**UNIVERSIDADE DO VALE DO ITAJAÍ**
**CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR**
**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

# IMPLEMENTAÇÃO E COMPARAÇÃO DE ALGORITMOS DE DETECÇÃO DE ATIVIDADE DE VOZ (VAD) EM DSP

Área de Sistemas Embarcados

por

Ronaldo Antunes Borges

Walter Antônio Gontijo, M. Eng
Orientador

São José (SC), junho de 2008

**UNIVERSIDADE DO VALE DO ITAJAÍ**
**CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR**
**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

# IMPLEMENTAÇÃO E COMPARAÇÃO DE ALGORITMOS DE DETECÇÃO DE ATIVIDADE DE VOZ (VAD) EM DSP

Área de Sistemas Embarcados

por

Ronaldo Antunes Borges

Relatório apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Engenharia de Computação para análise e aprovação.
Orientador: Walter Antônio Gontijo, M. Eng

São José (SC), junho de 2008.

# DEDICATÓRIA

*Dedico este trabalho a Deus, meus pais,*
*irmãos e amigos.*

# AGRADECIMENTOS

Primeiramente agradeço a Deus por tornar tudo isso possível, por ter me dado a vida e nunca ter me deixado sozinho. Por ter me abençoado com uma boa família que me deu condições para chegar até aqui.

Agradeço a minha mãe Lurdes por ter me ensinado a valorizar os estudos, por ter se dedicado tanto em me cuidar, esquecendo muitas vezes de seus próprios interesses para que eu pudesse me formar e ter um bom emprego. Palavras são pouco para expressar o quanto eu te amo.

Agradeço ao meu pai Pedro pelos ensinos que me deu para que eu pudesse ser um vencedor, tendo ânimo ao enfrentar os problemas que a vida apresenta, sempre com bom humor e otimismo.

Agradeço ao meu grande amigo Douglas que nunca me negou ajuda em tudo o que precisei, companheiro em grande parte dos trabalhos da faculdade. Também a todos os amigos graduandos em Engenharia da Computação, colegas que foram durante o curso.

Agradeço a todos os professores, que mais do que educadores também foram amigos, me tratando como pessoa e não como um número na lista de chamada. Agradeço especialmente ao professor e meu orientador, Walter, que com muito empenho sempre se dispôs a me ajudar desde o início deste trabalho e obrigado também pela contribuição que deu para a minha vida profissional. Agradeço também a professora Anita pelas orientações normativas, contribuindo para o aprimoramento deste trabalho.

Deixo também o meu agradecimento a todos aqueles que de alguma maneira contribuíram para a minha formação e realização deste trabalho de conclusão de curso.

# SUMÁRIO

# LISTA DE ABREVIATURAS

| | |
|---|---|
| ADC | Analog to Digital Converter |
| ALU | Arithmetic and Logic Unit |
| ANSI | American National Standards Institute |
| API | Application Programming Interface |
| AR | Auxiliary Register |
| ARAU | Auxiliary Register Arithmetic Unit |
| CAB | C Address Bus |
| CB | C Bus |
| CCS | Code Composer Studio |
| CD | Compact Disk |
| CNG | Comfort Noise Generator |
| CODEC | Coder/Decoder |
| CPU | Central Processing Unit |
| CSSU | Compare, Select, and Store Unit |
| DAB | D Address Bus |
| DAC | Digital to Analog Converter |
| DAG | Data Address Generator |
| DB | D Bus |
| DMA | Direct Memory Access |
| DP | Data memory page Pointer |
| DSK | DSP Starter Kit |
| DSP | Digital Signal Processor/Processing |
| DTX | Discontinuous Transmission |
| EAB | E Address Bus |
| EB | E Bus |
| EBD | Energy Based Detector |
| ETSI | European Telecommunications Standards Institute |
| FIR | Finite Impulse Response |
| GSM | Global System for Mobile communications |
| IDE | Integrated Development Environment |
| IEEE | Institute of Electrical and Electronics Engineers |
| IP | Internet Protocol |
| ITU-T | International Telecommunication Union -Telecommunication Standardization Sector |
| LPC | Linear Prediction Coding |
| LSF | Line Spectral Frequencies |
| MIPS | Millions of Instructions Per Second |
| PAB | Program Address Bus |
| PAGEN | Program Address Generation |
| PB | Program Bus |
| PCM | Pulse Code Modulation |
| PMST | Processor Mode Status Register |
| PSTN | Public Switched Telephone Network |
| RAM | Random Access Memory |
| ROM | Read Only Memory |
| SID | Silence Insertion Descriptor |

| | |
|---|---|
| SNR | Signal to Noise Ratio |
| TCP | Transmission Control Protocol |
| TDM | Time Division Multiplexing |
| TI | Texas Instruments |
| UDP | User Datagram Protocol |
| VAD | Voice Activity Detector/Detection |
| VLSI | Very Large Scale Integration |
| VoIP | Voice over Internet Protocol |
| ZCD | Zero Crossing Detector |

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE EQUAÇÕES

# RESUMO

BORGES, Ronaldo Antunes. **Implementação e comparação de algoritmos de Detecção de Atividade de Voz (VAD) em DSP**. São José, 2008. 75 f. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação)–Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, São José, 2008.

Durante uma conversação telefônica, cerca de 60% do tempo o canal fica inativo, isto é, o usuário não está falando. O algoritmo de Detecção de Atividade de Voz (VAD) faz a distinção entre voz (atividade de voz) e silêncio ou ruído ambiente (inatividade de voz) possibilitando aos CODECs que fazem uso do VAD ter uma maior taxa de compressão quando inatividade de voz é detectada. Devido à dificuldade na correta distinção entre atividade e inatividade de voz, vários algoritmos VAD vêm sendo propostos. Neste trabalho foi realizada a implementação de três algoritmos VAD, em linguagem assembly do Processador Digital de Sinais (DSP) TMS320C5410 da Texas Instruments. Posteriormente também foi efetuada uma comparação entre as implementações em assembly onde foram consideradas três características: ocupação de memória, complexidade computacional (mipagem) e desempenho. Os algoritmos analisados foram: *Energy Based Detector* (EBD), *Zero Crossing Detector* (ZCD) e o VAD do CODEC G.729 anexo B. O algoritmo EBD é o que apresentou a menor ocupação de memória e mipagem e ficou em segundo lugar em se tratando de alarmes-falsos. O ZCD ficou em segundo lugar levando-se em consideração a ocupação de memória e mipagem e teve a maior taxa de alarmes-falsos. O VAD do G.729B é o algoritmo mais robusto dos considerados neste trabalho, apresentando a maior ocupação de memória, mipagem e melhor desempenho na distinção correta entre atividade e inatividade de voz.

**Palavras-chave**: VAD. Detecção de Atividade de Voz. DSP.

# ABSTRACT

*During a phone conversation, almost 60% of the time the channel stays inactive, that is, the user is not speaking. The Voice Activity Detector (algorithm) makes the distinction between voice (active voice) and silence or background noise (inactive voice) making possible the CODECs that make use of VAD have a bigger rate of compression when inactive voice is detected. Due the difficulty in the correct distinction between active and inactive voice, some algorithms VAD are being proposed. In this work was carried out the implementation of three VAD algorithms, in assembly of the Digital Signal Processor (DSP) TMS320C5410 from Texas Instruments. After that was also made a comparison between the implementations in assembly where it were considered three features: memory occupation, computational complexity and performance. The analyzed algorithms were: Energy Based Detector (EBD), Zero Crossing Detector (ZCD) and the VAD of CODEC G.729 annex B. The algorithm EBD is what made the slightest memory occupation and computational complexity and stayed in second place in referring to false-alarms. The ZCD stayed in second place taking into consideration memory occupation and computational complexity and had the most rate of false-alarms among the algorithms. The VAD of the G.729B is the most robust algorithm among the ones considered in this work, presenting biggest memory occupation, computational complexity and performance in the correct distinction between active and inactive voice.*

***Keywords**: VAD. Voice Activity Detection. DSP.*

# 1 INTRODUÇÃO

Historicamente as conexões telefônicas ponto-a-ponto são feitas via comutação de circuitos, pela rede pública de telefonia comutada (PSTN - *Public Switched Telephone Network*). Entretanto, com a evolução das redes de dados, estas também vêm sendo empregadas para a comunicação de voz, utilizando a comutação por pacotes e a tecnologia IP (*Internet Protocol*). A comunicação de voz por redes de dados é denominada de VoIP (*Voice over Internet Protocol*).

A tecnologia VoIP define os procedimentos usados para realizar ligações telefônicas sobre uma rede IP, incluindo a digitalização, a codificação do sinal de voz, encapsulamento em pacotes de dados, e protocolos de transporte TCP (*Transmission Control Protocol*) ou UDP (*User Datagram Protocol*) (CISCO, 2007). Segundo Michael Powell (2004 apud RAMIRES & VIOTTI, 2005) "VoIP representa a mais significativa mudança de paradigma em toda a história das telecomunicações modernas, desde a invenção do telefone".

A voz humana é um sinal analógico. Para que ela possa trafegar em uma rede digital, precisará passar por um processo de digitalização[1], que transforma o sinal analógico em digital (BERNAL FILHO, 2003). Para melhor utilização da largura de banda da rede emprega-se codificadores de voz, conhecidos como CODECs (COdificador/DECodificador - *enCOder/DECoder*).

CODECs de voz são padronizados pela ITU-T (*International Telecommunication Union – Telecommunication Standardization Sector*), por meio das recomendações da série G.7XX, que especifica como a voz (sinal analógico) será codificada em uma seqüência de bits (sinal digital). (HARDY, 2003).

Em uma conversação telefônica, cerca de 60% do tempo o canal fica inativo (o usuário não está falando) (MINOLI & MINOLI, 1998 *apud* KHASNABISH, 2003). Portanto, algoritmos

[1] A digitalização é um processo que permite a representação de um sinal analógico em um sinal digital por meio de uma seqüência de bits (COUTINHO, 2005).

de Detecção de Atividade de Voz (VAD - *Voice Activity Detection*) estão se tornando cada vez mais importantes em sistemas relacionados à voz, dentre eles o VoIP (DAVIS & NORDHOLM, 2003).

O VAD (Detector de Atividade de Voz) é um classificador que indica a saída como 1 ou 0, significando respectivamente, a presença ou ausência de voz em cada quadro considerado (ESTEVEZ *et al*, 2005). Em sistemas VoIP os dados de voz (ou *payload* por pacote) são transmitidos junto com um cabeçalho (*header*) ao longo da rede. Um tamanho constante de *payload*, representando algumas amostras (entre 80 e 160) de voz, é conhecido como "quadro" (*frame*), e o seu tamanho é determinado baseando-se no tamanho do fonema, CODEC usado etc. Se um quadro não contém um sinal de áudio ele não precisa ser transmitido. O VAD aplicado em sistemas VoIP indica se um quadro contém um sinal de voz e conseqüentemente se ele será transmitido ou não (PRASAD *et al*, 2002).

Os Processadores Digitais de Sinais (DSP – *Digital Signal Processor*) vêm sendo largamente empregados em telecomunicações (codificação de voz, criptografia etc). Para codificação de voz um algoritmo eficiente de VAD resulta em melhor desempenho do sistema de codificação (BENYASSINE *et al*, 1997).

Neste trabalho são implementados três algoritmos de VAD em processador digital de sinais e efetuada uma análise comparativa de desempenho.

## 1.1 CONTEXTUALIZAÇÃO

Para que um sinal analógico transportando voz, trafegue pela rede de dados, como em um sistema VoIP, ele precisa ser digitalizado. A digitalização de um sinal analógico é realizada por um Conversor Analógico para Digital (ADC - *Analog to Digital Converter*), já a codificação do sinal digital é realizada pelo CODEC.

A codificação digital de sinais de voz é realizada para atender ao requisito de a capacidade dos meios de transmissão e/ou armazenagem dos dados ser finita. Deste modo, é necessário encontrar um meio termo entre duas necessidades antagônicas: diminuir a quantidade de bits necessária para a representação da informação (*encoding*), e manter a capacidade de recuperar

a informação original (*decoding*) com um nível de distorção aceitável (PEREZ MEANA, 2007).

Neste contexto, existem codificadores de voz que utilizam algoritmos VAD, pois estes detectam a atividade de voz em um *frame* permitindo classificar o sinal de entrada em ativo (voz) ou inativo (silêncio ou ruído ambiente). Tal classificação permite ao codificador de voz (CODEC) operar de um modo quando voz é detectada e de outro nos casos de silêncio. A codificação de segmentos de silêncio pode ser feita usando menos bits aumentando a taxa de compressão do sinal.

## 1.2 PROBLEMA

Tradicionalmente os algoritmos VAD têm se baseado em modelos como entropia espectral[2], energia do sinal, z*ero crossing rate* (taxa de cruzamento por zero) (PRASAD *et al*, 2006). Entretanto, nenhuma das estratégias tem conseguido ótimo global, isto é, conseguem bons resultados em uma característica, por exemplo, baixo processamento, mas demonstram deficiência em outras, por exemplo, compressão do sinal (DAVIS & NORDHOLM, 2003).

O desafio principal em se projetar um algoritmo VAD é o processo de identificar as características que podem classificar eficazmente uma fala em ativa ou inativa (SINGH & BOLAND, 2007). Este trabalho se propõe a implementar três algoritmos de VAD a fim de avaliar seus desempenhos em determinados ambientes.

## 1.3 OBJETIVOS

### 1.3.1 Objetivo geral

Este trabalho tem como objetivo, a implementação e comparação de três algoritmos VAD em um Processador Digital de Sinais (DSP).

[2] Estudos sobre o espectrograma de um sinal com muito ruído mostram que os segmentos contendo voz são mais "organizados" que segmentos contendo ruído. A aplicação do conceito de entropia para a detecção de atividade de voz assume que o espectro do sinal é mais organizado durante os segmentos de voz (RENEVEY & DRYGAJLO, 2001).

### 1.3.2 Objetivos específicos

- Analisar alguns algoritmos VAD disponíveis;
- Implementar os algoritmos VAD em linguagem “C”;
- Implementar os algoritmos VAD em linguagem assembly do processador DSP escolhido;
- Realizar testes de validação;
- Comparar os algoritmos considerados.

### 1.3.3 Escopo e delimitação do trabalho

Este trabalho se limita à implementação em DSP de alguns algoritmos VAD existentes e a realização da comparação entre os algoritmos considerados.

## 1.4 RESULTADOS ESPERADOS

Ao final deste trabalho, espera-se, dispor de:

- Implementações dos algoritmos de VAD em linguagem assembly do DSP escolhido;
- Análise comparativa de algumas características, tais como, complexidade computacional, ocupação de memória e desempenho dos algoritmos VAD considerados.

## 1.5 JUSTIFICATIVA

Com o desenvolvimento da tecnologia VLSI (*Very Large Scale Integration*), a capacidade de processamento dos processadores digitais de sinais (DSPs) tiveram forte crescimento, que viabilizou a implementação de algoritmos de processamento de sinais, contribuindo em vários ramos da indústria. Um destes ramos que teve um significativo desenvolvimento nos últimos anos é o das telecomunicações. Alguns importantes desenvolvimentos têm contribuído para este fato, tais como, algoritmos eficientes de codificação de voz (CODECs), equalizadores, canceladores de eco etc. (PEREZ MEANA, 2007).

Algoritmos de VAD têm sido implementados juntamente com os CODECs, pois estes detectam a atividade de voz, permitindo determinar se um quadro contém ou não voz, para a

sua eventual transmissão. A indicação do VAD possibilita diminuir o *bit-rate*[3] em uma transmissão de voz, liberando a capacidade do canal para aplicações concorrentes e com isso otimizando o uso do canal de transmissão.

## 1.6 ASPECTOS METODOLÓGICOS

### 1.6.1 Metodologia

A realização deste trabalho iniciou pelo estudo do estado da arte sobre VAD e o levantamento de algoritmos de VAD disponíveis na literatura para análise.

Posteriormente foi realizada a implementação dos algoritmos selecionados em linguagem "C". Finalmente, os algoritmos de VAD foram implementados em linguagem assembly do processador DSP escolhido, bem como foi realizada uma comparação entre os mesmos.

Assim, para que os objetivos propostos fossem alcançados, o presente trabalho seguiu o método científico de análise e síntese. A análise é a decomposição de um todo em suas partes, partindo do mais complexo para o menos complexo. Em contra-partida, a síntese é a reconstituição do todo decomposto pela análise, partindo do mais simples para o menos simples (CERVO & BERVIAN, 1996).

A pesquisa bibliográfica procura explicar determinado problema a partir de referências teóricas publicadas em documentos. Pode ser realizada de modo independente ou como parte da pesquisa descritiva ou experimental (CERVO & BERVIAN, 1996). Desta forma, as referências bibliográficas ajudaram no entendimento e resolução dos problemas encontrados.

As simulações em linguagem "C" permitiram compreender melhor os algoritmos de VAD. Este tipo de atividade refere-se à pesquisa experimental, que se caracteriza por manipular diretamente as variáveis relacionadas com o objeto de estudo (CERVO & BERVIAN, 1996).

[3] Num arquivo de áudio ou vídeo, *bit-rate* é o número de bits por segundo, usados para representar o conteúdo a ser exibido (GUIA DO HARDWARE, 2008).

Diante das classificações atribuídas aos processos relacionados a este trabalho, tais pesquisas foram realizadas de maneira qualitativa[4].

[4] Pesquisa qualitativa é aquela que busca entender um fenômeno específico em profundidade (CERVO & BERVIAN, 1996).

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é descrito o detector de atividade de voz (VAD – *Voice Activity Detector*), e os algoritmos de VAD (*Voice Activity Detection*) implementados e analisados neste trabalho.

Os algoritmos VAD analisados usam amostras PCM (*Pulse Code Modulation*) como dados de entrada. A vantagem no uso da codificação PCM linear é que ela é usada em codificadores de voz, tais como as recomendações G.711 ou G.729, da ITU-T (PRASAD *et al*, 2006). Portanto neste capítulo também será apresentado o método de codificação PCM.

## 2.1 MODULAÇÃO POR CÓDIGO DE PULSO (PCM)

A recomendação G.711 da ITU-T (1988) define o método PCM de modulação. Este tipo de modulação é usado para a transmissão de conversações telefônicas em ligações interurbanas. Este método é dividido em três etapas (Figura 1): amostragem, quantificação e codificação.

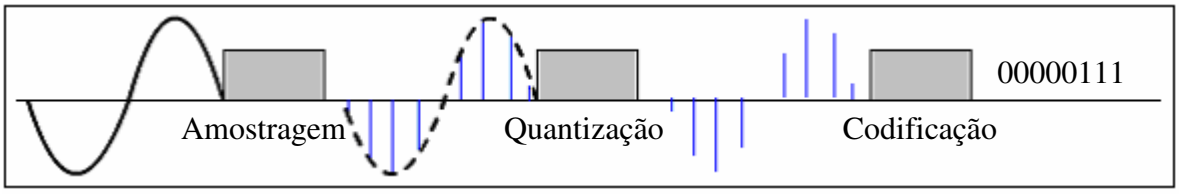


Figura 1 - Etapas da modulação PCM.

### 2.1.1 Amostragem

Em um processo de amostragem, um sinal de voz contínuo no tempo é transformado em um sinal discreto no tempo. Na amostragem são obtidas amostras (*samples*) do sinal analógico. Por meio dessas amostras o sinal poderá ser reconstruído. A freqüência na realização da amostragem é denominada de freqüência de amostragem, e de acordo com o teorema de Nyquist, para se reconstruir fielmente um sinal analógico a freqüência de amostragem deve ser maior ou igual a duas vezes a maior freqüência do sinal. Em aplicações de telefonia a taxa de amostragem usada é igual a 8000 amostras por segundo (SOARES *et al*, 1995).

### 2.1.2 Quantização

No processo de quantização são atribuídos valores discretos contáveis, aos níveis de tensão do sinal amostrado. Esses valores estão dentro de um conjunto de valores que é determinado pela

resolução (em bits) que as amostras terão. Para a modulação PCM geralmente usam-se 8 bits de resolução. Neste processo pode ocorrer um erro conhecido como erro de quantização, que consiste na diferença entre o valor original da amostra e o valor discreto atribuído a ela (SOARES *et al*, 1995).

### 2.1.3 Codificação

O sinal PCM a ser transmitido é obtido pela codificação dos valores discretos quantizados. O codificador atribui a cada valor quantizado um código de 8 bits, de acordo com a resolução para sistemas telefônicos. Temos com isso uma taxa de digitalização de 8000 amostras/segundo x 8 bits/amostra, ou 64 kbit/s (SOARES *et al*, 1995).

## 2.2 ALGORITMO DE DETECÇÃO DE ATIVIDADE DE VOZ (VAD)

O processo de separar voz e silêncio é chamado de detecção de atividade de voz (VAD). Algoritmos VAD foram usados primeiramente em sistemas de reconhecimento de voz, compressão e codificação de voz para encontrar o começo e o final das palavras em uma conversa (PRASAD *et al*, 2006). Dentre as recentes aplicações que fazem uso de VAD podemos citar áudio conferência, sistemas de reconhecimento de voz, sistemas VoIP, telefonia celular, cancelamento de eco e codificação de voz (PRASAD *et al*, 2002). Para o propósito de detectar segmentos de atividade de voz, vários tipos de algoritmos de VAD vêm sendo desenvolvidos.,Deve-se frisar que os diferentes algoritmos variam em sensibilidade, atraso, exatidão e custo computacional (processamento).

A função primária de um detector de atividade da voz é fornecer uma indicação da presença de voz a fim de facilitar o processamento, e, possivelmente, fornecer delimitadores para o início e fim de um segmento de fala. (APPIAH *et al*, 2005).

A Figura 2 mostra a decisão VAD para um arquivo de áudio amostrado com 8kHz, tendo o tamanho do *frame* de 80 amostras ou 10 ms. A linha sólida superposta no gráfico indica os segmentos do sinal que foram classificados como voz (1) ou ruído (0).

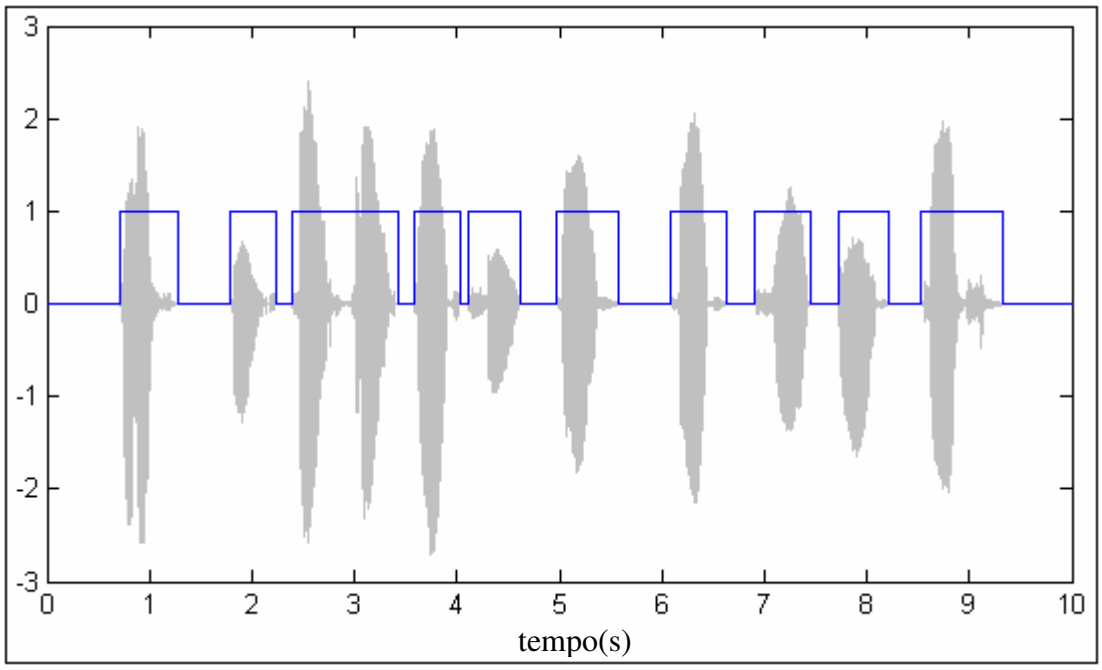

Figura 2 - Exemplo de uma decisão VAD.

O uso de VAD proporciona alguns benefícios, dentre estes, a melhor utilização da capacidade de dispositivos de armazenagem de voz e menor consumo na bateria de aparelhos celulares. Tais benefícios ocorrem, pois os segmentos com ruído não são transmitidos, visto que estes, geralmente, contêm menos informação, se comparado a segmentos contendo fala. (APPIAH *et al*, 2005).

A melhoria na detecção depende principalmente, da porcentagem de pausas durante uma conversação e da confiabilidade do algoritmo de VAD usado para detectar estes intervalos. Por um lado, é vantajoso ter baixa porcentagem de atividade de voz, mas, cortes nos segmentos onde há atividade de voz deve ser evitado para preservar a qualidade. Este é um problema para um algoritmo VAD em ambientes com muito ruído (APPIAH *et al*, 2005).

A variedade e variação na natureza da voz e ruídos do ambiente são um desafio na identificação das características que podem classificar eficazmente um segmento de fala em ativa ou inativa. Recentemente algoritmos de VAD têm se baseado em entropia espectral, nível de energia, *zero crossing rate* (taxa de cruzamento por zero) (PRASAD *et al*, 2006).

Infelizmente, estes algoritmos têm problemas em ambientes onde há muito ruído, especialmente quando o ruído não é estacionário[5].

Um algoritmo que é usado em ambientes abertos onde os ruídos não são estacionários deve ser capaz de detectar voz na presença de diferentes tipos de ruídos. Nestas condições de difícil detecção, é vital que o algoritmo VAD seja "a prova de falhas", indicando atividade de voz quando há dúvida na decisão, para evitar que cortes sejam introduzidos. A maior dificuldade na detecção de voz nestes ambientes é quando uma baixa razão entre o sinal e o ruído (SNR - *Signal to Noise Ratio*) é encontrada. Quando o sinal de segmentos contendo atividade de voz possui amplitude menor que a amplitude do sinal onde há ruídos, não há como se fazer a distinção entre voz e ruído, utilizando técnicas simples de detecção baseadas na energia do sinal. (APPIAH *et al*, 2005).

A maioria dos algoritmos de VAD faz uso de um nível limiar (*threshold*) para fazer a indicação de atividade de voz. Este *threshold* é freqüentemente ajustado para ser fixo, ou adaptativo, sendo atualizado em intervalos de silêncio (inatividade de voz) (APPIAH *et al*, 2005).

A Figura 3 apresenta o esquema genérico de um algoritmo de VAD decomposto em dois passos: a extração dos parâmetros (processamento) e a aplicação de uma regra de classificação (decisão). Parâmetros relevantes são extraídos do sinal de voz, a fim de permitir uma boa detecção dos segmentos contendo voz. Tais parâmetros devem oferecer uma variação discriminativa entre segmentos de voz e segmentos de ruído. Nesta etapa geralmente a extração dos parâmetros é feita com base em um "quadro" (*frame*) de amostras, isto é, o sinal digital é dividido em pequenos segmentos de cerca de 10 ms (80 amostras), considerando uma freqüência de amostragem de 8 kHz (APPIAH *et al*, 2005). A escolha na duração do *frame* pode variar, sendo dependente da aplicação a que o algoritmo de VAD é destinado. Normalmente este tem uma duração de 5 a 40 ms (PRASAD *et al*, 2002). A decisão é feita baseada no *threshold*, sendo ele aplicado aos parâmetros extraídos do sinal, para fazer a distinção entre segmentos de voz e segmentos de ruído. Este *threshold* pode ser fixo ou

[5] Um ruído estacionário apresenta características que são relativamente constantes no tempo, com flutuações inferiores a 5 dB (CCDR-ALG, 2005). Este pode apresentar desvio padrão constante ou variável ao longo do tempo (LOPES, 2004).

adaptativo e a decisão é sempre feita *frame* a *frame*. Independentemente do algoritmo VAD usado, a operação tem que evitar detectar voz como sendo ruído ou ruído como sendo voz (APPIAH *et al*, 2005).

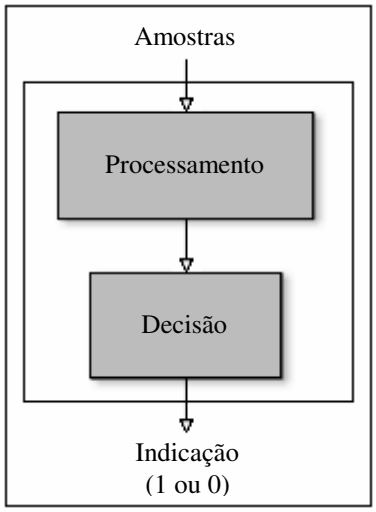


Figura 3 - Esquema genérico de um algoritmo VAD.

Uma representação esquemática de um sistema de comunicação que faz uso de um algoritmo VAD e um codificador de inatividade de voz (ruído) para uma maior taxa de compressão é descrito na Figura 4.

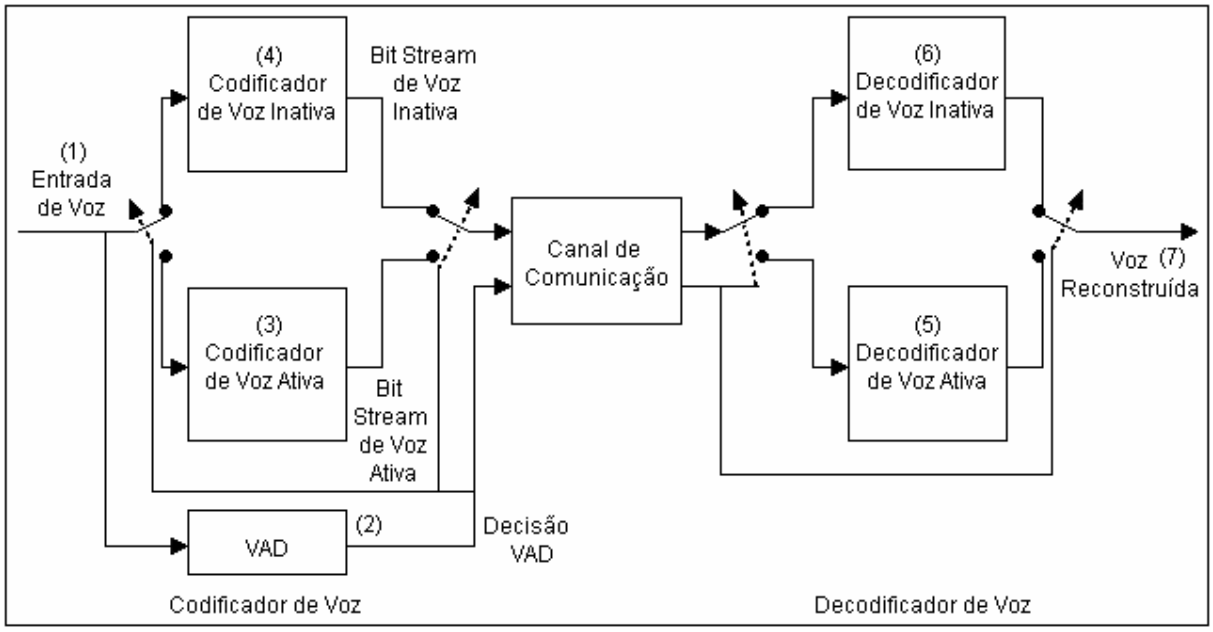


Figura 4 - Sistema de comunicação de voz com VAD.

Fonte: Adaptado de ITU-T (1996b).

A entrada para o codificador de voz é o sinal recebido de voz digitalizada (1). Para cada *frame* do sinal de entrada o VAD (2) fornece uma decisão de atividade de voz, que é usada para alternar entre os codificadores de voz ativa (voz) (3) e voz inativa (silêncio ou ruídos do

ambiente) (4). Quando o codificador de voz ativa está operacional, para cada *frame* um *bitstream* de voz ativa é enviado para o decodificador. Entretanto, durante segmentos de inatividade, o codificador de voz inativa (4) pode escolher em não transmitir qualquer informação ou enviar uma informação de atualização, conhecida como Descritor de Inserção de Silêncio (SID – *Silence Insertion Descriptor*) para o decodificador de voz inativa (6). Esta técnica é chamada de Transmissão Descontínua (DTX – *Discontinuous Transmission*). Para cada *frame*, a saída de ambos os decodificadores é usada como um sinal reconstruído (7) (BENYASSINE *et al*, 1997).

Durante os segmentos contendo atividade de voz, o decodificador de voz ativa (5) reconstruirá o sinal. No entanto, quando inatividade de voz for declarada pelo VAD, o sinal de saída será gerado pelo decodificador de voz inativa (6). A opção em deixar a saída completamente muda, durante segmentos de inatividade de voz, cria uma queda na energia do sinal, que faz com que o ouvinte tenha uma desconfortável sensação de súbita perda de informação. Portanto, para preencher estes segmentos de silêncio, uma descrição do ruído ambiente (SID) será enviada do codificador para o decodificador de voz inativa (6). Usando o SID, o decodificador de voz inativa fará a geração de um sinal de saída que é perceptivelmente equivalente ao ruído original. Tal sinal é comumente chamado ruído de conforto, e o decodificador de voz inativa é chamado de Gerador de Ruído de Conforto (CNG - *Comfort Noise Generator*) (BENYASSINE *et al*, 1997).

Dentre as características desejáveis para um algoritmo VAD estão (PRASAD *et al*, 2002):

- Implementar uma boa regra de decisão que explore as propriedades da fala, para classificar esta em ativa ou inativa;
- Adaptar-se de acordo com mudanças no ruído do ambiente;
- Ter baixa complexidade computacional, para ser utilizado em tempo real;
- Ter boa qualidade de voz, após a aplicação do algoritmo VAD;
- Maximizar a detecção de segmentos inativos de voz, para economizar largura de banda.

## 2.3 ENERGY BASED DETECTOR (EBD)

Muitos dos algoritmos VAD são baseados em medidas de energia. Estes algoritmos têm a vantagem de serem simples e de não fazerem suposições sobre as características do ruído. Entretanto, algoritmos baseados na energia são sensíveis ao ruído e, portanto, variações no ruído podem reduzir seu desempenho (YAMAMOTO *et al*, 2006). Tais algoritmos têm conseguido desempenho adequado a ambientes limpos de ruídos, principalmente quando a energia dos segmentos de voz é significativamente maior do que os segmentos onde é encontrado ruído somente (RENEVEY & DRYGAJLO, 2001).

### 2.3.1 Descrição do algoritmo

A energia de um *frame* indica a possível presença de voz e é um importante parâmetro para algoritmos VAD. Na Figura 5 pode ser observado o fluxograma do algoritmo VAD implementado, sendo baseado em Prasad *et al* (2002). Neste algoritmo, a característica utilizada para fazer a indicação de atividade de voz, será a energia do *frame*, que é obtida pela soma dos quadrados das amostras do *frame*, dividido pelo tamanho do *frame* (PRASAD *et al*, 2002).

Ao iniciar, o algoritmo recebe amostras em formato PCM referentes ao sinal de áudio de entrada. Estas são divididas em *frames* com 80 amostras cada. Formado um *frame* ( $f_j$ ), é calculado a energia do mesmo ( $E_j$ ), e com base nesta energia é feita uma comparação com um certo valor, chamado de *threshold* ( $E_{th}$ ). Caso a energia do *frame* seja maior que o valor do *threshold*, é indicada a presença de voz para este *frame*, senão, é indicado a ausência de voz para este *frame* e o valor do *threshold* atualizado. A indicação do VAD para cada *frame* é sinalizada, como por exemplo, com a armazenagem de "1" ou "0", representando respectivamente a presença ou não de atividade de voz no *frame* analisado.

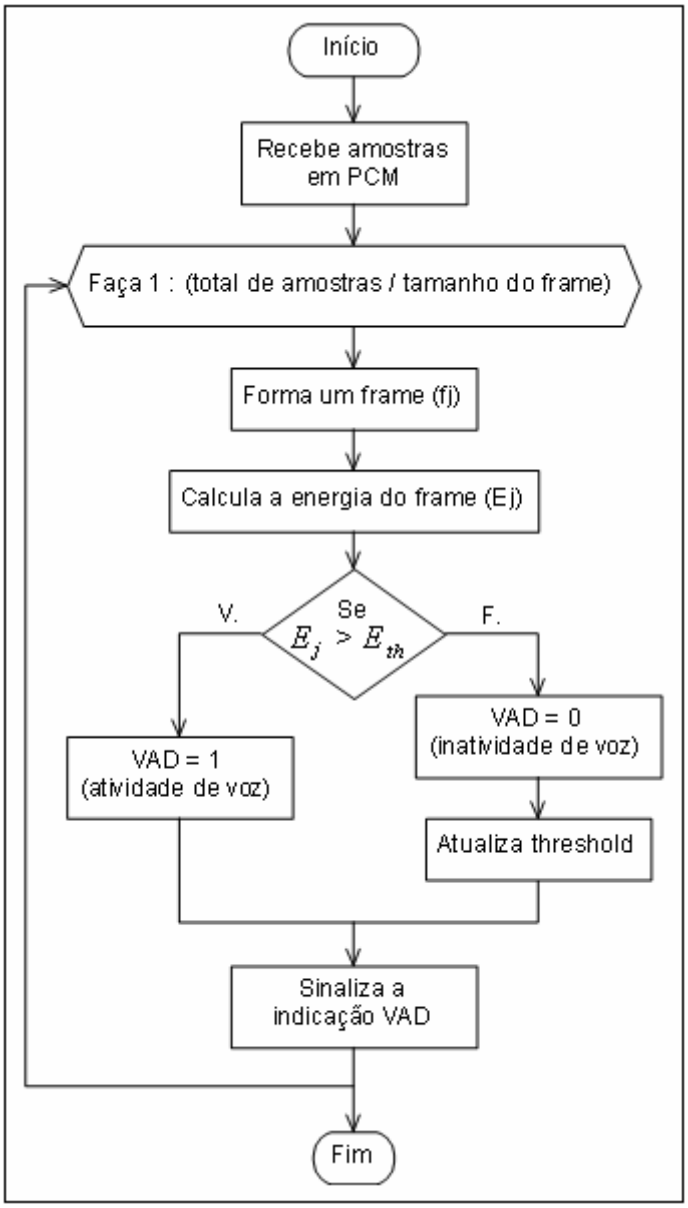


Figura 5 - Fluxograma do algoritmo Energy Based Detector.

Considerando $x(i)$ como a i-ésima amostra de voz. Se o tamanho do *frame* for de $k$ amostras, então o j-ésimo *frame* pode ser representado no domínio do tempo por uma seqüência como demonstra a Equação 1,

$$f_j = \{x(i)\}_{i=(j-1)k+1}^{jk} \qquad \text{Equação 1}$$

A energia $E_j$ do j-ésimo *frame* pode ser representada como na Equação 2,

$$E_j = \frac{1}{k} \sum_{i=(j-1)k+1}^{jk} x^2(i)$$ Equação 2

Onde, $E_j$ = energia do j-ésimo *frame* e $f_j$ é o j-ésimo *frame* sobre consideração.

A energia do *frame* é o parâmetro usado para a classificação do *frame* em ativo ou inativo. A energia de um *frame* ativo é maior que um *frame* inativo. A regra de classificação é,

Se ( $E_j > E_{th}$ ) então
    frame = 'ativo'
Senão
    frame = 'inativo' Equação 3

onde, $E_j$ representa a energia do *frame* que está sendo avaliado, $E_{th}$ representa a energia do *threshold*, sendo que, $E_{th} = 1{,}5 \times E_r$. O *threshold* será usado para fazer a classificação, sendo que com um fator escalar (1,5) permite maior diferença na energia do sinal para a distinção entre um *frame* contendo voz ou ruído.

### 2.3.2 Valor inicial do threshold

Dois métodos são propostos para ajustar um valor inicial para o *threshold* (PRASAD *et al*, 2002).

Método 1: O algoritmo VAD é "treinado" por um pequeno período de tempo por um arquivo de áudio PCM pré-gravado que contêm somente ruídos. O nível do *threshold* inicial é computado com base nesse arquivo. Por exemplo, a energia do *threshold* inicial é obtida pela média das energias de cada amostra, como na Equação 4,

$$E_r = \frac{1}{v} \sum_{m=0}^{v} E_m$$ Equação 4

Onde, $E_r$ = referência para a estimativa do *threshold* inicial,

$v$ = número de *frames* contidos no arquivo de áudio pré-gravado.

Método 2: Embora similar ao método anterior, aqui é assumido que os 200 ms iniciais de amostra não contém qualquer voz; isto é, estes 20 *frames* iniciais são considerados como 'inativos'. Sua energia média é calculada como na equação (4), com $v = 20$.

### 2.3.3 Atualização do threshold

Para uma decisão mais precisa durante a execução do algoritmo, o valor do *threshold* deve se adaptar a possíveis mudanças na quantidade de ruídos presentes no sinal de áudio. O modo de atualizar o valor do *threshold* é feito conforme a Equação 5 (PRASAD *et al*, 2002),

$$E_{rnovo} = (1 - p)E_{rvelho} + pE_{rruído} \qquad \text{Equação 5}$$

Onde,

$E_{rnovo}$ é a energia do $E_r$ atualizado.

$E_{rvelho}$ é a energia do antigo $E_r$.

$E_{rruído}$ é a energia do mais recente *frame* contendo ruídos.

A referência $E_r$ é atualizada como sendo uma combinação do *threshold* antigo e da energia do *frame* atual que contém ruído. O valor de $p$ usado é de 0.2, pois com este valor, pausas entre duas sílabas (algo em torno de 100 ms) não são consideradas como silêncio (PRASAD *et al*, 2002).

## 2.4 ZERO CROSSING DETECTOR (ZCD)

Algoritmos VAD baseados somente na energia do sinal têm problemas quando utilizados em condições onde há muito ruído, apresentando resultados insatisfatórios, principalmente na detecção de segmentos onde há baixa relação entre o sinal e o ruído (SNR). O baixo desempenho se deve ao fato de tais segmentos serem mascarados pelo ruído (YAMAMOTO *et al*, 2006).

*Zero Crossing* (Cruzamento por Zero) para um sinal é o número de vezes que ele cruza a 'linha zero'. O número de *zero crossings* para um sinal de voz encontra-se em um intervalo

fixo de valores. Por exemplo, para um *frame* de 10 ms, o número de *zero crossings* fica entre 5 e 15 vezes. O número de *zero crossings* para ruído é aleatório e imprevisível. Esta propriedade permite formular uma regra de decisão que é independente da energia do sinal, sendo que o número de *zero crossings* permite detectar, na maioria dos casos, os fonemas com baixa energia (PRASAD *et al*, 2002).

O número de *zero crossings* de um *frame* $f_j$ pode ser calculado como mostra a Equação 6:

$$N_{ZC}(f_j) = \frac{1}{2}\sum_{i=0}^{M-1}\left[\left|\operatorname{sgn}[x(i)] - \operatorname{sgn}[x(i-1)]\right|\right] \qquad \text{Equação 6}$$

Onde,

$$\begin{aligned}\operatorname{sgn}[x(n)] &= 1 \ se \ x(n) \geq 0 \\ &= -1 \ se \ x(n) < 0\end{aligned} \qquad \text{Equação 7}$$

Sendo $M = 80$ o número de amostras do *frame* e $x(i)$ a i-ésima amostra do *frame*.

Este algoritmo, baseado em Prasad *et al* (2002), é uma extensão do algoritmo *Energy Based Detector* (EBD) citado anteriormente. Quando o algoritmo ZCD indica inatividade de voz para um *frame* é utilizada uma segunda regra de classificação. Tal regra calcula o número de cruzamentos por zero (*zero crossings*) no *frame* considerado e compara seu resultado com o intervalo de valores. A segunda regra de classificação do algoritmo ZCD é dada por,

```
Se ( N_ZC(f_j) ∈ R ) então
    frame = 'ativo'
senão
    frame = 'inativo'
```

Equação 8

onde $N_{ZC}(f_j)$ é o número de *zero crossings* detectado no *frame* $f_j$ e $R$ é o conjunto de valores {5,6,7,...,15}, o número de *zero crossings* possível para um *frame* de 10 ms.

O fluxograma completo do algoritmo ZCD pode ser observado na Figura 6.

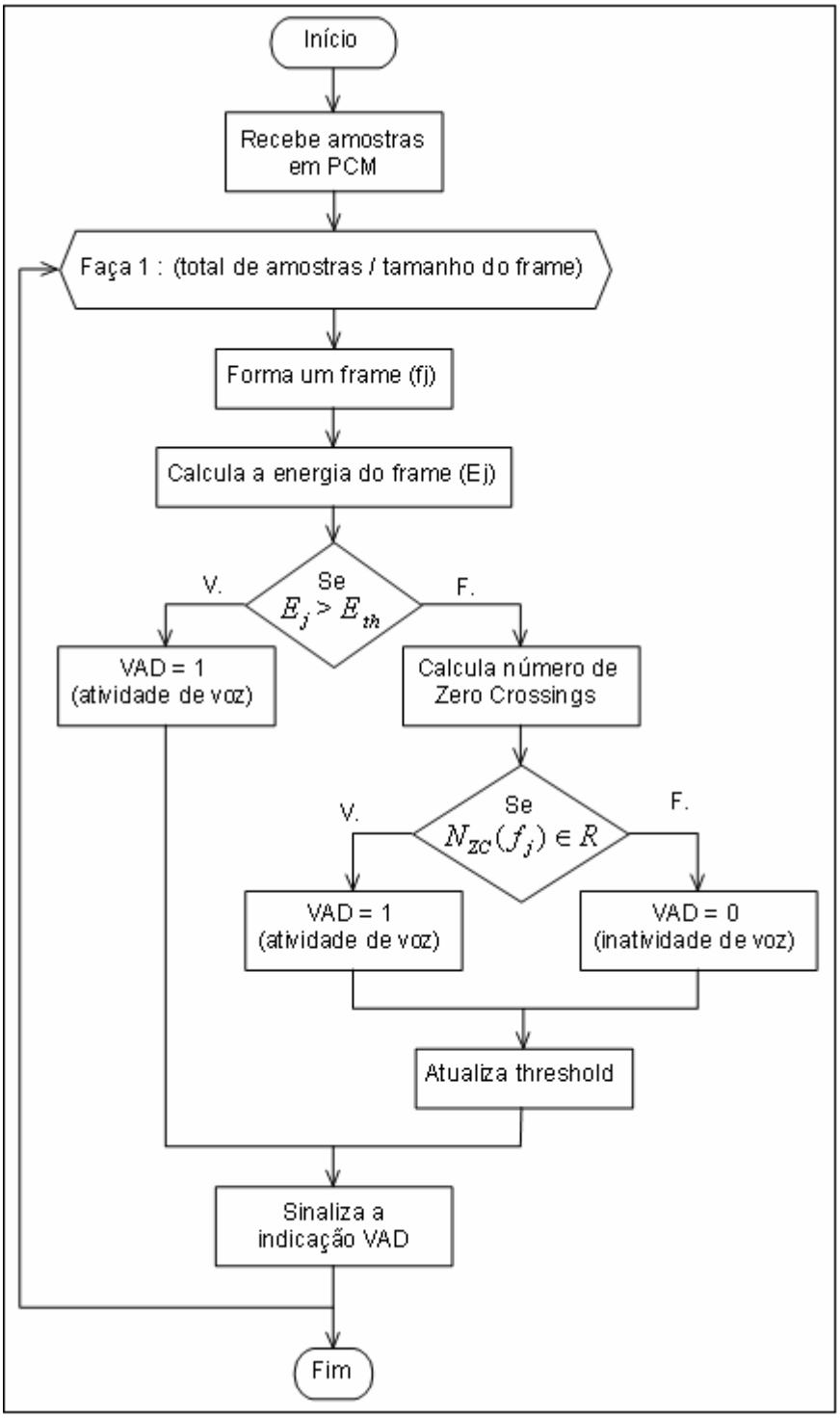


Figura 6 - Fluxograma do algoritmo ZCD.

O sinal de entrada é dividido em *frame*s, como no algoritmo de VAD *Energy Based Detector*. A energia do *frame* é calculada e se sua energia não for superior ao *threshold*, é calculado o número de *zero crossings* para o mesmo. Caso o número de *zero crossings* esteja compreendido no intervalo {5, 6, 7, ..., 15} o *frame* será classificado como 'ativo', do contrário, será classificado como 'inativo' e para qualquer uma das duas alternativas o valor

do novo *threshold* será calculado. A indicação VAD será sinalizada de modo semelhante ao usado no algoritmo EBD.

## 2.5 VAD DO CODEC G.729 ANEXO B

O codificador G.729, aprovado em 1996, foi desenvolvido para operar com uma taxa de transmissão de 8 kbit/s, sendo utilizado, por exemplo, em aplicações de voz por *frame relay*[6] para voz e dados. O sinal de entrada para o codificador é um sinal digitalizado, com largura de banda utilizada em sistemas telefônicos, conforme Recomendação G.712. O sinal analógico é amostrado em 8000 Hz, sendo então convertido em PCM linear de 16 bits, servindo de entrada para o codificador (ITU-T, 1996a).

Após a transmissão, a saída do decodificador é convertida para analógica, sendo novamente um PCM linear de 16 bits. Outra forma de entrada e saída é a descrita pela Recomendação G.711 para dados PCM a 64 kbit/s, sendo convertida em PCM linear de 16 bits antes da codificação e de PCM linear de 16 bits para o formato apropriado depois da decodificação (ITU-T, 1996a).

Conforme ITU-T (1996b) o anexo B do CODEC G.729 ou G.729B, também aprovado em 1996, define um método de detecção de atividade de voz de baixo *bit-rate*, desenvolvido e otimizado para operar juntamente com a versão G.729 ou o anexo A do CODEC G.729 (uma versão de menor complexidade).

Para conseguir um módulo de compressão de silêncio, de boa qualidade e baixo *bit-rate*, um robusto algoritmo de detecção de atividade de voz baseado em *frames* é essencial, para detectar *frames* inativos de voz (silêncio ou ruído ambiente). O CODEC G.729B incorpora um módulo de VAD, DTX e CNG. A união dos algoritmos VAD, DTX e CNG permite ao CODEC G.729B operar com uma taxa média de transmissão de 4 kbit/s durante conversações, enquanto mantém a qualidade na reprodução (BENYASSINE *et al*, 1997).

[6] Frame Relay é uma técnica de comutação de pacotes que se baseia em um conjunto de protocolos especificados pela ITU-T, sendo a técnica mais recomendada para a implementação de redes WAN para conectividade entre hosts e redes locais (DÍGITRO TECNOLOGIA, 2008).

Um fluxograma do algoritmo VAD do padrão G.729B é dado na Figura 7. O VAD opera em *frames* digitalizados de voz. Os *frames* são processados em ordem de chegada, sendo numerados consecutivamente no começo de cada conversação.

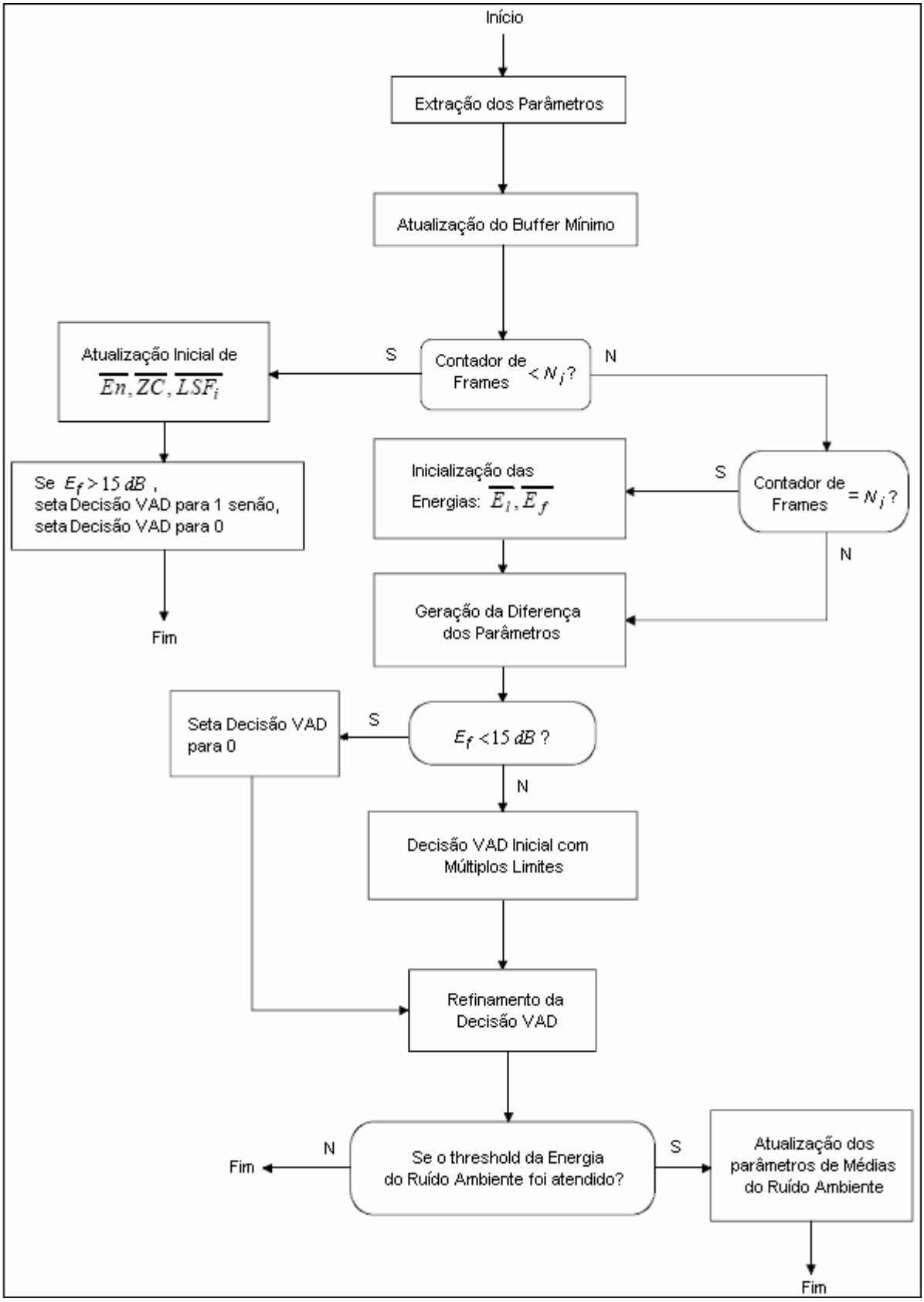


Figura 7 - Fluxograma do VAD do CODEC G.729B.

Fonte: Adaptado de ITU-T (1996b).

No primeiro estágio, quatro características paramétricas são extraídas do sinal de entrada. A extração dos parâmetros é compartilhada com o módulo de codificação para obter maior eficiência computacional. Os parâmetros são: A energia total do *frame*, a energia de baixa freqüência (0 a 1 kHz), um conjunto de *Line Spectral Frequencies*[7] e a taxa de cruzamento por zero (*zero crossing rate*) do *frame*.

Se o número do *frame* for menor que $N_i$, onde $N_i$ é uma constante de valor 32, um estágio de inicialização das médias de longo prazo ocorre, e a decisão de atividade de voz é forçada para 1 se a energia do *frame* na análise Codificação Linear Preditiva[8] (LPC - *Linear Prediction Coding*) for acima de 15 dB. De outra maneira, a decisão de atividade de voz é forçada para 0. Se o número do *frame* for igual a $N_i$, o estágio de inicialização para as energias características do ruído ambiente ocorre.

No estágio seguinte é calculado um conjunto de diferença de parâmetros. Este conjunto é gerado como uma medida da diferença entre os parâmetros atuais e as médias em execução das características do ruído ambiente. Quatro medidas diferentes são calculadas:

- Distorção espectral;
- Diferença na energia;
- Diferença na energia de baixa freqüência;
- Diferença nos cruzamentos por zero (*zero-crossing*).

A decisão de atividade de voz inicial é feita no estágio seguinte, usando regiões de decisão com múltiplos limites no espaço, das quatro medidas de diferença. A decisão de voz ativa é dada como sendo a união das regiões de decisão e a decisão de voz inativa como sendo o complemento da decisão lógica. A análise da energia, junto com decisões de *frames* vizinhos já processados, são usados para refinar a decisão.

[7] As Freqüências de Linhas Espectrais (LSF - *Line Spectral Frequencies*) são usualmente utilizadas para codificação de voz, devido à sua grande eficiência de codificação e suas propriedades atraentes para interpolação (SILVA, 2005).

[8] A idéia básica em relação ao modelo do LPC é que uma amostra de voz dada no tempo n, s(n), pode ser aproximada como uma combinação linear de p amostras passadas (RABINER & JUANG, 1993).

As médias de execução do ruído ambiente devem ser atualizadas somente na presença de ruído ambiente, e não na presença de voz ativa. Um *threshold* adaptativo é testado, e a atualização ocorre somente se o critério do *threshold* é encontrado.

## 2.6 CONSIDERAÇÕES

No início deste capítulo foi apresentada a codificação PCM, que é utilizada como dados de entrada para os algoritmos VAD. No decorrer do capítulo foi abordado o princípio de funcionamento de um algoritmo de detecção de atividade de voz. Este capítulo terminou com a apresentação dos três algoritmos escolhidos para implementação deste trabalho.

# 3 PROCESSADOR DIGITAL DE SINAIS

Neste capítulo é apresentado um histórico do processador digital de sinais e o modelo escolhido para a implementação dos algoritmos VAD considerados. No final do capítulo é abordada a ferramenta de desenvolvimento para programação do DSP, o Code Composer Studio.

## 3.1 DSP E SUAS APLICAÇÕES

O DSP (*Digital Signal Processor* - Processador Digital de Sinais) é um microprocessador com uma arquitetura otimizada para operar com processamento digital de sinais (KUO *et al*, 2006). DSP pode ter dois significados, podendo se tratar do Processamento Digital de Sinais (*Digital Signal Processing*) ou do Processador Digital de Sinais (*Digital Signal Processor*) ( KUO *et al*, 2006).

Processamento digital de sinais se refere à representação digital de sinais e o uso de sistemas digitais para analisar, modificar, armazenar, ou extrair informações destes sinais. Várias pesquisas têm sido realizadas para o desenvolvimento de algoritmos e sistemas DSP para aplicações no mundo real (KUO *et al*, 2006).

Processadores DSP (*chip*) com arquitetura e conjunto de instruções específicas tem sido desenvolvidos por empresas como Texas Instruments, Motorola, Analog Devices etc (VENKATARAMANI & BHASKAR, 2002). Além de criar seus processadores DSP elas também desenvolvem o ambiente de desenvolvimento (IDE - *Integrated Development Environment*), para seus respectivos *chips*, facilitando a criação de programas (KUO *et al*, 2006). O rápido crescimento e comercialização da tecnologia dos processadores DSP não é uma surpresa, considerando as vantagens comerciais em termos de rapidez, flexibilidade e potencialmente a capacidade de baixo custo no projeto oferecida por estes dispositivos (KUO *et al*, 2006).

Os processadores DSP são utilizados em várias aplicações, tais como: controladoras de disco rígido, telefones celulares, sistemas de reconhecimento de voz, processamento de imagem etc. Os DSPs estão se tornando comum e estão substituindo os microprocessadores de uso geral em diversas aplicações (VENKATARAMANI & BHASKAR, 2002).

Os processadores DSPs foram desenvolvidos considerando-se que as operações mais utilizadas em processamento digital de sinais são adições, multiplicações e transferências de memória. Para tal existem instruções, que realizam a multiplicação e a adição em apenas um ciclo de instrução. Por exemplo, a instrução MAC (*Multiply And Accumulate* – Multiplica e acumula) (KUO *et al*, 2006).

## 3.2 HISTÓRICO

As primeiras aplicações com processamento digital de sinais tiveram início das décadas de 60 (1960 a 1969) e 70 quando os computadores digitais se tornaram disponíveis. Computadores eram caros durante esta época, e o processamento digital de sinais estava limitado a algumas aplicações críticas. Tentativas pioneiras foram feitas em quatro áreas chaves: radar e sonar, onde a segurança nacional estava em risco; exploração de petróleo, onde grandes quantias de dinheiro poderiam ser obtidas; exploração espacial, onde os dados devem ter considerável precisão; e imagem médica, onde vidas poderiam ser salvas (SMITH, 1999).

A revolução do computador pessoal dos anos 1980 e 1990 causou uma explosão no uso do processamento digital de sinais, com novas aplicações. Sem ser motivado pelas forças armadas e o governo, o DSP passou de repente a ser dirigido pelo mercado comercial. O processador DSP atingiu o público com produtos tais como: telefones celulares, tocadores de CD (*Compact Disk*), e correio eletrônico de voz (SMITH, 1999).

Os primeiros processadores de único *chip* foram a base na qual os processadores DSP modernos foram construídos. Embora a maioria não fosse comercialmente bem sucedidos, os fabricantes foram rápidos em corrigir os erros que eles possuíam (TAN & HEINZELMAN, 2003).

Em 1978, a AMI lançou o S2811 que foi projetado para operar junto com um processador de propósito geral tal como o 6800 da Motorola. A função principal do S2811 era aliviar a carga de executar intensivas sub-rotinas matemáticas do processador principal. Resumidamente, ele comportou-se como um co-processador matemático e nunca foi utilizado em grandes quantidades em algum produto final (TAN & HEINZELMAN, 2003).

Um ano mais tarde, a Intel anunciou um "Processador de Sinais Analógicos" (2920) o qual possuía um ADC e um DAC (*Digital to Analog Converter* - Conversor Digital para Analógico) no mesmo *chip*. A desvantagem deste processador foi que ele não possuía uma unidade de multiplicação. A multiplicação era realizada por uma série de deslocamentos (*shifts*) adicionados a produtos parciais. Devido a isso o desempenho do 2920 era somente um pouco melhor do que um processador de propósito geral. Comercialmente, o *chip* foi usado somente em *modems* (TAN & HEINZELMAN, 2003).

A AT&T lançou o DSP1 em fevereiro de 1980. O DSP1 possuía a maioria das unidades funcionais vistas em processadores DSP atuais, tal como a MAC. Seu sucesso foi também devido ao fato de que as ferramentas de desenvolvimento estavam disponíveis, acelerando o desenvolvimento de aplicações. A arquitetura do DSP1 sobrevive até hoje, evoluindo na família de processadores DSP 1600 da Lucent Technologies (TAN & HEINZELMAN, 2003). Também em 1980 a NEC anunciou o uPD7720 e foi um dos dispositivos mais largamente usados entre os processadores DSP disponíveis até o momento (KUULUSA, 2000).

A classe de arquitetura representando a primeira largamente aceita de processadores DSP no mercado apareceram no início da década de 1980, com os processadores DSP TMS320C10 da Texas Instruments e o ADSP-2101 da Analog Devices. Os chips foram desenvolvidos com uma arquitetura Harvard com barramentos de dados e programa separados para memória de dados e de programa respectivamente. Os blocos funcionais chaves eram as unidades de multiplicação, adição e acumulação (TAN & HEINZELMAN, 2003).

## 3.3 ARQUITETURA

Os processadores DSP são desenvolvidos com características específicas para aplicações com processamento digital de sinais. Os microprocessadores convencionais são utilizados para aplicações de propósito geral e com isso eles não têm as características necessárias para se obter bom desempenho em algoritmos DSP (VENKATARAMANI & BHASKAR, 2002). Os processadores DSP geralmente utilizam a arquitetura Harvard modificada, podendo ser observada na Figura 8. Nesta arquitetura um conjunto de barramentos é usado para acessar a memória que tem tanto programa quanto dados e outra que tem somente dados. A arquitetura

Harvard modificada é utilizada em processadores DSP da Texas Instruments e Analog Devices, por exemplo (VENKATARAMANI & BHASKAR, 2002).

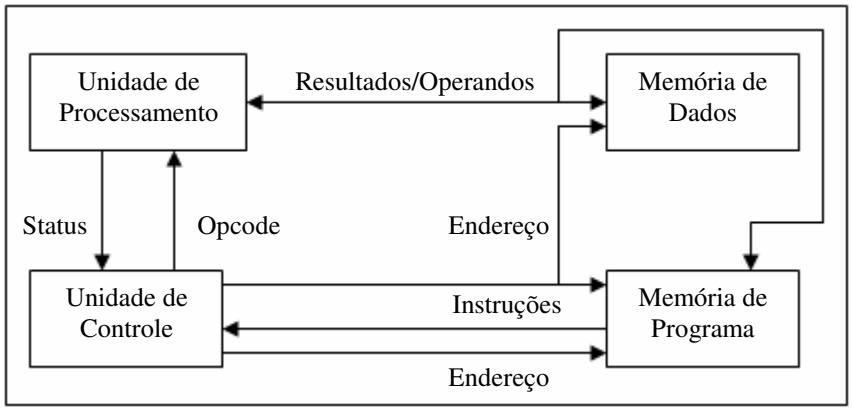


Figura 8 - Arquitetura Harvard modificada.

O número de acessos à memória é um fator que pode influenciar na velocidade de execução das instruções, por exemplo, a instrução MACD (*Multiply ACcumulate with Data shift* - Multiplicação e acumulação com deslocamento) requer quatro acessos à memória por ciclo de instrução[9]. Barramentos de programa e dados separados permitem acesso simultâneo a instruções na memória de programa e operandos na memória de dados, provendo alto grau de paralelismo. Por exemplo, enquanto ocorre uma multiplicação, um produto anterior pode ser carregado, adicionado ou subtraído do acumulador e, ao mesmo tempo, um novo endereço pode ser gerado. Tal paralelismo suporta variado conjunto de operações lógicas, aritméticas e manipulação de bits que podem todas serem executadas em um único ciclo de máquina (VENKATARAMANI & BHASKAR, 2002).

Muitas das instruções em um processador DSP, incluindo a instrução MACD requerem somente um ciclo de *clock* por de instrução (VENKATARAMANI & BHASKAR, 2002).

O diagrama de blocos da Figura 9 mostra os principais componentes encontrados em um processador DSP comum.

[9] Um ciclo de instrução é o tempo decorrido desde uma instrução ser obtida até a completa execução da instrução, incluindo o tempo necessário para armazenar o resultado em um registrador ou memória (VENKATARAMANI & BHASKAR, 2002).

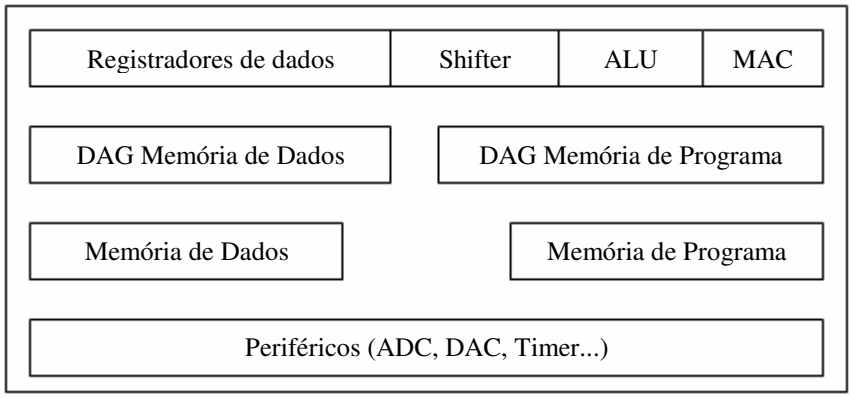

Figura 9 - Diagrama de blocos de um processador DSP convencional.

No topo do diagrama de blocos estão os registradores de dados, eles são utilizados do mesmo modo como em microprocessadores tradicionais. Estes registradores podem manter cálculos intermediários, preparar dados para o processador matemático, servir como *buffer* para transferência de dados, etc. Se necessário estes registradores também podem ser utilizados para controlar *loops* e contadores. A ALU (*Arithmetic Logic Unit* - Unidade Lógica e Aritmética) realiza adição, subtração, valor absoluto, operações lógicas (AND, OR, XOR, NOT), conversão entre formatos de ponto fixo e flutuante, e funções similares. Operações binárias elementares são realizadas pelo Shifter, tal como deslocamento e rotação de bits (SMITH, 1999).

Dois são os geradores de endereços de dados (DAG - *Data Address Generator*), um para cada tipo de memória, de dados e de programa. Estes controlam os endereços enviados para as memórias de programa e dados, especificam onde a informação será lida ou escrita.

Uma das operações mais comuns requeridas em aplicações com processamento digital de sinais é a multiplicação de *arrays* em tempo real (VENKATARAMANI & BHASKAR, 2002). Nestes, antes de uma nova amostra do sinal chegar, a multiplicação das amostras anteriores já deve estar concluída. Isto exige que a multiplicação, bem como a acumulação devem ser efetuadas utilizando elementos de hardware, para tornar essa operação mais rápida. Há duas abordagens para resolver este problema. Uma unidade MAC dedicada pode ser implementada em hardware, que integra um multiplicador e um acumulador em uma única unidade de hardware. Esta abordagem é adotada pelo processador DSP DSP5600X da Motorola. A alternativa é ter multiplicador e acumulador separados, sendo adotado pelos processadores C5x, da Texas Instruments, onde o resultado do multiplicador é adicionado ao

conteúdo do acumulador na ALU central. Em ambas as abordagens descritas a operação MAC é concluída em um ciclo de instrução (VENKATARAMANI & BHASKAR, 2002).

Processadores DSP costumam ter alguns periféricos *on-chip* que aliviam a CPU de algumas funções de uso comum. Estes periféricos também ajudam a reduzir o número de chips em um sistema baseado em processamento digital de sinais. Dentre os periféricos que um processador DSP costuma dispor estão: *Timer on-chip*, porta serial, porta paralela, porta serial TDM, porta *host*, ADC e DAC, portas Comm (são portas paralelas usadas na comunicação entre processos entre um número de processadores DSP idênticos em um sistema multi-processado) (VENKATARAMANI & BHASKAR, 2002).

## 3.4 FAMÍLIA TMS320C54X

A família TMS320C54x (C54x) são processadores DSPs de ponto-fixo de 16 bits da Texas Instruments. Os DSPs C54x contemplam necessidades específicas de aplicações embarcadas em tempo real, tais processadores são encontrados em sistemas de reconhecimento de voz, filtros adaptativos, telefones celulares e reconhecimento de voz, por exemplo (TI, 2001a).

A CPU (*Central Processing Unit* - Unidade Central de Processamento), com sua arquitetura Harvard modificada, tem por características consumo de energia minimizado e alto grau de paralelismo. Além destas características, os modos de endereçamento e o conjunto de instruções nos processadores DSP C54X melhoram o desempenho global do sistema (TI, 2001a).

Os DSPs C54x têm flexibilidade operacional e velocidade. Eles combinam uma avançada arquitetura (com um barramento de memória de programa, três barramentos de memória de dados, e quatro barramentos de endereço), memória *on-chip*, periféricos *on-chip* e um conjunto de instruções altamente especializado (TI, 2001a).

A arquitetura do DSP 54x é construída considerando um conjunto de oito barramentos maiores (quatro barramentos de programa/dados e quatro barramentos de endereços) como apresentado na Figura 10.

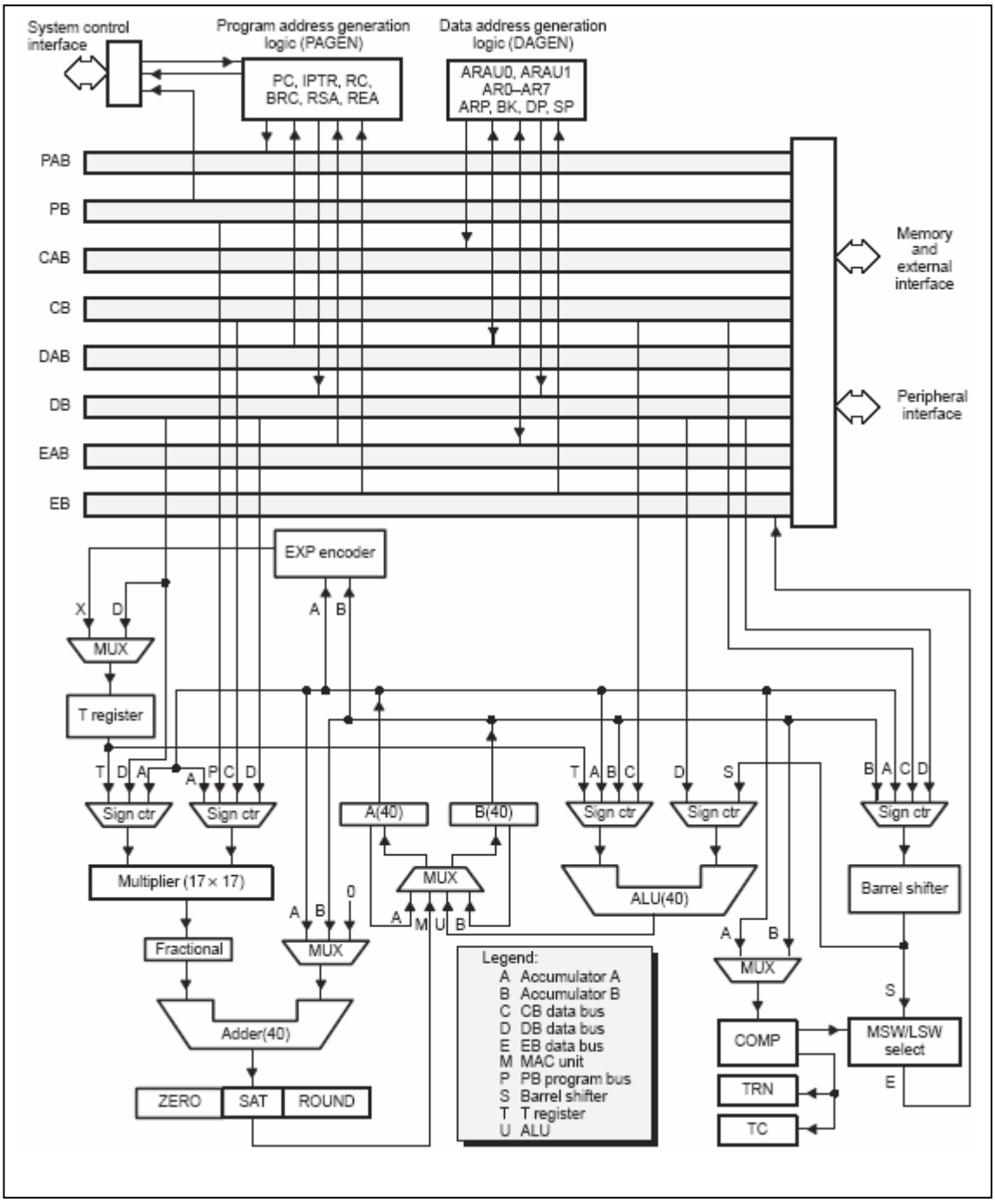


Figura 10 - Diagrama de blocos dos DSPs da família C54x.

Fonte: Adaptado de TI (2001a).

O barramento de programa (PB - *Program Bus*) leva o código de instrução e operandos imediatos da memória de programa. Três barramentos de dados (CB, DB e EB – *C, D and E Bus*) interconectam vários elementos, tais como a CPU, periféricos *on-chip* e memória de dados. Quatro barramentos de endereços (PAB, CAB, DAB e EAB – *Program, C, D and E Address Bus*) entregam o endereço necessário para a execução da instrução (TI, 2001a).

O DSP C54x pode gerar acima de dois endereços de memória de dados por ciclo utilizando as duas unidades auxiliares de registradores aritméticos (ARAU0 e ARAU1 - *Auxiliary Register Arithmetic Units*), elas executam operações aritméticas sobre os oito registradores auxiliares (AR – *Auxiliary Register*), AR0 a AR7 de 16 bits (TI, 2001a).

O barramento de programa (PB – *Program Bus*) pode levar operandos armazenados na memória de programa, por exemplo, um coeficiente de um filtro digital, para as unidades de multiplicação e adição para operações do tipo multiplica e acumula ou para um destino na memória de dados, nas instruções de transferência (MVPD e READA). Esta capacidade, em conjunto com a característica de leitura dupla de operandos, permite a execução em um único ciclo de instruções com três operandos, requeridas em um filtro FIR, por exemplo. O DSP C54x também tem um barramento bidirecional *on-chip* para acessos a periféricos *on-chip* (TI, 2001a).

A unidade central de processamento (CPU) é comum a todos os dispositivos C54x. A CPU C54x dispõe de: unidade lógica e aritmética (ALU) de 40 bits, dois acumuladores (A e B) de 40 bits, *barrel shifter*, multiplicador de 17 por 17 bits, adicionador de 40 bits, unidade de comparação, seleção e armazenagem (CSSU - *Compare, Select, and Store Unit*), unidade de geração de endereço de dados e programa (TI, 2001a).

O processador C54x tem três modos de endereçamento: imediato, direto (endereço de página + *offset*), e endereçamento indireto (TI, 1997).

O modo de endereçamento direto é o mais simples, por exemplo, para carregar a constante 0x77 no acumulador A, basta fazer:

```
LD #0x77, A
```

Outras instruções permitem endereçamento indireto com uma operação na ALU, por exemplo:

```
ANDM 0x00FF, *AR1
```

Este exemplo, em particular, não utiliza como destino um acumulador, isto é, a instrução realiza um AND lógico de uma constante com um valor na memória e armazena o resultado na memória (TI, 1997).

O endereçamento direto requer o uso de um apontador de página de dados (DP - *Data memory page Pointer*) e de um *offset* em uma página de 128 palavras de 16 bits. Apesar do uso de endereçamento direto, a maioria dos algoritmos utiliza o endereçamento indireto e registradores auxiliares (TI, 1997).

Os modos de endereçamento indiretos do C54x permitem algumas opções. Por exemplo, *ARn simplesmente acessa o valor apontado por ARn, podendo ARn ser de AR0 a AR7. Ainda a título de exemplo, *ARn+, *ARn+0, e *ARn+0% respectivamente, realizam pós-incremento, pós-incremento com tamanho de passo variável, e fila circular (TI, 1997).

Uma das abordagens adotadas para aumentar a eficiência dos processadores DSP avançados é o *pipeline* de instrução. Um *pipeline* de instrução consiste em uma seqüência de operações que ocorre durante a execução de uma instrução, isto é, uma instrução é dividida em partes que são executadas de maneira seqüencial. O *pipeline* da família de DSP C54x é composto de seis níveis: pré-busca, busca, decodificação, acesso, leitura e execução (TI, 2001a).

O C54x têm alguns mecanismos de controle de *loop* (laços de repetição), incluindo repetições de uma instrução ou de bloco de instruções. Ambos podem ser realizados ao mesmo tempo, permitindo *loops* aninhados. Uma única instrução pode ser repetida 16 vezes fazendo-se:

```
RPT #16-1
MVPD #tabela, *AR2+
```

Este exemplo também ilustra a transferência de dados da memória de programa para a memória de dados. O ponteiro para a tabela é gerenciado e automaticamente incrementado pela unidade de geração de endereços de programa (PAGEN - *Program Address Gerenation*), devendo estar dentro de um bloco de repetição de uma única instrução. Outra versão de repetição de uma única instrução é instrução RPTZ, que primeiro zera o valor de um dos acumuladores. Há também a instrução RPTB, para repetição de bloco de instruções, o contador de repetição de bloco (BRC - *Block Repeat Counter*) deve ser carregado antes do começo do *loop* (TI, 1997).

O processador DSP escolhido para as implementações é o TMS320C5410 (C5410), que é capaz de realizar 100 MIPS (*Millions of Instructions Per Second* - Milhões de Instruções por Segundo). Este DSP possui 16K palavras de 16 bits de memória ROM (*Read Only Memory* –

Memória Somente de Leitura) configurada para memória de programa, 64K palavras de memória RAM (*Random Access Memory* – Memória de Acesso Aleatório), sendo 8K palavras de memória de programa/dados com modo duplo de acesso e 56K palavras de memória de programa/dados com modo de acesso único. Dentre seus periféricos estão: gerador de *clock on-chip* com oscilador interno ou fonte externa de *clock*, *timer* de 16 bits, controladora DMA (*Direct Memory Access* - Acesso Direto à Memória) de 6 canais e pinos de I/O de propósito geral (TI, 2000).

## 3.5 A FERRAMENTA DE DESENVOLVIMENTO CODE COMPOSER STUDIO (CCS)

O CCS foi desenvolvido pela Texas Instruments para a programação de seus processadores DSP. O CCS permite a edição de código fonte em linguagem de baixo nível (assembly) ou em linguagem "C", como também uma mistura entre ambos os tipos. Este ambiente de desenvolvimento fornece ferramentas para o desenvolvimento, simulação, emulação, depuração e análise de programas em tempo real (TI, 2007).

### 3.5.1 Ambiente de desenvolvimento

A IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado), podendo ser vista na Figura 11, foi desenvolvida para auxiliar os desenvolvedores a maximizar a produtividade, sendo uma ferramenta intuitiva. A IDE do CCS é baseada em padrões populares da indústria, assim a navegação e os ícones são intuitivos, permitindo aos desenvolvedores tornarem-se produtivos rapidamente (TI, 2007).

Desenvolvedores podem ajustar a interface da IDE para servir as suas necessidades durante seu ciclo de desenvolvimento. Janelas podem ser movidas, adicionadas ou removidas como necessário, dando completo controle ao desenvolvedor (TI, 2007).

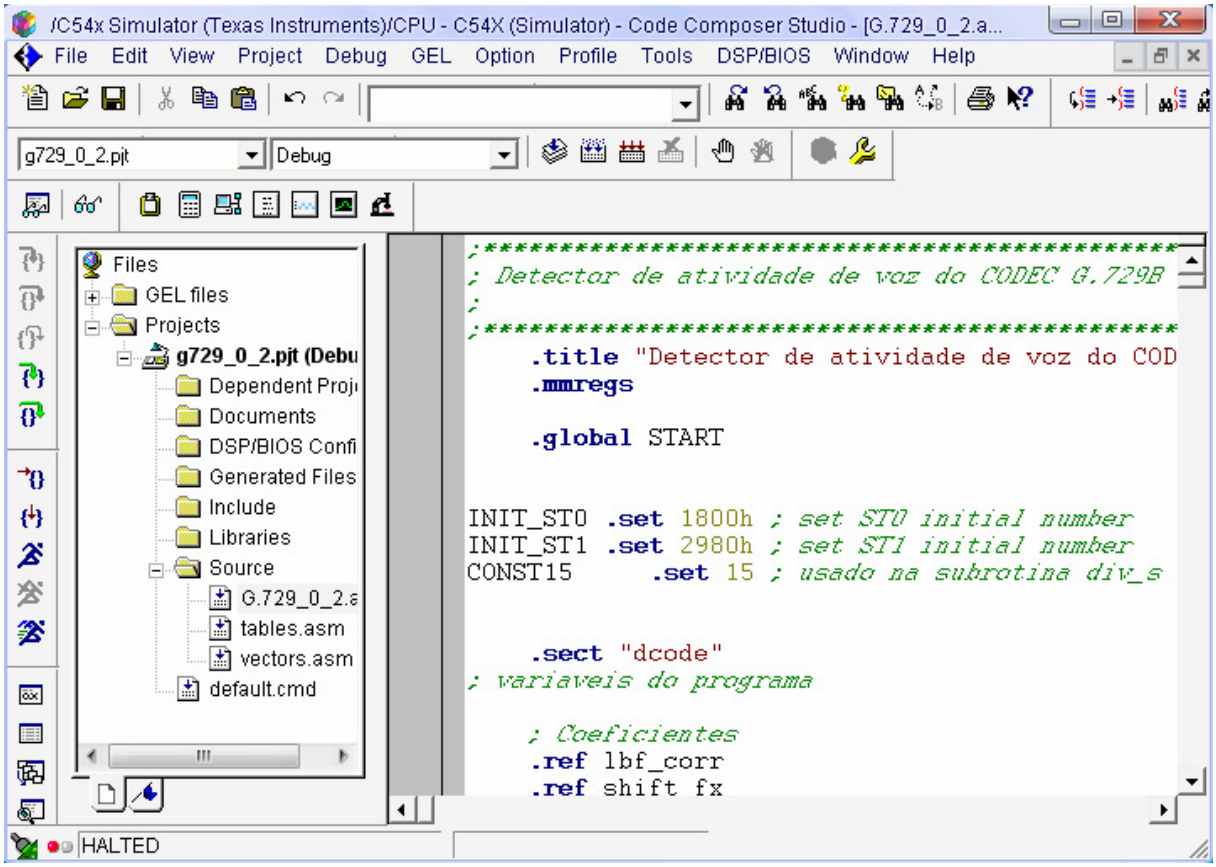

Figura 11 - IDE do Code Composer Studio.

A IDE também é construída com uma Interface de Programação de Aplicativos (API) aberta. Isto permite a terceiros ou clientes construir *plug-ins* que se integram a IDE do CCS e interagem com os componentes do Code Composer Studio. *Plug-ins* permitem o desenvolvimento de aplicações mais customizadas, com funcionalidades especiais, ajudando com isso no desenvolvimento mais rápido e eficaz de aplicações DSP (TI, 2007).

### 3.5.2 Análise em tempo real

Esta característica permite ao programador analisar e monitorar uma aplicação DSP enquanto ela está em execução. Mesmo depois de o programa ter sido parado, as informações já capturadas por meio das ferramentas de análise de tempo real fornecem ricos detalhes sobre a seqüência de eventos que levaram ao atual ponto de execução. As ferramentas de análise de tempo real são usadas no ciclo de desenvolvimento quando há a transição da fase de eliminação de erros (*debug phase*) para a fase de execução (*runtime phase*). Neste tempo, elas mostram os problemas sutis que aparecem da interação dependente do tempo dos componentes do programa (TI, 2007).

### 3.5.3 Visualização de dados da aplicação

A visualização de dados da aplicação permite medir o desempenho e obter um perfil do todo ou de partes do código que necessitam de uma maior otimização; é útil para identificar eventos do processador tais como: número de ciclos, número de *branches*, chamadas de sub-rotinas ou interrupções. Todas as informações estão disponíveis durante o desenvolvimento da aplicação e fase de depuração, aumentando com isso a produtividade (TI, 2007).

### 3.5.4 Gerenciador de projetos

O gerenciador de projetos fornece uma maneira rápida de visualizar, acessar, e manipular todos os arquivos do projeto na mesma janela. Com grandes esforços no desenvolvimento de uma aplicação ela geralmente é dividida em um número de sub-projetos diferentes ajudando a distribuir a carga de trabalho. O gerenciador de projetos do CCS é desenvolvido para satisfazer as necessidades dos usuários que trabalham em projetos grandes, com partes em vários locais (TI, 2007).

### 3.5.5 Ajuda on-line

O CCS dispõe de uma extensa ajuda *on line*, do tipo sensível ao contexto. O sistema de ajuda contém informações completas sobre os dispositivos DSP, incluindo a definição dos registradores e sumário das instruções (TI, 2007).

### 3.5.6 Seleção do dispositivo (chip DSP)

O utilitário de configuração do CCS permite ao desenvolvedor configurar rapidamente seu ambiente de desenvolvimento. As configurações incluem vários tipos de simuladores totalmente funcionais, bem como suporte para algumas placas, como as DSKs (DSP *Starter Kits* - kit de desenvolvimento DSP). Na configuração pode-se criar dispositivos customizados, selecionando o processador e a interface de emulação e salvá-lo com um nome único. Algumas das características de configuração do CCS permitem ao desenvolvedor: visualizar e modificar seu sistema de configuração, adicionar múltiplas placas de desenvolvimento e CPUs ao seu sistema, permitindo a depuração multi-processador. O CCS permite também importar configurações anteriores que foram salvas tornando mais rápida a definição ou mudança de configuração (TI, 2007).

## 3.6 CONSIDERAÇÕES

Neste capítulo foi apresentada uma breve história sobre o processador DSP, suas aplicações, arquitetura, periféricos, e as características do DSP utilizado neste trabalho para a implementação dos algoritmos VAD considerados. Neste capítulo também foi apresentado o Code Composer Studio, que é o ambiente de desenvolvimento utilizado para a programação do processador DSP escolhido.

# 4 IMPLEMENTAÇÃO DOS ALGORITMOS

Este capítulo trata da implementação dos três algoritmos de VAD analisados neste trabalho. São comentadas as particularidades das implementações em linguagem "C" e em assembly no processador DSP escolhido, o TMS320C5410 da Texas Instruments.

## 4.1 ALGORITMO EBD EM LINGUAGEM C

As implementações em linguagem "C" foram realizadas com o auxílio do ambiente de programação Bloodshed Dev-C++ para Windows, disponível em (Dev-C++, 2005). Os primeiros algoritmos EBD e ZCD foram desenvolvidos em linguagem "C" utilizando tipos de dados de ponto fixo, compatível com o processador DSP TMS320C5410 que é de 16 bits ponto fixo. Já a implementação em linguagem "C" do algoritmo VAD do CODEC G.729B, que também utiliza tipo de dados ponto fixo, está disponível em (Recommendation G.729 Annex B, 2000). O algoritmo *Energy Based Detector* (EBD) é o mais simples dos três algoritmos implementados. Seu fluxograma está disposto na Figura 5, e, será o primeiro algoritmo a ter o código em linguagem "C" comentado.

A função principal do algoritmo EBD é a que calcula a energia de um *frame*, apresentada no trecho de código da Figura 12. Esta função necessita de dois parâmetros. O primeiro é um ponteiro que aponta para a primeira posição de um vetor contendo amostras PCM linear. O segundo parâmetro, "tamanho", indica a quantidade de elementos que este vetor possui. Na utilização da função "calculaEnergiaDoFrame" o valor de "tamanho" é 80, que é passado como parâmetro pela constante "T_FRAME". O vetor é composto de amostras de 16 bits (representadas pelo tipo de dados *short int*, que utiliza 16 bits sinalizados).

```
unsigned long int calculaEnergiaDoFrame(short int *vetor,
unsigned char tamanho)
{
      int i;
      unsigned long long int soma = 0;
      unsigned long int resultado;
      for(i=0; i<tamanho; i++)
      {
            soma += (vetor[i]*vetor[i]);
      }
      resultado = soma / tamanho;
      return(resultado);
}
```

Figura 12 - Cálculo da energia do frame em linguagem C.

A variável "soma" foi definido do tipo *unsigned long long int* de 64 bits, pois teoricamente há a possibilidade de seu valor chegar a utilizar 37 bits (foram realizados experimentos e foi observada tal afirmação). Esta variável armazena o valor da soma das 80 amostras elevadas ao quadrado, com isso não há a necessidade de acondicionar números negativos (definido por *unsigned*).

A energia do *frame*, isto é, o retorno da função, é feito pela variável "resultado", do tipo *unsigned long int* (32 bits), que resulta da divisão de "soma" por "tamanho".

O algoritmo EBD inicia com os cálculos da referência do *threshold* e o próprio *threshold*, respectivamente identificados pelas variáveis "referenciaTh" e "threshold" conforme é apresentado no trecho de código na Figura 13.

```
#define T_FRAME 80
short int frame[T_FRAME];
unsigned long long int soma20Frames;
unsigned long int referenciaTh;
unsigned long int threshold;

soma20Frames = 0;
for (i=0; i<20; i++) {
      // forma um frame, nas simulações com a leitura em arquivo.
      soma20Frames += calculaEnergiaDoFrame(frame, T_FRAME);
}
referenciaTh = soma20Frames / 20;
threshold = ((referenciaTh*15)/10);
```

Figura 13 - trecho de código para o cálculo do valor inicial do threshold.

Os primeiros vinte *frames* formados são utilizados para o cálculo dos valores das variáveis "referenciaTh" e "threshold". A variável "soma20Frames" armazena a soma das suas energias, obtidas com a função "calculaEnergiaDoFrame". O valor da variável "referenciaTh" é obtido por meio da divisão da variável "soma20Frames" por vinte, já o valor de "threshold" é obtido pela multiplicação de "referenciaTh" por 1.5, conforme apresentado na Seção 2.3.1. Entretanto, como só foram utilizados tipos de dados inteiros não é possível fazer uma

multiplicação por 1.5, que é de ponto flutuante. Logo, pelo menos duas alternativas existem para se contornar este problema: multiplicar por quinze e dividir o resultado por dez ou dividir por dez e multiplicar o resultado por 15, sendo utilizada a primeira das alternativas, dado que nos testes realizados apresentou um resultado mais preciso.

Tendo sido calculados a referência do *threshold* e o correspondente valor do *threshold*, o detector de atividade de voz passará a atuar nos *frames* seguintes.

O trecho de código da Figura 14 trata da indicação do VAD e a eventual atualização do *threshold*.

```
unsigned long int energiaEj;

energiaEj = calculaEnergiaDoFrame(frame, T_FRAME);

if (energiaEj > threshold)
{
      printf("VAD = 1\n");
      vad = 1;
}
else
{
      printf("VAD = 0\n");
      vad = 0;

      // atualiza referência do treshold e treshold

      referenciaTh = ((8*referenciaTh)/10) + ((2*energiaEj)/10);
      threshold = ((referenciaTh*15)/10);
}
```

Figura 14 - Trecho de código da função principal do algoritmo EBD.

O trecho de código da Figura 14 inicia com o cálculo da energia do *frame*, sendo armazenada na variável "energiaEj". A "energiaEj" é comparada com o "threshold", caso ela seja maior que "threshold" é declarado atividade de voz para este *frame* (variável "vad" recebe o valor "1"), do contrário, a variável "vad" receberá o valor "0", indicando inatividade de voz para este *frame*. Caso inatividade de voz seja detectada, há a necessidade de atualização da referência do *threshold* e do *threshold*, conforme descrito na Seção 2.3.3.

Esta implementação foi utilizada como referência para a implementação deste algoritmo em linguagem assembly.

## 4.2 ALGORITMO EBD EM ASSEMBLY

O ambiente de desenvolvimento Code Composer Studio (CCS) foi utilizado na programação dos algoritmos em assembly. O algoritmo EBD inicia com o cálculo da energia de vinte *frames*, que são utilizados para o cálculo do valor inicial do *threshold*. O trecho de código da Figura 15 mostra como é realizada a leitura de 80 amostras de um arquivo contendo amostras PCM em formato hexadecimal para o vetor "amostras".

```
T_FRAME           .set  80
arqEntrada        .word   0 ; ponteiro para o arquivo de entrada
amostras          .space (T_FRAME*16) ; 80 amostras

STM #T_FRAME 80-1,AR6   ; recarrega valor para loop
STM arqEntrada,AR2
STM amostras,AR3
leituraAmostras:
      MVDD *AR2,*AR3+ ; copia 80 amostras do arquivo no vetor amostras
      BANZ  leituraAmostras,*AR6-   ; faz 80 vezes
```

Figura 15 - Preenchimento do frame com 80 amostras.

O trecho de código da Figura 15 inicia com a declaração da constante "T_FRAME" com o valor 80 decimal, após isto, a variável "arqEntrada", que será ligada ao arquivo com as amostras PCM por meio do recurso "Port Connect" do CCS. A terceira declaração é a do vetor "amostras", que com o auxílio da diretiva ".space" reserva 1280 bits (80 * 16). Sempre que referenciada a variável "amostras" é carregado o endereço do primeiro espaço de 16 bits deste vetor.

É importante comentar que no *loop* o registrador "AR2", que aponta para o vetor de entrada, incrementa automaticamente, sem a necessidade do modificador "+", como no caso do registrador "AR3" que aponta para o vetor "amostras".

Para o cálculo da soma de 80 amostras elevadas ao quadrado é utilizada a instrução SQURA como mostra a Figura 16.

```
LD    #0,A  ; zero o acumulador A
STM   #80-1,AR6   ; carrega valor para loop
STM amostras,AR3
calcEFrameValorInicial:
      SQURA *AR3+, A
      BANZ  calcEFrameValorInicial,*AR6-
```

Figura 16 - Trecho de código para soma do quadrado de 80 amostras.

Para a repetição da instrução SQURA é utilizado um tipo de *loop* disponível ao C5410. Neste *loop* é utilizado um registrador auxiliar (AR6) que com a instrução BANZ fará um salto para o *label* "calcEFrameValorInicial" enquanto "AR6" não for zero. A instrução SQURA carrega

um valor de 16 bits apontado pelo registrador “AR3”, calcula o quadrado deste valor e soma com o valor contido no acumulador “A”. O DSP C5410 dispõe de dois acumuladores, “A” e “B”, sendo ambos de 40 bits e utilizados em instruções que realizam *shift*, *load*, *store*, multiplicação, adição etc. Após a soma do quadrado de 80 amostras, é necessário fazer a divisão do resultado por 80. O C5410 não tem instruções para divisão em um único ciclo de *clock*. A rotina de divisão pode ser vista na Figura 17 e está disponível em (TI, 2001b).

```
STL A,d_NumL
SFTA A,-16 ; A = A >> 16
RPT #(16-1)
      SUBC VALOR,A
STL A, d_QuotH
XOR d_QuotH,A ; limpa AL
OR d_NumL,A ; AL = NumL
RPT #(16-1)
      SUBC VALOR,A
STM #d_QuotH,AR6
STL A,d_QuotL
DLD *AR6,A
```

Figura 17 - Rotina em assembly de divisão inteira.

O trecho de código da Figura 17 efetua a divisão sem sinal de um número de até 40 bits (estando no acumulador “A”) por outro de 16 bits (variável “VALOR” na figura). O resultado da rotina de divisão é armazenado no acumulador “A”.

Após o cálculo da média dos vinte primeiros *frames* do arquivo, o resultado será a referência para o valor inicial do *threshold*, “referenciaTh”, e pode utilizar mais que 16 bits. Portanto a variável em assembly também ocupa 32 bits, permitindo a utilização das instruções DLD e DST, que respectivamente, fazem um *load* e *store* de 32 bits em um único ciclo. Como no algoritmo em linguagem “C” o valor do *threshold* é obtido com a multiplicação da referência do *threshold* por quinze e posteriormente o resultado sendo dividido por dez. O processador C5410 possui instruções para multiplicação de 17 por 17 bits. A Figura 18 mostra a rotina desenvolvida que realiza a multiplicação sem sinal de um número de 32 bits por outro de 16 bits.

```
STM a_mult_h,AR2  ;parte alta de A
STM a_mult_l,AR3  ;parte baixa de A
STM CONST_M_15,AR6      ; aponta para o valor 15
STL A,*AR3
STH A,*AR2
LD *AR3,T
MPYU *AR6,A
LD A,B
LD *AR2,T
MPYU *AR6,A
SFTA A,8
SFTA A,8
ADD B,A
```

Figura 18 - Rotina em assembly para multiplicação de 32 por 16 bits.

No trecho de código da Figura 18 "CONST_M_15" é uma variável que tem o valor 15, já o valor de 32 bits para a multiplicação deve estar no acumulador "A" e ao final da rotina o resultado da multiplicação é armazenado no acumulador "A". O resultado da multiplicação pode ultrapassar 32 bits, o que não é problema visto que os acumuladores, "A" e "B", possuem 40 bits assim distribuídos: 16 bits (parte baixa) + 16 bits (parte alta) + 8 bits "guarda", para armazenarem os bits que de outra maneira poderiam causar um *overflow* no valor do acumulador.

A Figura 19 mostra como é a tomada de decisão para um *frame*. Nesta figura o registrador "AR3" aponta para o vetor "amostras" que formam o *frame*.

Como pode ser observado na Figura 19, após o cálculo da soma do quadrado de 80 amostras, o código foi suprimido (parte de código que efetua a divisão do acumulador "A" por 80).

```
calcEFrame:
      SQURA *AR3+,A
      BANZ  calcEFrame,*AR6- ; faz 80 vezes

      ; código suprimido. Aqui divide de A por 80
      DST A,*AR4 ; armazeno a energiaEj

      DLD *AR1,B ; carrego em B o threshold
      NEG B ;threshold fica negativo
      ADD B,A
      BC trataRuido,ALEQ      ; pula caso A <= 0

      LD #1,B     ; é voz
      B decisaoJahTratada
trataRuido:

      ; aqui tenho que atualizar o valor do threshold
      ; ((2 * energiaEj)/10)
      DLD *AR4,A
      SFTA A,1    ; multiplica por 2
      ; código suprimido. Rotina para dividir A por 10
      DST A,*AR4  ; armazeno temporariamente em energiaEj

      ; ((8 * referenciaTh)/10)
      DLD *AR5,A
      SFTA A,3    ; multiplicou referenciaTh por 8
      ; código suprimido. Rotina para dividir A por 10

      ; ((8 * referenciaTh)/10) + ((2 * energiaEj)/10)
      DLD *AR4,B
      ADD B,A
      DST A,*AR5  ; armazeno referenciaTh

      ; código suprimido. Rotina para multiplicar A por 15
      ; código suprimido. Rotina para dividir A por 10
      DST A,*AR1  ; A contém o threshold, então armazeno.

      LD #0,B     ; é ruído
decisaoJahTratada:

      ; já tratou decisão
      ; acumulador B contém a decisao VAD
```

Figura 19 - Tomada de decisão do algoritmo EBD em assembly.

Após o cálculo da energia do *frame*, o valor é armazenado por meio da instrução DST na variável "energiaEj", que é apontada pelo registrador "AR4". Em seguida há o teste condicional, para saber se a energia do *frame* é maior que o *threshold*. Para realizar este teste, o valor do *threshold*, que é apontado pelo registrador "AR1" é carregado para o acumulador "B". Posteriormente é realizada a negação do valor do acumulador "B", como o acumulador "A" já continha a energia do *frame* é feita uma adição destes acumuladores, e o resultado fica no acumulador "A". Se o resultado da adição for maior que zero, a execução do código continuará na instrução posterior à BC, sendo com a instrução LD carregada a constante "1" no acumulador "B" para indicar atividade de voz neste *frame*. Após a indicação de atividade de voz é realizado um *branch* para o *label* "decisaoJahTratada" onde o acumulador em questão é utilizado para fazer a gravação da indicação em um arquivo, por exemplo. Caso o

resultado da adição seja menor ou igual a zero é feito um salto para o *label* “trataRuido” onde a referência do *threshold* e o valor correspondente do *threshold* são atualizados e ao final é carregada a constante “0” no acumulador “B”, indicativo de inatividade de voz para este *frame*.

## 4.3 ALGORITMO ZCD EM LINGUAGEM C

O algoritmo ZCD (*Zero Crossing Detector*) é uma extensão do *Energy Based Detector* (algoritmo EBD) como observado na Seção 2.4. Portanto grande parte deste algoritmo é compartilhada com as implementações em linguagem “C” e assembly. As particularidades a serem discutidas neste e no próximo item tratam apenas das partes que diferenciam ambos os algoritmos, em linguagem “C” e em assembly.

O cálculo da referência do *threshold* e o correspondente valor inicial do *threshold* são calculados do mesmo modo que no algoritmo EBD. Quando o algoritmo ZCD estiver operacional (após os vinte primeiros *frames*, que são utilizados para calcular o valor inicial do *threshold*), o valor da energia do *frame* é calculada e posteriormente comparada com o *threshold*. Caso ela seja menor, é realizada a contagem dos cruzamentos pela linha “zero” para as 80 amostras do *frame*, como mostra o trecho de código na Figura 20.

```
char sinalAtual, sinalAnterior;
unsigned short int numCruzamentos;

energiaEj = calculaEnergiaDoFrame(frame, T_FRAME);

if (energiaEj > threshold)
{
      vad = 1;
}
else
{
    numCruzamentos = 0;
    for(i=0; i<T_FRAME; i++)
      {
            sinalAtual = sinal(frame[i]);
            numCruzamentos += abs (sinalAtual - sinalAnterior);
            sinalAnterior = sinalAtual;
      }
      numCruzamentos /= 2;
      if((numCruzamentos >=5) && (numCruzamentos <=15))
      {
            vad = 1;
      }
else
      {
            vad = 0;
      }

// atualiza referencia do treshold e treshold

      referenciaTh = ((8*referenciaTh)/10) + ((2*energiaEj)/10);
      threshold = ((referenciaTh*15)/10);
}

// pega o sinal da ultima amostra, para frames não consecutivos.
sinalAnterior = sinal(frame[79]);
```

Figura 20 - Trecho de código da função principal do algoritmo ZCD.

O algoritmo ZCD calcula o número de cruzamentos por zero com o auxílio da função "sinal", que retorna 1 caso o parâmetro passado seja maior ou igual a zero, e -1 se negativo. O critério para determinar se um *frame* contém atividade de voz ou não, levando-se em consideração o número de cruzamentos por zero é descrito na Seção 2.4.

Após ter sido calculado o número de cruzamentos por zero, caso o resultado esteja entre cinco e quinze, é declarada atividade de voz para este *frame*, do contrário inatividade de voz, e independente da decisão tomada, sempre que for calculado o número de cruzamentos por zero o valor do *threshold* será atualizado.

Na última linha da Figura 20 é possível observar a instrução que obtém o sinal da última amostra do *frame*, esta instrução é necessária, pois no início da contagem de cruzamentos por zero a variável "sinalAnterior" deve conter o sinal da última amostra, do último *frame* processado.

## 4.4 ALGORITMO ZCD EM ASSEMBLY

A implementação em assembly do algoritmo ZCD, como no algoritmo EBD necessita calcular o valor inicial do *threshold* com os vinte primeiros *frames* formados. A diferença entre estes algoritmos está quando o algoritmo ZCD passa a fazer a classificação dos *frames*. No trecho de código da Figura 21 é possível observar o cálculo da energia de um *frame*.

```
T_FRAME           .set  80    ; constante de valor 80
amostrasZCD .space (T_FRAME*16)     ; 80 amostras

      STM   #T_FRAME-1,AR6    ; recarrega valor para loop
      LD    #0,A
      DST A,*AR4 ; zero a variavel energiaEj apontada por AR4
      STM amostrasZCD,AR2     ; aponto para inicio do vetor
calcEFrame:
            SQURA *AR2+, A    ; soma o quadrado das 80 amostras
      BANZ  calcEFrame,*AR6-

      ; código suprimido. Divide o acumulador A por 80

      DST A,*AR4 ; armazena a energia do frame
```

Figura 21 - Cálculo da energia do frame do algoritmo ZCD em assembly.

A cada *frame* de 80 amostras processado, as amostras correspondentes são copiadas para o "amostrasZCD", sendo que no trecho de código da Figura 21 isso já foi realizado. O vetor "amostrasZCD" também é utilizado caso o *frame* seja considerado inativo para voz, na primeira classificação do ZCD. O trecho de código da Figura 21 inicia com a declaração da constante "T_FRAME" com o valor 80 decimal, após, é declarado o vetor "amostrasZCD". Após o *label* "calcEFrame" inicia o *loop* para o cálculo da energia e também da armazenagem das 80 amostras do *frame*. O *loop* é composto por uma única instrução, SQURA, que eleva o valor da amostra lida ao quadrado e soma com o conteúdo do acumulador "A", que ao final conterá a soma do quadrado das 80 amostras. Nesta instrução também é utilizado o operador "+" para incrementar o registrador "AR2" que está sendo usado como um ponteiro para o vetor "amostrasZCD".

Quando um *frame* primeiramente é classificado como inatividade de voz, o número de cruzamentos por zero deve ser calculado, e novamente o *frame* deve ser classificado, considerando não mais a energia, e sim o número de *zero crossings* calculado. O trecho de código da Figura 22 aborda o que é realizado quando um *frame* tem a sua primeira classificação como inativo.

```
; foi detectado inatividade de voz
      ST #0, numCruzamentos
      STM #amostrasZCD,AR2 ; aponto novamente o vetor
      STM T_FRAME-1, BRC      ; fará 80 vezes
      RPTB  fimContaCruzamentos -1
            LD *AR2+,A ; carrego uma amostra
            ST #1,sinalAtual
            BC ehPositivo,AGEQ
            ST #-1,sinalAtual
ehPositivo:
            LD sinalAtual,A
            LD sinalAnterior,B
            NEG B
            ADD B,A
            ABS A
            LD numCruzamentos,B
            ADD A,B
            STL B, numCruzamentos
            MVDK sinalAtual,sinalAnterior
fimContaCruzamentos:

      LD numCruzamentos,A
      SFTA A,-1   ; divide por 2
      SUB #5,A
      ST #0,vad
      BC eraMenorQue5,ALT     ; pula se < 0
      SUB #11,A
      BC eraMaiorQue15,AGEQ
      ST #1,vad   ; estava entre ((nCross >=5) && (nCross <=15))
eraMenorQue5:
eraMaiorQue15:
; atualiza valor do threshold
```

Figura 22 - Cálculo do número de cruzementos por zero do algoritmo ZCD em assembly.

Na Figura 22, o trecho de código inicia zerando a variável “numCruzamentos”. Depois é ajustado o registrador “AR2” para apontar para o primeiro endereço do vetor “amostrasZCD”. A terceira instrução, utilizando o registrador BRC carrega o valor de quantas vezes o *loop* RPTB será realizado. O registrador BRC é carregado com 80 – 1, pois o *loop* RPTB (que é uma instrução para repetição de bloco de código) sempre executa o bloco de código uma vez mais do que valor carregado em BRC.

No *loop* que inicia após a instrução RPTB e termina com a instrução anterior ao *label* “fimContaCruzamentos” é realizada a contagem do número de cruzamentos por zero do *frame*. Após o *loop* o número de cruzamentos por zero é carregado no acumulador “A” e dividido por dois, com a instrução SFTA, que realiza um *shift* aritmético de um *bit* para a direita no acumulador “A”, pois a contagem de cruzamentos por zero sai do *loop* multiplicada por dois.

Por fim o *frame* é novamente classificado, considerando o número de cruzamentos por zero, e independente da segunda classificação o valor da referência do *threshold* e o correspondente valor do *threshold* serão atualizados.

## 4.5 ALGORITMO VAD DO CODEC G.729B EM LINGUAGEM C

As organizações ITU (*International Telecommunication Union*) e ETSI (*European Telecommunications Standards Institute*) são responsáveis por muitas das recomendações de algoritmos voltados à comunicação com e sem fio, tais como os codificadores de voz GSM (*Global System for Mobile communications*), G.723.1 e também o G.729 anexo B. Estes algoritmos empregam tipos de dados inteiros ANSI C e implementam operações aritméticas fracionárias de 16 bits.

Um algoritmo que utiliza o estilo ITU/ETSI é desenvolvido de modo que possa ser executado em qualquer compilador compatível com ANSI C. Todos os cálculos são realizados utilizando um conjunto predefinido de operações básicas (contidas no arquivo "basic_op.c"), de modo a facilitar a sua implementação em assembly de qualquer processador DSP.

A implementação do CODEC G.729B disponível pela ITU-T é composta pelo codificador e decodificador, possuindo também um algoritmo de VAD (Detecção de Atividade de Voz), DTX (Transmissão Descontínua) e CNG (Gerador de Ruído de Conforto). Para implementar apenas o VAD do CODEC G.729B, se fez necessário separar os arquivos que realizam esta função, de todo o CODEC.

Na Figura 23 são apresentados os arquivos usados na implementação do algoritmo VAD do CODEC G.729B. Embora na figura somente apareçam os arquivos ".c" também são necessárias as respectivas bibliotecas (*.h) que além da declaração das funções dos arquivos ".c" também possuem as declarações de algumas constantes utilizadas.

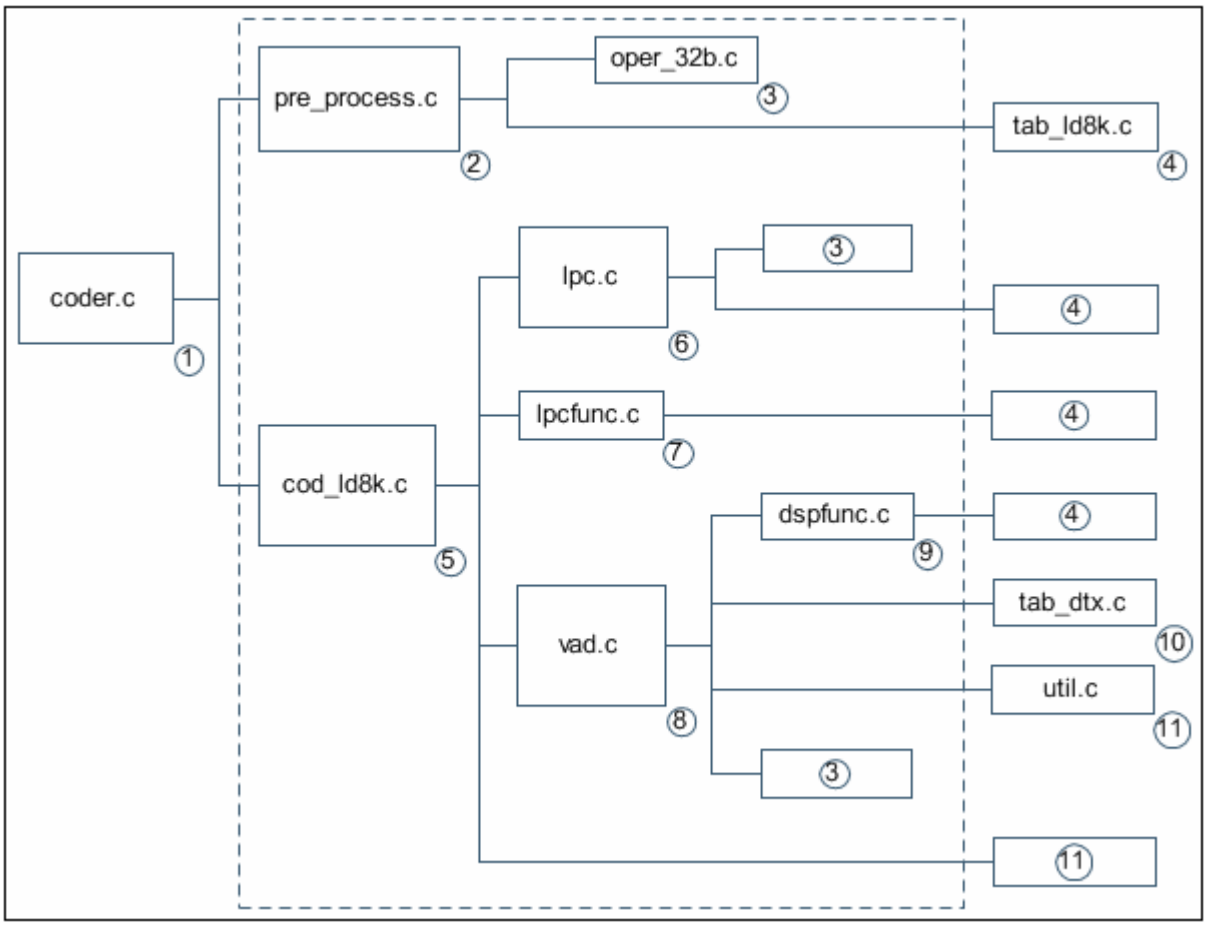


Figura 23 - Arquivos utilizados na implementação do VAD do G.729B.

Os arquivos que estão dentro da área tracejada na Figura 23 fazem uso do arquivo "basic_op.c", que facilita a implementação do algoritmo em um processador DSP. O arquivo "coder.c" é responsável pela leitura e escrita de dados de entrada/saída e também faz chamada a funções dos arquivos "pre_process.c" e "cod_ld8k.c". Os arquivos de número 4 e 10 contêm listas de valores constantes e o arquivo de número 11, "util.c", contém duas funções, a primeira zera os valores de um vetor e a segunda copia os valores de um vetor para outro. Os arquivos dentro da área tracejada da Figura 23 realizam o processamento e informam se a indicação de VAD deve ser setada ou não.

## 4.6 ALGORITMO VAD DO CODEC G.729B EM ASSEMBLY

A implementação em linguagem "C" utiliza a representação de números em modo fracionário, como por exemplo, Q15[10]. Quando dois números nesta representação são multiplicados, um bit extra de sinal é gerado. Portanto, um *shift* de um bit para a esquerda é necessário para remover o bit de sinal redundante. Na implementação em assembly isto foi realizado de maneira automática, setando o bit FRCT (*Fractional mode* - modo Fracionário) do DSP C5410.

Outro ponto é que na implementação em linguagem "C" foi utilizada a variável "Overflow", de um bit, que é setado quando o valor do número armazenado em um acumulador excede determinada faixa. Na implementação do VAD do G.729B, o valor do acumulador é sempre limitado à faixa de 0x80000000 a 0x7FFFFFFF; sendo estes valores os limites negativo e positivo, respectivamente. Na implementação em assembly o bit *overflow* foi tratado automaticamente por setar o bit OVM (*Overflow Mode* - Modo Overflow) do registrador PMST, que satura o acumulador quando da ocorrência de um *overflow*.

O trecho de código da Figura 24 mostra a função "L_mac", encontrada no arquivo "basic_op.c", que faz a multiplicação de duas variáveis de 16 bits e acumula o resultado a uma terceira variável, de 32 bits.

```
Word32 L_mac(Word32 L_var3, Word16 var1, Word16
var2)
{
      Word32 L_var_out;
      Word32 L_produit;

      L_produit = L_mult(var1,var2);
      L_var_out = L_add(L_var3,L_produit);
      return(L_var_out);
}
```

Figura 24 - Operação MAC em linguagem C do algoritmo VAD do G.729B.

A função "L_mac" é a implementação em linguagem "C" da instrução MAC do DSP, que realiza a multiplicação entre dois valores e acumula o resultado da operação a um terceiro valor.

[10] O formato Q15 é aquele onde o ponto binário está localizado à direita do bit 15 em um número de 16 bits. É possível representar números de -1 à +0.999969482421875, sendo, por exemplo, o número 0,5 decimal representado por 0x4000 no formato fracionário Q15 (DUQUE, 2004).

No trecho de código da Figura 25 é possível observar a implementação da função “L_mac” no assembly do DSP C5410.

```
STM #var2,AR2
STM #var1,AR3
; L_var3 deve estar no acumulador A
MAC *AR2, *AR3, A
; resultado no acumulador A
```

Figura 25 - Operação MAC em assembly do algoritmo VAD do G.729B.

Com exceção de três funções do arquivo “basic_op.c” todas as outras foram implementadas em assembly utilizando as próprias instruções do processador DSP, a exemplo da instrução MAC. Os arquivos “tab_ld8k.c” e “tab_dtx.c” da implementação em linguagem “C” contêm apenas listas de valores constantes, com isso na implementação em assembly esses valores foram carregados diretamente na memória de dados do DSP. A implementação dos outros arquivos “.c” se deu de modo similar à de “basic_op”, sendo que as partes mais complexas da implementação foram as de estruturas de controle aninhadas, como por exemplo, um *loop* do tipo “`for`” dentro de um *loop* do tipo “`while`”.

## 4.7 CONSIDERAÇÕES

Este capítulo apresentou algumas características relevantes sobre as implementações dos algoritmos VAD considerados neste trabalho. Foram discutidas as implementações realizadas em linguagem “C” e assembly do processador TMS320C5410.

A validação das implementações dos algoritmos foi realizada comparando-se os resultados em linguagem “C” com os obtidos na linguagem assembly. Observou-se em tal comparação resultados iguais, comprovando a equivalência das implementações.

# 5 RESULTADOS

Este capítulo apresenta os resultados da comparação entre os três algoritmos VAD considerados neste trabalho. São avaliados a ocupação de memória, complexidade computacional e desempenho dos algoritmos, considerando as implementações em linguagem assembly.

## 5.1 RECURSOS COMPUTACIONAIS

Os recursos computacionais avaliados dos algoritmos são: ocupação de memória e complexidade computacional (mipagem[11]). Os resultados são apresentados na Tabela 1.

Tabela 1 – Ocupação de memória e mipagem dos algoritmos.

| Implementação | Memória de programa | Memória de dados | Mipagem |
|---|---|---|---|
| Algoritmo EBD | 658 B | 194 B | 0,064 MIPS |
| Algoritmo ZCD | 770 B | 202 B | 0,228 MIPS |
| VAD do G.729B | 6568 B | 2658 B | 8,655 MIPS |

A ocupação de memória dos algoritmos é obtida a partir do arquivo ".map" que o CCS gera no momento da compilação. O algoritmo EBD (*Energy Based Detector*) é o menor dos três, ocupando 658 bytes de memória de programa e 194 bytes de memória de dados. Em segundo lugar fica o algoritmo ZCD (*Zero Crossing Detector*) que utiliza 770 bytes de memória de programa e 202 bytes de memória de dados. O maior dos algoritmos, e que mais ocupa memória, é o VAD do CODEC G.729B, ocupando 6568 bytes de memória de programa e 2658 bytes de memória de dados.

Para a realização do cálculo da mipagem dos algoritmos é utilizado o número de ciclos de *clock* gastos. Esta informação é obtida diretamente na ferramenta *profile* do CCS que informa os números mínimo, máximo e médio de ciclos de *clock* no trecho de código avaliado.

[11] O termo "mipagem" é uma medida que denota quantos MIPS (Milhões de Instruções Por Segundo) um processador é capaz de executar (SABBATINI, 1997).

O cálculo da mipagem dos algoritmos foi realizado com base em um *frame* de 80 amostras, freqüência de amostragem de 8 kHz e capacidade computacional do processador DSP utilizado, que é de 100 MIPS.

O algoritmo *Energy Based Detector* (EBD) é o que realiza menos processamento dentre os três implementados e sua mipagem é de 0,064 MIPS. O *Zero Crossing Detector* (ZCD) foi o segundo algoritmo de maior mipagem, realizando até 0,228 MIPS devido à segunda regra de classificação adicionada. Por fim, o algoritmo VAD do CODEC G.729B foi o de maior mipagem, com 8,655 MIPS, o que demonstra em parte a sua grande complexidade computacional na realização dos cálculos do algoritmo, e também sua robustez, como pode ser observado na comparação de desempenho dos algoritmos apresentada na seqüência.

## 5.2 COMPARAÇÃO DE DESEMPENHO ENTRE OS ALGORITMOS

A análise comparativa de desempenho entre os três algoritmos de VAD foi efetuada considerando-se um procedimento semelhante ao realizado por Davis e Nordholm (2003).

Os algoritmos de VAD foram avaliados utilizando quatro arquivos de áudio, sendo dois deles gravados por um interlocutor do sexo feminino e dois do sexo masculino. Ruídos do banco de dados SPIB (1995) do tipo *white noise*[12], *pink noise*[13], *babble noise*[14], e Volvo *car noise* (ruído gravado no interior de um carro Volvo 340 a 120 Km/h) foram adicionados aos arquivos de áudio originais. A cada arquivo de áudio foi adicionado somente um tipo de ruído, sendo que foram produzidas cinco relações sinal ruído (SNR), sendo elas: infinito (nenhum ruído foi adicionado), 20dB, 15dB, 10dB e 5dB. Sendo assim foram gerados no total vinte arquivos, quatro arquivos originais com cinco relações SNR cada.

Os algoritmos de VAD foram então testados utilizando-se os arquivos com ruídos e as decisões foram armazenadas em arquivos de saída. Estas decisões foram então comparadas

[12] Ruído branco (*white noise*) é um sinal composto por um espectro de freqüência onde todas as componentes possuem a mesma potência (ADOBE, 2007).
[13] Ruído rosa (*pink noise*) é, por definição, aquele cuja densidade espectral de potência é proporcional ao inverso da freqüência, sendo um tipo de ruído comumente encontrado na natureza (WIKIPEDIA, 2008).
[14] Ruído *babble* é tipicamente um discurso ao fundo que envolve múltiplos interlocutores falando ao mesmo tempo, produzindo assim um ruído de fala incompreensível (GORODNITSKY, 2008).

*frame* a *frame*, à um conjunto de decisões feitas manualmente (um arquivo "rotulado").A porcentagem de voz ou de silêncio, bem como a duração dos arquivos rotulados são apresentadas na Tabela 2.

Tabela 2 – Porcentagem de voz ou de silêncio e duração dos arquivos rotulados.

| Arquivo rotulado | % de voz | % de silêncio | Duração (segundos) |
|---|---|---|---|
| 1 | 36.35 | 63.65 | 23,0 |
| 2 | 46.60 | 53.40 | 23,4 |
| 3 | 49.61 | 50.39 | 19,2 |
| 4 | 58.61 | 41.39 | 20,4 |

Após a análise dos resultados a porcentagem de *frames* classificados pelos algoritmos como sendo voz (%V), silêncio ou ruído ambiente (%S) e alarme-falso (%AF) são apresentados na Tabela 3. Considera-se alarme-falso os *frames* que eram voz, mas foram classificados como silêncio ou silêncio que foram classificados como sendo voz.

Tabela 3 – Comparação de desempenho entre os algoritmos.

| Arquivo | Ambiente | | EBD | | | ZCD | | | G.729B | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Nº | Ruído | SNR | %V | %S | %AF | %V | %S | %AF | %V | %S | %AF |
| 1 | Silêncio | Inf. | 36,35 | 63,65 | 0,00 | 36,35 | 63,65 | 0,00 | 38,87 | 61,13 | 2,52 |
| 1 | White | 20dB | 80,91 | 19,09 | 44,57 | 80,91 | 19,09 | 44,57 | 53,96 | 46,04 | 18,74 |
| 1 | White | 15dB | 80,43 | 19,57 | 44,17 | 80,43 | 19,57 | 44,17 | 50,70 | 49,30 | 21,04 |
| 1 | White | 10dB | 80,26 | 19,74 | 44,35 | 80,26 | 19,74 | 44,35 | 44,91 | 55,09 | 25,87 |
| 1 | White | 5dB | 77,78 | 22,22 | 46,83 | 77,78 | 22,22 | 46,83 | 37,43 | 62,57 | 33,09 |
| 2 | Silêncio | Inf. | 46,60 | 53,40 | 0,00 | 46,60 | 53,40 | 0,00 | 49,08 | 50,92 | 2,48 |
| 2 | Pink | 20dB | 91,40 | 8,60 | 44,80 | 93,40 | 6,60 | 46,81 | 61,87 | 38,13 | 16,65 |
| 2 | Pink | 15dB | 91,31 | 8,69 | 44,72 | 93,50 | 6,50 | 46,90 | 60,98 | 39,02 | 18,49 |
| 2 | Pink | 10dB | 62,43 | 37,57 | 21,57 | 70,52 | 29,48 | 27,77 | 53,19 | 46,81 | 14,21 |
| 2 | Pink | 5dB | 74,88 | 25,12 | 38,13 | 79,55 | 20,45 | 40,56 | 45,10 | 54,90 | 23,83 |
| 3 | Silêncio | Inf. | 49,61 | 50,39 | 0,00 | 49,61 | 50,39 | 0,00 | 53,83 | 46,17 | 4,23 |
| 3 | Babble | 20dB | 68,13 | 31,87 | 18,62 | 92,91 | 7,09 | 43,30 | 57,64 | 42,36 | 9,39 |
| 3 | Babble | 15dB | 68,44 | 31,56 | 20,08 | 93,27 | 6,73 | 43,77 | 56,44 | 43,56 | 11,53 |
| 3 | Babble | 10dB | 65,52 | 34,48 | 21,13 | 92,44 | 7,56 | 43,77 | 49,77 | 50,23 | 16,54 |
| 3 | Babble | 5dB | 64,11 | 35,89 | 29,63 | 91,71 | 8,29 | 46,06 | 54,51 | 45,49 | 25,77 |
| 4 | Silêncio | Inf. | 58,62 | 41,38 | 0,05 | 58,62 | 41,38 | 0,05 | 60,79 | 39,21 | 2,41 |
| 4 | Volvo | 20dB | 92,58 | 7,42 | 34,40 | 94,35 | 5,65 | 35,87 | 75,97 | 24,03 | 17,99 |
| 4 | Volvo | 15dB | 92,43 | 7,57 | 34,94 | 94,55 | 5,45 | 36,17 | 74,74 | 25,26 | 18,03 |
| 4 | Volvo | 10dB | 92,04 | 7,96 | 36,22 | 94,10 | 5,90 | 37,10 | 75,18 | 24,82 | 21,62 |
| 4 | Volvo | 5dB | 91,65 | 8,35 | 37,40 | 93,91 | 6,09 | 37,79 | 73,17 | 26,83 | 27,76 |

Os algoritmos EBD (*Energy Based Detector*) e ZCD (*Zero Crossing Detector*) obtiveram melhor resultado em condições ideais (sem ruído presente no sinal de áudio), já na presença de ruído, o algoritmo VAD do CODEC G.729B obteve o melhor resultado. O VAD do G.729B demonstrou na prática a sua robustez na distinção correta entre voz e ruído ambiente, mesmo em condições com baixa relação SNR, com desempenho de até 37% melhor que os algoritmos EBD e ZCD.

Conforme apresentado na Seção 2.4 deste trabalho, a regra de classificação adicionada ao ZCD considera que para um *frame* de voz de 10 ms ou 80 amostras, o número de cruzamentos por zero é de 5 a 15 vezes e para um *frame* contendo somente ruído o número é imprevisível e aleatório. O algoritmo ZCD permitiu maior detecção de *frames* contendo voz, comparado ao EBD. Parte dos *frames* que ambos os algoritmos primeiramente classificaram como ruído, na segunda classificação do ZCD foram indicados como voz. O que foi observado nas simulações é que alguns dos *frames* contendo somente ruído ambiente, primeiramente classificados como ruído, na segunda classificação foram indicados como sendo voz, aumentando a porcentagem de alarmes-falsos do algoritmo ZCD. Por exemplo, o ambiente com o tipo de ruído *pink* e SNR de 5 dB, neste caso o algoritmo EBD indicou 74,88% dos *frames* como sendo voz e o ZCD 79,55%. O algoritmo ZCD indicou maior quantidade de *frames* contendo voz e apresentou uma maior quantidade de alarmes-falsos comparado ao EBD. Com isso, o algoritmo EBD fica em segundo lugar considerando-se a porcentagem de alarmes-falsos, mas em qualidade de áudio após ter sido aplicado o VAD, por ter classificado menos *frames* como sendo voz, fica com uma qualidade inferior ao ZCD.

Com base nos dados da Tabela 3 pode-se concluir que o algoritmo com o melhor desempenho na distinção correta entre voz e ruído ambiente foi o algoritmo VAD do CODEC G.729B. Este algoritmo, além de fazer parte de um codificador de voz, também é visto como uma referência em se tratando de detecção de atividade de voz, sendo utilizado para realizar comparações de desempenho quando um novo algoritmo de VAD é proposto.

## 5.3 CONSIDERAÇÕES

Neste capítulo foram apresentados resultados das comparações entre os algoritmos considerados neste trabalho. O algoritmo *Energy Based Detector* (EBD) pela sua simplicidade

foi o de menor ocupação de memória e mipagem, mas ficou em segundo considerando-se a porcentagem de alarmes-falsos. O *Zero Crossing Detector* apresentou uma ocupação de memória e mipagem superior ao EBD e foi o algoritmo de maior quantidade de alarmes-falsos dentre os algoritmos implementados. O VAD do CODEC G.729B foi o que teve a maior ocupação de memória e mipagem, porém foi o algoritmo que apresentou a menor quantidade de alarmes-falsos.

# 6 CONCLUSÕES

Neste trabalho foi apresentada a implementação em um DSP de três algoritmos de Detecção de Atividade de Voz (VAD – *Voice Activity Detection*) sendo eles: *Energy Based Detector*, *Zero Crossing Detector* e o VAD do CODEC G.729 anexo B. Posteriormente à implementação também foi realizada uma comparação entre os algoritmos escolhidos.

A realização deste trabalho teve início com pesquisas em literaturas, principalmente em artigos publicados na IEEE (*Institute of Electrical and Electronics Engineers*) que dispõe de um vasto conteúdo sobre o tema de detecção de atividade de voz.

Após os algoritmos VAD terem sido selecionados foi iniciada a implementação em linguagem "C". A implementação do CODEC G.729B em tal linguagem é disponibilizada pela ITU-T. Deve-se frisar que para implementar apenas o VAD do CODEC G.729B, se fez necessário separar os arquivos que realizam esta função de todo o CODEC.

A implementação em linguagem "C" foi utilizada como referência para a implementação em linguagem assembly do processador DSP TMS320C5410 da Texas Instruments. Com o propósito de validar a implementação em linguagem assembly, foram comparadas as indicações VAD dos algoritmos em linguagem "C" com as da linguagem assembly.

Para avaliar os algoritmos implementados neste trabalho, eles foram comparados considerando-se: ocupação de memória, complexidade computacional (mipagem) e desempenho.

O algoritmo *Energy Based Detector* (EBD) é o mais simples dos três implementados. Este possui a menor ocupação de memória e mipagem e apresentou uma porcentagem de alarmes-falsos (*frames* que foram classificados erroneamente) inferior ao *Zero Crossing Detector* (ZCD). O algoritmo ZCD ficou em segundo lugar levando-se em consideração a ocupação de memória e mipagem, mas em se tratando da comparação de desempenho, teve uma maior taxa de alarmes-falsos. Tal resultado deve-se ao fato de na segunda regra de classificação deste algoritmo ele ter indicado *frames* que eram ruído, como sendo voz. O algoritmo VAD do CODEC G.729B teve a maior ocupação de memória e mipagem e também o que apresentou o melhor desempenho, tendo menor quantidade de alarmes-falsos. Este algoritmo demonstrou

sua robustez em determinados ambientes, considerando-se quatro tipos de ruídos e cinco relações SNR.

Os objetivos definidos neste trabalho de conclusão de curso foram alcançados com êxito. Este trabalho não só possibilitou a aquisição de novos conhecimentos e experiência sobre algoritmos de detecção de atividade de voz (VAD – *Voice Activity Detection*), mas também a utilização dos conhecimentos teóricos e práticos adquiridos durante o curso de Engenharia de Computação.

## 6.1 TRABALHOS FUTUROS

Como sugestões para trabalhos futuros os seguintes itens podem ser considerados:

- Implementação em hardware dos algoritmos de VAD considerados;
- Implementação de outros algoritmos de VAD, como por exemplo, o algoritmo VAD do CODEC AMR (*Adaptive Multi-Rate*) ou então o VAD do CODEC GSM (*Global System for Mobile communications*);
- Implementação de novas estratégias de detecção de atividade de voz, por exemplo, lógica *Fuzzy*.

# REFERÊNCIAS BIBLIOGRÁFICAS


ADOBE. **Adobe Audition 3** - User guide. 2007. Disponível em: <http://help.adobe.com/en_US/Audition/3.0/audition_3_help.pdf>. Acesso em: 26 de maio de 2008.

APPIAH, M. Y. *et al*. **Robust Voice Activity Detection and Noise Reduction Mechanism Using Higher-Order Statistics.** Technical report, Institute of Eletronics Systems, Aalborg University, 2005.

BENYASSINE, A. *et al*. ITU-T Recommendation G.729 Annex B: A Silence Compression Scheme for Use with G.729 Optimized for V.70 Digital Simultaneous Voice and Data Applications. **IEEE Communications Magazine**, [S.l.], v. 35, ed. 9, p. 64-73, set. 1997.

BERNAL FILHO, Huber. **Telefonia IP**. 2003, disponível em: <http://www.teleco.com.br/tutoriais/tutorialtelip/default.asp>. Acesso em: 17 de agosto de 2007.

CCDR-ALG. **Recolha de Dados Acústicos.** 2005, disponível em: <http://www.ccdr-alg.pt/ccr/index.php?module=ContentExpress&func=display&ceid=344>. Acesso em: 5 de novembro de 2007.

CERVO, Amado Luiz.; BERVIAN, Pedro Alcino. **Metodologia científica.** 4ª. ed. São Paulo: Makron Books, 1996. ISBN 85-346-0521-1.

CISCO. **IP Telephony/Voice over IP (VoIP)**. 2007. disponível em: <http://www.cisco.com/en/US/tech/tk652/tk701/tsd_technology_support_protocol_home.html >. Acesso em: 17 de agosto de 2007.

COUTINHO, Pedro. **Tecnologia Multimédia.** Escola de Engenharia - Universidade do Minho, 2005. Disponível em: <http://piano.dsi.uminho.pt/disciplinas/TM/aulas/sessao3.pdf>. Acesso em: 11 de março de 2008.

DAVIS, Alan; NORDHOLM, Sven. A Low Complexity Statistical Voice Activity Detector with Performance Comparisons to ITU-T / ETSI Voice Activity Detectors. *In*: Joint Conference of The Fourth International Conference on Information, Communications & Signal Processing and Fourth Pacific-Rim Conference on Multimedia. 2003, Singapore. **Joint...** (Proceedings) [S.l.]: IEEE, 2003. p. 119-123.

Dev-C++. Version 4.9.9.2. [S.l.]: Bloodshed Software, 2005. Disponível em: <http://www.bloodshed.net/dev/devcpp.html>. Acesso em: 13 de dezembro de 2007.

DÍGITRO TECNOLOGIA. **Glossário Tecnológico.** Florianópolis, [2008]. Disponível em: <http://www.digitro.com/pt/tecnologia_glossario-tecnologico.php?index=F>. Acesso em: 20 fevereiro 2008.

DUQUE, Carlos Augusto. **O processador digital de sinais família Tms320f2xx.** 2004. Disponível em: http://www.mestradoeletrica.ufjf.br/professores/duque/microp_cap4.pdf. Acesso em: 01 de agosto de 2007.

ESTEVEZ, P.A. *et al.* **Genetic programming-based voice activity detection.** Electronics Letters, v. 41, ed. 20, p. 1141-1143, set. 2005.

GORODNITSKY, Irina. Evaluation of derivative time-delay modeling for robust pitch detection in very high and nonstationary noise. *In*: PCI - Problems of Cybernetics and

Informatics. 2., 2008, Baku. **The Second...** (Papers) [S.l.]: [s.n.], [2008?]. Disponível em: <www.science.az/cyber/pci2008/3/3-40.doc>. Acesso em: 11 de junho de 2008.

GUIA DO HARDWARE. **Bit-rate**. [200-?], Disponível em: <http://www.guiadohardware.net/termos/bit-rate>. Acesso em: 18 de fevereiro de 2008.

HARDY, William C. **VoIP Service Quality**: Measuring and Evaluating Packet-Switched Voice. New York: McGraw-Hill, 2003.

ITU-T. **ITU-T Recommendation G.711**: Pulse code modulation (PCM) of voice frequencies. 1988. Disponível em: <http://www.itu.int/rec/T-REC-G.711-198811-I/en>. Acesso em: 12 de novembro de 2007.

_____. **ITU-T Recommendation G.729**: Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP). 1996a. Disponível em: <http://www.itu.int/rec/T-REC-G.729-199603-S/en>. Acesso em: 4 de outubro de 2007.

_____. **ITU-T Recommendation G.729 Annex B**: A silence compression scheme for G.729 optimized for terminals conforming to Recommendation V.70 digital simultaneous voice and data applications. 1996b. Disponível em: <http://www.itu.int/rec/T-REC-G.729-199610-S!AnnB/en>. Acesso em: 28 de setembro de 2007.

KHASNABISH, Bhumip. **Implementing Voice over IP.** New Jersey: Wiley-Interscience, 2003. [Online], ISBN: 9780471225270.

KUO, Sen M.; LEE, Bob H.; TIAN, Wenshun. **Real-Time Digital Signal Processing: Implementations and Applications.** 2. ed. Chichester: John Wiley & Sons Ltd, 2006. ISBN 0-470-01495-4.

KUULUSA, Mika. **DSP Processor Core-Based Wireless System Design.** Tese de doutorado. Tampere University of Technology, Tampere, Finland, 2000. Disponível em: <http://edu.cs.tut.fi/kuulusa296.pdf>. Acesso em: 17 de março de 2008.

LOPES, João. **Processamento de dados.** 2004, Disponível em: <http://bsel.ist.utl.pt/2007/PortalQuimiometria/Contents/procdados/node2.html>. Acesso em: 5 de novembro de 2007.

PEREZ MEANA, Hector. **Advances in audio and speech signal processing**: technologies and applications. Hershey: Idea Group Publishing, 2007.

PRASAD, R. V. *et al.* Comparison of Voice Activity Detection Algorithms for VoIP. *In*: IEEE symposium on Computers and Communications, 7., 2002, Taormina. **The...** (Proceedings) [S.l.]: IEEE, 2002. p. 530-535.

PRASAD, R. V. *et al*. Voice Activity Detection for VoIP - An Information Theoretic Approach. *In*: IEEE Global Telecommunications Conference (IEEE GLOBECOM), 49. 2006, San Francisco, **IEEE...** (Proceedings) [S.l.]: IEEE, 2006.

RABINER, Lawrence; JUANG, Biing-Hwang. **Fundamentals of speech recognition.** Englewood Cliffs, New Jersey: Prentice Hall, 1993. ISBN 0-13-285826-6.

RAMIRES, Anderson; VIOTTI, Luiz. **Voice over Internet Protocol (VoIP) – Apenas uma moda ou uma nova tecnologia que irá mudar nossa forma de comunicação?** 2005, disponível em: <http://www.teleco.com.br/emdebate/ramires01.asp>. Acesso em: 18 de agosto de 2007.

Recommendation G.729 Annex B. Version 1.4. [S.l.]: International Telecommunication Union, 2000. Disponível em: <http://www.itu.int/rec/T-REC-G.729-199610-S!AnnB/en>. Acesso em: 28 de setembro de 2007.

RENEVEY, Philippe; DRYGAJLO, Andrzej. Entropy based voice activity detection in very noisy conditions. Proceedings of European Conference on Speech Communication and Technology, 7., 2001, Aalborg. **Seventh...** [S.l.]: [s.n.], 2001. p. 1887-1890.

SABBATINI, Renato M. E. **TeraFLOPS à vista**. 1997. Disponível em: <http://www.sabbatini.com/renato/correio/cp970214.htm>. Acesso em: 25 de maio de 2008.

SILVA, Danilo. Atributos de voz para reconhecimento. *In*:_____. **Códigos de Permutação para Compressão de Dados e Modulação.** [2005]. Dissertação (Mestrado em Engenharia Elétrica) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, [2005]. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/cgi-bin/PRG_0599.EXE/6201_5.PDF?NrOcoSis=17261&CdLinPrg=pt>, Acesso em: 17 de outubro de 2007.

SINGH, Deepti; BOLAND, Frank. **Voice Activity Detection.** ACM, New York. Setembro [2007]. v. 13, ed. 4, p. 7, Disponível em: <http://www.acm.org/crossroads/xrds13-4/voice_detection.html>. Acesso em: 21 de agosto de 2007.

SMITH, Steven W. **The Scientist and Engineer's Guide to Digital Signal Processing.** 2ª. ed. San Diego: California Technical Publishing, 1999. ISBN 0-9660176-6-8.

SOARES, Luiz Fernando Gomes *et al*. **Redes de Computadores: das LANs, MANs E WANs às redes ATM.** Rio de Janeiro: Campus, 1995.

SPIB. Signal Processing Information Base. **Noise data**. 1995, Disponível em: <http://spib.rice.edu/spib/select_noise.html>. Acesso em: 26 de maio de 2008.

TAN, Edwin J.; HEINZELMAN, Wendi B. DSP Architectures: Past, Present and Future. **Computer Architecture News**, [S.l.], v. 31, n. 3, p. 6-19, jun. 2003. Disponível em: <www.ece.rochester.edu/research/wcng/papers/journal/CAN_r1.pdf>. Acesso em: 21 de abril de 2008.

TI. Texas Instruments. **TMS320C54x DSP Programming Environment.** 1997. Disponível em: <http://focus.ti.com/lit/an/spra182/spra182.pdf>. Acesso em: 17 de abril de 2008.

_____. Texas Instruments. **TMS320VC5410 Fixed-Point Digital Signal Processor - Data Manual.** 2000. Disponível em: <http://focus.ti.com/lit/ds/symlink/tms320vc5410.pdf>. Acesso em: 26 de março de 2008.

_____. Texas Instruments. **TMS320C54x DSP Reference Set - Volume 1**: CPU and Peripherals. 2001a. Disponível em: <http://focus.ti.com/lit/ug/spru131g/spru131g.pdf>. Acesso em: 8 de fevereiro de 2008.

_____. Texas Instruments. **TMS320C54x DSP programmer's guide**. 2001b. Disponível em: <http://focus.ti.com/lit/ug/spru538/spru538.pdf>. Acesso em: 14 de janeiro de 2008.

_____. Texas Instruments. **Code Composer Studio IDE.** [2007?]. Disponível em: <http://focus.ti.com/dsp/docs/dspsupporto.tsp?sectionId=3&tabId=452>. Acesso em: 25 de março de 2008.

VENKATARAMANI, B.; BHASKAR, M. **Digital Signal Processors**: Architecture, Programming and Applications. New Delhi: Tata McGraw – Hill Publishing Company Limited, 2002. ISBN 0-07-047334-X.

WIKIPEDIA. **Pink noise**. 2008. Disponível em: <http://en.wikipedia.org/wiki/Pink_noise>. Acesso em: 15 de junho de 2008.

YAMAMOTO, K. *et al*. Robust Endpoint Detection for Speech Recognition Based on Discriminative Feature Extraction. *In*: IEEE International Conference on Acoustics, Speech, and Signal Processing, 2006, Toulouse. **ICASSP 2006** (Poster), [S.l.]: IEEE, 2006. p. 805-808.