mínimo múltiplos ACSs	**0,81**	398,00	426,21	**80,2**
		Mínimo ACSs únicos	**1,14**	397,70	426,25	**112,9**

Tabela 6.1 – Potência dissipada (consumo de energia) nos VDs.

Conforme a Tabela 6.1, ao contrário do exposto no Capítulo 5, onde o consumo de energia estática em circuitos com lógica CMOS é praticamente desprezível em relação ao de energia dinâmica, em PLDs este consumo tende a ser muito elevado. Isto porque a potência estática dissipada (consumo de energia estática) nos PLDs fornecido pelo PowerPlay Power Analyzer é, na realidade, a potência dissipada para manter o dispositivo ligado e operando, e é praticamente independente da atividade interna dos sinais. Por esta razão, pode ser observado que há pouca variação deste consumo entre os PLDs. Contudo, vale destacar que em todos os circuitos implementados a potência estática dissipada é muito superior à dinâmica, sendo que a potência estática representa pelo menos 57% da potência total, no caso do PLD da Cyclone II. Já a potência dinâmica dissipada representa no máximo 20% da potência total, sendo que a potência dinâmica dissipada nas FPGAs gira em torno de 1% da potência total devido ao circuito do VD utilizar, proporcionalmente, poucas células lógicas nestes dispositivos.

Como o consumo de energia estática independe praticamente do circuito implementado, o real consumo a ser analisado para descobrir qual decodificador consome menos é o consumo de energia dinâmica. Considerando isto, pode-se observar dois comportamentos dos decodificadores na Tabela 6.1:

(i) Na maioria dos PLDs utilizados, os VDs puncionados e o VD mínimo com múltiplos ACSs apresentaram consumos próximos entre si e inferiores aos dos demais VDs.

(ii) Curiosamente, o VD mínimo com ACSs únicos não apresentou o melhor desempenho e, em alguns casos, consumiu mais energia dinâmica que o VD convencional.

Estes resultados podem ser originados pelo fato dos decodificadores com treliça puncionada e com treliça mínima realizarem comparações de somas (distâncias calculadas + métricas) mais simples, pois apenas dois ramos entram em cada nó por fase. Entretanto, esta simplificação pode ser contra-argumentada pela menor quantidade de atualizações das métricas que ocorrem no VD convencional, em relação aos outros VDs, além do VD convencional não necessitar dos percursos intermediários. Já, analisando os VDs mínimos, seus consumos deveriam estar mais próximos devido

aos dois realizarem as mesmas operações. Com isto em mente, duas possíveis explicações para os resultados mais favoráveis para os VDs puncionados e o VD mínimo com múltiplos ACSs seriam:

(i) A maneira como os circuitos – principalmente combinacionais – são construídos nos PLDs pode impactar no consumo. Considerando que a principal diferença dos circuitos combinacionais entre os VDs são os blocos HD e os somadores e comparadores dos ACSs, o Quartus II pode implementá-los de uma maneira favorável aos VDs puncionados e ao mínimo com múltiplos ACSs.

(ii) Os dois VDs mínimos apresentaram consumos distintos possivelmente pela quantidade e complexidade superiores de roteamento presentes no VD mínimo com ACSs únicos. Trilhas e conexões internas nos circuitos aumentam a capacitância equivalente do circuito, fazendo com que mais carga seja demandada para carregar os nós, aumentando o consumo do circuito.

Apesar dos resultados obtidos e das hipóteses levantadas, o consumo de energia dinâmica ainda é pouco se comparado com o consumo total do circuito. Por esta razão, foi pesquisada a confiabilidade dos valores obtidos pelo PowerPlay Power Analyzer, a qual foi obtida através de contato por e-mail com a equipe de suporte da Altera. Estes informaram que há uma tolerância de ±15% para o consumo total de energia obtido pelo Quartus II. Entretanto, após inúmeras tentativas em obter informações sobre a precisão do consumo dinâmico, a equipe de suporte da Altera concluiu que os valores obtidos pelo software servem apenas para estimar o consumo de energia do circuito, de modo que o consumo real de energia deve ser medido na prática.

Apesar de ter sido possível tentar analisar o consumo real de um PLD, pela dificuldade de obter o consumo de energia de apenas o PLD – tendo em vista que este pertencia a uma placa de desenvolvimento e outros periféricos também estavam ligados e consumindo energia –, decidiu-se examinar o consumo de energia dos VDs em outro software, o PSpice. Além disto, como pode ser observado na Tabela 6.1, a relação do consumo de energia depende muito do dispositivo utilizado, sendo que os percentuais

da coluna da direita não seguem um padrão bem definido, dando indícios de resultados difíceis de serem precisamente obtidos. Os decodificadores foram, portanto, construídos a nível de transistor através da linguagem SPICE e seus resultados serão apresentados no Capítulo 7.

Vale destacar que os circuitos foram também construídos com a estratégia de *clock gating*, porém em todos os VDs o consumo de energia mostrou-se ser superior. Por esta razão, foi optado por utilizar o pino de *enable* dos DFFs para impedir chaveamentos desnecessários dos mesmos. Além disto, o PowerPlay Power Analyzer possibilita a visualização do consumo de energia por bloco do circuito, mostrando que o consumo de energia dinâmica do bloco TB representa em torno de 5% a 15% do consumo total de energia dinâmica dos VDs. Devido a este valor ser uma parcela consideravelmente pequena do consumo total, optou-se por analisar os circuitos mantendo-se o bloco TB.

- **Consumo de hardware**

A quantidade de elementos combinacionais e registradores nos circuitos pode ser obtida através do Quartus II. O bloco HD, o *Add* do ACS e o *Compare* do ACS são todos circuitos de lógica combinacional, sendo que a complexidade dos mesmos impacta na quantidade de portas lógicas e multiplexadores utilizados para construí-los. Registradores são utilizados para cada bit de informação que é armazenado nos circuitos. Os sinais registrados nos VDs estão listados nas Tabelas 6.2 a 6.6 a seguir, junto com suas dimensões (em bits) indicando a quantidade de DFFs utilizados. Vale lembrar que foi aplicada a terceira estratégia de construção – ACS e TB: minimizando os registradores dos percursos – nos decodificadores, de modo que são necessários apenas 14 DFFs para armazenar os percursos vencedores de cada nó. Além disto, a segunda estratégia de processamento – Bloco HD: atualizado apenas quando necessário – foi também aplicada nos decodificadores.

VD convencional				
Bloco	**Sinal**	**Quantidade**	**Bits**	**Total de DFFs**
VD	Contador de bits	1	3	3
VD	Contador de palavras-código	1	4	4
VD	Registrador de bit	1	2	2
HD	Registrador de palavra-código	1	3	3
ACS	Métrica	4	3	12
ACS	Percurso vencedor	4	2 x 14	112
TB	Memória de TB	4	2 x 14	112
VD	Palavra decodificada	1	2	2
TOTAL				**250**

Tabela 6.2 – Registradores do VD convencional.

VD puncionado com múltiplos ACSs				
Bloco	**Sinal**	**Quantidade**	**Bits**	**Total de DFFs**
VD	Contador de bits	1	3	3
VD	Contador de palavras-código	1	4	4
VD	Registrador de bit	1	1	1
HD	Registrador de palavra-código	1	3	3
ACS	Métrica	8	3	24
ACS	Percurso intermediário	4	1	4
ACS	Percurso vencedor	4	2 x 14	112
TB	Memória de TB	4	2 x 14	112
VD	Palavra decodificada	1	2	2
TOTAL				**265**

Tabela 6.3 – Registradores do VD puncionado com múltiplos ACSs.

VD puncionado com ACSs únicos				
Bloco	**Sinal**	**Quantidade**	**Bits**	**Total de DFFs**
VD	Contador de bits	1	3	3
VD	Contador de palavras-código	1	4	4
VD	Registrador de bit	1	1	1
HD	Registrador de palavra-código	1	3	3
ACS	Métrica	4	3	12
ACS	Percurso intermediário	4	1	4
ACS	Percurso vencedor	4	2 x 14	112
TB	Memória de TB	4	2 x 14	112
VD	Palavra decodificada	1	2	2
TOTAL				**253**

Tabela 6.4 – Registradores do VD puncionado com ACSs únicos.

VD mínimo com múltiplos ACSs				
Bloco	**Sinal**	**Quantidade**	**Bits**	**Total de DFFs**
VD	Contador de bits	1	3	3
VD	Contador de palavras-código	1	4	4
VD	Registrador de bit	3	1	3
ACS	Métrica	16	3	48
ACS	Percurso intermediário	4	1	4
ACS	Percurso vencedor	4	2 x 14	112
TB	Memória de TB	4	2 x 14	112
VD	Palavra decodificada	1	2	2
TOTAL				**288**

Tabela 6.5 – Registradores do VD mínimo com múltiplos ACSs.

VD mínimo com ACSs únicos				
Bloco	**Sinal**	**Quantidade**	**Bits**	**Total de DFFs**
VD	Contador de bits	1	3	3
VD	Contador de palavras-código	1	4	4
VD	Registrador de bit	1	1	1
ACS	Métrica	8	3	24
ACS	Percurso intermediário	4	1	4
ACS	Percurso vencedor	4	2 x 14	112
TB	Memória de TB	4	2 x 14	112
VD	Palavra decodificada	1	2	2
TOTAL				**262**

Tabela 6.6 – Registradores do VD mínimo com ACSs únicos.

Analisando as tabelas anteriores, fica evidente que os VDs com múltiplos ACSs apresentam a maior quantidade de registradores. O VD mínimo com múltiplos ACSs utiliza 288 DFFs, devido a este apresentar n=3 fases em que as métricas são atualizadas. Já o VD convencional apresenta a menor quantidade de DFFs – apenas 250 –, pois as métricas neste decodificador são atualizadas apenas uma vez por palavra-código recebida e, também, não apresenta percursos intermediários.

A Tabela 6.7 a seguir relaciona o consumo total de elementos lógicos, incluindo elementos de lógica combinacional e registradores. A quantidade destes elementos lógicos utilizados para construir um circuito não é necessariamente a soma dos elementos combinacionais com os registradores. Isto porque, conforme visto na seção 6.1, as células lógicas dos PLDs possuem circuitos combinacionais e também registradores, de modo que o Quartus II constrói o circuito de acordo com a

disponibilidade destas células e com o roteamento entre elas. Portanto, parte da lógica combinacional e parte da lógica seqüencial do VD podem estar na mesma célula lógica, sendo que o consumo real de hardware é melhor representado pelo consumo total de elementos lógicos, presente na Tabela 6.7. A coluna da direita desta tabela mostra a relação do consumo total de hardware do VD com o consumo total de hardware do VD convencional. O consumo total de hardware dos dispositivos da família Stratix é exibido em percentual da quantidade total de hardware disponível nos mesmos.

Família	Dispositivo	Viterbi *decoder*	Consumo de Hardware			
			Combinacional	Registradores	Total	%
MAX II	EPM570F100C4	Convencional	291	250	**403**	**100,0**
		Puncionado múltiplos ACSs	244	265	**358**	**88,8**
		Puncionado ACSs únicos	207	253	**322**	**79,9**
		Mínimo múltiplos ACSs	243	288	**354**	**87,8**
		Mínimo ACSs únicos	220	262	**334**	**82,9**
Cyclone II	EP2C5AF256A7	Convencional	291	250	**405**	**100,0**
		Puncionado múltiplos ACSs	239	265	**351**	**86,7**
		Puncionado ACSs únicos	208	253	**325**	**80,2**
		Mínimo múltiplos ACSs	235	288	**348**	**85,9**
		Mínimo ACSs únicos	212	262	**328**	**81,0**
Cyclone III	EP3C5E144A7	Convencional	291	250	**405**	**100,0**
		Puncionado múltiplos ACSs	239	265	**353**	**87,2**
		Puncionado ACSs únicos	208	253	**325**	**80,2**
		Mínimo múltiplos ACSs	235	288	**348**	**85,9**
		Mínimo ACSs únicos	212	262	**328**	**81,0**
Stratix II	EP2S15F484C3	Convencional	134	250	**3%**	**-**
		Puncionado múltiplos ACSs	102	265	**3%**	**-**
		Puncionado ACSs únicos	73	253	**2%**	**-**
		Mínimo múltiplos ACSs	126	288	**3%**	**-**
		Mínimo ACSs únicos	105	262	**3%**	**-**
Stratix III	EP3SE50F484C2	Convencional	132	250	**<1%**	**-**
		Puncionado múltiplos ACSs	102	265	**<1%**	**-**
		Puncionado ACSs únicos	71	253	**<1%**	**-**
		Mínimo múltiplos ACSs	126	288	**<1%**	**-**
		Mínimo ACSs únicos	104	262	**<1%**	**-**

Tabela 6.7 – Consumo de hardware dos VDs.

Apesar da quantidade superior de registradores nos demais decodificadores, o VD convencional apresentou o maior consumo de hardware em todos os PLDs testados. Devido à lógica combinacional mais complexa do VD convencional, com comparadores de 4 entradas – os quais equivalem a 3 comparadores de 2 entradas, acrescentado de um MUX de 2 entradas –, foram necessários mais elementos lógicos para implementá-lo. Já o VD puncionado com ACSs únicos teve o melhor desempenho, seguido muito próximo do VD mínimo com ACSs únicos. Esperar-se-ia que uma redução de hardware poderia resultar também em uma redução de consumo de energia dinâmica, por o circuito apresentar uma capacitância equivalente menor. Porém, isto não é garantido, pois se o hardware diminui, mas há um aumento em maior proporção da atividade do circuito, aumentar-se-ia o consumo de energia dinâmica. Desta maneira, é mais provável que pela reutilização dos blocos ACS foi demandado um roteamento mais complexo entre as células lógicas, o qual resultou em uma capacitância equivalente maior nos circuitos dos VDs com ACSs únicos, elevando, portanto, o consumo de energia.

Devido ao VD convencional consumir mais energia dinâmica na maioria dos casos testados e também consumir mais hardware em todos os casos, fica evidente que há uma relação entre estes dois consumos. Porém, quando se chega a consumos de hardware um pouco mais próximos – como é o caso dos VDs com múltiplos ACSs versus os VDs com ACSs únicos –, esta relação entre os consumos de energia dinâmica e hardware começa a não valer, sendo que detalhes mais profundos sobre a construção dos circuitos são necessários para concluir de fato qual o melhor circuito em ambos os termos.

- **Freqüência máxima de operação**

Como ocorreu com o consumo de energia, a freqüência máxima de operação é influenciada pela complexidade de roteamento dos circuitos. O Quartus II calcula a freqüência máxima de operação de um circuito através do maior tempo de propagação do sinal do nó origem ao nó destino dentro do circuito (caminho crítico). Em outras palavras, dentre todos os nós do circuito, o que possui o maior tempo de atualização em algum momento que ocorre variação é o que define a freqüência máxima do sistema, sendo que não é possível aplicar um clock com tempo menor que o tempo de estabilização do nível de tensão deste nó.

Os resultados deste experimento estão na Tabela 6.8, podendo ser observado que em todos os casos os VDs com múltiplos ACSs apresentaram os melhores resultados. Comparando estes resultados com os obtidos para o consumo de energia dinâmica, pode ser observada uma forte correlação, destacando o melhor desempenho para os VDs com múltiplos ACSs. Como o roteamento interno destes dois VDs é mais simples que o dos VDs com ACSs únicos, o tempo de propagação dos sinais é também menor, resultando em freqüências máximas de operação maiores. A coluna da direita mostra a relação da freqüência máxima de operação do VD com a freqüência máxima de operação do VD convencional.

Família	Dispositivo	Viterbi *decoder*	Freqüência Máxima (MHz)	%
MAX II	EPM570F100C4	Convencional	36,63	100,0
		Puncionado múltiplos ACSs	49,83	136,0
		Puncionado ACSs únicos	34,78	94,9
		Mínimo múltiplos ACSs	66,35	181,1
		Mínimo ACSs únicos	44,39	121,2
Cyclone II	EP2C5AF256A7	Convencional	66,43	100,0
		Puncionado múltiplos ACSs	92,56	139,3
		Puncionado ACSs únicos	67,16	101,1
		Mínimo múltiplos ACSs	108,72	163,7
		Mínimo ACSs únicos	80,01	120,4
Cyclone III	EP3C5E144A7	Convencional	69,97	100,0
		Puncionado múltiplos ACSs	110,94	158,6
		Puncionado ACSs únicos	89,06	127,3
		Mínimo múltiplos ACSs	125,72	179,7
		Mínimo ACSs únicos	105,60	150,9
Stratix II	EP2S15F484C3	Convencional	112,23	100,0
		Puncionado múltiplos ACSs	179,15	159,6
		Puncionado ACSs únicos	133,37	118,8
		Mínimo múltiplos ACSs	168,86	150,5
		Mínimo ACSs únicos	133,51	119,0
Stratix III	EP3SE50F484C2	Convencional	147,67	100,0
		Puncionado múltiplos ACSs	217,68	147,4
		Puncionado ACSs únicos	180,57	122,3
		Mínimo múltiplos ACSs	233,21	157,9
		Mínimo ACSs únicos	187,83	127,2

Tabela 6.8 – Freqüência máxima de operação dos VDs.

Com todos estes dados apresentados, permanece, porém, uma questão em aberto: qual o melhor VD? A resposta, no momento, seria: depende. Depende do que é realmente mais crítico para o sistema, mas depende também de qual PLD foi utilizado.

- Se for em termos de consumo de energia com FPGAs, os VDs com múltiplos ACSs seriam o ideal.
- Se for para utilizar FPGAs das famílias Cyclone II e III, os VDs com múltiplos ACSs apresentaram os melhores resultados e parecidos entre si, com o VD mínimo se destacando um pouco mais em termos de freqüência máxima de operação.
- Se for para utilizar CPLDs das famílias Stratix II e III, o VD puncionado com múltiplos ACSs seria o recomendado, a não ser se o PLD fosse operar em uma freqüência um pouco maior, restando o VD mínimo com múltiplos ACSs.
- Se o fator crítico for simplesmente consumo de hardware, o VD puncionado com ACSs únicos seria ideal.

Em resumo, os VDs puncionado e mínimo apresentaram desempenhos parecidos, ambos substancialmente superiores ao VD convencional, tornando interessante construir VDs com estas variações da treliça para o código (3,2,2,3).

Para confirmar os resultados de consumo de energia e consumo de hardware obtidos através do Quartus II, os circuitos foram também construídos em PSpice, através da linguagem SPICE. Apesar dos valores de consumo de hardware e de freqüência máxima de operação fornecidos pelo Quartus II serem bastante confiáveis, o consumo de energia, pelas razões de imprecisão citadas anteriormente, será analisado mais profundamente no Capítulo 7.

7. IMPLEMENTAÇÃO DOS DECODIFICADORES EM ASICs

Para validar os resultados obtidos com os circuitos apresentados no Capítulo 6, os decodificadores foram também construídos em circuitos integrados de aplicação específica – ASIC (do inglês, *Application Specific Integrated Circuit*). Em outras palavras, um ASIC é um circuito integrado fabricado especificamente para atender a uma certa aplicação – no caso, implementação dos VDs. Circuitos digitais construídos em ASICs são formados por transistores, sendo que as características destes elementos, como largura e comprimento do canal e tipo de substrato, são estipuladas pelo desenvolvedor. A grande vantagem de ASICs sobre PLDs é que em ASICs apenas os circuitos desejados são construídos, ao contrário de PLDs, que já possuem uma estrutura interna e apenas o roteamento entre as células lógicas é configurado na construção dos circuitos. Por esta razão, circuitos construídos em ASICs podem ter melhor aproveitamento de área do circuito integrado.

Como, por um lado, ASICs possuem a vantagem de seus circuitos poderem ser construídos da maneira desejada, em contrapartida há a necessidade de especificar cada elemento do circuito. Partindo de um material semicondutor “em branco”, é necessário que resistores, diodos, transistores, e qualquer outro elemento de circuitos eletrônicos sejam manualmente desenhados no ASIC. Antes de mandar um chip dedicado ser fabricado, é conveniente verificar o comportamento do circuito implementado. Para isto, pode-se utilizar softwares onde os componentes eletrônicos são especificados textualmente – em termos de seus valores e dimensões – e simulações dos mesmos podem ser realizadas. Para os circuitos dos VDs, foi optado em utilizar o software PSpice, sendo que a análise de consumo de energia foi realizada de forma gráfica, observando as formas de onda dos sinais dos circuitos geradas nas simulações. Já o consumo de hardware é simplesmente a quantidade de transistores utilizados para construir os circuitos, dado o qual é conhecido pois todos os circuitos precisam ser definidos pelo projetista.

7.1 SPICE

Simulation Program with Integrated Circuit Emphasis - SPICE (em inglês, programa de simulação com ênfase em circuitos integrados) foi desenvolvido no Electronics Research Laboratory na Universidade da California em Berkeley, nos EUA. SPICE é um simulador de circuitos eletrônicos utilizado para validar a integridade e analisar o comportamento do circuito antes do CI ser fabricado e foi apresentado pela primeira vez em 1973 [30]. Como os custos de fabricação são relativamente elevados, é conveniente analisar o circuito previamente, assegurando-se do seu comportamento. Dentre os vários simuladores, destaca-se o PSpice, criado em 1984 pela MicroSim, sendo encontrado atualmente no OrCAD, da Cadence Design Systems. O PSpice possibilita a simulação de circuitos eletrônicos e a análise do comportamento dos mesmos através das formas de onda de corrente e tensão em todos os componentes – incluindo resistores, capacitores, diodos, transistores, entre outros – presentes nos circuitos e nos nós que os interligam [31].

7.2 CONSIDERAÇÕES SOBRE AS SIMULAÇÕES E OS VDs

Os circuitos dos VDs foram todos construídos utilizando-se apenas transistores MOSFET. Além disto, os transistores foram interligados aplicando a lógica CMOS, salvo alguns circuitos (chaves e multiplexadores) que foram construídos com lógica *transmission gate*. Em ambas estas lógicas, tem-se que para cada nMOS há também um pMOS, de modo que a quantidade total de transistores utilizados para construir um dado circuito é formada metade por nMOS e metade por pMOS.

Foram realizadas diversas simulações nos circuitos para obter o consumo de energia. Os primeiros circuitos foram construídos utilizando um modelo mais atual do MOSFET (BSIM 3v3.1), porém as simulações foram realizadas em um tempo muito elevado, demorando alguns dias para simular uma mensagem composta de apenas 15 palavras-código, ou seja, 45 bits. Por esta razão foram alterados os transistores para um modelo mais antigo (MOS2), onde menos efeitos secundários do MOSFET – mais presentes em transistores menores ($\lambda \leq 0.5\mu m$) – são analisados, resultando equações

um pouco mais simples (porém sem eliminar nenhum dos parâmetros fundamentais). Com este modelo, foi possível analisar seqüências de 20 mensagens (formadas por 45 bits cada), com cada simulação durando, em média, três dias.

Como a mobilidade das lacunas é menor que a dos elétrons, usualmente constrói-se transistores pMOS com o dobro da largura (*W*) de canal dos nMOS, mantendo o mesmo comprimento do canal (*L*) [26]. Os transistores utilizados nos circuitos dos VDs foram construídos com as seguintes dimensões:

- nMOS: W=3μm e L=2μm;
- pMOS: W=6μm e L=2μm.

Foi aplicado um clock global de 100kHz nos VDs, sendo alimentados com uma tensão V_{DD}=3,3V. Para obter o consumo de energia, foi analisada a corrente $i(t)$ consumida pelo circuito, oriunda da fonte de alimentação. Integrando o produto $V_{DD}.i(t)$ e dividindo pelo intervalo de tempo [32], foi possível obter os valores de potência de cada circuito (na ordem de micro-Watts).

7.3 CONSUMO DE ENERGIA QUALITATIVO: ATIVIDADE DOS TRANSISTORES

Antes de apresentar os resultados de consumo de energia dos VDs, é possível obter (qualitativamente) uma estimativa do mesmo analisando a atividade de cada componente dos VDs individualmente, considerando quantas vezes o estado do componente é alterado, por palavra-código recebida. Em outras palavras, é estimado quantos transistores são acionados nos componentes de cada VD no intervalo de tempo de uma palavra-código. Os valores obtidos nestas estimativas são meramente aproximações, porém servem de base para o que será esperado do PSpice para os resultados de consumo de energia. A Tabela 7.1 a seguir mostra as funções realizadas dentro de cada VD e a utilização, medida em transistores, dos componentes ao longo de uma palavra-código, a qual é obtida pelo produto da quantidade de transistores dos componentes da função com a taxa de atividade (utilização) dos mesmos. A coluna da direita mostra o percentual de atividade do respectivo VD em relação à atividade do VD convencional.

VD	Função / Operação							%
	Clocks e *flags*	**Contadores**	**Palavra-código**	**HD**	**ACS**	**TB**	**Total**	
Convencional	224	288	228	90	5014	3008	**8852**	**100,0**
Puncionado com múltiplos ACSs	304	288	128	40	4886	3008	**8654**	**97,8**
Puncionado com ACSs únicos	338	288	128	88	5046	3008	**8896**	**100,5**
Mínimo com múltiplos ACSs	304	288	60	6	5782	3008	**9448**	**106,7**
Mínimo com ACSs únicos	186	288	60	6	5902	3008	**9450**	**106,8**

Tabela 7.1 – Estimativa da atividade dos transistores dos VDs, por palavra-código.

Os valores da Tabela 7.1 indicam que, se a estimativa é realmente precisa, os VDs mínimos apresentam um consumo de energia maior que o do VD convencional. Além disto, os dois VDs puncionados, apesar de realizarem as mesmas funções e suas estruturas serem praticamente iguais, apresentam atividades de transistores não muito próximas. Isto ocorre devido ao VD puncionado com ACSs únicos utilizar os mesmos somadores nas k=2 fases da treliça, os quais foram construídos para somar 4 bits (métricas) com 2 bits (distâncias calculadas). Já o VD puncionado com múltiplos ACSs utiliza um somador mais simples na segunda fase da treliça, aproveitando o fato da distância calculada nesta fase ser formada por apenas um bit.

Os VDs mínimos apresentaram uma atividade total de transistores similares. Entretanto, como o VD com múltiplos ACSs utiliza clocks distintos (através de *clock gating*) para atualizar as métricas em cada fase da treliça, este VD apresentou uma atividade maior de clocks e *flags*. Isto já não ocorre com o VD com ACSs únicos devido a este utilizar a mesma memória de métricas nas n=3 fases; porém, o bloco ACS deste VD apresenta atividade mais elevada devido aos multiplexadores utilizados para a seleção de sinais.

Apesar destas prévias considerações, vale destacar dois aspectos da estimativa do consumo de energia e do PSpice:

1. A estimativa considera que toda palavra-código recebida afetará o circuito em praticamente o máximo de sua atividade, sendo que, na prática, nem toda palavra-código recebida afeta os estados de todos os nós e DFFs dos circuitos. Por esta razão, foram realizadas simulações compostas de uma longa seqüência de palavras-código, aleatoriamente geradas, representando uma situação mais real.
2. O PSpice não leva em consideração o roteamento dos circuitos. Devido a isto, é possível que o consumo de energia dos VDs com ACSs únicos, os quais apresentam, na prática, roteamento mais complexo, não se assemelhe tanto ao valor real do consumo de energia.

7.4 RESULTADOS: CONSUMO DE ENERGIA

Como foi mencionado anteriormente, o consumo de energia dos VDs foi obtido através da análise das formas de onda geradas pela simulação dos circuitos, na decodificação de 20 mensagens seqüenciais (compostas de 15 palavras-código cada). A Tabela 7.2 mostra estes resultados, sendo que a coluna à direita mostra o consumo de energia do respectivo VD em relação ao VD convencional.

VD	Consumo de Energia (μW)	%
Convencional	45,86	100,0%
Puncionado com múltiplos ACSs	44,40	96,8%
Puncionado com ACSs únicos	45,03	98,2%
Mínimo com múltiplos ACSs	51,41	112,1%
Mínimo com ACSs únicos	52,60	114,7%

Tabela 7.2 – Consumo de energia dos VDs.

Como pode ser observado, o VD puncionado com múltiplos ACSs mostrou realmente ser o de menor consumo de energia, confirmando a estimativa. Curiosamente, o VD puncionado com ACSs únicos apresentou um consumo melhor que o do VD convencional, ao contrário do que estava previsto para ele. Isto mostra que a palavra-código recebida impacta bastante o consumo de energia do circuito. Além disto,

como os VDs puncionados realizam as mesmas operações e a única diferença mais significativa destas operações está no somador um pouco mais complexo do VD com ACSs únicos, o consumo destes VDs ficou muito próximo. Este resultado do consumo de energia entre os VDs puncionados ficou bastante similar ao obtido com os PLDs.

Em contrapartida, como os VDs mínimos diferem entre si em termos de seleção de sinais, pode-se concluir que os multiplexadores do VD com ACSs únicos contribuem com uma parcela proporcionalmente considerável do consumo de energia. Também obtido para os PLDs, o resultado do consumo de energia entre os VDs mínimos ficou similar, mostrando que o VD com múltiplos ACSs é mais econômico. Apesar disto, ambos os VDs mínimos apresentaram consumo superior ao do VD convencional, sendo que nos PLDs o VD com múltiplos ACSs apresentou menor consumo que o VD convencional em todos os casos analisados. Com isto é possível concluir que a estrutura mais complexa do comparador do VD convencional exige uma maior utilização de células lógicas nos PLDs, além de um roteamento mais complexo entre elas, elevando o consumo de energia. Já no PSpice, além do roteamento interno dos circuitos não ser contemplado, impactando em parte no consumo de energia, os circuitos são elaborados de maneira diferente dos implementados em PLDs. A estrutura dos circuitos dos VDs mínimos no PSpice, junto com o fato destes atualizarem as métricas n=3 vezes por palavra-código, justificam as estimativas da Tabela 7.1, de modo que nos ASICs os VDs mínimos realmente apresentam-se menos econômicos que o VD convencional.

7.5 RESULTADOS: CONSUMO DE HARDWARE

O consumo de hardware dos VDs é obtido através do PSpice, que contabiliza quantos transistores foram utilizados nos circuitos. A Tabela 7.3 mostra os resultados obtidos, lembrando que metade dos transistores é nMOS e a outra é pMOS, devido aos tipos de lógica aplicados nos circuitos.

VD	Transistores MOSFET	%
Convencional	8624	100,0%
Puncionado com múltiplos ACSs	8466	98,2%
Puncionado com ACSs únicos	7388	85,7%
Mínimo com múltiplos ACSs	9312	108,0%
Mínimo com ACSs únicos	7964	92,3%

Tabela 7.3 – Consumo de hardware dos VDs.

Se for desconsiderado o roteamento, analisando apenas a área de circuito integrado ocupada pelos transistores, os resultados novamente são parecidos com os dos PLDs para os VDs puncionados, porém diferem para os VDs mínimos. Isto mostra que os comparadores do VD convencional são circuitos que utilizam muitas células lógicas nos PLDs, se comparado com os demais circuitos dos VDs. Ao utilizar mais células lógicas, o roteamento interno do circuito torna-se mais complexo, afetando também, neste caso, o consumo de energia.

Através destes resultados, é possível obter as seguintes conclusões sobre os circuitos construídos em ASICs e analisados no PSpice:

1. Como o roteamento interno dos circuitos não foi contemplado pelo PSpice, o consumo de energia dos VDs mínimos foram, em parte, desfavoráveis se comparado com o consumo do VD convencional e com os resultados obtidos para os PLDs.
2. A seleção dos sinais no VD mínimo com ACSs únicos realmente eleva o consumo de energia, conforme foi também observado nos PLDs.
3. As maneiras diferentes com que o PSpice e o Quartus II implementam os circuitos dos VDs implicam em resultados divergentes, principalmente para os VDs mínimos.

Portanto, a construção dos circuitos em PSpice deixou claro que os VDs puncionados são realmente favoráveis em termos de consumo de energia e de hardware. Já não é possível obter uma conclusão definitiva sobre os consumos dos VDs mínimos. Devido à maneira diferente de construir circuitos no PSpice e em PLDs,

os consumos dos circuitos elaborados no PSpice foram bastante divergentes dos resultados do Quartus II, mostrando que cada caso deve ser analisado individualmente de acordo com o tipo de circuito integrado utilizado.

8. CONCLUSÕES

Este capítulo contempla os assuntos abordados ao longo da dissertação, além do conteúdo estudado durante o Mestrado e as dificuldades encontradas para chegar nos resultados. Por fim estão apresentadas as conclusões da dissertação e os trabalhos futuros que poderão ser realizados tendo em vista os resultados aqui obtidos.

8.1 CONTEÚDO ABORDADO

A seguir estão listados os assuntos apresentados e discutidos ao longo da dissertação:

a. Códigos convolucionais e suas características;
b. O codificador e o diagrama de estados do código (3,2,2,3);
c. As treliças do código (3,2,2,3) e as complexidades;
d. Decodificação usando o algoritmo de Viterbi;
e. O decodificador de Viterbi (VD) e seus componentes;
f. Particularidades dos VDs de acordo com a treliça utilizada;
g. Os circuitos dos VDs e as estratégias de construção e processamento;
h. Medições e estimativas do consumo de energia, do consumo de hardware e da freqüência máxima de operação dos VDs;
i. Resultados dos VDs construídos PLDs;
j. Resultados dos VDs construídos em ASICs.

8.2 APRENDIZADO E DIFICULDADES

Os dois assuntos mais explorados ao longo da dissertação foram a estrutura do VD e a implementação dos VDs em circuitos eletrônicos. O Capítulo 4 mostra o resultado que surgiu após muita análise das treliças e da operação do VD. Com esta análise foram levantadas as estratégias de construção e processamento dos decodificadores, as quais possibilitaram a implementação do VD com as três treliças.

Foram identificadas as vantagens e desvantagens dos VDs construídos, de acordo com a treliça utilizada, visando, principalmente, preservar uma comparação justa entre os circuitos implementados. Os VDs foram construídos em PLDs através da linguagem de programação de hardware VHDL, utilizando o Quartus II, e em ASICs através do PSpice, com os circuitos definidos neste a nível de transistor.

O Capítulo 5 abordou a metodologia de estimativa e medição do consumo de energia, do consumo de hardware e da freqüência máxima de operação dos VDs. Conforme visto, a segunda e terceira características dos decodificadores foram facilmente obtidas para os PLDs com o Quartus II. Já os resultados em termos de consumo de energia mostraram que os VDs com múltiplos ACSs são mais econômicos que o VD convencional, porém deixaram indícios de imprecisão, devido à relação de consumo entre os VDs variar consideravelmente de acordo com o PLD utilizado. Após inúmeras tentativas em obter a precisão da estimativa de consumo de energia gerada pelo Quartus II através do suporte da Altera, foi decidido analisar se era possível sanar esta dúvida construindo os VDs em ASICs, desenvolvendo os circuitos, agora, a nível de transistor.

Os circuitos dos VDs foram, portanto, analisados com o PSpice, mostrando que o consumo de energia ainda é algo difícil de se obter, principalmente por dois fatores: (i) pelo PSpice tomar um considerável tempo para realizar as simulações dos circuitos dos decodificadores, chegando a demorar vários dias para simular a decodificação de uma única mensagem; (ii) e pelos circuitos elaborados em PSpice serem construídos de maneira diferente daqueles implementados em PLDs, devido às características próprias de PLDs (estrutura interna fixa, configurando apenas as interligações entre as células lógicas e seus componentes) e ASICs (circuito especificado a nível de transistor). Além disto, o PSpice não leva em consideração o roteamento interno do circuito – algo que influencia o consumo de energia. Em relação ao consumo de hardware, pode haver discrepâncias de acordo com tipo de circuito integrado utilizado (PLD ou ASIC), novamente por causa das particularidades destes dispositivos e pelo PSpice não contemplar o roteamento. Portanto, os resultados divergentes de consumo de energia e hardware entre o PSpice e o Quartus II mostram que certas características dos circuitos são amplamente influenciadas pelo software compilador – na escolha das células

lógicas e roteamento interno destas – e pelo desenvolvedor – na lógica aplicada e na maneira de construir os circuitos.

8.3 CONCLUSÕES

As conclusões a seguir estão apresentadas para cada uma das características analisadas para os decodificadores do código (3,2,2,3). O quarto item expõe as conclusões finais.

8.3.1 Consumo de energia

- **Para implementações em PLDs**: O consumo de energia medido é um pouco menor nos VDs com múltiplos ACSs (na ordem de 75% a 95% do VD convencional, dependendo do dispositivo escolhido). Para os VDs com ACSs únicos, a vantagem é pequena, podendo inclusive ser pior do que o VD convencional para o caso do VD mínimo.
- **Para implementação em ASIC**: O consumo dos VDs puncionados é praticamente na mesma ordem do VD convencional. No entanto, o consumo medido para os VDs mínimos é significativamente pior (na ordem de 113% do VD convencional).

8.3.2 Consumo de hardware

- **Para implementações em PLDs**: O consumo de hardware medido (em relação ao VD convencional) é menor tanto nos VDs puncionados quanto nos VDs mínimos. Porem, como esperado, o consumo dos VDs com ACSs únicos foi menor (cerca de 80% do consumo do VD convencional) do que aquele dos VDs com múltiplos ACSs (na ordem de 85% do VD convencional).
- **Para implementação em ASIC**: O consumo de hardware medido (em relação ao VD convencional) do VD puncionado com múltiplos ACSs é similar ao do VD convencional, porem o VD puncionado com ACSs únicos é vantajoso (cerca de 85% dos transistores consumidos no VD convencional). No caso do VD mínimo com ACSs únicos o consumo é um pouco menor (cerca de 92%), porem é desvantajoso (na ordem de 108%) para o VD mínimo com múltiplos ACSs.

8.3.3 Freqüência máxima de operação

- **Para implementações em PLDs** (somente estas foram medidas): As freqüências máximas medidas para os VDs puncionados e mínimos foram superiores às do VD convencional em praticamente todas as implementações, destacando os VDs com múltiplos ACSs: o VD puncionado com 135% a 160% e o VD mínimo com 150% a 180%, em relação ao VD convencional.

8.3.4 Respostas às perguntas do Capítulo 2

Foram levantadas algumas perguntas ao final do Capítulo 2, as quais podem ser respondidas da seguinte maneira:

- Será que uma complexidade reduzida de treliça reflete em circuitos mais simples?
    - Não. Os circuitos das treliças puncionada e mínima possuem uma estrutura mais complexa em termos de seleção de sinal e de memória de percursos intermediários, apesar de poderem apresentar um consumo menor de hardware (se comparados com o VD convencional).
- Ou possivelmente circuitos que demandam menos computações para se obter os mesmos resultados?
    - Sim. Distâncias de Hamming, somas e comparações mais simples foram realizadas nos VDs puncionados e mínimos.
- Será que a estrutura irregular da treliça mínima não implicará na construção de um VD mais complexo?
    - Sim, principalmente quando utilizando a estratégia de ACSs únicos, tornando a seleção de sinais do VD mínimo bastante complexa.

As respostas acima mostram a complexidade de construção dos circuitos dos decodificadores, além de algumas vantagens observadas nas implementações. As conclusões finais, a seguir, resumem o que foi observado ao longo deste trabalho.

8.3.5 Conclusões finais

Apesar de alguns pontos complicados de serem analisados, pode-se concluir que há uma grande vantagem em construir os VDs com as variações das treliças

quando se visa velocidades maiores de processamento dos decodificadores em PLDs. Quanto ao consumo de hardware, pode-se esperar uma redução moderada se comparado com o VD convencional. Em termos de consumo de energia, as perspectivas são menos conclusivas, mas é de se esperar uma redução mínima a moderada. Porém, conforme recomendado pelo próprio pessoal de suporte da Altera, a melhor maneira de se obter valores precisos de consumo de energia é analisando individualmente cada circuito na prática, valendo isto tanto para PLDs quanto para ASICs. Apesar das vantagens que os VDs puncionados e mínimos apresentaram em alguns dos casos analisados, a complexidade das treliças apresentadas em [4] não se traduziu como um parâmetro preciso para o desempenho dos decodificadores.

8.4 TRABALHOS FUTUROS E QUESTÕES EM ABERTO

Após toda a análise e trabalho realizados ao longo deste Mestrado, alguns trabalhos futuros podem surgir baseando-se nas questões abertas listadas a seguir.

a. Existe outra medição de complexidade das treliças que se traduz melhor para os resultados obtidos?
b. Como a estrutura da treliça mínima do código (3,2,2,3) é pouco regular, é possível que um código com uma treliça mínima que possui a mesma quantidade de nós em todas as fases apresente um VD mínimo com melhor desempenho?
c. Como foi analisado apenas um código, é possível que códigos com outras taxas ou tamanhos de palavra-informação e/ou palavra-código apresentem resultados mais favoráveis ao VD convencional?
d. Os valores apresentados pelo Quartus II e pelo PSpice são confiáveis? Seria interessante fabricar um chip dedicado contendo os VDs para medir o consumo real de energia, bem como a freqüência máxima de operação.

REFERÊNCIAS

[1] VITERBI, A. J. **Error bounds for convolutional codes and an asymptotically optimum decoding algorithm**, IEEE Trans. Inform. Theory, vol. 13, no. 2, p. 260-269, abr. 1967.

[2] COSTELLO, D. J.; LIN, S. **Error Control Coding**. Upper Saddle River: Pearson Prentice Hall, 2nd ed., 2004.

[3] CAIN, J. B.; CLARK Jr., G. C.; GEIST, J. M. **Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Likelihood Decoding**. IEEE Trans. Info. Theory, Vol. IT-25, no. 1, p. 97-100, jan. 1979.

[4] MCELIECE, R. J.; LIN, W. **The Trellis Complexity of Convolutional Codes**. IEEE Trans. Inform. Theory, vol. 42, no. 6, p. 1855-1864, nov. 1996.

[5] KANG, I.; WILSON Jr., A. N. **Low-power Viterbi decoder for CDMA mobile terminals**. IEEE J. Solid-State Circuits, vol. 33, p. 473-482, mar. 1998.

[6] BOUGARD, B. *et al*. **Energy-scalability enhancement of wireless local area network transceivers**. Proc. IEEE Workshop on Signal Processing Advances in Wireless Communications, p. 449-453, jul. 2004.

[7] KIVIOJA, M.; ISOAHO, J.; VÄNSKÄ, L. **Design and Implementation of Viterbi Decoder with FPGAs**. Journal of VLSI Signal Processing 21, p. 5-14, maio 1999.

[8] CHANG, Y.; SUZUKI, H.; PARHI, K. K. **A 2-Mb/s 256-State 10-mW Rate-1/3 Viterbi Decoder**. IEEE Journal of Solid-State Circuits, vol. 35, no. 6, p. 826-834, jun. 2000.

[9] ALLAN, G.; SIMMONS, S. **A VLSI implementation of an adaptive-effort low-power Viterbi decoder for wireless communications**. Canadian Conference on Electrical and Computer Engineering, vol. 2, p. 1183-1188, maio 2001.

[10] TESSIER, R. *et al*. **A Reconfigurable, Power-Efficient Adaptive Viterbi Decoder**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13-4, p. 484-488, abr. 2005.

[11] SHANNON, C. E. **A mathematical theory of communication**. Bell System Technical Journal, vol. 27, p. 379-423 e 623-656, jul. e out. 1948.

[12] ELIAS, P. **Coding for Noisy Channels**. I.R.E. Convention, record part 4, vol. 3, p.37-46, 1955.

[13] WYNER, A. D.; ASH, R. B. **Analysis of Recurrent Codes**. IEEE Trans. Inform. Theory, vol. IT-9, p. 143-156, jul. 1963.

[14] MCELIECE, R. J. **On the BCJR trellis for linear block codes**. IEEE Trans. Inform. Theory, vol. 42, no. 4, p. 1072-1092, jul. 1996.

[15] SIDORENKO, V.; ZYABLOV, V. **Decoding of convolutional codes using a syndrome trellis**. IEEE Trans. Inform. Theory, vol. 40, no. 5, p. 1663-1666, set. 1994.

[16] VARDY, A. **Trellis Structure of Codes**. Elsevier, Handbook of Coding Theory, p. 1989-2118, 1998.

[17] WOZENCRAFT, J. M.; REIFFEN, B. **Sequential Decoding (Research Monograph)**, The MIT Press / John Wiley and Sons, mar. 1961.

[18] MASSEY, J. L. **Threshold Decoding**. Massachusetts: MIT Press, abr. 1963.

[19] HELLER, J. A.; JACOBS, I. M. **Viterbi Decoding for Satellite and Space Communications**. IEEE Trans. on Communication Technology, vol. 19, no 5, part 1, p. 835-848, out. 1971.

[20] STEPHENS, R.; PELLET, M. **Joint NASA/ESA Telemetry Channel Coding Guideline: Issue 1**. NASA/ESA Working Group (NEWG) Publication, jan. 1982.

[21] GEMMEKE, T.; GANSEN, M.; NOLL, T. G. **Implementation of Scalable Power and Area Efficient High-Throughput Viterbi Decoders**, IEEE Journal of Solid-State Circuits, vol. 37, no. 7, p. 941-948, jul. 2002.

[22] SUN, F.; ZHANG, T. **Parallel High-Throughput Limited Search Trellis Decoder VLSI Decoder**, IEEE Trans. on VLSI Systems, vol. 13, no. 9, p. 1013-1022, set. 2005.

[23] SWAMINATHAN, S.; TESSIER, R.; GOECKEL, D.; BURLESON, W. **A Dynamically Reconfigurable Adaptive Viterbi Decoder**. International Symposium on FPGAs, Session: Innovative Applications, p. 227-236, fev. 2002.

[24] EMNETT, F.; BIEGEL, M. **Power Reduction Through RTL Clock Gating**. SNUG, San Jose, 2000.

[25] INTEL CORPORATION. **Intel's First Microprocessor - The Intel 4004**.
Disponível em: <www.intel.com/museum/archives/4004.htm>.
Acesso em: 31 maio 2009.

[26] PEDRONI, V. A. **Digital Electronics and Design, with VHDL**. San Francisco: Morgan Kauffman / Elsevier, jan. 2008.

[27] ALTERA CORPORATION, **Quartus II 7.2 Handbook**. Out. 2007.
Disponível em: <www.altera.com>.

[28] ALTERA CORPORATION, **PowerPlay Power Analysis, Quartus II 9.0 Handbook**. Vol. 3, mar. 2009.

[29] IEEE STANDARD VHDL LANGUAGE REFERENCE MANUAL**. IEEE Std 1076-1987**, mar. 1988.
Disponível em: <www.ieee.org>.

[30] NAGEL, L. W; PEDERSON, D. O. **SPICE (Simulation Program with Integrated Circuit Emphasis), Memorandum No. ERL-M382**. University of California, Berkeley, abr. 1973.

[31] CADENCE DESIGN SYSTEMS. **PSpice 10.0 User Guide**. Jun. 2003.

[32] HARRIS, D.; WESTE, N. **CMOS VLSI Design: A Circuits and Systems Perspective.** Addison Wesley, 3rd ed., 2004.

Resumo

PEDRONI, Bruno Umbria. Implementação em hardware de um decodificador de Viterbi baseado nas treliças convencional, puncionada e mínima. 2009. 105 f. Dissertação – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná. Curitiba, 2009.

O decodificador de Viterbi (VD) é amplamente utilizado nos sistemas atuais de comunicação, principalmente os sem fio, onde o consumo de energia é crítico. O VD realiza a decodificação de códigos convolucionais, os quais podem ser representados por treliças, sendo que é através destas que o VD processa a decodificação. Conforme avaliado por McEliece e Lin, as treliças puncionada e mínima de um código de taxa 2/3 apresentam complexidade reduzida em relação à treliça convencional. Nesta dissertação, foram construídos VDs baseados nestas três estruturas de treliça do código de taxa 2/3. Os circuitos dos decodificadores foram implementados em dispositivos de lógica programável (PLDs) e em circuitos integrados de aplicação específica (ASICs), avaliando o impacto da treliça – e sua complexidade – sobre o consumo de energia, o consumo de hardware e a freqüência máxima de operação de cada VD.

Palavras-chave: Treliça de código convolucional. Decodificador de Viterbi. Desempenho de decodificador. Dispositivo de lógica programável. Circuito integrado de aplicação específica.

Área de Conhecimento

3.04.00.00-7 Engenharia Elétrica
3.04.06.00-5 Telecomunicações
3.04.03.03-0 Circuitos Eletrônicos
1.03.04.01-0 Hardware

Número da Ata

Dissertação N° 510

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