IMPLEMENTAÇÃO DE UM CIRCUITO DIGITAL INTEGRADO DEDICADO UTILIZANDO LINGUAGEM DE DESCRIÇÃO DE HARDWARE

Denner Paganoti de Almeida[1], Sibilla Batista da Luz França[2]
UFPR-Universidade Federal do Paraná
Curitiba-PR, Brasil
[1]dpaganoti7@gmail.com, [2]sibilla@eletrica.ufpr.br

Resumo— Os códigos corretores de erros permitem detectar e corrigir erros em uma mensagem possivelmente corrompida durante o processo de transmissão ou armazenamento. Para isso, são adicionados à mensagem bits de redundância, permitindo assim, que o decodificador recupere a mensagem original. Embasando-se numa arquitetura de hardware proposta para um desses decodificadores, a qual fundamenta-se em conjuntos da informação, implementou-se um circuito integrado digital dedicado correspondente à um dos blocos componentes desse decodificador. Esse circuito, que faz a função de um escalonador de matrizes, pode-se dizer que é o núcleo desse sistema de decodificação. É ele que fornece as matrizes de saída Gr e Gr0 resultantes do processo, a partir das quais gera-se a mensagem decodificada. Referenciando-se pelas etapas descritas por esse, foram realizados processos de codificação do circuito na linguagem VHDL, síntese, layout e verificações, que ao fim resultaram num ASIC pronto para a fabricação. Todo o processo de design foi feito utilizando a tecnologia bicmos8hp de 130nm.

I. INTRODUÇÃO

Os códigos corretores de erros são utilizados para aumentar a confiabilidade na transmissão e no armazenamento de dados. Devido à interferência e ruído no canal de comunicação e imperfeições na mídia de armazenamento, a mensagem original pode ser corrompida, comprometendo a informação. Para que a mensagem possa ser recuperada, dentro do limite de correção do código, uma série de bits, chamados de bits de paridade (ou redundância) são adicionados à mensagem original. Os códigos corretores de erros normalmente são representados pelo par (n, k), onde k é o número de bits de informação e n o número de bits da palavra-código, portanto $m=n-k$ é o número de bits de redundância. Outro parâmetro importante é a distância de Hamming mínima (d_{min}), que representa o número de posições nas quais duas palavras-código diferem. A utilização desses códigos é representada na figura 1. A mensagem **x** é codificada por um codificador que acrescenta uma série de bits de paridade. Durante o processo de transmissão, a mensagem é corrompida pelo canal ruidoso, que altera seu conteúdo original. Pelo fato de conter uma série extra de bits, que foram adicionados no processo de codificação, essa consegue ser restaurada pelo decodificador através de uma série de operações lógicas e aritméticas.

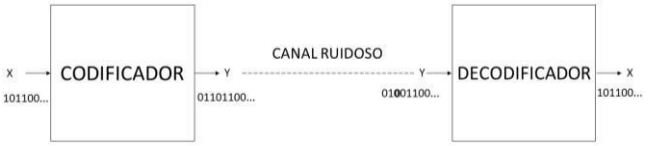

Fig. 1. *Funcionamento típico de um código corretor de erros [1].*

Diversos algoritmos que realizam a decodificação desse tipo de código foram estudados na literatura nos últimos anos. Dentre esses, existe um que, buscando uma implementação otimizada em hardware, propõe a construção de um decodificador para códigos de blocos baseado em conjuntos de informação [2]. Esses conjuntos, que são a base desse sistema, podem ser definidos como o agrupamento de mensagens candidatas baseando-se nos símbolos que tenham maior chance de estarem corretos. Os processos de decodificação que se embasam nessa abordagem, acabam por escolher como a mensagem decodificada, aquela que contida nesse conjunto, possui o maior número de símbolos considerados confiáveis em relação à uma métrica estabelecida pelo decodificador.

[2] Utilizando desses processos de decodificação baseados em conjuntos de informação, propôs-se uma arquitetura de um decodificador composta por cinco blocos, representado na figura 2. Esse realiza a decodificação da seguinte forma: a partir de uma mensagem analógica **x** que é recebida na entrada do decodificador, são extraídas uma mensagem **r** através de um processo de decisão abrupta (demodulação) e um vetor de confiabilidade **s**, cujos elementos são ordenados do mais ao menos confiável. A partir desse vetor de confiabilidade são realizadas operações de escalonamento na matriz geradora binária **G**, baseando-se no método de Gauss-Jordan, resultando em uma matriz chamada de **Gr**, que é **G** escalonada e em uma outra matriz **Gr0**, composta pelos pivôs utilizados durante as eliminações. A partir dessas duas matrizes, que sofrem uma série de operações até o fim do processo, são geradas algumas mensagens candidatas e posteriormente, algumas

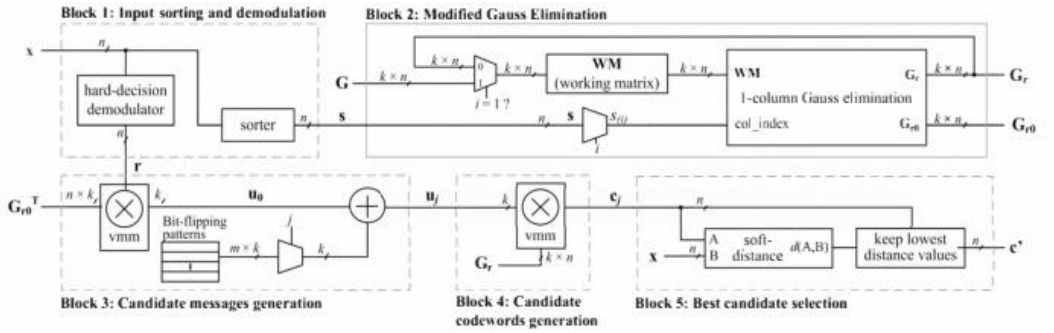

Fig.2. *Decodificador baseado em conjuntos da informação [2]*.

palavras-código candidatas. A mensagem vencedora é determinada comparando-se as palavras-código recebidas com a mensagem **x**, escolhendo-se a que apresentar maior semelhança.

Tendo conhecimento da arquitetura desse decodificador, implementou-se um circuito digital integrado dedicado referente à um dos seus blocos componentes. O bloco escolhido foi o de número dois do conjunto que realiza as operações de escalonamento na matriz geradora binária **G**, a partir da ordem estabelecida pelos elementos contidos no vetor de confiabilidade **s** de *n* posições. Essa matriz **G** tem dimensões *(n,k,d)*, onde *n* é o número de colunas, *k* o número de linhas e *d* a distância de Hamming, e é do tipo **G=[I|P]**, onde **I** é uma matriz identidade *kxk* e **P** é uma matriz de paridade *kx(n-k)*. O processo de escalonamento que é aplicado sobre ela é do tipo Gauss-Jordan e resulta ao final de *n-d+1* ciclos de *clock* em duas matrizes. Uma **Gr**, que é a matriz **G** escalonada com *k* colunas linearmente independentes e **Gr0**, que é uma matriz de mesmas dimensões de **G**, que é zerada em todas as suas posições, exceto nas posições dos pivôs usados no escalonamento, onde recebe elementos iguais à '1'.

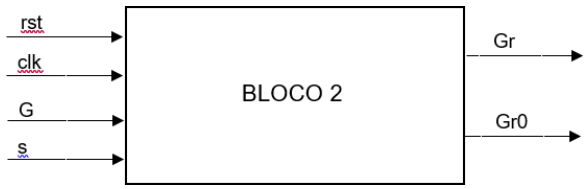

Fig.3. *Diagrama de blocos do escalonador*.

Utilizando dos conhecimentos do algoritmo utilizado no bloco 2 e do fluxo de projetos VLSI, foi possível implementar um circuito integrado digital dedicado correspondente ao escalonador. Devido à alta concentração e à miniaturização dos transistores nos circuitos digitais, fez-se necessário utilizar uma técnica que descreve-se todos os passos envolvidos no processo de design de um chip. Esses passos, conhecidos como fluxo de projeto VLSI, englobam todos os processos de concepção de um *ASIC* desde as definições das suas funcionalidades até sua fabricação numa *foundry*. Começando pela codificação do circuito numa HDL (linguagem de descrição de hardware), esse fluxo continua por etapas de síntese, geração de *layout*, simulações e verificações pós-*layout* e construção. Na etapa de síntese, o código é transformado num circuito RTL, que demonstra as funcionalidades do que vai ser implementado. A partir da netlist, que resulta desse processo, é gerado o *layout* padrão que compõe o projeto são posicionadas na futura área de sílicio, os barramentos de alimentação são estabelecidos, os pinos de entrada e saída são inseridos e ocorre o roteamento entre todos esses elementos. Na próxima etapa, são feitas as verificações de *layout*, para comprovar se o desenho desse ocorreu corretamente. Com tudo correto, simula-se, e comprovado que o circuito executa corretamente suas funções, faz-se seu envio à uma *foundry* para fabricação. Utilizando desse método de design, da linguagem VHDL e da tecnologia bicmos8hp, projetou-se o C.I. digital correspondente ao algoritmo realizado pelo bloco escalonador de Gauss-Jordan do decodificador da figura 2.

II. DESENVOLVIMENTO E RESULTADOS

No processo de design de um circuito integrado digital, onde segue-se os passos do fluxo VLSI, a primeira etapa é a implementação do código correspondente ao circuito em linguagem descrição hardware. Utilizando o VHDL, implementou-se um algoritmo que realizava as funções de escalonamento de Gauss-Jordan inerentes ao bloco 2, para matrizes de dimensão genérica.

Basicamente, esse algoritmo realizava o seguinte: a partir do vetor **s**, usava-se como pivô a posição definida por *Lx**s**(L)*, onde *L* é o índice de linha, na matriz **G**. Nessa posição, verificava-se se o elemento contido era igual à '1'. Caso não fosse, verificava-se as posições abaixo do pivô na coluna ***s**(L)*, e se um '1' fosse encontrado, era feita a troca entre as linhas. Se nenhuma das posições abaixo fosse '1', avançava-se à próxima coluna indicada por ***s**(L)*, tendo então o pivô o endereço *Lx**s**(L+1)*. Garantido a posição como '1', é feita uma operação XOR entre a linha do pivô e a de todos os elementos contidos em ***s**(L)* cujo valor é '1', garantindo que ao final ***s**(L)* seja uma coluna linearmente independente. Com todos os elementos da coluna zerados, exceto o pivô, *L* é incrementado em 1 e a matriz **Gr0**, que é composta inicialmente só por zeros, recebe '1' na posição pivoteada. Todo esse processo deve ocorrer *k* vezes, dentro de *n-d+1* ciclos de *clock*. Ao final, deve-se ter obtido uma matriz **Gr**, com *k* colunas linearmente independentes, e uma **Gr0**, com essas mesmas colunas e no resto '0'. Abaixo, tem-se ilustrado esse processo realizado numa matriz **G**(7,4,3) e com o vetor **s**=(6,5,7,3,2,1,4). Foi para essa dimensão que o circuito integrado foi implementado. Observa-se que o escalonamento ocorreu nas colunas 6, 5 e 7 da matriz **G**, ficando essas linearmente independentes. Ao tentar realizar o escalonamento em 3, não foi possível encontrar um pivô '1', avançando assim à 2. Ao fim de

5 ciclos de *clock*, obteve-se duas matrizes, **Gr** e **Gr0**, ambas com *k* colunas linearmente independentes, concluindo o processo.

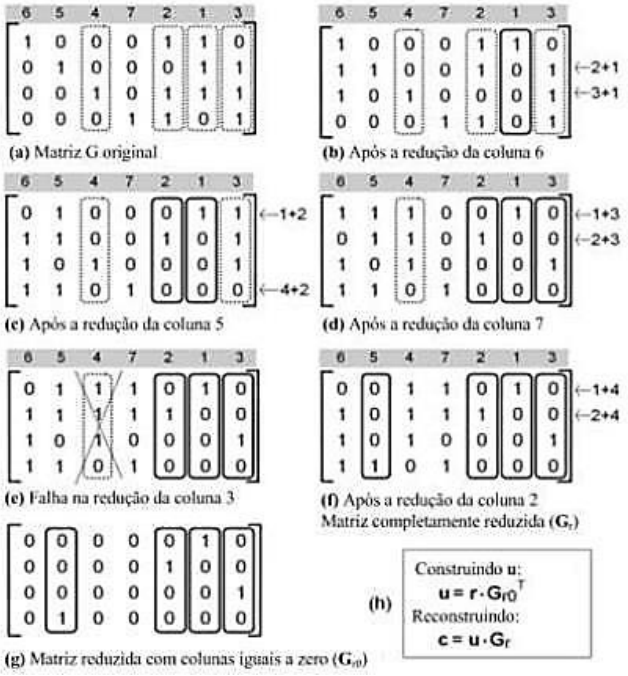

Fig.4. *Algoritmo desempenhado pelo bloco 2 [3]*.

Compreendendo o algoritmo de escalonamento, fez-se sua implementação em VHDL. Utilizando as dimensões citadas no parágrafo acima e do vetor **s**, realizou-se a simulação do código no simulador *ModelSim*, obtendo os resultados da tabela abaixo para um *clock* de período 100ns:

TABELA 1. RESULTADOS DE SAÍDA PARA O CÓDIGO VHDL

	1º clock	2º clock	3º clock	4º clock	5º clock
Gr	1000110	100011	1110010	1110010	0011010
	1100101	1100101	0110100	0110100	1011100
	1010001	1010001	1010001	1010001	1010001
	0001101	1101000	1101000	1101000	1101000

Gr0	0000010	0000010	0000010	0000010	0000010
	0000000	0000100	0000100	0000100	0000100
	0000000	0000000	0000001	0000001	0000001
	0000000	0000000	0000000	0000000	0100000

Nos resultados compilados na tabela, percebe-se que as matrizes **Gr** e **Gr0** apresentam os mesmos resultados da figura 3. É possível ver claramente as colunas que sofreram eliminação em **Gr** (colunas 6,5,7,2) e que os pivôs foram colocados de forma correta em **Gr0**. Assim, analisando esses resultados, percebe-se de forma clara que o código é funcional. Com isso, implementou-se esse código numa FPGA da família *Xilinx Spartan 3E* e obteve- se os resultados em relação ao hardware utilizado mostrado abaixo:

TABELA 2. NÚMERO DE ELEMENTOS UTILIZADOS NUMA FPGA XILINX SPARTAN 3E

	Número de elementos utilizados	Percentual em relação ao total de elementos
Flip Flops	71	1%
LUTs	1103	6%
Slices	583	6%

A partir do código VHDL funcional foi possível iniciar o processo de design do circuito integrado nas ferramentas de CAD da *Cadence*. A primeira etapa foi a síntese do código num circuito RTL, realizada no *Cadence RTL Compiler*. Aqui, o código VHDL do circuito foi sintetizado num circuito RTL, cujo resultado é uma *netlist* que contém as células padrão que serão utilizadas. É a partir dessa que é possível criar o *layout* do circuito nas fases mais avançadas do processo de *design.* Ao realizar a síntese, o circuito resultante teve as características mostradas na tabela abaixo:

TABELA 3. NÚMERO DE CÉLULAS PADRÃO E ÁREA DO CIRCUITO SINTETIZADO

Número de células padrão utilizadas	562
Área ocupada pelas células padrão	5938,56µm²

Simulando o circuito RTL no *ModelSim*, obteve-se para **Gr** e **Gr0** os mesmos resultados da tabela 1. Assim, com o circuito funcionando corretamente, pode-se avançar à geração do *layout* do circuito. Essa etapa foi realizada no *Cadence Encounter*. Aqui, foram adicionados os anéis de alimentação ao circuito, os barramentos de entrada e saída e ocorreu o posicionamento das células padrão da tecnologia, definidas na *netlist* geradas na etapa anterior, que foram roteadas entre si e entre os barramentos. Ao fim desse processo, obteve-se um *layout* do circuito com as características mostradas na tabela abaixo:

TABELA 4. ESTIMATIVAS CONSTRUTIVAS DO LAYOUT

Quantidade de pinos de I/O	79
Número total de camadas utilizadas	16
Quantidade de trilhas utilizadas para roteamento	591
Área total do layout circuito	85452,096µm²
Densidade total do circuito	38,56%

Com o *layout* pronto, foi possível realizar as verificações e simulações pós-*layout* no *Cadence Virtuoso*. O primeiro passo desse processo foi a realização de uma simulação transiente com o esquemático do circuito. Feito isso, cujo resultado foi idêntico aos já simulados, iniciou-se as verificações. Foram aplicadas as verificações de DRC e LVS, sendo que todos os erros apresentados nessas foram corrigidos. Após essas, foi realizada a extração de parasitas e a simulação pós-*layout*. Nessa simulação

constatou-se os mesmos resultados apresentados pelo código e pelo circuito RTL para as matrizes **Gr** e **Gr0**. Assim, o *layout* mostrou-se totalmente funcional e correspondendo aos resultados esperados inicialmente.

Com o circuito do escalonador pronto, foi feito um *padframe* para acomodá-lo. O encapsulamento escolhido foi do tipo dip 40, que conta com 38 pinos de I/O disponíveis, e o tamanho final desse é de 16 mm². Pelo fato de contar com um número menor de pinos do que o circuito do escalonador, foi necessário criar um circuito auxiliar para que fosse possível apresentar as saídas **Gr** e **Gr0**. Assim, criou-se uma espécie de conversor paralelo-serial, que mostra na saída ambas as matrizes, uma linha por ciclo de *clock*. Esse circuito auxiliar foi roteado junto ao principal e então roteou-se esses com o *padframe*. Ao fim, tendo tudo concluído, os resultados de saída para todo o conjunto foram os seguintes:

TABELA 5. RESULTADOS DE SAÍDA DO CONJUNTO PADFRAME, BLOCO 2 E CIRCUITO AUXILIAR

	1º clock	2º clock	3º clock	4º clock
Gr	0011010	1011100	1010001	0001101
Gr0	0000010	0000100	0000001	0100000

Analisando os resultados da tabela, percebe-se que as linhas de cada uma das matrizes, **Gr** e **Gr0**, são apresentadas sequencialmente a cada ciclo de *clock*. Ao fim, com todas as etapas simuladas e com os resultados comprovados funcionais, obteve-se o *layout* do circuito integrado digital dedicado da figura 4 mostrada abaixo. No seu processo de *design* foram utilizados os passos descritos no fluxo VLSI e esse foi projetado com a tecnologia bicmos8hp de 130nm, sendo que essa foi utilizada pela primeira vez no âmbito do GICS-UFPR para um projeto de tal natureza.

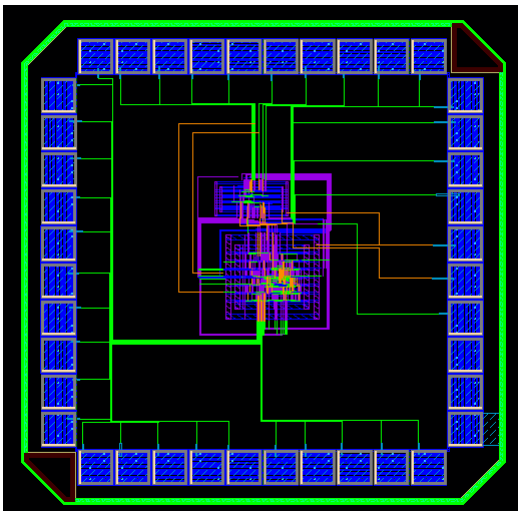

Fig.5. *Circuito resultante finalizado.*

III. CONCLUSÃO

Seguindo todos os passos do fluxo de projeto VLSI, foi possível, a partir de um código em VHDL, realizar todo o processo de design de um circuito digital integrado dedicado, que em tese, estaria pronto para a fabricação. Esse circuito, que em todas as fases do processo construtivo; codificação, síntese, layout e verificações; foi simulado, para que ao fim mostrasse-se funcional, o que de fato ocorreu e pode ser comprovado pela análise dos resultados apresentados. Assim, ao fim de toda a cadeia de processos, tem-se um circuito que pode ser usado de forma plena no sistema para o qual foi originalmente concebido, bem como em outras aplicações que envolvam o escalonamento de matrizes.

REFERÊNCIAS

[1] PEDRONI, Volnei A. *Digital Electronics and Design with VHDL*. Morgan Kaufmann, 2010.

[2] GORTAN, Antônio; JASINSKI, Ricardo P.; GODOY JR., Walter; PEDRONI, Volnei A.. *Achieving near-MLD performance with soft information-set decoders implemented in FPGAs*. Artigo- Departamento de Engenharia Eletrônica, UTFPR, 2010.

[3] FRANÇA, Sibilla Batista da Luz. Desenvolvimento e implementação de chips dedicados para um novo decodificador de códigos corretores de erros baseado em conjuntos de informação. Tese de Doutorado-UTFPR, 2013.

[4] PEDRONI, Volnei A. *Circuit design with VHDL*. Cambridge: MIT Press, 2004.

[5] D'AMORE, Roberto. *VHDL: Descrição e síntese de circuitos digitais*. 1ª ed. Rio de Janeiro: LTC editora, 2005.

[6] BRUNVAND, Erik. Digital VLSI Chip Design with Cadence and Synopsys CAD Tools. 1ª ed. Addison Wesley Longman, 2009.

[7] WESTE, Neil; HARRIS, David Money. *CMOS VLSI Design*. 4ª ed. Addison Wesley Longman, 2011.

[8] SENGUPTA, Anirban. *Design flow of a digital IC*. Artigo-IEEE Consumer Electronics Magazine, 2016.

[9] GORTAN, Antônio; JASINSKI, Ricardo P.; GODOY JR., Walter; PEDRONI, Volnei A.. *An improved GF(2) matrix inverter with linear time complexity*. Artigo-International Conference on Recofigurable Computing, 2010.

[10] KESSLER, A.; GANESAN, A..*An introduction to standard-cell VLSI design.* IEEE Potentials Magazine. 1985.