# Implementação de um Codificador LDPC para um Sistema de Televisão Digital

Fábio Lumertz, Fabbryccio A. C. M. Cardoso e Dalton S. Arantes †

***Resumo*— Este trabalho apresenta uma implementação em hardware de um codificador LDPC para um sistema de televisão digital. O codificador implementado é baseado em uma classe estendida de códigos de repetição e acumulação irregulares, eIRA, com palavra-código de 9792 bits e taxa 3/4. A implementação foi realizada utilizando tecnologias inovadoras de prototipagem rápida para FPGAs, como o System Generator do Design Flow da Xilinx, o qual será brevemente introduzido. Este trabalho insere-se em um projeto mais amplo de elaboração de uma tecnologia de modulação inovadora para o Sistema Brasileiro de Televisão Digital (SBTVD).**

***Palavras-Chave*— LDPC, eIRA, Televisão Digital, FPGAs, Xilinx System Generator.**

***Abstract*— This work presents the implementation of an LDPC encoder for a digital television system. The implemented encoder is based on an eIRA - extended Irregular Repeat Accumulate - with codeword-length equal to 9792 bits and rate 3/4. The implementation was developed using advanced technologies for rapid prototyping on FPGAs, like the System Generator of Xilinx´s Design Flow, which will be briefly introduced. This work is part of a more ambitious project for development of an advanced modulation standard for the Brazilian Digital Television System (SBTVD).**

***Keywords*— LDPC, eIRA, Digital Television, FPGAs, Xilinx System Generator.**

## I. INTRODUÇÃO

Durante o projeto do Sistema Brasileiro de Televisão Digital (SBTVD), financiado pelo Governo Brasileiro, o Instituto Nacional de Telecomunicações (INATEL), a Universidade Estadual de Campinas (UNICAMP), a Universidade Federal de Santa Catarina (UFSC), o Centro Federal de Educação Tecnológica do Paraná (CEFET-PR) e a Linear Equipamentos Eletrônicos S.A. uniram seus esforços de pesquisa na formação de um consórcio responsável por uma proposta de Modulação Inovadora (MI-SBTVD). Dentre as principais inovações do sistema proposto, que será apresentado na Seção II, encontra-se o emprego de um código LDPC (*Low Density Parity Check*) na codificação de canal. A implementação de um codificador LDPC para o sistema proposto, bem como as ferramentas de prototipagem rápida utilizadas, constituem o tema central deste trabalho.

Os códigos LDPC, ou códigos de matriz de paridade de baixa densidade, são códigos de bloco binários lineares que utilizam matrizes de paridade $\mathbf{H}$ esparsas, ou seja, de baixa densidade de elementos não nulos. Estes códigos foram apresentados pela primeira vez por Robert Gallager (1931-) em 1962 [1]. Em 1981 Tanner [2] apresentou uma nova interpretação para os códigos LDPC, a partir de uma representação gráfica da matriz de paridade, técnica que atualmente é chamada de gráfico de Tanner ou gráfico bipartido. Finalmente, em meados dos anos 90 os códigos LDPC foram redescobertos, passando então por diversas evoluções, especialmente no que se refere à regularidade de sua estrutura [3]. Apesar de Gallager ter proposto o uso do LDPC como corretor de erro, ele não propôs um método específico para a construção algébrica e sistemática dos códigos.

O método gráfico desenvolvido por Tanner oferece, entre outras vantagens, uma forma conveniente de representar a matriz de paridade ($\mathbf{H}$). Os índices das colunas da matriz $\mathbf{H}$ são representados por círculos chamados de nós de variável (ou mais especificamente de nós de bit) e os índices das linhas são representados por quadrados chamados de nós de paridade (ou mais comumente de nós de cheque). Cada bit na palavra-código corresponde a um nó de bit e cada equação de cheque de paridade corresponde a um nó de cheque.

A ligação entre um nó de bit e um nó de cheque é chamada de ramo. Cada ramo indica que um dado '1' da matriz $\mathbf{H}$ pertence à coluna do nó de bit de origem do ramo e também pertence à linha do nó de cheque de destino deste mesmo ramo. O número total de ramos corresponde ao número total de uns da matriz de paridade $\mathbf{H}$.

O grau de um dado nó de bit ou de um nó de cheque pode ser determinado de duas formas. Através da matriz $\mathbf{H}$, o grau de um nó de bit corresponde ao número de uns presentes em sua respectiva linha e o grau de um nó de cheque corresponde ao número de uns presentes em sua respectiva coluna. Graficamente, o grau de um nó pode ser determinado através do número de ramos que sai do nó.

Outro conceito bastante importante é referente à regularidade do código. Um código é dito regular quando o número de uns é o mesmo em todas as linhas e colunas da matriz $\mathbf{H}$, ou graficamente, quando todos os nós de cheque e todos os nós de bit apresentam o mesmo grau. Quando Gallager desenvolveu os códigos LDPC, nos anos 60, ele apresentou apenas códigos regulares. Segundo [4], os códigos irregulares têm melhor desempenho do que os regulares, embora apresentem uma complexidade maior de projeto.

Um ciclo em um gráfico de Tanner representa um caminho fechado. O número de saltos dentro de um ciclo é conhecido como grau do ciclo. É conveniente ressaltar que ciclos de baixo grau são também referenciados como ciclos curtos, devido à representação dos gráficos de Tanner. Segundo [5], ciclos

† Os autores atuam junto ao Departamento de Comunicações, Faculdade de Engenharia Elétrica e Computação, UNICAMP, Campinas, SP, lumertz@decom.fee.unicamp.br, cardoso@decom.fee.unicamp.br, dalton@decom.fee.unicamp.br. Este trabalho foi financiado pela FINEP, através do Projeto MI-SBTVD - RFP-18 e pelo CNPq, que financiou Bolsa de Mestrado para F. Lumertz.

curtos, especialmente de grau quatro, devem ser evitados para o projeto de códigos eficientes, pois provocam um elevado grau de erro (saturação na probabilidade de erro de bit - *error floor*).

Para a obtenção da matriz de paridade **H**, este trabalho considera as técnicas propostas em [5][6][7], usando códigos eIRA (*extended Irregular Repeat Accumulate*), que visam principalmente à viabilidade e diminuição da complexidade de implementação do código. O emprego dessas técnicas é apresentado na Seção IV.

A implementação em hardware descrita neste trabalho só foi possível de ser realizada em tempo hábil, isto é, compatível com o cronograma do projeto, graças ao emprego de ferramentas de prototipagem rápida, as quais permitem um incremento significativo na eficiência de equipes desenvolvedoras. Em um mundo que exige cada vez mais rapidez e agilidade na demonstração de resultados de pesquisas científicas e no desenvolvimento de novos produtos, encontrar soluções de uma forma eficiente e rápida é fundamental para manter-se competitivo. Segundo [8], "o único desafio com o qual se deparam os engenheiros de projeto/desenvolvimento atualmente é conceber sistemas cada vez mais complexos em um *time-to-market* (janela de tempo de desenvolvimento até sua chegada ao mercado) comprimido, para produtos que possuem ciclos de vida cada vez mais curtos". Por esta razão, as ferramentas de prototipagem rápida são apresentadas na Seção III deste trabalho.

## II. O SISTEMA DE TV DIGITAL PROPOSTO

O modelo de televisão digital no qual está inserido o codificador LDPC implementado neste trabalho é exibido na Figura 1. Este trabalho representa uma continuação das pesquisas desenvolvidas para o Sistema Brasileiro de Televisão Digital (SBTVD), pois desde então algumas modificações foram feitas no sistema para suportar, na camada de transporte, os protocolos UDP/IP.

Os blocos indicados na Figura 1 com os nomes Vídeo Bruto, Codificador de Vídeo, Empacotador UDP/IP, Decodificador de Vídeo, Desempacotador UDP/IP, Conversor Camada MAC e Reprodutor de Vídeo foram implementados em plataformas PC, tanto para o transmissor quanto para o receptor. Os fluxos de vídeo são gerados e multiplexados em um computador PC. Por intermédio de uma interface Ethernet e do bloco denominado Conversor Camada MAC, o *Transport Stream* é passado para o módulo de modulação, implementado em hardware sobre uma plataforma baseada em FPGAs. O encapsulamento dos dados em quadros MAC (*Media Access Protocol*) faz-se necessário para viabilizar a utilização da interface Ethernet.

Os dados binários são então entregues ao aleatorizador a fim de evitar que padrões periódicos sejam mantidos em longas seqüências de bits. Isto poderia causar uma indesejada concentração espectral de potência, o que tornaria o sistema mais suscetível a desvanescimentos seletivos em freqüência, e também poderia causar uma alta relação entre potência de pico e potência média, que por sua vez dificultaria o uso eficiente de amplificadores de potência [9].

A codificação de canal proposta é a combinação de uma codificação Reed-Solomon com a codificação LDPC. Após a codificação de canal, o sistema mapeia os bits resultantes para serem modulados em fase e fase-quadratura em múltiplas portadoras. Um mapeamento possível é o 64QAM (*Quadrature and Amplitude Modulation*).

Posteriormente tem-se a etapa de modulação propriamente dita. A proposta de modulação foi de um sistema que combina técnicas de OFDM (*Orthogonal Frequency Division Multiplexing*) [10] com um esquema de diversidade espacial com duas antenas transmissoras e uma antena receptora. A diversidade de transmissão foi implementada usando a técnica de Alamouti [11], adaptada para o OFDM, utilizando codificação espaço-temporal em pares de símbolos OFDM. O emprego da técnica de Alamouti e a utilização da codificação LDPC constituem as principais inovações propostas pelo Consórcio MI-SBTVD para o SBTVD.

Após a etapa de modulação os dados são convertidos de digital para analógico em uma freqüência intermediária (FI). O sinal resultante em FI é então convenientemente filtrado e transmitido a partir de módulos de rádio freqüência (RF).

## III. FERRAMENTAS DE PROTOTIPAGEM RÁPIDA PARA FPGAS

O fluxo de desenvolvimento utilizado neste trabalho é baseado no ISE (*Integrated Software Environment*) Design Flow da Xilinx Inc. Em linhas gerais, este fluxo de desenvolvimento é constituído pelas etapas de descrição (entrada), síntese e implementação do projeto, além da etapa de programação do dispositivo FPGA [12].

A etapa de descrição do projeto pode ser feita através da forma esquemática, através de linguagens HDL (como VHDL ou Verilog), através da entrada de arquivos de terminação *edif*, *ngc* ou *ngo*, obtidos a partir de ferramentas de síntese ou, ainda, através de projetos desenvolvidos em System Generator. Na etapa de descrição do projeto, podem ser utilizados Núcleos de Propriedade Intelectual (NPI ou *IP Cores*), tanto se a entrada for feita na forma esquemática quanto na forma textual de VHDL ou Verilog. Existe no Design Flow uma ferramenta chamada CORE Generator, que oferece um grande leque de funções já otimizadas para linguagens de descrição de hardware. Através desta ferramenta, cria-se um componente NPI desejado e adiciona-se este componente ao projeto.

Após a finalização da etapa de descrição, o usuário pode ainda, antes de partir para as etapas de síntese e implementação, criar restrições de temporização (*timing*) para o sistema, bem como definir o mapeamento dos pinos de entrada e saída do dispositivo. Caso o usuário opte por não criar restrições de *timing*, um arquivo padrão será criado pelo sistema no momento da síntese. O arquivo que determina o mapeamento das portas de entrada e saída do dispositivo deve obrigatoriamente ser elaborado antes da etapa de implementação. Sem a elaboração deste arquivo (de terminação UCF - *User Constraint File*), o processo de implementação não é iniciado, pois é ele que determina para quais pinos do dispositivo as entradas e saídas do sistema serão mapeadas.

O processo de síntese é realizado pelo XST (*Xilinx Synthesis Technology*), uma ferramenta da Xilinx que sintetiza o projeto após sua descrição para criar um arquivo específico de *netlist* chamado NGC (*Netlist File*), que contém tanto dados lógicos

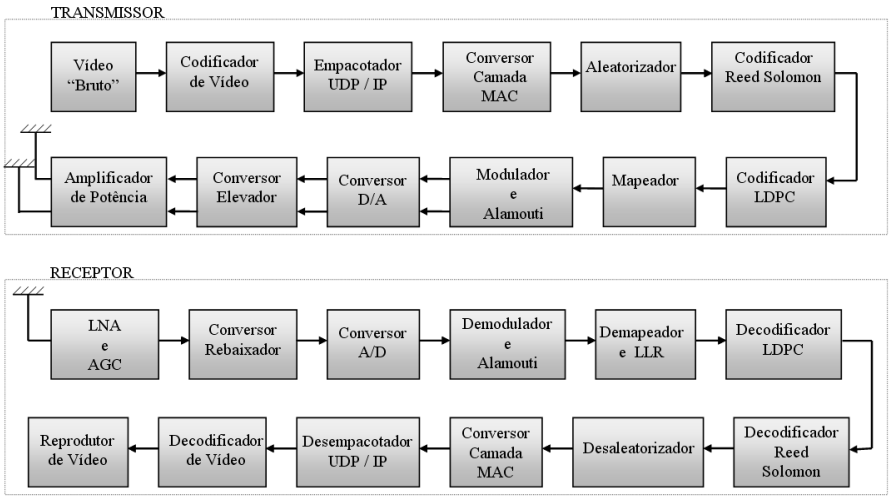


Fig. 1. Sistema de Televisão Digital.

como restrições que levam aos arquivos EDIF (*Electronic Design Interchange Format*) e NCF (*Netware Command File*).

As etapas de síntese e implementação podem ser otimizadas para melhorar o desempenho de velocidade ou para minimizar a área ocupada pelo circuito. A otimização para velocidade prioriza a redução do caminho percorrido por um sinal ao longo de unidades lógicas, possibilitando assim um aumento na freqüência de operação do circuito.

Por outro lado, na otimização para minimização de área, a prioridade é a redução da quantidade de portas lógicas utilizadas para execução do projeto. No final da síntese, a ferramenta XST relata informações de *timing* para o projeto. Vale destacar que existe a possibilidade de se utilizar outras ferramentas de síntese ao invés do XST. Ainda que estas ferramentas não constituam o Design Flow original da Xilinx, elas podem ser perfeitamente integradas ao seu fluxo de desenvolvimento.

O System Generator é a ferramenta de linguagem de mais alto nível que faz parte do Design Flow da Xilinx. A grande vantagem do System Generator, no entanto, está no fato de que seu desenvolvimento se dá sobre a plataforma Simulink do Matlab. No System Generator, a partir do esquema modelado, é possível gerar o código do projeto para implementação em hardware. O System Generator não visa substituir o VHDL ou Verilog, mas ele certamente elimina muitas das dificuldades de projeto ligadas à linguagem, obtendo-se um ganho de produtividade bastante significativo [13].

A implementação em System Generator torna-se especialmente eficiente a partir do uso dos núcleos de propriedade intelectual (NPIs), dos mais simples aos mais complexos, de operações aritméticas até algoritmos complexos de DSP, que já se encontram disponíveis nos *toolboxes* do Simulink.

O ponto de partida na utilização do System Generator é o emprego de dois blocos chamados *Gateway In* e *Gateway Out*. Cada bloco *Gateway In* representa uma interface de entrada do sistema. Em desenvolvimentos e simulações feitas no ambiente Simulink, ele representa a interface entre o universo de ponto flutuante, Simulink, e o universo de ponto fixo, que é efetivamente implementado em hardware. Em termos de implementação, cada bloco *Gateway In* representa uma entrada (fisicamente falando, pinos da FPGA) do dispositivo. No *Gateway In* é necessário informar o tipo de dado de entrada (booleano, ponto fixo com sinal ou ponto fixo sem sinal) e o seu tamanho em bits. Já o *Gateway Out*, por sua vez, representa a interface de saída do universo de ponto fixo para ponto flutuante, quando em simulação no Simulink, e representa pinos de saída no dispositivo (pinos do chip) em termos de implementação.

Todo sistema desenvolvido em System Generator deve obrigatoriamente possuir um bloco também denominado System Generator. Este bloco determina características do projeto como a família do dispositivo a ser utilizado, freqüência de operação do circuito, ferramenta utilizada para síntese, linguagem de descrição de hardware utilizada (VHDL ou Verilog), além do tipo de compilação a ser feita.

O tipo de compilação pode ser, por exemplo, HDL *netlist* ou compilação modo *bitstream*, através da qual, a partir do System Generator, gera-se diretamente o arquivo binário para ser gravado na FPGA. Outra importante opção de compilação é a compilação para co-simulação em hardware. Este recurso de co-simulação em hardware é bastante importante, pois permite que um sistema inteiro seja modelado e apresentado dentro de uma "caixa preta" para ser utilizado no ambiente Simulink, enquanto, na realidade, sua função é executada na FPGA.

A interface de transferência de dados entre a FPGA e o microcomputador pode ser feita, durante a co-simulação em hardware, através de um barramento PCI ou de um cabo JTAG/Paralelo proprietário Xilinx. O que esta compilação faz, na realidade, é gravar o projeto na FPGA e disponibilizar um bloco no ambiente Simulink que representa este projeto. Desta forma, duas grandes vantagens são obtidas. A primeira refere-se ao tempo de simulação. As restrições para processamento de funções complexas passam do processador do computador (CPU) para a FPGA. A segunda vantagem é que este recurso torna possível incorporar uma FPGA diretamente no ambiente Simulink, podendo-se, assim, observar seu efeito dentro de um

sistema. A partir desta co-simulação em hardware, a FPGA acrescentada ao Simulink passa a ser também uma "caixa preta" no projeto. Todo o fluxo de desenvolvimento, a partir do System Generator até a etapa de gravação do programa na FPGA, é apresentado na Figura 2.

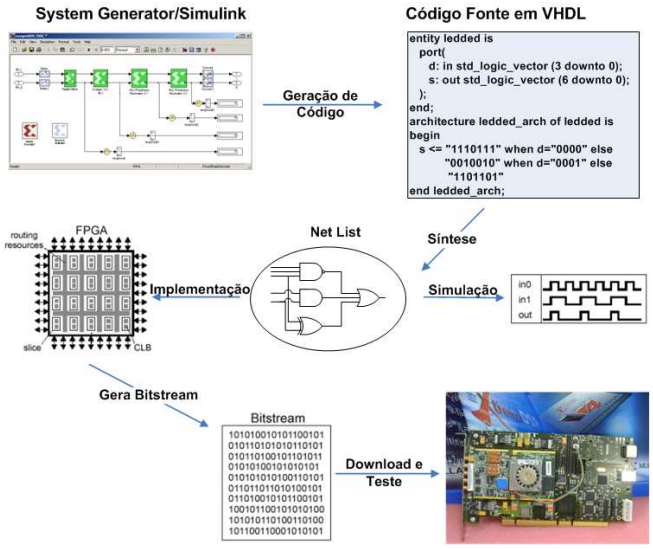


Fig. 2. Fluxo de Desenvolvimento do Design Flow da Xilinx.

## IV. CODIFICAÇÃO LDPC

Para a presente seção são necessárias algumas considerações. Seja $c$ o vetor palavra-código, um vetor coluna de $n$ posições. Assim, a palavra código terá $n$ bits. Seja também $x$ um vetor coluna, de $k$ bits, que representa a mensagem binária a ser codificada e seja $p$ o vetor paridade com $(n-k)$ posições. Para um código irregular, o grau de um dado nó será representado pelo índice $i$. Definindo $\lambda_i$ como a fração dos ramos conectados a nós de bit de grau $i$, a distribuição de graus para nós de bit é definida por

$$\lambda(y) = \sum_{i=2}^{dv} \lambda_i y^{i-1}, \tag{1}$$

onde $y$ é uma variável espectral e $dv$ representa o maior grau de nós de bit.

De forma análoga, para os ramos conectados aos nós de cheque (número de 1's das linhas da matriz $\mathbf{H}$) usa-se a variável $\rho_i$ para representar a fração dos ramos ligados aos nós de cheque de grau $i$, tal que

$$\rho(y) = \sum_{i=2}^{dc} \rho_i y^{i-1}, \tag{2}$$

onde $dc$ é o maior grau de nós de cheque.

Como descrito na sessão introdutória, a obtenção da matriz de paridade $\mathbf{H}$ segue a metodologia usada em [5][6][7]. Inicialmente, a matriz $\mathbf{H}$ deve ser dividida em duas matrizes $\mathbf{H_1}$ e $\mathbf{H_2}$, tais que

$$\mathbf{H} = [\mathbf{H}_1 \ \mathbf{H}_2]. \tag{3}$$

A matriz $\mathbf{H_1}$ possui dimensão $(n-k) \times k$ e será gerada aleatoriamente de forma a atender a distribuição necessária de graus. Já a matriz $\mathbf{H_2}$, de dimensão $(n-k) \times (n-k)$, é dada por

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{bmatrix}. \tag{4}$$

Uma vez que a estrutura de $\mathbf{H_2}$ é conhecida, o projeto de um código LDPC eIRA consiste em duas tarefas: otimização das distribuições dos graus de $\lambda_i$ e $\rho_i$ e a construção da matriz $\mathbf{H_1}$. A otimização de $\lambda(y)$ e $\rho(y)$ é uma tarefa com um alto custo computacional e nem um pouco trivial. Em geral, essas otimizações empregam técnicas complexas, entre as quais se destaca a técnica de evolução de densidade apresentada em [14]. Para a obtenção da matriz $\mathbf{H_1}$, deve-se recordar que a mesma tem dimensão $(n-k) \times k$. O primeiro passo para a sua construção, segundo [14], é criar uma matriz $\mathbf{M}$, submatriz de $\mathbf{H_1}$ e $m$ vezes menor do que ela, de dimensão $a \times b$. O fator $m$ determinará o maior grau de paralelização que poderá ser implementado. A matriz $\mathbf{M}$ não apenas deverá ser criada respeitando a otimização de $\lambda_i$ e $\rho_i$ como também evitando ciclos de baixa ordem, especialmente de grau quatro, conforme descrito previamente. De posse da matriz $\mathbf{M}$, cada um de seus elementos nulos será substituído por uma matriz nula de ordem $m$ e cada elemento '1' de $\mathbf{M}$ deverá ser substituído por permutações da matriz identidade de ordem $m$, construindo-se assim uma nova matriz $\mathbf{S}$ que possui a mesma dimensão de $\mathbf{H_1}$. Assim, de posse de $\mathbf{S}$, monta-se finalmente a matriz $\mathbf{H_1}$ através de

$$\mathbf{H}_1 = \begin{bmatrix} S_1 \\ S_{m+1} \\ \vdots \\ S_{(a-1)m+1} \\ S_2 \\ S_{m+2} \\ S_{(a-1)m+2} \\ \vdots \\ S_{n-k} \end{bmatrix}, \tag{5}$$

onde $s_i$ é a i-ésima linha da matriz $\mathbf{S}$.

Finalmente, uma vez obtida a matriz $\mathbf{H_1}$, a matriz de paridade $\mathbf{H}$ pode ser calculada através da Equação 3. É importante salientar que as técnicas citadas anteriormente diminuem significativamente o aparecimento de ciclos curtos, embora estes não sejam totalmente eliminados. Tendo em vista que $[x^T \ p^T]$ é uma palavra código, portanto ortogonal a $\mathbf{H}$, tem-se

$$[\mathbf{H}_1 \ \mathbf{H}_2] \begin{bmatrix} x \\ p \end{bmatrix} = 0, \tag{6}$$

que resulta em $\mathbf{H}_1 x = \mathbf{H}_2 p$.

Finalmente, a paridade $p$ é dada por

$$p = \mathbf{H}_2^{-1}\mathbf{H}_1 x. \quad (7)$$

Tendo em vista que $\mathbf{H_2}$ assume a forma mostrada na Equação 4, seu inverso é uma matriz triangular inferior e sua implementação pode ser obtida com um acumulador simples. Desta forma, a estrutura do codificador pode ser representada como na Figura 3.

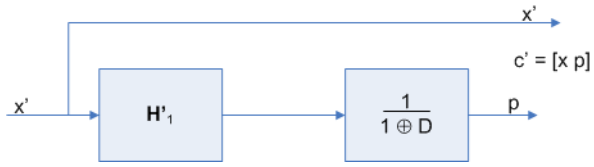


Fig. 3. Codificador LDPC.

Uma vez tendo-se implementado $\mathbf{H_2^{-1}}$ através de um acumulador, o grande obstáculo da codificação passa a ser a manipulação de $\mathbf{H_1}x$. O processo direto para a codificação é então reduzido à tarefa de armazenar toda a matriz de paridade $\mathbf{H_1}$ em hardware e efetuar a multiplicação pela mensagem a ser codificada, cujo resultado é definido por $q_i$. Posteriormente, aplicando-se uma operação de ou-exclusivo (*xor*) de $q_i$ com $q_{i-1}$ obtém-se a paridade $p_i$, dada então por

$$p_i = q_i \oplus q_{i-1}. \quad (8)$$

Um problema bastante expressivo é que a matriz $\mathbf{H_1}$, embora esparsa, é significativamente de grande dimensão. Nem sempre é possível armazená-la devido a limitações na quantidade de memórias do dispositivo onde a implementação do código é realizada. Nestes casos, uma técnica bastante difundida é aplicada, onde uma estrutura $\mathbf{T}$, significativamente menor que $\mathbf{H_1}$, é criada de tal forma que $\mathbf{T}$ possa emular a estrutura de $\mathbf{H_1}$ através de simples permutações. Ao invés de armazenar toda a matriz $\mathbf{H_1}$, é necessário apenas registrar a estrutura $\mathbf{T}$, a qual armazena os índices dos nós de cheque conectados aos $k/m$ nós de bit indexados pelos múltiplos de $m$. Dessa forma, a cada bloco de $m$ bits da mensagem de $k$ bits a ser codificada, a estrutura $\mathbf{T}$ é permutada.

## V. IMPLEMENTAÇÃO E RESULTADOS

O primeiro passo para a implementação do codificador é a definição do próprio código. Baseado nas características de robustez frente ao ruído e às interferências presentes no canal, necessárias a um sistema de televisão digital, o comprimento de código foi definido em 9792 bits. A taxa de código escolhida para implementação do protótipo foi de $3/4$, ou seja, dos 9792 bits da palavra-código, 7344 bits são de informação enquanto 2448 são bits de paridade. De posse do comprimento e taxa do código e adotando o descrito na Seção IV, sabe-se que a matriz de paridade $\mathbf{H}$ terá dimensão $2448(n-k) \times 9792(n)$, sendo a matriz $\mathbf{H}_1$ uma matriz de dimensão $2448(n-k) \times 7344(k)$ e $\mathbf{H}_2$ uma matriz de dimensão $2448(n-k) \times 2448(n-k)$.

Dando continuidade à etapa de projeto do código, determinou-se o grau máximo de paralelização $m$ como sendo igual a 51. Isso significa dizer que no máximo 51 ramos podem ser processados em paralelo. Essa determinação baseia-se no modelo apresentado em [15], no qual também encontram-se expostos os critérios utilizados para a otimização de $\lambda(y)$ e de $\rho(y)$. O grau máximo de paralelização $m$ foi determinado para a construção da matriz $\mathbf{M}$ respeitando a distribuição de $\lambda_i$ e de $\rho_i$ e também respeitando os ciclos de baixa ordem. Essa matriz foi desenvolvida por dois dos autores (Lopes e Pegoraro) de [15] que forneceram a estrutura $\mathbf{T}$, definida na Seção IV, que representa a tabela compacta a partir da qual pode-se montar a matriz $\mathbf{H}_1$. A estrutura $\mathbf{T}$ é de tamanho equivalente à quantidade de bits '1' contidos na matriz $\mathbf{M}$, porém o que ela armazena de fato são as posições em que estes bits se encontram na matriz $\mathbf{H}_1$. A estrutura $\mathbf{T}$ possui 720 elementos, de onde se pode concluir que a matriz $\mathbf{H}_1$ possui $720 \times 51$ elementos não nulos, ou seja, 36720.

Uma vez definidos os parâmetros do código, se considerou duas formas de desenvolvimento para implementar a multiplicação de $x$ pela matriz transposta de $\mathbf{H}_1$.

A primeira forma consiste em armazenar a estrutura $\mathbf{T}$ em memória, processando grupos de $m = 51$ colunas de $\mathbf{H}_1$, através de permutações das informações das linhas de $\mathbf{T}$. Uma linha da estrutura $\mathbf{T}$ apresenta a informação sobre a localização dos bits '1' em uma coluna da matriz $\mathbf{H}_1$. Porém, através de permutações implementadas no algoritmo do codificador, essa mesma informação possibilita estimar a posição dos bits '1' nas 50 colunas subseqüentes, baseando-se no grau de paralelismo $m = 51$, utilizado para a otimização de $\lambda_i$ e $\rho_i$, que deram origem à matriz $\mathbf{H}_1$.

A segunda alternativa à utilização da estrutura $\mathbf{T}$, consiste em fazer uso de uma estrutura maior, de 36720 elementos, que já contenha de uma forma direta todas as posições dos bits '1' ao longo da matriz $\mathbf{H}_1$. Esta proposta é vantajosa em um cenário no qual blocos de memória estão disponíveis no dispositivo de hardware onde o código será implementado e quando se visa um incremento em velocidade de processamento e diminuição da área de silício ocupada, uma vez que as permutações das linhas da estrutura $\mathbf{T}$ não serão mais necessárias. Assim, a segunda alternativa foi a efetivamente implementada neste trabalho e será apresentada a seguir.

A Figura 4 apresenta o codificador LDPC implementado. O algoritmo do codificador foi desenvolvido através de um bloco no System Generator denominado M-Code, o qual possibilita a descrição de algoritmos em linguagem Matlab usando bibliotecas de ponto fixo da Xilinx. A estrutura do codificador desenvolvido em System Generator pode ser observada na Figura 4. Associado ao bloco que executa a codificação (bloco no lado esquerdo da Figura 4) encontram-se dois blocos de memória. O primeiro deles possui 36720 posições, localizado no canto inferior direito da figura, e armazena as posições dos elementos não nulos na matriz $\mathbf{H}_1$. O segundo bloco de memória, localizado na parte direita da Figura 4, possui 2448 posições e armazena os 2448 bits resultantes da multiplicação de $x$ pela matriz transposta de $\mathbf{H}_1$, valor este para o qual adotou-se a nomenclatura $q_i$.

A Implementação do algoritmo de codificação pode ser melhor compreendida através de seu respectivo diagrama de máquina de estado exposto na Figura 5.

O hardware utilizado na implementação do codificador

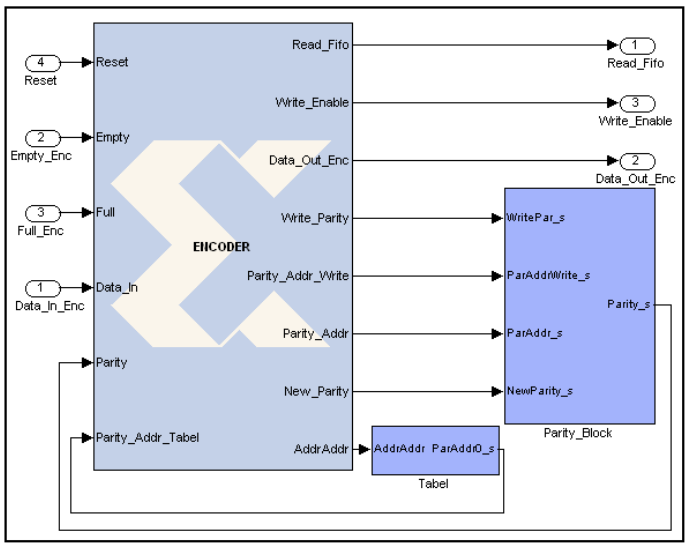


Fig. 4. Codificador LDPC Implementado.

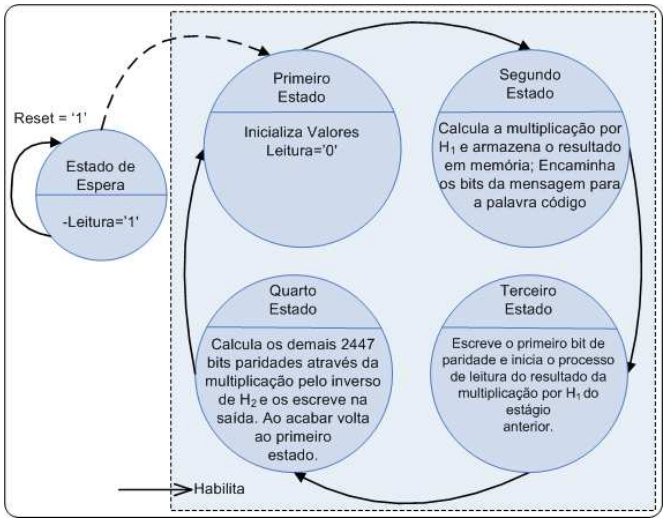


Fig. 5. Máquina de Estados do Codificador LDPC.

foi uma placa de desenvolvimento ML-402 [16], que utiliza uma FPGA Virtex-4 $XC4VSX35$ [17]. O codificador LDPC ocupou apenas 153 dos 15360 *slices* (onde *slice* é uma forma proprietária da Xilinx para medida de elementos lógicos do dispositivo) disponíveis na FPGA. Contudo, na implementação de todo o sistema de TV digital proposto, outras partes do mesmo puderam ser implementadas nesta mesma placa ML-402.

## VI. Conclusões

Este artigo apresentou uma implementação em hardware para um codificador LDPC em um sistema de televisão digital, utilizando ferramentas de prototipagem rápida para FPGAs. No esquema implementado, o codificador seguiu os preceitos de um código eIRA (*extended Irregular Repeat Accumulate*), com palavra-código de 9792 bits de comprimento e taxa 3/4. Ênfase especial foi dada às ferramentas de prototipagem rápida empregadas no desenvolvimento. Um dos aspectos importantes abordados neste trabalho refere-se à união da etapa de concepção do código com sua efetiva implementação em um dispositivo de hardware. Recentemente, dois pontos-chave têm recebido especial atenção na literatura. No primeiro deles enfatiza-se que as FPGAs têm sido uma excelente solução para a implementação em hardware, especialmente nas etapas de desenvolvimento e de prototipagem. No segundo ponto demonstra-se que as ferramentas de projeto vêm sendo aprimoradas para proporcionar soluções de desenvolvimento em linguagem de alto nível e que propiciam altos ganhos de produtividade. O presente trabalho procurou enfatizar essas propostas de desenvolvimento e tendências de novas tecnologias de projeto, apresentando resultados concretos através de uma implementação em FPGA.

## Referências


[1] R. G. Gallager; *"Low-density parity-check codes"*; IRE Transactions on Information Theory, vol. IT-8, pp. 21-28, January of 1962.

[2] R. M. Tanner; "A recursive approach to low-complexity codes"; *IEEE Trans. Information Theory, pp. 533-547, September of 1981.*

[3] D. MacKay; "Good error-correcting codes based on very sparse matrices"; *IEEE Trans. Information Theory, pp. 399-431, March of 1999.*

[4] M. Luby; M. Mitzenmacher; A. Shokrollahi; D. Spielman; "Improved low-density parity-check codes using irregular graphs"; *IEEE Transactions on Information Theory, vol.47, pp.585-598, February of 2001.*

[5] M. Yang; W. Ryan; and Y. Li; "Design of efficiently encodable moderate-length high-rate irregular LDPC codes"; *IEEE Trans. Communications, vol. 52, nº 4, pp. 564-571, April of 2004.*

[6] Y. Zhang; W. Ryan; and Y. Li; "Structured eIRA codes with low floors"; *Proceedings of the International Symposium on Information Theory - ISIT2005, pp. 74-178, September of 2005.*

[7] ETSI (European Telecommunications Standards Institute); "DVB-S.2 Standard Specification"; *Disponível em http://webapp.etsi.org/action/PU/20050322/en 302307v010101p.pdf. Acessado em junho de 2006.*

[8] Powell, S.R.; Cesear T.M.; "Rapid Design and Exploration of Signal Processing using a VHDL Model Generator Based Paradigm"; *Second Annual RASSP (Rapid Prototyping of Application Specific Signal Processors) Conference, 1996.*

[9] Ahmad R. S. Bahai and Burton R. Saltzberg; "Multi-Carrier Digital Communications: Theory and Applications of OFDM"; *Kluwer Academic Publishers, 2002.*

[10] K. Welling and M. Rice; "Coded Orthogonal Frequency Division Multiplexing for the Multipath Fading Channel"; *In Proceedings of the International Telemetering Conference, Las Vegas, NV, October 1999.*

[11] S. Alamouti; "A simple transmit diversity technique for wireless communications"; *IEEE J. Select. Areas Comm. , vol 16, no 8, pp. 1451-1458, 1998.*

[12] Xilinx Inc.; "FPGA Design Flow Overview"; *Disponível em http://toolbox.xilinx.com/docsan/xilinx7/help/iseguide/html/ise_fpga_design_flow_overview.htm. Acessado em outubro de 2006.*

[13] Xilinx Inc.; "System Generator For DSP"; *Disponível em http://www.xilinx.com/support/sw_manuals/sysgen_ug.pdf. Acessado em outubro de 2006.*

[14] T. J. Richardson and R. Urbanke; "The capacity of low-density parity check codes under message-passing decoding"; *IEEE Transactions on Information Theory, Vol. 27, nº 2. February of 2001.*

[15] Tarciano F. Pegoraro; Fábio A. L. Gomes; Renato R. Lopes; Roberto Gallo; José S. Panaro; Marcelo C. Paiva; Fabrício C. A. Oliveira and Fabio Lumertz; "Design, Simulation and Hardware Implementation of a Digital Television System: LDPC channel coding"; *IEEE 9th International Symposium on Spread Spectrum Techniques and Applications (ISSSTA), Manaus, Brazil, August 2006.*

[16] Xilinx Inc.; "ML40x Evaluation Platform User Guide"; *version 1.0.1, June of 2006. Disponível em http://www.xilinx.com/bvdocs/userguides/ug210.pdf . Acessado em novembro de 2006.*

[17] Xilinx Inc.; "Virtex-4 User Guide"; *version 1.6, October of 2006. Disponível em http://www.xilinx.com/bvdocs/userguides/ug070.pdf. Acessado em novembro de 2006.*