UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

# Desenvolvimento de Arquitetura Para Sistemas de Reconhecimento Automático de Voz Baseados em Modelos Ocultos de Markov

por

**JOSÉ LUIS GÓMEZ CIPRIANO**

Tese submetida à avaliação,
como requisito parcial para a obtenção do grau
de Doutor em Ciência da Computação

Prof. Dr. Dante Augusto Couto Barone
Orientador

Prof. Dr. Sergio Bampi
Co-Orientador

Porto Alegre, novembro de 2001.

**CIP - CATALOGAÇÃO NA PUBLICAÇÃO**

Gómez Cipriano, José Luis

Desenvolvimento de Arquitetura para Sistemas de Reconhecimento Automático de Voz Baseados em Modelos Ocultos de Markov / por José Luis Gómez Cipriano. - Porto Alegre: PPGC da UFRGS, 2001.

125 p. : il.

Tese (Doutorado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR - RS, 2001. Orientador: Barone, Dante A. C. Co-Orientador: Bampi, Sergio.

1. Voz: computadores. 2. Reconhecimento de Voz 3. Processamento de Sinais 4. FPGAs. 5. VLSI. 6. Projeto Digital. I. Barone, Dante. II. Bampi, Sergio. III.Título.

# Agradecimentos

Muitas pessoas ajudaram na realização do presente trabalho. Não posso mencionar nominalmente todos, pois, com certeza, faltaria espaço para isso:

Mencionarei meu orientador o Prof. Dr. Dante Barone, por ter incentivado, confiado, acreditado e orientado o presente trabalho. Também o Prof. Dr. Sergio Bampi, pelas discussões técnicas e a co-orientação deste trabalho.

Todos os colegas e Professores do GME-UFRGS que pelo espírito de companheirismo e ajuda, fizeram mais amenas as horas de trabalho. Não mencionarei nenhum nome, pois não gostaria de ser injusto esquecendo algum.

Mencionarei o Roger Nunes, que mostrou um bom desempenho como auxiliar de pesquisa. Também os funcionários do Instituto de Informática que sempre facilitaram ambientes e recursos necessários para trabalhar, fazendo isso com boa disposição.

O CNPq que auxiliou financeiramente este trabalho, por 4 anos. Com certeza, não teria sido possível sem essa ajuda.

Cultivei muitas amizades durante os 4 anos do doutorado. Ensinaram-me muito e ajudaram a amadurecer mais. Tudo isso influiu na maneira de enfrentar os desafios, por isso quero agradecer a todos meus amigos.

Agradeço também a meus pais José Gómez e Maria Cipriano, por ter sido abnegados e pacientes, me apoiaram e incentivaram para realizar o doutorado. Estou em dívida, pelo tempo que não pude passar com eles. Assim também minha esposa, Patrícia que esteve comigo, especialmente nos momentos mais difíceis. Ela me ajudou a ser perseverante e lembrar princípios valiosos e a dar prioridade às coisas mais importantes. Ela suportou muitas vezes meu mau humor e ajudou a lidar com o desânimo, abrindo mão de muitas coisas por minha causa.

Com certeza a pessoa mais importante é Deus, pois Ele nos dá a vida e tudo o que nos rodeia vem Dele. Nenhum trabalho seria possível sem Ele. Agradeço muito.

A todos vocês,

Muito obrigado por ajudar.

# Sumário

**Lista de Figuras** ..... **7**

**Lista de Tabelas** ..... **10**

**Lista de Abreviaturas** ..... **11**

**Resumo** ..... **13**

**Abstract** ..... **14**

**1 Introdução** ..... **15**

**1.1 Sistemas de Reconhecimento Automático de Voz (RAV)** ..... **18**

**1.2 O Problema do RAV** ..... **18**

**1.3 Pré-processamento do Sinal de Voz** ..... **19**

1.3.1 Captura e Conversão A/D do Sinal de Voz ..... 19

1.3.2 Análise Espectral ..... 20

1.3.3 Análise por Banco de Filtros ..... 21

1.3.4 Análise por Predição Linear (LPC) ..... 21

1.3.5 Extração de Parâmetros ..... 22

**1.4 Codificação do Sinal de Voz** ..... **23**

**2 Identificação de Padrões de Voz** ..... **26**

**2.1 Modelagem Deterministica - DTW** ..... **26**

**2.2 Modelagem Conexionista - ANN** ..... **27**

**2.3 Modelos Ocultos de Markov** ..... **30**

2.3.1 Exemplo de *HMM* ..... 31

2.3.2 Definição de *HMMs* ..... 32

2.3.3 Os Três Problemas Básicos para HMMs ..... 35

2.3.4 Avaliação ..... 35

2.3.5 Decodificação ..... 38

2.3.6 Treinamento ..... 41

2.3.7 RAV Baseado em *HMMs* ..... 45

**3 Sistemas de Reconhecimento de Voz em Hardware** ..... **48**

**3.1 Introdução** ..... **48**

**3.2 O Sistema SBR ..... 48**
**3.3 A Arquitetura GSM ..... 51**
**3.4 O sistema UCB-DTW ..... 51**
**3.5 O Subsistema de Processamento de Palavras UCB-HMM ..... 53**
**3.6 O Decodificador de Estados UPM-HMM ..... 55**
**3.7 O Sistema UCB-Infopad ..... 56**
**3.8 Extração de Características em Terminais Remotos ..... 57**
**3.9 O Co-processador de Funções Gaussianas ..... 58**
**3.10 Conclusão ..... 58**

**4 Modelagem em SW do RAV com HMMs ..... 60**
**4.1 Introdução ..... 60**
**4.2 HMMs vs. Outras Técnicas ..... 60**
**4.3 Comparação de Parâmetros Acústicos ..... 66**
**4.4 Análise de Tempo e Complexidade do RAV/DHMM ..... 67**
**4.5 Conclusão ..... 68**

**5 Arquitetura Especifica para RAV baseado em HMMs ..... 69**
**5.1 Motivação ..... 69**
**5.2 Trabalho Proposto ..... 69**
**5.3 Tempo de Resposta ..... 72**
**5.4 Tamanho dos Circuitos e Integração ..... 72**
**5.5 Formato dos Dados ..... 72**
**5.6 Taxa de Erro ..... 73**

**6 Subsistema de Pré-ênfase, Separação em quadros e Janelamento .... 74**
**6.1 Função de Pré-ênfase ..... 74**
**6.2 Função de Separação em Quadros ..... 78**
**6.3 Função de Janelamento ..... 85**

**7 O Subsistema de Extração dos Parâmetros Mel-cepstrais ..... 90**
**7.1 Função FFT ..... 91**
**7.2 Função de Filtros Triangulares ..... 93**
**7.3 Processador Logaritmo ..... 98**
**7.4 Transformada Coseno (DCT) ..... 103**

**8 Subsistema de Reconhecimento ..... 108**
**8.1 Decodificador de Viterbi ..... 109**

**8.2 União dos Subsistemas ..... 114**

**9 Conclusões ..... 115**

**Bibliografia ..... 118**

# Lista de Figuras

FIGURA 1.1 - Sub-áreas do processamento de voz. ..... 15
FIGURA 1.2 - Tarefas de um sistema de *RAV*. ..... 16
FIGURA 1.3 - Paradigma geral do reconhecimento automático de voz [RAB 93]. ..... 18
FIGURA 1.4 - Etapas do pré-processamento do sinal de voz. ..... 19
FIGURA 1.5 - Seqüência de operações na conversão do sinal analógico de voz em um sinal digital. ..... 20
FIGURA 1.6 - Divisão em quadros e transposição entre janelas adjacentes. ..... 20
FIGURA 1.7 - Método de análise por banco de filtros. ..... 21
FIGURA 1.8 - Cálculo dos parâmetros mel-cepstrais baseados em banco de filtros. ..... 22
FIGURA 1.9 - Banco de filtros triangulares [RAB 93]. ..... 23
FIGURA 1.10 - Fluxograma do algoritmo *LBG*. ..... 24
FIGURA 1.11 - Sistema de *RAV* baseado em *VQ*. ..... 25
FIGURA 2.1 - Representação da diferença temporal. ..... 26
FIGURA 2.2 - Ajuste temporal dinâmico. ..... 26
FIGURA 2.3 - Representação do *DTW*. ..... 27
FIGURA 2.4 - Exemplos de funções de ativação. ..... 28
FIGURA 2.5 - Modelo matemático do neurônio. ..... 28
FIGURA 2.6 - Arquitetura básica de uma rede neural. ..... 29
FIGURA 2.7 - *HMM* típico de três estados. ..... 30
FIGURA 2.8 - Modelo de *N* estados de urnas e bolas. ..... 31
FIGURA 2.9 - Modelo *left-right*. ..... 34
FIGURA 2.10 - Algoritmo *forward*. ..... 37
FIGURA 2.11 - Algoritmo *forward* em operação, mostrando o valor de $\alpha_t(j)$ em cada célula. ..... 37
FIGURA 2.12 - Algoritmo *backward*. ..... 38
FIGURA 2.13 - Treliça de estados e o algoritmo de *Viterbi*. ..... 41
FIGURA 2.14 - Cálculo dos valores de $\gamma_t(i,j)$ utilizando o algoritmo *Baum-Welch*. ..... 42
FIGURA 2.15 - Um modelo oculto de *Markov* hierarquicamente estruturado. ..... 46
FIGURA 2.16 - Sistema de *RAV* para palavras isoladas sem nenhuma gramática ..... 47
FIGURA 3.1 - Arquitetura do Sistema SBR. ..... 49
FIGURA 3.2 - Arquitetura do processador de DTW-SBR. ..... 50
FIGURA 3.3 - Operação do SBR. ..... 50
FIGURA 3.4 - Diagrama de blocos da parte operativa do subsistema UCB-DTW. ..... 52
FIGURA 3.5 - Diagrama de blocos do subsistema de processamento de palavras. ..... 53

FIGURA 3.6 - Arquitetura do processador de Viterbi. .....55
FIGURA 3.7 - Arquitetura do sistema de *RAV* da UPM. .....55
FIGURA 3.8 - Sistema Infopad. .....56
FIGURA 3.9 - Aquisição da voz e extração de características em terminal remoto. .....57
FIGURA 4.1 - Taxa de reconhecimento em função do número de estados para um sistema de *RAV* baseado em *DHMM*. .....61
FIGURA 4.2 - Diagrama de blocos do sistema *LBG/HMM*. .....64
FIGURA 4.3 - Taxa de erro no reconhecimento vs. Tamanho do dicionário. .....65
FIGURA 5.1 - Diagrama do Sistema de RAV proposto. .....71
FIGURA 6.1 - Subsistema de pré-ênfase, separação em quadros e janelamento. .....74
FIGURA 6.2 - Parte operativa da função de pré-ênfase, utilizando um único somador/substrator. .....75
FIGURA 6.3 - Parte operativa da função de pré-ênfase, utilizando um somador e um substrator. .....76
FIGURA 6.4 - Deslocamento à direita para realizar a divisão por 16. .....76
FIGURA 6.5 - Simulação da pré-ênfase. .....77
FIGURA 6.6 - Superposição entre quadros. .....78
FIGURA 6.7 - Segmentação em blocos do sinal de voz. .....79
FIGURA 6.8 - Seqüência de blocos e sua armazenagem nos segmentos da memória. .....80
FIGURA 6.9 - Operação de janelamento. .....81
FIGURA 6.10 - Multiplicação dos blocos pela janela de Hamming. .....81
FIGURA 6.11 - Operações no quadro $Q_{i-1}$, ao completar o processamento do último bloco. .....82
FIGURA 6.12 - Operações no quadro $Q_i$, ao processar o primeiro bloco do quadro. .....83
FIGURA 6.13 - Operações no quadro $Q_i$, ao processar o segundo bloco do quadro. .....83
FIGURA 6.14 - Operações no quadro $Q_i$, ao processar o último bloco do quadro. .....84
FIGURA 6.15 - Algoritmo de separação em quadros do sinal de voz. .....85
FIGURA 6.16 - Janela de Hammnig. .....86
FIGURA 6.17 - Parte operativa da janela de Hamming e a divisão de quadros. .....87
FIGURA 6.18 - Endereçamento das memórias RAM e ROM das funções de janelamento e divisão de quadros .....87
FIGURA 6.19 - Componente Mem_Index. .....88
FIGURA 6.20 - Simulação da janela de Hamming. .....89
FIGURA 7.1 - Extração dos parâmetros mel-cepstrais. .....90
FIGURA 7.2 - Operações cruzadas para a *FFT* (butterfly). .....92
FIGURA 7.3 - Entrada de dados nas memorias *RAM*. .....92
FIGURA 7.4 - Enderaçamento das memórias. .....93

FIGURA 7.5 - Parte operativa da função de filtros triangulares. ..... 95
FIGURA 7.6 - Endereçamento das memórias ROM. ..... 96
FIGURA 7.7 - Máximo resultado acumulado em cada canal do banco de filtros triangulares. ..... 97
FIGURA 7.8 - Erro entre os resultados de hardware e o modelo comportamental para o circuito *triang_filters_v2* . ..... 97
FIGURA 7.9 - Erro entre os resultados de hardware e o modelo comportamental para o circuito *triang_filters_v3* . ..... 98
FIGURA 7.10 - Diagrama de blocos da operação de escalonamento. ..... 99
FIGURA 7.11 - Cálculo do valor escalonado. ..... 100
FIGURA 7.12 - Cálculo do fator de escalonamento. ..... 100
FIGURA 7.13 - Parte operativa da função CORDIC. ..... 101
FIGURA 7.14 - Dispositivos *Shift_X* e *Shift_Y*. ..... 102
FIGURA 7.15 - Comparação do software com o hardware da função logaritmo. ..... 103
FIGURA 7.16 - Erro relativo produzido pela função logaritmo em relação ao modelo em software, para o circuito log_v2. ..... 103
FIGURA 7.17 - Parte operativa da função DCT. ..... 105
FIGURA 7.18 - Conexão entre memórias. ..... 105
FIGURA 7.19 - Erro relativo entre hardware e software para o circuito *dct_v2*. ..... 106
FIGURA 7.20 - Erro relativo entre hardware e software para o circuito *dct_v3*. ..... 107
FIGURA 8.1 - Quantização dos vetores de parâmetros mel-cepstrais. ..... 108
FIGURA 8.2 - Identificação da seqüência. ..... 109
FIGURA 8.3 - HMM left-right utilizado. ..... 109
FIGURA 8.4 - Parte operativa do subsistema de reconhecimento. ..... 112
FIGURA 8.5 - Endereçamento do subsistema de reconhecimento. ..... 113
FIGURA 8.6 - Simulação do subsistema de reconhecimento. ..... 113
FIGURA 8.7 – União dos subsistemas. ..... 114

# Lista de Tabelas

TABELA 3.1 - Sistemas de *RAV* em hardware. ..... 59
TABELA 4.1 - Taxas de reconhecimento, para sistemas de palavras isoladas. ..... 61
TABELA 4.2 - Taxas de reconhecimento, para a identificação de locutor. ..... 62
TABELA 4.3 - LBG/HMM vs. MSVQ/NN. ..... 64
TABELA 4.4 - Resultados individuais de *RAV* utilizando *VQ/HMM* [ZHA 95] ..... 66
TABELA 4.5 - Taxas de reconhecimento, comparando parâmetros acústicos. ..... 66
TABELA 4.6 - Tempo de processamento das etapas do RAV/DHMM. ..... 67
TABELA 4.7 - Complexidade dos algoritmos de RAV/DHMM. ..... 67
TABELA 6.1 - Comparação entre circuitos de pré-ênfase. ..... 77
TABELA 6.2 - Resultados da síntese da pré-ênfase, o algoritmo de divisão em quadros e o janelamento operando em conjunto. ..... 89
TABELA 7.1 - Síntese da função *FFT*. ..... 93
TABELA 7.2 - Banco de filtros triangulares. ..... 94
TABELA 7.3 - Resultados da síntese da função de filtros triangulares. ..... 96
TABELA 7.4 - Resultados da síntese da função CORDIC para o cálculo de logaritmos. 102
TABELA 7.5 - Síntese da DCT. ..... 106
TABELA 7.6 - Síntese da FFT e da função de extração de parâmetros mel-cepstrais. 107
TABELA 8.1 - Comparação entre subsistemas de reconhecimento. ..... 111
TABELA 8.2 – Tempo de resposta. ..... 114

# Lista de Abreviaturas

| | |
|---|---|
| A/D | *Analógico – Digital* |
| ANN | *Artificial Neural Network* |
| AVR | *Automatic Voice recognition* |
| CDPD | *Clock and Data Precharged Dynamic circuit technique* |
| CHMM | *Continuous Hidden Markov Model* |
| CPU | *Central Process Unit* |
| DARPA | *Defense Advanced Projects Research Agency* |
| DCT | *Discrete Cosine Transform* |
| DFT | *Discrete Fourier Transform* |
| DHMM | *Discrete Hidden Markov Model* |
| DRMD | *DARPA Resources Management Database* |
| DSP | *Digital Signal Processing* |
| DTW | *Dynamic Time Warping* |
| EM | *Expectation-Maximization* |
| FFT | *Fast Fourier Transform* |
| FIR | *Finite Impulse Response* |
| FPGA | *Field Programmable Gate Array* |
| GMM | *Gaussian Mixture Model* |
| HMM | *Hidden Markov Model* |
| IIR | *Infinite Impulse Response* |
| LBG | *Algoritmo Linde-Buzo Gray* |
| LPC | *Linear Prediction Coding* |
| MAC | *Função multiplica-acumula* |
| MFCC | *Mel frequency cepstral coefficients* |
| MGC | *Multiple Gaussian Clustering* |
| MHMM | *Multiple Hidden Markov Model* |
| MIT | *Massachusetts Institute of Technology* |
| ML | *Maximum Likelihood* |
| MLP | *Multilayer Perceptron* |
| Pdf | *Probability density function* |
| RAV | *Reconhecimento Automático de Voz* |
| SD | *Speaker Dependent* |

| | |
|---|---|
| SI | *Speaker Independent* |
| SNR | *Single-to-Noise-Ratio* |
| SOG | *Sea of Gates* |
| TDNN | *Time-Delay Neural Network* |
| TI | *Texas Instrument* |
| TSPC | *True Single Phase Clocking circuit technique* |
| TTL | *Transistor Transistor Logic* |
| UCB | *Universidade da California em Berkeley* |
| UPM | *Universidade Politécnica de Madri* |
| VLSI | *Very Large Scale of Integration* |
| VQ | *Vector Quantization* |

# Resumo

Este trabalho foi realizado dentro da área de reconhecimento automático de voz (*RAV*). Atualmente, a maioria dos sistemas de *RAV* é baseada nos modelos ocultos de *Markov* (*HMMs*) [GOM 99] [GOM 99b], quer utilizando-os exclusivamente, quer utilizando-os em conjunto com outras técnicas e constituindo sistemas híbridos. A abordagem estatística dos *HMMs* tem mostrado ser uma das mais poderosas ferramentas disponíveis para a modelagem acústica e temporal do sinal de voz.

A melhora da taxa de reconhecimento exige algoritmos mais complexos [RAV 96]. O aumento do tamanho do vocabulário ou do número de locutores exige um processamento computacional adicional. Certas aplicações, como a verificação de locutor ou o reconhecimento de diálogo podem exigir processamento em tempo real [DOD 85] [MAM 96]. Outras aplicações tais como brinquedos ou máquinas portáveis ainda podem agregar o requisito de portabilidade, e de baixo consumo, além de um sistema fisicamente compacto. Tais necessidades exigem uma solução em hardware.

O presente trabalho propõe a implementação de um sistema de *RAV* utilizando hardware baseado em *FPGAs* (*Field Programmable Gate Arrays*) e otimizando os algoritmos que se utilizam no *RAV*. Foi feito um estudo dos sistemas de *RAV* e das técnicas que a maioria dos sistemas utiliza em cada etapa que os conforma. Deu-se especial ênfase aos Modelos Ocultos de Markov, seus algoritmos de cálculo de probabilidades, de treinamento e de decodificação de estados, e sua aplicação nos sistemas de *RAV*.

Foi realizado um estudo comparativo dos sistemas em hardware, produzidos por outros centros de pesquisa, identificando algumas das suas características mais relevantes.

Foi implementado um modelo de software, descrito neste trabalho, utilizado para validar os algoritmos de *RAV* e auxiliar na especificação em hardware.

Um conjunto de funções digitais implementadas em *FPGA*, necessárias para o desenvolvimento de sistemas de *RAV* é descrito. Foram realizadas algumas modificações nos algoritmos de *RAV* para facilitar a implementação digital dos mesmos. A conexão, entre as funções digitais projetadas, para a implementação de um sistema de reconhecimento de palavras isoladas é aqui apresentado. A implementação em *FPGA* da etapa de pré-processamento, que inclui a pré-ênfase, janelamento e extração de características, e a implementação da etapa de reconhecimento são apresentadas finalmente neste trabalho.

**Palavras-chave:** RAV, HMM, reconhecimento de voz, modelagem acústica, fala, FPGAs.

**TITLE:** "SPECIFIC ARCHITECTURE FOR AUTOMATIC VOICE RECOGNITION SYSTEMS BASED IN HIDDEN MARKOV MODELS"

# Abstract

This work was realized in the automatic voice recognition (*AVR*) domain. Currently, most *AVR* systems are based on *Hidden Markov Models* (*HMMs*), used alone or with other approaches and forming hybrid systems. The *HMMs* statistical approach has been shown one of the most powerful statistical tools available for modeling speech signal.

A better recognition rate requires more complex recognition algorithms, which increments computational cost. The increasing of the vocabulary or the number of speakers requires more computational processing. Some applications, like speaker verification or dialog recognition could require real time processing. Other applications such as toys or portable machines still could aggregate the portability and low-power requirements; join with a physically compact system. These requirements could require a hardware solution.

The current work proposes the implementation of an *AVR* system using hardware based in *FPGAs (Field Programmable Gate Arrays)* and optimizing the *AVR* algorithms. It was made a study of *AVR* systems and the techniques that most systems use in each stage. A special emphasis on *Hidden Markov Models*, their probability calculus, training, state decoding algorithms, and their applications in *AVR* systems was given.

A comparative study of hardware systems produced by other research centers was made, identifying some of their more relevant features.

A software model implemented and described in this work was used for validating the *AVR* algorithms and help in the hardware specification.

A set of digital functions implemented in *FPGA*, and necessary for the developing of *RAV* systems are described. Some modifications in the *AVR* algorithms were made for facilitate their digital implementation. The connection between the digital functions designed, for implementation of an isolated recognition system is presented here. The *FPGA* implementation of the pre-processing stage, which includes the pre-emphasis, windowing and parameter extraction, and the implementation of the recognition stage are finally shown in this work

**Keywords:** AVR, HMM, voice recognition, acoustic modeling, speech, FPGAs.

# 1 Introdução

É cada vez maior o interesse do homem interagir de forma mais natural com as máquinas. A comunicação entre os seres humanos é feita de diversas maneiras, entre elas, gestos, textos impressos, desenhos, e voz. Entretanto, a voz certamente é um modo comum e natural de comunicação mais freqüentemente utilizado entre as pessoas. O processamento de voz, que auxilia neste tipo de comunicação, é formado por diversas subáreas, assim como mostrado na fig. 1.1. Entre estas áreas podemos mencionar a análise e síntese da voz o reconhecimento de voz e a codificação da voz [BAK 75] [CAM 97] [ALA 98].

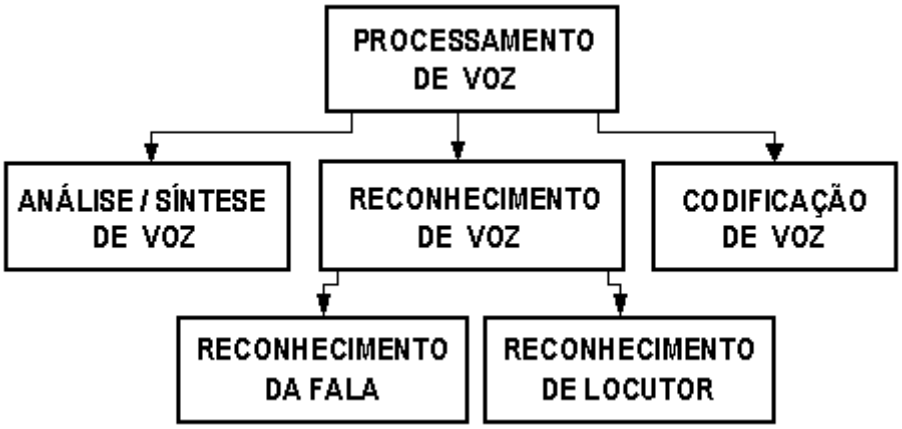



FIGURA 1.1 - Sub-áreas do processamento de voz.

O desenvolvimento de sistemas para a comunicação homem-máquina através da voz constitui uma área de pesquisa de grande potencial [SCH 94]. No entanto, a meta ambiciosa de dotar uma máquina com a capacidade de falar e entender naturalmente ainda se encontra distante.

Uma tarefa importante na comunicação homem-máquina através da voz é o reconhecimento da mensagem falada. Sistemas que se encarregam dessa tarefa são conhecidos pelo nome de sistemas de reconhecimento automático de voz (*RAV*) [FOS 93] [GOM 98].

O reconhecimento de voz é uma habilidade natural do ser humano, mas é uma tarefa difícil para os computadores e estações de trabalho atuais. No mercado, podem encontrar-se diversos produtos que vão de sistemas de reconhecimento de palavras isoladas e vocabulário pequeno a sistemas de reconhecimento de linguagem contínua e grandes vocabulários [LUF 94] [WAN 97] [GOM 98] [STR 98]. Porém, o computador que reconheça voz sem restrições de locutores, ambientes e vocabulário é ainda um sonho [ZUE 85] [PIC 95] [DES 97]. Este sonho tem envolvido algumas gerações de estudiosos de fonética e engenheiros dentro desta área.

Como todos os sons, a voz é uma onda analógica e contínua. Para que um sistema de *RAV* utilize os dados de voz, as formantes, os padrões de ruído, os silêncios, e os efeitos de coarticulação, devem ser capturados e convertidos ao formato digital. Este processo de conversão é feito através de técnicas de processamento digital de sinais (*DSP* - ***D**igital **S**ignal **P**rocessing*). Alguns sistemas incluem hardware para executar essa conversão. Outros sistemas utilizam a capacidade de processamento de sinais de

outros produtos como, por exemplo, placas de som e áudio digital [FOS 93]. O processamento digital de sinais (*DSP*) se refere à obtenção de uma representação discreta de sinais e à teoria, projeto e implementação de métodos numéricos para processar tal representação discreta.

Entre as aplicações atuais das técnicas de *DSP*, podemos mencionar: melhoramento da voz, codificação da voz, reconhecimento de voz, compressão de vídeo, reconhecimento de padrões visuais, etc.

Uma vez em formato digital, é necessário gerar um conjunto de parâmetros que representem informações relevantes do sinal de voz. A parametrização gerada deve ser robusta, quer dizer, o máximo possível invariante ao canal de transmissão, ao ruído de fundo, ao transdutor e ao locutor.

As aplicações para processamento digital de sinais ocupam em torno de 30% do mercado de circuitos integrados [MAD 95] [LAP 97]. A evolução da tecnologia *VLSI* tem sido uma grande aliada do progresso nos sistemas *DSP*. A execução de tarefas em tempo real torna necessária a implementação de sistemas *DSP* em silício [LEE 95] [VIS 95].

Um sistema de reconhecimento automático de voz (*RAV*) é um sistema capaz de transformar um sinal de voz em uma seqüência de dados com a qual uma máquina tomará decisões [GOM 99d]. Os sistemas de *RAV* executam três tarefas básicas, assim como mostrado na figura 1.2:

(1) **Pré-processamento**, que inclui a conversão *A/D*, filtragem e extração de parâmetros do sinal de voz. Identificam-se os componentes principais da representação e eliminam-se informações redundantes. Nesta etapa, estão incluídos alguns processamentos para melhoria da qualidade do sinal, devido à degradação do mesmo por alguns fatores como: variação da amplitude, ruído no canal, entre outros;

(2) **Identificação** da informação contida no sinal de voz. Esta tarefa tem como função a identificação dos padrões com representações existentes daqueles padrões;

(3) **Comunicação**, ou envio da informação à aplicação que tomará decisões [MAR 96].

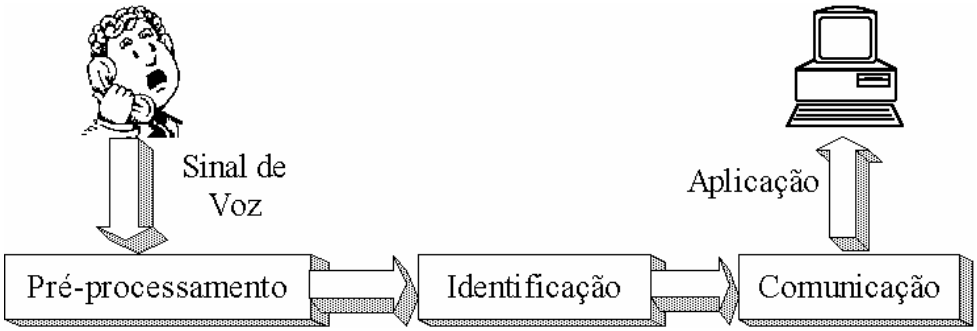



FIGURA 1.2 - Tarefas de um sistema de *RAV*.

Em relação aos aspectos computacionais, há grande interesse de pesquisa nas técnicas relacionadas à segunda tarefa, isto é o reconhecimento do sinal propriamente dito. Atualmente, na fronteira do estado da arte de sistemas de *RAV*, modelos estatísticos são extensivamente pesquisados e utilizados [GOM 99]. Esses modelos consideram que o sinal de voz pode ser muito bem caracterizado como um processo paramétrico estocástico.

A abordagem estatística mais amplamente utilizada e que vem apresentando os melhores resultados, nos sistemas de *RAV*, está baseada nos **Modelos Ocultos de *Markov*** ou *HMM* (***H**idden **M**arkov **M**odels*) [RAB 93] [DEL 93] [NAR 96]. Os modelos ocultos de *Markov* modelam o processo de geração da voz através de uma máquina de estados estocástica. Os sons da voz, ou parâmetros derivados a partir desses sons, são gerados segundo as funções de distribuição de probabilidades correspondentes a cada estado do *HMM*. Este método fornece uma maneira natural e altamente confiável de reconhecer a voz em diversas aplicações práticas [BUR 96] [NUN 96].

Neste contexto, desenvolve-se o presente trabalho, que abrange:

a) Estudo e implementação de um modelo eficiente em software para reconhecimento de voz, utilizável em aplicações industriais.

b) Estudo e pesquisa de algoritmos e técnicas de processamento digital de sinais, utilizadas no reconhecimento de voz, visando a sua implementação em *FPGAs*.

c) A concepção de um sistema de aplicação específico, para o reconhecimento de voz, baseado em *FPGAs*.

d) Dado o papel preponderante desta abordagem em *RAV* e devido ao fato de que o mesmo abre um enorme campo de investigação inovadora, propõe-se a utilização dos Modelos Ocultos de *Markov*, propondo e fornecendo alternativas para um desempenho ótimo em tempo real, estabelecendo-se comparativamente vantagens e desvantagens.

## 1.1 Sistemas de Reconhecimento Automático de Voz (RAV)

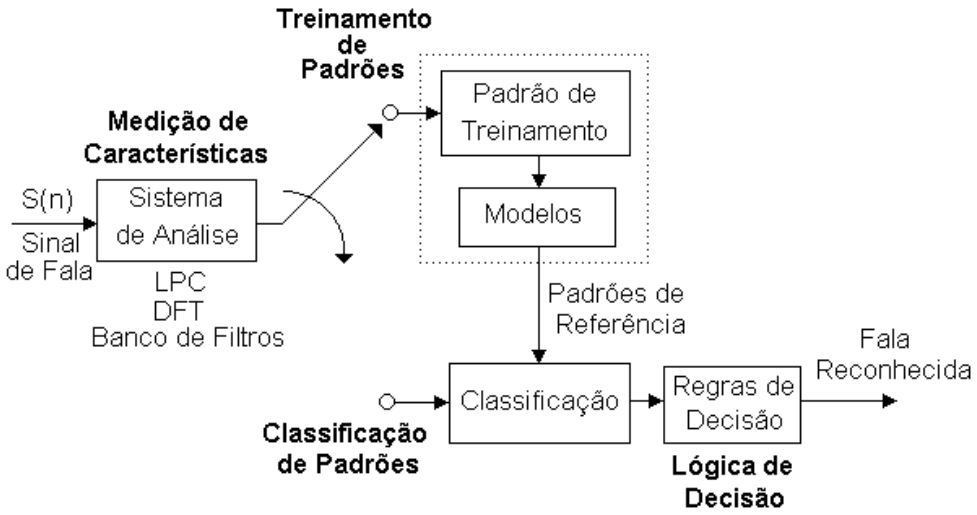



FIGURA 1.3 - Paradigma geral do reconhecimento automático de voz [RAB 93].

A figura 1.3 mostra o paradigma geral de reconhecimento de padrões aplicado ao *RAV* [RAB 93]. A extração de características obtém uma representação do sinal de voz, eliminando informações redundantes e gerando os padrões que o *RAV* utilizará [LEV 85]. No treinamento, se utilizam um ou mais padrões da mesma classe para criar um padrão ou modelo representativo dessa classe, o qual é armazenado. Na etapa de classificação de padrões, o padrão desconhecido é comparado com cada padrão de referência, obtendo uma medida de similaridade, e escolhendo o padrão de referência vencedor.

## 1.2 O Problema do RAV

O *RAV* de **palavras isoladas**, onde as palavras estão separadas por uma pausa, é uma tarefa mais simples do que o *RAV* de **fala contínua**, onde os limites das palavras não são facilmente distinguíveis [STR 98]. Na fala contínua, há confusão adicional entre palavras e frases, o espaço de busca é maior e a produção de fonemas e palavras é afetada pelos fonemas e palavras vizinhos (**coarticulação**) [TEB 95].

Vocabulários grandes requerem uma grande capacidade de armazenamento e alto custo computacional. Uma solução é não utilizar modelos de palavras, mas modelos de subpalavras (fonemas, grupos de fonemas ou sílabas) [MAR 96].

O *RAV* **dependente do locutor** tem boa precisão, mas requer treinamento para cada novo usuário o que pode ser inconveniente em certas aplicações. Em contraste, o *RAV* **independente do locutor** que reconhece a voz de usuários para os quais não houve treinamento, tem precisão menor, pois os parâmetros característicos da voz dependem fortemente do usuário (idade, sexo, sotaque, velocidade da fala, etc.) [PEA 90] [CAM 97].

O *RAV* também é afetado por ruído, distorções acústicas e pelo microfone/telefone (tipo, direcionamento e posição). As restrições na seqüência de palavras, também afetam o desempenho do *RAV*, sendo representadas por uma

**gramática**, que serve como filtro, com a finalidade de avaliar unicamente as sentenças possíveis [PIC 93].

Outro desafio no *RAV* é a variabilidade da voz em um único locutor e também para locutores diferentes. Existem duas classes de variabilidade: **acústica**, referente aos diversos acentos, pronunciações, entonações, volumes, etc. e **temporal**, mais fácil de controlar, referente às diversas velocidades de fala [TEB 95].

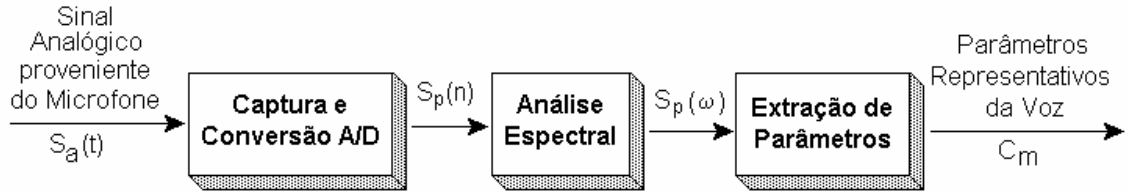



FIGURA 1.4 - Etapas do pré-processamento do sinal de voz.

# 1.3 Pré-processamento do Sinal de Voz

O pré-processamento do sinal de voz, que visa gerar um conjunto de parâmetros que contém informações relevantes à diferenciação de seus eventos, é o primeiro passo do processo de *RAV*. O pré-processamento do sinal de voz pode ser dividido em 3 tarefas, assim como mostrado na figura 1.4. Estas tarefas são: (1) Captura e conversão *A/D*; (2) Análise espectral e (3) Extração de parâmetros.

## 1.3.1 Captura e Conversão A/D do Sinal de Voz

A conversão *A/D* consiste na amostragem do sinal analógico, $S_a(t)$, cada $T$ segundos e a quantização das amostras, para obter o sinal digital $S(n)=S_a(n.T)$, $n=0, 1,...$ Dada uma onda analógica contínua de largura de banda finita (onde a máxima freqüência presente é $f_{max}$), a informação de entrada somente será preservada com uma freqüência de amostragem $f_s \geq 2.f_{max}$ (critério de *Nyquist*). Para evitar o *aliasing* [RAB 93], o sinal de voz resultante do transdutor é, em uma primeira etapa, passado através de um filtro analógico, passa-baixas, com freqüência de corte $f_c \leq f_s/2$, que elimina freqüências acima da metade da freqüência de amostragem. Posteriormente, o sinal passa por um conversor *A/D*.

O sinal de voz digitalizado passa através de um **filtro de pré-ênfase** com função de transferência:

$$H_{pre}(z) = 1 - a_{pre}.z^{-1} \tag{1.1}$$

onde $0,9 \leq a_{pre} \leq 1$ é o coeficiente de pré-ênfase [RAB 93]. Este filtro é utilizado para equalizar o espectro de voz e melhorar o desempenho da análise espectral, que constitui a etapa posterior [PIC 93]. A saída $S_p(n)$, do filtro de pré-ênfase, está relacionada com a entrada, $S(n)$, através da equação:

$$S_p(n) = S(n) - a_{pre} \cdot S(n-1) \tag{1.2}$$

As tarefas mencionadas, são esquematizadas na figura 1.5.

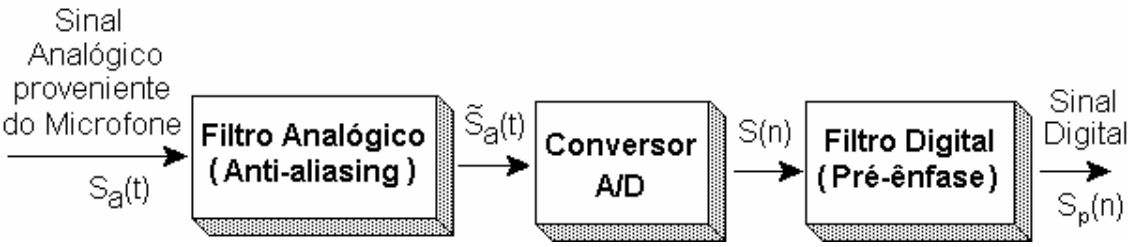



FIGURA 1.5 - Seqüência de operações na conversão do sinal analógico de voz em um sinal digital.

## 1.3.2 Análise Espectral

A figura 1.6 ilustra os passos necessários para a realização da análise espectral. Inicialmente, o sinal digital é dividido em quadros de duração fixa (l0ms~30ms), ***Tf***. A análise espectral se realiza sobre um intervalo de tempo ***Tw*** (duração da janela), existindo geralmente superposição entre janelas adjacentes ($T_w > T_f$) com uma porcentagem de superposição, $\%S = 100.(T_w - T_f)/T_w$, que varia de 0% a 70% na maioria dos sistemas. Uma *%S* alta permite uma transição mais suave dos parâmetros extraídos [NUN 96]. Porém, estimativas excessivamente suavizadas podem ocultar variações reais do sinal de voz. As descontinuidades no início e final de cada quadro são minimizadas aplicando janelas com bordas suaves (*Hamming*, *Kaiser,* etc.) [RAB 93].

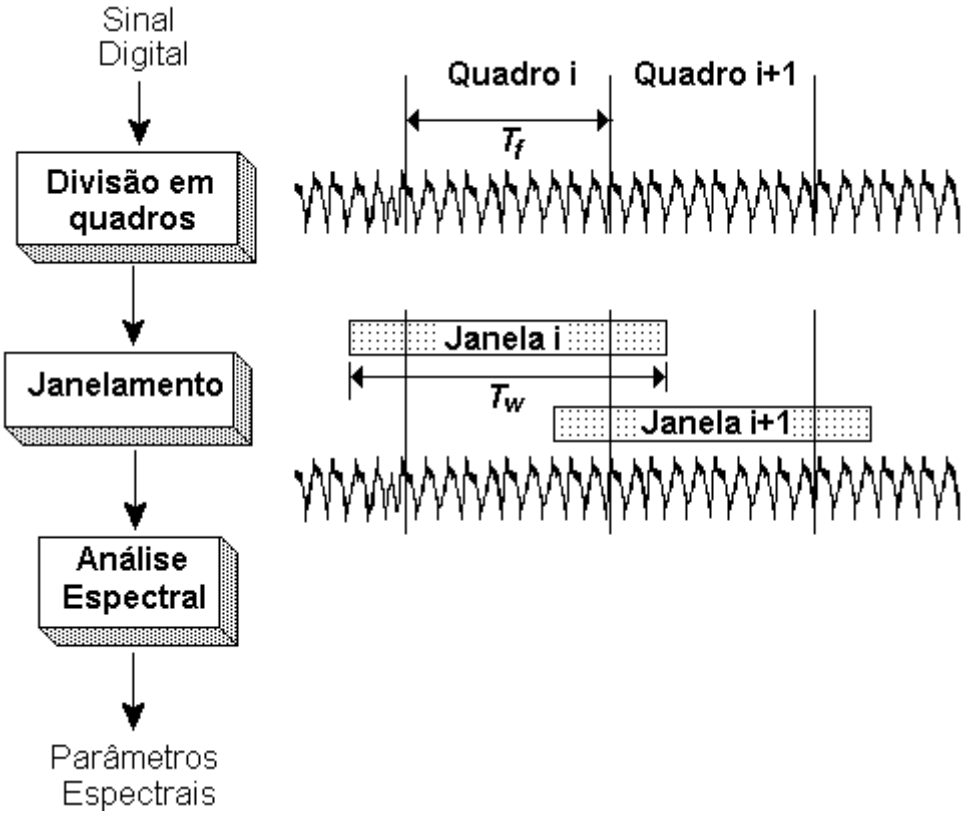



FIGURA 1.6 - Divisão em quadros e transposição entre janelas adjacentes.

Nos sistemas de *RAV* atuais predominam dois métodos de análise espectral: o **banco de filtros** a partir da *FFT* (***F**ast **F**ourier **T**ransform*) e a **codificação preditiva linear** *(LPC* ou ***L**inear **P**redictive **C**oding*). Os espectros resultantes de ambos métodos

são representações mais correlacionadas ao processo de audição e percepção humana que a representação temporal inicial, o que justifica a utilização de parâmetros extraídos a partir de tais representações espectrais [DEL 93] [PIT 96].

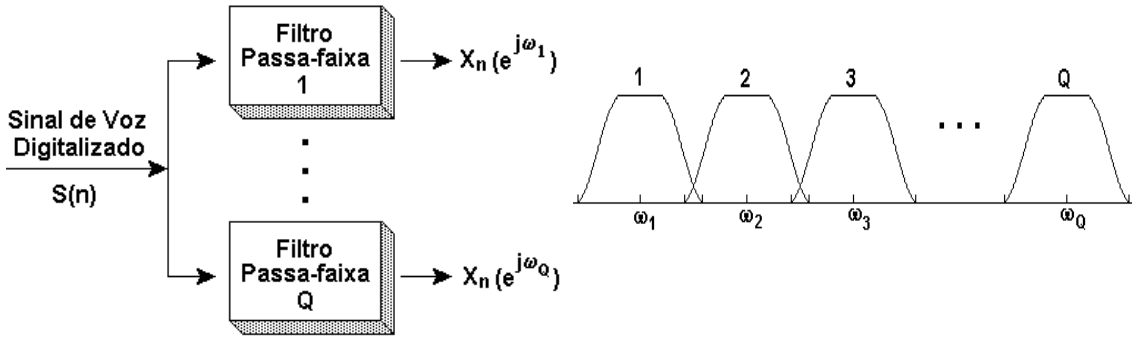
### 1.3.3 Análise por Banco de Filtros

A figura 1.7 (a) mostra a estrutura geral da técnica de banco de filtros. O sinal digitalizado de voz, *S(n)*, é processado através de um banco constituído de *Q* filtros passa-faixa que abrangem segmentos da faixa de freqüências de interesse. Além disso, pode existir uma superposição entre os filtros, no domínio da freqüência, como mostrado na figura 1.7 (b).

Freqüentemente se realizam mapeamentos de freqüências que utilizam uma escala não uniforme, com significado perceptual [PIC 93]. Uma escala deste tipo é a escala ***Mel***, que pode ser obtida para uma determinada freqüência através da seguinte expressão:

$$f_{mel}=2595.log_{10}(1+f/700) \tag{1.3}$$

onde $f_{mel}$ é a freqüência (real) em *mels*.




(a) Diagrama da análise por banco de filtros com Q canais.

(b) Resposta em freqüência de um banco de filtros de Q canais.

FIGURA 1.7 - Método de análise por banco de filtros.

### 1.3.4 Análise por Predição Linear (LPC)

A amostra *s(n)* pode ser aproximada como a combinação linear das *p* amostras passadas:

$$s(n)\approx a_1.s(n-1)+...+a_p.s(n-p) \tag{1.4}$$

onde os valores $a_1,...,a_p$ considerados constantes sobre a janela analisada, são os coeficientes *LPC* (Linear Predictive Coding). A autocorrelação que permite minimizar o erro quadrático médio entre o sinal modelado e o sinal original para obter os coeficientes *LPC* [RAB 78] [RAB 93], é calculada para cada quadro *l* como:

$$r_l(m)=\sum_{n=0}^{N-1-m}\bar{x}_l(n)\cdot\bar{x}_l(n+m) \quad , \quad m=0,1,...,p \tag{1.5}$$

onde o atraso máximo no cálculo da autocorrelação, *p*, é a ordem da análise *LPC* e tem valores típicos de 10 a 16 [RAB 93]. Os *p+l* valores da função de autocorrelação são convertidos em um conjunto de parâmetros *LPC*, utilizando o método de *Durbin*:

*Método de Durbin*

*INICIO:* $E^{(0)}=r(0)$

*RECURSÂO: Para* $1 \le i \le p$

$$k_i = \frac{r(i) - \sum_{j=1}^{i-1} \alpha_j^{i-1}.r(i-j)}{E^{(i-1)}},$$

*Para* $1 \le j \le i\text{-}1$

$$\alpha_i^{(i)} = k_i,\ \alpha_j^{(i)} = \alpha_j^{(i)} - k_i.\alpha_{i-j}^{(i-1)},$$

$$E^{(i)}=(1-k_i^2).\ E^{(i-1)}$$

O método de *Durbin* é aplicado recursivamente para *i=1,..., p,* e como solução final se obtêm os coeficientes *LPC* para uma análise de ordem *p*, dados por $a_m = \alpha_m^{(p)}$, *m=1...p,* resultantes da modelagem realizada sobre uma janela de voz.

## 1.3.5 Extração de Parâmetros

Após a realização de uma análise espectral do sinal de voz, é preciso determinar um conjunto de parâmetros capaz de diferenciar eventos da voz.

A seleção da melhor representação paramétrica dos dados acústicos é uma tarefa importante no projeto de qualquer sistema de *RAV*. O objetivo fundamental ao escolher uma representação paramétrica é comprimir os dados de voz eliminando informação não pertinente à análise fonética dos dados e salientar aqueles aspectos do sinal que contribuem significativamente à detecção das diferenças fonéticas [DAV 80].

Os parâmetros extraídos do sinal de voz devem ser o máximo possível invariantes ao canal de transmissão, ruído de fundo, transdutor e locutor, reduzindo as taxas computacionais sem muita perda de informação [DAV 80].

A figura 1.8 mostra como obter os *parâmetros mel-cepstrais*, através de uma análise espectral com um banco de filtros linearmente espaçados na escala *mel*. É calculado o espectro do sinal de voz que foi separado em quadros através de janelas. O espectro resultante é passado através se um banco de filtros.

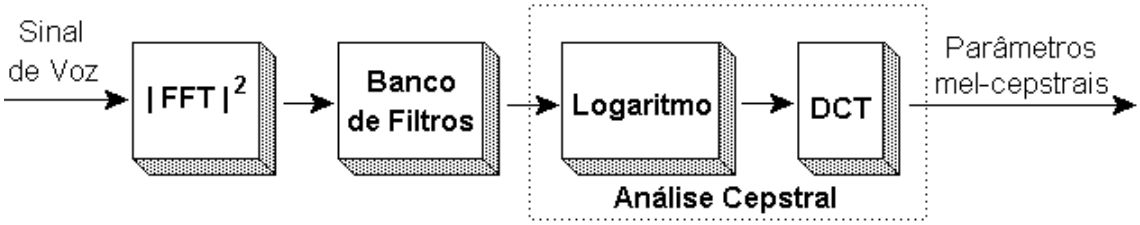



FIGURA 1.8 - Cálculo dos parâmetros mel-cepstrais baseados em banco de filtros.

A figura 1.9 mostra tal banco de filtros passa-banda, de resposta triangular com espaçamento e largura determinados por um intervalo constante de freqüência *mel*.

Calculam-se os logaritmos da energia nas saídas dos filtros e a Transformada Coseno Discreta (*DCT*) de tais valores, obtendo os parâmetros desejados $c^*_n$:

$$c^*_n = \sum_{k=1}^{K} (log(S^*_k) \cdot cos\left[ n \cdot \left( k - \frac{1}{2} \right) \cdot \frac{\pi}{K} \right], \; n=1,..., \; L \qquad (1.6)$$

onde $L$ é o número de coeficientes, $S^*_k(\omega)$, $k=1,...,K$, são os coeficientes de potência da saída do k-ésimo filtro e $S(\omega)$ é a entrada.

O número de parâmetros utilizados não necessariamente é igual ao número de filtros do banco de filtros. Freqüentemente, são descartadas algumas das últimas amostras obtidas na saída da *DCT*, as quais contém pouca informação sobre a forma do trato vocal utilizada para produzir o trecho de voz em análise [DAV 80].

Outros parâmetros existentes podem ser obtidos a partir dos coeficientes *LPC* e primeiras e segundas derivadas entre outros, que ajudam a caracterizar as mudanças rápidas no som de voz [PIC 93].

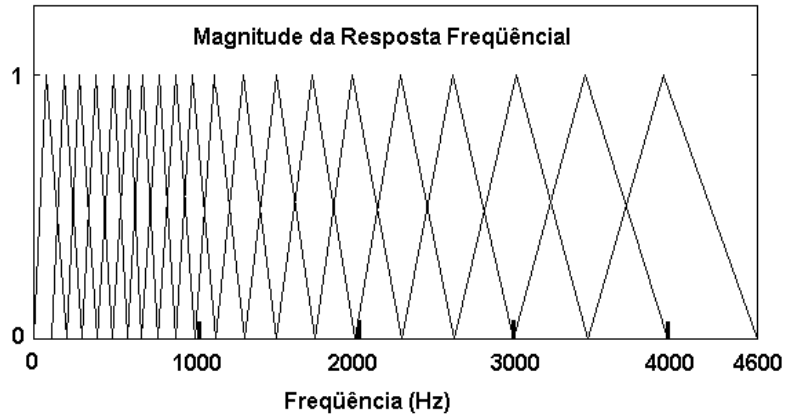



FIGURA 1.9 - Banco de filtros triangulares [RAB 93].

# 1.4 Codificação do Sinal de Voz

Uma vez extraídos os parâmetros do sinal de voz, procede-se à redução da informação utilizando algum método de codificação. A quantização vetorial (*VQ*) é uma técnica de compressão de dados que mapeia um conjunto de vetores de observação $\mathbf{X}=\{\mathbf{x_t} \in \Re^n / \; i=1...T\}$, gerando um conjunto $\mathbf{C}=\{\mathbf{y_i} \in \Re^n / \; i=1...N\}$ de $N$ vetores-código [SOO 87]. O conjunto **C** é o dicionário (*codebook*) do quantizador. O mapeamento é realizado calculando a distorção, entre cada vetor de observação $\mathbf{x_t} \in \mathbf{X}$ e os vetores-código $\mathbf{y_i} \in \mathbf{C}$, codificando $\mathbf{x_t} \in \mathbf{X}$ pelo índice $i$ do vetor-código $\mathbf{y_i} \in \mathbf{C}$ que apresenta a menor distorção [MAK 85] [WEI 92]. O objetivo do treinamento na *VQ* é gerar o melhor conjunto de vetores-código **C** que produza a menor distorção. Para tanto, existem diversos algoritmos.

A figura 1.10 mostra uma variante do algoritmo *LBG* (*Linde-Buzo-Gray*) [LEE 89]. Este algoritmo separa os dados de treinamento em *2,4,...,N* partições, com os centróides das partições representando os vetores-código desejados.

A partir de um conjunto de $M$ vetores-código $\boldsymbol{V}_i^M$, $i=1,...,M$, que represente uma partição dos vetores de treinamento, obtém-se $2M$ vetores $\boldsymbol{V}_i^{2M}$, $i=1,...,2M$, mediante algum método heurístico como:

$$\begin{aligned} V_i^{2M} &= V_i^M + 0{,}01 \cdot V_i^M \quad , \qquad i = 1,..., M \\ V_i^{2M} &= V_i^M - 0{,}01 \cdot V_i^M \quad , \qquad i = M+1,..., 2M \end{aligned} \tag{1.7}$$

Determinam-se, então, as $2M$ partições e seus centróides através de refinamento iterativo: cada vetor de treinamento é classificado como pertencente à partição cujo centróide apresenta a menor distorção em relação a este vetor. Em seguida, calcula-se um novo centróide para cada partição através da média de todos os vetores de treinamento correspondentes. A inicialização do algoritmo ($M=1$) é realizada calculando o centróide dos vetores de treinamento. O algoritmo termina quando o conjunto de treinamento é dividido em $N$ partições, e os $N$ centróides correspondentes convergem, sendo armazenados como os vetores-código do *dicionário* desejado.

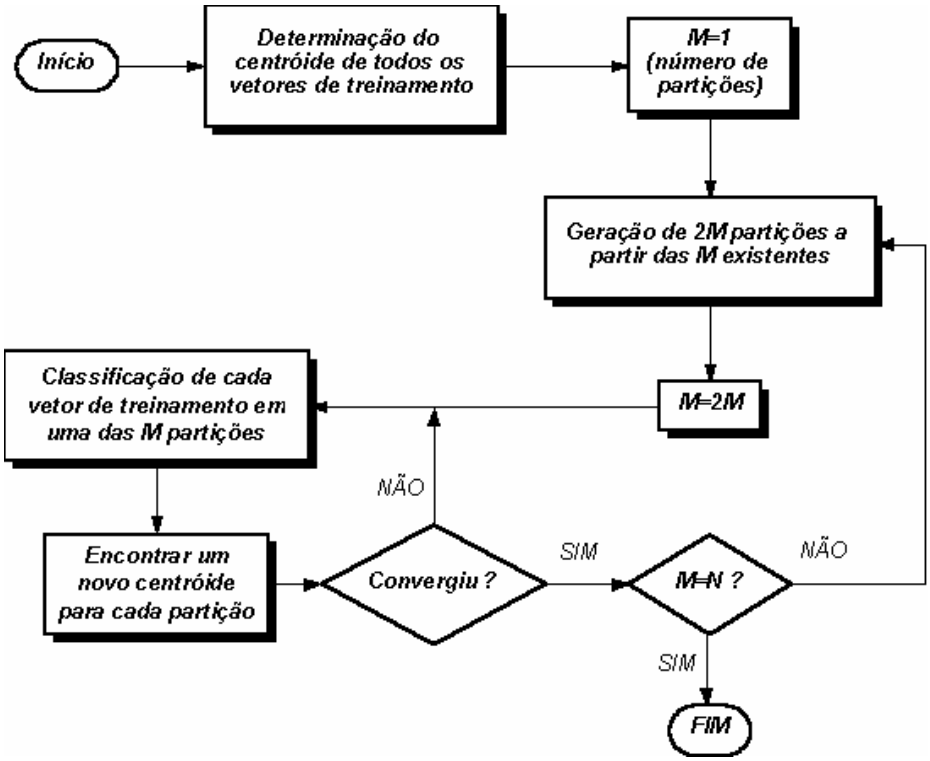



FIGURA 1.10 - Fluxograma do algoritmo *LBG*.

A *VQ* pode ser utilizada como técnica de reconhecimento [RAB 93], como indicado na figura 1.11. No treinamento, são gerados dicionários-modelo como padrões de referência. Na etapa de reconhecimento, o vetor de teste é utilizado para gerar um dicionário e a distorção em relação a cada um dos dicionários-modelo é calculada. O padrão ganhador é aquele que apresenta a menor distorção.

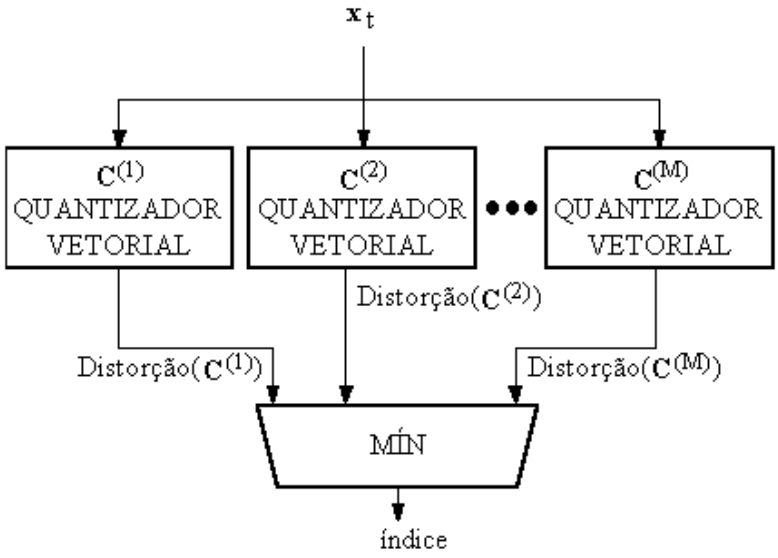



FIGURA 1.11 - Sistema de *RAV* baseado em *VQ*.

# 2 Identificação de Padrões de Voz

Uma vez realizado o pré-processamento da voz procedente de um determinado usuário, o sistema de *RAV* está preparado para realizar sua função primária: identificar aquilo que o usuário tenha dito. Basicamente, três grandes métodos são utilizados em *RAV* [RAB 93] e serão descritos a seguir.

## 2.1 Modelagem Deterministica - DTW

Esta modelagem utiliza os parâmetros do sinal de voz como padrões de teste e de referência, junto com uma métrica apropriada para a comparação dos mesmos, tendo baixo custo [SAK 78].

A diferença temporal entre ambos padrões causada pela compressão ou expansão de fonemas e pausas dentro das frases (figura 2.1), requer um alinhamento dos padrões, utilizando algoritmos de Ajuste Temporal Dinâmico (*DTW*) (ver figura 2.2).

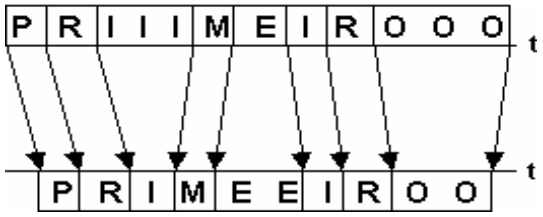
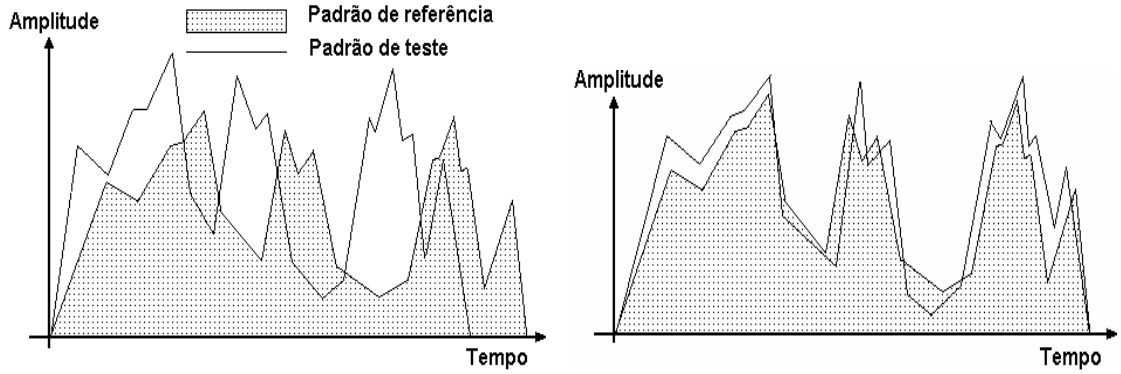



FIGURA 2.1 - Representação da diferença temporal.




(a) Antes do *DTW*. (b) Depois do *DTW*.

FIGURA 2.2 - Ajuste temporal dinâmico.

Considerando-se dois padrões de voz **R** e **T** com $N$ e $M$ quadros cada, correspondendo ao padrão de referência e ao padrão de teste, respectivamente, os padrões **R** e **T** podem ser representados, depois de uma adequada extração de parâmetros, por seqüências espectrais, da seguinte maneira:

$$\mathbf{R} = \mathbf{r}_1,...,\mathbf{r}_N, \qquad \mathbf{T} = \mathbf{t}_1,...,\mathbf{t}_M \tag{2.1}$$

onde $\mathbf{r_n}$ e $\mathbf{t_m}$, $n=1,...,N$ e $m=1,...,M$, são vetores de parâmetros de características acústicas. Os quadros de **R** e **T** são representados em um plano n-m, (ver figura 2.3) e as diferenças temporais são descritas por uma seqüência de pontos, ou **função de ajuste**, $(n(k),m(k))$ onde $k=1,...,K$, sendo $K$ o comprimento do caminho.

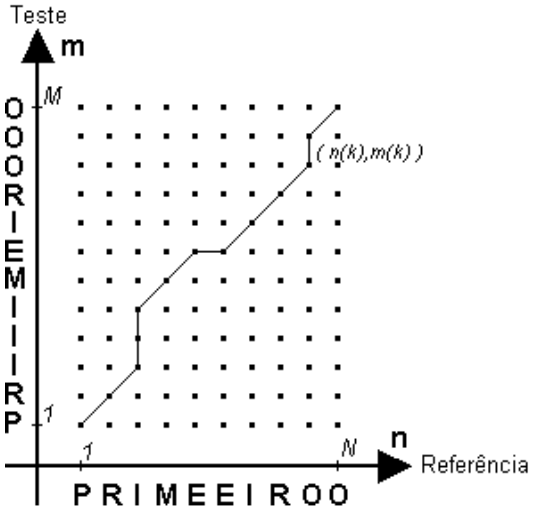



FIGURA 2.3 - Representação do *DTW*.

O problema que o *DTW* deve resolver é o de encontrar o caminho, parametrizado pelo par $(n(k),m(k))$, que minimize uma distância dada. Se $D_a(n,m)$ é a distância total mínima ao longo de qualquer caminho partindo do ponto $(1,1)$ até o ponto $(n,m)$, então:

$$D_a(n,m) = \min_{n',m'} \{ D_a(n',m') + d^*[(n',m'),(n,m)] \} \tag{2.2}$$

onde $d^*[(n',m'),(n,m)]$ indica a distância do ponto $(n',m')$ ao ponto $(n,m)$.

Visto que cada palavra deve ter modelo próprio, a preparação de modelos e o casamento de padrões se tornam pouco prático ao aumentar o tamanho do vocabulário. Bons resultados são obtidos com vocabulários pequenos [SAK 78].

## 2.2 Modelagem Conexionista - ANN

Atualmente, grandes esforços na área de reconhecimento de voz têm sido feitos com base em uma das técnicas emergentes da área de Inteligência Artificial: as Redes Neurais Artificiais (*ANN*). Existem centenas de arquiteturas de redes neurais propostas para aplicações que abrangem um grande número de áreas do conhecimento humano [FRI 96].

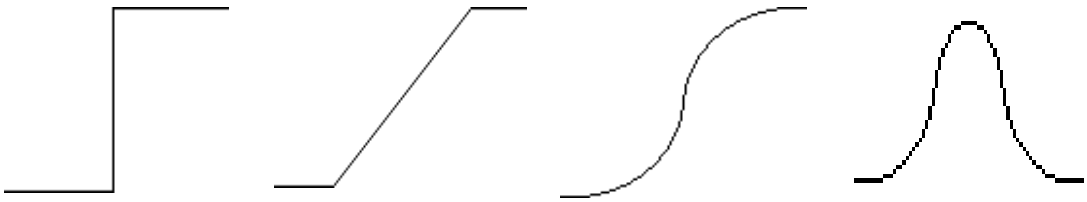
As redes neurais baseiam-se no funcionamento do próprio cérebro humano. Assim, possuem elementos de processamento que são modelos simplificados dos neurônios biológicos, e que simulam o funcionamento paralelo destes, em nosso cérebro, adquirindo características que os computadores seqüenciais carecem (paralelismo, capacidade de aprender, tolerância a falhas, etc.) [MOR 91] [BEN 96].

O *RAV* é uma tarefa onde uma estrutura exata dos padrões de voz, é difícil de ser definida em face da grande variabilidade existente, provocada pela entonação, timbre, sotaque e outros aspectos já considerados no capítulo 1. As redes neurais podem oferecer grande auxílio na busca da solução para o problema de *RAV*. Características, como a capacidade de generalização, são importantes para um problema que possui como objeto de estudo algo que varia enormemente, como o sinal de voz [DOS 95] [BEN 96].

Redes neurais são processadores paralelos que utilizam como unidades básicas neurônios artificiais. Cada neurônio calcula a soma ponderada de n sinais de entrada $x_j$, $j=1,...,n$ e aplica esta soma em uma função. O resultado desta operação será considerado como saída da unidade de processamento que, dependendo do tipo de rede, poderá ser discreta ou contínua. Matematicamente, a saída, $y$, da unidade de processamento será:

$$y = f(\sum_{j=1}^{n} w_j * x_j) \tag{2.3}$$

onde a função $f$ é conhecida como função de ativação do neurônio. Geralmente são utilizadas, como funções de ativação: funções degrau, linear, sigmóide e gaussiana. Estas funções são mostradas na figura 2.4.



a) Degrau b) Linear c) Sigmóide d) Gaussiana

FIGURA 2.4 - Exemplos de funções de ativação.

O modelo matemático do neurônio pode ser representado pela figura 2.5.

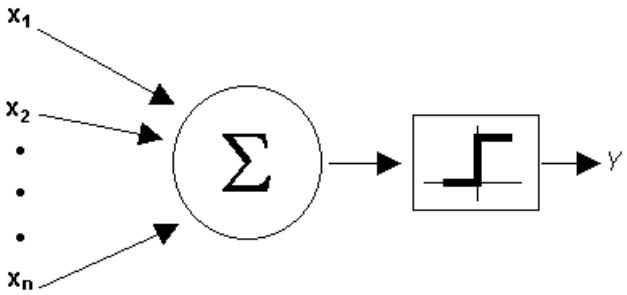



FIGURA 2.5 - Modelo matemático do neurônio.

A solução de problemas práticos requererá mais de um neurônio. Uma rede neural consiste em um conjunto de neurônios arranjados formando diversas camadas e interligados entre si. As redes neurais possuem três tipos de camadas: uma camada de

entrada, uma ou mais camadas escondidas, e uma camada de saída. A figura 2.6 ilustra essa arquitetura básica. A concatenação de um número variado das unidades de processamento em camadas, bem como as diferentes maneiras de como serão interligadas, darão origem a um número variado de arquiteturas com características e funcionamento diferentes.

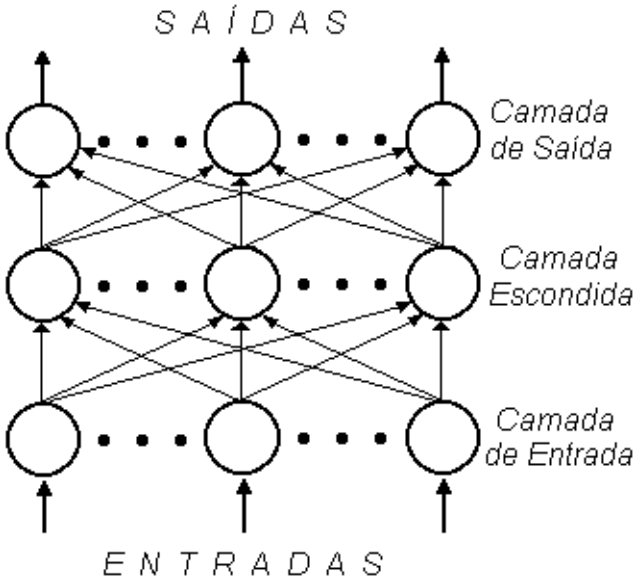



FIGURA 2.6 - Arquitetura básica de uma rede neural.

As redes neurais artificiais convencionais têm sido estruturadas para serem utilizadas em padrões estáticos [FAR 94] [BEN 96]. Um dos grandes problemas encontrados ao se utilizar redes neurais, nos sistemas de *RAV* é que a voz possui uma natureza dinâmica e a compressão ou a expansão que existe nas várias repetições das mesmas palavras produz seqüências de vetores de diferentes comprimentos. Como o número de unidades de entrada em uma rede neural é fixo, são necessárias algumas modificações às estruturas básicas de redes neurais [DOS 95] [YOU 95]. Uma forma de fazer isto é definindo um número fixo de janelas e variar o avanço e superposição entre janelas adjacentes de maneira que o número de vetores calculados sempre coincida com o número de unidades de entrada da rede neural [OGL 90] [BEN 96]. Uma outra maneira é fazer a superposição entre janelas fixas e variar o tamanho de cada janela de acordo com o tamanho da elocução sempre procurando coincidir o número de vetores calculados na elocução com o número de entradas da rede neural [MOR 91] [DOS 95] [BEN 96]. Pode-se também utilizar um algoritmo que realize o ajuste temporal dinâmico e mapeie a elocução de teste em uma elocução de tamanho padrão para utilizar com a rede neural [FAR 94] [BEN 96].

Nos sistemas de *RAV*, as redes neurais mais utilizadas tem sido: *perceptron multicamadas* ou *MLP* (**M**ulti **L**ayer **P**erceptron), *Hopfield*, *ART*, *SOM*, e em especial a rede *TDNN* (**T**ime **D**elay **N**eural **N**etwork) a qual incorpora a dinâmica do padrão de voz [FAR 94] [DOS 95] [YOU 95] [CAS 97].

Além dessas redes neurais, existem arquiteturas híbridas que se relacionam com outras técnicas tais como *DTW* e *HMM* [MOR 91] [BUR 96] [NAR 96] [MOR 97] [SMY 97].

## 2.3 Modelos Ocultos de Markov

Um processo de *Markov* de *j-ésima* ordem é um processo no qual para qualquer seqüência de eventos no domínio do tempo a densidade de probabilidade condicional do evento atual, dados todos os eventos passados e presentes, depende unicamente dos *j* eventos mais recentes [MOR 95].

No caso especial de uma cadeia de *Markov* de primeira ordem, esta descrição probabilística é truncada ao estado atual e ao estado predecessor [GON 94], ou melhor:

$$P[q_t=S_j|q_{t-1}=S_i,\ q_{t-2}=S_k,...]=P[q_t=S_j|q_{t-1}=S_i] \tag{2.4}$$

Os instantes de tempo associados com as mudanças de estado são denotados como $t=1, 2, ...,T$, e o estado atual no tempo $t$ será $q_t$. O processo assim descrito é denominado de modelo observável de *Markov*, já que a saída do processo é o conjunto de estados em cada instante de tempo, onde cada estado corresponde a um evento físico observável [RAB 93]. Porém, este modelo é restrito demais para ser aplicável à maioria de problemas de interesse. É necessário estender o conceito de modelos de *Markov* para incluir o caso quando a observação é uma função probabilística do estado e não um evento determinístico. O modelo resultante, que recebe o nome de modelo oculto de *Markov* ou *HMM (**H**idden **M**arkov **M**odel)*, é um processo estocástico oculto que pode ser observado unicamente através de outro processo estocástico que produz a seqüência de observações [DEL 93] [RAB 93].

Um *HMM* é uma máquina de estados finita estocástica constituída por um conjunto de estados ligados entre si através de transições. Ao ocorrer uma transição, gera-se um símbolo. Os *HMMs* são usualmente especificados por um conjunto de estados, um conjunto de probabilidades de transição para cada transição permitida entre estados, e uma função de densidade de probabilidade associada com a emissão de símbolos [MOR 95] [MAR 96]. A figura 2.7 mostra, como exemplo, um *HMM* de três estados $S_1$, $S_2$ e $S_3$. Existe uma probabilidade associada a cada transição entre estados e uma função de probabilidade com as observações $v_k$.

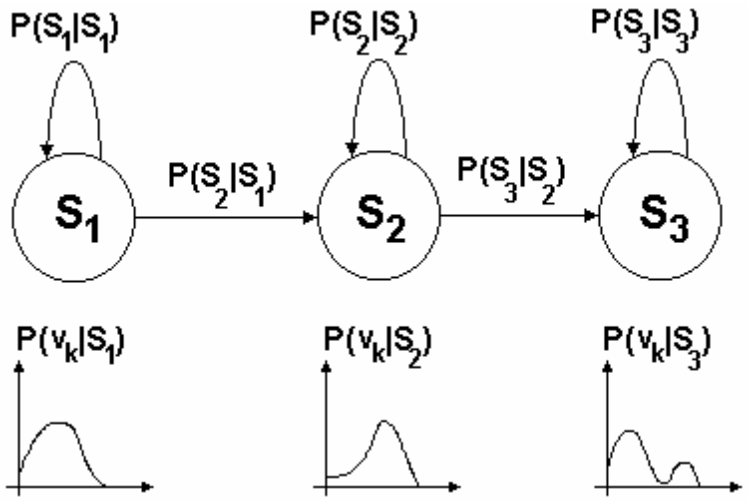



FIGURA 2.7 - *HMM* típico de três estados.

### 2.3.1 Exemplo de *HMM*

Consideraremos, a título de ilustração, o experimento de urnas e bolas ilustrado na figura 2.8. Assumindo que existem *N* urnas em uma sala contendo bolas coloridas de *M* cores diferentes. Uma pessoa escolhe de maneira aleatória uma urna inicial. Dessa urna, a pessoa escolhe uma bola também aleatoriamente. Em seguida a bola é recolocada na respectiva urna. Outra urna é então selecionada de acordo com um processo aleatório associado com a nova urna, e mais uma bola é retirada. Outra pessoa, impedida de ver o processo de retirada, ou seja, impedida de ver qual a seqüência de urnas utilizadas para retirar as bolas, anotará a cor das bolas retiradas que a primeira pessoa irá ditando. Este processo irá gerar uma seqüência finita de observações de cores, a qual deseja-se modelar como a saída de um *HMM*.

Uma forma intuitiva e simples de se modelar este processo é associar as urnas aos estados de um *HMM* e as cores das bolas aos símbolos possíveis de serem emitidos pelo *HMM* [RAB 93] [HUA 90]. Para cada estado (urna), existe uma distribuição de probabilidade de emissão de símbolos (cores). Além disso, a escolha da próxima urna está relacionada às transições entre estados, com cada transição possuindo uma dada probabilidade de ocorrência [DUG 96].

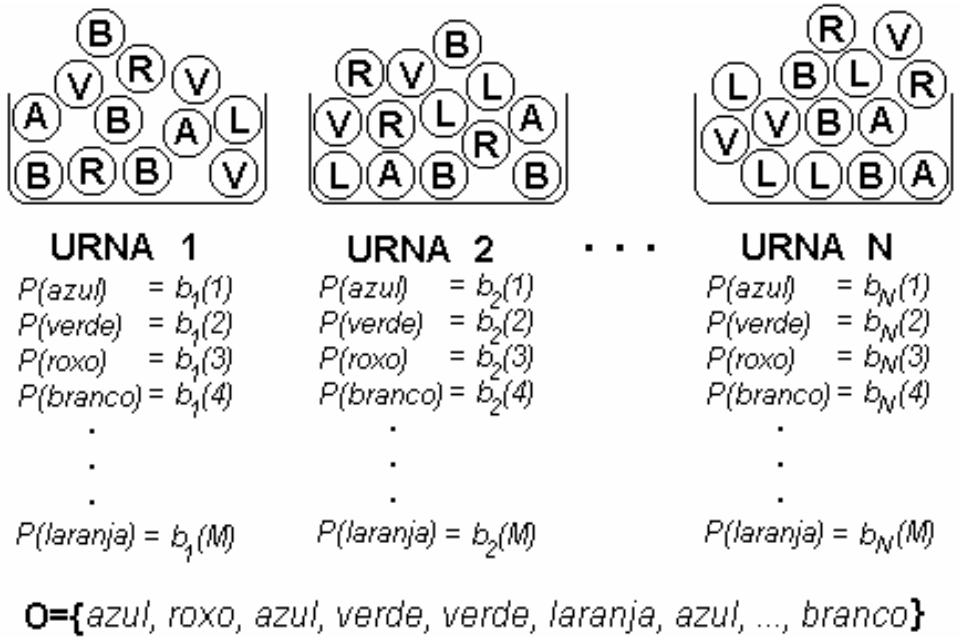



FIGURA 2.8 - Modelo de *N* estados de urnas e bolas.

O único parâmetro observável neste processo é a seqüência de símbolos emitidos, ou seja, a seqüência de cores. Não é possível ter acesso ao mecanismo gerador do processo e na melhor das hipóteses podem-se estimar seus parâmetros [DUG 96]. Isto quer dizer que nunca se terá certeza se o número de estados utilizado no *HMM* coincide realmente com o número de estados do processo. O que se consegue é um modelo que produz, com máxima probabilidade, a seqüência de observações para o qual foi treinado [MOR 95] [FOS 98].

### 2.3.2 Definição de *HMMs*

Um modelo de *Markov* é definido formalmente pelos seguintes parâmetros:

1. Um conjunto de estados $\boldsymbol{S}=\{S_i\}$, de tamanho $\boldsymbol{N}$. O estado no tempo $t$ é designado como $q_t$. Apesar dos estados da cadeia serem ocultos, para várias aplicações práticas existe algum significado físico para eles. No caso do modelo de urnas e bolas descrito acima, por exemplo, cada estado está associado a uma urna diferente.

2. Um conjunto de símbolos observáveis $\boldsymbol{V}=\{v_k\}$, de tamanho $M$, que corresponde à saída física do sistema modelado. Para o exemplo das urnas e bolas, acima citado, os símbolos correspondem às cores das bolas selecionadas das urnas; para o *RAV*, os símbolos devem corresponder a um conjunto de vetores de parâmetros extraídos do sinal de voz. Quando o conjunto de símbolos é finito, dizemos que o *HMM* correspondente é discreto, caso contrário temos um *HMM* contínuo no qual $M$ representa o número de misturas da função de distribuição de probabilidade associada com cada estado.

3. Uma matriz de probabilidades de transição entre estados $\mathbf{A}=\{a_{ij}\}$, onde $a_{ij}$ é a probabilidade de transição do estado $S_i$ para o estado $S_j$. A matriz **A** é de dimensão *NxN*:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix} \tag{2.5}$$

Formalmente pode-se definir:

$$a_{ij}=P[q_{t+1}=S_j|q_t=S_i], \qquad 1 \le i, j \le N \tag{2.6}$$

A forma como os estados estão ligados, ou seja, quais as transições possíveis de ocorrerem, define a estrutura ou topologia do *HMM*. Para as transições possíveis de ocorrerem, teremos $a_{ij} > 0$ e para as não permitidas, $a_{ij} = 0$.

4. Um conjunto de distribuições de probabilidade de observação de símbolos. Se o *HMM* é discreto (***DHMM***), tal conjunto é dado pela matriz $\mathbf{B}=\{b_j(k)\}$, de dimensões *NxM*:

$$\mathbf{B} = \begin{bmatrix} b_1(o_1) & b_1(o_2) & \dots & b_1(o_M) \\ b_2(o_1) & b_2(o_2) & \dots & b_2(o_M) \\ \dots & \dots & \dots & \dots \\ b_N(o_1) & b_N(o_2) & \dots & b_N(o_M) \end{bmatrix} \tag{2.7}$$

onde $b_j(k)$ é a probabilidade de emissão do símbolo $\boldsymbol{k}$ no estado $\boldsymbol{j}$, e pode ser definida formalmente como:

$$b_j(k)=P[o_t=\boldsymbol{v_k}|q_t=S_j], \qquad 1 \le j \le N;\ 1 \le k \le M \tag{2.8}$$

Se o *HMM* for contínuo (CHMM) será necessária a estimação das funções de distribuição de probabilidade que geram as observações dado o estado j, ou

seja $b_j(\boldsymbol{O})$ [HUA 90]. Normalmente utiliza-se uma mistura de Gaussianas [HUA 90].

5. Um vetor de probabilidades iniciais, $\boldsymbol{\pi}=\{\pi_i\}$, onde $\pi_i$ é a probabilidade do *HMM* iniciar no estado $S_i$. O vetor $\boldsymbol{\pi}$ pode ser representado mediante um vetor de dimensão *Nx1*:

$$\boldsymbol{\pi}=[\pi_1\ \pi_2\ \ldots\ \pi_N] \tag{2.9}$$

Cada elemento $\pi_i$ pode ser definido da seguinte maneira:

$$\pi_i=P[q_1=S_i], \qquad 1 \le i \le N \tag{2.10}$$

No caso de *HMMs* discretos [LEE 88] [YU 95], o inteiro espaço acústico é dividido ou "*clusterizado*" em um número razoável de regiões (256, por exemplo), através de um processo de quantização vetorial (*VQ*). O centróide de cada cluster é representado por um código escalar ou índice. O conjunto de códigos ou índices obtidos da quantização é conhecido, também, pelo nome de *dicionário de quantização*. O dicionário de quantização formará os símbolos ou observações com que os *HMMs* trabalharão. Portanto, a distribuição de probabilidades sobre o espaço acústico é representada por um simples histograma referente ao conjunto de índices. Esta abordagem é suscetível de erros de quantização se o dicionário de quantização for pequeno demais. Por outro lado, o incremento do tamanho do dicionário de quantização produzirá menos dados de treinamento para cada código do dicionário o qual também introduzirá uma degradação no desempenho [TEB 95].

Os erros da quantização podem ser eliminados utilizando um modelo de densidade contínua, em lugar de um dicionário de quantização. Neste caso, a distribuição de probabilidade sobre o espaço acústico é modelada diretamente, assumindo que ela possui uma certa forma paramétrica. Tipicamente essa forma paramétrica é constituída por uma mistura de *M* Gaussianas [TEB 95], tal como:

$$b_j(\mathbf{O}) = \sum_{k=1}^{M} c_{jk} \cdot G(\mathbf{O}, \mu_{jk}, \mathbf{U_{jk}}) \tag{2.11}$$

onde $\boldsymbol{O}$ é o vetor de observações sendo modelado, $c_{jk}$ é o fator de peso para cada Gaussiana *G* com média $\mu_{jk}$ e matriz de covariança $\mathbf{U_{jk}}$, satisfazendo o seguinte requisito estocástico:

$$\sum_{k=1}^{M} c_{jk} = 1, \qquad c_{jk} \ge 0, \quad 1 \le j \le N, \quad 1 \le k \le M \tag{2.12}$$

desta maneira função de distribuição de probabilidade está apropriadamente normalizada, ou seja:

$$\int_{-\infty}^{+\infty} b_j(\mathbf{O}).d\mathbf{O} = 1 \qquad 1 \le j \le N \tag{2.13}$$

Durante o treinamento, a reestimação de $b_j$, envolve a reestimação de $c_{jk}$, $\mu_{jk}$, e $\mathbf{U_{jk}}$, utilizando um conjunto adicional de fórmulas [DEL 93].

No presente trabalho serão utilizados *HMMs* discretos, em função da facilidade de implementação em relação aos *HMMs* contínuos.

A notação simplificada para representar um *HMM* é $\lambda(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$. A seqüência de observações é representada pelo vetor $\boldsymbol{O}=[o_1...o_T]$, onde $T$ é o número de observações da seqüência.

A estrutura da matriz $\boldsymbol{A}$, define a topologia do *HMM*. Uma topologia muito utilizada é a *left-right,* onde o incremento de tempo faz com que o índice que indica o estado atual aumente ou permaneça igual, mas nunca diminua. A figura 2.9 mostra um *HMM left-right* de 4 estados. O *HMM left-right* é apropriado para modelar sinais de voz, com propriedades variáveis no tempo de uma maneira sucessiva [RAB 93].

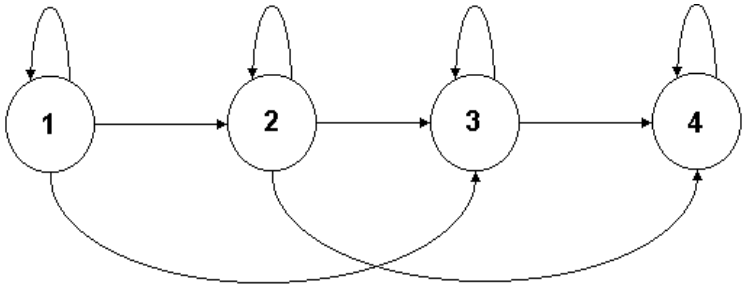



FIGURA 2.9 - Modelo *left-right*.

Na continuação considerar-se-á o caso discreto, utilizando quantização vetorial, antes do treinamento dos *HMMs*. Para tanto, deve ser utilizado um conjunto de observações extraídas de múltiplas seqüências correspondentes à mesma classe para gerar os vetores códigos. Posteriormente, qualquer seqüência de observações a ser utilizada com o *HMM*, seja para treinamento ou teste, deverá antes ser quantizada vetorialmente utilizando o dicionário obtido. Se existirem $M$ vetores código no dicionário, então é suficiente associar com cada observação um número inteiro $k$, $1 \leq k \leq M$. Formalmente, o processo aleatório vetorial $v(t)$ é substituído por um processo aleatório escalar $y(t)$ que pode assumir somente valores inteiros no intervalo $[1, M]$.

Para sistemas de *RAV*, geralmente são considerados apenas os modelos de *Markov* de primeira ordem [LEV 83] [HUA 90] [FOS 98] para os quais são feitas as duas hipóteses seguintes:

*(1) hipótese de Markov*, a qual determina que a probabilidade de uma cadeia estar em um dado estado no instante $t$ depende apenas de seu estado no instante $t-1$.

*(2) hipótese de independência de emissão de símbolos*, que determina que a probabilidade de emissão de um símbolo de saída no instante $t$ depende apenas do estado nesse instante de tempo sem importar quando e como chegou-se naquele estado.

Estas hipóteses adicionais, que caracterizam um modelo de primeira ordem, permitem uma grande redução no número de parâmetros do modelo, além de possibilitar algoritmos bastante eficientes para o treinamento e avaliação destes parâmetros, como será visto mais adiante.

### 2.3.3 Os Três Problemas Básicos para HMMs

Existem três questões básicas encontradas no desenvolvimento de sistemas modelados por *HMMs*, que devem ser resolvidas para os modelos serem úteis em aplicações do mundo real. Estas questões serão descritas a seguir.

### 2.3.4 Avaliação

O problema de avaliação se resume em dados um modelo $\lambda(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$, e uma seqüência de observações $\boldsymbol{O}=(o_1\ o_2\ ...\ o_T)$, calcular a probabilidade de $\lambda$ gerar a seqüência $\boldsymbol{O}$, ou seja, $P(\boldsymbol{O}|\lambda)$.

A seqüência de observações $\boldsymbol{O}$ pode ser gerada através de diferentes seqüências $\boldsymbol{q}$ de estados, cada uma delas com uma probabilidade determinada. Portanto, para calcular a probabilidade desejada, é preciso somar as probabilidades de observação da seqüência $\boldsymbol{O}$ para todas as seqüências de estados, de tamanho $T$, permitidas pela topologia do *HMM*. Existem $N^T$ seqüências possíveis de estados [RAB 93].Considerando $\boldsymbol{q}=(q_1\ q_2...\ q_T)$, uma de tais seqüências de estados, a probabilidade de acontecer esta seqüência particular é dada por:

$$P(\mathbf{q} \mid \lambda) = \pi_{q_1}.a_{q_1q_2}.a_{q_2q_3}...a_{q_{T-1}q_T} \tag{2.14}$$

Além disso, a probabilidade da seqüência $\boldsymbol{O}$ ser gerada, dada a seqüência de estados $\boldsymbol{q}$ e o modelo $\lambda$, é calculada por [LEV 83]:

$$P(\mathbf{O} \mid \mathbf{q}, \lambda) = \prod_{t=1}^{T} P(o_t \mid q_t, \lambda) \tag{2.15}$$

onde é considerada a independência estatística das observações. A probabilidade de que $\boldsymbol{O}$ e $\boldsymbol{q}$ ocorram simultaneamente, é o seguinte produto:

$$P(\mathbf{O}, \mathbf{q} \mid \lambda) = P(\mathbf{O} \mid \mathbf{q}, \lambda).P(\mathbf{q} \mid \lambda) \tag{2.16}$$

A probabilidade de $\boldsymbol{O}$, dado $\lambda$, é calculada somando a probabilidade conjunta de $\boldsymbol{O}$ e $\boldsymbol{q}$ sob todas as possíveis seqüências de estados $\boldsymbol{q}$, obtendo-se:

$$P(\mathbf{O} \mid \lambda) = \sum_{todo\ \mathbf{q}} P(\mathbf{O} \mid \mathbf{q}, \lambda).P(\mathbf{q} \mid \lambda) = \sum_{q_1 \ldots q_T} \pi_{q_1}.b_{q_1}(o_1).a_{q_1q_2}.b_{q_2}(o_2)...a_{q_{T-1}q_T}.b_{q_T}(o_T) \tag{2.17}$$

A equação acima indica que no instante inicial ($t=1$), $\lambda$ está no estado $q_1$ com probabilidade $\pi_{q_1}$ e $\lambda$ gera o símbolo $o_1$ com probabilidade $b_{q_1}(o_1)$. Ao avançar para o instante de tempo $t+1$ ($t=2$), há uma transição do estado $q_1$ para o estado $q_2$ com probabilidade $a_{q_1q_2}$ e $\lambda$ gera o símbolo $o_2$, com probabilidade $b_{q_2}(o_2)$. Este processo continua até, no instante $T$, fazer a transição do estado $q_{T-1}$ para o estado $q_T$ com probabilidade $a_{q_{T-1}q_T}$ e gerar o símbolo $o_T$ com probabilidade $b_{q_T}(o_T)$. Isto é feito para todas as seqüências possíveis somando-se os resultados.

A avaliação direta da equação (2.17) apresenta um crescimento exponencial com o tamanho da seqüência de observações. Porém, no caso de *HMMs* de primeira ordem, pode-se utilizar, para a avaliação de $P(\boldsymbol{O}|\lambda)$, o algoritmo *forward* [LEE 89].

O algoritmo *forward* fornece como resultado final o valor da verossimilhança, ou seja, o valor de $P(\mathbf{O}|\lambda)$. Considerando a variável $\alpha_t(i)$ definida como:

$$\alpha_t(i)=P(o_1\, o_2 \ldots o_t,\, q_t=S_i|\lambda) \tag{2.18}$$

ou seja, $\alpha_t(i)$ é igual à probabilidade do *HMM* ter gerado até o instante $t$, a seqüência de observações $(o_1\, o_2 \ldots o_t)$, e estar no estado $S_i$ nesse mesmo instante. O algoritmo *forward* calcula $\alpha_t(i)$, recursivamente em $t$, da maneira indicada na continuação:

***Algoritmo Forward***

*1. INICIALIZAÇÃO:*

$$\alpha_1(i)=P(o_1,q_1=S_i|\lambda)=\pi_i.b_i(o_1), \qquad 1\leq i\leq N$$

*2. RECURSÃO:*

*para* $1\leq t\leq T-1,\ 1\leq j\leq N,$

$$\alpha_{t+1}(j)=\left[\sum_{i=1}^{N}\alpha_t(i).a_{ij}\right].b_j(o_{t+1})$$

*3. FINALIZAÇÃO:*

$$P(\mathbf{O}\mid\lambda)=\sum_{i=1}^{N}\alpha_T(i)$$

O primeiro passo inicializa a probabilidade *forward* como a probabilidade de estar no estado $S_i$ e ter a observação inicial $o_1$, dado o modelo $\lambda$. A etapa de recursão, a qual é o centro do algoritmo *forward*, é ilustrada na figura 2.10. Essa figura mostra como o estado $j$ pode ser alcançado no tempo $t+1$ a partir dos $N$ possíveis estados, $S_i$, $1\leq i\leq N$, do tempo $t$. O produto $\alpha_t(i).a_{ij}$ é a probabilidade de que, conjuntamente, $o_1\, o_2 \ldots o_t$ sejam observados, e o estado $j$ seja alcançado no tempo $t+1$ através do estado $S_i$ no tempo $t$.

A cada instante de tempo $t$, os valores de $\alpha_{t+1}(i)$ são calculados multiplicando-se os valores de $\alpha_t(i)$ pela probabilidade de transição do estado $i$ para o estado $j$, $a_{ij}$, e pela probabilidade de emissão do símbolo $o_{t+1}$, no estado $j$, e somando-se estes valores para todo valor de $i$. O passo final do algoritmo calcula $P(\mathbf{O}|\lambda)$ como a soma das variáveis terminais $\alpha_T(i)=P(o_1 \ldots o_T,\, q_T=S_i|\lambda)$.

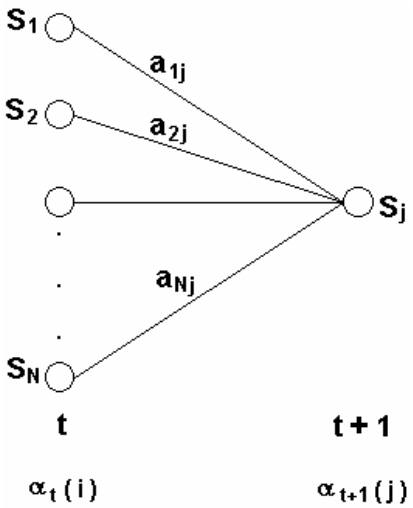



FIGURA 2.10 - Algoritmo *forward*.

Uma das vantagens deste algoritmo, além de sua eficiência computacional é sua capacidade de ser implementado em tempo real, pois a cada chegada de um novo símbolo os valores de $\alpha_t(i)$ podem ser atualizados [BUR 96] [NAR 96].

A figura 2.11 mostra um exemplo do algoritmo forward em operação, para um *HMM* de dois estados, no qual existem apenas dois símbolos: *A* e *B*. A probabilidade de emissão de cada símbolo é indicada para cada estado. O cálculo da probabilidade de que a seqüência de saída $\boldsymbol{O}=[A, A, B]$ possa ter sido gerada pelo *HMM* é mostrado ao lado. Cada célula em *(t, j)* mostra o valor de $\alpha_t(j)$, utilizando os valores fornecidos de $a_{ij}$ e $b_j(o_t)$. O cálculo procede do primeiro estado ao último estado dentro de um tempo de quadro, antes de proceder ao seguinte quadro. Na célula final, observa-se que a probabilidade de que este *HMM* gere a seqüência *[A, A, B]* é 0,096.

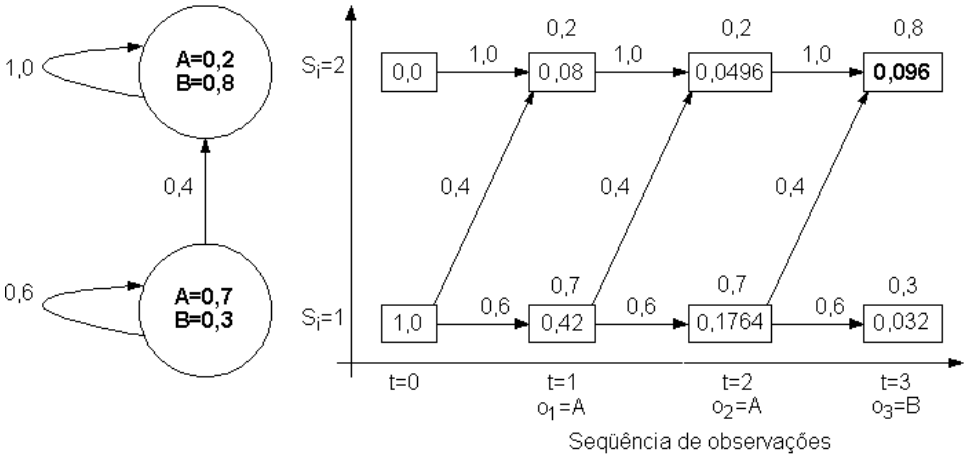



FIGURA 2.11 - Algoritmo *forward* em operação, mostrando o valor de $\alpha_t(j)$ em cada célula.

De maneira similar à variável *forward*, pode-se definir a variável *backward* $\beta_t(i)$, como:

$$\beta_t(i)=P(o_{t+1}\, o_{t+2} \ldots o_T|q_t=Si,\lambda) \qquad (2.19)$$

ou seja, $\beta_t(i)$ é igual à probabilidade do modelo de *Markov* gerar a seqüência de observações $o_{t+1}\ o_{t+2}\ ...\ o_T$, sendo $S_i$ o estado no instante $t$. Assim, $\beta_t(i)$ pode ser calculado recursivamente em $t$, como é mostrado abaixo.

***Algoritmo Backward***

*1. INICIALIZAÇÃO:*

$\beta_T(i)=1, \qquad 1 \le i \le N$

*2.RECURSÃO: para* $T-1 \ge t \ge 1,\ 1 \le i \le N,$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}.b_j(o_{t+1}).\beta_{t+1}(j)$$

O passo de inicialização define arbitrariamente $\beta_T(i)=1$, para todos os valores de $i$. A cada instante de tempo $t$, os valores de $\beta_t(i)$ são calculados multiplicando-se os valores de $\beta_{t+1}(j)$ pela probabilidade de transição do estado $i$ para o estado $j$, $a_{ij}$, e pela probabilidade de emissão do símbolo $o_{t+1}$, no estado $j$, e somando-se estes valores para todos os valores de $j$. Este procedimento é ilustrado na figura 2.12.

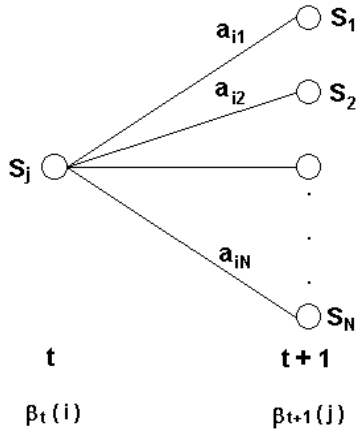



FIGURA 2.12 - Algoritmo *backward*.

## 2.3.5 Decodificação

O problema de decodificação visa à determinação da seqüência de estados $\boldsymbol{q_i}=(q_1\ q_2\ ...\ q_T)$, do modelo $\lambda$, que mais provavelmente produziu a seqüência de observações $\boldsymbol{O}=(o_1\ o_2\ ...\ o_T)$. Em contraste com o primeiro problema, no qual podia ser dada uma solução exata, existem diversas maneiras de se resolver o segundo problema [DEL 93].

Por definição, em um *HMM* a seqüência de estados é escondida. Existe apenas a possibilidade de reproduzir a seqüência de estados que tem a mais alta probabilidade de ter gerado a seqüência de observações dada [LEV 83].

Para determinar a seqüência de estados ótima, é necessário calcular a máxima probabilidade, ao longo de um caminho, que termina no estado $i$, estando no tempo $t$ e

produzindo as primeiras *t* observações. Esta probabilidade é definida utilizando a variável $\delta_t(i)$, onde:

$$\delta_t(i) = \max_{q_1 q_2 \cdots q_{t-1}} [P(q_1 \ q_2 \ \ldots \ q_{t-1}, q_t = S_i, o_1 \ o_2 \ \ldots \ o_t \mid \lambda)] \qquad (2.20)$$

Por indução tem-se:

$$\delta_{t+1}(j) = \{\max_i [\delta_t(i).a_{ij}]\}.b_j(o_{t+1}) \qquad (2.21)$$

Para recuperar a seqüência de estados, é necessário recuperar o argumento que maximiza a equação anterior, para cada *t* e para cada *j* [LEV 83]. Isto é feito através da matriz $\psi_t(j)$. O procedimento completo é mostrado na continuação e é conhecido pelo nome de ***algoritmo de Viterbi*** [RYA 93].

***Algoritmo de Viterbi***

1. *INICIALIZAÇÃO:*

   *Para* $1 \le i \le N$

   $\delta_1(i) = \pi_i . b_i(o_1), \ \psi_1(i) = 0$

2. *RECURSÃO:*

   *Para* $2 \le t \le T, \ 1 \le j \le N,$

   $$\delta_t(j) = \max_{1 \le i \le N} \left[\delta_{t-1}(i) \cdot a_{ij}\right] \cdot b_j(O_t)$$

   $$\psi_t(j) = \arg\max_{1 \le i \le N} \left[\delta_{t-1}(i) \cdot a_{ij}\right]$$

3. *FINALIZAÇÃO:*

   $$P^*(\mathbf{O} \mid \lambda) = \max_{1 \le i \le N} \left[\delta_T(i)\right]$$

   $$q_T = \arg\max_{1 \le i \le N} \left[\delta_T(i)\right]$$

4. *SEQUÊNCIA DE ESTADOS (backtracking):*

   *Para* $T-1 \le t \le 1$

   $q^*_t = \psi_{t+1}.(q^*_{t+1})$

O algoritmo de *Viterbi* é similar (exceto pelo passo de *backtracking*) em implementação ao algoritmo *forward*. A probabilidade forward é a soma de $P(\mathbf{O},\mathbf{q}|\lambda)$ sobre todas as possíveis seqüências de estados. No entanto, o *algoritmo de Viterbi* calcula eficientemente o valor máximo de $P(\mathbf{O},\mathbf{q}|\lambda)$ para todos os estados $\mathbf{q}$. Devido ao seu menor custo computacional, o algoritmo de *Viterbi* é utilizado, preferencialmente ao algoritmo *forward*, para classificar uma seqüência de observações como gerada por um determinado *HMM*.

Embora a classificação utilizando o algoritmo de *Viterbi* não seja ótima, resultando apenas em uma aproximação para a probabilidade de uma dada seqüência de

observações, os resultados utilizando-se o algoritmo de *Viterbi* e o algoritmo *forward* são praticamente idênticos [LEV 83] [DEL 93].

A probabilidade de uma palavra (ou sentença) tenderá a zero durante a utilização do algoritmo de *Viterbi*. Para um número grande de quadros de análise, esta probabilidade irá ultrapassar a precisão dos computadores digitais, resultando em *underflow* [LEV 83]. Conseqüentemente, é necessária a realização de algum tipo de escalonamento.

Um caminho para se evitar *underflow* é representar probabilidades pelos seus logaritmos [LEE 89]. Utilizando o logaritmo dos parâmetros de modelo no algoritmo de *Viterbi*, para o cálculo da verossimilhança, não há necessidade de normalização nem de multiplicações [RYA 93] [LAZ 97].

Definindo-se:

$$\tilde{\delta}_t(i) = \max_{q_1,q_2,\ldots,q_T} \{\log[P(q_1 q_2 \ldots q_t, o_1 o_2 \ldots o_t \mid \lambda)]\} \tag{2.22}$$

A versão logarítmica do algoritmo de *Viterbi* é mostrada abaixo.

***Implementação Logarítmica do Algoritmo de Viterbi***

*0. PRE-DEFINIÇÂO:*

*Para 1≤t≤T,*

*para 1≤i≤N, 1≤j≤N*

$$\tilde{\pi}_i = \log(\pi_i), \quad \tilde{b}_i(o_t) = \log(b_i(o_t)), \quad \tilde{a}_{ij} = \log(a_{ij})$$

*1. INICIALIZAÇÃO:*

*Para 1≤i≤N*

$$\tilde{\delta}_1(i) = \log(\delta_1(i)) = \tilde{\pi}_i + \tilde{b}_i(o_1), \quad \psi_1(i) = 0$$

*2. RECURSÃO:*

*Para 2≤t≤T, 1 ≤j≤ N,*

$$\tilde{\delta}_t(j) = \log(\delta_t(j)) = \max_{1\le i\le N}\left[\tilde{\delta}_{t-1}(i) + \tilde{a}_{ij}\right] + \tilde{b}_j(O_t)$$

$$\psi_t(j) = \arg\max_{1\le i\le N}\left[\tilde{\delta}_{t-1}(i) + \tilde{a}_{ij}\right]$$

*3. FINALIZAÇÃO:*

$$\tilde{P}^*(\mathbf{O} \mid \lambda) = \max_{1\le i\le N}\left[\tilde{\delta}_T(i)\right]$$

$$q_T = \arg\max_{1\le i\le N}\left[\tilde{\delta}_T(i)\right]$$

*3. SEQUÊNCIA DE ESTADOS (backtracking):*

*Para T-1≤t≤1*

$$q^*_t = \psi_{t+1}(q^*_{t+1})$$

O caminho ótimo produzido pelo algoritmo de *Viterbi* pode ser representado mediante um diagrama de treliça [FOR 73] [RYA 93]. A figura 2.13 mostra um diagrama de treliça típico.

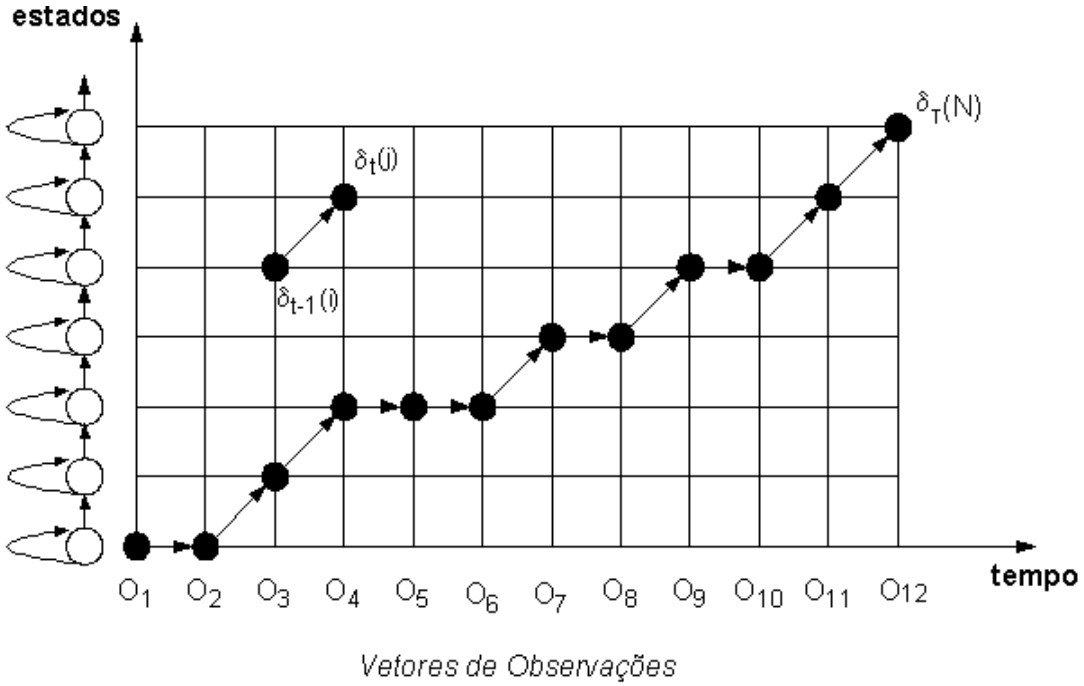



FIGURA 2.13 - Treliça de estados e o algoritmo de *Viterbi*.

## 2.3.6 Treinamento

O problema do treinamento consiste em ajustar os parâmetros do modelo $\lambda(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$ com o objetivo de maximizar a probabilidade da seqüência de observações, $P(\boldsymbol{O}/\lambda)$.

Dada qualquer seqüência de observações finita não existe um caminho ótimo para estimar os parâmetros do *HMM*. Pode-se, porém, escolher as matrizes $\boldsymbol{A}$ e $\boldsymbol{B}$ de forma que $P(\boldsymbol{O}/\lambda)$ seja localmente maximizado utilizando o algoritmo iterativo *forward-backward* [LEE 89], também conhecido como algoritmo de reestimação de *Baum-Welch*, visto a seguir.

Inicialmente, define-se a variável $\gamma_t(i, j)$ como:

$$\gamma_t(i, j)=P(q_t=S_i, q_{t+1}=S_j|\boldsymbol{O},\lambda) \tag{2.23}$$

ou, em outras palavras, $\gamma_t(i, j)$ é a probabilidade de um caminho estar no estado $S_i$ no tempo $t$ e realizar uma transição para o estado $S_j$ no tempo $t+1$, dada a seqüência de observações $\boldsymbol{O}$ e o modelo $\lambda$. Este evento composto acontece com probabilidade $\alpha_t(i)$, que calcula a probabilidade do caminho se encontrar no estado $i$ no tempo $t$, multiplicado pelo valor de $a_{ij}.b_j(O_{t+1})$, que calcula a transição local do estado $i$ para o estado $j$, multiplicado por $\beta_{t+1}(j)$, que calcula a probabilidade do caminho se encontrar no estado $j$ no tempo $t+1$. Isto é ilustrado na figura 2.14. A partir das definições das probabilidades *forward* e *backward*, $\gamma_t(i, j)$ pode ser reescrita da seguinte maneira:

$$\gamma_t(i,j) = \frac{P(q_t = i, q_{t+1} = j, \mathbf{O} \mid \lambda)}{P(\mathbf{O} \mid \lambda)} = \frac{\alpha_t(i).a_{ij}.b_j(o_{t+1}).\beta_{t+1}(j)}{P(\mathbf{O} \mid \lambda)} \tag{2.24}$$

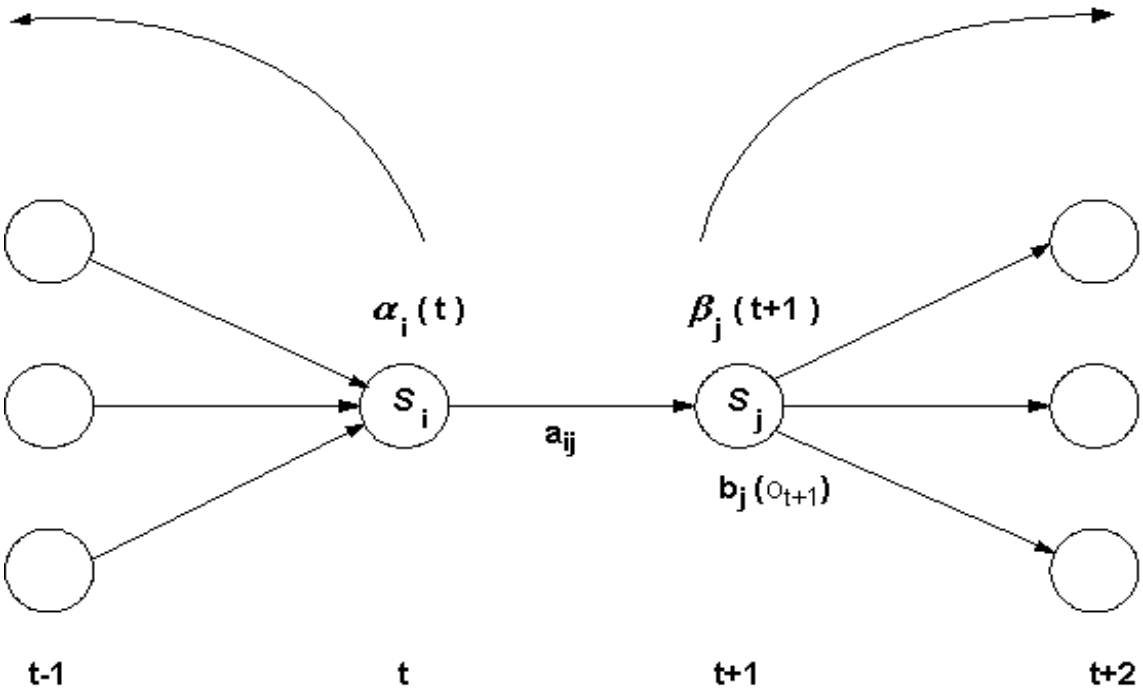



FIGURA 2.14 - Cálculo dos valores de $\gamma_t(i,j)$ utilizando o algoritmo *Baum-Welch*.

De maneira similar, a probabilidade posterior de estar no estado *i* no tempo *t*, $\gamma_t(i)$, dada a seqüência $\boldsymbol{O}$ e o modelo $\lambda$, é:

$$\gamma_t(i) = P(q_t = i \mid \mathbf{O}, \lambda) = \frac{\alpha_t(i).\beta_t(i)}{P(\mathbf{O} \mid \lambda)} \tag{2.25}$$

Das equações (2.24) e (2.25), observa-se que $\gamma_t(i)$ pode ser calculada como:

$$\gamma_t(i) = \sum_{j=1}^{N} \gamma_t(i,j), \qquad t < T \tag{2.26}$$

Utilizando novamente o experimento das urnas e bolas, mostrado na figura 2.15 (onde há uma correspondência entre os estados e as urnas, e entre as probabilidades de emissão e a distribuição das cores das bolas), então o número esperado de vezes que se obtém uma bola de cor azul da urna 1 (estado 1), dados a seqüência de observações $\boldsymbol{O}$ e o modelo $\lambda$, pode ser expresso como:

$$\sum_{t \in O_t = azul} \gamma_t(1) \tag{2.27}$$

Similarmente, o número esperado de vezes que se retira uma bola da urna 1, dada a seqüência de observações $\boldsymbol{O}$ e o modelo $\lambda$, pode ser expresso como:

$$\sum_{t=1}^{T} \gamma_t(1) \tag{2.28}$$

Intuitivamente, a partir das expressões (2.27) e (2.28), a probabilidade de emissão $b_1(azul)$ poderá ser calculada como:

$$b_1(azul) = \frac{\sum_{t \in O_t = azul} \gamma_t(1)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{2.29}$$

Desta maneira, criar-se-á um novo modelo $\overline{\lambda}$ que irá melhorando iterativamente a estimativa.

A soma de $\gamma_t(i)$ para todo o índice $t$, excluindo o último instante $T$, será o número esperado de transições para fora do estado $i$ [HUA 90]. Em geral, uma estimativa $\overline{a}_{ij}$ da probabilidade de transição do estado $i$ ao estado $j$, pode calcular-se como:

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \gamma_t(i,j)} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.30}$$

Similarmente, a probabilidade de observar o símbolo $k$ no estado $j$, $b_j(k)$, pode ser calculada como a freqüência de ocorrência do símbolo $k$ em relação à freqüência de ocorrência de qualquer símbolo no estado $j$. A soma de $\gamma_t(i)$ para todo o índice de tempo $t$ é o número esperado de vezes que o estado $i$ é visitado. Portanto, $b_j(k)$ poderá ser reestimado como:

$$\overline{b}_j(k) = \frac{\sum_{t=1 \; t \in O_t = k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.31}$$

Finalmente, uma nova estimativa das probabilidades de estado inicial pode ser calculada como:

$$\overline{\pi}_i = \gamma_1(i) \tag{2.32}$$

Para uma seqüência de treinamento $\boldsymbol{O}$, $P(\boldsymbol{O}|\lambda)$ é geralmente uma função não linear dos parâmetros que constituem o modelo $\lambda$. Esta função possui vários máximos locais no espaço multidimensional [TUR 98]. O modelo ótimo, $\lambda^*$, corresponde ao máximo global da função.

As equações (2.30)-(2.32) permitem calcular $\overline{\lambda}$, o qual é utilizado iterativamente para substituir $\lambda$. Com o modelo $\overline{\lambda}$ deve-se cumprir que $P(\boldsymbol{O}|\overline{\lambda}) \geq P(\boldsymbol{O}|\lambda)$, ou melhor dito, o novo modelo tem maior probabilidade de produzir a seqüência de observações $\boldsymbol{O}$, convergindo a um modelo $\overline{\lambda}$ correspondente a um máximo local de $P(\boldsymbol{O}|\lambda)$.

O modelo sempre melhorará quando realizado o processo de reestimação, a menos que seus parâmetros já representem um máximo local [HUA 90] [DEL 93]. Devido à existência de diversos máximos locais a reestimação não necessariamente produzirá o melhor modelo possível, $\lambda^*$ [LEV 83]. Assim, torna-se comum na prática executar o algoritmo várias vezes com diferentes conjuntos iniciais de parâmetros [DEL 93] [YU 95], e escolher o $\bar{\lambda}$ que produz o maior valor de $P(\boldsymbol{O}|\lambda)$, levando em consideração um valor de limiar $\varepsilon$, que pode ser inferior a 1% [HUA 90] [DEL 93].

O algoritmo de *Baum-Welch* é esquematizado da seguinte maneira [DEL 93]:

1. *INICIALIZAÇÃO:*

   *Inicia com valores arbitrários para o modelo* $\lambda$

2. *RECURSÃO:*

   *2.1 Utilizar* $\lambda(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$, $\boldsymbol{O}$ *para calcular (2.23) e (2.24).*

   *2.2 Reestimar o modelo* $\bar{\lambda}$ *, utilizando (2.29)-(2.31).*

   *2.3 Se* $P(\boldsymbol{O}|\bar{\lambda}) - P(\boldsymbol{O}|\lambda) \geq \varepsilon$

   *Então voltar ao passo 2.1, utilizando* $\lambda = \bar{\lambda}$

   *Caso contrário deter o algoritmo.*

   *2.4 Repetir os passos anteriores com diversos modelos iniciais para encontrar um apropriado máximo local de* $P(\boldsymbol{O}|\lambda)$.

É necessário observar que se algum conjunto de probabilidades de um *HMM* permanecer fixo, os parâmetros remanescentes ainda poderão ser estimados como indicado anteriormente e a probabilidade poderá melhorar iterativamente [HUA 90]. Além disso, é importante lembrar que $P(\boldsymbol{O}|\lambda)$ terá o mesmo valor quando calculado a partir das probabilidades *forward* ou *backward*:

$$P(\mathbf{O} \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) = \sum_{i=1}^{N} \pi_i . b_i(o_1) . \beta_1(i) = \sum_{i=1}^{N} \alpha_t(i) . \beta_t(i) \qquad (2.33)$$

Uma única seqüência de observações não é suficiente para a reestimação dos parâmetros *HMM* em sistemas práticos de *RAV*. Os dados apropriados de treinamento deverão estar constituídos de um conjunto de seqüências de observações independentes procedentes da mesma fonte. Por exemplo, se cada *HMM* representa uma palavra em um sistema de *RAV*, serão necessárias diversas repetições da mesma palavra para a reestimação dos parâmetros. Na prática, entre 5 e 10 repetições de um locutor são típicas para *RAV* dependente do locutor e mais de 100 palavras de diferentes locutores masculinos e femininos são necessárias para o *RAV* independente do locutor [ZHA 94] [ZHA 95]. Tais múltiplas seqüências de observações poderão utilizar as fórmulas (2.30)-(2.32) generalizadas [DEL 93] [RAB 93].

Define-se $\mathbf{O}=[\mathbf{O}^1,\mathbf{O}^2,...,\mathbf{O}^Q]$ como o conjunto de $Q$ seqüências de observações, onde $\mathbf{O}^q=(o_1^q,o_2^q,...,o_{T_q}^q)$ é a q-ésima seqüência de treinamento, com $T_q$ observações. Assumindo que as seqüências de observações são independentes umas das outras, a estimação dos parâmetros *HMM* estará baseada na maximização da seguinte expressão:

$$\log[P(\mathbf{O} \mid \lambda)] = \sum_{q=1}^{Q} \log[P(\mathbf{O}^q \mid \lambda)] \tag{2.34}$$

Sendo $\sum_{t=1}^{Tq} \gamma_t^q(i,j)$ o número esperado de transições do estado *i* ao estado *j*, estimado a partir de $\mathbf{O^q}$. Além disso, o número médio esperado de transições $\sum_{t=1}^{Tq} \gamma_t(i,j)$ é a soma de $\gamma_t^q(i,j)$ com relação a $\mathbf{O^q}$. Portanto, a equação de reestimação pode ser calculada como:

$$\overline{a}_{ij} = \frac{\sum_{q=1}^{Q} \sum_{t=1}^{Tq-1} \gamma_t^q(i,j)}{\sum_{q=1}^{Q} \sum_{t=1}^{Tq-1} \sum_{j=1}^{N} \gamma_t^q(i,j)} \tag{2.35}$$

Designando $\sum_{t=1}^{Tq} \gamma_t^q(i)$ como a expectativa de estar no estado *i* no tempo *t*, estimado a partir de $\mathbf{O^q}$, similarmente, a equação de reestimação (2.31) pode ser estendida como:

$$\overline{b}_j(k) = \frac{\sum_{q=1}^{Q} \sum_{\substack{t=1, \\ t \in O_t^q = k}}^{T_q} \gamma_t^q(j)}{\sum_{q=1}^{Q} \sum_{t=1}^{T_q} \gamma_t^q(j)} \tag{2.36}$$

É necessário impor um limite inferior na estimativa dos parâmetros $\lambda(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$, do modelo de modo a evitar que eles se tornem muito pequenos [DOS 95]. Como já foi indicado anteriormente na seção 2.3.5, a probabilidade de uma palavra (ou sentença) tende a zero, durante a utilização do algoritmo *Baum-Welch*. Para um número grande de quadros de análise, ocorrerá *underflow* [LEV 83]. Portanto, é necessário algum tipo de escalonamento, como dividir todas as probabilidades de estados para um dado instante *t* pela soma das probabilidades de todos os estados no mesmo instante de tempo, antes de fazer o cálculo das probabilidades no instante posterior. Outra maneira para evitar o *underflow* é representar as probabilidades pelos seus logaritmos [LEE 89].

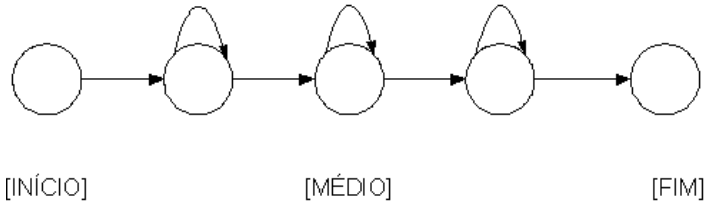
## 2.3.7 RAV Baseado em *HMMs*

Uma característica atrativa do conceito de *HMMs* é sua capacidade de fornecer um método matemático para a execução seqüencial das tarefas relacionadas com o reconhecimento de padrões de voz.

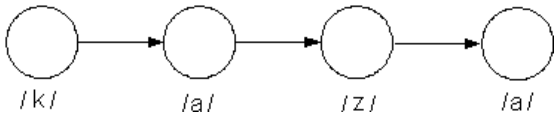
Quando os *HMMs* são utilizados no *RAV*, os estados são interpretados como modelos acústicos, indicando quais sons são prováveis de serem ouvidos durante seus correspondentes segmentos de voz [MAR 97] [GOM 99]. Além disso, as distribuições de probabilidade de saída modelam os eventos de voz e as probabilidades de transição modelam a duração de tais eventos. Assim, um *HMM* é capaz de absorver variações temporais entre elocuções de uma mesma palavra ou sentença. Esta é uma característica bastante desejável quando se quer modelar o sinal de voz, visto que locuções de uma

mesma palavra possuem diferentes durações dependendo, entre outros fatores, do contexto no qual a palavra está inserida, do estado emocional do locutor e das características deste locutor.

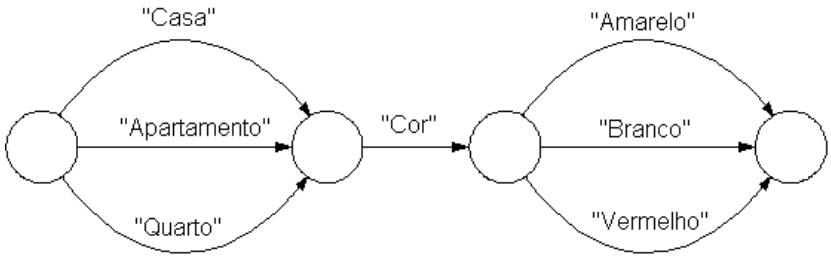
A elocução modelada pelo *HMM* pode ser uma subpalavra, uma palavra, uma sentença completa ou até mesmo um parágrafo. Em pequenos vocabulários, (por exemplo, para o reconhecimento de dígitos), normalmente os *HMMs* são utilizados para modelar palavras, enquanto que para grandes vocabulários os *HMMs* são utilizados para modelar subpalavras (fones, fonemas, etc.). A figura 2.16 ilustra como os estados e transições de um *HMM* podem ser estruturados hierarquicamente, para representar subpalavras (neste caso, fonemas), palavras e frases [TEB 95] [FOS 98].




(a) Nível de Subpalavras.




(b) Nível de Palavras.




(c) Nível de Sentenças.

FIGURA 2.15 - Um modelo oculto de *Markov* hierarquicamente estruturado.

A figura 2.16 mostra o diagrama de blocos de um sistema de *RAV* de palavras isoladas baseado em *HMMs*, com um vocabulário de $V$ palavras. Cada palavra do vocabulário tem um conjunto de treinamento de $K$ repetições (falada por um ou mais locutores). Para cada palavra $v$ no vocabulário, é construído um *HMM* $\lambda^v$, estimando-se os parâmetros de $\lambda(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$ que otimizem a probabilidade do conjunto de vetores de observações utilizados para o treinamento da *v-ésima* palavra. Para cada palavra que deve ser reconhecida, é lida a seqüência de observações $\boldsymbol{O}$, fazendo uma análise de características da elocução correspondente, seguido pelo cálculo das probabilidades de cada modelo $P(\boldsymbol{O}|\lambda^V)$, $1 \le v \le V$, e a seleção daquele cuja probabilidade seja a mais alta:

$$v^* = \arg \underset{1 \le v \le V}{máx} \left[ P(\mathbf{O} \mid \lambda^v) \right] \tag{2.37}$$

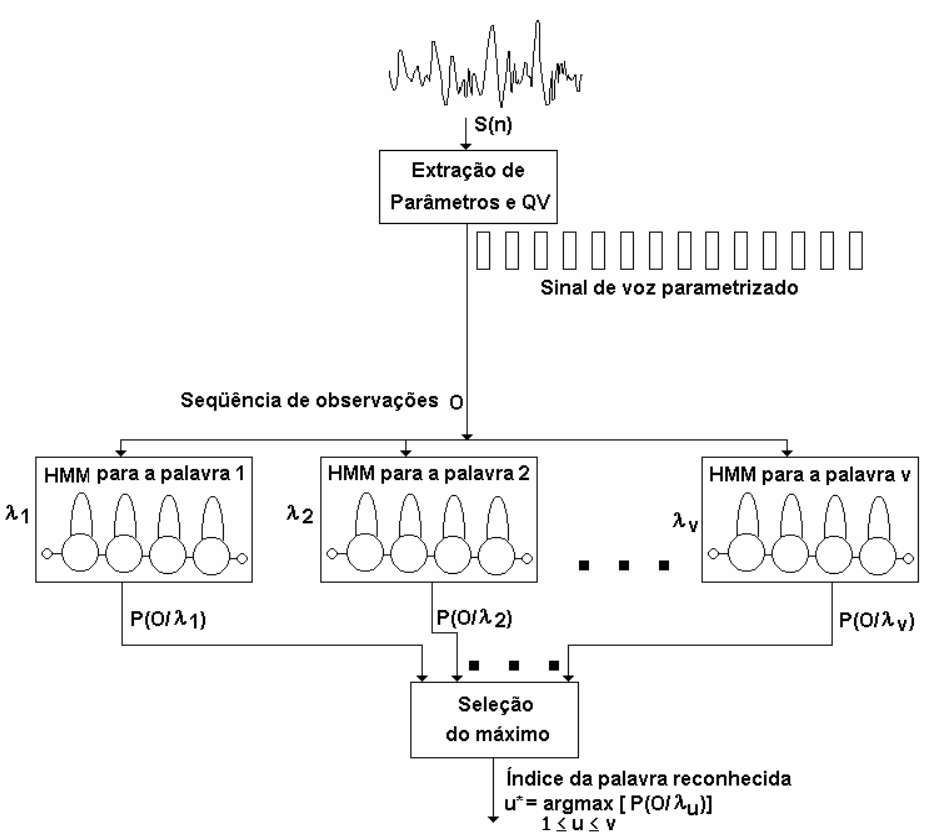



FIGURA 2.16 - Sistema de *RAV* para palavras isoladas sem nenhuma gramática

O treinamento de *HMMs* em fala contínua é muito similar ao treinamento de palavras isoladas. Uma das grandes vantagens da modelagem com *HMMs* é que estes podem absorver informação referente aos limites entre palavras para fala contínua [HUA 90] [TEB 95]. Visto que a seqüência de estados nos *HMMs* é oculta, então não interessa onde se encontram os limites entre palavras. Para treinar os parâmetros dos *HMMs*, precisa-se da seqüência de palavras dentro de cada sentença. Cada palavra é instanciada com seu modelo, o qual pode ser uma concatenação de *HMMs* de subpalavras [MOR 95] [FOS 98]. Depois disso, as palavras na sentença são concatenadas com modelos opcionais de silêncio entre elas. Este *HMM* concatenado da sentença é então treinado para a sentença completa. Visto que o *HMM* da sentença é treinado para a sentença completa, todos os limites das palavras estarão sendo considerados em cada quadro [LEE 89].

# 3 Sistemas de Reconhecimento de Voz em Hardware

## 3.1 Introdução

Diversos algoritmos *DSP* que são utilizados no *RAV* têm sido implementados em circuitos integrados, com o intuito de diminuir o tempo de processamento e também realizar o reconhecimento de mais palavras do que com processadores de propósito geral [DEN 85] [CAU 97].

As pesquisas em *RAV* em software têm procurado geralmente a maior taxa de reconhecimento e isto tornou necessário o desenvolvimento de novos e melhores algoritmos [RAV 96]. Na maioria dos casos, um algoritmo que ajude a obter taxas mais altas de reconhecimento, exige uma modelagem mais próxima do processo de produção e percepção da fala levando em consideração características reais. Uma modelagem assim, normalmente exige maior tempo de processo computacional. Neste caso, podem utilizar-se circuitos integrados de aplicação específica que podem servir para executar operações de maneira mais rápida do que um sistema de *RAV* implementado em software ou até mesmo com processadores de propósito geral.

Nos últimos anos, têm sido projetados alguns sistemas baseados em hardware de propósito específico, focalizando diversos aspectos complexos do *RAV*. Grande parte de tais sistemas foi implementada em uma placa de circuito impresso com uma entrada analógica e em operação conjunta com um computador, que comanda a comunicação entre os diversos subsistemas que compõem a placa e serve de interface com o usuário. Os algoritmos que são executados em tais placas são usualmente dependentes do locutor e para palavras isoladas. Isto quer dizer que cada usuário deve treinar a placa com sua própria voz, e para cada palavra, falando cada palavra claramente e com pausas entre elas.

Na continuação, descreveremos alguns trabalhos mais relevantes na implementação em hardware de sistemas de *RAV*.

## 3.2 O Sistema SBR

A *AT&T* implementou um sistema de *RAV*, denominado *SBR* (***S**ingle-**B**oard **R**ecognizer*) e descrito em [ACK 86]. Tal sistema está baseado na técnica de ajuste temporal dinâmico (*DTW*) e utiliza coeficientes *LPC*. O sistema *SBR* é dividido para realizar as tarefas de análise das características, comparação de padrões e controle do sistema, assim como mostrado na figura 3.1.

Para a extração de parâmetros, o sistema *SBR* utilizou o processador digital de sinais programável, *AT&T DSP20* [ACK 85], que recebe as amostras quantizadas do sinal de voz. O sinal de voz é então dividido em quadros de 300 amostras (45ms), calculando um vetor de características para cada quadro. Um novo quadro é iniciado cada 100 amostras (15ms). São produzidos 9 parâmetros *LPC* por cada quadro, codificados com 12 bits.

O controle de operações e a comunicação entre o processador de *DTW* e o de pré-processamento além da entrada e saída de dados, são executados por um microprocessador de 16 bits [ACK 86]. Tal microprocessador armazena valores *LPC* tanto dentro da memória de teste como na memória de referência. O sistema foi projetado para palavras isoladas, com um vocabulário de 150 palavras.

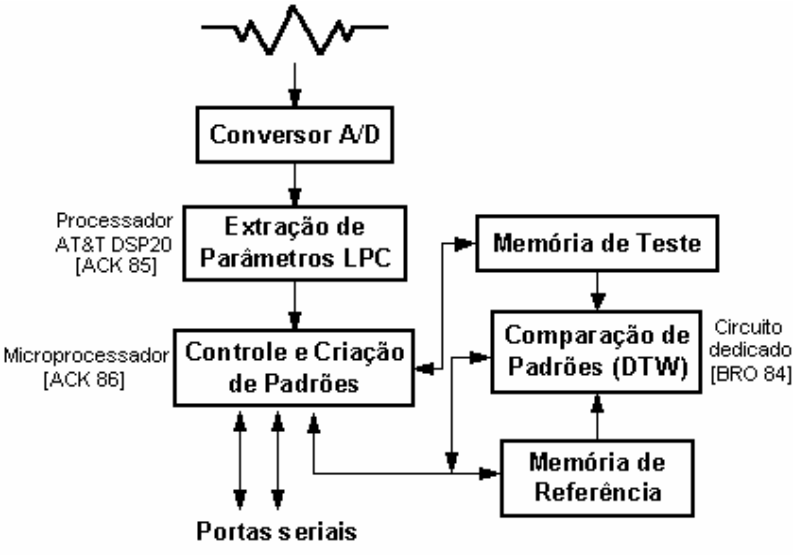



FIGURA 3.1 - Arquitetura do Sistema SBR.

A comparação de padrões é realizada utilizando um circuito integrado de aplicação específica [BRO 84], que calcula as distâncias entre o padrão de entrada desconhecido e os padrões de referência armazenados, utilizando *DTW*. Somando as distâncias entre o padrão de teste e o padrão de referência ao longo do caminho de alinhamento temporal ótimo, obtêm-se a distância total entre ambos padrões. Os valores de distância são enviados para o bloco de controle para processar as decisões. Finalmente, o subsistema de controle envia os resultados de reconhecimento para um computador central ou para um terminal através de portas seriais. Os três subsistemas e outros componentes são arranjados em uma placa de circuito impresso de 20cm x 25cm.

A arquitetura do subsistema de *DTW* é mostrada na figura 3.2. Depois do cálculo das distâncias parciais $d_{ij}$, entre os parâmetros do quadro $i$ do padrão de teste e os parâmetros do quadro $j$ do padrão de referência, calcula-se a distância total. Uma distância acumulada é calculada em cada ponto e consiste da distância local e o mínimo de três valores predecessores possíveis: $D(i-1, j)$, $D(i-1, j-1)$ e $D(i-1, j-2)$, de maneira que:

$$D(i,j)=d_{ij}+mín[D(i-1, j), D(i-1, j-1), D(i-1, j-2)] \qquad (3.1)$$

A distância acumulada é colocada dentro de um registrador de deslocamento de 14 estágios, como indicado na figura 3.2, onde estará disponível para os cálculos de distância entre o seguinte quadro de teste e os quadros de referência. Utilizam-se 96Kx12-bits de memória para os padrões de referência e 1Kx12bits para o armazenamento temporário dos padrões de teste.

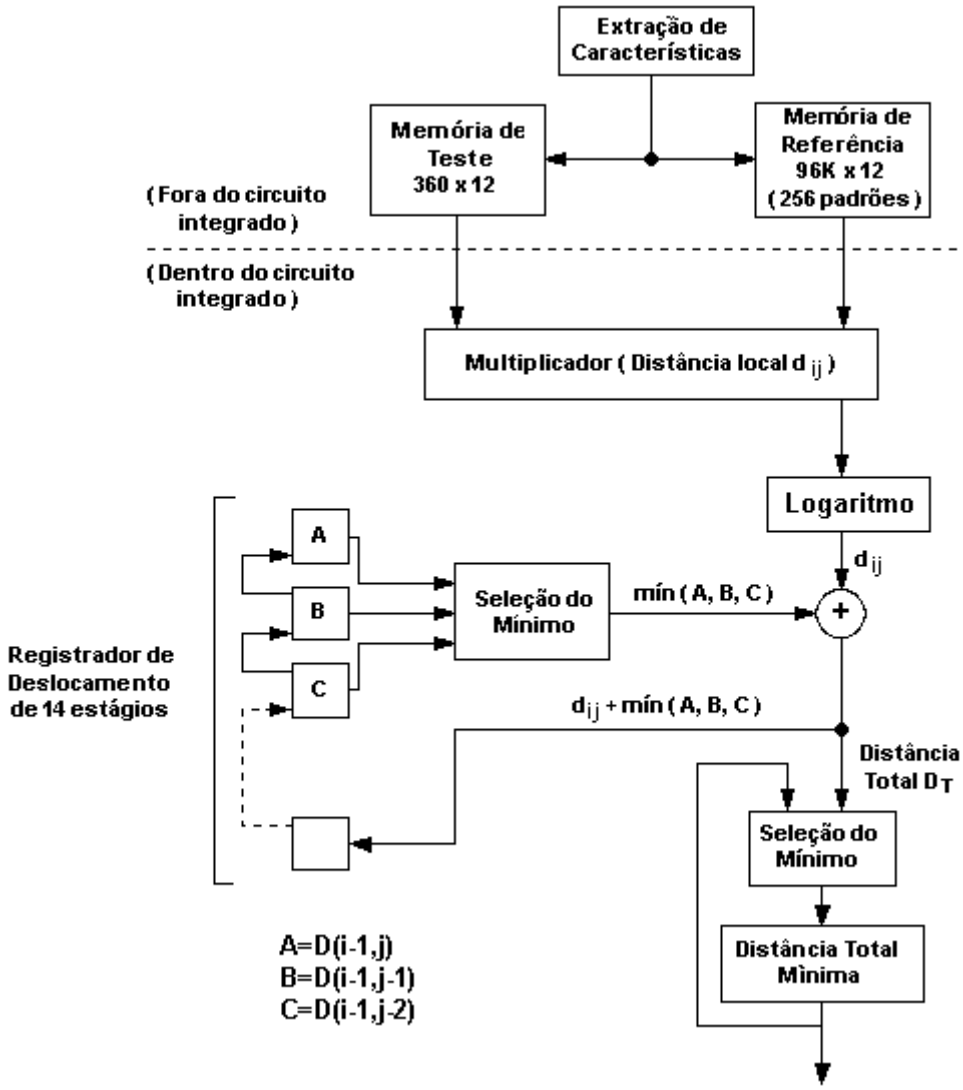



FIGURA 3.2 - Arquitetura do processador de DTW-SBR.

Um conjunto de módulos escritos em linguagem C para a avaliação, controle e o desenvolvimento de aplicações com o *SBR*, foi implementado em um computador pessoal, comunicando-se com outros dispositivos através das portas seriais do *SBR*, como mostra a figura 3.3.

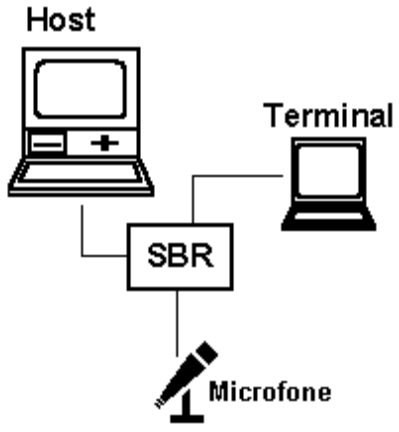



FIGURA 3.3 - Operação do SBR.

## 3.3 A Arquitetura GSM

A *AT&T* projetou também a arquitetura *GSM* (***G**raph **S**earch **M**achine*) uma arquitetura *VLSI* programável para o cálculo eficiente de diversas operações básicas para *RAV*, utilizando *HMMs* e *DTW* [GLI 87]. Após a conversão *A/D*, o resultado é transmitido para um processador *DSP*.

A arquitetura *GSM* foi projetada com o objetivo de não ser custosa para suportar configurações em arranjo, ser programável para suportar modificações nos algoritmos e seu uso em aplicações diversas, e ter uma comunicação simples com outros dispositivos.

Como muitas arquiteturas *DSP*, a *GSM* utiliza memórias tanto dentro como fora do processador integrado, para aumentar seu desempenho. As memórias internas consistem de uma *RAM* de 32x32 (cache de programa) e uma *ROM* de 32x32. Os programas do usuário são executados a partir da cache, que é carregada sob o controle de vários programas utilitários baseados em *ROM*. Outros programas utilitários de *ROM* são utilizados para transferir dados entre os dois barramentos externos, e para iniciar e deter a *GSM*. A cache permite que a toda a memória externa (local) seja utilizada para o armazenamento de programas. Devido a que são pouco freqüentes os acessos à memória local, a porta da memória local está liberada quase exclusivamente para os dados E/S. A memória local tem 64Kx16bits e mantém dados, assim como programas. Nesta memória acessível na velocidade do relógio *GSM*, armazenam-se os padrões de referência. Uma porta para processamento, de 8-bits e uma porta para memória local, de 16 bits adicionam flexibilidade e reduz a lógica de comunicação.

Em razão de que as operações de extração de características (filtragem, autocorrelação, *LPC*) e cálculo de distâncias têm, tipicamente, um grande número de operações de multiplicação-soma, um processador *DSP* é indicado para realizá-las. O restante de operações que possuem, geralmente, um grande número de operações soma-mínimo, é realizado pela arquitetura *GSM*.

O *GSM* foi projetado utilizando tecnologia *CMOS* de 1,75μm, ocupando uma área de 45mm$^2$ e operando a uma velocidade de 8MHz.

## 3.4 O sistema UCB-DTW

A Universidade de Califórnia, Berkeley (*UCB*), projetou um sistema de *RAV* para palavras isoladas e um vocabulário de 1000 palavras utilizando a técnica de *DTW* [KAV 87]. O sistema foi implementado em uma única placa de barramento múltiplo, seguindo a idéia de operação do *SBR*. O sistema contém três subsistemas: um subsistema de propósito geral, um subsistema de pré-processamento e um subsistema de casamento de padrões. Todos os subsistemas trabalham em paralelo, conectados através das linhas de endereços e de dados, fornecidas pelo subsistema de propósito geral utilizando arquiteturas especificas.

O subsistema de propósito geral foi projetado utilizando um microprocessador 80186 da Intel.

O subsistema de pré-processamento contém um conversor *A/D*, um banco de filtros digitais, e uma *FIFO* que interrompe o 80186 a cada 10ms, com 16 parâmetros espectrais de energia. Cada característica espectral é representada por 4 bits, sendo necessários 64 bits para representar os 16 parâmetros espectrais. O banco de filtros digitais do subsistema de pré-processamento foi implementado em um circuito integrado.

No referente ao subsistema de casamento de padrões, este possui uma memória para a armazenagem dos padrões de referência. Tal memória foi implementada utilizando circuitos integrados padrão de 64Kx1 de *RAM* dinâmica. Cada palavra da memória possui 16 bits, sendo que um quadro do sinal de voz que é representado por 16 vetores espectrais (4 bits cada) precisa de 64 bits ou de 4 palavras da memória. Além disso, utilizaram-se circuitos padrão de 16Kx4 de *RAM* dinâmica para armazenar os resultados parciais dos cálculos do algoritmo de *DTW*.

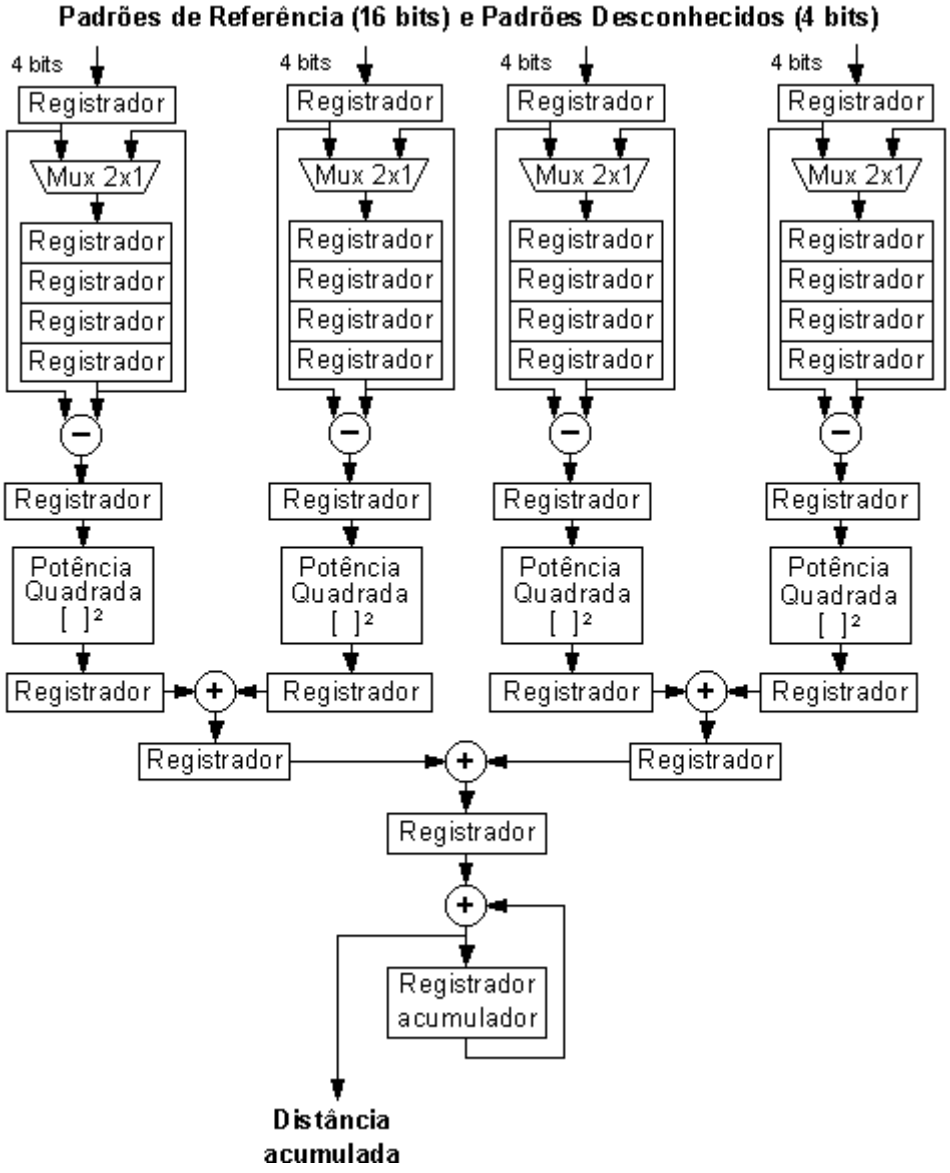



FIGURA 3.4 - Diagrama de blocos da parte operativa do subsistema UCB-DTW.

O centro do subsistema de casamento de padrões é o circuito integrado de *DTW*. Este circuito realiza o calculo das distâncias, endereçamento das memórias, e seqüenciamento de estados. O diagrama do datapath do subsistema de *DTW* é mostrado na figura 3.4. Cada quadro dos padrões de referência é lido da memória requerendo 4 ciclos de relógio, visto que ocupa 4 palavras da memória. Ao ser lida da memória, cada palavra de 16 bits é dividida em 4 grupos de 4 bits (*nibbles*) e diminuída dos *nibbles* correspondentes do padrão desconhecido. Cada valor resultante é elevado ao quadrado, gerando valores de 8 bits. Depois, os resultados são agrupados, somados e o resultado final é carregado dentro de um acumulador. A saída do acumulador será a distância Euclidiana.

O circuito de *DTW* foi implementado utilizando técnicas de paralelismo e pipeline para poder calcular de maneira rápida as medidas de distância. Os primeiros protótipos foram realizados utilizando componentes discretos *TTL* (***T**ransistor-**T**ransistor **L**ogic*). O circuito integrado final foi implementado em um processo *NMOS* de 4μm.

## 3.5 O Subsistema de Processamento de Palavras UCB-HMM

Um outro trabalho, realizado na *UCB*, é descrito em [STO 91], onde se apresenta a implementação de dois circuitos integrados que fazem parte de um subsistema de processamento de palavras para reconhecimento de linguagem contínua e vocabulário grande, em tempo real. O sistema está baseado em Modelos Ocultos de Markov, focalizando o processamento de Viterbi e a comunicação com memórias associadas.

O diagrama de blocos do subsistema é mostrado na figura 3.5. O subsistema possui três componentes principais, para o modelamento com *HMMs*, sendo que o pré-processamento, o cálculo de parâmetros e a quantização são realizados externamente.

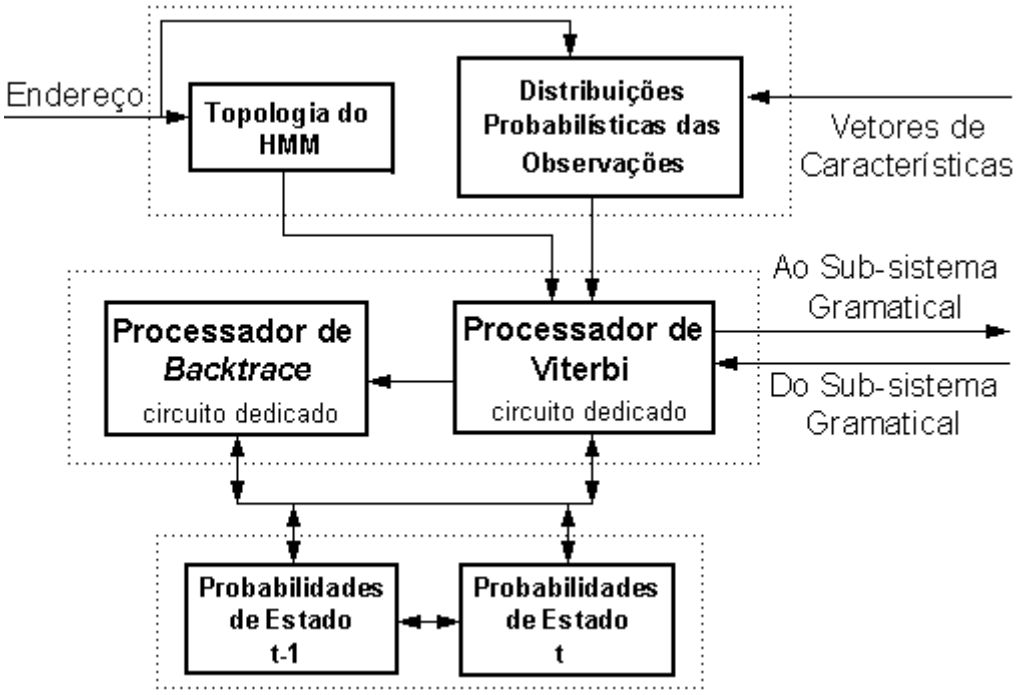



FIGURA 3.5 - Diagrama de blocos do subsistema de processamento de palavras.

Um dos componentes mantém toda a informação associada com os *HMMs* das palavras do vocabulário, dentro da memória que armazena a topologia do *HMM* e a memória de probabilidades de saída. Cada estado do *HMM* é associado com uma palavra da memória onde é descrita a posição de, no máximo, três estados predecessores que estão conectados ao estado que está sendo avaliado. As distribuições de probabilidade das observações são armazenadas em outra memória, a qual utiliza o mesmo endereço da memória que contém a topologia. Para a implementação deste subsistema foram utilizados dispositivos discretos.

Outro componente armazena os valores temporários das probabilidades de estado e os índices de backtrace para dois quadros consecutivos *t-1* e *t*. Este componente utiliza o mesmo endereço que as memórias que contem o *HMM* utilizam. Cada palavra da memória *t-1* armazena a probabilidade de estado e o índice de backtrace do estado associado com seu endereço. Em cada iteração do algoritmo de Viterbi, esses valores são lidos da memória *t-1*, para atualizar esses valores e escrevendo os resultados na memória *t*. Depois do processamento de um quadro, os valores da memória *t-1* são atualizados. Este subsistema também foi implementado com componentes discretos.

O terceiro componente, é um elemento de processamento, dividido em dois circuitos integrados dedicados, o processador de Viterbi e o processador de backtrace, que permitem fazer a decodificação de estados e os cálculos das probabilidades de observação de cada *HMM*. É utilizada a *VQ* das características do sinal de voz, e as densidades de observação são funções discretas. Contudo, o trabalho se concentra na implementação do processamento de Viterbi e o backtracing.

Todas as probabilidades associadas com os *HMMs* são representadas com seu logaritmo em valor absoluto. Portanto, todas as multiplicações envolvidas no cálculo de $P(\boldsymbol{O}|\lambda)$ podem ser implementadas com somas.

A figura 3.6 mostra a arquitetura do processador de Viterbi, que atualiza os valores das probabilidades segundo as equações recursivas do algoritmo de Viterbi. É considerado que, no máximo, existem três estados predecessores para cada estado e se utiliza uma modelagem de palavras, daí a utilização de três memórias. São utilizadas memórias de acesso de dupla porta que permitem o cálculo, em um único ciclo, dos três estados predecessores no modelo de uma palavra. O endereço de escrita que é o mesmo para todas as memórias é gerado por um contador. Os endereços de leitura são calculados de maneira relativa ao endereço de escrita utilizando um valor de deslocamento fornecido pela memória que contém a topologia.

As três probabilidades de estado na saída das memórias são multiplicadas pelas probabilidades de transição, e esta operação é implementada mediante a soma dos logaritmos das probabilidades Depois disso, a probabilidade com o menor valor é selecionada. A este último valor, é somada a probabilidade de observação, para assim calcular finalmente a probabilidade de estado. Para minimizar o tamanho de palavra dos valores de probabilidade, o processador normaliza os valores finais, utilizando a melhor probabilidade total do quadro anterior.

Os circuitos integrados de Viterbi e backtracing foram fabricados utilizando uma tecnologia *CMOS* de 2μm.

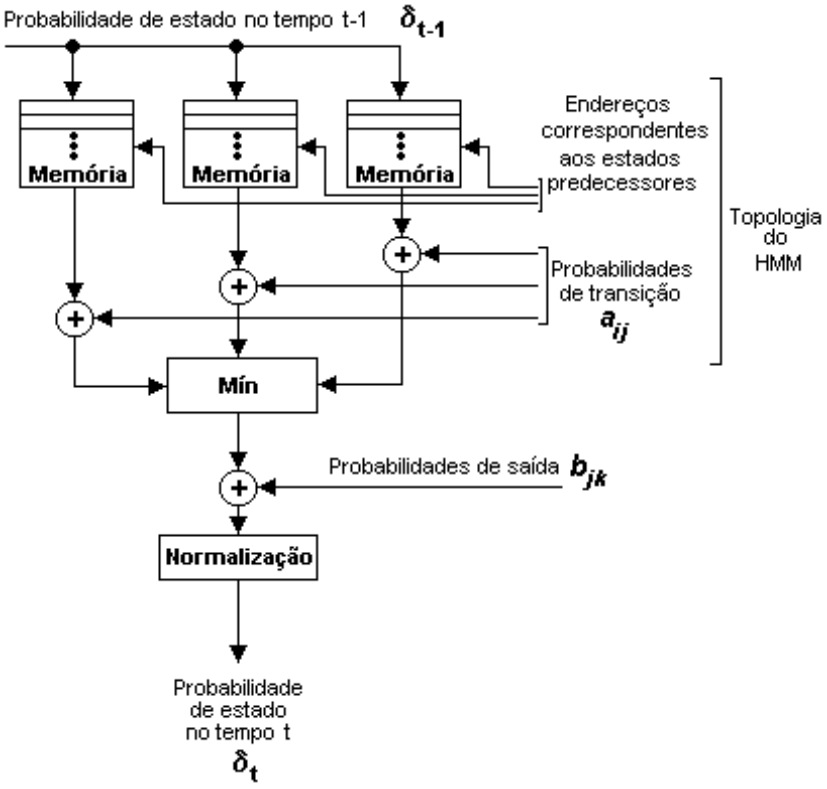



FIGURA 3.6 - Arquitetura do processador de Viterbi.

## 3.6 O Decodificador de Estados UPM-HMM

A Universidade Politécnica de Madri (UPM) realizou a implementação de um sistema para o reconhecimento de palavras isoladas com um vocabulário de 1000 palavras [MOR 93] [FER 94]. Uma primeira versão do sistema foi implementada utilizando uma rede de transputers, com a mesma etapa de pré-processamento.

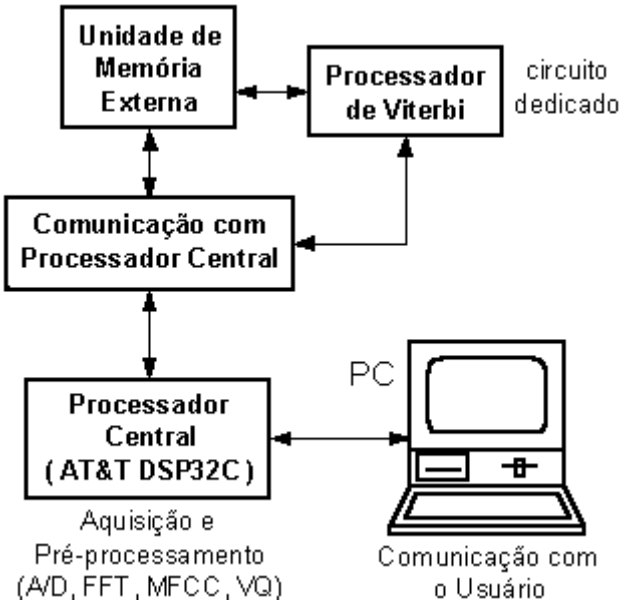



FIGURA 3.7 - Arquitetura do sistema de *RAV* da UPM.

A etapa de pré-processamento (que inclui a conversão *A/D*, filtro *antialiasing*, janelamento, *FFT*, extração de coeficientes mel-cepstrais (*MFCC*) e a quantização vetorial (*VQ*)) foi implementada utilizando um processador *AT&T DSP32C*. Foi projetado um circuito integrado em tecnologia *CMOS* de 1,5μm para as operações relacionadas com a decodificação de estados. A arquitetura da etapa de decodificação de estados é bastante similar ao do sistema apresentado no item 3.5. Os outros elementos foram implementados com componentes discretos padrão, entre eles a memória externa que armazena as probabilidades de observação, os identificadores de fonemas e as probabilidades do modelo. O processador *DSP32C* era o encarregado de coordenar também a comunicação entre todos os componentes. A figura 3.7 mostra o diagrama deste sistema.

O sistema final relata uma taxa média de reconhecimento de 68%, para um vocabulário de 1000 palavras e em modo de operação independente do locutor, para uma base de dados de 70 locutores [MOR 93] [FER 94].

## 3.7 O Sistema UCB-Infopad

Os trabalhos iniciais da *UCB* em sistemas de *RAV* em hardware, descritos em parágrafos anteriores, foram posteriormente utilizados para novos projetos. [CHA 94] descreve o Infopad, um projeto multidisciplinar realizado na *UC* em Berkeley, que tinha como objetivo a implementação de um sistema pessoal de comunicações para comunicação portável. A idéia era obter um terminal operacional pequeno, leve, de baixa potência (usando baterias) e capaz de operar com texto, gráficos, áudio, vídeo. O envio de dados multimídia deveria ser realizado através de uma conexão sem fio e em tempo real. A figura 3.8 mostra um diagrama do sistema *Infopad*.

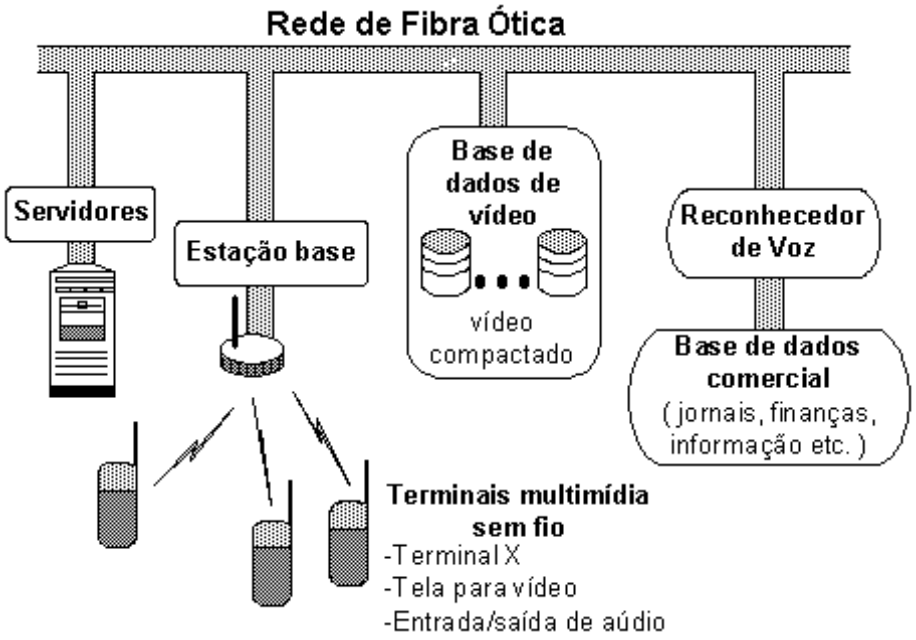



FIGURA 3.8 - Sistema Infopad.

A maior parte da carga computacional seria realizada, na rede remota e pelo servidor. O terminal somente deveria servir para a entrada e saída de dados, reduzindo o

custo, peso e consumo de potência. A entrada de dados deveria ser realizada através de caneta eletrônica e sinal de áudio, com as vantagens de não usar teclado.

O Infopad utiliza um sistema de *RAV* para fala continua, independente do locutor, vocabulário pequeno e baixa perplexidade, em software. Tal sistema de *RAV* deve funcionar na rede. Este tipo de reconhecimento é conhecido pelo nome de *reconhecimento remoto de voz*. Também foram exploradas técnicas de baixa potência para a implementação de circuitos integrados que se encarreguem das tarefas que são realizadas localmente, tais como multiplexação e demultiplexação de pacotes, decodificação de vídeo e atualização de buffers.

O objetivo do sistema Infopad, é colocar a maior carga operacional na rede. Por tal motivo o *RAV* deveria ser efetuado por um servidor utilizando um sistema de *RAV* em software. Porém, a possibilidade de realizar o *RAV* no próprio terminal portável permitiria a diminuição da largura de banda requerida para a transmissão de dados de áudio. Ao fazer o reconhecimento no terminal, unicamente seria necessária a transmissão de uma representação *ASCII* das palavras reconhecidas, permitindo reduzir o consumo e complexidade do enlace de rádio e a operação em regiões de baixa largura e banda. Tais condições são obtidas na medida que os algoritmos de *RAV* são implementados utilizando hardware específico e colocados junto com o terminal. Basicamente foram implementadas memórias com técnicas de baixa potência [BUR 96]. Parte do sistema foi implementada e outras partes constituem trabalhos futuros. Os objetivos foram demonstrar a possibilidade de realização de algoritmos de *RAV* em hardware portável e com baixo consumo e contribuir no projeto *CMOS* de baixa potência.

## 3.8 Extração de Características em Terminais Remotos

Um trabalho mais simples do que o Infopad foi realizado em [BOR 98], onde é descrita a implementação em silício de um algoritmo simplificado para a extração de características do sinal de voz. A implementação de tal algoritmo utilizando um *ASIC* é a principal contribuição do trabalho relatado em [BOR 98].

Os parâmetros extraídos são codificados pelo circuito e transferidos remotamente através de um sinal de rádio. Os dados são recebidos por uma estação base que completa a tarefa de reconhecimento. A figura 3.9, mostra a parte de aquisição e extração de características de um terminal portável executando de maneira remota o *RAV*.

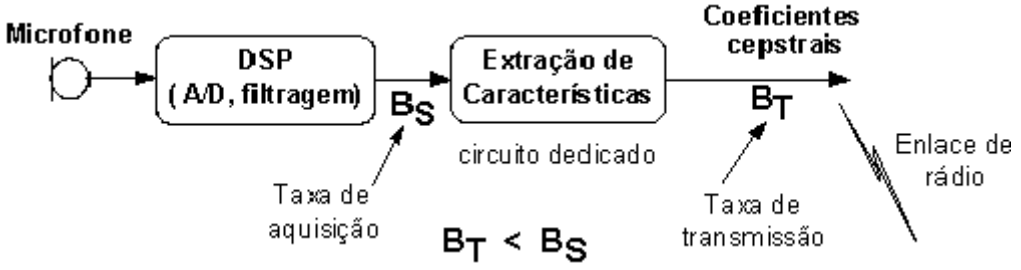



FIGURA 3.9 - Aquisição da voz e extração de características em terminal remoto.

A estação base utiliza para o reconhecimento, técnicas de ajuste temporal dinâmico (*DTW*).

Utilizou-se tecnologia *SOG* (*Sea of Gates*) *CMOS* de 0,5μm, calculando 15 parâmetros cepstrais, cada 8ms.

## 3.9 O Co-processador de Funções Gaussianas

[PIH 96] dá atenção à implementação das funções de distribuição de probabilidades (*pdf*) das observações dentro do problema de reconhecer linguagem contínua utilizando *HMMs*. Não é realizado nenhum projeto das etapas de extração de parâmetros, decodificação de estados e reconhecimento do sinal de voz. Procura-se melhorar o desempenho no cálculo das operações de multiplicação e potência quadrada como parte da avaliação de funções gaussianas correspondentes às funções de distribuição de probabilidade (*pdf*) de cada quadro.

O objetivo é a implementação de bibliotecas de células utilizando as técnicas de circuitos de *TSPC* (*True Single Phase Clocking circuit technique*) e *CDPD* (*Clock and Data Precharged Dynamic circuit technique*) dentro de ferramentas de síntese lógica e otimização [PIH 96a]. Outro objetivo é a geração de multiplicadores, circuitos para calcular o quadrado de números, somadores e subtratores de alta velocidade em aplicações *DSP* de alta velocidade. Tais técnicas são utilizadas para a implementação de um circuito *VLSI*, em tecnologia *CMOS* de 0,8μm.

O *ASIC* implementado serve como co-processador em um subsistema de reconhecimento com *HMMs*, podendo trabalhar em conjunto com um microprocessador ou um processador de sinais, no cálculo de funções gaussianas. A extração de características e o controle das tarefas podem ser realizados através de uma placa *DSP* e outros blocos funcionais, os quais não foram implementados em hardware. O circuito requer bancos de memória externa para armazenar os valores estimados dos parâmetros das gaussianas. O circuito co-processador utilizou 35k transistores, ocupando uma área de 16,5 $mm^2$ e uma freqüência de relógio máxima de 154 MHz.

## 3.10 Conclusão

Foram descritos os trabalhos existentes no que se refere a implementações em hardware, de sistemas de *RAV*. A tabela 3.1 mostra um resumo de tais sistemas, listando as características que cada sistema indica nas publicações que cada laboratório disponibiliza para a comunidade.

Unicamente o sistema UPM-HMM publica dados sobre taxa de reconhecimento, a qual é de 68%. Os demais trabalhos não relatam nenhum resultado quanto a que taxa de reconhecimento operam.

Existem alguns dispositivos comerciais dos quais não existe nenhum detalhamento de uso público sobre a implementação e, portanto, não serão descritos aqui.

TABELA 3.1 - Sistemas de *RAV* em hardware.

| Sistema | Técnica | Estilo da voz | N. de circuitos dedicados | Tempo de ciclo (ns) | Área ($mm^2$) | Tecnologia (μ) |
|---|---|---|---|---|---|---|
| **SBR [ACK 86]** | DTW | Palavras isoladas | 1 | 250 | 81 (**) | 2,5CMOS |
| **GSM [GLI 87]** | DTW e HMM | Palavras isoladas e ling. contínua | 1 | 125 | 45 | 1,75CMOS |
| **UCB-DTW [KAV 87]** | DTW | Palavras isoladas | 1 | 200 | 13 | 4NMOS |
| **UCB-HMM [STO 91]** | HMM | Ling. contínua | 2 | 200 | 114 (viterbi) 51 (backtrack) | 2CMOS |
| **UPM-HMM [FER 94]** | HMM | Palavras isoladas | 1 | 40 | 78 | 1,5CMOS |
| **Terminal Remoto [BOR 98]** | DTW | Palavras isoladas | 1 | - | Sea of Gates (100K) | 0,5CMOS |
| **Pdf Coproc. [PIH 96]** | HMM | Ling. contínua | 1 | 6,5 (*) | 16,5 | 0,8CMOS |

(*) = apenas para o coprocessador de pdf.
(**)= estimado por [BRO 84]

# 4 Modelagem em SW do RAV com HMMs

## 4.1 Introdução

Uma primeira etapa do trabalho proposto envolveu o estudo dos algoritmos utilizados em *RAV*. Isto foi necessário para ter um conhecimento intrínseco de tais algoritmos. Além disso, o estudo dos algoritmos de *RAV* tornou possível obter uma comparação quantitativa do desempenho das diversas técnicas de reconhecimento, diferentes parâmetros característicos da voz, modos de operação do *RAV* e outros. Além disso, o estudo de tais algoritmos permite identificar tarefas que podem ser otimizadas visando um melhor desempenho de velocidade, uma simplificação e uma especificação mais clara.

Com esse objetivo, foram implementadas rotinas de software para os algoritmos de pré-ênfase, extração de características, quantização vetorial, modelagem de *Markov* e *DTW*. Desta maneira tem-se uma descrição comportamental das técnicas que permitirá fazer um estudo posterior em um nível inferior

## 4.2 HMMs vs. Outras Técnicas

Foi realizado um estudo comparativo entre as técnicas *DTW*, *VQ* e *DHMM (Discrete HMM*) ou HMM discreto. Tal estudo foi feito com palavras isoladas, nos modos dependente e independente do locutor. A base de dados utilizada é a mesma do projeto *REVOX* [REV 98] [PET 99], formada por 10 palavras que vão da frase "primeiro andar" até a frase "décimo andar", para ser utilizada no controle de um elevador. Esta base de dados foi coletada, em um ambiente com ruído, com uma freqüência de amostragem de 11025Hz e resolução de 16 bits. Esta base foi utilizada, nos experimentos descritos na continuação.

Ao implementar a técnica de ajuste temporal dinâmico, utilizaram-se parâmetros mel-cepstrais quantizados previamente.

A quantização vetorial foi modificada, criando subdivisões dos arquivos de vozes e gerando um dicionário de códigos para cada subdivisão. Os parâmetros acústicos quantizados escolhidos nesta segunda técnica, também foram os parâmetros mel-cepstrais.

Assim como no caso do ajuste temporal dinâmico, os modelos de *Markov* utilizaram parâmetros mel-cepstrais quantizados, e uma modelagem de palavras.

A tabela 4.1 apresenta a comparação entre as taxas de reconhecimento obtidas com as técnicas de *RAV* descritas nos capítulos anteriores, para um sistema de palavras isoladas. O caso apresentado utiliza *HMMs* discretos (*DHMMs*). Da tabela indicada pode se observar que o melhor desempenho no modo independente do locutor é obtido pela técnica *DTW*. No modo dependente do locutor, a melhor taxa é obtida pela técnica *VQ* junto com o cálculo da menor distorção. Em todos os casos nesta tabela, os *HMMs* fornecem taxas de reconhecimento mais baixas em relação às outras técnicas. Um comportamento assim, também é observado em [RAB 89] e em [ZHA 95]. Uma

melhora na taxa de reconhecimento pode ser obtida pelo aumento dos arquivos de treinamento, como é indicado em [ZHA 94] e [ZHA 95]. No entanto, a base de dados utilizada no presente trabalho tem tamanho limitado o qual não permitiu verificar tal afirmação.

TABELA 4.1 - Taxas de reconhecimento, para sistemas de palavras isoladas.

| **Técnica** | **DTW** | | **VQ** | | **DHMM** | |
|---|---|---|---|---|---|---|
| **Modo** | **SD** | **SI** | **SD** | **SI** | **SD** | **SI** |
| **N$^{o}$. de Arquivos de Treinamento** | 740 | 740 | 740 | 740 | 740 | 740 |
| **N$^{o}$. de Arquivos de Teste** | 370 | 240 | 370 | 240 | 370 | 240 |
| **Taxa de Reconhecimento** | 94,65% | 97,56% | 97,84% | 92,92% | 91,35% | 85,96% |

SD = Dependente do locutor.
SI = Independente do locutor.

Na figura 4.1, são mostradas as taxas de reconhecimento para um sistema de *RAV* baseado em *DHMM*, em função do número de estados, no modo dependente do locutor, porém com um número de arquivos de treinamento diferente do caso anterior. Pode-se observar o aumento da taxa de reconhecimento quando se aumenta o número de estados. Porém, tal relação não continua indefinidamente, pois ao aumentar o número de estados a taxa de reconhecimento se degrada. Quando o número de estados é 8 ou 9, a taxa de reconhecimento fica maior. Um número de estados razoável seria da ordem de 8 estados. Outros pesquisadores recomendam valores que variam de 6 até 8 estados [RAB 82] [RAB 93] [NUN 96]. O aumento do número de estados, incrementa o tempo de treinamento e o requerimento de memória de parte do sistema *HMM*, não produz melhoras significativas no desempenho e inclusive observa-se degradação. Por tais motivos, decidiu-se por utilizar 8 estados

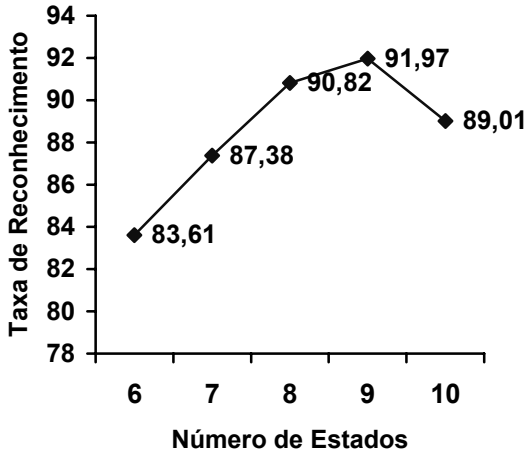



FIGURA 4.1 - Taxa de reconhecimento em função do número de estados para um sistema de *RAV* baseado em *DHMM*.

Quando se trata de reconhecimento de locutor (verificando o grau de aceitação), os resultados são qualitativamente similares, como pode ser observado na tabela 4.2. A relação qualitativa entre as técnicas apresentadas na tabela 4.2 é também observada em outros sistemas de reconhecimento de locutor tais como [MAT 94] e [YU 95], onde

melhoras nas taxas de reconhecimento com *HMMs* somente são obtidas a partir do uso de *HMMs* contínuos (*CHMMs*) [CAM 97].

TABELA 4.2 - Taxas de reconhecimento, para a identificação de locutor.

| **Técnica** | **DTW** | **VQ** | **DHMM** |
|---|---|---|---|
| *Taxa de Reconhecimento* | 94,65 | 85,68 | 83,33 |

Os resultados anteriores mostram que as taxas de reconhecimento, ao utilizar *HMMs* são inferiores às obtidas quando se utilizam técnicas tais como *VQ* e *DTW*. Porém, os *HMMs* são uma técnica muito utilizada por diversas razões. Uma das razões é que os *HMMs* oferecem a possibilidade de ser utilizados em sistemas híbridos *HMM/ANN* com os quais se obtêm melhores resultados do que com as técnicas anteriormente consideradas. Uma taxa de *97,71%* foi obtida para reconhecimento independente do locutor, quando se utiliza uma abordagem híbrida *HMM/ANN* e vozes gravadas em um canal com ruído [SCH 2000].

Outro motivo é o menor consumo de memória que os *HMMs* têm em relação às outras técnicas. Nos *HMMs* se utiliza um conjunto de modelos de uma estrutura particular, além de um conjunto grande de dados de treinamento para extrair os parâmetros de cada modelo *HMM*. No caso de um *HMM* de $N$ estados, com $M$ símbolos finitos por estado, é necessário estimar um total de $N^2+N.M$ parâmetros. Quando $N=8$ e $M=64$, serão necessários 576 parâmetros. Estes 576 parâmetros representarão uma palavra de um determinado vocabulário, independente do número de quadros e do número de parâmetros acústicos de uma palavra. O processo de treinamento dos *HMMs* tem um custo computacional maior do que no caso de *DTW* e *VQ*, mas é um processo que precisa ser feito apenas uma única vez. Depois de ter feito o treinamento, o processo para reconhecer palavras se torna muito mais simples, no caso dos *HMMs*.

Uma situação diferente é observada no caso da *VQ* e do *DTW* onde a memória requerida é dependente do número de *templates* de cada palavra do vocabulário e do número de quadros e parâmetros acústicos utilizados. O número de dados que devem ser armazenados em memória seria $Q.V.T.p$ dados, onde $Q$ é o número de *templates* por palavra, $V$ é o número de palavras, $T$ é o número de quadros de cada palavra e $p$ é o número de parâmetros acústicos. Quando $Q=12$, $V=10$, $T=40$ e $p=10$, então será necessário armazenar 48000 dados. Esta vantagem evidente dos *HMMs* se torna maior quando o vocabulário cresce ou quando se utiliza linguagem contínua, onde os *HMMs* podem modelar fonemas ou grupos de fonemas em lugar de palavras.

As técnicas *DTW*, *VQ* e redes neurais se tornam inviáveis quando se trata de linguagem contínua, pois elas passam a consumir uma grande quantidade de recursos computacionais. Tais técnicas confrontam diversos problemas nos modelos de treinamento para fala contínua, já que os limites entre palavras não são detectáveis automaticamente e amiúde é necessária uma separação manual [MYE 81] [JUA 84]. No entanto, os *HMMs* permitem uma modelagem diferente e de menor custo computacional para tais problemas.

A palavra é a menor unidade de voz com significado semântico, sendo menos sensível às variações contextuais do que unidades menores tais como o fonema [MAR 96] [YOU 96]. Como conseqüência modelos de palavras, quando adequadamente treinados, freqüentemente levam aos melhores desempenhos de reconhecimento. Porém,

nem sempre os modelos de palavras podem ser adequadamente treinados, o que vai se tornar mais grave na medida que o tamanho do vocabulário aumenta.

Utilizar modelos de palavras em grandes vocabulários pode ser proibitivo, pois o tamanho da base de dados de treinamento é proporcional ao número de palavras do vocabulário. É preferível construir modelos de palavras a partir de modelos de unidades menores, previamente treinados. Os fones representam uma escolha natural para tais unidades. Em virtude da existência de aproximadamente 40 fones no idioma português, o número de palavras (ou sentenças) necessário para o treinamento adequado de modelos de fones é da ordem de centenas mesmo para vocabulários grandes. A desvantagem de utilizar modelos de fones, porém, é a grande variação da pronúncia de um dado fonema dependendo do contexto no qual esteja inserido [LEE 89].

Atualmente, uma das mais populares soluções em *RAV* é a utilização da *VQ* na codificação de segmentos de voz junto com *HMMs* na modelagem e classificação de tais segmentos. O primeiro passo, a *VQ*, divide o espaço do sinal em um número de células para produzir um dicionário de vetores. Cada vetor no dicionário corresponde a uma célula e representa todos os vetores em tal célula. O principal objetivo da *VQ* é a compressão de dados reduzindo os cálculos de processamento de sinais. As saídas da *VQ* representam os índices dos vetores do dicionário. O segundo passo, a modelagem *HMM*, produz um conjunto de modelos que representam as possíveis seqüências de vetores do dicionário que fazem parte das palavras que o sistema deve reconhecer [RAB 89].

A figura 4.2 mostra um sistema, que utiliza o mais comum algoritmo de *VQ*: o algoritmo *LBG* (*Linde-Buzo-Gray*). As vantagens deste algoritmo são sua simplicidade e o baixo processamento computacional. O algoritmo *LBG* não garante que o espaço de voz seja classificado de uma maneira globalmente otimizada. Isto quer dizer que alguns dos vetores do dicionário podem não representar as células dos vetores de entrada típicos. As limitações do algoritmo *LBG* conduzem a uma classificação imprecisa do espaço de voz e um casamento inadequado com a modelagem *HMM*. Conseqüentemente o sistema completo produz uma precisão de reconhecimento limitada.

O algoritmo *LBG* é também conhecido como o algoritmo de agrupamento e divisão porque cada iteração consiste em uma etapa de agrupamento e outra de divisão. Na etapa de divisão, cada vetor gerado no dicionário na iteração precedente é dividido em dois, gerando um novo dicionário que tem duas vezes mais vetores do que o dicionário anterior. Na etapa de agrupamento, os novos vetores do dicionário são usados para agrupar todos os pontos dos dados da entrada e formar novos espaços.

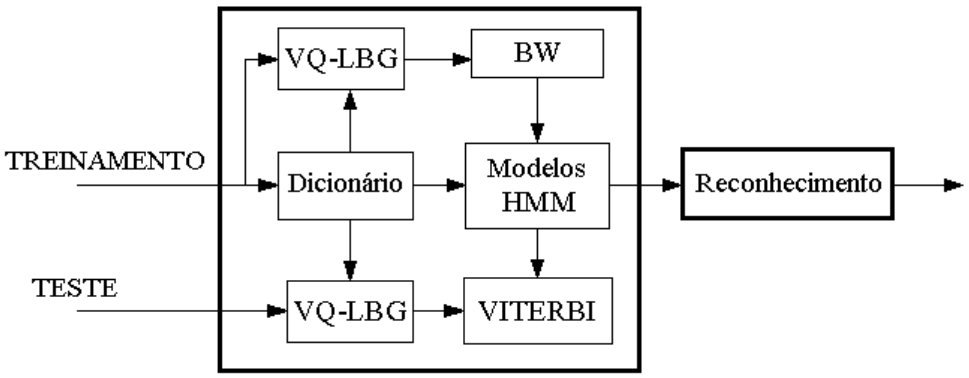



FIGURA 4.2 - Diagrama de blocos do sistema *LBG/HMM*.

A tabela 4.3 mostra a taxa de reconhecimento para a técnica *LBG/DHMM*. Os arquivos de voz utilizados foram gravados com ruído. Os resultados obtidos com o algoritmo *LBG/DHMM* são apresentados junto com os resultados do algoritmo *MSVQ* (*Multi-Section Vector Quantization*) em combinação com a regra do vizinho mais próximo (*Nearest Neighbor* ou *NN*). A taxa de reconhecimento do algoritmo *LBG/HMM* é menor do que a taxa obtida com *MSVQ/NN*. Uma implementação *MSVQ/DHMM* é uma alternativa que ainda não foi experimentada.

TABELA 4.3 - LBG/HMM vs. MSVQ/NN.

| **Algoritmo** | **LBG/DHMM** | **MSVQ/NN** |
|---|---|---|
| *Arquivos de Treinamento* | *740* | *740* |
| *Arquivos de Reconhecimento* | *370* | *370* |
| *Taxa de Reconhecimento* | *91,35%* | *95,95%* |

O *RAV* com *DHMMs* tem sido amplamente utilizado em diversas tarefas de reconhecimento, não apenas para palavras isoladas com vocabulário pequeno, mas também para reconhecimento de fala contínua tal como no caso do sistema *SPHINX* [LEE 89]. As principais vantagens dos *DHMMs* são a definição compacta e o baixo custo computacional.

Quando se utiliza um dicionário *VQ* para representar os vetores espectrais de voz, existe uma distorção inerente. A distorção diminui ao aumentar o nível da quantização, ou melhor, dito quando o tamanho do dicionário da *VQ* aumenta. Isto significa que quanto maior o tamanho do dicionário, maior deveria ser a taxa de reconhecimento. No entanto, em um sistema de *RAV/DHMM*, isto nem sempre acontece.

A figura 4.3 mostra o caso onde se tem um vocabulário pequeno de 10 comandos para a tarefa de *RAV* de comandos isolados. O dicionário da *VQ* com 64 e com 128 palavras produz os melhores resultados. Quando o tamanho do dicionário da *VQ* é aumentado para 256 ou mais, a distorção é menor do que nos casos anteriormente mencionados, mas a taxa de reconhecimento tende a piorar. Um dos motivos desta situação acontecer é que o espaço do sinal de voz consiste de grupos finitos representantes de classes acústicas, sendo que, cada classe revela algumas características estatísticas diferentes. Além disso, ao considerar as variações entre diferentes locutores e repetições do sinal de voz real, diversas palavras código poderão ser produzidas no conjunto ou na área onde há transposição de fonemas. Isto pode gerar

uma *VQ* ambígua e piorar o desempenho do reconhecimento. Outro motivo é que o algoritmo *LBG* agrupa o espaço do sinal de voz geometricamente, em vez de fazê-lo estatisticamente e assim tem aplicabilidade limitada nos *HMMs* os quais produzem uma modelagem baseada nas características estatísticas da seqüência de treinamento.

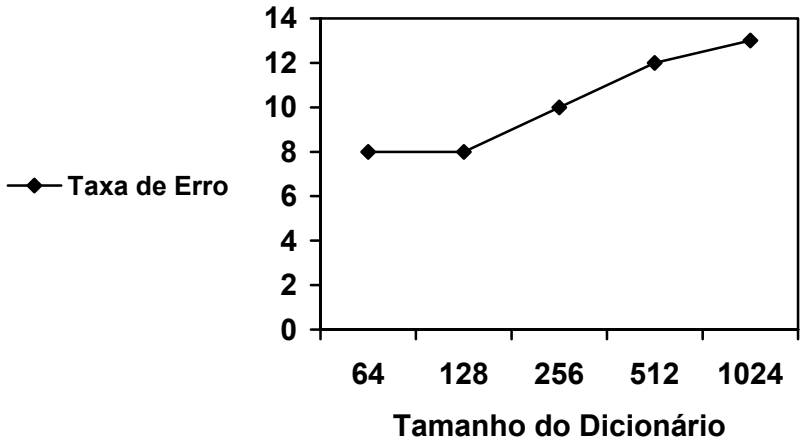



FIGURA 4.3 - Taxa de erro no reconhecimento vs. Tamanho do dicionário.

Uma *VQ* inadequada dos dados de treinamento leva à perda de informação estatística e à eventual diminuição da precisão do sistema total de reconhecimento. Para se adequar à modelagem *HMM* e reduzir o efeito do erro da *VQ*, são necessários um método de agrupamento estatístico na primeira etapa da modelagem.

As variações em torno da média ocorrem de maneira aleatória quando é considerada uma grande população de locutores, de maneira que os pontos dentro de um grupo devem ser distribuídos segundo uma função de densidade de probabilidade Gaussiana multidimensional. Esta visão do processo de produção de voz sugere que a classificação do espaço do sinal de voz é mais bem feita utilizando um modelo de mistura *Gaussiana* (***G**aussian **M**ixture **M**odel* ou *GMM*), nos pontos que pertencem a um determinado grupo ou "*cluster*" [TEB 95] [DEL 93] [CHO 96].

O algoritmo *EM* (***E**xpectation-**M**aximization*) pode ser utilizado como método alternativo para a *VQ* e o *GMM*, que é gerado como o dicionário da *VQ*. O *EM* é um algoritmo iterativo para a derivação da máxima verossimilhança (***M**aximum **L**ikelihood ou ML*) que estima os parâmetros de uma ampla variedade de modelos estatísticos e pode ser utilizado como substituto do algoritmo *LBG* para a quantização do espaço do sinal de voz [GOM 2000]. O algoritmo *EM* para *GMMs* é um método de quantizar o espaço do sinal de voz que reflete o processo de produção da voz de maneira mais próxima do que o algoritmo *LBG*. Porém, o custo computacional do algoritmo *EM* é maior do que o custo do algoritmo *LBG* [GOM 2000].

Outro método de classificação é o algoritmo de agrupamento por gaussianas múltiplas (***M**ultiple **G**aussian **C**lustering* ou *MGC*). Este algoritmo é similar ao algoritmo *EM*, porém com um custo computacional menor, produzindo uma classificação bastante precisa [CHO 96].

A tabela 4.4 mostra um experimento [ZHA 95] de *RAV* de palavras isoladas utilizando dígitos de zero ao nove. A tabela 4.4 mostra que entre os sistemas de *RAV* individuais, o *EM/HMM* mostra a melhor taxa de reconhecimento, e o *LBG/HMM* o pior.

TABELA 4.4 - Resultados individuais de *RAV* utilizando *VQ/HMM* [ZHA 95]

| **Tamanho do Dicionário** | | **128** | **256** | **512** |
|---|---|---|---|---|
| **Taxa de Erro** | *LBG/HMM* | *5,67* | *4,86* | *4,13* |
| | *EM/HMM* | *2,75* | *2,67* | *3,16* |
| | *MGC/HMM* | *4,62* | *4,53* | *5,10* |

Os três métodos de classificação dividem o espaço de dados de diferentes maneiras, e as divisões produzidas pelo algoritmo *EM* são as mais apropriadas.

Uma abordagem diferente para obter uma melhor taxa de reconhecimento com *VQ/HMMs* é a utilização de um sistema de *RAV HMM* múltiplo (*MHMM*) baseado nos três algoritmos acima mencionados. A idéia é que outros algoritmos podem corrigir os erros produzidos por cada algoritmo individualmente. Na etapa de treinamento, os três algoritmos de classificação mencionados são utilizados para uma *VQ* paralela e gera-se um dicionário para cada subsistema. São produzidos três modelos para cada palavra do vocabulário através do algoritmo *Baum-Welch*. Na etapa de reconhecimento, o algoritmo de *Viterbi* é utilizado com os três subsistemas em paralelo. Tal sistema requer um maior consumo computacional e tempo de reconhecimento do que aquele que é implementado em software pelos reconhecedores individuais. Porém, no caso de um sistema em hardware, os subsistemas poderiam ser implementados em paralelo para melhorar os resultados de reconhecimento sem incrementar o tempo de reconhecimento. Tal alternativa serve de motivação adicional para o projeto de um sistema *RAV* em hardware.

## 4.3 Comparação de Parâmetros Acústicos

A tabela 4.5 mostra uma comparação entre as taxas de reconhecimento obtidas, utilizando *HMMs* para palavras isoladas com diversos parâmetros acústicos, utilizando a base de dados mencionada neste capítulo.

TABELA 4.5 - Taxas de reconhecimento, comparando parâmetros acústicos.

| **Parâmetros** | **SD** | **SI** |
|---|---|---|
| LPC | 68,11% | 60,49% |
| Cepstrais | 87,30% | 80,32% |
| Cepstrais ponderados | 82,16% | 81,77% |
| MFCC | 85,68% | 83,07% |
| MFCC+Delta | 91,35% | 85,96% |
| MFCC+DeltaDelta | 86,22% | 80,17% |

Da tabela, é observável que os melhores resultados são obtidos para o reconhecimento utilizando parâmetros mel-cepstrais. Isto é amplamente aceito pela maioria dos projetistas de sistemas de *RAV* [MAR 96] [NUN 96] [BOU 98], e, por tal motivo são o conjunto de parâmetros mais utilizados.

## 4.4 Análise de Tempo e Complexidade do RAV/DHMM

A tabela 4.6 mostra o tempo de processamento envolvido nas diversas etapas de um sistema de *RAV* de palavras isoladas, com um vocabulário pequeno de 10 palavras baseado em *HMMs*.

TABELA 4.6 - Tempo de processamento das etapas do RAV/DHMM.

| Etapa | Tempo (ms) |
|---|---|
| *Pré-processamento* | 110 |
| *VQ* | 440 |
| *Viterbi* | 50 |
| *Treinamento* | 700 |

Como podem observar-se, a etapa que consume maior tempo de processamento é a etapa de treinamento seguida pela etapa de quantização vetorial. As etapas que consumem menor tempo são a etapa de pré-processamento (pré-ênfase, divisão em quadros, janelamento e extração de parâmetros) e a decodificação de estados com o algoritmo de *Viterbi*.

Também foi realizada uma análise de complexidade dos algoritmos de *RA*V *DHMM*. Os resultados são mostrados na tabela 4.7. *T* é o tamanho da seqüência de dados, *N* é o número de estados dos *HMMs* e *M* é o número de palavras do dicionário da *VQ*.

TABELA 4.7 - Complexidade dos algoritmos de RAV/DHMM.

| | Operações | Inicialização | Recursão | Finalização |
|---|---|---|---|---|
| ***Forward*** | Somas | | $N*(T-1)+N^2*(T-1)$ | $T$ |
| | Divisões | | $N*(T-1)$ | |
| | Multiplicações | | $N*(N+1)*(T-1)$ | |
| | Atribuições | $2$ | | |
| ***Backward*** | Somas | | $N^2*(T-1)$ | |
| | Divisões | $N$ | $N*(T-1)$ | |
| | Multiplicações | | $2*N^2*(T-1)$ | |
| ***Viterbi*** | Somas | | $N*(N+1)*(T-1)$ | |
| | Atribuições | $N$ | | $2*(N-1)$ |
| ***Reestimação de A*** | Somas | $[(T+1)*N+(T-1)]*N$ | | |
| | Multiplicações | $(T-1)*(3*N+2)*N$ | | |
| ***Reestimação de B*** | Somas | $T*N+T*M*N+2*M*N$ | | |

| | | |
|---|---|---|
| | Multiplicações | *2*T*M*N* |

## 4.5 Conclusão

Ao avaliar as técnicas de *RAV*, observa-se que para o problema de palavras isoladas abordado, o melhor desempenho é obtido pela *VQ*. No entanto, ao aumentar o tamanho do vocabulário, aumenta a exigência de recursos do sistema tais como número de parâmetros e a memória necessária para os padrões. Isto não ocorre ao trabalhar com *HMMs*.

Outra vantagem dos *HMMs* é que eles permitem o trabalho com conjuntos de subpalavras, o qual torna a taxa de reconhecimento dos *HMMs* superior ao das outras técnicas.

# 5 Arquitetura Especifica para RAV baseado em HMMs

## 5.1 Motivação

A maior parte dos sistemas de *RAV* existentes possui também limitações de desempenho que dificultam ou inclusive impedem o seu uso com grandes quantidades de dados em alta velocidade. Para a solução destas dificuldades, existem diversas abordagens.

Uma primeira abordagem se refere a aqueles sistemas que adotam algoritmos otimizados em termos de velocidade e que na sua implementação utilizam arquiteturas de hardware convencionais em processadores de propósito geral. Em alguns casos, os algoritmos de *RAV,* são normalmente modificados para trabalhar apropriadamente em máquinas de propósito geral sem a influência de implementações futuras de hardware de propósito específico.

Uma segunda abordagem envolve o uso de algoritmos otimizados, utilizando estruturas de hardware especializadas com um pipeline de instruções eficientes. É nesta segunda abordagem que as placas de processamento digital de sinais (*DSP*) de tarefas específicas, amplamente utilizadas na atualidade, se encontram.

Em certas aplicações é requerido um sistema flexível que ocupe pouco espaço físico, que trabalhe com grandes vocabulários, com linguagem contínua, e em tempo real, tornando maior o problema de processamento e armazenamento. Em tal caso surge uma terceira abordagem do problema apresentado anteriormente, que contempla o projeto e fabricação de circuitos integrados de aplicação específica, visando seu uso em algoritmos de tratamento e reconhecimento de voz.

O processamento de sinais e, em especial o *RAV* são áreas que possuem um grande potencial para novas aplicações de circuitos integrados. Sem o uso de circuitos integrados específicos para o *RAV*, o tamanho do vocabulário se torna limitado.

No capítulo anterior foi realizada uma análise de um modelo em software que servirá de base para a especificação de um sistema de *RAV* em hardware. Na continuação, serão consideradas algumas condições que serão levadas em conta na implementação do sistema de *RAV* em hardware.

## 5.2 Trabalho Proposto

A maioria dos sistemas de *RAV* existentes atualmente consiste em um software projetado para ser utilizado em computadores pessoais. O programa agregado opera continuamente dentro de um sistema operacional (*windows*, *OS/2*, etc.) e requer que o computador esteja equipado com uma placa de som compatível. A principal desvantagem de tais sistemas é a necessidade de um computador pessoal. Muitos dos sistemas existentes são bastante sofisticados [YOU 94]. No entanto, não é economicamente viável utilizar um sistema completamente computadorizado para controlar, por exemplo, uma máquina de lavar ou um aparelho de *TV*. Além disso, tendo

em vista que os programas de *RAV* exigem tempo de processamento de *CPU* do computador, a operação e o funcionamento do computador ficam geralmente afetados quando se habilita o *RAV* para executar certas tarefas a partir do computador.

A implementação de um sistema de *RAV* em hardware é uma tarefa mais difícil e demorada do que fazê-lo em software. Porém, o espaço físico ocupado por um sistema de *RAV* em hardware é menor do que um sistema de *RAV* em software.

Os sistemas de *RAV,* baseados em hardware, descritos no capítulo 3 tem sido implementados em uma placa com uma entrada analógica e um computador que comanda a comunicação entre os diversos subsistemas servindo de interface com o usuário.

Diversos algoritmos de *RAV* têm sido implementados em circuitos integrados por pesquisadores interessados no reconhecimento de um número de palavras maior daquele que é possível com processadores de propósito geral. Um sistema de *RAV* implementado com circuitos integrados de aplicação específica pode executar operações de maneira muito mais rápida do que aquele implementado com processadores de propósito geral, quando a parte operativa é especificada segundo os algoritmos de *RAV*. Ao utilizar máquinas de propósito geral, tópicos tais como largura de bits e paralelismo não podem ser explorados e por tal motivo eles são ignorados em favor de um aumento da complexidade algorítmica. Neste trabalho propõe-se uma alternativa prática de aplicação específica.

Ao examinar a tabela 3.1 e o estudo realizado no capítulo 3, observa-se que os trabalhos existentes que abrangem mais etapas do processo de *RAV* utilizam *DTW*. Tem havido pouca pesquisa em sistemas baseados em *HMMs*. Os sistemas existentes utilizando *HMMs*, que são o *GSM* [GLI 87], o *UCB-HMM* [STO 91], o *UPM-HMM* [FER 94] e o co-processador de pdf projetado por [PIH 96], implementam apenas uma pequena parte do processo de *RAV*. Tais sistemas fazem parte de uma arquitetura baseada em componentes discretos e pelo menos uma placa *DSP* para aquisição e extração de parâmetros característicos do sinal de voz. Além disso, existe pouca informação no referente às taxas de reconhecimento obtidas em relação a tais trabalhos.

Neste trabalho propõe-se a *definição de uma arquitetura para RAV baseada em HMMs* que possa ser utilizada na comunicação e controle de outros objetos e máquinas, tais como eletrodomésticos, robôs, instrumentos de teste, etc. Tal arquitetura propõe uma solução para os problemas mencionados anteriormente.

Nenhum dos trabalhos descritos no capítulo 3 utilizou lógica programável na implementação de algoritmos de *RAV*. Um sistema utilizando lógica programável tornaria possível a prototipação e teste de algoritmos de *RAV* de maneira mais rápida do que ao utilizar um *ASIC*.

Além de propor uma nova arquitetura, o objetivo deste trabalho é também sua especificação utilizando uma linguagem de descrição de hardware. Isto abre a possibilidade de sintetizar a arquitetura especificada em diversas plataformas, gerando circuitos integrados específicos.

Uma outra característica que guiará a proposta do sistema em menção é o potencial de ampliar o tamanho do vocabulário, das palavras e do número de locutores,

agregando mais módulos integrados, comunicáveis entre si. Além desta característica, se encontra o potencial de utilização tanto para vocabulário isolado como para linguagem contínua.

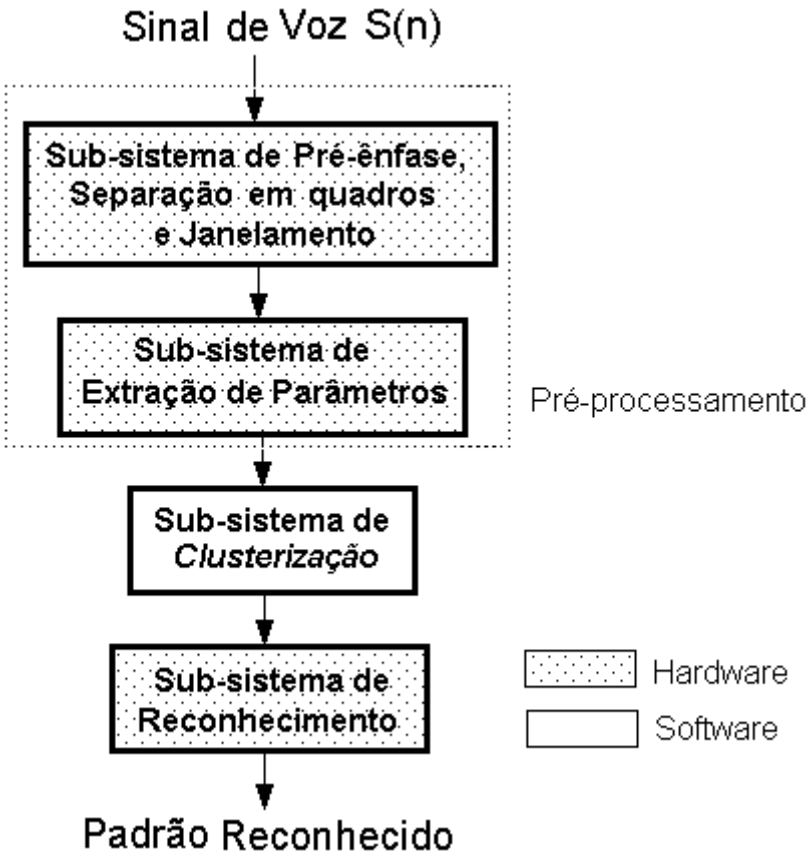



FIGURA 5.1 - Diagrama do Sistema de RAV proposto.

A figura 5.1 mostra o diagrama de blocos da proposta inicial do sistema. A tarefa de *RAV* é realizada por 4 subsistemas:

(a) O subsistema que realiza a pré-ênfase a separação em quadros e o janelamento, do sinal de voz. Este subsistema foi projetado utilizando *FPGAs*.

(b) O subsistema de extração de parâmetros, que tem como função a extração de parâmetros acústicos do sinal de voz. Este subsistema também foi projetado utilizando *FPGAs*.

(c) O subsistema de clusterização, que realiza a quantização vetorial dos parâmetros extraídos do sinal de voz. Este subsistema tem como função a separação do espaço acústico de vetores característicos para diminuir a quantidade de informação e definindo grupos diferenciáveis. Para tanto é efetuada a quantização vetorial dos parâmetros extraídos. Este subsistema não será projetado em *FPGAs*, mas apenas se utilizará a modelagem em software descrita no capitulo 3.

(d) O subsistema de reconhecimento, que realiza a identificação da palavra falada. Este subsistema tem como função o casamento dos padrões de teste

com os padrões armazenados, calculando a probabilidade de cada modelo ter gerado a seqüência de observações. Está formado basicamente por um decodificador de estados baseado no algoritmo de Viterbi para calcular a probabilidade de cada modelo armazenado ter gerado a seqüência de observações. Este subsistema foi projetado utilizando *FPGAs*.

Antes de descrever tais subsistemas, falaremos sobre algumas variáveis que serão de influência na implementação do hardware para *RAV*.

## 5.3 Tempo de Resposta

Hoje em dia, a maioria de aplicações práticas para *RAV* utiliza palavras isoladas, especialmente para a seleção de alternativas, ou para responder questões básicas. No caso de *RAV* para palavras isoladas, o tempo de resposta é muito importante. O tempo de resposta é o tempo para que uma palavra seja reconhecida imediatamente depois que ela foi falada por completo. O tempo de resposta também é crítico quando o sistema é utilizado com palavras conectadas. Em muitos casos essas palavras são faladas com mais de 250ms de silêncio entre elas.

Dependendo do tipo de aplicação, o tempo de resposta precisa ser tão curto quanto possível. A implementação do sistema proposto procurou otimizar os tempos de cada subsistema para assim produzir, um tempo mínimo total.

## 5.4 Tamanho dos Circuitos e Integração

O tamanho dos circuitos constitui um fator que afeta a implementação. Além da implicação na área utilizada, também existe o problema do número de pinos. Dependendo do tamanho do vocabulário e da quantidade de elementos que se pretende implementar, pode ser necessária a partição do circuito projetado em mais de um processador. Um trabalho importante, pouco abordado, mas nada trivial é a integração de partes operacionais e de controle, que implementam algoritmos de *RAV,* dentro de um sistema completo.

Um dos requisitos procurados é a portabilidade do sistema, portanto, o sistema proposto visa reduzir o espaço físico. Além disso, procura-se usar o menor número de células possíveis dos *FPGAs*.

## 5.5 Formato dos Dados

Uma questão importante se refere ao formato numérico dos dados utilizados. A maioria de processadores *DSP* utiliza o formato de dados ponto-fixo em lugar do formato ponto-flutuante que é muito comum em aplicações científicas. No formato de ponto-fixo, o ponto binário (análogo ao ponto decimal em base 10) é localizado em uma posição fixa dentro do número. No entanto, no formato de ponto-flutuante os números são expressos utilizando um expoente e uma mantissa e o ponto binário flutua baseado no valor do expoente. O formato de ponto-flutuante permite representar um grande intervalo de valores, eliminando praticamente o perigo de *overflow* na maioria das aplicações. Porém, o formato de ponto-fixo produz processadores mais baratos e com

um consumo de energia menor do que o formato de ponto flutuante em velocidades comparáveis, pois o formato em ponto-flutuante requer a implementação de um hardware mais complexo. Por tal motivo, existem poucos processadores de ponto-flutuante.

O custo e o consumo de energia também influem na largura dos dados utilizados. Os processadores *DSP* geralmente utilizam a menor largura de dados necessária para uma precisão adequada na aplicação final. A maioria dos processadores *DSP* de ponto-fixo utiliza palavras de 16-bits, sendo esta uma largura suficiente para diversas aplicações *DSP*. Outros processadores *DSP* utilizam larguras de 20, 24, ou ainda de 32 bits para obter uma melhor precisão em aplicações que são difíceis de implementar bem como dados de 16 bits, tais como o processamento de sinais de áudio de alta fidelidade.

Com o objetivo de obter uma adequada qualidade do sinal enquanto se utilizam dados de ponto-fixo, os processadores *DSP* incluem um hardware especializado para permitir com que o programador mantenha fidelidade numérica através de uma série de cálculos. Por exemplo, a maioria dos processadores *DSP* inclui tipicamente um ou mais registradores acumuladores para manter os resultados ao somar vários produtos da multiplicação. Os registradores acumuladores tipicamente são mais largos que outros registradores, fornecendo amiúde alguns bits adicionais conhecidos como bits de guarda, para estender o intervalo de valores que pode ser representado e assim evitar o *overflow*. Além disso, os processadores *DSP* usualmente incluem bom suporte para saturação aritmética, arredondamento e deslocamento, todos os quais são úteis para manter fidelidade numérica.

Ao multiplicar números de ponto-fixo, deve ser levado em consideração o tamanho dos operandos. Em geral, ao multiplicar dois números de *n-bits*, serão necessários *2.n-bits* para representar o produto sem introduzir algum erro.

O processamento realizado em cada etapa do *RAV*, normalmente utiliza representações em ponto flutuante, pois se trabalha com valores reais e, no caso da etapa de modelagem estatística, são utilizadas matrizes estocásticas onde os dados manipulados são inferiores a 1. Porém, a representação em ponto flutuante produziria blocos operacionais complexos e memórias muito grandes. Para evitar este aumento de complexidade, no sistema de *RAV* proposto, optou-se por uma representação em ponto fixo para todos os subsistemas.

## 5.6 Taxa de Erro

A obtenção de uma baixa taxa de erro no reconhecimento é um requisito desejável em todo sistema de *RAV*. Uma vez que a otimização do modelo comportamental é realizada, a obtenção de uma baixa taxa de erro requer também uma otimização no nível de arquitetura. É previsível uma diferença entre as taxas que o modelo comportamental produz e as taxas que a implementação em hardware de tal modelo produz, causado pela perda de precisão de um para o outro. A voz existe no mundo analógico na forma de ondas de som e deve ser transformada em um sinal analógico elétrico, antes do processo de reconhecimento começar. Esses componentes analógicos do sistema são uma poderosa fonte de erro que deve ser considerada.

# 6 Subsistema de Pré-ênfase, Separação em quadros e Janelamento

Este subsistema é o encarregado de processar a voz de entrada, fazer a pré-ênfase, a separação em quadros e o janelamento do sinal digital de voz. A entrada deste subsistema é constituída pelos valores do sinal de voz que foram passados por um filtro anti-aliasing e discretizados através de um conversor *A/D* externo ao sistema. O sinal de voz foi gravado anteriormente no formato *RAW*, 16 bits, a uma taxa de amostragem de 11025Hz. A saída deste subsistema é formada pelos parâmetros extraídos.

A figura 6.1 mostra o subsistema proposto. Este subsistema possui um bloco funcional que realiza a pré-ênfase, o segundo bloco funcional realiza a separação em quadros e o terceiro bloco funcional faz o janelamento do sinal de voz.

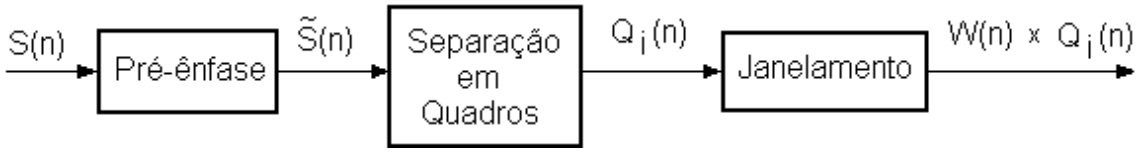



FIGURA 6.1 - Subsistema de pré-ênfase, separação em quadros e janelamento.

## 6.1 Função de Pré-ênfase

Ao implementar o filtro de pré-ênfase no sinal, são propostas algumas modificações à definição fornecida na seção 1.3.1, para facilitar a implementação em ponto-fixo. O coeficiente de pré-ênfase, $a_{pre}$, é aproximado pelo valor de 15/16 em função da simplificação que significa realizar uma divisão por um número do tipo $2^n$, sendo n inteiro. A saída $\tilde{S}(n) = S(n) - a_{pre}.S(n-1)$ será então:

$$\tilde{S}(n) = S(n) - \frac{15}{16}.S(n-1) = S(n) - S(n-1) + \frac{S(n-1)}{16} \qquad (6.1)$$

A figura 6.2 mostra uma arquitetura implementada para a parte operativa do filtro de pré-ênfase. A implementação desta arquitetura utiliza um único somador/subtrator de 20 bits para as operações do filtro de pré-ênfase, em conjunto com 3 multiplexadores *Mx1*, *Mx2* e *MxCl*. Em um primeiro instante é selecionada a operação de subtração entre *S(n)* e *S(n-1)*. Depois de estes dados serem selecionados pelos multiplexadores *Mx1* e *Mx2*, aparecem na entrada do somador/subtrator. No passo seguinte, realiza-se a soma dos valores no acumulador e o resultado de *S(n-1)/16*, que são colocados na entrada do somador/subtrator, pelos multiplexadores *mux1* e *mux2*.

O objetivo desta implementação foi utilizar um número reduzido de células lógicas, apesar de custar um tempo de latência maior. No entanto, o resultado não foi satisfatório e optou-se por uma alternativa mais rápida e que, na síntese total, utilizou menos células lógicas.

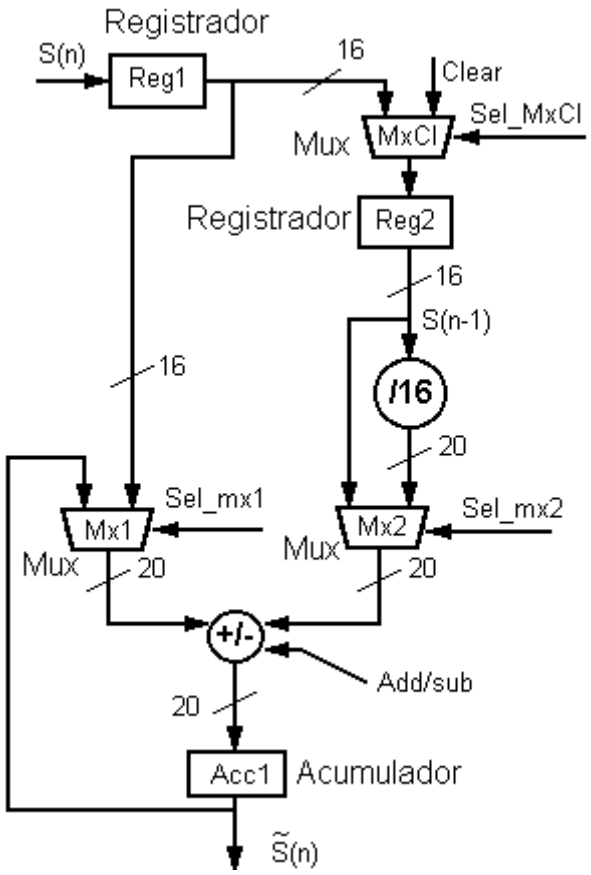



FIGURA 6.2 - Parte operativa da função de pré-ênfase, utilizando um único somador/substrator.

A figura 6.3 mostra uma arquitetura simplificada para a realização da pré-ênfase. Esta arquitetura consiste de um circuito de divisão, um somador, um subtrator, um acumulador, e dois registradores de deslocamento. A pré-ênfase é realizada em um único pulso de relógio, sendo também necessários dois pulsos adicionais na inicialização dos registradores. É uma alternativa mais simples do que a anterior e consume um tempo de latência menor. Na etapa de inicialização, a primeira amostra do sinal de voz digitalizado, *S(1)*, é armazenada no registrador *Reg1*. Na etapa seguinte, é lido um novo dado, a amostra *S(2)*. Este novo dado é armazenado em *Reg1* e o valor anterior de *Reg1* é transferido para *Reg2*. As outras operações são realizadas em paralelo e o resultado final é armazenado no acumulador *Acc1*. Este procedimento é repetido até processar o total das amostras do sinal de voz.

Em ambas as arquiteturas das figuras 6.2 e 6.3 utilizaram-se externamente dados de 16 bits. No entanto, internamente os dados foram convertidos para 20 bits, pois isto permite diminuir a perda de precisão do sistema.

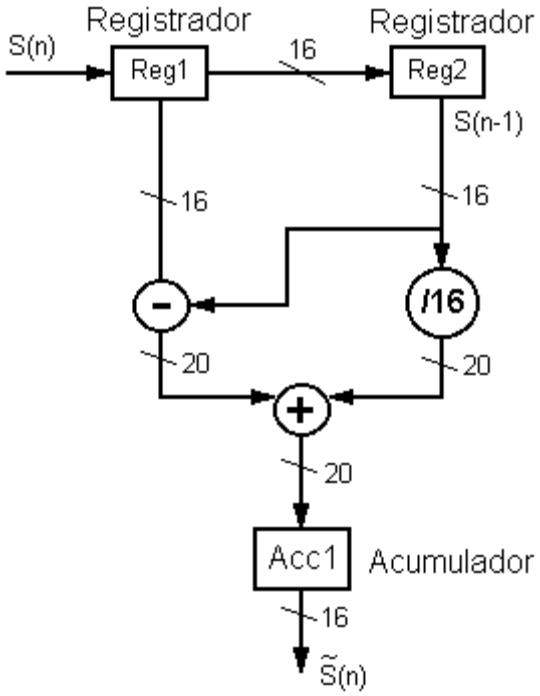



FIGURA 6.3 - Parte operativa da função de pré-ênfase, utilizando um somador e um substrator.

A divisão por 16 é obtida fazendo um deslocamento de quatro bits para a direita, introduzindo quatro zeros nos bits mais significativos, à esquerda (ver fig. 6.4). Tal solução simples permite realizar a divisão em um único pulso de relógio. A palavra original é de 16 bits, e optou-se por calcular uma palavra final de 20 bits, para não causar uma modificação significativa da precisão.

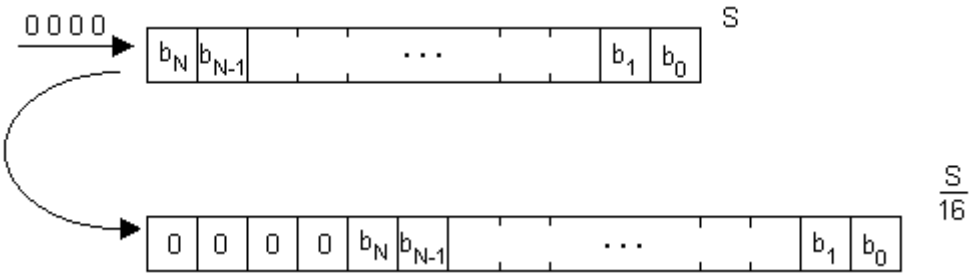
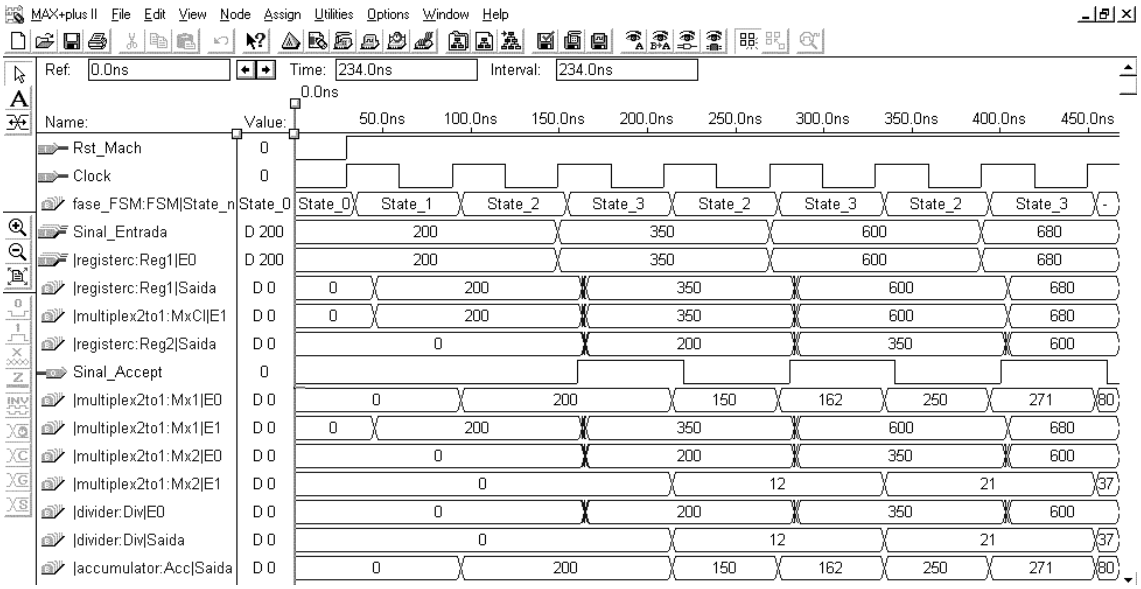
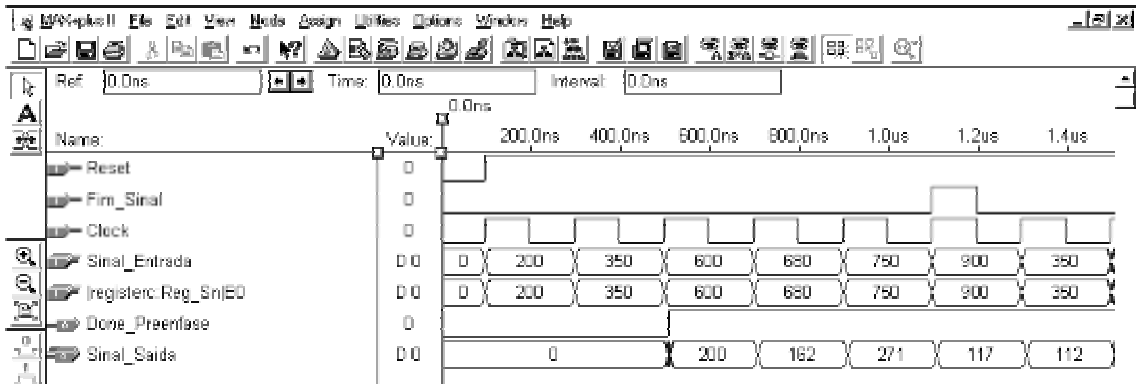



FIGURA 6.4 - Deslocamento à direita para realizar a divisão por 16.

A figura 6.5 mostra as simulações das funções de pré-ênfase da figura 6.2 e 6.3. A entrada da função de pré-ênfase é o sinal *Sinal_Entrada*.




(a) Circuito com um único somador.




(b) Circuito com um somador e um subtrator.

FIGURA 6.5 - Simulação da pré-ênfase.

A tabela 6.1 mostra os resultados de síntese dos circuitos de pré-ênfase, implementados. Utilizar um somador e um subtrator separados é uma alternativa mais simples, pois apesar de incluir mais um circuito somador, evita-se utilizar multiplexadores obtendo, no final, uma redução de células lógicas. Além disso, o tempo de latência é reduzido pela metade, pois todo o processo é realizado em um único pulso de relógio.

TABELA 6.1 - Comparação entre circuitos de pré-ênfase.

| Circuito | Dispositivo | Células | % Ocupação | Tempo de latência |
|---|---|---|---|---|
| Com um único somador-subtrator | EPF10K10LC84-3 | 100 | 17 | 2 pulsos/dado |
| Com somador e subtrator separados | EPF10K10LC84-3 | 91 | 15 | 1 pulso/dado |

Os resultados foram comparados com o modelo em software, produzindo um erro máximo inferior a 1%.

## 6.2 Função de Separação em Quadros

Após a pré-ênfase, o sinal deve ser separado em *quadros* $Q_i$*=0,..., T-1*, onde *T* é o número de quadros em que o sinal de voz pode ser dividido. Utilizou-se uma freqüência de amostragem de 11025Hz e cada quadro foi gerado a partir de 252 amostras. O número 252 foi escolhido, pois além de produzir um quadro com duração dentro do intervalo especificado por [RAB 93] que garanta que a parte analisada do sinal permaneça estacionário. Também, 252 é um número divisível por 3, o qual facilitará a implementação em hardware da superposição entre quadros. Além disso, o número 252 também é divisível por 9, o qual permitirá a implementação do algoritmo, que será explicado posteriormente, que divide um quadro em três partes iguais.

Devido ao tamanho do quadro e à existência de superposição entre quadros, cada amostra corresponde ao primeiro terço do atual quadro de análise, ou ao segundo terço do quadro anterior ou ao terço final do penúltimo quadro. Depois de cada 84 amostras, valor que corresponde a um terço do total de amostras do quadro, um dos três quadros é completado. Quando isto ocorre, a relação entre os quadros de análise é mudada ciclicamente. A figura 6.6 mostra a superposição existente entre os quadros gerados a partir do sinal de voz.

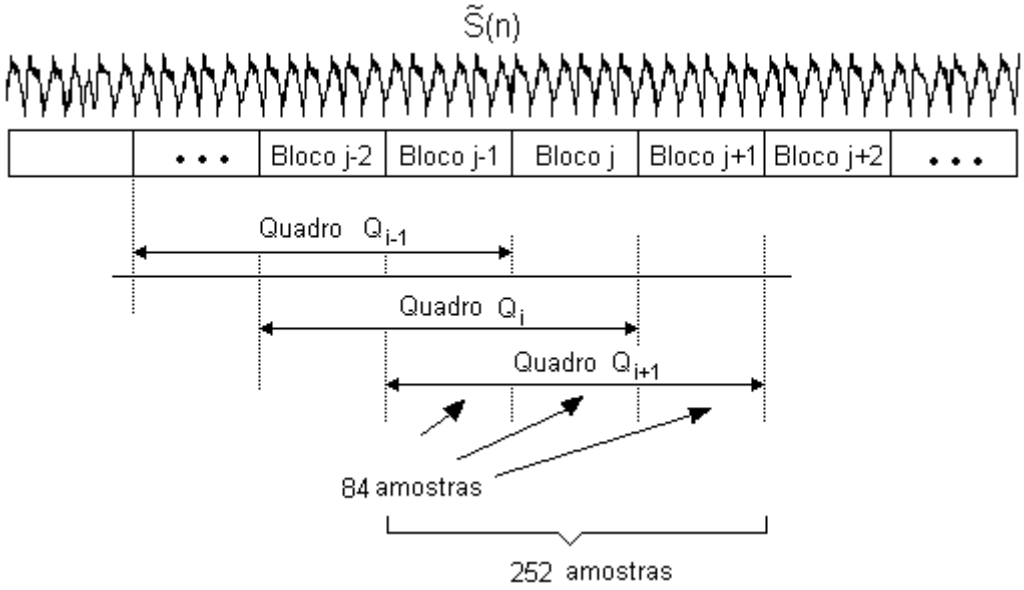



FIGURA 6.6 - Superposição entre quadros.

**Algoritmo de Separação em Quadros**

Foi desenvolvido um algoritmo para a separação em quadros que permita sua implementação otimizada em hardware. Cada quadro, $Q_i$, deve ser multiplicado pelos valores correspondentes da janela de *Hamming*, processo conhecido como janelamento. Porém, dentro do algoritmo que será descrito na continuação, o janelamento é realizado enquanto a separação em quadros é realizada.

Para formar os quadros, as amostras depois de serem processadas pela função de pré-ênfase são inicialmente segmentadas em blocos, onde cada bloco tem 84 amostras, não existindo superposição entre blocos. Cada quadro, $Q_i$, será composto por três blocos sucessivos, totalizando 252 amostras. A partir destas definições, pode-se deduzir que o número de blocos que podem ser formados a partir do sinal de voz será $T+2$, onde $T$ é o número de quadros.

A figura 6.7 mostra a segmentação em blocos $B_i$, $i=0...T+1$, do sinal de voz. As amostras de cada quadro, composto por três blocos consecutivos, devem ser multiplicadas pelos valores correspondentes da janela de *Hamming*, $W(n)$, $n=0...251$.

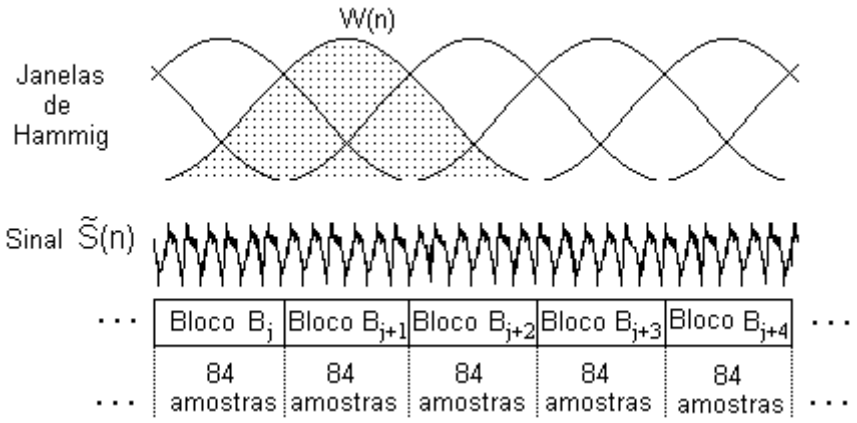



FIGURA 6.7 - Segmentação em blocos do sinal de voz.

Para a implementação em hardware do algoritmo de divisão em quadros, utiliza-se uma memória *RAM* de dupla porta, externa, que permite a leitura e a escrita de forma paralela. Esta memória é dividida em três segmentos *M(0)*, *M(1)* e *M(2)*, de 84 amostras cada um. Em cada segmento de memória serão armazenadas as amostras dos blocos que formarão um quadro. O resultado da multiplicação de cada amostra do quadro final pelo valor correspondente da janela de *Hamming* é armazenado em um registrador.

Os quadros são formados pelo agrupamento dos segmentos da memória. Para permitir o trabalho em paralelo das operações de leitura e escrita na memória, a seqüência em que os segmentos de memória são agrupados ocorre ciclicamente, como mostrado na figura 6.8.

As amostras do primeiro bloco (o bloco $B_0$) não precisam ser armazenadas na memória, pois serão processadas diretamente, multiplicando-as pela porção da janela de *Hamming* correspondente. As amostras do segundo bloco (o bloco $B_1$) são armazenadas no segmento de memória *M(0)* e as amostras do terceiro bloco (o bloco $B_2$) são armazenadas no segmento de memória *M(1)*. Assim, o primeiro quadro $Q_0$ será formado pelas amostras do primeiro bloco (que não precisou ser armazenado) e as amostras armazenadas nos segmentos de memória *M(0)* e *M(1)*.

Em um tempo posterior, as amostras do quarto bloco (o bloco $B_3$) são armazenadas no segmento de memória *M(2)*. Neste momento não é necessário alterar o conteúdo nos segmentos *M(0)* e *M(1)*. Assim, teremos o segundo quadro, $Q_1$, que é formado pelas amostras armazenadas nos segmentos de memória *M(0)*, *M(1)* e *M(2)*.

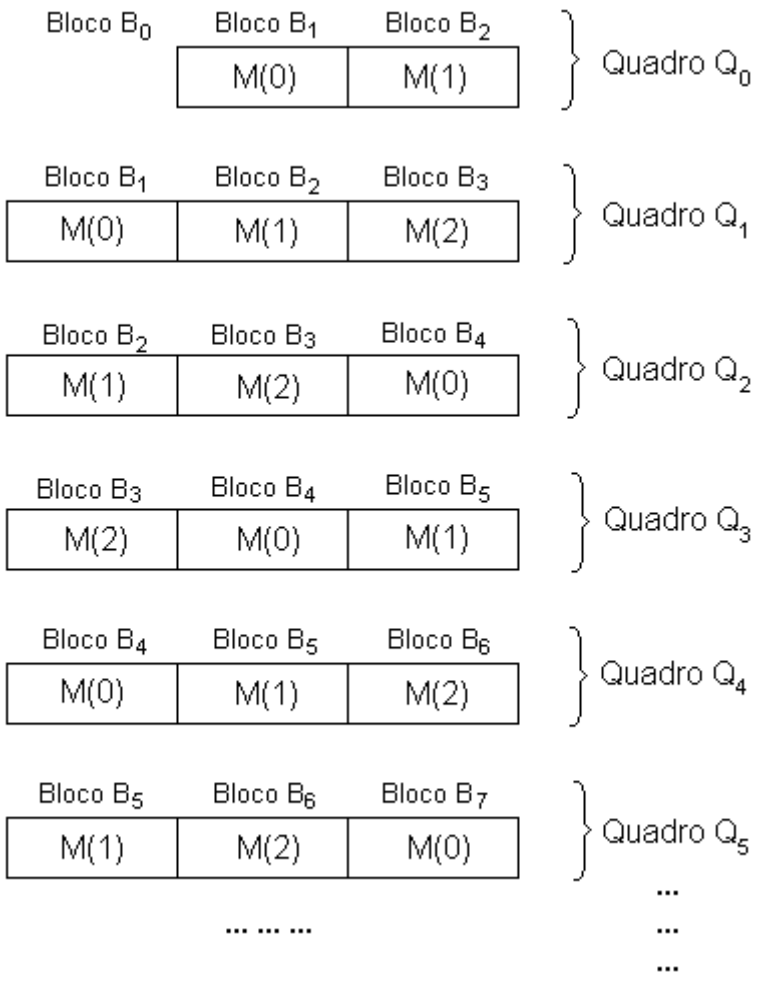



FIGURA 6.8 - Seqüência de blocos e sua armazenagem nos segmentos da memória.

Continuando com o processamento, as amostras do quinto bloco (o bloco $B_4$) são armazenadas no segmento de memória *M(0)*. O conteúdo anterior deste segmento (o bloco $B_1$), não será mais necessário para o processamento posterior. Neste momento não é necessário alterar os segmentos *M(1)* e *M(2)*, que contém os blocos $B_2$ e $B_3$. Assim, teremos o terceiro quadro, $Q_2$, o qual é formado pelas amostras armazenadas nos segmentos de memória *M(1)*, *M(2)* e *M(0)*, gerando uma nova seqüência de segmentos.

Na continuação, as amostras do sexto bloco (o bloco $B_5$) são armazenadas no segmento *M(2)*. O conteúdo anterior deste segmento (o bloco $B_2$), não será mais utilizado no processamento posterior. No entanto, o conteúdo dos segmentos *M(1)* e *M(0)* não é alterado (blocos $B_3$ e $B_4$), neste momento. Assim, teremos o quarto quadro (o quadro $Q_3$), formado pelas amostras armazenadas nos segmentos de memória *M(2)*, *M(0)* e *M(1)*.

O processamento dos demais blocos é realizado da maneira mostrada na figura 6.8, até processar o total de blocos que conformam o sinal de voz.

Uma vez realizada a separação em quadros, será necessária a multiplicação de cada quadro $Q_i$ formado pelos blocos $B_j$, $B_{j+1}$ e $B_{j+2}$, pela janela de *Hamming* (ver figura 6.9). No entanto, o algoritmo desenvolvido faz a multiplicação de cada bloco pela porção da janela de *Hamming* correspondente, assim como mostrado nas figuras 6.10

(a), (b) e (c). Este processo é realizado pelo algoritmo de separação em quadros, no mesmo momento em que cada quadro vai sendo formado.

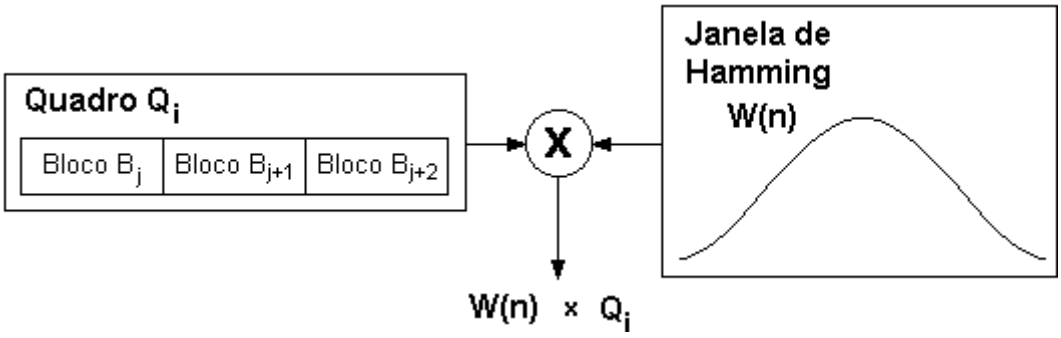
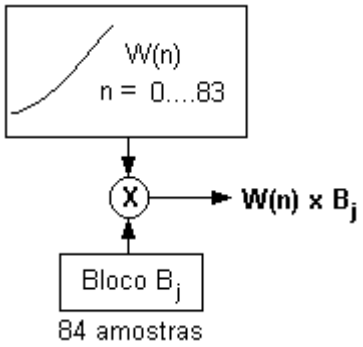
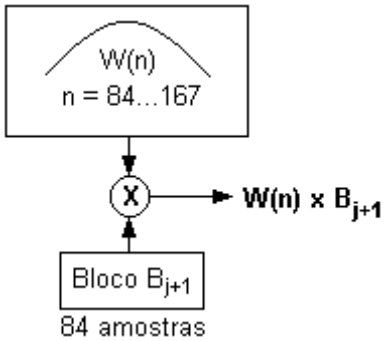



FIGURA 6.9 - Operação de janelamento.

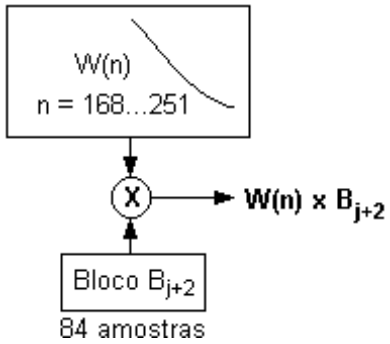
A escrita de amostras nos segmentos da memória é realizada paralelamente com a operação de leitura dos segmentos. Isto significa que, enquanto um segmento da memória está sendo escrito, outro segmento da memória está sendo lido e multiplicado pela janela de *Hamming*. Para tanto, foi necessária uma velocidade da operação de leitura da memória igual ao triplo da velocidade da operação de escrita. Isto significa que, enquanto são lidas 84 amostras que estão armazenadas em um segmento da memória, são recebidas do sinal de voz e armazenadas em um outro segmento da memória, apenas 28 amostras.




(a) Janelamento do primeiro bloco.




(b) Janelamento do segundo bloco.




(c) Janelamento do terceiro bloco.

FIGURA 6.10 - Multiplicação dos blocos pela janela de Hamming.

Para ilustrar o funcionamento deste algoritmo, consideremos o quadro $Q_i$ formado pelos blocos $B_j$, $B_{j+1}$ e $B_{j+2}$ armazenados nos segmentos de memória *M(0)*, *M(1)* e *M(2)* respectivamente. O quadro anterior (quadro $Q_{i-1}$) foi formado pelos blocos $B_{j-1}$, $B_j$ e $B_{j+1}$, que ocupavam os segmentos de memória *M(2)*, *M(0)* e *M(1)* respectivamente, assim como mostrado na figura 6.11.

Quando o quadro $Q_{i-1}$ estava sendo processado, as amostras do bloco $B_{j-1}$ estavam armazenadas no segmento de memória *M(2)*. O último bloco, do quadro $Q_{i-1}$, que foi processado, isto é suas amostras lidas da memória e multiplicadas pela janela de Hamming, foi o bloco $B_{j+1}$. Este bloco se encontrava no segmento de memória *M(1)*. Uma vez que o último bloco do quadro $Q_{i-1}$ está sendo processado, foi observado que o próximo quadro a ser processado (quadro $Q_i$) não precisará mais as amostras do bloco $B_{j-1}$ que se encontravam no segmento *M(2)*. Portanto, utilizou-se o segmento *M(2)* para armazenar os valores do próximo bloco (bloco $B_{j+2}$). Para tanto o segmento *M(2)* foi dividido em três partes iguais. O bloco $B_{j+2}$ foi armazenado em *M(2)* em três etapas.

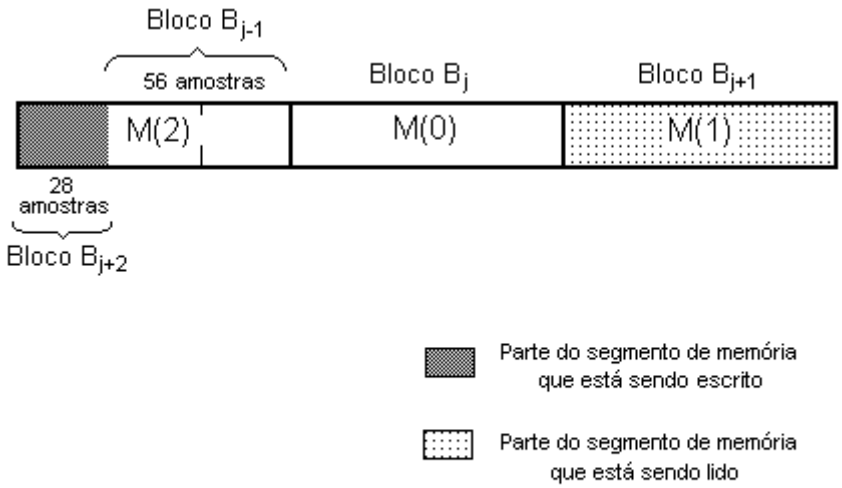



FIGURA 6.11 - Operações no quadro $Q_{i-1}$, ao completar o processamento do último bloco.

A primeira etapa, mostrada na figura 6.11, ocorre enquanto o bloco $B_{j+1}$ está sendo processado dentro do quadro $Q_{i-1}$. É realizada uma operação de escrita no segmento *M(2)* em paralelo à operação de leitura do segmento *M(1)*. São escritas as primeiras 28 amostras do bloco $B_{j+2}$. Esta operação de escrita afeta apenas a primeira parte do segmento de memória *M(2)*, ou seja, as primeiras 28 amostras do segmento. Tudo isto ocorre terminando o processamento do quadro $Q_{i-1}$.

A segunda etapa é mostrada na figura 6.12. Nesta etapa começa o processamento de um novo quadro, o quadro $Q_i$. O primeiro bloco do quadro $Q_i$ (bloco $B_j$) é processado, isto quer dizer que os valores armazenados no segmento de memória *M(0)* são lidos e multiplicados pela porção da janela de *Hamming* correspondente. Em paralelo à operação de leitura do segmento *M(0)*, escrevem-se outras 28 amostras do bloco $B_{j+2}$ no segmento *M(2)*. Esta operação de escrita afeta unicamente a segunda parte do segmento *M(2)*, formada por 28 amostras. Até este momento 56 amostras do bloco $B_{j+2}$ foram escritas no segmento *M(2)*.

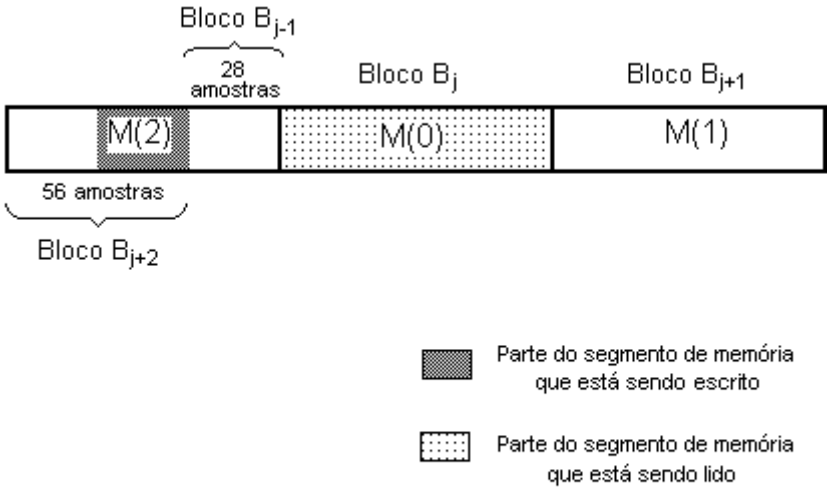



FIGURA 6.12 - Operações no quadro $Q_i$, ao processar o primeiro bloco do quadro.

A última etapa, mostrada na figura 6.13, continua com o processamento do quadro $Q_i$. O segundo bloco do quadro $Q_i$ (bloco $B_{j+1}$) é agora processado, isto quer dizer que os valores armazenados no segmento de memória *M(1)* são lidos e multiplicados pela porção da janela de *Hamming* correspondente. Em paralelo à operação de leitura do segmento *M(1)*, escrevem-se as 28 amostras finais do bloco $B_{j+2}$ no segmento *M(2)*. Esta operação de escrita afeta a terceira parte de 28 amostras do segmento de memória *M(2)*. Neste momento todas as 84 amostras do bloco $B_{j+2}$ são escritas no segmento *M(2)*.

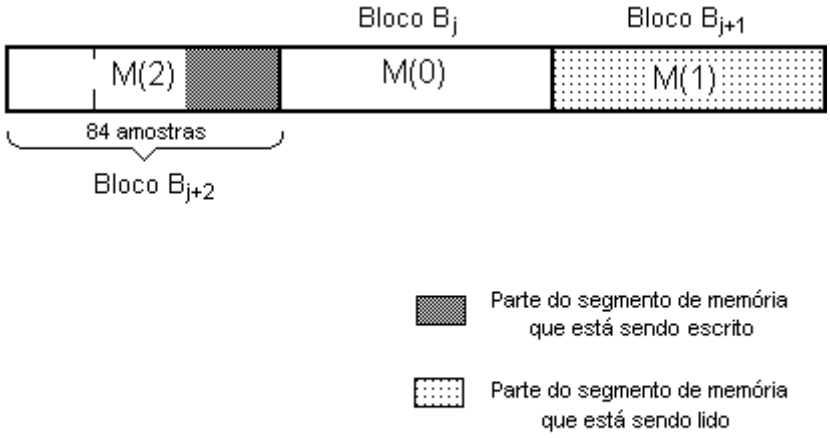



FIGURA 6.13 - Operações no quadro $Q_i$, ao processar o segundo bloco do quadro.

Uma vez completada a escrita do bloco $B_{j+2}$ no segmento *M(2)*, é realizada a escrita do bloco $B_{j+3}$, também em três etapas da mesma maneira em que foi armazenado o bloco $B_{j+2}$.

Para concluir o processamento do quadro $Q_i$, realiza-se o processamento do bloco $B_{j+2}$, que está armazenado no segmento *M(2)*. Assim como mostrado na figura 6.14, as primeiras 28 amostras do bloco $B_{j+3}$ são escritas no segmento *M(0)*, em paralelo à operação de leitura do segmento *M(2)*. Esta operação de escrita afeta a primeira parte de 28 amostras do segmento de memória *M(0)*. Isto dá início ao armazenamento das amostras do bloco $B_{j+3}$, que será utilizado para formar o quadro $Q_{i+1}$. As três etapas explicadas anteriormente são agora realizadas para armazenar o bloco $B_{j+3}$.

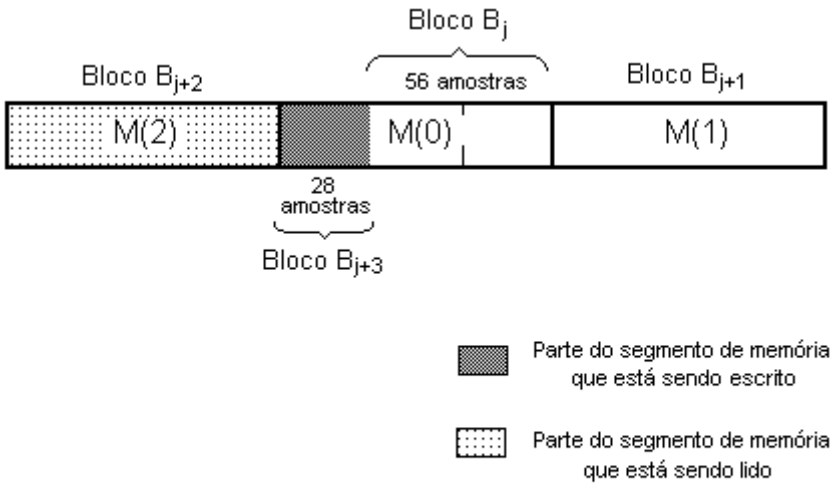



FIGURA 6.14 - Operações no quadro $Q_i$, ao processar o último bloco do quadro.

A figura 6.15 mostra o diagrama de fluxo do algoritmo proposto para a separação em quadros. Inicialmente o primeiro bloco $B_0$ de 84 amostras é multiplicado pelos primeiros 84 valores da janela de *Hamming*. O mesmo procedimento é seguido pelo segundo e terceiro blocos $B_1$ e $B_2$, que conformam o primeiro quadro. Além disso, as amostras do segundo e o terceiro bloco são armazenadas nos segmentos *M(1)* e *M(2)*, para ser utilizadas posteriormente, na formação do segundo quadro. O bloco $B_0$ não precisa ser armazenado em nenhum segmento de memória, pois não será utilizado nos quadros seguintes.

A partir do quarto bloco, $B_3$, cada bloco é processado em 3 etapas. Inicialmente o primeiro terço do bloco $B_3$, formado por 28 amostras, é armazenado no primeiro terço do segmento de memória *M(0)*. Na etapa seguinte, é lido o segmento *M(1)*, que contêm os valores do bloco $B_1$, multiplicando estes valores pela janela de *Hamming* e o resultado é armazenado no registrador de saída. Além disso, o segundo terço de 28 amostras do bloco $B_3$ é armazenado no segundo terço do segmento de memória *M(0)*. Na etapa seguinte, o segmento de memória *M(2)*, que contêm os valores do bloco $B_2$, é multiplicado pela janela de *Hamming* e o resultado é armazenado no registrador de saída. Nesta mesma etapa, o último terço de 28 amostras do bloco $B_3$ é armazenado no último terço do segmento de memória *M(0)*. Na próxima etapa, é lido o segmento de memória *M(0)* que agora contêm os valores do bloco $B_3$, sendo multiplicado pela janela de *Hamming* e o resultado é armazenado no registrador de saída. Além disso, nesta etapa, o primeiro terço de 28 amostras do bloco $B_4$ é armazenado no primeiro terço do segmento *M(1)*. O processo é repetido com os blocos seguintes e os segmentos de

memória *M(0)*, *M(1)* e *M(2)* são preenchidos da maneira em que foi mostrada na figura 6.7.

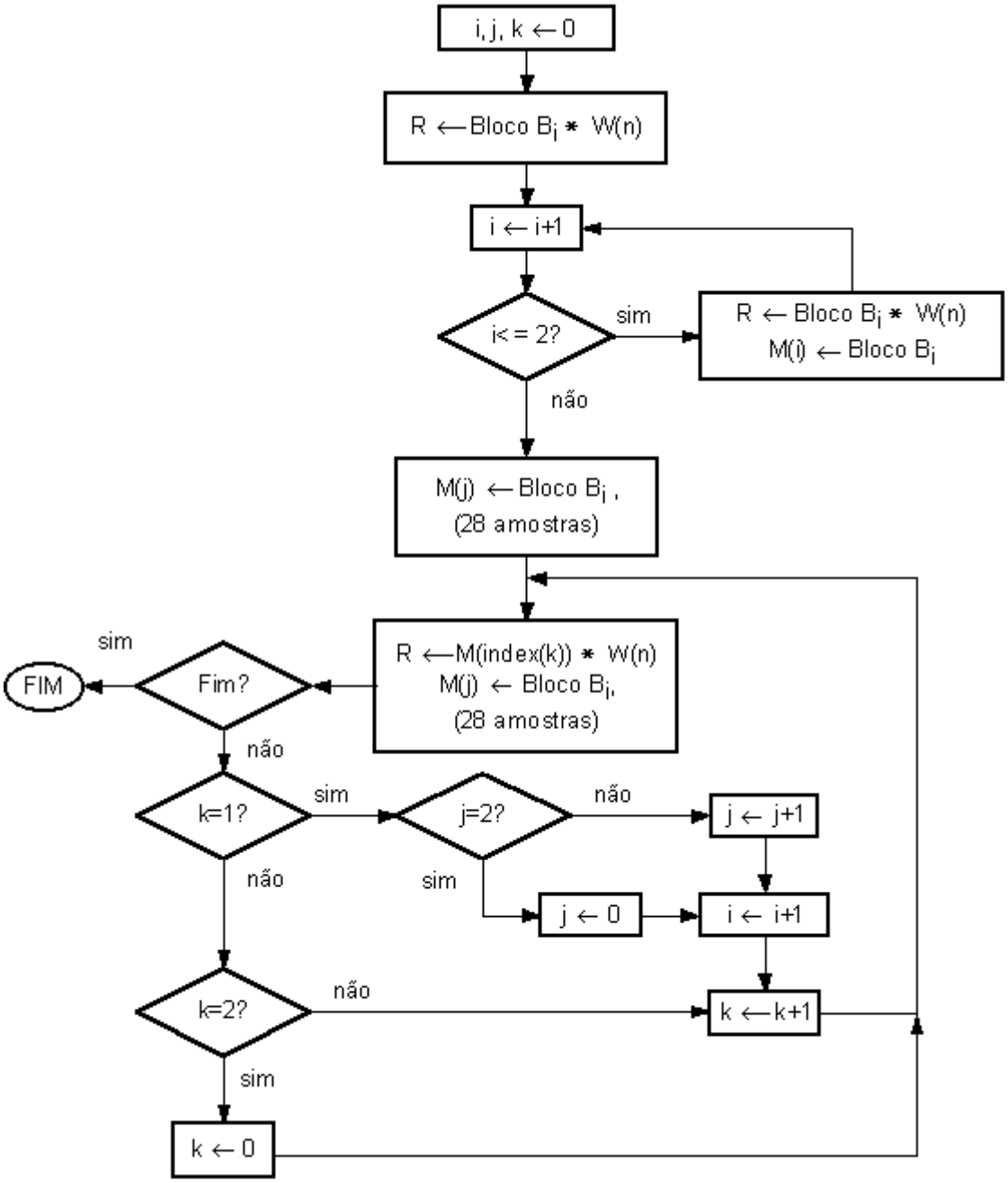



FIGURA 6.15 - Algoritmo de separação em quadros do sinal de voz.

## 6.3 Função de Janelamento

Para minimizar as descontinuidades no início e no final de cada quadro, aplica-se uma janela de *Hamming* em cada quadro. A janela de Hamming, *W(n)*, tem equação:

$$W(n) = 0{,}54 - 0{,}46 \cdot \cos\left(\frac{2\pi \cdot n}{Ns - 1}\right), 0 \le n \le Ns - 1 \qquad (6.2)$$

onde *Ns* é o número de amostras de cada janela e *n* é a amostra que está sendo avaliada.

O tamanho escolhido para cada janela é o mesmo que o do tamanho do quadro, isto significa 252 amostras, ou aproximadamente 23ms do sinal. O deslocamento da janela é de 84 amostras. O tamanho de 252 amostras facilita a implementação em ponto-fixo da janela de *Hamming*.

A implementação da janela de *Hamming* foi realizada com uma memória *ROM* que armazena os primeiros 126 valores da janela *W(n)* (*n=0...125*). A partir destes valores se obtêm os 126 valores restantes de *W(n)* (*n= 126...251*), utilizando a propriedade de simetria da janela. A figura 6.16 mostra a simulação da janela de *Hamming*.

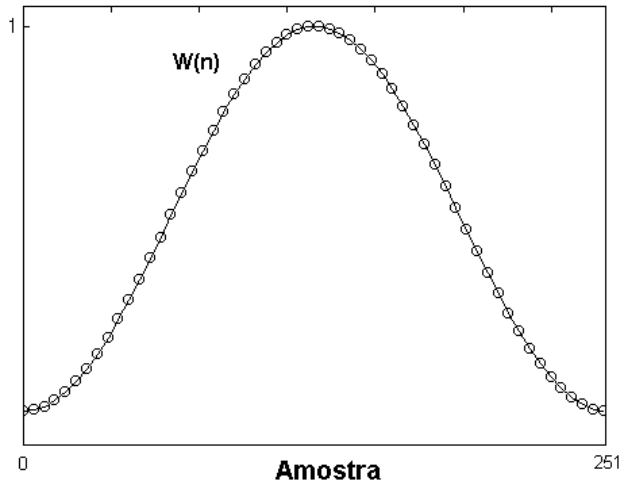



FIGURA 6.16 - Janela de Hammnig.

O registrador de endereçamento da memória que contêm os valores da janela de *Hamming* é implementado com um contador binário de 0...125, que opera nos modos crescente/decrescente.

A figura 6.17 mostra parcialmente a parte operativa da função de divisão de quadros e janelamento. Os valores da janela de *Hamming* se encontram armazenados na memória *rom_memory*, de 12 bits, de onde são lidos e multiplicados pelos valores do quadro do sinal de voz pré-enfatizado, $\tilde{s}(n)$.

Os valores de um quadro do sinal de voz pré-enfatizado, $\tilde{s}(n)$, se encontram armazenados na memória *ram_memory*. A memória *ram_memory* é uma memória de duplo acesso assíncrona, de 16 bits, que permite a leitura e a escrita de valores em paralelo. Ela foi dividida em três segmentos, para a implementação do algoritmo de separação em quadros explicado anteriormente.

Foi incluído um multiplexador que seleciona, através do sinal *Sel_Mux_Read*, entre realizar a leitura de um quadro $Q_i$ a partir da memória *RAM* ou a partir do sinal pré-enfatizado, $\tilde{s}(n)$. Isto permite o processamento do primeiro quadro, $Q_1$, que é multiplicado diretamente pela janela, sem necessidade de lê-lo da memória *RAM*.

Para multiplicar a janela de *Hamming* por cada quadro formado, a partir do sinal de voz, utilizou-se um multiplicador de 12 bits x 16 bits e o resultado de 28 bits foi

reduzido para 16 bits e armazenado em um registrador de saída, para ser utilizado na seguinte etapa: a extração de parâmetros.

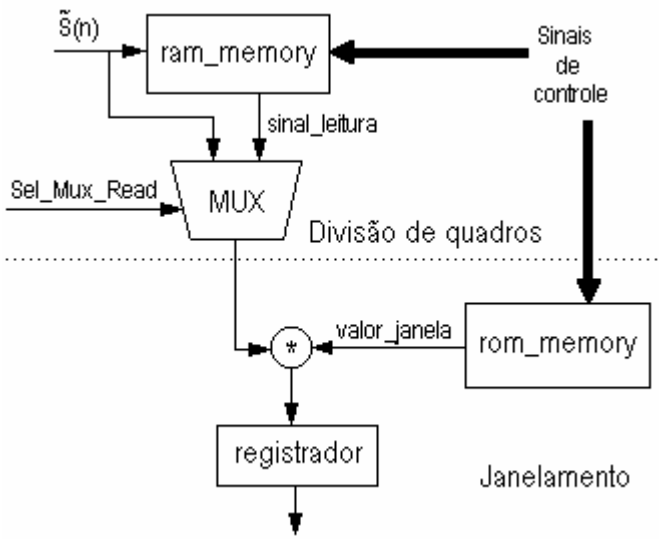



FIGURA 6.17 - Parte operativa da janela de Hamming e a divisão de quadros.

A figura 6.18 mostra como é realizado o endereçamento das memórias utilizadas. Ambas memórias são externas. A leitura da memória *ROM* é realizada através do *up_down counter* com duas freqüências de relógio diferentes $f_{ck2}$ e $f_{ck1}$, onde $f_{ck2}=3.f_{ck1}$. A freqüência $f_{ck1}$ é a freqüência com que os dados do sinal de voz são recebidos da pré-ênfase.

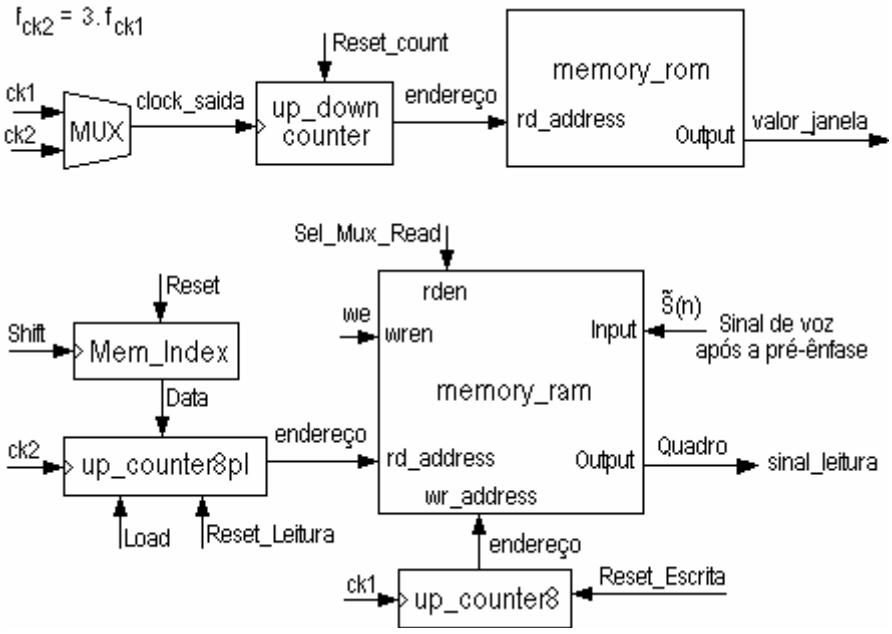



FIGURA 6.18 - Endereçamento das memórias RAM e ROM das funções de janelamento e divisão de quadros

O primeiro quadro não é lido da memória, como mencionado anteriormente, mas é processado diretamente do sinal de voz pré-enfatizado. Isto torna necessária a leitura

dos valores da janela de *Hamming* com uma freqüência, $f_{ck1}$. A partir do segundo quadro, todos os quadros processados são lidos da memória *RAM*.

A escrita na memória *RAM* é realizada conforme as amostras do sinal de voz pré-enfatizado, são recebidas. Isto significa que a escrita na memória *RAM* deve ser realizada com uma freqüência $f_{ck1}$. São lidos três blocos $B_i$ da memória *RAM*, enquanto é escrito apenas um. Portanto, a freqüência de leitura da *RAM* deve ser três vezes a freqüência de escrita. Isto significa que a freqüência de leitura da *RAM* é $f_{ck2}$.

Os valores da janela de Hamming devem ser lidos da memória *ROM* com a mesma freqüência com que está sendo feita a leitura da *RAM*, isto significa $f_{ck2}$.

A memória *RAM* utiliza o contador *up_counter8*, de 8 bits, para o cálculo dos endereços de escrita. Na memória são armazenadas 252 amostras, portanto, uma vez que o contador atinge o valor 251 é reinicializado e a contagem recomeça.

O cálculo dos endereços de leitura da memória *RAM* é feito através de um contador *up_counter8pl*, de 8 bits, com uma entrada *Load* para carregar o valor a partir do qual o contador inicia a contagem. Isto permite selecionar o endereço da *RAM* a partir do qual as amostras serão lidas. Este endereço pode ser 0, 84 ou 168, que são os endereços onde começam os três segmentos em que a *RAM* foi dividida. Este valor é fornecido pelo circuito *Mem_Index*, mostrado na figura 6.19.

O circuito *Mem_index* foi implementado com três registradores em cascata e um contador *Counter7*, de 7 bits, que faz a contagem de 0...83, contando o número de valores lidos de um segmento de memória (até 84 valores). Em cada registrador é armazenado um dos endereços em que cada segmento de memória inicia e o sinal de controle *Shift*, servirá como relógio indicando a circulação dos valores entre cada registrador. Na saída, *Data*, ter-se-á o endereço inicial para ser carregado no contador up_*counter8pl*.

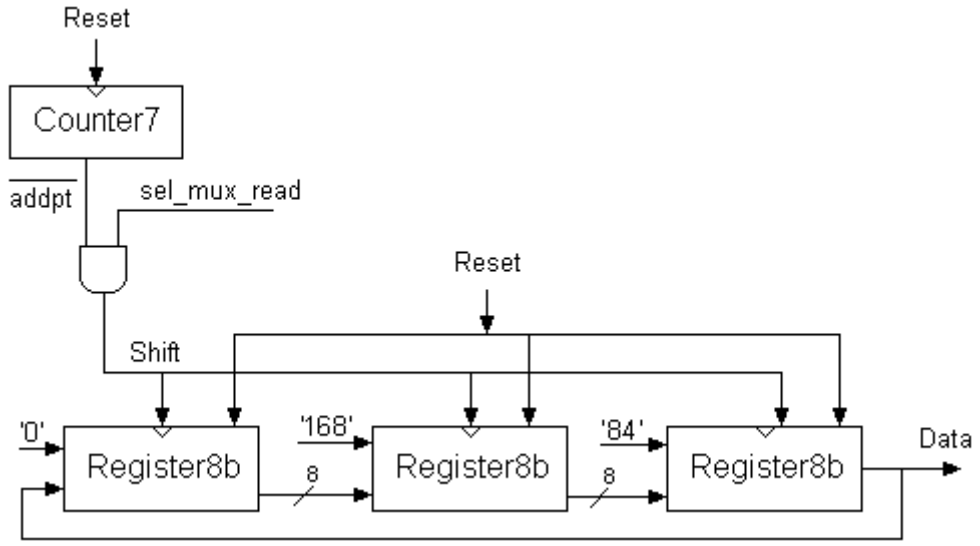



FIGURA 6.19 - Componente Mem_Index.

A figura 6.20 mostra os resultados da simulação da operação da função de janelamento em conjunto com a função de separação em quadros. As implementações em ponto-fixo, em *FPGA*, das funções de pré-ênfase, divisão em quadros e janelamento

que foram descritas neste capítulo, foram simuladas em *Maxplus II*, operando em conjunto. Os resultados da implementação dessas três etapas, trabalhando em conjunto em *FPGA*, foram comparados com os resultados do modelo de software descrito no capítulo 4. Uma medida de comparação apropriada seria o **erro relativo** entre o modelo em software e o hardware, que mede a diferença entre os resultados produzidos por estas implementações. Os resultados fornecem um erro relativo **máximo** de 4,88%, quando utilizamos palavras dentro da base de dados *REVOX*. Este valor de erro máximo é aceitável, considerando que mais da metade dos resultados obtidos com o circuito apresentado neste capítulo produz um erro relativo inferior a 1%.

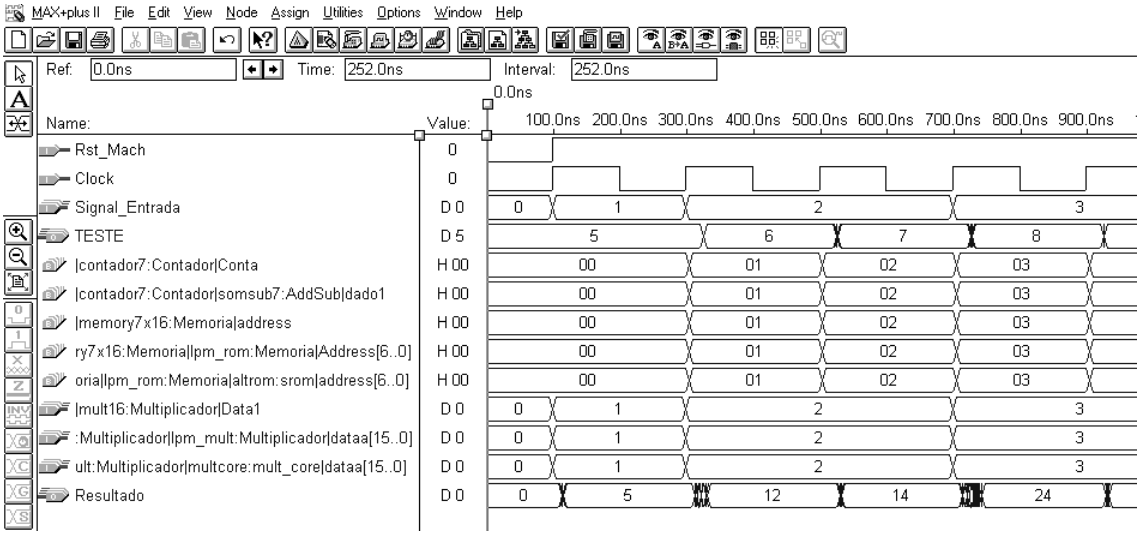



FIGURA 6.20 - Simulação da janela de Hamming.

A tabela 6.2 mostra os resultados da síntese das três funções operando em conjunto. As memórias foram sintetizadas em conjunto como se fossem memórias internas. Esta é a alternativa que se escolheu para efeitos de teste. No entanto, ao colocar em uma placa se optará pelo uso de memórias externas, como especificado inicialmente, na seção 6.2, pois isto exigirá um *FPGA* menor ao indicado na tabela 6.2.

TABELA 6.2 - Resultados da síntese da pré-ênfase, o algoritmo de divisão em quadros e o janelamento operando em conjunto.

| **Dispositivo** | **No de Células lógicas** | **% de utilização** | **Freqüência máxima** |
|---|---|---|---|
| EPF10K30ETC144-1 | 844 | 48 % | 18,21 Mhz |

Visto que a freqüência de amostragem do sinal de voz, para a base de dados utilizada, é 11025Hz então as duas freqüências necessárias para a operação das funções descritas neste capítulo são $f_{ck1}$=11025Hz e $f_{ck2}$=33075Hz, ambas menores do que a freqüência máxima de operação do circuito, indicada na tabela 6.2 (18,21 MHz).

# 7 O Subsistema de Extração dos Parâmetros Mel-cepstrais

O estudo prévio do capítulo 4 indica que os parâmetros que, na maioria dos modos de operação dos sistemas de *RAV*, produzem a melhor taxa de reconhecimento são os parâmetros mel-cepstrais ou a derivada destes. Isto é observado quando se trabalha quer em modo dependente, quer em modo independente do locutor, ou com palavras isoladas ou com linguagem contínua, para reconhecer/verificar locutor ou para reconhecer comandos. Portanto, o subsistema de extração de parâmetros proposto está baseado em parâmetros mel-cepstrais.

Depois da aplicação da janela de *Hamming*, de 23ms, é realizado o cálculo dos parâmetros mel-cepstrais de cada quadro. Neste trabalho, foram calculados 27 parâmetros mel-cepstrais. A figura 7.1 mostra a maneira de obter os parâmetros mel-cepstrais.

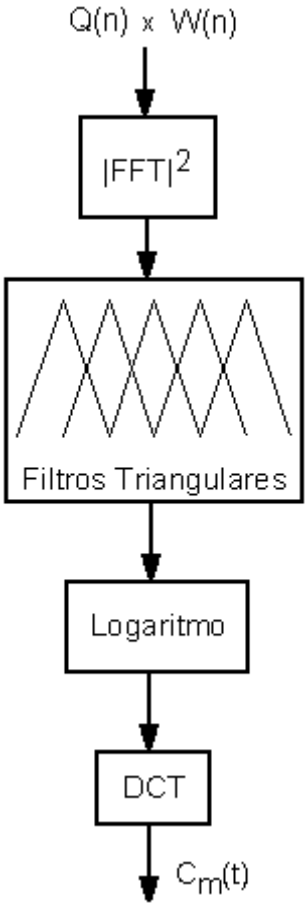



FIGURA 7.1 - Extração dos parâmetros mel-cepstrais.

Os quadros do sinal de voz, após terem sido multiplicados pela janela de *Hamming*, são processados da seguinte maneira:

1) Calcula-se o espectro de energia (LPC ou FFT).

2) Através de um banco de filtros triangulares, calcula-se a energia de cada canal.

3) Calculam-se o logaritmo do valor da energia de cada canal.

4) Calcula-se a transformada coseno.

Na continuação descreveremos cada uma destas etapas.

## 7.1 Função FFT

Um primeiro passo para a extração dos parâmetros mel-cepstrais é o cálculo do espectro de energia, o qual é feito através do módulo da *FFT* do sinal segmentado em quadros.

A função *FFT* implementa a seguinte equação:

$$X(k) = \sum_{n=0}^{Nw-1} S_Q(n) \cdot e^{-j \cdot 2\pi \cdot k.n / Nw}, \qquad 0 \le k \le Nw-1 \tag{7.1}$$

onde $S_Q(n)$ é o sinal de voz proveniente da separação em quadros após ter sido processado pela janela de *Hamming*. $N_w=251$ corresponde ao número de amostras dentro de um quadro.

O espectro de energia é dado por:

$$F_k = |X(k)|^2, \qquad 0 \le k \le \frac{Nw}{2} - 1 \tag{7.2}$$

A função *FFT* implementa a equação 7.1 iterativamente, através das seguintes equações:

$$\begin{aligned} X_l(k) &= X_{l-1}(k) + W^p \cdot X_{l-1}\left(k + N/2^l\right) \\ X_l\left(k + N/2^l\right) &= X_{l-1}(k) - W^p \cdot X_{l-1}\left(k + N/2^l\right) \end{aligned} \tag{7.3}$$

onde $X_l(k)$ é o valor de $X(k)$ na iteração $l$. O número de iterações necessário para obter os valores finais de $X(k)$ é $l=log_2N$. Foram utilizadas 256 amostras, pois os 252 valores provenientes da divisão em quadros foram completados com zeros, para obter um número de amostras de entrada do tipo $2^n$, sendo n inteiro. O fator $W_p$ é definido como:

$$W^p = e^{-j \cdot 2\pi \cdot p / Nw}, \qquad p = n.k \tag{7.4}$$

A figura 7.2, mostra a estrutura de operações necessárias para a implementação da FFT. Duas memórias ROM *Twiddle_Re_Mem* e *Twiddle_Im_Mem* de 128 palavras de 16 bits são utilizadas para armazenar a parte real e a parte imaginária do fator $W^p$, da equação 7.4. Quatro memórias *RAM Re0*, *Im0*, *Re1* e *Im1*, de 128 palavras de 16 bits cada uma, são utilizadas para armazenar os valores intermediários de cada iteração. Estas memórias têm dupla porta de acesso.

As saídas a cada iteração foram separadas, para facilitar o processamento em hardware, em suas partes real e imaginária da seguinte maneira:

$$a = Re\{X_l(k)\}, \qquad b = Im\{X_l(k)\}$$
$$a^{dn} = Re\left\{X_l\left(k + N/2^l\right)\right\}, \qquad b^{dn} = Im\left\{X_l\left(k + N/2^l\right)\right\} \tag{7.5}$$

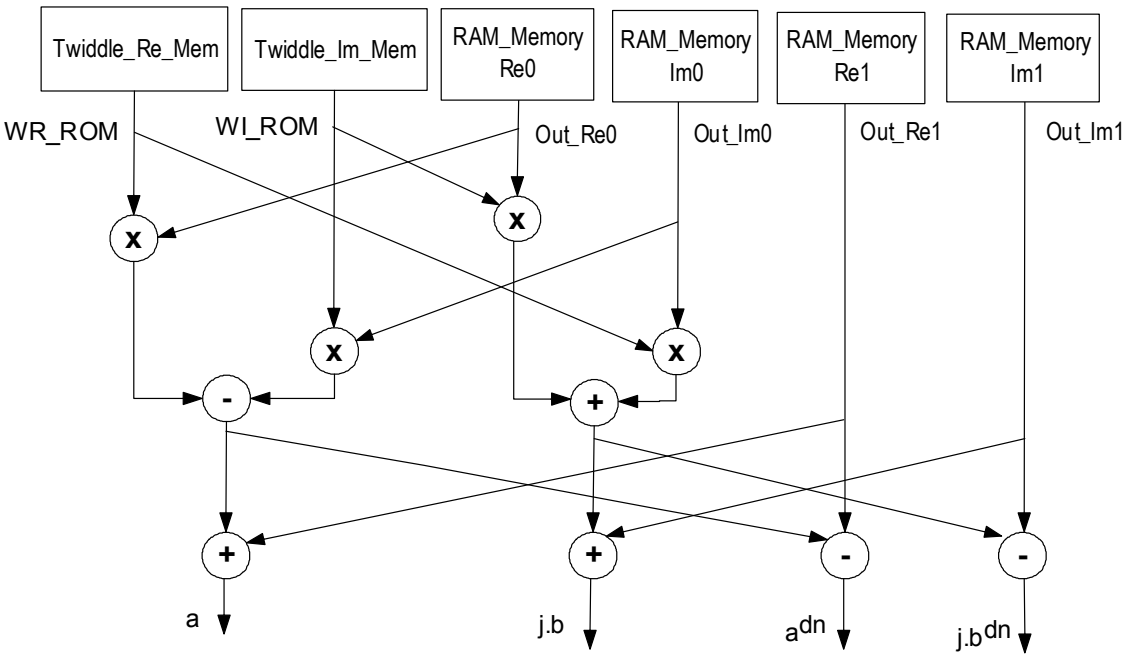



FIGURA 7.2 - Operações cruzadas para a *FFT* (butterfly).

A figura 7.3 mostra a entrada de dados nas memórias *RAM*, que armazenam os valores intermediários em cada iteração.

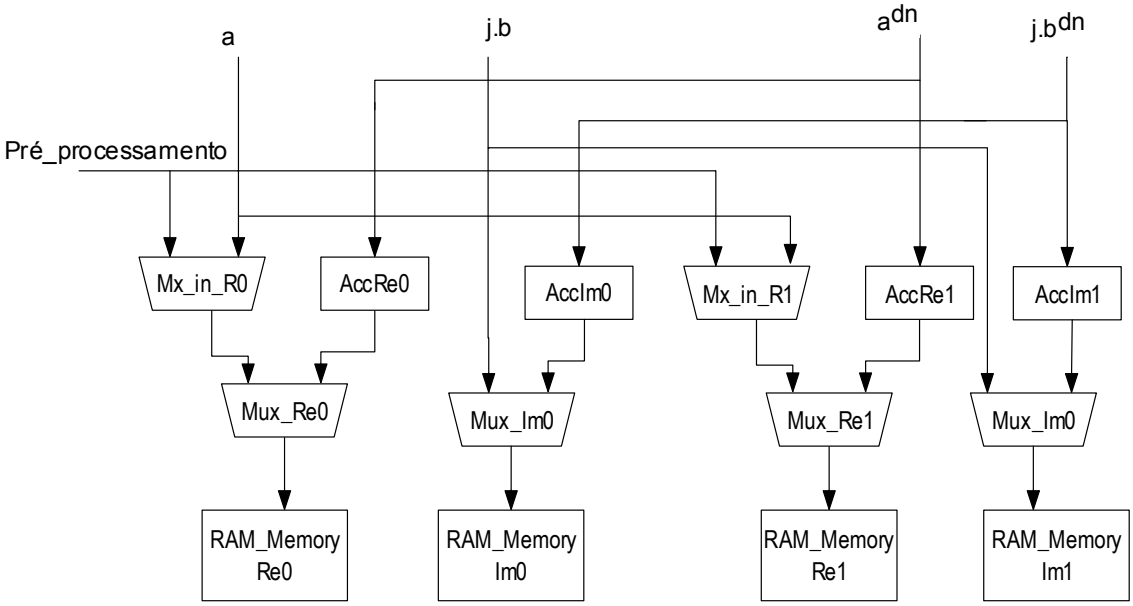



FIGURA 7.3 - Entrada de dados nas memorias *RAM*.

Os valores de *X(k)* a cada iteração são novamente introduzidos nas memórias *RAM* para o cálculo das iterações seguintes.

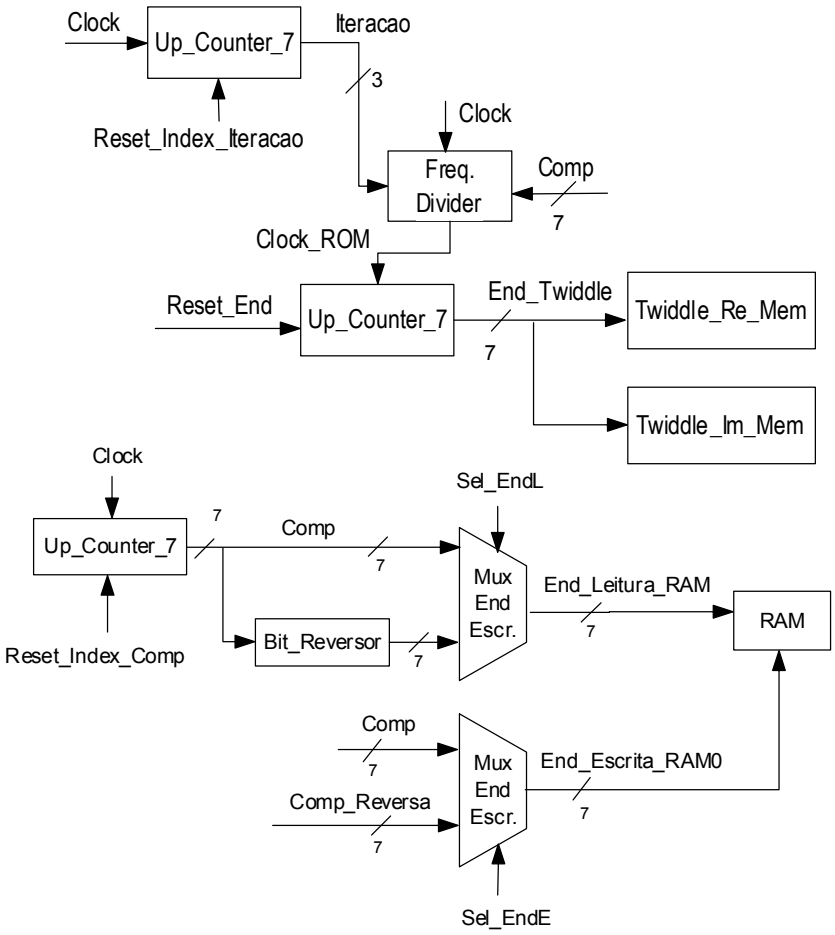



FIGURA 7.4 - Enderaçamento das memórias.

Os resultados da síntese são mostrados na tabela 7.1. A função *FFT* foi sintetizada em um dispositivo EP1K50FC484-1 da família *ACEX*. Foram utilizadas 1776 células lógicas ocupando 61% do *FPGA*. A freqüência máxima de operação é de 16,72MHz.

TABELA 7.1 - Síntese da função *FFT*.

| **Dispositivo** | **Células lógicas utilizadas** | **% de Ocupação** | **Freqüência Máxima** |
|---|---|---|---|
| EP1K50FC484-1 | 1776 | 61% | 16,72MHz |

## 7.2 Função de Filtros Triangulares

O espectro do sinal de voz é processado por um banco de filtros triangulares, similar ao mostrado na figura 1.9, no capítulo 1.

Foram utilizados 27 filtros triangulares para cobrir a faixa de freqüências do sinal de voz. Visto que o sinal de voz foi amostrado com uma freqüência $f_s=11025Hz$,

pelo critério de Nyquist [RAB 93], o espectro do sinal terá freqüências abrangendo até $f_s/2 = 5512{,}5\ Hz$.

TABELA 7.2 - Banco de filtros triangulares.

| Índice | Freq. Central (Hz) | Largura de Banda (Hz) | Índice | Freq. Central (Hz) | Largura de Banda (Hz) |
|---|---|---|---|---|---|
| 0 | 100 | 200 | 14 | 1635 | 322 |
| 1 | 200 | 200 | 15 | 1808 | 354 |
| 2 | 300 | 200 | 16 | 1989 | 390 |
| 3 | 400 | 200 | 17 | 2198 | 429 |
| 4 | 500 | 200 | 18 | 2418 | 472 |
| 5 | 600 | 200 | 19 | 2670 | 519 |
| 6 | 700 | 200 | 20 | 2937 | 571 |
| 7 | 800 | 200 | 21 | 3240 | 628 |
| 8 | 900 | 200 | 22 | 3564 | 690 |
| 9 | 1000 | 200 | 23 | 3931 | 759 |
| 10 | 1100 | 220 | 24 | 4324 | 835 |
| 11 | 1220 | 242 | 25 | 4766 | 919 |
| 12 | 1342 | 266 | 26 | 5243 | 1011 |
| 13 | 1486 | 293 | | | |

Visto que unicamente a metade do espectro de freqüência é considerada, a entrada da função de filtros triangulares é um vetor com 128 valores correspondentes ao espectro de um quadro do sinal de voz. O banco de filtros triangulares calcula a energia $E_j$ em cada canal $j$:

$$E_j = \sum_{k=0}^{K-1} \phi_j(k) \cdot F_k, \qquad 0 \le j \le J-1 \tag{7.6}$$

onde $J=27$, é o número de filtros triangulares $\phi_j$ utilizados. Desta maneira, observa-se que a saída é constituída por 27 valores correspondentes a cada canal do banco de filtros triangulares. A tabela 7.2 mostra as freqüências centrais e larguras de banda do banco de filtros implementado. Esta distribuição de freqüências obedece aos valores sugeridos em [SAN 94], e têm sido muito utilizada por acreditar-se que permite avaliar as freqüências que o ouvido humano discrimina melhor.

Visto que unicamente 128 valores do espectro de freqüência são introduzidos no banco de filtros triangulares, então também serão necessários unicamente 128 dos valores do banco de filtros triangulares. Estes valores foram armazenados em uma memória *ROM*, *Valor_Filtro_ROM*, de 128 palavras e largura de 12 bits. Em outra ROM, *Numero_Amostras_ROM*, armazenou-se a quantidade de valores da função triangular que cada um dos 27 canais utiliza. Isto serve para poder calcular o número de vezes que será realizada a operação soma-acumula. Esta estratégia reduz o tempo de processamento.

A figura 7.5 mostra a parte operativa da função de filtros triangulares. Os registradores *St0*, *St0_ROM*, *St1* e *St2* implementam um pipeline de três estágios:

1. No primeiro estágio, cada amostra do espectro proveniente do bloco de *FFT* é armazenada no registrador *St0*. Ao mesmo tempo, o ganho do filtro triangular correspondente é lido da ROM *Valor_Filtro_ROM* e armazenado no registrador *St0_ROM*.

2. No segundo estágio, são lidos os valores dos registradores *St0* e *St0_ROM*, multiplicando-os e armazenando o resultado no registrador *St1*.

3. No terceiro estágio, o valor armazenado em *St1* é lido e escalonado mediante um circuito combinacional de divisão (deslocamento de 3 bits) por um fator de 8. Ainda neste estágio, realiza-se a soma com o valor que está armazenado no acumulador *St2*. Este processo de acumulação é repetido até completar todos os valores que correspondem a um canal do banco de filtros triangulares. Uma vez concluído este processo para um canal, o multiplexador *Mux_feed* reinicializa o acumulador *St2* com o primeiro valor correspondente ao próximo canal e o processo é repetido até completar todos os canais. A saída do acumulador *St2* é convertida para 16 bits através de um circuito de arredondamento.

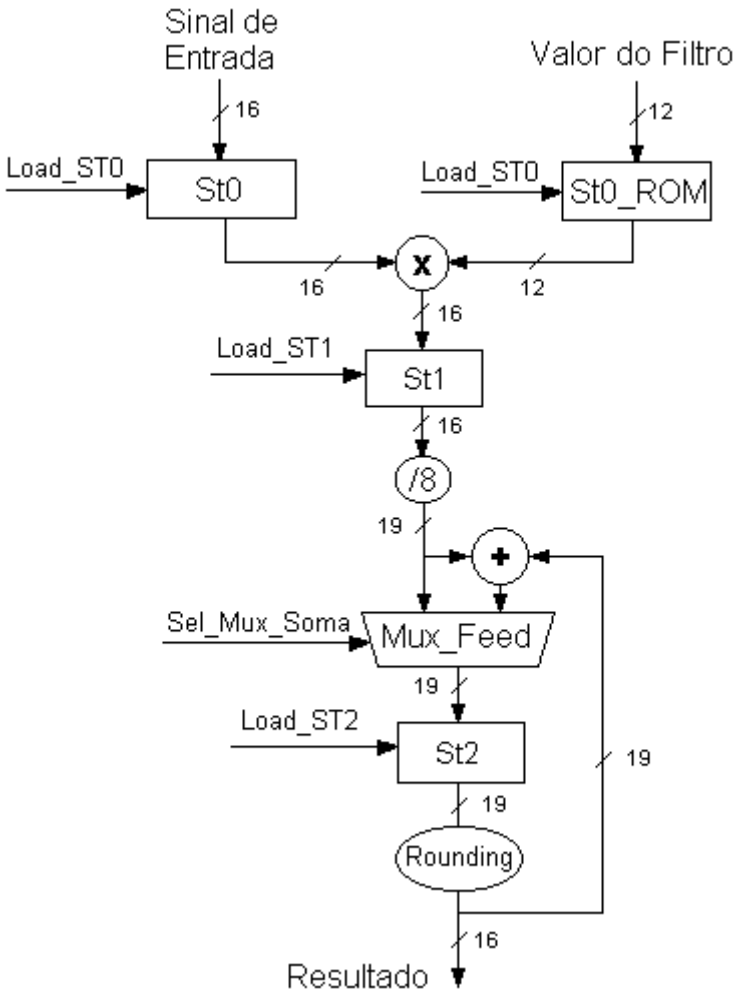



FIGURA 7.5 - Parte operativa da função de filtros triangulares.

As memórias são endereçadas, de maneira simples, através de contadores, como mostra a figura 7.6. Um contador de 7 bits, *Up_counter7*, é utilizado para endereçar os

128 valores armazenados em *Valor_Filtro_ROM*. Um contador de 5 bits *Up_Counter5* (que fará a contagem de 0...26) é utilizado para endereçar os 27 valores armazenados em *Numero_Amostras_ROM*, e que correspondem a cada canal do banco de filtros. Um contador de 4 bits é utilizado para indicar a ordem do valor que está sendo lido da memória ROM, *Valor_Filtro_ROM*. Uma vez que *Up_counter4* atinge o valor igual ao total de valores utilizados dentro de um canal, ele é reinicializado e sua contagem recomeça para o próximo canal.

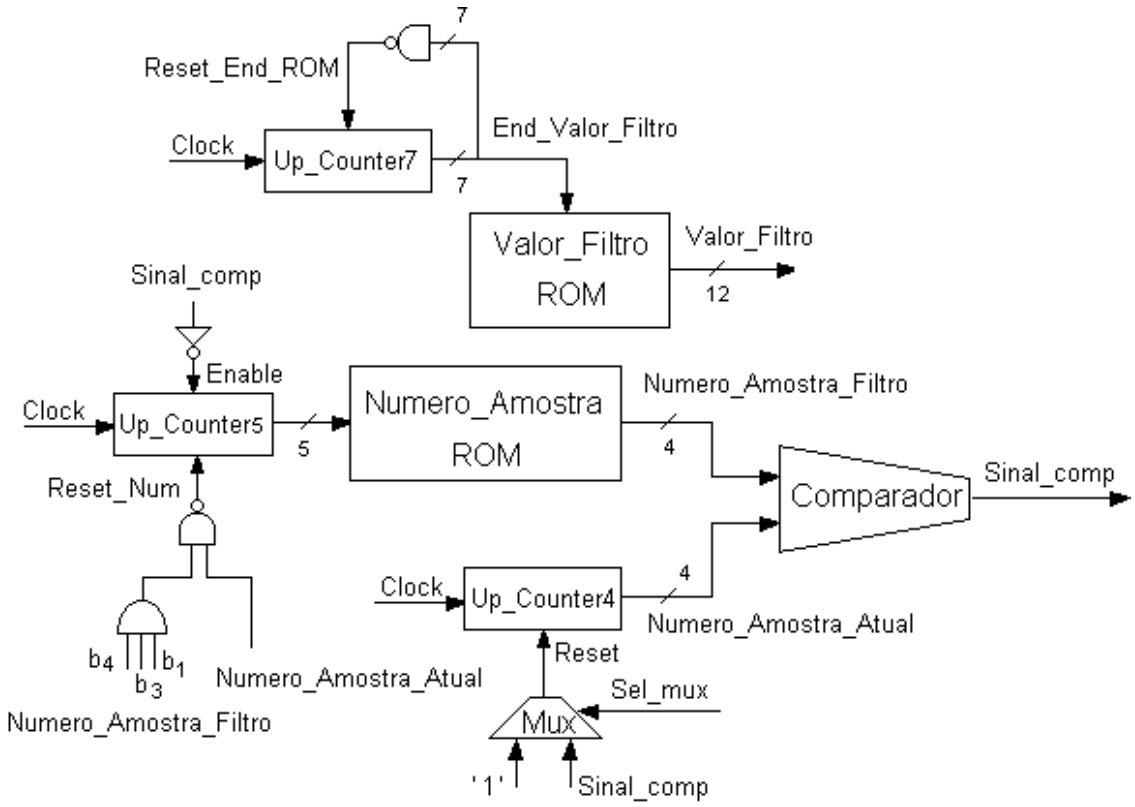



FIGURA 7.6 - Endereçamento das memórias ROM.

A tabela 7.3 mostra os resultados de síntese da função de filtros triangulares implementado em *FPGA*. Foram testadas três implementações, alterando o barramento interno. A primeira implementação, o circuito *triang_filters_v1*, utilizou um barramento interno de 16 bits, não realizando nenhum escalonamento entre etapas. Esta implementação utilizou 932 células lógicas (LCs) ou 53% do FPGA EPF10K30ETC144-1. A freqüência de operação máxima é de 20,83MHz. No entanto, a comparação entre os resultados produzidos pelo hardware e o modelo comportamental produziu erros acima de 100%.

TABELA 7.3 - Resultados da síntese da função de filtros triangulares.

| **Circuito** | **Dispositivo** | **Células lógicas utilizadas** | **% de Ocupação** | **Freqüência máxima** |
|---|---|---|---|---|
| triang_filters_v1 | EPF10K30ETC144-1 | 932 | 53% | 20,83 MHz |
| triang_filters_v2 | EPF10K30ETC144-1 | 961 | 55% | 22,12 MHz |
| triang_filters_v3 | EPF10K30ETC144-1 | 750 | 43% | 24,57 MHz |

Uma melhor precisão foi obtida ao aumentar o barramento interno. Isto foi feito na implementação de uma segunda versão do circuito de filtros triangulares: *triang_filters_v2*. Após a multiplicação e armazenar o resultado em *St1*, aumento-se o número de bits para 19 fazendo um escalonamento (divisão por $2^3$) do resultado do produto. O fator de escalonamento foi escolhido, pois demonstrou ser o mais apropriado, após examinar os máximos valores que a saída de cada canal do banco de filtros produz. Tais valores são mostrados na figura 7.7, onde se pode observar que o maior valor produzido por um canal do banco de filtros é 5,9, correspondente ao canal de índice 25. Portanto, para manter os valores, produzidos pelos filtros, a um número menor que 1, o fator $2^3$ é suficiente.

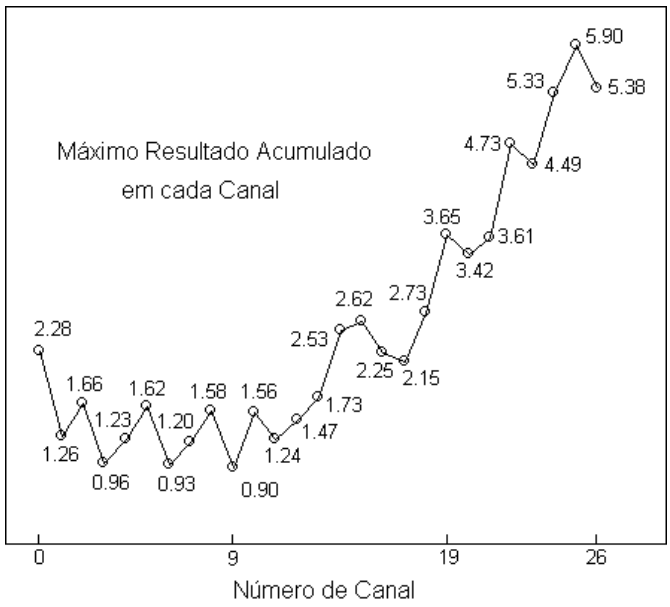



FIGURA 7.7 - Máximo resultado acumulado em cada canal do banco de filtros triangulares.

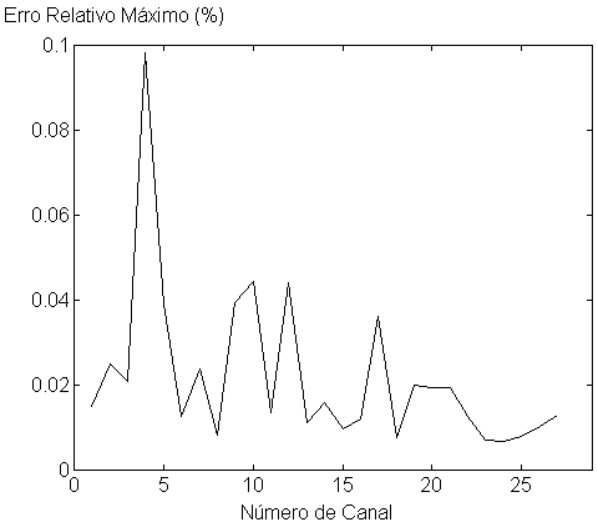



FIGURA 7.8 - Erro entre os resultados de hardware e o modelo comportamental para o circuito *triang_filters_v2* .

Esta implementação utilizou 961 células lógicas (LCs) ou 55% do FPGA EPF10K30ETC144-1. A freqüência de operação máxima é de 22,12MHz. Os erros relativos máximos entre os resultados produzidos pelo banco de filtros triangulares em hardware e o modelo comportamental são mostrados para cada um dos canais do banco de filtros do circuito *triang_filters_v2*, na figura 7.8. Unicamente são mostrados os máximos erros por cada canal. O máximo erro entre todos os canais foi de 0,098%.

Analisou-se a redução de bits dentro da memória ROM *Valor_Filtro_ROM*. Considerando diversas larguras, concluiu-se que uma largura de 12 bits seria suficiente para garantir uma boa precisão. Por este motivo, reduziu-se a largura de bits da *Valor_Filtro_ROM*, de 12 para 16 bits, na implementação do circuito *triang_filters_v3*.

Esta implementação utilizou 750 células lógicas (LCs) ou 43% do FPGA EPF10K30ETC144-1. A freqüência de operação máxima é de 24,57MHz. Os erros relativos máximos entre os resultados produzidos pelo banco de filtros triangulares em hardware e o modelo comportamental são mostrados para cada um dos canais do banco de filtros do circuito *triang_filters_v3*, na figura 7.9. O máximo erro entre todos os canais foi de 0,11%.

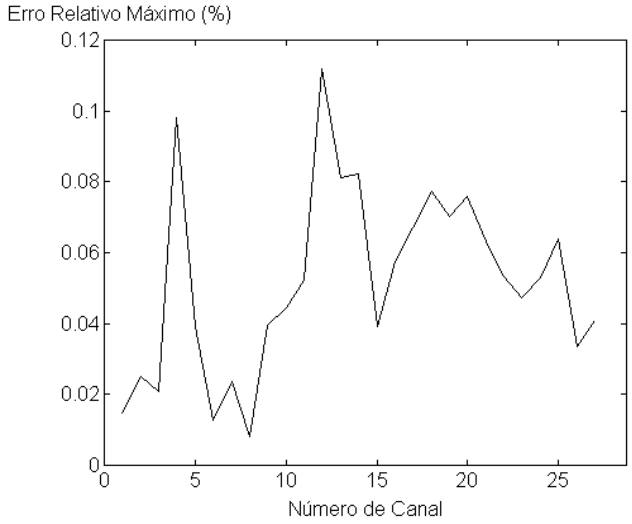



FIGURA 7.9 - Erro entre os resultados de hardware e o modelo comportamental para o circuito *triang_filters_v3* .

Comparando as três implementações, acima descritas, se observaram melhores resultados para a terceira implementação, que ocupa um menor número de células, tem uma freqüência de operação maior e os erros em relação ao modelo comportamental são menores que 0,11%.

## 7.3 Processador Logaritmo

Foi implementado o algoritmo CORDIC (COordinate Rotation DIgital Computer), que permite o cálculo de funções transcendentais de maneira iterativa.

O algoritmo CORDIC consiste na rotação de um vetor *(x,y)* em um angulo (modo de rotação) ou no alinhamento com um dos eixos coordenados (modo de vetoração). Estas operações podem ser realizadas em sistemas de coordenadas lineares,

circulares e hiperbólicas. Para o cálculo do logaritmo são necessárias coordenadas hiperbólicas. O algoritmo utiliza rotações em ângulos elementares de valor conhecido ($\tanh^{-1}(2^{-i})$ em coordenadas hiperbólicas) armazenadas em uma memória. A iteração básica ou microrotação em coordenadas hiperbólicas é:

$$\begin{aligned} x_{i+1} &= x_i - m \cdot y_i \cdot \delta_i \cdot 2^{-i} \\ y_{i+1} &= y_i + x_i \cdot \delta_i \cdot 2^{-i} \\ z_{i+1} &= z_i - \delta_i \cdot arctgh(2^{-i}) \end{aligned} \tag{7.7}$$

onde $(x_0, y_0)$ são as coordenadas iniciais do vetor e a coordenada $z$ acumula o valor do ângulo. O coeficiente $\delta_i$ indica a direção de cada microrotação.

O domínio de convergência do CORDIC está limitado pelo intervalo $[-\pi/2, \pi/2]$, devido ao uso de $2^0$ para a tangente hiperbólica na primeira iteração [WAL 71] [AND 96] [EDW 96]. Para números fora deste intervalo, aplica-se o algoritmo para o valor escalonado. A relação entre o arco tangente hiperbólico de um número $\beta$ e o logaritmo natural do mesmo é:

$$ln(\beta) = 2 \cdot arctgh \cdot \left( \frac{\beta - 1}{\beta + 1} \right) \tag{7.8}$$

Devido à divergência do arco tangente hiperbólico no ponto 1, o intervalo para os valores de cálculo deve ser $0.5 \le \beta < 1$. Desta maneira, os valores dentro do intervalo $1 < \beta < 0.5$ são multiplicados ou divididos por 2 até satisfazer a regra anterior. Em forma analítica:

$$ln(\beta) = ln(\beta' \cdot 2^n) = ln(\beta') + n \cdot ln(2) \tag{7.9}$$

onde n será um número inteiro positivo quando $\beta < 0.5$, zero quando $0.5 \le \beta < 1$, ou um número inteiro negativo quando $\beta > 1$.

O sistema proposto está composto de duas etapas de processamento. Inicialmente, o valor de entrada é analisado e, se necessário, realiza-se o escalonamento do valor. Posteriormente, o algoritmo CORDIC é aplicado ao valor residual e o resultado é somado com o fator proveniente da etapa de escalonamento, assim como é mostrado na figura 7.10.

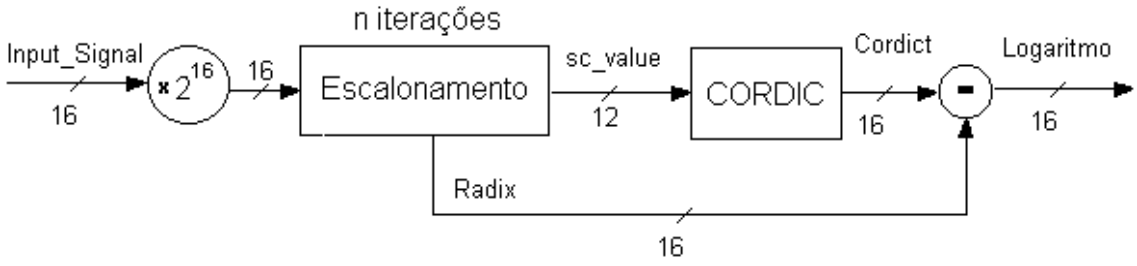



FIGURA 7.10 - Diagrama de blocos da operação de escalonamento.

Na figura 7.11, é mostrado como é realizado o escalonamento. Inicialmente, o sinal de entrada é armazenado em um registrador de deslocamento de 16 bits, *In_Shift_Reg*. Em cada pulso de relógio, um bit vai sendo armazenado no registrador *Out_Shift_Reg*. O deslocamento é desativado, quando o escalonamento é concluído, ativando-se o sinal *Done*.

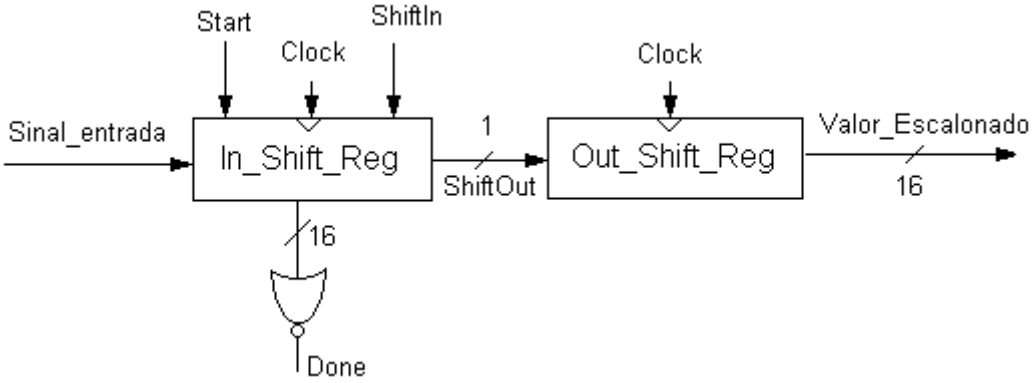



FIGURA 7.11 - Cálculo do valor escalonado.

A figura 7.12 mostra o cálculo do fator de escalonamento, *Radix*. O fator de escalonamento é $n.ln(2)$, onde n é o número de vezes que é realizado o deslocamento do valor inicial do sinal de entrada. Para uma melhor precisão é utilizado um barramento de dados interno de 20 bits, sendo externamente aproximado para 16 bits.

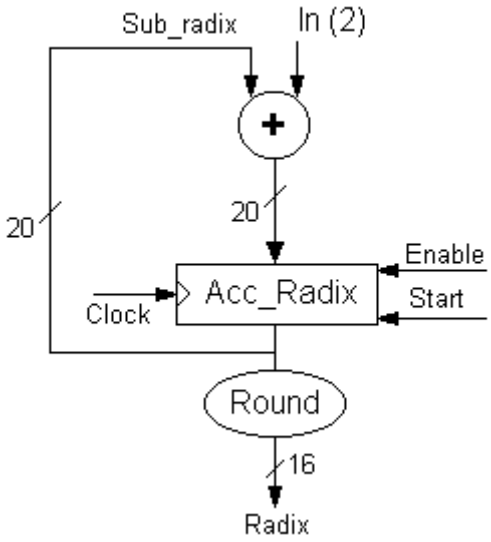



FIGURA 7.12 - Cálculo do fator de escalonamento.

A arquitetura CORDIC projetada é mostrada nas figuras 7.13 e 7.14. Inicialmente, o valor escalonado é selecionado pelos multiplexadores *L*, assim também o valor '1' é selecionado pelos multiplexadores *P*. Com estes valores é executada uma soma ou subtração e o resultado é armazenado nos registradores *Reg_X* e *Reg_Y*. As saídas destes registradores estão conectadas Nas seguintes iterações, os multiplexadores L selecionam os conteúdos dos registradores *Reg_X* e *Reg_Y* e os multiplexadores P selecionam os conteúdos dos registradores *shift_x* e *shift_y*. Depois disso, o algoritmo

para o cálculo do logaritmo é executado e o valor final $Ln(\beta')$ é gerado para ser somado ao número prévio calculado por escalonamento.

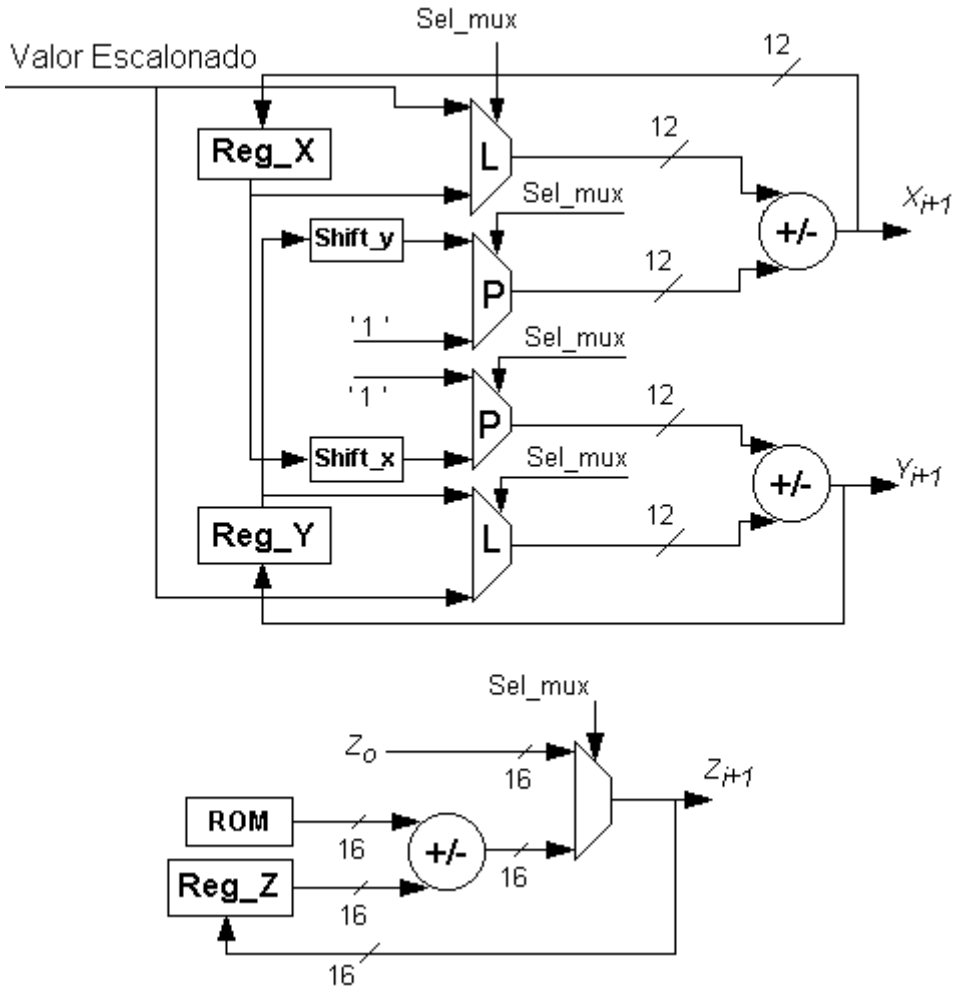



FIGURA 7.13 - Parte operativa da função CORDIC.

Utilizou-se uma representação numérica em ponto-fixo com 16 bits de precisão e representação em complemento a dois. Internamente, trabalhou-se em algumas etapas com 20 bits para aumentar a precisão. Algumas etapas em que a diminuição para 12 bits acrescentou pouca diminuição da precisão, utilizaram menos bits, para diminuir o hardware. Todo o sistema foi testado, comparando os resultados do Maxplus II e a modelagem do algoritmo em software.

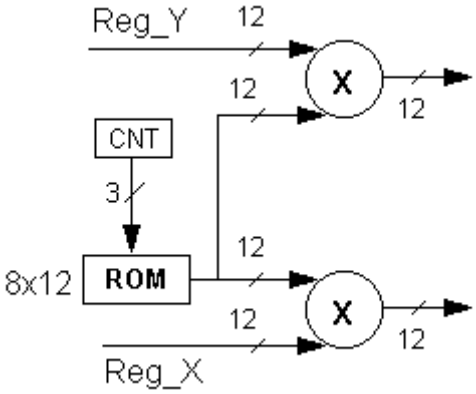



FIGURA 7.14 - Dispositivos *Shift_X* e *Shift_Y*.

A tabela 7.4 mostra os resultados da síntese em *FPGA* da função CORDIC, juntamente com a função de escalonamento, para o cálculo de logaritmos. Duas implementações foram realizadas, mudando a largura dos dados. No primeiro circuito, *log_v1*, todos os componentes utilizam dados com largura de 16 bits. Esta implementação produziu erros elevados (acima de 100%) em relação ao modelo em software. Melhor precisão foi conseguida mudando a largura dos dados entre algumas das etapas dentro do circuito, tal como foi explicado anteriormente. Estas mudanças foram implementadas no circuito *log_v2*.

TABELA 7.4 - Resultados da síntese da função CORDIC para o cálculo de logaritmos.

| **Circuito** | **Dispositivo** | **Células Lógicas Utilizadas** | **% de Ocupação** | **Freqüência máxima** |
|---|---|---|---|---|
| *log_v1* | EPF10K30ETC144-1 | 650 | 37% | 28,73 MHz |
| *log_v2* | EPF10K30ETC144-1 | 904 | 52% | 23,80 MHz |

A figura 7.15 mostra a comparação do software com o hardware da função logaritmo para o circuito *log_v2*. Como pode se observar, as curvas ficam praticamente superpostas. A figura 7.16 mostra o erro relativo. Pode observar-se que os erros são inferiores a 0,3%.

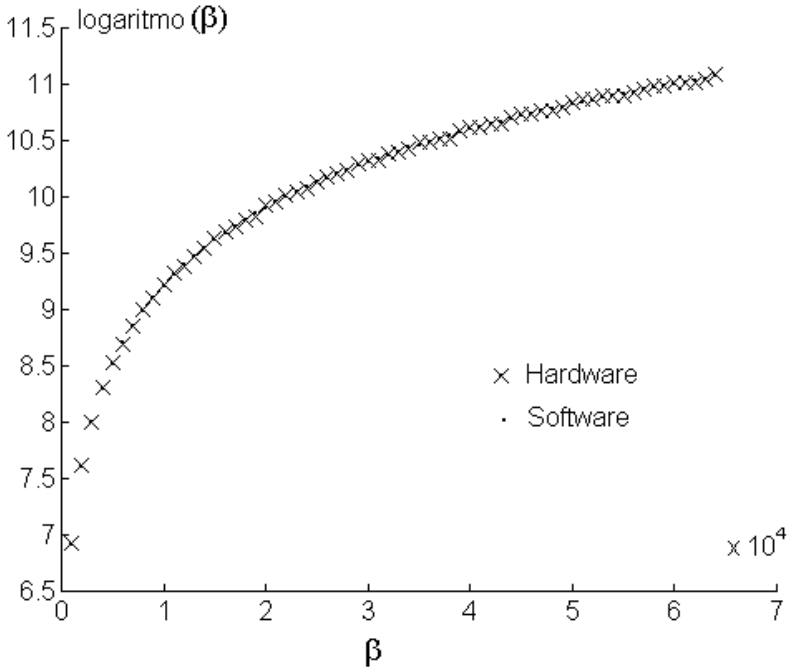



FIGURA 7.15 - Comparação do software com o hardware da função logaritmo.

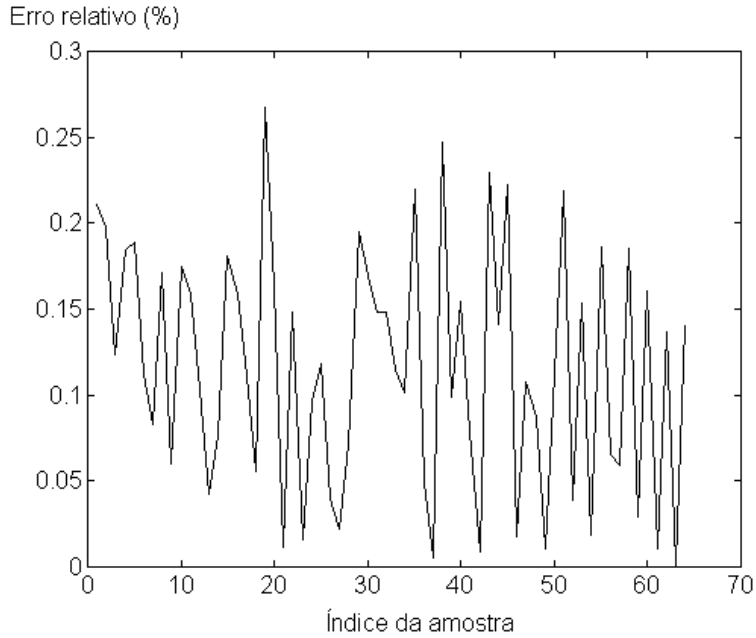



FIGURA 7.16 - Erro relativo produzido pela função logaritmo em relação ao modelo em software, para o circuito log_v2.

## 7.4 Transformada Coseno (DCT)

A *DCT* é aplicada a um conjunto de valores proveniente do logaritmo natural do espectro à saída dos filtros triangulares.

A função *DCT* calcula a seguinte operação:

$$C_m = \beta \cdot \sum_{j=0}^{N-1} cos\left( m\frac{\pi}{N}\left(j+0.5\right)\right) \cdot f_j \quad , \qquad \beta = \begin{cases} \sqrt{1/N}, & se \quad m = 0 \\ \sqrt{2/N}, & se \quad m > 0 \end{cases} \qquad (7.10)$$

onde $C_m$ indica a transformada coseno da função $f_j$. $N$ é o número de amostras das quais é calculada a transformada coseno. $f_i$ são as 27 saídas do banco de filtros triangulares. Para a extração de parâmetros mel-cepstrais foi necessário o cálculo da *DCT* de 27 valores, correspondentes às saídas dos 27 filtros.

A *DCT* foi implementada de maneira similar à dos filtros triangulares. Foi implementado mediante um circuito de multiplicação e acumulação (*MAC*). São calculados os 27 produtos da equação 7.10 e acumulados para obter o resultado final.

A figura 7.17 mostra a implementação da parte operativa da função transformada coseno (*DCT*). O conjunto de valores da função coseno ($27^2$) dentro da transformada foi armazenado em uma memória *ROM*. Alguns registradores *St0*, *St1* e *St2* foram utilizados para implementar o pipeline do circuito.

Ao processar o primeiro quadro, o multiplexador *Mux_Feed* seleciona o cálculo do primeiro produto e armazena-o no acumulador *St2*. Os produtos que são calculados depois são selecionados pelo multiplexador *Mux_Feed*, para ser somados com o valor inicial do acumulador *St2*, até completar os 27 produtos correspondentes ao primeiro quadro. O processo é repetido com o primeiro produto do seguinte quadro e depois com os valores restantes do quadro. Este processo continua até completar o total dos quadros.

A figura 7.18 mostra a conexão entre as memórias utilizadas para a função *DCT*. O sinal de entrada, constituído pelos 27 valores provenientes do banco de filtros triangulares, é armazenado temporariamente numa memória *RAM* de dupla porta de acesso. A escrita na memória *RAM* é realizada durante a borda de descida do sinal de relógio *clock* e a leitura é realizada durante a borda de subida. Um contador de 5 bits, *up_counter5*, é utilizado para calcular os endereços desta memória, que realiza a contagem 0...26. Este contador é reinicializado uma vez atingido o valor 26 ("*11010*"). O sinal *Reset* deste contador é utilizado para o controle do multiplexador *Mux_Feed*. A memória *ROM* de cosenos é endereçada por um contador de 10 bits, *up_counter10*, que faz a contagem 0...$27^2$-1.

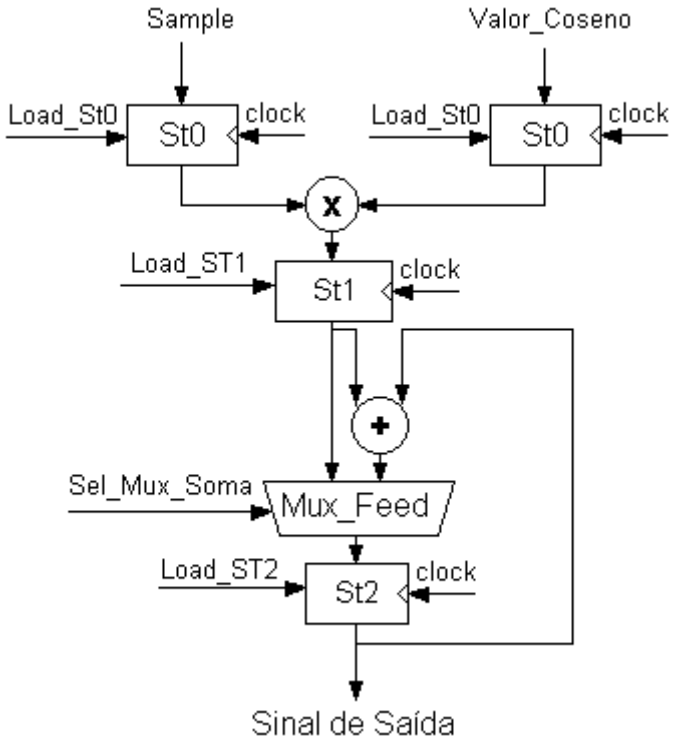



FIGURA 7.17 - Parte operativa da função DCT.

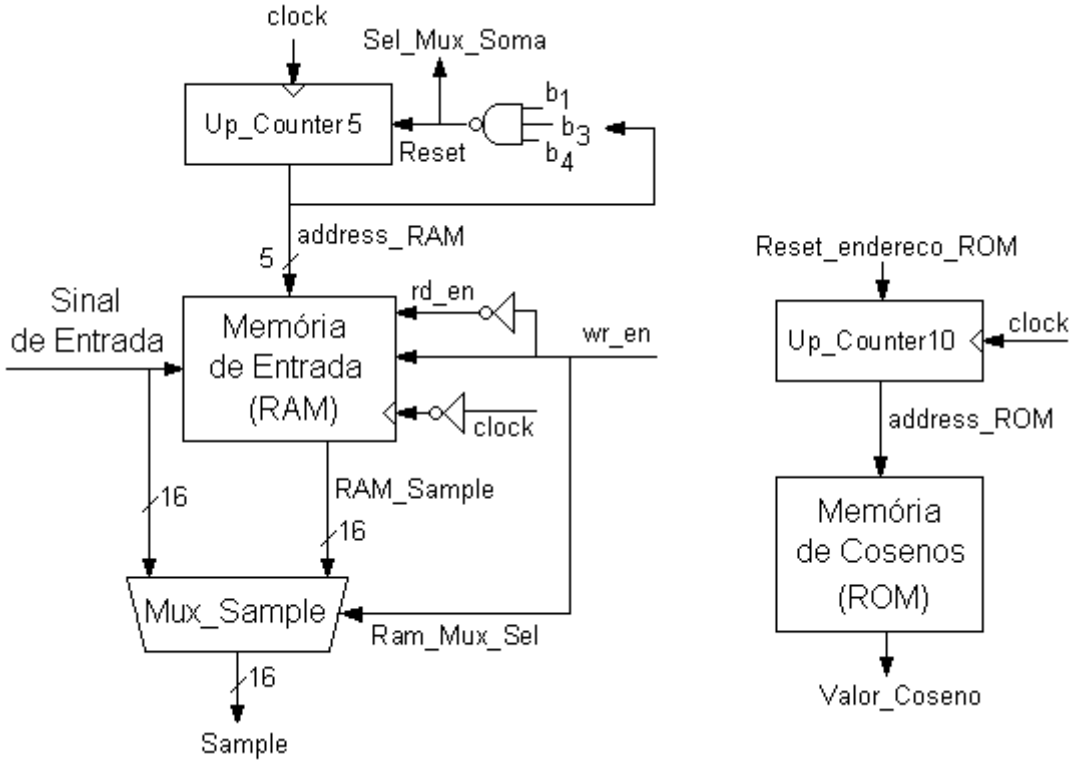



FIGURA 7.18 - Conexão entre memórias.

A tabela 7.5 mostra os resultados da síntese. Foram realizadas três implementações. No circuito *dct_v1*, a função *DCT* foi sintetizada em um

EPF10K20TC144-3 e ocupou 910 células lógicas ou 78% do *FPGA*. A máxima freqüência de operação foi de 9,07MHz. A memória *ROM* foi incluída dentro do *FPGA*. A memória *RAM* foi colocada externamente ao *FPGA*. O registrador de saída *St1*, o somador, o multiplexador *Mux_Feed* e o acumulador *St2* utilizaram 32 bits.

TABELA 7.5 - Síntese da DCT.

| Circuito | Dispositivo | Células Lógicas Utilizadas | % de Utilização | Freqüência máxima |
|---|---|---|---|---|
| dct_v1 | EPF10K20TC144-3 | 910 | 78% | 9,07MHz |
| dct_v2 | EPF10K30ETC144-1 | 945 | 54 % | 24.5MHz |
| dct_v3 | EPF10K30ETC144-1 | 954 | 55 % | 23.36 MHz |

Foi implementada uma segunda arquitetura, *dct_v2*, sintetizada em um EPF10K30ETC144-1 e ocupou 945 células lógicas ou 54% do *FPGA*. A máxima freqüência de operação foi de 24,5MHz. As memórias neste caso foram colocadas dentro do *FPGA*. Após o multiplicador, a largura dos dados foi ajustada para 16 bits. Após o registrador *St1*, foram acrescentados dois bits à parte inteira do resultado, para permitir o cálculo com dados que têm parte inteira e que são produzidos ao fazer o processo de acumulação. Ao mesmo tempo, a parte decimal foi ajustada para poder manter a largura de dados em 16 bits. O multiplexador e o acumulador foram agora implementados para 16 bits. A figura 7.19 mostra o erro relativo entre o circuito *dct_v2* e o modelo em software, para cada um dos 27 parâmetros produzidos pelo circuito de *DCT*. O máximo erro é de 4,9%.

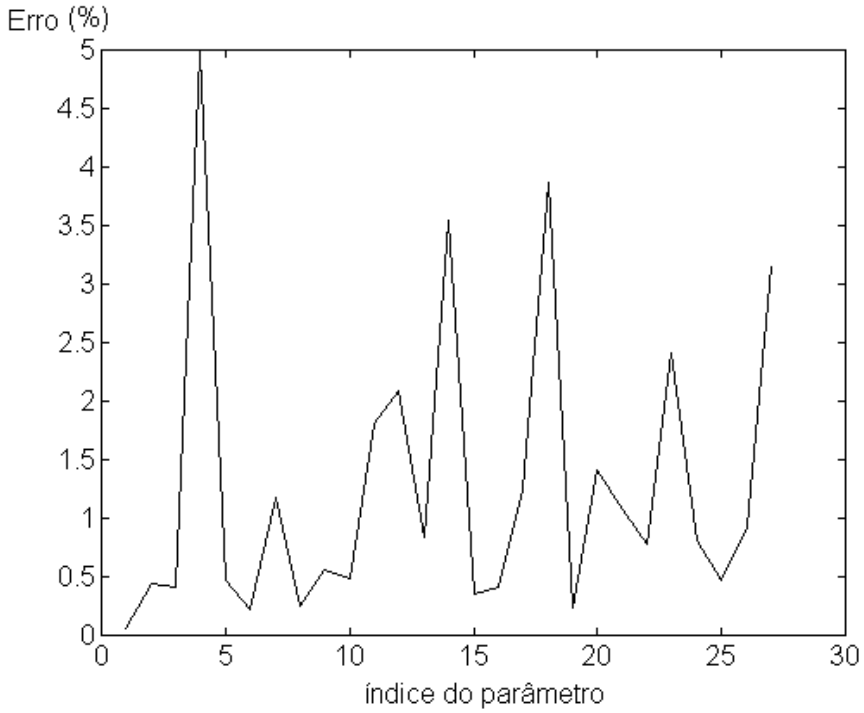



FIGURA 7.19 - Erro relativo entre hardware e software para o circuito *dct_v2*.

O terceiro circuito, *dct_v3*, foi implementado em um EPF10K30ETC144-1, ocupando 954 células lógicas ou 55% do *FPGA*. A máxima freqüência de operação foi

de 23,36MHz. A diferença em relação ao circuito anterior ocorre após o registrador *St1*, a parte decimal não foi ajustada, obtendo-se então 2 bits para a parte inteira e 15 bits para a parte decimal, além do bit de sinal. A palavra final tem uma largura de 18 bits, portanto, o multiplexador e o acumulador são implementados para trabalhar com18 bits. A saída da função de *DCT* é ajustada para 16 bits, e é com esta largura que é enviada para o processo de quantização. A figura 7.20 mostra o erro relativo entre o circuito *dct_v3* e o modelo em software, para cada um dos 27 parâmetros produzidos pelo circuito de *DCT*. O máximo erro é de 1,45%.

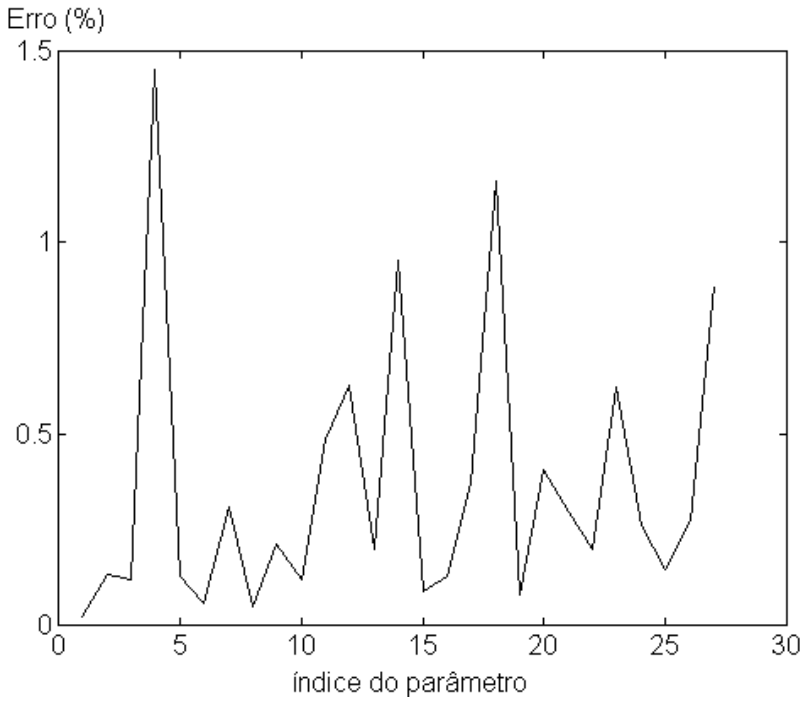



FIGURA 7.20 - Erro relativo entre hardware e software para o circuito *dct_v3*.

As funções anteriormente descritas foram unidas para formar o subsistema de extração de parâmetros. Foram necessários dois FPGAs, um para o circuito de FFT e outro para as demais funções. A tabela 7.6 mostra o resumo da síntese destes dois blocos.

TABELA 7.6 - Síntese da FFT e da função de extração de parâmetros mel-cepstrais.

| **Circuito** | **Dispositivo** | **Células Lógicas Utilizadas** | **% de Utilização** | **Freqüência máxima** |
|---|---|---|---|---|
| FFT | EP1K50FC484-1 | 1776 | 61% | 16,72MHz |
| mel_ceps | EP1K100QC208-1 | 2604 | 52 % | 13,26MHz |

# 8 Subsistema de Reconhecimento

O subsistema de reconhecimento constitui a última etapa do sistema de *RAV*. Após extrair os parâmetros mel-cepstrais, $c_m$, de cada quadro do sinal de voz, foi gerado um conjunto de vetores, de parâmetros mel-cepstrais, $MFCC^i$:

$$MFCC^i = \left(c_0^i, c_1^i, ..., c_M^i\right) \quad , \qquad i = 0,...,T-1 \tag{8.1}$$

onde o índice $i$ indica o quadro ao qual corresponde o vetor de parâmetros mel-cepstrais. $T$ é o número de quadros da elocução e $M$ é o número de parâmetros mel-cepstrais para cada quadro.

A figura 8.1 ilustra como os vetores $MFCC^i$ são quantizados vetorialmente. Nesta mesma figura, a separação em quadros é mostrada sem superposição, para facilitar a visualização. O processo de quantização vetorial foi explicado no capítulo 1. A quantização mapeia os vetores $MFCC^i$**,** gerando um conjunto de vetores-código. Após a quantização, cada repetição de uma elocução será representada por uma seqüência **O** de índices do código de quantização (*codebook*). Assim, uma palavra que era representada por uma seqüência de vetores de parâmetros é transformada em uma seqüência de números inteiros (índices do dicionário da quantização).

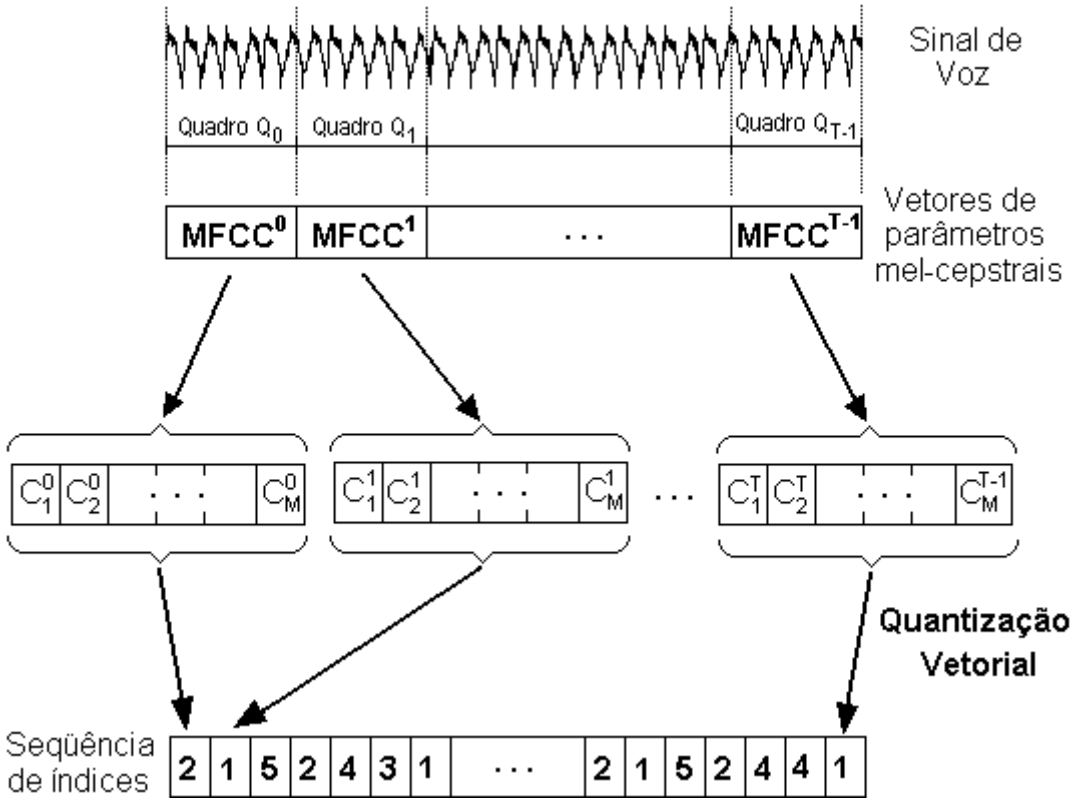



FIGURA 8.1 - Quantização dos vetores de parâmetros mel-cepstrais.

O processo de quantização vetorial foi modelado em software. Sua implementação em hardware não é o objetivo deste trabalho, devido à dimensão do mesmo.

Uma vez que os vetores de parâmetros do sinal de voz foram quantizados vetorialmente, procede-se à identificação da seqüência de índices. A figura 8.2 ilustra este processo. São calculadas as probabilidades, $P(\mathbf{O}|\lambda_1)$ ... $P(\mathbf{O}|\lambda_v)$, de a seqüência O ter sido gerada por cada um dos modelos, $\lambda_1$ ... $\lambda_v$, existentes no sistema. Depois disso, é calculada a máxima probabilidade, a qual indicará o modelo que corresponde 'a seqüência **O** de índices.

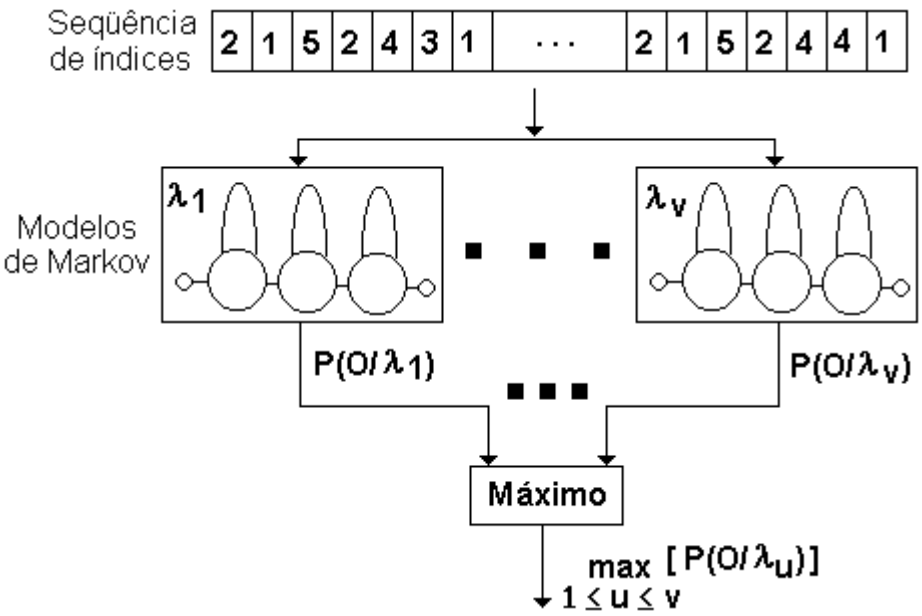



FIGURA 8.2 - Identificação da seqüência.

O cálculo da máxima probabilidade de um modelo ter produzido a seqüência **O**, é realizado através do algoritmo de Viterbi, como explicado no capítulo 2.

O subsistema de reconhecimento está constituído por um decodificador de Viterbi, implementado em *FPGA*, que faz o cálculo da máxima probabilidade para cada seqüência.

# 8.1 Decodificador de Viterbi

Diversas simplificações podem ser feitas no algoritmo de decodificação de Viterbi, ao utilizar modelos *left-right* do tipo mostrado na figura 8.3.

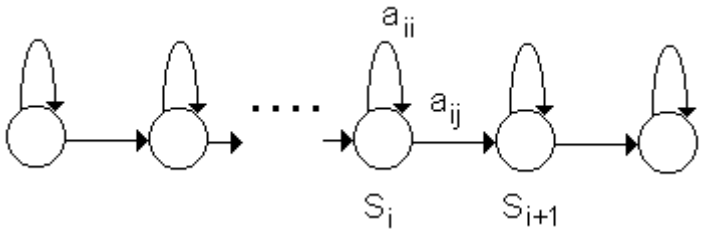



FIGURA 8.3 - HMM left-right utilizado.

O estado inicial é também o primeiro estado $S_i=1$ e o estado que segue será ele mesmo ou o estado $S_i=2$. Se o estado atual é $S_i$ e o estado seguinte é $S_j$, então o valor de

$a_{ij}$ será diferente de zero unicamente nos casos em que $S_j=S_i$ ou $S_j=S_{i+1}$, ou melhor, dito quando ocorre uma transição para o mesmo estado ou uma transição para o estado adjacente. Portanto, a matriz de probabilidades de transição $\mathbf{A_{ij}}$, é especificada da seguinte maneira:

$$\mathbf{A_{ij}} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & 0 & ... & ... & 0 & 0 \\ 0 & 0 & 0 & a_{ii} & a_{ii+1} & 0 \\ 0 & 0 & 0 & 0 & ... & .... \\ 0 & 0 & 0 & 0 & 0 & a_{NN} \end{bmatrix} \tag{8.2}$$

O cálculo de caminhos, mostrado no capítulo 2, utilizou a seguinte equação:

$$\delta_{t+1}(j) = \{\underset{i}{máx}[\delta_t(i).a_{ij}]\}.b_j(o_{t+1}) \tag{8.3}$$

onde $\delta_t(i)$ define a máxima probabilidade ao longo de um caminho, que termina no estado $i$, estando no tempo $t$ e produzindo as primeiras $t$ observações. Esta equação pode ser simplificada, considerando a topologia definida pela equação 8.2, onde existem unicamente probabilidades de transição do tipo $a_{ii}$ e $a_{ii+1}$, onde $i=1...N$, sendo N o número de estados. A equação 8.3 é reduzida à seguinte equação:

$$\delta_{t+1}(i) = máx\{\delta_t(i)\cdot a_{ii}, \delta_t(i+1)\cdot a_{ii+1}\}\cdot b_i(o_{t+1}) \tag{8.4}$$

onde se calculam apenas as probabilidades através de dois caminhos $\delta_t(i)$ e $\delta_t(i+1)$, pois alternativas de transição de estado não existem, em função da topologia dos modelos *left-right*.

Com o objetivo de minimizar a quantidade de hardware necessário para implementar o subsistema de reconhecimento, utiliza-se o logaritmo das probabilidades. Então a equação anterior será modificada da seguinte maneira:

$$\tilde{\delta}_t(i) = log(\delta_t(i)) = máx\left\{\tilde{\delta}_{t-1}(i) + \tilde{a}_{ii}, \tilde{\delta}_{t-1}(i+1) + \tilde{a}_{ii+1}\right\} + \tilde{b}_i(O_t) \tag{8.5}$$

Portanto, todas as multiplicações são reduzidas a somas. A carga computacional principal reside na equação de recursão (8.5) e consiste de $2.N.T$ operações de cálculo do mínimo e $3.N.T$ operações de soma, onde $N$ é o número de estados e $T$ é o número de quadros processados. Sem as simplificações mencionadas, seriam necessárias $N^2T$ operações soma e minimiza.

Visto que a matriz de probabilidades de transição e o vetor de probabilidades iniciais contêm diversos valores zero, ao utilizar o logaritmo das probabilidades é necessário definir uma representação para o valor: $log(0)=-\infty$. Uma solução simples encontrada foi definir este valor como o menor valor negativo representável em ponto fixo e 16 bits.

Para uma implementação em ponto-fixo, é também necessário utilizar um procedimento de escalonamento para manter limitado o intervalo das verossimilhanças acumuladas $\delta_t(i)$.

Para efeitos de teste, utilizaram-se 4 estados e memórias internas ao *FPGA*. Isto permitiu a verificação e comparação da implementação em relação ao modelo em software [GOM 99a] [GOM 99c] [GOM 99e]. No entanto, o circuito final utiliza memórias externas, para reduzir o número de células ocupadas e assim o tamanho do FPGA. Além disso, utilizaram-se 8 estados no circuito final, em lugar de 4, visto que na análise da seção 4.2 este número mostrou ser mais apropriado.

A quantização vetorial utilizou um dicionário (codebook) de 128 valores, pois este mostrou ser o tamanho mais apropriado para o trabalho com palavras isoladas, conforme mostrado na figura 4.3 do capítulo 4.

Foram realizadas 3 implementações:

(1) Uma implementação comportamental (viterbi bhvsd), que consiste em uma descrição comportamental do algoritmo de Viterbi, sintetizado em FPGA. Foi sintetizado em um EPF10K100GC503-3, ocupando 297 células lógicas ou 5% do total de células do FPGA. Nenhuma simplificação foi realizada, precisando 4600 pulsos para processar uma palavra de 100 quadros.

(2) Uma implementação RTL (viterbi rtl), consiste em uma descrição estrutural do algoritmo de Viterbi. Foi sintetizado em um EPF10K50RC240-3, ocupando 133 células lógicas ou 4% do total de células do FPGA e precisando 1025 pulsos para processar uma palavra de 100 quadros.

(3) Uma implementação RTL com a parte de controle simplificada (viterbi rtl-s). Para uma palavra de 100 quadros, precisou 410 pulsos de relógio e foi sintetizado em um EPF10K30ETC144-1, ocupando 138 células lógicas ou 7% do total do FPGA.

Na tabela 8.1, o desempenho destes 3 circuitos é mostrado, quantizado com parâmetros mel-cepstrais. O melhor desempenho foi obtido para o terceiro circuito.

TABELA 8.1 - Comparação entre subsistemas de reconhecimento.

| **Circuito** | **Dispositivo** | **Células** | **% Ocupação** | **Tempo de latência** |
|---|---|---|---|---|
| viterbi bhvsd | EPF10K100GC503-3 | 297 | 5 | 4600 pulsos |
| viterbi rtl | EPF10K50RC240-3 | 133 | 4 | 1025 pulsos |
| viterbi rtl-s | EPF10K30ETC144-1 | 138 | 7 | 410 pulsos |

Na figura 8.4, a parte operativa do subsistema de reconhecimento é mostrada, utilizando a implementação viterbi rtl-s. A probabilidade de transição foi armazenada em duas memórias RAM:

(1) *logaii_ram*, memória de 8 palavras e 16 bits, que armazena as probabilidades de transição para o mesmo estado. Em outras palavras armazena as 8 probabilidades $a_{ii}$, onde i indica o estado atual.

(2) *logaij_ram*, memória também de 8 palavras e 16 bits, que armazena as probabilidades de transição ao estado adjacente $a_{ii+1}$. Visto que o modelo utilizado é do tipo left-right, ao se alcançar o ultimo estado haverá unicamente transição para o mesmo estado, Isto significa que o último valor é 0. Para simplificar o endereçamento, as probabilidades foram escritas na memória *logaij_ram*, a partir da segunda posição de memória.

Esta estratégia reduz o tamanho dos registros de endereços e permite a leitura destas 2 memórias no mesmo ciclo de relógio e com um único registro de endereços.

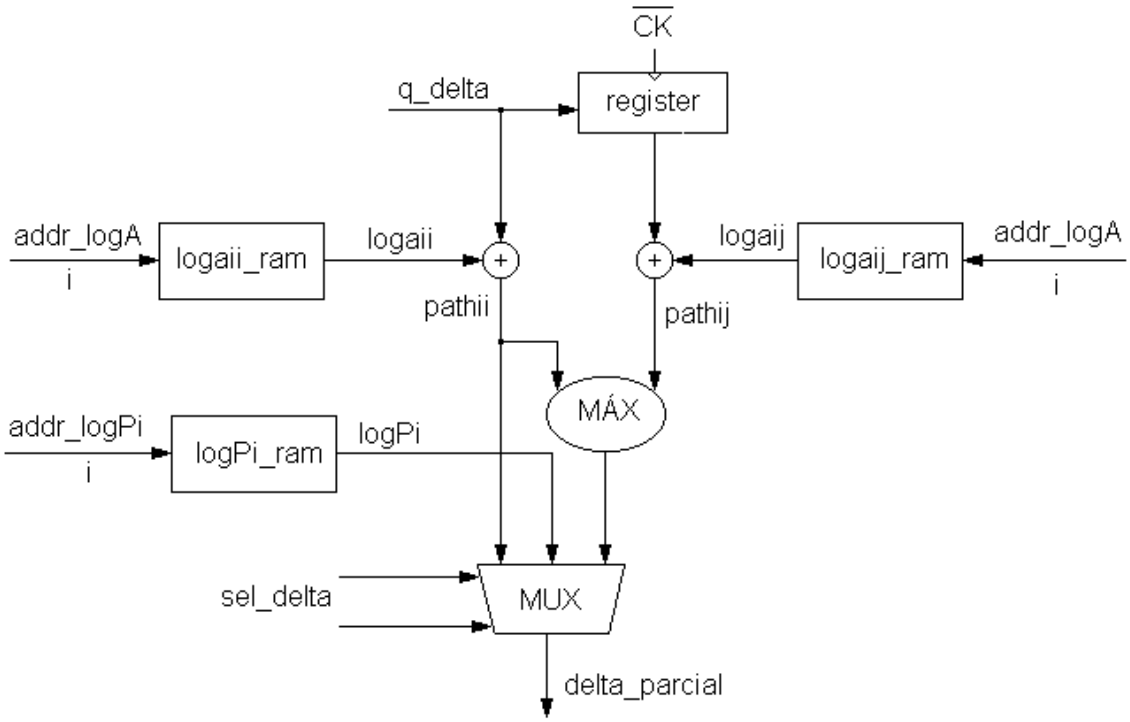



FIGURA 8.4 - Parte operativa do subsistema de reconhecimento.

Uma memória *RAM delta_ram* de 8 palavras e 16 bits foi utilizada para armazenar os valores de $\delta_t(i)$. Um multiplexador reduz o número de somadores para o cálculo $\delta_t(i)$. Em vista do uso de modelos *left-right*, foi feita uma simplificação, evitando o uso de uma memória RAM adicional para armazenar os valores de $\delta_{t-1}(i)$, sendo necessário unicamente um registrador.

Uma memória *RAM* de 1024 palavras e 16 bits é utilizada para armazenar as probabilidades de observação, $b_j(k)$, outra *RAM*, *logPi_ram* de 8 palavras e 16 bits, é utilizada para armazenar os valores $\pi_i$, e outra para armazenar a seqüência de observações (*ram_obs*).

A figura 8.5 mostra o circuito para o endereçamento das memórias que fazem parte do subsistema de reconhecimento.

Quando utilizado para identificar uma seqüência de 100 vetores de parâmetros mel-cepstrais, quantizados, correspondentes à palavra primeiro, o subsistema de

reconhecimento viterbi rtl-s, utilizou 41μs. para calcular a probabilidade de observação desta palavra.

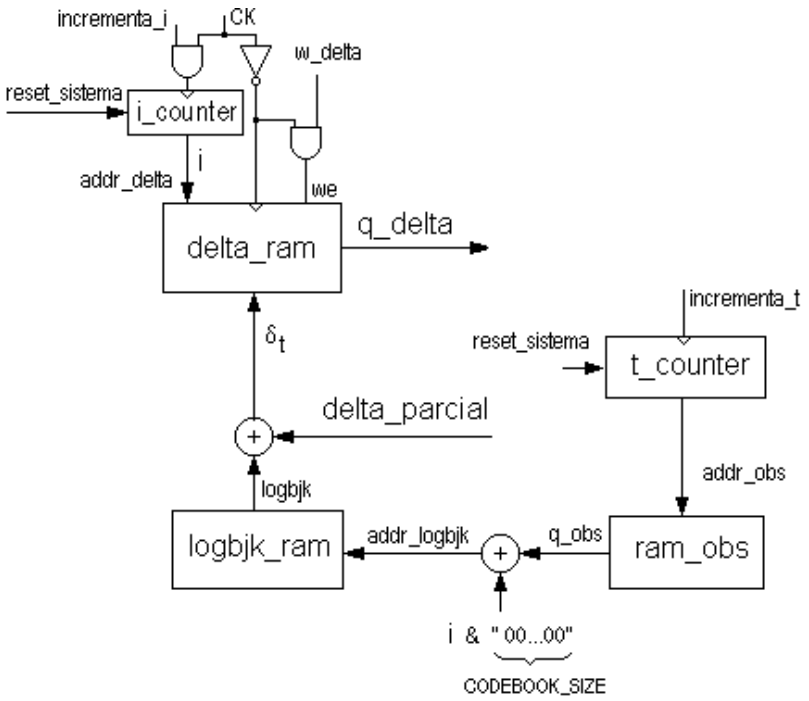



FIGURA 8.5 - Endereçamento do subsistema de reconhecimento.

A figura 8.6, mostra a simulação deste subsistema. A simulação do circuito produziu bons resultados. Os testes foram realizados utilizando a base de dados *REVOX* mencionada no capítulo 4.

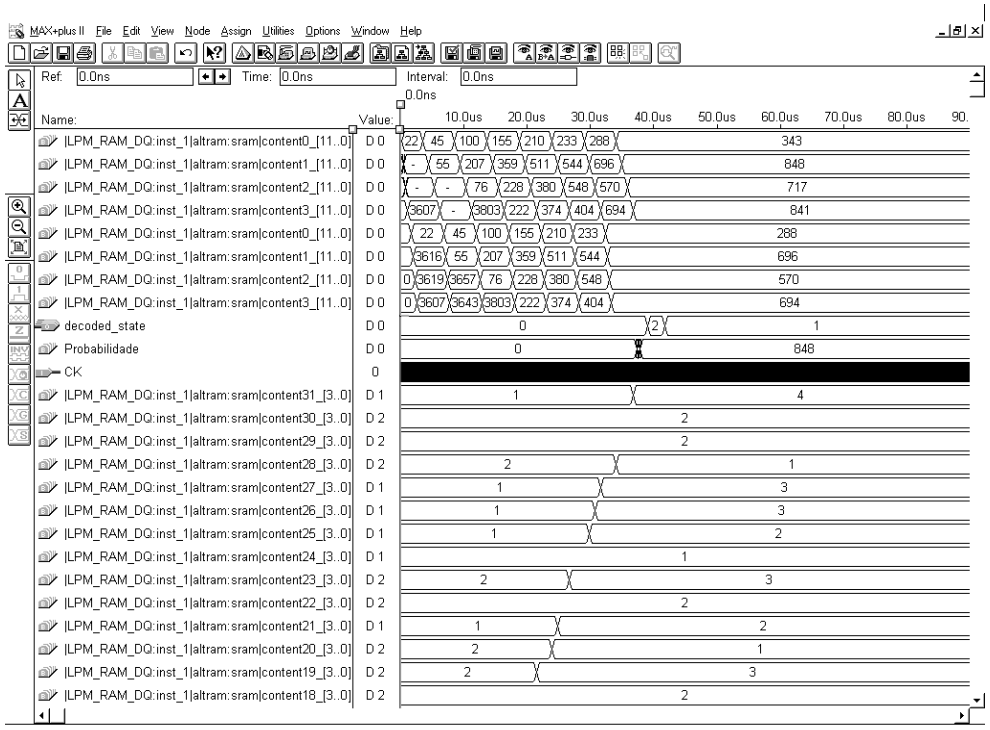



FIGURA 8.6 - Simulação do subsistema de reconhecimento.

## 8.2 União dos Subsistemas

Cada um dos subsistemas, descritos anteriormente, foram testados isoladamente e em conjunto. A figura 8.7 mostra a conexão entre os diversos subsistemas. O controle sincroniza a comunicação entre os subsistemas, cada qual com sua própria memória. Para testar todos os subsistemas operando em conjunto, a etapa de quantização vetorial foi implementada em software. A comunicação do bloco de quantização, com os outros blocos, também foi modelada em software. A saída do subsistema de reconhecimento, *q_delta* é introduzida em um circuito de maximização com um registrador a fim de calcular o modelo que produz a máxima probabilidade (*best_word*) para todos os HMMs.

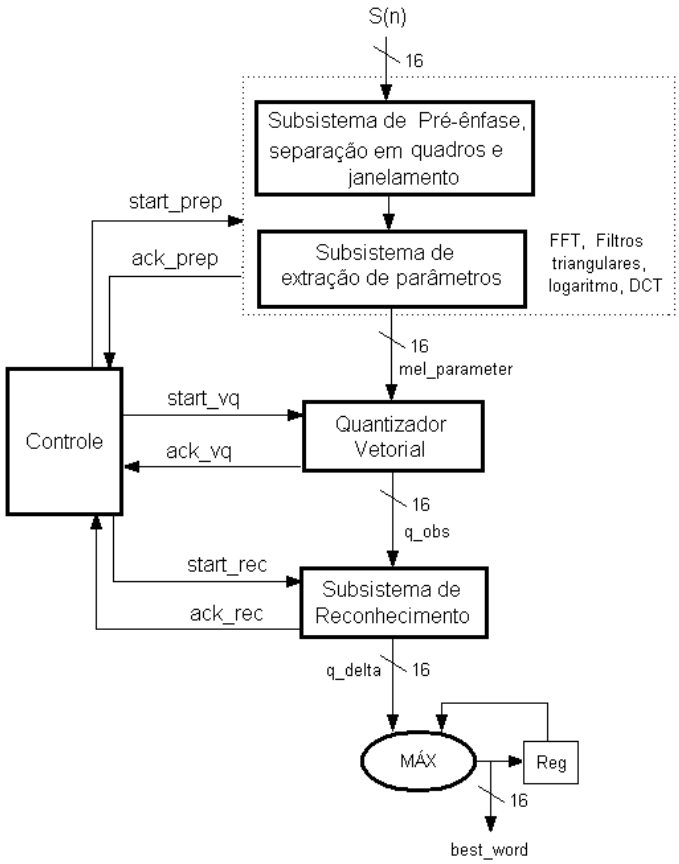



FIGURA 8.7 – União dos subsistemas.

A tabela 8.2 mostra o tempo de resposta dos subsistemas implementados em hardware comparados com seus correspondentes em software. É observada a grande vantagem quanto ao tempo do sistema em hardware.

TABELA 8.2 – Tempo de resposta.

| | **HW** | **SW** |
|---|---|---|
| **Etapa** | **Tempo (μs)** | **Tempo (μs)** |
| Pré-processamento (Subsistema de pré-ênfase e extração de parâmetros) | 2085 | 110000 |
| Reconhecimento | 3,75 | 50000 |

# 9 Conclusões

Em se tratando de sistemas de *RAV,* refere-se geralmente aos sistemas de software. Uma alternativa atraente (mas pouco explorada) é a utilização de circuitos específicos ou hardware dedicado que elimine a necessidade de utilizar um computador pessoal.

A pouca informação existente quanto a sistemas de hardware dedicado baseados em *HMMs* e suas taxas de reconhecimento são aspectos que motivaram esta tese.

Na primeira parte deste trabalho, foram analisadas e revisadas as tarefas que os sistemas de *RAV* utilizam. Além disso, foram consideradas as técnicas mais utilizadas nos sistemas de *RAV* atuais.

Foram apresentadas as características dos sistemas de *RAV* em hardware que tornam este tipo de sistemas em uma nova solução para o problema de *RAV*. A partir da análise de alguns circuitos mais relevantes, encontrados na literatura, observou-se a pouca informação e trabalho neste tipo de sistemas, decorrente da falta de trabalhos que reúnam pesquisa dentro das áreas de RAV e projeto de circuitos integrados.

Deu-se especial ênfase aos Modelos Ocultos de Markov como uma técnica de identificação de padrões de voz. Foi feita uma comparação entre os *HMMs* e outras técnicas, mostrando a vantagem quando utilizada com vocabulários grandes. Outra vantagem dos *HMMs* é que eles permitem o trabalho com conjuntos de subpalavras. Foi mostrado que a taxa de reconhecimento dos *HMMs* é superior ao das outras técnicas apenas quando se trata de sistemas de vocabulário grande e/ou linguagem contínua.

Foi proposto um sistema de *RAV* em hardware. Este trabalho apresentou desenvolvimento de um conjunto de circuitos dedicados para o reconhecimento automático de voz, utilizando *FPGAs*. A principal tarefa foi especificar o sistema, seguida pela descrição dos subsistemas em linguagem de alto nível, tendo como alvo a síntese em *FPGA*.

A tarefa de reconhecimento de voz foi dividida em etapas a fim de realizar a sua implementação em hardware. As etapas de pré-processamento e reconhecimento foram implementadas em hardware. A etapas de quantização vetorial assim como a etapa de treinamento dos *HMMs* não foi implementada, em hardware. A etapa de treinamento é feita uma única vez ao definir o vocabulário do sistema e introduzir os parâmetros do modelo de cada palavra. Assim, o tempo de treinamento não é crítico para o funcionamento do sistema de *RAV*. O processo de quantização vetorial é uma etapa importante a ser implementada em *FPGA*. No entanto, o tempo de projeto desta tese não permitiu a implementação em hardware desta etapa. Um *FPGA* para a quantização vetorial constitui um dos trabalhos futuros a ser realizado.

Foi desenvolvido um novo algoritmo para a realização em tempo real da separação em quadros. Este algoritmo foi posteriormente implementado em hardware. Foi descrito cada um dos subsistemas que compõem o sistema proposto. O subsistema de pré-ênfase, separação em quadros e janelamento foi implementado em um único *FPGA*. O subsistema de extração de parâmetros e o subsistema de reconhecimento

também foram implementados em outro *FPGA*. A etapa de reconhecimento foi colocada em outro *FPGA*.

A memória consumida nas operações de *RAV* constituiu um fator crítico. Ao implementar os sistemas de *RAV* em hardware, a grande quantidade de área consumida pertence aos bancos de memória utilizados.

A melhor solução foi utilizar memórias externas. Esta solução implica no aumento do tamanho físico do sistema, mas necessária por causa da inviabilidade de implementação dentro de um único circuito FPGA.

Utilizaram-se bancos separados de memória, cada um dos quais é acessado pelo seu próprio barramento e pode ser lido ou escrito durante cada ciclo de relógio. Foi utilizado hardware dedicado para o cálculo dos endereços de memória. As unidades que geram tais endereços trabalham em paralelo com as principais unidades de processamento, permitindo o acesso aos dados em novas posições de memória sem parar para calcular um novo endereço.

Inicialmente, o teste de tais circuitos realizado separadamente mostrou o correto funcionamento dos mesmos, quando comparados com os correspondentes modelos em software. Também se realizou o teste dentro de um sistema e para isto foi necessário utilizar alguns módulos de software que modelem as etapas não implementadas. O teste foi feito com um vocabulário para o controle de elevadores, formado pelas palavras primeiro,..., décimo.

Um problema que surgiu na implementação em hardware das técnicas de *RAV* é a definição das condições de fronteira. Uma solução para isto foi utilizar valores grandes, chamados de infinidades. Tais infinidades são valores que não serão escolhidos nos circuitos de minimização ou maximização dentro dos algoritmos de *RAV,* porém sua especificação é necessária para a implementação de uma lógica de operação simples e funcional.

O sistema proposto constitui um sistema portátil, de baixo peso e consumo. Além disto, o sistema faz uso de operações em paralelo, o que conduz a um aumento do desempenho do processamento. O sistema é capaz de realizar todas as operações necessárias para o reconhecimento de padrões de voz previamente aprendidos. Por fim, o hardware para a realização deste sistema pode vir a ser o esqueleto para uma placa de prototipação rápida onde podem ser desenvolvidos diversos algoritmos para o processamento de voz em *VHDL* e estes servirem como uma base de biblioteca para futuros trabalhos em hardware ou hardware/software codesign.

As características de velocidade de um circuito dedicado, o espaço físico ocupado, a flexibilidade da descrição em *VHDL* e o potencial dos sistemas em *FPGA* tornam esta proposta útil para o desenvolvimento de novas aplicações. O sistema proposto será utilizado em problemas que exigem um pequeno vocabulário e número de locutores limitado.

Otimizações de potência no nível comportamental se apresentam como possíveis trabalhos. Um circuito assim pode ser utilizado na experimentação com diversos aspectos da tecnologia de *RAV*.

Para aumentar a capacidade de processamento pode-se implementar a fase de treinamento dos *HMMs* de modo a melhorar o desempenho e com isto permitir o treinamento on-line do circuito. Algo similar poderá ser realizado com a etapa de quantização vetorial.

Com certeza, em um futuro próximo será possível ter uma implementação do sistema como um *SOC* (***S****ystem* ***O****n* ***C****hip*), ou utilizar microprocessadores *DSP* com parte reconfigurável.

# Bibliografia


[ACK 85] ACKENHUSEN, J. G. et al. Single-Chip Implementation of Feature Measurement for LPC-Based Speech Recognition. **AT&T Technical Journal**, USA, v.64, n.8, p.1787-1805, Oct. 1985.

[ACK 86] ACKENHUSEN, J. G. et al. Single-Board General-Purpose Speech Recognition System. **AT&T Technical Journal**, USA, v.65 n.5, p. 48-59, Sept/Oct. 1986.

[ALA 98] AL-AKAIDI, M. A Simulation Support in the Search for an Efficient Speech Coding. **Simulation**, California, v.70, n.5, p. 279-290, May 1998.

[BAK 75] BAKER, J. The Dragon System-An Overview. **IEEE Transactions on Acoustics, Speech and Signal Processing**, New York, v.23, n.1, p. 24-29, Feb. 1975.

[BEN 96] BENGIO, Y. **Neural Networks for Speech and Sequence Recognition**. New York: Thomson Computer Press, 1996. 167p.

[BOR 98] BORGATTI, M. et al. A Low-Power Integrated Circuit for Remote Speech Recognition. **IEEE Journal of Solid-State Circuits**, New York, v.33, n.7, p. 1082-1089, July 1998.

[BOU 98] BOU-GHAZALE, S. E.; HANSEN, J. H. L. *HMM*-Based Stressed Speech Modeling with Application to Improved Synthesis and Recognition of Isolated Speech Under Stress. **IEEE Transactions on Speech e Audio Processing**, New York, v.6, n.3, p. 201-216, May 1998.

[BRO 84] BROWN, M. K. et al. The DTWP: An LPC-Based Dynamic Time-Warping Processor for Isolated Word Recognition. **AT&T Bell Laboratories Technical Journal**, USA, v.63, n.3, p.441-457, Mar. 1984.

[BUR 96] BURSTEIN, A. J. **Speech Recognition for Portable Multimedia Terminals**. Berkeley: University of California, 1996. Ph. D. Thesis.

[CAM 97] CAMPBELL, J. Speaker Recognition: A Tutorial. **Proceedings of the IEEE,** New York, v.85, n.9, p. 1436-1462, Sept. 1997.

[CAS 97] CASAGRANDE, R.; CABRAL, E. Time-Delay Neural Network Applied to Speaker Recognition. In: CONGRESSO BRASILEIRO DE REDES NEURAIS, 3., 1997, Florianópolis. **Proceedings...** Florianópolis: SBC, 1997. p.319-323.

[CAU 97] CAUWENBERGHS, G.; PEDRONI, V. A Low-Power CMOS Analog Vector Quantizer. **IEEE Journal of Solid State Circuits**, New York, v.32, n.8, p. 1278-1283, Aug. 1997.

[CHO 96] CHONG, J.; TOGNERI, R. Speaker Independent Recognition of Small Vocabulary. In: AUSTRALIAN INTERNATIONAL CONFERENCE ON SPEECH SCIENCE AND TECHNOLOGY, 6., 1996, Adelaide, Australia. **Proceedings...** Adelaide: [s.n.], 1996. p.133-137.

[DAV 80] DAVIS, S. B.; MERMELSTEIN, P. Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. **IEEE Transactions on Acoustics, Speech, and Signal Processing**, New York, v.28, n.4, p. 357-366, Aug. 1980.

[DEL 93] DELLER, J. R.; PROAKIS, J. G.; HANSEN, J. H. L. **Discrete-Time Processing of Speech Signals**. New Jersey: Prentice Hall, 1993. 908p.

[DEN 85] DENYER, P.; RENSHAW, D**. VLSI Signal Processing - A Bit Serial Approach.** New York: Addison Wesley, 1985. 312p.

[DES 97] DESHMUKH, N. et al. Benchmarking Human Performance for Continuous Speech Recognition. In: IEEE SOUTHEASTCON, 1997, Virginia, USA. **Proceedings...** Virginia: IEEE, 1997. p.97-99.

[DOD 85] DODDINGTON, G. Speaker Recognition-Identifying People by their Voices. **Proceedings of the IEEE**, New York, v.73, n.11, p. 1651-1664, Nov. 1985.

[DOS 95] DOS SANTOS, S. B.; ALCAIM, A. **Fundamentos de Reconhecimento de Voz**. Rio de Janeiro: Centro de Estudos em Telecomunicações da Pontifícia Universidade Católica do Rio de Janeiro, 1995. 88p. (Relatório Técnico, n. CETUC-DID-01/95).

[DUG 96] DUGAD, R.; DESAI, U. B. **A Tutorial on Hidden Markov Models**. Bombay: Signal Processing and Artificial Neural Networks Laboratory, Indian Institute of Technology, 1996. 16p. (Technical Report n. SPANN-96.1).

[EDW 97] EDWARDS, J. Voice-Based Interfaces Make PCs Better Listeners. **Computer**, Los Alamitos, CA, v. 30 , n.8, p. 14-16, Aug. 1997.

[FAR 94] FARREL, K. R. et al. Speaker Recognition Using Neural Networks and Conventional Classifiers. **IEEE Transactions on Acoustics, Speech, and Signal Processing**, New York, v. 2, n.1, p. 194-205, Jan. 1994.

[FER 94] FERNÁNDEZ, J. M. et al. A Flexible VLSI-Based Hardware System for Medium-Large Vocabulary Real-Time Speech Recognition. **Microprocessing and Microprogramming**, Amsterdam, v.40, p. 825-828, 1994.

[FOR 73] FORNEY, G. D. The Viterbi Algorithm. **Proceedings of the IEEE**, New York, v.61, n.3, p. 268-278, Mar. 1973.

[FOS 93] FOSTER, P.; SCHALK, T. B. **Speech Recognition. The Complete Practical Reference Guide**. New York: Telecom Library, 1993. 377p.

[FOS 98] FOSLER-LUSSIER, E. **Markov Models and Hidden Markov Models**: A Brief Tutorial. Berkeley: International Computer Science Institute, University of California, 1998. 7p. (Technical Report n.98-041).

[FRI 96] FRITSCH, J. **Modular neural networks for Speech Recognition**. 1996. Tese (Doutorado) - Carnegie Mellon University, Pittsburgh.

[GLI 87] GLINSKI, S. et al. The Graph Search Machine (GSM): A *VLSI* Architecture for Connected Speech Recognition and Other Applications. **Proceedings of the IEEE**, New York, v.75, n.9, p. 1172-1184, Sept. 1987.

[GOM 98] GÓMEZ CIPRIANO, J. L. et al. Speech Recognition Systems. In: MICROELECTRONICS SEMINAR, 13., 1998, Bento Gonçalves. **Proceedings...** Bento Gonçalves: SBC, 1998. p. 189-192.

[GOM 99] GÓMEZ CIPRIANO, J. L. **Modelos Ocultos de Markov para Sistemas de Reconhecimento Automático de Voz**. 1999. 89p. Exame de Qualificação (PPGC) – Universidade Federal do Rio Grande do Sul, Porto Alegre.

[GOM 99a] GÓMEZ CIPRIANO, J. L. et al. Design of a Hidden Markov Model State Decoding System. In: WORKSHOP IBERCHIP, 5., 1999, Lima-Perú. **Proceedings...** Lima: Iberchip, 1999. p. 302-307.

[GOM 99b] GÓMEZ CIPRIANO, J. L. et al. Hidden Markov Models for Speech Recognition. In: MICROELECTRONICS SEMINAR, 14., 1999, Pelotas. **Proceedings...** Pelotas: SBC, 1999. p. 117-120.

[GOM 99c] GÓMEZ CIPRIANO, J. L. et al. Design of a Real-Time Hidden Markov Model State Decoding System. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN-CACIC 1999, Tandil-Argentina. **Proceedings...** Tandil: UNCPBA, 1999. CD-ROM.

[GOM 99d] GÓMEZ-CIPRIANO, J. .; BARONE, D. Introdução ao Reconhecimento de Voz. In: ESCOLA REGIONAL DE INFORMÁTICA, 1999, Salvador, Bahia. **Cursos...** Salvador: SBC, 1999.

[GOM 99e] GÓMEZ CIPRIANO, J. L. et al. Design of a Reconfigurable State Decoding System for Hidden Markov Models. In: INTERNATIONAL CONFERENCE ON ELECTRICAL AND ELECTRONICS ENGINEERING-ELECO 1999, Bursa-Turquia. **Proceedings...** Bursa: The Chamber of Turkish Electrical Engineers, 1999.

[GOM 2000] GÓMEZ-CIPRIANO, J. L. et al. Classification Methods in Vector Quantization for Speech Recognition In: MICROELECTRONICS SEMINAR, 15., 2000, Torres. **Proceedings...** Torres: SBC, 2000. p. 103-106.

[GON 94] GONG, Y. et al. **Issues in Acoustic Modeling for Automatic Speech Recognition**. Paris: INRIA, 1994. 15p. (Technical Report n. 2368).

[HUA 90] HUANG, X. D.; ARIKI, Y.; JACK, M. A. **Hidden Markov Models for Speech Recognition**. Edinburgh: Edinburgh University Press, 1990. 276p.

[JUA 84] JUANG, B. H. On the Hidden Markov Model and Dynamic Time Warping for Speech Recognition – A Unified View. **AT&T Bell Laboratories Technical Journal**, USA, v.63 n.7, p. 1213-1243, Sept. 1984.

[KAV 87] KAVALER, R. A. et al. A Dynamic-Time-Warp Integrated Circuit for a 1000-Word Speech Recognition System. **IEEE Journal of Solid-State Circuits**, New York, v. sc-22, n.1, p. 3-14, Feb. 1987.

[LAP 97] LAPSLEY, P. et al. **DSP Processor Fundamentals**: architectures and features. New Jersey: IEEE Press, 1997. 210p.

[LAZ 97] LAZZARO, J. et al. A Micropower Analog Circuit Implementation of Hidden Markov Model State Decoding. **IEEE Journal of Solid-State Circuits**, New York, v.32, n.8, p. 1200-1209, Aug. 1997.

[LEE 89] LEE, K.- F. **Automatic Speech Recognition - The Development of the SPHINX System**. London: Kluwer Academic Publishers, 1989. 207p.

[LEV 83] LEVINSON, S. E.; RABINER, L. R.; SONDHI, M. M. An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. **The Bell System Technical Journal**, USA, v.62 n.4, p. 1035-1074, Apr. 1983.

[LEV 85] LEVINSON, S. Structural Methods in Automatic Speech Recognition. **Proceedings of the IEEE**, New York, v.73, n.11, p. 1625-1650, Nov. 1985.

[LUF 94] LUFT, J. **Reconhecimento Automático de Voz para palavras isoladas e Independente do Locutor**. 1994. Dissertação (Mestrado-PPGEMM) - Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MAD 95] MADISETTI, V. K. **VLSI Digital Signal Processors**: An Introduction to Rapid Prototyping and Design Synthesis. New Jersey: IEEE Press, 1995. 524p.

[MAK 85] MAKHOUL, J. et al. Vector Quantization in Speech Coding**. Proceedings of the IEEE**, New York, v.73, n.11, p. 1551-1589, Nov. 1985.

[MAM 96] MAMMONE, R. J. et al. Robust Speaker Recognition. A Feature Based Approach. **IEEE Signal Processing Magazine**, New York, v.13, n.5, p. 58-71, Sept. 1996.

[MAR 96] MARKOWITZ, J. A. **Using Speech Recognition**. New Jersey: Prentice-Hall, 1996. 292p.

[MAR 97] MARI, J. -F. et al. Automatic Word Recognition Based on Second Order Hidden Markov Models. **IEEE Transactions on Speech and Audio Processing**, New York, v.5, n.1, p. 22-25, Jan. 1997.

[MAT 94] MATSUI, T.; FURUI, S. Comparison of Text-Independent Speaker Recognition Using VQ-Distortion and Discrete/Continuous HMM's. **IEEE Transactions on Speech and Audio Processing**, New York, v.2, n.3, p. 456-459, July 1994.

[MOR 91] MORGAN, D. P.; SCOFIELD, C. L. **Neural Networks and Speech Processing**. Boston: Kluwer, 1991. 391p.

[MOR 93] MORENO, F. et al. A Flexible Architecture for Real-Time Speech Recognition. **Microprocessing and Microprogramming**, Amsterdam, v. 37, p. 69-72, 1993.

[MOR 95] MORGAN, N.; BOURLARD, H. Continuous Speech Recognition. **IEEE Signal Processing Magazine**, New York, v.12, n.3, p.25-42, May 1995.

[MOR 97] MORAIS, E. et al. Modelo Oculto de Markov Parametrizado Através de uma ANN e sua Aplicação no Reconhecimento de Fala Contínua. In: CONGRESSO BRASILEIRO DE REDES NEURAIS, 3., 1997, Florianópolis, Brasil. **Proceedings...** Florianópolis: SBC, 1997. p.324-329.

[MYE 81] MYERS, C. S. et al. On the Use of Dynamic Time Warping for Word Spotting and Connected Word Recognition. **The Bell System Technical Journal**, New Jersey, v.60, n.3, p. 303-325, Mar. 1981.

[NAR 96] NARAYANASWAMY, S. **Pen and Speech Recognition in the User Interface for Mobile Multimedia Terminals**. 1996. Tese (Doutorado) - University of California, Berkeley.

[NUN 96] NUNES, H. F. **Reconhecimento Automático de Fala Baseado em HMM**. 1996. Dissertação (Mestrado-FEE-DECOM) - Universidade Estadual de Campinas, Campinas.

[OGL 90] OGLESBY, J.; MASON, J. S. Optimization of Neural Models for Speaker Identification. In: INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 1990, Albuquerque. **Proceedings...** Albuquerque: IEEE, 1990. p.261-264.

[PEA 90] PEACOCKE, R. D.; GRAF, D. H. An Introduction to Speech and Speaker Recognition. **Computer Magazine**, New York, v.23, n.8, p. 26-33, Aug. 1990.

[PET 99] PETRY, A. et al. REVOX Voice Recognition System Applied to Industry Control. In: PROTEM-CC-PHASE III – PROJECTS INTERNATIONAL EVALUATION, 2., 1999, Rio de Janeiro. **Proceedings...** Brasília: CNPq, 1999. p.347-376.

[PIC 93] PICONE, J.W. Signal Modeling Techniques in Speech Recognition. **Proceedings of the IEEE**, New York, v.81, n.9, p.1215-1247, Sept. 1993.

[PIC 95] PICONE, J. et al. Automated Speech Understanding: The Next Generation. **Digital Signal Processing Technology**, Texas, v.CR57, p. 101-114, 1995.

[PIH 96] PIHL, J. **Design Automation of High Speed Digital Signal Processing in VLSI with Application in Speech Recognition Systems Based on Hidden Markov Models**. 1996. Tese (Doutorado) - Norwegian Institute of Technology, Noruega.

[PIH 96a] PIHL, J. et al. A VLSI Implementation of Pdf Computations in HMM Based Speech Recognition. In: IEEE TEN CONFERENCE ON DIGITAL SIGNAL PROCESSING APPLICATIONS, TENCON, 1996, Australia. **Proceedings...** Australia: IEEE, 1996. p.241-246.

[PIT 96] PITTON, J. W. et al. Time Frequency Analysis and Auditory Modeling for Automatic Speech Recognition. **Proceedings of the IEEE**, New York, v.84, n.9, p.1199-1215, Sept. 1996.

[RAB 78] RABINER, L. R.; SCHAFER, R. W. **Digital Processing of Speech Signals**. New Jersey: Prentice-Hall, 1978. 512p.

[RAB 82] RABINER, L. R. et al. On the Application of Vector Quantization and Hidden Markov Models to Speaker-Independent, Isolated Word Recognition. **The Bell System Technical Journal**, New Jersey, v.62, n.4, p.1075-1105, Apr. 1983.

[RAB 89] RABINER, L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. **Proceedings of the IEEE**, New York, v.77, n.2, 257-286, Feb. 1989.

[RAB 93] RABINER, L.; JUANG, B. - H. **Fundamentals of Speech Recognition**. New Jersey: Prentice Hall, 1993. 507p.

[RAV 96] RAVISHANKAR, M. K. **Efficient Algorithms for Speech Recognition**. 1996. Tese (Doutorado) - Carnegie Mellon University, Pittsburgh.

[REV 98] REVOX: Sistema de Reconhecimento de Voz Aplicado ao Controle Industrial. Porto Alegre: Instituto de Informática da UFRGS, 1998. 123p. Relatório Final.

[RYA 93] RYAN, M. S. **The Viterbi Algorithm**. England: Department of Computer Science of Warwick, 1993, 17p. (Technical Report RR238).

[SAK 78] SAKOE. H.; CHIBA, S. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. **IEEE Transactions on Acoustics, Speech and Signal Processing**, New York, v.26, n.1, p. 43-49, Feb. 1978.

[SAN 94] SANDHU, S. **A Comparative Study of Mel Cepstra and EIH for Phone Classification under Adverse Conditions**. 1994. Tese (Mestrado-Department of Electrical Engineering and Computer Science) – Massachusetts Institute of Technology, Massachusetts.

[SCH 2000] SCHRAMM, M. C. et al. A Brazilian Portuguese Language Corpus Development. In: INTERNATIONAL CONFERENCE ON SPEECH AND LANGUAGE PROCESSING, ICSLP, 2000, Beijing. **Proceedings...** Beijing: [S.l.:s.n.], 2000. p.579-582.

[SCH 94] SCHAFER, R. W. Scientific Bases of Human-Machine Communication by Voice. In: ROE, D. B.; WILPON J. G. **Voice Communication Between Humans and Machines**. Washington: National Academic Press, 1994. p.15-33.

[SMY 97] SMYTH, P. et al. Probabilistic Independence Networks for Hidden Markov Probability Models. **Neural Computation**, Massachusetts, v.9, p. 227-269, Sept. 1997.

[SOO 87] SOONG, F. K.; JUANG, B.- H. A Vector Quantization Approach to Speaker Recognition. **AT&T Technical Journal**, New Jersey, v.66, n.2, p. 16-26, Mar./Apr. 1987.

[STR 98] STROPE, B.; ALWAN, A. Robust Word Recognition Using Threaded Spectral Peaks. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, ICASSP, 1998, Seattle. **Proceedings...** Seattle: IEEE, 1998. v.3, p.625-629.

[TEB 95] TEBELSKIS, J. **Speech Recognition Using Neural Networks**. 1995. Tese (Doutorado) - Carnegie Mellon University, Pittsburgh.

[TUR 98] TURIN, W. Unidirectional and Parallel Baum-Welch Algorithms. **IEEE Transactions on Speech and Audio Processing**, New York, v.6, n.6, p.516-523, Nov. 1998.

[WAN 97] WANG, X. ; POLS, L. C. W. A Preliminary Study About Robust Speech Recognition for A Robotics Application. **Proceedings of the Institute of Phonetics**, Amsterdam, v.21, p.11-20, 1997.

[WEI 92] WEIHMANN, T. **Processamento Digital de Sinais Aplicado à Transmissão de Voz**. 1992. Dissertação (Mestrado-PPGEMM)-Universidade Federal do Rio Grande do Sul, Porto Alegre.

[YOU 94] YOUNG, S. J. **The HTK Hidden Markov Model Toolkit**: Design and Philosophy. Cambridge: Cambridge University Engineering Department, 1994. 27p. (Technical Report TR.152).

[YOU 95] YOUNG, D. P. **A Fuzzy ART MAP Neural Network Speech Recognition System Based on Formant Ratios**. 1995. Tese (Mestrado - Department of Electrical and Computer Engineering) - Royal Military College of Canada, Kingston.

[YOU 96] YOUNG, S. A Review of Large-Vocabulary Continuous-Speech Recognition. **IEEE Signal Processing Magazine**, New York, v.13, n.5, p. 45-57, Sept. 1996.

[YU 95] YU, K.; MASON, J.; OGLESBY, J. Speaker recognition using hidden Markov models, dynamic time warping and vector quantisation. **IEE Proceedings on Vision, Image and Signal Processing**, New York, v. 142, p. 313-318, 1995.

[ZHA 94] ZHANG, Y. et al. Speaker-Independent Isolated Word Recognition Using Multiple Hidden Markov Models. **IEE Proceedings on Vision, Image and Signal Processing**, London, v.141, n.3, p. 197-202, June 1994.

[ZHA 95] ZHANG, Y. **Speaker-Independent Isolated Word Recognition**. 1995. Tese (Doutorado) - University of Western Australia, Crawley.

[ZUE 85] ZUE, W. Z. The Use of Speech Knowledge in Automatic Speech Recognition. **Proceedings of the IEEE**, New York, v.73, n.11, p. 1602-1615, Nov. 1985.