UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA

"AVALIAÇÃO DE TÉCNICAS DE COMPRESSÃO DE IMAGENS PARA IMPLEMENTAÇÃO EM FPGA DE GRANULARIDADE FINA"

WALTER SOTO ENCINAS JÚNIOR

Tese apresentada ao Instituto de Física de São Carlos, Universidade de São Paulo, para obtenção do título de Doutor em Ciências "Física Aplicada – Opção Física Computacional"

USP/IFSC/SBI
8-2-001361

Orientador:
Prof. Dr. Edson dos Santos Moreira

Comissão Julgadora:
Prof. Dr. Edson dos Santos Moreira (ICMC-USP)
Prof. Dr. Jan Frans Willem Slaets (IFSC-USP)
Prof. Dr. Celso Massaki Hirata (ITA)
Prof. Dr. Valentin Obac Roda (EESC-USP)
Profa. Dra. Tereza Cristina Melo de Brito Carvalho (EP-USP)

São Carlos – São Paulo
2000

IFSC-USP SERVIÇO DE BIBLIOTECA E INFORMAÇÃO

Encinas Júnior, Walter Soto

"Avaliação de técnicas de compressão de imagens para implementação em FPGA de granularidade fina"/ Walter Soto Encinas Júnior – São Carlos, 2000

Tese (Doutorado) – Instituto de Física de São Carlos da Universidade de São Paulo, 2000 – páginas: 151
Área: Física Aplicada-Opção: Física Computacional
Prof. Dr. Edson dos Santos Moreira

1. Transformada Wavelet, 2. FPGA, 3. Processamento de Imagens

I. Título

UNIVERSIDADE
DE SÃO PAULO
Instituto de Física de São Carlos

Caixa Postal 369 - CEP 13560-970 - São Carlos - SP
Fone: (0xx16) 273-9777 - Fax: (0xx16) 273-9777
E-mail: wladerez@if.sc.usp.br

MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE WALTER SOTO ENCINAS JÚNIOR APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 28 DE SETEMBRO DE 2000.

COMISSÃO JULGADORA:

Prof. Dr. Edson dos Santos Moreira/ICMC-SCE-USP

Prof. Dr. Jan Frans Willem Slaets/IFSC-FFI-USP

Prof. Dr. Celso Massaki Hirata/ITA

Prof. Dr. Valentin Obac Roda/EESC-SEL-USP

Profa. Dra. Tereza Cristina Melo de Brito Carvalho/EP-USP

Dedico este trabalho à minha família, com muito amor

Índice

Índice i

Índice de Figuras v

Índice de Tabelas viii

Resumo x

Abstract xi

Abreviaturas Utilizadas xii

1. Introdução 1
   - 1.1. Objetivos 4
   - 1.2. Contribuições 4
   - 1.3. Estrutura do trabalho 5

2. Compressão de imagens 6
   - 2.1. Introdução 6
   - 2.2. Redundância 6
     - 2.2.1. Estatística 6
     - 2.2.2. Perceptiva 8
   - 2.3. Codificação 9
     - 2.3.1. Predição 9
     - 2.3.2. Transformação 10
       - 2.3.2.1 Transformada Karhunen-Loève 11
       - 2.3.2.2 DCT 12
       - 2.3.2.3 Transformada Wavelet 13
     - 2.3.3. Interpolação/Extrapolação 14
     - 2.3.4. Codificação Estatística 14
       - 2.3.4.1 Huffman 14
       - 2.3.4.2 Codificação Aritmética 15
     - 2.3.5. Codificação Baseada em Objetos e Modelos 15
   - 2.4. Quantização 16
   - 2.5. Avaliação 16
   - 2.6. Padrões de compressão 19
   - 2.7. Implementação das Técnicas de Compressão 21
   - 2.8. Implementação em Software 21
   - 2.9. Implementação em Hardware Dedicado 24
   - 2.10. Soluções de Compromisso 25

2.11. Considerações Finais 26

3. Lógicas Reconfiguráveis 27
3.1. Histórico 27
3.2. Arquiteturas 28
3.2.1. Tipos de Programação 29
3.2.2. Arquitetura de Blocos Lógicos 30
3.2.3. Arquitetura de Roteamento 31
3.3. Aplicações 32
3.3.1. FPGAs como processadores VLIW 33
3.3.2. A FPGA XC6200 35
3.4. Paradigmas de aceleração 39
3.4.1. Unidade Funcional 41
3.4.2. Coprocessador 42
3.4.3. Processador 43
3.4.4. Servidor 44
3.5. Considerações Finais 46

4. A Transformada Wavelet 47
4.1. Introdução 47
4.2. A Transformada de Fourier Limitada 48
4.3. Análise Multiresolução 52
4.4. Wavelets Ortonormais 58
4.5. Wavelets Biortogonais 60
4.6. Considerações Finais 61

5. Análise dos Erros de Quantização em Bancos de Filtros Multiescala 63
5.1. Introdução 63
5.1.1. Revisão Estatística 65
5.2. Erro em Estágio Único 67
5.2.1. Quantização de Coeficientes 67
5.2.2. Quantização de Barramento 68
5.3. Erro de Estágios Múltiplos em Cascata 69
5.3.1. Quantização de Coeficientes 70
5.3.2. Quantização de Barramento 72
5.4. Considerações para a Etapa de Análise 75
5.5. Considerações para a Etapa de Síntese 75
5.6. Avaliação 77
5.6.1. D4 79
5.6.2. D6 84

5.7. Considerações Finais 89

6. Alternativas de Implementação 90
6.1. Introdução 90
6.2. Estruturação das Entidades 90
6.2.1. Abordagem *Bit-Slice* 91
6.2.2. Abordagem de Macros Estruturadas 91
6.3. Aritmética Distribuída 92
6.4. Multiplicadores 95
6.5. Processamento Bit-Serial 97
6.6. Somadores 99
6.7. Considerações Finais 105

7. Implementação 107
7.1. Introdução 107
7.2. Fluxo de Projeto 107
7.3. Considerações de Implementação 111
7.4. Bibliotecas Parametrizáveis 112
7.4.1. ADD_TAM_S 113
7.4.2. MUL_ADD_S 113
7.4.3. ROUNDER 113
7.4.4. SAT_S 114
7.5. Filtros FIR 114
7.5.1. FIR1_INI 115
7.5.2. FIR1 115
7.5.3. FIR2 116
7.6. Implementação da Transformada Wavelet 118
7.6.1. Wavelet D4 Sem Multiplicadores 121
7.6.1.1 Transformada Direta 121
7.6.1.2 Transformada Inversa 126
7.6.1.3 Resultados 129
7.6.2. Wavelet D4 Utilizando FIR 130
7.6.3. Wavelet 9/7 Utilizando FIR 132
7.7. Coprocessador de Granularidade Elevada 133
7.8. Considerações Finais 134

8. Conclusão 136
8.1. Análise de Resultados 136
8.2. Contribuições 136
8.3. Originalidade 137

8.4. Trabalhos Futuros 137

9. Bibliografia 139

10. Apêndice 1

10.1. Dados de área e atraso das entidades implementadas 1

10.2. Publicações geradas por este trabalho 5

Índice de Figuras

Figura 2.1: Modelo do Sistema Visual Humano 8
Figura 2.2: Diagrama simplificado da codificação/decodificação segundo padrão MPEG 22
Figura 3.1: Estrutura de um FPGA. 28
Figura 3.2: Elementos de comutação a) transistor como chave e b) multiplexador 30
Figura 3.3: Estrutura do bloco lógico 36
Figura 3.4: Layout de um FPGA XC6216 36
Figura 3.5: Detalhe do roteamento entre blocos lógicos 37
Figura 3.6: Célula básica da família XC6200 38
Figura 3.7: Paradigmas de aceleração com FPGAs 41
Figura 3.8: Topologia de interconexão *mesh* e *crossbar* 45
Figura 4.1: a) Função-janela Gaussiana no domínio do espaço b) Base da STFT no domínio do espaço c) Base da STFT no domínio da frequência 48
Figura 4.2: Região de discriminação da STFT 49
Figura 4.3: Região de discriminação de uma base ideal 50
Figura 4.4: Linhas de referência na figura Lena 50
Figura 4.5: Luminância nas linhas de referência primeira, intermediária e última 51
Figura 4.6: Comportamento das bases $h_{a,\tau}(x)$ em várias escalas 51
Figura 4.7: Comportamento das bases $s(x)$ em várias frequências 52
Figura 4.8: Diagrama do processo de análise 56
Figura 4.9 Diagrama de análise com bancos de filtros 56
Figura 4.10: Diagrama do processo de síntese 57
Figura 4.11 Diagrama de síntese com bancos de filtros 57
Figura 4.12: Função wavelet de Daubechies-4 59
Figura 4.13: Função wavelet de Daubechies-20 59
Figura 5.1: Esquema de um bancos de filtros para DWT 64
Figura 5.2: Estrutura de um filtro FIR 67
Figura 5.3: Modelo estatístico da quantização de barramento 69
Figura 5.4: Filtro composto por N filtros em cascata 69
Figura 5.6: Modelo estatístico da etapa de síntese de um sistema com M bandas 76
Figura 5.7: Caminho do sinal na transformada wavelet que introduz mais erros de quantização 78
Figura.5.8: Legenda para gráficos de erro de quantização 80
Figura.5.9: "Lena" processada com 3 escalas 80
Figura.5.10: "Camera" processada com 3 escalas 81
Figura 5.11: Variável aleatória sem autocorrelação processada com 3 escalas 81
Figura 5.12: Variável aleatória com autocorrelação de 0.95 processada com 3 escalas 82
Figura.5.13: "Lena" processada com 4 escalas 82

Figura.5.14: "Camera" processada com 4 escalas....83
Figura 5.15: Variável aleatória sem autocorrelação processada com 4 escalas....83
Figura 5.16: Variável aleatória com autocorrelação de 0.95 processada com 4 escalas....84
Figura 5.17: "Lena" processada com 3 escalas....85
Figura 5.18:"Camera" processada com 3 escalas....85
Figura 5.19: Variável aleatória sem autocorrelação processada com 3 escalas....86
Figura 5.20: Variável aleatória com autocorrelação de 0.95 processada com 3 escalas....86
Figura 5.21: "Lena" processada com 4 escalas....87
Figura 5.22:"Camera" processada com 4 escalas....87
Figura 5.23: Variável aleatória sem autocorrelação processada com 4 escalas....88
Figura 5.24: Variável aleatória com autocorrelação de 0.95 processada com 4 escalas....88
Figura 6.1: FIR utilizando aritmética distribuída....93
Figura 6.2: LUT para a) 2 b) 3 c) 4 bits....94
Figura 6.3 Multiplicador utilizando aritmética distribuída....96
Figura 6.4: Somador bit-serial....98
Figura 6.5: Área e performance de somadores implementados em XC4000 {Xing e Yu, 1998} 100
Figura 6.6: Somador *Carry-Skip*....101
Figura 6.7: Célula básica do somador completo prevendo *carry-skip*....102
Figura 6.8: Pior caso de propagação do *carry* em somador *carry-skip*....103
Figura 6.9: Tamanho ótimo dos grupos de somadores em somador *carry-skip*....104
Figura 6.10: Ganho de performance do somador *carry-skip* sobre o *ripple-carry*....105
Figura 7.1: Esquema do fluxo de projeto....109
Figura 7.2: Simulação de um filtro FIR....110
Figura 7.3: Detalhe da simulação de um filtro FIR....110
Figura 7.4: Utilização de componentes otimizando a) área b) atraso de roteamento....111
Figura 7.5: Tamanhos de contador e registrador de deslocamento....112
Figura 7.6: Diagrama do somador....113
Figura 7.7: Diagrama do multiplicador por constante....113
Figura 7.8: Diagrama da entidade de arredondamento....114
Figura 7.9: Diagrama da entidade de saturação....114
Figura 7.10: Estrutura do estágio inicial do filtro FIR....115
Figura 7.11: Estrutura do estágio intermediário do filtro FIR....116
Figura 7.12: Construção de filtro FIR utilizando as entidades FIR_INI e FIR1....116
Figura 7.13: Estrutura do filtro FIR simétrico....117
Figura 7.14: Diagrama geral do sistema de transformada wavelet....119
Figura 7.15: Fluxo dos dados para WT direta na arquitetura de {Lewis e Knowles, 1991}....121
Figura 7.16: Multiplicador por 3, 5, 11, 19....122
Figura 7.17: Arquitetura do processador para WT direta....123

Figura 7.18: *Lay-out* da implementação da WT direta no FPGA XC6216 125
Figura 7.19: Distribuição de entidades da WT direta no FPGA XC6216 126
Figura 7.20: Fluxo dos dados da WT inversa na arquitetura de {Lewis e Knowles, 1991}..... 126
Figura 7.21: Arquitetura do processador para WT inversa 127
Figura 7.22: *Lay-out* da implementação da WT inversa no FPGA XC6216 128
Figura 7.23: Distribuição de entidades da WT inversa no FPGA XC6216 128
Figura 7.24: "Lena" original 129
Figura 7.25: "Lena" comprimida usando D4 e 3 escalas 130
Figura 7.26: "Lena" comprimida usando D4 e 6 escalas 130
Figura 7.27: *Lay-out* da implementação do filtro H_0 da WT D4 no FPGA XC6216 131
Figura 7.28: Distribuição de entidades do filtro H_0 da WT D4 no FPGA XC6216 132
Figura 7.29: *Lay-out* da implementação do filtro H_0 da WT 9/7 no FPGA XC6216 133
Figura 7.30: Processador wavelet D4 e unidades de acesso a memória e controle no FPGA XC6216 134

IFSC-USP SERVIÇO DE BIBLIOTECA E INFORMAÇÃO

Índice de Tabelas

Tabela 2.1: Padrões de compressão de imagens estáticas 20
Tabela 2.2: Padrões de compressão de imagens dinâmicas 20
Tabela 2.3: Operações necessárias para descompressão vídeo de qualidade NTSC 22
Tabela 2.4: Operações necessárias para descompressão vídeo de qualidade NTSC, utilizando extensões multimídia 23
Tabela 2.5: Operações necessárias para compressão de imagem VGA 23
Tabela 3.1: Características de reconfiguração de FPGAs comerciais 35
Tabela 4.1: Coeficientes $\mathbf{h_0}$ de algumas wavelets ortonormais 60
Tabela 4.2: Coeficientes $\mathbf{h_0}$ e $\mathbf{g_0}$ de algumas wavelets biortogonais 61
Tabela 6.1: Funções booleanas para implementação de LUT de 2 entradas 94
Tabela 7.1: Complexidade de várias arquiteturas de wavelet propostas 120
Tabela 7.2: Área e desempenho da entidade multiplicador 122

Resumo

Este trabalho analisa a viabilidade de implementação de um sistema de compressão de vídeo em FPGAs com restrições de área e de recursos de roteamento, de modo a estudar as possibilidades oferecidas por pequenas áreas de lógica reconfigurável associadas a processadores clássicos. A técnica de eliminação de redundância espacial adotada é a Transformada Wavelet (WT), ao invés da já bastante explorada Transformada Cosseno (DCT). A arquitetura selecionada para efetuar a DWT é implementada no FPGA Xilinx XC6200, que apresenta características muito interessantes de reconfiguração rápida e dinâmica. Também foi desenvolvida um modelo estatístico dos erros de quantização associados à implementação da Transformada Wavelet em aritmética de ponto fixo, auxiliando uma implementação em hardware mais eficiente em termos de área e desempenho. Algumas arquiteturas alternativas para a DWT são implementadas, para avaliação dos méritos de cada uma delas. Para permitir esta implementação, uma biblioteca de funções para processamento de sinais otimizada para o FPGA XC6200 também foi desenvolvida. Os resultados mostram que um sistema completo de compressão, utilizando os filtros propostos por Daubechies pode ser implementado utilizando o equivalente a 2500 portas lógicas e atinge taxas de compressão de aproximadamente 8 vezes sem perda significativa de qualidade em imagens monocromáticas de 256x256 *pixels* a 30 quadros por segundo.

Abstract

The purpose of this work is to study the problems associated with the implementation of a image compression system implemented with fine-grained FPGAs with scarce routing resources. The study intends to explore possibilities offered by small areas of reconfigurable logic coupled to classical processors. The image compression technique adopted was Discrete Wavelet Transform (DWT), instead the well-known Discrete Cosine Transform. The architecture was built in a Xilinx FPGA, the XC6200, a very interesting device due to its dynamic reconfigurable features. It was also developed a study about quantization errors in DWT processed with fixed point arithmetic. Some other DWT architectures are presented and the advantages are discussed. The implementation of these architectures led to the development of signal processing libraries optimized to XC6200. The implementation was made with only 2500 gates and showed that is possible to reach video rates. Frame sizes are 256x256 with 256 gray levels, and the compression rate is about 8 times.

Abreviaturas Utilizadas

ALU ..... Arithmetic Logic Unit

ASIC ..... Application Specific Integrated Circuit

CCM ..... Custom Computing Machines

CPLD ..... Complex Programmable Logic Device

CSD ..... Canonic Signed Digit

DCT ..... Discrete Cosine Transform

DFT ..... Discrete Fourier Transform

DM ..... Delta Modulation

DPCM ..... Diferential Pulse Code Modulation

DSP ..... Digital Signal Processor

DWT ..... Discrete Wavelet Transform

EDA ..... Electronic Design Automation

EDIF ..... Electronic Data Interchange Format

EEPROM ..... Electrically-Erasable Programable Read Only Memory

EPROM ..... Erasable Programmable Read Only Memory

FIR ..... Finite Impulse Response

FPGA ..... Field-Programmable Gate Array

IDCT ..... Inverse Discrete Cosine Transform

IMC ..... Inverse Motion Compensation

IQ ..... Inverse Quantization

ISDN ..... Integrated Services Data Network

ISO ..... International Standards Organization

ISP ..... In-System Programmability

IVLC ..... Inverse Variable Length Code

JPEG ..... Joint Picture Expert Group

KLT ..... Karhunen-Loève Transform

LSB ..... Lower Significant Bit

LUT ..... Look-Up Table

MAC ..... Multiply and Accumulate

MAD ..... Mean Absolute Difference

MC ..... Motion Compensation

MIMD ..... Multiple Instruction Multiple Data

MIPS ..... Millions Instructions per Second

MPEG ..... Motion Picture Expert Group

MPGA ..... Mask Programmable Gate Array

NTSC ..... National Television System Commitee

PCI ..... Peripheral Component Interconnect

PLD ..... Programmable Logic Device

PSNR ..... Peak Signal-Noise Relation

RAM ..... Random Access Memory

RISC ..... Reduced Instructions-Set Code

RMS ..... Root Mean Square

RMSE ..... Root Mean Square Error

ROM ..... Read Only Memory

RPU ..... Reconfigurable Processing Unit

SDF ..... Standard Delay Format

SIMD ..... Single Instruction Multiple Data

SNR ..... Signal-Noise Relation

SOC ..... System On a Chip

SRAM ..... Static RAM

1. Introdução

A quantidade de informação armazenada, transmitida ou processada de alguma forma por computadores tem crescido exponencialmente nos últimos 10 anos. A principal causa disso é a popularização de sistemas e aplicações multimídia {Sweldens, 1995}, causada pelo aumento da capacidade de processamento e armazenamento dos computadores pessoais. A era onde computadores processavam apenas texto e números foi substituída por uma outra onde o computador se tornou também uma ferramenta de comunicação e deve armazenar, transmitir e processar imagens estáticas, em movimento, sons, etc. O conjunto de todos estes meios de representação de informação é chamado de informação multimídia.

Imagens são uma classe de informação multimídia muito exigente em termos capacidade de armazenamento e banda para transmissão. Por exemplo

- uma foto 35 mm digitalizada com qualidade (resolução de 12 μm) exige 18 MBytes
- um segundo de vídeo colorido padrão NTSC exige 23 MBytes

Daí compreende-se porque compressão de imagens é um tema central na área de processamento de sinais e teoria de informação {Cohen e Ryan, 1995}.

Entre as várias técnicas existentes para a compressão de imagens estáticas, o uso de transformadas tem sido cada vez mais frequente, ocupando um espaço importante ao lado das tradicionais técnicas de predição. Mas transformadas são computacionalmente complexas, da ordem de $O(N\log N)$, $O(N^2)$ ou mesmo $O(N^3)$, o que implica em uma grande quantidade de processamento a ser realizada. No caso de imagens dinâmicas e/ou de alta resolução, as exigências de processamento são ainda mais difíceis de serem atendidas.

As soluções para este problema podem ser classificadas de maneira ampla dentro de duas abordagens: utilização de software executando em processadores de uso geral de alta performance ou utilização de algum tipo de hardware dedicado.

A implementação de um algoritmo de compressão de imagens utilizando processadores de uso geral exige antes de tudo um processador com taxa de execução de instruções muito elevada. Devido à natureza sequencial dos processadores de uso geral, o paralelismo inerente ao processamento de imagens não pode ser explorado. Arquiteturas superescalares conseguem um paralelismo limitado, às custas de estruturas de controle complexas.

A utilização de hardware para acelerar tarefas de compressão de vídeo pode se dar num número variado de graus. Desde circuitos ASIC específicos cuja arquitetura é adaptada para implementar com eficiência um determinado algoritmo de compressão, até unidades funcionais especializadas em certas tarefas de compressão, integradas a processadores sequenciais ou não. Circuitos ASIC apresentam características interessantes, como a implementação eficiente, tanto em área de silício consumida quanto em energia dissipada, através da adaptação da arquitetura ao paralelismo da tarefa. Por outro lado esta solução peca no que diz respeito à flexibilidade. Não existe, ou existe de modo muito limitado, a possibilidade de se alterar o tipo de processamento executado pelo dispositivo. Isto significa que ele é restrito a um determinado tipo de aplicação, o que implica em volumes de produção menores e custos mais elevados. O mesmo também se aplica a coprocessadores, circuitos dedicados ao processamento de uma determinada etapa da compressão e que operam em conjunto com processadores de uso geral para uma solução completa. Processadores dedicados ao processamento de informação multimídia (processadores de mídia) também sofrem desta restrição, embora em menor grau, pois são programáveis. Mas suas unidades funcionais são projetadas para implementar determinadas técnicas de compressão, e dificilmente trazem algum benefício em outro contexto.

Os problemas de inflexibilidade e especificidade de aplicação inerentes ao hardware são resolvidos se o algoritmo de compressão a ser implementado for um algoritmo maduro, completamente desenvolvido, sem espaço para alterações e amplamente aceito.

Os padrões MPEG-1 e MPEG-2 são exemplos de padrões maduros e bem sucedidos. Se utilizam de técnicas de compressão bastante testadas, como a Transformada Coseno Discreta (DCT). Desenvolvidos para compressão de vídeo para multimídia doméstica e para difusão de vídeo de qualidade, respectivamente, estes padrões atingiram um estágio que pouco se beneficiariam da flexibilidade oferecida por implementações em software. Soluções baseadas em hardware seriam ideais, principalmente devido à demanda de *set-top boxes* de baixo custo para receptores domésticos gerada pela popularização do padrão MPEG-2.

Mas o avanço das técnicas de compressão é uma via de dois sentidos. Ao mesmo tempo que oferece novas possibilidades de codificação mais eficiente também exige novos avanços para oferecer soluções a problemas não considerados anteriormente. Novas técnicas de compressão, conhecidas por técnicas de compressão de segunda geração, são um exemplo desta tendência. Elas permitem obter taxas de compressão mais elevadas, através da exploração das características do sistema visual humano ou de técnicas de codificação baseada em modelos de alto nível

semântico. Estas técnicas motivaram a proposição de um novo padrão, o MPEG-4, inicialmente com foco em comunicação visual em baixas taxas de bits, mas que, muito em função dos avanços nas técnicas de compressão, passou a mirar alvos bem mais ambiciosos. O novo objetivo passou a ser representar a informação multimídia nos seus 3 grandes domínios de aplicação: a comunicação em duas vias, a transmissão em *broadcast* e o armazenamento/reprodução/interação. Devido ao grande número algoritmos compreendidos neste padrão, e por serem novas muitas destas técnicas, apenas a implementação em software é capaz de oferecer a flexibilidade exigida.

Isso leva a se buscar uma maneira de combinar a flexibilidade do software com o desempenho do hardware, permitindo que processadores menos complexos possam executar uma ampla gama de algoritmos de processamento de imagens e que possibilitem alterações nestes algoritmos a medida que técnicas mais eficientes sejam desenvolvidas. Uma solução que tem o potencial de combinar flexibilidade e desempenho são os FPGAs. Estes dispositivos contém um arranjo bidimensional de portas lógicas individuais, ou conjuntos de portas, que podem ser interligadas pelo usuário final sem a necessidade de elaboração de máscaras ou processos de produção complexos. Estas portas podem ser interligadas para implementar uma arquitetura especialmente adaptada ao processamento desejado. Alguns tipos de FPGAs oferecem a possibilidade de reprogramação, tornando possível um "hardware virtual".

FPGAs não são uma panacéia. Para permitir a capacidade de reprogramação são necessários recursos de roteamento e de interconexão que ocupam grande parte da área do dispositivo, o que reduz a complexidade dos circuitos que podem ser implementados e tornam seu desempenho mais lento se comparados com um circuito ASIC. Por outro lado, por serem circuitos de uso geral, os seus volumes de fabricação são muito maiores que soluções específicas para uma determinada aplicação.

Uma tendência que se tem observado é a integração de FPGAs e processadores em um mesmo *chip*. Neste caso o FPGA integrado é de pequena capacidade, e seu objetivo é facilitar a *interface* do processador a outros dispositivos, ou implementar dispositivos de I/O especializados. Um processamento complexo como é a compressão de imagens não é normalmente considerado viável nestas condições.

Este trabalho pretende verificar se é possível implementar algoritmos de compressão de vídeo utilizando FPGAs associados à processadores de recursos limitados. Existem vários algoritmos de processamento de imagens implementados em FPGAs, mas estas implementações

normalmente utilizam mais de um FPGA de grande capacidade, com ganhos de performance significativos sobre processadores de alto desempenho.

1.1. Objetivos

O principal objetivo deste trabalho é o estudo de alternativas para a implementação da transformada wavelet, um algoritmo de compressão de imagens de segunda geração, em FPGAs de capacidade limitada. Isto é interessante para permitir o uso de áreas de lógica reconfigurável integradas a processadores de baixo custo para esta tarefa, habilitando uma série de novas possibilidades para estes processadores. Isto diferencia este trabalho de outros trabalhos nesta área, cuja ênfase é em implementações utilizando várias FPGAs de capacidade elevada. Neste trabalho o foco é a redução da complexidade da implementação.

A transformada wavelet foi escolhida em lugar da DCT devido a sua maior eficiência e simplicidade de implementação. Foram desenvolvidas algumas técnicas de análise para oferecer subsídios para tornar mais eficiente este tipo de implementação.

1.2. Contribuições

A contribuição principal é apresentar uma análise detalhada das alternativas de implementação disponíveis e detalhar o processo de decisão que levou aos resultados finais. Uma metodologia de análise estatística de erros foi desenvolvida e aplicada na implementação. Isto permite uma compreensão de como certos aspectos da arquitetura influenciam a qualidade da compressão e descompressão das imagens.

A implementação utilizou o FPGA Xilinx XC6200, uma família de FPGAs com poucas implementações mencionadas na literatura. Esta família apresenta algumas características interessantes de reconfiguração rápida e dinâmica. O trabalho levou ao desenvolvimento de bibliotecas de macros estruturadas otimizadas para esta arquitetura. Entre estas macros podem-se mencionar macros especializadas para processamento de sinais, que permitem a implementação de outros tipo de wavelets.

O trabalho se conclui com algumas implementações compactas e eficientes da transformada wavelet baseada em wavelets ortonormais e biortogonais.

1.3. Estrutura do trabalho

Inicialmente é feita uma revisão abrangente nas técnicas de codificação de imagens. As vantagens e desvantagens de cada técnica são citadas. Uma relação de padrões de compressão de imagens mais comuns e as técnicas neles utilizadas é apresentada. O capítulo termina enumerando algumas arquiteturas e processadores para implementação destes algoritmos.

FPGAs são introduzidos a seguir, mostrando seus pontos fortes e deficiências. A família de FPGAs Xilinx XC6200 que será utilizada no trabalho é descrita com mais detalhe.

O capítulo seguinte introduz os conceitos por trás da transformada wavelet, com foco em aspectos práticos e na abordagem de processamento de sinais.

O capítulo 5 desenvolve uma técnica de modelamento estatístico de erros adaptada para wavelets. Este modelamento foi testado experimentalmente para wavelets ortononormais Daubechies-4 e Daubechies-6.

O capítulo 6 descreve as alternativas para a implementação em profundidade, mostrando as opções disponíveis e as vantagens e desvantagens de cada opção no contexto do FPGA escolhido.

O capítulo seguinte mostra os resultados da implementação.

Finalmente comentam-se os resultados obtidos, com destaque para as contribuições do trabalho.

2. Compressão de imagens

2.1. Introdução

A idéia básica por trás de qualquer esquema de compressão é remover a redundância presente na informação a ser codificada. No caso de sinais analógicos processados digitalmente, a compressão envolve também uma quantização eficiente desta informação {Cohen e Ryan, 1995}. Quando a informação a ser comprimida consiste em imagens destinadas a observação, a compressão pode envolver uma certa perda na qualidade da imagem. Graças às características do sistema visual humano, pequenas distorções podem ser imperceptíveis ao observador.

Sistemas de compressão de imagens exploram estas deficiências para conseguirem a representação mais eficiente da imagem com qualidade. As técnicas de codificação de imagens mais comuns são apresentadas, de modo a evidenciar as características de cada técnica. Também são descritos alguns critérios de avaliação de qualidade para imagens e técnicas de codificação.

2.2. Redundância

Imagens naturais contém uma quantidade expressiva de informação supérflua. Elementos da imagem (*pixels*) adjacentes no espaço ou no tempo possuem um grau elevado de correlação que, se for explorado, permite uma representação mais compacta. Se as imagens são destinadas únicamente para visualização, maiores reduções ainda podem ser obtidas devido ao fato de que o sistema visual humano não detectar pequenas alterações na imagem. Em outras palavras, a compressão de uma imagem pode ser feita explorando as redundâncias estatística e perceptiva.

2.2.1. Estatística

A redundância estatística em uma imagem está relacionada com a quantidade de informação contida nela. A redundância pode ser medida através da entropia. Seja a luminância dos *pixels* da imagem uma sequência de N variáveis aleatórias individuais $\mathbf{b}$, todas com a mesma função distribuição de probabilidade $P(b)$, $b \in \{\mathbf{b}\}$ onde $\{\mathbf{b}\}$ é o conjunto de todos os valores possíveis para $\mathbf{b}$. A entropia desta imagem pode ser calculada como:

$$H(\mathbf{b}) = -\sum_{b \in \{\mathbf{b}\}} P(b) \log_2 P(b) \tag{2.1}$$

A entropia define a quantidade de bits necessária para representar a informação. Assim, se para cada variável aleatória **b** são necessários H(**b**) bits, para representar toda a imagem são necessários NH(**b**) bits. Considerando que a luminância de cada *pixel* na imagem original seja codificada com N_b bits, a taxa de compressão que pode ser obtida eliminando a redundância estatística é:

$$C = \frac{N_b}{\mathrm{H}(\mathbf{b})}$$

A entropia em (2.1) é denominada de entropia de ordem zero e considera os *pixels* individualmente. Mas imagens naturais possuem correlação elevada tanto entre *pixels* adjacentes no espaço quanto no tempo, permitindo maiores compressões. Supondo que seja considerada uma sequência de S+1 *pixels* ao invés de um *pixel* individual, ou seja, um vetor **B** com S variáveis aleatórias. A entropia de ordem S é:

$$H(\mathbf{B}) = -\sum_{B \in \{\mathbf{B}\}} P(B) \log_2 P(B) \qquad (2.2)$$

Se não existe correlação entre os elementos do vetor **B**, o número bits necessários para representar cada *pixel* deste conjunto é o mesmo necessário para se representar um *pixel* individualmente. Mas existindo correlação este valor diminui. Em testes com imagens naturais cuja luminância foi representada com 6 bits, a entropia de ordem 0, 1 e 2 obtida foi de, respectivamente, 4.4, 1.9 e 1.5 bits/*pixel* {Jain, 1981}.

A eliminação da correlação entre *pixels* pode ser conseguida através de codificação estatística dos mesmos considerando a entropia de ordens mais elevadas. Mas na prática isto é inviável devido às exigências de memória, pois são necessárias tabelas cujo tamanho é da ordem $2^{(S+1)Nb}$. Para se calcular a entropia de ordem 3, ou seja, considerando apenas grupos de 4 *pixels*, S=3 e N_b=8, a memória utilizada seria da ordem de 32 MBytes. Por isso a eliminação da redundância na prática é feita através de técnicas de predição ou utilização de transformadas que reduzem a correlação entre os *pixels*.

Estas mesmas considerações sobre correlação de *pixels* num único quadro de imagem podem ser aplicadas correlação de *pixels* em quadros sucessivos de imagem.

2.2.2. Perceptiva

A redundância perceptiva, ou subjetiva, em uma imagem é a informação irrelevante para o ser humano que, se omitida, não causa diferença perceptível ao observador. Ao contrário da redundância estatística, a remoção da redundância perceptiva implica em perda de informação e não é reversível. Enquanto a redundância estatística pode ser avaliada quantitativamente, a redundância perceptiva é avaliada qualitativamente, através de testes psicométricos aplicados a grupos de observadores {Netravali e Haskell, 1988}. Isto porque o conhecimento sobre o sistema visual humano ainda não é suficiente para desenvolver modelos que avaliem a qualidade subjetiva de uma imagem da mesma forma que o ser humano {Jayant e outros, 1993}.

O diagrama simplificado que sumariza o mecanismo de percepção do sistema visual humano está na Figura 2.1 {Kunt e outros, 1985}. A imagem, após ser filtrada pelas lentes do globo ocular e sensibilizar a retina de forma não-linear, tem seus componentes direcionais percebidos separadamente no córtex cerebral. Esta filtragem ocorre de modo similar a operadores utilizados para deteção de bordas, como o operador Sobel. Cada componente direcional é então dividido em bandas, ou oitavas {Jayant e outros, 1993}, e cada banda é percebida com um grau diferente de sensibilidade.

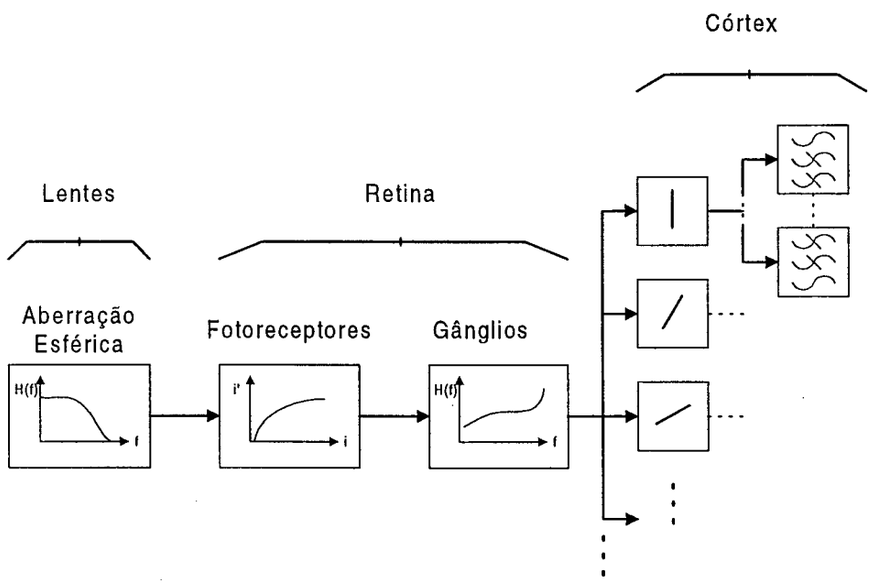


Figura 2.1: Modelo do Sistema Visual Humano

Um sistema de compressão eficiente deveria processar imagens de maneira similar ao do córtex humano, permitindo que os seus componentes sejam quantificados com o mesmo peso

USP SERVIÇO DE BIBLIOTECA E INFORMAÇÃO

atribuído a eles pelo sistema visual humano. Este processo permitiria eliminação da redundância perceptiva, levando a taxas de compressão maiores que aquelas obtidas apenas eliminando a redundância estatística.

2.3. Codificação

Existem várias técnicas pesquisadas de codificação de imagens. A grosso modo elas podem ser divididas em 5 grandes categorias:

- Predição
- Transformação
- Interpolação/Extrapolação
- Codificação Estatística
- Codificação Baseada em Modelos

A maior parte dos padrões de compressão de imagens utilizam-se de mais de uma destas técnicas. A codificação estatística, por exemplo, é utilizada com quase todas as técnicas. Neste mesmo cenário, é muito comum a combinação de técnicas de predição e transformação, que é denominada codificação híbrida.

2.3.1. Predição

Nesta técnica o sinal transmitido é a diferença entre o sinal real e uma previsão feita com base nas suas amostras anteriores. Para sinais com alto índice de correlação, como imagens, sua taxa de variação não é elevada e a faixa dinâmica do erro de previsão é menor que a do sinal, podendo portanto ser codificado com um número menor de bits. Entre as técnicas de codificação por predição estão a Modulação Delta (DM) e a Codificação Diferencial por Modulação de Pulso (DPCM), bastante utilizadas na codificação de voz. Esta última é utilizada no modo de compressão sem perda do padrão JPEG {Pennbecker, 1993}, mas oferece taxas de compressão limitadas. A sua grande vantagem é ser uma técnica de implementação simples.

Preditores são utilizados para eliminação da redundância temporal entre quadros sucessivos de uma sequência de imagens em praticamente todos os padrões de compressão de imagens dinâmicas. Variam desde preditores simples que codificam as diferenças entre quadros, como no H.261, até esquemas que levam em consideração o movimento de translação de regiões da imagem para minimizar a diferença entre os quadros (compensação de movimento).

Um aspecto importante desta técnica é sua suscetibilidade a erros de transmissão, pois erros em um *pixel* pode afetar todos os subsequentes, incluindo *pixels* de outros quadros da imagem. A técnica mais utilizada para minimizar este problema é o envio periódico de um quadro completo da imagem, anulando os erros de previsão.

2.3.2. Transformação

Transforma o conjunto de *pixels* de uma imagem em um conjunto de coeficientes estatisticamente independentes ou com menor índice de correlação {Kunt e outros, 1985}, onde a energia é concentrada em um número menor de coeficientes. Como os coeficientes não são correlacionados, eles podem ser codificados estatisticamente com eficiência mesmo individualmente. Normalmente estas transformadas são lineares. A compressão ocorre na quantização dos coeficientes, que pode ser feita com alguma perda de precisão; e na consideração de coeficientes de pequena magnitude como sendo nulos {Netravali e Haskell, 1988}. Naturalmente isto implica em um erro na codificação, mas, se comparado a codificação baseada na previsão, este erro é distribuído em um conjunto muito mais amplo de *pixels*. Por isso é menos perceptível ao sistema visual humano e preferível em altas taxas de compressão {Netravali e Haskell, 1988}.

A transformação linear de uma sequência unidimensional de *N pixels*, representada pelo vetor **v**, pode ser expressa como a multiplicação entre este vetor uma matriz de decorrelação **A** de dimensão *NxN*, produzindo um vetor **u** de dimensão *N*. Como as dimensões de **v** e **u** são as mesmas, fica claro que neste processo não há nenhum tipo de compressão. A transformada inversa é obtida multiplicando-se a matriz de reconstrução **B** pelo vetor **v**.

$$\begin{aligned} \mathbf{u} &= \mathbf{A}\mathbf{v} \\ \mathbf{v} &= \mathbf{B}\mathbf{u} \end{aligned} \qquad (2.3)$$

Se a transformação for ortonormal, também denominada unitária, **B** é a transposta conjugada de **A**.

$$\mathbf{v} = \mathbf{B}\mathbf{u} = \mathbf{A}'\mathbf{u}$$

Esta transformação é da ordem $O(N^2)$. Algumas transformadas possuem algoritmos rápidos, de ordem $O(N\log N)$ ou menor, tornando possível o processamento de imagens (onde *N* é elevado) em tempo real.

Como imagens são matrizes bidimensionais de *pixels*, os vetores **u** e **v**. passam a ser as matrizes **V** de *pixels* e **U** de coeficientes. Se os elementos da matriz de *LxL* forem agrupados em um vetor de L^2 elementos, a transformada pode ser feita normalmente. Mas ela reduzirá apenas a correlação de uma das dimensões da matriz. Isso pode ser corrigido aplicando-se a transformada separadamente, ou seja, primeiro nas linhas e depois nas colunas, ou vice-versa.

$$\mathbf{U} = \mathbf{A}\mathbf{V}\mathbf{A}^t$$
$$\mathbf{V} = \mathbf{B}\mathbf{U}\mathbf{B}^t$$

Se a transformada for ortonormal:

$$\mathbf{V} = \mathbf{A}'\mathbf{U}\mathbf{A}^*$$

A transformada separada exige $2N^3$ operações de multiplicação, menos que as L^4 necessárias se a transformada for feita em um único vetor de L^2 elementos.

2.3.2.1 Transformada Karhunen-Loève

É a transformada linear mais eficiente na eliminação da correlação dos coeficientes {Kunt e outros, 1985}. É pouco utilizada na prática devido sua complexidade computacional $O(N^2)$ e necessitar utilizar informações precisas da covariância da imagem, que não são conhecidas com precisão, e cuja extração é de complexidade $O(N^3)$. Existe um algoritmo rápido para esta transformada, mas ele é válido apenas para certos modelos estatísticos de imagens {Jain, 1981}.

Seja **R** a matriz de autocovariância de **v**. As linhas da matriz de transformação **A** são formadas por autovetores normalizados de **R**.

A diferença entre as taxas de compressão obtidas a uma determinada distorção quando codificadas utilizando DPCM e a transformada KL foi determinada analiticamente {Jain, 1981}, considerando como entrada uma sequência Gauss-Markov de primeira ordem. Para distorções da ordem de 1% e sequências finitas de *pixels*, a taxa de compressão oferecida pela transformada KL é até 30% superior que aquela oferecida pela codificação DPCM. Para imagens reais, a diferença na compressão é ainda maior.

2.3.2.2 DCT

A transformada cosseno discreta é uma variação da transformada discreta de Fourier (DFT), com a vantagem de minimizar as distorções causadas pelo fenômeno de Gibbs introduzidos pelas descontinuidades na extensão periódica dos dados em **v** {Netravali e Haskell, 1988}. Tem como base apenas funções cosseno e por isso não apresenta números complexos em sua matriz de transformação. Isso é interessante pois como as intensidades dos *pixels* também são números reais, a transformação não utiliza de aritmética complexa {Egger e outros, 1999}.

A matriz **A** de transformação é definida como:

$$A_{l,c} = a_l \cos\left(\frac{2(c+1)\pi l}{2N}\right), \quad \forall l, c \in \{0..N-1\},$$

onde

$$a_i = \begin{cases} \sqrt{1/N}, & i = 0 \\ \sqrt{2/N}, & \forall i \in \{1..N-1\} \end{cases}$$

Existem algoritmos rápidos para a DCT, cuja complexidade é da ordem de $O(N\log_2 N)$ {Ersoy e Nouira, 1992}.

No que diz respeito a redução da correlação das imagens, a DCT (assim como a DFT) se aproxima da KLT se a imagem apresentar certas características estatísticas. Se a intensidade dos *pixels* tiver uma covariância proporcional a $\rho^{|d|}$, onde d é a distância entre os *pixels*, e se ρ se aproxima de 1, a DCT é uma substituta para a KLT. Em imagens naturais ρ possui um valor típico de 0.95 {Jain, 1981} e nesta situação a DCT apresenta resultados similares à KLT {Netravali e Haskell, 1988}.

A DCT é a transformada utilizada na maioria dos padrões de compressão de imagem existentes, como o JPEG, H.261, H.263, MPEG-1, MPEG-2 e MPEG-4. Para tornar possível o processamento em tempo real, em todos estes padrões a DCT não é aplicada na imagem como um todo, mas em blocos de *LxL pixels*, onde $8 \leq L \leq 16$. Esta segmentação não apresenta problemas pois a correlação dos *pixels* diminui a medida que a distância entre eles aumenta. Também é interessante para que erros de quantização de um coeficientes não causem distorções em toda a imagem.

A DCT segmentada é eficiente para compressões de até 30-40 vezes. Para compressões mais elevadas ela causa distorções na imagem muito visíveis chamadas de *blocking artifacts* {Egger e outros, 1999}, quando contornos dos blocos ficam *LxL* são bastante perceptíveis.

2.3.2.3 Transformada Wavelet

A transformada wavelet é uma transformada de desenvolvimento recente e com um comportamento similar à DCT no que diz respeito à eliminação da correlação espacial. Ela decompõe um sinal em níveis de resolução cada vez menores, gerando dois conjuntos de componentes para cada nível de resolução; um deles com uma representação aproximada do sinal naquela resolução, e outro com os detalhes perdidos nesta aproximação. Do ponto de vista de processamento de sinais, estes componentes são equivalentes ao resultado da filtragem do sinal respectivamente em um filtro passa baixas e outro passa altas.

Uma das grandes vantagens da WT quando comparada com a DCT é a sua complexidade de ordem O(*N*). Esta característica permite que a WT seja aplicada a imagens sem a necessidade de segmentá-las em blocos, evitando assim as perturbadoras distorções de bloco (*blocking artifacts*). Em compressões de 50 vezes ou mais as distorções características da WT são chamadas de *ringing effects* {Egger e outros, 1999} e são causadas pelo fenômeno de Gibbs.

Outra importante vantagem é a escalabilidade de resolução temporal, resolução espacial e de codificação. Uma imagem codificada em resoluções sucessivas pode ser decodificada de modo a ter sua resolução e tamanho incrementados a cada iteração. A decodificação não precisa ser feita até o último nível de resolução, permitindo que o compromisso de qualidade e tamanho da imagem possa ser estabelecido na decodificação, em função da capacidade de processamento disponível.

Além disso, como a decomposição da imagem em várias bandas de frequências emula aproximadamente a maneira que o sistema visual humano percebe as imagens, é possível levar em consideração na quantização de cada banda a importância que o cérebro humano dá a ela. Isso traz uma maior eficiência na eliminação da redundância subjetiva.

Por permitir explorar as peculiaridades do sistema visual humano na redução da redundância, a WT pode ser considerada como uma técnica de codificação de segunda geração {Kunt e outros, 1985}.

2.3.3. Interpolação/Extrapolação

Este tipo de codificação procura extrair de uma imagem um subconjunto de *pixels* que através de interpolações ou extrapolações, pode ser utilizado para a reconstrução da imagem completa. É bastante utilizado na compressão de quadros sucessivos de imagens dinâmicas, como na criação de *frames* do tipo B no padrão MPEG-2.

Outra técnica que pode ser classificada dentro desta categoria é a compressão fractal. Ela se baseia na propriedade de que regiões de uma imagem podem ser expressas a partir de um conjunto de *pixels* de dentro dessa região. Taxas de compressão de 70 a 80 vezes são possíveis {Egger e outros, 1999}.

2.3.4. Codificação Estatística

A codificação estatística procura definir uma codificação binária eficiente para os símbolos que representam a imagem (*pixels*, coeficientes, erros de predição, etc). Para que esta codificação seja eficiente e prática, é necessário que a sequência de símbolos a ser codificada seja não-correlacionada. Por exemplo, a taxa de compressão da imagem clássica de teste "Lena" utilizando codificação estatística mas sem eliminar a correlação entre os *pixels* é de apenas 20% {Egger e outros, 1999}.

Por isso esta técnica de codificação de imagens normalmente é associada com técnicas de remoção de correlação, como a codificação preditiva e de transformação.

2.3.4.1 Huffman

Esta codificação estatística utiliza palavras binárias de tamanho inteiro variável. Se baseia na probabilidade de ocorrência de cada símbolo, alocando menos bits para símbolos mais frequentes e mais bits para os menos frequentes. O tamanho mínimo da palavra binária não pode ser menor que 1 bit. O que é inconveniente quando se codifica símbolos cuja distribuição estatística é extremamente concentrada, como os coeficientes nulos gerados pela quantização dos coeficientes obtidos após as transformadas DCT. Daí a associação com a codificação *run-length*, que substitui toda uma sequência de símbolos iguais por um único valor indicando o número de repetições deste símbolo {Netravali e Haskell, 1988}.

A utilização dos códigos de Huffman torna a codificação da informação próxima do

limite inferior teórico, segundo indicado pela sua entropia {Egger e outros, 1999}.

2.3.4.2 Codificação Aritmética

Ao contrário do método de Huffman, esta codificação pode representar símbolos utilizando frações de bit. Por isso pode ser utilizada na codificação a distribuição estatística precisa dos símbolos, mesmo quando ela é extremamente concentrada. Por outro lado, é mais sensível a erros na transmissão.

2.3.5. Codificação Baseada em Objetos e Modelos

Técnicas de codificação de imagens baseadas na redução da redundância estatística são consideradas técnicas de primeira geração. Estas técnicas se baseiam modelos estatísticos de imagens, que são aproximações matematicamente convenientes mas nem sempre válidas no mundo real. Também não exploram em profundidade as peculiaridades do sistema visual humano na redução da redundância perceptiva.

Para maior eficiência na codificação novas técnicas foram propostas que analisem a estrutura da imagem, decompondo-a em elementos relevantes para o sistema visual humano, como bordas, texturas, primitivas geométricas, etc, cujo nível semântico é mais elevado que um simples arranjo de *pixels*. Estas técnicas são denominadas técnicas de segunda geração, e por serem mais complexas que as de primeira geração, se tornam vantajosas apenas para taxas de compressão acima de 50 vezes {Ebrahimi e Kunt, 1998}. Entre estas técnicas estão as baseadas em objetos e baseadas em modelos.

A codificação baseada em objetos decompõe a imagem em objetos de formato arbitrário e os estes objetos nas suas primitivas (contornos, texturas, translações, etc). Esta codificação é utilizada no padrão MPEG-4, que pode representar uma imagem como um conjunto de imagens de formato arbitrário.

A codificação baseada em modelos visa extrair da imagem (análise) parâmetros que controlem um modelo disponível no receptor (síntese). Esta técnica permite uma compressão muito elevada, pois apenas os parâmetros são transmitidos. Mas devido a complexidade, é de recente aplicação na compressão de imagens. Sua utilização está prevista no padrão MPEG-4, na forma de modelos para síntese de faces e corpo humano, permitindo aplicações de vídeotelefonia a taxas de bits muito reduzidas.

2.4. Quantização

A quantização é a representação de uma grandeza de determinada resolução em uma resolução inferior. A resolução de sinais analógicos não sujeitos a ruído, por exemplo, é infinita. A quantização é uma operação não-reversível, e implica na introdução de um erro na grandeza representada, chamado de erro de quantização {Porat, 1997}.

Em sistemas de codificação de imagens os erros de quantização ocorrem na entrada, na codificação e na saída. O erro de quantização da entrada consiste na conversão da intensidade dos *pixels*, uma grandeza analógica, para uma grandeza digital de resolução finita. O erro de quantização na saída consiste em representar a informação já quantizada utilizando um número de bits menor que o necessário. E o erro de quantização na codificação é o erro cumulativo provocado pela representação inexata dos parâmetros envolvidos no processo de codificação.

Como em todos estes processos é introduzido o erro de quantização, é importante poder se estimar o efeito que estes erros causam na imagem reconstruída.

A quantização na etapa de codificação depende do tipo de codificação adotada e até mesmo do tipo de algoritmo que implementa determinado esquema de codificação. A análise do erro de quantização específico para a codificação utilizada será proposta em capítulo a parte. Para outras técnicas de codificação mais detalhes podem ser obtidos em {Kim e Sung, 1998} {Wong, 1991} {Lim e Liu, 1988} {Porat, 1997} {Oppenheim e Schafer, 1975} {Jain, 1981} {Netravali e Haskell, 1988}.

A quantização da saída pode ser utilizada para se obter compressão adicional dos dados. Quando este processo é feito com critério e, de preferência, explorando características do sistema visual humano, o erro de quantização acrescentado pode ser imperceptível ao observador {Jayant e outros, 1993}. Uma situação frequente é quando se deseja transmitir a informação relativa aos bits comprimidos a uma taxa fixa. Para isso o número de bits disponível para a representação dos coeficientes ou do erro de predição deve ser alocado de forma a minimizar a distorção. A solução deste problema tem abordagens diferentes para codificação preditiva {Jain, 1981} e por transformação {Egger e outros, 1999} {Strang e Nguyen, 1996} {Netravali e Haskell, 1988}.

2.5. Avaliação

A avaliação de um algoritmo de compressão de uma imagem pode ser feita segundo os

seguintes critérios {Jayant e outros, 1993}:

- Qualidade
- Taxa de bits
- Complexidade
- Latência
- Simetria
- Escalabilidade

O sistema visual humano é muito complexo e por isso ainda não foi possível estabelecer uma medida única de qualidade de imagem que indique precisamente o quanto de distorção é efetivamente percebida por um observador. Por isso existem vários critérios de medida de qualidade, divididos entre objetivos ou subjetivos.

Critérios objetivos são utilizados tanto para avaliar o resultado de uma determinada compressão, como para auxiliar definição de parâmetros em um algoritmo de compressão de modo a minimizar ou otimizar o erro na saída. Embora não correspondam diretamente ao modo que o sistema visual humano avalia a fidelidade a imagem reconstruída, são critérios úteis. A medida objetiva mais comum do erro entre a imagem original e a comprimida é o Erro Quadrático Médio (RMSE):

$$e_{RMS} = \sqrt{\frac{1}{AL}\sum_{i=1}^{L}\sum_{j=1}^{A}\left(img_{i,j} - \tilde{img}_{i,j}\right)^2}$$

Outras medidas objetivas de qualidade de imagem se baseiam na relação sinal-ruído (SNR). Duas medidas de relação sinal-ruído são utilizadas na área de processamento de imagens.

$$SNR = 10\log\left(\frac{\sigma_{img}^2}{e_{RMS}^2}\right)\text{dB}$$

$$PSNR = 10\log\left(\frac{I_{\max}^2}{e_{RMS}^2}\right)\text{dB}$$

A primeira equação, onde σ_{img} é o desvio padrão da distribuição de *pixels* na imagem e relaciona a potência do sinal com a potência do ruído. A segunda equação, onde $I_{\max}$ é o valor máximo de pico-a-pico que um *pixel* pode assumir, é a simplificação de uma medida de qualidade desenvolvida para televisão preto e branco {Netravali e Haskell, 1988}. Ela possui uma relação

aproximadamente linear com os resultados de testes qualitativos de fidelidade de reprodução feitos para televisão, e por isso foi adotada como medida de qualidade na área de imagens.

Testes qualitativos se baseiam na avaliação por um grupo de observadores de imagens e padrões normalizados. A avaliação feita com imagens é chamada de avaliação explícita, e com padrões normalizados é a avaliação implícita. Na avaliação explícita os observadores avaliam subjetivamente a qualidade da imagem, ou sua distorção, sob diversas condições de iluminação, distância do observador, tamanho da imagem, e atribuem um valor dentro de uma escala de 5 níveis. O julgamento de todos os observadores é ponderado para se obter um valor médio de qualidade {Jayant e outros, 1993}.

A taxa de bits obtida é uma das avaliações mais importantes de um algoritmo de compressão. Esta taxa pode ser medida em bits/*pixel* ou bits/segundo.

A complexidade de uma técnica de compressão pode ser avaliada segundo várias métricas, como número de operações aritméticas necessárias, quantidade de memória exigida, número de portas utilizadas, potência dissipada, etc.

Um fator importante principalmente em aplicações de compressão de vídeo dedicado a comunicação em duas direções é a latência. Latência é o atraso entre o início da codificação e o término da decodificação de um quadro. A latência é menos relevante para aplicações de que envolvem comunicação em uma direção, como *broadcast*.

Outro fator a ser considerado na avaliação de uma técnica de codificação de imagem é a simetria do algoritmo, ou seja, a diferença de complexidade do algoritmo de compressão e descompressão. Técnicas de compressão que utilizam a codificação preditiva para eliminar redundância entre quadros são assimétricas, pois a compressão é muito mais complexa que a descompressão. Padrões como H.261, H.263, e MPEG exigem que a maior parte das tarefas do estágio de decodificação sejam executadas também na codificação, sendo portanto bastante assimétricos.

Do mesmo modo que algoritmos, aplicações também são classificadas segundo o critério de simetria {Furht, 1995}. São assimétricas aplicações que se utilizem poucas vezes da compressão e muitas da descompressão, como televisão digital ou reprodução de informação multimídia. Aplicações que fazem o mesmo uso da compressão e descompressão, como videoconferência e videotelefonia, são classificadas como simétricas. A simetria das técnicas de codificação é importante quando a codificação acontece em tempo real. Se a aplicação for

assimétrica, a assimetria da codificação não apresenta inconveniência.

A escalabilidade de uma técnica de codificação é a capacidade do sinal codificado se adaptar a uma variedade de condições. A escalabilidade é vital quando imagens estáticas ou dinâmicas codificadas devem ser transmitidas em uma variedade ampla de canais de transmissão, com características diversas de taxa de erros, velocidade, etc, e com destino a uma ampla variedade de receptores, como *set-top boxes*, computadores, dispositivos portáteis, etc, cada um com uma capacidade de processamento diferente. Neste contexto os fatores de escalabilidade mais importantes estão na resolução temporal, na resolução espacial, na dimensão e na decodificação.

A escalabilidade na resolução temporal no contexto de codificação de imagens é a possibilidade da mesma informação ser decodificada em taxas de atualização menores que aquela da imagem original. Escalabilidade na resolução espacial é a capacidade de se reconstruir imagens de diferentes níveis de qualidade a partir da mesma informação codificada. O mesmo é válido para a escalabilidade na dimensão da imagem. A escalabilidade na decodificação é a capacidade da informação ser parcialmente decodificada e apresentar informação inteligível.

Em um ambiente de comunicação multimídia ubíqua é desejável que informação multimídia codificada possa ser decodificada por uma grande variedade de receptores, inclusive aqueles de capacidade computacional limitada, mesmo que penalizando a resolução temporal ou algum outro parâmetro de qualidade. Esta característica é satisfeita se técnicas de codificação apresentem escalabilidade na decodificação. Ou seja, um receptor decodifica a quantidade de informação que for possível para ele. A escalabilidade na decodificação permite que o receptor faça a decodificação da imagem dentro do melhor compromisso entre resolução temporal, espacial e dimensão, independente do que foi previsto na codificação.

No caso de canais de transmissão lentos, é interessante que a decodificação da imagem seja feita de forma incremental, tanto no seu tamanho quanto na sua resolução, de forma a permitir que o seu conteúdo seja avaliado parcialmente pelo observador sem a necessidade da recepção completa da imagem. Para isso é necessário que a codificação utilizada apresente escalabilidade no tamanho e nas resoluções temporais e espaciais da imagem.

2.6. Padrões de compressão

Na Tabela 2.1 são sumarizadas as técnicas de codificação utilizadas nos padrões de

compressão de imagens estáticas mais comuns. A Tabela 2.2 se refere a imagens dinâmicas. Todos estes padrões são baseados em transformações, predição e codificação estatística. A transformação mais utilizada é a DCT, mas os padrões mais recentes adotam a transformada wavelet (WT). Para imagens dinâmicas utiliza-se a codificação híbrida, ou seja, transformações para eliminar a redundância dentro de um quadro e codificação preditiva para eliminá-la entre quadros.

Tabela 2.1: Padrões de compressão de imagens estáticas

Padrão	Entidade	Taxa de compressão	Técnicas de compressão	Aplicações
JPEG	ISO	2-30	DCT Huffman	imagens de resolução intermediária fotografia digital imagens
JPEG2000	ISO	2-50	Wavelets (em definição)	Inclusive imagens de alta resolução imagens médicas imagens de satélite fax colorido

Tabela 2.2: Padrões de compressão de imagens dinâmicas

Padrão	Entidade	Taxa de bits	Técnicas de compressão	Aplicações
H.261	ITU	múltiplos de 64K	DCT Quantização Compensação de movimento preditiva Huffman	Vídeotelefonia sobre ISDN Vídeoconferência sobre ISDN
H.263	ITU	8 K a 1.5 M	DCT Quantização Compensação de movimento refinada e bidirecional e Huffman Codificação aritmética	Vídeotelefonia sobre linhas telefônicas Vídeotelefonia móvel
MPEG-1	ISO	1.5 M	DCT Quantização perceptual Compensação de movimento refinada e bidirecional Huffman Codificação aritmética	Vídeo para armazenamento em CD-ROM Multimídia doméstica
MPEG-2	ISO	1.5 M a 35 M	DCT Quantização perceptual e adaptatival Compensação de movimento refinada e bidirecional Níveis e perfis de operação Huffman Codificação aritmética	TV digital (broadcast, satelite e cabo) TV digital de alta resolução Vídeo profissional
MPEG-4	ISO	8 K a 35 M	DCT	Vídeo interativo

			Wavelet Quantização perceptual e adaptativa Compensação de movimento refinada e bidirecional Escalabilidade espacial, temporal, de qualidade e de complexidade Objetos de forma arbitrária Número arbitrário de streams Codificação baseada em modelos Huffman Codificação aritmética	Vídeo profissional Convergência de mídias Multimídia móvel

2.7. Implementação das Técnicas de Compressão

O processamento da informação multimídia envolve de um modo ou de outro codificação e/ou decodificação da informação para armazenamento, transmissão ou exibição. A codificação/decodificação é necessária para permitir o armazenamento eficiente da informação multimídia em dispositivos de armazenamento restritos e sua transmissão em canais limitados. Este processo também é descrito como compressão/descompressão.

A implementação dos algoritmos de compressão de informação multimídia em computadores de uso geral apresenta um espectro amplo de soluções, que se localizam em algum ponto intermediário entre dois paradigmas: a implementação baseada em software e a implementação baseada em hardware.

2.8. Implementação em Software

A implementação de algoritmos de codificação/decodificação de imagens em tempo real é uma tarefa computacionalmente muito complexa. O padrão MPEG-2, cujo diagrama está na Figura 2.2, exige 4.10^8 operações para a descompressão de imagens de qualidade equivalente ao padrão NTSC, e para a compressão, 3.10^{10} operações por segundo {Kuroda e Nishitani, 1998}. Sendo que apenas operações aritméticas são consideradas, sem levar em conta as operações de *load* e *store*.

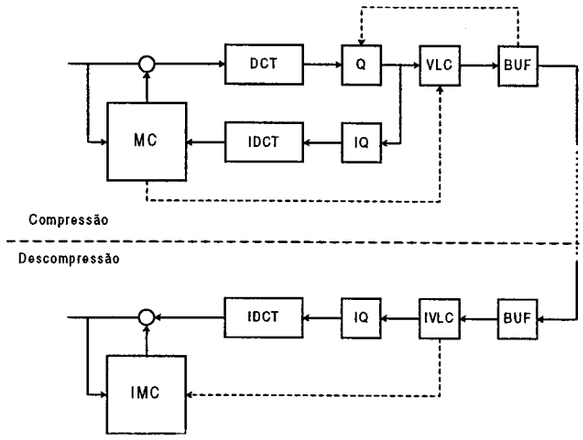


Figura 2.2: Diagrama simplificado da codificação/decodificação segundo padrão MPEG

A maior parte destas operações está distribuída nas tarefas de eliminação de redundância espacial (DCT), eliminação de redundância temporal (MC), e quantização/codificação estatística de tamanho variável (Q, VLC) e nas suas operações inversas (IDCT, IMC, IQ, IVLC). As operações necessárias para DCT e MC e suas inversas são na sua maior parte operações de multiplicação, soma e subtração em palavras de 8 ou 16 bits. As tarefas VLC e IVLC utilizam principalmente operações em nível de bits individuais. A distribuição aproximada de operações para cada bloco na descompressão MPEG-1 e MPEG-2 {Kuroda e Nishitani, 1998} é apresentada na Tabela 2.3.

Tabela 2.3: Operações necessárias para descompressão vídeo de qualidade NTSC

	MPEG-1	MPEG-2
IDCT	42.4 Mops	173.3 Mops
IMC	36.5 Mops	149.3 Mops
IQ+IVLC	13.6 Mops	36.2 Mops

A arquitetura de processadores de uso geral para *workstations* e computadores pessoais tem sido estendida para executar operações de processamento de informação multimídia com maior eficiência. Estas extensões, como a MMX da Intel {Peleg e Weiser, 1996} e a ISP da HP {Lee, 1995}, que exploram o paralelismo SIMD obtido com a divisão das ALUs em unidades menores, são bastante apropriadas para processamento de *pixels*. E também são consistentes com a filosofia por trás da arquitetura RISC, que é prover primitivas e não soluções. A Tabela 2.4 mostra a redução obtida na descompressão MPEG-1 e MPEG-2 se utilizadas extensões multimídia. No levantamento destes dados não foi considerada nenhuma extensão multimídia específica, mas reduções similares na DCT utilizando MMX {Peleg e outros, 1997}, permitem

considerar estes resultados como típicos.

Tabela 2.4: Operações necessárias para descompressão vídeo de qualidade NTSC, utilizando extensões multimídia

	MPEG-1	MPEG-2
IDCT	11.9 Mops	48.6 Mops
IMC	12.1 Mops	49.8 Mops
IQ+IVLC	9.9 Mops	26.4 Mops

Com estes resultados, a decodificação MPEG-2 passa a utilizar aproximadamente 1.10^8 operações, o que está dentro das possibilidades de processadores de uso geral superescalares {Johnson, 1991}. O elevado número de MIPS necessário para implementar estas operações em um único processador implica num processador com alta frequência de *clock*, exploração intensa de *pipeline* e latências elevadas na execução das instruções.

A contrapartida disto é o alto consumo e a necessidade de um mecanismo complexo de controle de execução de instruções, de modo a executar instruções fora de ordem e minimizar problemas de dependência entre dados {Patterson e Hennessy, 1995}. Esta combinação se traduz em pouca eficiência na utilização da energia e da área de silício.

Mesmo algoritmos menos exigentes do ponto de vista computacional ainda são bastante complexos. {Sakamoto e Hase, 1997} descreve uma implementação do padrão JPEG utilizando um microcontrolador RISC de 32 bits de uso geral, para aplicação em câmeras fotográficas digitais. Os números de operações necessárias nesta implementação estão na Tabela 2.5. Esta avaliação foi feita em duas implementações, uma delas altamente otimizada para a arquitetura do processador. A imagem processada possui tamanho de 640x480 *pixels*, próxima ao tamanho da imagem processada pelo MPEG-2, cujos dados estão na tabela anterior. Diferentemente daquela avaliação, a Tabela 2.5 é mais detalhada e leva em consideração inclusive operações de *load* e *store*. O desempenho relatado da implementação é de 1 imagem comprimida em 0.5 s.

Tabela 2.5: Operações necessárias para compressão de imagem VGA

	JPEG	JPEG otimiz
DCT	6.0 Mops	4.1 Mops
Q+VLC	5.6 Mops	2.9 Mops

Estes resultados mostram que a implementação de algoritmos de compressão de imagens ainda está distante dos processadores de baixo custo, embora sejam viáveis para processadores de uso geral sofisticados.

2.9. Implementação em Hardware Dedicado

Uma solução para tornar possível a utilização de multimídia em dispositivos mais simples que computadores pessoais ou *workstations*, como *set-top-boxes*, e sistemas *embbeded*, seria implementar os algoritmos de codificação/decodificação diretamente em hardware. Implementações de algoritmos diretas em hardware apresentam vantagens evidentes de velocidade e consumo {Pirsch e outros, 1995}.

A maior velocidade é conseguida pela especialização da arquitetura do processador para o algoritmo pretendido. Nesta arquitetura o paralelismo oferecido pelo algoritmo é explorado através da utilização de várias unidades funcionais e múltiplos barramentos, etc. O consumo é reduzido pois estas entidades, operando em paralelo, não necessitam de taxas de *clock* tão elevadas quanto processadores de uso geral, operando sequencialmente.

Mas existem desvantagens substanciais. A perda da flexibilidade imposta pela implementação em hardware não é importante quando os algoritmos a serem implementados estão maduros. Mas o advento novos padrões, como o MPEG-4 e a sua filosofia "caixa-de-ferramentas" alterou o cenário do processamento de informação multimídia.

Embora inicialmente direcionado para uma tarefa muito específica – a videofonia em taxas de bits muito reduzidas, o MPEG-4 converteu-se em padrão para representação de informação multimídia nas suas 3 grandes classes de aplicações: comunicação em duas vias, transmissão em *broadcast* e armazenamento/reprodução/interação {Sikora, 1997}. Devido a essa proposta ambiciosa, o padrão MPEG-4 utiliza-se de um conjunto muito mais amplo de técnicas de codificação, incluindo técnicas de segunda geração, como a transformada wavelet e codificação baseada em modelos e objetos {Koenen e outros, 1997}. Além de também utilizar as técnicas de codificação maduras como a codificação híbrida baseada em DCT. Por serem relativamente recentes, as técnicas de segunda geração ainda são motivo de pesquisa e refinamento, desencorajando a sua implementação em hardware, e a estrutura flexível do padrão é melhor suportada por implementações em software {Koenen e outros, 1997} {He e outros, 1998}.

A eficiência da utilização do hardware é outra questão importante. Se todas as funcionalidades de um padrão como o MPEG-4 fossem implementadas em hardware a sua utilização seria muito reduzida, pois nem todas estas funcionalidades são utilizadas simultaneamente. Deve haver um compromisso entre as tarefas implementadas em hardware e

aquelas implementadas em software, mas esse compromisso depende de uma série de fatores (processador utilizado, consumo esperado, nível de qualidade desejado na decodificação, etc) e deveria ser deixado para o projetista da aplicação e não definido com antecedência pelo projetista do hardware de compressão/descompressão.

Finalmente, soluções baseadas em hardware apenas são viáveis se produzidas em grande volume, como é o caso de padrões maduros como MPEG-2 {Kondo e outros, 1996}.

2.10. Soluções de Compromisso

A compressão de informação multimídia é uma tarefa com alto grau de paralelismo – grupos de *pixels* podem ser comprimidos independentemente, vários algoritmos de compressão são executados simultaneamente, imagens e sons são processados em paralelo e em sincronismo {Lee, 1995}. Por isso uma solução para se atingir os requisitos de performance requeridos pela compressão/descompressão de imagens em tempo real é a exploração do paralelismo destas tarefas através de um conjunto de processadores relativamente simples operando em paralelo.

Um exemplo desta abordagem é o processador TMS32C080 da Texas Instruments {Guttag e outros, 1992}, que integra em um único *chip* 1 processador RISC de uso geral, 4 DSPs, memórias cache de dados e instruções distribuídas entre os 5 processadores, unidades de chaveamento (*crossbar*) para interligar o conjunto e unidade de transferência independente para sustentar a transferência de dados para os processadores. O processador RISC possui uma arquitetura de 64 bits, unidade de ponto flutuante compatível com IEEE-754, suporte para chaveamento de tarefas, e sua função é coordenar a execução dos outros processadores, além de poder implementar algoritmos que exijam aritmética de ponto flutuante. Os DSPs possuem arquitetura de 32 bits, e instruções de 64 bits (VLIW). Estas instruções permitem a operação paralela de várias unidades funcionais do DSP. Sua ALU é inteira de 32 bits, e a aplicação destes DSPs é o processamento paralelo de imagem. Esta abordagem é interessante, pois o paralelismo MIMD do conjunto de processadores permite a execução de algoritmos complexos sem necessitar de taxas de *clock* elevadíssimas {Golston, 1996} {Kuroda e Nishitani, 1998}. Também simplifica o projeto da unidade de controle de cada processador, pois não são necessários *pipelines* longos e nem um circuito de controle complexo. E cada um dos DSPs pode ser individualmente desativado para maior economia de energia. Mas não é uma solução econômica, pois o *chip* completo utiliza aproximadamente 4 milhões de transistores {Pirsch e outros, 1995}.

Outra proposta para processamento de multimídia sem exigir processadores de complexidade elevada ou altas taxas de *clock* são os processadores com arquitetura VLIW, que se baseiam em instruções com palavras longas controlando a execução simultânea de várias unidades funcionais do processador {Patterson e Hennessy, 1995}. Este paralelismo do tipo SIMD oferece um desempenho elevado sem a complexidade dos processadores superescalares {Kuroda e Nishitani, 1998}. Quando aplicada especificamente a processamento de informação multimídia esta arquitetura é frequentemente enriquecida com unidades funcionais dedicadas ao processamento de mídia, como por exemplo, uma unidade para executar as operações com bits necessárias na codificação estatística {Pirsch e outros, 1995}. Processadores baseados nesta arquitetura são chamados de processadores de mídia. O processador i750 da Intel {Harney e outros, 1991} é uma utilização pioneira desta abordagem para o processamento multimídia. Um exemplo mais recente é o processador Mpact da Toshiba, com capacidade de executar de 2 a 5 instruções simultâneas {Kuroda e Nishitani, 1998}. Mas para efetivamente extrair essa capacidade do processador é necessária a programação em baixo nível, utilizando a linguagem montadora, limitando a utilização de linguagens de alto nível.

2.11. Considerações Finais

Todos os padrões de compressão adotam algum tipo de transformação. Os padrões consolidados se utilizam da DCT, mas os padrões emergentes como o JPEG2000 e o MPEG-4 utilizam-se da transformada wavelet (WT). A razão é que esta transformação decompõe a imagem de modo similar ao sistema visual humano e permite a eliminação redundância subjetiva de forma mais eficiente.

3. Lógicas Reconfiguráveis

Uma maneira de se acelerar o processamento de informação multimídia, combinando a velocidade do processamento em hardware com a flexibilidade do processamento em software, é a utilização de circuitos digitais configuráveis.

Circuitos digitais configuráveis pelo usuário se dividem a grosso modo em dispositivos lógicos programáveis (PLD) e *gate arrays* programáveis em campo (FPGA). Os primeiros são utilizados para interligar circuitos integrados em um sistema digital (*glue logic*). Possuem capacidade limitada e estrutura de roteamento inflexível, não permitindo a implementação de algoritmos complexos. FPGAs possuem capacidade suficiente para efetivamente implementar estes algoritmos, além de utilizarem uma estrutura de roteamento flexível que permite o aproveitamento de toda a capacidade do dispositivo. FPGAs reprogramáveis oferecem a possibilidade de algoritmos diferentes serem implementados em um mesmo dispositivo.

3.1. Histórico

A primeira patente de um dispositivo digital programável similar a um FPGA foi requisitada em 1966 (pat. #3.473.160, *Electronically Controlled Microelectronic Cellular Logic Array*) {Tredennick, 1998}, mas este conceito permaneceu inviável por 2 décadas devido a pequena capacidade dos circuitos integrados de então. Os primeiros FPGAs foram introduzidos em 1984 pela Xilinx. Estes dispositivos foram desenvolvidos com base nos MPGAs, circuitos integrados *semicustom* que contém um arranjo de células, cada célula sendo uma porta lógica completamente funcional {Lala, 1990}. Interligando as saídas destas portas lógicas através de uma camada metalizada se obtém o circuito desejado. Apesar de propiciarem um ciclo desenvolvimento mais rápido que integrados *full custom*, ainda exigem a confecção da máscara para a camada metalizada de interligação. FPGAs são MPGAs cujas interligações são pré-fabricadas, e que podem ser configuradas pelo usuário final.

As vantagens das FPGAs sobre circuitos integrados *semicustom* e *full custom* são

- não exige a fabricação de máscaras
- pequeno tempo de desenvolvimento
- facilidade no teste do circuito final
- simplicidade na correção de erros de projeto

Por outro lado, as desvantagens são

- eficiência reduzida na utilização de área de silício
- custo unitário mais elevado
- desempenho inferior
- consumo mais elevado

Em vista dos prós e contras, as FPGAs são alternativas aos circuitos integrados específicos em projetos cujo volume de fabricação é reduzido e performance não é crítica {Hopkins, 1998}.

3.2. Arquiteturas

Existem várias arquiteturas de FPGAs comercialmente disponíveis por aproximadamente 15 fabricantes {Chow e outros, 1999}. Três aspectos principais definem a arquitetura de um FPGA:

- tipo de tecnologia de programação
- arquitetura das células
- estrutura de roteamento.

A ilustração de um FPGA típico está na Figura 3.1.

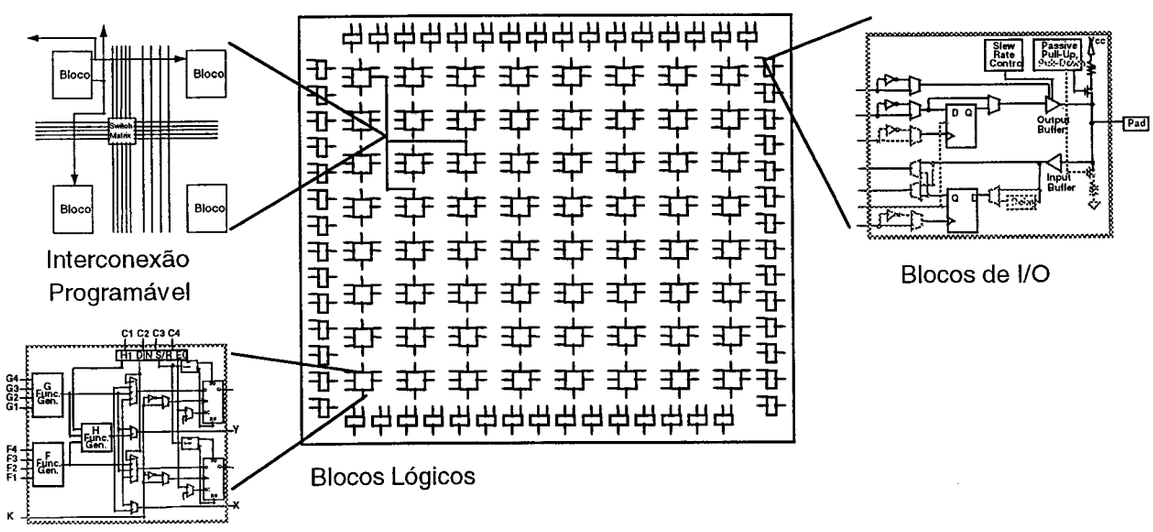


Figura 3.1: Estrutura de um FPGA.

3.2.1. Tipos de Programação

Um FPGA é configurado através de comutadores programáveis eletricamente. Existem várias maneiras de se implementar estes comutadores, cada uma delas com características próprias de tamanho, resistência, capacitância e volatilidade que determinam aspectos importantes de um FPGA, como o desempenho e a complexidade do dispositivo.

Qualquer que seja a tecnologia de implementação destes comutadores programáveis, eles sempre terão área, resistência e capacitância maiores que as de um simples contato metálico {Turley, 1997}. Por essa razão, um FPGA terá sempre o desempenho e a eficiência de utilização de área inferiores se comparado a um MPGA de mesma tecnologia de fabricação {Hauser e Wawrzynek, 1997}.

As tecnologias de programação encontradas com maior frequência em FPGAs são:

- antifuse
- EPROM
- SRAM

Antifuses são dispositivos que quando não programados apresentam uma impedância elevada, e que quando programados por meio de pulsos de tensão elevada (da ordem de 20 V {Hauck, 1998}) passam a conduzir. O nome *antifuse* vem do fato de possuírem um comportamento contrário aos fusíveis. A programação é irreversível, por isso estes dispositivos não são reconfiguráveis. Por outro lado, a área de silício ocupada por cada elemento de comutação é reduzida, sua capacitância é mínima e a resistência de condução é pequena {Brown, 1996}, o que contribui para a velocidade de operação do dispositivo.

EPROMs utilizam como elemento de comutação transistores com *gate* flutuante. Sua programação é feita utilizando tensões de programação elevadas, como no *antifuse*, mas ao contrário do *antifuse*, podem ser reprogramados. O apagamento da programação pode ser feito por meio luz ultra-violeta (UVEPROM) ou elétricamente (EEPROM).

O processo de fabricação de elementos de chaveamento baseados em EPROM é o mais complexo de todos os elementos analisados. A área ocupada por eles é maior que aquela dos *antifuses*, e especialmente no caso da EEPROM {Brown, 1996}. Por outro lado a tecnologia EEPROM permite a programação do *chip* com o mesmo instalado na placa. Este recurso, denominado ISP (*In System Programmability*), facilita muito correções e atualizações na

programação do dispositivo.

A tecnologia de programação SRAM tem como elementos de comutação transistores operando como chaves, ou multiplexadores. São controlados por células de RAM estática distribuídas pelo FPGA, conforme mostra a Figura 3.2.

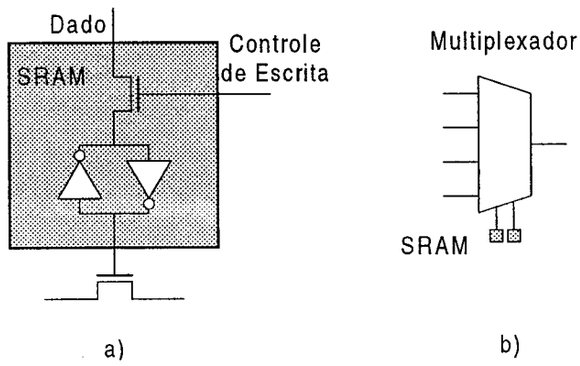


Figura 3.2: Elementos de comutação a) transistor como chave e b) multiplexador

A área ocupada pelo elemento de comutação é a maior entre todas as tecnologias, pois todo elemento possui uma célula de SRAM associada a ele. E a configuração é volátil, devendo ser restaurada sempre que o dispositivo for religado. Por outro lado, a reprogramação é rápida e pode ser feita com o dispositivo ligado ao circuito, em um número ilimitado de vezes. O processo de fabricação desta tecnologia é o mais simples, e é o mesmo utilizado na fabricação de memórias RAM, fazendo com que os avanços nesta área possam ser imediatamente aplicados na fabricação de FPGAs.

Vista como um problema para aplicações *stand-alone*, a volatilidade deste tipo de elemento de comutação é encarada como uma característica positiva para aplicações onde o FPGA é utilizada como um acelerador de processamento {Hauck, 1998}. Permite que um mesmo dispositivo implemente vários algoritmos de processamento. Por isso FPGAs baseados em SRAM se tornaram os mais utilizados nesta área {O'Donnell, 1998}. Quando se omitir a especificação da tecnologia de programação de um FPGA, entenda-se que seja baseada em SRAM.

3.2.2. Arquitetura de Blocos Lógicos

Os blocos lógicos de um FPGA são as unidades de processamento do dispositivo. Normalmente são limitados a implementar funções booleanas de 2 a 5 bits gerando saídas de 1 ou 2 bits, sendo que estes bits podem ser registrados por meio flip-flops, para implementação de circuitos digitais sequenciais.

Os blocos lógicos dos FPGAs são classificados a grosso modo em blocos de granularidade fina ou grossa {Brown e Rose, 1996}. Blocos de granularidade fina são aqueles formados por portas lógicas de 2 ou 3 entradas, e os de granularidade grossa, formados por LUTs de até 5 entradas.

A principal vantagem no uso de blocos de granularidade fina é a eficiência no aproveitamento dos seus recursos. Um bloco utilizado aproveita maior parte dos seus recursos, não importa quão simples seja a função implementada. A desvantagem reside no fato de ser necessário em um número muito grande de blocos lógicos para implementar funções complexas, exigindo mais recursos de interconexão nos dispositivos {Hauck, 1998}, o que ocupa maior área no dispositivo e torna mais lenta a operação do FPGA.

FPGAs de granularidade grossa são baseados em *look-up tables* (LUTs). Uma LUT é uma pequena memória de um bit de largura que possui suas linhas de endereçamento ligadas aos sinais de entrada do bloco lógico. Esta memória pode implementar qualquer função booleana que envolva as suas entradas, através da programação adequada do conteúdo da memória. LUTs apresentam um alto grau de funcionalidade, mas são limitadas a um número reduzido de entradas, pois sua área cresce exponencialmente com o aumento do número de entradas. Um compromisso entre área e funcionalidade do bloco lógico sugere a utilização de LUTs de 4 entradas {Chow e outros, 1999}. Outra estratégia é a utilização de *clusters* de LUTs nos blocos lógicos {Wasson, 1994}, permitindo uma maior granularidade e reduzindo a utilização de roteamento entre blocos. Isto é feito na família de FPGAs Xilinx XC4000 {Xilinx, 1998}, onde são utilizadas 2 LUTs 4x1 e 1 LUT 3x1.

Para FPGAs com blocos lógicos de granularidade grossa a inconveniência é que se a função implementada for simples, grande parte do potencial do bloco fica desperdiçado. Isso é tão mais grave quanto maior for a granularidade do bloco lógico.

3.2.3. Arquitetura de Roteamento

A arquitetura de roteamento de um FPGA é a maneira pela qual os comutadores programáveis e segmentos de trilha são posicionados para permitir a interconexão dos blocos lógicos. Entre os parâmetros que definem uma arquitetura de roteamento estão o número de trilhas e o seu comprimento. Estes parâmetros estão associados com a granularidade do bloco lógico. Quanto maior a granularidade, menos blocos lógicos serão necessários para implementar

determinada função e por isso menos trilhas serão utilizadas na conexão.

O comprimento das trilhas de roteamento afeta a velocidade de operação do dispositivo. Um roteamento a longas distâncias construído a partir de rotas curtas impõe um atraso muito grande ao sinal a ser conduzido. Uma rota longa é considerada uma rota rápida, mas diminui a densidade de roteamento, por ocupar uma grande área para transmissão de apenas um sinal.

FPGAs com recursos de roteamento bem distribuídos facilitam o trabalho das ferramentas automação de projeto eletrônico (EDA) por oferecerem mais alternativas para o roteamento, não exigindo um posicionamento otimizado dos blocos lógicos. Para FPGAs com recursos de roteamento abundantes, como a Xilinx XC4000 {Xilinx, 1998}, ferramentas EDA conseguem posicionar e rotear automaticamente projetos que utilizam quase toda a capacidade de um determinado dispositivo {Xilinx, 1998}. Já para FPGAs com recursos de roteamento escassos, como os da família de FPGAs de granularidade fina Xilinx XC6200, as ferramentas apenas conseguem posicionar e rotear automaticamente projetos que utilizem menos de 25% da capacidade do dispositivo {Hartenstein e outros, 1998}. Nestes casos é necessária a intervenção do projetista para posicionar manualmente os blocos lógicos de modo a facilitar o roteamento automático.

3.3. Aplicações

Inicialmente FPGAs foram desenvolvidos para implementação de *glue logic* em sistemas digitais. O aumento da sua capacidade tornou possível a implementação de sistemas digitais completos em um único *chip*, conhecidos por SOC (*System-On-Chip*). Apesar de apenas recentemente (início da década de 90) serem desenvolvidos FPGAs com capacidade suficiente para viabilizar o conceito de SOC, os sistemas integrados em um *chip* evoluíram significativamente, passando da integração de um número limitado de subsistemas pouco complexos até a integração de múltiplos processadores, periféricos, subsistemas de comunicação, hierarquias de barramentos, etc {Rincon e outros, 1999}.

A possibilidade de programação pelo usuário associada aos FPGAs de alta capacidade permitiu o florescimento de uma indústria de produção de *cores*, que são subsistemas digitais pré-testados. Existem *cores* que implementam processadores, interfaces de redes, arquiteturas especializadas para processamento de sinais, etc. Isto reduz o tempo de desenvolvimento de uma aplicação complexa, pois basta ao projetista selecionar um ou mais *cores* entre uma ampla gama a

sua disposição no mercado, e se preocupar com a sua integração {Tuck, 1996}. Isto é cada vez mais importante a medida que os projetos aumentam de complexidade.

A emulação e prototipação é uma área onde FPGAs tem aplicação crescente. Como os circuitos integrados estão cada vez mais complexos, sua validação é muito lenta quando feita por meio de simuladores baseados em software. Por isso os testes não são tão completos quanto poderiam ser, permitindo que um projeto defeituoso seja aprovado para fabricação. Se um conjunto de FPGAs for utilizado para emular o circuito em teste, a velocidade de simulação será várias ordens de grandeza maior que a simulação em software {Hauck, 1998} e mais completos poderão ser os testes. A medida que problemas são detectados e corrigidos, os FPGAs podem ser rapidamente reprogramados para refletir estas alterações. Esta estratégia foi utilizada pela Intel no desenvolvimento do processador Pentium {Chan e Mourad, 1994}.

FPGAs também podem ser vistos como elementos de processamento, com características muito diversas dos processadores clássicos. FPGAs possuem um massivo paralelismo de granularidade fina, sendo que cada bloco lógico do FPGA pode ser considerado uma unidade de processamento com palavra de um ou poucos bits {Hauck, 1998}. Configurando e interligando estas unidades de processamento de modo a explorar o paralelismo do algoritmo a ser processado, é possível conseguir acelerações substanciais sobre o processamento sequencial executado por processadores clássicos. FPGAs assim configurados implementam processadores *data-flow**, em contrapartida com os processadores *control-flow* tradicionais {Brebner, 1998a}.

Aproveitando a característica de reconfigurabilidade de FPGAs baseadas em SRAM, arquiteturas *data-flow* podem ser construídas "sob medida" para cada algoritmo, confundindo e misturando os domínios do hardware e do software. Por isso FPGAs são o agente catalisador para a criação de processadores adaptados à aplicação, ou *Custom Computing Machines* (CCM) {Bergmann e Mudge, 1994}.

3.3.1. FPGAs como processadores VLIW

FPGAs também podem ser encarados como processadores de palavra de instrução muito longa (VLIW), onde cada bloco lógico é uma unidade funcional controlada diretamente por alguns bits (uma ou duas centenas nos FPGAs baseados em LUT mais comuns {DeHon, 1996}) da palavra de instrução. A palavra de instrução seria composta pela concatenação dos bits de

configuração de todos os blocos lógicos e pode atingir vários milhares de bits de largura {Shiratsuchi, 1997}. Seguindo este conceito, FPGAs seriam como processadores VLIW sem unidade de *fetch* de instrução e com cache de exatamente 1 instrução, em muitos casos a única executada durante o processamento.

Analisando sob este ponto de vista, a capacidade de reconfiguração de FPGAs baseados em SRAM é fundamental, pois assim é possível alterar o cache de instruções para a permitir a execução de mais de uma instrução (outras configurações do FPGA).

A maioria dos FPGAs comerciais existentes foi desenvolvida para substituir circuitos ASIC em pequeno volume, e seus fabricantes encaram a capacidade de reconfiguração como um mero recurso de atualização de hardware {Shiratsuchi, 1997}. Analisar um FPGA sob a ótica de um processador VLIW ajuda a evidenciar algumas deficiências das arquiteturas existentes e apontar características que seriam bem-vindas:

- reconfiguração rápida
- reconfiguração dinâmica
- múltiplos contextos de configuração

A reconfiguração rápida permite que mais instruções sejam lidas por intervalo de tempo, tornando aceitável em termos de desempenho dividir a solução de um problema em múltiplas instruções. A redução do número de bits na palavra de instrução através de uma codificação mais eficiente traz benefícios neste sentido e {DeHon, 1996} aponta alguns resultados nesta direção.

A reconfiguração dinâmica é a capacidade de reprogramar o FPGA sem interromper a sua operação. Esta capacidade permite a reconfiguração parcial do dispositivo e também contribui para a rapidez da operação. Alguns trabalhos significativos tratando de reconfiguração dinâmica em FPGAs podem ser encontrados em {Spillane e Owen, 1998} {Brebner e Donlin, 1997} {Brebner, 1998b} {Wirthlin e Hutchings, 1999}.

A existência de múltiplos contextos de configuração é outra característica que contribui para a eficiência da divisão de tarefas em várias instruções (ou configurações) para o FPGA. Contextos múltiplos em um FPGA são análogos ao cache de instruções em processadores, permitindo o armazenamento de mais de uma instrução, além daquela que está sendo executada no momento. Este conceito foi explorado por vários pesquisadores na proposta de novas

---

* O termo *data-flow* aqui não se aplica a processadores *data-flow*, mas ao conceito de arquitetura *data-driven*

arquiteturas de FPGA {Tau e outros, 1995} {DeHon, 1996} {DeHon, 1996} {Faura e outros, 1997}.

Não existe nenhuma arquitetura de FPGA comercialmente disponível que combine estas características todas. A que melhor preenche estes requisitos é a família de FPGAs Xilinx XC6200 {Shiratsuchi, 1997}. Ela não suporta múltiplos contextos de configuração, mas permite reconfiguração rápida, dinâmica e parcial. Uma comparação do tamanho da palavra de configuração e dos tempos de configuração desta arquitetura com outras convencionais do mesmo fabricante pode ser vista na Tabela 3.1. O tamanho da palavra de reconfiguração para o XC6216 depende do número de blocos lógicos que serão reconfigurados, e também da regularidade da configuração a ser programada.

Tabela 3.1: Características de reconfiguração de FPGAs comerciais

FPGA	Palavra config (bits)	Tempo config (ms)	Arranjo dos blocos	Complexidade (K portas)	Modo config
XC4003	53976	5.4	10x10	2-5	serial
XC4013	247960	24.8	24x24	10-30	serial
XC4028	668167	8.4	32x32	18-50	paralelo
XC4062	1433847	17.9	48x48	40-130	paralelo
XC6216	depende	40ns/bloco	64x64	16-24	paralelo/parcial

3.3.2. A FPGA XC6200

A família de FPGAs Xilinx XC6200 é uma família de FPGAs de granularidade fina. Cada bloco lógico (Figura 3.3) pode implementar qualquer função booleana de 2 entradas, inclusive um multiplex 2:1, por meio de multiplexadores de 2 e 4 entradas. Estas funções podem ser registradas por um flip-flop contido dentro do bloco. É uma arquitetura minimalista, se comparada com aquelas baseadas em LUT (*look-up table*) utilizadas em outros FPGAs, como os Altera FLEX 10K {Altera, 1998} e Xilinx XC4000 {Xilinx, 1998}.

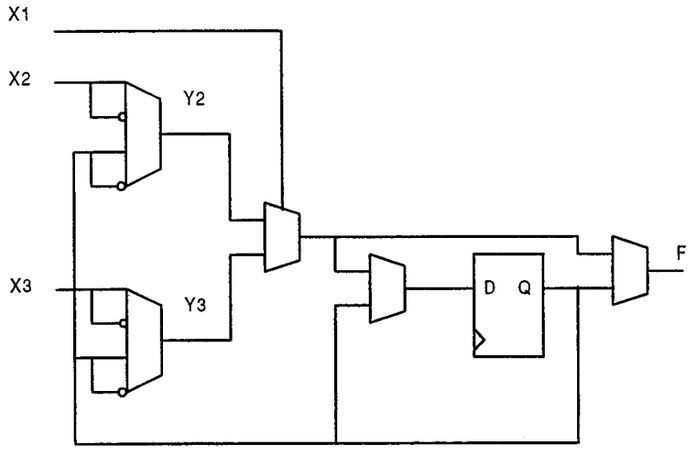


Figura 3.3: Estrutura do bloco lógico

Os blocos lógicos são dispostos em um arranjo retangular de células, arranjo este cujo tamanho varia nos diferentes dispositivos da família. No modelo XC6216 (Figura 3.4) o arranjo possui 64x64 células, e no XC6264, 128x128 células {Xilinx, 1998}.

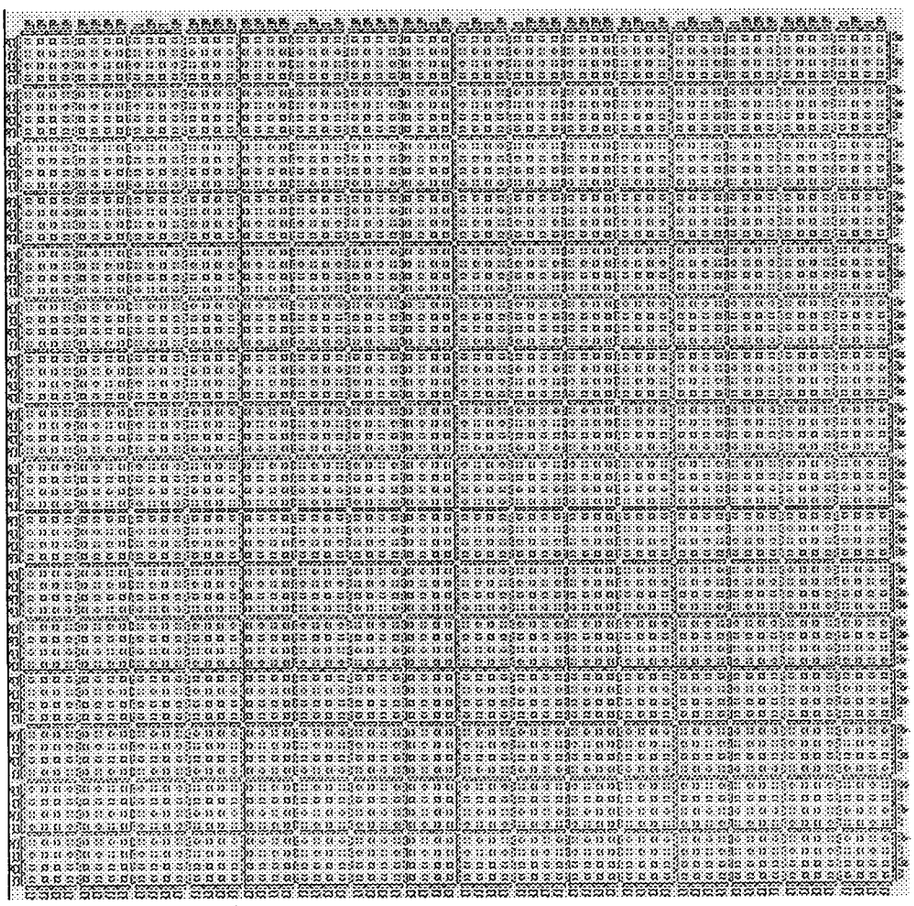


Figura 3.4: Layout de um FPGA XC6216

Os flip-flops são do tipo D síncronos, e podem ser configurados para *set* ou *reset* assíncrono. Não possuem controle de habilitação, que deve ser implementado utilizando a lógica combinacional presente no bloco lógico. As rotas de distribuição de *clock* e *set/reset* são exclusivas, embora possam ser utilizadas para conduzir outros sinais em certos casos. Cada linha de *clock* corre verticalmente e alimenta 4 blocos lógicos. As linhas de *set/reset* também correm verticalmente, mas cada uma delas alimenta 16 blocos lógicos. Cada linha de *clock* e *set/reset* pode conduzir um sinal diferente, permitindo circuitos síncronos com múltiplos *clocks*. As redes de roteamento destes sinais são distribuídas por toda a área do *chip* de modo a minimizar o *skew* e garantir um comportamento previsível em circuitos síncronos. Além destas rotas, existem também duas outras redes de distribuição globais de baixo *skew* para sinais de controle definidos pelo usuário.

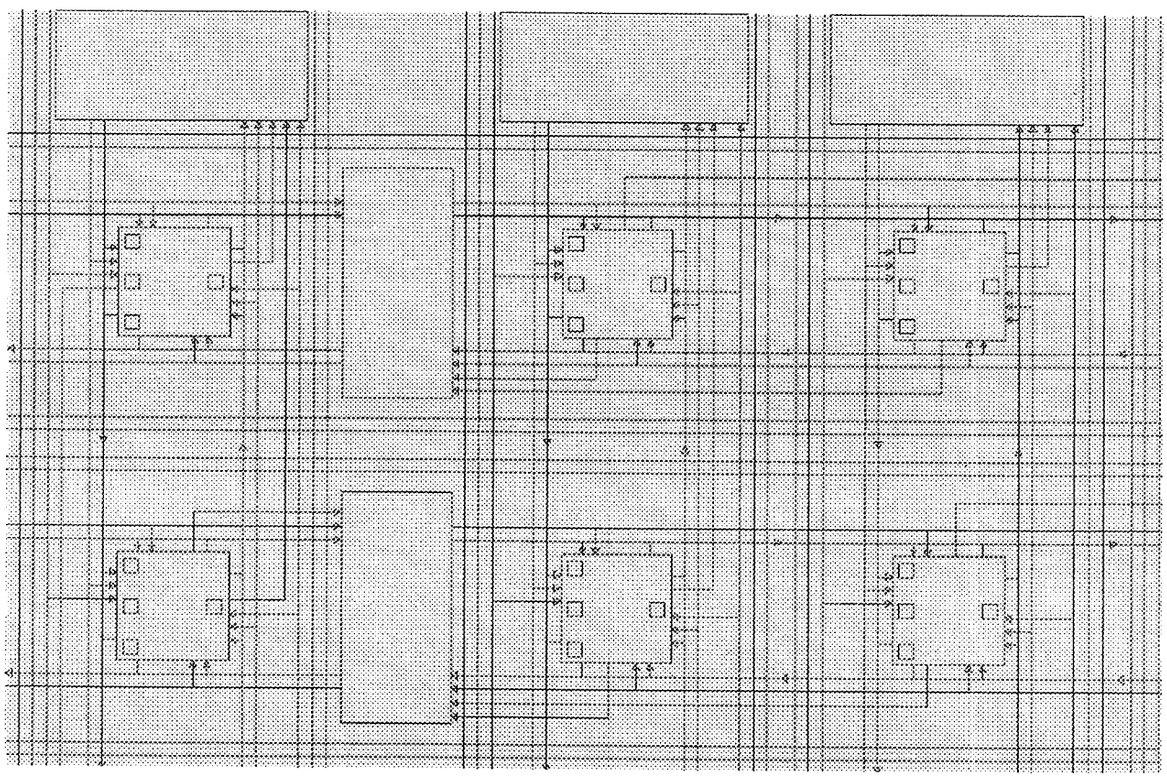


Figura 3.5: Detalhe do roteamento entre blocos lógicos

A arquitetura deste FPGA é hierárquica. Células vizinhas são agrupadas em blocos com 4x4 células, formando a primeira hierarquia. Esses blocos comunicam-se com seus vizinhos, formando blocos maiores de 16x16 células. No dispositivo XC6216, como ilustra a Figura 3.4, um arranjo 4x4 desses blocos de 16x16 células forma um arranjo de 64x64 células, que possui uma bloco de I/O em cada célula da borda. Cada nível de hierarquia está associado a um plano de

roteamento que possui rotas horizontais e verticais e está interligado com os outros para maximizar as possibilidades de roteamento. Cada célula pode se comunicar com suas vizinhas através de rotas locais, e com células distantes utilizando roteamento dos planos superiores, que atravessam 4, 16 ou 64 células. Esses segmentos longos são denominados FastLANEs™.

O bloco lógico da Figura 3.3 pode ter suas entradas X1, X2 e X3 alimentadas diretamente a partir de linhas locais ou de comprimento 4 vindas de qualquer direção. A saída do bloco lógico também pode ser programada para alimentar uma ou mais linhas locais de qualquer direção. Existe também um recurso extra de roteamento chamado de *magic lines*, que permite fazer curvas de 90° com os sinais. Estes recursos são mostrados na Figura 3.6.

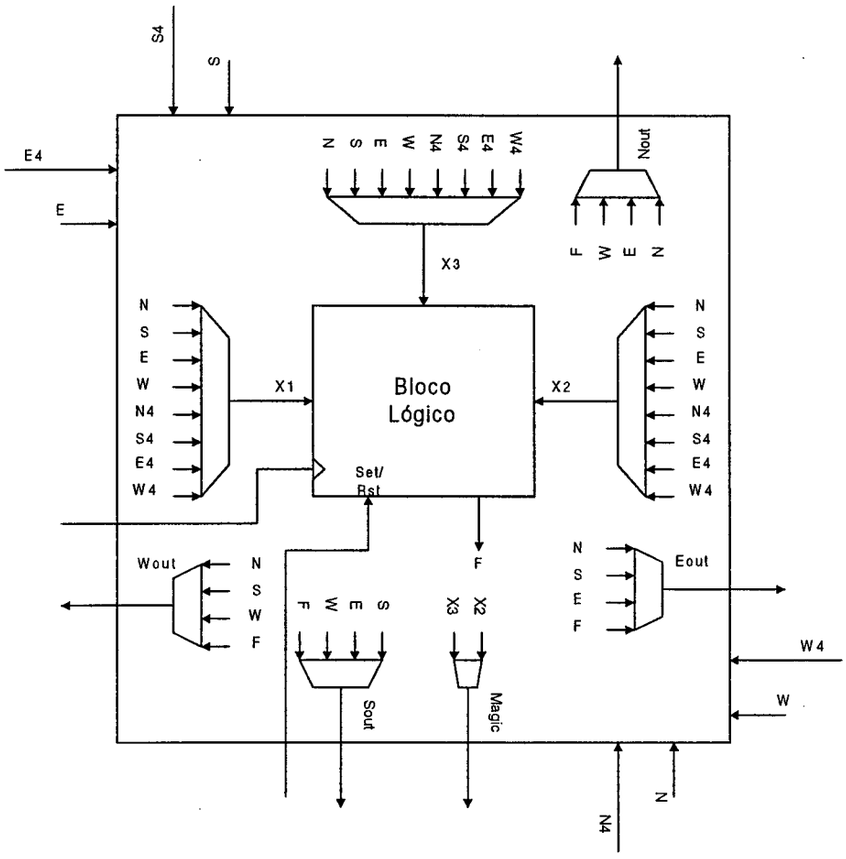


Figura 3.6: Célula básica da família XC6200

Comparada com outras arquiteturas de FPGAs, esta apresenta poucos recursos de roteamento, especialmente se considerando sua granularidade fina. Por outro lado existe uma proporção elevada de registros em relação à lógica combinacional - um para cada função de 2 entradas. Isto é muito conveniente para estruturas regulares que explorem intensivamente o *pipelining*.

A arquitetura da família de FPGA XC6200 é altamente simétrica. Não existem blocos lógicos com funções especiais, e com exceção das rotas globais, nem recursos de roteamento dedicados. Não possuem linhas roteamento rápido para *carry*, que podem oferecer ganhos de velocidade na operação de circuitos aritméticos da ordem de 4 vezes{Gschwind e Salapura, 1995b}.

Mas o ponto onde esta arquitetura efetivamente brilha é o seu suporte à reconfiguração dinâmica e a sua interface simplificada com processadores. Daí a razão da denominação também aplicada a este dispositivo, de RPU (*Reconfigurable Processor Unit*). A capacidade de reconfiguração desta arquitetura inclui reconfiguração parcial, e reconfiguração múltipla. A reconfiguração múltipla consiste em reconfigurar dinâmicamente vários blocos lógicos de uma mesma coluna através da configuração de um único bloco lógico. Aliada à reconfiguração parcial, este recurso reduz significativamente o tempo de reconfiguração.

A interface simplificada com processadores denominada FastMap™ {Churcher e outros, 1995} é uma característica única entre todas as FPGAs comerciais {Barr, 1998} {Davis e outros, 1998}. É a primeira FPGA projetada visando cooperação com processadores {Kean e outros, 1996}. Cada célula tem seus bits de configuração mapeados em um endereço. Desse modo, a reconfiguração pode ser feita de um modo seletivo, apenas sobre as células que devem ser alteradas. E as saídas de todos os blocos lógicos podem ser lidas pelo processador diretamente através da interface FastMap™, sem utilizar pinos de I/O externos e nem recursos de roteamento internos. A interface FastMap™ é semelhante à interface de uma memória, e permite que o estado dos blocos lógicos seja lido ou alterado pelo processador do mesmo modo que se acessa o conteúdo de uma memória. Embora o termo "memória ativa" tenha sido cunhado tendo em vista outra arquitetura de FPGA {Vuillemin e outros, 1996}, é a melhor palavra para descrever a esta.

Este conjunto de características é extremamente útil em aplicações de computação reconfigurável, principalmente aplicações que exigem reconfiguração dinâmica e parcial {Bouridane e outros, 1999} {Fawcett, 1996} {Brebner e Gray, 1995} {Trainor e outros, 1998} {Woods e outros, 1998}.

3.4. Paradigmas de aceleração

Um algoritmo pode ser acelerado através da sua implementação em hardware. As lógicas reconfiguráveis, como FPGAs baseados em SRAM, permitem a implementação de soluções

específicas para uma determinada aplicação sem perda de generalidade. Este tipo de soluções são denominadas *Custom Computing Machines* (CCMs). Mas existem alguns problemas nesta abordagem:

- Complexidade
- Eficiência

A complexidade de algoritmos úteis normalmente não permite que eles sejam completamente implementados em hardware. O processamento *data-flow* inerente ao hardware não é tão eficiente quanto o *control-flow* na execução de máquinas de estado complexas {Brebner, 1998a}, e processadores clássicos implementados em hardware fixo são otimizados para a máxima eficiência neste tipo de tarefa {Hauser e Wawrzynek, 1997}.

Por isso a maior parte das soluções que utilizam lógicas reconfiguráveis para aceleração de um determinado algoritmo também utilizam processadores associados a elas. O princípio por trás desta abordagem é executar em hardware os 10% de código que tomam 90% do tempo de processamento {Patterson e Hennessy, 1995} e os 90% de código restantes em processadores clássicos.

Mas a execução de parte do algoritmo em hardware implica na comunicação entre a lógica reconfigurável e o processador clássico. A aceleração oferecida pelo hardware deve ser tal que compense estes atrasos de comunicação.

Seja T_H o tempo de execução de uma tarefa em hardware, T_S o tempo de execução desta tarefa em software, e T_C o tempo de comunicação entre a lógica reconfigurável e o processador necessário para a tarefa em questão. Para que a lógica reconfigurável acelere determinada tarefa:

$$T_H + T_C < T_S$$

A intensidade da aceleração é avaliada pela relação:

$$\frac{T_H + T_C}{T_S} = \frac{T_H}{T_S} + \frac{T_C}{T_S}$$

Se o tempo de comunicação é grande em comparação com o tempo de processamento toda a aceleração trazida pelo hardware pode ser anulada. Para evitar isso as alternativas são reduzir o tempo de comunicação e/ou aumentar o tempo de execução da tarefa, através do aumento da granularidade da tarefa. Como o tempo de comunicação não é linearmente dependente da granularidade da tarefa, uma tarefa de granularidade elevada pode permitir uma

aceleração maior que uma tarefa de granularidade reduzida.

{Page, 1996} e {Wittig e Chow, 1996} classificam CCMs quanto à granularidade da tarefa executada pela lógica reconfigurável e seu grau de independência em relação ao processador, características que na prática tem alto grau de correlação. {Hauck, 1998} propôs uma classificação baseada localização da lógica reconfigurável na hierarquia de barramentos do processador, como mostrado na Figura 3.7. Lógicas reconfiguráveis desempenhariam os seguintes papéis:

- Unidade funcional
- Coprocessador
- Processador
- Servidor de processamento

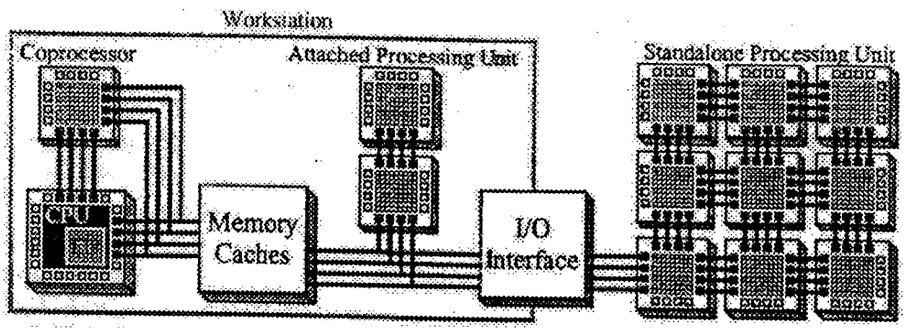


Figura 3.7: Paradigmas de aceleração com FPGAs

3.4.1. Unidade Funcional

A lógica reconfigurável pode acelerar o processamento implementando novas unidades funcionais especializadas dentro do processador. A criação destas novas unidades funcionais pode implicar na extensão do conjunto de instruções com novas instruções especializadas.

Por serem tarefas de granularidade fina, para haver aceleração significativa o tempo de comunicação deve ser mínimo. Por isso a lógica reconfigurável é integrada ao processador dentro do mesmo *chip*, compartilhando o barramento de alta velocidade, registradores e sinais de controle internos do processador.

Soluções utilizando este paradigma de aceleração variam desde a implementação de unidades funcionais em paralelo com a ALU de um processador clássico {Gustin, 1999} até a implementação de todo o processador em lógica reconfigurável {Wirthlin e outros, 1994}

{Wirthlin e Hutchings, 1995}. Implementar um processador inteiro em FPGA tem a desvantagem de empregar lógica reconfigurável para implementar estruturas que, por não precisarem ser especializadas ou serem muito complexas, são melhor implementadas em hardware fixo. Contador de programa, multiplicadores, somadores, são exemplos de unidades que podem ser implementadas diretamente em hardware sem perda de generalidade e com muito maior eficiência de área e desempenho. Por isso processadores implementados completamente em lógica reconfigurável (*soft cores*) são limitados a processadores simples cujas instruções são executadas sequencialmente.

Processadores mais complexos podem ser implementados em lógica fixa (*hard core*) na mesma pastilha da lógica reconfigurável. {Wittig e Chow, 1996} utiliza um *hard core* de processador de 32 bits associado a uma certa quantidade de lógica reconfigurável e justifica esta estratégia analisando a implementação em lógica reconfigurável de unidades típicas de um processador, como controlador de RAM dinâmica e unidade de UART. Os resultados mostram que se implementados como *hard core* ocupam área de silício 20 a 25 vezes maior que se fossem implementados como um *soft core**.

A criação de novas instruções implica na extensão da linguagem montadora do processador. Assemblers parametrizáveis foram propostos por {Clark, 1996}, e solucionam o desenvolvimento em baixo nível. Mas no caso de linguagens de alto nível, os compiladores teriam que traduzir as operações descritas na linguagem utilizando um conjunto de instruções potencialmente infinito. Seriam muito complexos, e consequentemente, pouco eficientes {Patterson e Hennessy, 1995}.

Outra questão levantada por {Hauck, 1998} é se a área utilizada na implementação da lógica reconfigurável não seria melhor utilizada se implementasse caches maiores para o processador. É uma questão deixada em aberto, mas pertinente, pois a área de silício normalizada do processador MIPS-X de 32 bits completo é 23 vezes menor {Wittig e Chow, 1996} que a de uma FPGA Xilinx XC4010, com apenas 400 blocos lógicos e capacidade de aproximadamente 10K portas lógicas {Xilinx, 1998}.

3.4.2. Coprocessador

Lógica reconfigurável operando como um coprocessador não compartilha o *datapath* do

processador, mas tem acesso ao cache de dados. Deve implementar tarefas de granularidade mais elevada, uma vez que a comunicação com o processador não é tão rápida. Pode ser implementada em *chip* separado, neste caso a comunicação com o processador se dá por meio de barramentos e sinais externos.

As tarefas são explicitamente invocadas através da execução sequências de instrução no processador principal que habilitem o processamento na lógica reconfigurável, como em {Hauser e Wawrzynek, 1997}. Os dados podem ser fornecidos pelo processador ou obtidos pela própria lógica reconfigurável. Como as tarefas são de granularidade mais elevada elas são executadas em paralelo com as tarefas executadas no processador, como processos paralelos.

Além de auxiliar em processamento, em alguns casos a lógica reconfigurável pode ser útil se funcionar como um coprocessador de I/O. Executando tarefas de baixo nível com os dados de entrada e saída, como detecção de erros de paridade ou monitorando e respondendo a eventos externos, ela alivia a carga de processamento do processador principal. Esta é uma tendência forte para processadores que devem ser utilizados em sistemas com poucos ou nenhum componente além do processador, como é o caso dos microcontroladores. A lógica atuando como coprocessador de I/O reconfigurável permite que periféricos especializados sejam acrescentados ao microcontrolador de acordo com as necessidades da aplicação, conforme relatado por {Anon, 1998} {Triscend, 1998}. Maior versatilidade foi obtida por {Faura e outros, 1997} que integrou ao *core* de um processador da família Intel 8051 lógica reconfigurável analógica e lógica reconfigurável digital multicontexto.

Este tipo de aplicação não exige que a lógica reconfigurável seja complexa, uma vez que a intenção é implementar periféricos simples. Por isso ela é integrada no mesmo *chip* do processador, não com o objetivo de compartilhar o *datapath*, mas para permitir uma interface entre ambos sem utilização de pinos adicionais.

3.4.3. Processador

A lógica reconfigurável operando como processador está conectada ao barramento de memória ou ao barramento de extensão do computador (ISA, PCI, etc). Executa tarefas de granularidade mais elevada em um conjunto de dados, que podem ser fornecidos pelo processador ou lidos da memória.

---

* Comparação feita normalizando a utilização da área de silício

Uma grande quantidade de aplicações aceleradas por lógicas reconfiguráveis atuando como processadores paralelos tem sido relatada. Especificamente na área de processamento de imagem existem muitos exemplos desta abordagem, implementando filtros {Hamid, 1994} {Singh e Bellec, 1994}, transformadas DCT {Altera, 1996} {Bergmann e Chung, 1997} {Woods e outros, 1998} e outras transformadas {Cucchiara e outros, 1998} {Dick, 1996} {Guermeur, 1998}.

A lógica reconfigurável pode assumir também o papel de um pré-processador paralelo. Exemplos disso são {Encinas e Moreira, 1999} e {Woodward e outros, 1995} fazendo o pré-processamento de bits em alta velocidade e {Greenfield e outros, 1997} aliviando um DSP da tarefa de filtragem digital de sinais.

Uma vez que lógicas reconfiguráveis operando como processos paralelos podem estar conectadas no barramento de extensão, não necessitando estar integradas no mesmo *chip* do processador, é possível construir processadores reconfiguráveis genéricos. Implementados em placas de extensão com um ou mais FPGAs ligado a memórias locais, comunicando-se com o processador principal através de um barramento de alta velocidade, estes sistemas genéricos permitem a implementação de processadores reconfiguráveis adaptáveis a uma ampla gama de aplicações. Exemplos são {Kostarnov e outros, 1997}, que descreve Riley, um sistema que associa uma FPGA Xilinx XC4013 com um processador Intel i960; PAMette {Shand, 1996}, que integra 4 FPGAs Xilinx 4010 em uma placa PCI e cujas aplicações são descritas em {Shand, 1995}; {Nisbet e Guccione, 1997}, que descreve uma placa PCI com um FPGA XC6216 e 2 MB de SRAM, utilizada por {Donlin e outros, 1997} para aceleração do processamento da linguagem PostScript; Hades {Ludwig, 1996}, uma placa de expansão específica para a *workstation* Ceres-2 utilizando um FPGA XC6216 e 256 KB de SRAM, empregada para testes na área de computação reconfigurável {Wirth, 1998} {Gehring e Ludwig, 1996}.

3.4.4. Servidor

Lógicas reconfiguráveis atuando como servidores de processamento se comunicam com o processador principal através de canais de entrada e saída de uso geral, como interfaces seriais ou de rede, e executam tarefas de granularidade muito elevada. Muitas vezes efetuam todo o processamento necessário sem ajuda de processadores associados, que são utilizados apenas para alimentar os dados e/ou apresentar os resultados {Guccione e Gonzalez, 1995}. Como a maior parte do processamento é feito pela lógica reconfigurável, o atraso da comunicação com o

processador principal não é um fator tão importante no atraso da aceleração.

Servidores de processamento implementados em lógicas reconfiguráveis são conceitualmente similares aos processadores paralelos, mas são muito mais complexos. Geralmente são compostos por dezenas de FPGAs de capacidade elevada e com grande quantidade de memória à disposição. Para tornar este conjunto o mais genérico possível e para aproveitamento de todo o seu potencial, mecanismos de interconexão são necessários. As topologias de interconexão mais utilizadas são a de barramento, onde conjuntos de FPGAs e memórias compartilham o mesmo meio de transmissão, a de *mesh* , em que FPGAs/memórias vizinhos são ligados entre si, e a de *crossbar*, onde FPGAs/memórias são interligados de maneira a oferecer a mesma distância em qualquer ligação {Hauck, 1998}.

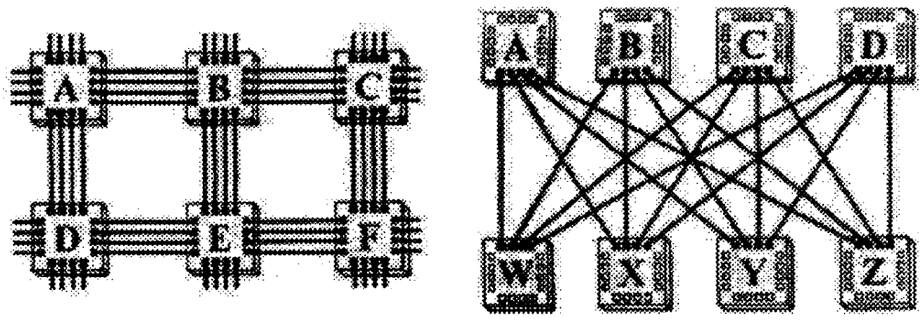


Figura 3.8: Topologia de interconexão *mesh* e *crossbar*

Estas topologias e suas combinações, algumas vezes em diferentes hierarquias, são utilizadas nos servidores de processamento. Exemplos destes servidores são o Splash {Gokale e outros, 1994}, uma topologia híbrida de *mesh* linear de 16 FPGAs Xilinx XC4010 conectados a memórias locais com um *crossbar* implementado com outro FPGA interligando estes conjuntos. Esta topologia é apropriada para implementar arquiteturas sistólicas, que utilizam intensamente comunicação local.

Outro processador de alta capacidade é o DECPeRLe {Vuillemin e outros, 1996}, cuja topologia é um híbrido de *mesh* bidimensional 4x4 de FPGAs Xilinx XC3090 associado com barramentos horizontais e verticais que cruzam estes arranjo. Esta topologia reproduz em maior escala a topologia de um FPGA, com ligações locais entre os blocos lógicos e rotas longas para ligações entre blocos não adjacentes. As memórias SRAM estão conectadas aos 4 FPGAs que controlam os barramentos, sendo 4 MB no total. As aplicações implementadas neste processador são relatadas em {Bertin e outros, 1993} e variam desde multiplicações em palavras muito longas

(2048 bits) até a implementação de transformadas DCT.

3.5. Considerações Finais

FPGAs são utilizadas para processamento e não só para implementar *glue logic*. A capacidade de reconfiguração é importante para permitir o reaproveitamento dos recursos lógicos do FPGA em outras tarefas. Quanto mais rápida for a velocidade de reconfiguração, maiores serão as oportunidades para utilizar este reaproveitamento com eficiência. Reconfiguração parcial é interessante pois permite a reconfiguração durante a operação do dispositivo.

Uma das poucas famílias de FPGA que suportam estas características é a família XC6200. O fato de ser projetada tendo em vista a comunicação com um processador torna este dispositivo muito adequado para a implementação de CCMs do tipo coprocessador e processador paralelo.

4. A Transformada Wavelet

4.1. Introdução

Conforme discutido no Capítulo 2, um modo de evidenciar correlações na imagem seria utilizar algumas bases mais convenientes para representar a informação. Existe uma maneira de se determinar a base ideal para representação, se é conhecida de antemão a correlação presente no conjunto de dados {Sweldens, 1995}. A transformada Karhunen-Loève (KLT) é muito eficiente sob o ponto de eliminação da redundância nos *pixels* da imagem. Mas esta transformada teoricamente ótima, apresenta alguns problemas de ordem prática:

- Dependência dos dados
- Conhecimento da matriz de correlação
- Complexidade

A KLT é ótima para um conjunto de dados que siga o modelo estatístico representado pela matriz de correlação. Como normalmente este modelo estatístico não é conhecido, ele pode ser estimado a partir dos dados a serem transformados. Este processo envolve a determinação da matriz de covariância e seus autovetores, e é um algoritmo de complexidade $O(N^3)$. Com a matriz de transformação ótima determinada, a KLT é um algoritmo cuja complexidade é de ordem $O(N^2)$.

Por isso a representação Karhunen-Loève não é utilizada diretamente na compressão de imagens em tempo real. O que se necessita é uma transformada que seja independente do conjunto de dados, rápida (de ordem menor que $O(N^2)$), boa capacidade de eliminação de correlação e capacidade de concentração da energia do sinal em poucos coeficientes. A DCT é uma transformada que atende parcialmente os requisitos acima, e por isso vem sendo utilizada largamente na compressão de imagens. Embora as bases seno e cosseno tenham alta resolução espectral, pois são locais (ou localizadas) no domínio da frequência, elas não são de modo algum locais no espaço* devido ao fato delas terem comprimento infinito {Vetterli e Herley, 1992}. Por isso não é possível representar impulsos ou descontinuidades em uma região limitada no domínio do espaço com um conjunto limitado de componentes no domínio da frequência.

---

* Como esta discussão se refere especificamente à compressão de imagens estáticas, as referências ao domínio do tempo, em oposição ao domínio da frequência, serão substituídas por referências ao domínio do espaço.

4.2. A Transformada de Fourier Limitada

Para minimizar este problema Gabor propôs a Transformada de Fourier Limitada ou *Short Time Fourier Transform* (STFT), também conhecida como *Windowed Fourier Transform* (WFT) {Fournier, 1995}. Ela se utiliza de uma função-janela $w(x)$ centrada na região de interesse, modulando as bases seno/cosseno {Rioul e Vetterli, 1991}.

$$F_{ST}(\tau, f) = \int_{-\infty}^{\infty} f(x) w^*(x-\tau) e^{-i2\pi fx} \, dx$$

Utilizando a função Gaussiana como função-janela (Figura 4.1a), pode ser considerado que a transformada utiliza uma nova base:

$$s(x,\tau) = w(x-\tau)e^{-2i\pi fx} \tag{4.1}$$

A base $s(x,\tau)$ é limitada no espaço (Figura 4.1b). Como a Figura 4.1c mostra, a discriminação de frequências é muito menos precisa a discriminação das bases seno/cosseno, cujo espectro é um impulso. Quanto menor o decaimento da função Gaussiana, mais as bases no domínio da frequência se aproximarão do impulso, e melhor será a resolução no domínio da frequência.

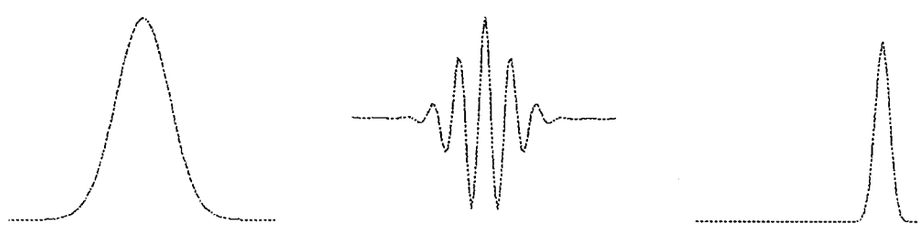


Figura 4.1: a) Função-janela Gaussiana no domínio do espaço b) Base da STFT no domínio do espaço c) Base da STFT no domínio da frequência

Isso pode ser representado no plano posição-frequência da Figura 4.2. O coeficiente da STFT associado a uma determinada posição τ_i e frequência f_j é uma componente que representa um conjunto de frequências em torno de f_j naquela posição do espaço. Isso significa que a capacidade de resolução de detalhes no domínio da frequência não é mais infinita como na transformada de Fourier.

A capacidade de resolução de detalhes no espaço e na frequência é inversamente relacionada com o menor detalhe que pode ser resolvido nestes domínios. $\Delta\tau$ é o menor detalhe

no domínio do espaço e Δf no domínio da frequência. Seja $w(t)$ a função janela da STFT e $W(f)$ a sua transformada de Fourier, o menor detalhe na frequência e no espaço resolvido pela STFT pode ser determinada por (4.2) (4.3) {Rioul e Vetterli, 1991}.

$$\Delta f^2 = \frac{\int f^2 |W(f)|^2 \, \mathrm{d} f}{\int |W(f)|^2 \, \mathrm{d} f} \tag{4.2}$$

$$\Delta \tau^2 = \frac{\int \tau^2 |w(\tau)|^2 \, \mathrm{d} \tau}{\int |w(\tau)|^2 \, \mathrm{d} \tau} \tag{4.3}$$

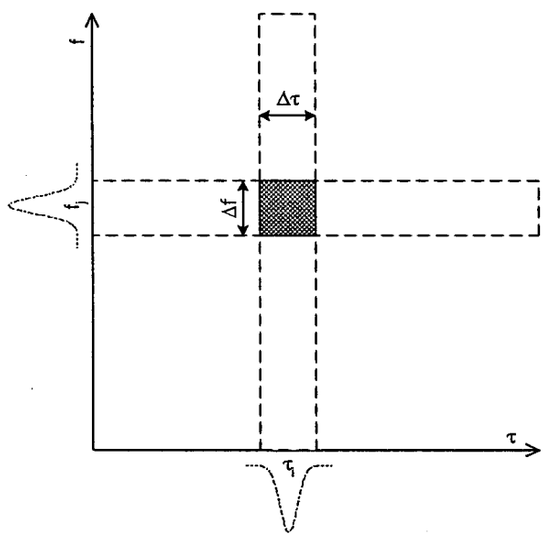


Figura 4.2: Região de discriminação da STFT

Estas resoluções não podem ser arbitrariamente pequenas. O Princípio da Incerteza, ou Desigualdade de Heisenberg {Fournier, 1995} afirma que o produto dessas resoluções é limitado por uma constante.

$$\Delta \tau \Delta f \geq \frac{1}{4\pi} \tag{4.4}$$

Na STFT estas resoluções são constantes em qualquer frequência ou posição da janela no espaço. Mas este comportamento das bases não é interessante se o sinal a ser transformado é uma imagem. Imagens são compostas por regiões relativamente uniformes seguidas de áreas de transição abrupta, como mostra a Figura 4.4. Estas áreas de transição representam as bordas dos objetos exibidos. São ricas em harmônicos e locais no espaço, ou seja, restritas a uma região limitada, como mostra a Figura 4.5. As transições poderiam ser representadas com eficiência se a

base utilizada pudesse discriminar pequenos detalhes no espaço, mesmo perdendo resolução na frequência. As áreas uniformes na imagem apresentam poucas variações e por isso precisam de bases com boa discriminação no domínio da frequência, mesmo que se estendam no espaço.

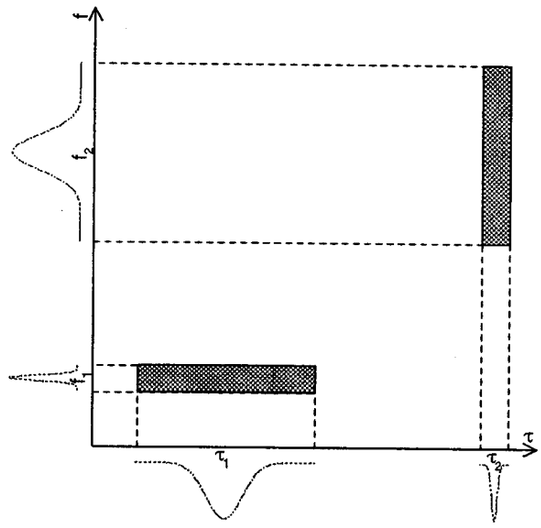


Figura 4.3: Região de discriminação de uma base ideal

Figura 4.4: Linhas de referência na figura Lena

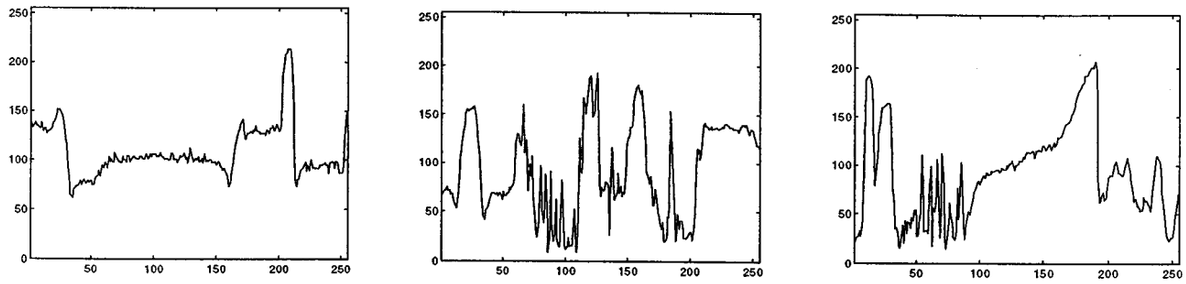


Figura 4.5: Luminância nas linhas de referência primeira, intermediária e última

Uma base assim deveria poder trocar a resolução temporal com a resolução espacial quando necessário. Isso pode ser conseguido através da introdução de um fator de escala no espaço e na frequência. Este fator de escala comprimiria ou dilataria a base de acordo com a necessidade. Considerando uma base muito parecida com a utilizada na STFT:

$$h(x) = w(x)e^{-2i\pi f_0 x} \tag{4.5}$$

Acrescentado um fator de escala a e um deslocamento τ à base h(x), a nova base seria:

$$h_{a,\tau}(x) = \sqrt{|a|}h(ax-\tau) \tag{4.6}$$

O fator $\sqrt{|a|}$ é necessário para normalizar a energia da nova base em suas várias escalas. Com essa nova base a STFT pode ser re-escrita como:

$$F_W(\tau, a) = \int_{-\infty}^{\infty} f(x)h_{a,\tau}(x)\,\mathrm{d}x \tag{4.7}$$

É importante observar que nesta base, assim como a largura da janela, a frequência também é associada ao fator de escala da forma $f = f_0/a$. Isto pode ser observado comparando o comportamento destas bases (Figura 4.6) com o comportamento das bases da equação (4.1), onde a largura da janela é fixa e apenas a frequência varia (Figura 4.7).

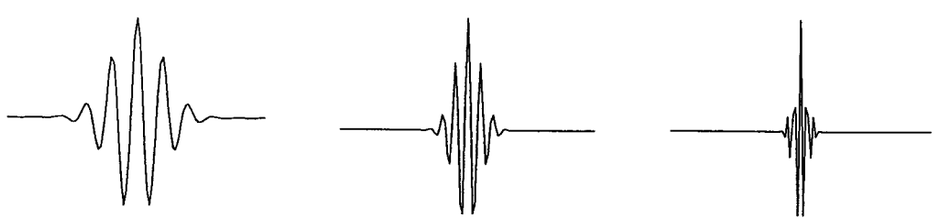


Figura 4.6: Comportamento das bases $h_{a,\tau}(x)$ em várias escalas

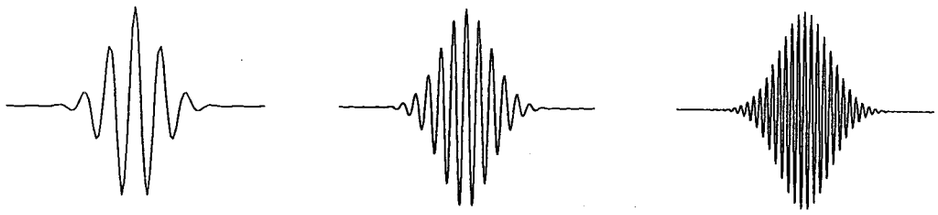


Figura 4.7: Comportamento das bases $s(x)$ em várias frequências

A equação (4.7) é a Transformada Wavelet (WT), e a base $h_{a,\tau}(x)$ é a wavelet básica. Esta base representa o sinal através de translações τ e dilatações sucessivas a. Como τ e a são variáveis contínuas, existe muita redundância nesta representação, assim com existe redundância na transformação da STFT {Vetterli e Herley, 1992}. Para remover esta redundância é necessário utilizar funções $h_{a,\tau}(x)$ que sejam bases ortonormais para uma sequência discreta de τ e a, o que infelizmente não se consegue com a função representada em (4.5).

Uma vez que se deseja discretizar os valores de τ e a, é interessante que a discretização seja feita em uma grade diádica {Vetterli e Herley, 1992}, onde $a = 2^j$, $j \in \mathbf{Z}$ e $\tau \in \mathbf{Z}$. Se for possível determinar funções que possam gerar bases ortonormais apenas utilizando suas dilatações e deslocamentos, como em (4.6), então pode-se utilizá-las para obter uma representação eficiente da imagem.

4.3. Análise Multiresolução

A análise multiresolução, um *framework* matemático para análise de wavelets, foi desenvolvido por {Mallat, 1989} e implica na análise de um sinal em várias escalas simultâneas. Na análise multiresolução é necessária a existência de uma sequência de espaços V^j, $j \in \mathbf{Z}$ que satisfaçam as condições {Cohen e Ryan, 1995}:

$$V^j \subset V^{j+1} \tag{4.8a}$$

$$f(x) \in V^j \Leftrightarrow f(2x) \in V^{j+1} \tag{4.8b}$$

$$\phi(x) \in V^j \Rightarrow \{\phi(x-i)\}_{i \in Z} \text{ são bases para } V^j \tag{4.8c}$$

$$\bigcap_{j \in Z} V^j = \{0\} \tag{4.8d}$$

$$\bigcup_{j\in Z} V^j \rightarrow L^2 \tag{4.8e}$$

Estes espaços são chamados de espaços aninhados (*nested spaces*) pois todos os elementos de V^j estão contidos no espaço V^{j+1} (4.8a). As bases para estes espaços são chamadas de funções de escala (*scaling functions*) ou "wavelet-pai", e representadas por ϕ. Um espaço V^j é gerado pelas bases $\phi_i^j(x) = \phi(2^j x - i)$ (4.8c), que são as funções geradas pelos deslocamentos da função de escala em uma determinada escala j. A união de todos estes espaços tende ao espaço completo das funções L^2 (4.8e) e a sua interseção resulta no espaço vazio (4.8d).

Uma consequência destas condições é que a função de escala em uma determinada escala j pode ser descrita como uma combinação de funções de escala numa escala j+1.

$$\phi_i^j(x) = \sqrt{2}\sum_i c_i \phi_i^{j+1}(x) \tag{4.9}$$

A Equação (4.9) é chamada de Equação de Dilatação e possui algumas características interessantes. Se a função de escala for não-nula em um intervalo limitado, ou em outras palavras, tiver suporte compacto, o número de coeficientes c_i não-nulos é limitado. Isso implica que a função de escala possuirá suporte compacto em todas as escalas. Isso não aconteceria se (4.9) fosse uma equação diferencial ordinária com todas as funções na mesma escala {Strang e Nguyen, 1996}.

Satisfeitas as condições (4.8a)-(4.8e) pode-se introduzir as bases ortonormais da transformada wavelet. Seja W^j um espaço que é o complemento ortogonal entre espaço V^j e V^{j+1}, que representa a informação contida no espaço V^{j+1} e que não pode ser representada no espaço V^j.

$$W^j = V^j \perp V^{j+1} \Leftrightarrow V^{j+1} = V^j \oplus W^j$$

Isso significa que (4.8e) pode ser escrita como:

$$\bigoplus_{j\in Z} W^j \rightarrow L^2$$

As bases ortogonais que geram os espaços W^j são as wavelets, representadas por ψ. Estas wavelets, que geram os espaços através de deslocamentos e dilatações, são determinadas pela função de escala e podem ser representadas como uma combinação linear das funções de escala dilatadas e deslocadas.

$$\psi_i^j(x) = \sqrt{2}\sum_i d_i \phi_i^{j+1}(x) \tag{4.10}$$

Diferentemente das funções de escala, as wavelets em uma escala também podem ser ortogonais às wavelets em outras escalas. Isso pode ser explicado sabendo-se que W^j é ortogonal a V^j, mas está contido em V^{j+1}. Como W^{j+1} é ortogonal a V^{j+1}, então W^{j+1} é ortogonal a W^j. E se os espaços são ortogonais suas funções base também o são {Strang e Nguyen, 1996}.

$$W^j \perp V^j$$
$$W^j \subset V^{j+1}$$
$$W^{j+1} \perp V^{j+1} \Rightarrow W^j \perp W^{j+1}$$

Estes espaços ortogonais W^j definidos pela wavelets são os espaços que podem representar L^2 sem redundância. Projetando-se uma função $f(x)$ nestes espaços pode-se conseguir a representação eficiente e não-redundante pretendida inicialmente.Esta decomposição pode ser feita parcialmente, ou seja, a função $f(x)$ pode ser projetada sem redundância num espaço V^n e nos sucessivos espaços W^n , W^{n+1}.... pois:

$$V^n \perp W^j, \forall j \geq n$$
$$V^n \bigoplus_{j \geq n} W^j \rightarrow L^2$$

.O cálculo da projeção de $f(x)$ é feita da seguinte maneira. Seja $f_1(x)$ a projeção de $f(x)$ no espaço V^1. Isto é feito pelo produto escalar de $f(x)$ com as bases do espaço V^1.

$$f_1(x) = \left\langle f \middle| \phi_i^1 \right\rangle = \int f(x)\phi_i^1(x)\,\mathrm{d}x$$

Esta projeção resulta numa combinação linear das funções de escala $\phi(2x-i)$, que são as bases de V^1.

$$f_1(x) = \sum_i a_{1,i}\phi_i^1(x) = \sum_i a_{1,i}\phi(2x-i) \tag{4.11}$$

Mas $V^1 = V^0 \oplus W^0$, cujas bases são respectivamente $\phi_i^0(x) = \phi(x-i)$ e $\psi_i^0(x) = \psi(x-i)$. Portanto $f_1(x)$ pode ser escrita como uma combinação linear de $\phi_i^0(x)$ e $\psi_i^0(x)$. Seja a projeção de $f_1(x)$ em V^0:

$$\left\langle f_1 \middle| \phi_i^0 \right\rangle = \sum_i a_{0,i}\phi(x-i) = f_0(x)$$
$$\left\langle f_1 \middle| \psi_i^0 \right\rangle = \sum_i b_{0,i}\psi(x-i)$$

Calculando-se o produto escalar utilizando a combinação linear (4.11):

$$\begin{aligned}\left\langle f_1 \middle| \phi_i^0 \right\rangle &= \int f_1(x)\phi_i^0(x)\,\mathrm{d}\,x \\ &= \int \sum_k a_{1,k}\phi_k^1(x)\phi_i^0(x)\,\mathrm{d}\,x\end{aligned} \tag{4.12}$$

Mas $\phi_i^0(x)$ pode ser representada por uma combinação linear de $\phi_i^1(x)$.

$$\phi(x) = \sum_n c_n\phi(2x-n)$$
$$\phi(x-i) = \sum_n c_n\phi(2x-2i-n)$$

Reescrevendo (4.11):

$$\left\langle f_1 \middle| \phi_i^0 \right\rangle = \int \sum_k a_{1,k}\phi(2x-k)\sum_n c_n\phi(2x-2i-n)\,\mathrm{d}\,x$$

Considerando que as bases $\phi_i^j(x)$ são ortonormais, ou seja, $\left\langle \phi_i^j \middle| \phi_k^j \right\rangle = \delta_{i-k}$, a integral não é nula somente onde $2i+n=k$.

$$\left\langle f_1 \middle| \phi_i^0 \right\rangle = \sum_k a_{1,k}c_{k-2i} = a_{0,i} \tag{4.13}$$

O mesmo processo também pode ser utilizado na determinação da projeção de $f_1(x)$ em $\psi_i^0(x)$. O resultado é:

$$\left\langle f_1 \middle| \psi_i^0 \right\rangle = \sum_k a_{1,k}d_{k-2i} = b_{0,i}$$

Estas equações significam que a projeção de uma função $f(x)$ em um espaço V^j pode ser obtida a partir da projeção de $f(x)$ no espaço V^{j+1}. É simples mostrar que isto ocorre para todas as escalas. Assim:

$$\begin{aligned}a_{j,i} &= \sum_k a_{j+1,k}c_{k-2i} \\ b_{j,i} &= \sum_k a_{j+1,k}d_{k-2i}\end{aligned} \tag{4.14}$$

O cálculo das projeções, chamado de análise, pode ser representado graficamente pela Figura 4.8.

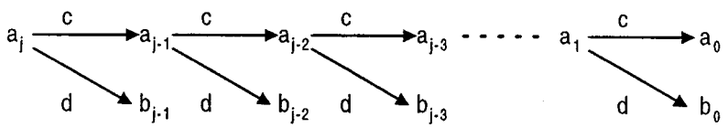


Figura 4.8: Diagrama do processo de análise

Do ponto de vista de processamento de sinais, o cálculo das projeções $a_{j,i}$ pode ser encarado como uma filtragem nas projeções anteriores $a_{j+1,i}$ feita por um filtro digital H_0 de N estágios cujos coeficientes são $h_0[i]=c[N-1-i]$, onde $i \in \{0..N-1\}$, e cuja saída é subamostrada por um fator de 2 (*downsampling*). Esta subamostragem também é chamada de decimação {Strang e Nguyen, 1996}. Um outro filtro H_1 de N estágios, cujos coeficientes são $h_1[i]=d[N-1-i]$, onde $i \in \{0..N-1\}$, calcula as projeções $b_{j,i}$. A estes filtros associados em cascata se dá o nome de bancos de filtros {Stollnitz e outros, 1995}. Este banco de filtros é chamado de banco de filtros de análise.

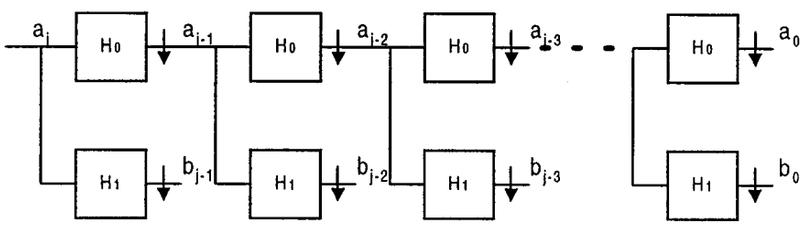


Figura 4.9 Diagrama de análise com bancos de filtros

A reconstrução da projeção de $f_j(x)$ em um espaço V^{j+1} a partir das projeções no espaço V^j e W^j podem ser calculadas da seguinte maneira {Strang e Nguyen, 1996}:

$$a_{j+1,i} = \sum_k \left(r_{2k-i} a_{j,k} + s_{2k-i} b_{j,k} \right)$$

Os coeficientes **r** e **s** podem ser determinados a partir dos coeficientes **c** e **d**. Este processo, chamado de síntese, pode ser representado graficamente como:

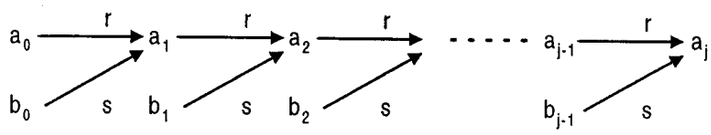


Figura 4.10: Diagrama do processo de síntese

Esta operação pode ser representada como uma sequência de filtragens por filtros G_0 e G_1, cujos coeficientes são respectivamente $g_0[i]=r[N\text{-}1\text{-}i]$ e $g_1[i]=s[N\text{-}1\text{-}i]$, onde $i \in \{0..N\text{-}1\}$, seguidas de uma operação de interpolação (*upsampling*) .

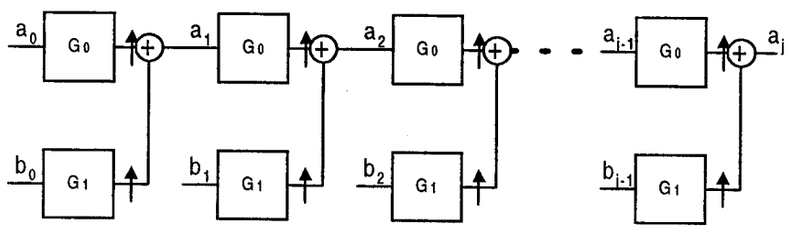


Figura 4.11 Diagrama de síntese com bancos de filtros

Devido ao fato das bases $\psi_i^j(x)$ precisarem ser ortogonais às bases $\phi_i^j(x)$, os coeficientes **d** podem ser determinados a partir de **c**, ou vice-versa. Seja N o número de coeficientes de **c**.

$$d[i] = -1^i c[N-i-1], \quad i \in \{0..N-1\} \tag{4.15}$$

Em resumo, se existem as bases $\phi_i^j(x)$ e $\psi_i^j(x)$ e elas satisfazem (4.9) e (4.10) então a partir dos coeficientes **c** e **d** de (4.9) e (4.10) é possível decompor o sinal nos vários espaços W^j. Esta decomposição é um processo recursivo que envolve uma convolução com os filtro H_0 e H_1 associada com subamostragens para cada etapa de decomposição. Devido ao fato das bases terem um suporte compacto, o número de coeficientes de **c** e **d** não nulos é limitado, o que torna a complexidade do algoritmo de análise da ordem O(N). Isto também vale para o algoritmo de síntese, utilizando um conjunto de filtros diferente e sobreamostragens. Considerando a transformada do ponto de vista matricial, como em (2.3), isto significa que as matrizes transformada direta e inversa são esparsas. Isso pode ser visto abaixo, na representação matricial de (4.14).

$$\begin{bmatrix} \vdots \\ a_{j-1,i} \\ a_{j-1,i+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & & & & & & & \\ & c_0 & c_1 & \cdots & c_{N-1} & & & \\ & & & c_0 & c_1 & \cdots & c_{N-1} & \\ & & & & & & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ a_{j,i} \\ a_{j,i+1} \\ \vdots \end{bmatrix}$$

$$\begin{bmatrix} \vdots \\ b_{j-1,i} \\ b_{j-1,i+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & & & & & & & \\ & d_0 & d_1 & \cdots & d_{N-1} & & & \\ & & & d_0 & d_1 & \cdots & d_{N-1} & \\ & & & & & & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ a_{j,i} \\ a_{j,i+1} \\ \vdots \end{bmatrix}$$

Foge do escopo deste trabalho resolver a equação de diferença para determinar as funções de escala (4.9), wavelets (4.10) e os coeficientes. Uma maneira sistemática de construir wavelets que satisfaçam a estas condições pode ser encontrada em {Strang e Nguyen, 1996} e {Mallat, 1989}.

4.4. Wavelets Ortonormais

Wavelets ortonormais são aquelas que geram bases ortonormais. O desenvolvimento anterior foi feito considerando que as bases $\psi_i^j(x)$ são ortonormais, o que permitiu simplificações em alguns passos do desenvolvimento, como em (4.13).

Daubechies descobriu uma família de wavelets ortogonais com suporte compacto, ou seja, com número reduzido de coeficientes **c** e **d**. Esta família é parametrizada por um número inteiro que representa o número de coeficientes **c** e **d**, e consequentemente o tamanho dos filtros de síntese e análise. Cada wavelet desta família é conhecido por Dn , onde n indica é o número de coeficientes. Este número indica o grau de suavidade da wavelet, e é um fator importante para o seu desempenho na compressão de imagens. Quanto maior o valor de n mais suave são as wavelets. Wavelets de ordem menor são irregulares (Figura 4.12) enquanto nas de ordem maior as irregularidades aparecem apenas em derivadas de ordem superior (Figura 4.13). Por outro lado, nesta família de wavelets, filtros de comprimento maior que 8 ou 10 não apresentam vantagens significativas {Lu e outros, 1996} em aplicações de compressão de imagens.

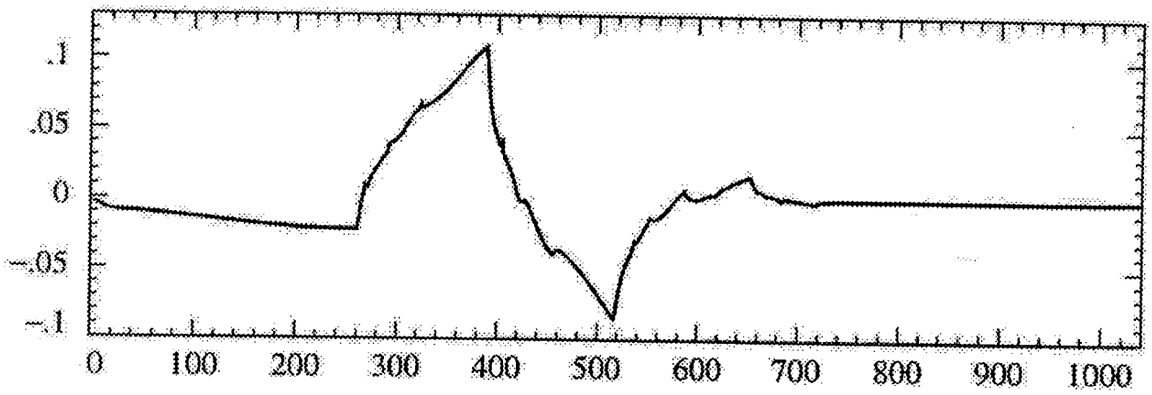


Figura 4.12: Função wavelet de Daubechies-4

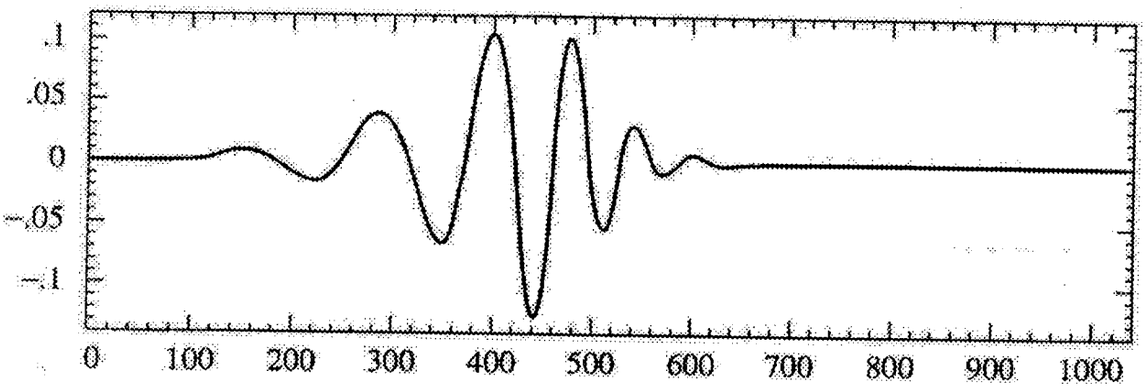


Figura 4.13: Função wavelet de Daubechies-20

Dados os coeficientes da equação de **c** e **d**, os coeficientes dos filtros de síntese e análise são:

$$h_0[i] = c[N-i-1]$$
$$h_1[i] = d[N-i-1]$$
$$g_0[i] = r[N-i-1]$$
$$g_1[i] = s[N-i-1]$$
$$g_0[i] = -1^i h_1[i]$$
$$g_1[i] = -1^i(-h_0[i])$$

Devido a ortogonalidade:

$$g_0[i] = c[i]$$

Daí pode-se concluir que nas wavelets ortonormais os coeficientes basta se conhecer um dos conjuntos de coeficientes **c** ou **d** para se determinar os coeficientes de todos os filtros

envolvidos.

$$h_1[i] = -1^i c[i]$$
$$= -1^i h_0[N-i-1]$$

Desenvolvendo as equações anteriores, os filtros H_0, H_1, G_0 e G_1 para wavelets ortonormais podem ser determinados a partir dos coeficientes **c**, havendo apenas a variação de posição e sinal dos coeficientes:

$$d[i] = -1^i(-c[N-i-1])$$
$$r[i] = c[N-i-1]$$
$$s[i] = -1^i c[i]$$

A Tabela 4.1 mostra os coeficientes $\mathbf{h_0}$ de algumas wavelets ortonormais.

Tabela 4.1: Coeficientes $\mathbf{h_0}$ de algumas wavelets ortonormais

	Haar	D4	D6
i	$h_0[i]$	$h_0[i]$	$h_0[i]$
0	0.70710678118655	-0.12940952255126	0.33267055295008
1	0.70710678118655	0.22414386804201	0.80689150931109
2		0.83651730373781	0.45987750211849
3		0.48296291314453	-0.13501102001025
4			-0.08544127388203
5			0.03522629188571

A maior desvantagem de wavelets ortonormais com suporte compacto é a sua assimetria. Isto significa que os filtros que implementam a wavelet não possuirão fase linear {Oppenheim e Schafer, 1975}, o que causa distorções nas bordas da imagem processada com estas wavelets {Lu e outros, 1996}. A única wavelet ortonormal simétrica é a wavelet Haar, que devido ao seu tamanho limitado, possui baixo desempenho na compressão {Villasenor e outros, 1995}.

4.5. Wavelets Biortogonais

Wavelets biortogonais permitem tanto suporte compacto quanto a simetria. A diferença entre estas wavelets e as ortonormais é que a condição de ortogonalidade entre os espaços W^j e os espaços V^j foi relaxada e (4.15) não é necessária. Isso oferece um maior grau de liberdade na escolha das wavelets, permitindo filtros de fase linear com coeficientes binários {Rassau e outros, 1999} {Strang e Nguyen, 1996}, que são coeficientes formados por números inteiros divididos

por uma potência de 2. Isso é muito interessante para a implementação em hardware. Wavelets biortogonais também diferem das ortonormais por permitirem filtros H_0 e G_1 de tamanhos diferentes dos tamanhos dos seus pares H_1 e G_0.

Uma vez determinados os coeficientes $c[i]$ e $d[i]$, conforme descrito em {Strang e Nguyen, 1996}, os coeficientes dos filtros de análise (H_0 e H_1) e síntese (G_0 e G_1) são:

$$h_0[i] = c[N_0 - i - 1] = c[i], \quad i \in \{0..N_0 - 1\}$$
$$h_1[i] = d[N_1 - i - 1] = d[i], \quad i \in \{0..N_1 - 1\}$$
$$g_0[i] = -1^i h_1[i], \quad i \in \{0..N_1 - 1\}$$
$$g_1[i] = -1^i(-h_0[i]), \quad i \in \{0..N_0 - 1\}$$

Algumas wavelets biortogonais interessantes do ponto de vista de simplicidade de implementação {Rassau e outros, 1999} {Villasenor e outros, 1995} são apresentados na Tabela 4.2. Estas wavelets são conhecidas pelo tamanho de seus filtros H_0 e G_0.

Tabela 4.2: Coeficientes $\mathbf{h_0}$ e $\mathbf{g_0}$ de algumas wavelets biortogonais

	9/3		5/3		2/6		Triangular	
i	$h_0[i]$*64	$g_0[i]$*16	$h_0[i]$*8	$g_0[i]$*2	$h_0[i]$*2	$g_0[i]$*8	$h_0[i]$	$g_0[i]$
0	1	-1	-1	1	1	-1	0	1/2
1	0	0	2	2	1	1	1	1
2	-8	9	6	1		8	0	1/2
3	16	16	2			8		
4	46	9	-1			1		
5	16	0				-1		
6	-8	-1						
7	0							
8	1							

4.6. Considerações Finais

Além de serem mais eficientes na representação de imagens que a DCT, a transformada wavelet é de complexidade de ordem O(N). Esta complexidade reduzida é resultado da matriz de transformação ser uma matriz esparsa. Além disso a característica da decomposição da imagem em escalas sucessivas de menor resolução permite uma escalabilidade que não é oferecida pela DCT.

Transformadas wavelet ortonormais possuem filtros com os mesmos coeficientes, mudando apenas a sua posição e sinais. Esta característica pode ser explorada na implementação

em hardware, pois os mesmos multiplicadores por constante podem ser utilizados na implementação dos filtros H_0, H_1, G_0 e G_1.

Transformadas wavelet biortogonais permitem que se utilizem apenas coeficientes inteiros nestas matrizes de transformação, o que também é conveniente para a implementação em hardware.

5. Análise dos Erros de Quantização em Bancos de Filtros Multiescala

5.1. Introdução

Em um sistema digital grandezas contínuas são representadas com um número finito de bits, o que causa erros de representação chamados de erros de quantização. A medida que operações matemáticas são efetuadas sobre estas representações aproximadas, estes erros se propagam e acumulam, podendo vir a ser significativos {Koren, 1993}. Quando o sistema de representação binária adotado é o de ponto flutuante, estes erros normalmente são de pequena magnitude e não necessitam de análise detalhada na implementação de um algoritmo. Mas no caso de um sistema de representação binária de ponto fixo, erros de quantização são muito significativos e demandam análise precisa.

O algoritmo para a DWT é uma sequência de multiplicações e somas (MAC), que podem ser implementadas por meio de filtros FIR. A análise da propagação dos erros de quantização em filtros FIR utilizando representação de ponto fixo foi tema de muitas pesquisas, mas tem sido deixada em segundo plano devido ao aumento do tamanho de palavra dos processadores modernos, e da popularização de processadores com coprocessadores de ponto flutuante embutidos. Mas o advento das FPGAs de alta densidade, capazes de implementar subsistemas completos em um único *chip*, trouxe este tema de volta a atualidade.

Diferentemente de processadores clássicos, cujo tamanho de palavra é predeterminado, FPGAs podem implementar arquiteturas com o tamanho de palavra que for necessário à aplicação {Kostarnov e outros, 1997}. Naturalmente processadores clássicos podem simular palavras maiores com ajuda do software, mas com prejuízo de performance {Hesener, 1996}. Para esta flexibilidade ser aproveitada é necessário um estudo cuidadoso dos erros de quantização introduzidos e seu impacto no resultado final do processamento. Qualquer bit desnecessário no tamanho da palavra escolhido implica em desperdício de recursos e, possivelmente, menor velocidade de operação. Já um tamanho de palavra menor que o necessário implica em problemas de precisão no cálculo e distorções no resultado.

Existem 3 fatores que determinam a magnitude do erro no processamento da transformada wavelet em aritmética de ponto fixo:

- Tamanho da palavra de representação dos coeficientes dos filtros
- Tamanho da palavra de representação dos dados a serem processados
- Número de escalas

A motivação deste capítulo é desenvolver uma técnica para estimar a influência destes fatores no erro do processamento da transformada wavelet em aritmética de ponto fixo, de modo a auxiliar a determiná-los.

Como visto anteriormente, a DWT pode ser eficientemente implementada por meio de filtros FIR e operações de decimação (subamostragem) e interpolação (sobreamostragem), conforme mostra a Figura 5.1, que representa a transformada wavelet com N escalas.

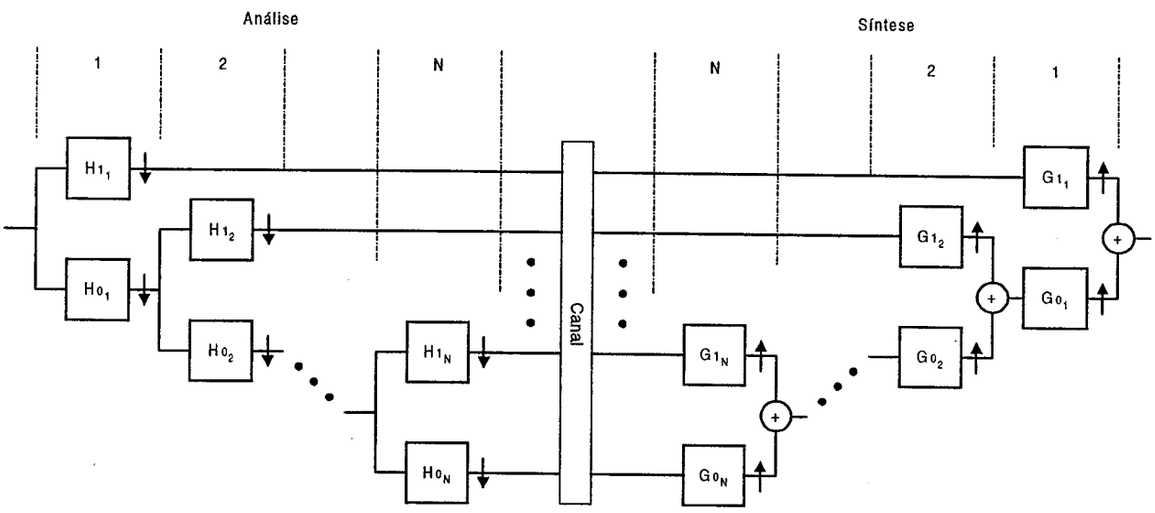


Figura 5.1: Esquema de um bancos de filtros para DWT

Um sinal com erro de quantização pode ser representado como:

$$\tilde{y}(t) = y(t) + \Delta y(t) \tag{5.1}$$

A relação sinal-ruído (SNR) mede o erro introduzido no sinal:

$$SNR = \frac{\int_0^T (y(t) - \tilde{y}(t))^2 dt}{\int_0^T y(t)^2 dt} = \frac{\int_0^T \Delta y(t)^2 dt}{\int_0^T y(t)^2 dt}$$

Se estes sinais forem considerados sequências aleatórias, a relação sinal ruído pode ser expressa como:

$$SNR = \frac{E[\Delta \mathbf{Y}^2]}{E[\mathbf{Y}^2]}$$

Um parâmetro muito utilizado para medida da qualidade perceptual de uma imagem após seu processamento é a relação sinal ruído de pico (PSNR) {Strang e Nguyen, 1996}.

$$PSNR = 10\log_{10}\left(\frac{y_{\max}^{\ 2}}{\frac{1}{N_l N_c}\sum_{l}^{N_l}\sum_{c}^{N_c}\left(y(l,c)-\tilde{y}(l,c)\right)^2}\right) \tag{5.2}$$

5.1.1. Revisão Estatística

Esta análise de erros de quantização em sistema de DWT é feita através de um modelo estatístico. Se feita de maneira determinística, considerando sempre os piores casos na propagação do erro, o erro estimado é pouco realista, muito maior do que se observa na prática.

A análise proposta não visa determinar a função densidade de probabilidade do erro introduzido pela quantização, mas apenas determinar a magnitude deste erro. Isso já é suficiente para se estimar parâmetros importantes do sistema sob análise, como a relação sinal-ruído na saída e a relação sinal-ruído de pico (PSNR), muito utilizada para avaliação da qualidade da codificação de imagens.

Seja **X** uma variável aleatória (v.a.) e $f_{\mathbf{X}}(x)$ a sua função densidade de probabilidade (f.d.p.). O momento ordem n desta v.a. é:

$$E[\mathbf{X}^n] = \int_{-\infty}^{\infty} x^n f_{\mathbf{X}}(x)\,dx$$

Sob o ponto de vista da área de processamento de sinais, o momento de primeira ordem é a média do sinal e o de segunda ordem é o seu valor RMS.

Sejam **X** e **Y** v.a. reais e independentes entre si, portanto, não correlacionadas.

$$\begin{aligned} &E[\mathbf{X}+\mathbf{Y}] = E[\mathbf{X}] + E[\mathbf{Y}] \\ &E[\mathbf{XY}] = E[\mathbf{X}]\,E[\mathbf{Y}] \\ &E[a\mathbf{X}] = aE[\mathbf{X}] \end{aligned} \tag{5.3}$$

Sejam $\mathbf{X}_i$ e $\mathbf{Y}_i$ sequências aleatórias reais e independentes. No modelo estatístico

proposto elas representam os sinais de entrada e saída do sistema sob análise.

$$\begin{aligned} &E[\mathbf{X}_i + \mathbf{Y}_i] = E[\mathbf{X}_i] + E[\mathbf{Y}_i] \\ &E[\mathbf{X}_i \mathbf{Y}_i] = E[\mathbf{X}_i]\, E[\mathbf{Y}_i] \\ &E[a\mathbf{X}_i] = aE[\mathbf{X}_i] \end{aligned} \tag{5.4}$$

Seja $\mathbf{X}_i$ uma sequência aleatória independente nas mesmas condições anteriores, formando um processo estocástico, que seja estacionária na média e na média dos quadrados {Stark e Woods, 1994}. Essa consideração é muito frequente na análise e modelamento de sinais aleatórios {Porat, 1997} e permite uma série de simplificações no modelamento. Todas as sequências são consideradas estacionárias nesta análise.

$$\begin{aligned} &E[\mathbf{X}_i] = E[\mathbf{X}], \forall i \\ &E[\mathbf{X}_i^{\,2}] = E[\mathbf{X}^2], \forall i \end{aligned} \tag{5.5}$$

Sejam $\mathbf{X}_i$ e $\mathbf{Y}_i$ sequências aleatórias nas mesmas condições anteriores, mas cuja média seja nula:

$$\begin{aligned} &E\left[\sum_i a_i \mathbf{X}_i\right] = \sum_i a_i E[\mathbf{X}_i] = \sum_i a_i E[\mathbf{X}] \\ &E\left[\left(\sum_i a_i \mathbf{X}_i\right)^2\right] = \sum_i a_i^{\,2} E[\mathbf{X}_i^{\,2}] = \|a\|_2^2 E[\mathbf{X}^2] \\ &E\left[\left(\sum_i^N \mathbf{X}_i \mathbf{Y}_i\right)^2\right] = \sum_i E[\mathbf{X}_i^{\,2}]\, E[\mathbf{Y}_i^{\,2}] = N\, E[\mathbf{X}^2]\, E[\mathbf{Y}^2] \\ &E\left[\left(\sum_i^N (\mathbf{X}_i + \mathbf{Y}_i)\right)^2\right] = \sum_i (E[\mathbf{X}_i^{\,2}] + E[\mathbf{Y}_i^{\,2}]) = N E[\mathbf{X}^2] + N E[\mathbf{Y}^2] \end{aligned} \tag{5.6}$$

Estas igualdades podem ser deduzidas utilizando os mesmos princípios utilizados neste desenvolvimento:

$$\begin{aligned} E\left[\left(\sum_i^N \mathbf{X}_i\right)^2\right] &= E\left[\sum_i^N \mathbf{X}_i \sum_j^N \mathbf{X}_j\right] = E\left[\sum_i^N \mathbf{X}_i^{\,2} + \sum_{i \neq j}^N \mathbf{X}_i \sum_{j \neq i}^N \mathbf{X}_j\right] \\ &= E\left[\sum_i^N \mathbf{X}_i^{\,2}\right] + E\left[\sum_{i \neq j}^N \mathbf{X}_i \sum_{j \neq i}^N \mathbf{X}_j\right] \\ &= \sum_i^N E[\mathbf{X}_i^{\,2}] + \sum_{i \neq j}^N \sum_{j \neq i}^N E[\mathbf{X}_i \mathbf{X}_j] \end{aligned}$$

Se as v.a. $\mathbf{X}_i$ forem independentes:

$$E\left[\left(\sum_{i}^{N}\mathbf{X}_i\right)^2\right]=\sum_{i}^{N}E[\mathbf{X}_i^{\,2}]+\sum_{i\neq j}^{N}\sum_{j\neq i}^{N}E[\mathbf{X}_i]E[\mathbf{X}_j]$$

Se as v.a. $\mathbf{X}_i$ forem estacionárias:

$$E\left[\left(\sum_{i}^{N}\mathbf{X}_i\right)^2\right]=NE[\mathbf{X}^2]+(N^2-N)E[\mathbf{X}]^2$$

Se as v.a. $\mathbf{X}_i$ também tiverem média nula:

$$E\left[\left(\sum_{i}^{N}\mathbf{X}_i\right)^2\right]=NE[\mathbf{X}^2]$$

5.2. Erro em Estágio Único

Cada bloco H_0, H_1, G_0 e G_1 é um filtro FIR, representado na Figura 5.2. Um filtro FIR é composto pelo cascateamento de estágios chamados de *taps* que atrasam o sinal e executam uma multiplicação e uma soma.

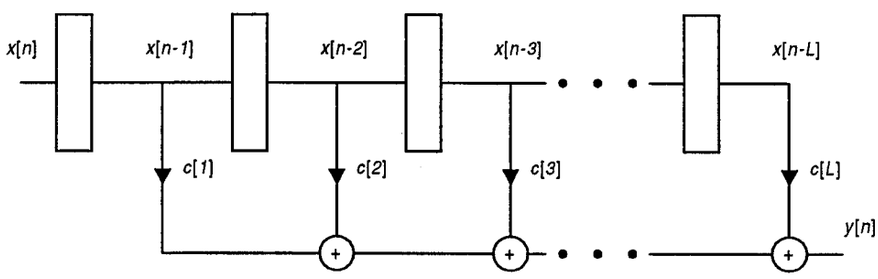


Figura 5.2: Estrutura de um filtro FIR

A saída de um filtro FIR com L taps é:

$$y[n]=\sum_{i=1}^{L}x[n-i]\,c[i] \tag{5.7}$$

5.2.1. Quantização de Coeficientes

O coeficiente c_i quantizado com número de bits limitado pode ser representado conforme (5.1):

$$\tilde{c}[i] = c[i] + \Delta c[i]$$

Segundo (5.7) a saída com erro devido à quantização dos coeficientes é:

$$\begin{aligned}\tilde{y}_c[n] &= \sum_{i=1}^{L} x[n-i]\,(c[i] + \Delta c[i]) \\ &= \sum_{i=1}^{L} x[n-i]\,c[i] + \sum_{i=1}^{L} x[n-i]\,\Delta c[i]\end{aligned}$$

O que implica numa relação sinal-ruído de:

$$SNR = \frac{\left(\sum_{i=1}^{L} x[n-i]\,\Delta c[i]\right)^2}{\left(\sum_{i=1}^{L} x[n-i]\,c[i]\right)^2}$$

Considerando as variáveis $x[i]$ como amostras de uma sequência aleatória, a relação (5.6) pode ser escrita como:

$$\frac{E(\Delta \mathbf{Y}^2)}{E(\mathbf{Y}^2)} = \frac{E\left[\left(\sum_{i=1}^{L} \Delta c_i\,\mathbf{X}_i\right)^2\right]}{E\left[\left(\sum_{i=1}^{L} c_i\,\mathbf{X}_i\right)^2\right]} = \frac{\|\Delta c\|_2^2\,E[\mathbf{X}^2]}{\|c\|_2^2\,E[\mathbf{X}^2]} = \frac{\|\Delta c\|_2^2}{\|c\|_2^2} \tag{5.8}$$

Pode-se considerar sem perda de precisão cada um dos componentes de erro $\Delta c[i]$ como um sinal de ruído branco independente {Oppenheim e Schafer, 1975}, ou seja, uma v.a. $\Delta\mathbf{C}$, com f.d.p. uniforme e média nula. Os coeficientes $c[i]$ também serão considerados v.a. $\mathbf{C}$, nas mesmas condições de $\Delta\mathbf{C}$. Assim (5.8) torna-se:

$$\frac{E(\Delta \mathbf{Y}^2)}{E(\mathbf{Y}^2)} = L\frac{E[\Delta \mathbf{C}^2]}{E[\mathbf{C}^2]} \tag{5.9}$$

5.2.2. Quantização de Barramento

Algumas vezes a saída de um filtro deve ter sua amplitude reduzida (*scaled*) {Porat, 1997} de modo que este resultado possa ser representado com o número de bits disponível no barramento do sistema digital. Essa operação pode ser representada pela divisão da saída a ser

reduzida por uma constante de ajuste de escala K. Na representação em ponto fixo, esta redução implica na utilização de menos bits para representar a saída, gerando um erro de quantização chamado de erro de quantização de barramento.

O erro de quantização de barramento, representado na Figura 5.3 pelo sinal $\Delta b[n]$ também pode ser considerado sem perda de precisão um sinal do tipo ruído branco {Oppenheim e Schafer, 1975}, ou seja, uma v.a. $\Delta\mathbf{B}$ de uniformemente distribuída e de média nula.

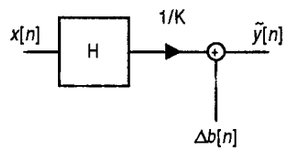


Figura 5.3: Modelo estatístico da quantização de barramento

O sinal de saída se considerada a quantização de barramento passa a ser:

$$\tilde{y}[n] = y[n] + \Delta y[n] \tag{5.10}$$

O erro devido à quantização de barramento é simplesmente:

$$\frac{E(\Delta\mathbf{Y}^2)}{E(\mathbf{Y}^2)} = \frac{E[\Delta\mathbf{B}^2]}{E[\mathbf{C}^2]} \tag{5.11}$$

5.3. Erro de Estágios Múltiplos em Cascata

Como visto anteriormente, a DWT pode ser implementada com filtragens sucessivas através de um conjunto de filtros ligados em cascata. A Figura 5.4 ilustra N filtros FIR ligados em cascata, cada um com coeficientes e constante de escala próprios. Para maior generalidade do modelo, os erros de quantização de barramento também estão representados.

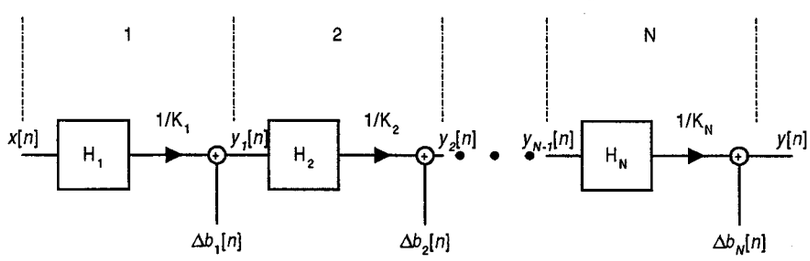


Figura 5.4: Filtro composto por N filtros em cascata

Considerando-se tanto os erros de quantização de coeficiente e barramento, a saída $y[n]$

pode ser representada da seguinte maneira:

$$\tilde{y}_N[n] = \frac{1}{K_N}\left(\sum_{i=1}^{L} \tilde{c}[i]\,\tilde{y}_{N-1}[n-i]\right) + \Delta b_N[n]$$

5.3.1. Quantização de Coeficientes

Para se considerar apenas os erros introduzidos pela quantização de coeficientes e se ignorar os erros de quantização de barramento basta tornar K_i e Δb_i respectivamente 1 e 0. Neste caso a saída do estágio j do conjunto de filtros passa a ser:

$$\tilde{y}_j[n] = \sum_{i=1}^{L} \tilde{c}_j[i]\,\tilde{x}[n-i] = \sum_{i=1}^{L} c_j[i]\,\tilde{x}[n-i] + \sum_{i=1}^{L} \Delta c_j[i]\,\tilde{x}[n-i] \qquad (5.12)$$

Como os filtros estão em cascata, a saída de um estágio de filtro é a entrada do próximo estágio.

$$\begin{aligned} y_j[n] &= x_{j+1}[n] \\ y_0[n] &= x[n] \\ y_j[n] &= \sum_{i=1}^{L} c_j[i]\,y_{j-1}[n-i] \end{aligned} \qquad (5.13)$$

Utilizando (5.13) em (5.12):

$$\begin{aligned} \tilde{y}_j[n] = &\sum_{i=1}^{L_j} c_j[i]\,y_{j-1}[n-i] + \sum_{i=1}^{L_j} c_j[i]\,\Delta y_{j-1}[n-i] + \sum_{i=1}^{L_j} \Delta c_{j1}[i]\,y_{j-1}[n-i] + \\ &+ \sum_{i=1}^{L_j} \Delta c_i[i]\,\Delta y_{j-1}[n-i] \end{aligned} \qquad (5.14)$$

Desprezando o último termo de (5.14) por ser muito menor que os outros, o erro no estágio j é:

$$\Delta y_j[n] = \sum_{i=1}^{L} c_j[i]\,\Delta y_{j-1}[n-i] + \sum_{i=1}^{L} \Delta c_j[i]\,y_{j-1}[n-i]$$

Os erros $\Delta c_j[i]$ e $c_j[i]$ podem ser considerados variáveis aleatórias $\Delta \mathbf{C}_{j,i}$ e $\mathbf{C}_{j,i}$, não correlacionadas, com média nula e distribuição uniforme.

$$\Delta \mathbf{Y}_j = \sum_i^{L_j} \mathbf{C}_{j,i} \Delta \mathbf{Y}_{j-1} + \sum_i^{L_j} \Delta \mathbf{C}_{j,i} \mathbf{Y}_{j-1}$$
$$\mathbf{Y}_j = \sum_i^{L_j} \mathbf{C}_{j,i} \mathbf{Y}_{j-1} \tag{5.15}$$

Calculando a potência do erro de quantização de coeficientes:

$$E[\Delta \mathbf{Y}_j{}^2] = E\left[\left(\sum_i^{L_j} \mathbf{C}_{j,i} \Delta \mathbf{Y}_{j-1} + \sum_i^{L_j} \Delta \mathbf{C}_{j,i} \mathbf{Y}_{j-1}\right)^2\right]$$
$$= E\left[\left(\sum_i^{L_j} \mathbf{C}_{j,i} \Delta \mathbf{Y}_{j-1}\right)^2\right] + E\left[\left(\sum_i^{L_j} \Delta \mathbf{C}_{j,i} \mathbf{Y}_{j-1}\right)^2\right]$$

Considerando $E[\Delta \mathbf{C}_{j,i}] = E[\Delta \mathbf{C}_j]$ e $E[\mathbf{C}_{j,i}] = E[\mathbf{C}_j]$, $\forall\, i \in \{1..L\}$, $\forall\, j \in \{1..N\}$:

$$E[\Delta \mathbf{Y}_j{}^2] = L_j E[\mathbf{C}_j{}^2] E[\Delta \mathbf{Y}_{j-1}{}^2] + L_j E[\Delta \mathbf{C}_j{}^2] E[\mathbf{Y}_{j-1}{}^2] \tag{5.16}$$

Aplicando a equação anterior para todos os estágios de filtragem da Figura 5.1, a potência do erro na saída do estágio N pode ser então calculada:

$$E[\Delta \mathbf{Y}_N{}^2] = \prod_{j=1}^{N} L_j{}^2 E[\mathbf{C}_j{}^2] \left(\sum_{i=1}^{N} \frac{E[\Delta \mathbf{C}_i{}^2]}{E[\mathbf{C}_i{}^2]} E[\mathbf{X}^2] + E[\Delta \mathbf{X}^2] \right) \tag{5.17}$$

A equação anterior mostra que a potência do erro de quantização na saída após os N estágios de filtragem depende dos coeficientes de cada estágio ($E[\Delta \mathbf{C}_i^2]$ e $E[\mathbf{C}_i^2]$) e das características estatísticas do sinal de entrada ($E[\Delta \mathbf{X}^2]$ e $E[\mathbf{X}^2]$). O termo $E[\Delta \mathbf{X}^2]$ se refere ao erro de quantização do sinal de entrada.

O erro relativo pode então ser calculado:

$$\frac{E[\Delta \mathbf{Y}_N{}^2]}{E[\mathbf{Y}_N{}^2]} = \frac{\prod_{j=1}^{N} L_j{}^2 E[\mathbf{C}_j{}^2] \left(\sum_{i=1}^{N} \frac{E[\Delta \mathbf{C}_j{}^2]}{E[\mathbf{C}_j{}^2]} E[\mathbf{X}^2] + E[\Delta \mathbf{X}^2] \right)}{\prod_{j=1}^{N} L_j{}^2 E[\mathbf{C}_j{}^2] E[\mathbf{X}^2]}$$

Simplificando:

$$\frac{E[\Delta \mathbf{Y}_N{}^2]}{E[\mathbf{Y}_N{}^2]} = \sum_{i=1}^{N} \frac{E[\Delta \mathbf{C}_j{}^2]}{E[\mathbf{C}_j{}^2]} + \frac{E[\Delta \mathbf{X}^2]}{E[\mathbf{X}^2]} \tag{5.18}$$

Esta equação indica que o erro relativo pode ser estimado através da soma dos erros relativos de quantização dos coeficientes de cada estágio de filtragem. O termo $E[\Delta \mathbf{X}^2]/E[\mathbf{X}^2]$ de (5.18) representa o erro relativo do sinal de entrada, que pode ser introduzido por erros de quantização gerados em um processamento anterior. Desconsiderando o erro do sinal de entrada, a equação simplifica-se em:

$$\frac{E[\Delta \mathbf{Y}_N{}^2]}{E[\mathbf{Y}_N{}^2]} = \sum_{i=1}^{N} \frac{E[\Delta \mathbf{C}_j{}^2]}{E[\mathbf{C}_j{}^2]}$$

5.3.2. Quantização de Barramento

O erro de quantização de barramento pode ser modelado segundo o diagrama da Figura 5.4. Repetindo o equacionamento anterior, mas considerando também os erros de quantização de barramento além dos erros de quantização de coeficientes, o sinal na saída do estágio j pode ser expresso como:

$$\begin{aligned}\tilde{y}_j[n] &= \frac{1}{K_j}\sum_{i=1}^{L} \tilde{c}_j[i]\tilde{x}_j[n-i] + \Delta b_j[n] \\ &= \frac{1}{K_j}\sum_{i=1}^{L} c_j[i]\tilde{x}_j[n-i] + \frac{1}{K_j}\sum_{i=1}^{L} \Delta c_j[i]\tilde{x}_j[n-i] + \Delta b_j[n]\end{aligned} \tag{5.19}$$

Da mesma forma que (5.13):

$$\begin{aligned}y_j[n] &= x_{j+1}[n] \\ y_0[n] &= x[n] \\ y_j[n] &= \frac{1}{K_j}\sum_{i=1}^{L} c_j[i]\, y_{j-1}[n-i]\end{aligned} \tag{5.20}$$

Desenvolvendo (5.19) de modo semelhante a (5.14):

$$\begin{aligned}\tilde{y}_j[n] &= \frac{1}{K_j}\sum_{i=1}^{L_j} \tilde{c}_j[i]\,\tilde{y}_{j-1}[n-i] + \Delta b_j[n] \\ &= \frac{1}{K_j}\sum_{i=1}^{L_j} c_j[i]\, y_{j-1}[n-i] + \frac{1}{K_j}\sum_{i=1}^{L_j} c_j[i]\Delta y_{j-1}[n-i] \\ &+ \frac{1}{K_j}\sum_{i=1}^{L_j} \Delta c_{j1}[i]\, y_{j-1}[n-i] + \frac{1}{K_j}\sum_{i=1}^{L_j} \Delta c_i[i]\Delta y_{j-1}[n-i] \\ &+ \Delta b_j[n]\end{aligned} \tag{5.21}$$

Desprezando o penúltimo termo de (5.21) por ser muito menor que os outros, o erro do sinal na saída do estágio j é:

$$\Delta y_j[n] = \frac{1}{K_j}\sum_{i=1}^{L} c_j[i]\Delta y_{j-1}[n] + \frac{1}{K_j}\sum_{i=1}^{L} \Delta c_j[i] y_{j-1}[n] + \Delta b_j[n] \tag{5.22}$$

Considerando os erros Δc_j e c_j como variáveis aleatórias $\Delta\mathbf{C}_j$ e $\mathbf{C}_j$, não correlacionadas, com média nula e distribuição uniforme, a Equação (5.22) pode ser escrita como:

$$\Delta\mathbf{Y}_j = \frac{1}{K_j}\sum_{i}^{L_j} \mathbf{C}_{j,i}\Delta\mathbf{Y}_{j-1} + \frac{1}{K_j}\sum_{i}^{L_j} \Delta\mathbf{C}_{j,i}\mathbf{Y}_{j-1} + \Delta\mathbf{B}_j \tag{5.23}$$

$$\mathbf{Y}_j = \frac{1}{K_j}\sum_{i}^{L_j} \mathbf{C}_{j,i}\mathbf{Y}_{j-1} \tag{5.24}$$

Considerando $\Delta\mathbf{C}_{j,i} = \Delta\mathbf{C}_j$ e $\mathbf{C}_{j,i} = \mathbf{C}_j$, $\forall\, i \in \{1..L\}$, $\forall\, j \in \{1..N\}$, a potência dos erros de quantização de coeficientes e barramento na saída do filtro j é:

$$\begin{aligned} E[\Delta\mathbf{Y}_j{}^2] &= E[(K_j{}^{-1}\sum_{i}^{L_j} \mathbf{C}_j\Delta\mathbf{Y}_{j-1} + K_j{}^{-1}\sum_{i}^{L_j} \Delta\mathbf{C}_j\mathbf{Y}_{j-1} + \Delta\mathbf{B}_j)^2] \\ &= K_j{}^{-2}E\left[\left(\sum_{i}^{L_j} \mathbf{C}_j\Delta\mathbf{Y}_{j-1}\right)^2\right] + K_j{}^{-2}E\left[\left(\sum_{i}^{L_j} \Delta\mathbf{C}_j\mathbf{Y}_{j-1}\right)^2\right] + E[\Delta\mathbf{B}_j{}^2] \\ &= \frac{L_j}{K_j{}^2}\left(E[\mathbf{C}_j{}^2]E[\Delta\mathbf{Y}_{j-1}{}^2] + E[\Delta\mathbf{C}_j{}^2]E[\mathbf{Y}_{j-1}{}^2]\right) + E[\Delta\mathbf{B}_j{}^2] \end{aligned} \tag{5.25}$$

Desenvolvendo a Equação (5.25) obtém-se o erro total na saída do último estágio do filtro:

$$E[\Delta\mathbf{Y}_N{}^2] = \frac{L_N}{K_N^2}\left\{E[\mathbf{C}_N{}^2]E[\Delta\mathbf{Y}_{N-1}{}^2] + E[\Delta\mathbf{C}_N{}^2]E[\mathbf{Y}_{N-1}{}^2]\right\} + E[\Delta\mathbf{B}_N{}^2]$$

Utilizando-se novamente (5.25) para tornar mais claro o processo de recursão:

$$E[\Delta\mathbf{Y}_N{}^2] = \frac{L_N}{K_N^2}\left\{E[\mathbf{C}_N^2]\left(\frac{L_{N-1}}{K_{N-1}^2}\left\{E[\mathbf{C}_{N-1}^2]E[\Delta\mathbf{Y}_{N-2}^2] + E[\Delta\mathbf{C}_{N-1}^2]E[\mathbf{Y}_{N-2}^2]\right\} + E[\Delta\mathbf{B}_{N-1}^2]\right) + E[\Delta\mathbf{C}_N^2]E[\mathbf{Y}_{N-1}^2]\right\} + E[\Delta\mathbf{B}_N^2]$$

Desenvolvendo a equação anterior aplicando-se recursivamente (5.25), obtém-se o erro total na saída do último estágio do filtro:

$$E[\Delta \mathbf{Y}_N{}^2] = \prod_{j=1}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)E[\Delta \mathbf{Y}_0{}^2] +$$
$$+\sum_{i=1}^{N}\left(\prod_{j=i}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)\frac{E[\Delta \mathbf{C}_i{}^2]}{E[\mathbf{C}_i{}^2]}E[\mathbf{Y}_{i-1}{}^2]\right) +$$
$$+\sum_{i=2}^{N}\left(\prod_{j=i}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)E[\Delta \mathbf{B}_{i-1}{}^2]\right) + E[\Delta \mathbf{B}_N{}^2]$$

Como:

$$E[\mathbf{Y}_j{}^2] = \prod_{i=1}^{j}\left(\frac{L_i}{K_i{}^2}E[\mathbf{C}_i{}^2]\right)E[\mathbf{X}^2]$$
$$E[\Delta \mathbf{Y}_0{}^2] = E[\Delta \mathbf{X}^2]$$

Portanto:

$$E[\Delta \mathbf{Y}_N{}^2] = \prod_{j=1}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)\left(E[\Delta \mathbf{X}^2] + \sum_{i=1}^{N}\frac{E[\Delta \mathbf{C}_i{}^2]}{E[\mathbf{C}_i{}^2]}E[\mathbf{X}^2]\right) +$$
$$+\sum_{i=2}^{N}\left(\prod_{j=i}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)E[\Delta \mathbf{B}_{i-1}{}^2]\right) + E[\Delta \mathbf{B}_N{}^2] \qquad (5.26)$$

E a relação entre a potência dos erros (Equação (5.26)) e do sinal (Equação (5.24)) é:

$$E[\Delta \mathbf{Y}_N{}^2] = \frac{\prod_{j=1}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)\left(E[\Delta \mathbf{X}^2] + \sum_{i=1}^{N}\frac{E[\Delta \mathbf{C}_i{}^2]}{E[\mathbf{C}_i{}^2]}E[\mathbf{X}^2]\right) + \sum_{i=2}^{N}\left(\prod_{j=i}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)E[\Delta \mathbf{B}_{i-1}{}^2]\right) + E[\Delta \mathbf{B}_N{}^2]}{\prod_{j=1}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)E[\mathbf{X}^2]}$$

O primeiro termo do numerador é a parcela de erro referente ao erro de quantização de coeficientes, conforme (5.18). O último termo se refere ao erro de quantização de barramento, no último estágio de filtragem e o termo intermediário representa os erros de quantização de barramento nos estágios anteriores de filtragem. O erro relativo pode então ser simplificado para:

$$\frac{E[\Delta \mathbf{Y}_N{}^2]}{E[\mathbf{Y}_N{}^2]} = \frac{E[\Delta \mathbf{X}^2]}{E[\mathbf{X}^2]} + \sum_{i=1}^{N}\frac{E[\Delta \mathbf{C}_i{}^2]}{E[\mathbf{C}_i{}^2]} + \sum_{i=1}^{N-1}\frac{E[\Delta \mathbf{B}_i{}^2]}{\prod_{j=1}^{i}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)E[\mathbf{X}^2]} + \frac{E[\Delta \mathbf{B}_N{}^2]}{\prod_{j=1}^{N}\left(\frac{L_j}{K_j{}^2}E[\mathbf{C}_j{}^2]\right)E[\mathbf{X}^2]}$$

Na transformada wavelet cada escala corresponde a um estágio de filtragem, e todos os

estágios de filtro são iguais. Portanto $L_j = L$, $K_j = K$, $\mathbf{C}_j = \mathbf{C}$, $\Delta\mathbf{B}_j = \Delta\mathbf{B}$, $\forall\, j \in \{1..N\}$ e o erro relativo referente aos erros de quantização de coeficientes e barramento pode simplificado para:

$$\frac{E[\Delta \mathbf{Y}_N{}^2]}{E[\mathbf{Y}_N{}^2]} = \frac{E[\Delta \mathbf{X}^2]}{E[\mathbf{X}^2]} + N\frac{E[\Delta \mathbf{C}^2]}{E[\mathbf{C}^2]} + \sum_{i=1}^{N}\left(\frac{K^2}{L\,E[\mathbf{C}^2]}\right)^i \frac{E[\Delta \mathbf{B}^2]}{E[\mathbf{X}^2]} \tag{5.27}$$

5.4. Considerações para a Etapa de Análise

Conforme apresentado no capítulo anterior, os coeficientes dos filtros H_0 e H_1 utilizados na transformada wavelet ortonormal possuem as mesmas magnitudes, mas com posição e sinais alterados. Como estes coeficientes foram considerados variáveis aleatórias na análise de erros, e apenas o segundo momento destas variáveis é considerado no modelamento dos erros de quantização, os filtros H_0 e H_1 de wavelets ortonormais podem ser considerados "estatísticamente iguais", ou seja, eles contribuem de maneira idêntica para o erro de quantização total.

Segundo indica a Figura 5.1, a etapa de análise decompõe o sinal de entrada em várias bandas, e cada uma delas é gerada por uma sequência diferente de filtros. O segundo e terceiro termos de (5.27) mostram claramente que o erro de quantização é proporcional ao número de estágios de filtro em cascata, uma vez que todos os estágios de filtragem são iguais. Assim sendo, o erro de quantização estimado nas saídas dos filtros H_{0N} e H_{1N} da Figura 5.1 pode ser considerado o erro máximo.

As decimações (subamostragens) feitas na saída de cada filtro não alteram significativamente as considerações feitas no modelamento. O efeito desta operação é eliminar amostras pares do sinal, reduzindo o coeficiente de autocorrelação dos sinais nas entrada dos filtros, o que está de acordo com as considerações feitas para simplificação da análise.

5.5. Considerações para a Etapa de Síntese

A etapa de síntese difere da etapa de análise pela operação de interpolação (sobreamostragem) realizada a cada estágio, e pela soma das bandas na saída no mesmo estágio. A interpolação significa inserir uma amostra nula a cada amostra do sinal de entrada, e não afeta significativamente as propriedades estatísticas do sinal. Já a soma dos sinais (e dos erros) de cada banda deve ser equacionada para determinar sua influência sobre o erro de quantização. Para maior generalidade na análise será considerado um sistema de M bandas, como mostra a Figura

5.5 e não apenas com 2 bandas, como na Figura 5.1.

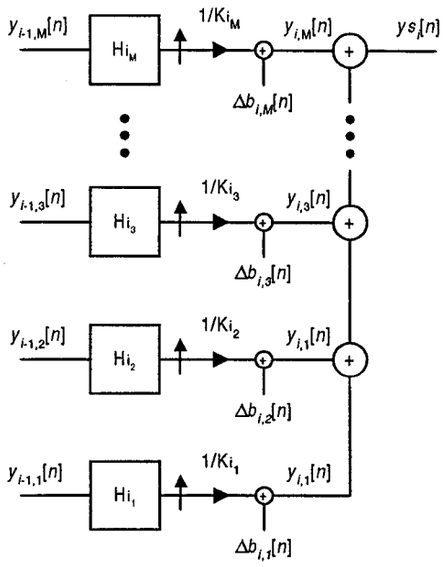


Figura 5.5: Modelo estatístico da etapa de síntese de um sistema com *M* bandas

A saída com erro de quantização nesta etapa é a soma das saídas com erro de cada banda.

$$\begin{aligned}\tilde{y}_{s_j}[n] &= \sum_i^M \tilde{y}_j[n] \\ &= \sum_i^M y_j[n] + \sum_i^M \Delta y_j[n]\end{aligned} \tag{5.28}$$

A saída com erro de quantização pode ser decomposta em erro de quantização e saída sem erros.

$$y_{s_j}[n] = \sum_i^M y_j[n]$$

$$\Delta y_{s_j}[n] = \sum_i^M \Delta y_j[n]$$

Considerando as saídas de cada banda $y[n]$ e $\Delta y[n]$ como sequências aleatórias conforme (5.6), o erro de quantização relativo de saída é:

$$\frac{E[\Delta \mathbf{Ys}_j{}^2]}{E[\mathbf{Ys}_j{}^2]} = \frac{\sum_i^M E[\Delta \mathbf{Y}_j{}^2]}{\sum_i^M E[\mathbf{Y}_j{}^2]}$$

Mas a média de um conjunto de dados nunca é maior que o maior dado do conjunto. Portanto o erro relativo calculado após a soma dos sinais das M bandas na etapa de síntese é menor ou igual ao maior erro relativo entre todas estas bandas.

$$\frac{E[\Delta \mathbf{Ys}_j{}^2]}{E[\mathbf{Ys}_j{}^2]} \leq \max\left(\frac{E[\Delta \mathbf{Y}_{j,i}{}^2]}{E[\mathbf{Y}_{j,i}{}^2]}\right), \quad \forall i \in \{1..M\}$$

No caso de filtros para transformadas wavelet ortonormais, os filtros de síntese possuem os mesmos coeficientes que os de análise, mudando apenas a sua ordem e seus sinais. Como no modelamento de erro os coeficientes foram considerados variáveis aleatórias, e estas alterações não afetam a magnitude dos coeficientes, pode-se considerar que os coeficientes na etapa de síntese são "estatísticamente iguais" aos da etapa de análise. Como na etapa de análise os filtros H_0 e H_1 também são "estatísticamente iguais" e o maior erro é gerado pelo caminho com maior número de estágios de filtragem, é possível uma simplificação na estimativa dos erros na síntese. Na etapa de síntese de wavelets ortonormais o maior erro entre os sinais de várias bandas será sempre aquele produzido pelo filtro ligado ao maior número de filtros anteriores.

5.6. Avaliação

A avaliação do modelo estatístico é feita comparando o erro estimado por (5.27) com aquele obtido empiricamente após a transformada wavelet direta (etapa de análise) e inversa. (etapa de síntese), conforme mostra a Figura 5.6. Como não há nenhum tipo de processamento no sinal transformado, idealmente o sinal reconstruído na saída seria uma reprodução do sinal de entrada. Mas devido aos vários erros de quantização envolvidos o que se tem é uma aproximação do sinal de entrada. A diferença entre esta aproximação e o sinal original é comparada com o erro teórico estimado. A diferença entre o erro de quantização medido e o teórico mostra quão boa é a aproximação do modelo estatístico. Esta técnica de avaliação foi utilizada por {Dick, 1996}.

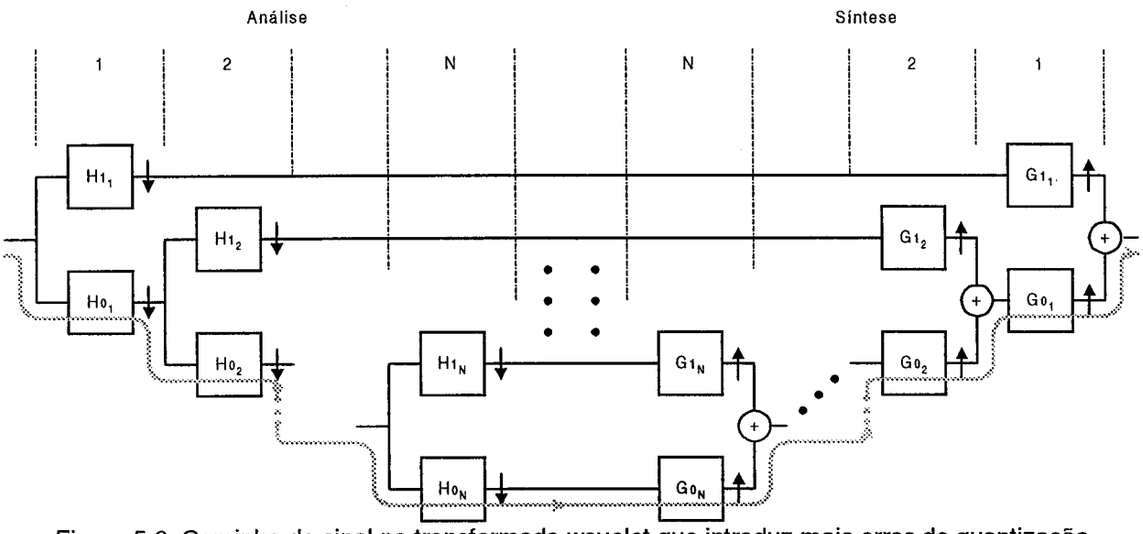


Figura 5.6: Caminho do sinal na transformada wavelet que introduz mais erros de quantização

As transformadas wavelet direta e inversa são implementadas por meio de um programa que utiliza aritmética de ponto fixo e palavra de tamanho limitado, e introduz os mesmos erros de quantização de coeficientes e de barramento que uma implementação em hardware apresentaria. Este programa é estimulado com algumas imagens padrões e sua saída reconstruída é comparada com a entrada original, sem erros de quantização.

O estímulo é feito por meio de imagens padronizadas de teste e sinais aleatórios. As imagens de teste são imagens monocromáticas, com 8 bits de luminância e tamanho de 256x256 *pixels*. Dois tipos de sinais aleatórios também são utilizados, um deles sendo um sinal de ruído branco com distribuição uniforme e sem autocorrelação e o outro um sinal autoregressivo com coeficiente de correlação de 0.95, que é um modelo largamente adotado na representação da intensidade dos *pixels* de uma imagem {Stark e Woods, 1994}.

Devido ao fato de filtros de wavelets biortogonais geralmente terem coeficientes inteiros, ou seja, capazes de serem representados sem erros de quantização, apenas filtros de wavelets ortonormais tem seus erros analisados. A análise investiga a dependência dos erros de análise/síntese com o número de bits utilizados na representação dos coeficientes, o tamanho do barramento e o número de escalas.

Outra variável considerada em (5.27) e que afeta diretamente o erro de quantização de barramento é a constante de normalização K. Ela deve ser definida de modo a evitar que o sinal de saída supere o valor máximo, mas que preserve a faixa dinâmica dos sinal. Existem vários critérios de normalização {Porat, 1997}, e entre eles foi adotado aquele que determina a

normalização pela primeira norma do filtro. Ou seja:

$$Ke = \sum_{i}^{L} |c_i| \tag{5.29}$$

Esta normalização é a mais conservadora de todas, e impede a ocorrência de *overflow* na saída. Outras normalizações utilizam coeficientes menores, mas calculados de modo a manter a probabilidade de *overflow* muito baixa. Neste caso, recomenda-se que o filtro utilize aritmética de saturação {Peleg e outros, 1997} para garantir que no caso da ocorrência de *overflow* ou *underflow* a saída se mantenha nos seus valores extremos {Porat, 1997}.

Como a normalização implica na divisão do sinal de saída pela constante de normalização, seria muito conveniente para a implementação que esta divisão pudesse ser feita apenas com deslocamentos. Para isso *Ke* deveria ser uma potência de 2, o que nem sempre ocorre. Neste caso, o procedimento adotado foi o arredondamento de (5.29) para a menor potência de 2 mais próxima, mesmo com a possibilidade de haver saturação da saída.

$$\log_2 Ke = \left\lceil \log_2 \left(\sum_{i}^{L} |c_i| \right) \right\rceil$$

5.6.1. D4

A wavelet Daubechies-4 é uma wavelet ortonormal, e seus coeficientes estão na Tabela 4.1. A Equação (5.27) é utilizada para a estimativa do erro, tanto na etapa de análise quanto na etapa de síntese. Como a transformação é de uma imagem e a transformada wavelet é aplicada horizontalmente e verticalmente, portanto estes erros devem ser duplicados. As constantes de normalização são respectivamente 2^{Nbc} e 2^{Nbc-1} para a etapa de análise e síntese, e os coeficientes dos filtros H_0, H_1, G_0 e G_1 são tornados inteiros pela multiplicação pela constante $2^{Nbc-1/2}$, onde *Nbc* é o número de bits dos coeficientes. A legenda para os gráficos de erro de quantização está na Figura.5.7.

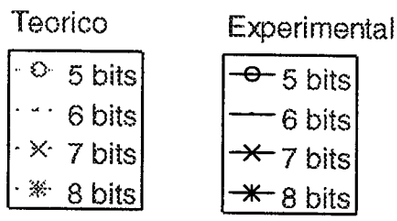


Figura.5.7: Legenda para gráficos de erro de quantização

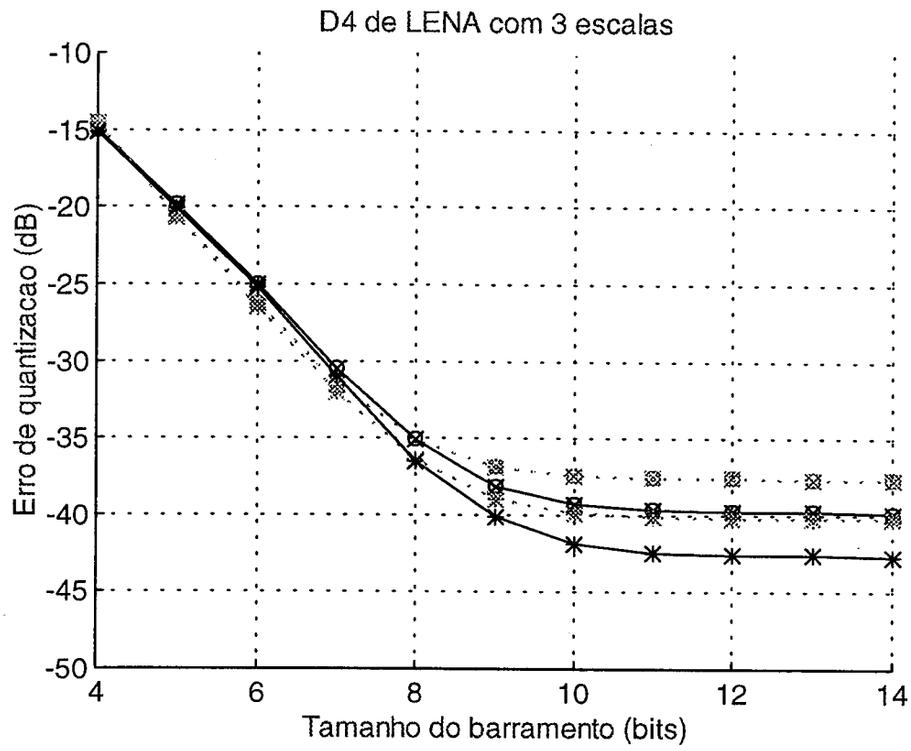


Figura.5.8: "Lena" processada com 3 escalas

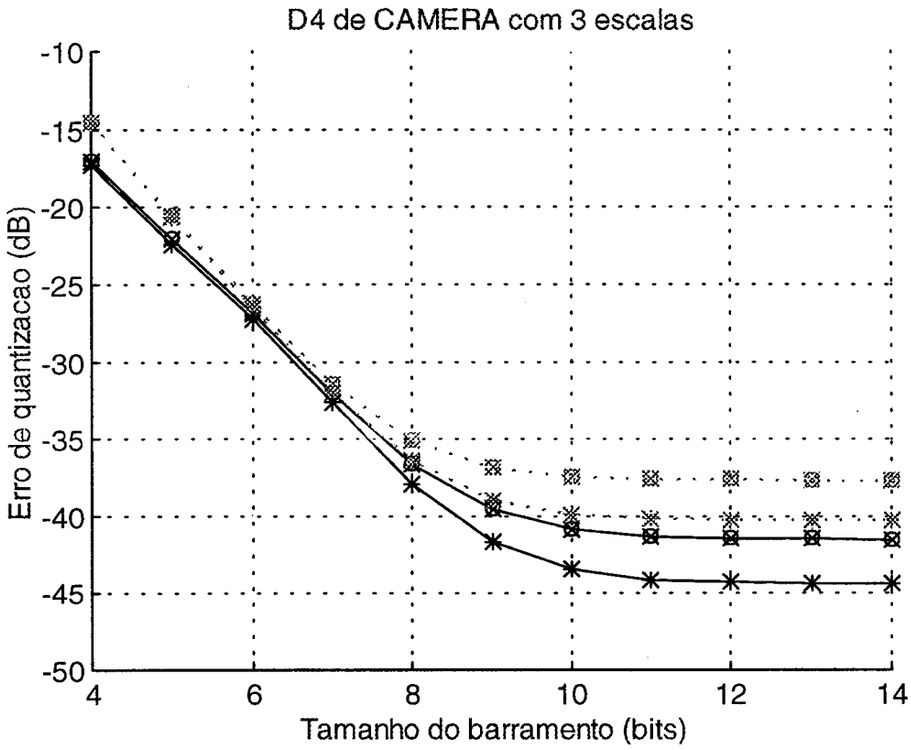


Figura.5.9: "Camera" processada com 3 escalas

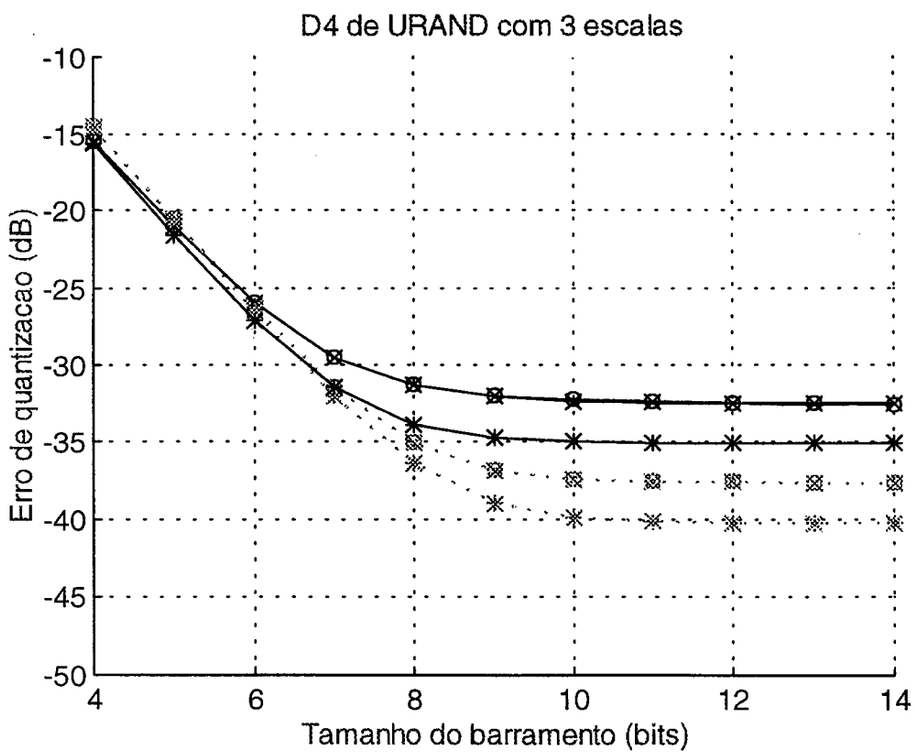


Figura 5.10: Variável aleatória sem autocorrelação processada com 3 escalas

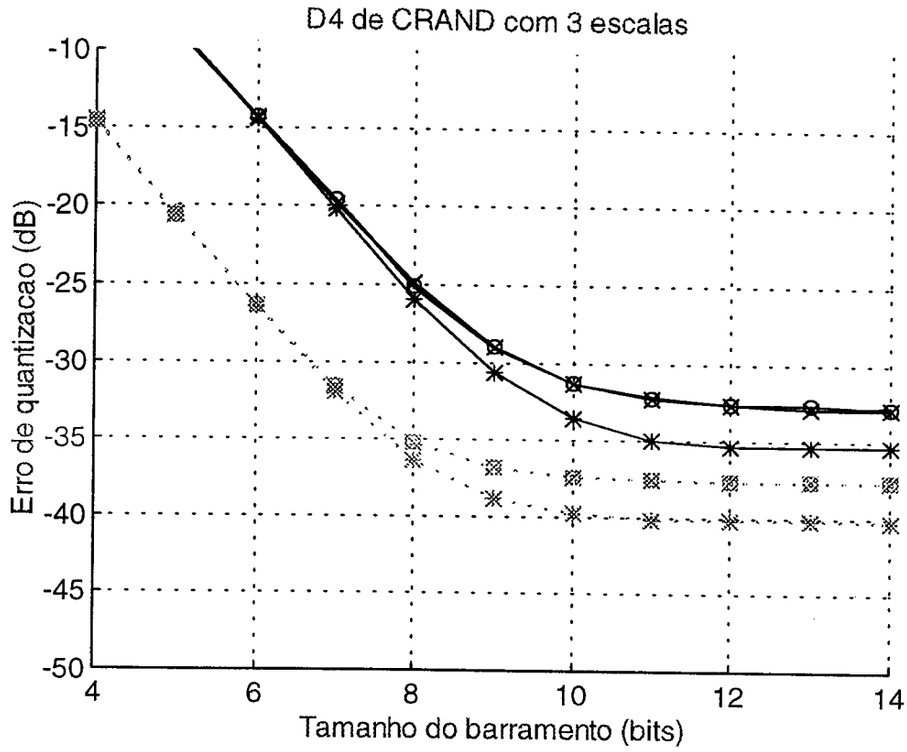


Figura 5.11: Variável aleatória com autocorrelação de 0.95 processada com 3 escalas

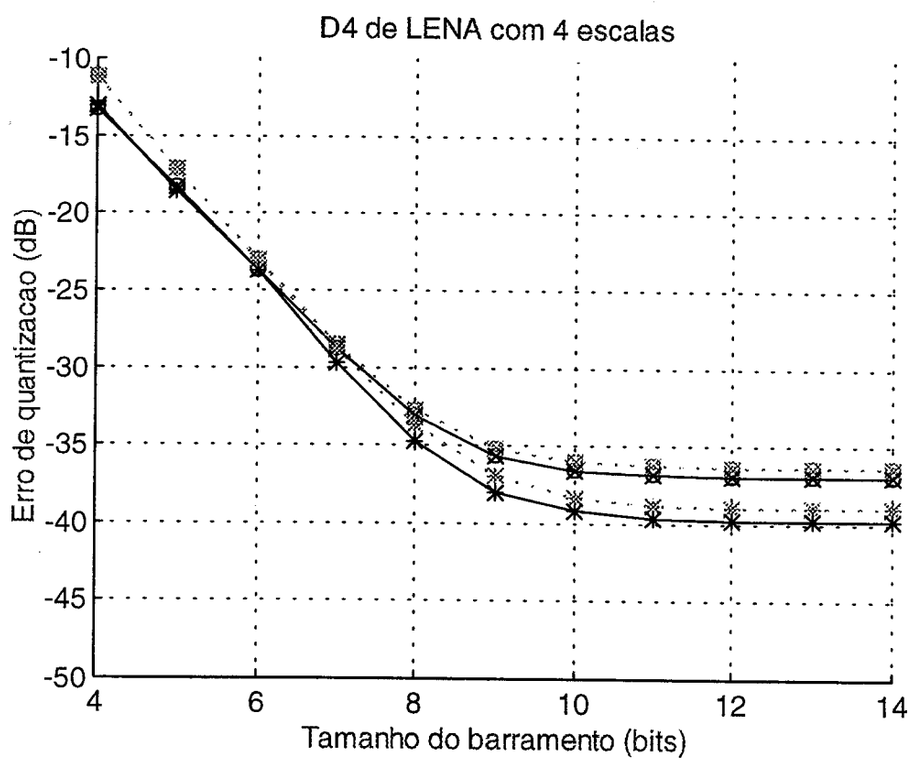


Figura.5.12: "Lena" processada com 4 escalas

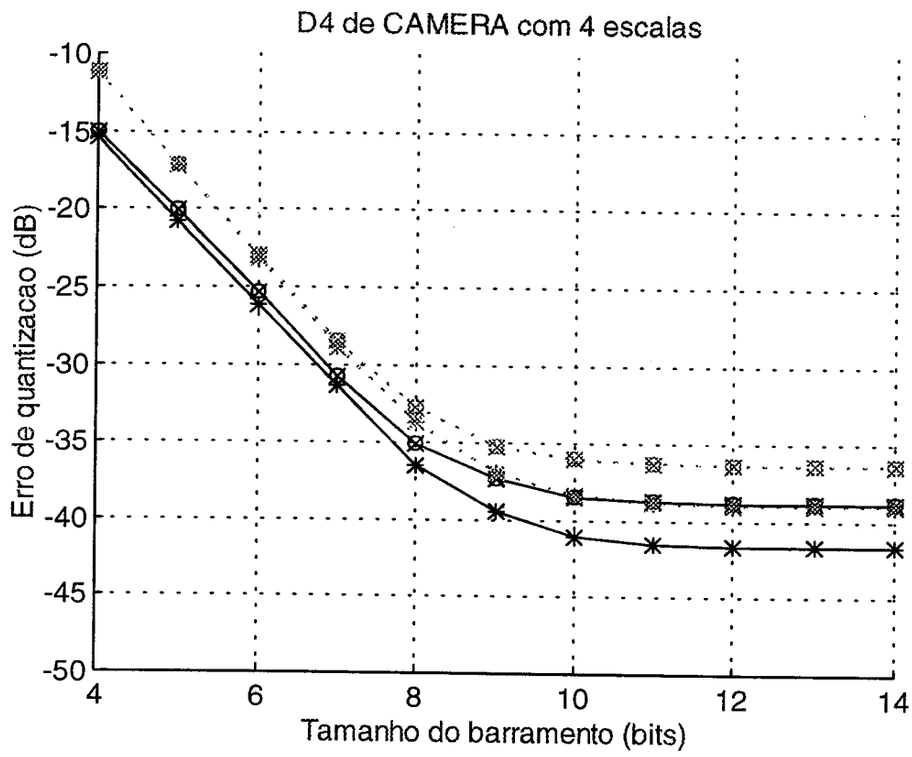


Figura.5.13: "Camera" processada com 4 escalas

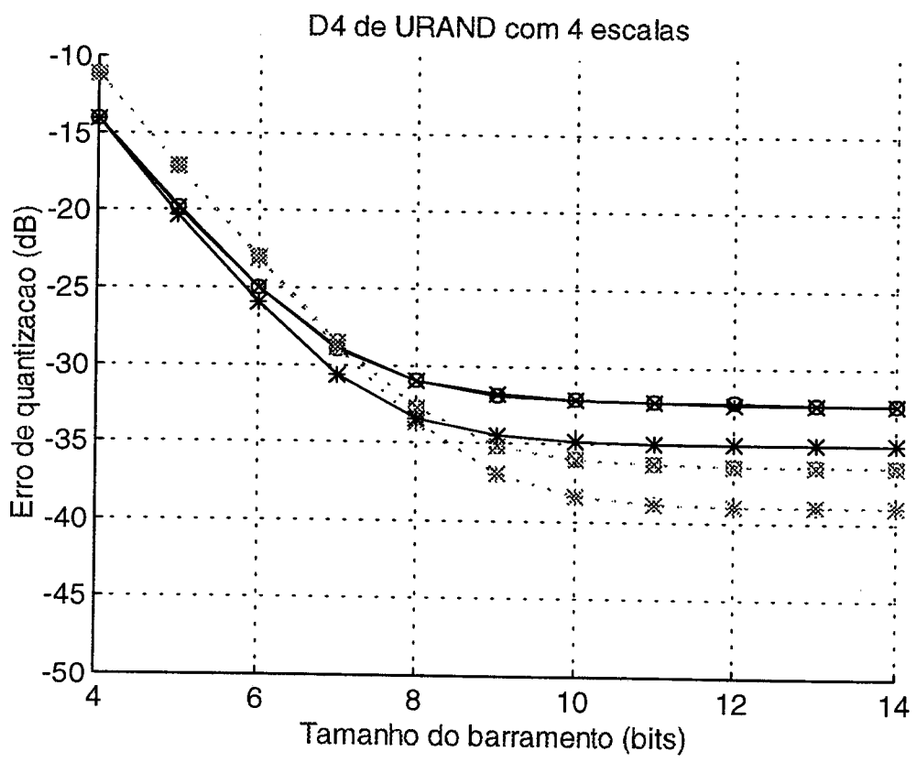


Figura 5.14: Variável aleatória sem autocorrelação processada com 4 escalas

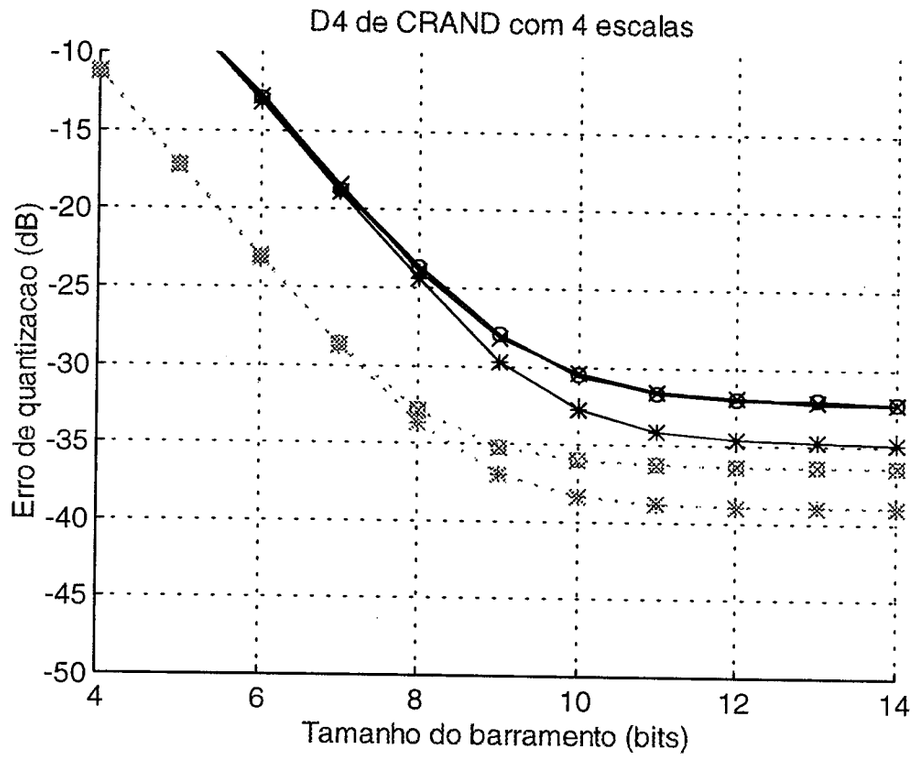


Figura 5.15: Variável aleatória com autocorrelação de 0.95 processada com 4 escalas

5.6.2. D6

A wavelet Daubechies-6 também é uma transformada ortonormal e por isso a análise feita foi similar à de Daubechies-4.

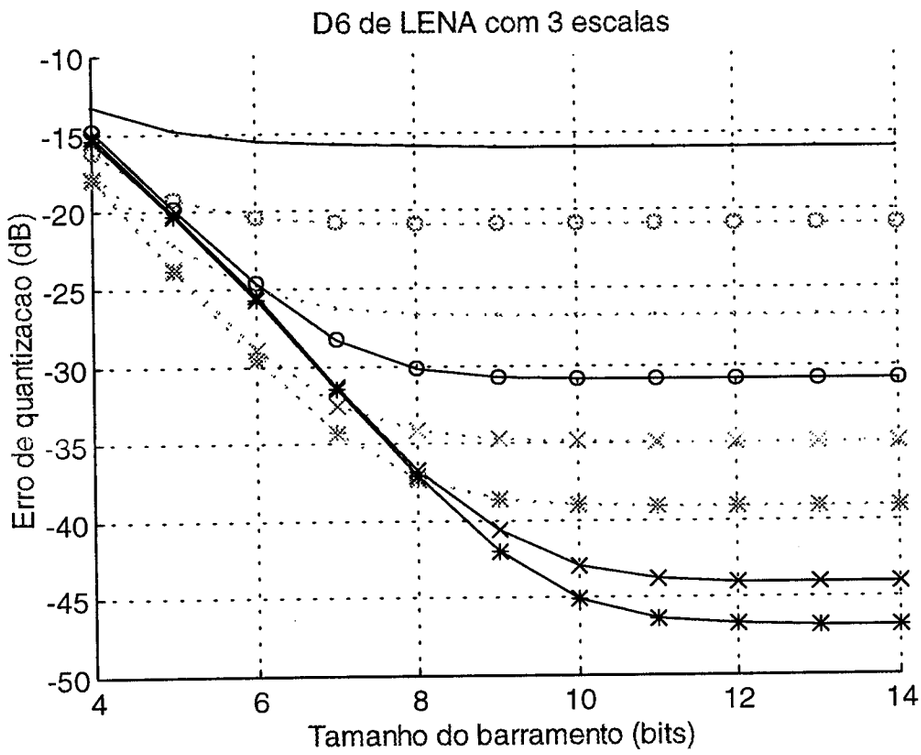


Figura 5.16: "Lena" processada com 3 escalas

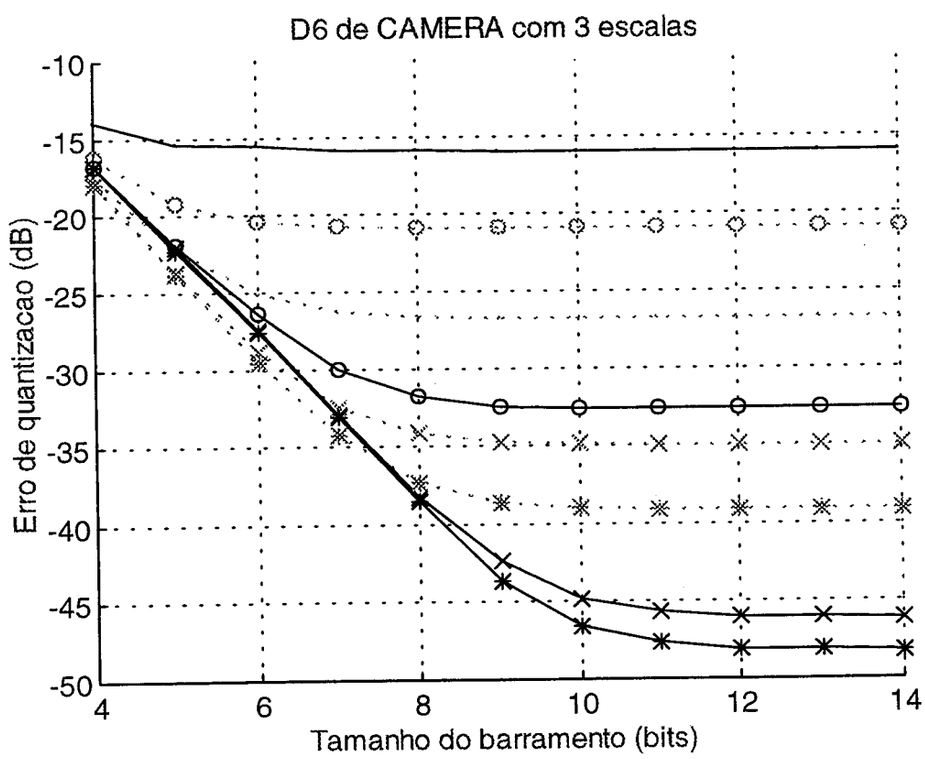


Figura 5.17:"Camera" processada com 3 escalas

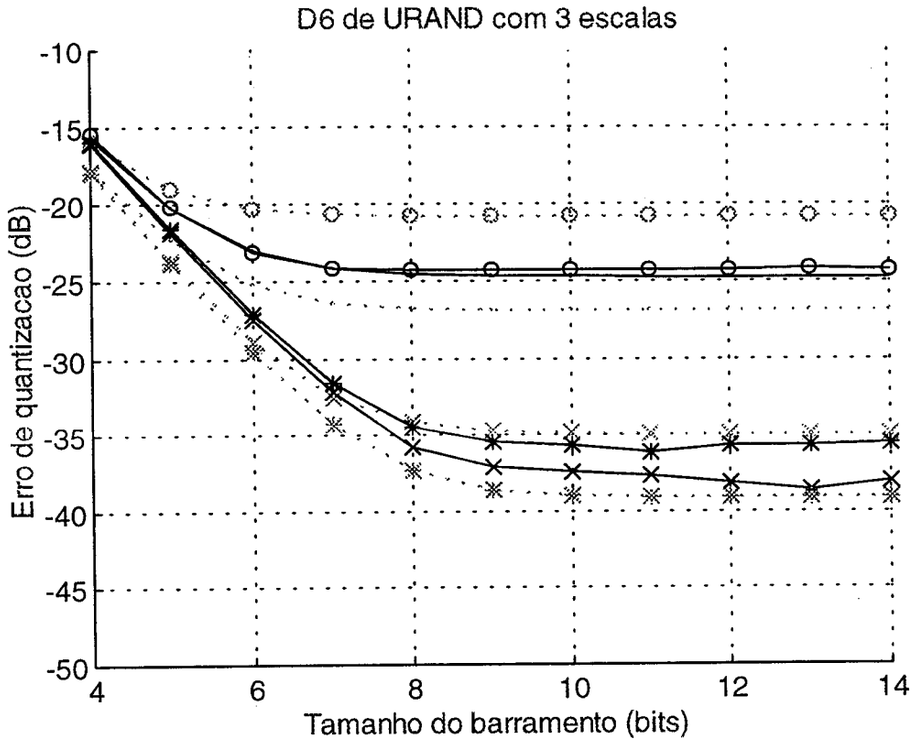


Figura 5.18: Variável aleatória sem autocorrelação processada com 3 escalas

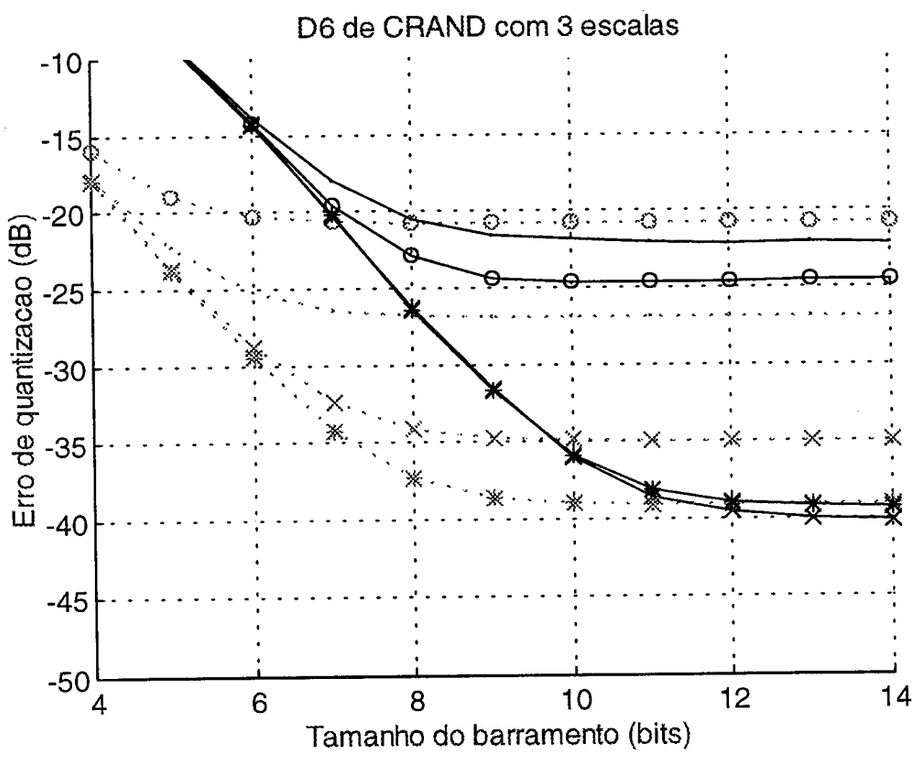


Figura 5.19: Variável aleatória com autocorrelação de 0.95 processada com 3 escalas

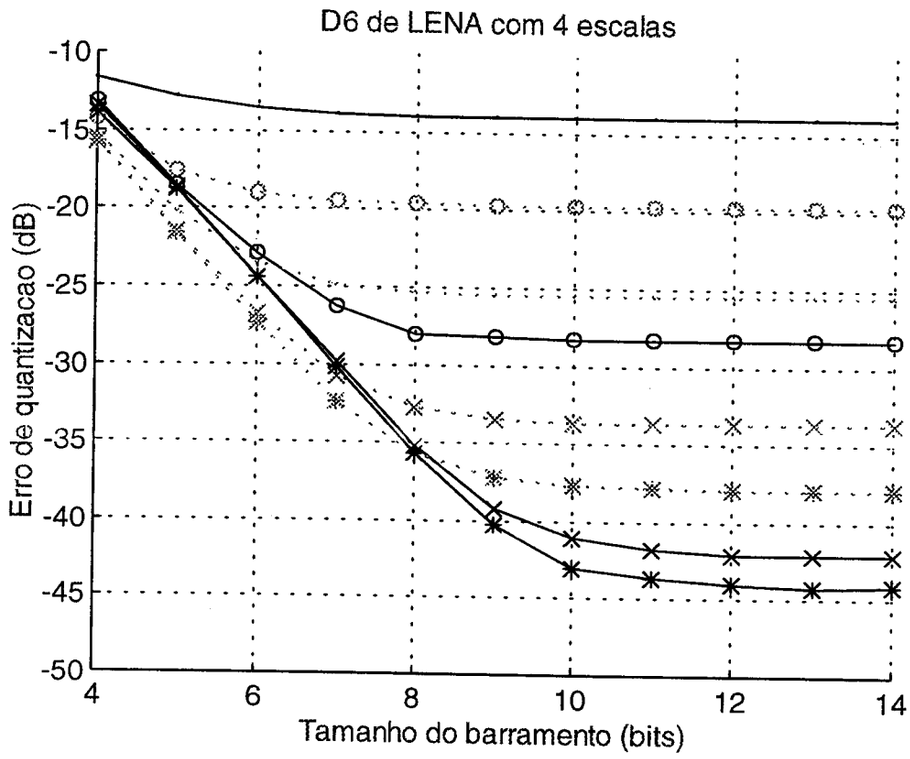


Figura 5.20: "Lena" processada com 4 escalas

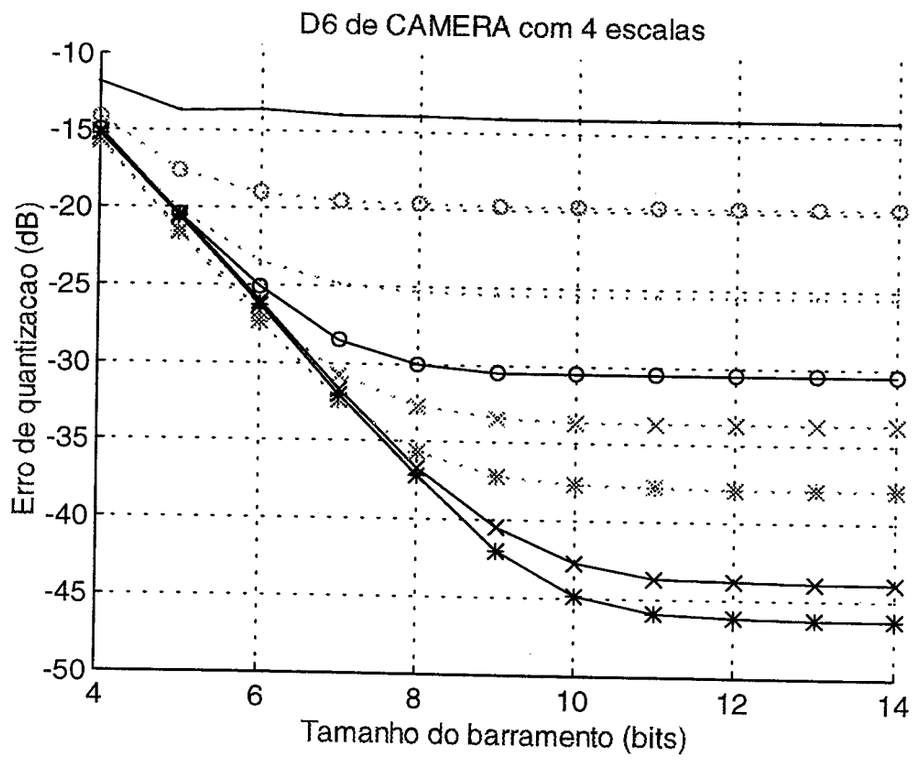


Figura 5.21:"Camera" processada com 4 escalas

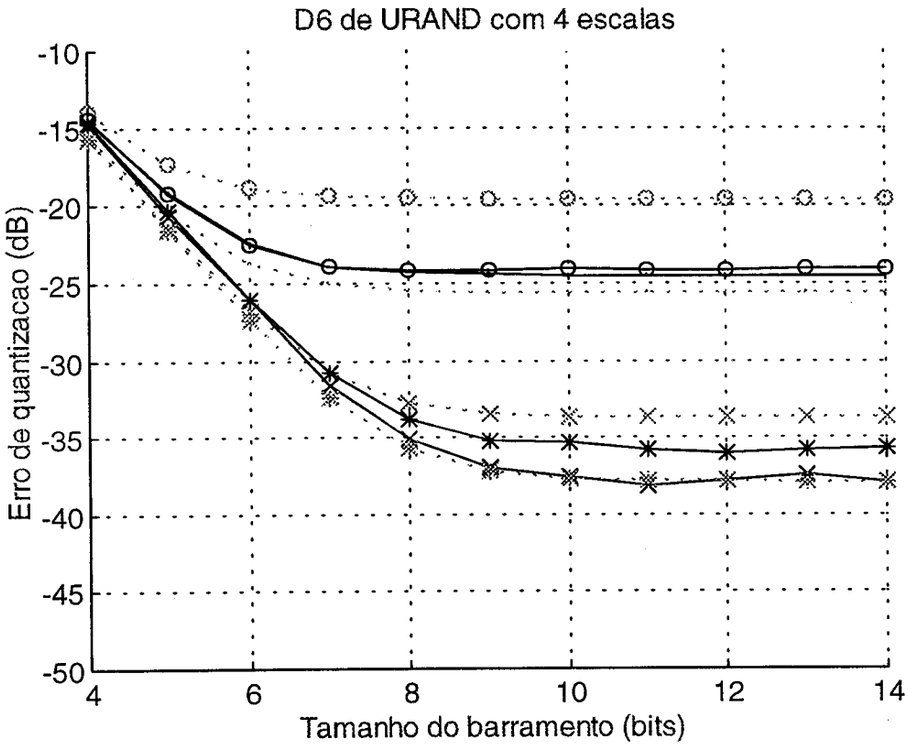


Figura 5.22: Variável aleatória sem autocorrelação processada com 4 escalas

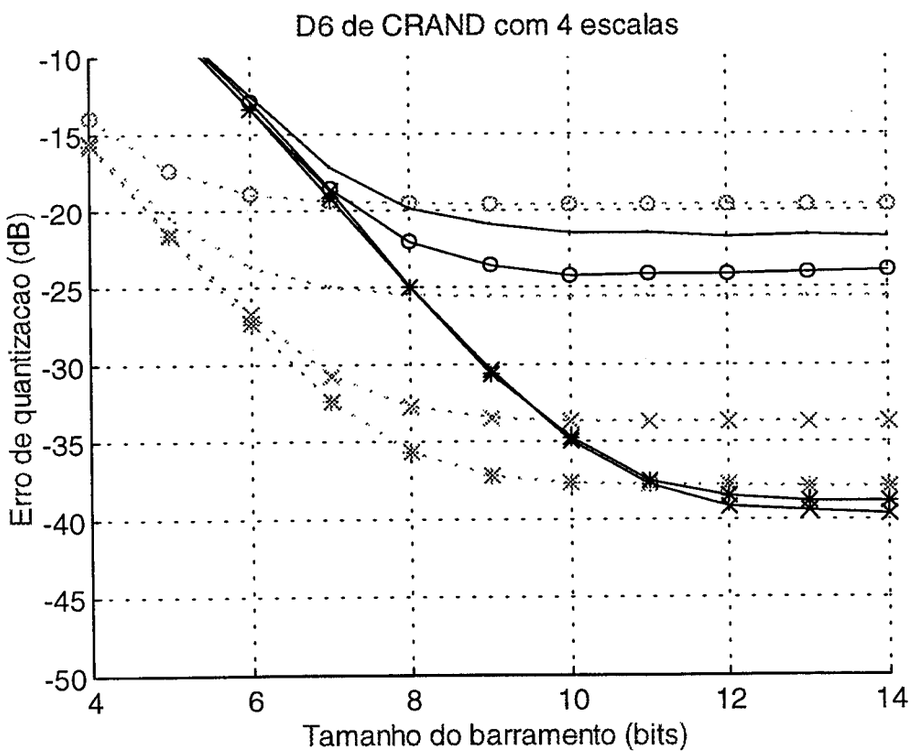


Figura 5.23: Variável aleatória com autocorrelação de 0.95 processada com 4 escalas

5.7. Considerações Finais

As simulações mostram que o modelamento reproduz o erro de quantização da wavelet Daubechies-4 com fidelidade. Fica evidente que a partir de um certo valor não existe vantagem em aumentar o tamanho das palavras, pois o erro relativo se mantém. Analisando a equação (5.27), observa-se que isso se dá quando o termo referente ao erro de quantização no barramento se torna desprezível e o erro de quantização dos coeficientes passa a ser o termo dominante.

Outra característica que merece menção é o fato de apenas haver melhora significativa no erro relativo quando os coeficientes são representados com 8 bits. Com 5, 6 e 7 praticamente não existe diferença significativa no erro relativo, que está na faixa de 40 dB, ou seja, é aproximadamente de 1%. A explicação é que os o erro de quantização absoluto dos coeficientes Δc_i é 0.072 quando Nbc=5. Assim, quando se usam mais bits para representar os coeficientes, o erro de quantização de coeficientes aumenta na mesma proporção e por isso não há ganho no erro relativo. Só existe alguma alteração neste comportamento quando $\Delta c_i > 0.5$, o que ocorre quando $Nbc = 8$.

A análise dos erros do Daubechies-6 mostra que o modelo não apresenta fidelidade tão boa quanto no caso das Daubechies-4, particularmente quando os coeficientes são representados com poucos bits. O caso que mais chama a atenção e que se repete sistematicamente nas 4 imagens de estímulo é o erro relativo com Nbc=5. Isso pode-se explicar pois com estes coeficientes os erros Δc_i são todos arredondados em direção ao maior valor, fazendo com que a média deste erros seja não nula. Isso mostra a sensibilidade do modelo às condições propostas no seu desenvolvimento, que considerava os erros Δc como sequências aleatórias de média zero. Isto também pode ser observado no erro apresentado quando o sinal é uma v.a. correlacionada. O modelamento deste erro é sistematicamente menos preciso que quando o erro é uma v.a. não-correlacionada, que foi uma das condições utilizadas no equacionamento.

Comparando o perfil dos erros das wavelets D4 e D6, pode-se observar que a D4 possui uma saída muito menos sujeita a erros que o D6. Para implementação em hardware, a D6 exige uma palavra para representação de coeficientes maior, o que faz sua implementação muito custosa. Para a D4 o melhor tamanho de barramento é de 11 bits, e os coeficientes são inteiros, com um pequeno erro de arredondamento (3, 5, 11 e 19).

6. Alternativas de Implementação

6.1. Introdução

Existem algumas técnicas de implementação de algoritmos largamente utilizadas em FPGAs. Particularmente em implementações de algoritmos de processamento de sinais para FPGAs, as técnicas de processamento baseadas em aritmética distribuída {Bergmann e Chung, 1997} {Dick e Harris, 1998} {Greenfield e outros, 1997} {Woods e outros, 1998}, arquitetura bit-serial {Panneerselvam e outros, 1995} {Andraka, 1993} {Hesener, 1996} {Lim e outros, 1993} {Turner e outros, 1995}, e implementações otimizadas de multiplicadores {Kean e outros, 1996} {Haynes e Cheung, 1999} e somadores {New, 1996a} {New, 1996b} {Xing e Yu, 1998} são muito comuns. É necessário avaliar os méritos destas técnicas para a implementação na arquitetura de FPGA escolhida.

A família de FPGAs XC6200 possui uma granularidade fina e poucos recursos de roteamento. A maior parte dos trabalhos publicados utilizando FPGAs para processamento de sinais trata de FPGAs baseadas em LUTs. Por isso é necessário avaliar as alternativas de implementação cuidadosamente sob estas restrições.

6.2. Estruturação das Entidades

A escassez de recursos de roteamento na família XC6200, associada com a simplicidade de cada unidade funcional, oferecem grandes dificuldades na implementação de circuitos. Diferentemente de outras FPGAs, a FPGA XC6200 possui um número muito restrito de recursos de roteamento {Woods e outros, 1998} à disposição das ferramentas de posicionamento e roteamento (P&R). A ferramenta de posicionamento e roteamento oferecida pelo fabricante para esta família de FPGAs, a XACT6000, apenas consegue rotear circuitos que exigem menos de 25% da capacidade do dispositivo {Hartenstein e outros, 1998}. Como não faz sentido desperdiçar 75% da capacidade do dispositivo, é necessário que o projetista dê um auxílio à ferramenta na forma de um posicionamento manual das entidades. Este posicionamento pode ser inserido no *netlist* de modo a ser reconhecido pela ferramenta XACT6000. Ou seja, o *floorplanning* manual, uma exceção em outras famílias de FPGA é regra na família XC6200.

Isto dificulta a utilização de bibliotecas de entidades padronizadas como aquelas

disponíveis em ferramentas de síntese lógica, como as da Synopsys™. É necessário que estas bibliotecas incorporem também informações de posicionamento para garantir a regularidade do posicionamento, e consequentemente o roteamento satisfatório.

Uma vez que a síntese direta não pode ser utilizada, duas abordagens se apresentam como interessantes para orientar o desenvolvimento nesta FPGA: a abordagem *bit-slice* e a de macros estruturadas.

6.2.1. Abordagem *Bit-Slice*

Nesta abordagem, procura-se identificar o conjunto de componentes responsável pelo processamento de 1 bit. Cria-se então uma entidade agrupando estes componentes cuidadosamente posicionados e roteados. Finalmente repete-se esta estrutura para todos os bits que formam a palavra, criando-se assim a arquitetura final {Heron e Woods, 1996}.

Existem uma grande vantagem nesta abordagem: uma vez roteada a entidade, o roteamento de quantos bits forem necessários está assegurada, além de ser um roteamento regular e previsível. Isto é muito interessante para FPGAs que permitem roteamento parcial, pois esta regularidade permite que apenas alguns bits da entidade sejam reconfigurados, sem alterar a configuração dos outros {Kean e outros, 1996}. Mas esta abordagem funciona apenas em circuitos de pequena complexidade, e as entidades geradas, por serem específicas para cada implementação, não permitiem o seu reaproveitamento na forma de bibliotecas de uso geral.

6.2.2. Abordagem de Macros Estruturadas

Enquanto a abordagem *bit-slice* considera a segmentação do circuito no sentido paralelo ao barramento, esta considera a segmentação perpendicular ao mesmo. Os blocos segmentados seriam os sucessivos estágios de processamento da arquitetura, como os somadores, registradores, etc. Como no caso da abordagem *bit-slice*, estes blocos também seriam posicionados e roteados manualmente, e este posicionamento seria de algum modo gravado na descrição da macro, tornando-a pronta para reutilização. É uma estratégia muito interessante, pois estes blocos se repetem sem alteração em várias arquiteturas, enquanto que os blocos formados na *bit-slice* são específicos. Por outro lado, o roteamento dos sinais que interligam as macros são deixados para o nível hierárquico superior.

Estas macros são também chamadas de macros parametrizáveis pois, para torná-las mais

gerais, frequentemente uma ou mais características (tamanho, por exemplo) é parametrizada.

Estudos comparativos {Heron e Woods, 1996} entre esta abordagem e a bit-*slice* mostram que em circuitos simples, esta última apresenta uma utilização de área de 100%, enquanto que a primeira apresenta 82%, que é uma diferença pequena considerando-se que as macros estruturadas não são específicas para determinado circuito e podem ser reutilizadas.

6.3. Aritmética Distribuída

Esta técnica minimiza o número de multiplicações em uma estrutura do tipo FIR quando os coeficientes dos filtros são constantes. Seja $\mathbf{x}^j$ um vetor de B_x bits, $j \in \{0..\ B_x\text{-}1\}$, representando os dados de entrada do filtro, e $\mathbf{h}^j$ um vetor de B_h bits, $j \in \{0..\ B_h\text{-}1\}$, representando os coeficientes de um filtro de L *taps*.

$$x_i = \sum_{j=0}^{B_x-1} \mathbf{x}_i^j\, 2^j \qquad h_i = \sum_{k=0}^{B_h-1} \mathbf{h}_i^k\, 2^k \tag{6.1}$$

A saída deste filtro será:

$$y = \sum_{i=0}^{L-1} x_i\, h_i = \sum_{i=0}^{L-1} \left(\sum_{j=0}^{B_x-1} \mathbf{x}_i^j\, 2^j \sum_{k=0}^{B_h-1} \mathbf{h}_i^k\, 2^k \right)$$

Que pode ser reescrito como:

$$y = \sum_{j=0}^{B_x-1} 2^j \left(\sum_{i=0}^{L-1} \mathbf{x}_i^j \sum_{k=0}^{B_h-1} \mathbf{h}_i^k\, 2^k \right) \tag{6.2}$$

O termo interior da somatória depende apenas dos L bits dos dados, uma vez que os valores de $\mathbf{h}$ são constantes, e esta somatória pode ser predeterminada para todas as combinações dos L dados. Uma memória ROM ou RAM com L bits de endereços e $(B_h + L)$ bits de dados pode armazenar os resultados desta somatória. O resultado final seria obtido somando as saídas destas memórias, com o peso correto (termo 2^j na Equação (6.2)). Dessa forma os L multiplicadores poderiam ser substituídos por B_x memórias de 2^L palavras de $(B_h + L)$ bits, com vantagem no desempenho {Jaggernauth e outros, 1985}.

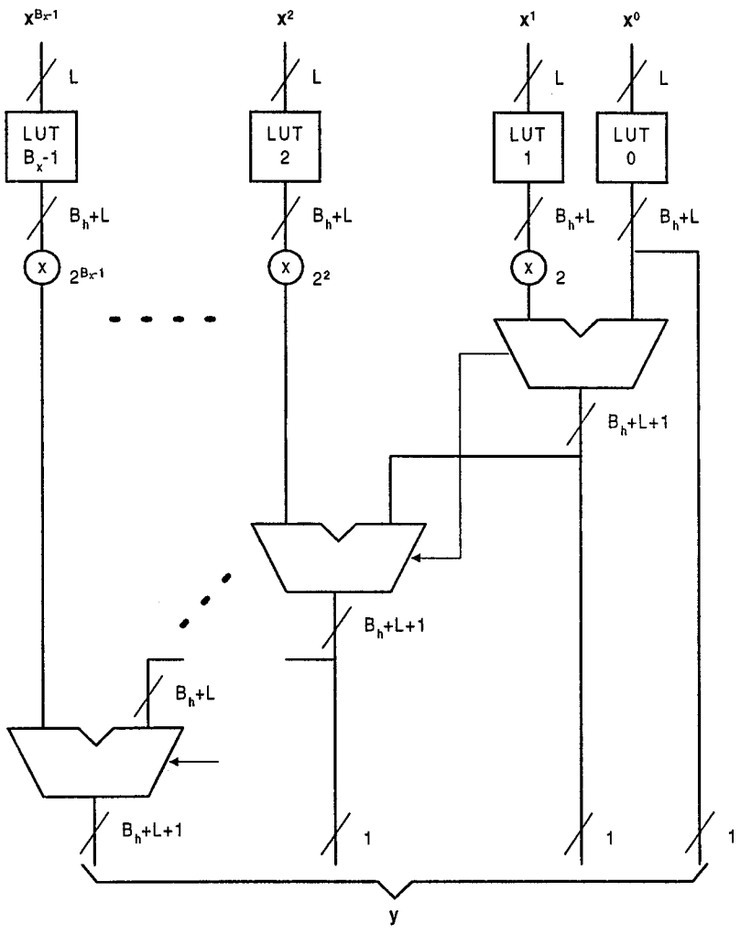


Figura 6.1: FIR utilizando aritmética distribuída

Esta técnica é muito interessante para FPGAs baseados em *look-up table* (LUT), pois estas podem ser utilizadas em lugar das memórias RAM ou ROM no armazenamento das somas em (6.2). FPGAs comerciais possuem LUTs de tamanho 16x1, permitindo a implementação direta para L=4, o que corresponde a filtros FIR de fase linear de 8 *taps* (como os filtros para DCTs de imagem {Trainor e outros, 1998}). Os resultados de {Pirsch e outros, 1995} mostram a eficiência do uso de aritmética distribuída em implementações VLSI e os de {Greenfield e outros, 1997} e {Altera, 1998} em implementações para lógicas reconfiguráveis.

Nos FPGAs de granularidade fina, como a família Xilinx XC6200 e Atmel 6000 os elementos lógicos não são LUTs, mas portas lógicas de 2 entradas. Isto não impede o uso de aritmética distribuída, pois LUTs podem ser implementadas eficientemente com ajuda de multiplexadores e de portas lógicas de 2 entradas. A Figura 6.2 mostra esta implementação. As entidades L são LUTs de 2 entradas e que podem ser implementadas por funções booleanas de 2

entradas, conforme a Tabela 6.1.

Tabela 6.1: Funções booleanas para implementação de LUT de 2 entradas

A=1 B=1	A=1 B=0	A=0 B=1	A=0 B=0	Função
0	0	0	0	0
0	0	0	1	$\overline{A+B}$
0	0	1	0	$\overline{A}.B$
0	0	1	1	$\overline{A}$
0	1	0	0	$A.\overline{B}$
0	1	0	1	$\overline{B}$
0	1	1	0	$A \oplus B$
0	1	1	1	$\overline{A.B}$
1	0	0	0	$A.B$
1	0	0	1	$\overline{A \oplus B}$
1	0	1	0	B
1	0	1	1	$\overline{A}+B$
1	1	0	0	A
1	1	0	1	$A+\overline{B}$
1	1	1	0	$A+B$
1	1	1	1	1

Uma vez que os elementos lógicos do FPGA XC6200 utilizam área de silício 10.6 vezes menor que os da família de FPGAs baseados em LUT Xilinx XC4000* {Kean e outros, 1996}, está solução é competitiva em termos de área, mas oferece atrasos maiores, pois possui vários níveis de lógica combinacional enquanto a LUT possui apenas um.

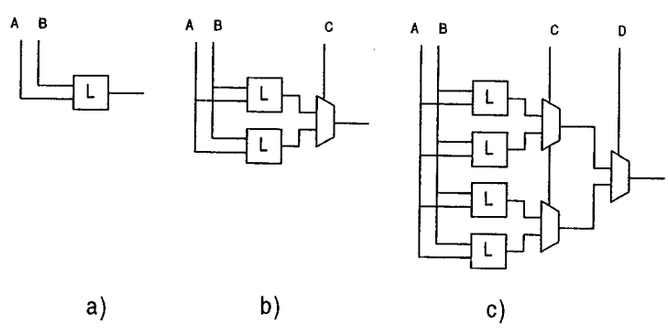


Figura 6.2: LUT para a) 2 b) 3 c) 4 bits

Um exemplo desta implementação de um filtro FIR de 4 *taps*, com palavras de 12 bits e coeficientes de 4 bits no XC6200 exigiria 11 somadores de 8 bits e mais 4 LUTs 4x1 de 8 bits.

---

* Quando implementados utilizando a mesma tecnologia

Cada LUT 4x1 ocupa 7 blocos lógicos, o que daria um total de 672 blocos lógicos, além dos somadores, e sem contar possíveis registros de *pipeline*. É um valor adequado se os coeficientes do filtro FIR não forem na sua maior parte potências de 2. Neste caso a implementação do filtro FIR na sua forma canônica é mais vantajosa.

6.4. Multiplicadores

A aritmética distribuída também pode ser utilizada para implementar eficientemente multiplicadores por constante. Considere-se a multiplicação dos vetores de bits da Equação (6.1). Uma multiplicação entre dois números binários se escreve como:

$$y = xh = \sum_{j=0}^{B_x-1} \mathbf{x}^j \, 2^j \sum_{k=0}^{B_h-1} \mathbf{h}^k \, 2^k \tag{6.3}$$

Se existe à disposição LUTs de E entradas, o vetor de B_x bits $\mathbf{x}$ da Equação (6.3) pode ser dividido em N vetores menores $\mathbf{x}_i$, $i \in \{0..N\text{-}1\}$, de tamanho E ou menor:

$$y = \left(\sum_{j=0}^{E-1} \mathbf{x}_0^j \, 2^j + \sum_{j=0}^{E-1} \mathbf{x}_1^j \, 2^{E+j} + \sum_{j=0}^{E-1} \mathbf{x}_2^j \, 2^{2E+j} + \ldots \right) \sum_{k=0}^{B_h-1} \mathbf{h}^k \, 2^k = \sum_{i=0}^{N-1} 2^{iE} \left(\sum_{j=0}^{E-1} \mathbf{x}_i^j \sum_{k=0}^{B_h-1} \mathbf{h}^k \, 2^k \right)$$

Esta equação é basicamente (6.2), com a diferença que os E bits que endereçam a LUT pertencem ao mesmo vetor $\mathbf{x}$, e os pesos para a soma das saídas das LUTs é 2^{iE}. Esta técnica foi implementada no FPGA XC6200 por {Kean e outros, 1996}.

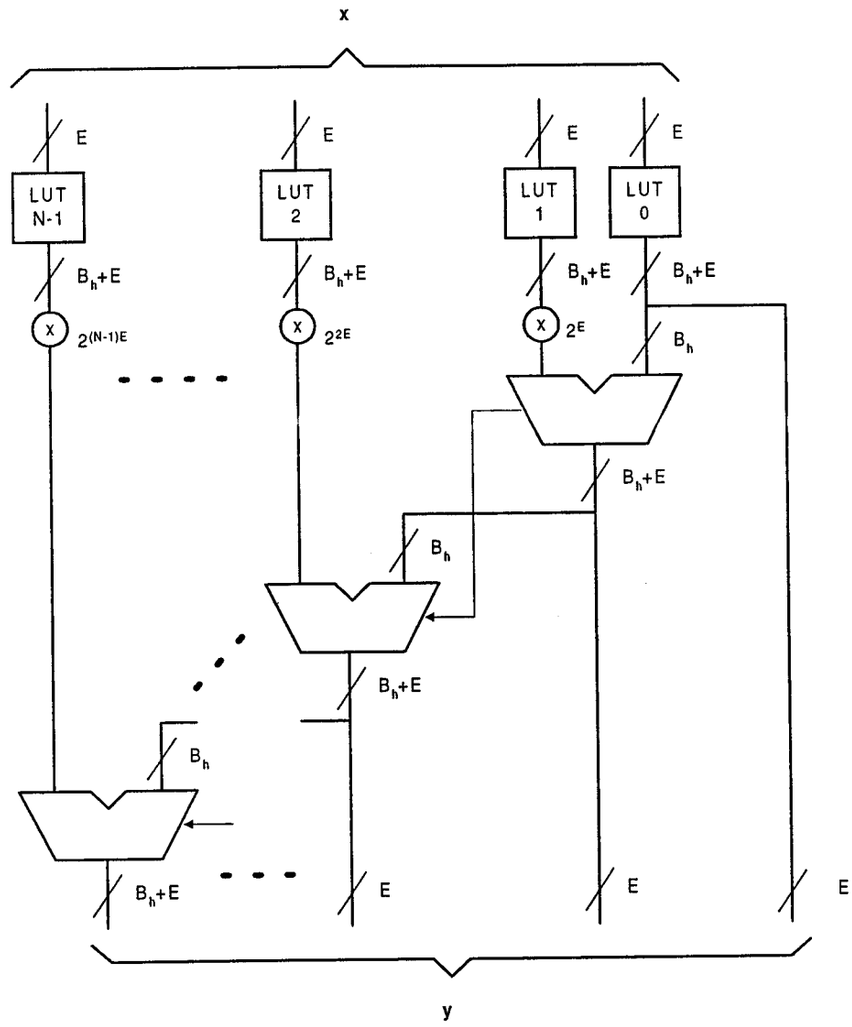


Figura 6.3 Multiplicador utilizando aritmética distribuída

O número de LUTs e somadores utilizados para implementar um filtro FIR é dado por (6.4). As LUTs devem ter L linhas de endereço e B_h+L linhas de dados, e os somadores devem ser de B_h+L bits.

$$N_{LUT} = B_x$$
$$N_{SOM} = N_{LUT} - 1 \qquad (6.4)$$

O número de LUTs e somadores utilizados para implementar um multiplicador é dado por (6.5). As LUTs devem ter E linhas de endereço e B_h+E linhas de dados, e os somadores devem ser de B_h+E bits.

$$N_{LUT} = \left\lceil \frac{B_x}{E} \right\rceil \qquad (6.5)$$
$$N_{SOM} = N_{LUT} - 1$$

Para multiplicação de uma palavra de 12 bits por constantes de 8 bits, são necessárias 3 LUTs de 4x12 mais 2 somadores de 12 bits. Apenas para as LUTs são utilizados 252 blocos lógicos do XC6200, o que não é excessivo se a constante de multiplicação possui muitos bits em 1. Se, ao contrário, tiver poucos bits em 1, a implementação por meio de somas e deslocamentos é mais vantajosa. Esta implementação de multiplicação por uma constante de 8 bits, com 3 bits em 1, exige 3 somadores de 20 bits cada, ou seja, 160 blocos lógicos.

6.5. Processamento Bit-Serial

A arquitetura bit-serial processa o sinal bit a bit, e não paralelamente. Isso reduz a área da implementação e simplifica o roteamento, pois as palavras binárias são transmitidas bit-a-bit por um única rota, ao invés de várias rotas em paralelo. Tipicamente, um algoritmo que processe uma palavra de N bits quando executado serialmente exige aproximadamente $1/N$ da área e demora N vezes mais tempo (ou ciclos de *clock*) para ser executado do que se fosse executada paralelamente {Hesener, 1996}. Ou seja, a arquitetura bit-serial é equivalente em termos do produto área-tempo (AT) à paralela. Como atrasos de roteamento são bastante significativos em FPGAs, a arquitetura bit-serial muitas vezes acaba tendo o produto AT menor {Andraka, 1993} que o da abordagem paralela, por exigir um menor número de rotas e rotas de menor comprimento {Lim e outros, 1993},.

Algoritmos de processamento de sinais normalmente são dominados por operações de somas e multiplicações. Como estas operações são iniciadas a partir do bit menos significativo (LSB) das palavras, é interessante que as sequências seriais de bits que representam as palavras tenham o LSB como o primeiro bit da sequência. Outras operações menos frequentes, como a ordenação e divisão, exigem o arranjo inverso. É possível inverter a ordem dos bits para estas operações, mantendo a ordem direta nas outras, mas isto implica no aumento da latência na saída dos dados.

A representação complemento-2 também é atraente para operações de soma e multiplicação {Denyer e Renshaw, 1985}, e é preferida em comparação com a representação complemento-1 e bit de magnitude.

Um sistema de processamento de sinais bit-serial pode ser generalizado como sendo composto por um conjunto de operadores interligados {Denyer e Renshaw, 1985}. Estes operadores recebem zero ou mais entradas, seriais e não-seriais, e geram uma ou mais saídas seriais após uma latência associada ao processamento efetuado. Esta latência é medida em ciclos de *clock*. As entradas e saídas podem ser entradas de dados ou de controle. O controle é necessário para sincronizar a operação dos operadores. Um sinal de controle óbvio é o *clock*. Outro seria aquele que indica o início de uma palavra, necessário para iniciar o estado interno dos operadores para o processamento de uma nova palavra.

Um exemplo de operador seria um somador bit-serial, conforme mostra a Figura 6.4. Nela podem ser observados sinais de entrada de dados *A* e *B*, de controle *Ctrl LSB* e de saída *S*. Este operador é importante pois é a base para outros operadores e portanto é quem determina o período de *clock* mínimo do sistema {Lim e outros, 1993}. Os outros operadores são construídos a partir de composições deste.

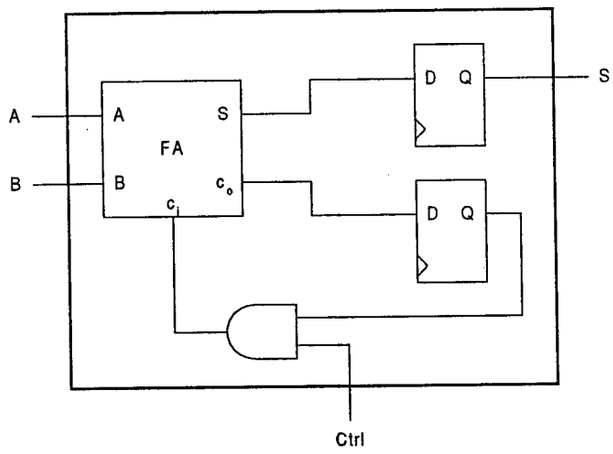


Figura 6.4: Somador bit-serial

Outros tipos de somadores podem ser utilizados no lugar do somador *ripple-carry*, mas como o *carry* no somador *ripple-carry* bit-serial faz uso de *pipeline*, não existe grande diferença entre o desempenho de cada implementação.

O controle dos operadores bit-serial pode ser bastante simplificado se os operadores exibirem latência constante. Dessa forma é simples garantir que os LSBs dos dados atinjam os operadores simultaneamente, basta distribuir convenientemente elementos de atraso pelo *datapath*. Isto simplifica também a geração dos sinais de controle. Em algumas implementações de algoritmos de processamento de sinais apenas o *clock* e um sinal de controle de início de

palavra são necessários {Trainor e outros, 1998}. Isto é muito interessante pois estes sinais de controle, que tem que ser distribuído por todos os operadores com o mínimo de atraso e *skew*, podem fazer uso dos escassos recursos de roteamento de alta velocidade presentes em algumas arquiteturas de FPGA. Por exemplo, o FPGA XC6200 oferece 4 rotas de distribuição global de sinais, 2 dedicadas ao roteamento do *clock* e *clear* e 2 para sinais de uso geral, ideais para conduzir os sinais de controle.

O processamento bit-serial pode ser utilizado associado à aritmética distribuída, como em {Woods e outros, 1998} ou mesmo com outros sistemas de representação numérica, como {Turner e outros, 1995} e {Panneerselvam e outros, 1995}, que utilizam-se da representação numérica CSD, onde cada dígito binário é sinalizado {Hwang, 1979}. Este sistema de numeração é utilizado para aumentar a velocidade de operação em processamento bit-paralelo, mas neste caso a vantagem é reduzir a área da implementação.

Uma implementação de um FIR de 5 taps com palavras de 8 bits para o XC6200 utilizando a arquitetura bit-serial atinge um desempenho de 1.87 M amostras/s. A área desta implementação é de 400 blocos lógicos {Xilinx, 1998}. É uma implementação com produto AT satisfatório, mas a implementação trabalha apenas com números sem sinal, o que não é possível para a implementação de filtros FIR para imagens. Se fosse necessário acrescentar o bit de sinal, a performance cairia 10% aproximadamente. E também esta taxa de processamento não permitira a compressão de vídeo em tempo real.

6.6. Somadores

Somadores são os blocos fundamentais das unidades lógico-aritméticas, e são determinantes no desempenho de algoritmos de processamento de sinais. A performance dos somadores é importante pois são empregados em todos os algoritmos de processamento de sinal, e também em conjunto com outras estratégias, como aritmética distribuída.

Existe um grande número de estudos sobre implementações VLSI de somadores. Um deles, feito por {Zimmermann, 1999} comparando desempenho e área de vários somadores visando implementação VLSI utilizando *standard-cells* (células de granularidade fina de complexidade similar àquelas da FPGA XC6200), aponta que o somador com o melhor produto AT é o somador *carry-skip*.

Estes resultados não são integralmente verdadeiros quando a implementação é feita em

FPGAs, devido às peculiaridades das arquiteturas destes dispositivos, principalmente limitações de recursos de roteamento {Zimmermann, 1998}. Por isso a regularidade da arquitetura do somador também é um fator a ser considerado na sua avaliação. Algumas famílias de FPGAs possuem recursos dedicados à propagação rápida do *carry*, permitindo a implementação eficiente de somadores *ripple-carry*. Na família Altera FLEX 10K {Altera, 1998} este recurso é chamado "*Carry Chain*", e o atraso na propagação do *carry* é de apenas 0.2 ns {Altera, 1998}. Na família Xilinx XC4000 este recurso é chamado de "*Fast Carry Logic*" e oferece atrasos da mesma ordem que aqueles da família FLEX 10K {New, 1996a}. Estes recursos tornam dispensável o uso de esquemas de aceleração de *carry* para somadores de até 24 bits {New, 1996b}. Isto é confirmado por {Xing e Yu, 1998}, que na Figura 6.5 mostra que somadores *carry skip* oferecem vantagens de velocidade apenas no caso de somadores maiores que 64 bits.

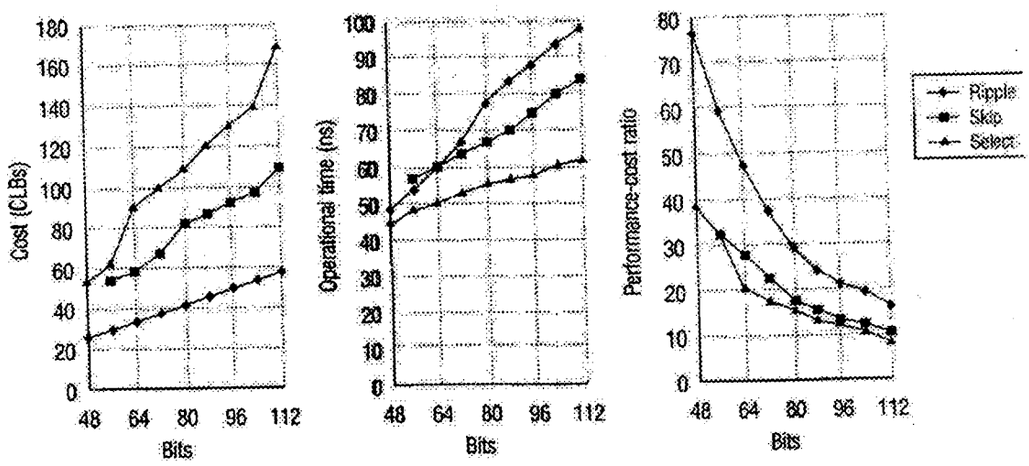


Figura 6.5: Área e performance de somadores implementados em XC4000 {Xing e Yu, 1998}

Algumas famílias de FPGAs, como a Xilinx XC6200 não oferecem recursos dedicados de propagação de *carry*, e por isso somadores de alta performance merecem ser considerados. Tanto {Xing e Yu, 1998} e {Zimmermann, 1999} apontam o somador *carry-skip* como de melhor compromisso entre velocidade e área, sendo que a análise de {Zimmermann, 1999} foi feita tendo como arquitetura de implementação uma muito similar à da família XC6200. Portanto este somador deve ser estudado para implementação em FPGAs sem recursos de propagação rápida de *carry*.

O somador *carry-skip* reduz o tempo necessário para propagar o *carry* utilizando uma lógica que avança o *carry* sobre um grupo de somadores consecutivos, economizando o tempo de propagação do *carry* dentro destes somadores.

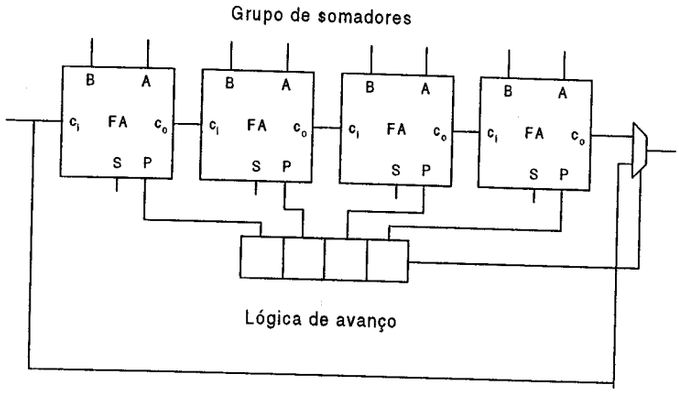


Figura 6.6: Somador *Carry-Skip*

Este somador se baseia no fato de que em um grupo de S somadores cujas entradas a_i e b_i satisfaçam $P_i = a_i \oplus b_i = 1, \forall i \in \{1..S\}$ o *carry* de saída é igual ao de entrada e pode ser avançado sobre o grupo de somadores {Koren, 1993}. Isto é útil se o produto $P_1\ P_{2...}\ P_S$ puder ser calculado e propagado mais rapidamente que o *carry* percorre os somadores do grupo. Isto é verdadeiro em uma implementação VLSI, onde se tem a liberdade de utilizar portas de várias entradas e lógica *wired*. FPGAs de granularidade grossa, como as baseadas em LUTs, também podem implementar funções de várias entradas com baixo atraso. Mas no caso de FPGAs de granularidade fina, cujos blocos lógicos são normalmente limitados a funções booleanas de 2 entradas, a implementação de funções de mais entradas é feita compondo estas funções menores. Este arranjo de portas simples pode gerar atrasos de propagação da mesma ordem daqueles envolvidos na propagação do *carry*, sem benefício algum para a velocidade do somador.

Como este tipo de somador apresenta caminhos redundantes para o *carry*, a determinação do caminho crítico feita automaticamente por meio de ferramentas EDA gera resultados incorretos pois ignora o caminho mais rápido do avanço do *carry*. E o cálculo de atraso feito por {Zimmermann, 1999} não pode ser aceito integralmente pois ele considera que portas XOR são compostas por portas AND e OR, e por isso tem o dobro de área e impõem o dobro do atraso que as outras portas. O que não é verdadeiro em se tratando de FPGAs de granularidade fina. Portanto o cálculo do atraso de propagação do *carry* deve ser feito analiticamente.

Quando a arquitetura da FPGA dispõe de multiplex 2:1, como é o caso da família XC6200, é vantajoso implementar o *full adder* como indicado na Figura 6.7. A porta AND

representa o circuito adicional necessário para implementar o avanço do *carry*. Se os tempo de atraso t_c e t_s são iguais, os dois caminhos impõe o mesmo atraso e o circuito de avanço não contribui para aceleração na geração do *carry*. Se estes tempos forem diferentes, então pode haver vantagem, que deve ser quantificada.

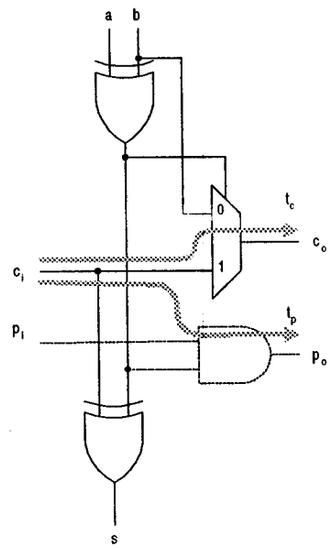


Figura 6.7: Célula básica do somador completo prevendo *carry-skip*

Seja t_g o atraso típico de um porta e t_c o atraso de *carry* de um único estágio somador. Para um grupo composto por S somadores *ripple-carry* o atraso t_r do *carry* na saída do grupo de somadores é:

$$t_r = t_c S$$

Considerando que a lógica de avanço também seja implementada com funções booleanas de 2 entradas em arranjo linear, como na Figura 6.7, o atraso t_a imposto pela lógica de avanço no grupo de S somadores é:

$$t_a = t_p S$$

Seja um somador *ripple-carry* com N estágios. O caso mais crítico na propagação de *carry* é quando ele é gerado no primeiro estágio e se propaga até o último. Este atraso é determinado por t_{rn} :

$$t_{rn} = t_c N$$

Considerando esta mesma situação para o somador *carry-skip*, o tempo máximo de

propagação do *carry* t_{sn} é determinado pela geração e propagação do *carry* no primeiro grupo de somadores *ripple-carry*, pelo seu avanço sobre os grupos de somadores intermediários através da lógica de aceleração do *carry*, e na sua propagação no último grupo, que ocorre do mesmo modo que no primeiro. Neste caso a lógica de aceleração de *carry* faz com que este percorra um caminho mais rápido nos estágios intermediários. Isto é ilustrado pela Figura 6.8, onde o somador *ripple-carry* utiliza 4 grupos de somadores *carry-skip*.

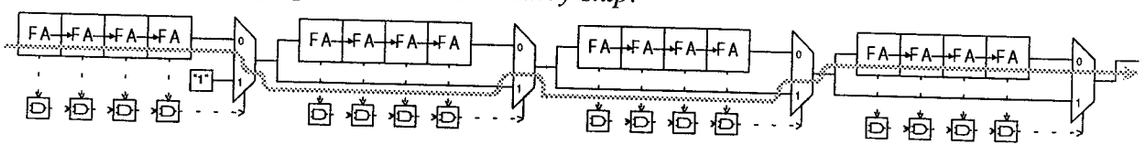


Figura 6.8: Pior caso de propagação do *carry* em somador *carry-skip*

Considerando o tempo t_r igual ao atraso t_m do multiplexador e o tempo t_p igual ao atraso t_g de uma porta genérica de 2 entradas:

$$
\begin{aligned}
t_{sn} &= t_r S + \left(\frac{N}{S} - 2\right)(t_a + t_g) + t_r S \\
&= 2St_m + \left(\frac{N}{S} - 2\right)(S+1)t_g \\
&= 2t_m S + Nt_g - 2St_g + Nt_g S^{-1} - 2t_g
\end{aligned}
$$

$$t_{sn} = 2(t_m - t_g)S + (N-2)t_g + Nt_g S^{-1} \qquad (6.6)$$

O tamanho S_o do grupo de somadores que acarreta o mínimo t_{sn} pode ser determinado derivando (6.6) e é.

$$S_o = \sqrt{\frac{N}{2}}\sqrt{\frac{t_g}{t_m - t_g}}$$

Seja R a relação entre o atraso do *carry* e da lógica de avanço.

$$R = \frac{t_m}{t_g}$$

Se $R \rightarrow 1$, $S_o \rightarrow \infty$ o que é impossível. Para manter as condições propostas no equacionamento de (6.6) deve haver pelo menos 3 grupos de somadores, logo $S \leq N/3$. Assim:

$$S = \min\left(\text{round}\left(\sqrt{\frac{N}{2}}\sqrt{\frac{1}{R-1}}\right), \left\lfloor \frac{N}{3} \right\rfloor\right)$$

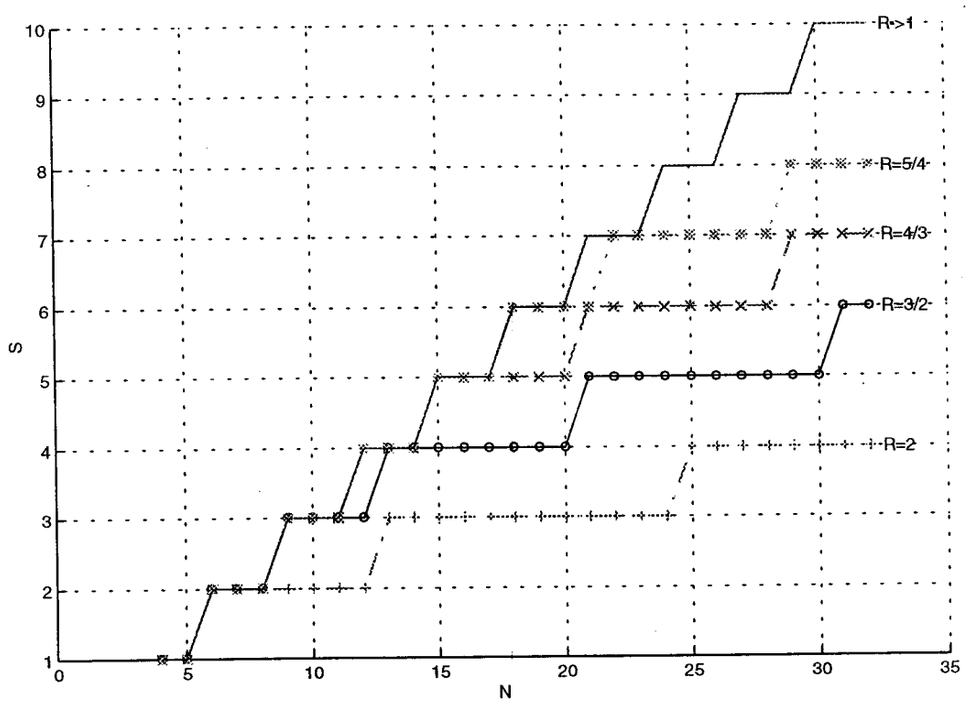


Figura 6.9: Tamanho ótimo dos grupos de somadores em somador *carry-skip*

Utilizando estes tamanhos ótimos de grupos somadores pode-se estimar o ganho de performance oferecido pelo somador *carry-skip* para FPGAs de granularidade fina através da relação t_{rn}/t_{sn}. A Figura 6.10 mostra que maiores ganhos podem ser obtidos quanto maior for a diferença entre o atraso do *carry* e o atraso da lógica de avanço. Também deve ser observado que nesta estimativa não se levou em consideração os atrasos de roteamento dos sinais, mas a Equação (6.6) ainda assim pode ser utilizada para se estimar a aceleração, desde que se acrescentem os tempos de roteamento no cálculo de R. Isto provavelmente reduzirá seu valor, pois a Figura 6.7 mostra que o roteamento *carry* e da lógica de avanço seguem caminhos semelhantes.

Um arranjo em árvore das portas que implementam a lógica de avanço tornaria o atraso desta lógica proporcional a $\log_2(S)$, em lugar de proporcional a S, como o arranjo linear equacionado. Mas como mostra a Figura 6.9, os tamanhos de grupo são da ordem de 2^2, e por isso o arranjo em árvore apresenta um ganho de velocidade reduzido. E este ganho pode ser facilmente anulado pelo roteamento mais complexo deste arranjo, principalmente se os tempos de roteamento são da mesma ordem dos tempos de propagação das portas.

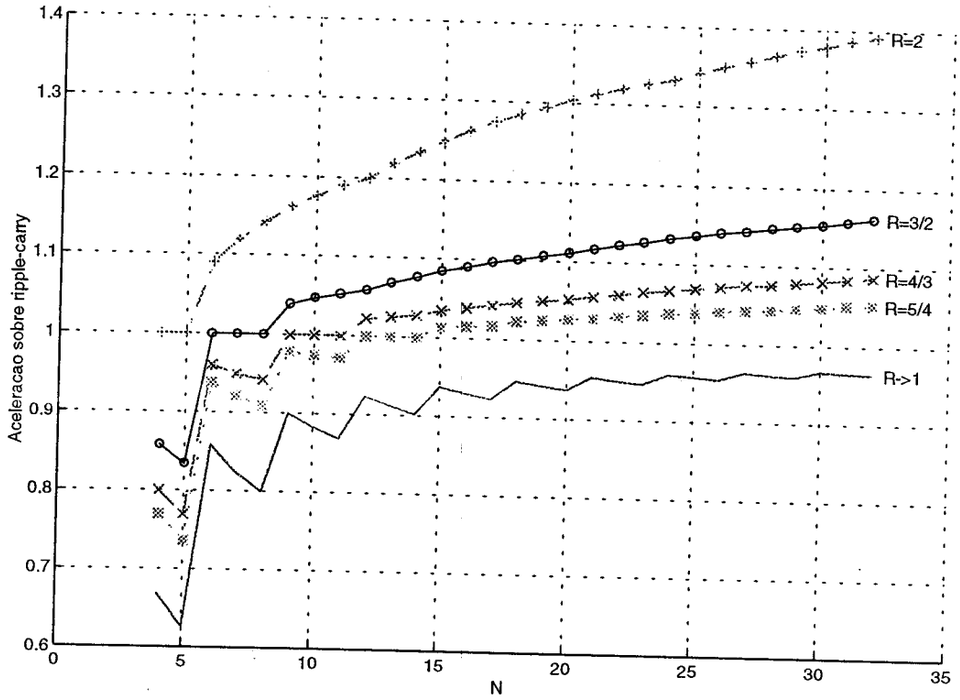


Figura 6.10: Ganho de performance do somador *carry-skip* sobre o *ripple-carry*

A segundo o manual do FPGA XC6200 {Xilinx, 1998}, a relação entre o atraso do multiplexador 2:1 e uma porta lógica normal é 3 ns / 2 ns no pior caso. Os tempos de roteamento de uma trilha local são de proximadamente 0.7 ns. Assim a relação R seria de 1.37, ou seja, aproximadamente 4/3. Isso significaria que o somador *carry-skip* traria uma aceleração na soma de 16 bits de 5% a 10%, o que não compensa a maior complexidade deste somador. Tempos melhores podem ser obtidos através da divisão da soma em vários estágios de pipeline.

6.7. Considerações Finais

Devido a complexidade do circuito a ser implementado, a forma de estruturação escolhida foi a de macros estruturadas parametrizáveis. Esta solução permite a elaboração de entidades parametrizáveis de uso geral, como somadores, multiplicadores, registradores, formando uma biblioteca. A parametrização é utilizada para configurar as entidade da biblioteca com as características necessárias para a aplicação. Entre estas características estão o tamanho da palavra, a constante de comparação em comparadores, a constante de multiplicação em multiplicadores, o número de estágios de *pipeline* em um estágio de filtro FIR, etc. Por serem de

uso geral, estas macros podem formar uma biblioteca de entidades para outros projetos.

Devido ao fato de filtros biortogonais binários serem compostos por coeficientes inteiros, a utilização da aritmética distribuída não é muito interessante. Já no caso dos filtros ortonormais, com coeficientes irracionais, pode ser uma alternativa interessante e sua utilizacão deve ser analisada em função dos coeficientes da wavelet em questão.

A arquitetura do FPGA XC6200 com seus recursos de roteamento escassos é bastante apropriada à implementações bit-serial. O fato de barramentos serem de apenas um bit simplifica bastante a interligação das entidades de processamento. Mas, como mostrado em {Xilinx, 1997}, o desempenho não é viável para aplicações de tempo real, a menos que se utilize mais de um sistema de compressão wavelet operando em paralelo, como sugerido em algumas arquiteturas de processamento wavelet {Chang e outros, 1997} {Parhi e Nishitani, 1993} {Hung e outros, 1998} {Chakrabarti e Mumford, 1999} {Paek e Kim, 1998} {Yu e Chen, 1999b} {Grzeszczak e outros, 1996} {Yu e Chen, 1997a} {Chakrabarti e Vishwanath, 1995} {Knowles, 1990} {Vishwanath e outros, 1995}.

7. Implementação

7.1. Introdução

Este capítulo discute e descreve a implementação em hardware de um sistema para executar a transformada wavelet em tempo real sobre imagens dinâmicas de dimensão da ordem 256x256. O alvo da implementação é o FPGA Xilinx XC6200. Devido a sua arquitetura de granularidade fina, rica em flip-flops, com poucos recursos de roteamento, e ausência de ferramentas de desenvolvimento de alto nível, são necessárias algumas considerações sobre o fluxo de projeto adotado. Estas informações são úteis para outros tipos de FPGA de granularidade fina.

Uma vez que foi adotada a criação de macros parametrizáveis como estratégia de estruturação do projeto, alguns componentes úteis para aplicações de processamento digital de sinais foram desenvolvidos. Eles serão descritos e utilizados nas implementações da transformada wavelet apresentadas na sequência.

Como o objetivo deste trabalho é o desenvolvimento de um sistema de transformada wavelet compacto para implementação em FPGAs limitados, serão discutidas e implementadas algumas arquiteturas com estas características. Os resultados das implementações são apresentados. Um sistema de processamento de um quadro completo utilizando estes processadores wavelet também é descrito.

7.2. Fluxo de Projeto

Devido as características desta família de FPGAs as metodologias de desenvolvimento utilizadas em outras arquiteturas {Gschwind e Salapura, 1995a} não se aplicam a ela. O ciclo de desenvolvimento em uma FPGA convencional é composto das seguintes etapas

- Criação
- Síntese
- Geração de *netlist*
- Posicionamento e Roteamento (P&R)
- Simulação Detalhada

Normalmente as etapas de criação, síntese e simulação são as que mais consomem o tempo do projetista, enquanto as outras são efetuadas automaticamente pelas ferramentas de projeto eletrônico. Eventualmente é necessária a intervenção do projetista no processo de P&R quando se deseja alguma otimização.

Uma vez que a ferramenta de posicionamento e roteamento XACT6000 {Xilinx, 1997} {Xilinx, 1998} aceita parâmetros de posicionamento inseridos no *netlist*, é possível efetuar o posicionamento ótimo manualmente e inserir estes parâmetros de posição no *netlist*, permitindo que o circuito seja alterado ou mesmo utilizado em outro projeto sem necessidade do P&R manual.

Explorando este recurso podem ser construídas bibliotecas de entidades parametrizáveis e pré-posicionadas, chamadas de macros. Os parâmetros de posicionamento devem ser inseridos de alguma forma no *netlist*. A linguagem VHDL prevê meios de se acrescentar informações complementares, na forma de atributos definidos pelo usuário, ao circuito lógico descrito {Dewey, 1997}. Mas como ferramentas de síntese não-especializadas para uma determinada arquitetura de FPGA, como o Synopsys™, ignoram estes atributos na geração do *netlist*, outros meios devem ser encontrados.

Uma maneira seria utilizar a linguagem de *scripting* oferecida pela ferramenta de síntese para extrair os atributos da descrição do circuito e inserí-las no *netlist*. Para o Design Compiler da Synopsys™, o comando `set_attribute {add1} RLOC ''X1Y2'' -type string` inseriria no *netlist* o string "*X1Y2*" como valor do atributo *RLOC*, indicador da posição do componente *add1*.

Outra solução seria utilizar ferramentas de síntese específicas para a família XC6200, como o Velab {Grant, 1997}, um analisador VHDL e elaborador de *netlist* que aceita como entrada um subconjunto de estruturas VHDL'93 restrito àquelas utilizadas em descrição estrutural, e que é capaz de reconhecer os atributos de posicionamento. É um programa de domínio público, com versões para DOS e Solaris e muito mais acessível que ferramentas de síntese profissionais. Como macros são repetições regulares de estruturas simples e o Velab aceita as diretrizes *if-generate*, *for-generate* e *generic* da linguagem VHDL, as macros podem ser desenvolvidas e parametrizadas com eficiência. A desvantagem desta solução é que não se pode utilizar da conveniência da síntese lógica a partir da descrição comportamental em alto nível.

Para minimizar este problema, uma estratégia é dividir um projeto em entidades de

controle e *datapath*. Entidades de controle são estruturas irregulares (*random logic*), geralmente máquinas de estado, que no caso de arquiteturas orientadas para processamento de sinais, utilizam muito menos área na implementação que o *datapath*. Sendo assim, o posicionamento e roteamento automáticos das entidades de controle pode ser utilizados, pois mesmo tendo uma eficiência reduzida, o desperdício de área é não é significativo. Esta abordagem permite o emprego de ferramentas de síntese de alto nível para traduzir o modelo comportamental da entidade de controle para o nível de portas lógicas.

Ao contrário da entidade de controle, a entidade que contém o *datapath* é regular e sua área é muito maior. Por isso sua implementação é crítica e deve ser feita instanciando as entidades manualmente pré-posicionadas descritas nas macros.

A Figura 7.1 mostra o ciclo de desenvolvimento utilizado. Ele é semelhante ao proposto por {Hartenstein e outros, 1998}.

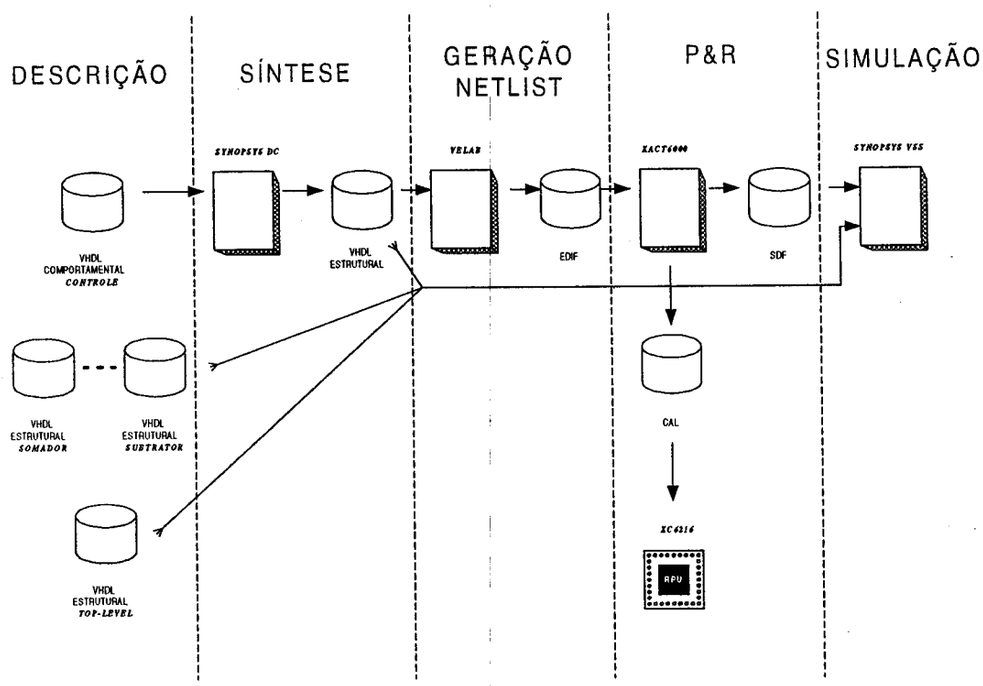


Figura 7.1: Esquema do fluxo de projeto

A fase de descrição consiste na descrição estrutural das macros do *datapath* e da entidade de topo da hierarquia (*top-level*), que integra todas as entidades do projeto, e na descrição comportamental da unidade de controle. Esta entidade deve ser sintetizada, e a ferramenta utilizada para isso foi o Synopsys Design Compiler 1999.10, com biblioteca de tecnologia alvo específica para o XC6200 e compatível com VITAL. Esta ferramenta pode gerar

netlists no formato EDIF diretamente, mas foi escolhida a geração de *netlist* em formato VHDL estrutural. Juntamente com todas as outras entidades envolvidas no projeto, esta descrição é processada pelo Velab. Este programa gera o *netlist* no formato EDIF, que é lido pela ferramenta XACT6000. Ela posiciona e roteia as entidades respeitando as informações de posicionamento das macros. Ela gera um arquivo para programação da FPGA, e um arquivo no formato SDF com informações exatas sobre os atrasos de roteamento. Este arquivo é lido pelo simulador (*backannotation*) para a simulação detalhada, com os atraso exatos utilizados em lugar dos parâmetros estimados da biblioteca VITAL.

O projeto utiliza o simulador Synopsys™ VSS 1999.10 para validação. A validação se inicia com a inspeção de formas de onda geradas na simulação, como mostra a Figura 7.2 e Figura 7.3. Este procedimento permite avaliar o comportamento do circuito em um conjunto limitado de dados. Para a validação do projeto, foi utilizada a metodologia de *testbench* {Biederman, 1997}, onde se compara as saídas do circuito sintetizado com as saídas do *testbench*. Isto permite a validação automática extensiva, utilizando uma grande quantidade de vetores de teste.

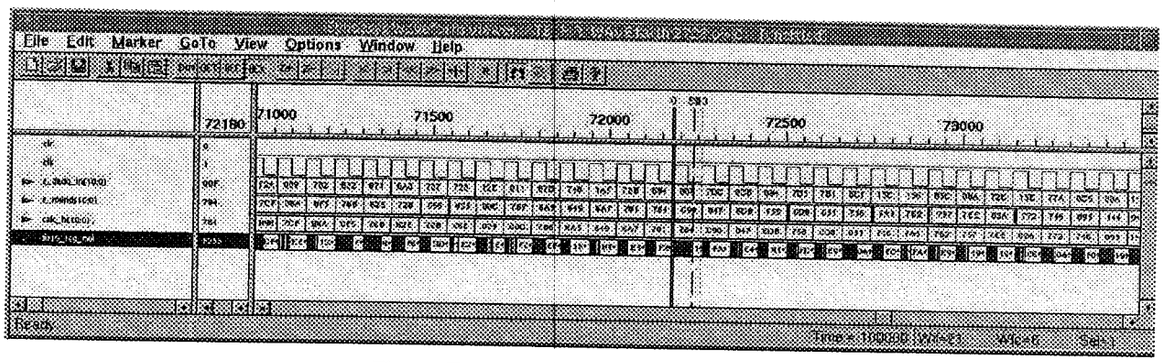


Figura 7.2: Simulação de um filtro FIR

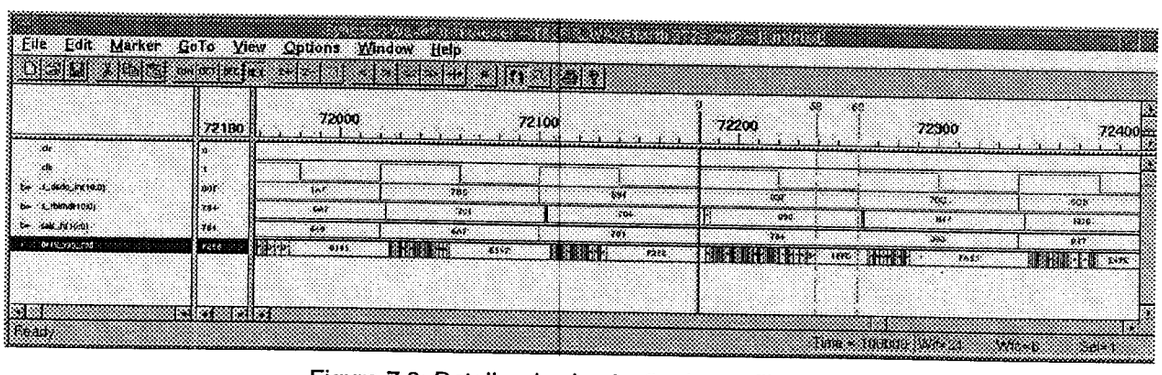


Figura 7.3: Detalhe da simulação de um filtro FIR

7.3. Considerações de Implementação

A implementação da lógica combinacional é síncrona, utilizando um único *clock* de fase única. Isso facilita muito a análise de performance e torna a operação do circuito mais previsível. Também permite que se disponha registradores horizontalmente sem desperdiçar linhas de roteamento de *clock*, embora esta liberdade não tenha sido aproveitada. Os registradores foram dispostos na vertical, com os barramentos correndo horizontalmente.

Como o *clock* é único, foi necessário implementar uma habilitação de *clock* em alguns flip-flops. Uma vez que a família de FPGA XC6200 não oferece flip-flops com este recurso, a estratégia adotada foi utilizar habilitação de *clock* combinacional. Este circuito pode ser implementado utilizando a porção combinacional do bloco lógico que contém o flip-flop, não exigindo recursos adicionais. Apenas é necessária uma linha de roteamento para distribuir a habilitação para todos os pares multiplexadores/flip-flops do registrador, mas como ela segue a disposição do registrador (vertical), ela não compete por roteamento com os sinais do barramento. Foi evitado o uso de *gated clocks*, por introduzirem atrasos na transição do *clock* (*clock skew*) que poderiam gerar problemas de meta-estabilidade {Wakerly, 1994}, além de pulsos espúrios gerando transições indesejadas {Chan e Mourad, 1994}.

Sempre que possível foi utilizado *pipelining*, para conseguir maior de velocidade de operação, uma vez que muitas vezes ele não exige nenhuma lógica extra. Outro esforço feito para minimizar atrasos foi a duplicação de componentes de área não significativa e que tivessem suas saídas utilizadas em circuitos muito distantes entre si. Esta técnica consome alguma área de lógica a mais em troca da redução do atraso de roteamento {Trainor e outros, 1998}, conforme mostra a Figura 7.4:

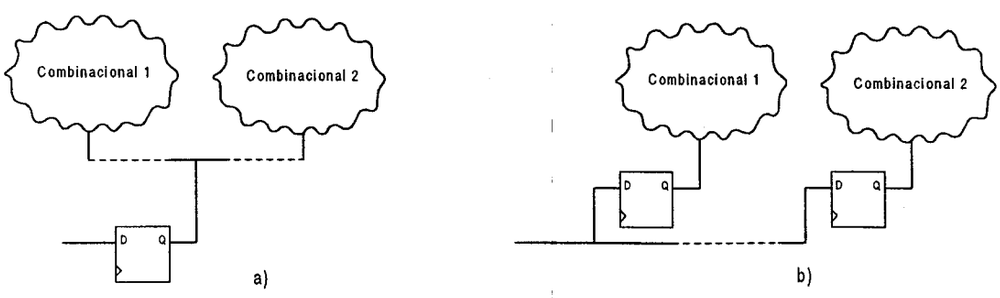


Figura 7.4: Utilização de componentes otimizando a) área b) atraso de roteamento

O uso de codificação *one-hot* nas máquinas de estado {Wakerly, 1994} foi feito sempre

que possível, por ser mais adequado para FPGAs, normalmente ricas em flip-flops {Gschwind e Salapura, 1995a}. Contadores de poucos bits foram substituídos por registradores de deslocamento quando o seu objetivo era apenas gerar um sinal de fim de contagem. A Figura 7.5 mostra quando utilizar ou não esta estratégia.

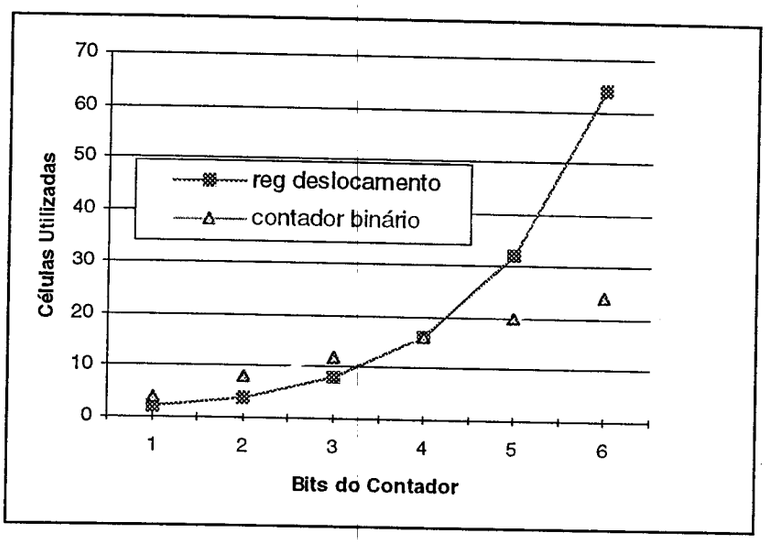


Figura 7.5: Tamanhos de contador e registrador de deslocamento

7.4. Bibliotecas Parametrizáveis

Foram desenvolvidas algumas entidades para este projeto que, pelo fato de serem parametrizáveis, podem ser utilizadas em outras aplicações. A maior parte delas é orientada para processamento de sinais, mas também existem algumas com outras funções, como implementar acesso a memória. Elas foram desenvolvidas em VHDL, respeitando as limitações do Velab {Grant, 1997}, um programa de domínio público com versões para plataforma DOS e UNIX (Solaris, SunOS 4.1 e Linux). Como as equações para gerar o posicionamento dos blocos lógicos podem ser complexas em alguns casos, foi necessário utilizar o recurso do Velab que permite inserir no código VHDL comandos para invocar a execução de programas externos. Estes programas são executáveis, desenvolvidos em qualquer linguagem, que devem receber os parâmetros por meio da *standard input* e retornar o resultado por meio do *standard output*.

Uma lista completa de entidades se encontra nos Apêndices, com dados de área e desempenho das entidades. As entidades mais importantes são:

7.4.1. ADD_TAM_S

Esta entidade é um somador de números binários com sinal, na representação complemento-2. Ele é um *ripple-carry* simples, que oferece a melhor relação performance/custo. Não utiliza *pipeline*. Os tamanhos das suas entradas e da saída são parâmetros independentes.

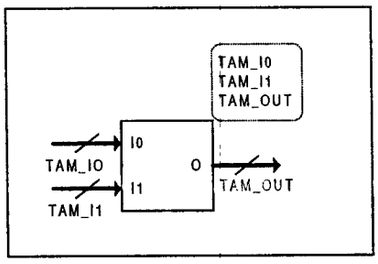


Figura 7.6: Diagrama do somador

7.4.2. MUL_ADD_S

Esta entidade é um multiplicador por constante implementado por meio de um arranjo de somadores ADD_TAM_S. Ele utiliza uma rede de somadores para implementar a multiplicação. Este arranjo apenas é interessante se o valor da constante de multiplicação puder ser representado com um número pequeno de bits em 1, pois a entidade utiliza 1 somador para cada um destes bits, com exceção do primeiro.

A entidade não possui *pipeline* interno. Além da constante de multiplicação ela pode ser parametrizada também pelo tamanho da entrada e da saída.

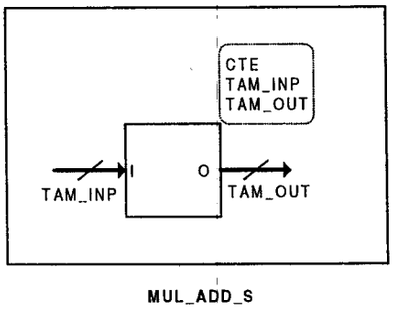
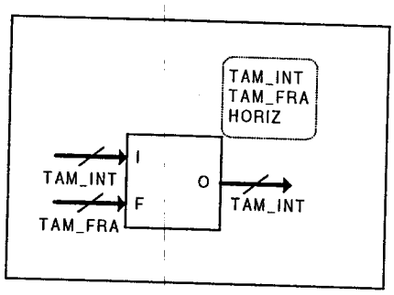


Figura 7.7: Diagrama do multiplicador por constante

7.4.3. ROUNDER

Esta entidade arredonda um número binário de ponto fixo, com uma parte inteira e uma

parte fracionária. O arredondamento é feito somando o bit mais significativo da parte fracionária à parte inteira. Esta entidade opera também com números binários com sinal.

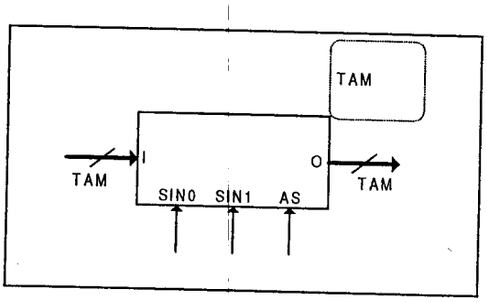


ROUNDER

Figura 7.8: Diagrama da entidade de arredondamento

7.4.4. SAT_S

Esta entidade implementa aritmética de saturação. Ela trunca números binários na representação complemento-2 quando ocorre *overflow* ou *underflow*. Esta entidade deve ser utilizada após um estágio de soma ou de subtração. Ela faz a saturação do sinal de entrada (I) com base nos sinais dos 2 operandos (SIN0 e SIN1), no sinal do resultado da operação, e no tipo de operação efetuada (AS=1, subtração).



SAT_S

Figura 7.9: Diagrama da entidade de saturação

7.5. Filtros FIR

Uma família de entidades foi desenvolvida para implementar filtros FIR. Estas entidades tem o posicionamento definido de modo a simplificar o cascateamento entre os estágios do filtro.

As entidades que implementam estes filtros se utilizam da entidade MUL_ADD_S para multiplicação e ADD_TAM_S para soma. Diferentemente das outras entidades anteriores, esta possui *pipeline* interno. Para construir um filtro FIR basta instanciar estas entidades com os

parâmetros desejados. Foram desenvolvidos 3 tipos de estágios para filtros FIR. Todos estes filtros são parametrizados pela constante de multiplicação (CTE), pelo sinal da constante de multiplicação (CTE_NEG=1 indica sinal negativo), pelo número de registros de pipeline após o multiplicador (NUM_REG), pelo tamanho da entrada de dados (TAM_INP) e pelo tamanho da palavra na saída do multiplicador (TAM_MUL).

7.5.1. FIR1_INI

Esta entidade implementa o estágio inicial do FIR. Por ser o primeiro estágio, ele não necessita da entidade somadora, e por isso economiza alguns recursos se comparada com o estágio de FIR convencional.

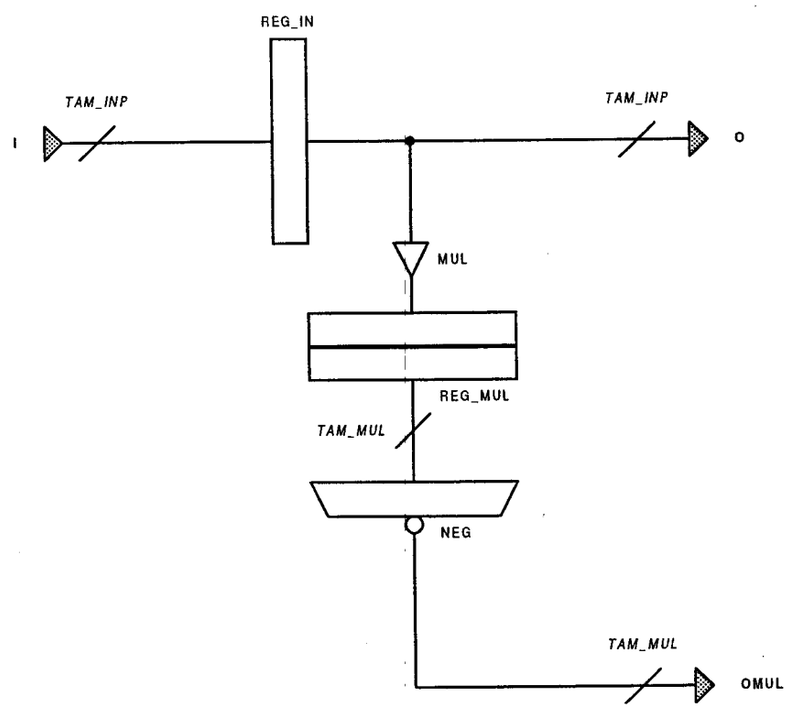


Figura 7.10: Estrutura do estágio inicial do filtro FIR

7.5.2. FIR1

Esta entidade implementa um estágio de filtro FIR intermediário, com o somador. Ela pode ser parametrizada também no tamanho da entrada do somador e no tamanho da sua saída.

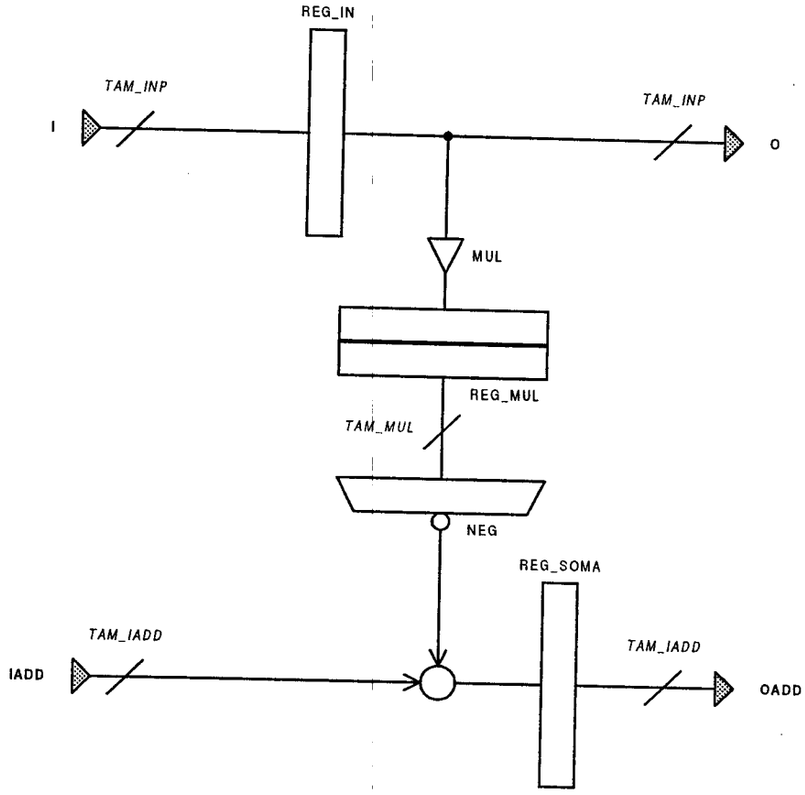


Figura 7.11: Estrutura do estágio intermediário do filtro FIR

Um exemplo de utilização destas entidades na construção de um filtro FIR completo e com *pipeline* extensivo pode ser visto a seguir.

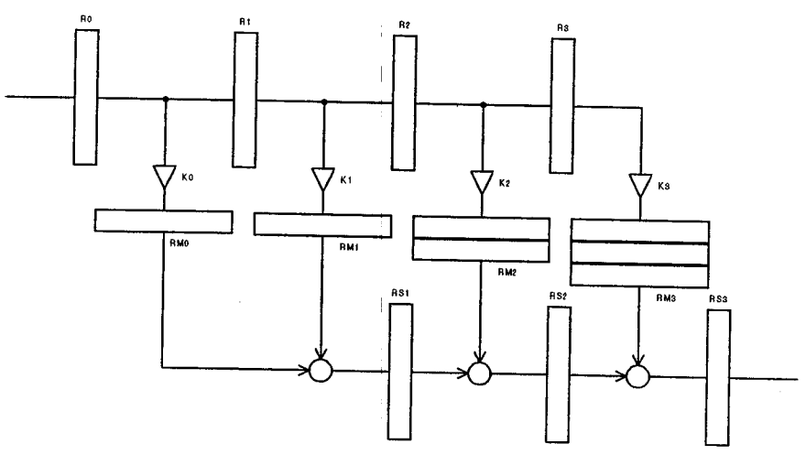


Figura 7.12: Construção de filtro FIR utilizando as entidades FIR_INI e FIR1

7.5.3. FIR2

Esta entidade é otimizada para implementar filtros FIR simétricos. Neste caso pode-se

aproveitar a simetria para reduzir o número de multiplicadores, que é o estágio mais lento e que ocupa maior área. Por isso foi desenvolvido um estágio de filtro FIR que explora esta simetria. O cascateamento dos filtros exige mais recursos de roteamento que no caso do estágio FIR não-simétrico, mas como as novas rotas correm em direção contrária, apenas se aproveitam recursos de roteamento que eram desperdiçados. Assim o cascateamento ainda é simples.

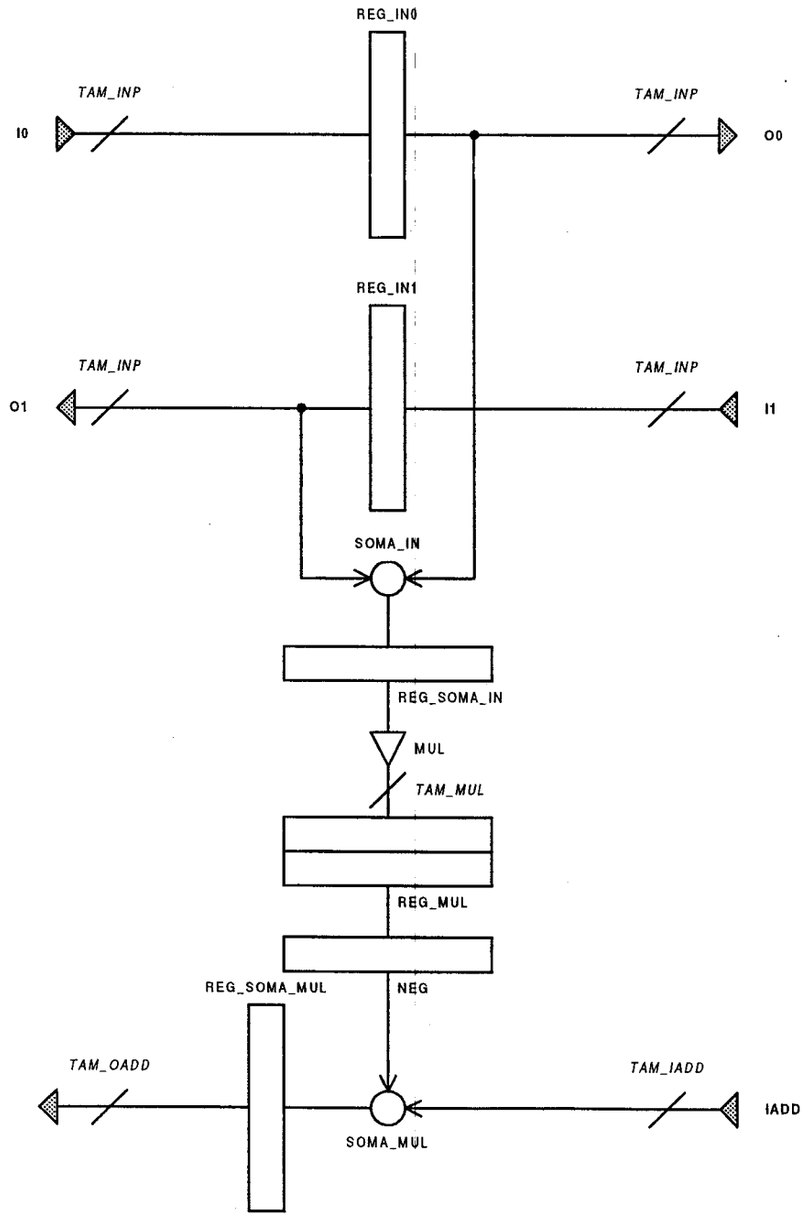


Figura 7.13: Estrutura do filtro FIR simétrico

7.6. Implementação da Transformada Wavelet

A transformada wavelet deve ser implementada em FPGAs de pequena capacidade. Por isso a arquitetura para a transformada deve ser a mais econômica, mas sem prejuízo do desempenho.

Devido ao tamanho limitado do FPGA XC6216, não é possível implementar nele todas as etapas de um sistema de aquisição e compressão de imagem. Por isso optou-se em utilizar o FPGA como um coprocessador dedicado à transformada wavelet, operando em paralelo com um processador clássico. A granularidade da tarefa do coprocessador pode ser reduzida a ponto de processar apenas um *pixel* enviado pelo processador, ou pode ser mais elevada, processando independentemente quadros completos obtidos em um *buffer*. Ao processador ficaria a tarefa de carregar este *buffer* com os dados a serem transformados, obtidos por meio dos dispositivos de entrada ligados ao processador, e, ao término do processamento do quadro, ler o resultado e eventualmente executar algum processamento adicional. O quadro geral do processamento está na Figura 7.14.

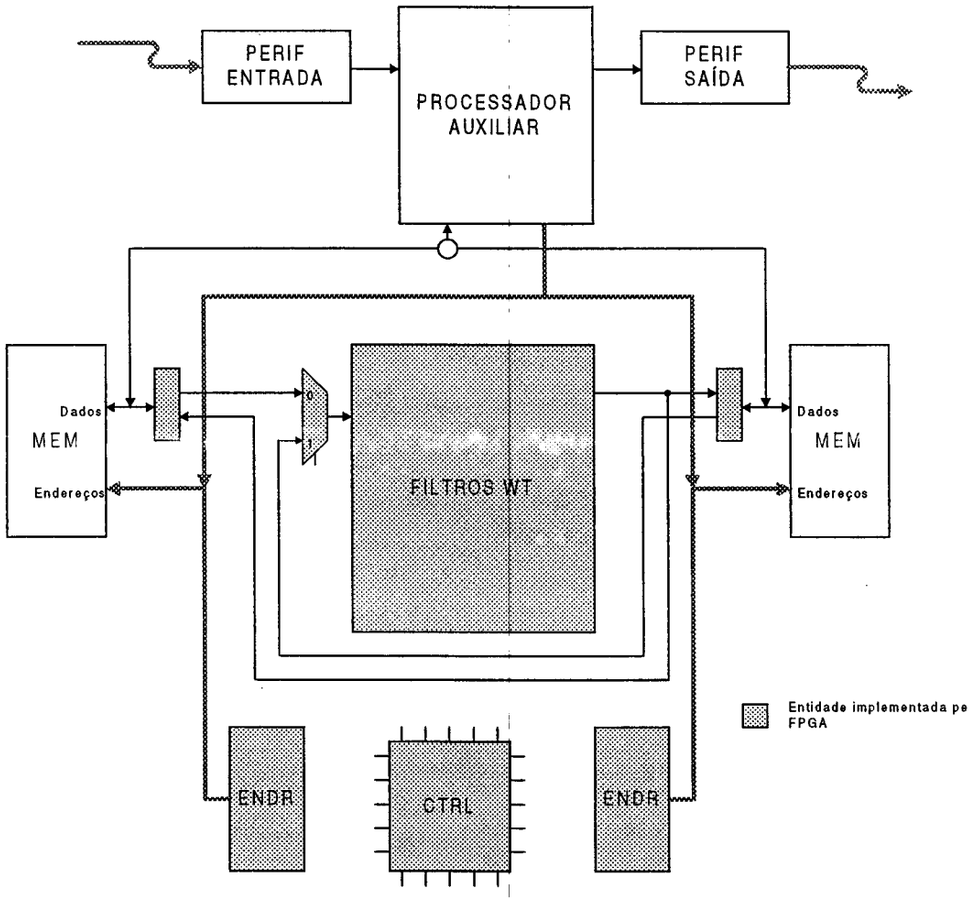


Figura 7.14: Diagrama geral do sistema de transformada wavelet

Caso se deseje apenas um coprocessador de granularidade fina, basta implementar no FPGA os filtros da transformada wavelet desejada. É uma solução simples, mas não muito eficiente, pois o processador auxiliar fica incumbido de alimentar o coprocessador *pixel* por *pixel*, o que impede que ele se dedique a outras tarefas. Uma melhor divisão de tarefas é implementar um coprocessador de granularidade mais elevada, que possa processar todo um quadro da imagem. Para isso deve-se implementar no FPGA, além dos filtros da transformada, também unidades de acesso à memória e uma lógica de controle para comandar o processamento nas várias escalas e nas duas dimensões. Para manter os filtros sempre operando, é interessante que existam dois *buffers*, permitindo leitura e escrita simultâneas.

Existem várias arquiteturas otimizadas para implementação da WT bidimensional, mas a maior parte delas é voltada para a implementação VLSI, sem restrições de área. Isso se reflete no grande número de somadores e multiplicadores exigidos. A Tabela 7.1 lista a quantidade de multiplicadores e somadores de 16 bits utilizados nestas arquiteturas propostas, além de classificar a entidade de controle utilizada.

Tabela 7.1: Complexidade de várias arquiteturas de wavelet propostas

Ref	Mul	Soma	Ctrl
{Parhi e Nishitani, 1993} (folded)	16	12	complexo
{Parhi e Nishitani, 1993} (digit)	14	12	simples
{Vishwanath e outros, 1995}	24	24	simples
{Chang e outros, 1997}	14	12	simples
{Yu e Chen, 1997a}	8	11	simples

É bastante claro que estas arquiteturas não podem ser implementadas no XC6216. Por exemplo, dois multiplicadores por constante de 16 bits implementado pela entidade MUL_ADD_S ocupam aproximadamente ¼ da capacidade do dispositivo. Uma solução seria utilizar a estratégia bit-serial para implementar estas arquiteturas. Outra seria utilizar arquiteturas mais simples.

A forma mais simples de se implementar a WT bidimensional é através da transformada separável canônica, primeiro processando com os filtros H_0, H_1, G_0 e G_1 as linhas (ou colunas) e depois as colunas (ou linhas) separadamente. Esta abordagem canônica apresentam maior latência e é menos eficiente que as arquiteturas acima, mas é melhor adaptada às restrições de espaço.

Outra maneira de se reduzir o espaço destas implementações é utilizar filtros biortogonais com coeficientes binários. Existem alguns com bom desempenho para compressão, e como seus coeficientes são inteiros, as multiplicações são eficientemente implementadas pela entidade MUL_ADD_S. A multiplicação de uma palavra de 16 bits por uma constante com 3 bits em 1 (como 11 por exemplo) utiliza apenas 3% da capacidade do dispositivo. Por isso a implementação canônica da WT com coeficientes binários é a opção preferida.

Um outro conjunto de filtros bastante adequado às condições de restrição de área são os filtros para a wavelet Daubechies-4. Apesar de serem ortonormais, com coeficientes irracionais, se eles forem multiplicados por $16\sqrt{2}$ passam a ser muito próximos de números inteiros, com erros de arredondamento de no máximo 0.072. Os coeficientes D4 passam a ser 3, 5, -19 e 11, o que permite uma implementação eficiente. Mas estes coeficientes possuem uma característica ainda mais interessante: 11=8+3, 19=16+3.

Considerando esta propriedade, {Lewis e Knowles, 1991} sugeriram uma arquitetura para reduzir a complexidade da implementação desta transformada. Esta arquitetura prescinde completamente de multiplicadores, utilizando apenas somadores na sua implementação.

7.6.1. Wavelet D4 Sem Multiplicadores

Esta implementação utiliza a arquitetura de {Lewis e Knowles, 1991}. É uma arquitetura muito compacta, pois não utiliza multiplicadores. Também é eficiente, pois não há o descarte de um resultado a cada dois resultados gerados devido à subamostragem. A cada par de ciclos de *clock* dois resultados são gerados.

Os tamanhos das palavras e coeficientes são aqueles definidos pela análise estatística de erros. O tamanho da palavra é de 11 bits, incluindo o bit de sinal. O tamanho dos coeficientes varia de 2 a 5 bits.

7.6.1.1 Transformada Direta

O diagrama de fluxo de dados do processamento nesta arquitetura difere bastante daquele baseado nos filtros FIR. Ele é mostrado na Figura 7.15. A arquitetura que executa este processamento é representada na Figura 7.16 e Figura 7.17.

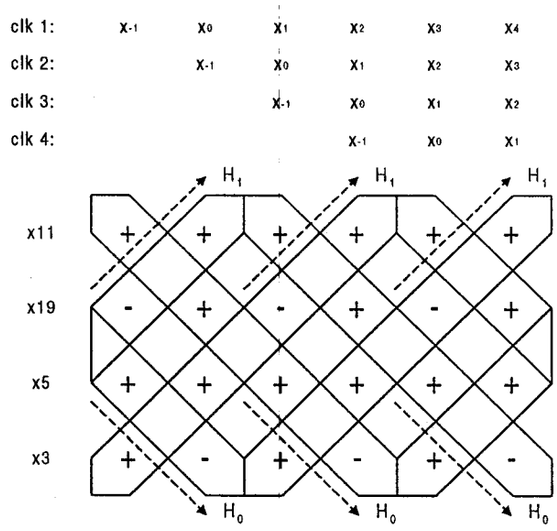


Figura 7.15: Fluxo dos dados para WT direta na arquitetura de {Lewis e Knowles, 1991}

Esta arquitetura é dividida 2 blocos. O primeiro bloco (Figura 7.16) multiplica a entrada pelos 4 coeficientes da wavelet, e os resultados destas multiplicações são processados conforme a Figura 7.15 pelo circuito da Figura 7.17.

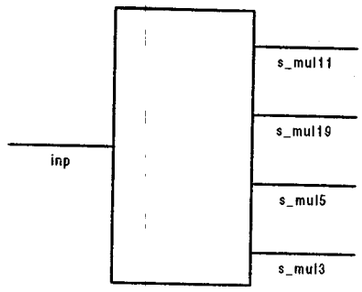


Figura 7.16: Multiplicador por 3, 5, 11, 19

O desempenho e área do multiplicador são muito satisfatórios, pois ele é implementado utilizando apenas somas em lugar de multiplicações. Mesmo as multiplicações por 11 e por 19, que são obtidas utilizando o resultado da multiplicação por 3 são relativamente rápidas. A Tabela 7.2 mostra o desempenho e área desta entidade para vários tamanhos de palavra.

Tabela 7.2: Área e desempenho da entidade multiplicador

TAM_INP	TAM_x3	TAM_x5	TAM_x11	TAM_x19	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	6	7	8	9	8	16	128	54	LOCAL	50,31
8	10	11	12	13	8	24	192	104	LOCAL	67,81
12	14	15	16	17	8	32	256	150	LOCAL	78,88

Os blocos ADD_B e ADD_D da Figura 7.17 são somadores/subtratores cuja operação é controlada pelos sinais sub_b e sub_d. Os blocos ADD_A e ADD_D efetuam aritmética de saturação através da entidade SAT_S.

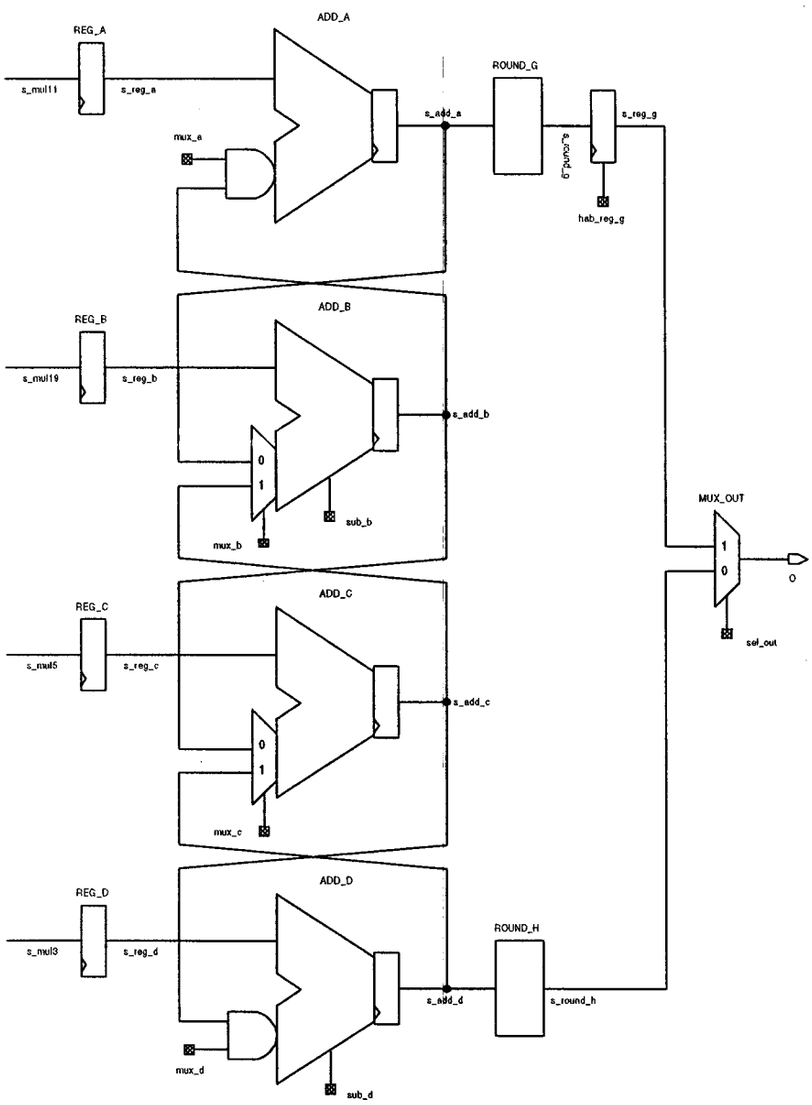


Figura 7.17: Arquitetura do processador para WT direta

Não está representado nas figuras, mas ainda existe uma entidade de controle para geração dos sinais de controle para as entidades da arquitetura de modo a implementar o fluxo da Figura 7.15. Mas a entidade de controle é muito simples, ocupando apenas 2 blocos lógicos do FPGA. Como pode ser visto, o circuito faz uso extensivo de *pipeline*, as saídas de todos somadores estão ligadas a registradores.

O período do *clock* para a arquitetura é determinado pela entidade mais lenta, que é o multiplicador. Por isso o período do *clock* deveria ser de no mínimo de 80 ns, ou seja, uma frequência de *clock* de 12.5 MHz. A estimativa do atraso de cada entidade é feita pela ferramenta XACT6000 após o posicionamento e roteamento. Mas os dados da Tabela 7.2 foram levantados

considerando a entidade independente, sem levar em consideração a conexão com outras entidades. Fazendo a análise de tempo em toda a arquitetura o resultado é um atraso maior, da ordem de 91 ns. Por isso a frequência de *clock* real é de 11 MHz.

O desempenho deste processador pode ser calculado considerando-se que a imagem possui *N pixels*. Para cada escala o número de *pixels* processados diminui de ¼. Para *E* escalas o número de *pixels* processados em apenas uma dimensão é de $N+N/4+N/16+...N/4^{E-1}$.

$$NPP1D(N,E) = N\sum_{i=0}^{E-1} 4^{-i}$$

Como a transformada separável exige o processamento em 2 dimensões, o número total de *pixels* processados em um quadro é $2.NPP1D(N,E)$. Para uma imagem de 256x256 *pixels*, o tempo para a transformação de um quadro com 4 escalas é 16 ms. Isso resulta numa taxa de 63 quadros processados por segundo.

A implementação desta arquitetura no FPGA XC6216 pode ser vista nas Figura 7.18 e Figura 7.19. A área ocupada pela implementação é uma região retangular de 37x37 blocos lógicos.

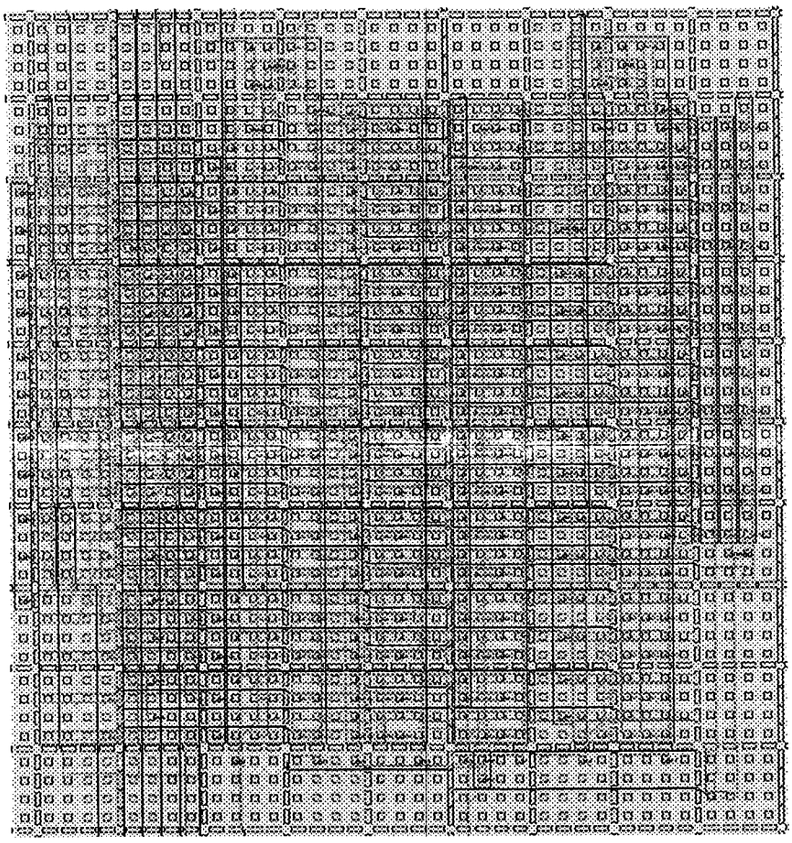


Figura 7.18: *Lay-out* da implementação da WT direta no FPGA XC6216

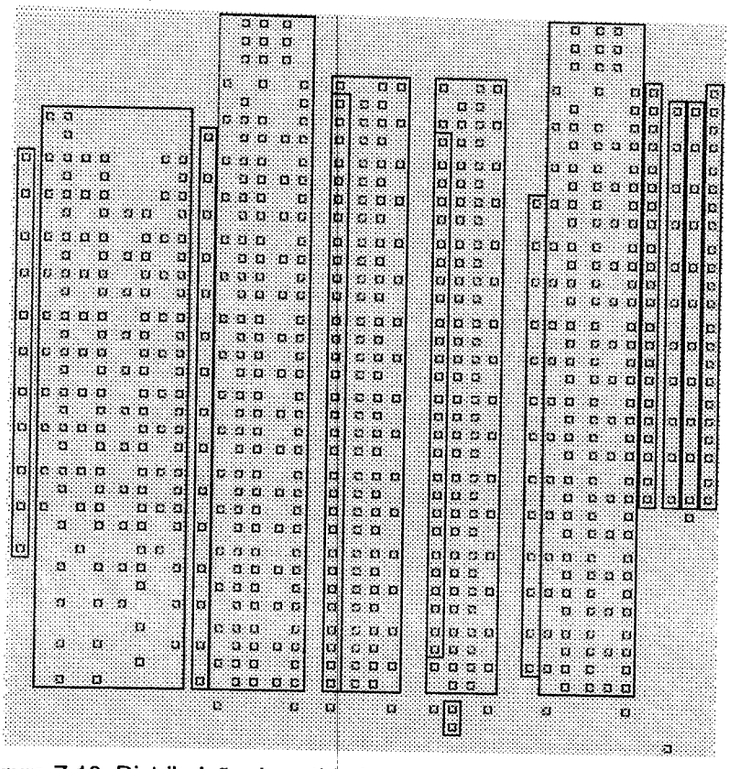


Figura 7.19: Distribuição de entidades da WT direta no FPGA XC6216

7.6.1.2 Transformada Inversa

O diagrama de sinal da transformada inversa nesta arquitetura é muito parecida com a transformada direta, como pode-se ver na Figura 7.20. Como a transformada é ortonormal, o que se altera entre a transformada direta e a inversa é apenas a posição e os sinais dos coeficientes.

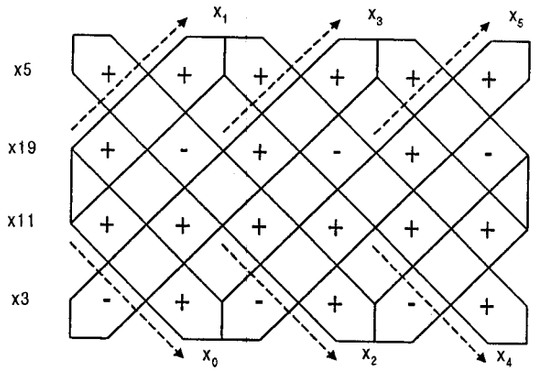


Figura 7.20: Fluxo dos dados da WT inversa na arquitetura de {Lewis e Knowles, 1991}

O diagrama da arquitetura para a WT inversa segundo {Lewis e Knowles, 1991} está na

Figura 7.21. As alterações se limitam à entidade de controle desta entidade, e ao posicionamento dos somadores. O desempenho é o mesmo que o da WT direta e a área também.

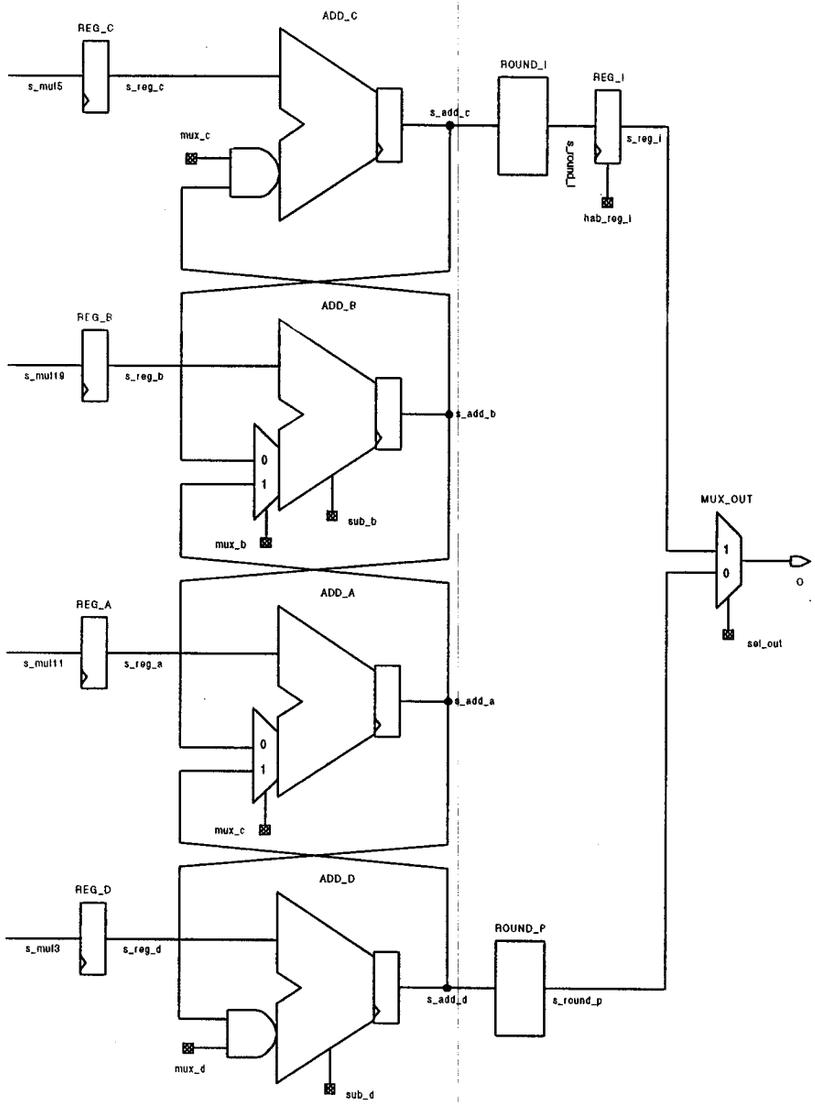


Figura 7.21: Arquitetura do processador para WT inversa

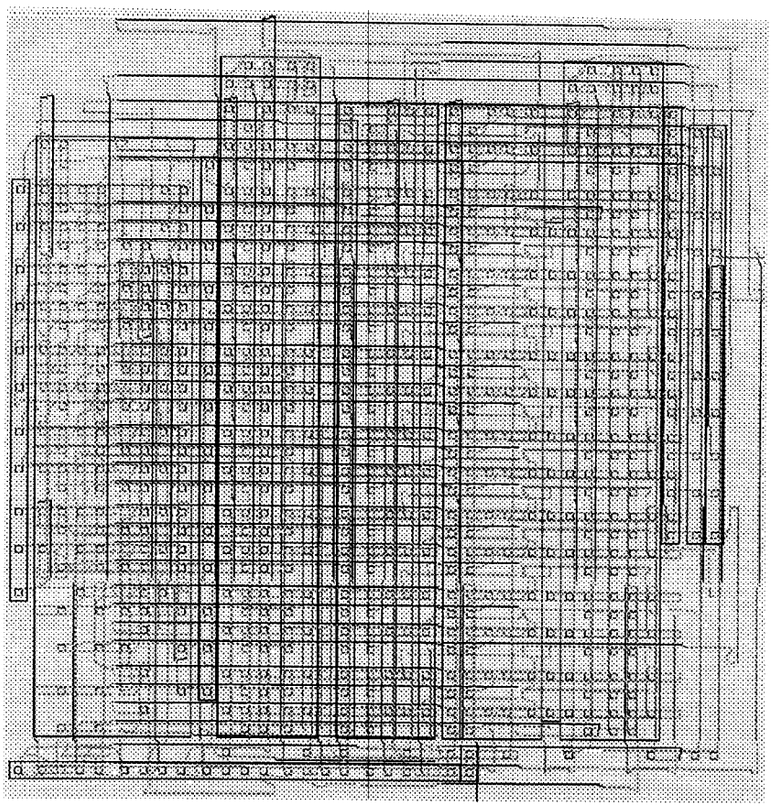


Figura 7.22: *Lay-out* da implementação da WT inversa no FPGA XC6216

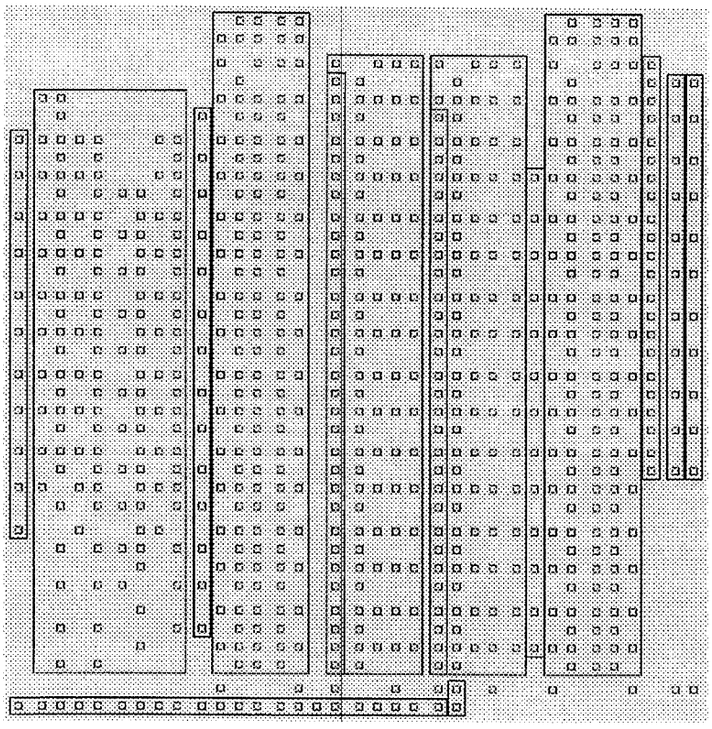


Figura 7.23: Distribuição de entidades da WT inversa no FPGA XC6216

7.6.1.3 Resultados

Várias imagens foram padronizadas de teste foram processadas e concluiu-se que a compressão utilizando esta arquitetura é limitada a valores inferiores a 8 vezes, pois as distorções da imagem para valores maiores são bastante perceptíveis. A taxa de compressão foi determinada em função do número de coeficientes descartados (zerados). A relação sinal-ruído PSNR foi de aproximadamente 35 dB nestas imagens. Os resultados qualitativos podem ser vistos nas figuras a seguir.

Imagem original gera 65536 coefs

Figura 7.24: "Lena" original

Imagem reconstruida utilizando 7505 coefs

Figura 7.25: "Lena" comprimida usando D4 e 3 escalas

Imagem reconstruida utilizando 7378 coefs

Figura 7.26: "Lena" comprimida usando D4 e 6 escalas

7.6.2. Wavelet D4 Utilizando FIR

Embora não seja tão eficiente quanto a implementação anterior, a implementação canônica da D4 utilizando filtros FIR é útil pois valida as entidades que implementam os filtros FIR e permite uma comparação entre esta implementação e a anterior, que é específica para a

wavelet D4.

A comunicação entre os estágios adjacentes do filtro se dá utilizando linhas locais, e por isso os estágios podem ser ligados em cascata sem maiores dificuldades no roteamento. A Figura 7.27 mostra que praticamente apenas linhas locais estão sendo utilizadas, o que torna o roteamento muito mais simples que o da arquitetura anterior. Isso contribui para o melhor desempenho desta arquitetura, da ordem de 78 ns.

Por outro lado, a área utilizada é muito maior que na implementação anterior. A Figura 7.27 representa apenas o filtro H_0 da transformada wavelet, que ocupa uma área retangular de 31x32 blocos lógicos. Seria necessário outra entidade das mesmas dimensões para implementar o filtro H_1, enquanto a implementação anterior faz a transformada completa, ou seja, faz o processamento equivalente ao dos dois filtros.

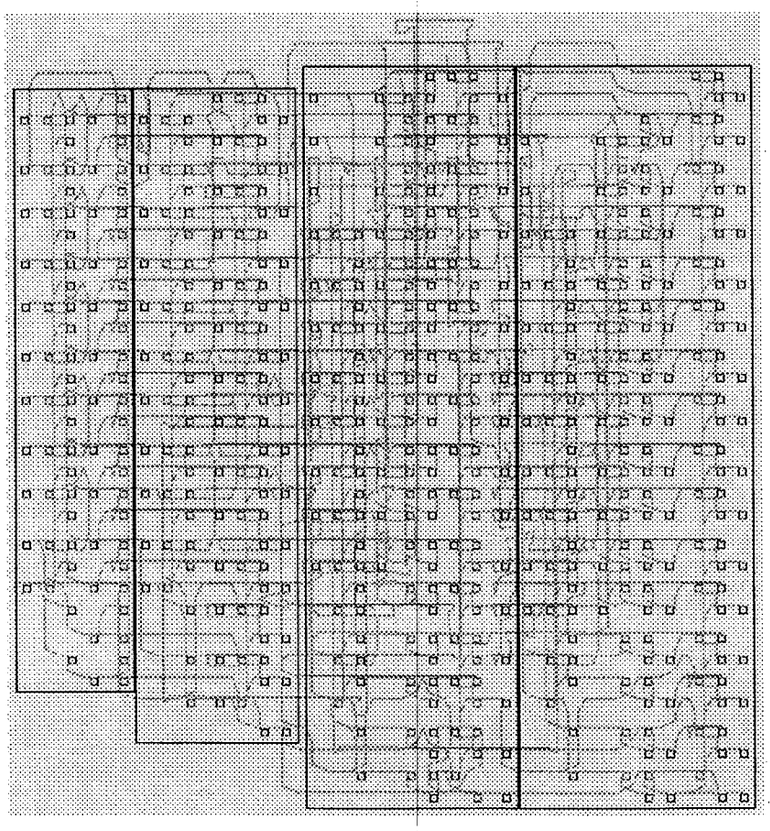


Figura 7.27: *Lay-out* da implementação do filtro H_0 da WT D4 no FPGA XC6216

Nas Figura 7.27 e Figura 7.28 pode-se averiguar a implementação das entidades FIR_INI e FIR1. A primeira entidade à esquerda é a FIR_INI, que ocupa uma área menor por dispensar o somador. Todas as outras entidades são instâncias da FIR1. Os coeficientes destes

estágios de filtragem são -3, 5, 19 e 11.

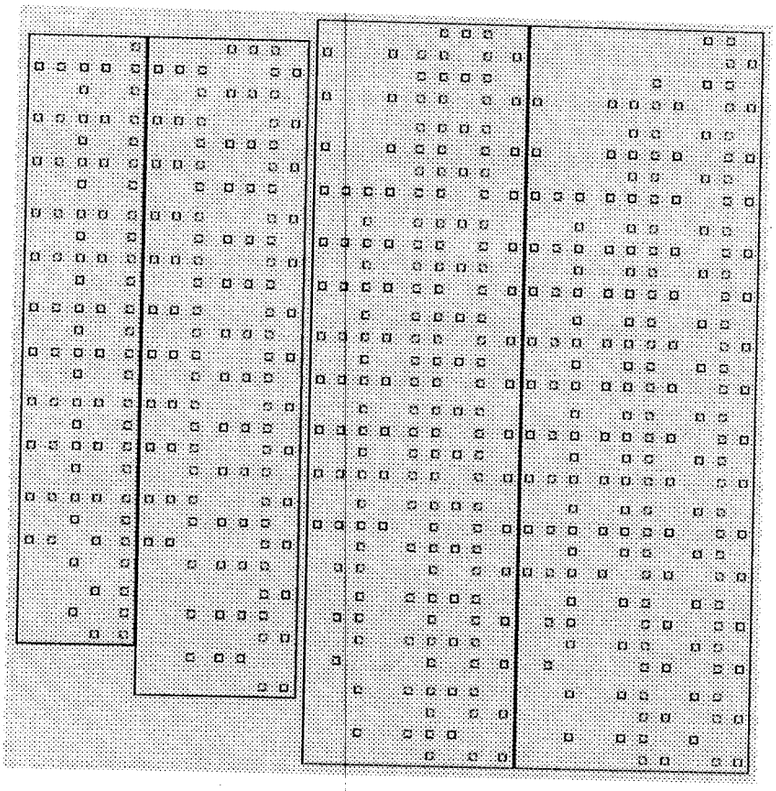


Figura 7.28: Distribuição de entidades do filtro H_0 da WT D4 no FPGA XC6216

7.6.3. Wavelet 9/7 Utilizando FIR

A wavelet 9/7 é uma wavelet biortogonal com coeficientes binários. Seus filtros FIR são simétricos, o que permite utilizar as entidades FIR2 para economia de área de implementação. Um filtro de 9 estágios se implementado utilizando instâncias da entidade FIR1 seria muito maior que a implementação obtida e apresentada na Figura 7.29.

O atraso máximo das entidades desta implementação é de 90ns. Este tempo maior que no caso da wavelet D4 não é devido ao número de estágios (9 ao invés de 4), pois o filtro faz utilização extensiva de *pipelining*. Ele se explica pelo uso de rotas de comprimento 4, além do roteamento local. Isto é necessário porque a entidade FIR2 utiliza um roteamento mais denso que a entidade FIR1, tanto na direção esquerda para direita quanto na oposta.

O filtro exibido na Figura 7.29 é o filtro H_0, de 9 *taps* e cujos coeficientes são [1, 0, -8, 16, 46, 16, -8, 0, 1]/64. O filtro H_1 possui coeficientes iguais a [-1 0 9 16 9 0 –1]/16. É um filtro longo que pode ser implementado numa área de 43x35 blocos lógicos.

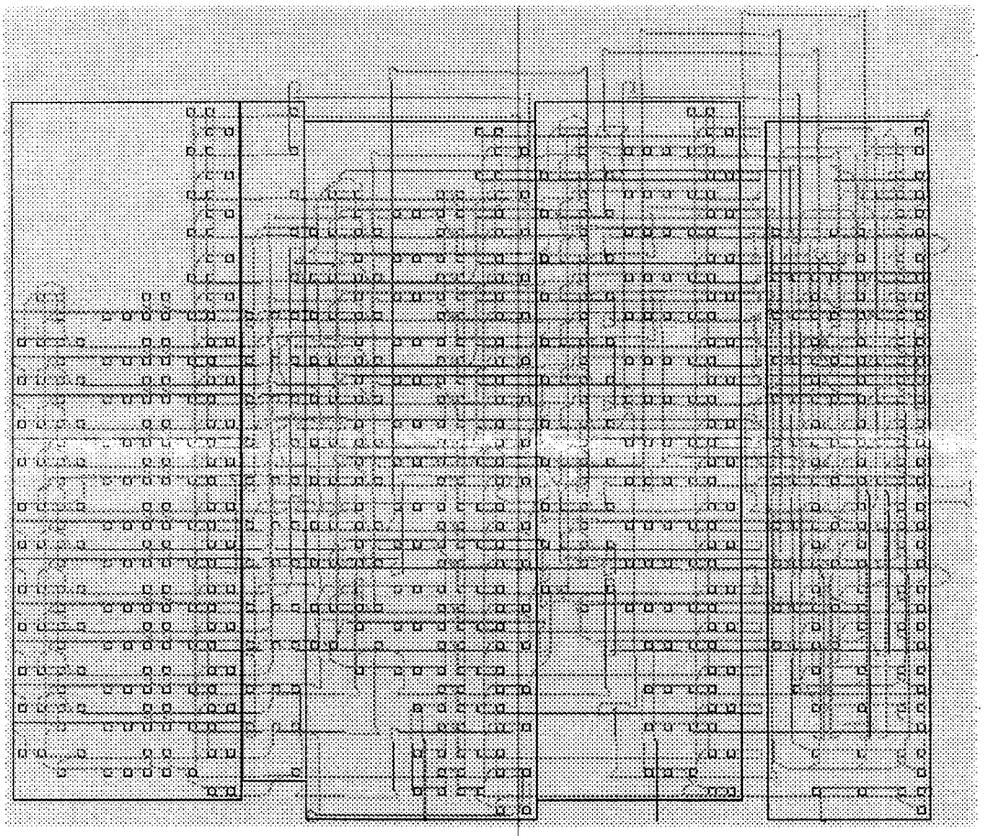


Figura 7.29: *Lay-out* da implementação do filtro H_0 da WT 9/7 no FPGA XC6216

7.7. Coprocessador de Granularidade Elevada

As implementações anteriores se limitaram a implementar apenas o núcleo do processamento da transformada wavelet, que são os filtros H_0 e H_1. Outro cenário discutido anteriormente envolve a implementação do coprocessador de granularidade mais elevada. Assim o próprio FPGA deve ser capaz de ler os *pixels* de imagem e armazenar os resultados, sem participação do processador. Apenas haveria a comunicação com o processador ao término do processamento do quadro.

Para isto foi desenvolvida uma entidade parametrizável de acesso à memória, permitindo a sua leitura e a sua escrita. As unidades de endereçamento possuem o recurso de transpor a matriz de *pixels* para implementar a WT separável. Para simplificar sua implementação, as dimensões da imagem devem ser uma potência de 2. As unidades de endereçamento representadas na Figura 7.30 possuem 4 bits, permitindo acesso a imagens de 16x16 *pixels*, mas são facilmente parametrizáveis para outros tamanhos, até um máximo de 16 bits.

A unidade de controle permite o processamento em várias escalas, nas duas dimensões. As linhas são tratadas como uma sequência única de *pixels*, o mesmo para as colunas. Isso causa uma distorção nas bordas da imagem e poderia ser resolvida com técnicas de *padding* {Strang e Nguyen, 1996}, que não foram implementadas.

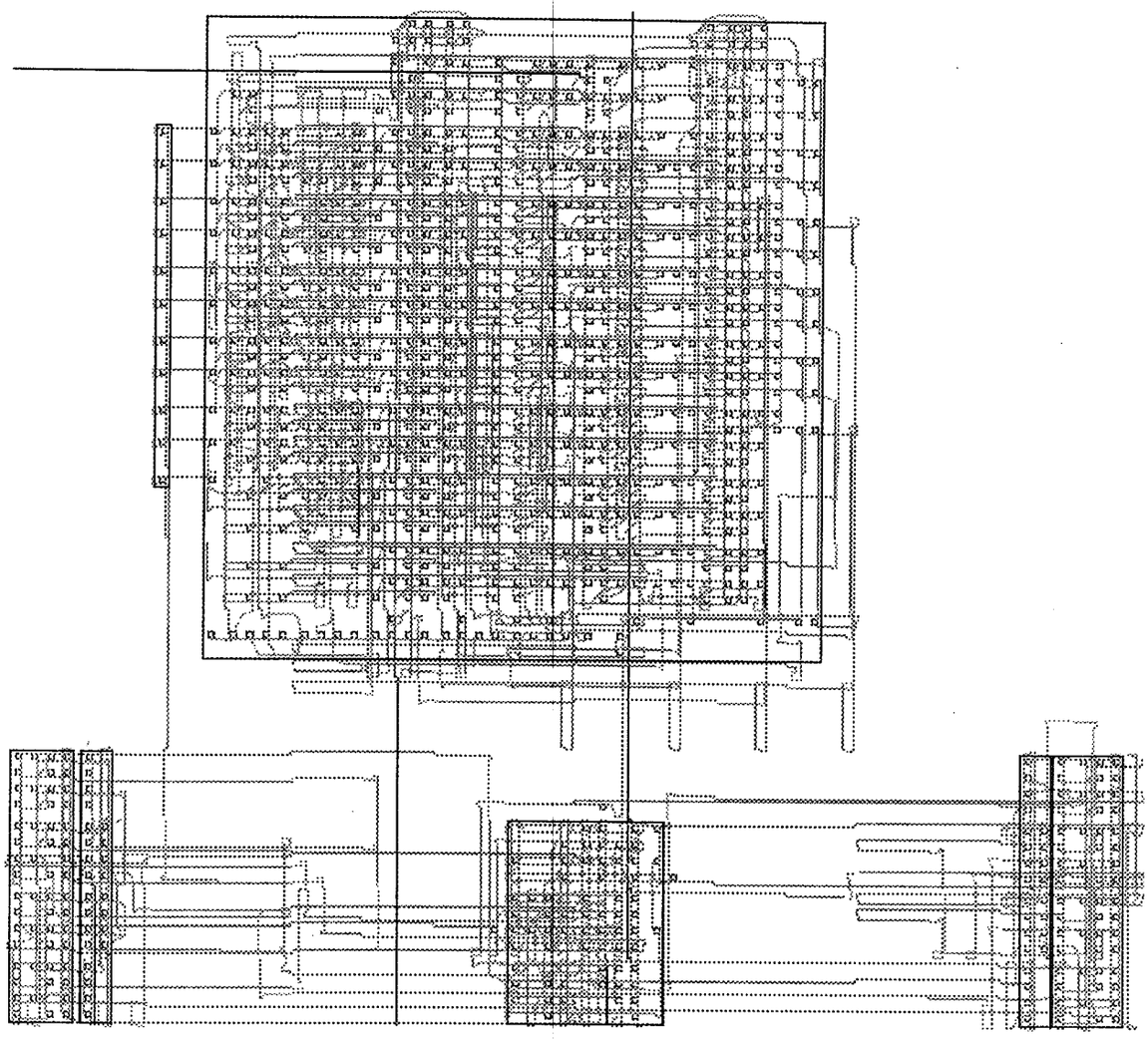


Figura 7.30: Processador wavelet D4 e unidades de acesso a memória e controle no FPGA XC6216

7.8. Considerações Finais

Como demonstrado, a arquitetura sugerida por {Lewis e Knowles, 1991} leva a uma implementação muito eficiente para áreas pequenas, no FPGA escolhido. As implementações baseadas em FIR não são menores, ao contrário, apresentam praticamente o dobro da área. Filtros binários foram implementados usando as entidades FIR2 e a eficiência da solução pode ser observada, devido ao fato dos coeficientes terem um pequeno número de 1 na sua representação

binária. Por isso os multiplicadores baseados em LUT não foram considerados.

A pequena área consumida pela implementação da arquitetura de Lewis & Knowles tornou possível a implementação das entidades de controle e de acesso à memória no limitado FPGA XC6216, de modo a tornar o coprocessador mais independente do processador auxiliar. A granularidade elevada deste coprocessador permite inclusive que o processador auxiliar seja um processador simples, como um microcontrolador, uma vez que a maior parte das tarefas está sendo executada pelo coprocessador no FPGA.

8. Conclusão

8.1. Análise de Resultados

Como se pode observar pelas imagens comprimidas, uma compressão de 8 vezes pode ser obtida pelo sistema proposto sem apresentar distorções significativas. E esta compressão é conseguida através da forma mais direta, que é simplesmente anular os coeficientes abaixo de uma certo limite. Não foi feito nenhum tipo de codificação estatística nem uma distribuição dos bits em regiões de maior sensibilidade do sistema visual humano. Portanto ainda há possibilidades de se conseguir maiores taxas de compressão.

As implementações alternativas mostraram que não existe maneira simples de se conseguir uma implementação mais compacta e eficiente que a proposta por {Lewis e Knowles, 1991}. Utilizando-se a wavelet de Haar certamente se conseguiria uma implementação compacta, mas esta wavelet é muito pouco utilizada para compressão de imagens devido a distorção que ela introduz {Rassau e outros, 1999}.

8.2. Contribuições

Além de obter a implementação de um sistema de compressão de vídeo em uma FPGA com limitações de área e roteamento, outras contribuições foram trazidos pelo trabalho.

Foram desenvolvidas entidades para aplicações de processamento de sinais para FPGAs da família Xilinx XC6200. Estas entidades foram testadas extensivamente e provaram sua eficiência. O seu posicionamento cuidadoso torna simples o roteamento e interligação com outras entidades, facilitando muito o desenvolvimento de aplicações para esta arquitetura.

A adoção de uma ferramenta de síntese de domínio público foi uma decisão que tornou o trabalho mais difícil, pelas limitações de recursos oferecidos. Mas por outro lado permite que estas bibliotecas sejam utilizadas por um público mais amplo.

Neste trabalho também foi necessário fazer algumas implementações utilizando ferramentas de síntese comerciais. Para isso o autor teve que instalar, configurar e operar as ferramentas da Synopsys™, o que ajudou na estruturação do Laboratório de Computação Reconfigurável do ICMC e disseminar a cultura sobre ferramentas profissionais de síntese e

simulação digital no grupo de trabalho.

8.3. Originalidade

A proposta do trabalho é original no sentido que existem poucas implementações na literatura de aplicações de compressão de vídeo em FPGAs com restrições de área. A maior parte das referências encontradas tratam de implementações utilizando vários FPGAs, ou FPGAs de capacidade elevada.

Outra contribuição original foi a adaptação de técnicas de modelamento de erros em filtros FIR na análise de erros em bancos de filtros multiescala. As simulações feitas mostram que este modelo pode ser utilizado para prever com alguma precisão o comportamento dos erros de quantização envolvidos em implementações de filtros para wavelets utilizando aritmética de ponto fixo. Isto é muito importante no contexto de implementações em FPGAs, que utilizam principalmente de aritmética de ponto fixo e possuem a flexibilidade de adotarem o tamanho de palavra que for mais indicado.

Uma exaustiva revisão bibliográfica não apontou nenhuma implementação de algoritmos de transformada wavelet para a arquitetura da família de FPGA XC6200. Nem mesmo implementações de filtros FIR para esta arquitetura. Por isso acredita-se que esta implementação descrita é inédita. Assim como a análise da implementação de algumas técnicas de processamento de sinais nesta arquitetura.

8.4. Trabalhos Futuros

Uma constatação feita pelo autor é que a arquitetura do FPGA XC6200 é muito apropriada para implementações bit-serial. As maiores dificuldades na implementação se deram no roteamento de barramentos paralelos entre entidades. Um trabalho que foi iniciado e não pode ser concluído foi a utilização de arquiteturas para WT não-separável em conjunto com a implementação bit-serial. Como estas arquiteturas se utilizam de vários processadores de wavelets, talvez a implementação bit-serial pudesse implementá-las de modo a obter um desempenho ao menos comparável ao implementação descrita.

Como existe alguma área livre no FPGA XC6216, especialmente na implementação da arquitetura de {Lewis e Knowles, 1991}, poderia-se agregar ao coprocessador mais funções de compressão, desde que não sejam muito complexas. Entre as funções que podem ser

implementadas estão uma unidade de compressão *run-length* e uma unidade de descarte de coeficientes. Outra sugestão seria implementar uma técnica simples de eliminação de redundância temporal, através da diferença entre quadros sucessivos. Isto poderia aumentar muito as taxas de compressão para imagens dinâmicas.

Uma outra perspectiva interessante é a utilização deste trabalho na implementação de um sistema móvel de captura de vídeo. Estudos preliminares mostram que um processador de uso geral simples (~20 MIPS) associado ao sistema implementado seria suficiente para este tipo de aplicação.

9. Bibliografia

Altera; "Image Processing in Altera FLEX 10K Devices", 1996, www.altera.com/document/papers/cpimag.pdf.

Altera; **1998 Data Book,** 1998.

Altera; "Implementing FIR Filters in FLEX Devices", http://www.altera.com/document/an/an073.pdf, 1998.

Andraka, R.; "FIR Filter Fits in an FPGA Using a Bit Serial Approach", *Proceedings of the Third Annual PLD Design Conference and Exhibit*, 1993, http://users.ids.net/~randraka/files/fir.pdf.

Anon; "Reconfigurable Processor Reincarnates the Venerable 8032", *Computer Design*, Vol. 37, No. 10, pp. 34-38, Oct 1998.

Barr, M.; "Controlling Run-Time Reconfigurable Hardware Designs With Java", *DesignCon98*, 1998, www.tsi-telsys.com.

Bergmann, N. e Chung, Y.; "Video Compression With Custom Computers", *Ieee Transactions On Consumer Electronics*, Vol. 43, No. 3, pp. 925-933, Apr 1997.

Bergmann, N. e Mudge, C.; "Comparing the Performance of FPGA-Based Custom Computers With General-Purpose Computers for DSP Applications", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1994.

Bertin, P.; Roncin, D. e Vuillemin, J.; "Programmable Active Memories: a Performance Assessment", 1993, Seattle, WA, Symposium on Integrated Systems.

Biederman, D.; "An Overview on Writing a VHDL Testbench", *Ieee Design & Test Of Computers*, pp. 384-388, 1997.

Bouridane, A.; Crookes, D.; Donachy, P.; Alotaibi, K. e Benkrid, K.; "A High Level FPGA-Based Abstract Machine for Image Processing", *Journal Of Systems Architecture*, Vol. 45, No. 1, pp. 809-824, 1999.

Brebner, G., "Circlets: Circuits As Applets", *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1998b, http://www.dcs.ed.ac.uk/home/gordon/fan/soft-circuitry/fccm98.ps.

Brebner, G.; "Field-Programmable Logic: Catalyst for New Computing Paradigms", *Lecture Notes in Computer Science*, Vol. 1482, pp. 49-58, 1998a.

Brebner, G. e Donlin, A.; "Runtime Reconfigurable Routing", *Lecture Notes in Computer Science*, Vol. 1304, 1997, Springer-Verlag, http://www.dcs.ed.ac.uk/~adamd/research/raw98.ps.gz.

Brebner, G. e Gray, J.; "Use of Reconfigurability in Variable-Length Code Detection at Video Rates", *Lecture Notes in Computer Science*, Vol. 975, pp. 429-438, 1995, Springer-Verlag.

Brown, S.; "FPGA Architectural Research: A Survey", *Ieee Design & Test Of Computers*, Vol. 13, No. 4, pp. 9-15, 1996.

Brown, S. e Rose, J.; "FPGA and CPLD Architectures: A Tutorial", *Ieee Design & Test Of Computers*, Vol. 13, No. 2, pp. 42-57, 1996, http://www.eecg.toronto.edu/~jayar/pubs/brown/survey.html.

Chakrabarti, C. e Mumford, C.; "Efficient Realizations of Encoders and Decoders Based on the 2-D Discrete Wavelet Transform", *Ieee Transactions On Very Large Scale Integration (Vlsi) Systems*, Vol. 7, No. 3, pp. 289-298, Sep 1999.

Chakrabarti, C. e Vishwanath, M.; "Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers", *IEEE Transactions on Signal Processing*, Vol. 43, No. 3, pp. 759-771, Mar 1995.

Chan, P. K. e Mourad, S.; **Digital Design Using Field Programmable Gate Arrays,** 1994, Englewood Cliffs, Prentice-Hall.

Chang, S. J.; Lee, M. H. e Park, J. Y.; "A High Speed VLSI Architecture of Discrete Wavelet Transform for MPEG-4", *Ieee Transactions On Consumer Electronics*, Vol. 43, No. 3, pp. 623-627, Aug 1997.

Chow, P.; Seo, S. O.; Rose, J.; Chung, K.; Paéz-Monzón, G. e Rahadja, I.; "The Design of an SRAM-Based Field Programmable Gate Array, Part I: Architecture", *Ieee Transactions On Very Large Scale Integration (Vlsi) Systems*, Vol. XX, No. YY, pp. -110, 1999.

Churcher, S.; Kean, T. e Wilkie, B.; "The XC6200 FastMap™ Processor Interface", *Lecture Notes in Computer Science*, Vol. 975, pp. 36-43, 1995, Springer-Verlag.

Clark, D., "Supporting FPGA Microprocessor Through Retargetable Software Tools", Master Thesis, Brigham Young University, Apr 1996, splish.ee.byu.edu/docs/dasm-fccm.ps.gz.

Cohen, A. e Ryan, R. D.; **Wavelets and Multiscale Signal Processing,** Applied Mathematics and Mathematical Computation, 1995, London, Chapman & Hall.

Cucchiara, R.; Neri, G. e Piccardi, M.; "A Real-Time Hardware Implementation of the Hough Transform", *Journal Of Systems Architecture*, Vol. 45, No. 1, pp. 31-45, 1998.

Davis, D.; Barr, M.; Bennett, T.; Edwards, S.; Harris, J.; Miller, I. e Schanck, C.; "A Java Development and Runtime Environment for Reconfigurable Computing", *Lecture Notes in Computer Science*, Vol. 1388, pp. 43-48, 1998, Spring-Verlag.

DeHon, A.; "DPGA Utilization and Application", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 1996.

DeHon, A.; "Dynamically Programed Gate Arrays: A Step Toward Increased Computational Density", *Fourth Canadian Workshop of Field-Programmable Devices*, 1996.

DeHon, A., "Reconfigurables Architectures for General-Purpose Computing", Report 1586, 1996, www.ai.mit.edu/projects/transit/rcgp/rcgp.html.

Denyer, P. e Renshaw, D.; **VLSI Signal Processing: A Bit-Serial Approach,** 1985, Addison-Wesley.

Dewey, A.; **Analysis and Design of Digital Systems With VHDL,** 1997, PWS Publishing Company.

Dick, C. e Harris, F.; "Virtual Signal Processors", *Microprocessors And Microsystems*, Vol. 22, No. 3-4, pp. 135-148, 1998.

Dick, C. H.; "Computing 2-D DFTs Using FPGAs", *Lecture Notes in Computer Science*, Vol. 1142, pp. 96-105, 1996, Springer-Verlag.

Donlin, A.; Hogg, J.; Singh, S. e DeWit, M., "A Dynamic Reconfiguration Run-Time System", 1997, IEEE Symposium on FPGA-Based Custom Computing Machines.

Ebrahimi, T. e Kunt, M.; "Visual Data Compression for Multimedia Applications", *Proceedings Of The Ieee*, Vol. 86, No. 6, pp. 1109-1125, Jun 1998.

Egger, O.; Fleury, P.; Ebrahimi, T. e Kunt, M.; "High-Performance Compression of Visual Information - A Tutorial Review - Part I: Still Pictures", *Proceedings Of The Ieee*, Vol. 87, No. 6, pp. 976-1011, Jun 1999.

Encinas, W. S. e Moreira, E.; "Hardware and Software Partition With Microcontrollers and CPLDs: A Case Study", *PDPTA'99 International Conference*, pp. 3002-3008, 1999, Las Vegas, NV.

Ersoy, O. K. e Nouira, A.; "Image Coding With the Discrete Cosine-III Transform", *IEEE Journal on Selected Areas in Communications*, Vol. 10, No. 5, pp. 884-891, Jun 1992.

Faura, J.; Moreno, J. M.; Aguirre, M. A.; Duong, P. e Insenser, J. M.; "Multicontext Dynamic Reconfiguration and Real-Time Probing on a Novel Mixed Signal Programmable Device With On-Chip Microprocessor", *Lecture Notes in Computer Science*, Vol. 1304, pp. 1-10, 1997, Springer-Verlag.

Fawcett, B., "Reconfigurable Processing With Field Programmable Gate Arrays", *Proc. of Int'l. Conf. on Application Specific Systems, Architectures and Processors (ASAP96),* pp. 293-302, 1996.

Fournier, A.; **Intoduction,** in **Wavelets and Their Applications in Computer Graphics,** No. I, pp. 5-33, 1995, SIGGRAPH 95.

Furht, B.; "A Survey of Multimedia Compression Techniques and Standards. Part II: Video Compression", *Real-Time Imaging*, pp. 319-337, 1995.

Gehring, S. e Ludwig, S. H.; "The Trianus System and Its Application to Custom Computing", *Lecture Notes in Computer Science* , pp. 176-184, 1996, Springer-Verlag.

Gokale, M.; Holmes, B.; Kopser, A.; Kunze, D.; Lopresti, D.; Lucas, S. e Minnich, R.; SPLASH: A Reconfigurable Linear Logic Array, ftp.super.org/pub/fpga/splash-1/splash-1.ps, 1994.

Golston, J.; "Single-Chip H.324 Videoconferencing", *Ieee Micro*, pp. 21-33, Aug 1996.

Grant, D. M.; Velab: VHDL Elaborator for the XC6200, 1997, www.xilinx.com/apps/velabrel.htm.

Greenfield, D.; Crome, C.; Won, M. S. e Amos, D.; "Enhancing Fixed Point DSP Processor Performance by Adding CPLDs As Coprocessor Elements", *Lecture Notes in Computer Science*, Vol. 1304, pp. 324-332, 1997, Springer-Verlag.

Grzeszczak, A.; Mandal, M.; PANCHANATHAN, S. e Yeap, T.; "VLSI Implementation of Discrete Wavelet Transform", *Ieee Transactions On Very Large Scale Integration (Vlsi) Systems*, Vol. 4, No. 4, pp. 421-433, Dec 1996.

Gschwind, M. e Salapura, V.; "A VHDL Design Metodology for FPGAs", *Lecture Notes in Computer Science*, Vol. 975, pp. 208-217, 1995a, Springer-Verlag.

Gschwind, M. e Salapura, V., "Optimizing VHDL Code for FPGA Targets", Report 95/11, 1995b, ftp.vlsivie.tuwien.ac.at/pub/vhdl4fpga/tr.ps.

Guccione, S. A. e Gonzalez, M. J.; "Classification and Performance Od Reconfigurable Architectures", *Lecture Notes in Computer Science*, Vol. 975, pp. 439-448, 1995, Springer-Verlag.

Guermeur, P.; "A Real-Time Discrete Wavelet Transform Implementation on an FPGA Architecture", *ISAS'98*, 1998, www.ensta.fr/~guermeur/publications/publications.html.

Gustin, V.; "An FPGA Extension to ALU Functions", *Microprocessors And Microsystems*, Vol. 22, No. 9, pp. 501-508, 1999.

Guttag, K.; Gove, R. J. e Van Aken, J. R.; "A Single-Chip Multiprocessor for Multimedia: The MVP", *IEEE Computer Graphics and Applications*, pp. 53-64, Nov 1992.

Hamid, G.; "An FPGA-Based Coprocessor for Image Processing", *IEE Electronics Division Colloquium on Integrated Imaging Sensors and Processing*, Vol. 238, pp. 1-4, 1994.

Harney, K.; Keith, M.; Lavelle, G.; Ryah, L. e Stark, D. J.; "The I750 Video Processor: A Total Multimedia Solution", *Communications Of The Acm*, Vol. 34, No. 4, pp. 64-78, Apr 1991.

Hartenstein, R.; Herz, M. e Gilbert, F.; Hartenstein, R. and Keevallik, A.; "Designing for Xilinx XC6200 FPGAs", *Lecture Notes in Computer Science*, Vol. 1482, 1998, Springer-Verlag, xputers.informatik.uni-kl.de.

Hauck, S.; "The Roles of FPGA's in Reprogrammable Systems", *Proceedings Of The Ieee*, Vol. 86, No. 4, pp. 615-638, 1998.

Hauser, J. R. e Wawrzynek, J.; "Garp: A MIPS Processor With a Reconfigurable Coprocessor", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1997.

Haynes, S. e Cheung, P. Y. K., "Configurable Multiplier Blocks for Embedding in FPGAs", 1999.

He, Y.; Ahmad, I. e Liou, M. L.; "A Software-Based MPEG-4 Video Encoder Using Parallel Processing", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 8, No. 7, pp. 909-920, Nov 1998.

Heron, J. P. e Woods, R.; "Architectural Strategies for Implementing an Image Processing Algorithm on XC6200 FPGA", *Lecture Notes in Computer Science*, Vol. 1142, pp. 317-326, 1996, Springer-Verlag.

Hesener, A.; "Implement Reconfigurable Datapaths in FPGAs for Adaptive Filter Designs", *Lecture Notes in Computer Science*, Vol. 1142, pp. 221-229, 1996, Springer-Verlag.

Hopkins, V.; "Cost or Performance - ASIC Still Beat FPGAs", *Integrated System Design*, Mar 1998.

Hung, K. C.; Huang, Y. J.; Truong, T. K. e Wang, C. M.; "FPGA Implementation for 2D Discrete Wavelet Transform", *Electronics Letters*, Vol. 34, No. 7, pp. 639-640, 1998.

Hwang, K.; **Computer Arithmetic: Principles, Architecture and Design,** 1979, New York, John Wiley & Sons.

Jaggernauth, J.; Loui, A. C. P. e Venetsanopoulos, A. N.; "Real-Time Image-Processing by Distributed Arithmetic Implementation of Two-Dimensional Digital-Filters", *Ieee*

Transactions On Acoustics Speech And Signal Processing, Vol. 33, No. 6, pp. 1546-1555, 1985.

Jain, A. K.; "Image Data Compression: A Review", *Proceedings Of The Ieee*, Vol. 69, No. 3, pp. 349-389, Mar 1981.

Jayant, N.; Johnston, J. e Safranek, R.; "Signal Compression Based on Models of Human Perception", *Proceedings Of The Ieee*, Vol. 81, No. 10, pp. 1385-1421, Oct 1993.

Johnson, M.; **Superscalar Microprocessor Design,** 1991, Englewood Cliffs, Prentice-Hall.

Kean, T.; New, B. e Slous, B.; "A Fast Constant Coefficient Multiplier for the XC6200", *Lecture Notes in Computer Science*, Vol. 1142, pp. 230-236, 1996, Springer-Verlag.

Kim, S. e Sung, W.; "Fixed-Point Error Analysis and Word Length Optimization of 8x8 IDCT Architectures", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 8, No. 8, pp. 935-940, Dec 1998.

Knowles, G.; "VLSI Architeture for the Discrete Wavelet Transform", *Electronics Letters*, Vol. 26, No. 15, pp. 1184-1185, Jul 1990.

Koenen, R.; Pereira, F. e Chiariglione, L.; MPEG-4: Context and Objectives, drogo.cselt.stet.it/leonardo/icjfiles/mpeg-4_si/paper1.htm, 1997.

Kondo, T.; Suguri, K.; Ikeda, M.; Abe, T.; Matsuda, H.; Okubo, T.; Ogura, K.; Tashiro, Y.; Ono, N.; Minami, T.; Kusaba, R.; Ikegata, T.; Nobutaro, S. e Kasai, R.; "Two-Chip MPEG-2 Video Encoder", *Ieee Micro*, pp. 51-58, Apr 1996.

Koren, I.; **Computer Arithmetic Algorithms,** 1993, Englewood Cliffs, Prentice-Hall.

Kostarnov, I.; Morley, S.; Osmany, J. e Solomon, C.; "A Reconfigurable Approach to Low Cost Media Processing", *Lecture Notes in Computer Science*, Vol. 1304, pp. 79-90, 1997, Springer-Verlag.

Kunt, M.; Ikonomopoulos, A. e Koecher, M.; "Second Generation Image-Coding Techniques", *Proceedings Of The Ieee*, Vol. 73, No. 4, pp. 549-574, Apr 1985.

Kuroda, I. e Nishitani, T.; "Multimedia Processors", *Proceedings Of The Ieee*, Vol. 86, No. 6, pp. 1203-1221, Jun 1998.

Lala, P. K.; **Digital System Design Using Programmable Logic Devices,** 1990, Englewood Cliffs, Prentice Hall.

Lee, R. B.; "Accelerating Multimedia With Enhanced Microprocessors", *Ieee Micro*, pp. 22-32, Apr 1995.

Lewis, A. S. e Knowles, G.; "VLSI Architecture for the 2-D Daubechies Wavelet Transform Without Multipliers", *Electronics Letters*, Vol. 27, No. 2, pp. 171-173, Jan 1991.

Lim, Y. C.; Evans, J. B. e Liu, B.; "An Efficient Bit-Serial FIR Filter Architecture", *Circuits, Systems, and Signal Processing*, Vol. 14, No. 5, Apr 1993.

Lim, Y. C. e Liu, B.; "Design of Cascade Form FIR Filters With Discrete Valued Coefficients", *Ieee Transactions On Acoustics Speech And Signal Processing*, Vol. 36, No. 11, pp. 1735-1739, Nov 1988.

Lu, J.; Algazi, V. R. e Estes, R. R.; "Comparative Study of Wavelet Image Coders", *Optical Engineering*, Vol. 35, No. 9, pp. 2605-2619, Sep 1996.

Ludwig, S. H.; Hartenstein, R. and Glesner, M.; "The Design of a Coprocessor Board Using Xilinx's XC6200 FPGA - An Experience Report", *Lecture Notes in Computer Science*, Vol. 1146, 1996, Springer-Verlag, ftp://ftp.inf.ethz.ch/pub/publications/papers/cs/FPL96.Coproc.ps.gz.

Mallat, S. G.; "A Theory for Multiresolution Signal Decomposition: the Wavelet Representation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 7, pp. 674-693, Jul 1989.

Netravali, A. N. e Haskell, B. G.; **Digital Pictures: Representation and Compression,** 1988, Plenum Press.

New, B., "Estimating the Performance of XC4000E Adders and Counters", Report XAPP 018, 1996b, www.xilinx.com/xapp/xapp018.pdf.

New, B., "Using the Dedicated Carry Logic in XC4000E", Report XAPP 013, 1996a, www.xilinx.com/xapp/xapp013.pdf.

Nisbet, S. e Guccione, S. A.; "The XC6200DS Development System", *Lecture Notes in Computer Science*, Vol. 1304, pp. 61-68, 1997, Springer-Verlag.

O'Donnell, R.; "Prolog to the Roles of FPGA's in Reprogrammable Systems", *Proceedings Of The Ieee*, Vol. 86, No. 4, pp. 613-614, 1998.

Oppenheim, A. e Schafer, R.; **Digital Signal Processing,** 1975, Englewood Cliffs, Prentice-Hall.

Paek, S.-K. e Kim, L.-S.; "2D DWT VLSI Architecture for Wavelet Image Processing", *Electronics Letters*, Vol. 34, No. 6, pp. 537-538, Mar 1998.

Page, I.; "Reconfigurable Processor Architectures", *Microprocessors And Microsystems*, Oct 1996, ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Ian.Page/proc_arch.ps.gz.

Panneerselvam, G.; Graumann, P. J. W. e Turner, L. E.; "Implementation of Fast Fourier Transforms and Discrete Cosine Transforms in FPGAs", *Lecture Notes in Computer Science*, Vol. 975, pp. 272-281, 1995, Springer-Verlag.

Parhi, K. K. e Nishitani, T.; "VLSI Architectures for Discrete Wavelet Transforms", *Ieee Transactions On Very Large Scale Integration (Vlsi) Systems*, Vol. 1, No. 2, pp. 191-202, Jan 1993.

Patterson, D. A. e Hennessy, J. L.; **Computer Architecture: A Quantitative Approach,** 1995, San Francisco, Morgan Kaufmann Publishers.

Peleg, A. e Weiser, U.; "MMX Technology Extension to the Intel Architecture", *Ieee Micro*, pp. 42-50, Aug 1996.

Peleg, A.; Wilkie, S. e Weiser, U.; "Intel MMX for Multimedia PCs", *Communications Of The Acm*, Vol. 40, No. 1, pp. 25-38, Jan 1997.

Pennbecker, W. B.; **JPEG Still-Image Data Compression Standard,** 1993, New York, Van Nostrand.

Pirsch, P.; Demassieux, N. e Gehrke, W.; "VLSI Architectures for Video Compression-A Survey", *Proceedings Of The Ieee*, Vol. 83, No. 2, pp. 220-246, Feb 1995.

Porat, B.; **A Course on Digital Signal Processing,** 1997, John Wiley & Sons.

Rassau, A. M.; Chae, S. e Eshraghian, K.; "Simple Binary Wavelet Filters for Efficient VLSI Implementation", *Electronics Letters*, Vol. 35, No. 7, pp. 555-557, Apr 1999.

Rincon, A. M.; Lee, W. R. e Slattery, M.; "The Changing Landscape of System-on-a-Chip Design", *Micro News*, Vol. 5, No. 3, 1999, http://www.chips.ibm.com/micronews/vol5_no3/rincon.html.

Rioul, O. e Vetterli, M.; "Wavelets and Signal Processing", *IEEE Signal Processing Magazine*, pp. 14-38, Oct 1991.

Sakamoto, T. e Hase, T.; "JPEG Software Solution for a 32-Bit MCU", *Ieee Transactions On Consumer Electronics*, Vol. 43, No. 3, pp. 410-417, Aug 1997.

Shand, M.; "Flexible Image Acquisition Using Reconfigurable Hardware", *IEEE Workshop on FPGAs for Custom Computing Machines*, 1995, ftp://ftp.digital.com/pub/DEC/PRL/research-articles/SHA951.ps.Z.

Shand, M.; PCI Pamette: Generic PCI Board, http://research.compaq.com/SRC/pamette/overview/sld001.html, 1996.

Shiratsuchi, S.; "FPGA As a Key Component for Reconfigurable System", *Lecture Notes in Computer Science*, Vol. 1259, pp. 23-32, 1997, Springer-Verlag.

Sikora, T.; "The MPEG-4 Video Standard Verification Model", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 7, No. 1, pp. 19-31, Feb 1997.

Singh, S. e Bellec, P.; "Virtual Hardware for Graphics Applications Using FPGAs", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1994.

Spillane, J. e Owen, H.; "Temporal Partitioning for Partially-Reconfigurable-Field-Programmable Gate", *Lecture Notes in Computer Science*, Vol. 1388, pp. 37-42, 1998, Springer-Verlag.

Stark, H. e Woods, J. W.; **Probability, Random Processes and Estimation Theory for Engineers,** 2nd Ed, 1994, Englewood Cliffs, Prentice-Hall.

Stollnitz, E. J.; DeRose, T. D. e Salesin, D. H.; "Wavelet for Computer Graphics: A Primer, Part 2", *IEEE Computer Graphics and Applications*, pp. 75-84, Jul 1995.

Strang, G. e Nguyen, T.; **Wavelets and Filter Banks,** 1996, Massachusetts, Wellesley-Cambridge Press.

Sweldens, W.; **Wavelets, Signal Compression and Image Processing,** in **Wavelets and Their Applications in Computer Graphics,** No. IV, 1995, SIGGRAPH 95.

Tau, E.; Chen, D.; Eslick, I.; Brown, J. e DeHon, A.; "A First Generation DPGA Implementation", *Canadian Workshop of Field-Programmable Devices*, 1995.

Trainor, D.; Heron, J. P. e Woods, R.; "Implementation of the 2D DCT Using a Xilinx XC6264 FPGA", *SPIE'98*, 1998, http://www.ee.qub.ac.uk/dsp/research/vlsi/reconf_logic/.

Tredennick, N.; "Get Ready for Reconfigurable Computing", *Computer Design*, pp. 55-63, Apr 1998.

Triscend; Configurable Processors: An Emerging Solution For Embedded System Design, http://www.triscendcorp.com/products/Triscend_white_paper_v1-0.pdf, 1998.

Tuck, B.; "Choosing FPGAs, ASICs, or Cores for DSP-Based System Design", *Computer Design*, pp. 85-98, Feb 1996.

Turley, J.; "Soft Computing Reconfigures Designer Options", *Computer Design*, pp. 76-82, Apr 1997.

Turner, L. E.; Graumann, P. J. W. e Gibb, S. G.; "Bit-Serial FIR Filters With CSD Coefficients for FPGAs", *Lecture Notes in Computer Science*, Vol. 975, pp. 310-320, 1995, Springer-Verlag.

Vetterli, M. e Herley, C.; "Wavelets and Filter Banks: Theory and Design", *IEEE Transactions on Signal Processing*, Vol. 40, No. 9, pp. 2207-2232, Sep 1992.

Villasenor, J.; Belzer, B. e Liao, J.; "Wavelet Filter Evaluation for Image Compression", *IEEE Transactions on Image Processing*, Vol. 8, No. 4, pp. 1053-1060, Aug 1995.

Vishwanath, M.; OWENS, R. M. e IRWIN, M. J.; "VLSI Architectures for the Discrete Wavelet Transform", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 42, No. 5, pp. 305-316, May 1995.

Vuillemin, J. E.; Bertin, P.; Roncin, D.; Shand, M.; Touati, H. H. e Boucard, P.; "Programmable Active Memories: Reconfigurable Systems Come of Age", *Ieee Transactions On Very Large Scale Integration (Vlsi) Systems*, Vol. 4, No. 1, pp. 56-69, 1996.

Wakerly, J. F.; **Digital Design, Principles & Practices,** 2nd Ed, 1994, Englewood Cliffs, Prentice-Hall.

Wasson, S. L.; "Design Tips for High Performance FPGA Design", *ASIC & EDA*, pp. 41-51, Oct 1994.

Wirth, N.; "Hardware Compilation: Translating Programs Into Circuits", *Computer*, Vol. 31, No. 6, pp. 25-31, Jun 1998, http://www.lola.ethz.ch/lola/hcl/.

Wirthlin, M. J. e Hutchings, B. L.; "A Dynamic Instruction Set Computer", *IEEE Workshop on FPGAs for Custom Computing Machines*, 1995.

Wirthlin, M. J. e Hutchings, B. L.; "Implementations Approaches for Reconfigurable Logic Applications", 1999, www.ee.byu.edu/faculty/wirthlin/pubs/ths.ps.gz.

Wirthlin, M. J.; Hutchings, B. L. e Gilson, K. L.; "The Nano Processor: a Low Resource Reconfigurable Processor", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 23-30, 1994.

Wittig, R. D. e Chow, P.; "OneChip: An FPGA Processor With Reconfigurable Logic", *IEEE Symposium on FPGAs For Custom Computing Machines (FCCM)*, 1996, http://www.eecg.toronto.edu/~wittig/.

Wong, P.; "Quantization and Roundoff Noises in Fixed-Point FIR Digital Filters", *IEEE Transactions on Signal Processing*, Vol. 39, No. 7, pp. 1552-1563, Jul 1991.

Woods, R.; Trainor, D. e Heron, J. P.; "Applying an XC6200 to Real-Time Image Processing", *Ieee Design & Test Of Computers*, Vol. 15, No. 1, pp. 30-38, 1998.

Woodward, D. R.; Page, I. e Harley, R. G.; "A Programmable I/O System for Real-Time AC Drive Control Applications", *Lecture Notes in Computer Science*, Vol. 975, pp. 370-379, 1995, Springer-Verlag.

Xilinx; "Synopsys Interface to XACT 6000", www.xilinx.com/xapp/synopsys_flow.pdf, 1997.

Xilinx; **The Programmable Logic Data Book,** 1998.

Xing, S. e Yu, W. W. H.; "FPGA Adders: Performance Evaluation and Optimal Design", *Ieee Design & Test Of Computers*, pp. 24-29, Jan 1998.

Yu, C. e Chen, S.-J.; "VLSI Implementation of 2-D Discrete Wavelet Transform for Real-Time Video Signal Processing", *Ieee Transactions On Consumer Electronics*, Vol. 43, No. 4, pp. 1270-1279, Nov 1997a.

Yu, C. e Chen, S.-J.; "Design of an Efficient VLSI Architecture for 2-D Discrete Wavelet Transforms", *Ieee Transactions On Consumer Electronics*, Vol. 45, No. 1, pp. 135-140, Feb 1999b.

Zimmermann, R.; "VHDL Library of Arithmetic Units", *Forum on Design Languages*, 1998, www.iis.ee.ethz.ch/~zimmi.

Zimmermann, R.; Lecture Notes on Computer Arithmetic: Principles, Architectures and VLSI Design, www.iis.ee.ethz.ch/zimmi/publications/comp_arith_notes.ps.gz, 1999.

10. Apêndice

10.1. Dados de área e atraso das entidades implementadas

MUL_ADD_S

CTE	TAM_INP	TAM_OUT	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso(ns)
3	8	10	2	18	36	24	X1Y1	34,9
5	8	11	2	20	40	25	X1Y1	39,53
7	8	11	4	20	80	49	X1Y1	57,51
11	8	12	4	22	88	42	X1Y1	63,92
15	8	12	6	22	132	75	X1Y1	69,88
3	12	14	2	26	52	36	X1Y1	46,94
5	12	15	2	28	56	37	X1Y1	52,08
7	12	15	4	28	112	73	X1Y1	63,28
11	12	16	4	30	120	74	X1Y1	66,21
15	12	16	6	30	180	111	X1Y1	72,23

ADD_TAM_S

TAM_I0	TAM_I1	TAM_OUT	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	8	12	2	16	32	27	X1Y1	52,89
8	8	16	2	16	32	31	X1Y1	63,80
6	12	18	2	24	48	41	X1Y1	74,76
12	12	24	2	24	48	47	X1Y1	91,74
8	16	24	2	32	64	55	X1Y1	93,06
16	16	32	2	32	64	63	X1Y1	114,16
16	32	48	2	64	128	111	X1Y1	153,00
32	32	64	2	64	128	127	X1Y1	183,43

ADD_TAM_U

TAM_I0	TAM_I1	TAM_OUT	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	8	12	2	16	32	19	X1Y1	35,98
6	12	16	2	24	48	29	X1Y1	48,84
8	16	24	2	32	64	39	X1Y1	60,52
16	32	48	2	64	128	93	X1Y1	171,00

ADDSUB

TAM	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	3	8	24	15	X1Y1	31,43
8	3	16	48	31	X1Y1	51,3
12	3	24	72	47	X1Y1	76,41
16	3	32	96	63	X1Y1	110,95
32	3	64	192	143	X1Y1	141,1

CONST_EQ

N	I1	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	11	1	7	7	4	X1Y1	18,71
8	1	1	15	15	8	X1Y1	31,18
8	128	1	15	15	8	X1Y1	30,67
8	255	1	15	15	8	X1Y1	31,50
16	255	1	31	31	16	X1Y1	56,60
32	255	1	63	63	32	X1Y1	106,80

NEG

TAM	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	1	8	8	8	X1Y1	17,9
8	1	16	16	16	X1Y1	30,55
12	1	24	24	24	X1Y1	44,57

16	1	32	32	32	X1Y1	55,83
32	1	64	64	64	X1Y1	108,9

ROUNDER

TAM	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	1	8	8	8	X1Y1	22,70
8	1	16	16	16	X1Y1	43,18
12	1	24	24	24	X1Y1	51,29
16	1	32	32	32	X1Y1	58,19
32	1	64	64	64	X1Y1	104,21

SAT_S

TAM	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	3	8	24	16	X1Y1	26,27
8	3	16	48	24	X1Y4	29,75
12	3	24	72	31	X1Y4	41,83
16	3	32	96	39	X1Y4	51,15
32	3	64	192	71	X1Y4	51,64

SHF_REG_LD

TAM	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	1	8	8	8	X1Y1	14,5
8	1	16	16	16	X1Y1	19,39
12	1	24	24	24	X1Y1	28,35
16	1	32	32	32	X1Y1	34,23
32	1	64	64	64	X1Y1	64,64

SHF_REG_ST

TAM	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	1	8	8	8	X1Y1	12,03
8	1	16	16	16	X1Y1	17,47
12	1	24	24	24	X1Y1	22,97
16	1	32	32	32	X1Y1	27,76
32	1	64	64	64	X1Y1	45,41

VAR_EQ

TAM	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso (ns)
4	1	7	7	7	X1Y1	19,41
8	1	15	15	15	X1Y1	31,78
12	1	23	23	23	X1Y1	43,56
16	1	31	31	31	X1Y1	56,54
32	1	63	63	63	X1Y1	106,35

COUNT_ADD

TAM_REG	TAM_PAS	Largura	Altura	Área Total	Área Útil	Roteamento	Atraso(ns)
8	4	3	16	48	39	X1Y1	45,03
16	4	3	32	96	79	X1Y1	60,63
32	4	3	64	192	159	X1Y1	112,57

10.2. Publicações geradas por este trabalho

Moreira, E.; Encinas, W.; Ramos, A, "Bringing Wireless Video Into Networked Multimedia Systems", *Displays*, Vol. 17, pp.207-215, 1997

Encinas, W. S., Moreira, E.; "Hardware and Software Partition With Microcontrollers and CPLDs: A Case Study", *PDPTA'99 International Conference*, pp. 3002-3008, 1999, Las Vegas, NV.

Encinas, W. S., Moreira, E.; "Design of a Micrometric Measurement System Using Hardware/Software Partitioning", a ser publicado nos Anais do 2o Congresso de Computação Reconfigurável CORE'2000, Fundação Eurípides de Marília, 2000

Encinas, W. S., Moreira, E.; "Implementação da Transformada Wavelet Discreta Utilizando a FPGA XC6216", a ser publicado nos Anais do 2o Congresso de Computação Reconfigurável CORE'2000, Fundação Eurípides de Marília, 2000